Team Design Project & Skills Final Report

TEAM 27 Frende

GUID	Name
2357509T	Tang Hao
2357600B	Bian Qing
2357490Y	Yin Xiangyu
2357505L	Li Yichu
2357503W	Wang Jiapeng
2357476L	Li Wenyu
2357480C	Chen Qianyi
2357805X	Xia Xiaoyou
2357512X	Xie Kunyang
2357504L	Li Zhaoyu

Abstract

In this course, students are required to work in a group and complete a complicated design project, and the project will be purely simulation-based using the Cyberbotics Webot. After analyzing the requirement, we divided ourselves into 5 groups who take charge of image recognition, distance recognition, casting, car motor, and graph design to complete tasks such as path tracing and food casting.

The main body of the report is divided into 4 parts: introduction, experimental details, results, and conclusions & recommendations, which present the whole process of completing the design project exhaustively. The table of contents is attached at the beginning, while the reference and appendix (the code we used) at the end.

Also, for the team collaboration and arrangement part, Group Structure is shown in **Appendix II**.

Contents

1. Introduction

The Team Design Project and Skills is aimed to complete 5 tasks with Cyberbotics Webots. In this part, task requirements and detailed assignments for each group will be introduced.

1.1. Requirement Analysis

The project aims to build experience of working in a team to design and construct an integrated electronic system that must perform a specific function within a budget. For this time, because of the coronavirus, the whole process is simulated in Webots.

The first requirement is to build the patio below in Webots:

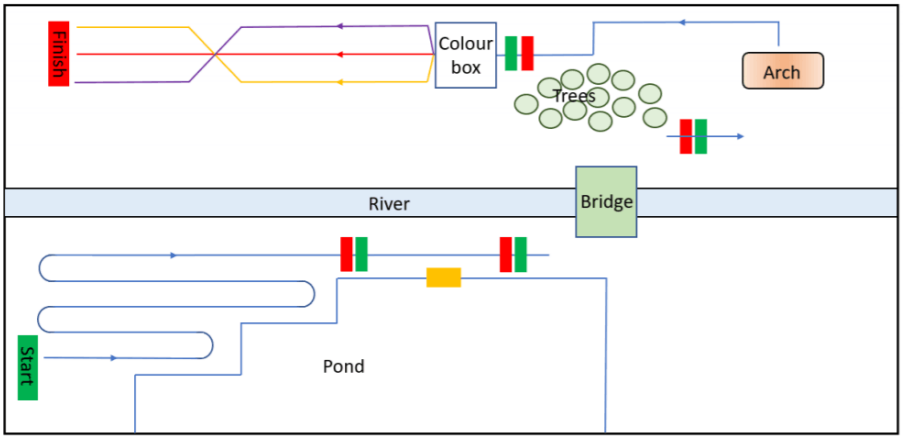


Figure 1 The patio

1.2. Task Structure

Then there are 5 tasks:

- Follow the path of the lines on the floor.
- Find the orange box and release the object/food in the pond.
- Find the bridge, cross it, and then detect the trees and turn right.
- Find the arch and go through it and then follow the line on the ground.
- Reads the color of the box, follow the correct color until it reaches the finish line.

Notice that the bridge should be 100 cm wide and 3 meters long, which includes the ramps that will be used to roll up and off the bridge. And after passing through the arch, the car should follow the line on the ground and avoid colliding with the trees by turning right. Besides, the color in task 5 could be set manually or automatically using a random color generator.

1.2.1. Graph Design

This section gives a detailed explanation of how we built the patio as required in Webots. The designing task is divided into 4 parts including checking the requirement, building patio, building objects, and other improvements. This chapter will give an exhaustive illustration of how we use Webots to build the ground, road, and other objects such as the color box. Besides, during our building process, we find that some designs can be added into the patio to help to complete our task, which will be given a specific introduction in the last part.

1.2.2. Casting

The releasing task uses multiple functions from other tasks to finalize the objective of releasing the food. The task will be divided into three separate procedures: reaching the spot for casting, casting, and leaving the spot & back to the track.

There exists an orange box along the pond, and the principal purpose of the casting task is to detect the orange box and throw or release the object ("fish food" in this experiment) into this box. When the automatic car reaches the spot of the orange sign, it will adjust the direction and position for releasing. After the object has been released, the car will be redirected and move back to the tracking line.

To achieve these goals, a mechanical arm and several modules are selected and performed in the whole process of casting.

1.2.3. Car Design and Control

The task of our group is to link the achievements of distance group and vision group together so that the car becomes a whole system, besides, our group also has to complete the code of the main task flow. We need to understand the use of the various sensors in Webots and the relevant algorithms used by two groups. In this report, I will describe the state machine for this task in detail.

1.2.4. Distance Recognition

The goal of the distance recognition is to determine the distance and the angle between the detected target and the middle position of the car, which provide a precise moving velocity and direction to the car motor.

To achieve the aim, we introduce 2 modules: The IMU module and the Lidar module.

The IMU module can determine the current moving angle of the car, and

the lidar module can transfer the position of the detected target into coordinate points.

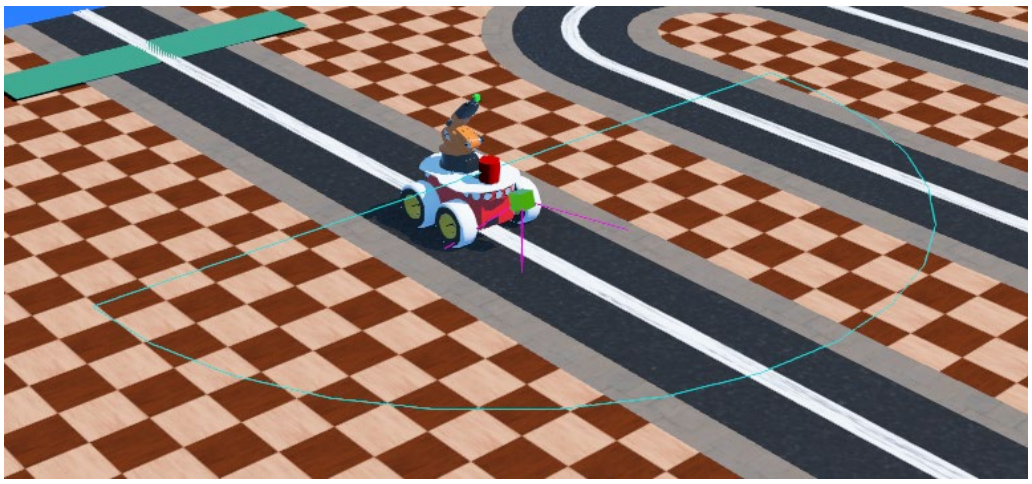


Figure 2 The Lidar module

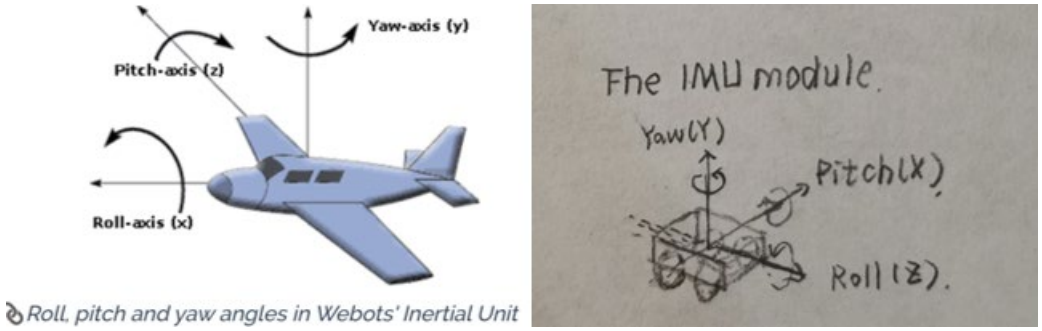


Figure 3 The IMU module

1.2.5. Image Recognition

The vision task plays a crucial role in the design of the whole project as it allows the robot car to take the image of the environment as the input and performs different tasks correspondingly. To realize this goal, we divide the task into two parts, namely color recognition and line tracing. In both parts, programs are developed based on C++ using the relevant functions in Webots. The results show that our car can successfully move along the line at straight lines, turnings, and intersections. Also, in the last task, our car can recognize the color in the color box and follow the line of the same color afterward.

2. Experiment Details

2.1. Graph Design (Li Yichu & Xia Xiaoyou)

2.1.1. Patro Building

2.1.1.1. Background

This part will explain how we built the ground, river, and light source.

Webots have provided us a powerful background setting pack, we can find it in 'add-objects-floors' [1]. The size of the floor is 100×30.

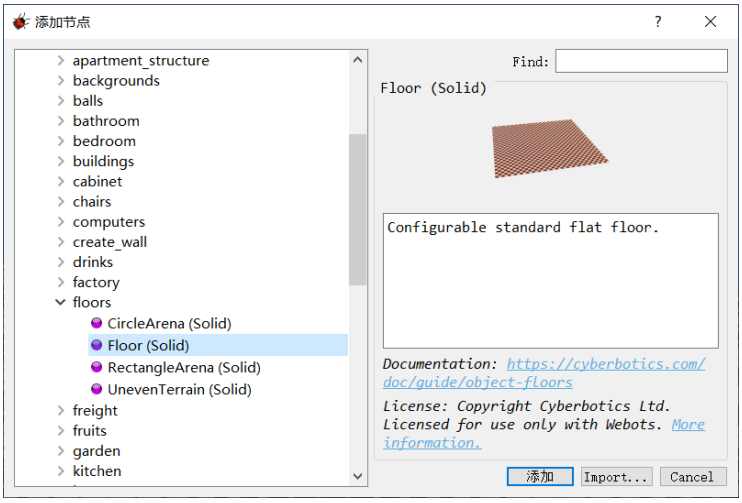


Figure 4 Size of Background

2.1.1.2. Roads

This part will explain how we arranged the roads [2].

In Webots, the roads can be simply arranged in 'add-objects-road'.

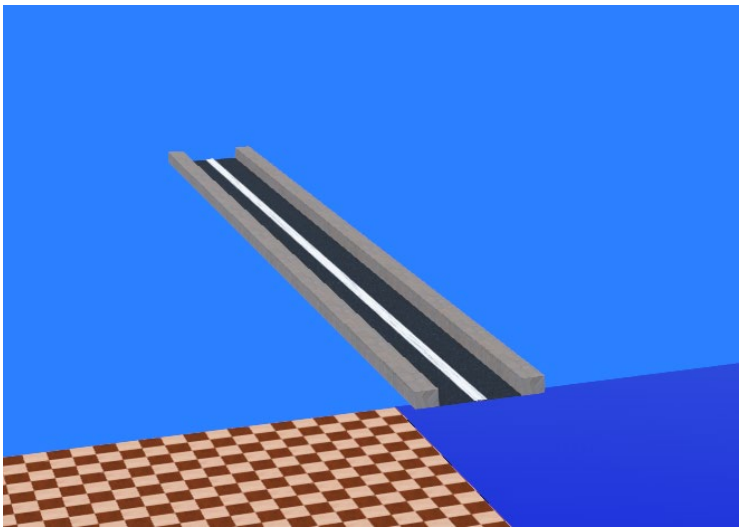


Figure 5 Road

In our design, the parameters of the road are set as

Table 1 Parameters of Roads

Width	0.75
Road border width	0.2
Road border height	0
Road line type	Continuous
White line width	0.15

To build the line with which aims for the car goes along, we find that the

central line can be set as 'continuous', which is shown in the graph. Also, the color of the line can be changed so that in the last task we can meet the requirement.

To make sure the car can successfully move out of the road and release the object, we set the height of the border 0.

2.1.1.3. Objects

This part will explain how we design the markers, signs, and trees

According to the task book, the positions of each marker are shown in follow.

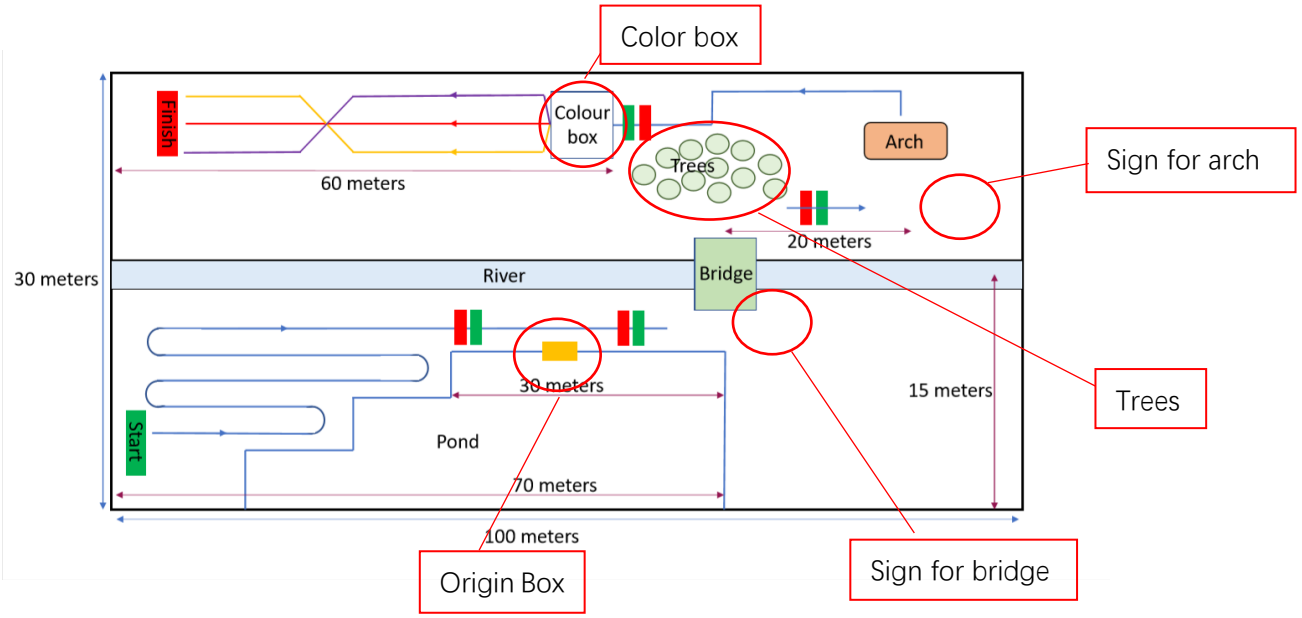


Figure 6 Objects and their Locations

In our design, most of the signs (sign for the arch, releasing, and bridge) and trees are provided by Webots, only the color box is designed by ourselves.

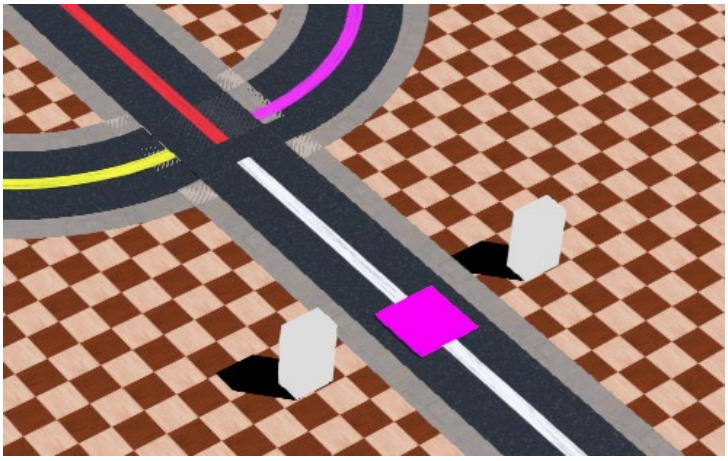
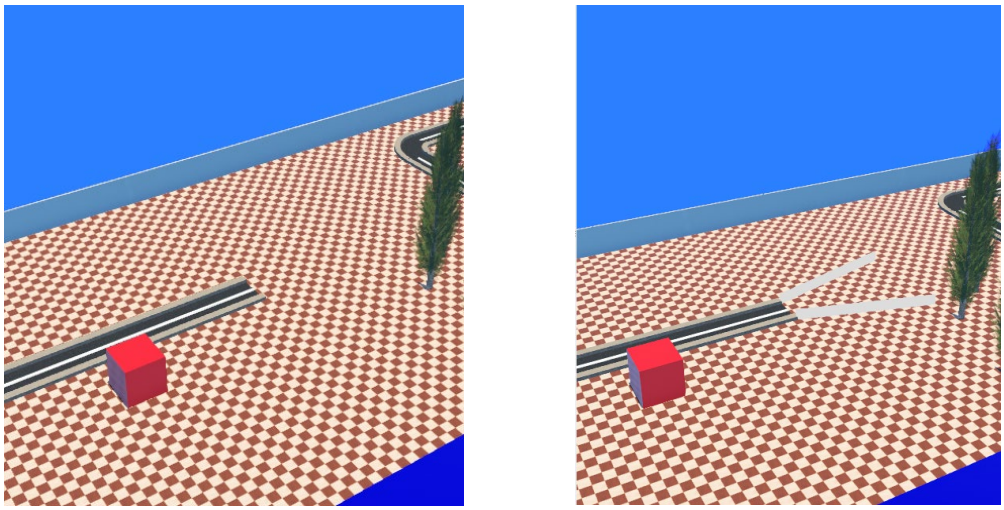


Figure 7 Color Box

We located the color box on the ground with no height, so that car move over it as soon as shot the color[3] with its camera.

2.1.1.4. Redirection Barriers

After crossing the arch and turned left for twice, the car will have to detect the tree again and turn right. However, it is hard to ensure the final direction after turning so that car might not go back to the road. As a result, we add two barriers as a triangle, thus the car can go back to the road by detecting.

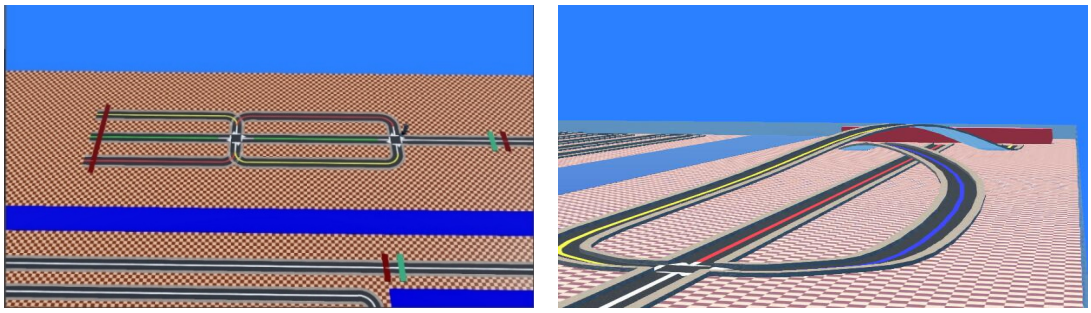


Previous After

Figure 8 Redirection Barriers

2.1.1.5. Overpass

In our teams' previously designed patio, the road for TASK 5 contains an intersection of three different colored roads. It seems that the road is complicated. We adjust the shape of the three roads into an overpass.



Previous After

Figure 9 Overpass

However, we found these designs are not necessary but make the patio out of order. So, we abandoned these in the later test.

2.2. Casting

The first task is to detect the orange box. In this part, we use a radar [4] that has already been done by the group who do the distance & angle recognition. The radar will detect the orange signal and ask the car to control its speed and go approach to the signal. (The code for radar is in the appendix.)

For the second part: Casting, we thought about my methods to finish this task, like catapult and track belt and the robotic arm.

2.2.1. Feasible trial (Bian Qing)

2.2.1.1. Track belt

Since the track belt can deliver the food in any direction, so we planned to use a track belt to deliver the fish food. And because the way out is always the rear of the car, for simplicity, we placed the belt in a horizontal direction instead of at an angle to the car. For testing, we designed a small cube to represent the fish food.

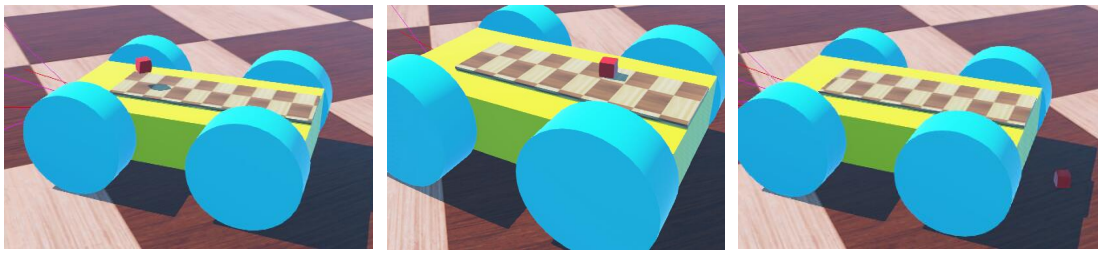
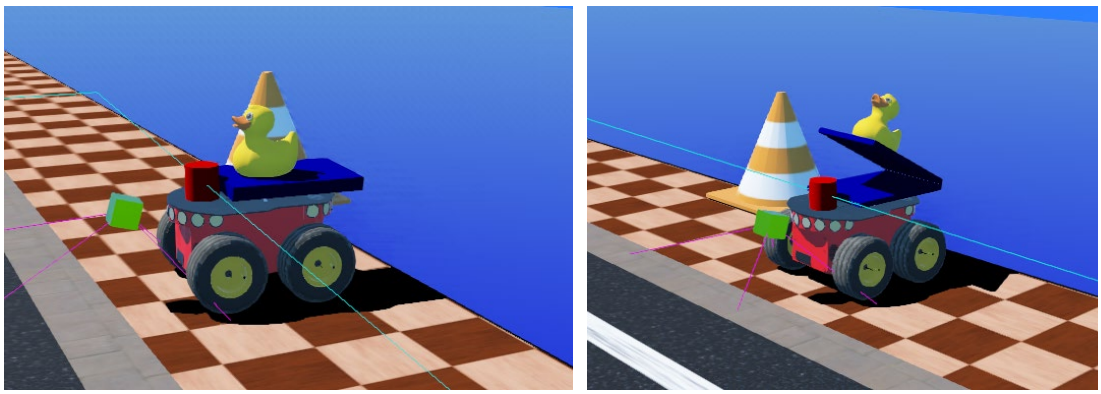


Figure 10 How a Track Belt Works

2.2.1.2. Catapult

The car will be equipped with an ejector that consists of two plates and a rotational motor, which in this case, one edge of each plate is connected by the rotational motor. By using the rotational motor, we can set both the speed and the angle of the rotation. Therefore, the whole equipment can be regarded as a Catapult, being able to emit objects from the plate to a certain distance and in a specific direction by controlling the rotational speed and motor angle respectively.



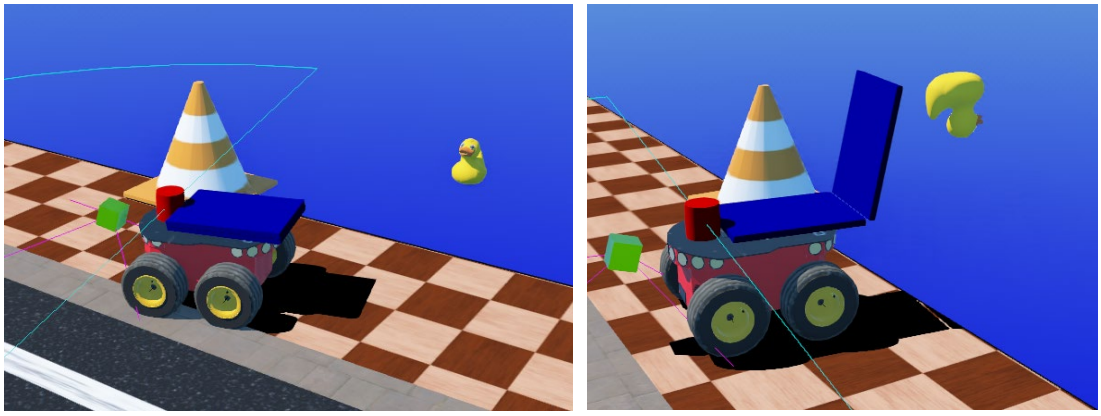


Figure 11 How a Catapult Works

However, we can find that if we use these two methods, we can deliver the fish food easily, but we cannot cast the fish food at a certain place, so the process for testing will be harder. As a result, we decide to choose is using a mechanical arm and put it on the car to deliver the fish food.

2.2.1.3. Robotic arm

We finally decided to use a robotic arm as the task ask us to deliver the fish food into a certain place (the orange box). The robotic arm can stretch into the box and put the fish food into it. This can significantly increase the success rate of casting.

2.2.2. The Robotic Arm(Tang Hao & Bian Qing & Xie Kunyang)

2.2.2.1. Robotic arm choice (Bian Qing)

We firstly wanted to use the ready-made robotic arm in Webots.

	Tinkerbots	Kinematics GMBH	*Robotics kit*
	IPR	Neuronics	*Robotics arm*
	IRB 4600/40	ABB	*Robotics arm*
	PUMA	Unimation	*Robotics arm*
	UR3e, UR5e and UR10e	Universal Robots	*Robotics arm*

Figure 12 Ready-made Robotic Arms

However, we find that these robotic arms neither too small nor too big.

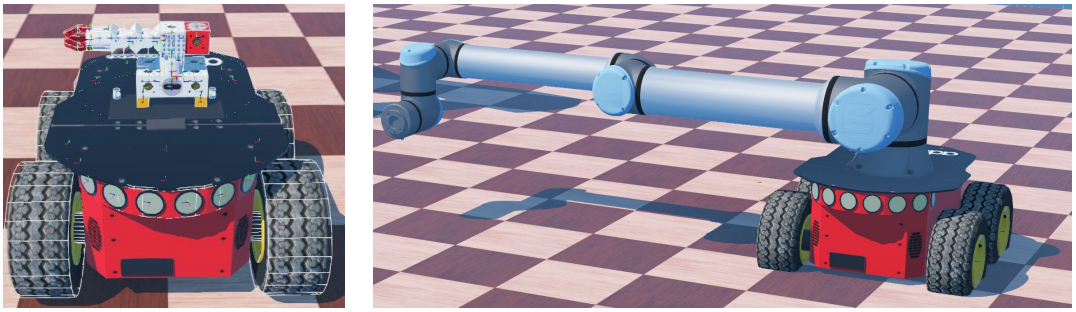


Figure 13 Unfitted Robotic Arms

So, we decided to use the mobile arm on the Youbot [5]. We move out of the arm part from the car and combine it with the car we want.

	youBot	

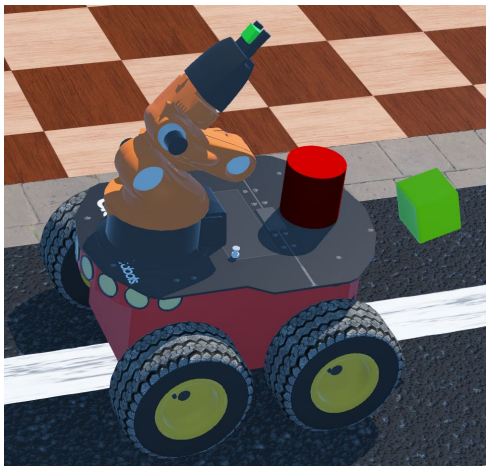


Figure 14 Final Car Module

Here, we have already chosen the robotic arm we want and finished the car construction.

2.2.2.2. Function realization (Tang Hao & Xie Kunyang)

The function of the robotic arm mainly including three parts: Gripper (grip & release), Arm (stretch out & drawback) and the general control function (which control the car to finish the whole task)

2.2.2.2.1. Gripper

Step Function	Control the finger and make it gap as that we desire	
	bound()	gripper_zet_gap()

```
1.  double bound(double v, double a, double b) {
2.    return (v > b) ? b : (v < a) ? a : v;
3.  }
```

```
4.  void gripper_set_gap(double gap) {
5.    double v = bound(0.5 * (gap - OFFSET_WHEN_LOCKED), MIN_POS, MAX_POS);
6.    wb_motor_set_position(fingers[LEFT], v);
7.    wb_motor_set_position(fingers[RIGHT], v);
8.  }
```

Step Function	Initialize the finger part of the arm
	gripper_init()

```
9.  void gripper_init() {
10.   fingers[LEFT] = wb_robot_get_device("finger1");
11.   fingers[RIGHT] = wb_robot_get_device("finger2");
```

Step Function	Control the finger moving (gripping & releasing)	
	gripper_grip()	gripper_release()

```
12. void gripper_grip() {
```

```
  wb_motor_set_position(fingers[LEFT], MIN_POS);
  wb_motor_set_position(fingers[RIGHT], MIN_POS);
}

void gripper_release() {
  wb_motor_set_position(fingers[LEFT], MAX_POS);
  wb_motor_set_position(fingers[RIGHT], MAX_POS);
}
```

2.2.2.2.2 Arm & Preparation for casting

Step	Hold the last state for every motor (Make every part of the arms move separately)
Function	passive_wail()

```
static void step(){
  if (wb_robot_step(TIME_STEP) == -1){
  wb_robot_cleanup();
  exit(EXIT_SUCCESS);
  }
  }

static void passive_wait(double sec){
  double start_time = wb_robot_get_time();
  do{
    step();
  }while(start_time + sec >wb_robot_get_time());
  }
```

Step	Control the arm position	
Function	go()	back()

```
void go(enum Height height1){
   arm_set_height(height1);
}

void back(enum Height height2){
   arm_set_height(height2);
   }
```

Step	Make the car closer to the releasing spot (If we find the car is not close enough to put the object into the box)
Function	Distance sensor [6]

```
WbDeviceTag ds[2];
ds_names[2][10] = {"ds_11", "ds_12"};
```

```
double ds_value[2];
for (int i = 0; i<2; i++){
   //get a handler to the sensor
   ds[i] = wb_robot_get_device(ds_names[i]);
   //perform distance measurment every time step
   wb_distance_sensor_enable(ds[i], TIME_STEP);
   //receive the value from sensors
   ds_value[i] = wb_distance_sensor_get_value(ds[i])
   }
```

Step Function	Judge the process of the task for casting move_seq0[ ]

```
unsigned short move_seq0[10] = {0}
```

2.2.3. Controller code (Tang Hao)

Here is the key code of the control function to show how this part been finished.

Step 1: The car will gradually control its speed and reach the position that is tangent to the orange box by using the lidar module [4].

```
if (move_seq0[0] ==0 ){
   // When radar detects the object, the car will control its speed and go approach to the object
   if (obj_cen.second < func.lidar_max_range){
      Mobile_.speed = 10*(obj_cen.second-0.1);
      // cout<< obj_cen.second<<endl;

   }
   // When the distance satisfies our needs, go to the next process
   if (obj_cen.second - 0.2 < 0.05){
      move_seq0[0] =1;
   }
   return ;
```

Step 2: The car will rotate itself to right and keep moving until it is very close to the box.

```
   // Adjust the front of the car to the right
   Mobile_.dir = -3.1415926;
   if (fabs(imu_val[2] -(-3.1415926))<0.3 && fabs(obj_cen.second - 0.55) < 0.16){
      move_seq0[1] =1;
   }
   return ;
```

Step 3: The car will rotate the left and move back until it is very close to the

box again.

After that, it will stretch out its arm and do casting, and then draw back its arm.

```
// Adjust the back of the car to right to make it convenient to cast
    Mobile_.dir = 0;
    Mobile_.speed = 0;
    if (fabs(imu_val[2] -(0)) < 0.1){
      // move backward
      SetSpeed(-0.5)
      // detect the optimal position for releasing
      if (ds_value[0]<=500 && ds_value[1]<=500){
        SetSpeed(0);
        }
      // Casting
     // After casting, draw back the robotic arm
       go(ARM_FRONT_FLOOR);
       passive_wait(3.0);
       gripper_release();
       passive_wait(4.0);
       back(ARM_RESET);
       passive_wait(2.0);
     if (fabs(wb_position_sensor_get_value(arm_pos[ARM3]) + 2.635) <0.04){

         move_seq0[2] =1;

    }
  }
```

Step 4: At last, it will turn the direction at 45 degrees and move toward until the car continue to line patrol.

```
// Redirect the car motion and move forward
    Mobile_.dir = -1.5708+0.2 ;
    Mobile_.speed = 6.3;
    // Continue line patrol
    if (func.count_ > 500){
      move_seq0[3] =1;
    }
    return ;
    }
```

2.3. Car Design and Control

2.3.1. Car Design (Xie Kunyang & Chen Qianyi & Tang

Hao)

2.3.1.1. Main Body (Xie Kunyang)

We need to choose the main body of the rover. First, we built a car using the basic geometry body, but it was too ugly, and it couldn't climb the bridge, so we change the plan.

Considering that the software's robot library contains many ready-made models, we decided to use the rover provided by the official. The volume of the car should be within the range of 50 * 50 * 50, or it cannot cross the bridge and go through the arch.

Finally, pioneer 3AT [7] is what we want:

Figure 15 Pioneer 3AT

2.3.1.2. Sensors (Chen Qianyi)

In this project, the rover should have the vision to patrol the line, so the camera module is necessary. We process the pictures and make the rover find the correct direction. Besides, the car should be able to avoid obstacles, so the distance sensor is also needed. Pioneer has 16 sonar, but the project has no limit on sensors, so we decided to use a more powerful one, that is, the lidar. It can emitter 512 beams and return 512 distance values, which increase the resolution. Finally, since the car needs to read its current direction and elevation angles, we added an inertia unit.

Camera Inertia Unit Lidar

Figure 16 Sensors

2.3.1.3. Arm (Tang Hao)

The mechanical arm used is based on the prototype of a robot named YouBot [5]. The whole structure of the arm can be simplified as two separate elements: arm body and fingers. For the arm body, it is constructed by using a black cylinder as the base and four orange blocks as the main limbs, which are all connected by different sizes of rotational motors. As for the finger part, it uses a black block as the arm palm and two black bars as the fingers, and the fingers are manipulated by two linear motors inside the black block.

Pioneer 3AT YouBot Our Rover

Figure 17 The Components of our Rover

2.3.2. Control (Xie Kunyang)

2.3.2.1. Speed & Direction Control

First, we need to control the direction and speed as we want, we created a speed structure variable:

```
1.  struct Mobile
2.  {
3.    float dir = pi;
4.    float speed = 6.3;
5.  }Mobile;
```

In this case, we can control the rover's direction and speed by Mobile.dir

and Mobile.speed.

The device data is assigned before each processing, real_distance, real_angle are the values acquired by lidar while line_dir and line_angle are values acquired by the camera.

```
1. Obj_center obstacle = funcs.obj_dis_info();
2. Obj_center lin_imag = funcs.imag_process(imag_val);
3. float real_distance = sqrt(obstacle.obj_x * obstacle.obj_x + obstacle.obj_y
   * obstacle.obj_y);
4. float real_angle = acos(obstacle.obj_x / real_distance) - pi/2;
5. float line_dir = sqrt(lin_imag.obj_x * lin_imag.obj_x + lin_imag.obj_y * lin
   _imag.obj_y);
6. float line_angle = acos(lin_imag.obj_x / line_dir) - pi/2;
7. Mobile.dir = imu_val[2] + line_angle;
8. gripper_grip();
```

Now, we can step in the state machines.

2.3.2.2. PID Controller

A proportional–integral–derivative controller (PID controller) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. In practical terms, it automatically applies an accurate and responsive correction to a control function [8].

In this project, PD control is enough. Here is the code:

```
1. float pd_control(float dis, float k, float d)
2. {
3.   float temp = 0.;
4.   float dis_2 = 0.;
5.   dis_2 = dis - err_yaw_last;
6.   temp = k * dis + d * abs(0-dis_2);
7.   err_yaw_last = dis;
8.   return temp;
9. }
```

2.3.2.3. Motor Control

We define a structure variable to store the speed of the wheels:

```
1. struct Speed
```

```
2.  {
3.    float wheel_l = 0.;
4.    float wheel_r = 0.;
5.  };
```

The two wheels on the left side of the car have the same speed. If the left wheel is backward and the right wheel is forward, then the car will turn left, vice versa. Therefore, when receiving the instruction from Mobile.dir, the wheel can be directly controlled to make the car steering.

In the middle, we also use PID control to make them change smoothly. The variable yaw means the current yaw value while yaw_d means the target value. And then process the difference value in the PID algorithm, and finally we can analyze the statement of the wheel in three conditions, left, straight, and right.

```
1.  Speed motion_keep(double yaw, double yaw_d, float L)
2.  {
3.    Speed temp;
4.    double dis = yaw - yaw_d;
5.    // If dis < 0, turn left, else, turn right
6.    double yaw_w = pd_control(abs(dis), 20., 5.);
7.    double speed_d = val_limit(yaw_w*L, -MAX_SPEED, MAX_SPEED);
8.    if(dis < 0)
9.    {
10.     temp.wheel_l = -speed_d;
11.     temp.wheel_r = speed_d;
12.   }
13.   else if (dis == 0)
14.   {
15.     temp.wheel_l = speed_d;
16.     temp.wheel_r = speed_d;
17.   }
18.   else
19.   {
20.     temp.wheel_l = speed_d;
21.     temp.wheel_r = -speed_d;
22.   }
23.   return temp;
24. }
```

2.3.2.4. Patrolling

We split the entire task into three state machines [9], they are cast_seq[], bridge_seq[], and color_seq[]. They are used to control, patrolling (including casting), bridge crossing, and color recognition processes respectively.

There is a structure variable used to store the target direction point of color recognition. This direction point can control the direction of the vehicle, At the beginning of the task, the car can complete the line patrol by identifying the white line on the road. The whole task is in the state of line patrol by default, and the special state is programmed only when a specific signal is triggered. In this task, we controlled the direction of the car by the PID controller.

```
// Start
  if(cast_seq[0] == 0)
  {
    Mobile.speed = 6.3;
    if(obstacle.obj_y < funcs.lidar_max_range)
      Mobile.speed = 10*(obstacle.obj_y - 0.9);
    if(obstacle.obj_y - 0.9 < 0.05)
      cast_seq[0] = 1;  // State over, to another state
  }
  else if(cast_seq[1] == 0)
  {
    Mobile.dir = south;
    if(fabs(imu_val[2] + pi < 0.2))
      Mobile.speed = 25*(obstacle.obj_y - 0.32);
    if(fabs(obstacle.obj_y - 0.32) < 0.03)
      cast_seq[1] = 1;
  }
```

2.3.2.5. Casting

In the procedure of casting, when the orange box is identified by lidar, the rover will down and turn around, then start casting with the mechanical arm when its back towards the orange box. After casting, the direction will be set to pi /4 to go back to the road. At the time when the camera detects the road, it will return to the right track until the white line cannot be found. In this case, cast_seq[] is over.

```
  else if(cast_seq[3] == 0)
```

```
2.    {
3.      Mobile.dir = east + pi/8;
4.      Mobile.speed = 6.3;
5.      if (funcs.count > 500)
6.        cast_seq[3] = 1;
7.    }
8.    else if(cast_seq[4] == 0)
9.    {
10.     Mobile.dir = imu_val[2] + line_angle;
11.     if(funcs.count < 40)
12.     {
13.       Mobile.dir = east;
14.       cast_seq[4] = 1;
15.     }
16.   }
```

2.3.2.6. Bridge & Arch

The process of crossing the bridge is simple. When lidar detects obstacles and turns the rover around. Here, if the car is not directly aimed at the obstacle, as we know the angle between the center of the obstacle and the center of the car, we can first let the car face the obstacle, which avoids the car to a wrong direction this is useful when going through the arch.

```
1.  // Before the arch
2.  else if(bridge_seq[6] == 0)
3.  {
4.    Mobile.dir = north + real_angle;  // Correct direction
5.    Mobile.speed = 5;
6.    // Have Detected the white line
7.    if(funcs.count > 100)
8.    {
9.      Mobile.speed = 6.3;
10.     bridge_seq[6] = 1;
11.   }
12. }
```

Besides, we need to control the speed of the rover when it is crossing the bridge, or it may be upside down.

2.3.2.7. Color Recognition

The last part is color recognition, the condition for entering this state is

that the rover is oriented to the west and there is an obstacle. Three basic colors of each pixel in a frame of the image are calculated respectively. If the specific color is larger than other colors, set the color_trace as the patrolling color. The rover will patrol according to the color_trace. In this way, the basic logical flow is implemented.

```
1.   else if(color_seq[0] == 0)
2.   {
3.     if(real_distance < 1 && abs(imu_val[2] - west) < 0.05)
4.       color_seq[0] = 1;
5.   }
6.   else if(color_seq[1] == 0)
7.   {
8.     Mobile.speed = 5;
9.     if(funcs.c_r > funcs.c_y and funcs.c_r > funcs.c_p)
10.    {
11.      funcs.color_trace = RED;
12.      cout << "red" <<endl;
13.    }
14.    else if(funcs.c_p > funcs.c_y and funcs.c_p > funcs.c_r)
15.    {
16.      funcs.color_trace = PURPLE;
17.      cout << "purple" <<endl;
18.    }
19.    else
20.    {
21.      funcs.color_trace = YELLOW;
22.      cout << "yellow" <<endl;
23.    }
24.    color_seq[1] = 1;
25.  }
```

2.4. Distance Recognition

2.4.1. The application of IMU Module (Li zhaoyu & Yin Xiangyu)

The IMU node computes and returns the roll, pitch, and yaw angles of the robot concerning to the global coordinate system defined in the WorldInfo node [10]. These parameters it provides are important in controlling and tracking the movements (turn left, turn right, keep going straight) of the robot.

As the figures below, they show that the Yaw and Pitch angles of the robot concerning to our global coordinate that the angle points to the bridge is 0 rad, and the angle points to the first beacon is -1.57 rad.

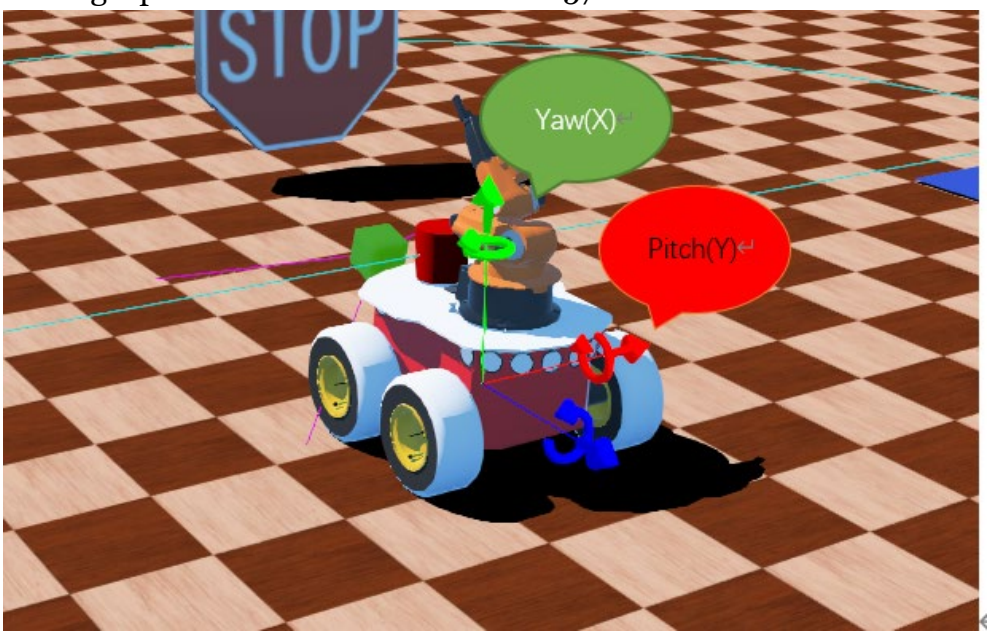


Figure 18 IMU on the car

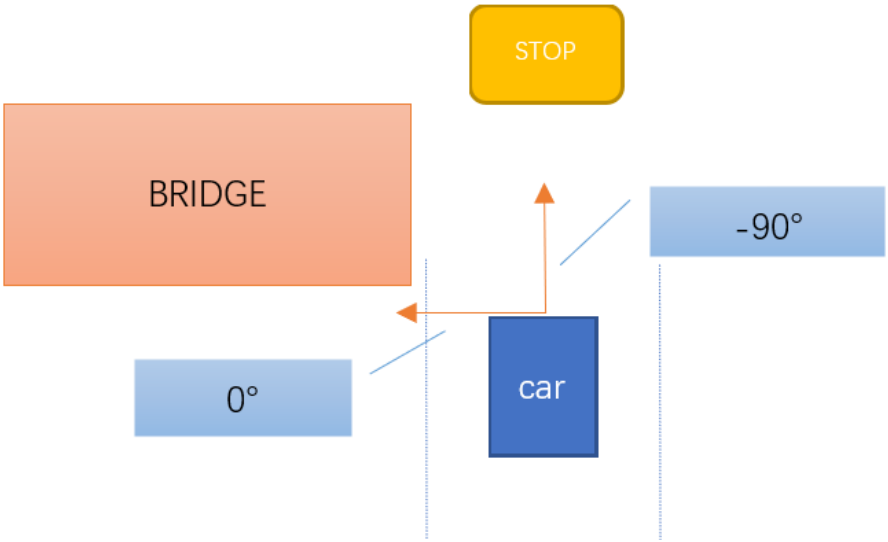


Figure 19 IMU on the car

2.4.2. The application of Lidar Module (Li Zhaoyu & Yin Xiangyu)

To obtain the relative distance and relative angle between the detected target and the car, we decide to use the lidar module [10].

It is the key node for the robot to recognize the objects. The basic principle

is to emit the laser and get the point cloud data so that it achieves the recognition of the objects [11]. By adjusting some relative parameters like the number of layers and lines, the effects will be different.

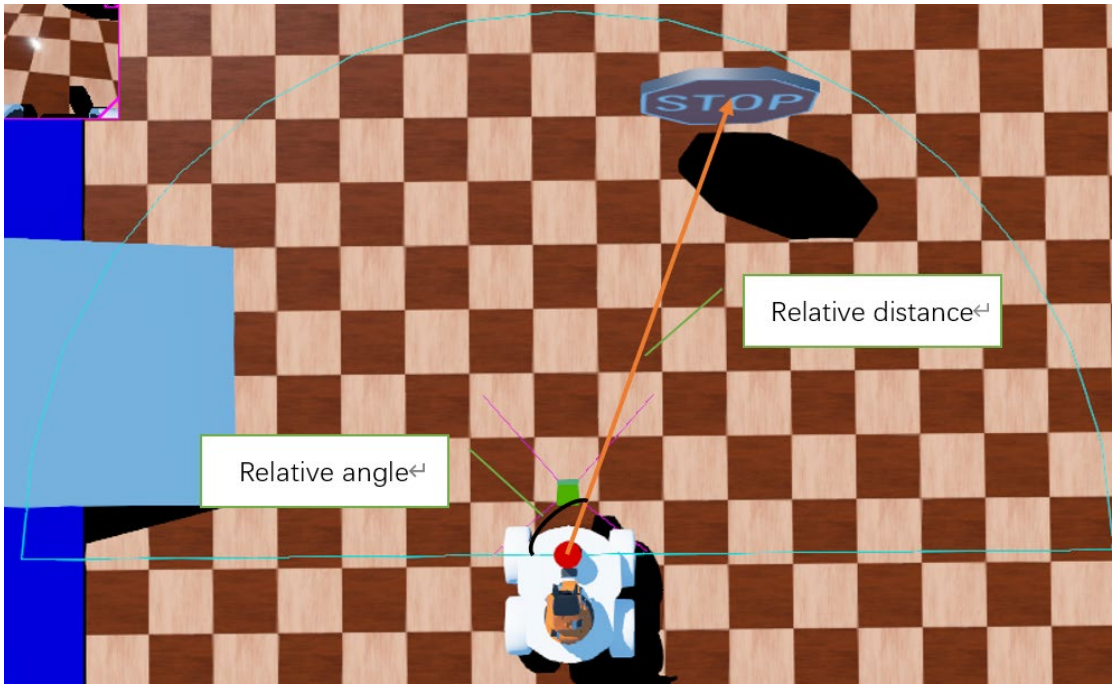


Figure 20 The lidar module on the car

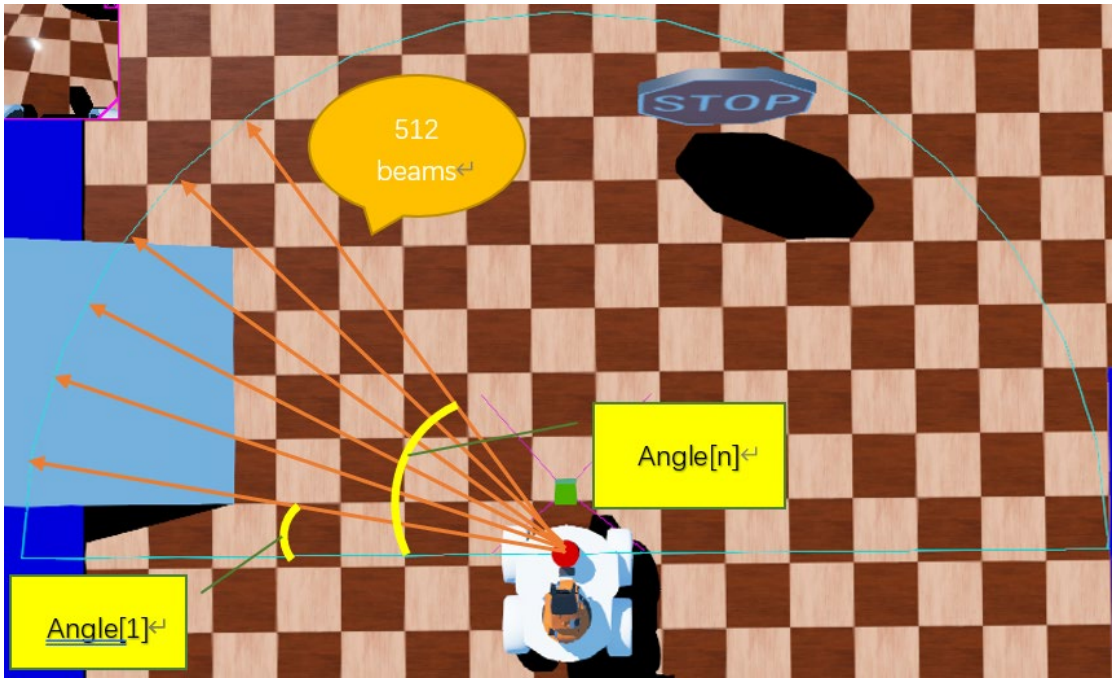


Figure 21 Analytical graph of lidar module

For the lidar module, we have 512 beams, for each beam, we can get a distance value stored in lidar_val[n], n indicates the nth beam.

$$\text{Angle}[1]=\frac{\pi*1}{512},\ \text{Angle}[n]=\frac{\pi*n}{512}$$

Relevant Code:

```
1.  //Transfer the lidar value into coordinate points
2.      void obj_distance_get(const float * lidar_val,double ang_range,
3.       double z_bias, double x_bias){
4.
5.          double piece_ = ang_range / lidar_num;//pi/512
6.
7.          for ( int i=0;i <lidar_num;i++ ){
8.              //transfer to x,y-axis
9.              double ang_ = ang_range + ( pi_ - ang_range) /2 - piece_*i;//lid
    ar_val i=0->pi
10.              lidar_dcrx[i] = (float)(lidar_val[i] * cos(ang_) + x_bias);
11.              lidar_dcry[i] = (float)(lidar_val[i] * sin(ang_) + z_bias);
12.
13.              //for undetected points, we define it as invalid
14.              if (lidar_val[i]/lidar_max_range>0.95){ //max_range=2m
15.                  laser_effective[i] = 0;
16.              }else {
17.                  laser_effective[i] = 1;
18.              }
19.          }
20.
21.      }
```

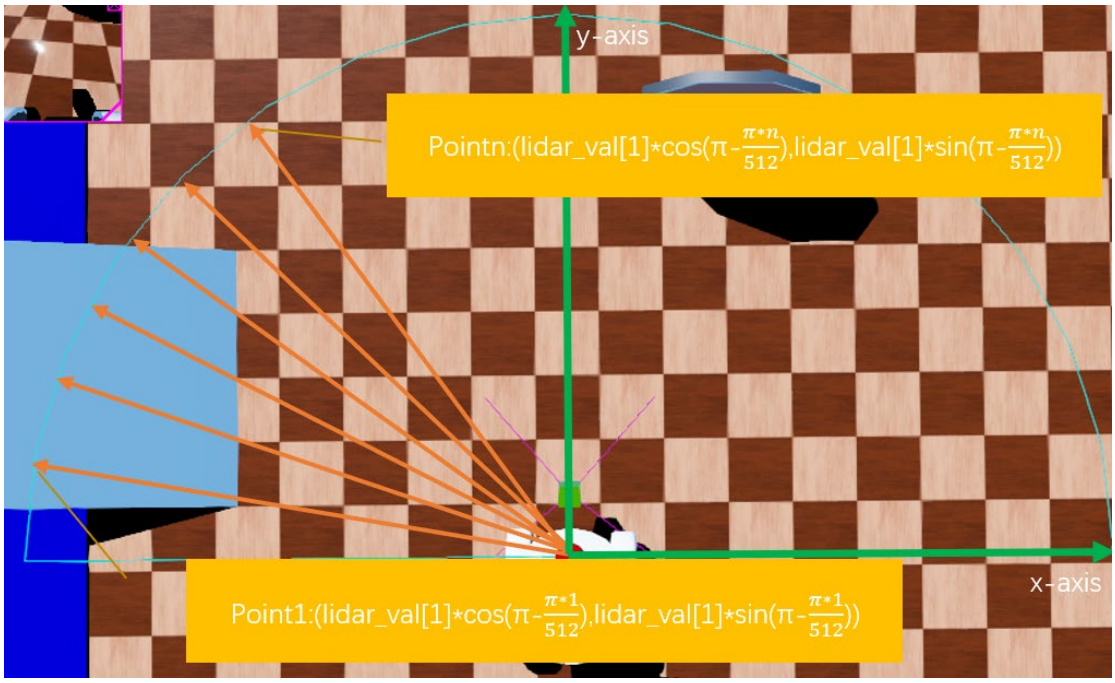


Figure 22 The process of getting the middle point

For the first step, we need to transfer the returned value of lidar into coordinate points.

As the figure above, we know that Point n:

$$(\text{lidar\_val}[1]*\cos(\pi-\frac{\pi*n}{512}),\text{lidar\_val}[1]*\sin(\pi-\frac{\pi*n}{512}))$$

The second step is to obtain the middle point of the target we detected since we already know the coordinate points of 512 detected positions. We can simply calculate the x-center and y-center:

$$obj\_cen.first = \frac{\sum lidar\_dcrx}{count}$$

$$obj\_cen.second = \frac{\sum lidar\_dcry}{count}$$

obj_cen.first represents the x-center and obj_cen.second represents the y-center.

Relevant Code:

```
1.  //obtain the middle point of the detected object,
2.      //returned value is X-middle and Y-middle
3.      pair<double, double> obj_dis_info(void){
4.
5.        pair<double, double> temp_;//form parameter
```

```
6.        double temp_x = 0., temp_y = 0.;
7.        short count = 0;
8.        for ( int i=0;i <lidar_num;i++ ){  //i=0 to 511
9.            if (laser_effective[i] ==1){

10.               // cout <<func.lidar_dcry[i] <<endl;

11.               temp_x+=lidar_dcrx[i];

12.               temp_y+=lidar_dcry[i];

13.               count+=1;
14.           }
15.       }
16.       //If no target is detected, default:(0,3)
17.       if (count ==0){ temp_x = 0.;temp_y = 3.;count=1;}
18.       temp_.first = temp_x/count; //x center of inertia
19.       temp_.second = temp_y/count;//y center of inertia
20.       return temp_;
21.   }
```

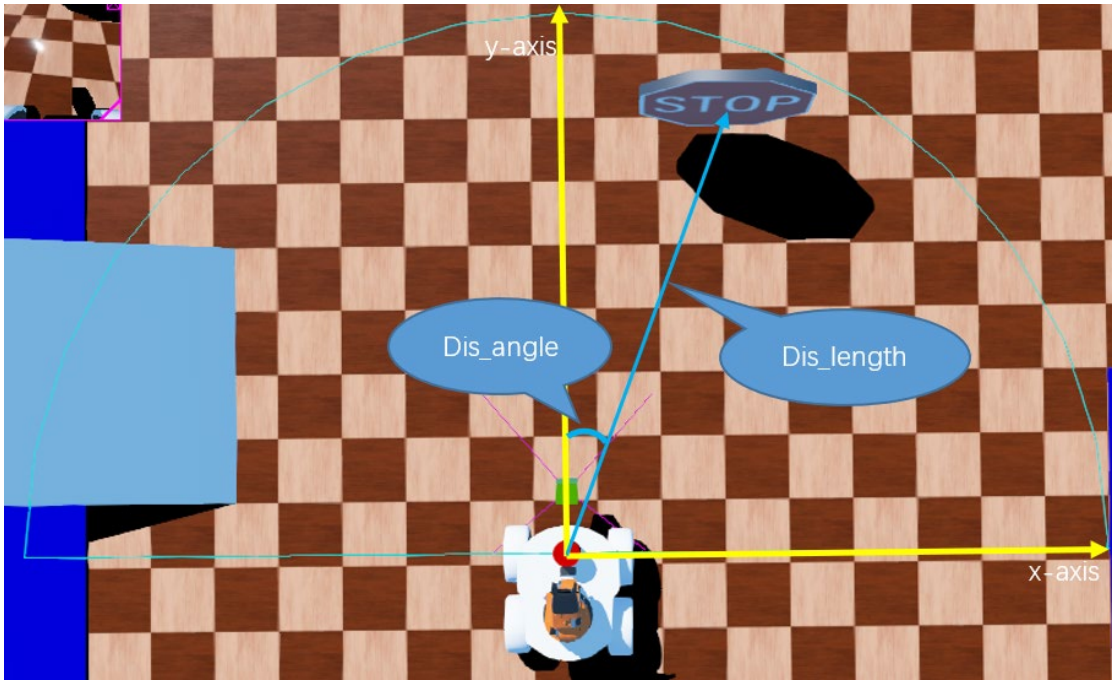


Figure 23 Notes

dis_length: The distance between the center of the obstacle and the center of the vehicle:

$$dis\_len = \sqrt{obj\_cen.first^2 + obj\_cen.second^2}$$

dis_angle: The Angle of the target point relative to the body center:

$$dis\_ang = \cos^{-1}\left(\frac{obj\_cen.first}{dis\_len}\right) - \frac{\pi}{2}$$

So that we can send the correct information of angle and distance to the car motor.

2.4.3. The combination with the car motor (Li Zhaoyu & Yin Xiangyu & Xie Kunyang)

According to the application of IMU[12] and lidar, we can send a precise current moving angle and the difference between the current angle and target angle to the operator of the motor, which can operate the car to get rid of the obstacle correctly.

The imu_val values are used to judge the conditions of the robot. By comparing that with some specific values we set, we can get the states of the movement of the robot. What's more, we set a flag value move_seq to represent different movement states(turn left, turn right, keep straight, cross the bridge) so that we can ensure the robot will implement specific movement according to states. And the Mobile_.dir will control the direction of the robot.

Relevant Code (Detect and cross the Bridge):

```
1.  //keep the car move forward yaw=-1.5708
2.      if (move_seq[0] ==0){
3.          Mobile_.dir = -1.5708 + ang_k*dis_ang;
4.          if (obj_cen.second < 1.1){//the distance in y-axis
5.              move_seq[0] =1;
6.          }
7.
8.      }else if (move_seq[1] ==0){
9.          //the bridge detection
10.         if (imu_val[0] > 0.5){
11.           move_seq[2] =1;
12.         }else if (imu_val[0] < 0.5){
13.           move_seq[3] =1;
14.         }
15.         //set yaw to 0, make the car turn right
16.         if (fabs(imu_val[2] - 0) > 0.05 and !one_taken[0] ){
17.             Mobile_.dir = 0;
18.         }else {
19.         //move forward to the mid-position of the bridge
20.             one_taken[0]=true;
```

```
            Mobile_.dir = 0 +ang_k*dis_ang;
        }
```

2.5. Image Recognition

2.5.1. Color recognition (Wenyu Li)

2.5.1.1. Task Analysis

The color recognition is mainly used in test 5 where our robot car is required to identify the color in the color box and follow the route of the same color before arriving at the finishing line. In our design, the color box shows red yellow or purple at one time. Therefore, our program should at least recognize four colors (including white, which is the color of the normal route) to complete the whole task successfully.

2.5.1.2. Method and Code development

In the digital systems, color can be defined within different color models. Among those color models, RGB and HSV are two commonly used ones. RGB defines one color by the amount of red blue and green lights contained in it while HSV distinguishes one color by its hue, saturation, and lightness. A wide range of colors can be defined in these two models by a 3-number array that represents three parameters. In the practical situation, HSV is commonly used in analyzing the color captured in the physical world as the effect of light is considered. In our project, since we do not need to consider the light in the simulation environment, we directly use RGB to describe the color in the color box. The following table shows some common colors and their RGB representations.

Table 2 RGB of Colors

Color	HTML / CSS Name	Hex Code #RRGGBB	Decimal Code (R,G,B)
	Black	#000000	(0,0,0)
	White	#FFFFFF	(255,255,255)
	Red	#FF0000	(255,0,0)
	Lime	#00FF00	(0,255,0)
	Blue	#0000FF	(0,0,255)
	Yellow	#FFFF00	(255,255,0)
	Cyan / Aqua	#00FFFF	(0,255,255)
	Magenta / Fuchsia	#FF00FF	(255,0,255)
	Silver	#C0C0C0	(192,192,192)
	Gray	#808080	(128,128,128)
	Maroon	#800000	(128,0,0)
	Olive	#808000	(128,128,0)
	Green	#008000	(0,128,0)
	Purple	#800080	(128,0,128)
	Teal	#008080	(0,128,128)
	Navy	#000080	(0,0,128)

To complete the color recognition task, we plan to build two main functions. One to define red purple yellow and white using RGB color mode, another to take the image input and distinguish the main color of the image by returning a color output. As the second function is built based on the first one, we can define the function color_judge as follows:

```
1.  bool color_judge( vector<unsigned char> col, short id){
2.
3.          short max = 190, min_ = 90;
4.
5.          if (id == WHITE){
6.              if (col[0] > max && col[1] > max && col[2] > max){return true;
    }
7.              else {return false;}
8.          }
9.          else if (id == RED){
```

```
            if (col[0] > max && col[1]<  min_ && col[2] < min_){return true;}
            else {return false;}
        }
        else if (id == YELLOW){
            if (col[0] >  max && col[1]>  max && col[2] < min_){return true;}
            else {return false;}
        }else if (id == PRU){
            if (col[0] < min_ && col[1]< min_ && col[2] >  max){return true;}
            else {return false;}
        }
        return false;
    }
```

This function takes two inputs. *col* is the RGB representation of one color and *id* represents the color to be judged. If the color of the *col* is the same as *id*, color_judge returns TRUE. Otherwise, it returns FALSE. The judging criterion is based on the table shown above.

Next, we use function process_img to connect the image input from the camera with the colors defined in color_judge. In process_img, there are three key parameters, namely, c_r c_y and c_b. Each of them represents the number of pixels taken by red, yellow, and purple respectively. Therefore, in the main function, we can determine the main color of the image, thus distinguishing the color in the color box. Also, another important step is to use the gaussian filter to denoise the image by taking the average of nearby.

```
pair<double, double> process_img(const unsigned char * img){
        pair<double, double> temp__;

          img_temp = img_copy(img);

          mat temp_ = gussian_handler(img_temp );

          int r_add =0,c_add=0;
          count_ =0;
          c_r =0,c_y =0,c_b = 0;
          for (short r = 1; r < img_w-1 ;r+=1){
            for (short c = 1; c < img_h-1 ;c+=1){

                  if ( color_judge(temp_[r][c],WHITE) or color_judge(temp_[r][c],color_trace))
```

```
                    {
                        r_add+=r;
                        c_add+=c;
                        count_+=1;
                    }

                    //judge the color in the box
                    if (color_judge(temp_[r][c],RED)){
                        c_r +=1;
                    }else if (color_judge(temp_[r][c],YELLOW)){
                        c_y +=1;
                    }else if(color_judge(temp_[r][c],PRU)){
                        c_b +=1;
                    }
                    //test red yellow and purple. If the color is the same,
  //corresponding parameter plus one

              }
            }

    }
```

Then, the two functions defined above can be used into the main function to control the robot car in task 5.

```
//judge color and trace the corresponding line
            if (func.c_r > func.c_y and func.c_r > func.c_b){
                func.color_trace = RED;
                cout << "red"<<endl;
            }else if (func.c_b > func.c_y and func.c_b > func.c_r){
                func.color_trace = PRU;
                cout << "purple"<<endl;
            }else {
                func.color_trace = YELLOW;
                cout << "yellow"<<endl;
            }
```

2.5.2. Line Tracing (Wang Jiapeng)

2.5.2.1. the principle

we know that our camera [13] can catch the pictures and videos which consist of pictures of each frame. So for our robot, when the camera catches a picture, it would iterate every pixel point. As we know, each pixel point contains RGB values, which decide the color of the pixel. So we turn it to a three-

dimensional vector that our computer can understand and process.

2.5.2.2. Algorithm

About the line tracing algorithm [14], we set an XY-coordinate system, the computer needs to find the reference point which belongs to one of the pixels, if there is no reference point, we would choose the point (0, 0.5* height)

```
float y =0, x=0;
            //if there is no reference point, set x=0,y=0.5*height
            if (count_==0){
                nothing_get = true;
                count_ =1;
                x=0;
                y=0.5*height;
            }else {
            //get the reference point
                nothing_get = false;
                y = height - r_add / count_;
                x = c_add / count_ - 0.5* width;
            }
```

The next step is getting the length and angle of the line according to the point, the length of the line equal to $\sqrt{x^2 + y^2}$, the angel equals to arccos （x/the distance of length.

```
//Image processing to obtain basis points for line inspection
 pair<double , double>obj_img = func.process_img(img);
 //Get the length of the line according to the point
 double dis_len = sqrt(obj_img.first *obj_img.first +  obj_img.second *obj_i
mg.second);
 double ang_temp = (acos(obj_img.first /dis_len) - pi_/2);
 //Gain angel
 double dis_ang =  2.5* ang_temp ;
```

So the robot can measure the length and adjust the angel every frame when moving.

2.5.2.3. Some important functions

process_img: this function is used to transfer the picture that the camera caught copy to a temp file in the memory, after doing some preprocessing like Gaussian filtering and iterate all pixels in this picture.

```
pair<double, double> process_img(const unsigned char * img){
pair<double, double> temp__;
            //copy the picture to the temp
img_temp = img_copy(img);
            //gaussian handler
            mat temp_ = gussian_handler(img_temp );

            int r_add =0,c_add=0;
            count_ =0;
            c_r =0,c_y =0,c_b = 0;
            for (short r = 1; r < img_w-1 ;r+=1){
              for (short c = 1; c < img_h-1 ;c+=1){
                        r_add+=r;
                        c_add+=c;
                        count_+=1;
                    }
    }
}
```

img_copy: this function is used to get the width and height from the picture, and let it to read every pixel in this picture one by one.

```
mat img_copy(const unsigned char * img){

          mat img_temp_  = img_def(width,height);

          int  num_pix = img_w * img_h ;
          const unsigned char *img_ = img;
          int temp_r = 0, temp_c = 0;

          for (int i=0 ; i< num_pix ; i +=1, img_ +=4){

              temp_c = i % img_w;
              temp_r = (int)i / img_w;

              img_temp_[temp_r][temp_c][2] = (unsigned char )*img_;
              img_temp_[temp_r][temp_c][1] = (unsigned char )*(img_ +1);
              img_temp_[temp_r][temp_c][0] = (unsigned char )*(img_+2);

          }

          return img_temp_;
```

```
21. }
```

gussion_handler: This function is used to do the Gaussian filtering [15], which is the process of reducing and averaging the entire image. The specific o peration of Gaussian conversion for each point value is: using a template (or c onvolution, alignment) nominal average gray value Replace the value of the po int in the center of the template.

```
1.  mat gaussian(mat imag)
2.  {
3.    mat temp = imag_size(width,height);
4.    for (short r = 1; r < width-1; r += 1)
5.    {
6.      for (short c = 1; c < height-1; c += 1)
7.      {
8.        for (short p = 0; p < 3; p += 1)
9.        {
10.         float add= 0;
11.         for (short i = 0; i < 3; i += 1)
12.         {
13.           for (short j = 0; j < 3; j += 1)
14.             add += (float)imag[r-1 +i][c-1 + j][p] * gaus[i][j];
15.         }
16.           temp[r][c][p] = (unsigned char) add;
17.       }
18.     }
19.   }
20.   return temp;
21. }
```

3. Results

3.1. Graph Design

The final graph is shown in follow:

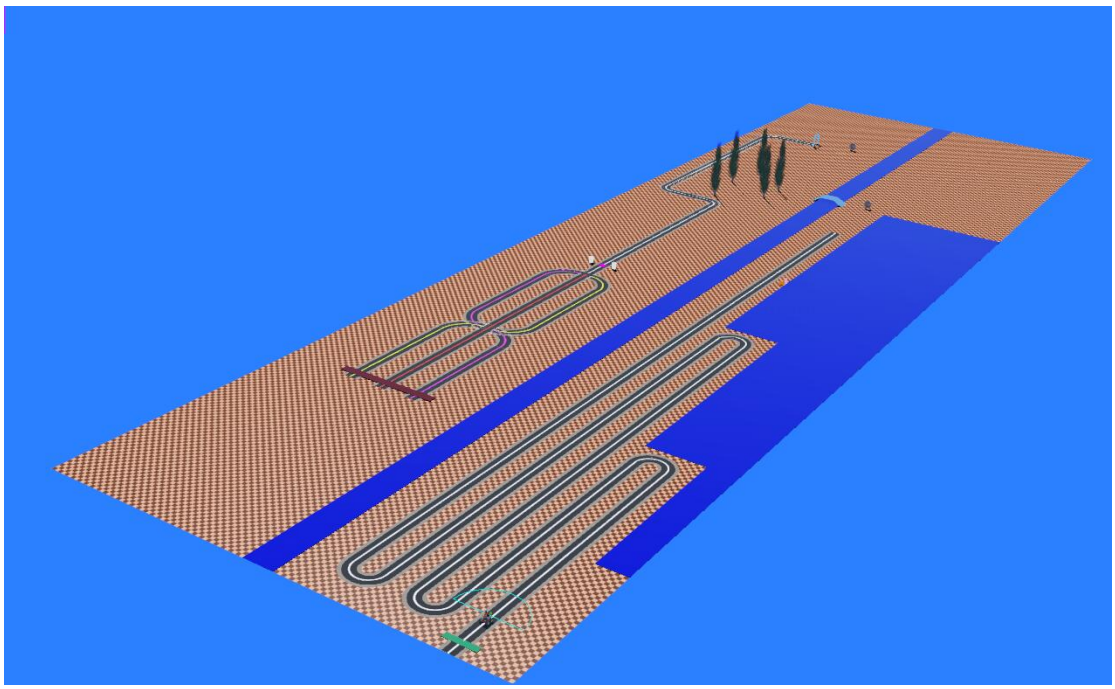


Figure 24 Whole Patio

As can be seen in the graph, the car can set up from the bottom on the right side, and cross the line on the left side.

During designing, we kept the patio simple and clear as possible, and the detailed design is shown in follow.

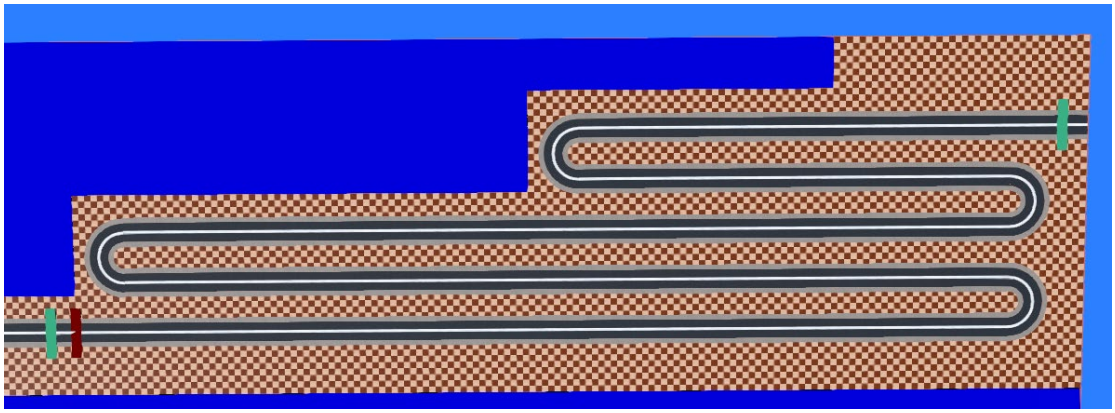


Figure 25 Task 1

In task 1, the car should go along the white line in the middle of the road and turn to make turns for 4 times.

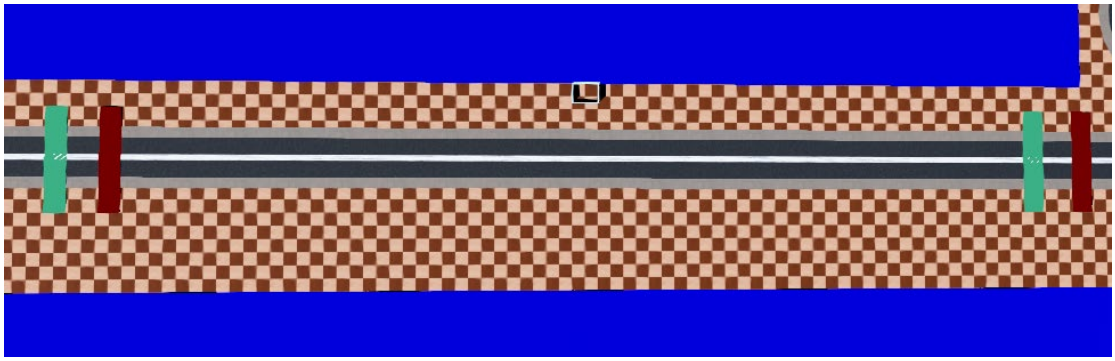


Figure 26 Task 2

In task 2, the car should release food when detected by the object beside the river. To make sure the car can freely go out of the road, the heights of all the components are 0.

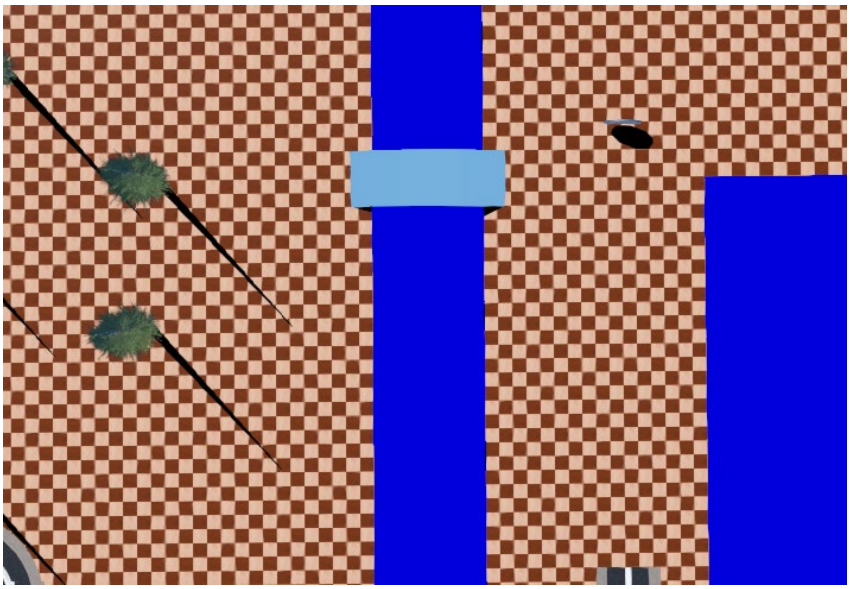


Figure 27 Task 3

The bridge is 100 cm wide and 3 meters long, which includes the ramps used to roll up and off the bridge.

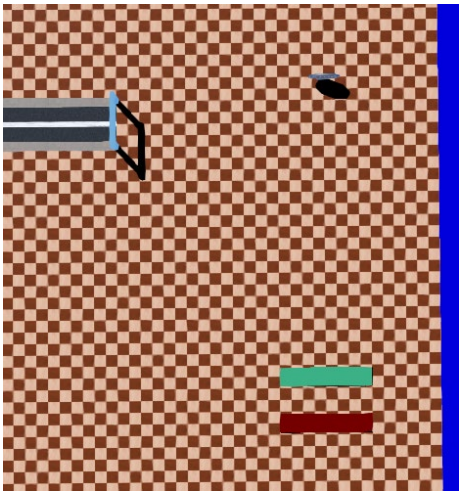


Figure 28 Task 4

The arch is simply built by three solid boxes, whose size is exactly a little bigger than the car.

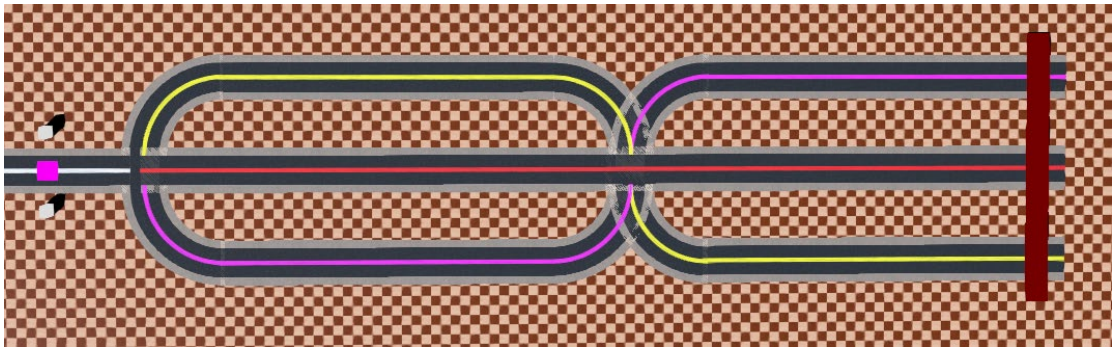


Figure 29 Task 5

We find the intercrosses do not affect the final line patrolling, so the simple design can be feasible.

3.2. Casting

Step1: The orange object comes into the range of the detection range of the radar. (The range is shown as a semicircle.)

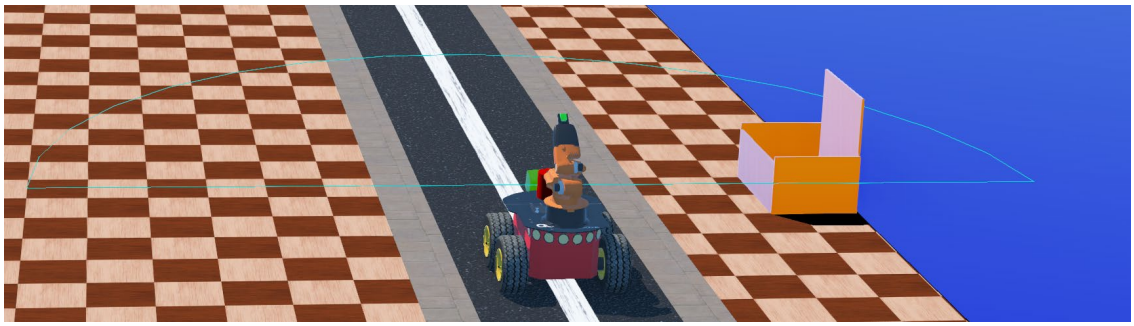


Figure 30 Step 1

Step2: The car changes its direction and moves towards the orange object.

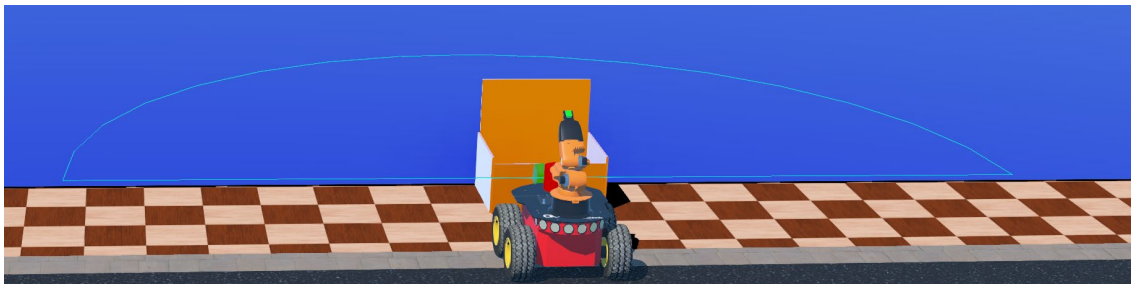


Figure 31 Step 2

Step3: The car turns back, stretches its robotic arm, and releases the objects into the pond.

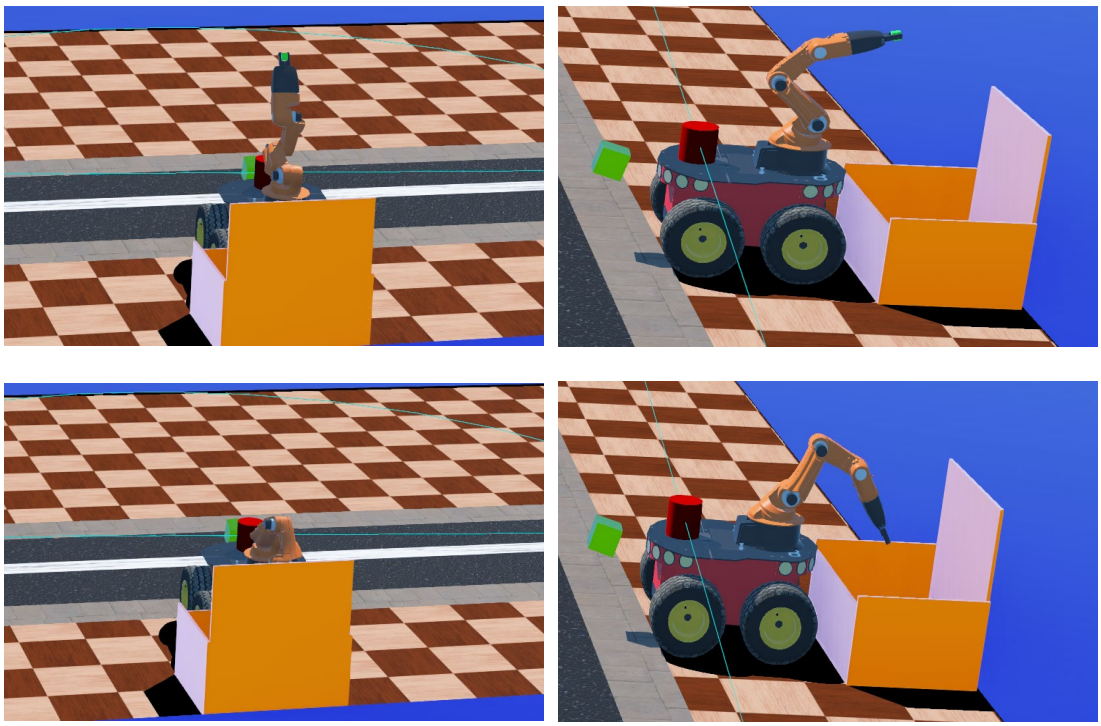


Figure 32 Step 3

Step4: The car draws back its robotic arm and goes back to find the line.

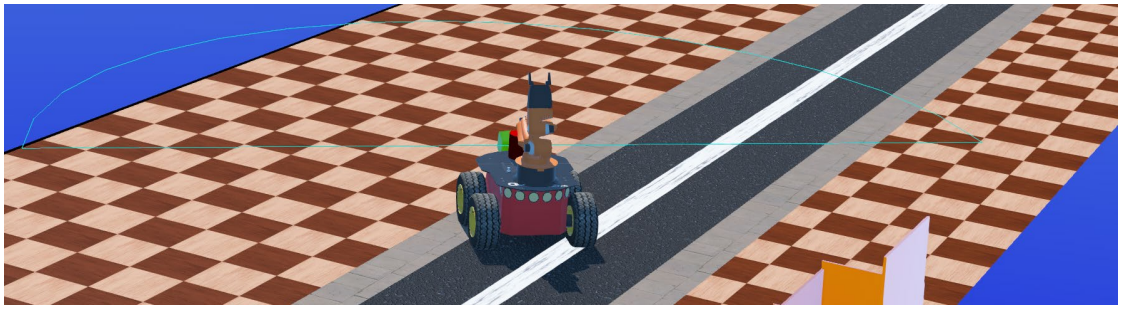


Figure 33 Step 4

Step5: The car goes along the line again, and task 2 has been finished.

Figure 34 Step 5

The total process is shown below:

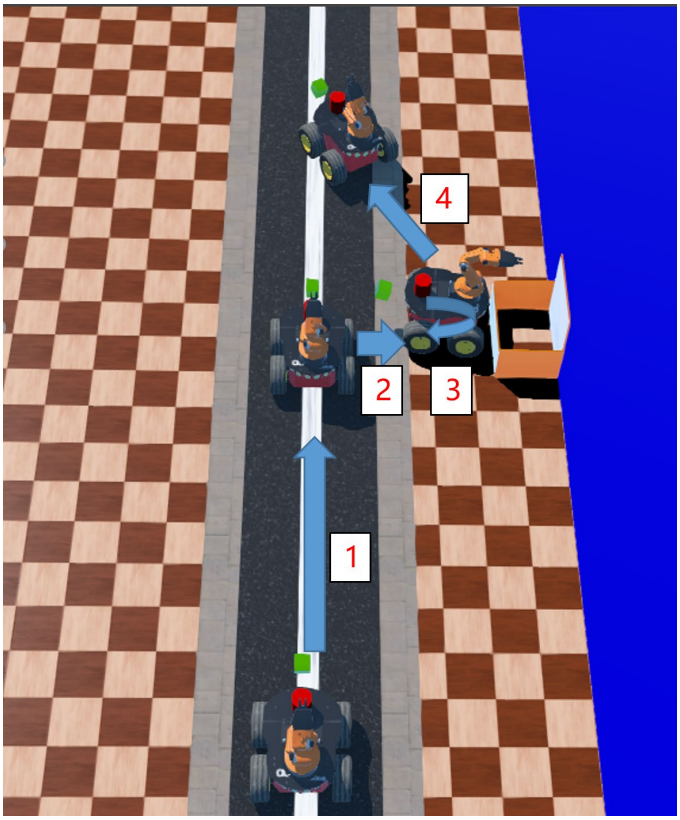


Figure 35 Total

3.3. Car Design and Control

3.3.1. Car Design

Figure 36 Car Design

We made a car within the size of requirement and wrote its main task flow code. As a result, the car can travel at the required speed and direction. Then we linked the achievements of other groups together

Finally, the car contains all the functions needed to complete 5 tasks.

3.3.2. Combination with Other Groups

We merged vision, distance recognition, casting parts into the robot so that the car becomes a whole system. The final result is shown as follows.

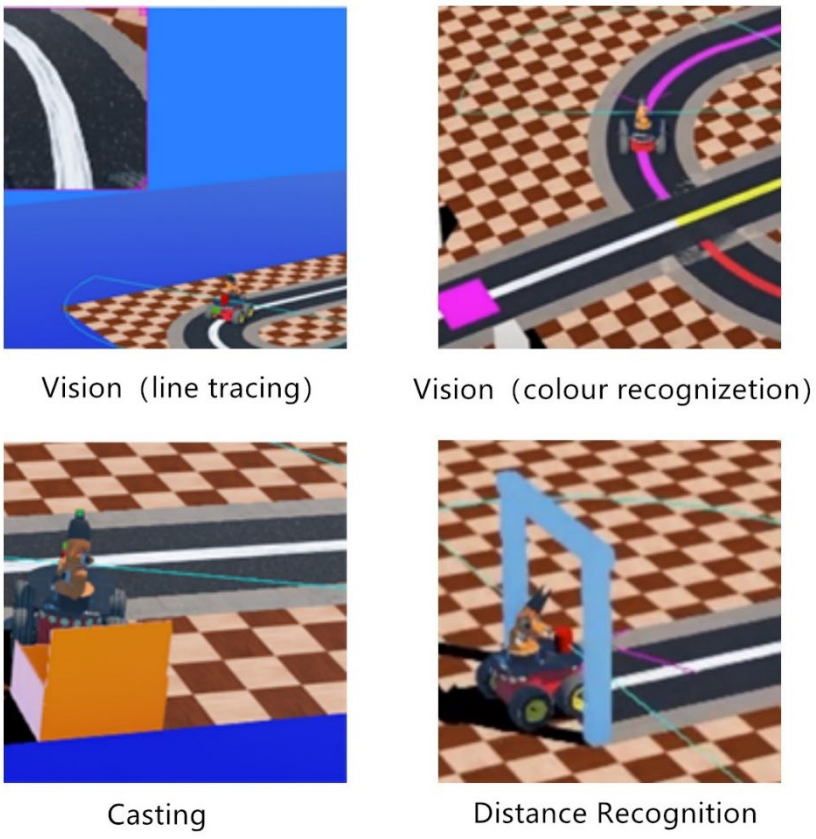


Figure 37 Results

3.4. Distance Recognition

3.4.1. Results

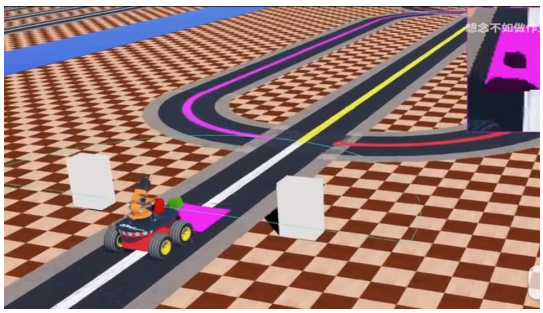
Through simulation, the robot went across the bridge and the arch without any mistake, it completed the mission perfectly. To be more detailed, the robot successfully makes a turn at a suitable position that we set in the controller and go straight forward after turning without a line to follow. Besides, the whole process was fluent which means the judgments, control, and adjustments of different movement states in the codes are logical and clear. The results showed that the robot can recognize the objects, make a turn to the correct direction at a suitable position, and keep going straight without a line on the ground.

3.4.2. Discussions

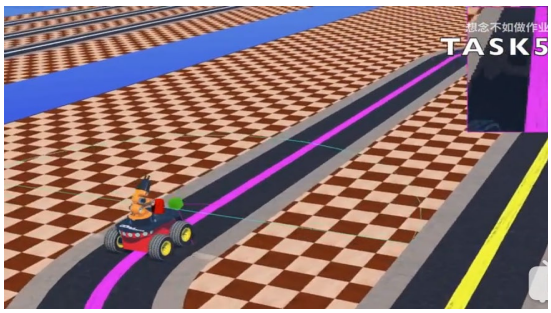
Although the robot can cross the bridge and the arch, it can not be perfectly cross the middle line of them. To solve this, we can adjust the value of the recognition distance and the changing rate of the angle when the robot is turning.

3.5. Image Recognition

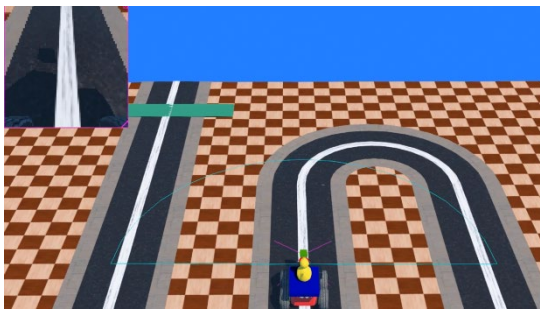
By implementing the code described 3.5.1.2 into webots, we can see that our car can successfully distinguish the color, which is purple in this test, and complete the following task according to the result.



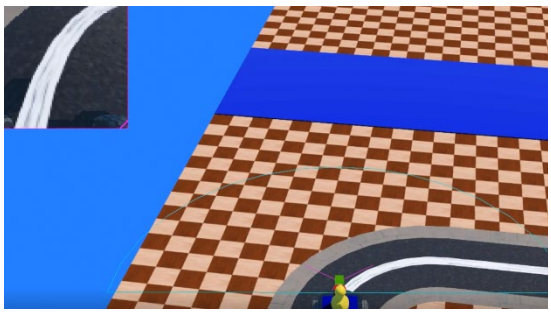
Before the Cross



After the Cross



Straight



Turning

Figure 38 Postures of Rover

4. Conclusions and Recommendation

4.1. Graph Design

To sum up, after the operation and design above, the patio discussed above can meet mostly every requirement given in the project textbook.

However, there is still a lot of room for improvement. First is the appearance, the lattice floor makes people's eyes uncomfortable. In the future test, we are planning to change it into the wooden floor. What else, we find that the patio is still empty. We did not design any other decoration in case of reducing the environmental impact on the sensor. At present, we are planning to put our team's logo on it.

Figure 39 Pioneer 3AT

What else, we find some attempt we made, the overpass, for example, can make the patio diversified. We will give these a reconsideration and try to make the patio better.

4.2. Casting

From the above, we can know there exit 3 methods if we want to throw out the finished food. They are Track belt, Catapult, and Robotic arm. However, we also find that the first two methods – Track belt and Catapult, are really hard to test to make sure the position the object will drop. So, if we want to deliver the fish food into a certain position such as the orange box in the task, the robotic

arm may be the best choice.

When controlling the robotic arm, we divided into two parts, one is the gripper and the other one in the arm. As a result, it can de free rotation as we want. At last, we write a general control code to let it finish the total task.
As we can see, the car can detect the object and deliver the fish food into the box as what we want, and that is all for task 2.

4.3. Car design and control

The task of our group is to design the main body of the car and make a combination of the achievement of other groups.

We chose pioneer 3AT as the main body of the rover and modified the car, added a camera, and reserved space for the casting group to place the robotic arm. We also wrote the code of the main task flow. To merge other parts into the robot, we learned the use of the various sensors in Webots and some algorithms used by two groups.

Finally, we made a car whose specifications meet requirements and contains all the functions needed to finish the 5 tasks.

4.4. Distance Recognition

Only if the car can recognize its moving direction and the distance & angle between the obstacle and itself. Then it can operate stably on the command direction, moving towards the middle position of the obstacle and avoiding it.

To operate these functions, we utilize the IMU module and the lidar module. The IMU module can help us determine the current moving angle of the car. The lidar module can help us determine the relative angle & distance between the middle point of the detected target and the car.

After we send the instant distance and angle data to the motor, the car can detect the bridge, the trees, and the arch successfully.

4.5. Image Recognition

4.5.1. Color Recognition

In this part, we are required to take the image input from the camera and judge the main color based on it. To realize this task, we first choose to use the RGB color model since the image of the simulation environment is not affected by the light. Then we built two key functions to judge the color and process the image respectively. Within these two functions, the general logic to count the pixels taken by each color, thus determining the main color by comparing the number of pixels. The results show that our design can determine the color and follow the corresponding route to the finishing line successfully.

4.5.2. Line Tracing

The method of line tracing works well and can meet the requirement. However, we can see that it shakes a lot when driving, this is because the robot is adjusting the position from the camera. So for this project, the algorithm is good, but if we want to do some big projects such as real auto-driving cars, we need to think more reliable algorithms.

Reference

[1]. Cyberbotics, Floors, https://cyberbotics.com/doc/guide/object-floors, [Accessed on 10/April/2020].

[2]. Cyberbotics, Road, https://cyberbotics.com/doc/guide/object-road, [Accessed on 10/April/2020].

[3]. Cyberbotics, Material, Field Summery, https://cyberbotics.com/doc/reference/material, [Accessed on 10/April/2020].

[4]. Cyberbotics, Lidar Sensors, http://www.cyberbotics.com/doc/guide/lidar-sensors, [Accessed on 22/May/2020].

[5]. Cyberbotics, KUKA's youBot, http://www.cyberbotics.com/doc/guide/youbot, [Accessed on 2/May/2020].

[6]. Cyberbotics, Distance Sensors, http://www.cyberbotics.com/doc/guide/distancesensor-sensors, [Accessed on 14/May/2020].

[7]. Cyberbotics, Pioneer 3AT, https://cyberbotics.com/doc/guide/pioneer-3at, [Accessed on 11/Apri/2020].

[8]. Wikipedia, PID Controller, https://en.wikipedia.org/wiki/PID_controller, [Accessed on 20/Mar/2020].

[9]. Wikipedia, Finite State Machine, https://en.wikipedia.org/wiki/State_machine, [Accessed on 17/Apri/2020].

[10]. Willem Dirk Van Dirk Van Driel, O.P., Sensor Systems Simulations: From Concept to Solution. 2018: p. 160-170.

[11]. F. Castano, G.B.R.H., Obstacle recognition based on machine learning for On-Chip LiDAR sensors in a cyber system. 2017: p. Sensors 17(9).

[12]. Cyberbotics, Inertia Unit, https://cyberbotics.com/doc/reference/inertialunit?version=R2020a-rev1, [Accessed on 28/May/2020].

[13]. Cyberbotics, camera-sensors, https://cyberbotics.com/doc/guide/camera-sensors, [Accessed on

18/May/2020].
[14]. OpenMV, follow-lines, https://book.openmv.cc/project/follow-lines.html, [Accessed on 14/April/2020].
[15]. Wikipedia, Gaussian_filter, https://en.wikipedia.org/wiki/Gaussian_filter, [Accessed on 18/May/2020].

Appendix I (codes)

Code 1: TDPS.cpp

```
1.  // Initial car coordinate = [-49 0.1 11.5]
2.  // Corner car coordinate = [-31.5 0.1 11.5]
3.  // Bridge car coordinate = [18 0.1 3.5]
4.  // Cast car coordinate = [-1.7 0.1 3.5]
5.  // Initial food coordinate = [-48.9373 0.64 11.5065]
6.  #include <stdio.h>
7.  #include <stdlib.h>
8.  #include <iostream>
9.  #include <arm.h>
10. #include <gripper.h>
11. #include <string.h>
12. #include <math.h>
13. #include <webots/robot.h>
14. #include <webots/motor.h>
15. #include <webots/lidar.h>
16. #include <webots/camera.h>
17. #include <webots/inertial_unit.h>
18. #include <webots/position_sensor.h>
19. #include <funcs.hpp>
20. using namespace std;
21.
22. #define north 0
23. #define south -pi
24. #define east -pi/2
25. #define west pi/2
26.
27. // Work Frequency
28. // float TIME_STEP = 10.;
29. // Class declaration
30. Funcs funcs;
31. // Data variables
```

```
32. const float *lidar_val = NULL;
33. const double *imu_val = NULL;
34. const unsigned char *imag_val = NULL;
35. // Define the height and width of the camera
36. int camera_w = 0;
37. int camera_h = 0;
38. // Number of lidar beams
39. int lidar_width = 0;
40. // State machines
41. unsigned short cast_seq[5] = {0};
42. unsigned short bridge_seq[7] = {0};
43. unsigned short color_seq[2] = {0};
44. // The ith stop sign has been taken
45. // The tree can be regarded as a stop sign
46. bool taken[3] = {0};
47. // Speed-angle coefficeint
48. // This coefficient connects dirction difference and wheel speed
49. float L = 2.;
50. // Target status of car
51. struct Mobile
52. {
53.   // float dir = -pi/2;
54.   float dir = pi;
55.   float speed = 6.3;
56. }Mobile;
57.
58. // Sensors
59. WbDeviceTag lidar;
60. WbDeviceTag imu;
61. WbDeviceTag camera;
62. WbDeviceTag front_left_wheel;
63. WbDeviceTag front_right_wheel;
64. WbDeviceTag back_left_wheel;
65. WbDeviceTag back_right_wheel;
66.
67. // Functions initializations
68. void robot_init(void);
69. void mainloop(void);
70. void set_speed(Speed spe);
71.
72. static void step()
73. {
74.   if (wb_robot_step(TIME_STEP) == -1)
75.   {
```

```
76.     wb_robot_cleanup();
77.     exit(EXIT_SUCCESS);
78.   }
79. }
80.
81. static void passive_wait(double sec)
82. {
83.   double start_time = wb_robot_get_time();
84.   do
85.   {
86.     step();
87.   }while(start_time + sec >wb_robot_get_time());
88. }
89.
90. void go(enum Height height1)
91. {
92.   arm_set_height(height1);
93. }
94.
95. void back(enum Height height2)
96. {
97.   arm_set_height(height2);
98. }
99.
100. int main(int argc, char **argv)
101. {
102.   // Initialize
103.   robot_init();
104.   arm_init();
105.   gripper_init();
106.   // Mainloop
107.   while (wb_robot_step(TIME_STEP) != -1)
108.     mainloop();
109.   // Clear memory
110.   wb_robot_cleanup();
111.   delete lidar_val;
112.   delete imu_val;
113.   delete imag_val;
114.   return 0;
115. }
116.
117. void robot_init(void)
118. {
119.   wb_robot_init();
```

```
120.   // Get the equipments
121.   lidar = wb_robot_get_device("lidar");
122.   imu = wb_robot_get_device("imu");
123.   camera = wb_robot_get_device("camera");
124.   front_left_wheel = wb_robot_get_device("front left wheel");
125.   front_right_wheel = wb_robot_get_device("front right wheel");
126.   back_left_wheel = wb_robot_get_device("back left wheel");
127.   back_right_wheel = wb_robot_get_device("back right wheel");
128.   // Set the motor end angle to infinity
129.   wb_motor_set_position(front_left_wheel, INFINITY);
130.   wb_motor_set_position(front_right_wheel, INFINITY);
131.   wb_motor_set_position(back_left_wheel, INFINITY);
132.   wb_motor_set_position(back_right_wheel, INFINITY);
133.   // Enable
134.   wb_lidar_enable(lidar, TIME_STEP);
135.   wb_inertial_unit_enable(imu, TIME_STEP);
136.   wb_camera_enable(camera, TIME_STEP);
137.   // Number of lidar beams
138.   lidar_width = wb_lidar_get_horizontal_resolution(lidar);
139.   funcs.init_lidar(lidar_width);
140.   // Number of pixels in height and width
141.   camera_w = wb_camera_get_width(camera);
142.   camera_h = wb_camera_get_height(camera);
143. }
144.
145. void mainloop(void)
146. {
147.   // Get data
148.   lidar_val = wb_lidar_get_range_image(lidar);
149.   imu_val = wb_inertial_unit_get_roll_pitch_yaw(imu);
150.   imag_val = wb_camera_get_image(camera);
151.   // Get distance
152.   funcs.distance_get(lidar_val, pi, 0.15, 0);
153.   // Set speed
154.   Speed temp;
155.   temp = funcs.motion_keep(imu_val[2], Mobile.dir, L);
156.   temp.wheel_l += Mobile.speed;
157.   temp.wheel_r += Mobile.speed;
158.   set_speed(temp);
159.   // Get obstacle information
160.   Obj_center obstacle = funcs.obj_dis_info();
161.   Obj_center lin_imag = funcs.imag_process(imag_val);
162.   float real_distance = sqrt(obstacle.obj_x * obstacle.obj_x + obstacle.obj
    _y * obstacle.obj_y);
```

```
163.   float real_angle = acos(obstacle.obj_x / real_distance) - pi/2;
164.   float line_dir = sqrt(lin_imag.obj_x * lin_imag.obj_x + lin_imag.obj_y *
       lin_imag.obj_y);
165.   float line_angle = acos(lin_imag.obj_x / line_dir) - pi/2;
166.   Mobile.dir = imu_val[2] + line_angle;
167.   gripper_grip();
168.   // Main logic
169.   // Start
170.   if(cast_seq[0] == 0)
171.   {
172.     Mobile.speed = 6.3;
173.     if(obstacle.obj_y < funcs.lidar_max_range)
174.       Mobile.speed = 10*(obstacle.obj_y - 0.9);
175.     if(obstacle.obj_y - 0.9 < 0.05)
176.       cast_seq[0] = 1;
177.   }
178.   else if(cast_seq[1] == 0)
179.   {
180.     Mobile.dir = south;
181.     if(fabs(imu_val[2] + pi < 0.2))
182.       Mobile.speed = 25*(obstacle.obj_y - 0.32);
183.     if(fabs(obstacle.obj_y - 0.32) < 0.03)
184.       cast_seq[1] = 1;
185.   }
186.   else if(cast_seq[2] == 0)
187.   {
188.     Mobile.dir = north;
189.     Mobile.speed = 0;
190.     if(fabs(imu_val[2] < 0.1))
191.     {
192.       wb_motor_set_velocity(front_left_wheel, 0);
193.       wb_motor_set_velocity(front_right_wheel, 0);
194.       wb_motor_set_velocity(back_left_wheel, 0);
195.       wb_motor_set_velocity(back_right_wheel, 0);
196.       passive_wait(0.5);
197.       wb_motor_set_velocity(front_left_wheel, -1);
198.       wb_motor_set_velocity(front_right_wheel, -1);
199.       wb_motor_set_velocity(back_left_wheel, -1);
200.       wb_motor_set_velocity(back_right_wheel, -1);
201.       passive_wait(1);
202.       wb_motor_set_velocity(front_left_wheel, 0);
203.       wb_motor_set_velocity(front_right_wheel, 0);
204.       wb_motor_set_velocity(back_left_wheel, 0);
205.       wb_motor_set_velocity(back_right_wheel, 0);
```

```
206.       go(ARM_FRONT_FLOOR);
207.       passive_wait(6.0);
208.       gripper_release();
209.       passive_wait(3);
210.       back(ARM_RESET);
211.       passive_wait(1.0);
212.       if (fabs(wb_position_sensor_get_value(arm_pos[ARM3]) + 2.635) <0.04)

213.         cast_seq[2] = 1;
214.     }
215.   }
216.   else if(cast_seq[3] == 0)
217.   {
218.     Mobile.dir = east + pi/8;
219.     Mobile.speed = 6.3;
220.     if (funcs.count > 500)
221.       cast_seq[3] = 1;
222.   }
223.   else if(cast_seq[4] == 0)
224.   {
225.     Mobile.dir = imu_val[2] + line_angle;
226.     if(funcs.count < 40)
227.     {
228.       Mobile.dir = east;
229.       cast_seq[4] = 1;
230.     }
231.   }
232.   // Before the first stop sign
233.   else if(bridge_seq[0] == 0)
234.   {
235.     Mobile.dir = east;
236.     if(real_distance < 1.3)
237.     {
238.       Mobile.dir = north;
239.       // When the aim direction is setting to 0, seq0 is over
240.       bridge_seq[0] = 1;
241.     }
242.   }
243.   // Before the bridge
244.   else if(bridge_seq[1] == 0)
245.   {
246.     // Keep straight
247.     Mobile.dir = north;
248.     // Slow down
```

```
249.     Mobile.speed = 3;
250.     if(abs(pi/6 - imu_val[0]) < 0.05)
251.       bridge_seq[1] = 1;
252.   }
253.   // Down
254.   else if(bridge_seq[2] == 0)
255.   {
256.     Mobile.dir = north;
257.     if(abs(pi/6 + imu_val[0]) < 0.05)
258.       bridge_seq[2] = 1;
259.   }
260.   // Finished downhill
261.   else if(bridge_seq[3] == 0)
262.   {
263.     Mobile.dir = north;
264.     if(abs(imu_val[0]) < 0.05)
265.       bridge_seq[3] = 1;
266.   }
267.   // Before the tree
268.   else if(bridge_seq[4] == 0)
269.   {
270.     Mobile.dir = north;
271.     Mobile.speed = 6.3;
272.     if(real_distance < 1)
273.     {
274.       Mobile.dir = east;
275.       // When after the bridge and
276.       // the direction is close to -pi/2, seq2 is over
277.       if(abs(-pi/2 - imu_val[2]) < 0.3)
278.         bridge_seq[4] = 1;
279.     }
280.   }
281.   // Before the second stop
282.   else if(bridge_seq[5] == 0)
283.   {
284.     Mobile.dir = east;
285.     if(real_distance < 1.5)
286.     {
287.       Mobile.dir = north;
288.       if(abs(0 - imu_val[2]) < 0.2)
289.         bridge_seq[5] = 1;
290.     }
291.   }
292.   // Before the arch
```

```
293.   else if(bridge_seq[6] == 0)
294.   {
295.     Mobile.dir = north + real_angle;
296.     Mobile.speed = 5;
297.     // Have Detected the white line
298.     if(funcs.count > 100)
299.     {
300.       Mobile.speed = 6.3;
301.       bridge_seq[6] = 1;
302.     }
303.   }
304.   // Before color
305.   else if(color_seq[0] == 0)
306.   {
307.     if(real_distance < 1 && abs(imu_val[2] - west) < 0.05)
308.       color_seq[0] = 1;
309.   }
310.   else if(color_seq[1] == 0)
311.   {
312.     Mobile.speed = 5;
313.     if(funcs.c_r > funcs.c_y and funcs.c_r > funcs.c_p)
314.     {
315.       funcs.color_trace = RED;
316.       cout << "red" <<endl;
317.     }
318.     else if(funcs.c_p > funcs.c_y and funcs.c_p > funcs.c_r)
319.     {
320.       funcs.color_trace = PURPLE;
321.       cout << "purple" <<endl;
322.     }
323.     else
324.     {
325.       funcs.color_trace = YELLOW;
326.       cout << "yellow" <<endl;
327.     }
328.     color_seq[1] = 1;
329.   }
330. }
331.
332. void set_speed(Speed spe)
333. {
334.   // Limit the speed (-6.4,6.4)
335.   spe.wheel_l = funcs.val_limit(spe.wheel_l,-6.3,6.3);
336.   spe.wheel_r = funcs.val_limit(spe.wheel_r,-6.3,6.3);
```

```
337.   // Set speed
338.   wb_motor_set_velocity(front_left_wheel, spe.wheel_l);
339.   wb_motor_set_velocity(front_right_wheel, spe.wheel_r);
340.   wb_motor_set_velocity(back_left_wheel, spe.wheel_l);
341.   wb_motor_set_velocity(back_right_wheel, spe.wheel_r);
342. }
```

Code 2: funcs.hpp

```
1.  #ifndef _FUNCS_HPP_
2.  #define _FUNCS_HPP_
3.    #include <iostream>
4.    #include <string>
5.    #include <math.h>
6.    #include <vector>
7.    using namespace std;
8.
9.    #define pi 3.1415926
10.   #define WHITE 0
11.   #define RED 1
12.   #define YELLOW 2
13.   #define PURPLE 3
14.   #define width 86
15.   #define height 86
16.   #define mat vector<vector<vector<unsigned char>>>
17.
18.   // Speed of wheels
19.   struct Speed
20.   {
21.     float wheel_l = 0.;
22.     float wheel_r = 0.;
23.   };
24.   // Center of obstacle
25.   struct Obj_center
26.   {
27.     float obj_x = 0.;
28.     float obj_y = 0.;
29.   };
30.
31.   class Funcs
32.   {
33.     private:
34.       // History error for PD control
```

```
35.       float err_yaw_last = 0.;
36.       float MAX_SPEED = 5.;
37.       float gaus[3][3] =
38.       {
39.         {0.0751,0.1238,0.0751},
40.         {0.1238,0.2042,0.1238},
41.         {0.0751,0.1238,0.0751},
42.       };
43.
44.     public:
45.       // Cartesian coordinate
46.       float *lidar_dcrx = NULL;
47.       float *lidar_dcry = NULL;
48.       // Lidar effective or not
49.       short *lidar_effective = NULL;
50.       // Number of lidar beams (512)
51.       int lidar_num = 0;
52.       // Range of lidar = 2m
53.       float lidar_max_range = 2.;
54.       // Number of pixels of a picture
55.       mat imag_temp;
56.       short color_trace = RED;
57.       int count = 0;
58.       int c_r = 0;
59.       int c_y = 0;
60.       int c_p = 0;
61.
62.       ~Funcs()
63.       {
64.         delete lidar_dcrx;
65.         delete lidar_dcry;
66.         delete lidar_effective;
67.       }
68.
69.       // Color judge
70.       bool color_judge(vector<unsigned char> color, short id)
71.       {
72.         short max = 190;
73.         short min = 90;
74.         // 255 255 255
75.         if (id == WHITE)
76.         {
77.             if (color[0] > max && color[1] > max && color[2] > max)
78.               return true;
```

```
79.             else
80.               return false;
81.         }
82.
83.         // 255 0 0
84.         else if (id == RED)
85.         {
86.             if (color[0] > max && color[1] <  min && color[2] < min)
87.               return true;
88.             else
89.               return false;
90.         }
91.
92.         // 255 255 0
93.         else if (id == YELLOW)
94.         {
95.             if (color[0] >  max && color[1] >  max && color[2] < min)
96.               return true;
97.             else
98.               return false;
99.         }
100.
101.         // 0 0 255
102.         else if (id == PURPLE)
103.         {
104.             if (color[0] > max && color[1] < min && color[2] >  max)
105.               return true;
106.             else
107.               return false;
108.         }
109.         return false;
110.       }
111.
112.       // Gaussian Filter
113.       mat gaussian(mat imag)
114.       {
115.         mat temp = imag_size(width,height);
116.         for (short r = 1; r < width-1; r += 1)
117.         {
118.           for (short c = 1; c < height-1; c += 1)
119.           {
120.             for (short p = 0; p < 3; p += 1)
121.             {
122.               float add= 0;
```

```
123.                for (short i = 0; i < 3; i += 1)
124.                {
125.                  for (short j = 0; j < 3; j += 1)
126.                    add += (float)imag[r-1 +i][c-1 + j][p] * gaus[i][j];
127.                }
128.                  temp[r][c][p] = (unsigned char) add;
129.              }
130.            }
131.          }
132.          return temp;
133.        }
134.
135.        // Image Processing
136.        Obj_center imag_process(const unsigned char *imag)
137.        {
138.          Obj_center temp_out;
139.          imag_temp = imag_copy(imag);
140.          mat temp = gaussian(imag_temp);
141.          int r_add = 0;
142.          int c_add = 0;
143.          count = 0;
144.          c_r = 0;
145.          c_y = 0;
146.          c_p = 0;
147.          for(short r = 1; r < width-1; r++)
148.          {
149.            for(short c = 1; c < height-1; c++)
150.            {
151.              if(color_judge(temp[r][c],WHITE) or color_judge(temp[r][c],colo
    r_trace))
152.              {
153.                r_add += r;
154.                c_add += c;
155.                count += 1;
156.              }
157.
158.              if(color_judge(temp[r][c],RED))
159.                c_r += 1;
160.              else if(color_judge(temp[r][c],YELLOW))
161.                c_y += 1;
162.              else if(color_judge(temp[r][c],PURPLE))
163.                c_p += 1;
164.            }
165.          }
```

```
166.          float y = 0;
167.          float x = 0;
168.          if(count == 0)
169.          {
170.            count = 1;
171.            x = 0;
172.            y = 0.5*height;
173.          }
174.          else
175.          {
176.            y = height - r_add/count;
177.            x = c_add / count - 0.5*width;
178.          }
179.          temp_out.obj_x = x;
180.          temp_out.obj_y = y;
181.          // (x,y) are direction points
182.          return temp_out;
183.        }
184.
185.        // Transfer image from "const unsigned char *" to "mat"
186.        mat imag_copy(const unsigned char *imag)
187.        {
188.          mat temp = imag_size(width, height);
189.          int pixel = width * height;
190.          const unsigned char *imag_ = imag;
191.          // Row
192.          int temp_r = 0;
193.          // Column
194.          int temp_c = 0;
195.          for(int i = 0; i < pixel; i++, imag_ += 4)
196.          {
197.            temp_c = i % width;
198.            temp_r = (int) i / height;
199.            // The image is coded as a 3-bytes sequence
200.            // representing red, green and blue levels of a pixel
201.            // Red
202.            temp[temp_r][temp_c][0] = imag_[2];
203.            // Green
204.            temp[temp_r][temp_c][1] = imag_[1];
205.            // Blue
206.            temp[temp_r][temp_c][2] = imag_[0];
207.          }
208.          return temp;
209.        }
```

```
210.
211.        // Size of image, initialize the mat
212.        mat imag_size(short wid, short hei)
213.        {
214.          mat temp;
215.          temp.resize(hei);
216.          for(int i = 0; i < hei; i++)
217.          {
218.            temp[i].resize(wid);
219.            for(int j = 0; j < wid; j++)
220.              temp[i][j].resize(3);
221.          }
222.          return temp;
223.        }
224.
225.        // Init lidar in Funcs
226.        void init_lidar(int num)
227.        {
228.          lidar_dcrx = new float [num];
229.          lidar_dcry = new float [num];
230.          lidar_num = num;
231.          lidar_effective = new short [num];
232.        }
233.
234.        // yaw: current value, yaw_d: target value, L: speed coefficient
235.        Speed motion_keep(double yaw, double yaw_d, float L)
236.        {
237.          Speed temp;
238.          double dis = yaw - yaw_d;
239.          // If dis < 0, turn left, else, turn right
240.          double yaw_w = pd_control(abs(dis), 20., 5.);
241.          double speed_d = val_limit(yaw_w*L, -MAX_SPEED, MAX_SPEED);
242.          if(dis < 0)
243.          {
244.            temp.wheel_l = -speed_d;
245.            temp.wheel_r = speed_d;
246.          }
247.          else if (dis == 0)
248.          {
249.            temp.wheel_l = speed_d;
250.            temp.wheel_r = speed_d;
251.          }
252.          else
253.          {
```

```
254.            temp.wheel_l = speed_d;
255.            temp.wheel_r = -speed_d;
256.          }
257.          return temp;
258.        }
259.
260.        // Return the center information of obstacle
261.        Obj_center obj_dis_info(void)
262.        {
263.          Obj_center temp;
264.          float temp_x = 0.;
265.          float temp_y = 0.;
266.          short counter = 0;
267.          // Sum
268.          for(int i = 0; i < lidar_num; i++)
269.          {
270.            if(lidar_effective[i] == 1)
271.            {
272.              temp_x += lidar_dcrx[i];
273.              temp_y += lidar_dcry[i];
274.              counter += 1;
275.            }
276.          }
277.          // Defaut values
278.          if(counter == 0)
279.          {
280.            temp_x = 0.;
281.            temp_y = 100.;
282.            counter = 1;
283.          }
284.          // Average
285.          temp.obj_x = temp_x / counter;
286.          temp.obj_y = temp_y / counter;
287.          return temp;
288.        }
289.
290.        // Transfer to cartesian coordinate
291.        // x_bias and z_bias are the difference between center of car and cen
    ter of lidar module
292.        // In this project, ang_range is pi
293.        void distance_get(const float *lidar_val, float ang_range, float z_bi
    as, float x_bias)
294.        {
295.          float piece = ang_range / lidar_num;
```

```
296.          for(int i = 0; i < lidar_num; i++)
297.          {
298.            double ang = ang_range + (pi - ang_range)/2 - piece * i;
299.            lidar_dcrx[i] = (float)(lidar_val[i] * cos(ang) + x_bias);
300.            lidar_dcry[i] = (float)(lidar_val[i] * sin(ang) + z_bias);
301.            // If no detection
302.            if (lidar_val[i]/lidar_max_range>0.95)
303.              lidar_effective[i] = 0;
304.            else
305.              lidar_effective[i] = 1;
306.          }
307.        }
308.
309.        // Restrict function
310.        float val_limit(float val, float min, float max)
311.        {
312.          float temp = 0;
313.          if(val < min)
314.            temp = min;
315.          else if(val > max)
316.            temp = max;
317.          else
318.            temp = val;
319.          return temp;
320.        }
321.
322.        // PD Control
323.        float pd_control(float dis, float k, float d)
324.        {
325.          float temp = 0.;
326.          float dis_2 = 0.;
327.          dis_2 = dis - err_yaw_last;
328.          temp = k * dis + d * abs(0-dis_2);
329.          err_yaw_last = dis;
330.          return temp;
331.        }
332.    };
333. #endif
```

Code 3: arm.h

```
1.  #include <webots/motor.h>
2.  #include <webots/robot.h>
```

```
3.  #include <webots/position_sensor.h>
4.  #include <math.h>
5.  #include <stdio.h>
6.
7.  #define TIME_STEP 10
8.
9.  static WbDeviceTag arm_elements[5];
10. static WbDeviceTag arm_pos[5];
11.
12. enum Height {
13.   ARM_FRONT_FLOOR,
14.   ARM_FRONT_PLATE,
15.   ARM_HANOI_PREPARE,
16.   ARM_FRONT_CARDBOARD_BOX,
17.   ARM_RESET,
18.   ARM_BACK_PLATE_HIGH,
19.   ARM_BACK_PLATE_LOW,
20.   ARM_MAX_HEIGHT
21. };
22.
23. enum Orientation {
24.   ARM_BACK_LEFT,
25.   ARM_LEFT,
26.   ARM_FRONT_LEFT,
27.   ARM_FRONT,
28.   ARM_FRONT_RIGHT,
29.   ARM_RIGHT,
30.   ARM_BACK_RIGHT,
31.   ARM_MAX_SIDE
32. };
33.
34. enum Arm { ARM1, ARM2, ARM3, ARM4, ARM5 };
35.
36. static enum Height current_height = ARM_RESET;
37. static enum Orientation current_orientation = ARM_FRONT;
38.
39. void arm_reset() {
40.   wb_motor_set_position(arm_elements[ARM1], 0.0);
41.   wb_motor_set_position(arm_elements[ARM2], 1.57);
42.   wb_motor_set_position(arm_elements[ARM3], -2.635);
43.   wb_motor_set_position(arm_elements[ARM4], 1.78);
44.   wb_motor_set_position(arm_elements[ARM5], 0.0);
45. }
46.
```

```
47. void arm_set_height(enum Height h_height) {
48.   switch (h_height) {
49.     case ARM_FRONT_FLOOR:
50.       wb_motor_set_position(arm_elements[ARM2], -0.97);
51.       wb_motor_set_position(arm_elements[ARM3], -1.55);
52.       wb_motor_set_position(arm_elements[ARM4], -0.61);
53.       wb_motor_set_position(arm_elements[ARM5], 0.0);
54.       break;
55.     case ARM_FRONT_PLATE:
56.       wb_motor_set_position(arm_elements[ARM2], -0.62);
57.       wb_motor_set_position(arm_elements[ARM3], -0.98);
58.       wb_motor_set_position(arm_elements[ARM4], -1.53);
59.       wb_motor_set_position(arm_elements[ARM5], 0.0);
60.       break;
61.     case ARM_FRONT_CARDBOARD_BOX:
62.       wb_motor_set_position(arm_elements[ARM2], 0.0);
63.       wb_motor_set_position(arm_elements[ARM3], -0.77);
64.       wb_motor_set_position(arm_elements[ARM4], -1.21);
65.       wb_motor_set_position(arm_elements[ARM5], 0.0);
66.       break;
67.     case ARM_RESET:
68.       wb_motor_set_position(arm_elements[ARM2], 1.57);
69.       wb_motor_set_position(arm_elements[ARM3], -2.635);
70.       wb_motor_set_position(arm_elements[ARM4], 1.78);
71.       wb_motor_set_position(arm_elements[ARM5], 0.0);
72.       break;
73.     case ARM_BACK_PLATE_HIGH:
74.       wb_motor_set_position(arm_elements[ARM2], 0.678);
75.       wb_motor_set_position(arm_elements[ARM3], 0.682);
76.       wb_motor_set_position(arm_elements[ARM4], 1.74);
77.       wb_motor_set_position(arm_elements[ARM5], 0.0);
78.       break;
79.     case ARM_BACK_PLATE_LOW:
80.       wb_motor_set_position(arm_elements[ARM2], 0.92);
81.       wb_motor_set_position(arm_elements[ARM3], 0.42);
82.       wb_motor_set_position(arm_elements[ARM4], 1.78);
83.       wb_motor_set_position(arm_elements[ARM5], 0.0);
84.       break;
85.     case ARM_HANOI_PREPARE:
86.       wb_motor_set_position(arm_elements[ARM2], -0.4);
87.       wb_motor_set_position(arm_elements[ARM3], -1.2);
88.       wb_motor_set_position(arm_elements[ARM4], -M_PI_2);
89.       wb_motor_set_position(arm_elements[ARM5], M_PI_2);
90.       break;
```

```
91.     default:
92.       fprintf(stderr, "arm_height() called with a wrong argument\n");
93.       return;
94.   }
95.   current_height = h_height;
96. }
97.
98. void arm_set_orientation(enum Orientation orientation) {
99.   switch (orientation) {
100.     case ARM_BACK_LEFT:
101.       wb_motor_set_position(arm_elements[ARM1], -2.949);
102.       break;
103.     case ARM_LEFT:
104.       wb_motor_set_position(arm_elements[ARM1], -M_PI_2);
105.       break;
106.     case ARM_FRONT_LEFT:
107.       wb_motor_set_position(arm_elements[ARM1], -0.2);
108.       break;
109.     case ARM_FRONT:
110.       wb_motor_set_position(arm_elements[ARM1], 0.0);
111.       break;
112.     case ARM_FRONT_RIGHT:
113.       wb_motor_set_position(arm_elements[ARM1], 0.2);
114.       break;
115.     case ARM_RIGHT:
116.       wb_motor_set_position(arm_elements[ARM1], M_PI_2);
117.       break;
118.     case ARM_BACK_RIGHT:
119.       wb_motor_set_position(arm_elements[ARM1], 2.949);
120.       break;
121.     default:
122.       fprintf(stderr, "arm_set_side() called with a wrong argument\n");
123.       return;
124.   }
125.   current_orientation = orientation;
126. }
127.
128. void arm_init() {
129.   arm_elements[ARM1] = wb_robot_get_device("arm1");
130.   arm_elements[ARM2] = wb_robot_get_device("arm2");
131.   arm_elements[ARM3] = wb_robot_get_device("arm3");
132.   arm_elements[ARM4] = wb_robot_get_device("arm4");
133.   arm_elements[ARM5] = wb_robot_get_device("arm5");
134.
```

```
135.   arm_pos[ARM1] = wb_motor_get_position_sensor(arm_elements[ARM1]);
136.   arm_pos[ARM2] = wb_motor_get_position_sensor(arm_elements[ARM2]);
137.   arm_pos[ARM3] = wb_motor_get_position_sensor(arm_elements[ARM3]);
138.   arm_pos[ARM4] = wb_motor_get_position_sensor(arm_elements[ARM4]);
139.   arm_pos[ARM5] = wb_motor_get_position_sensor(arm_elements[ARM5]);
140.
141.   wb_position_sensor_enable(arm_pos[ARM1],TIME_STEP);
142.   wb_position_sensor_enable(arm_pos[ARM2],TIME_STEP);
143.   wb_position_sensor_enable(arm_pos[ARM3],TIME_STEP);
144.   wb_position_sensor_enable(arm_pos[ARM4],TIME_STEP);
145.   wb_position_sensor_enable(arm_pos[ARM5],TIME_STEP);
146.
147.   wb_motor_set_velocity(arm_elements[ARM2], 0.5);
148.
149.   arm_set_height(ARM_RESET);
150.   arm_set_orientation(ARM_FRONT);
151. }
```

Code 4: gripper.h

```
1.  #include <webots/motor.h>
2.  #include <webots/robot.h>
3.
4.  #define LEFT 0
5.  #define RIGHT 1
6.
7.  #define MIN_POS 0.0
8.  #define MAX_POS 0.025
9.  #define OFFSET_WHEN_LOCKED 0.021
10.
11. static WbDeviceTag fingers[2];
12.
13. double bound(double v, double a, double b) {
14.   return (v > b) ? b : (v < a) ? a : v;
15. }
16.
17. void gripper_init() {
18.   fingers[LEFT] = wb_robot_get_device("finger1");
19.   fingers[RIGHT] = wb_robot_get_device("finger2");
20.
21.   wb_motor_set_velocity(fingers[LEFT], 0.03);
22.   wb_motor_set_velocity(fingers[RIGHT], 0.03);
23. }
```

```
24.
25. void gripper_grip() {
26.   wb_motor_set_position(fingers[LEFT], MIN_POS);
27.   wb_motor_set_position(fingers[RIGHT], MIN_POS);
28. }
29.
30. void gripper_release() {
31.   wb_motor_set_position(fingers[LEFT], MAX_POS);
32.   wb_motor_set_position(fingers[RIGHT], MAX_POS);
33. }
34.
35. void gripper_set_gap(double gap) {
36.   double v = bound(0.5 * (gap - OFFSET_WHEN_LOCKED), MIN_POS, MAX_POS);
37.   wb_motor_set_position(fingers[LEFT], v);
38.   wb_motor_set_position(fingers[RIGHT], v);
39. }
```

Code 5: gripper_open_size_controller.c

```
1.  #include <webots/motor.h>
2.  #include <webots/keyboard.h>
3.  #include <webots/robot.h>
4.  #include <webots/camera.h>
5.  #include <webots/distance_sensor.h>
6.
7.  #include "arm.h"
8.  #include "gripper.h"
9.
10. #include <math.h>
11. #include <stdio.h>
12. #include <string.h>
13. #include <stdlib.h>
14.
15. #define TIME_STEP 64
16.
17. double leftspeed = 0.0;
18. double rightspeed = 0.0;
19.
20. WbDeviceTag wheels[4];
21.
22. void check_keyboard(int key){
23.     switch(key){
24.
```

```
25.       case WB_KEYBOARD_UP:
26.           leftspeed = 4.0;
27.           rightspeed = 4.0;
28.           break;
29.         case WB_KEYBOARD_DOWN:
30.           leftspeed = -4.0;
31.           rightspeed = -4.0;
32.           break;
33.         case WB_KEYBOARD_RIGHT:
34.           leftspeed = 2.0;
35.           rightspeed = -2.0;
36.           break;
37.         case WB_KEYBOARD_LEFT:
38.           leftspeed = -2.0;
39.           rightspeed = 2.0;
40.           break;
41.         case '=':
42.         case 388:
43.         case 65585:
44.           printf("Grip\n");
45.           gripper_grip();
46.           break;
47.         case '-':
48.         case 390:
49.           printf("Ungrip\n");
50.           gripper_release();
51.           break;
52.
53.         case 332:
54.         case WB_KEYBOARD_UP | WB_KEYBOARD_SHIFT:
55.           printf("Increase arm height\n");
56.           arm_increase_height();
57.           break;
58.         case 326:
59.         case WB_KEYBOARD_DOWN | WB_KEYBOARD_SHIFT:
60.           printf("Decrease arm height\n");
61.           arm_decrease_height();
62.           break;
63.         case 330:
64.         case WB_KEYBOARD_RIGHT | WB_KEYBOARD_SHIFT:
65.           printf("Increase arm orientation\n");
66.           arm_increase_orientation();
67.           break;
68.         case 328:
```

```
69.         case WB_KEYBOARD_LEFT | WB_KEYBOARD_SHIFT:
70.           printf("Decrease arm orientation\n");
71.           arm_decrease_orientation();
72.           
73.         default :
74.           leftspeed = 0;
75.           rightspeed = 0;
76.     }
77. }
78. 
79. void set_wheels_velocity(){
80.         wb_motor_set_velocity(wheels[0],leftspeed);
81.         wb_motor_set_velocity(wheels[1],rightspeed);
82.         wb_motor_set_velocity(wheels[2],leftspeed);
83.         wb_motor_set_velocity(wheels[3],rightspeed);
84. }
85. 
86. 
87. int main(int argc, char **argv){
88.     wb_robot_init();
89. 
90.     arm_init();
91.     gripper_init();
92. 
93.     wb_keyboard_enable(TIME_STEP);
94. 
95. 
96.     char wheels_names[4][8] = {"wheel1", "wheel2", "wheel3", "wheel4"};
97.     for (int i=0; i<4; i++){
98.       wheels[i] = wb_robot_get_device(wheels_names[i]);
99.       wb_motor_set_position(wheels[i],INFINITY);
100.       wb_motor_set_velocity(wheels[i],0.0);
101.     }
102. 
103.     while (wb_robot_step(TIME_STEP) != -1){
104.       //drop();
105.       gripper_grip();
106.       int key = wb_keyboard_get_key();
107.       check_keyboard(key);
108.       set_wheels_velocity();
109. 
110. }
111.     wb_robot_cleanup();
112.     return 0;
```

```
113. }
```

Appendix II (Team Arrangement)

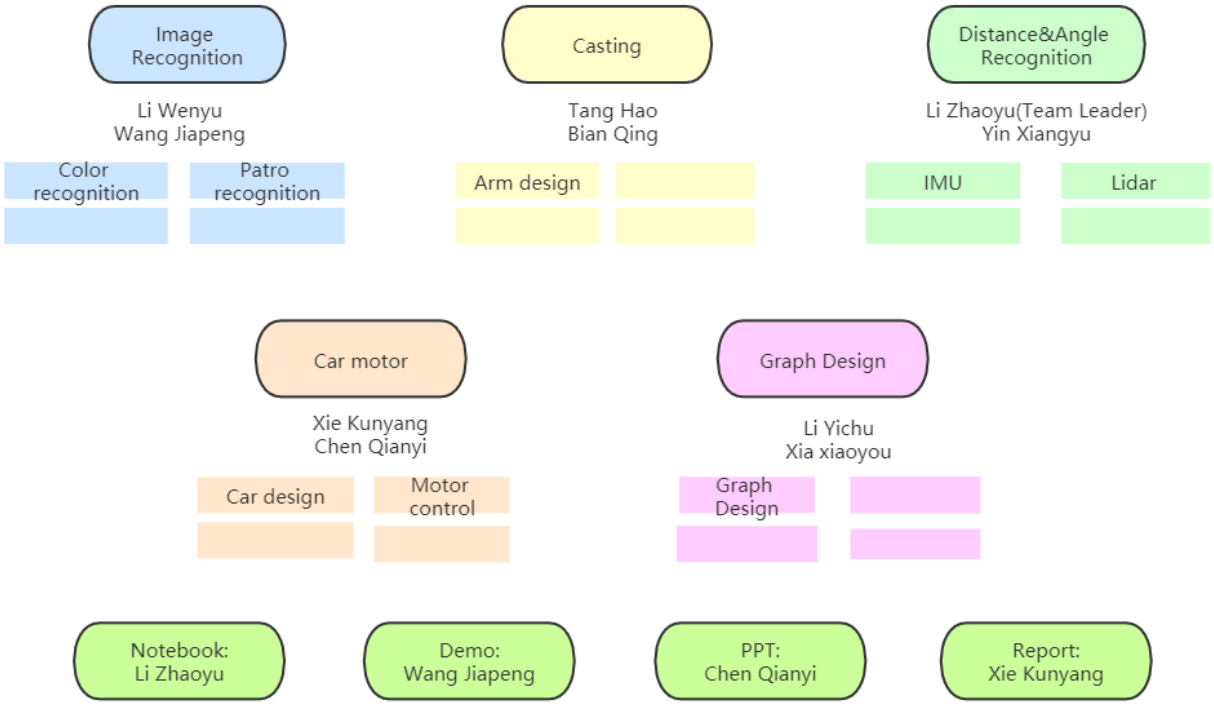


Figure 40 Team Arrangement