
电子科技大学
格拉斯哥学院
Glasgow College, UESTC

TEAM DESIGN PROJECT & SKILLS

University of Electronic Science and Technology of China

Glasgow College

# Multifunctional Rover Design on Webots

*Team:*
Team 32 - Mihotel

*Supervisor:*
Dr. Wasim Ahmad,
Dr. Abdullah Al-Khalidi

June 20, 2020

**Abstract**

This report is a comprehensive summary of our team designed multifunctional rover. Our rover is endowed with a wide range of features, including color recognition, path patrol, object detection, item releasing and so on. During the project development, we adopt an interactive designing strategy with a joint effort of the visual, decision and chassis groups. Our final design is able to finish the five specific tasks in a stable and efficient way. To promote work efficiency, we also employ the project management skills and base our work on two professional team development platforms, ZenHub and GitHub. We open-source our project to the community for further research[1].

[1] https://github.com/TDPS-Mihotel/Mihotel

**Acknowledgements**

This project is a joint effort of the Team 32 and we would like to acknowledge the contribution of our all members in our team. In particular, we would like to thank our team leader Zhuheng SONG for his time, patience and guidance. We would also like to thank our project manager Jinwei CHU for his excellent project management.

The Mihotel Rover is designed and developed online and we would like to thank Cyberbotics Webots, GitHub and ZenHub for their technical support. Also, we are grateful to Dr.Wasim Ahmad and Dr. Abdullah Al-Khalidi for delivering the course to us.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This document is a report for the team design project "Multifunctional Rover Design on Webots", which is part of the course Team Design Project and Skills[1]. In this course, our team is assigned with the design of an integrated electronic and electrical system that is capable of performing specific functions. In particular, we need to design and construct a line-patrol rover and finish five given tasks. This project is purely simulation-based with the Cyberbotics Webots[2].

During this project, we not only focus on the development of electronic and electrical systems, but also regard it as an excellent chance to put the project management skills learned in the course Engineering Project Management and Finance[3] into practice. To be specific, we follow the guidelines of the project planning, employing useful project management tools and techniques to promote the work efficiency. We also explore some widely-used team development applications, including ZenHub and GitHub to facilitate our work.

## 1.1 Aims & Objectives of the Project

In this report, we limit the scope to the design of the multifunctional rover, although it can be generalized to other electronic and electric system design tasks easily. The goal of our project is to develop a multifunctional intelligent line-patrol rover named **Mihotel Rover**. It is endowed with the following features:

- Autonomous driving - No external instructions needed to control the rover
- Colored path patrol - Capable of finding and following the path of black lines or lines with specific colors
- Object detection - Capable of recognizing the bridge over the river and the arch
- Color recognition - Capable of detecting the color of beacons and the color box
- Items Releasing - Capable of dropping the fish food on the orange fish tank
- Uphill & downhill moving - Capable of going up and down the bridge along the ramps
- Omnidirectional rotation - Capable of making a turn in any direction at any spot

However, since we design the rover using Webots simulation, our implementation cannot be the same as the rover designed in reality. We will return to the limitations in Section 6.2.

## 1.2 Background Research

Robots can be autonomous or semi-autonomous machines that simulate human behavior or thoughts and other creatures (such as robot dogs, robot cats, etc.)[1]. The intelligent robot can not only perceive the environment, but also think and judge to make reactive actions. Nowadays,

[1]The TDPS course is available at: https://moodle.gla.ac.uk/course/view.php?id=14648
[2]The Cyberbotics Webots can be accessed from: https://cyberbotics/com/
[3]The EPMF course is available at:https://moodle.gla.ac.uk/course/view.php?id=18377

intelligent robots can be used in industry, agriculture, medical treatment, military, and other fields. [2] It reduces the risk of people's work and brings many conveniences, so as to improve people's quality of life. Therefore, intelligent robot is one of the hot spots of scientific research. Common products in the market are: UAV, sweeping robot and smart home. In our project, we try to make robots complete certain tasks and become intelligent to some extent. Due to the impact of the COVID-19, the overall design changed from hardware to Webots simulation. The overall completion process is also different from the traditional intelligent robot design.

The remainder of this report is organized as follows: first, an overview of the project, including detailed description of tasks and the design workflow, is given in Chapter 2; then, we discuss our project management skills in Chapter 3 ; next, the subsystem design is introduced in Chapter 4 from five scopes; Chapter 5 is about the evaluation of the integrated system; finally, we discuss future research directions, together with a conclusion in Chapter 6. Other information, including the details of individual contribution can be found in the appendices.

# Chapter 2

# Overview

A comprehensive overview of the project is given in this chapter. Firstly, an in-depth examination of the five given tasks is made. In the second part of this chapter, we briefly describe our designing workflow for dealing with these tasks. Then we introduce our proposed solutions under the guidance of it.

## 2.1 Project Overview

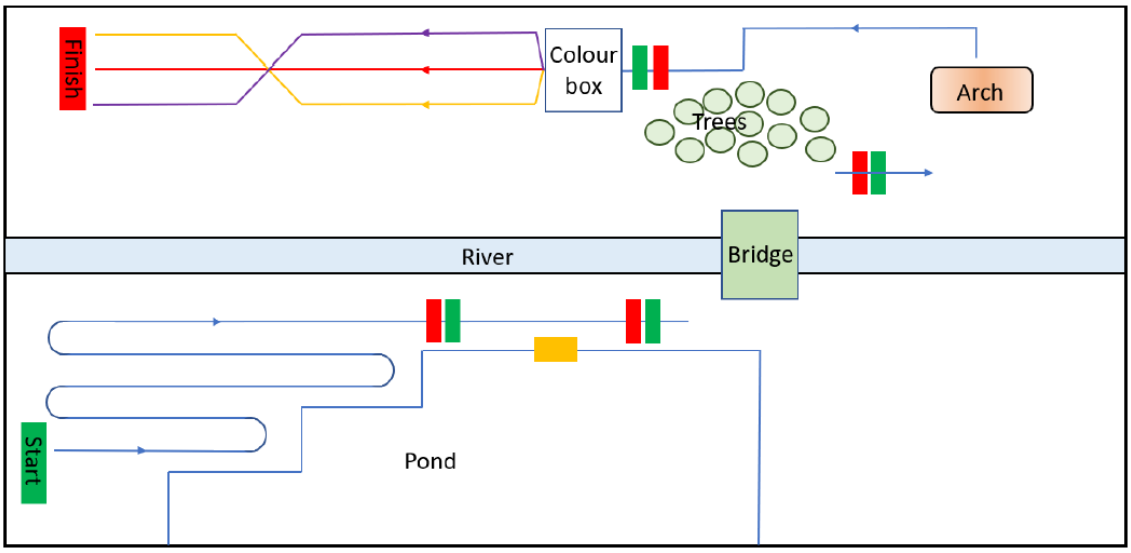



Figure 2.1: Five tasks marked by Green and Red boxes

According to the course specifications, our proposed design is expected to accomplish the following five tasks. In the first task, the rover starts from the green box as indicted in Figure 2.1 and then move along the lines until it reaches the first red box. The second task is from the second green box to the second red box, between which the rover should recognize the orange fish tank and drop the fish food right onto it. In the next task, the rover is expected to move autonomously, crossing the bridge and turning right before crashing into the forest. Then, the rover should pass the arch and then follow the lines to the starting point of the Task 5, where the rover is required to recognize the color of a randomly initialized color box, and follow the path with the same color till the red box at the end.[1]

There are some specific requirements for this project, which are listed in Table 2.1.

[1]Note: The green and red boxes are only for illustration. They do not really exist in our design

| Object | Specification | Note |
|---|---|---|
| patio | 100 m × 30 m | length × width |
| rover | 50 cm × 50 cm | maximum size |
| bridge | 100 cm × 3 m | width × length |
| arch | 100 cm × 100 cm | width × height (suggested size) |
| beacon | no more than 2 | |
| sensor | no number limitations | |

Table 2.1: The project specifications

## 2.2 Designing Workflow

During the project development, we adopt an interactive designing strategy. The whole process is shown in Figure 2.2. Both the Visual and the Chassis Group firstly search for possible solutions and have them examined by the Decision Group. The Decision Group then choose optimal ones from them and split the whole project into small achievable subtasks. If the provided solution does not fit the requirement of the Decision Group, it will be rejected and modifications will be made to improve it. Our program is developed with Python.

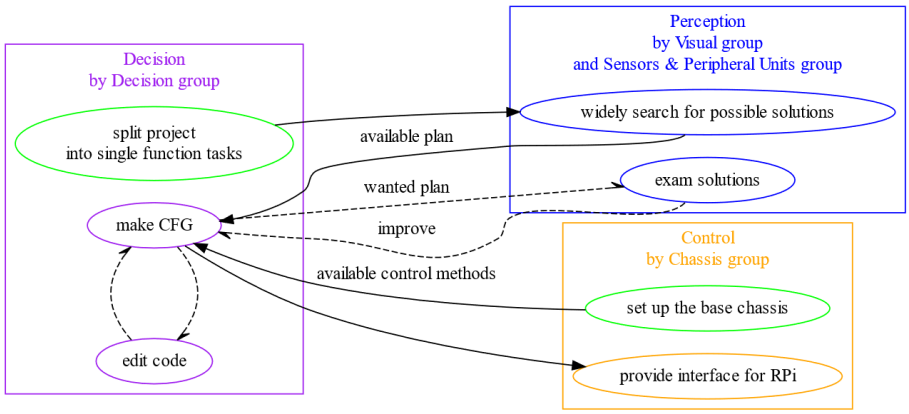



Figure 2.2: The development strategy

With a joint effort of the three main groups, an overall design is finally made. Here, we introduce it task by task.

- **Task 1:** When the path camera identifies the path, a "Path_Direction" signal of the angle between the car and the path direction could be sent from visual group to decision group. Then, a "Turn" command with the angle are sent from decision group to chassis group, leading the car to drive following the line.

- **Task 2:** When the path camera recognizes the orange box, the "Beacon" signal which is "tank" will be sent to decision group. The bool "isFeeded" turn to True to promise the "Feed" command could only be sent to chassis group for one time.

- **Task 3:** When the line cannot be found for more than 7 second, the "Path_Direction" signal will be replaced by "Direction_x", the angle between the car and x axis, which is measured by compass. After the left camera identifies the bridge at the center of view, the "Direction_x" signal will be replaced by "Direction_-z", leading the car turn to aim to the bridge. The, the "Turn" command with the angle with z axis are sent together to chassis group, helping the car crosses the bridge along z axis. After the path camera recognizes the green beacon behind the bridge, the "Beacon" signal which is "after bridge" will be sent to decision group. Then, the "Direction_x" signal takes place of "Direction_-z". As a result, the car could run along x axis again.

- **Task 4:** When the left camera identifies the gate at the center of view, the signal "Gate_Direction" could be received by decision group. "Turn" command with the angle of "Direction_-z" are

sent together to chassis group in order to let the car turn to the gate and drive ahead to cross it. After crossing the gate, the line can be seen again by the path camera. Therefore, the "Path_Direction" signal leads the car again to drive along the line.

- **Task 5:** When the car drive to the start of color line, it recognize its color first. Then we filter input images from the path camera by preserving this color only. In this way, this task could be simplified like the Task 1. Although the line may be separated at the intersection, we use the lost_count signal to count for 7 seconds to ensure the car can keep driving along the line until it reaches the real finish position.

# Chapter 3

# Project Management

This chapter is about the management skills employed during the development. We first introduce our division of work as a result of the Statement of Work and the Work Breakdown Structure. The development strategy is also discussed in this section. To enhance the work efficiency, we utilize two online platforms, ZenHub and GitHub, for the schedule management and version control respectively.

## 3.1 Division of Work

In order to divide the project into achievable tasks and assign tasks to the most suitable team member, the first step of the project management is to do the Task Orientation so that the requirements are able to be achieved efficiently and completely.

In the part of task orientation, we have strictly obeyed the process of the project management. First of all, the Statement of Work (SoW) is made to convert the whole project into realistic and achievable goals. Also, it has the function of establishing expectation of the final result. On top of the SoW, a Work Breakdown Structure (WBS) is needed to divide the whole team into several groups. The function of the WBS is to figure out all the works required to be done to complete the project and structure the work into logic components and subcomponents. The individual responsibilities can also be assigned.

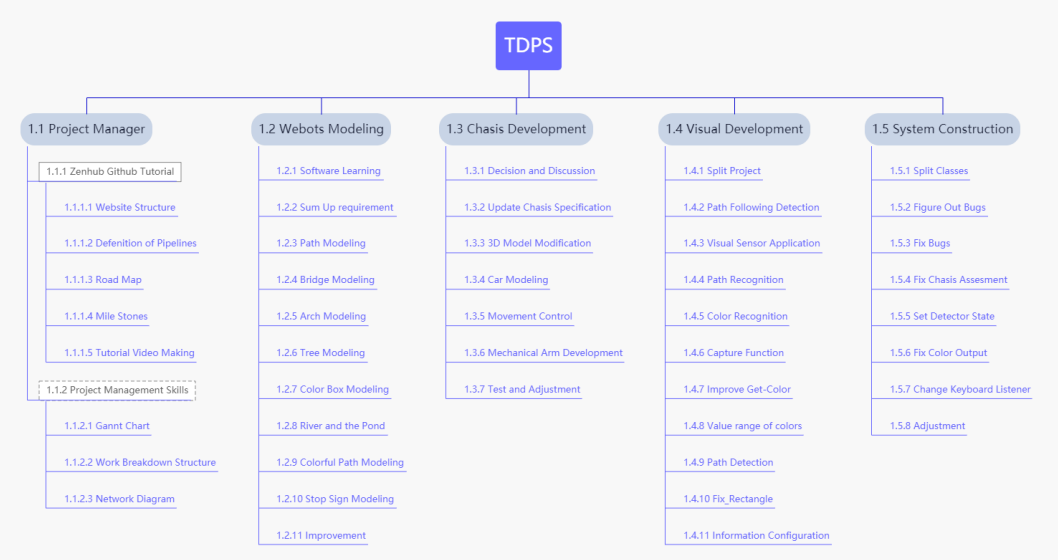



Figure 3.1: The Work Breakdown Structure

Our WBS is shown in Figure 3.1. According to it, we form 5 technical subgroups, including the Environment Group, the Visual Group, the Decision Group, the Chassis Group and the System Group, and one Documentation Group. In addition, we assign a project manager and a tech leader

to take responsible for progress control and tech support. For a detailed version of personnel division and individual contribution, please refer to the Appendix A.

## 3.2 Schedule Management & Version Control

### 3.2.1 Schedule Management

Due to the outbreak of the COVID-19, this project is forced to be finished online this year. Hence, reasonable scheduling to track the progress of the whole project becomes an indispensable part of the project management. ZenHub is an excellent web application for team work, on which we decide to base the development of our project. Figure 3.2 is a screenshot of the ZenHub board, which is made up of seven pipelines. They are New Issues, Epics, Help Wanted, In Progress, Backlog, Bugs and Closed.

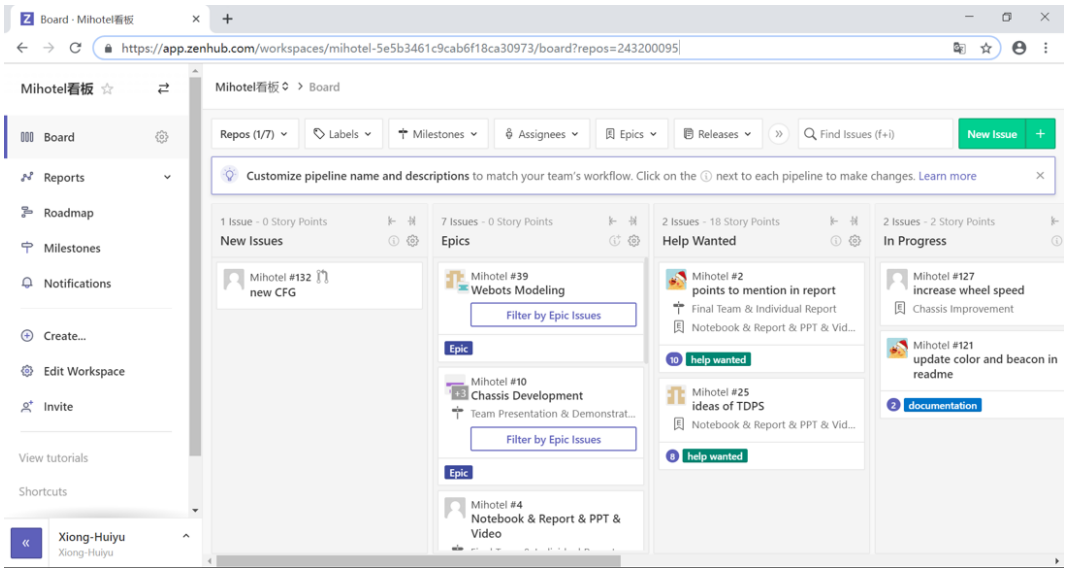



Figure 3.2: The user interface of ZenHub

ZenHub Pipelines are built on the basis of GitHub repositories and each "card" on the Board represents a GitHub Issue or Pull Request. Once a issue is newly created, it will appear immediately in the New Issues pipeline before moved to other regions. ZenHub Epics bundle similar groups of Issues together, providing a visual progress bar of work across related or dependent Issues. This panel can be regarded as a group panel, each group will have their own Epic panel containing their design tasks. After it follows the Help Wanted, which consists of problems requiring suggestions or solution from each member of the team. The next pipeline is In Progress, indicating that someone in the team is dealing with issues or pull requests inside it. The Backlog pipeline contains issues needing to be completed before the end of the project but not of the highest priority at present, or not able to be done without enough accumulation. After one issue is finished, it goes directly into the Closed pipeline. Team members can browse this pipeline to check what they have done so far or reopen specific issues if they find some problems with these issues later on.

Another important feature of ZenHub is the Roadmap, which is a in-built Gantt Chart automatically generated by ZenHub. As we discuss before, ZenHub groups issues using Epics. For example, issues proposed by the Visual Group are all tagged the Visual Development Epic. As is shown in Figure 3.3, different epics are represented by blocks of distinct colors. When creating a issue, we need to set the estimate time to finish it. Thus, the length of each colored block stands for the sum of estimate time of all issues in this epic. The black line in each colored block tracks how many issues are resolved within this epic. If no issue is left, the black line will have the same length as the colored block. The Roadmap is beneficial to our schedule management. We effectively track our progress by checking whether the black line in each epic is aligned with today's date. In short, the Roadmap contributes a lot to our schedule management, offering a broad picture about the project progress.

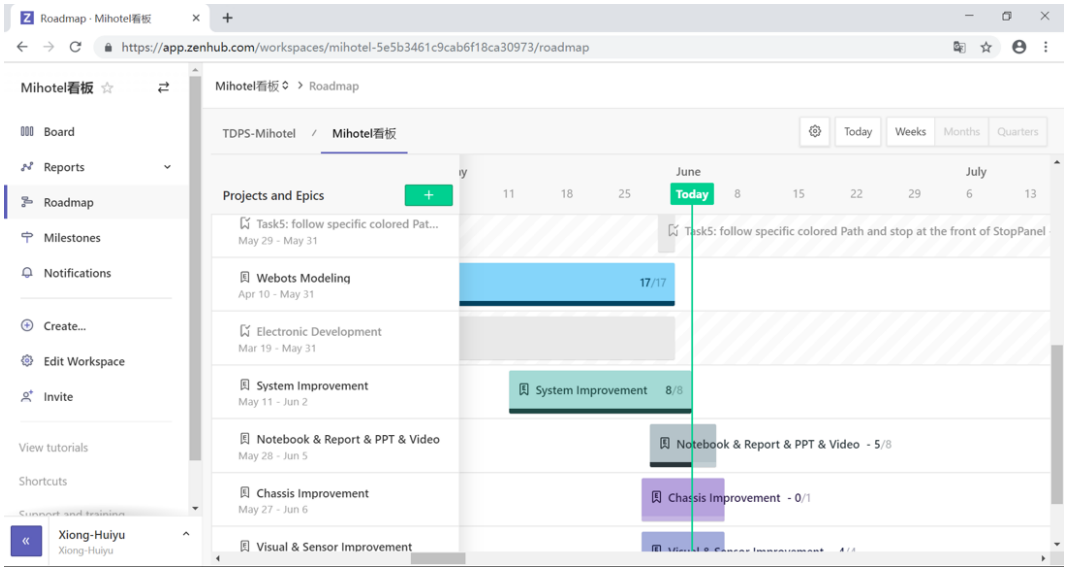



Figure 3.3: Roadmap: A simpler version of the Gantt Chart on ZenHub

### 3.2.2 Version Control

During team development, another frequently occurring problem is the version conflict. Supposing two team members are editing the same program, chances are that the one of them is unaware of changes made by the other. In that case, the code will not work if they directly merge two programs together. What's more, it is also time-consuming for people to merge codes manually. With an attempt to avoid this problem, we base our development process on GitHub.

There are two key benefits of using GitHub. One is the discussion board within each issue, which provides us with a unique platform to communicate with team member. The other is the Pull Request. Figure 3.4 illustrates the use of Pull Request. The black line on the top represents the master branch, which is the base version of our codes. In order to tackle different issues independently, several branches are created. They are the colored lines in Figure 3.4. Once the assignees solve issues in each branch, request will be pulled to merge this branch into the master. Reviewers are responsible to check if their requests really fix the problem.

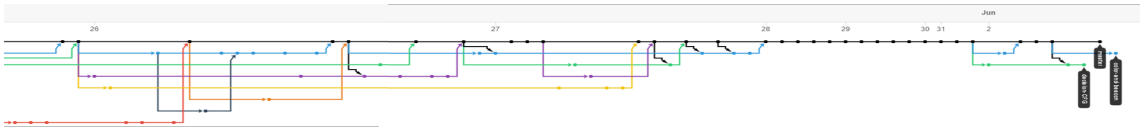



Figure 3.4: Requests merged into the master branch

As we mention before, the issues can be viewed as small cards in different pipelines on ZenHub. Similar to cards, we can write our ideas on them. Figure 3.5 takes one issue during our development as an example. It share the same strength as the E-mail, keeping important dialog between team members. If someone want to check some details about the design, he can find corresponding notes easily in this discussion board.


Figure 3.5: Discussion board of issues

# Chapter 4

# Subsystem Design

In this chapter, we discuss the subsystem design from 5 scopes. In Section 4.1, the construction of environment in Webots is introduced. Then we present the visual processing in Section 4.2. The decision making part comes after it in Section 4.3, followed by the rover modeling in Section 4.4. The system architecture comes last in Section 4.5.

## 4.1 Environment Building

The construction of environment is the first step of the project. Only based on a high degree of reduction environment, could the developed rover's functions truly meet the requirements. The construction of environment can be divided into six steps: (1) learning Webots tutorials[3]; (2) sorting out project requirements; (3) selecting model; (4) building model and selecting alternatives; (5) organizing component location; (6) adjusting parameters. There are two members in the Environment Group, Huiyu xiong and Jinwei Chu, who deal with requirements sorting & components selection and model Construction & parameters adjustment respectively.

### 4.1.1 Requirements sorting & Components selection

Author: Huiyu XIONG, UoG ID: 357680X

As the basis of intelligent rover project, environment is a very important part. Because an untrue environment may cause some problems in the R & D stage of the project and then affect its practical application. In the environment, any trace element may have a great impact on the project design. For example, light intensity will affect the selection of visual sensor and algorithm, etc. In this case, considering the real reduction environment and careful analysis of project requirements, it is necessary to organize and summarize the project model.

The first is to interpret the project requirements. From the TDPS Course on Moodle 1, we downloaded the requirements for this project and the latest supplementary requirements for the online simulation project. Combining the information of these PDF files with the supervisors' reply to the students' questions on the MS team, we sorted out the overall requirements of the environment: to build a world marked with size as shown in Figure 4.2, in which some patrol lines are drawn from start, so that the rover (50 cm x 50 cm) can follow. Next, there is an orange beacon or box on the edge of the pond, where fish feed. Additionally, the bridge (100cm(W) x 3m(L)), including the ramp used to get up and down the bridge, will be aligned with the edge of the pond in the middle. After crossing the bridge, several trees scattered randomly. Avoiding it, the gate (slightly larger than the rover, the recommended size is (100cm (W) x 100cm (H)) can be found by beacon. Then, go through it and follow the line on the ground. Move to the color box. After the rover reads the color of the box, the rover should be able to follow the correct color until it reaches finish.

According to this requirement, we drew out several important models to build the world: line patrol; pond & river; bridge; gate; trees; beacons and some invisible conditions. By comparing the requirements of each node and project, we have found several suitable components.

The first is line patrol, which is composed of four curved road segments with different radii and five straight road segments. The different curve design is to make the project more difficult, so as

to verify the powerful patrol function of our rover.

In order to build a real pond and river, we choose the fluid node. The yellow Duckling floating on the water in Figure 4.1 reflects the buoyancy of liquid water, which shows that our environment is close to reality. One more thing to be mentioned here is that this duckling causes no interference to the recognition of bridge, which verifies the robustness of our visual algorithms.

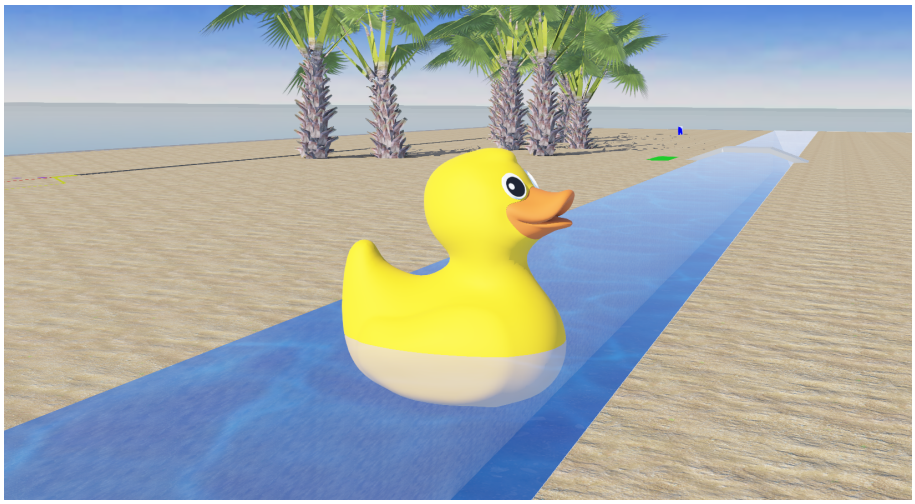


Figure 4.1: The floating yellow ducking

After the choice of sandy land, the choice of forest is of course the palm tree. In order to reflect the strong visual recognition ability of rover, we use the beacon as little as possible in the whole project. Color boxes are considered as beacons, which only use three places in total: Orange boxes for fish; green boxes after crossing the bridge and color boxes with undetermined colors in the final task.

Figure 4.2: The construction of palm trees

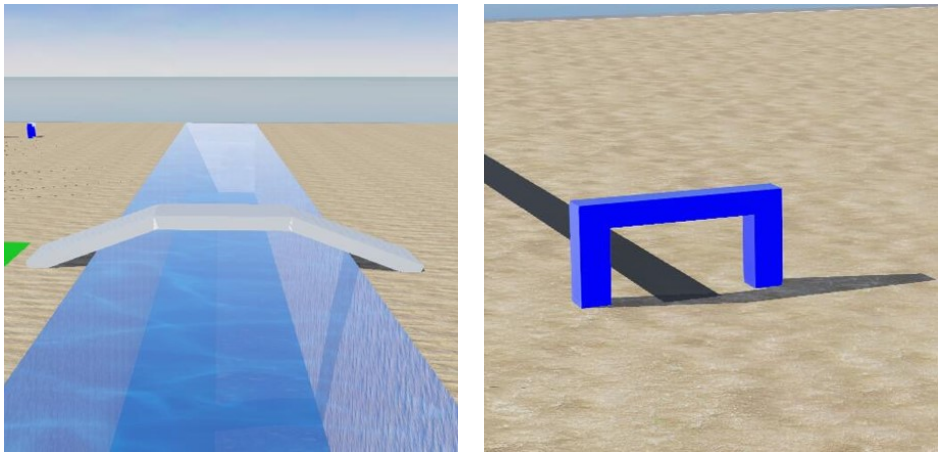
After browsing all the nodes, we found that it is difficult to find the appropriate nodes for components with special shape and size, such as gates and bridges. The specific solutions to this part will be described in the next section written by Jinwei Chu.

Finally, it is worth mentioning some invisible conditions, such as background conditions: light, background ocean and ground. We use "noon-sunny-empty" here. In order to simulate the intensity of sunlight at noon. It is said that light intensity will affect visual judgment, so we want to consider this extreme environmental impact factor. On the periphery of the environment, we added an ocean. The world itself is covered by solid in the sand. Together, it forms a picturesque summer island.

### 4.1.2 Environment Modeling & Parameter Adjustment

Author: Jinwei CHU, UoG ID: 2357643C

The key part of the environment modeling is the construction of the bridge and the gate. After seeking for the substitution of the bridge and the gate, we found that there is not a satisfying substitution for these two articles in the repository. Finally, we decided to build our own model to import the model into Webots. Our constructed bridge and gate are shown in Figure 4.3. In order to meet the requirement of the handbook, we set the width of the bridge to one meter and the length of the bridge to three meters. In order to enhance the precision of the detection of the gate, we set the gate just as large as the size of the rover, which is about 50 centimeters high and 30 centimeters wide.



(a) The bridge (b) The gate

Figure 4.3: The construction of the bridge and gate

There are also some other items needed to be modeled in Webots. In task 2, the rover is expected to throw the fish food onto an orange box, where it can slide into the pond. As is shown in Figure 4.4, we put a slope onto the top of the fish box so that the fish food can be put onto the slope and then it can roll-off the slope automatically.

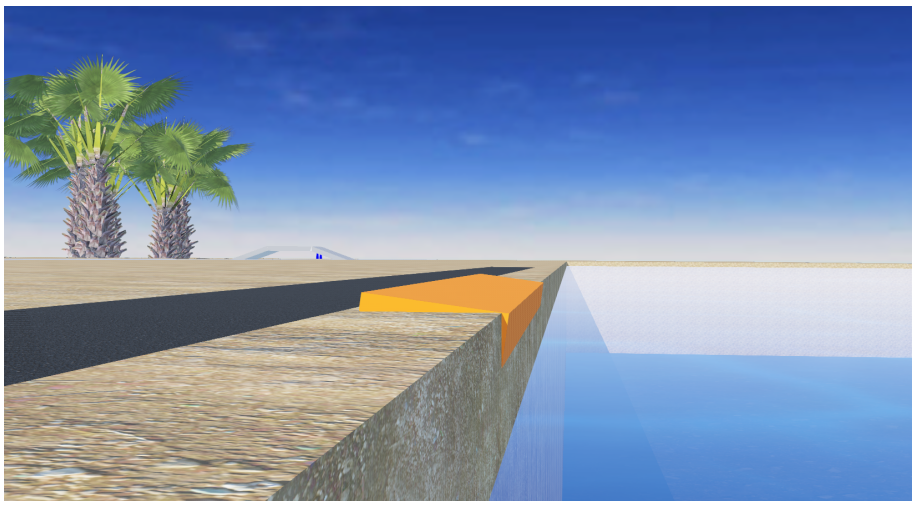


Figure 4.4: The orange box with slope

On top of the orange box, another color box is required to be built in order to complete the Task 5. We set the color paths into three different colors: purple, yellow, and red. The width of the colorful paths is different from the width of the road, which is about 0.1 meter wide. We do this deliberately to test the robustness of our path finding algorithm. Our constructed color box

and path is shown in Figure 4.5.

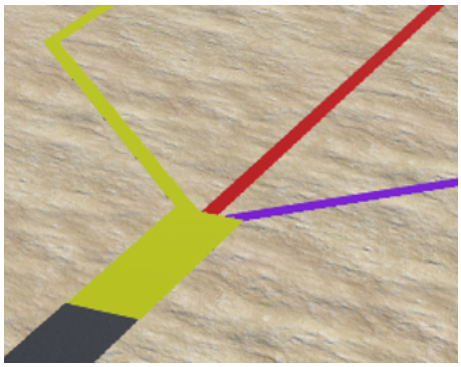


Figure 4.5: The constructed color box and path in Task 5

In summary, the advantages of our design are listed below:

- In order to make the environment more realistic, we converted the texture of the floor into sand, which looks just the same as the ground near the east lake of our school.
- In order to modify the texture of the ground, we use this several boxes to form the ground rather than use the default ground.
- The water used in our environment is constructed with the node fluid. In order to show its physical property, we put a rubber duck on the water, which can be seen that the rubber duck is floating on the water.
- In order to test the stability of our system. When we were constructing the road, at the corner of the road, we used different radius so that if the rover can turn left or turn right successfully with different radius, it shows that the stability of our system is truly good.
- In order to throw the fish food successfully into the pond, we put a slope on the top of the orange fish food box so that the food can be thrilled successfully into the pond.
- Also, in order to test the stability of power system, the gate is just as high as the height of the rover, and also the width of the gate is designed just as wide as the rover.

## 4.2 Visual Processing

Serving as "eyes" of the Mihotel Rover, the main work of our visual system is to perceive information from external environment and feed it to the decision module, where instructions are made to control the movement of the rover. To ensure a robust perception of the external information, our team propose to deal with the visual processing in a gradual, goal-oriented fashion. Our goal is to choose the most suitable sensors to finish the five tasks smoothly. In particular, the rover need to find the path and detect objects such as the bridge and the gate.

However, the selection and implementation of sensors is dependent of our solution to do the tasks, so we develop the path finding and object detection algorithms ahead of it. Before developing algorithms to do the detection, we do the image pre-processing first to obtain and remove the color information of the captured image. Three members in the Visual Group, Chang Shu, Bo Wen and Haoran Han are responsible for them respectively.

### 4.2.1 Beacon Detection & Color Filtering

Author: Chang SHU, UoG ID: 2357702S

Similar to the real-world scenario, the world in Webots is riotous with color. The color is a special attribute to every item in the world and is of vital importance to our project. To be specific, the color helps us identify the fish tank and the beacon. On top of this, we also utilize color information to follow the right path in Task 5.

However, the color can sometimes be problematic. When constructing the environment in Webots, we not only set the scene, but try to make it true to life as well. Our target is to develop a robust visual system that has as good performance even in the real world. The color variation caused by the reflection of the ground, the shadow of the sunlight are taken into consideration.

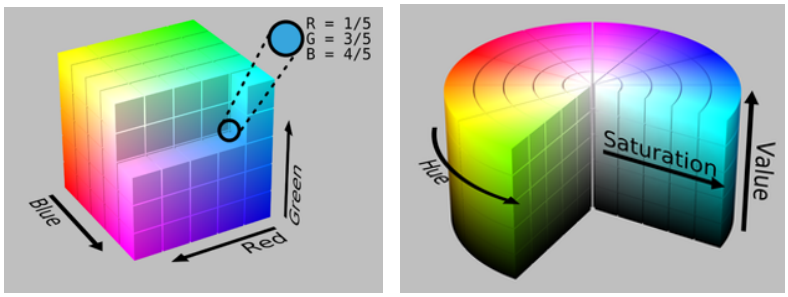
(a) Detection of fish tank with orange box

(b) Detection of turning point with green box

Figure 4.6: Beacon detection with color

To make full use of the color information and reduce its side effects on our path finding to the least, we design a **beacon detection algorithm** and a **color filtering algorithm**. As is illustrated in Figure 4.6, the beacon detection algorithm looks for interested colors and returns corresponding signals when they are detected. In contrast, the color filtering does not care about the color information. It processes the unsmooth image captured by the camera and converts it into the gray image (path with black line) or binary image (path with color in Task 5) so that our path finding algorithm only needs to deal with non-color images.

The first problem to be settled is in which color space should we capture and then remove the color of images. There are some forms of color space, such as RGB (an abbreviation for Red, Green, Blue) and HSV (an abbreviation for Hue, Saturation, Value), where we can do the image processing. Although RGB is the most commonly used color space in the digital image processing, we choose to do the image processing with color in HSV space. This is because the HSV color space has the strength of separating color apart easily.




(a) RGB space

(b) HSV space

Figure 4.7: Illustration of two color spaces

As is shown in Figure 4.7, the RGB color space is in the rectangular coordinate system while the HSV color space is a cylindrical-coordinate color model. A typical color in RGB space is a weighted sum of three-primary colors, which is not intuitive. However, in the HSV space, we can find the range of each color by determining its hue first, then the saturation, and finally the value. This makes it easy for us to fix the threshold of different colors. The range of each color is found using Python GUI.

Another issue is to remove noise in the unsmooth images. As is mentioned before, the color is not a fixed value in the real-world scenrio. Even a change in the angle of sunlight may influence the range of colors. Also, the dust on the path will make it difficult for our path finding algorithm to successfully detect the path. Therefore, we propose to smooth the images with the low pass filter (LPF) and remove the noise with the opening operation. Among a wide range of LPFs,

including average filter, Gaussian filter, median filter and bilateral filter, we select the Gaussian filter empirically[4]. The opening operation is developed from mathematical morphology, which is an erosion followed by a dilation. The erosion is used remove unwanted small pixels, like the noise in the image, while the dilation can fill the small gaps.

To mimic the reality and ensure the robustness of our visual algorithms, We manually add some Gaussian noise to our images during experiments. In Figure 4.8, we convert the HSV image of roads in Task 5 into binary image with only the color box and corresponding path preserved. Unfortunately, the thresholding cannot exclude the white noise efficiently. Hence, Gaussian filter and opening operation are applied to deal with it.

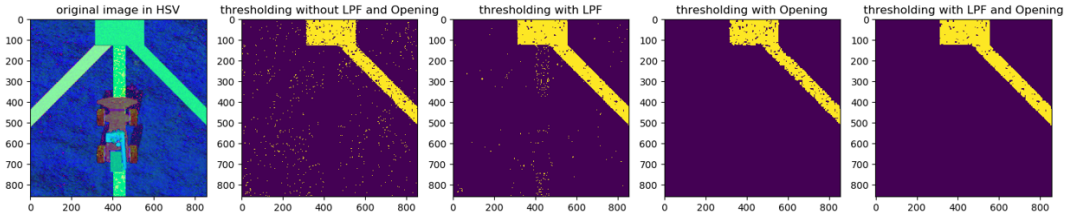



Figure 4.8: The pre-processing on the binary HSV image with 1% Gaussian noise

As we can observe, the opening operation does a even better job than the LPF, successfully removing all the noise in the image. However, the integrity of the wanted color suffers slight loss. This problem is more significant in Figure 4.9, where the color box and path directly disappears after the opening operation. This is caused by an increased amount of noise added to the image, which makes region of the wanted color discontinuous.

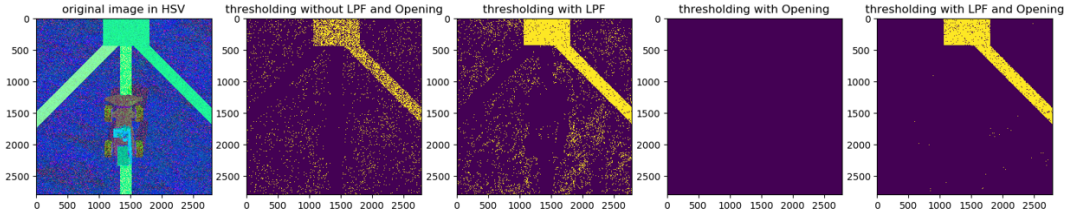



Figure 4.9: The pre-processing on the binary HSV image with 5% Gaussian noise

The problem above is easily solved with the help of the LPF. As is discussed above, the Gaussian filter provides no better results than the opening operation. The reason is that the Gaussian filter actually works on the whole color space rather than the interested color. However, this does not mean the LPF is useless. In fact, it prevents the noise from making the region of the wanted color become discontinuous. The last image in Figure 4.9 shows that a combination of LPF and opening operation proves effective, both removing the noise and avoiding over-filtering.

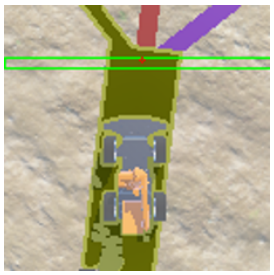


Figure 4.10: The principle of color path following in Task 5

Finally, we are able to obtain a clear and clean binary image. To explain the principle of how we do the task 5, this is added to the original image in Figure 4.10 as a mask. Although the input to the visual system is the RGB image, our rover is actually following the color filtered line in the binary image.

### 4.2.2 Path Finding & Object Detection

Author: Bo WEN, UoG ID: 2357658W

As is discussed in Section 2.1, in Task 1 and 5, and the period between Task 4 and 5, the rover needs to detect and follow the path on the ground; In task 2,3 and 4, the rover needs to detect specific objects and react correspondingly at the correct location. We offer solutions by developing a **path finding algorithm** and a **object detection algorithm**.

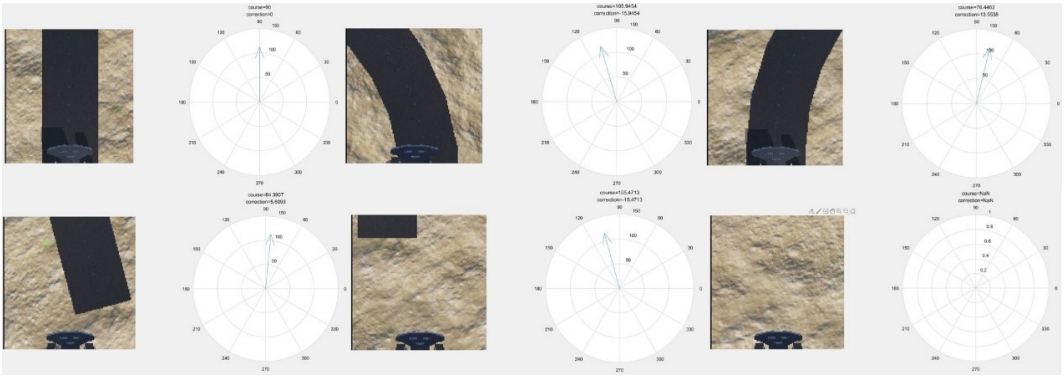


Figure 4.11: path finding by outputting the angle of the trajectory

To give an accurate and effective instruction to the rover of where it should head towards, we need to both detect the trajectory of the path and analysis the relative position of the rover to the path, then calculate an optimal correction angle for the Decision Group. With the help of the color filtering algorithm in Section 4.2.1, the RGB images are converted into smooth and noise-free gray (for black path) or binary (for path with color) images. Based on the grayscale feature of images to identified path pixels in the image, we vertically divide the image into several segments and calculate the geometric center of path in each segments, which form the trajectory of the path[5]. Then, we determine the geometric relationship between the path and the rover by both the morphology of the path and the relative position to calculate an optimal target pixel. Consequently, our algorithm is able to give useful instructions to the rover under various of scenarios. The results are illustrated in Figure 4.11.

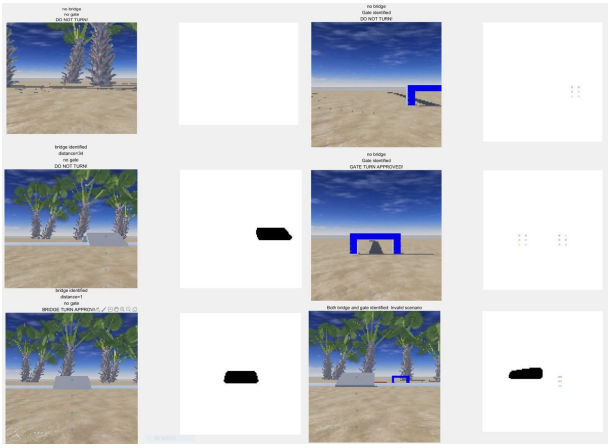


Figure 4.12: The illustration of the bridge detection (left) and the gate detection (right)

In order to give the rover corresponding instruction at the correct location in task 2 to 4, our object detection algorithm involves two parts: object detection and location acquisition.

1. **Object detection**:
   We detect the objects in tasks 2 to 4, namely the fish tank, the bridge and the gate, by some of their most significant image features, respectively. The fish tank is already detected by the beacon detection algorithm in Section 4.2.1. The bridge is detected by the grayscale

feature and the gate is detected by its gradient feature on the edge of its pillar. However, judging whether the object is detected from sole feature of these objects can sometimes be inaccurate. Thus, we further process these features by matrixes computation, including erosion and columns unification, etc. to extract pixels in the image that truly belongs to our target.

2. **Location acquisition**:
   We calculate the geometric center of the object according to its extracted features, and compare it with the center of our rover (which is calculated by the camera location) to decide whether the rover is at correct location to execute the next order.

Finally, our rover is able to successfully detect the objects and return corresponding signals so that the Decision Group can give further instructions to control the rover.

### 4.2.3 Sensor Selection & Implementation

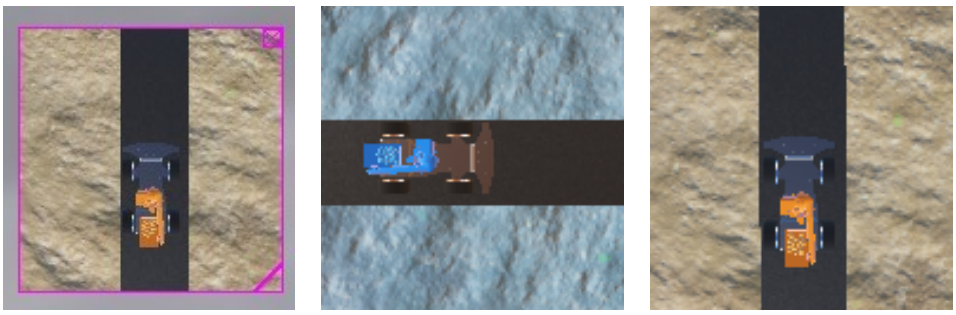
Author: Haoran HAN, UoG ID: 2357662H

In order to accomplish all algorithms described above, we need to choose the right sensors to capture the information from the environment so that the rover could make decisions according to the environment.

At the very beginning of this project, five types of sensor have been tested, including Distance Sensor, GPS, Accelerator, Compass and the Camera.

The **Distance Sensor** could provide the distance between the rover and the nearby object. The maximum distance that this sensor could detect and the corresponding precision could be modified in the "LookupTable" property. This sensor is originally designed to detect trees after the rover passing the bridge. The **GPS** could provide the position of the rover at the coordinate of the whole world. The returned value is the same as the "translation" property of the rover. Moreover, as the name indicates, the **Accelerator** could return the acceleration of the rover, both the magnitude and the direction in the Cartesian coordinate form. However, as the project developed, team members come up with more brilliant method. In that case, the position, speed, acceleration and the distance between the rover and the environment becomes unnecessary. In that case, even though these three sensors have been tested thoroughly, they are deleted in the final project design.

The **Compass** will return direction where the x-axis of sensor pointing to. By modifying the x-axis of the Compass to the same as the head of the rover, we could provide the moving direction of the rover in real-time. However, the original returned value is in the Cartesian form which is not easy for the decision group to analysis. In that case, the direction is transformed to the angle that the moving direction deviate from the x-axis and negative z-axis by applying the "arctan" function.



(a) Picture from Webots GUI (b) Picture before processing (c) Picture after processing

Figure 4.13: The comparison among pictures before and after the processing

The most important sensor in this project is the **Camera**. The direction that the camera scene towards is the along the negative z-axis and the upper while the upper direction of the camera is along the y-axis. By modifying the "height" and "weight" property, the number of pixels could be changed. However, there exist some problems with the camera: (1) the direction of the picture

returned by the python function is not accord with that shown in the GUI; (2) the returned picture is in the RGB form which is not with the BGR required by the opencv package; (3) there is a thin black line (where its RGB channel are all zeros) which could negatively influence the path and object detection algorithm. In that case, the rotation, reshape and crop operation is added after the Camera returning the picture. The comparison among pictures before and after the processing is shown in Figure 4.13. To make it easier to test our path finding algorithm, we define a "capture" function to save the image.

The final design utilizes two Cameras plus a Compass. One Camera towards down directly. This camera is named as "path_cam", and it is used for the **color filtering algorithm** and **path finding algorithm** developed before. Another camera named as "left_cam" which towards left is mainly used for the **bridge detection and gate detection algorithm**. The compass provides the moving direction which will serves as the guidance when there is no path. As illustrate before, we set two direction parameters. This is because when the rover turns to the bridge or the gate, it needs to turn to the negative z-axis while when the rover avoids the tree, it needs to turn to the x-axis. That's why these to direction is selected.

Table 4.1: The detailed information of signals processed by the Visual Group

| **Signal Name** | **Type** | **Description** |
|---|---|---|
| Direction_x | Float | Range from -180 to 180. Return the moving direction of the rover that deviates from the x-axis. Left deviation returns negative value, right for positive one |
| Direction_-z | Float | Range from -180 to 180. Return the moving direction of the rover that deviates from the negative z-axis. Left deviation returns negative value, right for positive one |
| Beacon | String | Return "tank detected" when the orange is detected; Return "after bridge" when the green is detected |
| Path_Color | String | Return the color of the path in Task 5 |
| Path_Direction | Float | Return the moving direction of the rover that deviates from the path. Left deviation returns negative value, right for positive one |
| Bridge_Detection | Boolean | Return "True" when the bridge is detected |
| Gate_Detection | Boolean | Return "True" when the ate is detected |

Overall, we explore a variety of sensors to take advantage of our proposed algorithms and complete the tasks. The final solution is only using two cameras and one compass for the sake of simplicity.

## 4.3 Decision Making

The Decision Group works as the "brain" of the Mihotel Rover, receiving and processing the signal information from the Visual Group and send commands to the Chassis Group to lead the rover finishing the whole mission. Our work can be divided into two parts, task identification and decision code writing. There are two member in the Decision Group, who are Zijian Wang and Hanpeng Xu. Their work is introduced separated in this section.

### 4.3.1 Task Identification & Path Following Decision Making

Author: Zijian WANG, UoG ID: 2357660W

As for the task identification, in order to accomplish the whole process accurately and efficiently, at the beginning of the project, we check our mission and decided to make our initial flow chart. It can help us clear what we need to achieve and what information we can get and use from the project requirement. In the project process, especially when there are several people do the co-operation, the flow chart can significantly increase the readability of the code, help us clear the logic and debug.

The Graphviz really helps us a lot in finishing and improving our logic graph step by step. Comparing with traditional freehand sketching to record our work, the Graphviz has its natural

advantages because it takes descriptions of graphs in a simple text language and make diagrams in useful formats. It means that we can easily correct our wrong scheme and add details directly to the flow chart by changing our code.

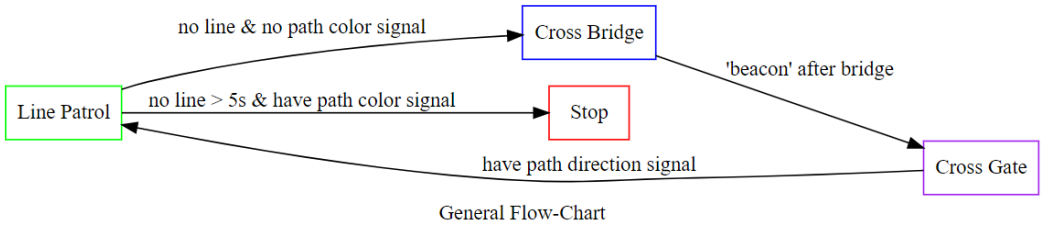



Figure 4.14: The general flow chart

As is shown in Figure 4.14, the whole process is finished by three states of our rover, line patrol, crossing bridge and crossing gate. The connection relationship and trigger condition can be seen clearly in our GFC. The 3 states include 5 subtasks in the whole process, normal line patrol, feeding, crossing bridge, crossing gate and the color line patrol. Through our merger and simplification, we use the streamlined code and as few as possible states to finish our project. And the transition conditions between each two states were discussed again and again with the Visual Group.

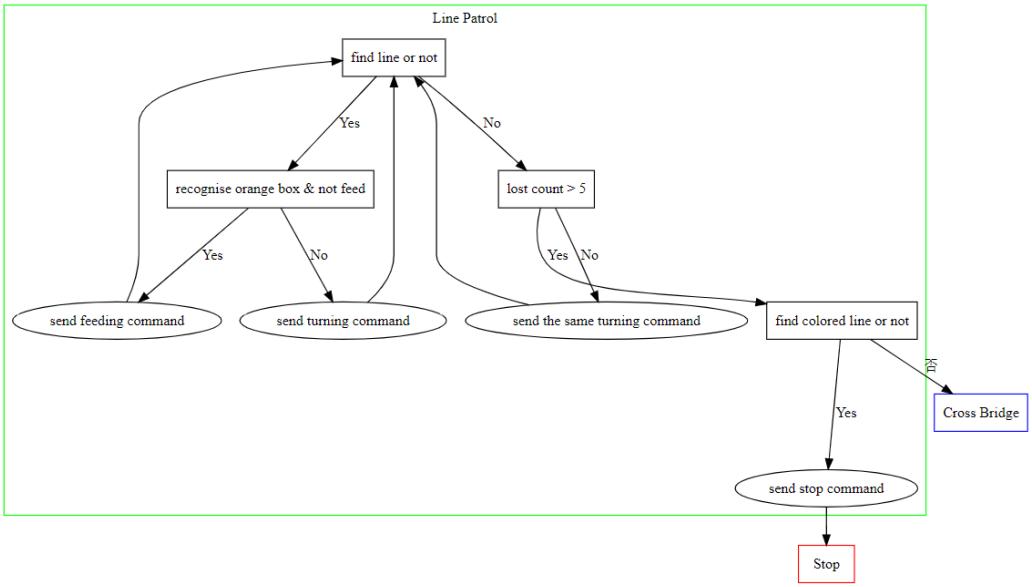


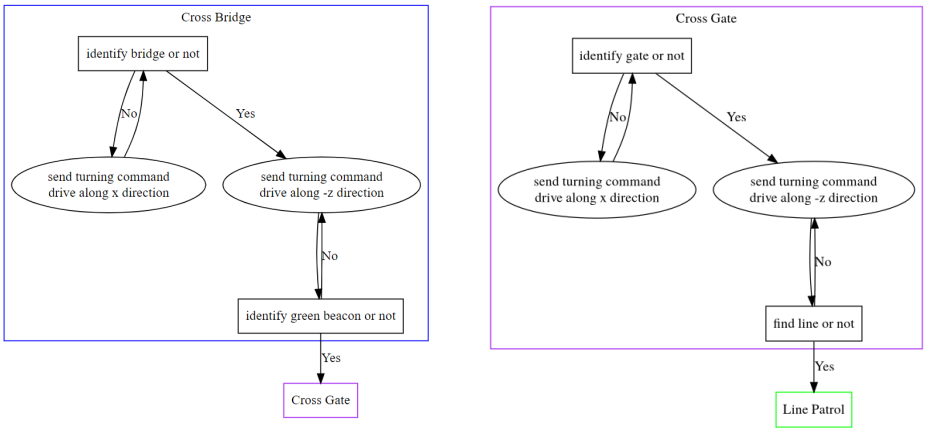

Figure 4.15: The line patrol logic graph

In the line patrol logic graph shown in Figure 4.15, several important signal judgements can be noticed. We design them as the key nodes to send correct commands to our chassis. “find line or not” is the basic condition for the rover to follow the line direction from the camera image. The orange box signal from visual group and the “no feed” status record work together to guarantee it can execute feed command at orange box for just one time. And the color signal helps us to know whether we have finished our whole process or not. The lost count signal is one of the key designs of our decision group. Even though the color line may be separated at the intersection, the rover can still keep driving ahead until it reaches the real finish position. Because of the tight logic design and detail correction again and again, our rover can deliver the correct command for each signal accurately.

### 4.3.2 Bridge & Gate Crossing Decision Making

Author: Hanpeng XU, UoG ID: 2357916X

The crossing of the gate and bridge is another focus of our task. In the process of drawing up the flow chart, we overcame many difficulties: (1) How to stop the rover from turning when it came to the bridge; (2) How to prevent the rover from falling on the bridge and make it cross the bridge smoothly. For the first question, after discussing with the vision group, we decided to use the left camera to capture the bridge, so that the camera and the center line of the bridge can be aligned before turning. The second problem was solved by the technical team, who made the rover go wirelessly according to the gyroscope so that it wouldn't fall off the bridge.




(a) cross bridge (b) cross gate

Figure 4.16: The flow chart of crossing the bridge and gate

After finalizing the final version of the flow chart, the focus of our task shifted to translating the flowchart into Python code. In this process the task, the key to the work is communication. In constant consultation with other technical groups, we make decisions based on the information and conditions that the visual and chassis groups can provide. So, we often have meetings with other technical groups during this period. For example, What code structure is used to execute the command, and what is the overall framework of the code. In addition, we will also go to GitHub above reference others code structure is how to build, and then try to run in the Jupyter Notebook.

```
class decision_command:
    def _init_(self,signal):
        self.signals = {
            'Position': [],
            'Direction_x': [],
            'Direction_-z': [],
            'Speed': [],
            'Distance': [],
            'Color': [],
            'Path_Direction': [],
            'time':[]
        }

    def data_filling(self,signal):
        #接收来自视觉组信息
        self.signals['Position']=signal.['Position']
        self.signals['Direction_x']=signal.['Direction_x']
        self.signals['Direction_-z']=signal.['Direction_-z']
        self.signals['Speed']=signal.['Speed']
        self.signals['Distance']=signal.['Distance']
        self.signals['Color']=signal.['Color']
        self.signals['Path_Direction']=signal.['Path_Direction']
        self.signals['time']=signal.['time']

    def data_processing(self):
        if self.signals['Path_direction'] != []


    def standby(self):
        if self.signals['Path_Direction'] != []:
```

Figure 4.17: The decision code

## 4.4 Chassis Designing

The chassis functions as the "body" of our Mihotel Rover. It receives instructions from the Decision Group and control the movement of the rover. There are two member in the Chassis Group. Haotain Wang takes charge of the modeling and importing of the arm and chassis of our rover while Chaofan Shi build the design the control algorithms.

### 4.4.1 Chassis & Arm Modeling

Author: Haotian WANG, UoG ID: 2357667W

In order to model the rover, the main work of the chassis modeling is to model our design in SolidWorks and transfer it to Webots, and add some necessary node to finish the task. We propose a four-fold modeling process, including chassis selection, coordinate assigning, model simplification and texture assigning.

At the beginning, we planed to assemble a real rover. So we browsed on the e-shop and found several choices. They are shown in Figure 4.18. Considering the tasks we have to accomplish, we finally decided to use the four wheel drive chassis(the last one of the following). The chassis had to be stable enough to hold the feeder and cross the bridge. A lower center of mass would be preferred. However, the chassis cannot be too low so that it won't hit the ground when it crosses the bridge. Thus, the other three chastises were abandoned. After we were informed to simulate the task in Webots, we found the 3D model of chosen chassis and proceed the following work.

(a) model one

(b) model two

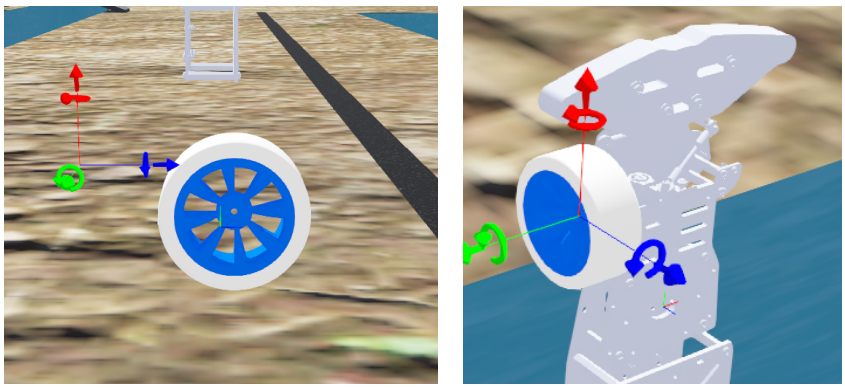
(c) model three

(d) model four

Figure 4.18: The 3D models on e-shops

Before we import the chassis into the simulated world, we had to set the coordinates of each component, especially for the wheels. If the axes of the wheels were not properly assigned. Wheels

won't rotate as expected. We shows the difference between wrongly and properly assigning coordinates to the wheel in Figure 4.19.



(a) not assigned to the wheel (b) properly assigned to the wheel

Figure 4.19: Coordinates assigned to the chassis

The original model contained too much unnecessary details which affect the performance of the simulation. Therefore, we had to delete some less important components while keeping the its appearance unchanged. Simplification is accomplished using SolidWorks. The simplified model is shown in Figure 4.20. The simplified model could run smoothly in the simulation world.

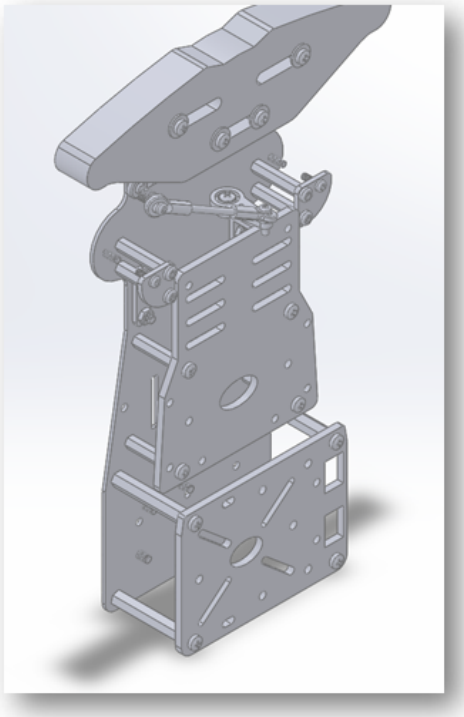


Figure 4.20: The simplified model

The imported models are unpainted at first. In order to make the rover looks more fancy, we set an iron material to its body and rubber to its wheels.

The final specifications of the chassis are listed in Table 4.2. The density of the chassis and feeder is set as different materials. The feeder is much lighter than the chassis. this design is for maintaining the mass center of the rover at its geometry center, so the rover can make turns as expected.

Figure 4.21: The model with texture assigned

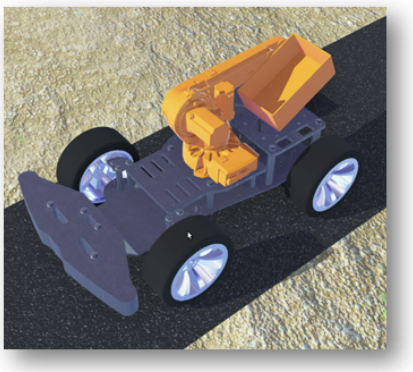


Figure 4.22: The final design

| | **Item** | **Specification** | **Note** |
|---|---|---|---|
| Body | size | 0.15 m × 0.23 m × 0.05 m | width × length × height |
| | density | 7.85 $\times 10^3$ kg/$m^3$ | density of metal |
| Wheel | radius | 0.033 m | |
| | max velocity | 100 rad/s | move forward when velocity is negative |
| | max torque | 100 N · m | |
| | front track | 0.12 m | velocity distance from center is 0.068 m |
| | rear track | 0.12 m | velocity distance from center is 0.07 m |
| | motor control method | velocity control | PID is not used in velocity control |
| Arm | density | 0.8 $\times 10^3$ kg/$m^3$ | |
| | arm length | 0.035 m | |
| | max velocity | 20 rad/s | |
| | max torque | 100 N · m | |
| | holder size | 0.04 m × 0.066 m × 0.024 m | width × length × height |

Table 4.2: The final specifications of the chassis

### 4.4.2 Chassis & Arm PID Control

Author: Chaofan SHI, UoG ID: 2357686S

The chassis controlling of the Chassis Group focuses on solving the controlling problems and writing the chassis controlling algorithms. We break down this into three parts: (1) adjust the centre of mass of the rover; (2) design the arm control algorithm; (3) adjust the PID control of the chassis.

In order to control the rover, the first thing needed to be solved is the rotation problem. When we want it rotate in place, the centre of mass will move around the centre. If the speed is fast, the rover will even fall apart. So, we tried some methods to solve this problem. We firstly tried to adjust the mass of four wheels and the mass of the chassis. Then we found the rover will oscillate greatly, even separate the rover if we increase the mass of the wheel greater than the chassis. Secondly, we tried to adjust the relative position of four wheels to make it a square, and it proves to be useless. Thirdly, we compare the car with 4-Wheels robot in the tutorial[6], and we realized it may only related with the centre of mass. After we finally found that the reason is caused by the centre of mass, we moved the bounding object of the rover and adjust the position to solve the problem. The centre of mass of our rover is illustrated by Figure 4.23.

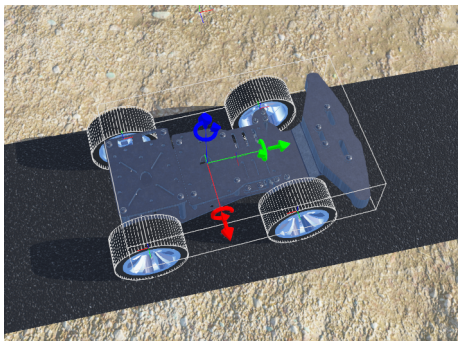


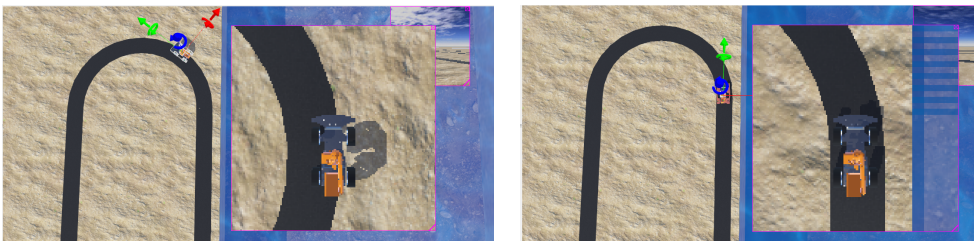
Figure 4.23: The centre of mass of the rover

Next, we need to control the arm to deliver the fish food, and we want to release the fish food both safely and elegantly. In fact, we choose to set its position directly at first, However, in the former version, we choose to throw the food without stop, so we have to set speed and us position limitation, because we can not precisely control the number of loops during a period. Using time delay and the speed control also contains some problems, it can not precisely control the position of the arm, so sometimes the position would be a little different.

```
            commandInfo(self.state)
        elif(self.flag == 2):
            if time - self.feed_start < self.feed_time:
                pass
            elif time - self.feed_start < self.feed_time * 3:
                self.velocityDict['Shoulder'] = 0
                self.velocityDict['Elbow'] = 0
                self.velocityDict['Wrist'] = 0
            elif time - self.feed_start < self.feed_time * 15:
                self.velocityDict['Shoulder'] = -shoulder_vel / 10.6
                self.velocityDict['Elbow'] = -elbow_vel / 10.6
                self.velocityDict['Wrist'] = wrist_vel / 10.6
            else:
                self.velocityDict['Shoulder'] = 0
                self.velocityDict['Elbow'] = 0
                self.velocityDict['Wrist'] = 0
                commandInfo('Feeder recovered')
                self.feed_action = False
                self.flag = 0
```

Figure 4.24: arm control algorithm

The last control algorithm is about the chassis. Our target is to move the car as fast as possible and finish the whole project in a rapid and steady fashion. Since the chassis receives the angle, which can be further converted into speed, the only thing for the chassis control is to write a PID program to transfer angle to speed, which is shown in Figure 4.25.



(a) (b)

Figure 4.25: The PID control of chassis

Our controlling method is writing a forward speed and adding the steering speed by the angle we received from decision group. At first, we want to use the PID control method, because it can both reduce the overshoot in curve and make the oscillation better in the straight road. However, it dose not work because the command the rover received has a delay compared the simulation time, so the derivative and integral part will make the performance worse. Then we found some frames have lost because the sensors wound cost a lot of times to refreshing and processing its data. Therefore, we add the limitation of the system to solve the former controlling problem.


(a) with frame loss (b) without frame loss

Figure 4.26: The PID control with and without frame loss

During experiments, the frame has 3 or more loss in each second, thus causing the rover to oscillate greatly. Hence, we have to add the limitation although we can get a faster simulation speed without the limitation. After we add the limitation, we can observe that the angle reduced from above 30 degree to about 15 degree, and the setting time reduced a lot. the rover perform much better.

At last we reduced the speed of the rover and increased the proportional control of it. As a result, we effectively prevent the overshoot problem around the corner.

## 4.5 System Controlling

The system controlling part is in the charge of our tech leader, Zhuheng Song.

### 4.5.1 Multiprocessing & Code Refactoring

Author: Zhuheng SONG, UoG ID: 2357651S


Figure 4.27: The regular development process

There is no doubt that a whole control process goes like this:

1. Abstract data from sensors into signals to show state of the rover
2. Make decision of following behavior depending on current state
3. control the rover to do the required behavior

Since here we do not have very complex situations to make decisions and we do not have to very complex tasks to do, it is obvious that **each stage takes different amount of time**. I suppose **Perception** takes a lot time, while time cost by **Decision** stage may be really short that we could even ignore it. In addition, **Control** stage actually need an **separate**, **high frequency** loop to keep high precision control of the rover.

Therefore, we need three separate loops:

- A perception loop keeps processing sensors data and transmit signals to decision loop. It should send out signals every time a signal is updated, but not send once when all signals are updated. In this way, we could always do decisions with the latest signals.
- A decision loop keeps sending commands to control loop depending on signals received from perception loop.
- A control loop keeps controlling the rover to act like what the decision loop asks to do, for example applying a PID control on velocity of four wheels.

Then we found two normal ways to allow three separate loops in one program: (1) multi-processing; (2) multi-threading

After some research we found that: If your program is a **CPU intensive** application, multi-processing is recommended, While multi-threading is more suitable for **I/O intensive** applications.

- A CPU intensive application is: for example, which has lots of loops or calculating steps.
- A I/O intensive application is: for example, mainly deal with files, or a web crawler.

Since multi-threading is running concurrently, most implements use a counter to decide whether switch to another thread. Therefore a CPU intensive application could reaches the counter threshold very soon, and need to compete for time to execute again.

On the other hand, for I/O intensive applications which have lots of situations where has to wait for **external** events, multi-threading could save a lot of resource, as for these situations, we could switch to another thread.

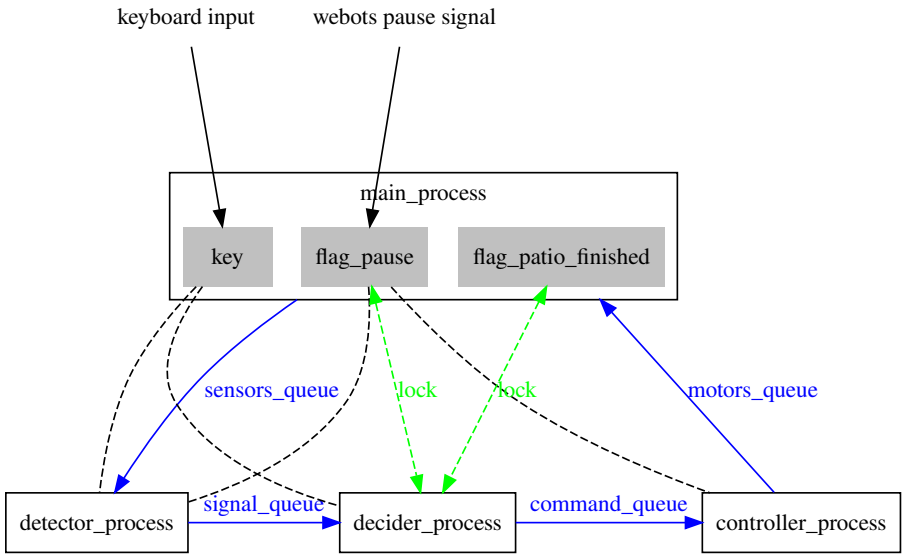



Figure 4.28: The final structure of our system

Given this, we decide to design a **multiprocessing** system. After lots of experiments, the system is developed to allow multiprocessing even in Webots, cross-platform and with high performance. The final structure of our system is shown in Figure 4.28.

The **detector process** will process sensors data and send all signals to the **decider process** every time a signal is updated, then the **decider process** will send command to the **controller process**, where the command is parsed into a dictionary of motors velocity. In addition, no matter a command is sent to **controller process** or not, it will send the motors velocity dictionary to the main process once every 5 ms. And before receiving the motors velocity dictionary, the main process will be blocked for 20ms, leave time for the three processes to do the work. Therefore, the simulation speed is now $\frac{32}{30+a+b}$, almost the best we could do, since we could not reduce a and b from the code. According to my test, value of $a + b$ could vary **from 10ms to 70ms**.

Figure 4.29 shows delays in the cycle.

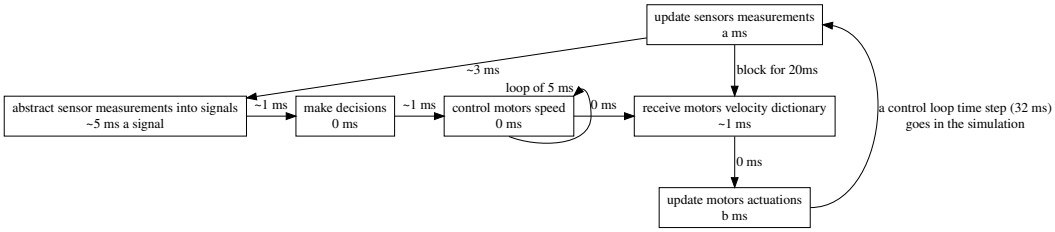



Figure 4.29: The final control loop

To prove the system works as expected, every step and the time it cost is logged, to show the exact time and order. It can be seen from the output shown in Figure 4.30 that there is at most delay of one frame between the simulation and motors velocity, which means in the simulation the motors velocity has a delay of at most 32ms. This is what expected and it is acceptable for this program.

```
[Debug] Simulation frame: 60
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Debug] a = 33
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 59 spend 6ms
[Command] Motor frame: 59
[Debug] x = 31
[Detected] Detector frame: 60 stage1: spend 9ms
[State] Decider frame: 60 spend 0ms
[Command] Chassis frame: 59 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Detected] Detector frame: 60 stage2: spend 12ms
[State] Decider frame: 60 spend 0ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Debug] b = 48
[Debug] Simulation frame: 61
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Debug] a = 29
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Command] Chassis frame: 60 spend 6ms
[Detected] Detector frame: 61 stage1: spend 9ms
[Command] Chassis frame: 60 spend 6ms
[State] Decider frame: 61 spend 0ms
[Command] Chassis frame: 61 spend 6ms
[Command] Motor frame: 61
[Debug] x = 34
[Detected] Detector frame: 61 stage2: spend 10ms
[State] Decider frame: 61 spend 0ms
[Command] Chassis frame: 61 spend 6ms
[Command] Chassis frame: 61 spend 6ms
[Command] Chassis frame: 61 spend 7ms
[Command] Chassis frame: 61 spend 6ms
[Command] Chassis frame: 61 spend 6ms
[Command] Chassis frame: 61 spend 6ms
[Command] Chassis frame: 61 spend 7ms
[Command] Chassis frame: 61 spend 7ms
[Command] Chassis frame: 61 spend 6ms
[Debug] b = 66
```

Figure 4.30: The system output

# Chapter 5

# Evaluation

This project is aimed at integrating a multifunctional intelligent rover that can perform specific tasks. This chapter of the report attempts to evaluate whether this integration of the system is successful from five perspectives. Some evaluation has been made in the subsystem design and is not repeated in this chapter.

## 5.1 Moving Speed

In the final demonstration, we are required to finish the simulation within 10 minutes. We manage to shorten this into around 3 minutes with a wheel velocity of 35 rad/s. Even though our rover moves pretty fast, we optimize our control algorithm to prevent it from overshooting around the corner. We also apply the mulitprocessing to let the visual, decision and chassis processes work in a parallel way. This saves the time to wait for the output of the last process, which occurs frequently in the series processing.

## 5.2 Operating Efficiency

We have considered the operating efficiency of the program as an important point of our design. In particular, we optimize each algorithm of us with respect to the time-complexity and space-complexity. Taking the color filtering algorithm as an example, the initial design follows the common practice of finding the contours of each color in OpenCV and then making a summation of the contour areas. During our analysis, we find that this method is too redundant. We simplify our algorithm by summing all the areas with an intensity higher than a certain threshold. On top of this, we also notice that using too many sensors will bring down our simulation speed. This is because initializing sensors in Webots consumes a large amount of time. For simplicity, we provide an easy solution with two cameras and one distance sensor only. As a result, our program is of light weight ang high operating efficiency.

## 5.3 Comprehensive State Machine

Our rover is not restricted to follow a specific routine to finish all the tasks due to the comprehensive state machine. In fact, it can start from any spot in from the Task 1 to the Task 5. For example, if we want to test whether our solution to Task 5 work or not, we can simply set the starting point of our rover at the end of Task 4. Even though it does not cross the bridge or gate, our color filtering and path finding algorithms can still operate normally.

## 5.4 Complete Encapsulation

During the development of our rover, we make a complete encapsulation of our program. In other words, each group can directly test its code without interfering the work of other groups. For example, the Visual Group can test its path finding algorithm within the visual program while the Chassis Group can also run the PID control algorithm within the chassis program. Their codes

are independent of each other. Finally, codes from different groups are imported and integrated in the controller program to make the rover run properly.

## 5.5 Excellent Project Management

Since this is a team design project, the management performance is evaluated here. With the assistance of project planning techniques, such as SoW, WBS and the Gantt Chart, we manage the project successfully, finishing all the tasks one week ahead of the deadline. We make a clear division of work at the very beginning of the project. Thus, every member in our team contributes a lot to our system designing. What's more, we base our project development on ZenHub and GitHub, leading to the tight cooperation among team members.

However, as it is our first time to do the project management, we still make some mistakes, which could be avoided next time with this experience. For example, we do not realize the necessity to write down a summary of each meeting. Some team members could only watch the recording to review the contents, which is a waste of time.

# Chapter 6

# Conclusion and Future Work

## 6.1 Conclusion

In this project, we work as a team to design and implement a multifunctional intelligent rover using Webots simulation. The designed rover succeeds in finishing all assigned tasks. However, our team does not stop here. We go a little bit further by including what might happen in the real world in our design. We consider the external environment, such as angle of the sun, the dusk on the road, and the internal physical property of the rover, like the centre of mass during development, ensuring that our design is robust even in real-world scenarios. These results come from a joint effort of our team members. We also attribute our success to the proper use of project management skills.

## 6.2 Future Work

Although we have tried our best to make our design close to reality, our design still some limitations since Webots is, as a matter of fact, not real and we do not explore it to the full extent. Due to time restriction, we only choose solutions from a limited amount of image processing or chassis control algorithms. The following points are possible extensions in the future:

1. Improve the path finding and object detection by taking advantage of cutting-edge image processing methods
2. Make a more detailed comparison among solutions of different sensors and choose a optimal one
3. Improve the chassis control algorithm by receiving more signals from the visual system

# Appendix A

# Member List & Individual Contribution

| Name | UoG ID | Personnel Division |
|---|---|---|
| Jinwei Chu | 2357643C | Environment Group & Project Manager |
| Huiyu Xiong | 2357680X | Environment Group & Document Group |
| Zhuheng Song | 2357651S | System Group & Tech Leader |
| Zijian Wang | 2357660W | Decision Group |
| Hanpeng Xu | 2357916X | Decision Group |
| Chang Shu | 2357702S | Visual Group & Document Group |
| Bo Wen | 2357658W | Visual Group |
| Haoran Han | 2357662H | Visual Group & Sensor Group |
| Haotian Wang | 2357667W | Chassis Group |
| Chaofan Shi | 2357686S | Chassis Group |

Table A.1: Member list with personnel division

## Roles Description:

### Project Manager:

- manage budget, in charge of reimbursement - Jinwei Chu
- supervise project progress - Jinwei Chu
- coordinate experiment timetable of every sub-team - Jinwei Chu

### Environment:

- list out the specifications about the environment building - Huiyu Xiong
- build the Webots world file to fit the specifications - Jinwei Chu

### Visual:

- beacon detection (including the orange box) - Chang Shu
- color recognition - Chang Shu
- line patrol with MATLAB- Bo Wen
- object detection (bridge and gate) with MATLAB - Bo Wen
- convert MATLAB programs developed by Bo Wen into Python - Haoran Han

### Sensor:

- figure out sensors (distance sensor, ultrasonic sensor, Gyroscope sensor) sheme and code for them - Haoran Han
- determine the position and angle of camera and sensors on the rover, help members from chassis group to draw them into the 3D model - Haoran Han

### Decision:

- in charge of decision making with respect different tasks
- rover behavioral decision making - Zijian Wang
- line patrol decision making (cooperate with the Visual and Sensor Group) - Zijian Wang
- cross bridge and gate decision making (cooperate with the Visual and Sensor Group) - Hanpeng Xu
- decision code writing - Hanpeng Xu

### Chassis:

- in charge of drive function of the chassis
- build the 3D model the whole rover by SolidWorks (including the chassis, sensors, fish food dropper and so on) - Haotian Wang
- modify the chassis to fix the mother board, sensors and so on - Haotian Wang
- design and build arm to drop the fish food - Chaofan Shi
- develop control algorithms for the arm and chassis - Chaofan Shi

### System Architecture:

- configure the environment needed by other sub-teams - Zhuheng Song
- synthesize modules into a system - Zhuheng Song
- develop the Mihotel script for a set of command used for configuration, debug and execution - Zhuheng Song

### Documentation:

- in charge of slides making - Zhuheng Song, Huiyu Xiong
- in charge of demo video - Zhuheng Song
- in charge of the final report writing - Chang Shu, Huiyu Xiong

# Bibliography


[1] "Robot." https://en.wikipedia.org/wiki/Robot Accessed on: Jun. 20, 2020. [Online].

[2] J.-H. Kim, E. T. Matson, H. Myung, P. Xu, and F. Karray, *Robot Intelligence Technology and Applications 2: Results from the 2nd International Conference on Robot Intelligence Technology and Applications*, vol. 274. Springer Science & Business Media, 2014.

[3] "Webots documents: Tutorials." https://cyberbotics.com/doc/guide/tutorials Accessed on: Jun. 20, 2020. [Online].

[4] M. Basu, "Gaussian-based edge-detection methods-a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 3, pp. 252–260, 2002.

[5] H. Gao, Y. Peng, Z. Dai, and F. Xie, "A new detection algorithm of moving objects based on human morphology," in *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 411–414, IEEE, 2012.

[6] "Webots documents: Tutorials 6: 4-wheels robot." https://cyberbotics.com/doc/guide/tutorial-6-4-wheels-robot Accessed on: Jun. 20, 2020. [Online].