

Faculty of Engineering and Computer Science Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation¹. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

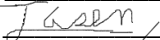
You must write one of the following statements on each piece of work that you submit:

For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"**, with your signature, I.D. #, and the date.

For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"**, with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number: ENGR 290
Name: Jasen Ratnam
Signature: 

Instructor: Dr. Khashayar Khorasani
I.D. # 40094237
Date: 2020-04-29

¹ Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at <http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf>.

Faculty of Engineering and Computer Science Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation¹. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

You must write one of the following statements on each piece of work that you submit:

For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"**, with your signature, I.D. #, and the date.

For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"**, with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number: Engr 290
Name: Andre Saul
Signature: AS

Instructor: Dr. K. Khorrami
I.D. #: 40076579
Date: 4/26/2020

¹ Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at <http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf>.
Approved by the ENCS Faculty Council February 10, 2012

Faculty of Engineering and Computer Science Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation¹. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

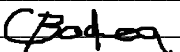
You must write one of the following statements on each piece of work that you submit:

For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"**, with your signature, I.D. #, and the date.

For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"**, with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number: 40027683
Name: Vicentiu-Cristian Badea
Signature: 

Instructor: Dr. Khashayar Khorasani
I.D. # 40027683
Date: 2020-04-24

¹ Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at <http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf>.

Faculty of Engineering and Computer Science Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation¹. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

You must write one of the following statements on each piece of work that you submit:

For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"**, with your signature, I.D. #, and the date.

For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"**, with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number: ENGR 290
Name: Brian Nordio
Signature: 

Instructor: Khaghayar Khorasani
I.D. # 46055620
Date: 28/04/2020

¹ Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at <http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf>.

Final Design Report

Autonomous hovercraft design

APRIL 2020

Concordia University

ENGR 290

ISSUED BY

TEAM #21

REPRESENTATIVE

| | |
|-------------------------|----------|
| JASEN RATNAM | 40094237 |
| BRIAN NORDIO | 40055620 |
| ANDRÉ SAAD | 40076579 |
| VICENTIU-CRISTIAN BADEA | 40027683 |

I. ABSTRACT

Developed in the scope of the ENGR 290 course at Concordia University, the project goal is to create a simplified version of a hovercraft. The end-product should have contained a working hovercraft model that is able to finish the track in the time limit autonomously, complemented with the well-documented process of design and implementation. Due to the COVID-19 pandemic, we were ordered by the government to not enter school property, therefore we could not possibly build a real hovercraft without access to the workshop and materials required. Due to this unforeseen situation, we had to use a simulation software (CoppeliaSim[4]) to design a virtual hovercraft to traverse a virtual maze that is provided to us. This goal is accomplished by following the steps of the engineering design process to make the required calculations and using CoppeliaSim[4] to simulate the hovercraft chosen in the PDR using the decision matrix to choose one design.

We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality.

II. TABLE OF CONTENTS

| | |
|---|-----------|
| I. ABSTRACT | 1 |
| II. TABLE OF CONTENTS | 2 |
| III. LIST OF FIGURES | 4 |
| IV. LIST OF TABLES | 5 |
| 1.INTRODUCTION | 6 |
| 1.1 Document Purpose | 6 |
| 1.2 Document Scope | 6 |
| 1.3 Change Control and Update Procedures | 6 |
| 2.REFERENCE DOCUMENTS | 7 |
| 3.REQUIREMENTS | 8 |
| 3.1 Mission Objectives | 8 |
| 3.2 Design Goals & Technical Goals | 8 |
| 3.3 Competition Requirements and Operations | 9 |
| 3.4 Target Score Analysis & Competition | 9 |
| 4.HOVERCRAFT DESIGN | 10 |
| 4.1 Principle of Operation | 10 |
| 4.2 System Design | 10 |
| 4.3 Structure Design | 11 |
| 4.4 Lift Design | 11 |
| 4.5 Propulsion Design | 11 |
| 4.6 Mechanical Design | 11 |
| 4.7 Calculations and Path Breakdown | 11 |
| 4.8 Decision Matrix | 12 |
| 4.9 SWOT Analysis | 13 |
| 4.10 Calculations & Simulations Summary | 14 |
| 4.11 Derivations used in Matlab tests | 16 |
| 4.12 Matlab test results | 21 |
| 5.HOVERCRAFT CONSTRUCTION PLAN | 26 |
| 5.1 Virtual Fabrication | 26 |
| 5.2 CoppeliaSim Changes to Design | 26 |
| 5.3 Implementation | 26 |
| 5.4 Verification and Testing | 26 |

| | |
|---|-----------|
| 6.COPPELIASIM DETAILS | 27 |
| 6.1 Hovercraft Details | 27 |
| 6.2 Proximity Sensors | 27 |
| 6.3 Vision Sensors | 28 |
| 6.4 Vision vs Proximity Sensors | 28 |
| 6.5 Final Simulation Videos | 28 |
| 6.6 Final Score | 29 |
| 7.MISSION OPERATIONS | 30 |
| 7.1 Competition Requirements and Operations | 30 |
| 8. MASTER SCHEDULE | 31 |
| 8.1 Master Schedule | 31 |
| 8.2 WBS (Work Breakdown Structure): | 33 |
| 8.3 Timeline and Milestones (Gantt Chart) | 34 |
| 8.4 Original Project Work Done | 34 |
| 9.RISK MANAGEMENT AND CONTINGENCY | 35 |
| 10.SUMMARY | 36 |
| 10.1 Overall Goal | 36 |
| 10.2 Overall Score | 36 |
| 10.3 Project Changes | 36 |
| 11.GLOSSARY | 37 |
| 11.1 Youtube Page | 37 |
| 11.2 Source Code | 37 |
| 11.2.1 Proximity sensor: | 37 |
| 11.2.2 Vision sensor: | 43 |
| 12.REFERENCES | 50 |

III. LIST OF FIGURES

| | |
|--|----|
| Figure 2.1: Track of the Hovercraft | 8 |
| Figure 4.1: Hovercraft design | 10 |
| Figure 4.2: Hovercraft base sketch | 16 |
| Figure 4.3: Side view of component masses sketch | 16 |
| Figure 4.4: Hovercraft skirt sketch | 17 |
| Figure 4.5: Hovercraft detailed sketch | 19 |
| Figure 4.6: Matlab test results with $d_{CG2R} = r_{fan} + W_{fs}$ and $X = 0.65$ [1-6] | 22 |
| Figure 4.7: Matlab all tests result with $d_{CG2R} = r_{fan} + W_{fs}$ and $X = 0.65$ | 23 |
| Figure 4.8: Matlab test results with $d_{CG2R} = L/2$ and $X = 0.70$ [1-6] | 24 |
| Figure 4.9: Matlab all tests result with $d_{CG2R} = L/2$ and $X = 0.70$ | 25 |
| Figure 6.1: Settings of proximity sensors | 27 |
| Figure 6.2: Settings of vision sensors | 28 |
| Figure 8.1: Master Schedule Part 1 | 31 |
| Figure 8.2: Master Schedule Part 2 | 32 |
| Figure 8.3: Master Schedule Part 3 | 33 |
| Figure 8.4: Work Breakdown Structure | 33 |
| Figure 8.5: Timeline and Milestones | 34 |

IV. LIST OF TABLES

| | |
|--|----|
| Table 4.1: Decision Matrix | 12 |
| Table 4.2: Swot analysis | 13 |
| Table 4.3: Component details | 19 |
| Table 9.1: Risk Management and Contingency Planning | 35 |

1.INTRODUCTION

A hovercraft is a vehicle that traps an air cushion underneath itself and then floats on top of any surface, hence it hovers over the floor and obstacles. One can consider this as a boat, airplane, and helicopter all in one because of its amphibious quality. One of the many uses of a hovercraft is its military application to access disaster zones. For our application, we will make a virtual hovercraft to traverse a maze. The simulation videos will be posted on [youtube](#).

1.1 Document Purpose

This document specifies the project's requirements and constraints that must be followed in the design and implementation process. In this document, we establish the necessary conditions and requirements given to us to design and simulate a hovercraft that respects them and is able to traverse the virtual track.

1.2 Document Scope

This report was created in the scope of the ENGR 290 course and shows the details on the virtual simulation of the hovercraft design chosen in the PDR that we will use to traverse a virtual track. It also shows how the virtual simulation using CoppeliaSim[4] will be implemented in detail with a time schedule. Our final product is a simulated design going around a given maze using proximity and vision sensors separately because of the COVID-19 situation since it is not possible to build a real hovercraft anymore. This product will be evaluated using the new grading formula given by the professor:

$$\text{Score} = (\text{the time taken to complete the track} * \text{number of proximity sensors}) + (\text{the time taken to complete the track} * \text{number of vision sensors}) .$$

The team with the lowest score for both scenarios will be ranked first, hence to have a simulation with the lowest score possible. Hence we want to use the least amount of components possible but also have the fastest track time.

1.3 Change Control and Update Procedures

The update procedure that we use, is to document any changes done to our design and include them in this report. One possible change that we had foreseen, is to change the size of fans and batteries or reduce the number of components to get a higher score. Unfortunately, due to the COVID-19 pandemic, our entire project had to be changed and we were left with little time to learn a new software (CoppeliaSim[4]) and virtually create a hovercraft that will traverse a virtual maze. We had multiple issues to create a hovercraft that works perfectly and completes the given track. This is a significant change, hence we couldn't document properly all the changes made to our original designs. The major changes that are important will be noted below but minor changes that had to be made were not documented and cannot be shown. We can sum up those minor changes by saying we had to do some changes to adapt the hovercraft design made for real life work on the simulation software.

2.REFERENCE DOCUMENTS

To simulate our hovercraft design in [CoppeliaSim](#)[4] we had to use a couple of reference documents to get to know the software and its [features](#) to implement an autonomous hovercraft. CoppeliaSim's website[4] provides great documentation on the available [APIs](#) to control our hovercraft design.

In addition to the documentation of the software available on its website, we also used a youtube [channel](#) as a reference to learn how to do different operations on the software with visual backing.

These documents were extremely helpful to simulate a hovercraft design with proximity sensors and vision sensors.

3.REQUIREMENTS

3.1 Mission Objectives

In order to have a functional hovercraft that respects all the requirements and constraints given, we need to list them in detail. The following set is essential to design and create a hovercraft. This set is used as a resource while building the hovercraft on CoppeliaSim[4], ensuring that the requirements are met at every step of the simulation creation. These requirements must be followed to ensure that our hovercraft completes the track below, for which a virtual copy is given in CoppeliaSim[4] by the TA. The final product that we will have has to be capable of all the updated requirements for the virtual simulation below.

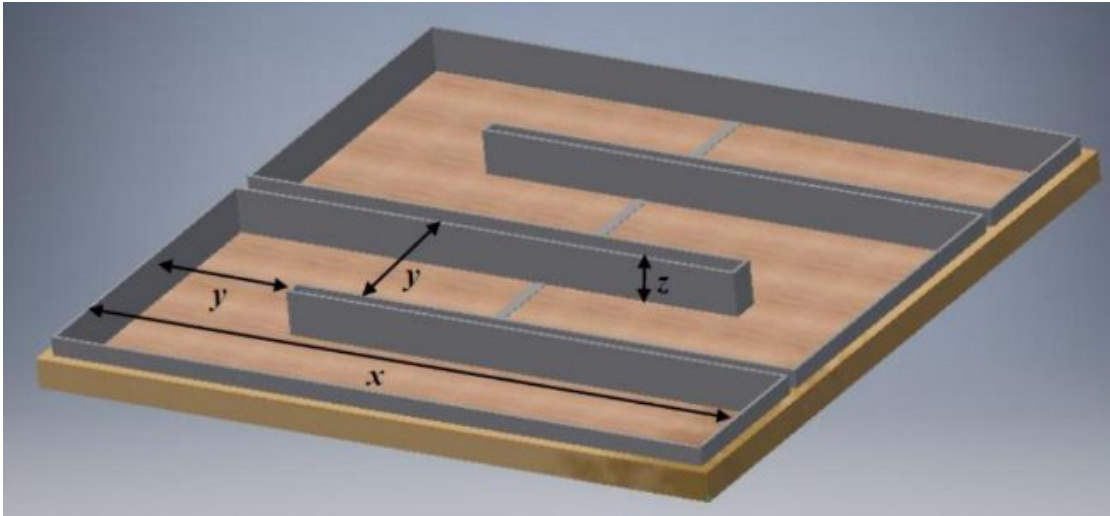


Figure 2.1: Track of the Hovercraft[1]

- x: 235cm
- y: 50-55cm
- z: 15cm

3.2 Design Goals & Technical Goals

Due to the change of the project scope, some of the requirements and goals had to be changed.

- The front of the hovercraft should be circular to facilitate going around corners
- Have a width less than 50cm, preferably around 30 cm
- The design should be shown with the help of sketches and models
- The design should be original and any inspiration should be shown
- Utilizing the least number of sensors. (Proximity sensors, and vision sensors)
- The geometric features and dimensions of our virtual reality-based track and obstacle simulations should be as close as possible the same as the one we chose in our PDR

3.3 Competition Requirements and Operations

Due to the change of the project, the competition requirements and operations also needed to be updated for our new project scope. The updated list is below:

- Should finish the track in Figure 1 virtually with the hovercraft simulation using CoppeliSim
- The entire body should rotate when turning
- Has to travel the track autonomously, using proximity sensors, vision sensors, and servos controlled with script code in the simulation.
- Have to travel over obstacles placed on the track of a height of a maximum 3 mm.
 - I.e. have to hover at a minimum of 3 mm.
- There are two sets of scenarios that we should implement and develop for our design. One with the use of the “Proximity sensors” and the second with the use of the “vision sensors” to travel the maze virtually.

3.4 Target Score Analysis & Competition

$$\text{Score} = (\text{the time taken to complete the track} * \text{number of proximity sensors}) + (\text{the time taken to complete the track} * \text{number of vision sensors}) .$$

This formula is used to analyze the hovercraft design we have simulated to traverse a virtual maze. Each team's hovercraft simulation will receive a score from the formula above for the simulation of both proximity sensors and vision sensors. The team with the lowest score will be ranked first, hence we want to get the lowest score possible. To accomplish this we want the shortest time taken to complete the track with the least number of components.

We will not be competing against one another, due to the current situation, each team's hovercraft simulation time and total score will be measured against others in order to rank the teams. Since we cannot view other teams hovercraft in person anymore, a video of our virtual simulation will be posted on [youtube](#) to allow other teams to have a visual of our simulation.

4.HOVERCRAFT DESIGN

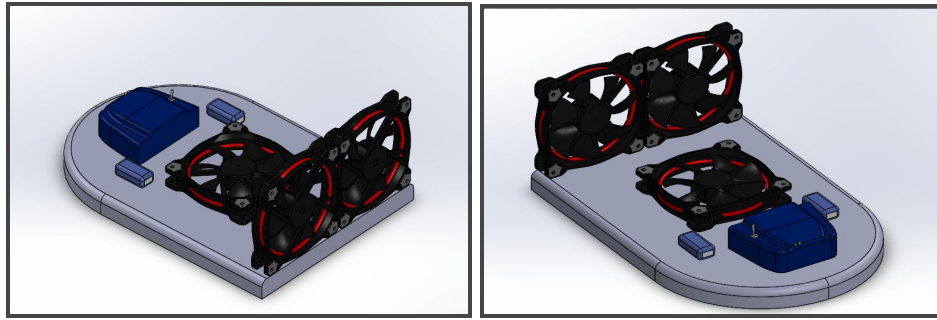


Figure 4.1: Hovercraft design

4.1 Principle of Operation

The model we chose to build is equipped with two fans in the back and 1 fan in the middle. This hovercraft is not equipped with a servo. Turning on and off one of the fans on the back will allow us to turn the craft, this eliminates the need for a servo. We cannot build a hovercraft anymore, therefore we will simulate it using CoppeliaSim[4]. The principle of operation of the hovercraft will remain the same for the simulation. The one difference being that the 1 fan in the middle for lift is swapped out with the quadcopter base. This provides better stability in the simulation since it does not use a skirt. We tried having the same lift fan but it caused the simulation to be extremely unstable. Using the quadcopter as advised by the TA solved our problems.

4.2 System Design

The first fan is placed in the middle face down where the airflow will go under the hovercraft through a hole, in a skirt. This will allow us to lift the hovercraft from the ground. The second and third fans are placed vertically in the back of the hovercraft. These fans will be used for direction changing and the horizontal force to go forward. The batteries and controller are placed at the front of the hovercraft to have a proper center of mass. Since we are now simulating a hovercraft, we don't need the information about the fans, batteries and the skirt given in the PDR anymore. The hovercraft simulation does not require batteries and a skirt to work and all the fans are the same with variable fan speeds. Therefore, we can ignore these in our system design. The placement of the fans do not change, except we use the quadcopter fan base to lift the hovercraft as stated above in the principle of operation.

4.3 Structure Design

The chosen model will have a rectangular shape with a rounded front. The rounded front was chosen to reduce the chance of the craft getting stuck in corners. This model comes with some weaknesses too, though. Notably, this shape could still get stuck in corners as it is not perfectly circular. Also, the presence of 3 fans, the heaviest components by far, adds a lot of mass which makes linear and angular acceleration more difficult. The hope is that the two fans in the back that allowed us to access more power and more control to overcome this mass and travel the maze with ease and stability. The structure design did not change for the simulation adaptation. We kept the overall structure design the same with the same weight. The simulation also shows the weakness of the shape by occasionally getting stuck in corners.

4.4 Lift Design

The lift of our design is from the first fan that is placed in the middle face down where the airflow will go under the hovercraft through a hole, in a skirt. This has changed for our simulation adaptation, the one lift fan has been swapped out with the quadcopter base. This provides better stability in the simulation since it does not use a skirt.

4.5 Propulsion Design

The propulsion of our design is from the second and third fans that are placed vertically in the back of the hovercraft. These fans are used for direction changing and the horizontal force to go forward. We use the same propulsion design for our simulation.

4.6 Mechanical Design

Our hovercraft design does not require any mechanical designing since we do not use moving parts such as servos. The only moving parts are the fans for the lift and propulsion. Since the fans are provided in both situations if building an actual hovercraft and in the simulation of one, we do not need to worry about the mechanical design of them.

4.7 Calculations and Path Breakdown

The track shown in Figure 2.1, shows that our hovercraft needs to go through three 180 degrees turns and overcome obstacles placed on the track of a maximum height of 3mm. We need to traverse 940 cm with a 180 degree turn every 235cm autonomously without human intervention. In our simulation our hovercraft should traverse a virtual version of that maze with the same dimensions and obstacles. We no longer have a time limit of 2 minutes but we need to complete the maze autonomously using the available proximity and vision sensors.

4.8 Decision Matrix

| Comparison Criteria | Importance | Design 1 | | Design 2 | | Design 3 | |
|----------------------|------------|-------------|--------------|-----------|----------------|-----------|---------------|
| | | Score | Result | Score | Result | Score | Result |
| Number of components | 0.1176 | 8 | 0.9408 | 9 | 1.0584 | 10 | 1.176 |
| Lift | 0.338 | 10 | 3.38 | 10 | 3.38 | 9 | 3.042 |
| Propulsion | 0.2561 | 10 | 2.561 | 9 | 2.3049 | 8 | 2.0488 |
| Weight | 0.1402 | 8.5 | 1.1917 | 9 | 1.2618 | 10 | 1.402 |
| Algorithm difficulty | 0.099 | 10 | 0.99 | 9 | 0.891 | 8 | 0.792 |
| Power Consumption | 0.04917 | 9 | 0.4425 | 8 | 0.39336 | 10 | 0.4917 |
| Total | 1 | 55.5 | 9.506 | 54 | 9.28946 | 55 | 8.9525 |

Table 4.1: Decision Matrix

From this decision matrix, we can see why we chose the first design to simulate. This table shows that the first design has the highest score out of the three. Some of our criteria have better results with other designs but the first design shows that the average of scores is better. This table was shown with more detail and explanation in the PDR.

4.9 SWOT Analysis

From the SWOT analysis done in the PDR, we found the strength and weaknesses of our chosen design shown below.

| Design | Strength | Weaknesses | Opportunity | Threats |
|---|---|--|---|--|
| Hovercraft Model 1 (1 fan on the bottom, 2 fans at back) | <ul style="list-style-type: none">- Airflow under the skirt for lift- Airflow for propulsion, velocity, and acceleration | <ul style="list-style-type: none">- Difficulty control algorithm- Volume of design- Weight- Power consumption-Number of components | <ul style="list-style-type: none">- Three opportunities to complete track- Knowledge of track and grading scheme- Coding framework provided | <ul style="list-style-type: none">- Imperfections in floor- Airflow- Competition |

Table 4.2: Swot analysis

4.10 Calculations & Simulations Summary

Now that we choose a design to implement, we had to do some simulations and calculations to choose which parts to use on our hovercraft. We used methods and tools learned in class to do so, notably Lagrangian and Newtonian mechanics and the Bernoulli equation when calculating fluid dynamics. Although the design calculations and simulations that we did show the behavior of the hovercraft, it is important to note it is only a close approximation in assumed conditions. Real-life factors that were not considered might affect our results with a real hovercraft model.

Although more calculations could have been made to see if the large fan was necessary to lift the craft, we chose it as it seemed like a safe choice due to its higher overall static pressure. If the larger fan was unnecessary, we could have saved a small amount of mass which would have made the craft more maneuverable (the smaller fan has 82% of the mass of the larger one). The thrust fans were similarly chosen without much calculation, but since there were 2, we assumed that they were powerful enough to maneuver the craft. We later confirmed this assumption in our calculations.

A simple power analysis was conducted to determine which batteries were needed to power the craft. Since the maximum allowed time was 2 minutes per run, the craft would only need to run for a maximum of 6 minutes. The total energy used by each component was estimated as its power consumption multiplied by 6 minutes using the following relations. The battery capacities were also converted to total energy.

$$P = VI \qquad 1 \text{ W} = 1 \text{ J/sec} \qquad mAh \cdot V \cdot 3.6 = J$$

Energy consumed: 6579.26 J

Energy of each Turnigy 180 mAh battery: 4795.2 J

$$2 \cdot 4795.2 \text{ J} = 9590.4 \text{ J} \qquad 9590.4 \text{ J} - 6579.26 \text{ J} = 3011.14 \text{ J}$$

Since 2 batteries are needed to achieve the fan voltage, there is an excess of total energy even when using the smallest batteries.

With these calculations done, we still wanted to conduct an in depth analysis of the dynamics of the craft to validate the component configuration we had chosen. The goal of the mathematical analysis was to determine a length 'L' (length of the rectangular part of the body) and 'r' (radius of the nose) that would produce a hovercraft that met all the following requirements:

1. The total length of the craft must be smaller than the width of the track hallways (50mm).
2. The craft had to be balanced lengthwise about the CG with the counterweight components available. The counterweight had to also fit entirely in the rectangular section of the craft.
3. There must exist a static pressure for the craft dimensions that satisfies the Bernoulli equation of the craft with a 3mm hover gap. (Based on the Q vs P graph in the fan's datasheet)
4. There must exist a static pressure for the craft dimensions that satisfies the vertical force balance equation of the craft. (Based on the Q vs P graph in the fan's datasheet)
5. A static pressure that satisfies requirement 4 must be present in the set of pressures that satisfy requirement 3 for the given dimensions. (These three steps were mathematically equivalent to equating the Bernoulli equation of the craft to its force balance equation).
6. The turning torque provided must be capable of achieving the angular velocity needed to make the corners with a goal of finishing the track under 2 minutes, given the moment of inertia of the craft.

The set of dimensions that satisfied all 6 requirements would represent all possible body sizes that we could use. The building material used throughout the calculations was $\frac{1}{4}$ inch balsa wood board. The skirt material was assumed to be weightless. In addition, two CGs, and consequently positions of the lift fan, were tested. These two things can be treated as the same since it is most convenient for the lift fan to act on the CG of the craft. The first scenario was that the fan was pressed up against the steering fans and did not move relative to the craft dimensions. The second was that the lift fan was placed on the centroid of the rectangular part of the body, and thus moved relative to the dimensions. Pressing the lift fan as far back as possible against the steering fans proved to be the most effective setup.

The analysis yielded a set of possible dimensions, but sadly since the project was moved to simulation, almost none of these results aided us in making the virtual hovercraft. However, this analysis represents the craft we would have physically made if given the opportunity.

4.11 Derivations used in Matlab tests

1. Test one did not require derivations to perform.

Test 1 pass conditions:

- $L + r < 50mm$

Dimensions of craft:

r : Radius of circular nose of craft

L : Length of rectangular part of craft

A_c : Area of craft

A_{nose} : Area of nose of craft

A_{rect} : Area of rectangular

A_s : Area of skirt opening

A_{s-nose} : Area of skirt opening under nose

A_{s-rect} : Area of skirt opening under rectangular section

h : Hover gap height

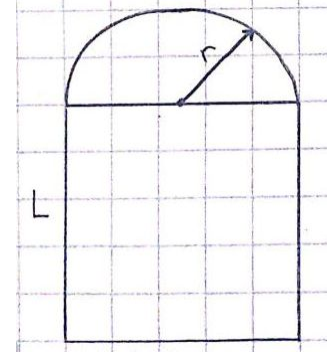


Figure 4.2: Hovercraft base sketch

$$A_{nose} = \frac{\pi r^2}{2} \quad A_{rect} = 2rL \quad A_c = A_{nose} + A_{rect} = \frac{\pi r^2}{2} + 2rL$$

$$A_{s-nose} = \pi r h \quad A_{s-rect} = 2h(L + r) \quad A_s = A_{s-nose} + A_{s-rect} = \pi r h + 2h(L + r)$$

2. Test two required the torque of each component about the CG of the craft. Since the craft would be designed so that the left and right sides are symmetric (facing forward in the craft), the craft only needed to be balanced from the side view.

$$m_{nose} = A_{nose} \cdot \rho_{A-wood}$$

$$\rho_{wood} = 170kg/m^3$$

$$\rho_{A-wood} = 170kg/m^3 \cdot 0.00635m = 1.0795kg/m^2$$

$$m_{cw} = 0.06345kg \quad (\text{mass of counterweight})$$

$$m_{rect} = A_{rect} \cdot \rho_{A-wood}$$

$$m_{fs} = 0.0161.93kg \quad (\text{mass of small fan})$$

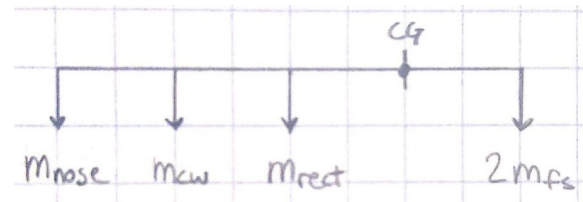


Figure 4.3: Side view of component masses sketch

Each of these masses acted on the craft at the following distances:

$$d_{nose} = \frac{4r}{3\pi} + L - d_{CG2R}$$

$$d_{rect} = L/2 - d_{CG2R}$$

$$d_{fs} = d_{CG2R} - W_{fs}/2$$

$$W_{fs} = 0.025m \quad (\text{width of small fan})$$

d_{CG2R} : Distance from CG to rear of craft. The two cases considered were:

$$d_{CG2R} = r_{fan} + W_{fs} \quad (\text{this gave better results}) \quad r_{fan} = 0.060m$$

$$d_{CG2R} = L/2$$

The goal is to find the distance d_{cw} (distance the counterweight needs to be from CG). Gravity is omitted as it appears in every term.

$$m_{nose} \cdot d_{nose} + m_{cw} \cdot d_{cw} + m_{rect} \cdot d_{rect} = 2m_{fs} \cdot d_{fs}$$

$$d_{cw} = \frac{2m_{fs} \cdot d_{fs} - m_{nose} \cdot d_{nose} - m_{rect} \cdot d_{rect}}{m_{cw}}$$

Test 2 pass conditions:

- $d_{cw} > r_{fan} + L_{cw}/2$
- $L > d_{cw} + L_{cw}/2$

Where $L_{cw} = 0.0534m$ (width of microcontroller)

3. Test three required Bernoulli's equation of the craft with a hover height h . Although there is a potential energy component at point 1, it was omitted as the mass density of air is rather small and the height between point 1 and 2 was unknown (skirt height) and small).

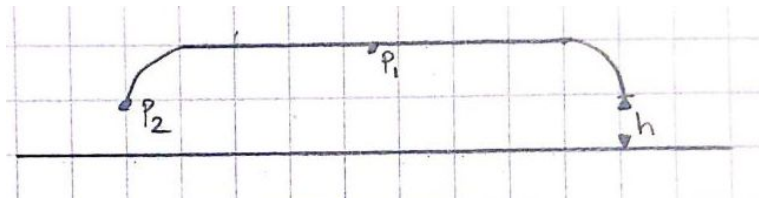


Figure 4.4: Hovercraft skirt sketch

$$P_1 = P_2 + \frac{1}{2} \cdot \rho_{air} \cdot V_2^2$$

$$Q = VA$$

$$P_1 = P_2 + \frac{1}{2} \cdot \rho_{air} \cdot (Q/A_s)^2$$

$$P_s = P_1 - P_2$$

$$A_s = \pi r h + 2h(L + r) = h \cdot (\pi r + 2L + 2r)$$

$$A_s = \frac{Q}{\sqrt{2P_s/\rho_{air}}}$$

$$h = \frac{Q}{\sqrt{2P_s/\rho_{air}}} \cdot \frac{1}{\pi r^2 + 2L + 2r}$$

$P_s(Q)$ was estimated with a best fit polynomial of the Q vs P graph provided in the datasheet of the large fan.

Q and subsequently P_s was bounded by the min and max values of this graph.

h was therefore a function of Q , L and r .

Test 3 pass conditions:

- $h > 3mm$
- $h < 3.1mm$
- Lowest 15% of range $< Q <$ Highest 15% of range
(this was supposed to simulate real life conditions where the fan would never perform at its extremes)

4. Test four required establishing the vertical (z axis) force balance equation for the static craft.

$$P_s \cdot A_c = (\rho_{A-wood} \cdot A_c + m_c)g$$

$$m_c = 0.58508kg \quad (\text{mass of all components})$$

$$P_s = \frac{(\rho_{A-wood} \cdot A_c + m_c)g}{A_c}$$

Test 4 pass conditions:

- Lowest 15% of range $< P_s <$ Highest 15% of range
(this was supposed to simulate real life conditions where the fan would never perform at its extremes)

5. Test five related the previous two equations.

$$P_s = \frac{1}{2} \cdot \rho_{air} \cdot (Q/A_s)^2 = \frac{(\rho_{A-wood} \cdot A_c + m_c)g}{A_c}$$

Test 5 pass conditions:

- Is there a pressure for values L and r in the set that passed test 3 that equals the pressure for the same values of L and r in the set that passed test 4
(equals ± 0.25 Pa to account for rounding errors in Matlab)

6. Test 6 required finding the moment of inertia of the craft about the z axis of its CG. From this we could roughly estimate if the angular torque provided by the maneuvering fans was enough to make the turns while the craft was travelling at an average velocity fast enough to finish the track in 2 minutes.

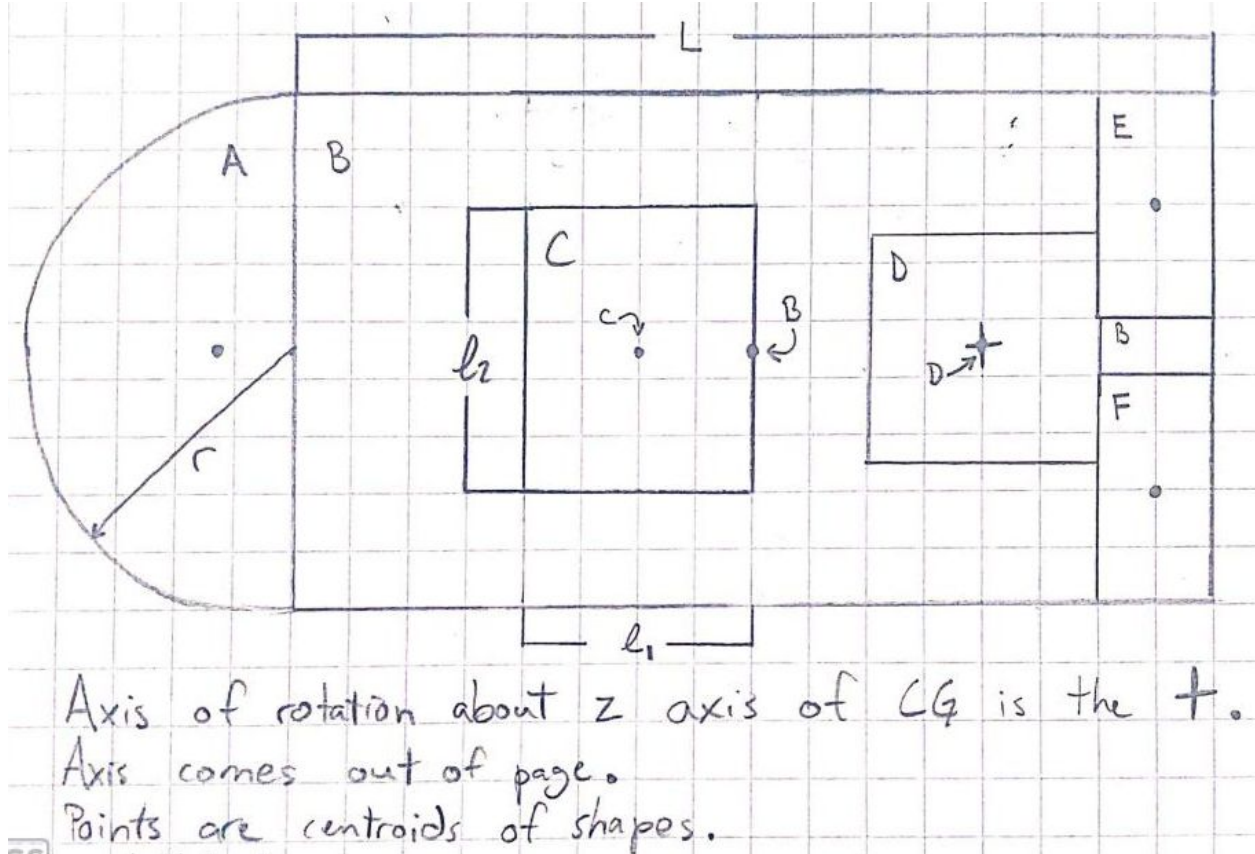


Figure 4.5: Hovercraft detailed sketch

| Shape / Object | Mass | Inertia about centroid (z axis) | Area | Distance of centroid to axis of rotation |
|----------------|------------|--|---------------------------|--|
| A | m_{nose} | $(\frac{1}{2} - \frac{16}{9\pi^2})m_A r^2$ | $\frac{1}{2}\pi r^2$ | $L + \frac{4r}{3\pi} - d_{CG2R}$ |
| B | m_{rect} | $\frac{m_B}{12}(L^2 + 4r^2)$ | $2r \cdot L$ | $\frac{L}{2} - d_{CG2R}$ |
| C | m_{cw} | $\frac{m_C}{12}(l_1^2 + l_2^2)$ | $l_1 \cdot l_2$ | d_{cw} |
| D | m_{fl} | $\frac{m_D}{12}(2(2 \cdot r_{fan})^2)$ | $(2r_{fan})^2$ | 0 |
| E/F | m_{fs} | $\frac{m_{E/F}}{12}((2 \cdot r_{fan})^2 + W_{fs}^2)$ | $(2r_{fan}) \cdot W_{fs}$ | $\sqrt{(d_{CG2R} - W_{fs}/2)^2 + (r + r_{fan})^2}$ |

Table 4.3: Component details

$$m_{fl} = 0.19777kg \quad (\text{mass of large fan})$$

$$l_1 = 0.0534m$$

$$l_2 = 0.0686m$$

Parallel axis theorem:

$$I_z = \sum_i (\bar{I}_i + A_i d_i^2)$$

Now we need to find the angular momentum gain/loss in 1 turn.

$$L_{track} = 8.75619449m$$

$$t_{goal} = 120s$$

$$r_{turn} = 0.025m$$

$$X > 0.65 \quad (\text{worst expected fan performance} - 65\%)$$

$$Q_{max-real} = Q_{max} \cdot X \quad Q_{max} = 3.03/60Pa$$

$$v_{avg} = \frac{L_{track}}{t_{goal}}$$

$$\omega_{avg} = \frac{v_{avg}}{r_{turn}}$$

$$t_{turn} = \frac{\pi}{\omega_{avg}}$$

$$V_{air} = Q_{max-real} \cdot t_{turn}$$

$$m_{air} = V_{air} \cdot \rho_{air} \quad \rho_{air} = 1.225kg/m^3$$

$$p_{air} = m_{air} \cdot \frac{Q_{max-real}}{\pi r_{fan}^2} \quad (\text{linear momentum of air moved})$$

$$L_{air} = p_{air} \cdot (r - r_{fan})$$

$$\omega = \frac{L_{air}}{I_z}$$

Test 6 pass conditions:

- $\omega > \omega_{avg}$

4.12 Matlab test results

Using the previously stated derivations, the 6 tests were conducted on the model of the hovercraft. Each test produced a set of dimensions L and r that either passed or failed the criteria. The union of all these sets produced the dimensions that pass all criteria.

Test 1 pass conditions:

- $L + r < 50mm$

Test 2 pass conditions:

- $d_{cw} > r_{fan} + L_{cw}/2$
- $L > d_{cw} + L_{cw}/2$

Where $L_{cw} = 0.0534m$ (width of microcontroller)

Test 3 pass conditions:

- $h > 3mm$
- $h < 3.1mm$
- Lowest 15% of range $< Q <$ Highest 15% of range
(this was supposed to simulate real life conditions where the fan would never perform at its extremes)

Test 4 pass conditions:

- Lowest 15% of range $< P_s <$ Highest 15% of range
(this was supposed to simulate real life conditions where the fan would never perform at its extremes)

Test 5 pass conditions:

- Is there a pressure for values L and r in the set that passed test 3 that equals the pressure for the same values of L and r in the set that passed test 4
(equals ± 0.25 Pa to account for rounding errors in Matlab)

Test 6 pass conditions:

- $\omega > \omega_{avg}$
- Since there are a few things that we could vary that would heavily affect the result of the tests, we chose to vary the location of the CG. Both L and r axis are in units of metres. All tests results are displayed in order 1 through 6 and then the union test.

Tests with $d_{CG2R} = r_{fan} + W_{fs}$ and $X = 0.65$ for test 6

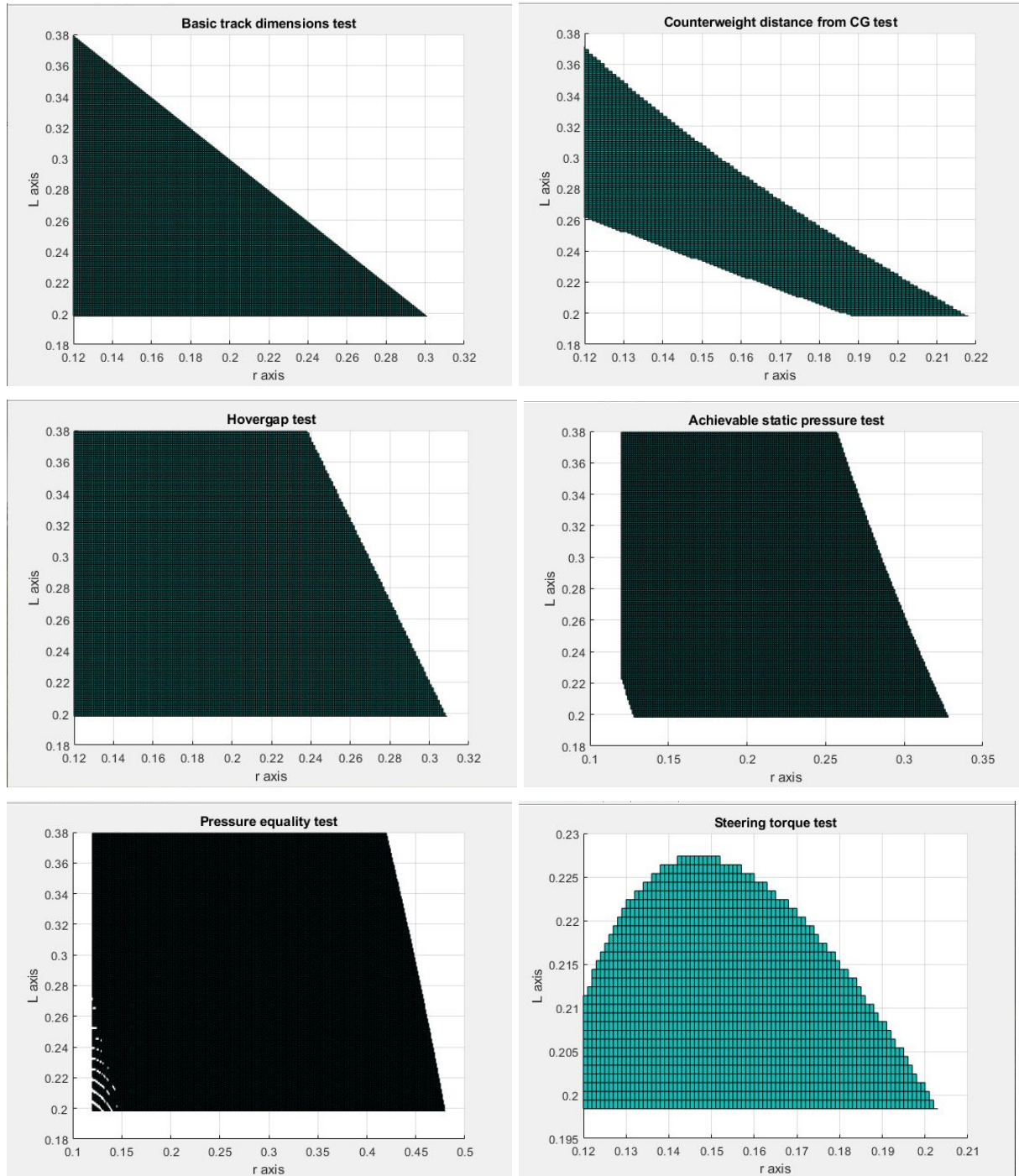


Figure 4.6: Matlab test results with $d_{CG2R} = r_{fan} + W_{fs}$ and $X = 0.65$ [1-6]

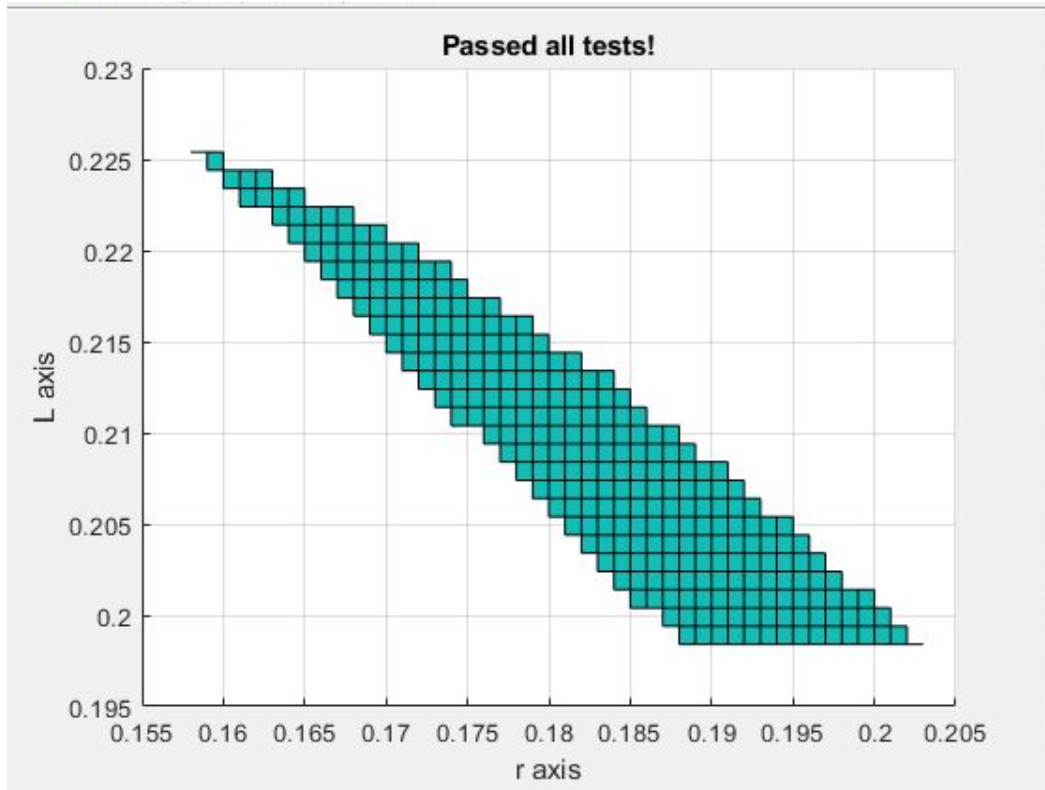


Figure 4.7: Matlab all tests result with $d_{CG2R} = r_{fan} + W_{fs}$ and $X = 0.65$

Tests with $d_{CG2R} = L/2$ and $X = 0.70$ for test 6

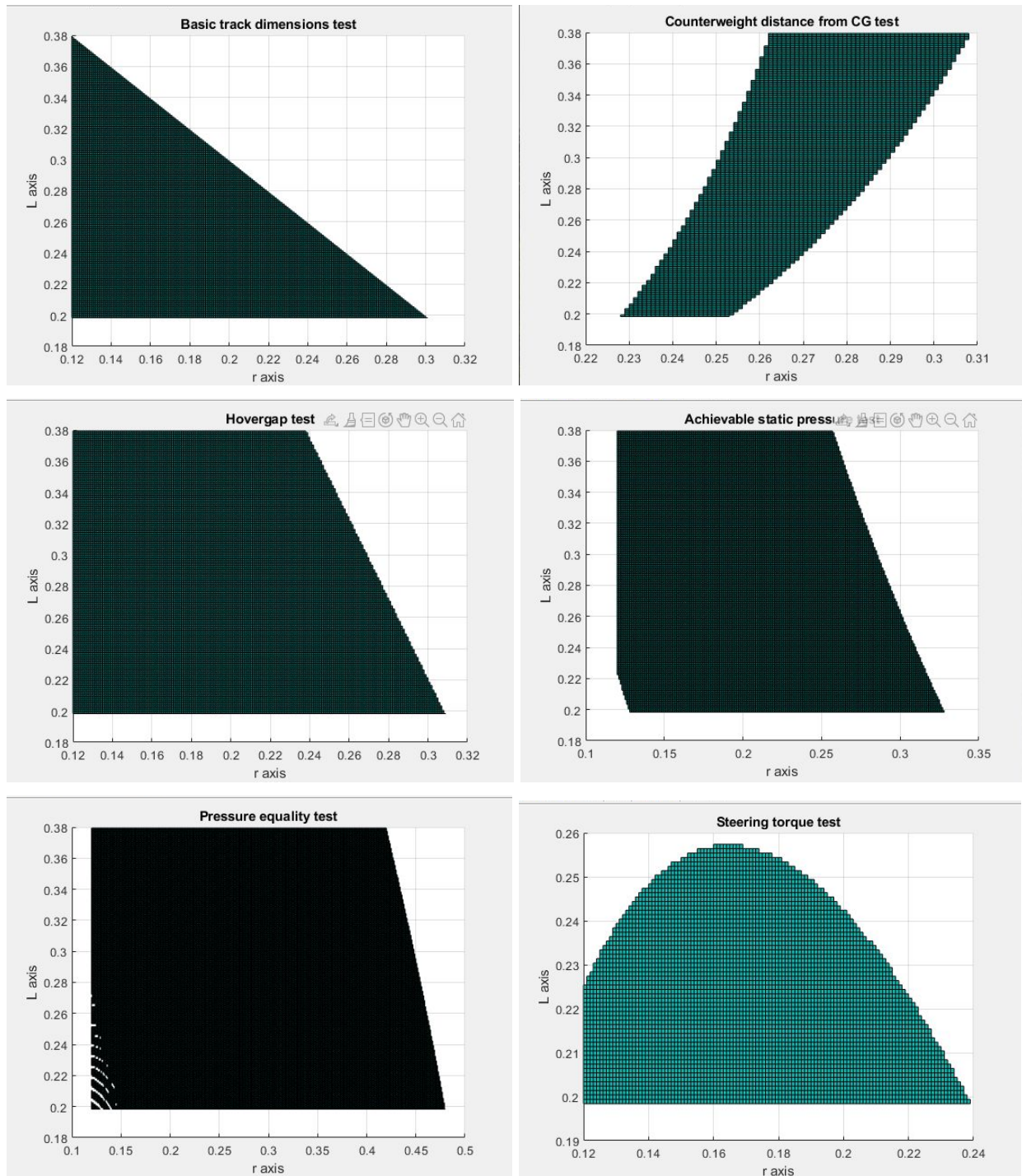


Figure 4.8: Matlab test results with $d_{CG2R} = L/2$ and $X = 0.70$ [1-6]

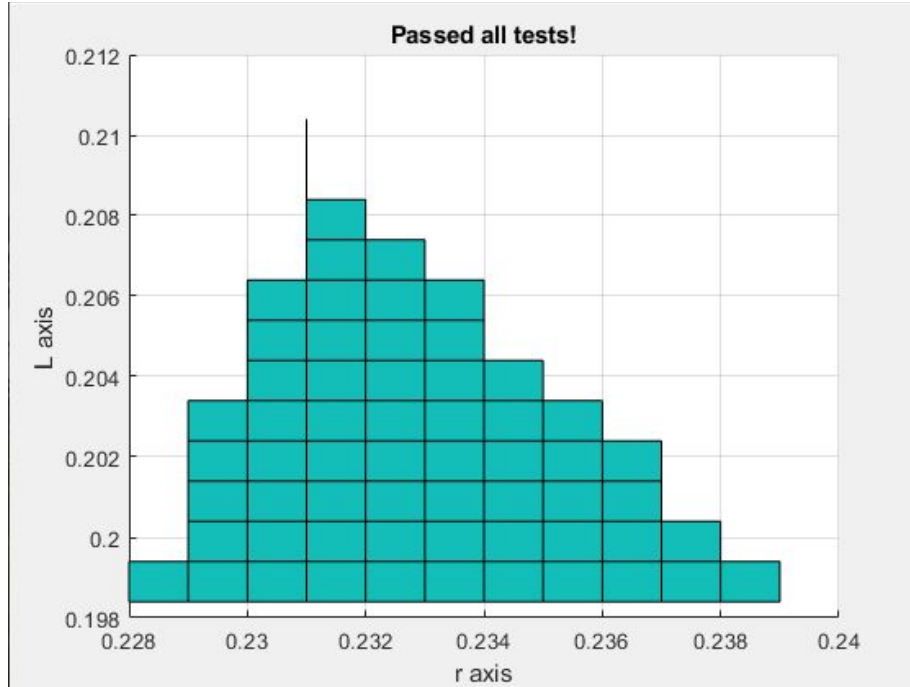


Figure 4.9: Matlab all tests result with $d_{CG2R} = L/2$ and $X = 0.70$

Results:

From the results displayed, it is clear that the first CG location was better than the second, as it gave results with a worse turning fan performance of 65% instead of 70%. However, both gave results, which means that the hovercraft layout was validated for any configuration of L and r found in the final union set!

5.HOVERCRAFT CONSTRUCTION PLAN

We had to change our plan to build a real hovercraft, to making a virtual hovercraft simulation. Therefore, we don't need a rigorous construction plan anymore. But some construction planning is still needed to build a virtual hovercraft.

5.1 Virtual Fabrication

We used primitive shapes to make the base of the hovercraft. Since, our design calls for a circular rectangle we had to combine a cylinder with a cuboid to get the correct shape. We made sure to keep the measurements of the base accurate to the measurements we stated in our PDR. We tried to keep the measurements exactly the same and we ended up with the base measuring 30 cm wide and 40 cm long. Then we needed to fabricate fans and place them on the hovercraft base. Instead of building our own fans from scratch, we used the pre-build fans available in the CoppeliaSim[4] library. The fans have been resized to 12 cm and placed in their position as mentioned in the above sections. Two fans were placed at the back of the hovercraft base to provide thrust and turning control. A third and last fan was placed at the middle of the hovercraft to provide lift. All three fans are identical and give the same force.

5.2 CoppeliaSim Changes to Design

After virtually building our hovercraft to be exactly the same design that we described in the PDR, we had to do some changes to make the hovercraft work properly and traverse the maze. After creating the hovercraft described above in CoppeliaSim[4], we couldn't get the hovercraft to lift properly and stay stable while running. Therefore, we used the quadcopter base with 4 fans to lift our hovercraft instead of the one fan, as advised by our TA.

5.3 Implementation

With the basic foundation of our hovercraft built virtually, we had to add sensors and a script to make it travel the track autonomously. The requirements of the project was to make the hovercraft traverse the maze using proximity sensors only and then vision sensors only on the same hovercraft base. The details of both these sensors are shown in the following section. While implementing our design to traverse the maze, we figured out quickly that 3 sensors is needed to efficiently traverse the maze, to know if there's a wall at the front or the sides. Hence, we used three sensors for each type of implementation (proximity sensors and vision sensors).

5.4 Verification and Testing

Verification and testing our design has become harder because we had to simulate it virtually. To verify that our design works as planned from our simulations and calculations, we had to run our simulation multiple times and essentially make the design work by trial and error. The verification process we used is if the hovercraft is capable of traversing the maze, then it passes and works good enough. This method was chosen because of the difficulty of learning a new software and ensuring it works perfectly. We also had a lack of time to perfectionate the design.

6.COPPELIASIM DETAILS

To simulate the hovercraft virtually in CoppeliaSim[4], we had to note some important details down. To make sure that our design is equivalent to our design in the PDR.

6.1 Hovercraft Details

In the simulation, we made the total hovercraft to have a total weight that we used for the simulations. Two of the sensors were placed on both sides and one at the front of it. All the fans for lift and thrust are equivalent with the same velocity and force at the start.

6.2 Proximity Sensors

The first situation we needed to simulate is using proximity sensors. Our solution uses three proximity sensors with the following settings.

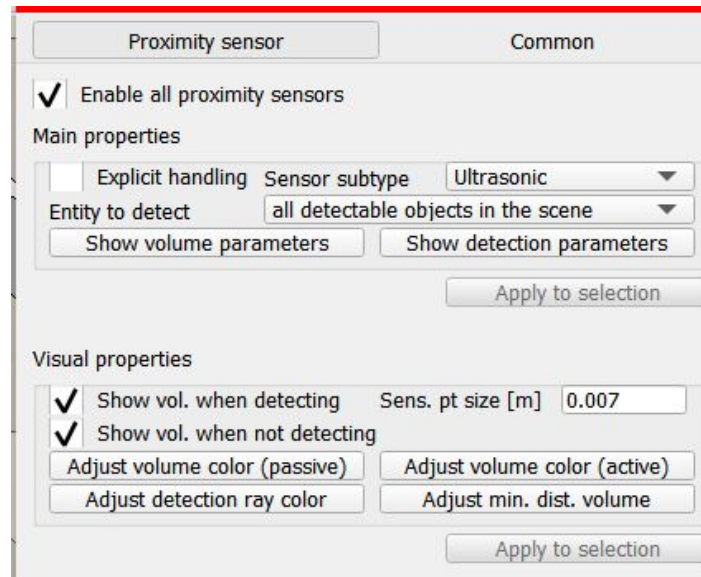


Figure 6.1: Settings of proximity sensors

We use these sensors to know when to turn. The hovercraft will know when it is getting too close to a wall and will turn in the opposite direction. The proximity sensors are a simulation of the real life ultrasonic sensors we would have used without the pandemic situation.

6.3 Vision Sensors

The second situation we needed to simulate is using vision sensors. Our solution uses three vision sensors with the following settings.

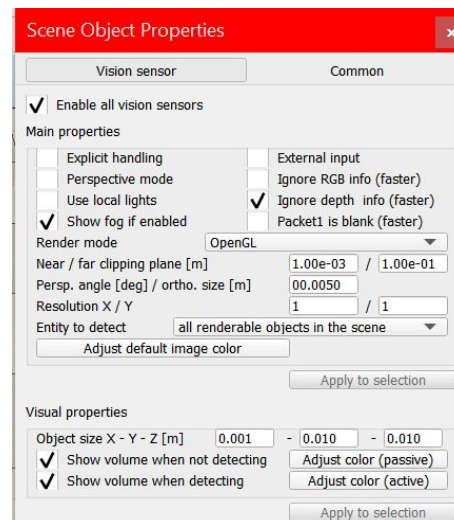


Figure 6.2: Settings of vision sensors

We use these sensors to know when to turn. The hovercraft will know when it is getting too close to a wall and will turn in the opposite direction. The proximity sensors are a simulation of the real life ultrasonic sensors we would have used without the pandemic situation. All the vision sensors are of the same size and have the same properties.

6.4 Vision vs Proximity Sensors

The proximity sensor can be used to model multiple types of sensors, we use the ultrasonic type. The sensor simulation can detect detectable entities such as the walls of the maze. The second type of sensor we used is vision sensors. Vision sensors on the other hand, sensors that are viewable objects. They operate in a similar way as camera objects. Vision sensors are a better option to use over proximity sensors when color, light or structure affects the detection process. In our case, both types of sensors offer the same information to control the hovercraft. But we observed that the vision sensors are a little bit slower than proximity sensors. Although, we did get better results using vision sensors with a shorter execution time.

6.5 Final Simulation Videos

A video of the final simulation of our hovercraft with both types of sensors is uploaded on youtube. The proximity sensor simulation is available [here](#) and the vision sensor simulation is available [here](#). From the videos, we can see that we had difficulties simulating our design in CoppeliaSim[4]. In fact, we had to run the code multiple times to get it to complete the maze in one way or the other. Therefore, if we rerun the code it will most likely not complete the track on the first try, you will most likely need to re-run the code multiple times to get the maze travelled. Our solution is not replicable everytime.

6.6 Final Score

Now that our simulations are complete with the corresponding videos, we can calculate the total score of our project to define our ranking in the class. We use the formula defined above to find the total score of our project. Knowing that our proximity sensor simulation uses 3 sensors and lasts 30 seconds, and that our vision sensor simulation uses 3 sensors and lasts 20 seconds. We can plug in these numbers in the formula.

$$\text{Score} = (\text{the time taken to complete the track} * \text{number of proximity sensors}) + \\ (\text{the time taken to complete the track} * \text{number of vision sensors})$$

$$\text{Score} = (30 \text{ seconds} * 3) + (20 \text{ seconds} * 3) = 150$$

Our total score is 150 for our project simulation.

7.MISSION OPERATIONS

7.1 Competition Requirements and Operations

Because of the pandemic, there will be no in person competitions. The teams will not be competing against one another in person, instead each team's hovercraft simulation videos and resulting performance (i.e. time of simulation) will be measured against others in order to rank the teams. The team with the lowest score will be ranked first, hence we want to have the lowest score possible.

The operations of our competitions is everyone does a virtual simulation of their hovercraft and uploads a video of it completing the maze on youtube. The video should have a complete simulation of a hovercraft traversing the maze, going around the corners using sensors. Going around the corners was hard to accomplish and we couldn't get to turn perfectly, currently our video shows that it hits the walls and goes back and forward before accomplishing the turn. We had to run the simulation multiple times to get the hovercraft to travel the maze completely, hence our videos are not replicable

8. MASTER SCHEDULE

8.1 Master Schedule

The Master Schedule can be represented with a node network, whereby we estimate the time requisites for the envisioned tasks ahead. The following network can be broken down into several stages. The first stage represents the beginning of the term, where we are in the planning phase of our structure. This is where we work on brainstorming ideas for the hovercraft designs, as well as evaluating our options with several analysis strategies. The goal is to derive a hierarchy of performance criterias and rank our ideas against each other.

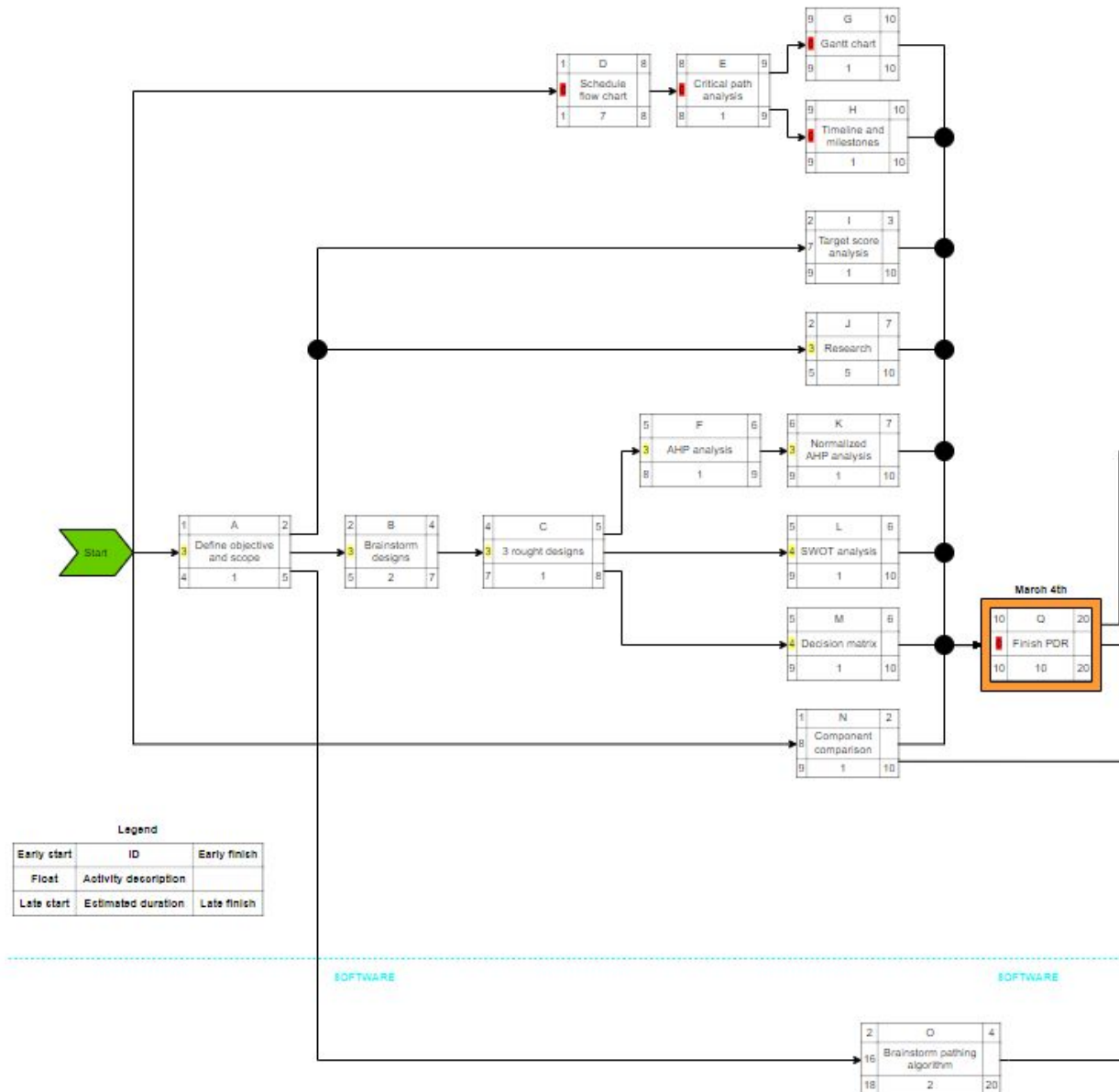


Figure 8.1: Master Schedule Part 1

Following our design stage, we may hone in on the superior design based on model analysis. In this stage, we derive our equations of motion, do hand calculations to develop an estimated model of our hovercraft, and finally we develop a 3d model and simulate the hovercraft equations using matlab. In tandem to our analysis, we can also begin preparing the VR environment and build a model.

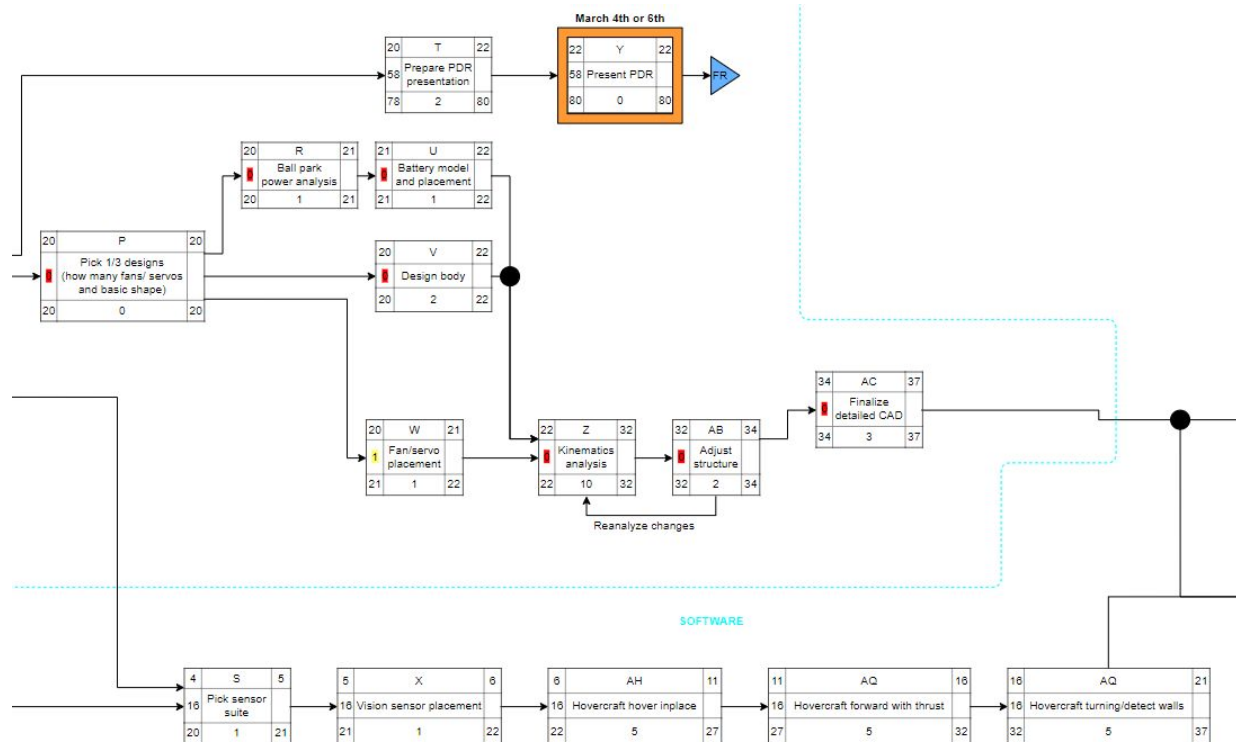
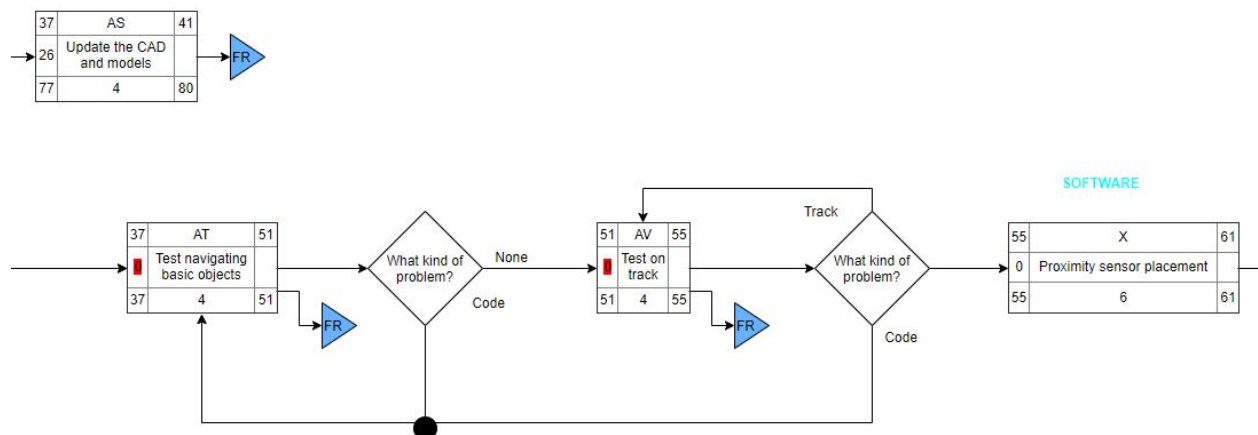


Figure 8.2: Master Schedule Part 2

With a finalized 3D model of the hovercraft, we are ready to simulate it in its VR environment. In the following stage, we may continue to work on the VR simulation portion of our project, whereby we implement strategies to navigate the maze using vision sensors and proximity sensors respectively.



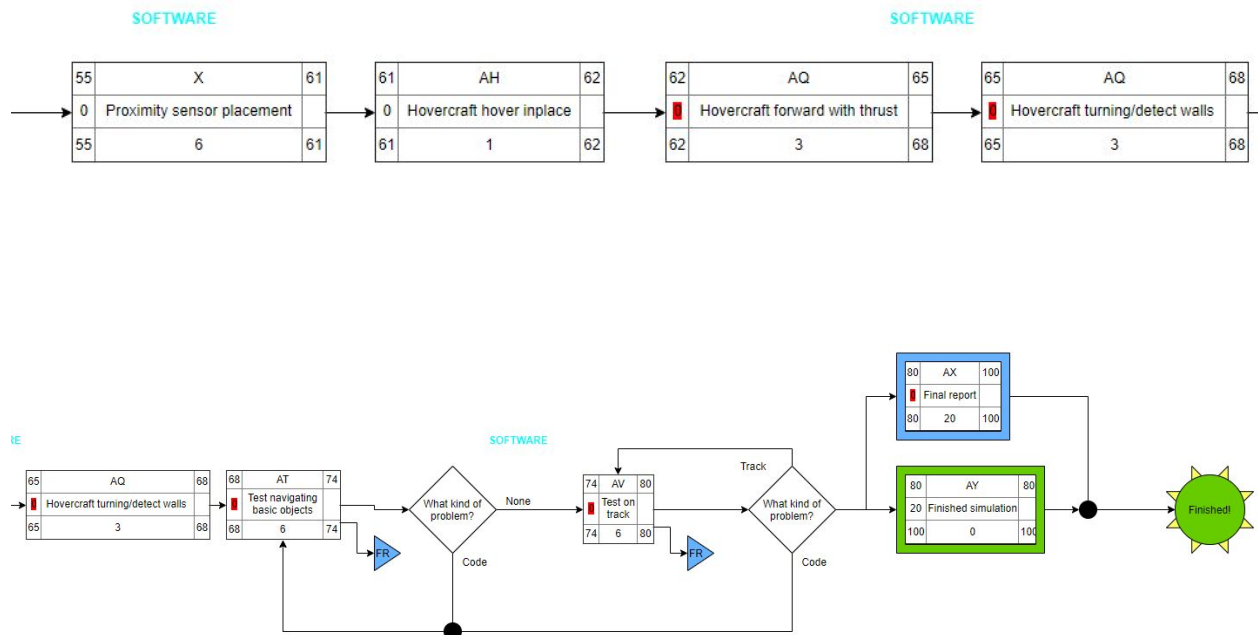


Figure 8.3: Master Schedule Part 3

8.2 WBS (Work Breakdown Structure):

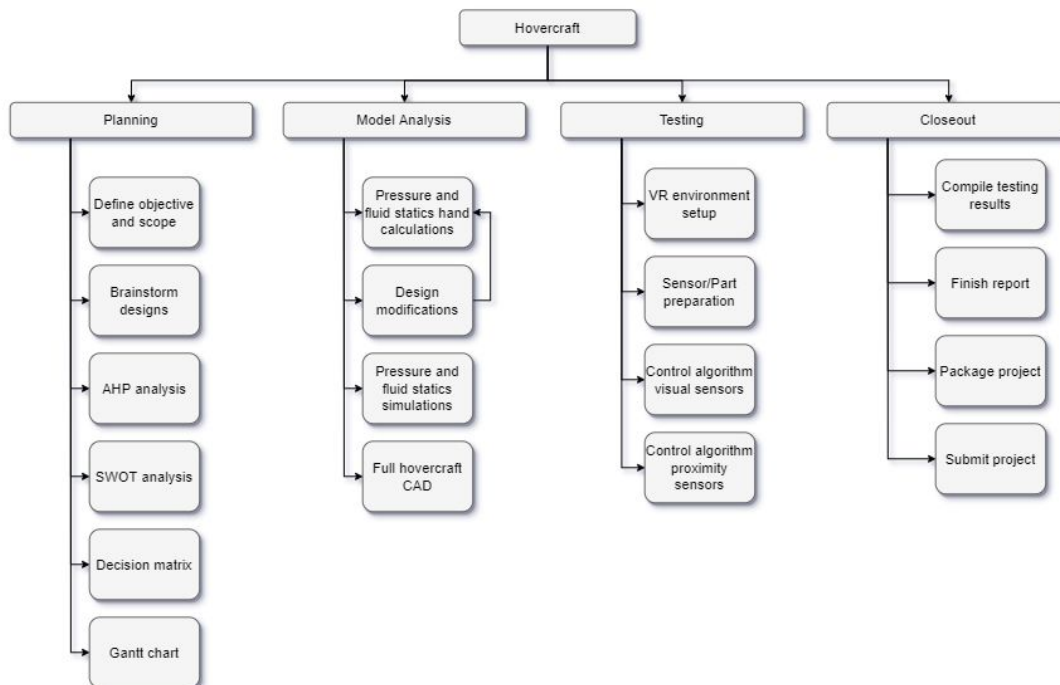


Figure 8.4: Work Breakdown Structure

8.3 Timeline and Milestones (Gantt Chart)

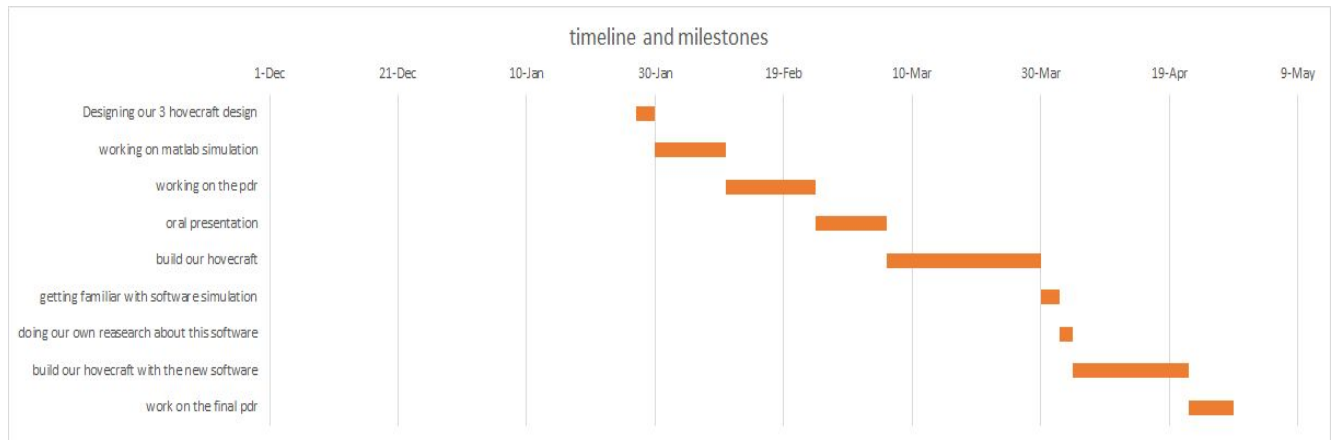


Figure 8.5: Timeline and Milestones

8.4 Original Project Work Done

Before the pandemic situation project changed on March 23rd, we had done some work on the original project. We had started to run some simulations on our chosen designs and started doing this final report. Therefore, we had to adapt this report for our new project guidelines. The simulations could be continued as normal, since it does not change with the new requirements.

9.RISK MANAGEMENT AND CONTINGENCY

Our project will require a risk management and contingency planning strategy in order to deal with identified or unknown risks. These strategies will allow us to anticipate when things may go wrong and how we will deal with these issues as they arise. We may also use them as guidelines for our project deliverables, as a means of avoiding obstacles when possible.

| Risk Management and Contingency Planning | | |
|--|---|---|
| Risks | Definition | Strategy |
| Cost | The cost of the project may exceed the planned cost. The cost in this case is measured in labor hours. | In order to gain control over the labor hours consumed in the implementation of the project, we will set realistic goals for the performance of the simulation, and only improve on it with given excess time. |
| Schedule | The desired product may not be submitted on time due to unforeseen obstacles | Submission on time is crucial. Therefore, we will plan our delivery date to be 1 week prior to the hard deadline of April 29th. Our new soft deadline will be scheduled for April 22nd. |
| Technical | Performance objects may not be met, as defined in the requirements | Given that a minimum performance goal is defined in the requirements of the project, we will focus on that. If that goal is too difficult, we shall seek help from advisors, such as our TA or professor through the Q&A forum. |
| Management | Task division, completion and/or social interference between members | If team cooperation is challenging due to personal schedules or task division, we shall schedule meetings when possible to discuss pivoting strategies. |
| Support | Ability to find resources to overcome challenges | If resources are scarce, we shall ask for advice from our TA or professor through the Q&A forum. |
| Production | Packaging the project, submission task | If packaging the project is deemed impossible, we shall communicate as quickly as possible with our TA or professor. |
| Feasibility | Ability to meet requirements as mentioned | If the minimum requirements cannot be met by the submission deadline, we shall write a report describing our progress and our challenges in doing so. |
| Test | Testing environment and/or simulation not set up properly | If the environment is proving unstable or unreliable, we shall seek advice from our TA or professor through the Q&A forum. |

Table 9.1: Risk Management and Contingency Planning

10.SUMMARY

10.1 Overall Goal

The goal of this project was to build a hovercraft that will be able to complete the track with a certain maximum amount of time. We had to make sure that our hovercraft had the lowest score based on the number of components and the time the hovercraft took to finish the course. We made 3 designs and tested them all to make sure we chose the right one. Based on the SWOT, WOT and AHP we were able to choose the best one with the lowest score. We choose the model with two fans in the back for the thrust and one in the middle facing the ground for the lift.

10.2 Overall Score

With the final simulations of the hovercraft done we calculated our final score below:

$$\text{Score} = (30 \text{ seconds} * 3) + (20 \text{ seconds} * 3) = 150$$

Our total score is 150 for our project simulation, this score could be lowered if the number of sensors was lowered but we couldn't reduce the number of sensors and keep the hovercraft stable while traversing the maze. We also made some calculation using matlab to make sure that our hovercraft can either turn fast enough will lift high enough to pass all the obstacle. We had to made sure that everything can fit on the hovercraft and that it still balance

10.3 Project Changes

We first had to build a hovercraft with giving parts by Dr Dimitry. We had to code our sensor using Arduino. Due to the pandemic situation, we had to do some changes to our project. We had to make a virtual simulation of our hovercraft on CoppeliaSim[4]. It was difficult to adapt to the new requirements and the new software with only a month left to the semester. With the help of our TA that gave us two class on zoom about this software helped us a lot. But considering all the circumstances, we were able to complete the project in time. We would love to refine our simulation to be faster and act better. We simply lacked time and experience using this software to do so. With a total score of 150, we are satisfied with our project.

11.GLOSSARY

11.1 Youtube Page

Our hovercraft simulation is available at our [youtube](https://www.youtube.com/channel/UCcxSa1OLhOfCuZCnMNzWJUA) channel. [6]
<https://www.youtube.com/channel/UCcxSa1OLhOfCuZCnMNzWJUA>

11.2 Source Code

11.2.1 Proximity sensor:

```
function getDist(sonar,max_dist)
    local detected, distance
    detected , distance = sim.readProximitySensor(sonar)
    if (detected < 0.9) then
        distance = max_dist
    end
    return distance
end

function getDistLeft(sonar,max_dist)
    local detected, distance
    detected , distance = sim.readProximitySensor(sonar)
    if (detected < 0.5 ) then
        distance = max_dist
    end
    return distance
end

function getDistRight(sonar,max_dist)
    local detected, distance
    detected , distance = sim.readProximitySensor(sonar)
    if (detected < 0.5 ) then
        distance = max_dist
    end
    return distance
end
```

```

function sysCall_init()
    -- Make sure we have version 2.4.13 or above (the particles are not
       supported otherwise)
    v=sim.getInt32Parameter(sim.intparam_program_version)
    if (v<20413) then
        sim.displayDialog('Warning','The propeller model is only fully
                           supported from CoppeliaSim version 2.4.13 and
                           above.&&This simulation will not run as
                           Expected!',
                           sim.dlgstyle_ok,false,'',nil,{0.8,0,0,0,0,0})
    end

    -- Detach the manipulation sphere:
    targetObj=sim.getObjectHandle('Quadricopter_target')
    sim.setObjectParent(targetObj,-1,true)

    -- This control algo was quickly written and is dirty and not optimal. It
       just serves as a SIMPLE example
    d=sim.getObjectHandle('Quadricopter_base')

    particlesAreVisible=sim.getScriptSimulationParameter(sim.handle_self,
                                                         'particlesAreVisible')
    sim.setScriptSimulationParameter(sim.handle_tree,'particlesAreVisible',
                                     tostring(particlesAreVisible))
    simulateParticles=sim.getScriptSimulationParameter(sim.handle_self,
                                                         'simulateParticles')
    sim.setScriptSimulationParameter(sim.handle_tree,'simulateParticles',
                                     tostring(simulateParticles))
    propellerScripts={-1,-1,-1,-1}

    for i=1,4,1 do
        propellerScripts[i]=sim.getScriptHandle(
                                'Quadricopter_propeller_respondable'..i)
    end

    heli=sim.getObjectAssociatedWithScript(sim.handle_self)
    particlesTargetVelocities={0,0,0,0}

```

```

pParam=2
iParam=0
dParam=0
vParam=-2

cumul=0
lastE=0
pAlphaE=0
pBetaE=0
psp2=0
psp1=0

prevEuler=0

fakeShadow=sim.getScriptSimulationParameter(sim.handle_self,'fakeShadow')
if (fakeShadow) then
    shadowCont=sim.addDrawingObject(sim.drawing_discpoints+
        sim.drawing_cyclic+sim.drawing_25percenttransparency+
        sim.drawing_50percenttransparency+
        sim.drawing_itemsizes,0.2,0,-1,1)
end

-- Prepare 2 floating views with the camera views:
floorCam=sim.getObjectHandle('Quadricopter_floorCamera')
frontCam=sim.getObjectHandle('Quadricopter_frontCamera')
floorView=sim.floatingViewAdd(0.9,0.9,0.2,0.2,0)
frontView=sim.floatingViewAdd(0.7,0.9,0.2,0.2,0)
sim.adjustView(floorView,floorCam,64)
sim.adjustView(frontView,frontCam,64)

sonar = sim.getObjectHandle('PS')
leftSonar = sim.getObjectHandle('PSLeft')
rightSonar = sim.getObjectHandle('PSRight')
max_dist = 2
last_time = sim.getSimulationTime()
ccw = false
quad = sim.getObjectHandle("lift")
leftPropeller = sim.getScriptHandle('left_propeller#0')
rightPropeller = sim.getScriptHandle('left_propeller')

end

```

```

function sysCall_cleanup()
    sim.removeDrawingObject(shadowCont)
    sim.floatingViewRemove(floorView)
    sim.floatingViewRemove(frontView)
end

function sysCall_actuation()
    s=sim.getObjectSizeFactor(d)
    pos=sim.getObjectPosition(d,-1)
    if (fakeShadow) then
        itemData={pos[1],pos[2],0.002,0,0,1,0.2*s}
        sim.addDrawingObjectItem(shadowCont,itemData)
    end

    -- Vertical control:
    targetPos=sim.getObjectPosition(targetObj,-1)
    pos=sim.getObjectPosition(d,-1)
    l=sim.getVelocity(heli)
    e=(targetPos[3]-pos[3])
    cumul=cumul+e
    pv=pParam*e
    thrust=5.335+pv+iParam*cumul+dParam*(e-lastE)+l[3]*vParam
    lastE=e

    -- Horizontal control:
    sp=sim.getObjectPosition(targetObj,d)
    -----
    --print(targetPos)
    dist = getDist(sonar,max_dist)
    distLeft = getDistLeft(leftSonar,max_dist)
    distRight = getDistRight(rightSonar,max_dist)
    --print(dist)
    step = 0.01

    sim.setScriptSimulationParameter(leftPropeller , 'particleVelocity' , 0.5)
    sim.setScriptSimulationParameter(rightPropeller , 'particleVelocity' , 0.5)

```

```

if dist > 0.9 then
    sim.setScriptSimulationParameter(leftPropeller,'particleVelocity',0.5)
    sim.setScriptSimulationParameter(rightPropeller,'particleVelocity',0.5)
    print('no wall')
    if distLeft < 0.1 then
        --turn right
        for i=1,5 do
            sim.setScriptSimulationParameter(rightPropeller
                                                , 'particleVelocity' , -1)
        end
    end
end

if distRight < 0.1 then
    --turn left
    for i=1,5 do
        sim.setScriptSimulationParameter(leftPropeller ,
                                          'particleVelocity' , -1)
    end
end
else
    for i=1,50 do
        sim.setScriptSimulationParameter(leftPropeller ,
                                          'particleVelocity' , -1)
        sim.setScriptSimulationParameter(rightPropeller ,
                                          'particleVelocity' , -1)
    end

    print('wall')
    if distLeft < distRight then
        for i=1,3 do
            --reverse
            sim.setScriptSimulationParameter(leftPropeller ,
                                              'particleVelocity' , -50)
            sim.setScriptSimulationParameter(rightPropeller ,
                                              'particleVelocity' , -50)
        end
    end
end

```

```

        print('go right')
        for i=1,300 do
            --turn right
            sim.setScriptSimulationParameter(leftPropeller ,
                                                'particleVelocity' , 150)
            sim.setScriptSimulationParameter(rightPropeller ,
                                                'particleVelocity' , -150)
        end
    else
        if distRight < distLeft then
            for i=1,5 do
                --turn reverse
                sim.setScriptSimulationParameter(leftPropeller ,
                                                    'particleVelocity' , -50)
                sim.setScriptSimulationParameter(rightPropeller ,
                                                    'particleVelocity' , -50)
            end

            print('go left')
            for i=1,300 do
                --turn Left
                sim.setScriptSimulationParameter(leftPropeller ,
                                                    'particleVelocity' , -150)
                sim.setScriptSimulationParameter(rightPropeller ,
                                                    'particleVelocity' , 150)
            end
        end
    end
end
end
-----
m=sim.getObjectMatrix(d,-1)
vx={1,0,0}
vx=sim.multiplyVector(m,vx)
vy={0,1,0}
vy=sim.multiplyVector(m,vy)

```

11.2.2 Vision sensor:

```

function sysCall_init()
    -- Make sure we have version 2.4.13 or above (the particles are not
    supported otherwise)
    v=sim.getInt32Parameter(sim.intparam_program_version)

```

```

if (v<20413) then
    sim.displayDialog('Warning','The propeller model is only fully
        supported from CoppeliaSim version 2.4.13 and
        above.&&This simulation will not run as
        Expected!',sim.dlgstyle_ok,false,'',
        ,nil,{0.8,0,0,0,0,0})
end

-- Detach the manipulation sphere:
targetObj=sim.getObjectHandle('Quadricopter_target')
sim.setObjectParent(targetObj,-1,true)

-- This control algo was quickly written and is dirty and not optimal.
  It just serves as a SIMPLE example

d=sim.getObjectHandle('Quadricopter_base')
particlesAreVisible=sim.getScriptSimulationParameter(sim.handle_self,
    'particlesAreVisible')
sim.setScriptSimulationParameter(sim.handle_tree,'particlesAreVisible',
    tostring(particlesAreVisible))
simulateParticles=sim.getScriptSimulationParameter(sim.handle_self,
    'simulateParticles')
sim.setScriptSimulationParameter(sim.handle_tree,'simulateParticles',
    tostring(simulateParticles))
propellerScripts={-1,-1,-1,-1}

for i=1,4,1 do
    propellerScripts[i]=sim.getScriptHandle(
        'Quadricopter_propeller_respondable'..i)
end

heli=sim.getObjectAssociatedWithScript(sim.handle_self)
particlesTargetVelocities={0,0,0,0}

pParam=2
iParam=0
dParam=0
vParam=-2

cumul=0

```



```

lastE=0
pAlphaE=0
pBetaE=0
psp2=0
psp1=0

prevEuler=0

fakeShadow=sim.getScriptSimulationParameter(sim.handle_self,
                                             'fakeShadow')

if (fakeShadow) then
    shadowCont=sim.addDrawingObject(sim.drawing_discontinuity+
                                    sim.drawing_cyclic+sim.drawing_25percenttransparency+
                                    sim.drawing_50percenttransparency+
                                    sim.drawing_itemsizes,0.2,0,-1,1)
end

-- Prepare 2 floating views with the camera views:
floorCam=sim.getObjectHandle('Quadricopter_floorCamera')
frontCam=sim.getObjectHandle('Quadricopter_frontCamera')
VSfront = sim.getObjectHandle('VSFront')
VSleft = sim.getObjectHandle('VSLeft')
VSright = sim.getObjectHandle('VSRight')
floorView=sim.floatingViewAdd(0.9,0.9,0.2,0.2,0)
frontView=sim.floatingViewAdd(0.9,0.7,0.2,0.2,0)
vsViewFront=sim.floatingViewAdd(0.9,0.5,0.2,0.2,0)
vsViewLeft=sim.floatingViewAdd(0.9,0.3,0.2,0.2,0)
vsViewRight=sim.floatingViewAdd(0.9,0.1,0.2,0.2,0)
sim.adjustView(floorView,floorCam,64)
sim.adjustView(frontView,frontCam,64)
sim.adjustView(vsViewFront,VSfront,64)
sim.adjustView(vsViewLeft,VSleft,64)
sim.adjustView(vsViewRight,VSright,64)

max_dist = 2

last_time = sim.getSimulationTime()
ccw = false

quad = sim.getObjectHandle("lift")

```

```

leftPropeller = sim.getScriptHandle('left_propeller')
rightPropeller = sim.getScriptHandle('left_propeller#0')

lineC = 0
lineCleft = 0
lineCright = 0
i=1
end

function sysCall_cleanup()
    sim.removeDrawingObject(shadowCont)
    sim.floatingViewRemove(floorView)
    sim.floatingViewRemove(frontView)
    sim.floatingViewRemove(vsViewFront)
    sim.floatingViewRemove(vsViewLeft)
    sim.floatingViewRemove(vsViewRight)
end

function sysCall_actuation()
    lineC = getLINEC(VSfront)
    lineCleft = getLINEC(VSleft)
    lineCright = getLINEC(VSright)

    s=sim.getObjectSizeFactor(d)
    pos=sim.getObjectPosition(d,-1)

    if (fakeShadow) then
        itemData={pos[1],pos[2],0.002,0,0,1,0.2*s}
        sim.addDrawingObjectItem(shadowCont,itemData)
    end

    -- Vertical control:
    targetPos=sim.getObjectPosition(targetObj,-1)

    pos=sim.getObjectPosition(d,-1)

    l=sim.getVelocity(heli)

```

```

e=(targetPos[3]-pos[3])
cumul=cumul+e
pv=pParam*e
thrust=5.335+pv+iParam*cumul+dParam*(e-lastE)+l[3]*vParam
lastE=e

-- Horizontal control:
sp=sim.getObjectPosition(targetObj,d)
-----
--print(targetPos)
step = 0.01

--0.09 makes HC not move use it as 0
sim.setScriptSimulationParameter(leftPropeller , 'particleVelocity', 20)
sim.setScriptSimulationParameter(rightPropeller, 'particleVelocity', 20)

if (lineC > 0.1)then
    print('wall')
    if(i==1) then
        print('first corner')
        for i=1,100 do
            sim.setScriptSimulationParameter(leftPropeller ,
                                                'particleVelocity' , 100)
            sim.setScriptSimulationParameter(rightPropeller ,
                                                'particleVelocity' , -100)
        end

        for i=1,500 do

        end
        i=2

    elseif(i==2) then
        print('second corner')
        for i=1,100 do
            sim.setScriptSimulationParameter(leftPropeller ,
                                                'particleVelocity' , -100)
            sim.setScriptSimulationParameter(rightPropeller ,

```

```

                                'particleVelocity' , 100)
    end

    for i=1,500 do

        end
        i=3
    elseif(i==3) then
        print('third corner')
        for i=1,100 do
            sim.setScriptSimulationParameter(leftPropeller ,
                                                'particleVelocity' , 100)
            sim.setScriptSimulationParameter(rightPropeller ,
                                                'particleVelocity' , -100)
        end

        for i=1,500 do

            end
            i=1
        end
    else
        print('no wall')
        if (lineCleft > 0.1)then
            print('left wall-no front')
            sim.setScriptSimulationParameter(leftPropeller ,
                                                'particleVelocity' , 10)
            sim.setScriptSimulationParameter(rightPropeller ,
                                                'particleVelocity' , 0.09)
        elseif (lineCright > 0.1)then
            print('right wall-no front')
            sim.setScriptSimulationParameter(leftPropeller ,
                                                'particleVelocity' , 0.09)
            sim.setScriptSimulationParameter(rightPropeller ,
                                                'particleVelocity' , 10)
        else
            sim.setScriptSimulationParameter(leftPropeller ,
                                                'particleVelocity' , 20)
            sim.setScriptSimulationParameter(rightPropeller ,
                                                'particleVelocity' , 20)
        end
    end
end

```

```

end
-----
m=sim.getObjectMatrix(d,-1)
vx={1,0,0}
vx=sim.multiplyVector(m,vx)
vy={0,1,0}
vy=sim.multiplyVector(m,vy)
alphaE=(vy[3]-m[12])
alphaCorr=0.25*alphaE+2.1*(alphaE-pAlphaE)
betaE=(vx[3]-m[12])
betaCorr=-0.25*betaE-2.1*(betaE-pBetaE)
pAlphaE=alphaE
pBetaE=betaE
alphaCorr=alphaCorr+sp[2]*0.005+1*(sp[2]-psp2)
betaCorr=betaCorr-sp[1]*0.005-1*(sp[1]-psp1)
psp2=sp[2]
psp1=sp[1]

-- Rotational control:
euler=sim.getObjectOrientation(d,targetObj)
rotCorr=euler[3]*0.1+2*(euler[3]-prevEuler)
prevEuler=euler[3]

-- Decide of the motor velocities:
particlesTargetVelocities[1]=thrust*(1-alphaCorr+betaCorr+rotCorr)
particlesTargetVelocities[2]=thrust*(1-alphaCorr-betaCorr-rotCorr)
particlesTargetVelocities[3]=thrust*(1+alphaCorr-betaCorr+rotCorr)
particlesTargetVelocities[4]=thrust*(1+alphaCorr+betaCorr-rotCorr)

-- Send the desired motor velocities to the 4 rotors:
for i=1,4,1 do
    sim.setScriptSimulationParameter(propellerScripts[i],
                                    'particleVelocity',particlesTargetVelocities[i])
end
end
function change(a,i)
    a[i] = a[i] + 0.5
    return a
end

function getLINEC(VS)

```

```
    value = sim.getVisionSensorImage(VS + sim.handleflag_greyscale)
    return value[1]
end
```

12.REFERENCES

- [1]: Competition_rule_ENGR290_2020.pdf (available on moodle)
- [2]:https://ocw.mit.edu/courses/mechanical-engineering/2-017j-design-of-electromechanical-robotic-systems-fall-2009/assignments/MIT2_017JF09_p29.pdf
- [3]:<https://www.cds.caltech.edu/~murray/wiki/images/f/fc/Hovercraft-dynamics.pdf>
- [4]:<https://www.coppeliarobotics.com>
- [5]:<https://www.youtube.com/channel/UC2P7h5Vik9tLafPQ2hIKVLA>
- [6]:<https://www.youtube.com/channel/UCcxSa1OLhOfCuZCnMNzWJUA>