Abstract

This report describes the design and development of a physical hovercraft in a team for the ENGR 290 competition. The goal is to make the hovercraft complete the specific track bounded by certain constraints which need to be maintained throughout the competition. This report contains certain procedures based on analyzing the problem, brainstorming through discussion, and researching over the internet to generate the ideas. There are three designs proposed which meet the specific requirements. A model or prototype is made for each design and carried them to testing and evaluating phases using MATLAB, WOT, AHP and SWOT analysis. Based on the analyses among the three designs, one final choice is narrowed down. Finally, the goal is to simulate the original hovercraft designs in a VR environment called CoppeliaSim.

Concordia University

Final Project Report

ENGR 290 - Introductory Engineering Team Project

Prof: Khashayar Khorasani

Due date: 21st April

Team 17:

Jose Ricardo Monegro Quezada, 40087821

Hashim Nakhuda, 40068968

Kaiyu Nie, 40121808

Erfan Reza, 40080730

Akshay Bhaskaran, 40068230

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

Table of Contents

List of Figures

List of Tables

1.0 Introduction

1.1 What is a hovercraft?

A hovercraft is a means of transportation. Hovercrafts are used to traverse through a variety of terrain without hindrances. They work by a fan (or multiple fans) to produce thrust and lift so that the hovercraft can steer and hover over the terrain. Hovercrafts can have many different designs meant to optimize for specific uses.

1.2 Objective

In Engr 290 the objective is to design and build a miniature hovercraft that can traverse autonomously through a maze with three obstacles. This document is the results of the design phase. It will discuss the requirements, ideas and research, design selection, progress to date and the schedule used to help narrow down which design to select when moving onto the simulator and building phases.

2.0 Requirements

2.1 Competition Requirements

- Must complete the track in 60 seconds or less
- Must be able to navigate through the maze autonomously
- Must be able to identify turns and take them
- Must be able to identify the end of the track and stop when it reaches it *
- Must hover above the obstacles on the track (3mm max from the ground)
- Should use the least amount of components possible
- Must be able to fit inside the track, with enough headroom to maneuver (shown in Figure 1)

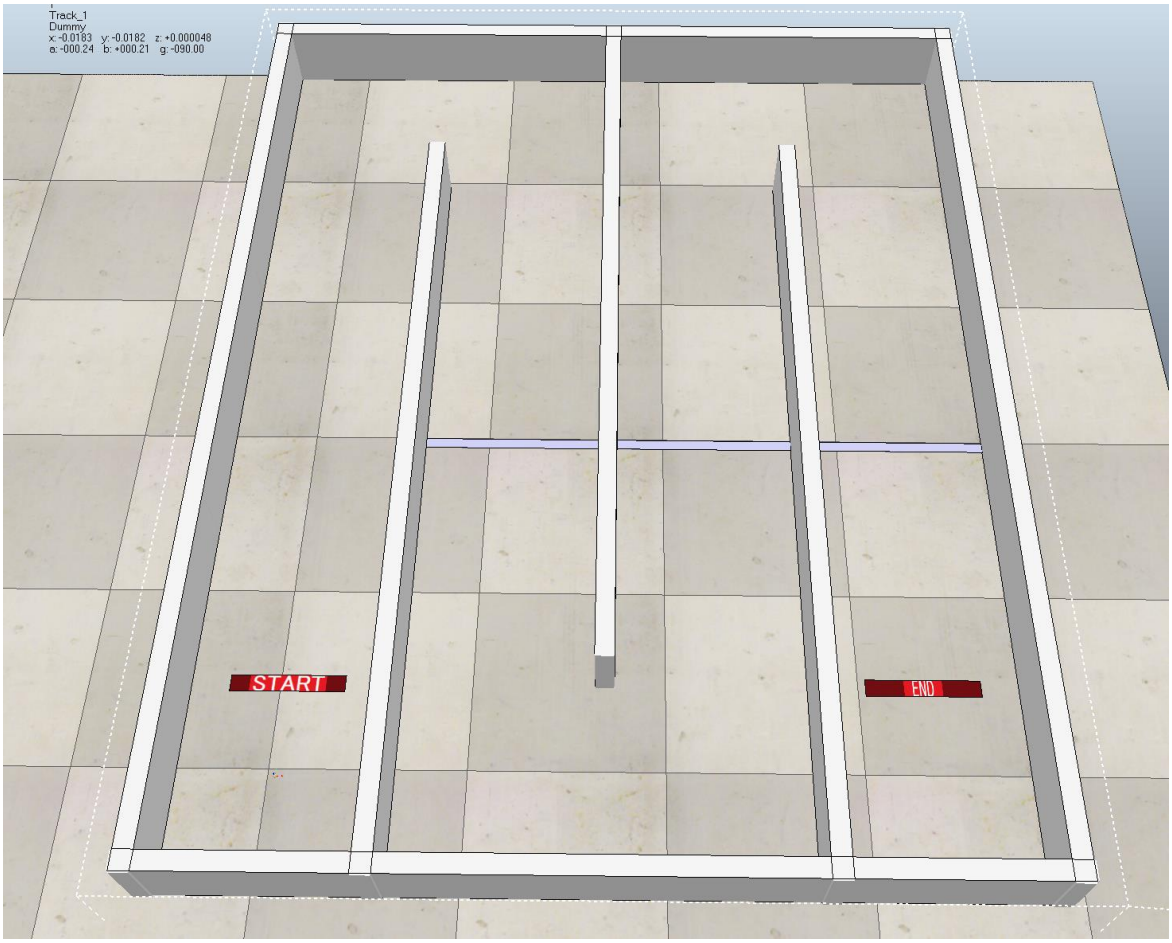


Fig 1. Track measurements

2.2 Competition Rule

1. Complete as much of the specified track as possible, operating autonomously.

2. Traverse as many of the increasingly challenging obstacles along the track.

3. Complete the course in as short a time as possible (1 min. time limit per each attempt)

4. Accomplish the Objectives 1-3 without using more resources (i.e. fans and servos) than necessary.

The scoring formula being = $d_{completed} /(N_c \times t_{course})$

Where $d_{completed}$ is the distance along the track that was successfully autonomously completed.

N_c is the number of components used.

t_{course} is the time (in seconds) taken to complete the course from the start line to the finish line.

3.0 Ideas and Research

By looking up the equation for the score, the ultimate goal is to make a hovercraft with the least number of components and finish the track as soon as possible. However, having the smallest number of components may cause more time to finish the track. Different designs may have certain advantages and disadvantages. A hovercraft may impossibly pass the obstacles on track since the lifting force is too weak. Having two or more fans will cause an increase in weight and the number of components used, but the hovercraft may have a higher speed and lift force. To get a higher score, the team needs to find a balance point where the hovercraft uses the least parts of components and maintains a decent speed and lift. After careful consideration, the following three designs find the best balance of the trade-off with the given requirements.

3.1 Design 1

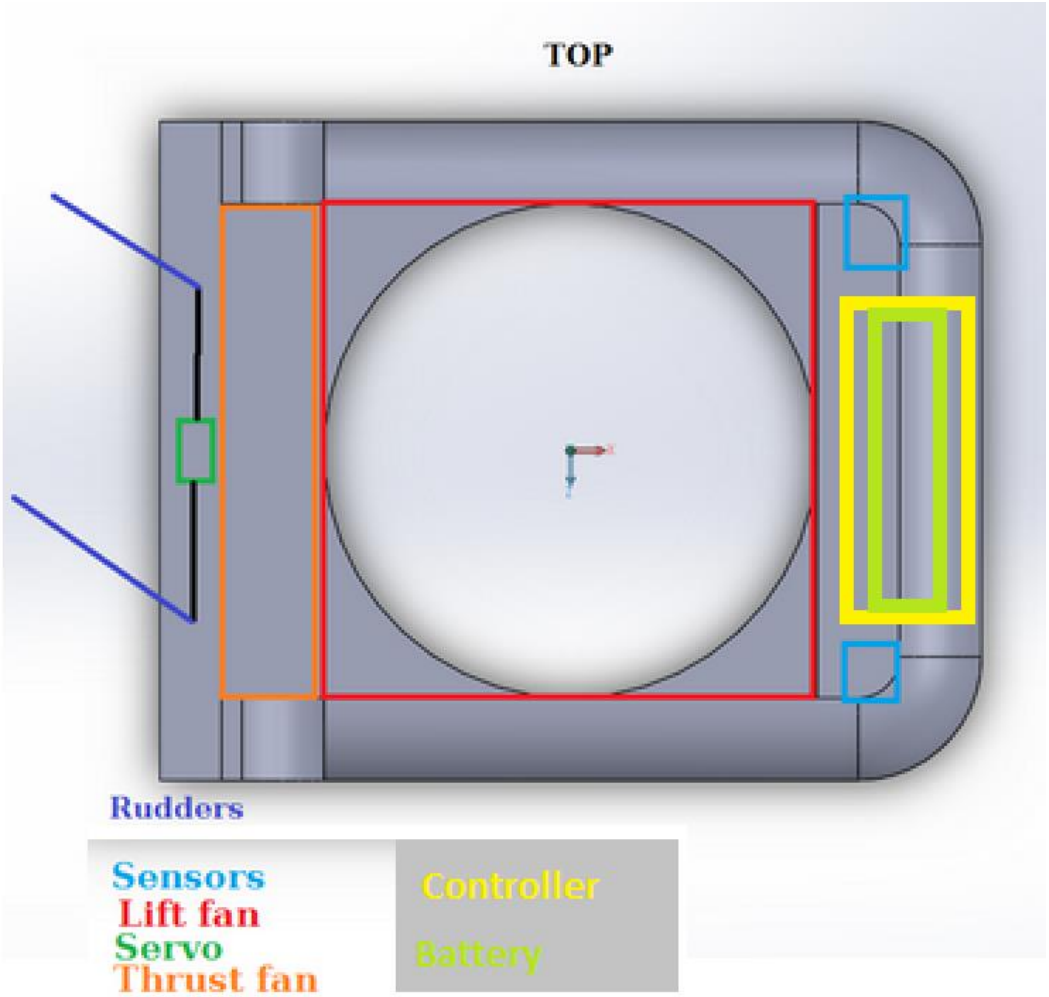


Fig 2.a Design 1 Top View

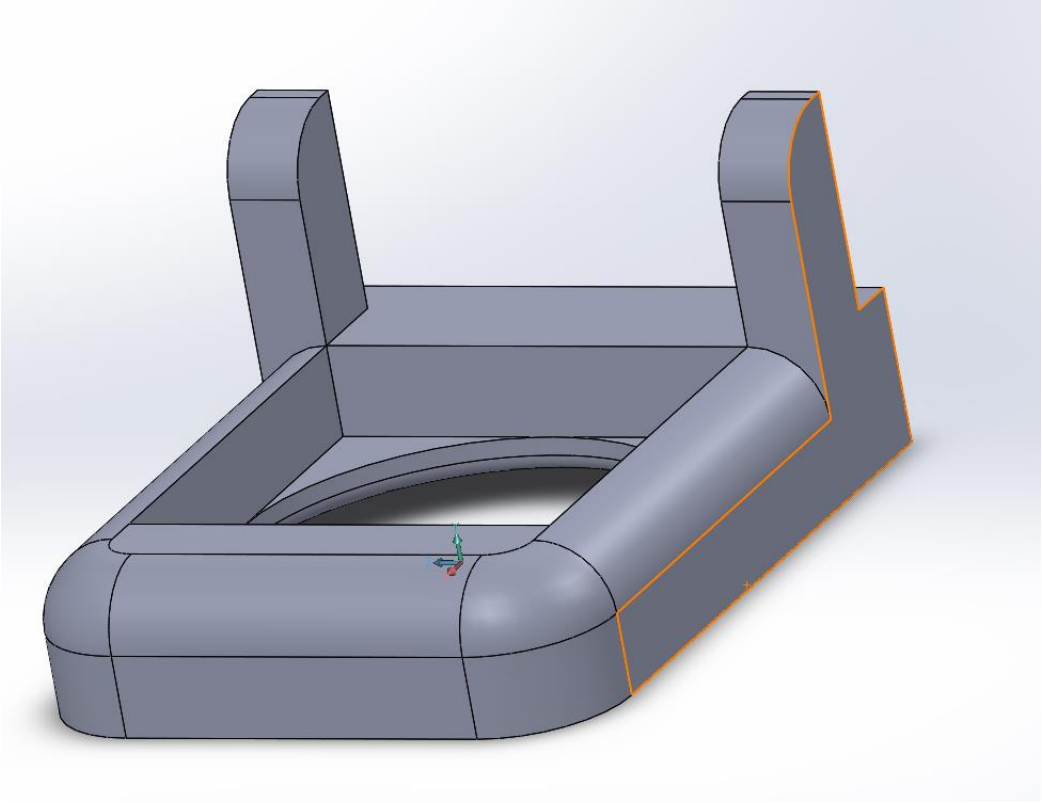
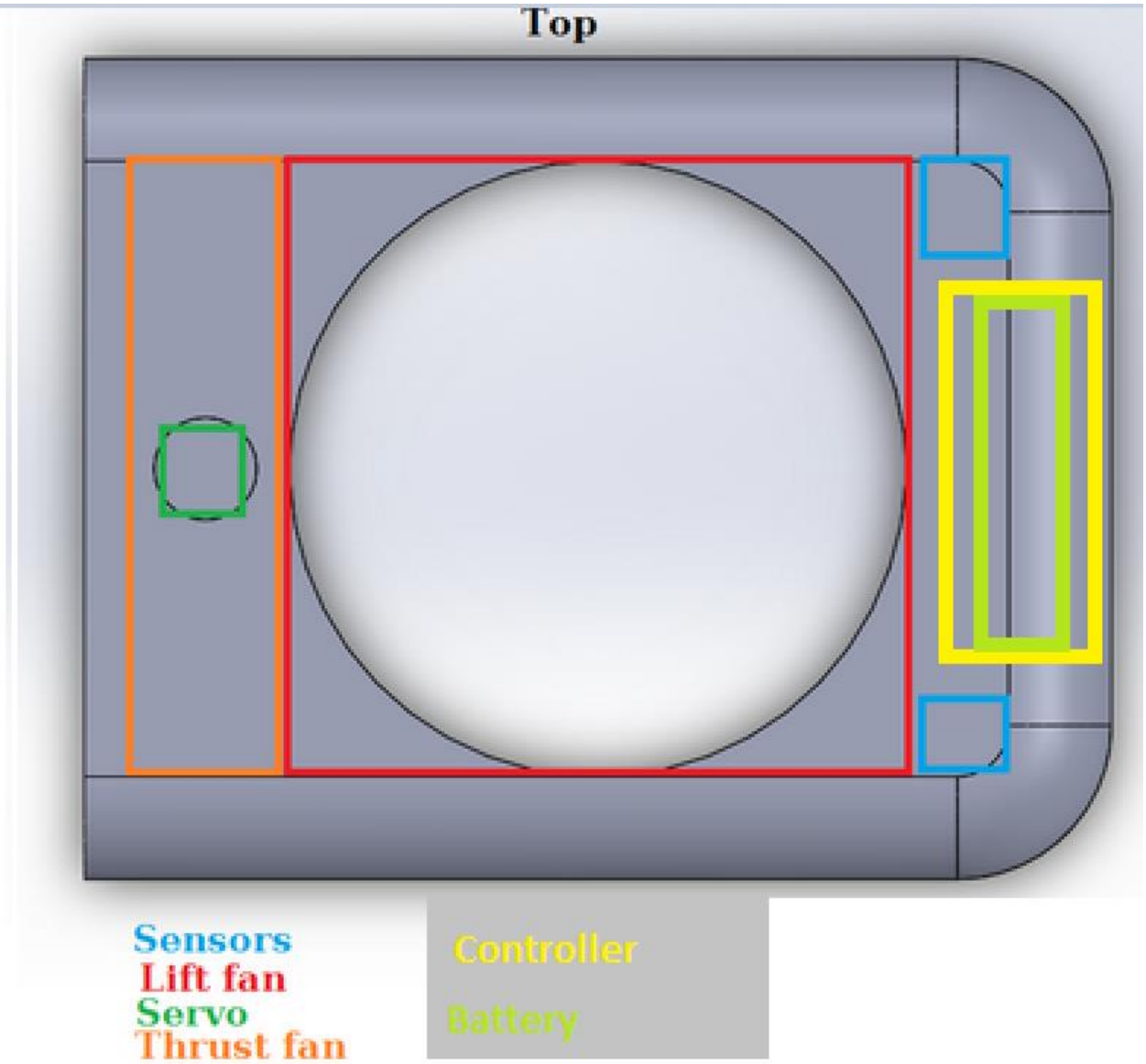


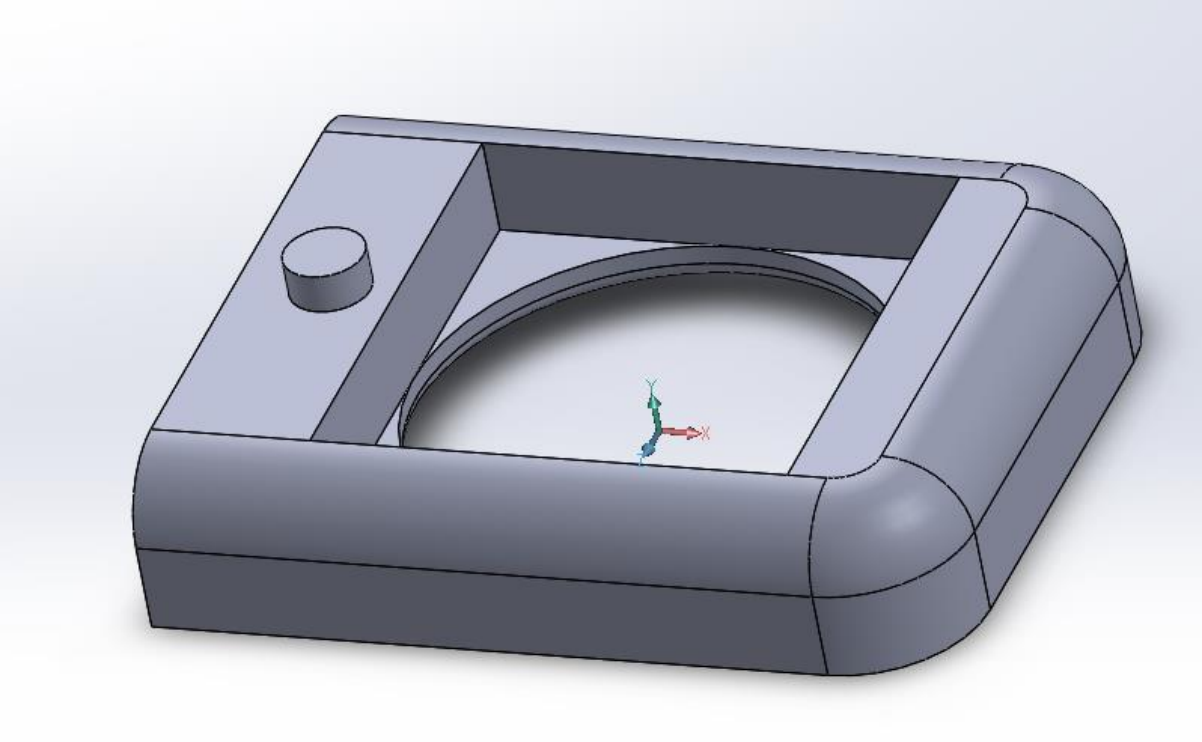
Fig 2.b Design 1 Side View

This idea comes from the basic concept of a hovercraft, where one fan provides lifting force and the other fan provides the thrust. Two sensors will be added to the front of the hovercraft pointing 45 degrees on both sides away from the center. There will be a rudder behind the fan, which provides the steering for the hovercraft.

3.2 Design 2



3.a Design 2 Top View



3.b Design 2 Side View

The idea comes from the Bell Boeing V-22 Osprey, an aircraft, which can do a vertical takeoff and landing by spinning its engines on the vertical axis. Using this concept this design was made. A fan with 360 degrees turning ability is used to provide the thrust that allows for optimal mobility. In comparison, another fan is used to provide lifting force. There will be two sensors for this design. One will be installed at the front of the hovercraft, pointing directly forward. The other will be installed on the right side of the hovercraft and facing to the right.

3.3 Design 3

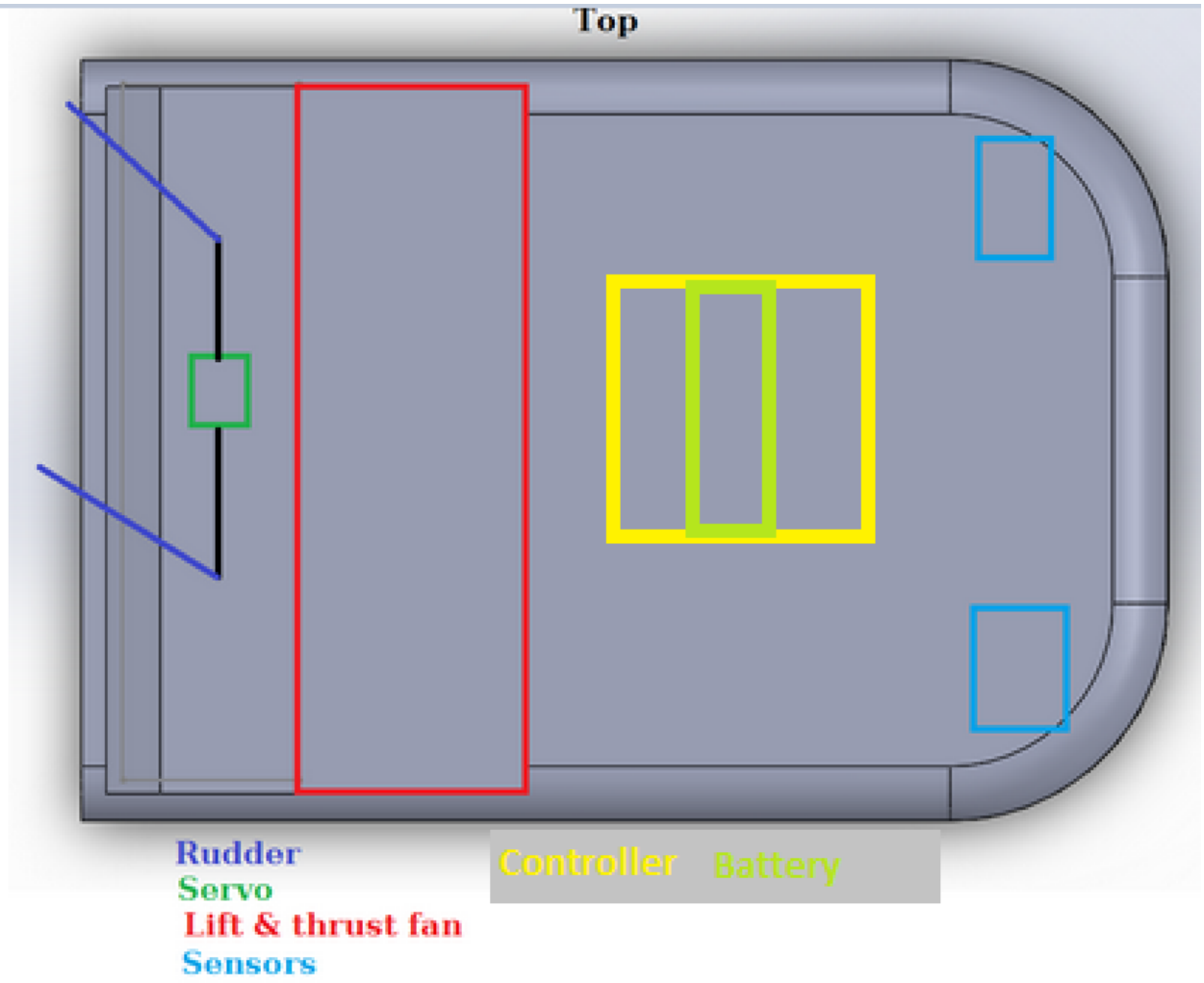


Fig 4.a Design 3 Top View

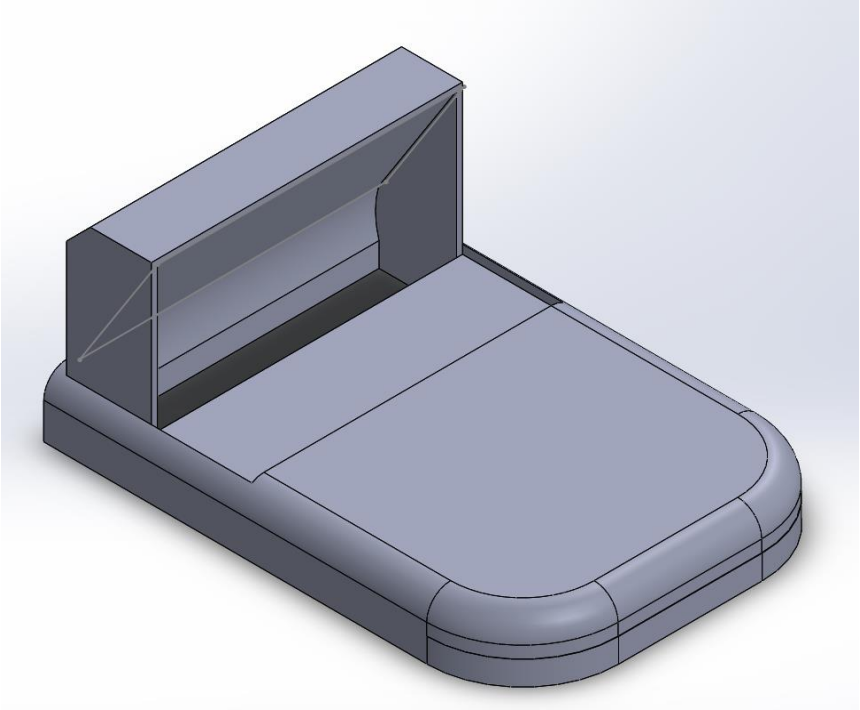


Fig 4.bDesign 3 Side View

The idea comes from a YouTube video[1] where the YouTuber made a mini 3D printed hovercraft with a radio control system. In order to minimize the number of components used, there will be only one fan for both lifting and pushing the hovercraft. Two sensors will be added to the front of the hovercraft and pointing 45 degrees to both sides since the goal is to design an automated hovercraft. There will be a rudder behind the fan which provides the thrust.

3.4 Summary of the Final Design Prototypes in Coppeliasim

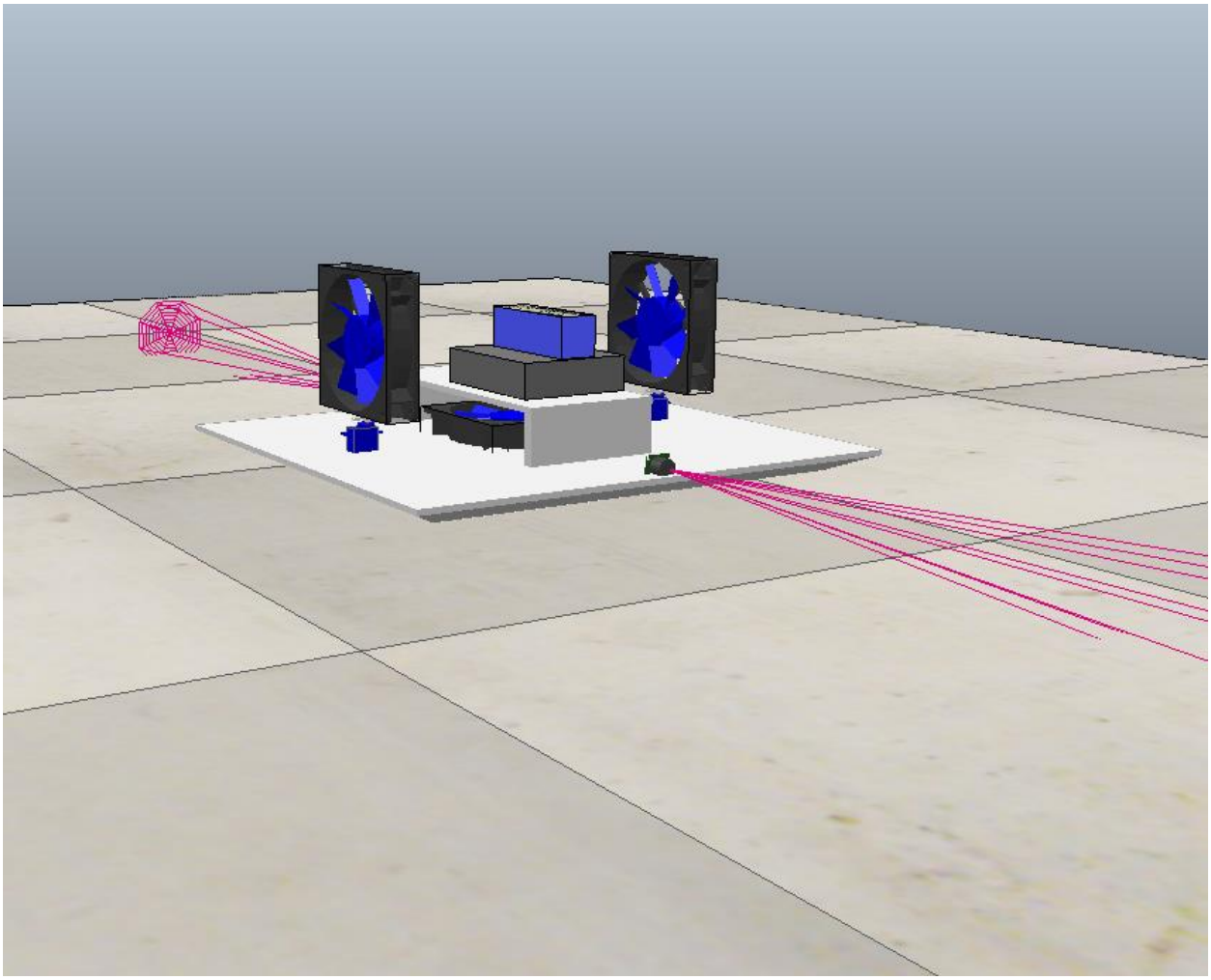


Fig 5.a Final Design Front/Side View

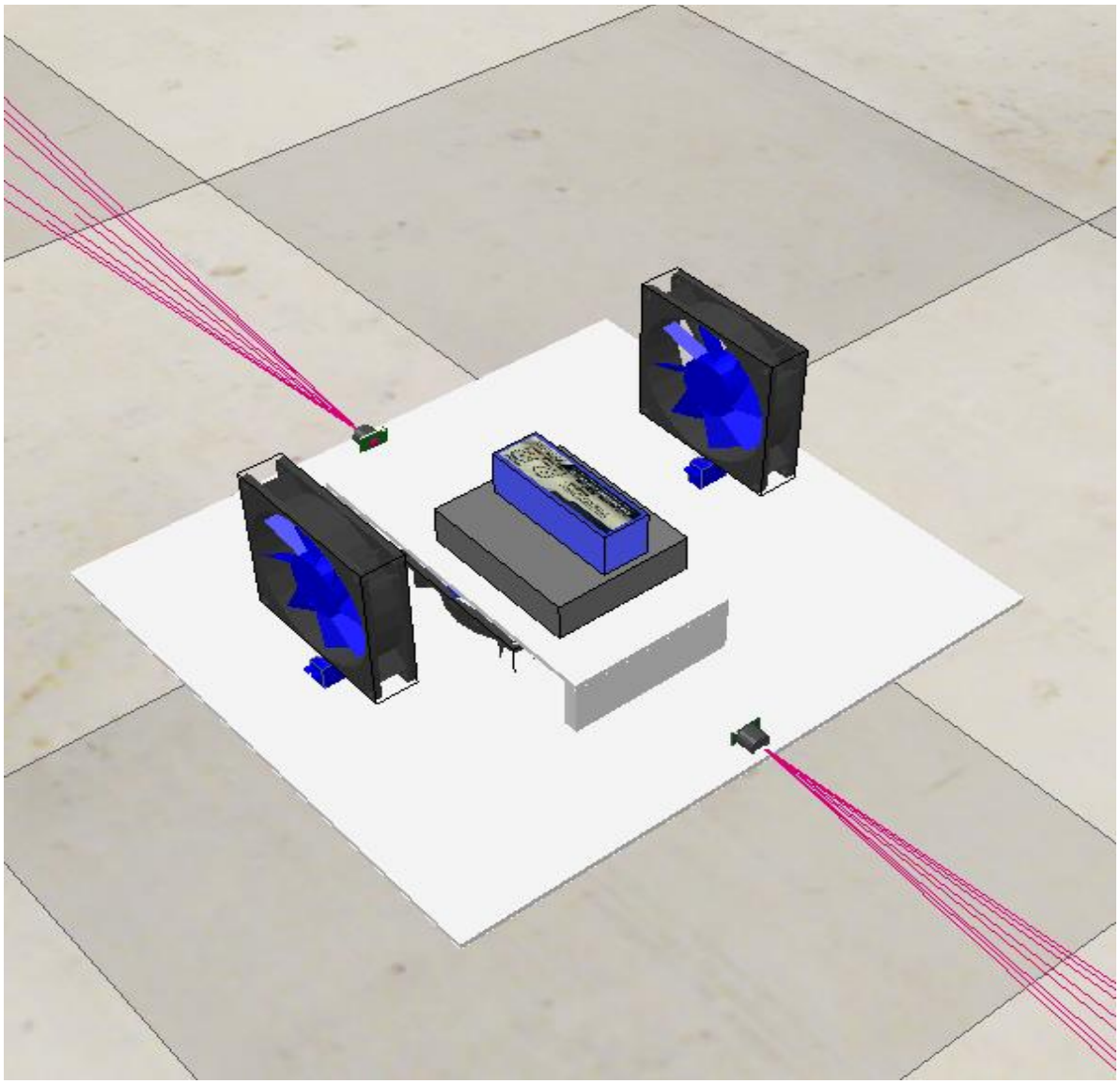


Fig 5.b Final Design Top Side View

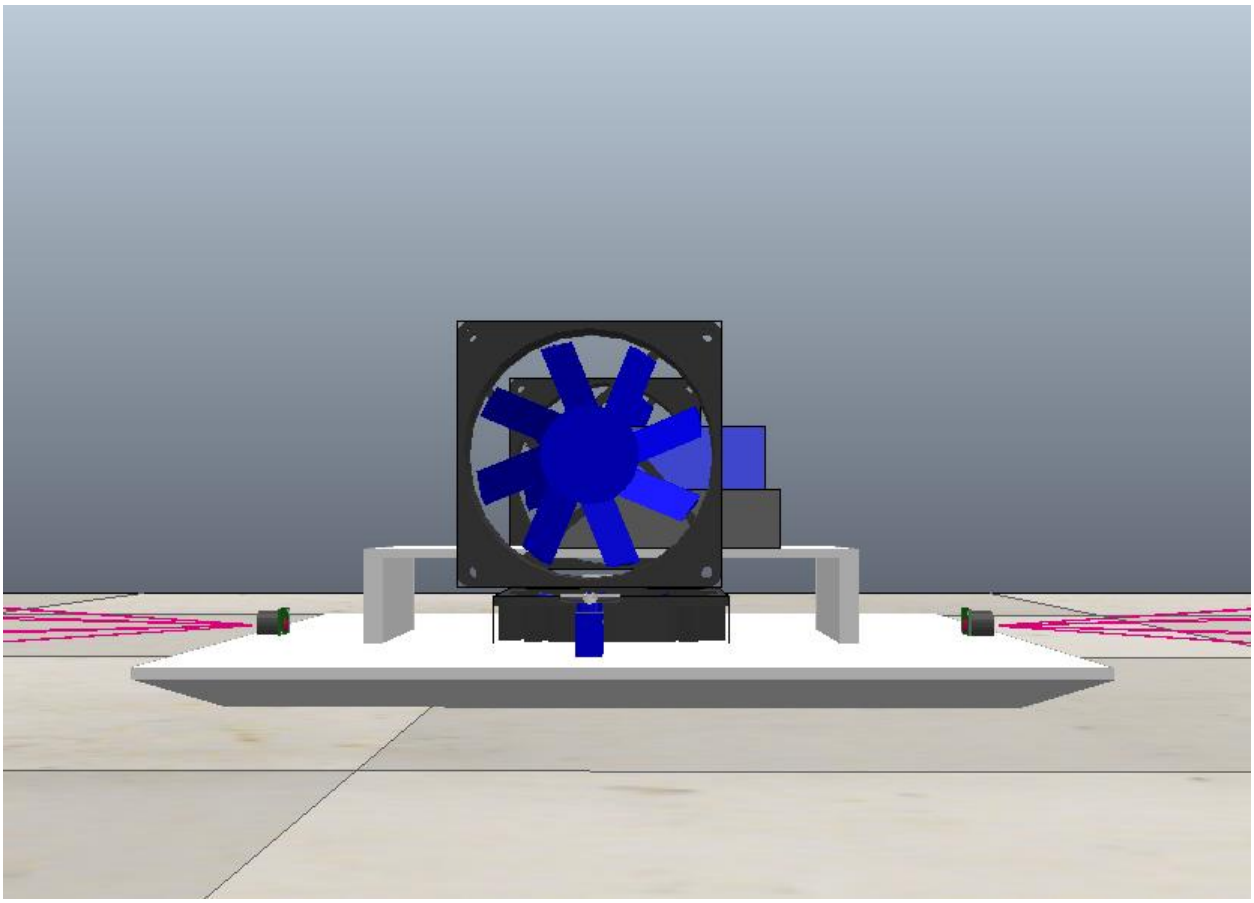


Fig 5.c Final Design Side View

After selecting design 2 as the model to be used when transitioning to the simulator we came across many dilemmas such as center of mass, steering and not enough lift. With this in mind we improved our design. We made a much larger base to help increase our lift as well as selected a heavier fan. We made a second platform over the lift fan to put more weight in the center to improve our center of mass. We added an extra thrust fan to help steering as well as speed during the course. With the new improvements the concept of how design 2 works is still preserved and is now implementable on the simulator.

4.0 Design Selection

4.1 Simple Decision Matrix:

		Design 1		Design 2		Design 3	
Criteria	Key Score	Score	Weighted Score	Score	Weighted Score	Score	Weighted Score
Weight	5	2	10	3	15	5	25
Angular Acceleration	3	5	15	6	18	3	9
Linear Acceleration	4	4	16	4	16	2	8
Max Speed	2	2	4	2	4	1	2
Number of Components	6	4	24	4	24	5	30
Lift	3	4	12	4	12	2	6
Total			81		89		80

Table 1. WOT analysis for 3 Designs

Design 2 has the highest total score.

By taking the equation for the score into consideration, the number of components has the highest score. Since the equation is $d_{completed}/(N_c \times t_{course})$, where $d_{completed}$ is the distance along the track that was successfully autonomously completed. N_c is the number of components used and t_{course} is the time taken to complete the course from the start line to the finish line. As was shown in the equation, once the track is finished, the number of components and the time consumed are the only two criteria that will influence the score. In this case, however, time criteria were divided into several different criteria. The total time can be calculated by total distance/ average speed. The average speed can be calculated using weight, angular acceleration, linear acceleration, and max speed.

Consider the obstacles on the track, the weight of a hovercraft becomes more important since the hovercraft has to pass those obstacles. For design 1 and design 2, both of them have a fan to provide lift

force. For design 3, however, there will be a single fan to provide both thrust and lift force. Therefore, the weight of the hovercraft cannot exceed a certain value. To fulfill the requirement in terms of weight, cardboard will be the material used to build the hovercraft.

- Number of Components: The number of components used is calculated simply based on how many components are used in the design.
- Weight: The total weight is calculated simply by adding all the components and the main body weight.
- Linear Acceleration & Angular acceleration: Linear and angular acceleration score is calculated based on the time needed for the hovercraft to complete a linear segment and a turn.
- Max Speed: Based on the distance covered by the hovercraft ratio with the time taken to finish the track, the max speed can be calculated.

SWOT Analysis:

Design	Strengths	Weakness	Opportunities	Threats
1	Maneuverability Stability	More power consumption Heaviest due to rudders Higher cost Lower score because of parts	Three tries	Has to complete the track
2	Best speed Maneuverability Stability Able to reverse	More power consumption Higher cost Blind when reversing Lower score because of parts	Three tries Know the track	Has to complete the track
3	Lowest weight Least parts used Low power consumption Less cost	Worst maneuverability Less stability Worst speed	Three tries	Time may prevent completion 3mm obstacle may prevent completion

Table 2. SWOT Analysis Table

4.2 Calculations

Calculation:

The material of the hovercraft will be styrofoam which is 0.5g/cm^3.

For designs 1 and 2, both of them are using the same components, therefore, the total mass of the component is the same.

2 fans (MEC0251V1-000U-A99) = 2 x 161.93g = 323.86g

1 battery (Rhino 460 mAh 20C series) = 28.5g

1 servo motor (DFRobot Micro Servo Motor) = 9g

1 controller = 36.85g

2 Ultrasonic Sensor = 2 x 4.3g = 8.6g

total mass of component = 517.96g

For design 3, there will be using only 1 fan, other parts remain the same.

Therefore, the total mass = 356.03g

From SolidWorks:

with the density of the main body = 0.05g/cm^3

	Design 1	Design 2	Design 3
Volume	673.5 cm^3	613.21 cm^3	637.78 cm^3
Mass	33.675 g	30.6605 g	31.889 g

Table 3. Mass/Volume comparison

Design 1

Parts	Mass(g)	Distance from Center (cm)	X Position (cm)	Y Position (cm)	Moment of Inertia ($kg*m^2$)
Lift Fan	161.93	0.0	0	0	0.00
Thrust Fan	161.93	7.25	-7.25	0	0.00085
Battery	28.50	8	8	0	0.00018
Controller	36.85	7.25	7.25	0	0.00019
Ultrasonic Sensor 1	4.30	9.2195	7	6	0.000037
Ultrasonic Sensor 2	4.30	9.2195	7	-6	0.000037
Servo motor	9.00	8.5	-8.5	0	0.000065

Table 4.a Table of Parts in Design 1

The total moment of inertia is 0.001359

$I_{Lift\ Fan} = m_{Lift\ Fan} \times r^2_{Lift\ Fan} = 161.93g \times (0cm)^2 = 0\ kg*m^2$

$I_{Thrust\ Fan} = m_{Thrust\ Fan} \times r^2_{Thrust\ Fan} = 161.93g \times (7.25cm)^2 = 0.00085\ kg*m^2$

$I_{battery} = m_{battery} \times r^2_{battery} = 28.5g \times (8.0cm)^2 = 0.00018\ kg*m^2$

$I_{Controller} = m_{Controller} \times r^2_{Controller} = 36.85g \times (7.25cm)^2 = 0.00023584\ kg*m^2$

$I_{Ultrasonic\ Sensor\ 1} = m_{Ultrasonic\ Sensor\ 1} \times r^2_{Ultrasonic\ Sensor\ 1} = 4.30g \times (9.2195cm)^2 = 0.000037\ kg*m^2$

$I_{Ultrasonic\ Sensor\ 2} = m_{Ultrasonic\ Sensor\ 2} \times r^2_{Ultrasonic\ Sensor\ 2} = 4.30g \times (9.2195cm)^2 = 0.000037\ kg*m^2$

$I_{Servo\ motor} = m_{Servo\ motor} \times r^2_{Servo\ motor} = 9.00g \times (8.50cm)^2 = 0.000065\ kg*m^2$

Design 2

Parts	Mass(g)	Distance from Center (cm)	X Position (cm)	Y Position (cm)	Moment of Inertia ($kg*m^2$)
Lift Fan	161.93	0.0	0	0	0.00
Thrust Fan	161.93	8.36	-8.36	0	0.001131722
Battery	28.50	8.05106	8	0.90526	0.000184735
Controller	36.85	8	8	0	0.00023584
Ultrasonic Sensor 1	4.30	8	8	0	0.00001548
Ultrasonic Sensor 2	4.30	6	0	-6	0.00001548
Servo motor	9.00	8.36	-8.36	0	0.000629006

Table 4.b Table of Parts in Design 2

The total moment of inertia is 0.002216

$I_{Lift\ Fan} = m_{Lift\ Fan} \times r^2_{Lift\ Fan} = 161.93g \times (0cm)^2 = 0\ kg*m^2$

$I_{Thrust\ Fan} = m_{Thrust\ Fan} \times r^2_{Thrust\ Fan} = 161.93g \times (8.36cm)^2 = 0.001131722\ kg*m^2$

$I_{battery} = m_{battery} \times r^2_{battery} = 28.5g \times (8.05106cm)^2 = 0.000184735\ kg*m^2$

$I_{Controller} = m_{Controller} \times r^2_{Controller} = 36.85g \times (8cm)^2 = 0.00023584\ kg*m^2$

$I_{Ultrasonic\ Sensor\ 1} = m_{Ultrasonic\ Sensor\ 1} \times r^2_{Ultrasonic\ Sensor\ 1} = 4.30g \times (6cm)^2 = 0.00001548\ kg*m^2$

$I_{Ultrasonic\ Sensor\ 2} = m_{Ultrasonic\ Sensor\ 2} \times r^2_{Ultrasonic\ Sensor\ 2} = 4.30g \times (6cm)^2 = 0.00001548\ kg*m^2$

$I_{Servo\ motor} = m_{Servo\ motor} \times r^2_{ervo\ motor} = 9.00g \times (8.36cm)^2 = 0.000629006\ kg*m^2$

Design3

Parts	Mass(g)	Distance from Center (cm)	X Position (cm)	Y Position (cm)	Moment of Inertia ($kg*m^2$)
Thrust Fan	161.93	2.5	-2.5	0	0.000101206
Battery	28.50	5.5	5.5	0	0.0015675
Controller	36.85	5.5	5.5	0	0.00202675
Ultrasonic Sensor 1	4.30	12.35	10.8	6	0.000065584
Ultrasonic Sensor 2	4.30	12.35	10.8	-6	0.000065584
Servo motor	9.00	5.25	-5.25	0	0.000024806

Table 4.c Table of Parts in Design 3

The total moment of inertia is 0.003857;

$I_{Thrust\ Fan} = m_{Thrust\ Fan} \times r^2_{Thrust\ Fan} = 161.93g \times (2.5cm)^2 = 0.000101206\ kg*m^2$

$I_{battery} = m_{battery} \times r^2_{battery} = 28.5g \times (5.5cm)^2 = 0.0015675\ kg*m^2$

$I_{Controller} = m_{Controller} \times r^2_{Controller} = 36.85g \times (5.5cm)^2 = 0.00202675\ kg*m^2$

$I_{Ultrasonic\ Sensor\ 1} = m_{Ultrasonic\ Sensor\ 1} \times r^2_{Ultrasonic\ Sensor\ 1} = 4.30g \times (12.35cm)^2 = 0.000065584\ kg*m^2$

$I_{Ultrasonic\ Sensor\ 2} = m_{Ultrasonic\ Sensor\ 2} \times r^2_{Ultrasonic\ Sensor\ 2} = 4.30g \times (12.35cm)^2 = 0.000065584\ kg*m^2$

$I_{Servo\ motor} = m_{Servo\ motor} \times r^2_{Servo\ motor} = 9.00g \times (5.25cm)^2 = 0.000024806\ kg*m^2$

4.3 Matlab Simulation

Acceleration: acceleration = force/mass

Velocity: velocity = acceleration .* time

Linear Displacement: linear displacement = average velocity .* t = current velocity/2 .* t

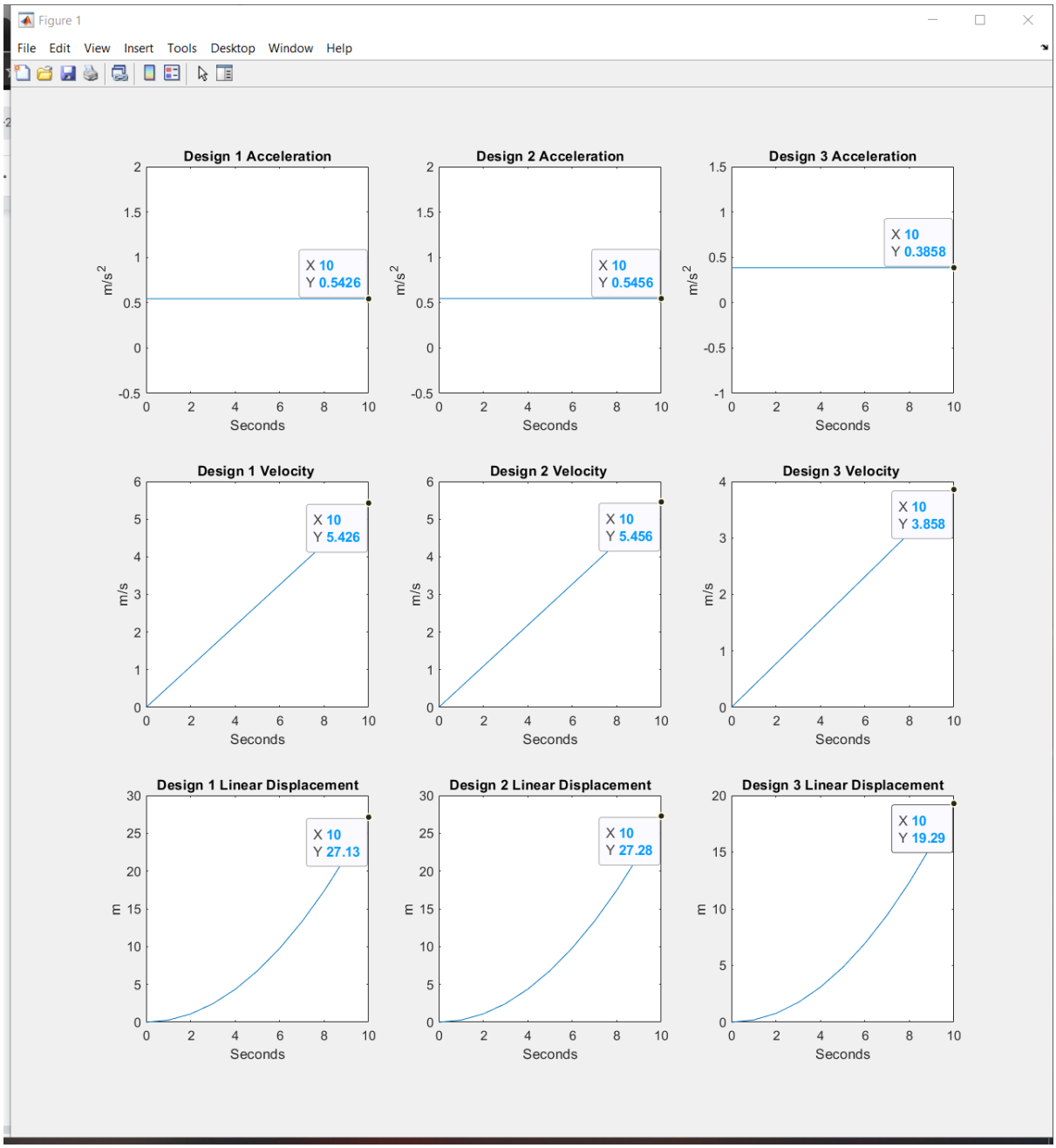


Fig 6.a MatLab Simulation of Three Designs (Linear)

Angular Acceleration: Angular Acceleration = distance between center of mass and fan * force * sqrt(2)/2

Angular Velocity: Angular Velocity = angular acceleration .* time

Angular Linear Displacement: Angular Linear Displacement = average angular velocity .* t
= (angular velocity/2).*t

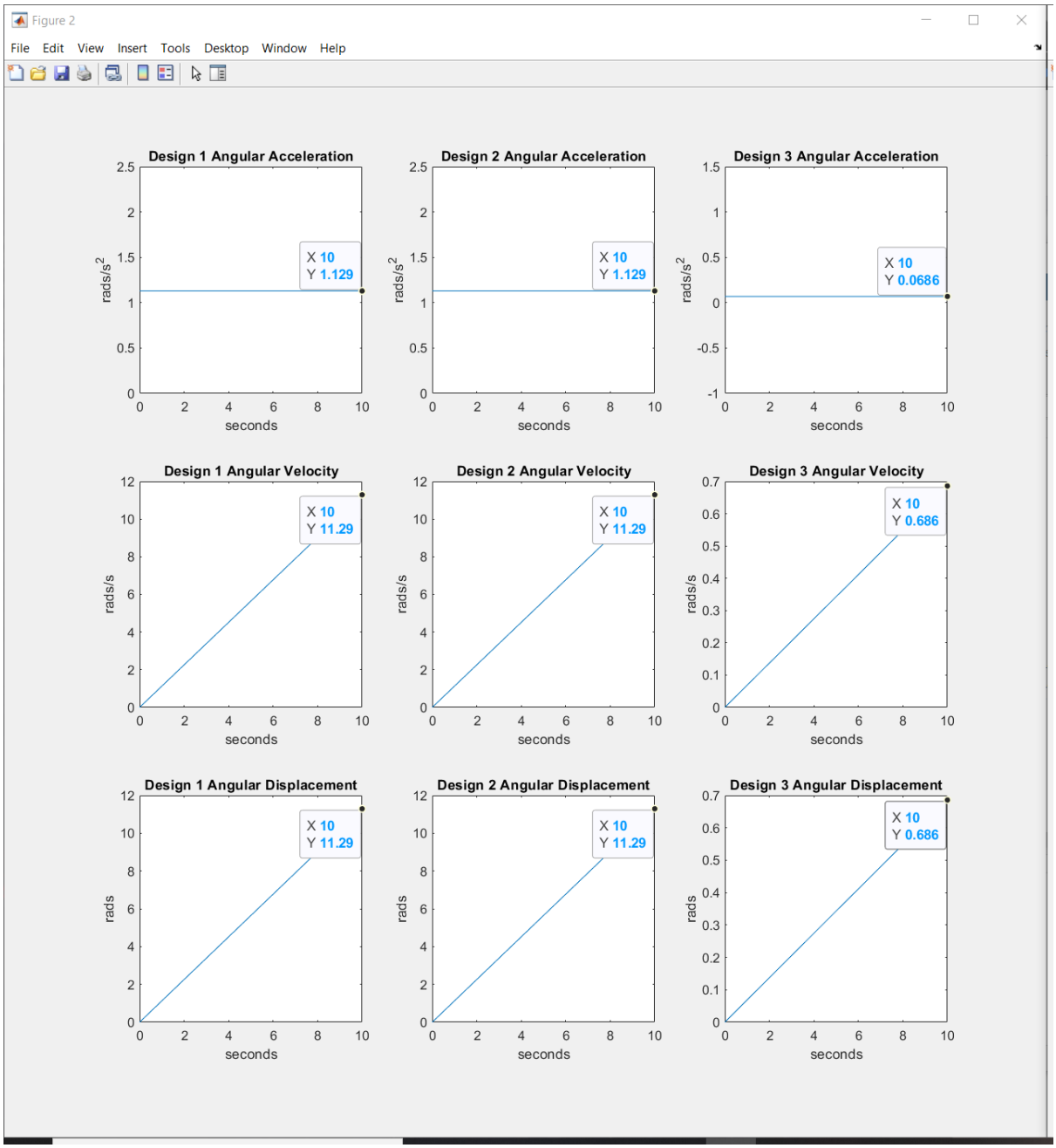


Fig 6.b MatLab Simulation of Three Designs (Angular)

4.4 Analytic Hierarchy Process

An Analytic Hierarchy Process (AHP) analysis was made comparing all of the hovercraft designs with the estimated weights of each of the criteria. The weights of each criteria were calculated using the comparative method specified in the AHP process, the results of which can be observed in Tables 6.a, 6.b and 6.c. After the Priority Vector (Table 6.b) was computed, the Consistency Ration (CR) was later obtained. By confirming that the CR (Table 7.b) was under the acceptable limit (10%), the rest of the comparisons were carried out accordingly.

After all the values obtained during the mass (Tables 7.a, 7.b and 7.c), speed (Tables 8.a, 8.b and 8.c), lift (Tables 9.a, 9.b and 9.c) and angular acceleration (Tables 10.a, 10.b and 10.c) cross comparisons are computed, Design 2 ended up being the one favored by the analysis (Table 11).

4.4.1 Criteria Weights

Criteria Pair Comparison Matrix					
Criteria	Mass	Speed, Linear	Angular Acceleration	Number of Components	Lift
Mass	1.00	0.25	0.13	0.17	0.50
Speed, Linear Acceleration	4.00	1.00	0.25	0.50	2.00
Angular Acceleration	8.00	4.00	1.00	2.00	6.00
Number of Components	6.00	2.00	0.50	1.00	4.00
Lift	2.00	0.50	0.13	0.17	1.00
Total	21.00	7.75	2.00	3.83	13.50

Table 5.a Criteria Pair Comparison Matrix

Criteria Pair Comparison Matrix (Normalized)					Priority Vector
0.04761904762	0.03225806452	0.0625	0.04347826087	0.03703703704	0.04457848201
0.1904761905	0.1290322581	0.125	0.1304347826	0.1481481481	0.1446182759
0.380952381	0.5161290323	0.5	0.5217391304	0.4444444444	0.4726529976
0.2857142857	0.2580645161	0.25	0.2608695652	0.2962962963	0.2701889327
0.09523809524	0.06451612903	0.0625	0.04347826087	0.07407407407	0.06796131184
1	1	1	1	1	1

Table 5.b. Criteria Pair Comparison Matrix (Normalized)

Criteria	Weights
Mass	0.045
Speed, Linear Acceleration	0.145
Angular Acceleration	0.473
Number of Components	0.270
Lift	0.068
Total	1.000

Table 5.c Criteria Weights

4.4.2 Consistency Ratio

Criteria	Ax	X
Mass	0.2188268204	4.90879928
Speed, Linear Acceleration	0.7121125433	4.924084035
Angular Acceleration	2.355899694	4.984417121
Number of Components	1.335068123	4.941239115
Lift	0.3335405273	4.907800015

Table 5.d Criteria Ax and X

Lambda	CI	RI	CR
4.933267913	-0.01668302172	1.12	-0.01489555511

Table 5.e Criteria Lambda, CI, RI and CR

4.4.3 Mass Comparison

Mass Comparison Matrix			
Mass	Design 1	Design 2	Design 3
Design 1	1.00	0.75	0.25
Design 2	1.33	1.00	0.33
Design 3	4.00	3.00	1.00
Total	6.33	4.75	1.58

Table 6.a Mass Pair Comparison Matrix

Mass Comparison Matrix (Normalized)			
Mass	Design 1	Design 2	Design 3
Design 1	0.1578947368	0.1578947368	0.1578947368
Design 2	0.2105263158	0.2105263158	0.2105263158
Design 3	0.6315789474	0.6315789474	0.6315789474

Table 6.b Mass Pair Comparison Matrix (Normalized)

Design	Priority vector
Design 1	0.1578947368
Design 2	0.2105263158
Design 3	0.6315789474

Table 6.c Mass Priority Vector

4.4.4 Speed, Linear Acceleration Comparison

Speed Comparison Matrix			
Mass	Design 1	Design 2	Design 3
Design 1	1.00	0.50	3.00
Design 2	2.00	1.00	4.00
Design 3	0.33	0.25	1.00
Total	3.33	1.75	8.00

Table 7.a Speed Pair Comparison Matrix

Speed Comparison Matrix (Normalized)			
Mass	Design 1	Design 2	Design 3
Design 1	0.3	0.28571428573	0.375
Design 2	0.6	0.5714285714	0.5
Design 3	0.1	0.1428571429	0.125

Table 7.b Speed Pair Comparison Matrix (Normalized)

Design	Priority vector
Design 1	0.3202380952
Design 2	0.5571428571
Design 3	0.1226190476

Table 7.c Speed Priority Vector

4.4.5 Lift Comparison

Lift Comparison Matrix			
Mass	Design 1	Design 2	Design 3
Design 1	1.00	0.67	2.00
Design 2	1.50	1.00	3.00
Design 3	0.50	0.33	1.00
Total	3.00	2.00	6.00

Table 8.a Lift Pair Comparison Matrix

Lift Comparison Matrix (Normalized)			
Mass	Design 1	Design 2	Design 3
Design 1	0.3333333333	0.3333333333	0.3333333333
Design 2	0.5	0.5	0.5
Design 3	0.1666666667	0.1666666667	0.1666666667

Table 8.b Lift Pair Comparison Matrix (Normalized)

Design	Priority vector
Design 1	0.3333333333
Design 2	0.5
Design 3	0.1666666667

Table 8.c Lift Priority Vector

4.4.6 Angular acceleration Comparison

Maneuverability Comparison Matrix			
Mass	Design 1	Design 2	Design 3
Design 1	1.00	0.33	0.50
Design 2	3.00	1.00	1.50
Design 3	2	0.67	1.00
Total	2.25	2.250	9.00

Table 9.a Angular Acceleration Pair Comparison Matrix

Angular Acceleration Comparison Matrix (Normalized)			
Mass	Design 1	Design 2	Design 3
Design 1	0.4444444444	0.4444444444	0.4444444444
Design 2	0.4444444444	0.4444444444	0.4444444444
Design 3	0.1111111111	0.11111111113	0.1111111111

Table 9.b Angular Acceleration Pair Comparison Matrix (Normalized)

Design	Priority vector
Design 1	0.4444444444
Design 2	0.4444444444
Design 3	0.1111111111

Table 9.c Angular Acceleration Priority Vector

4.4.7 Final Design Comparison

Design	Mass	Speed, Linear Acceleration	Angular Acceleration	Parts	Lift	Total
Design 1	0.15789474	0.3202380952	0.4444444444	0.363636364	0.333333333	0.3843232794
Design 2	0.210526316	0.5571428571	0.4444444444	0.363636364	0.5	0.4322571588
Design 3	0.631578947	0.1226190476	0.1111111111	0.272727273	0.166666667	0.1834195618

Table 10. Final Comparison Table

4.5 Sensor selection

During this project there were two sensor options given: a visual sensor and a proximity sensor. They both differ widely in what they perceive and provide data of. The proximity sensor, for one, can provide us with the distance in between itself and the closest object to it that lies within its range. In contrast the vision sensor creates an image based around the objects contained within its range, providing two-dimensional information about multiple entities with each reading. That being said, while both sensors perform different tasks and are meant for different applications, they can be both used for the same purpose in the context of our project. Given that the sensor's function, at least when it comes to how our control algorithm works, is only to inform the controller when a wall has been reached, a comparison can be formed based on both sensor's strength and weaknesses.

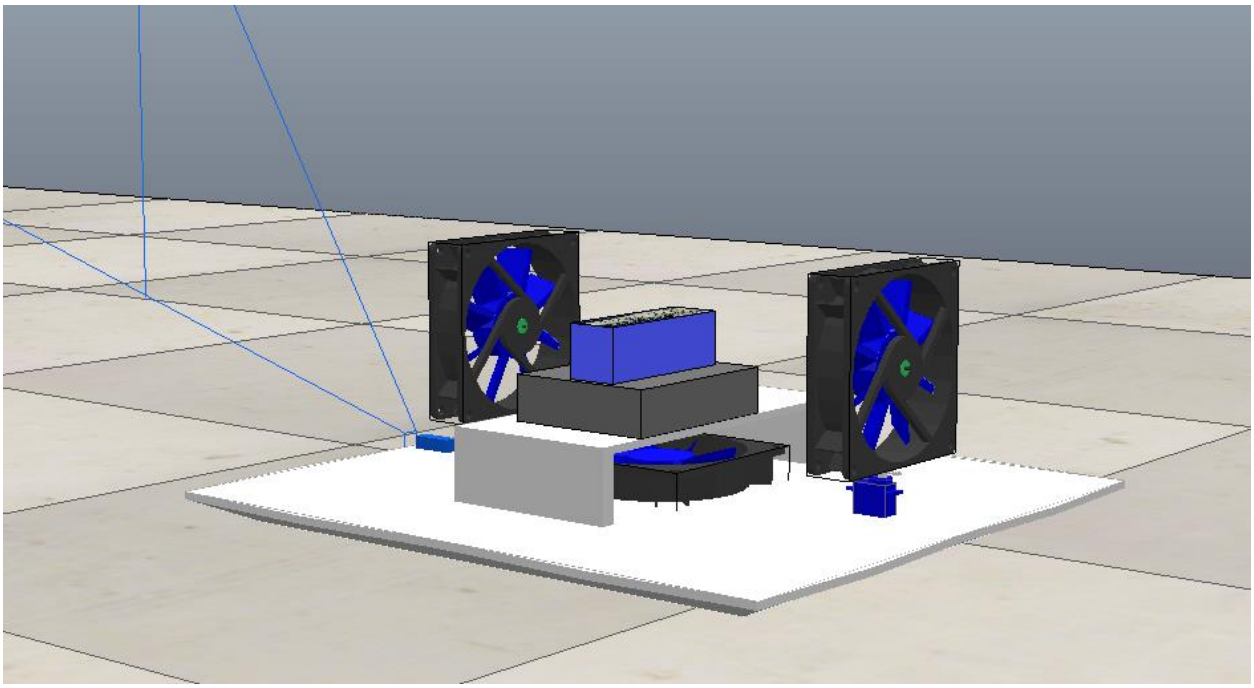


Fig 7.a Vision sensor implementation (Side view)

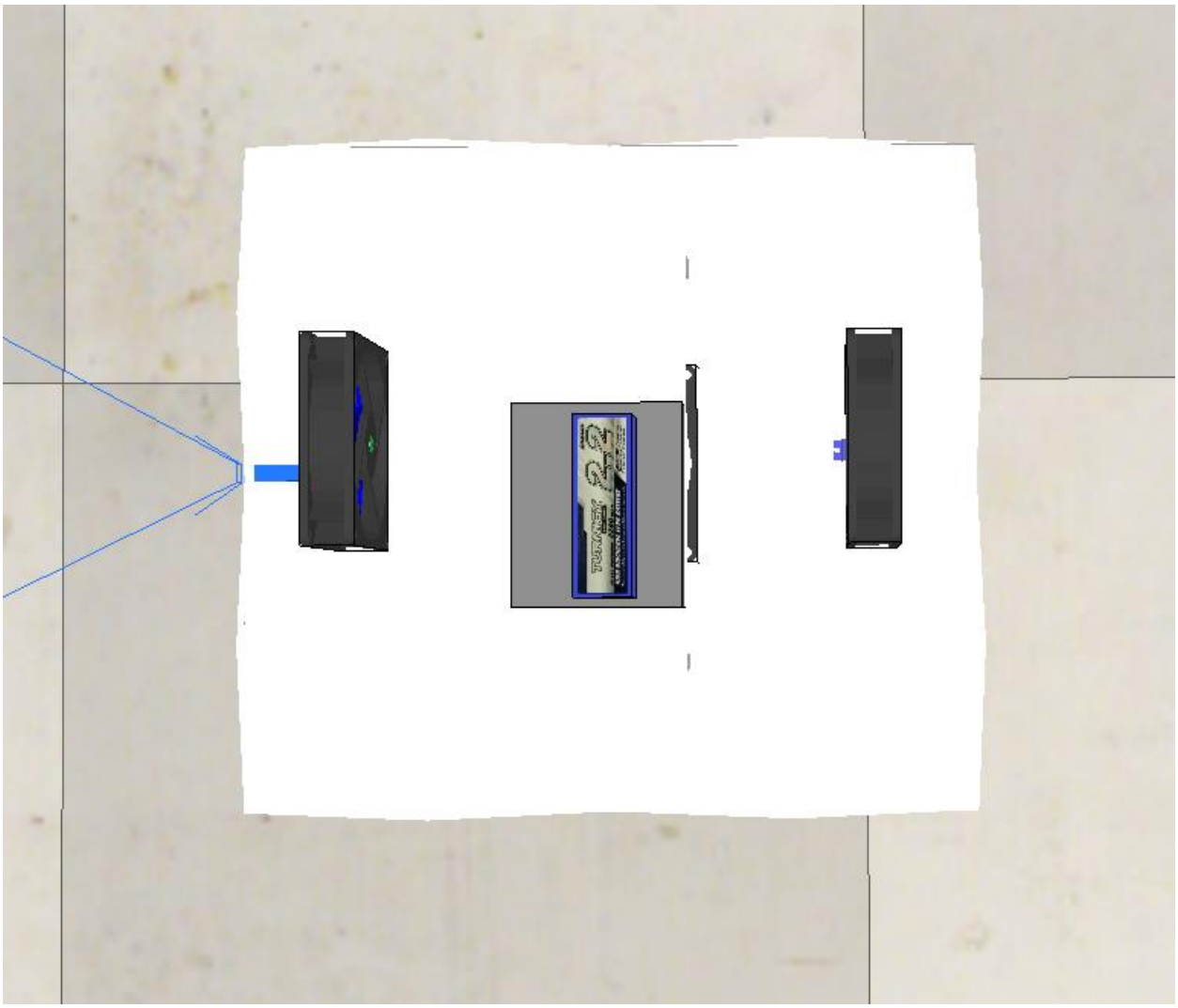


Fig 7.b Vision sensor implementation (top view)

4.5.1 Reliability

While both sensors perceive different aspects of the world, a comparison can be made in the consistency and accuracy in which both perform their respective tasks. The proximity sensor, although it does a much simpler job due to it providing a one-dimensional reading, it's results can be easily affected by vibration or deviations from the track. Furthermore, while testing the sensor, it was noted that certain "spikes" (sudden increments in the values perceived by it) can occur over the course of the simulation, making it harder to rely on its data.

On the other hand, the vision sensor provides accurate information about the locations of objects located in its field of view, and while not impossible, fewer, if any, inconsistencies on its data have been found while testing it on an active simulation.

4.5.2 Complexity

Since the vision sensor can only provide binary data about objects in an scene (that is, whether or not a pixel in the image is being occupied by an object or not), interpreting the data coming from it and using it in a practical way is a non-trivial problem to solve. For instance, if the distance from the sensor to an object is to be calculated, one would have to relate the location of the pixels in the two-dimensional array to a magnitude in the real world, a process that can be somewhat taxing to compute. It is also worth noting that the amount of data received from the vision sensor is much bigger than the proximity sensor, as each bit on the screen would have to be represented by a number and the amount of pixels needed to form an image increase dramatically with higher resolutions.

The proximity sensor, on the other hand, only returns the distance from itself, to the most immediate object in front of it, making it trivial to utilize on a control system's algorithm.

4.5.3 Conclusion

Despite its higher complexity and the larger data set needed to analyze its output, the vision sensor's precision and consistency proved invaluable at making our hovercraft perform as expected given the specified constraints, making it the ideal choice for this project.

The proximity sensors did a great job at determining how far the hovercraft was from the wall however can get fairly unreliable and inaccurate when the sensor was to determine distances that were close to it. There were multiple issues for having it follow along a wall at a certain distance since there would be unpredictable spikes. These spikes would tell the hovercraft that

it was either -100 cm away from the wall or jump a random number for the distance even if it was still 2-5 cm away. This issue could not be fixed either since it continually would give unpredictable results making it tough to distinguish between real values and spikes. In the end we chose the sensors placements to be for farther distances rather than close ones to help correct this issue. That is why we placed one in front and one in back of the hovercraft instead of one in front and one on the side.

5.0 Design Result

5.1 Revisited Calculations

Length(cm)	Width(cm)	Base(cm)	Height(cm)	Volume(cm^3)	Density(g/cm^3)	Mass(g)
45	41	1845	0.5	922.5	0.05	46.125

Table 11.a Final Size Table

Parts	Mass (g)	Distance from Center (cm)	X Positin (cm)	Y Positin (cm)	Moment of Interia (g*cm^2)	Moment of Interia (kg*m^2)
Lift Fan	198	0	0	0	0	0.00000
Thrust Fan 1	198	11.25	-11.25	0	25059.375	0.00251
Thrust Fan 2	198	11.25	11.25	0	25059.375	0.00251
Battery	258	0	0	0	0	0.00000
Arduino Controller	150	0	0	0	0	0.00000
Sensor 1	0	22.5	0	-22.5	0	0.00000
Sensor 2	0	22.5	0	22.5	0	0.00000
Small Servo Motor	9	11.25	-11.25	0	1139.0625	0.00011
Small Servo Motor	9	11.25	11.25	0	1139.0625	0.00011

Table 11.b Final Components Table

The total moment of inertia is 0.00524;

$I_{\text{Lift Fan}} = m_{\text{Lift Fan}} \times r^2_{\text{Lift Fan}} = 198g \times (0cm)^2 = 0\ kg*m^2$

$I_{\text{Thrust Fan}} = m_{\text{Thrust Fan}} \times r^2_{\text{Thrust Fan}} = 198g \times (11.25cm)^2 = 0.002\ kg*m^2$

$I_{\text{Thrust Fan}} = m_{\text{Thrust Fan}} \times r^2_{\text{Thrust Fan}} = 198g \times (11.25cm)^2 = 0.002\ kg*m^2$

$I_{\text{battery}} = m_{\text{battery}} \times r^2_{\text{battery}} = 258g \times (0cm)^2 = 0\ kg*m^2$

$I_{\text{Controller}} = m_{\text{Controller}} \times r^2_{\text{Controller}} = 150g \times (0cm)^2 = 0\ kg*m^2$

$I_{\text{Prox Sensor 1}} = m_{\text{Prox Sensor 1}} \times r^2_{\text{Ultrasonic Sensor 1}} = 0g \times (22.25cm)^2 = 0\ kg*m^2$

$I_{\text{Prox Sensor 2}} = m_{\text{Prox Sensor 2}} \times r^2_{\text{Ultrasonic Sensor 2}} = 0g \times (22.25cm)^2 = 0\ kg*m^2$

$I_{\text{Servo motor}} = m_{\text{Servo motor}} \times r^2_{\text{Servo motor}} = 9.00g \times (11.25cm)^2 = 0.00011\ kg*m^2$

$I_{\text{Servo motor}} = m_{\text{Servo motor}} \times r^2_{\text{Servo motor}} = 9.00g \times (11.25cm)^2 = 0.00011\ kg*m^2$

6.0 Schedule

This section of the report will cover all the scheduling related to the project.

Date of Meeting	Location	Time Allocated (h)	Members
Jan 23, 2021	Discord	2	all
Jan 30, 2021	Discord	1.5	all
Feb 6, 2021	Discord	3	all
Feb 13, 2021	Discord	3	all
Feb 20, 2021	Discord	2.5	all
Feb 27, 2021	Discord	2	all
March 1, 2021	Discord	5	all
March 2, 2021	Discord	4.5	all
March 3, 2021	Discord	4	all
March 5, 2021	Discord	2.5	all
March 6, 2021	Discord	2	all
March 13, 2021	Discord	2	all

March 27, 2021	Discord	1	all
April 3	Discord	2	all
April 7	Discord	1	all
April 9	Discord	1	all
April 10	Discord	1	all
April 14	Discord	2	all
April 16	Discord	4	all
April 17	Discord	5	all
April 18	Discord	3	all
April 19	Discord	4	all
April 20	Discord	3	all
April 21	Discord	2	all

Table 12. Meeting and Events

The above table shows the details for all the meetings that took place by the time the final project report was written. It should be noted that a fixed schedule of one meeting per week on Saturday was kept for all weeks except the “March Break” week (from March 1st to

March 6). This decision was made to allow all members to work individually on their assigned tasks, while also keeping the rest of the team up to date with their progress. After the PDR, we were assigned 5 tasks among us to shift to a CoppeliaSim environment. Since April 1st, we did not have lectures, we decided to meet and work during the lecture hours to utilize our process as well as to boost the project.

As for the overall project duration, a Gantt chart was prepared with the different steps planned for the project. Said chart is composed of eleven steps:

- Problem Description
- Research and Brainstorming
- Design Options Discussion
- Design Options Analysis
- Modeling, Prototyping, Analysis
- Simulation Environment Setup
- Simulation Programming
- Testing
- Fabrication
- Demonstration
- Report

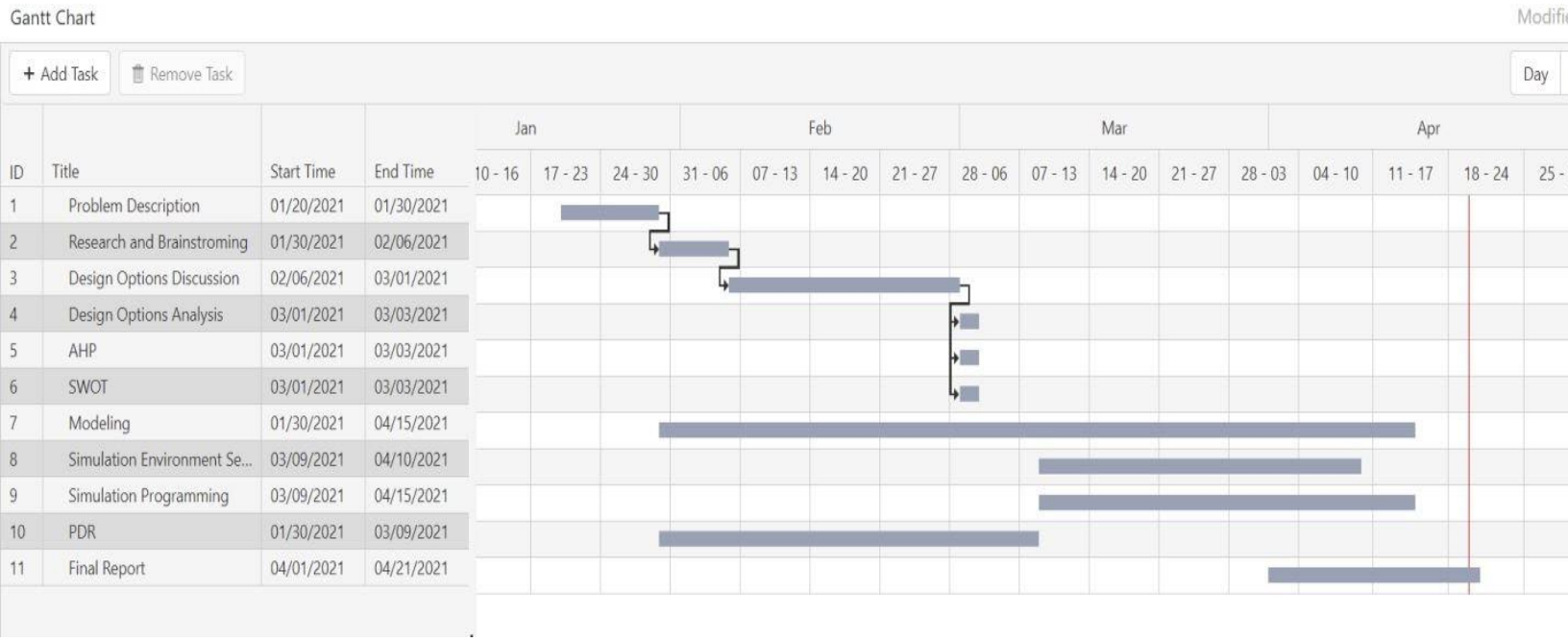


Fig 8. Scheduling Gantt Chart

7.0 Summary

This report has demonstrated the design process of a hovercraft. The objective was to create a fully autonomous hovercraft that can complete the predefined track within the time limit of 60 seconds. Another objective was to use the least amount of components for the design. Based on our simulation, all the objectives were met. By using a rectangular (45cm*41cm) body, the hovercraft was able to complete the track and go over the three obstacles. The rectangular body did not allow the craft to overturn to start going in the opposite direction. The autonomy algorithm made the hovercraft to always follow the wall on its right. When the hovercraft detects an obstacle on the right wall and the front wall, the craft will turn left otherwise it will move forward if there is an obstacle on the right and in front of it. This autonomous approach was implemented to achieve the least number of components which eventually secured a better score for our final design. The most important thing is that the hovercraft was able to finish the given track within 47 seconds. Overall, it was a great team effort that helped us to solve different challenges.

One of the most important outcomes from this project is the mechanical knowledge of a hovercraft such as kinematics and fluid dynamics. A successfully simulated project to get a small-scale experience in real life. However, the hardware and software limitations were affecting this project since it is not demonstrated physically. There are different results showing for each of the hardware and operating systems since the configuration of the each team members is different. Running more tests helped to optimize the hovercraft simulation in CoppeliaSim environment. At the end, it was interesting and helpful to get introduced to CoppeliaSim, Sketchup and Arduino Controller,

8.0 References

[1] Hovercraft reference idea: https://www.youtube.com/watch?v=mpqYlHTlXtA

[2] Density of Polystyrene: https://www.aqua-calc.com/page/density-table/substance/polystyrene-blank-foam

[3] Proximity Sensor: https://youtu.be/YIsWYySLVXQ

[4] Vision Sensor: https://youtu.be/E_ddDjjd5FM

9.0 Appendix

Matlab Code:

```
clc;
clear;
close all;
mass1 = 0.551635;
mass2 = 0.5486205;
mass3 = 0.387919;
t=[0:10];

f = 0.299335;
f3 = f/2;

a1 =
[f/mass1,f/mass1,f/mass1,f/mass1,f/mass1,f/mass1,f/mass1,f/mass1,f/mass1,f/mass1,f/mass1];
a2 =
[f/mass2,f/mass2,f/mass2,f/mass2,f/mass2,f/mass2,f/mass2,f/mass2,f/mass2,f/mass2,f/mass2];
a3 =
[f3/mass3,f3/mass3,f3/mass3,f3/mass3,f3/mass3,f3/mass3,f3/mass3,f3/mass3,f3/mass3,f3/mass3,
f3/mass3];

figure(1);
subplot(3,3,1);
plot(t,a1);
xlabel('Seconds');
ylabel('m/s^2');
title('Design 1 Acceleration')

subplot(3,3,2);
plot(t,a2);
xlabel('Seconds');
ylabel('m/s^2');
title('Design 2 Acceleration')

subplot(3,3,3);
plot(t,a3);
xlabel('Seconds');
ylabel('m/s^2');
title('Design 3 Acceleration')

v1 = a1.*t;
v2 = a2.*t;
v3 = a3.*t;
```

```
subplot(3,3,4);
plot(t,v1);
xlabel('Seconds');
ylabel('m/s');
title('Design 1 Velocity ');


subplot(3,3,5);
plot(t,v2);
xlabel('Seconds');
ylabel('m/s');
title('Design 2 Velocity ');

subplot(3,3,6);
plot(t,v3);
xlabel('Seconds');
ylabel('m/s');
title('Design 3 Velocity ');

d1=(v1/2).*t;
d2=(v2/2).*t;
d3=(v3/2).*t;

subplot(3,3,7);
plot(t,d1);
xlabel('Seconds');
ylabel('m');
title('Design 1 Linear Displacement');

subplot(3,3,8);
plot(t,d2);
xlabel('Seconds');
ylabel('m');
title('Design 2 Linear Displacement');

subplot(3,3,9);
plot(t,d3);
xlabel('Seconds');
ylabel('m');
title('Design 3 Linear Displacement');

%t for torque
t1 = 7.25*f*sqrt(2)/2;
t2 = 8.36*f;
t3 = 2.5*f3*sqrt(2)/2;
```

```
%l for alpha
l1 =
[t1/1.359,t1/1.359,t1/1.359,t1/1.359,t1/1.359,t1/1.359,t1/1.359,t1/1.359,t1/1.359,t1/1.359,t1/1.35
9];
l2 =
[t2/2.216,t2/2.216,t2/2.216,t2/2.216,t2/2.216,t2/2.216,t2/2.216,t2/2.216,t2/2.216,t2/2.216,t2/2.21
6];
l3 =
[t3/3.857,t3/3.857,t3/3.857,t3/3.857,t3/3.857,t3/3.857,t3/3.857,t3/3.857,t3/3.857,t3/3.857,t3/3.85
7];

figure(2);
subplot(3,3,1);
plot(t,l1);
xlabel('seconds');
ylabel('rads/s^2');
title('Design 1 Angular Acceleration');

subplot(3,3,2);
plot(t,l2);
xlabel('seconds');
ylabel('rads/s^2');
title('Design 2 Angular Acceleration');

subplot(3,3,3);
plot(t,l3);
xlabel('seconds');
ylabel('rads/s^2');
title('Design 3 Angular Acceleration');


av1 = l1.*t;
av2 = l2.*t;
av3 = l3.*t;


subplot(3,3,4);
plot(t,av1);
xlabel('seconds');
ylabel('rads/s');
title('Design 1 Angular Velocity');

subplot(3,3,5);
plot(t,av2);
xlabel('seconds');
ylabel('rads/s');
```

```
title('Design 2 Angular Velocity');

subplot(3,3,6);
plot(t,av3);
xlabel('seconds');
ylabel('rads/s');
title('Design 3 Angular Velocity');


ad1 = (av1/2).*t;
ad2 = (av2/2).*t;
ad3 = (av3/2).*t;

subplot(3,3,7);
plot(t,av1);
xlabel('seconds');
ylabel('rads');
title('Design 1 Angular Displacement');

subplot(3,3,8);
plot(t,av2);
xlabel('seconds');
ylabel('rads');
title('Design 2 Angular Displacement');

subplot(3,3,9);
plot(t,av3);
xlabel('seconds');
ylabel('rads');
title('Design 3 Angular Displacement');
```

Code for the arduino controller

//controller.ino

```
#include <stdint.h>

#include "utils.h"
#include "debug.h"
#include "serialdata.h"
#include "hovercraft.h"
#include <string.h>
#include <stdio.h>

// ------------------ OPTIONS -------------------------
#define BAUD_RATE 115200

// Comment if you'd like to disable camera support (will ignore proximity sensors)
#define ENABLE_VISION

// Uncomment if you'd like to disable the sensors (used for debugging)
//#define DISABLE_SENSORS

// ------------------ CONSTANTS -------------------------
#define THRUST_FAN_SPEED 100.0
#define WALL_STOP_DELAY 750

#ifdef DISABLE_SENSORS
#define OPTIMAL_SIDE_DISTANCE 2.0
#define SWITCH_LANE_DELAY 3500
#else
#define OPTIMAL_SIDE_DISTANCE 10.0
#define SWITCH_LANE_DELAY 4350
#endif

#define LEFT true
#define RIGHT false
#define UP_ANGLE 0.0
#define RIGHT_ANGLE -90.0
#define LEFT_ANGLE 90.0

#define ON true
#define OFF false

#define LANES_COUNT 4
#define LANE_1_DURATION 4000
#define LANE_2_DURATION 4500
#define LANE_3_DURATION 4200
#define LANE_4_DURATION 4800
```

```
#ifdef ENABLE_VISION

#define SERIAL_DELAY 90 //~2*8*1000000/BAUDRATE
#define IMG_SENSOR_YDIM 32
uint32_t img[IMG_SENSOR_YDIM];

#endif

// ------------------ VARIABLES ------------------------------
bool current_direction = LEFT;
float sensor_reading = 0.0;
float turn_angle = 0.0;
int lane_count = 0;
SerialData serialData;
Hovercraft hovercraft(&serialData);

// --------------------------  SETUP FUNCTION -------------------------
void setup()
{
  Serial.begin(BAUD_RATE);
  current_direction = LEFT;

#if ! defined(DISABLE_SENSORS) && ! defined(ENABLE_VISION)

  // Do this to:
  // - Get the initial time for the simulation
  // - Make the Arduino wait for the simulator
  receive_update();

  float ls = hovercraft.getLeftSensorDistance();
  if(ls < hovercraft.getRightSensorDistance())
    current_direction = LEFT;
  else
    current_direction = RIGHT;

#endif

#ifdef ENABLE_VISION

  Serial.setTimeout(1);

#endif

  switch(current_direction)
  {
    case LEFT:
      set_servos(LEFT_ANGLE);
      break;
    case RIGHT:
      set_servos(RIGHT_ANGLE);
```

```
      break;
  }

  //turn on fans
  set_all_fans_state(ON);
  set_thrust_fans_thrust(THRUST_FAN_SPEED);

  send_update();
}

// --------------------------  LANE SWITCH  --------------------------
void switch_lanes()
{
  // Move up to the next lane
  set_servos(UP_ANGLE);
  send_update();

#ifdef ENABLE_VISION

  while(checkDistance() > 0)
  {
    serialWait();
    Serial.println(" ");
  }

#else

  delay(SWITCH_LANE_DELAY);

#endif

  set_servos(turn_angle);
  current_direction = !current_direction;
  send_update();

  lane_count++;
}

// --------------------------  MAIN LOOP  --------------------------
void loop()
{

#ifdef ENABLE_VISION

  // Wait for an update
  serialWait();

#else

  // Wait for an update
```

```
  receive_update();

#endif

#ifndef ENABLE_VISION
#ifndef DISABLE_SENSORS

  // --------------------------  WITH SENSORS  --------------------------
  // Depending on the direction, read the sensor and decide the turn angle
  switch(current_direction)
  {
    case LEFT:
      sensor_reading = hovercraft.getLeftSensorDistance();
      turn_angle = RIGHT_ANGLE;
      break;
    case RIGHT:
      sensor_reading = hovercraft.getRightSensorDistance();
      turn_angle = LEFT_ANGLE;
      break;
  }

  // If you are OPTIMAL_SIDE_DISTANCE cm away from a side wall...
  if(sensor_reading > OPTIMAL_SIDE_DISTANCE and not (sensor_reading < 0))
  {
    delay(WALL_STOP_DELAY);
    switch_lanes();
  }

#else

  // --------------------------  WITHOUT SENSORS  --------------------------
  // Depending on the direction, read the sensor and decide the turn angle
  switch(current_direction)
  {
    case LEFT:
      turn_angle = RIGHT_ANGLE;
      break;
    case RIGHT:
      turn_angle = LEFT_ANGLE;
      break;
  }
  unsigned int timings[LANES_COUNT] = {
    LANE_1_DURATION,
    LANE_2_DURATION,
    LANE_3_DURATION,
    LANE_4_DURATION
  };

  // If we are not on the last track, move in the current one direction
  // for timings[lane_count]
```

```
  delay(timings[lane_count]);
  switch_lanes();

#endif
#else

  switch(current_direction)
  {
    case LEFT:
      turn_angle = RIGHT_ANGLE;
      break;
    case RIGHT:
      turn_angle = LEFT_ANGLE;
      break;
  }

  int dist = checkDistance();
  if(dist > 1 && dist < 10)
    switch_lanes();

#endif

  // If the last lane has beem reached, stop
  if(lane_count >= LANES_COUNT)
  {
    set_all_fans_state(OFF);
    send_update();
    while(1)
      delay(1000);
  }

  // Sending an update should guarantee that we'll
  // receive one back
  send_update();
}

//---------------------- FAN HANDLERS ----------------------

// Set thrust fans state
static inline void set_thrust_fans_state(bool s)
{
  hovercraft.right_fan_state = s;
  hovercraft.left_fan_state = s;
}

// Set thrust fans thrust
static inline void set_thrust_fans_thrust(float t)
{
  hovercraft.right_fan_thrust = t;
  hovercraft.left_fan_thrust = t;
```

```
}

// Set lift fans state
static inline void set_lift_fan_state(bool s)
{
  hovercraft.lift_fan_state = s;
}

// Set all fans state
static inline void set_all_fans_state(bool s)
{
  set_thrust_fans_state(s);
  set_lift_fan_state(s);
}

// ----------------------  SERVO HELPERS ------------------------------

// Set all servos state
static inline void set_servos(float a)
{
  hovercraft.right_servo = a;
  hovercraft.left_servo = a;
}

// ----------------------  DATA HANDLERS ---------------------------------

// Sends current hovercraft settings to the simulator
static inline void send_update()
{
  hovercraft.sendUpdate();
}

// Receives updates from the simulator (will block until
// updates are received)
static inline void receive_update()
{
  hovercraft.receiveUpdate();
}


// ---------------------- CAMERA HANDLERS ---------------------------------
#ifdef ENABLE_VISION
void serialWait()
{
  float simTime = Serial.parseFloat();
  int   result = Serial.parseInt();
  // digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  while(Serial.peek() != 's')
  {
    if (Serial.peek() == 'i')
```

```
  { //Read image data coming on the Serial bus
    char  t = Serial.read(); //discard the i
    delayMicroseconds(6*SERIAL_DELAY);
    for (int j = IMG_SENSOR_YDIM-1; j >= 0; j--)
    {
      char longarr[8];
      for  (int i = 0; i < 8; i++)
      {
        longarr[i] = Serial.read();
        if (Serial.available()< 4)
        {
          delayMicroseconds(SERIAL_DELAY);
        }
      }
      unsigned long hexval =  strtoul((const char*)longarr, 0, 16);
      img[j] = hexval;
    }
    if (Serial.peek() == '@')
    {
      char t = Serial.read(); //discard the @
      //Print out the image, to check if it was received properly
      Serial.write(0x23);Serial.write(0x23);Serial.write("\r\n");
      Serial.write("printing image...");
      Serial.write("\r\n");
      for  (int b = 0; b < IMG_SENSOR_YDIM; b++)
      {
        for (int c = 0; c < 32; c++) //for (int c = 31; c >= 0; c--)
        {
          Serial.print(bitRead(img[b],c));
          Serial.write(" ");
        }
        Serial.write("\r\n");
      }
      Serial.write(0x40);Serial.write(0x40);Serial.write("\r\n");
    }
  }
  if (Serial.peek() == 's')
  {
    //char t = Serial.read(); //discard the s
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
    return;// break out of the while loop and continue to the parsing steps.
  }
  if (Serial.available() > 2)
  {
    char t = Serial.read();
  }
 }
 // digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
}

void serialEmpty()
```

```
{
  while(Serial.available())
  {
    char t = Serial.read();
  }
}

short readImage(short imgX, short imgY)
{
  return bitRead(img[imgY],imgX);
}

// Made so it will start scanning from line
// two to ignore line at the bottom
int checkDistance()
{
  for (int i = 2; i < 29; i++)
  {
    if (readImage(14,i) == 1)
    {
      return i;
    }
  }
  return -1;
}
#endif
```

//debug.h

```
#pragma once

#include <Arduino.h>

//Debug macros, use when you can but pls no touchy
#define DEBUG_LOG(msg)        Serial.print(msg)

#define DEBUG_FILE_INFO()     DEBUG_LOG( __FILE__ ":")
#define DEBUG_LINE_INFO()     DEBUG_LOG( __LINE__)

#define DEBUG_VAR(var) \
                        DEBUG_FILE_INFO();                  \
                        DEBUG_LINE_INFO();                  \
                        DEBUG_LOG(": " #var " = " );        \
                        DEBUG_LOG(var);                   \
                        DEBUG_LOG(ENDL);
```

//hovercraft.cpp

```
#include "hovercraft.h"
```

```
Hovercraft::Hovercraft(SerialData* serialdata): serialdata(serialdata) {}

void Hovercraft::sendUpdate()
{
  if(this->serialdata == NULL)
    return;

  // Lift Fan
  this->internalBuffer[0] = this->lift_fan_state;

  // Right Fan
  this->internalBuffer[1] = this->right_fan_state;
  this->internalBuffer[2] = this->right_fan_thrust;
  this->internalBuffer[3] = this->right_servo;

  // Left Fan
  this->internalBuffer[4] = this->left_fan_state;
  this->internalBuffer[5] = this->left_fan_thrust;
  this->internalBuffer[6] = this->left_servo;

  this->serialdata->sendData(this->internalBuffer, 7);
}

void Hovercraft::receiveUpdate()
{
  if(this->serialdata == NULL)
    return;

  this->serialdata->parseData();

  if(this->serialdata->lenght() < this->input_count)
    return;

  this->sim_time      = this->serialdata->at(0);
  this->left_sensor   = this->serialdata->at(1);
  this->right_sensor  = this->serialdata->at(2);
}
```

//hovercraft.h

```
#pragma once

#include <Arduino.h>
#include "serialdata.h"
#include "utils.h"

class Hovercraft
{
private:
  const unsigned int input_count = 3;
```

```
public:
  Hovercraft(SerialData* serialdata);

  void sendUpdate();
  void receiveUpdate();

  float getTime()
    { return this->sim_time; }

  float getRightSensorDistance()
    { return this->right_sensor; }
  float getLeftSensorDistance()
    { return this->left_sensor; }

public:
  bool lift_fan_state     = false;
  bool right_fan_state    = false;
  bool left_fan_state     = false;

  float right_fan_thrust  = 0.0;
  float right_servo       = 0.0;

  float left_fan_thrust   = 0.0;
  float left_servo        = 0.0;

private:
  SerialData* serialdata;
  SERIAL_T internalBuffer[8];

  float sim_time          = 0.0;
  float right_sensor      = 0.0;
  float left_sensor       = 0.0;
};
```

//serialdata.cpp

```
#include "serialdata.h"

char SerialData::serial_buffer[MAX_SERIAL_BUFFER_SIZE] = {0};

SerialData::SerialData(): len(0) {}

void SerialData::sendData(SERIAL_T * data, size_t len)
{
  while(!Serial)
    delay(1);

  for(size_t i = 0; i < len; i++)
  {
    Serial.print(data[i]);
    Serial.write(PACKET_SEND_DEL);
```

```
  }
  Serial.write("\n\r");
}

size_t SerialData::readIntoBuffer()
{
  while(!Serial)
    delay(1);

  size_t bytes_read = 0;
  char c = 'a';

  // Wait for s to arrive. 's' indicates the
  // start of a packet
  while(c != PACKET_START)
    c = Serial.read();
  serial_buffer[bytes_read++] = c;

  // Read until either buffer is filled up or packet is over
  while(c != '\r' && c != '\n' && bytes_read < MAX_SERIAL_BUFFER_SIZE)
  {
    while(Serial.available() > 0)
      c = serial_buffer[bytes_read++] = Serial.read();
  }

#ifdef DEBUG
  DEBUG_VAR(bytes_read)
#endif

  return bytes_read;
}

void SerialData::parseData(char * data, size_t lenght)
{
  size_t i = 0;
  size_t len = 1;
  size_t start = 0;
#ifdef DEBUG
  DEBUG_VAR(lenght)
#endif

  // Read the amount of numbers received
  for(; i < lenght; i++)
  {
    if(data[i] != PACKET_START && data[i] != PACKET_RECV_DEL)
    {
      start = i;
      break;
    }
  }
```

```
  for(; i < lenght && len < 10; i++)
  {
    if(data[i] == PACKET_RECV_DEL) len++;
    else if (data[i] == '\n' || data[i] == '\r') break;
  }
#ifdef DEBUG
  DEBUG_VAR(len)
#endif

  // For every number found, store them into
  // this->data
  this->len = len;
  size_t index = 0;
  float buf = 0.0;
  float dec_count = 1.0;
  bool is_decimal = false;
  bool is_negative = false;
  for(i = start; i < lenght && index < this->len; i++)
  {
    int c = data[i];
#ifdef DEBUG
        DEBUG_VAR(char(c))
#endif
    switch(c)
    {
      //For negative numbers
      case '-':
        is_negative = true;
        break;

      //For decimal numbers
      case '.':
        is_decimal = true;
        break;

      //For the numbers
      case '0':
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
      case '8':
      case '9':
        c -= 48;
        if(is_decimal)
          buf += ((float)c)/(pow(10.0,dec_count++));
        else
          buf = buf * 10.0 + (float)c;
```

```
#ifdef DEBUG
        DEBUG_VAR(buf)
#endif
        break;

      //For the endings
      case ' ':
      case '\n':
      case '\r':


#ifdef DEBUG
        this->data[index] = (SERIAL_T)(buf*(is_negative ? -1.0 : 1.0));
        DEBUG_VAR(this->data[index])
        index++;
#else
        this->data[index++] = (SERIAL_T)(buf*(is_negative ? -1.0 : 1.0));
#endif

        buf = 0.0;
        dec_count = 1.0;
        is_decimal = false;
        is_negative = false;
        break;
      default:
        continue;
    }
  }
}

void SerialData::parseData()
{
  size_t s = this->readIntoBuffer();
  char* b = this->getBuffer();
  this->parseData(b,s);
}
```

//serialdata.h

```
#pragma once

//#define DEBUG

#include <Arduino.h>
#include <string.h>
#include <stdint.h>

#include "debug.h"

#define SERIAL_T float
```

```
#define MAX_SERIAL_BUFFER_SIZE 100

#define PACKET_START 's'

#define PACKET_SEND_DEL ','
#define PACKET_RECV_DEL ' '

class SerialData
{
private:
  static char serial_buffer[MAX_SERIAL_BUFFER_SIZE];                // Buffer for allocated data

public:
  //Static functions for data transfer
  void sendData(SERIAL_T * data, size_t len);                       // Sends data in the ###, ###, ...
\n\r format
  void parseData(char * data, size_t len);                          // Takes raw serial data from buffer
and parses it with the ###, ###, ... \n\r. Allocates results in data
  static size_t readIntoBuffer();                                   // Receives data from the serial buffer and
stores it into serial_buffer. Returns the amount of data recevied
  static char* getBuffer() {return serial_buffer;}                  // Returns pointer to serial_buffer
  void parseData();                                                 // Receive data from serial port, saves into
the buffer, and parses it

public:
  //Object memebers
  SerialData();                                                     // Constructor.

  SERIAL_T at(const size_t &i) {return this->data[i];}              // Gets data member at i index
  size_t lenght() {return this->len;}                               // Gets the lenght of data

private:
  SERIAL_T data[10];                                                // Stores parsed data
  size_t len = 0;                                                   // Stores data size
};
```

//util.h

```
#ifndef UTILS_H_
#define UTILS_H_

#define CAP_AT_NUM(val, num)  (val < 0.0 ? num : val)
#define MAP_AT_CAP(REF, VAL, CAP)  (REF * ((VAL > CAP ? CAP : VAL)/CAP))
#define ABS(x,y) (x < y ? y-x :x-y)

#endif
```

Lua script for the hovercraft

```
local USE_ARDUINO          = true
```

```
local SERIAL_PORT_NAME  = "COM3"
local SERIAL_BAUDRATE       = 115200
local SENSOR_PRESCISION     = 0.001

local CONTACT_PATCH_NAME    = "hv"
local LIFT_FAN_NAME         = "liftFan"
local RIGTH_FAN_NAME        = "ThrustFan#1"
local RIGHT_SERVO_NAME      = "SmallServo#1"
local RIGHT_SENSOR_NAME     = "MB1030#0"
local LEFT_FAN_NAME         = "ThrustFan1"
local LEFT_SERVO_NAME       = "SmallServo"
local LEFT_SENSOR_NAME      = "MB1030#1"

local DEBUG = false

function coroutineMain()
    print(_VERSION)

    -- simulator settings
    sim.setThreadSwitchTiming(10)
    sim.setIntegerSignal('Consoles',0)
    sim.setIntegerSignal('lift',0)

    keyboard_ctrl = 0
    init = true
    serial = nil

    --Open serial port communication
    print(SERIAL_PORT_NAME)
    serial = sim.serialOpen(SERIAL_PORT_NAME,SERIAL_BAUDRATE)

    -- Print serial info
    print(serial)
    if serial == -1 then
        error("Serial port error. Check port name/path and USB connection to Arduino board")
    end

    contact_patch_handle        = sim.getObjectHandle(CONTACT_PATCH_NAME)
    lift_fan_handle             = sim.getObjectHandle(LIFT_FAN_NAME)
    rigth_servo_handle          = sim.getObjectHandle(RIGHT_SERVO_NAME)
    rigth_fan_handle            = sim.getObjectHandle(RIGTH_FAN_NAME)
    rigth_sensor_handle         = sim.getObjectHandle(RIGHT_SENSOR_NAME)
    left_servo_handle           = sim.getObjectHandle(LEFT_SERVO_NAME)
    left_fan_handle             = sim.getObjectHandle(LEFT_FAN_NAME)
    left_sensor_handle          = sim.getObjectHandle(LEFT_SENSOR_NAME)

    -- initialize sensors
    imu_init()

    local arduinoData = {}
```

```
arduinoData.lift_fan_state      = 0
arduinoData.rigth_fan_state     = 0
arduinoData.rigth_fan_thrust    = 0
arduinoData.rigth_servo         = 0
arduinoData.left_fan_state      = 0
arduinoData.left_fan_thrust     = 0
arduinoData.left_servo          = 0

rxData = nil

-- Main simulation loop
while sim.getSimulationState() ~= sim.simulation_advancing_abouttostop do

    -- Get sensor data in centimeters
    detected, rigth_sensor_reading = read_sensor(rigth_sensor_handle)
    rigth_sensor_reading = 100 * rigth_sensor_reading

    detected, left_sensor_reading = read_sensor(left_sensor_handle)
    left_sensor_reading = 100 * left_sensor_reading

    if DEBUG then
        print("rigth_sensor_reading: " .. rigth_sensor_reading)
        print("left_sensor_reading: " .. left_sensor_reading)
    end

    -- Get data from arduino
    if serial ~= -1 and init == false then
        rxData = receive_data()
        print(rxData)
    end

    -- If data is valid, react accordingly

    -- Arduino Receive Data Format
    -- rxData[1] lift_fan_state
    -- rxData[2] rigth_fan_state
    -- rxData[3] rigth_fan_thrust
    -- rxData[4] rigth_servo
    -- rxData[5] left_fan_state
    -- rxData[6] left_fan_thrust
    -- rxData[7] left_servo

    if rxData and table.getn(rxData)>=7 then

        -- Get arduino data into arduinoData
        -- Lift Fan
        arduinoData.lift_fan_state = rxData[1]

        -- Right fan
        arduinoData.rigth_fan_state = rxData[2]
```

```
            arduinoData.rigth_fan_thrust = rxData[3]
            arduinoData.rigth_servo = angle_to_radiants(rxData[4])

            -- Left fan
            arduinoData.left_fan_state = rxData[5]
            arduinoData.left_fan_thrust = rxData[6]
            arduinoData.left_servo = angle_to_radiants(rxData[7])

            -- Lift Fan Update
            set_lift(lift_fan_handle, arduinoData.lift_fan_state, contact_patch_handle)

            -- Right Fan Update
            set_thrust(rigth_fan_handle, arduinoData.rigth_fan_state,
arduinoData.rigth_fan_thrust/100)
            set_servo(rigth_servo_handle, arduinoData.rigth_servo)

            -- Left Fan Update
            set_thrust(left_fan_handle, arduinoData.left_fan_state, arduinoData.left_fan_thrust/100)
            set_servo(left_servo_handle, arduinoData.left_servo)
        end

        -- Update Arduino with sensor data
        -- Arduino Send Data Format
        -- txData[1] simTime
        -- txData[2] left_sensor_reading
        -- txData[3] rigth_sensor_reading
        if serial ~= -1 then
            sim_time = round(sim.getSimulationTime(), SENSOR_PRESCISION)
            rigth_sensor_reading = round(rigth_sensor_reading, SENSOR_PRESCISION)
            left_sensor_reading = round(left_sensor_reading, SENSOR_PRESCISION)
            send_data(serial, {sim_time, left_sensor_reading, rigth_sensor_reading})
            init = false
        end
        sim.switchThread() -- resume in next simulation step
    end

end

function angle_to_radiants(angle)
    return angle*3.14*2/180
end

function imu_init()
-- This function will initialize the IMU communication
-- Run only once in the initialization section
-- Usage
    -- imu_init()
    gyroCommunicationTube=sim.tubeOpen(0,'gyroData'..sim.getNameSuffix(nil),1)
    accelSelfCommunicationTube=sim.tubeOpen(0,'accelerometerSelfData'..sim.getNameSuffix(
nil),1)
```

```
end

function read_imu()
-- This function will read the IMU
-- it will return instantaneous  values of angular speed (rad/s) and linear acceleration (m/s^2)
-- Returned values are 2 tables with 3 numbers each
-- {XangVel, YangVel, ZangVel}, {Xaccel, Yaccel, Zaccel}
-- Usage
    -- angVel, linAccel = read_imu()
    local angularSpeeds = {0,0,0}
    local data1=sim.tubeRead(gyroCommunicationTube)
    if (data1) then
        angularSpeeds=sim.unpackFloatTable(data1)
        angularSpeeds = round(angularSpeeds, sensorPrecision)
    end

    local accelSelf = {0,0,0}
    local data2=sim.tubeRead(accelSelfCommunicationTube)
    if (data2) then
        accelSelf =sim.unpackFloatTable(data2)
        accelSelf = round(accelSelf, sensorPrecision)
    end

    return angularSpeeds, accelSelf
end

function read_sensor(sensorHandle)
-- This function will read one proximity sensor using its Object Handle
-- sensorHandle: Input argument  must be the sensor model base's Object Handle
-- Output arguments are result & distance
-- If no object is sensed by the sensor,
    -- result = 0, distance = -1
-- If an object is sensed by the sensor,
    -- result = 1, distance = distance (m) between sensor and object (rounded to mm)
-- Usage
    -- result,distance = read_sensor(sensorHandle)

    result,distance = sim.readProximitySensor(sensorHandle)
    if distance == nil then
        distance = -1
    else
        distance = round(distance,0.001)
    end

    return result,distance
end

function send_data(localSerial, localdata)
-- This function sends data to the arduino board
-- nil values will be replaced, as much as possible, with -1
```

```
-- it may take the following arguments
    -- single number (single proximity sensors)
    -- array of numbers (multiple proximity sensors)
    -- array of arrays (proximity sensor(s) + IMU)
    -- numbers will be sent to the serial port as a string in this format
    -- "s #### #### #### ..."

    if localdata and table.getn(localdata)>0 then
        local data = unroll(localdata)
        local data_str = ""
        local ind = table.getn(data)
        for i = 1,ind do
            data_str = data_str.." "..data[i]
        end

        if localSerial ~= -1 then
            print("Sending: "..s ".." "..data_str)
            data_str = "s ".." "..data_str..'\n'
            charsSent = sim.serialSend(localSerial, data_str)
        else
            print("Serial port is not available")
        end
    end
end

function unroll(localdata, priorData)
-- This recursive function creates a single layer table from multi dimensional tables
-- it may take the following arguments
    -- data
        --any form of table of table of table (etc) as data
    -- priorData
        --is optional, is assumed to be a single layer table
    result = priorData or {}
    if type(localdata) == "number" then
        result = {localdata}
    elseif type(localdata) == "table" then
        ind = table.getn(localdata)
        for i = 1,ind do
            if type(localdata[i]) == "number" then
                table.insert(result,localdata[i])
            elseif type(localdata[i]) == "table" then
                unroll(localdata[i], result)
            elseif type(localdata[i]) == "nil" then
                table.insert(result,-1)
                print("Attempting to send nil, sending -1 instead")
            else
                table.insert(result,-1)
                print("unknown data type in subtable, sending -1 instead")
            end
        end
        --print("results")
```

```
        --print(result)
    else
        print('No new data to unroll?')
    end

    return result
end

function receive_data()
-- This function receives data from the arduino board
-- The data sent from the arduino MUST BE a comma-separated string
-- terminated by a carriage & line return "\r\n"
-- "###,###,###,###,...\r\n"
-- Those numbers are then returned by this function as a table
    -- {###,###,###,###,...}

    if serial ~= -1 then
        str = sim.serialRead(serial,1000,true,'\n',2)
    else
        return nil
    end
    if str ~= nil then
        local token
        ctrl_val = {}
        cpt=0
        for token in string.gmatch(str, "[^,]+") do
            if type(tonumber(token))=='number' then
                --print(token)
                cpt = cpt+1
                ctrl_val[cpt] = tonumber(token)
            end
        end
        --if ctrl_val == nil or cpt ~= arduino_arg_number then
        if ctrl_val == nil then
            print('unexpected data length, check arduino_arg_number var')
            return nil
        end
    else
        return nil
    end
    --print(ctrl_val)
    return ctrl_val
end

function set_servo(servoHandle,posCmd)
-- This function will actuate the RC Servo to reach the commanded position
-- Only commanded position in the -pi/2 to pi/2 range are allowed
-- servoHandle: Input argument must be the servo model base's Object Handle
-- posCmd: angle in radian from -pi/2 to pi/2
-- Function usage
```

```
    -- set_servo(servoHandle,servoPosition)

    if (sim.getObjectType(servoHandle)==sim.object_forcesensor_type) then
        temp = sim.getObjectChild(servoHandle,0)
        child1 = sim.getObjectChild(temp,0)
        if sim.getObjectType(child1)==sim.object_joint_type then
            servoHandle = child1
        else
            child2 = sim.getObjectChild(temp,1)
            if sim.getObjectType(child2)==sim.object_joint_type then
                servoHandle = child2
            end
        end
    end

    if posCmd > 3.1416/2 then
        print('Commanded servo position out of range')
        posCmd = 3.141/2
    elseif posCmd < -3.1416/2 then
        print('Commanded servo position out of range')
        posCmd = -3.141/2
    end
    sim.setJointTargetPosition(servoHandle,posCmd)
end

function set_thrust(fanHandle,state,localThrottle)
-- This function will control the thrust fan with three arguments
-- fanHandle: must be the handle of the component model base
-- state: Activity state of the fan
    -- state = 0    (fan is OFF)
    -- state = 1    (fan is ON)
-- Throttle: should be a number between 0 and 1.
    -- Throttle = 0     (will turn off the fan)
    -- Throttle = 0.5   (will give partial thrust from the fan)
    -- Throttle = 1     (will give the maximum thrust from the fan)
-- Function usage
    -- set_thrust(fanHandle, state, throttle)

    if (sim.getObjectType(fanHandle)==sim.object_forcesensor_type) then
        fanHandle = sim.getObjectChild(fanHandle,0)
    end


    if localThrottle > 1 then
        print('Throttle out of range')
        localThrottle = 1
    elseif localThrottle < 0 then
        print('Throttle out of range')
        localThrottle = 0
        state = 0
    elseif localThrottle == 0 then
```

```
        state = 0
    elseif state == 0 then
        localThrottle = 0
    else
        state = 1
    end

    sim.setUserParameter(fanHandle,'lift',0)
    sim.setUserParameter(fanHandle,'state',state)
    sim.setUserParameter(fanHandle,'throttle',localThrottle)
    --add lift = 0
end


function set_lift(liftFan, liftState, localContactPatch)
-- This function will activate and deactivate the lift simulation
-- This is done by setting the appropriate physical properties on the main hovercraft body
-- liftState: Activity state of the fan
    -- liftState = 0     (lift OFF)
    -- liftState = 1     (lift ON)
-- liftfan: must be the handle of the fan providing lift component model base
-- localContactPatch: must be the handle of the contact patch of your hovercraft
-- friction: returned value for the lift friction coefficient
-- Function usage
    -- friction = set_lift(liftFan, liftState, localContactPatch)

    if (sim.getObjectType(liftFan)==sim.object_forcesensor_type) then
        liftFan = sim.getObjectChild(liftFan,0)
    end

    sim.setUserParameter(liftFan,'throttle',1)
    sim.setUserParameter(liftFan,'lift',1)

    if liftState == 1 and sim.getIntegerSignal('lift') == 0 then

        print(  "Lift is activated")
        XYSize = getBoundingBoxXYSize(localContactPatch)
        meanRadius = (XYSize[1] + XYSize[2]) / 2 / 2
        area = XYSize[1] * XYSize[2]
        COMlocation, COMDelta, mass = getCenterOfMass(localContactPatch)
        --print("COMlocation X: "..COMlocation[1].."; COMlocation Y: ".. COMlocation[2]..";
COMlocation Z: ".. COMlocation[3].."; area: "..area.."; meanRadius: "..meanRadius)
        body_pressure = mass * 9.8 / area
        fan_pressure = sim.getUserParameter(liftFan,"fanPressure")

        if body_pressure >= fan_pressure then

            friction = 0.1
        elseif body_pressure >= fan_pressure/1.25 then
```

```
            friction = -0.07/0.25 * (fan_pressure/body_pressure)  + 0.38

        elseif body_pressure >= fan_pressure/2.5 then
            friction = -0.02/1.25 * (fan_pressure/body_pressure)  + 0.05
        elseif body_pressure < fan_pressure/2.5 then
            friction = 0.01
        end

        print(  "body pressure: "..round(body_pressure,0.01).." Pa; "..
                "fan pressure: "..round(fan_pressure,0.01).." Pa; ")

        if COMDelta <= 0.01*meanRadius then
            --Friction coef is untouched
        elseif COMDelta <= 0.5*meanRadius then
            friction = friction + (0.1-friction)/0.49 * (COMDelta/meanRadius-0.01)
        elseif COMDelta > 0.5*meanRadius then
            friction = 0.1


        end

        print(  "Distance between CoM and centroid: ".. 1000*round(COMDelta,0.00001).." mm; "..
                "meanRadius: "..1000*round(meanRadius,0.00001).." mm; "..
                "Contact patch friction: "..round(friction,0.0001))

        sim.setUserParameter(liftFan,'state',1)
        sim.setIntegerSignal('lift',1)

        sim.setEngineFloatParameter(sim.newton_body_staticfriction,localContactPatch,friction)
        sim.setEngineFloatParameter(sim.newton_body_kineticfriction,localContactPatch,friction)
        sim.resetDynamicObject(localContactPatch)

    elseif liftState == 0 and sim.getIntegerSignal('lift') == 1 then
        sim.setUserParameter(liftFan,'state',0)
        sim.setIntegerSignal('lift',0)

        sim.setEngineFloatParameter(sim.newton_body_staticfriction,localContactPatch,0.2)
        sim.setEngineFloatParameter(sim.newton_body_kineticfriction,localContactPatch,0.09)
        sim.resetDynamicObject(localContactPatch)
    end
    return round(friction,0.0001)
end

function getBoundingBoxXYSize(obj)
    a, size1min = sim.getObjectFloatParameter(obj, 15)
    a, size2min = sim.getObjectFloatParameter(obj, 16)
    a, size3min = sim.getObjectFloatParameter(obj, 17)
    a, size1max = sim.getObjectFloatParameter(obj, 18)
    a, size2max = sim.getObjectFloatParameter(obj, 19)
    a, size3max = sim.getObjectFloatParameter(obj, 20)
```

```
    size1 = size1max-size1min
    size2 = size2max-size2min
    size3 = size3max-size3min
    sizes = {size1, size2, size3}
    minSize = math.min(size1,size2,size3)
    for i = 1,3 do
        if sizes[i] == minSize then
            min = i
        end
    end
    table.remove(sizes,min)
    return sizes
end

function getCenterOfMass(modelBase)
-- reference: https://forum.coppeliarobotics.com/viewtopic.php?t=1719
-- Function returns the CoM for a given model in this format
-- {{CoMX, CoMY, CoMZ},{deltaX, deltaY, deltaZ},totalMass}
-- First find all non-static shapes in our model:

    allNonStaticShapes={}
    allObjectsToExplore={modelBase}
    while (#allObjectsToExplore>0) do
        obj=allObjectsToExplore[1]
        table.remove(allObjectsToExplore,1)
        if (sim.getObjectType(obj)==sim.object_shape_type) then
            --print("object# "..obj)
            r,v=sim.getObjectInt32Parameter(obj,3003)
            if (v==0) then -- is the shape non-static?
                table.insert(allNonStaticShapes,obj)
            end
        end
        index=0
        while true do
            child=sim.getObjectChild(obj,index)
            if (child==-1) then
                break
            end
            table.insert(allObjectsToExplore,child)
            index=index+1
        end
    end

    -- Now compute the center of mass of our model (in absolute coordinates):

    mass,inertia,base_com=sim.getShapeMassAndInertia(modelBase,nil)
    miri={0,0,0}
    totalMass=0
    loc_com = {}
    for i=1,#allNonStaticShapes,1 do
```

```
        --print(sim.getObjectName(allNonStaticShapes[i]))
        mass,inertia,com=sim.getShapeMassAndInertia(allNonStaticShapes[i],nil)
        miri[1]=miri[1]+mass*com[1]
        miri[2]=miri[2]+mass*com[2]
        miri[3]=miri[3]+mass*com[3]
        totalMass=totalMass+mass
    end
    final_com = {}
    final_com[1]=miri[1]/totalMass
    final_com[2]=miri[2]/totalMass
    final_com[3]=miri[3]/totalMass

    delta = math.sqrt((final_com[1]-base_com[1])^2+(final_com[2]-base_com[2])^2)

    return final_com,delta,totalMass
end


function round(exact, quantum)
-- Rounding function
-- https://stackoverflow.com/questions/18313171/lua-rounding-numbers-and-then-truncate
    if type(exact) == "number" then
        local quant,frac = math.modf(exact/quantum)
        return quantum * (quant + (frac > 0.5 and 1 or 0))
    elseif type(exact) == "table" then
        out = {}
        for i = 1,table.getn(exact) do
            out[i] = round(exact[i], quantum)
        end
        return out
    else
        print("Unexpected type sent to round() function")
    end
end




function sysCall_cleanup()
    -- Put some clean-up code here
    if USE_ARDUINO then
        if serial ~= -1 then
            sim.serialClose(serial)
        else
            print('Your serial port was not correctly opened at simulation start')
        end
    end
end
-- See the user manual or the available code snippets for additional callback functions and
details
```

Lua script for vision implementation

```
local USE_ARDUINO = true
local SERIAL_PORT_NAME  = "COM3"
local SERIAL_BAUDRATE   = 115200
local SENSOR_PRESCISION = 0.001

local CONTACT_PATCH_NAME    = "hv"
local LIFT_FAN_NAME         = "liftFan"
local BOTTOM_FAN_NAME       = "ThrustFan#1"
local BOTTOM_SERVO_NAME     = "SmallServo#1"
local TOP_FAN_NAME          = "ThrustFan1"
local TOP_SERVO_NAME        = "SmallServo"
local CAMERA_SENSOR_NAME    = "VisionSensor"

local DEBUG = true

function coroutineMain()
    print(_VERSION)

    -- simulator settings
    sim.setThreadSwitchTiming(10)
    sim.setIntegerSignal('Consoles',0)
    sim.setIntegerSignal('lift',0)
    sim.clearStringSignal('intToRemAPI')
    sim.clearStringSignal('floatToRemAPI')
    sim.clearStringSignal('fromRemAPI')

    keyboard_ctrl = 0
    init = true
    serial = nil

    --Open serial port communication
    print(SERIAL_PORT_NAME)
    serial = sim.serialOpen(SERIAL_PORT_NAME,SERIAL_BAUDRATE)

    -- Print serial info
    print(serial)
    if serial == -1 then
        error("Serial port error. Check port name/path and USB connection to Arduino board")
    end

    contact_patch_handle        = sim.getObjectHandle(CONTACT_PATCH_NAME)
    lift_fan_handle             = sim.getObjectHandle(LIFT_FAN_NAME)
    bottom_servo_handle         = sim.getObjectHandle(BOTTOM_SERVO_NAME)
    bottom_fan_handle           = sim.getObjectHandle(BOTTOM_FAN_NAME)
    top_servo_handle            = sim.getObjectHandle(TOP_SERVO_NAME)
    top_fan_handle              = sim.getObjectHandle(TOP_FAN_NAME)
    vision_sensor_handle        = sim.getObjectHandle(CAMERA_SENSOR_NAME)
```

```
-- initialize sensors
imu_init()

local arduinoData               = {}
arduinoData.lift_fan_state      = 0
arduinoData.bottom_fan_state    = 0
arduinoData.bottom_fan_thrust   = 0
arduinoData.bottom_servo        = 0
arduinoData.top_fan_state       = 0
arduinoData.top_fan_thrust      = 0
arduinoData.top_servo           = 0

-- Main simulation loop
while sim.getSimulationState() ~= sim.simulation_advancing_abouttostop do
    -- Read camera image
    imgstr = read_vision(vision_sensor_handle)
    -- print(image)

    -- Get data from arduino
    rxData = nil
    if serial ~= -1 and init == false then
        rxData = receive_data()
        -- print(rxData)
    end

    -- If data is valid, react accordingly

    -- Arduino Receive Data Format
    -- rxData[1] lift_fan_state
    -- rxData[2] bottom_fan_state
    -- rxData[3] bottom_fan_thrust
    -- rxData[4] bottom_servo
    -- rxData[5] top_fan_state
    -- rxData[6] top_fan_thrust
    -- rxData[7] top_servo

    if rxData and table.getn(rxData)>=7 then

        -- Get arduino data into arduinoData
        -- Lift Fan
        arduinoData.lift_fan_state = rxData[1]

        -- Right fan
        arduinoData.bottom_fan_state = rxData[2]
        arduinoData.bottom_fan_thrust = rxData[3]
        arduinoData.bottom_servo = angle_to_radiants(rxData[4])

        -- Left fan
        arduinoData.top_fan_state = rxData[5]
```

```
            arduinoData.top_fan_thrust = rxData[6]
            arduinoData.top_servo = angle_to_radiants(rxData[7])

            -- Lift Fan Update
            set_lift(lift_fan_handle, arduinoData.lift_fan_state, contact_patch_handle)

            -- Right Fan Update
            set_thrust(bottom_fan_handle, arduinoData.bottom_fan_state,
arduinoData.bottom_fan_thrust/100)
            set_servo(bottom_servo_handle, arduinoData.bottom_servo)

            -- Left Fan Update
            set_thrust(top_fan_handle, arduinoData.top_fan_state, arduinoData.top_fan_thrust/100)
            set_servo(top_servo_handle, arduinoData.top_servo)
        end

        -- Update Arduino with sensor data
        -- Arduino Send Data Format
        -- txData[1] simTime
        -- txData[2] top_sensor_reading
        -- txData[3] bottom_sensor_reading
        if serial ~= -1 then
            sim_time = round(sim.getSimulationTime(), SENSOR_PRESCISION)
            send_data(serial, {sim_time,0.0},imgstr)
            init = false
        end
        sim.switchThread() -- resume in next simulation step
    end

end

function read_vision(vis)
    local img = sim.getVisionSensorImage(vis)
    local res = sim.getVisionSensorResolution(vis)

    out = ""
    for i = 1, 3*res[1]*res[2],3 do
       out = out..(img[i])
    end
    for i = 1, res[1] do
        for j = 3*res[2]-2, 1 ,-3 do
            k = (i-1)*3*res[2] + j
            --print("pos: "..i.."", "..j.." = "..k)
            out = out..(img[k])
        end
    end

    --print(out)
    send = out:bittohex()
    --print(out:bittohex())
```

```
    return send
end

function string.bittohex(str)
--Inspiration: Michal Kottman https://stackoverflow.com/questions/9137415/lua-writing-
hexadecimal-values-as-a-binary-file
    return (str:gsub('....', function (c)
        return string.format('%01X', tonumber(c, 2))
    end))
end

function angle_to_radiants(angle)
    return angle*3.14*2/180
end

function imu_init()
-- This function will initialize the IMU communication
-- Run only once in the initialization section
-- Usage
    -- imu_init()
    gyroCommunicationTube=sim.tubeOpen(0,'gyroData'..sim.getNameSuffix(nil),1)
    accelSelfCommunicationTube=sim.tubeOpen(0,'accelerometerSelfData'..sim.getNameSuffix(
nil),1)
end

function read_imu()
-- This function will read the IMU
-- it will return instantaneous  values of angular speed (rad/s) and linear acceleration (m/s^2)
-- Returned values are 2 tables with 3 numbers each
-- {XangVel, YangVel, ZangVel}, {Xaccel, Yaccel, Zaccel}
-- Usage
    -- angVel, linAccel = read_imu()
    local angularSpeeds = {0,0,0}
    local data1=sim.tubeRead(gyroCommunicationTube)
    if (data1) then
        angularSpeeds=sim.unpackFloatTable(data1)
        angularSpeeds = round(angularSpeeds, sensorPrecision)
    end

    local accelSelf = {0,0,0}
    local data2=sim.tubeRead(accelSelfCommunicationTube)
    if (data2) then
        accelSelf =sim.unpackFloatTable(data2)
        accelSelf = round(accelSelf, sensorPrecision)
    end

    return angularSpeeds, accelSelf
end

function read_sensor(sensorHandle)
```

```
-- This function will read one proximity sensor using its Object Handle
-- sensorHandle: Input argument  must be the sensor model base's Object Handle
-- Output arguments are result & distance
-- If no object is sensed by the sensor,
   -- result = 0, distance = -1
-- If an object is sensed by the sensor,
   -- result = 1, distance = distance (m) between sensor and object (rounded to mm)
-- Usage
   -- result,distance = read_sensor(sensorHandle)

   result,distance = sim.readProximitySensor(sensorHandle)
   if distance == nil then
      distance = -1
   else
      distance = round(distance,0.001)
   end

   return result,distance
end

function send_data(localSerial, localdata, localImgStr)
-- This function sends data to the arduino board
-- nil values will be replaced, as much as possible, with -1
-- it may take the following arguments
   -- single number (single proximity sensors)
   -- array of numbers (multiple proximity sensors)
   -- array of arrays (proximity sensor(s) + IMU)
   -- numbers will be sent to the serial port as a string in this format
   -- "s #### #### #### ..."

   if localdata and table.getn(localdata)>0 then
      local data = unroll(localdata)
      local data_str = ""
      local ind = table.getn(data)
      for i = 1,ind do
         data_str = data_str.." "..data[i]
      end

      if string.len(data_str) > 60 then
         print("Attempting to send more than 60 characters, your data may get lost!")
      end

      if localSerial ~= -1 then
         if localImgStr then
            -- print("Sending: ".."i"..localImgStr.."s "..data_str)
            data_str = "i"..localImgStr.."s "..data_str.."\n"
         else
            -- print("Sending: ".."s "..data_str)
            data_str = "s "..data_str.."\n"
         end
```

```
            charsSent = sim.serialSend(localSerial, data_str)
        else
            print("Serial port is not available")
        end
    end
end

-- function send_data(localSerial, localdata)
-- -- This function sends data to the arduino board
-- -- nil values will be replaced, as much as possible, with -1
-- -- it may take the following arguments
--     -- single number (single proximity sensors)
--     -- array of numbers (multiple proximity sensors)
--     -- array of arrays (proximity sensor(s) + IMU)
--     -- numbers will be sent to the serial port as a string in this format
--     -- "s #### #### #### ..."

--     if localdata and table.getn(localdata)>0 then
--         local data = unroll(localdata)
--         local data_str = ""
--         local ind = table.getn(data)
--         for i = 1,ind do
--             data_str = data_str.." "..data[i]
--         end

--         if localSerial ~= -1 then
--             print("Sending: "..s ".." "..data_str)
--             data_str = "s ".." "..data_str..'\n'
--             charsSent = sim.serialSend(localSerial, data_str)
--         else
--             print("Serial port is not available")
--         end
--     end
-- end

function unroll(localdata, priorData)
-- This recursive function creates a single layer table from multi dimensional tables
-- it may take the following arguments
    -- data
        --any form of table of table of table (etc) as data
    -- priorData
        --is optional, is assumed to be a single layer table
    result = priorData or {}
    if type(localdata) == "number" then
        result = {localdata}
    elseif type(localdata) == "table" then
        ind = table.getn(localdata)
        for i = 1,ind do
            if type(localdata[i]) == "number" then
                table.insert(result,localdata[i])
```

```
            elseif type(localdata[i]) == "table" then
                unroll(localdata[i], result)
            elseif type(localdata[i]) == "nil" then
                table.insert(result,-1)
                print("Attempting to send nil, sending -1 instead")
            else
                table.insert(result,-1)
                print("unknown data type in subtable, sending -1 instead")
            end
        end
        --print("results")
        --print(result)
    else
        print('No new data to unroll?')
    end

    return result
end

function receive_data()
-- This function receives data from the arduino board
-- The data sent from the arduino MUST BE a comma-separated string
-- terminated by a carriage & line return "\r\n"
-- "###,###,###,###,...\r\n"
-- Those numbers are then returned by this function as a table
    -- {###,###,###,###,...}

    if serial ~= -1 then
        str = sim.serialRead(serial,300,true,'\n',2)
    else
        return nil
    end
    if str ~= nil then
        -- print("received: " .. str)
        local token
        ctrl_val = {}
        cpt=0
        for token in string.gmatch(str, "[^,]+") do
            if type(tonumber(token))=='number' then
                --print(token)
                cpt = cpt+1
                ctrl_val[cpt] = tonumber(token)
            end
        end
        --if ctrl_val == nil or cpt ~= arduino_arg_number then
        if ctrl_val == nil then
            print('unexpected data length, check arduino_arg_number var')
            return nil
        end
    else
```

```
        return nil
    end
    --print(ctrl_val)
    return ctrl_val
end

function set_servo(servoHandle,posCmd)
-- This function will actuate the RC Servo to reach the commanded position
-- Only commanded position in the -pi/2 to pi/2 range are allowed
-- servoHandle: Input argument must be the servo model base's Object Handle
-- posCmd: angle in radian from -pi/2 to pi/2
-- Function usage
    -- set_servo(servoHandle,servoPosition)

    if (sim.getObjectType(servoHandle)==sim.object_forcesensor_type) then
        temp = sim.getObjectChild(servoHandle,0)
        child1 = sim.getObjectChild(temp,0)
        if sim.getObjectType(child1)==sim.object_joint_type then
            servoHandle = child1
        else
            child2 = sim.getObjectChild(temp,1)
            if sim.getObjectType(child2)==sim.object_joint_type then
                servoHandle = child2
            end
        end
    end

    if posCmd > 3.1416/2 then
        print('Commanded servo position out of range')
        posCmd = 3.141/2
    elseif posCmd < -3.1416/2 then
        print('Commanded servo position out of range')
        posCmd = -3.141/2
    end
    sim.setJointTargetPosition(servoHandle,posCmd)
end

function set_thrust(fanHandle,state,localThrottle)
-- This function will control the thrust fan with three arguments
-- fanHandle: must be the handle of the component model base
-- state: Activity state of the fan
    -- state = 0    (fan is OFF)
    -- state = 1    (fan is ON)
-- Throttle: should be a number between 0 and 1.
    -- Throttle = 0     (will turn off the fan)
    -- Throttle = 0.5   (will give partial thrust from the fan)
    -- Throttle = 1     (will give the maximum thrust from the fan)
-- Function usage
    -- set_thrust(fanHandle, state, throttle)

    if (sim.getObjectType(fanHandle)==sim.object_forcesensor_type) then
```

```
        fanHandle = sim.getObjectChild(fanHandle,0)
    end


    if localThrottle > 1 then
        print('Throttle out of range')
        localThrottle = 1
    elseif localThrottle < 0 then
        print('Throttle out of range')
        localThrottle = 0
        state = 0
    elseif localThrottle == 0 then
        state = 0
    elseif state == 0 then
        localThrottle = 0
    else
        state = 1
    end

    sim.setUserParameter(fanHandle,'lift',0)
    sim.setUserParameter(fanHandle,'state',state)
    sim.setUserParameter(fanHandle,'throttle',localThrottle)
    --add lift = 0
end


function set_lift(liftFan, liftState, localContactPatch)
-- This function will activate and deactivate the lift simulation
-- This is done by setting the appropriate physical properties on the main hovercraft body
-- liftState: Activity state of the fan
    -- liftState = 0     (lift OFF)
    -- liftState = 1     (lift ON)
-- liftfan: must be the handle of the fan providing lift component model base
-- localContactPatch: must be the handle of the contact patch of your hovercraft
-- friction: returned value for the lift friction coefficient
-- Function usage
    -- friction = set_lift(liftFan, liftState, localContactPatch)

    if (sim.getObjectType(liftFan)==sim.object_forcesensor_type) then
        liftFan = sim.getObjectChild(liftFan,0)
    end

    sim.setUserParameter(liftFan,'throttle',1)
    sim.setUserParameter(liftFan,'lift',1)

    if liftState == 1 and sim.getIntegerSignal('lift') == 0 then

        print(  "Lift is activated")
        XYSize = getBoundingBoxXYSize(localContactPatch)
        meanRadius = (XYSize[1] + XYSize[2]) / 2 / 2
        area = XYSize[1] * XYSize[2]
```

```
    COMlocation, COMDelta, mass = getCenterOfMass(localContactPatch)
    --print("COMlocation X: "..COMlocation[1].."; COMlocation Y: ".. COMlocation[2]..";
COMlocation Z: ".. COMlocation[3].."; area: "..area.."; meanRadius: "..meanRadius)
    body_pressure = mass * 9.8 / area
    fan_pressure = sim.getUserParameter(liftFan,"fanPressure")

    if body_pressure >= fan_pressure then

       friction = 0.1
    elseif body_pressure >= fan_pressure/1.25 then

       friction = -0.07/0.25 * (fan_pressure/body_pressure)  + 0.38

    elseif body_pressure >= fan_pressure/2.5 then
       friction = -0.02/1.25 * (fan_pressure/body_pressure)  + 0.05
    elseif body_pressure < fan_pressure/2.5 then
       friction = 0.01
    end

    print(  "body pressure: "..round(body_pressure,0.01).." Pa; "..
          "fan pressure: "..round(fan_pressure,0.01).." Pa; ")

    if COMDelta <= 0.01*meanRadius then
       --Friction coef is untouched
    elseif COMDelta <= 0.5*meanRadius then
       friction = friction + (0.1-friction)/0.49 * (COMDelta/meanRadius-0.01)
    elseif COMDelta > 0.5*meanRadius then
       friction = 0.1


    end

    print(  "Distance between CoM and centroid: ".. 1000*round(COMDelta,0.00001).." mm; "..
          "meanRadius: "..1000*round(meanRadius,0.00001).." mm; "..
          "Contact patch friction: "..round(friction,0.0001))

    sim.setUserParameter(liftFan,'state',1)
    sim.setIntegerSignal('lift',1)

    sim.setEngineFloatParameter(sim.newton_body_staticfriction,localContactPatch,friction)
    sim.setEngineFloatParameter(sim.newton_body_kineticfriction,localContactPatch,friction)
    sim.resetDynamicObject(localContactPatch)

  elseif liftState == 0 and sim.getIntegerSignal('lift') == 1 then
    sim.setUserParameter(liftFan,'state',0)
    sim.setIntegerSignal('lift',0)

    sim.setEngineFloatParameter(sim.newton_body_staticfriction,localContactPatch,0.2)
    sim.setEngineFloatParameter(sim.newton_body_kineticfriction,localContactPatch,0.09)
    sim.resetDynamicObject(localContactPatch)
```

```
    end
    return round(friction,0.0001)
end

function getBoundingBoxXYSize(obj)
    a, size1min = sim.getObjectFloatParameter(obj, 15)
    a, size2min = sim.getObjectFloatParameter(obj, 16)
    a, size3min = sim.getObjectFloatParameter(obj, 17)
    a, size1max = sim.getObjectFloatParameter(obj, 18)
    a, size2max = sim.getObjectFloatParameter(obj, 19)
    a, size3max = sim.getObjectFloatParameter(obj, 20)
    size1 = size1max-size1min
    size2 = size2max-size2min
    size3 = size3max-size3min
    sizes = {size1, size2, size3}
    minSize = math.min(size1,size2,size3)
    for i = 1,3 do
        if sizes[i] == minSize then
            min = i
        end
    end
    table.remove(sizes,min)
    return sizes
end

function getCenterOfMass(modelBase)
-- reference: https://forum.coppeliarobotics.com/viewtopic.php?t=1719
-- Function returns the CoM for a given model in this format
-- {{CoMX, CoMY, CoMZ},{deltaX, deltaY, deltaZ},totalMass}
-- First find all non-static shapes in our model:

    allNonStaticShapes={}
    allObjectsToExplore={modelBase}
    while (#allObjectsToExplore>0) do
        obj=allObjectsToExplore[1]
        table.remove(allObjectsToExplore,1)
        if (sim.getObjectType(obj)==sim.object_shape_type) then
            --print("object# "..obj)
            r,v=sim.getObjectInt32Parameter(obj,3003)
            if (v==0) then -- is the shape non-static?
                table.insert(allNonStaticShapes,obj)
            end
        end
        index=0
        while true do
            child=sim.getObjectChild(obj,index)
            if (child==-1) then
                break
            end
            table.insert(allObjectsToExplore,child)
```

```
            index=index+1
        end
    end

    -- Now compute the center of mass of our model (in absolute coordinates):

    mass,inertia,base_com=sim.getShapeMassAndInertia(modelBase,nil)
    miri={0,0,0}
    totalMass=0
    loc_com = {}
    for i=1,#allNonStaticShapes,1 do
        --print(sim.getObjectName(allNonStaticShapes[i]))
        mass,inertia,com=sim.getShapeMassAndInertia(allNonStaticShapes[i],nil)
        miri[1]=miri[1]+mass*com[1]
        miri[2]=miri[2]+mass*com[2]
        miri[3]=miri[3]+mass*com[3]
        totalMass=totalMass+mass
    end
    final_com = {}
    final_com[1]=miri[1]/totalMass
    final_com[2]=miri[2]/totalMass
    final_com[3]=miri[3]/totalMass

    delta = math.sqrt((final_com[1]-base_com[1])^2+(final_com[2]-base_com[2])^2)

    return final_com,delta,totalMass
end


function round(exact, quantum)
-- Rounding function
-- https://stackoverflow.com/questions/18313171/lua-rounding-numbers-and-then-truncate
    if type(exact) == "number" then
        local quant,frac = math.modf(exact/quantum)
        return quantum * (quant + (frac > 0.5 and 1 or 0))
    elseif type(exact) == "table" then
        out = {}
        for i = 1,table.getn(exact) do
            out[i] = round(exact[i], quantum)
        end
        return out
    else
        print("Unexpected type sent to round() function")
    end
end




function sysCall_cleanup()
```

```
    -- Put some clean-up code here
    if USE_ARDUINO then
        if serial ~= -1 then
            sim.serialClose(serial)
        else
            print('Your serial port was not correctly opened at simulation start')
        end
    end
end
-- See the user manual or the available code snippets for additional callback functions and
details
```