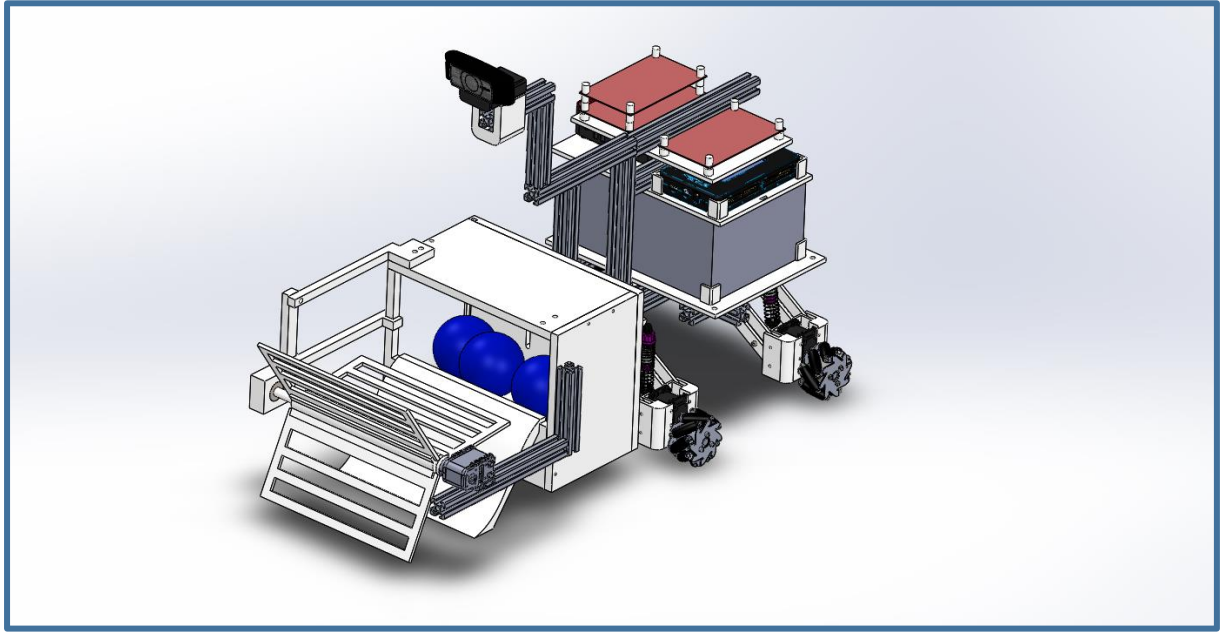


2018 Spring Capstone Design 1 Final Report

Group #3 (Team Hybrid System)

Ashar Alam (20140901)



Acknowledgement

A special thanks to the Professor and the TA of our group for their relentless effort and help. I would also like to thank professors of this course and other TAs for the guidance and feedback they provided.

Contents

1. Capstone Design 1: Background and Objective

1.1. Introduction.....	3
1.2. My Part: Labview.....	4

2. LabVIEW Progress: Progress, Guideline and Do's vs Dont's*

2.1. March.....	5
2.2. April.....	6
2.3. May	7
2.4. June: Final Code.....	8
2.5. Troubleshooting.....	9
2.6. References for future.....	10
2.7. Final Clean Code.....	11

3. Car Design and Iterations: Evolution

3.1. Motivation and Evolution.....	12
3.2. Suspension Evolution.....	16

4. Data Analysis: Heat Transfer and Vibration

4.1 Heat Transfer	17
4.2 Vibration.....	20

5. Conclusion.....22

6. Appendix

6.1 LabVIEW Assignments and Progress Report.....	23
--	----

1. Capstone Design 1: Background and Objective

1.1 Introduction

This year's Capstone Design 1 was radically different from past years'. Unlike past students; we were assigned a specific task: to build *a ball collecting autonomous robot*.

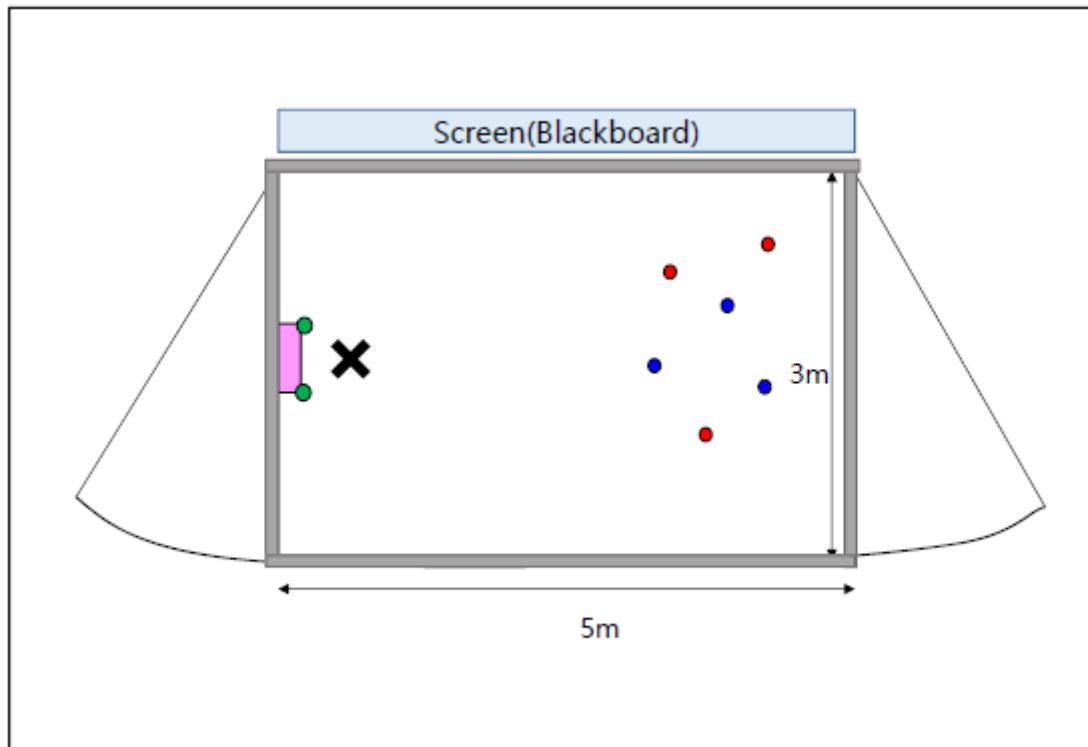


Figure 1 The final battleground

In the beginning, the task sounded simple. We had to collect 3 blue balls among a combination of 6 blue and red balls. Later, we had to deposit the three blue balls in a designated basket. The challenge was that all of this had to be done *autonomously*. For that, we had to pre-dominantly deal with softwares like LabVIEW, C++, ROS and Solidworks. So, we set out this task with a team of 7 students to perform the following objectives:

- **Pickup Mission:** We had to collect 3 blue balls within 5 minutes
- **Energy Mission:** We had to ensure that temperature of any part of our car doesn't rise upto 70 °C.

So we divided the members of our group into 4 division. Moreover, we set up a common contact point between each division so that a person in each team was in contact with the other divisions and that really helped us in the progress.

1.2 My part: LabVIEW

Labview forms the Measurement and Control part of the entire system. It is responsible for primarily three tasks as mentioned below:

- Measurement of various sensor data using my RIO.
- Actuation and control of DYNAMIXEL using my RIO.
- Control of whole system using myRIO.

In this project, we came up with efficient graphical programs with labview to integrate ROS and Open CV data to control the motors for driving the mechanum wheels.

It was a challenging task to control the motion of the car using mechanum wheel because these are omnidirectional wheels and this was our first time to interact with such a driving mechanism. Regardless, of that we worked hard and were successfully able to achieve the desired task with the help of codes.

Control system forms one of the most important part of integrated systems like aircrafts, spacecrafts and also vehicles. It includes gathering data (from sensors), processing it and activating actuators in response to the feedback from the sensors.

We used following as data sources for our final program:

- Temperature sensor data: For heat transfer and temperature analysis
- Camera: Using open CV for image processing
- LIDAR: Distance sensor using light as pulsed laser
- ROS: Robot operating system will help to integrate the above data and communicate with myRIO to run the LabVIEW code.

Based on the data and information obtained from above, LabVIEW worked on the feedback to control the following actuators:

- DYNAMIXEL motors and mechanum wheels: Based on the decision from ROS, we decided the minimum distance path to follow and controlled the motors to give desired motion using the omnidirectional mechanum wheels.
- We actuated the gripping motor when we are near the ball to collect it.

Since, we were new to LabVIEW, our TAs designed weekly assignments for us to progress through the learning of LabVIEW. After we had learnt them; we starting designing our own codes to solve the problems as and when required by us. We will talk about the details in the next section.

2. LabVIEW Progress: Progress, Guideline and Do's vs Dont's*

2.1 March

The first month (March) was dedicated to learning LabVIEW and my Rio interface. We were not exposed to this device beforehand and not having an electrical background makes matters worse. So we spent most of the time learning the LabVIEW interface and installing labview. Moreover, this month we were doing most of the assignments which was supposed to help us in learning LabVIEW. We only had two lectures for LabVIEW and I **strongly** suggest the course administrators to conduct more LabVIEW lectures. The homework's do help us to learn but it will be better if we get lectures to better understand things.


Don'ts:


- Do not leave assignments for the last day. These are not the usual assignments where you can take an equation and solve by using plug and chug. These are programming assignments which will take time. So start in advance to avoid hassle.
- Do not hesitate to contact TAs if you need any help.


Do's:

- Do view online lectures and materials related to LabVIEW to get acquainted with it in the beginning.
- Do ask the TA's for help if you do not understand any part. Sometimes, it's the fastest way of getting the problem solved.
- Do try to install LabVIEW on your personal computer to get a headstart on the software.

Guide*:

 [Detailed Labview Basics \(https://k12lab-support-pages.s3.amazonaws.com/lvThisbasichome1.html\)](https://k12lab-support-pages.s3.amazonaws.com/lvThisbasichome1.html): This is a very detailed introduction of basic LabVIEW commands. Highly recommended if you already know basics of loops like for/while and conditionals like if/else and want to learn how to implement them in LabVIEW.

 [Brief \(http://home.hit.no/~hansha/documents/lab/Lab%20Work/LabVIEW%20Programming/Basic%20LabVIEW%20Programming/Basic%20LabVIEW%20Programming.pdf\)](http://home.hit.no/~hansha/documents/lab/Lab%20Work/LabVIEW%20Programming/Basic%20LabVIEW%20Programming/Basic%20LabVIEW%20Programming.pdf)) A very brief and short intro to LabVIEW.

 [Video Tutorials \(https://www.labviewmakerhub.com/doku.php?id=learn:tutorials:labview:basics\)](https://www.labviewmakerhub.com/doku.php?id=learn:tutorials:labview:basics)
This one is a very helpful video guide. This one is extremely helpful if you are one of those people who prefer learning from video lectures.

* This is not an exhaustive list. These references are in addition to what TA's will give you.

2.2 April

In the second month (April), we actively started working on communication between an input device (keyboard/ Xbox Controller) and NUC PC. First we had to tackle bi-directional communication TCP/IP communication between myRIO and Xbox controller and also control the motors using the keyboard.

There were many challenges related to TCP/IP communication. The sample code looked very complicated but once we tried to understand the whole thing it became very easy to understand and operate. We also connected the DYNAMIXEL motors to myRIO using the U2D2. We were successful in establishing the communication and control but the TCP/IP communication was still slow.

We also used Dynamixel Wizard to identify the motors. You might have some problems in setting up motor IDs only if the U2D2 is not working.

Don'ts:


- Do not start using the sample TCP/IP code without understanding. It might seem to be a tedious task to understand the whole TCP/IP code when you are beginning to learn LabVIEW but a small amount of time spent now will go a long way to solving communication problems
- Do not hesitate to contact TAs if you need any help.


Do's:

- Do try to make a simple code from the sample code provided for TCP/IP communication.
- Do try to use Xbox communication along with the keyboard communication. It makes it easier to test.
- Do check the motor connections and U2D2 for error if you fail to control the motors.
- Make sure that the client starts before the server code otherwise it shows error.

Guide*:

 [Basic TCP/IP Communication](http://www.ni.com/white-paper/2710/en/)(<http://www.ni.com/white-paper/2710/en/>): This is a very basic intro to TCP/IP communication

 [Dynamixel Wizard](http://support.robotis.com/en/software/roboplus/dynamixel_monitor.htm) (http://support.robotis.com/en/software/roboplus/dynamixel_monitor.htm) A guideline to use dynamixel wizard [Your wizard might have a different interface as shown in this website]

 [Detailed Dynamixel](http://emanual.robotis.com/docs/en/software/rplus1/dynamixel_wizard/)(http://emanual.robotis.com/docs/en/software/rplus1/dynamixel_wizard/) This is a more detailed version of how to use and operate Dynamixel Wizard. This also helps in firmware recovery in cases of error.

* This is not an exhaustive list. These references are in addition to what TA's will give you.

2.3 May

In the second month (April), we were getting close to the final competition. After establishing active communications, we had to make the communication faster by changing the TCP/IP Code. Moreover, we had to establish the direction of motors in-order to make the rotation of various wheels culminate into movement of the car in a particular direction.

We first realized that the sample code provided to us was bi-directional communication and hence the practical approach would be to make it uni-directional. However, making it needs to find specific bytes we are sending from Xbox controller code to the server code. For that, if you use structures you have to look up the help section of LabVIEW to verify its size of data variables.

Don'ts:

- Do not use the sample bi-directional TCP/IP Communication for final race as it will be very slow.
- Do not use large data size for communication. Try to keep it simple.
- Do not assume that data types used in C++/Python have the same size as in LabVIEW. (float in LabVIEW might have different size as compared to C++ due to dependence on 32/64 bit variations.)
- Do not use local variables as they can cause errors in the code.
- Do not hesitate to contact TAs if you need any help.

Do's:

- Do understand how the data size affects the various functions.
- Do try to use *notifiers* instead of local variables if your code needs local variables.
- Do use *Indicators* to check for data output to debug the code. It is very simple to using print statements in various programming languages for debugging.

Guide*:

- 🔗 [Data Types](http://zone.ni.com/reference/en-XX/help/371361P-01/lvhowto/numeric_data_types_table/) (http://zone.ni.com/reference/en-XX/help/371361P-01/lvhowto/numeric_data_types_table/): These are the sizes of various data types used in LabVIEW.
- 🔗 [Advanced LabVIEW Tutorial](ftp://ftp.ni.com/pub/branches/uk/devdays_2012/Data_Communication_with_LabVIEW.pdf) (ftp://ftp.ni.com/pub/branches/uk/devdays_2012/Data_Communication_with_LabVIEW.pdf) This is a very useful advanced tutorial for LabVIEW of use for you at this stage. It deals with TCP/IP, Queues, Notifiers and Local Variables and more.

* This is not an exhaustive list. These references are in addition to what TA's will give you.

2.4 June

This was the final month and we had to solve the error problems. We successfully replaced the code with a uni-directional TCP/IP Communication. However, we were implementing delays in the loop time of the LabVIEW code which was causing communication delays in the main code of ROS.

So, we removed the delay from LabVIEW and tried to implement it in the ROS code. This solved our delay problem and we were able to run the car properly.

We were using MX type Dynamixel and we tried to connect an AX type Dynamixel for the gripper. However, the inherent code for MX is a bit different from AX. For AX to work, you should check the Dynamixel SubVI and make the necessary changes.

A week before final race, our motors stopped working. We spent a lot of time wondering if the motors had short circuited or had broken down. We thought that we might have changed the code but once we realized that we were using the same code previously; we concluded that it must be a hardware issue. Finally, a broken U2D2 was found out to be the root of the problem.

We also verified acceleration data using myRio's inbuilt accelerometer to verify superior performance of our vehicle with the help of an independent suspension.

We further tried to Deploy our code directly onto myRio by building Real Time Application. However, it also caused communication delays. Hence, we went back to running LabVIEW directly on PC.



Don'ts:

- Do not panic if an error is discovered at the end. Try to be calm and identify it in a sequential manner.
- Do not implement delays in the while/for loops of LabVIEW as it might lead to delay in communications between myRio and NUC.
- Do not use the same VI's for different type of Dynamixel motors.
- Do not hesitate to contact TAs if you need any help.

Do's:

- Do try to make the code simple and straightforward.
- Do try to use accelerometer of myRio for vibration analysis of the vehicle. It is very helpful.
- Do ask TAs in the event if some part fails very near to the race. They might have some spare parts which might help you.

Guide*:

-  [Code clean up utility](http://www.ni.com/tutorial/7386/ko/)(<http://www.ni.com/tutorial/7386/ko/>): There is a clean up utility in LabVIEW in case your code has too many wires.
-  [LabVIEW Debug Tools](http://www.ni.com/getting-started/labview-basics/ko/debug) (<http://www.ni.com/getting-started/labview-basics/ko/debug>) Some debug tools which might be helpful in LabVIEW.

* This is not an exhaustive list. These references are in addition to what TA's will give you.

2.5 Troubleshooting

LabVIEW Debug Tools (<http://www.ni.com/getting-started/labview-basics/ko/debug>)

Some debug tools which might be helpful in LabVIEW.











As you can see above, I have already introduced basic debug tools of LabVIEW which I used. Moreover, I have listed down some problem's which I faced and how I solved them. This might help you next time you are stuck with some problem in LabVIEW.*

- × **Error connecting with Rio Wifi:** When myRio Wifi connected to my internet; it showed ('Connected, Limited'). I thought the limited word might be connection error. But it's not an error; it still perfectly works. So, just ignore this message.
- × **Delay in communication:** Use uni-directional code instead of bi-directional communication in TCP/IP.
- × **Code error:** Sometimes just using indicators in the code can help you a lot in debugging the codes.
- × **Weird Accelerometer data:** The accelerometer data might look haphazard. Try adjusting the loop time or properties of the graph to get a proper representation.
- × **Local Variable not transferring data:** Try to use notifiers instead. Local variables are considered a big 'NO'.
- × **Differences in AX/MX motor speed:** The speed of AX and MX motors are to be set with different values. The same speed of MX motor might not even run AX motor slowly. Look at the Sub Vis for more information.
- × **Corrupted Dynamixel Firmware:** Sometimes, DYNAMIXEL Ids maybe mixed up due to non-termination of codes after finishing program. You may have to use the Firmware utility to perform a firmware recovery using Dynamixel Advanced Tutorial in Section 2.2.
- × **Data Type Mismatch between LabVIEW and ROS:** There might be a data type mismatch between LabVIEW and ROS. The number of bits/bytes being sent by ROS should be equal to the ones being received by LabVIEW. Thus, you should be very careful about data types like Structures and also size of arrays.
- × **The wheels not working properly:** Sometimes turning the wheels do not make the car turn properly in a particular direction. It might seem that the code is to blame; but the real culprit is the friction created due to loose wheels. Try to tightly attach the wheels to the motor as well as try to modify the speed of wheel rotation to counter the friction issue.
- × **Apparent Motor breakdown:** At one point it seemed like all our motors broke down. However, days of searching; we found that problem was with U2D2. So please look at U2D2 if all the motors don't work. If only 1 motor doesn't work, only then focus on the motor. Because we encountered this problem 2 times; we think that U2D2 is one of the device which creates many problems. Moreover for connecting Dynamixel's; the Daisy Chain connections doesn't have to be a closed loop.

* My solutions to these problems might not be the recommended way to solve these or might even be wrong. But please try to understand that I am a student not having a very detailed knowledge on this subject. My intention is to just help you with some problems.

2.6 References for Future

I have listed some references in each section before. However, I have compiled these references here again for your easy reference:

-  [**Detailed Labview Basics**](https://k12lab-support-pages.s3.amazonaws.com/lvThisbasichome1.html) (<https://k12lab-support-pages.s3.amazonaws.com/lvThisbasichome1.html>): This is a very detailed introduction of basic LabVIEW commands. Highly recommended if you already know basics of loops like for/while and conditionals like if/else and want to learn how to implement them in LabVIEW.
-  [**Brief**](http://home.hit.no/~hansha/documents/lab/Lab%20Work/LabVIEW%20Programming/Basic%20LabVIEW%20Programming/Basic%20LabVIEW%20Programming.pdf)
(<http://home.hit.no/~hansha/documents/lab/Lab%20Work/LabVIEW%20Programming/Basic%20LabVIEW%20Programming/Basic%20LabVIEW%20Programming.pdf>) A very brief and short intro to LabVIEW.
-  [**Video Tutorials**](https://www.labviewmakerhub.com/doku.php?id=learn:tutorials:labview:basics)
(<https://www.labviewmakerhub.com/doku.php?id=learn:tutorials:labview:basics>) This one is a very helpful video guide. This one is extremely helpful if you are one of those people who prefer learning from video lectures.
-  [**Data Types**](http://zone.ni.com/reference/en-XX/help/371361P-01/lvhowto/numeric_data_types_table/) (http://zone.ni.com/reference/en-XX/help/371361P-01/lvhowto/numeric_data_types_table/): These are the sizes of various data types used in LabVIEW.
-  [**Code clean up utility**](http://www.ni.com/tutorial/7386/ko/) (<http://www.ni.com/tutorial/7386/ko/>): There is a clean up utility in LabVIEW in case your code has too many wires.
-  [**LabVIEW Debug Tools**](http://www.ni.com/getting-started/labview-basics/ko/debug) (<http://www.ni.com/getting-started/labview-basics/ko/debug>) Some debug tools which might be helpful in LabVIEW.
-  [**Advanced LabVIEW Tutorial**](ftp://ftp.ni.com/pub/branches/uk/devdays_2012/Data_Communication_with_LabVIEW.pdf)
(ftp://ftp.ni.com/pub/branches/uk/devdays_2012/Data_Communication_with_LabVIEW.pdf) This is a very useful advanced tutorial for LabVIEW of use for you at this stage. It deals with TCP/IP, Queues, Notifiers and Local Variables and more.
-  [**Basic TCP/IP Communication**](http://www.ni.com/white-paper/2710/en/) (<http://www.ni.com/white-paper/2710/en/>): This is a very basic intro to TCP/IP communication
-  [**Dynamixel Wizard**](http://support.robotis.com/en/software/roboplus/dynamixel_monitor.htm)
(http://support.robotis.com/en/software/roboplus/dynamixel_monitor.htm) A guideline to use dynamixel wizard [Your wizard might have a different interface as shown in this website]
-  [**Detailed Dynamixel**](http://emanual.robotis.com/docs/en/software/rplus1/dynamixel_wizard/) (http://emanual.robotis.com/docs/en/software/rplus1/dynamixel_wizard/) This is a more detailed version of how to use and operate Dynamixel Wizard. This also helps in firmware recovery in cases of error.

2.7 Final Code

As you can see below, we designed our final code, which is very simple and easy to use, understand and debug. Also, you can include comments in the code for easier understanding.

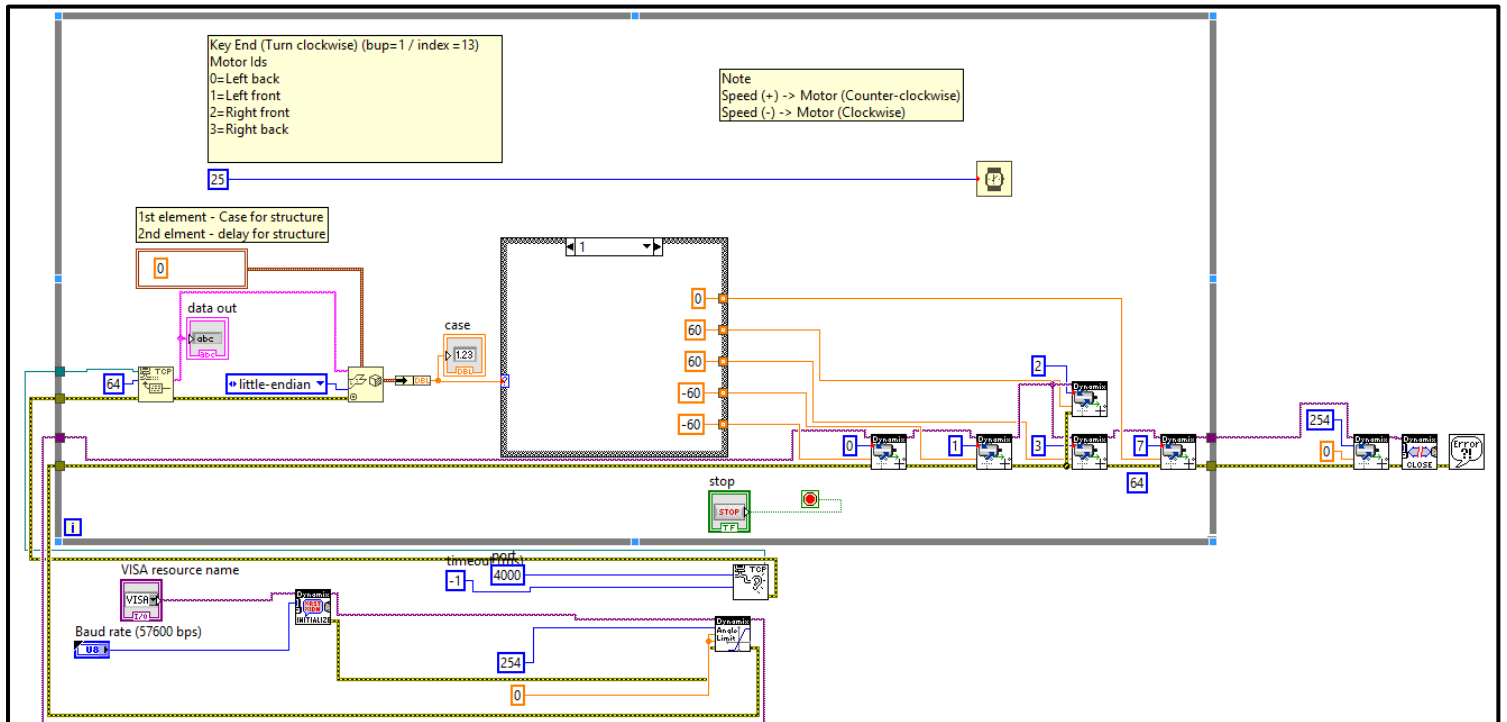


Figure 2 A simple code which we used to run the actuators

You can see as the above code is very simple and straightforward. This allows easier method to debug and deconstruct the code. It also includes simple comments for reference. As you can see below, we designed our final code, which is very simple and easy to use, understand and debug. Also, you can include comments in the code for easier understanding.

3. Car Design and Iterations: Evolution

3.1 Motivation and Evolution

We drew our inspiration from a concept we found on the internet as shown below:

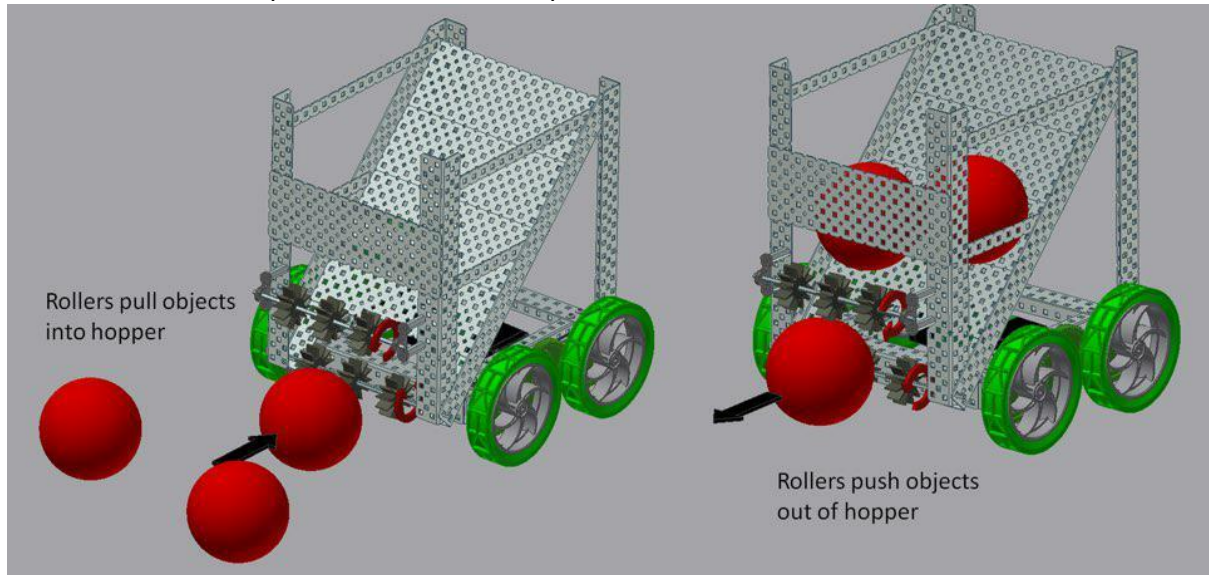


Figure 3 Inspiration for our design

This was a good idea, however, this was not suitable as there was fear of the balls getting stuck in rollers. Hence, we decided to modify the design as per our requirements:

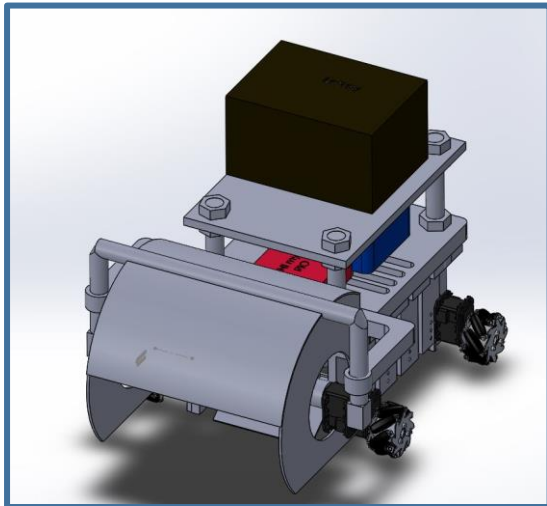


Figure 4 First Design

This was our first design. However, this has had issues due to improper wheel functioning and rotation.

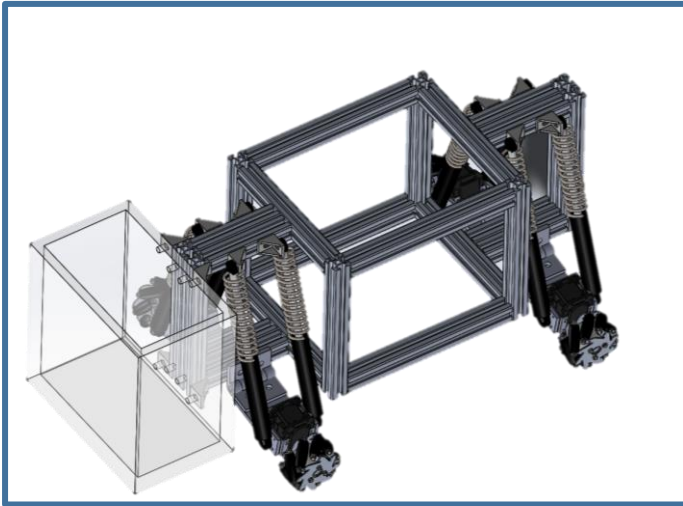


Figure 5 Our second design with McPherson Strut suspension

This was our second design with McPherson Strut type suspension. However, this had a problem with proper functioning, so we also changed this design.

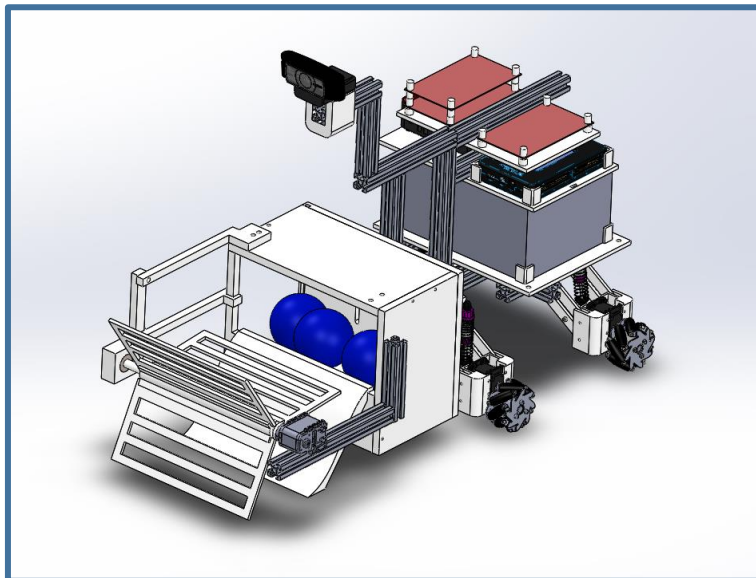


Figure 6 Final Design

This was our final design. As you can see this shows the evolution of design and how our design was modular.

This design features a simple and flexible picking mechanism along with an independent suspension and the camera mounted on top.

The battery and other electronic equipment were mounted above the car at the highest position. The suspension system helped to reduce the vertical vibration which improved the camera's detection performance.

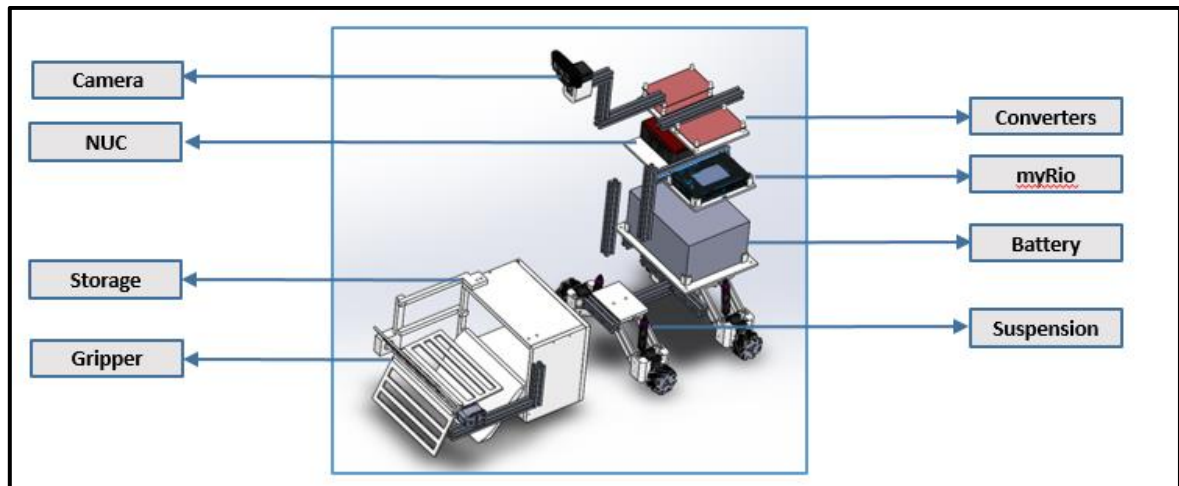


Figure 7 Final Car Design with various parts labelled

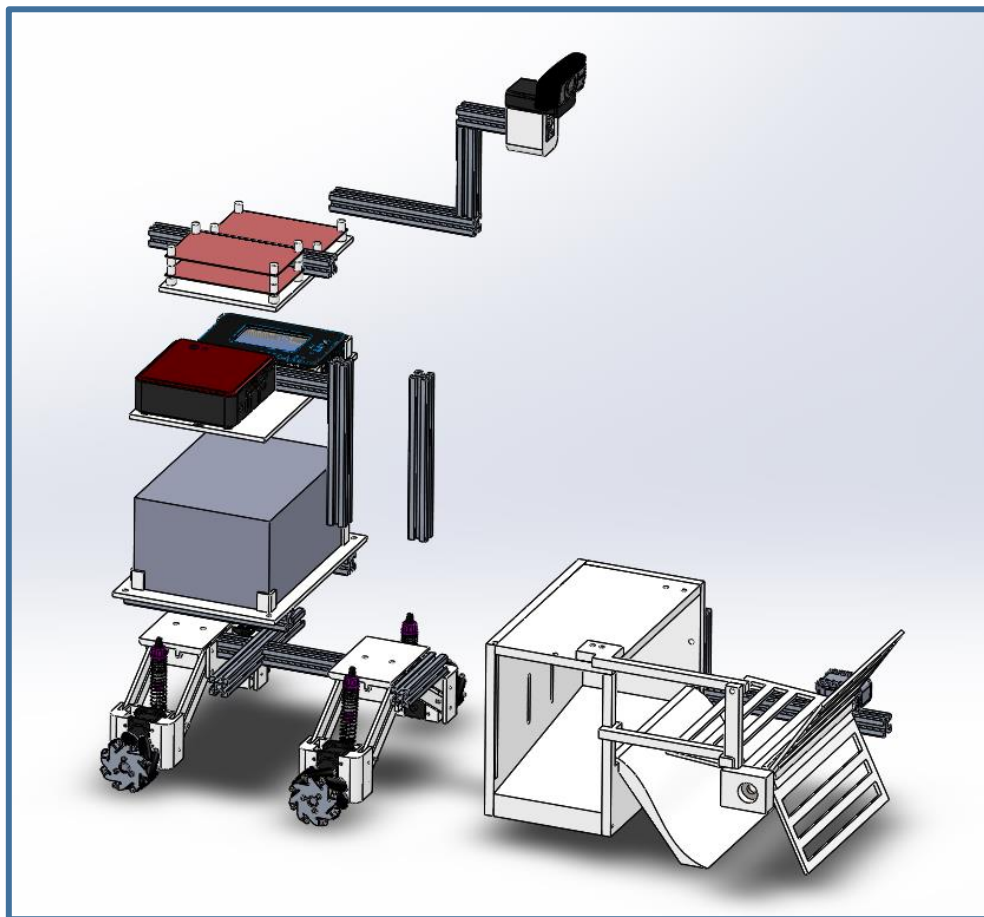


Figure 8 Modular approach of our car; different parts can be attached

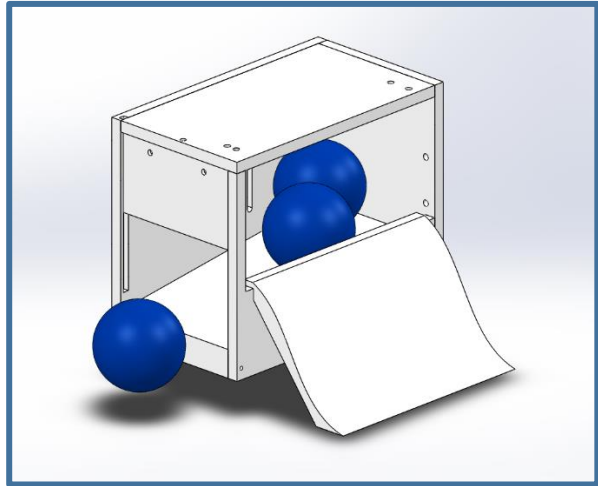
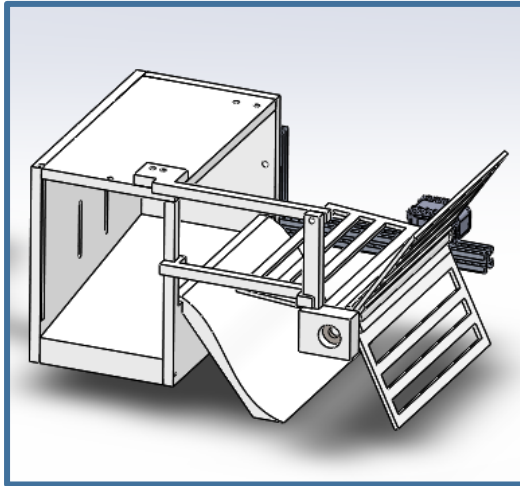


Figure 9 Our gripping and storage mechanism

3.2 Suspension Evolution

Suspension was another USP of our car to reduce vibration. Let's see the evolution below:

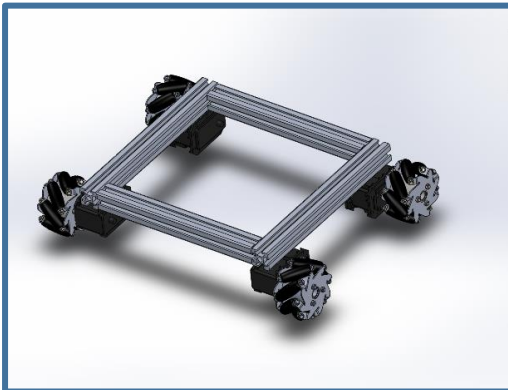


Figure 10 First chassis without suspension

This was our first chassis; however it lead to large vibrations due to lack of suspension. Also, the wheel vibrations lead to improper turning of the wheels.

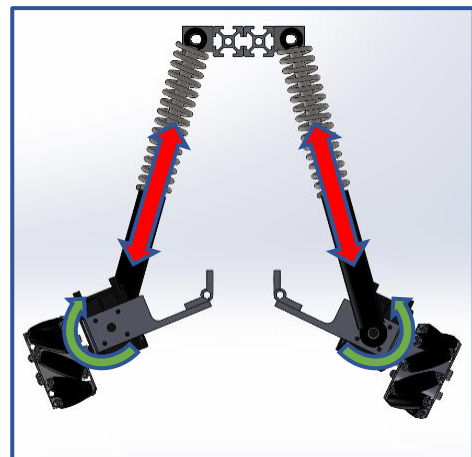
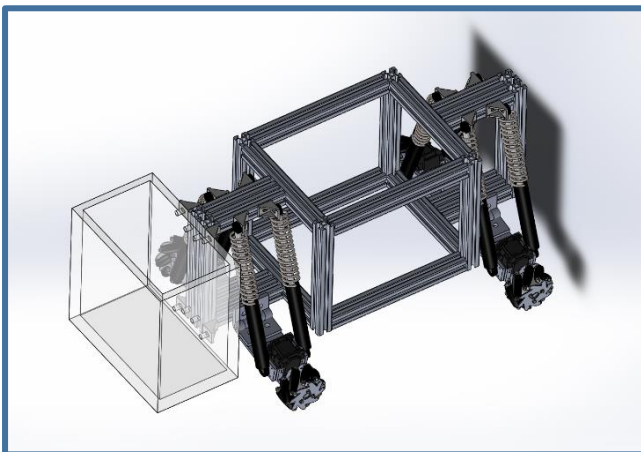


Figure 11 Our second design of suspension with a MacPherson Strut type suspension

You can see in the above design that the McPherson strut type suspension also had some problems. This was because our wheels are not just rubber wheels but Mecanum wheels. Their motion leads to change in the camber angle causing the rim of the mecanum to touch the ground.

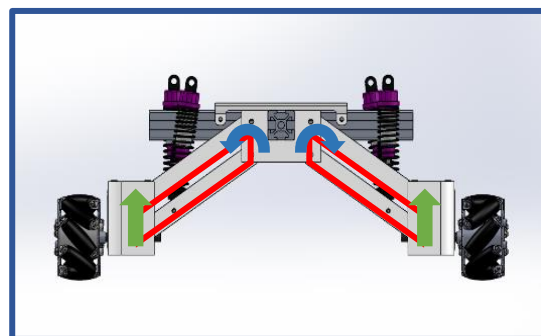
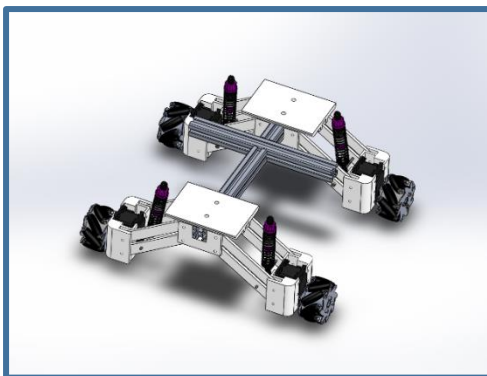


Figure 12 Our final independent suspension design: Camber angle is not changed even if the vehicle bounces and provides stability to the camera mount from base excitation

4. Data Analysis: Heat Transfer

4.1 Heat Transfer

We took pictures of the hot spots from the infra-red camera to identify the highest temperature areas of the car as shown below:

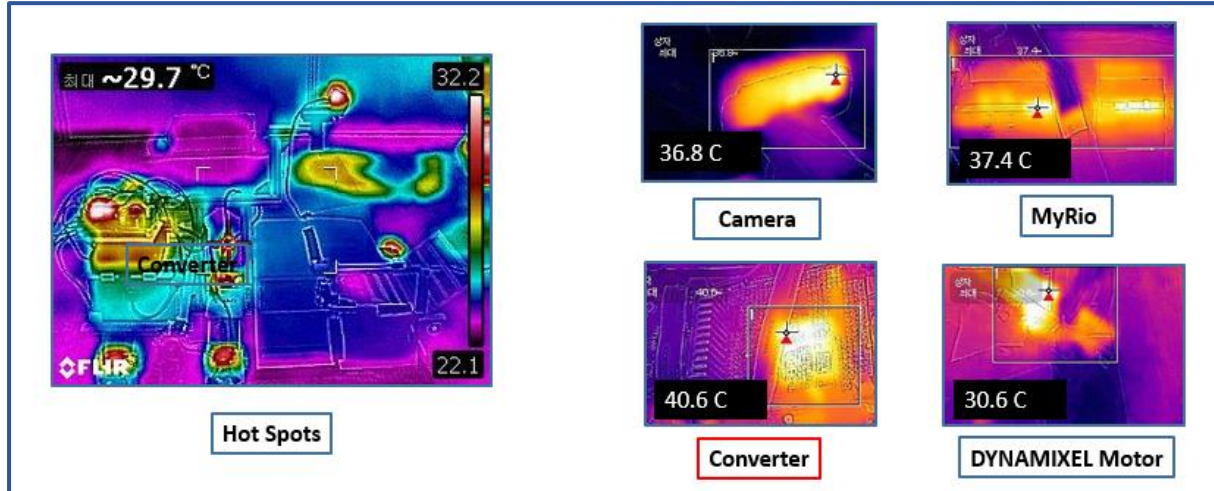


Figure 13 The hot spots and temperatures found with the IR camera

We further observed that the Converter had the highest temperature so we tried to solve the problem by focusing on the converted and finding theoretical temperatures.

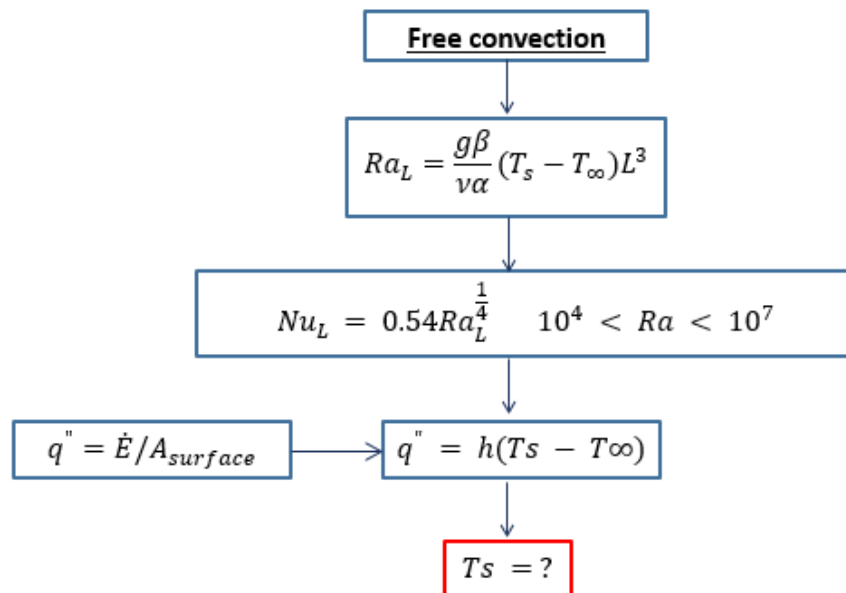


Figure 14 Some formulas used in calculation

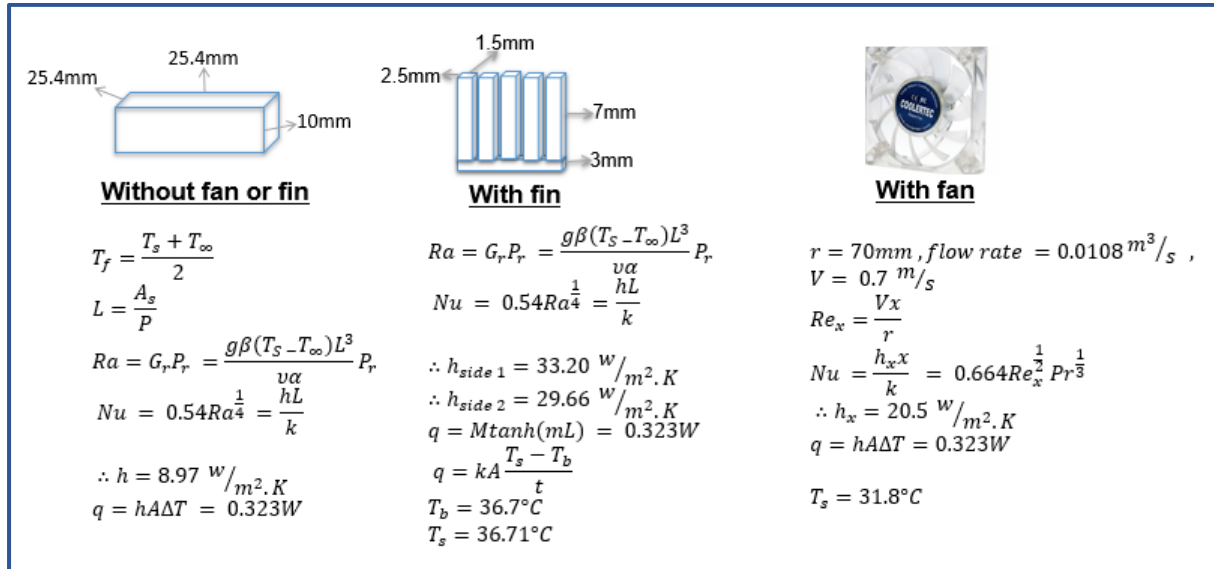


Figure 15 Our calculations for the temperatures

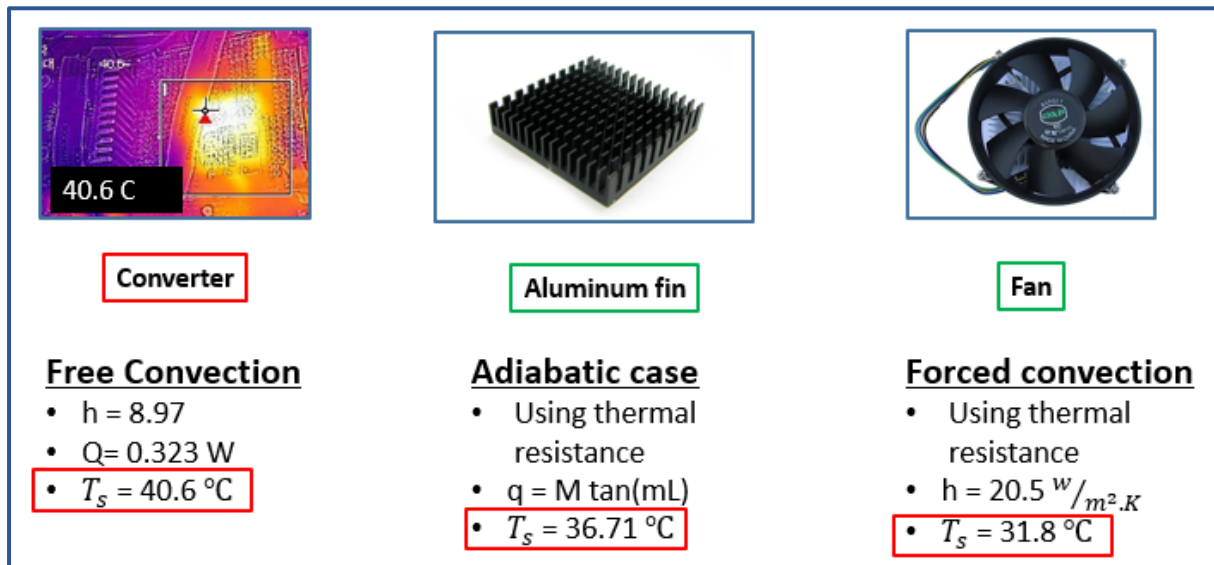


Figure 16 Our results for calculation

Our theoretical results with fin and fan tend to reduce the temperature. We did some calculations to obtain surface temperature of converter in the free convection case adiabatic case and forced convection case to obtain the prospects of using fin/fan. The calculations are shown in figure 15.

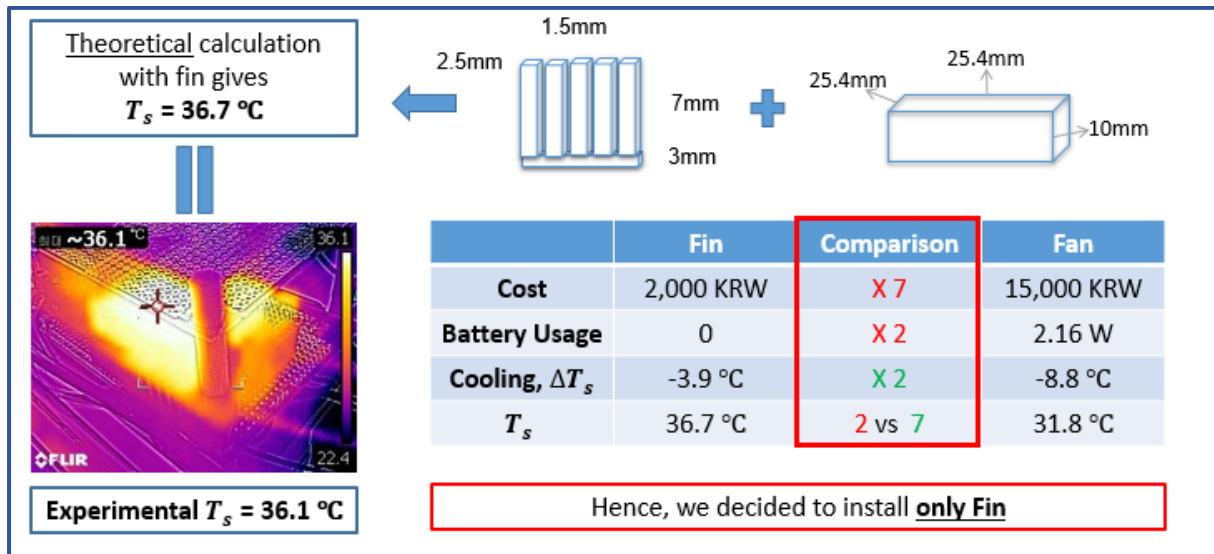


Figure 17 Our final conclusion for heat transfer

As we can show here our theoretical calculation matches well with the experimental observation after adding fin.

We used a cost vs benefit analysis to decide upon using a fin or a fan. We compared the cost and benefits of using fin and fan and settled for a fan because it offered better cost/benefit ratio. This was the conclusion of our heat transfer analysis. Thus, we were able to keep all the temperatures well below the required 70 $^{\circ}\text{C}$.

4.2 Vibration

Our suspension is the USP of our vehicle. Below is the theoretical modelling of our suspension:

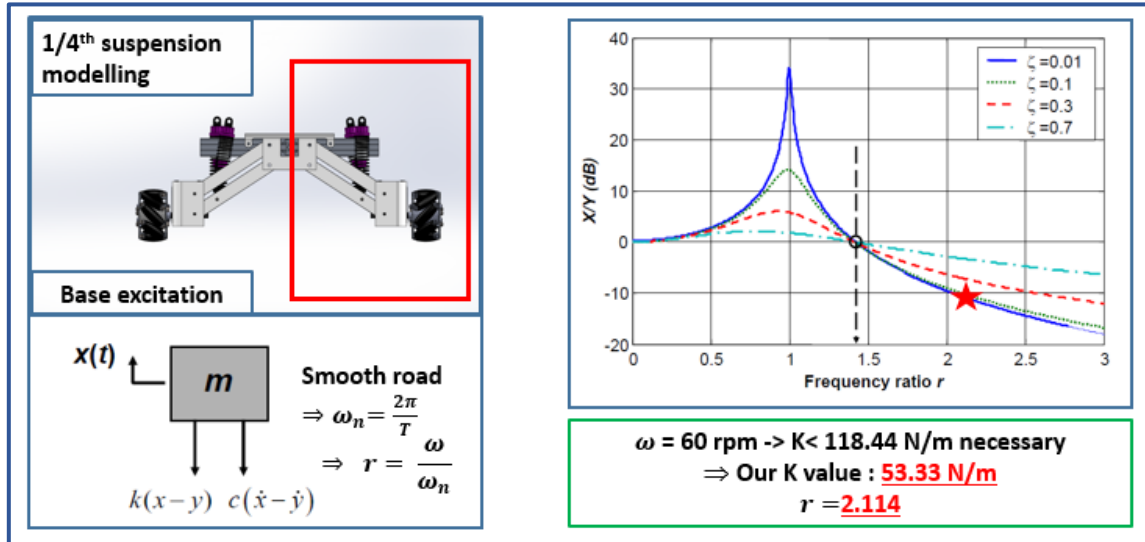


Figure 18 Physical vibration modelling of our suspension

When we observed the full operation of our vehicle, we concluded that almost all of our vibration comes from mecanum wheel. Therefore, we approximate our suspension model with a quarter car model.

As we estimated our vehicle to have 60 rpm velocity, the stiffness should be less than 118 N/m. We calculated proper stiffness value from equation of coil spring constant. By controlling each diameter, we fixed the value to 53N/m so that our design reduced vibration about **-10db** of magnitude theoretically

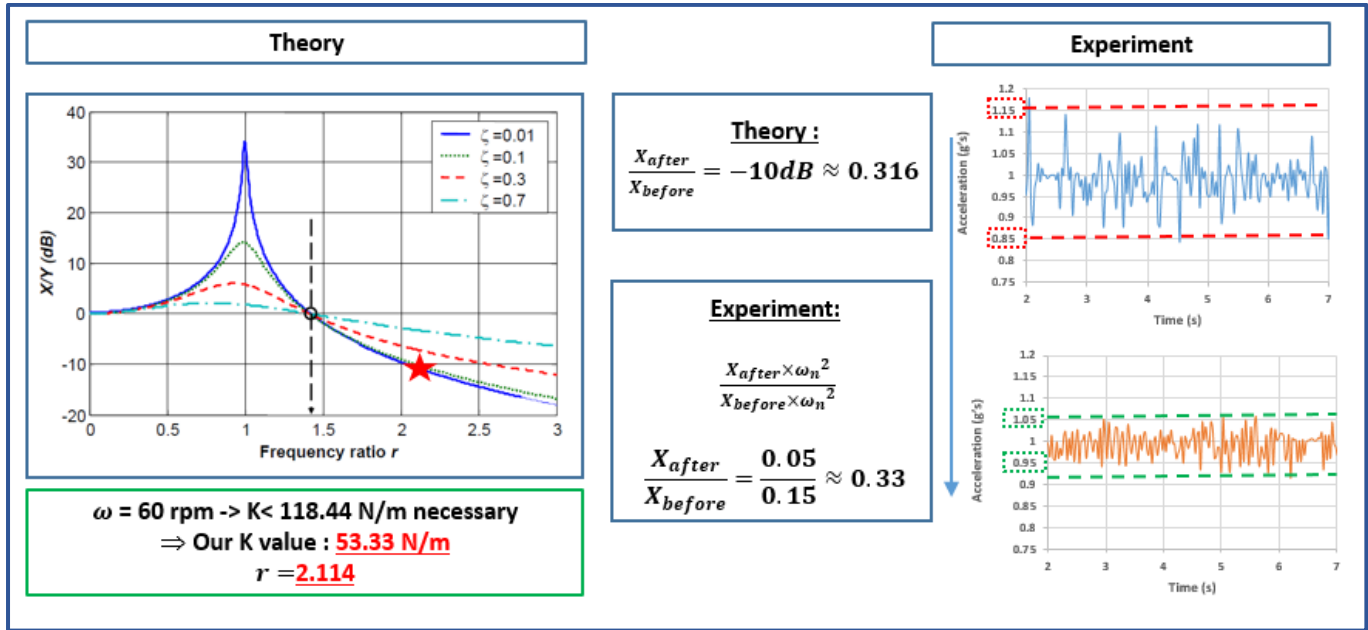


Figure 19 Comparison between our theoretical and experimental calculations

As shown in the figure 19 above, our theoretical calculations showed a reduction of about -10 dB or 0.316 in vibrations. Moreover, our experiments confirm the results with a 0.3 reduction of vibration. Thus, it shows that our suspension vibration modelling was very robust and our perfect camera detection confirms this fact.

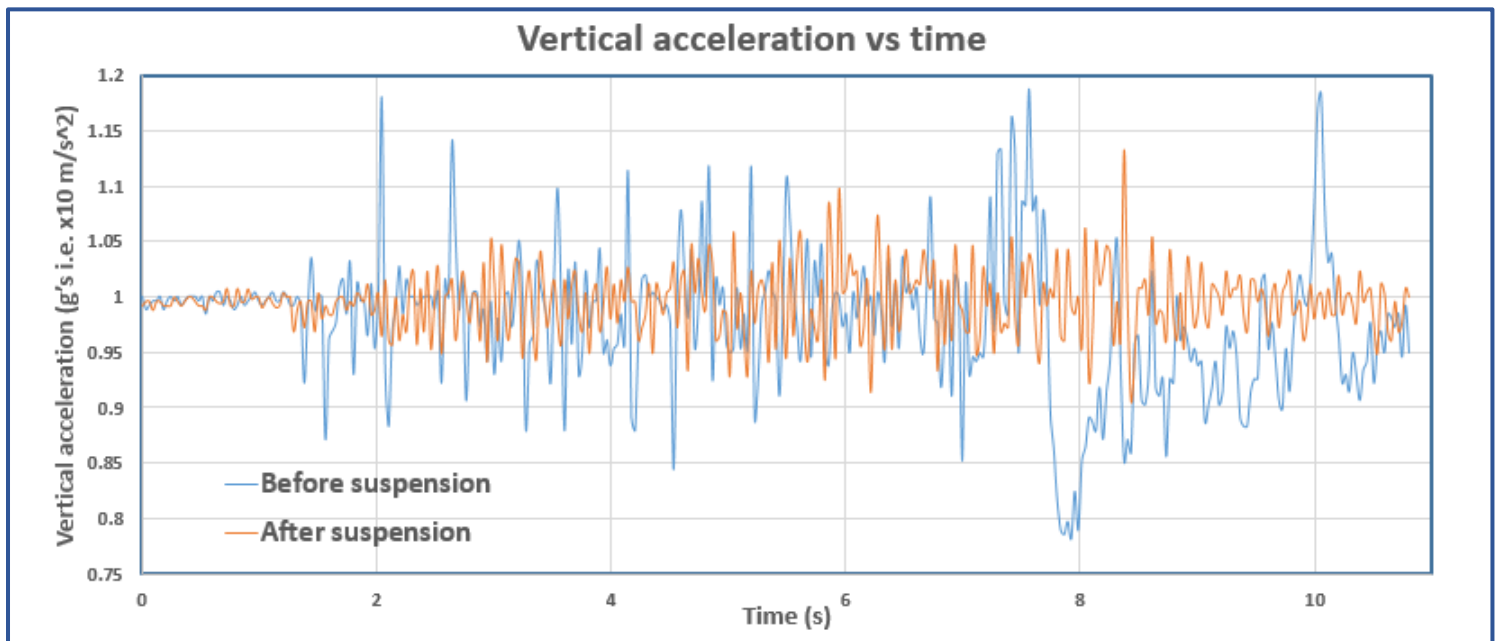


Figure 20 Vibration before and after suspension was incorporated

Figure 20 above shows that the vibration after suspension was incorporated (in orange color) is substantially reduced as compared to vertical vibration without suspension (in blue color)

5. Conclusion

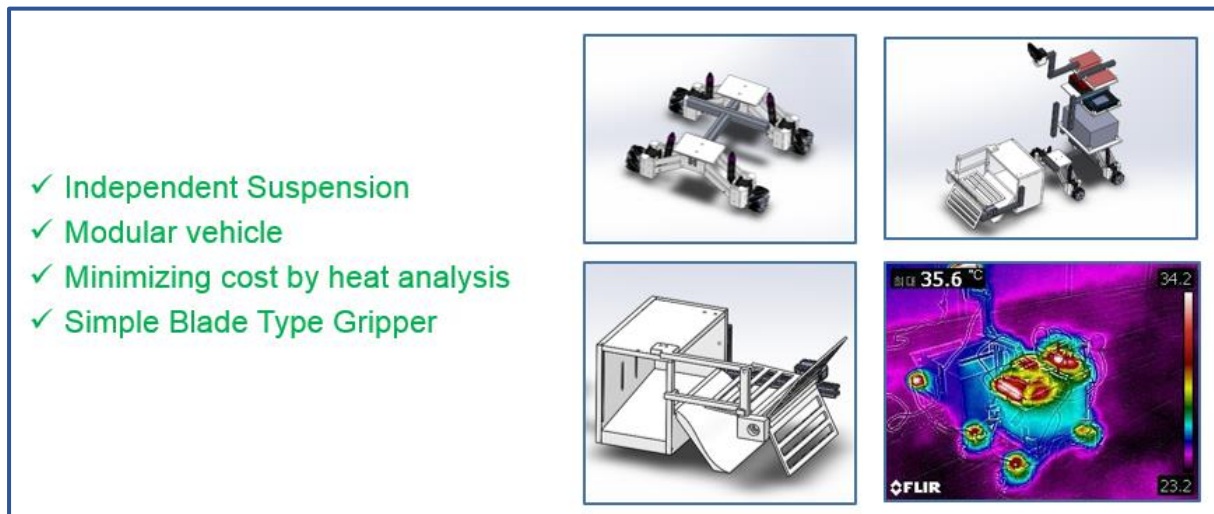


Figure 21 Key features of our car

The above figure represents the key features of our car. We have an independent suspension system which reduces camera vibration.

Our system is modular which can help to easily repair and maintain. The design changes if required, follow a smooth and swift transition. It is also easy to implement (add or remove) new features. Our team had foresight for Capstone Design II as it will be easy to attach a new gripper system or implement any other modifications for Capstone Design II.

We have been able to reduce temperature rise by using minimum cost fins by a carefully calculated and tested heat transfer analysis.

Our blade type gripper is a flexible and simple, lightweight design which does not falter in collecting the balls precisely and perfectly with minimum power usage.

This project was very challenging. However, our team tried our best. We worked together for days and nights and even though we could not win the competition; we are satisfied with our progress.

6. Appendix

6.1 Assignments (Week 1-4) and Compiled Progress Reports (Week 1 – 11)

Assignment #1 (Submitted March 16 – for 1st two weeks)

Overview of our assignment:

- **Configuring my RIO (Connections and basic measurements):** We were introduced to myRIO and performed initial configuration of myRIO like learning how to connect myRIO with the computer and setting up the programming environment. Following screenshots illustrate the process:

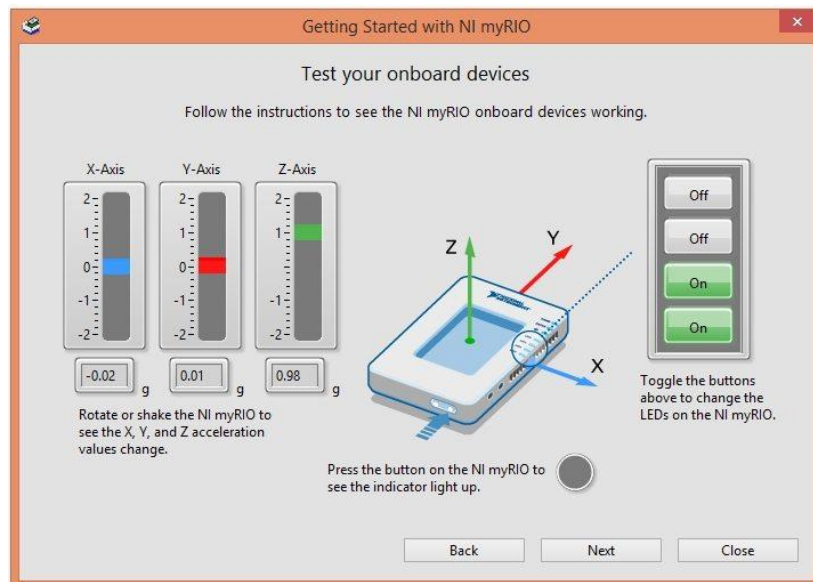


Figure 22 Calibration of myRIO accelerometer and LED

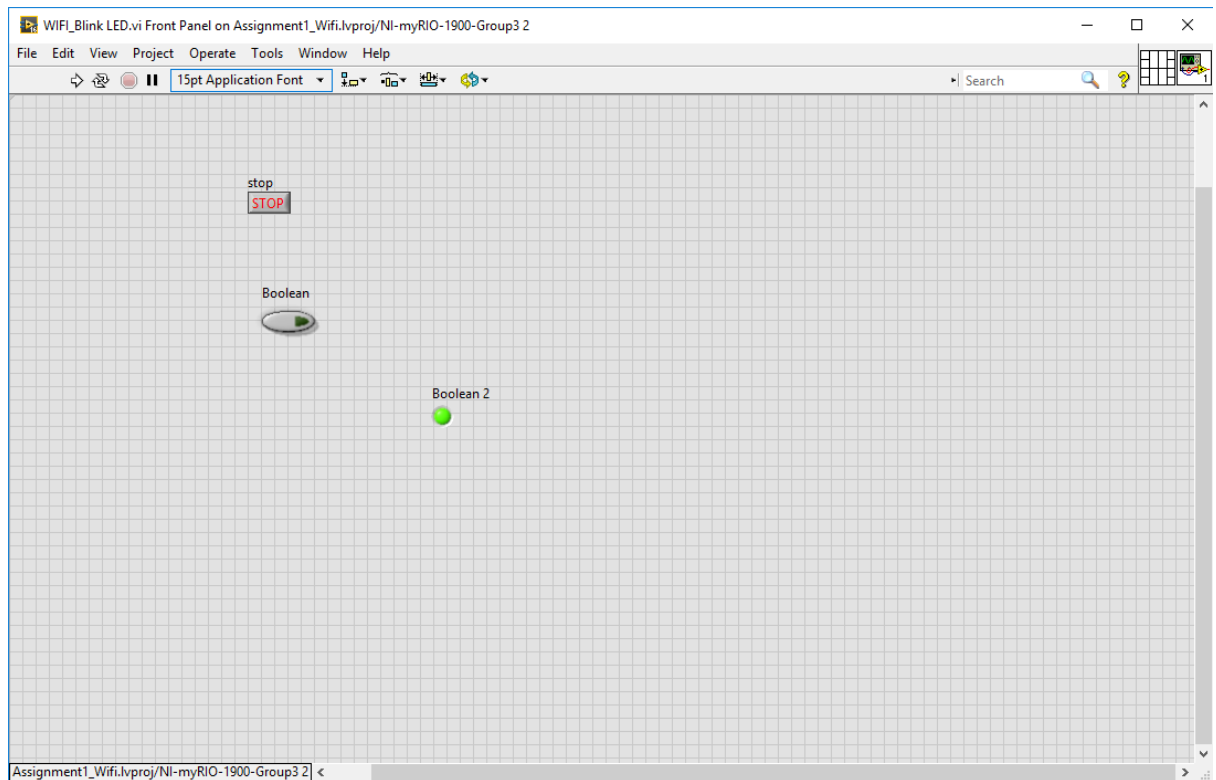


Figure 23 Blinking LED Example Front Panel

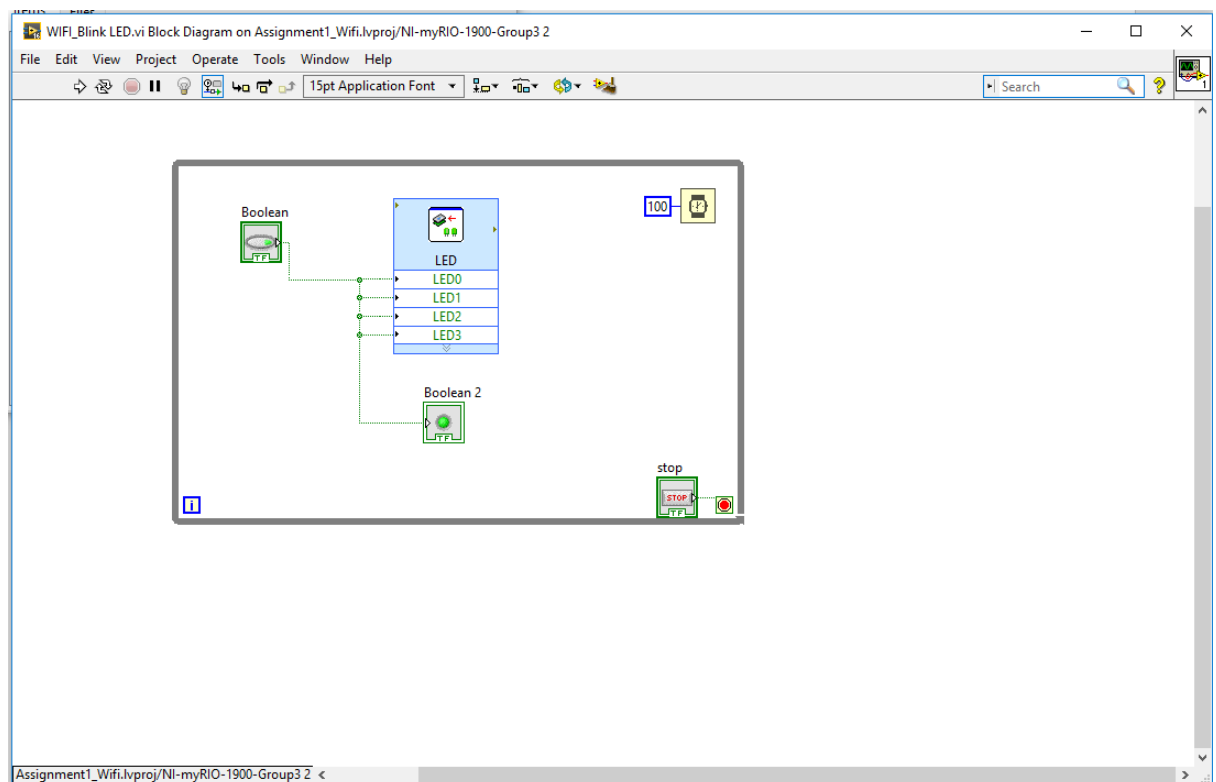
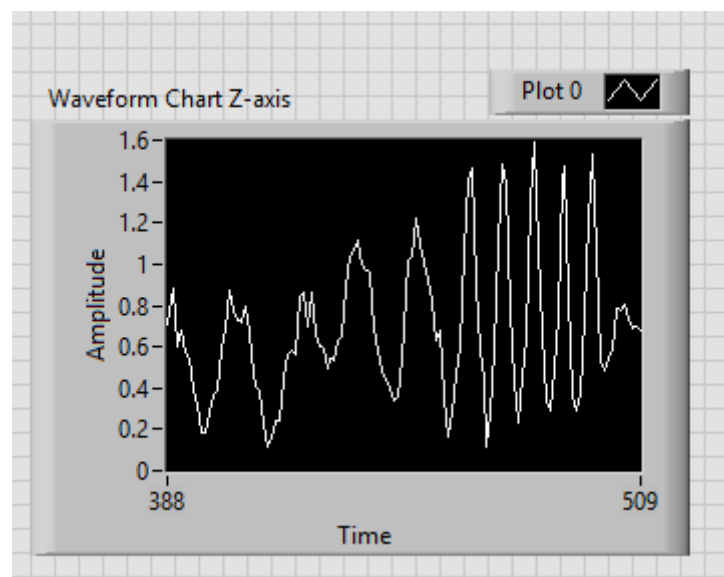
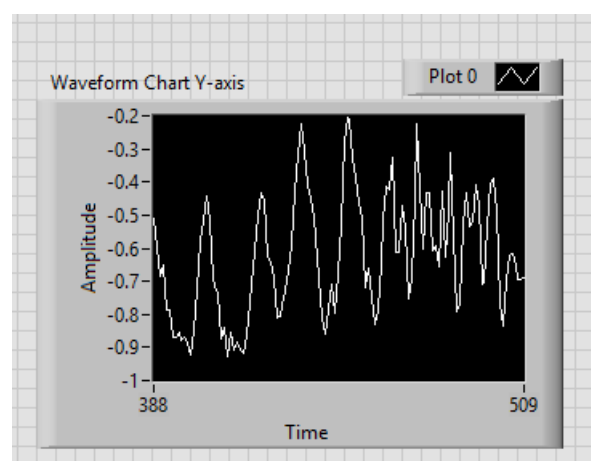
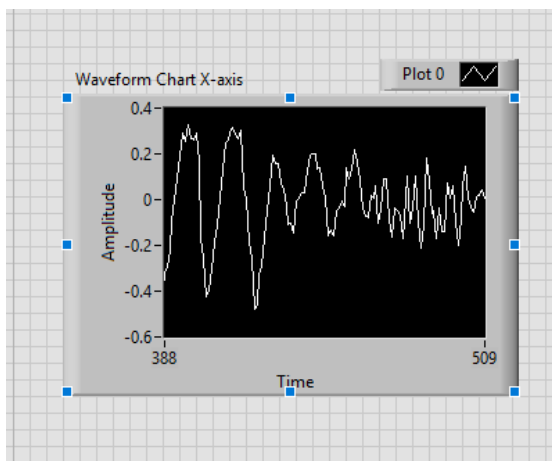


Figure 24 Blinking LED Block Diagram

Assignment #2 (Submitted March 22 – for 3rd week)

Overview of our assignment:

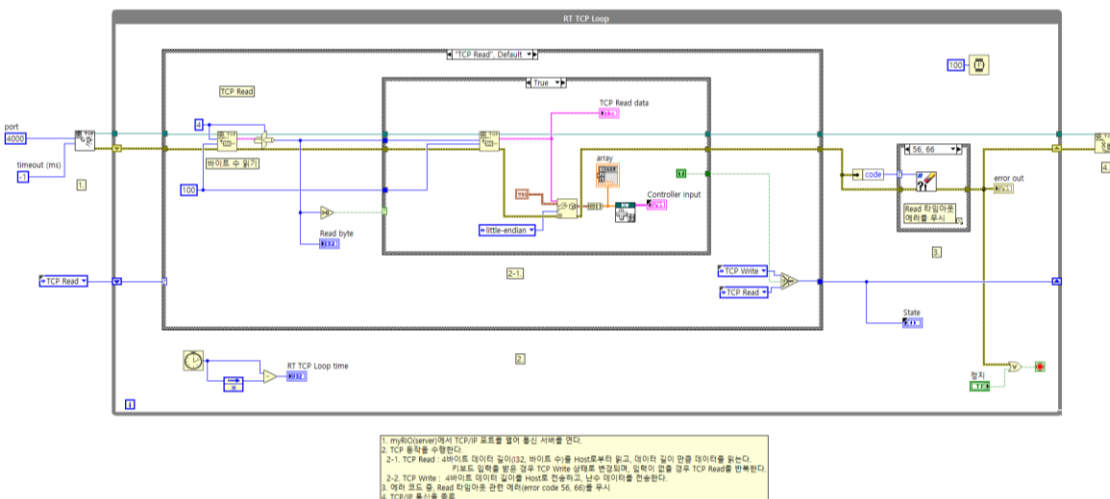
- **LED examples:** We developed some LabVIEW codes to come up with various LED Blinking examples (Ranging from timed blinking to sequential blinking). This assignment helped us to become familiar with the basic functions of LabVIEW programming including Case Structure (Similar to If-Else statement), Wait function, While Loops and also understand the whole programming flow.
- **Accelerometer measurement:** We developed LabVIEW codes to measure the accelerometer data from the in-built accelerometer embedded into myRIO device. We also learnt how to control actuators (in this case myRIO LED) using accelerometer data.



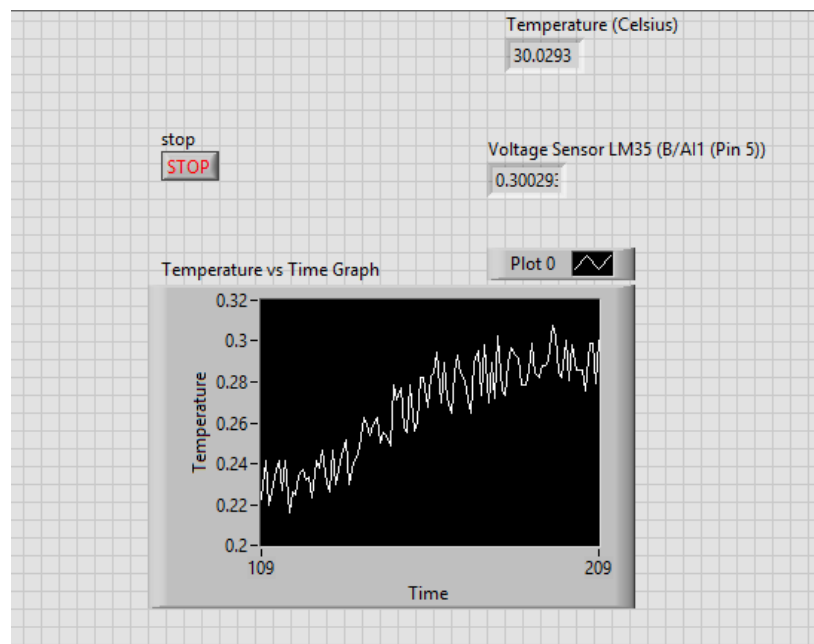
Assignment #3 (To be Submitted April 2 – for 3rd and 4th week)

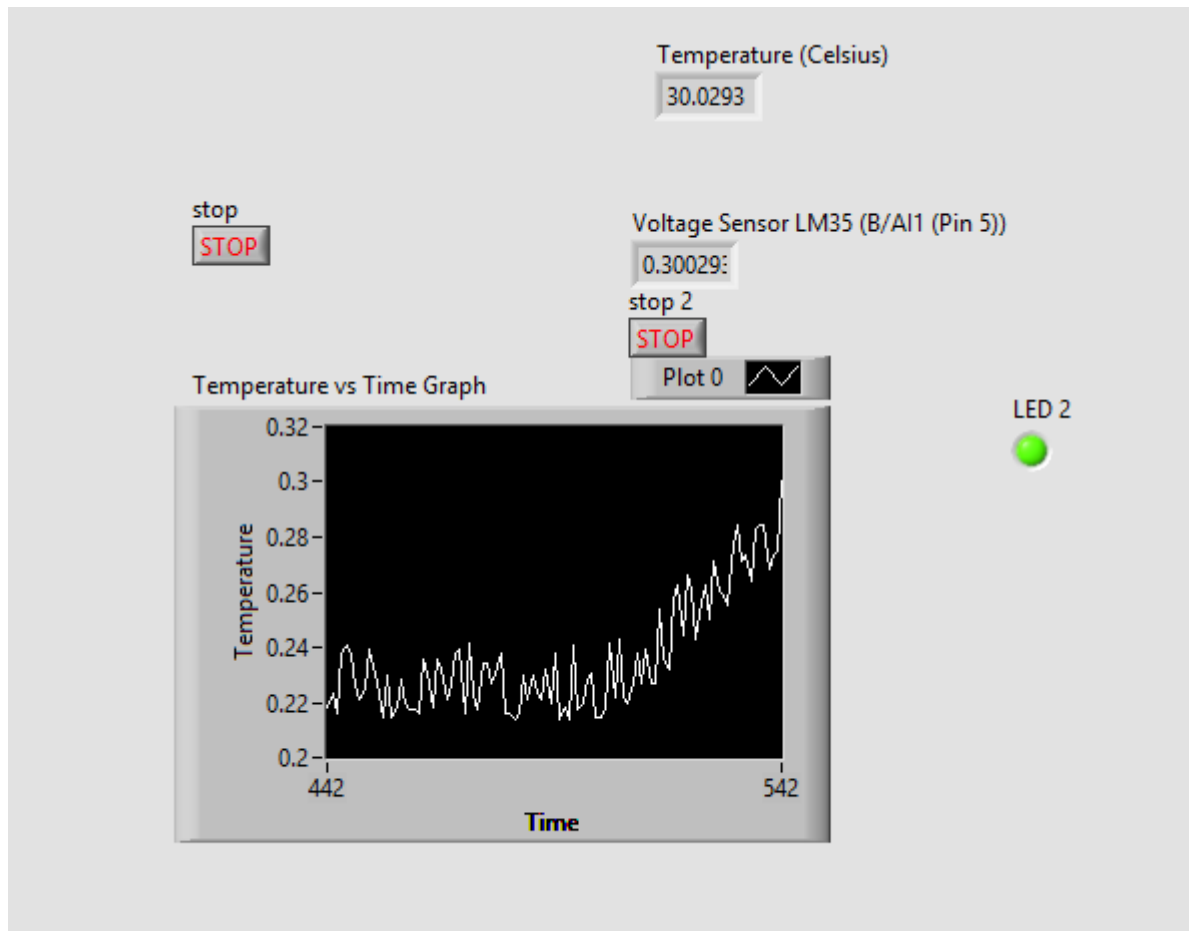
Overview of our assignment:

- **TCP/IP:** We learnt how to implement the TCP (Transfer Control Protocol)/ IP (Internet Protocol) Communication protocols in LabVIEW.



- **Temperature sensor:** We developed LabVIEW codes to measure temperature data from the temperature sensor and also learnt how to control myRIO LEDs using temperature data.





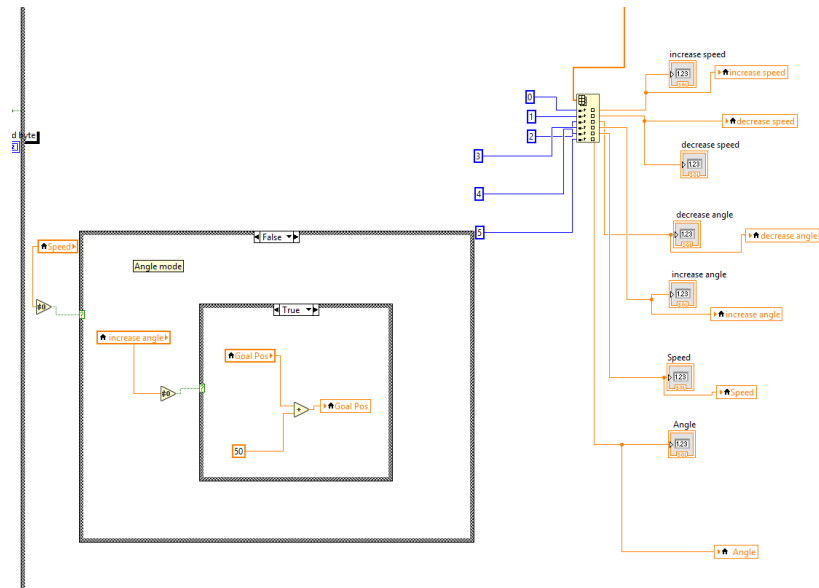
- **DYNAMIXEL (DXL) motor control:** We are currently working on LabVIEW codes to control Dynamixel motor using myRIO (Changing speed and angle of the servo motors) to control the wheels.

4th week (April 2 – April 8)

Overview of Assignment

Motor control: We established motor control using Keyboard and my RIO. We were able to increase/decrease speed of the motor using keyboard and able to control their behavior.

2)-1 Choose mode(wheel mode, position mode) some typical command from PC.



Multiple motor test: We were able to connect the motors together and setup the IDs. Moreover, we connected the wheels to one of the motors to test the behavior:



Figure 25 Motors connected with each other

Following is the snippet of the code used to do so:

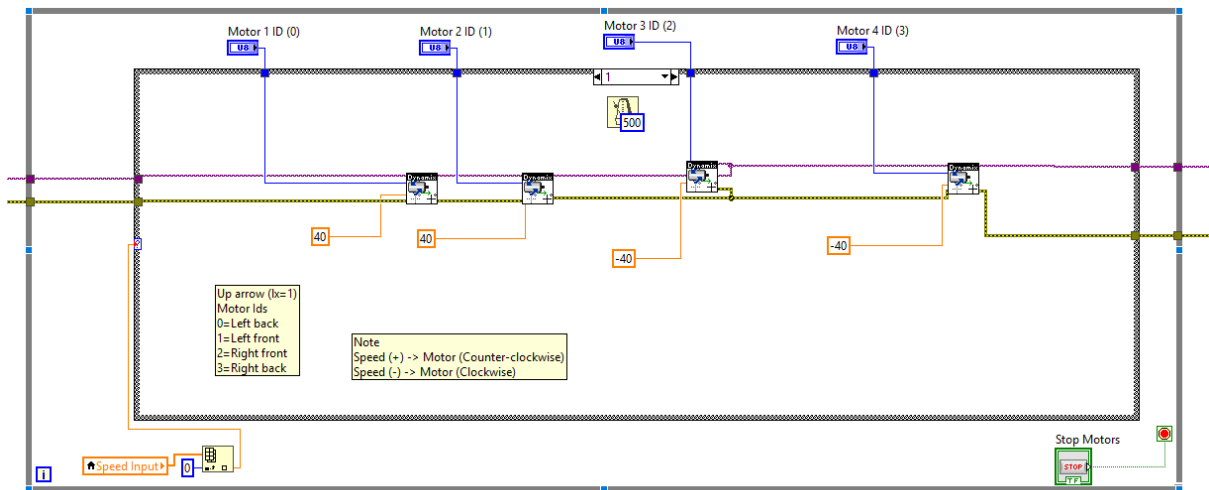


Figure 28 Code Snippet of motor code which checks for the case (up/down/left/right)

As shown in **Figure 7**, the keyboard input is checked with Case structure and based on the case (key pressed up/down/left/right); we can control the motors correspondingly to turn the wheels in a combination of clockwise/counterclockwise.

- **Multiple motor control [Xbox controller]:** After being able to successfully control the motors with keyboard, our next task was to control all the motors using Xbox. We made a code to use 4 keys of Xbox (bup, bdown, bl, br) to rotate the mecanum wheels such that they would correspond to the 4 different directions.

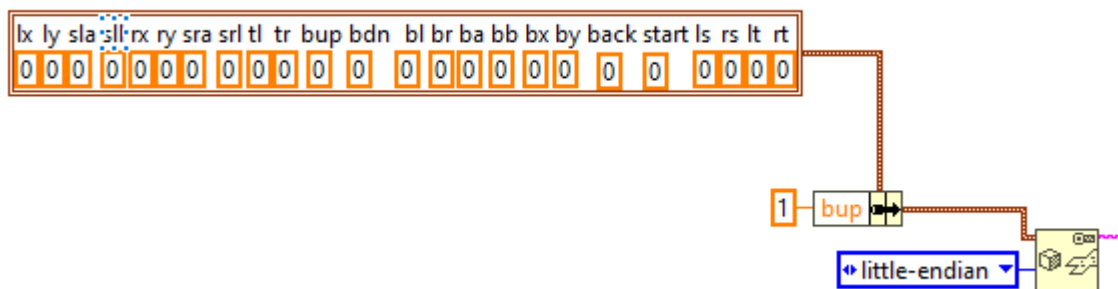


Figure 29 Code Snippet of the keyboard control block [Client Side]

As shown in **Figure 8**, the keyboard control corresponds to changing the corresponding Xbox controller variable to 1 to correspond to the 4 different cases for controlling the motors direction.

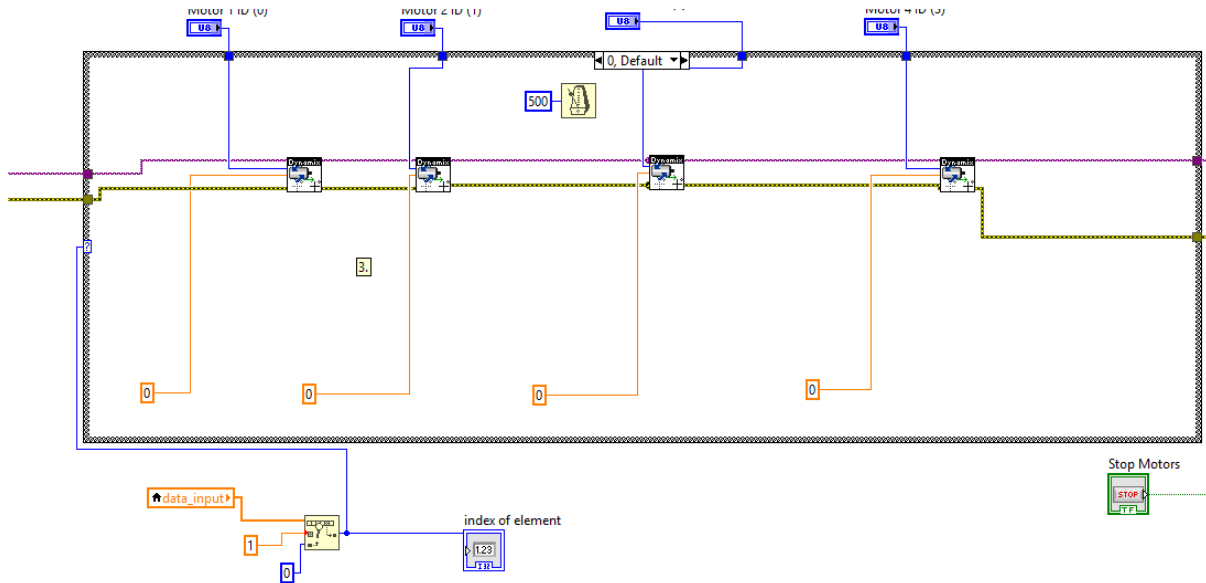


Figure 30 Code Snippet of motor code which checks for the case (based on Xbox Controller buttons)

As shown in **Figure 9**, the Xbox Controller has to be checked with Case structure and based on the case (Xbox up/down/left/right); we can control the motors correspondingly to turn the wheels in a combination of clockwise/counterclockwise.

However, when we tried this code; this did not work as the integration between ROS and myRio had problems due to incompatibility of data types. As suggested by the TA, mapping between Xbox and LabVIEW should be done again. We also need to change the bi-directional code (which we are using now) into one-directional so that the response speed can be increased. Thus, our plan is to be able to control the motors using myRio, ROS and Xbox controller by next week after midterms finish.

6th week (April 16 – April 23)

Overview

- **Multiple motor control for 10 directions [Keyboard]:** We wanted to make a code for all direction movement with the mecanum wheels. Thus, we worked on the code to make 10 direction code possible for the car using keyboard as described below:

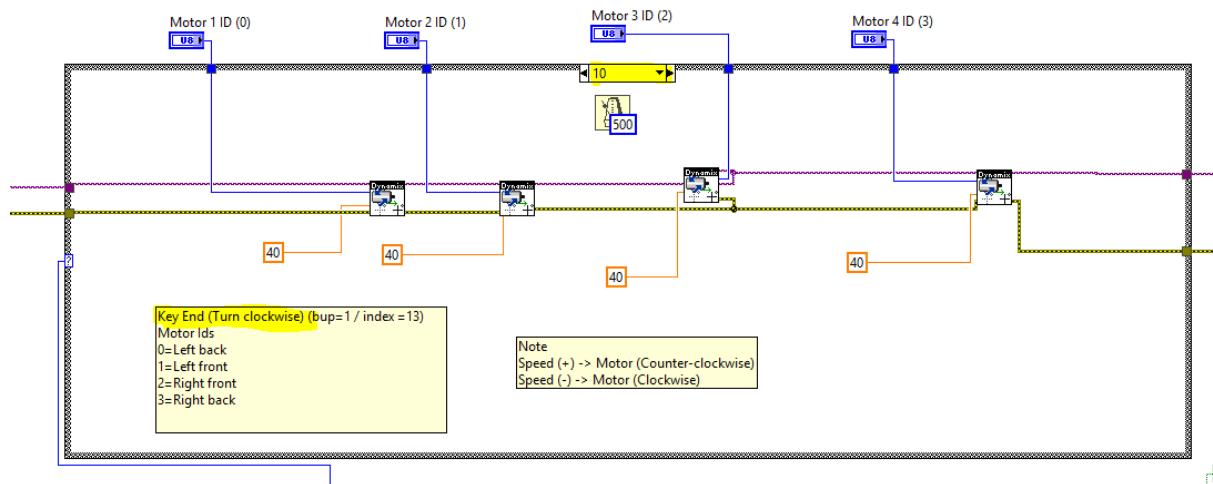


Figure 31 Code Snippet of motor code which checks for the case (based on Keyboard input)/ All ten directions

Following are the inputs for 10 different direction of the motors:

(+) represents Counterclockwise, (–) represents clockwise, (0) represents stopped

Direction	Left Back	Left Front	Right Front	Right Back
Forward	+	+	-	-
Reverse	-	-	+	+
Strafe Left	-	+	+	-
Strafe Right	+	-	-	+
Diagonal right up	+	0	-	0
Diagonal left up	0	+	0	-
Diagonal left down	-	0	+	0
Diagonal right down	0	-	0	+
Counterclockwise	-	-	-	-
Clockwise	+	+	+	+

7th week (April 23 – April 29)

- **Multiple motor control [Xbox controller]:** After being able to successfully control the 10 directions with keyboard, our next task was to control all the motors using Xbox. We were unable to do that

before midterm as the integration between ROS and myRio had problems due to incompatibility of data types. As suggested by the TA, we did the mapping again between XboX and LabVIEWQ and it worked this time.

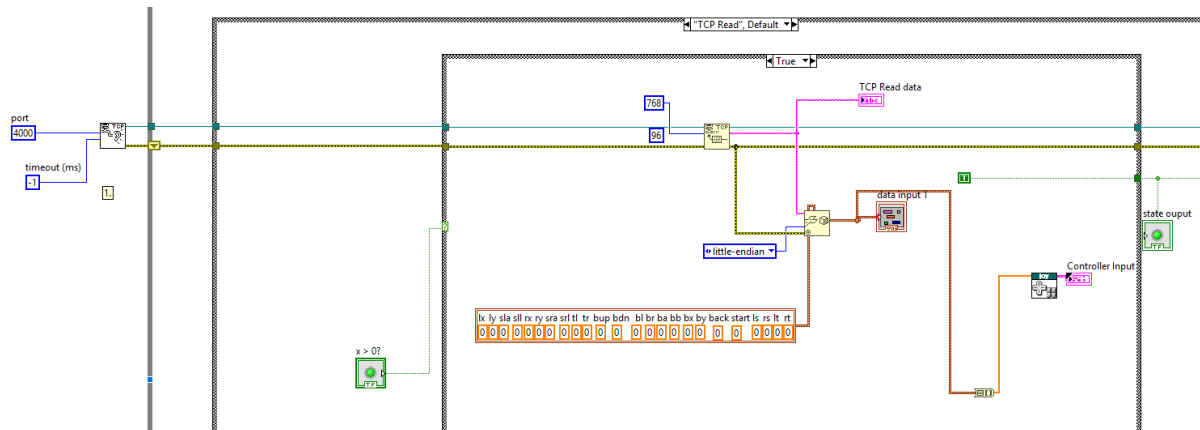


Figure 32 Code Snippet of the successful communication between Xbox, NUC PC and my Rio

After successfully establishing communication; we wrote a code for controlling 10 different directions using X-box keys.

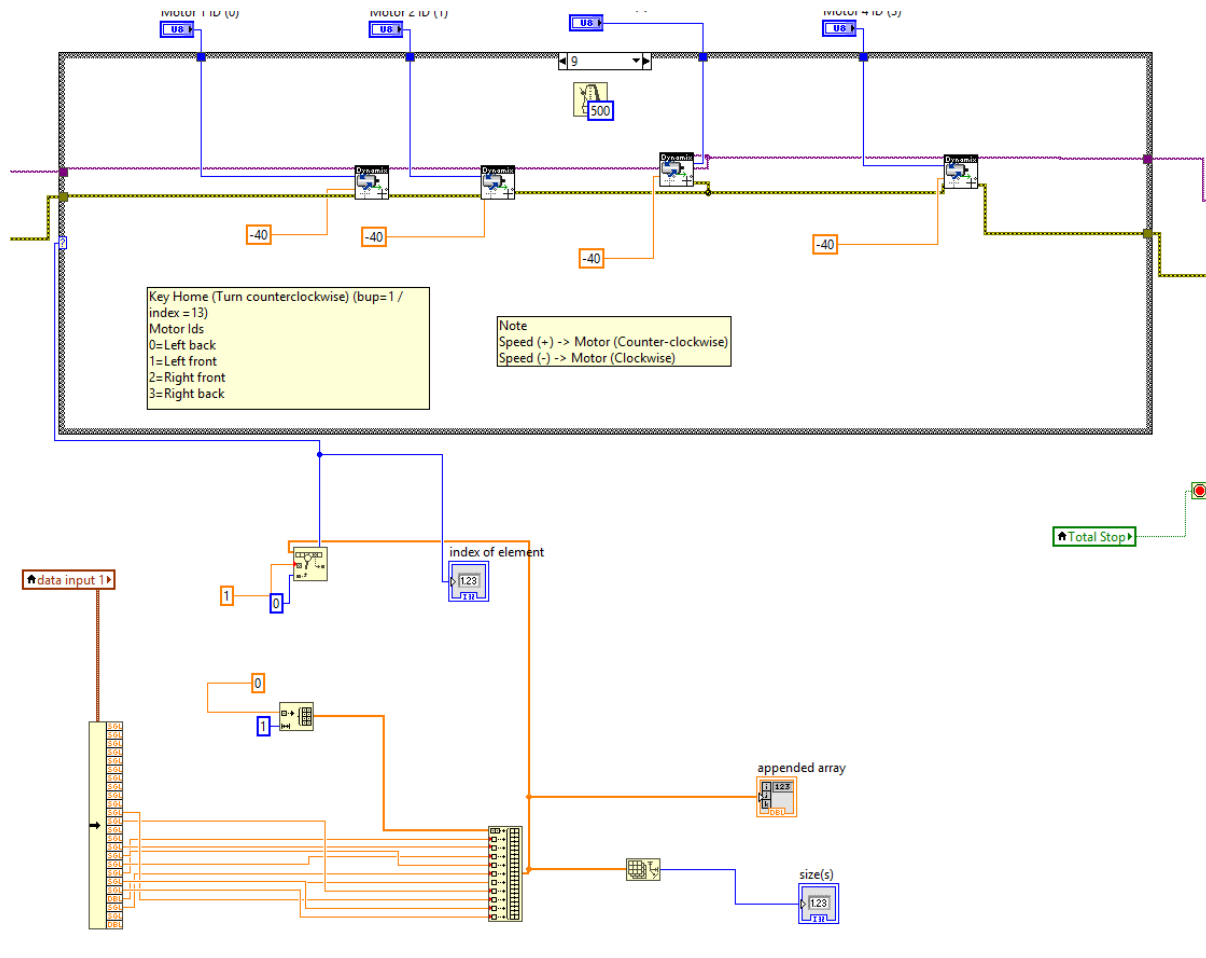


Figure 33 Code Snippet of the successful controlling 10 different directions using Xbox controller

Thus, we were able to control 10 different direction using Xbox controller, NUC PC and myRio. And we have also recorded the video for the presentation.

In the coming weeks; we want to make the code faster and also develop a code for integrating DC motor for actuator control in LabVIEW and also want to add a code for speed increase or decrease.

8th week (April 30 – May 6)

- **Delay removed from code [Xbox controller]:** As we had mentioned earlier; our code had a delay, which led to our car not responding fast enough to our instructions. So, we decided to make the code faster by integrating a single loop.

Earlier, our code for the run included codes for bi-directional TCP/IP communication. Hence, we tried to make the code unidirectional. Also, we added conditions so our car can run at different speeds while executing different directions (forward, reverse or rotation).

- **Making Stand-alone code:** After removing the delay, our next task was to be able to transfer the LabVIEW code to myRIO's flash memory so it can run without needing the LabVIEW program on the computer. TO do that, I followed the myRIO guide to deploy a startup application to myRIO using the Real Time Module as shown in the code snippets below:

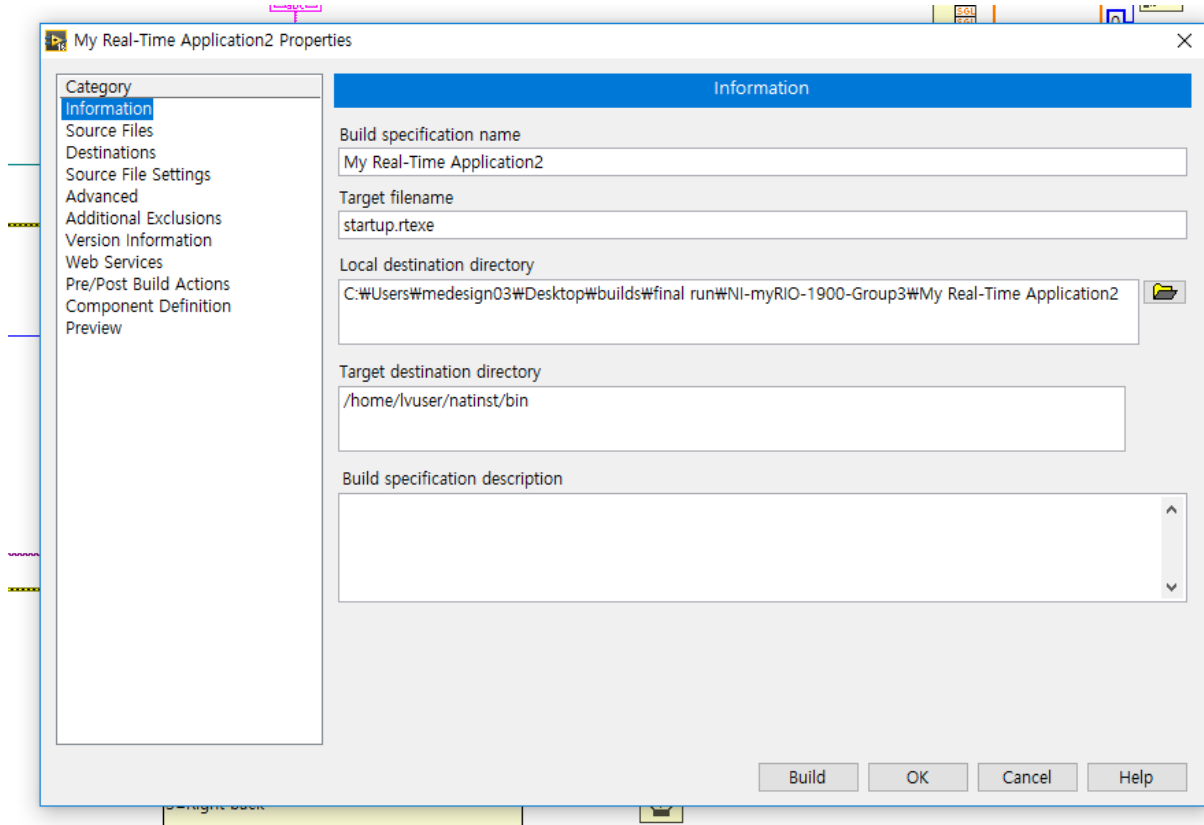


Figure 36 Various options employed to create stand-alone application

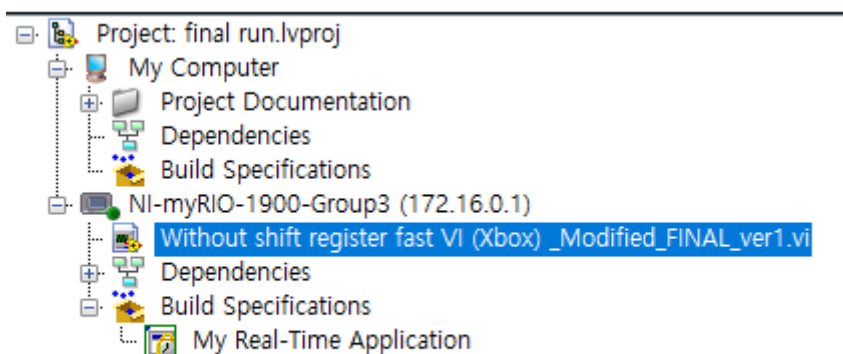


Figure 37 Final step in deploying final real time code

Figure 16 shows the Real Time Application created under the Build Specifications subheading. Thus, after doing this we no longer need to direct the car using LabVIEW being run on the computer. We

still have some error as we cannot determine how to restart my RIO every time ROS runs the code and we also have to figure out any changes requested by the ROS team to make our car run more efficiently for the competition.

10th week (May 14 – May 21)

- **Introducing delay time:** Earlier, we were trying to implement delay i.e. so the car can run for a fixed duration of time in a particular direction for a certain time in ROS code. However, ROS team was having difficulties to implement delays in the code. So, I tried to implement delay in LabVIEW code itself within the switch cases. You can see the implementation in the code snippet below:

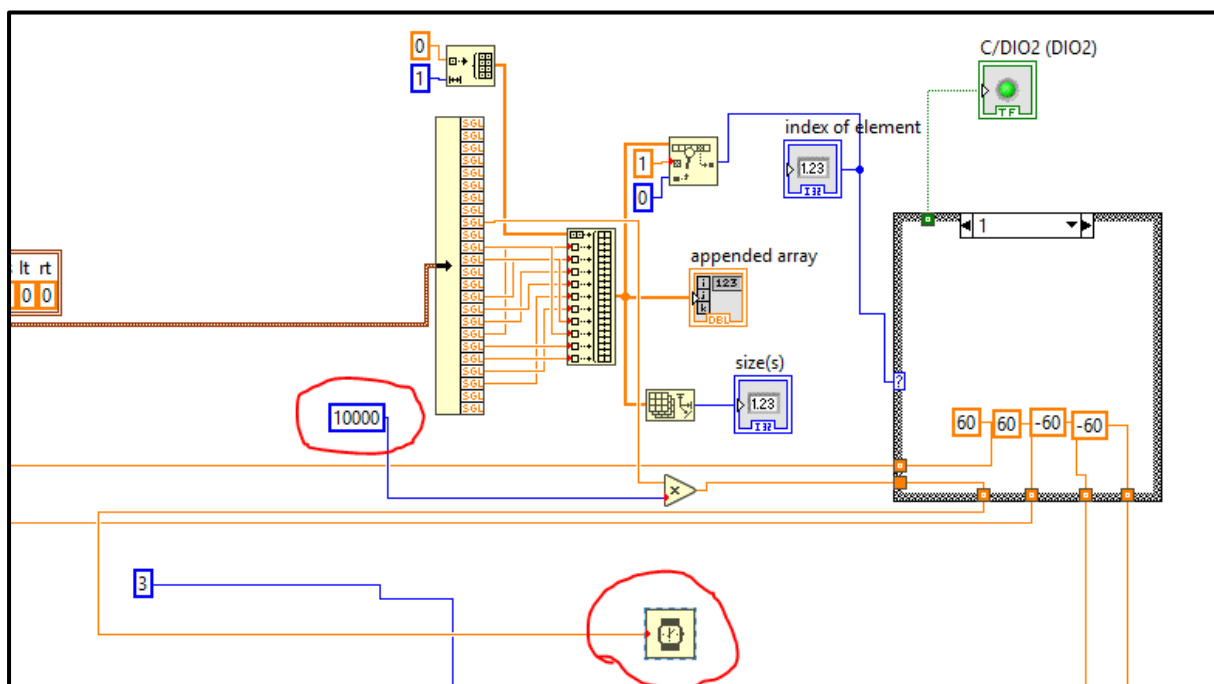


Figure 38 Delay introduced in the code using another variable to control the loop time

Thus, this delay time helped us to run a particular direction of car for a particular time duration (say 10 seconds). However, this was not very efficient as we were still using Xbox controller variables from the beginning and also it was difficult to track the code changes. Thus, we decided to try to change the Xbox variables into the floating point numbers.

11th week (May 22 – May 27)

- **Removing Xbox code to make the code simpler and faster:** Earlier, we were using Xbox controller data format which had large size and used array traversing and searching to find the case. We wanted to make the code simpler and faster and thus, we tried various methods to send data between ROS and myRIO. Finally, we were able to send data using floating point numbers as shown in the snippet below:

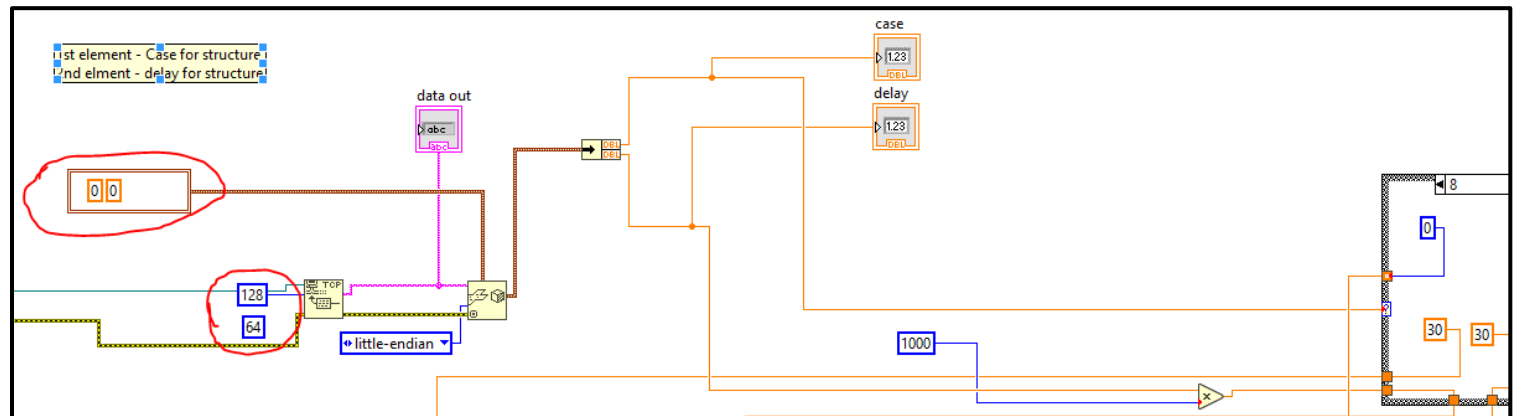


Figure 39 Xbox code modified into simple code for communication between ROS and myRIO

As shown in the figure above, we were able to considerably simplify the code from array of Xbox numbers to simple two variables: cases (dealing with directions) and delay (dealing with the amount of time the car follows a direction). In the first red circle on the top you can see the cluster of two variables case and delay being set to default 0. We use double-precision floating point numbers (each 64 bit). Hence, the bytes to read in TCP/IP communication becomes $64 \times 2 = 128$ bits. This cluster is then unbundled to send one data for direction to determine the case and the second is used to set the delay time for the loop running the motor. Thus, we were able to make the unidirectional communication fast to make the car run properly. Below is our code snippet: clean, simple and easy to debug:

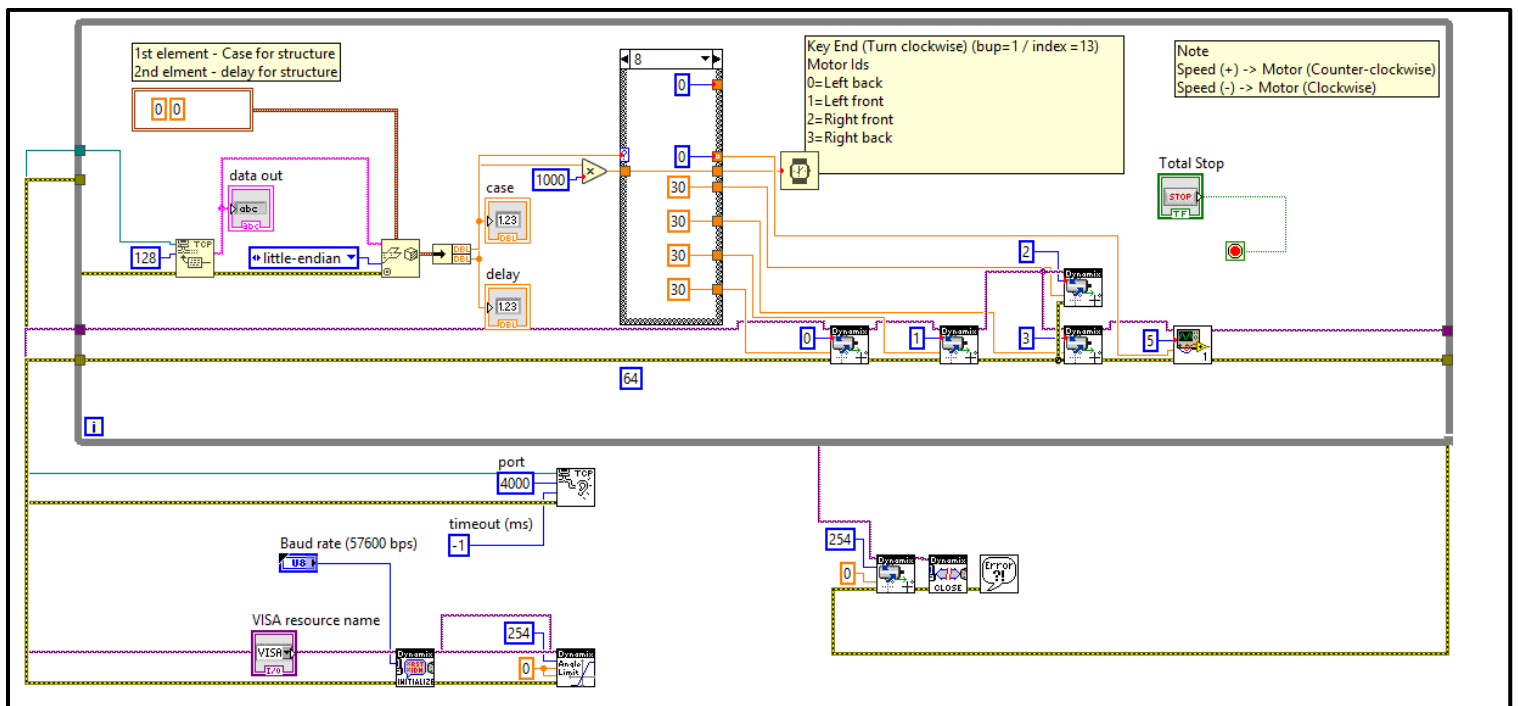


Figure 40 Clean and simple code which is readable, featuring comments and easy to modify/debug

The above figure features the code we wrote to make debugging and modifying simpler for us. It features 4 different wheels for dynamixels and other dynamixel as actuator.

- Adding AX-18 Dynamixel actuator:** For our gripping mechanism, we ordered another Dynamixel Servo (AX-18), however its control was different than the MX models used for wheels. Hence, we had to create a subVI with different parameters for this Dynamixel as shown in the code snippets below:

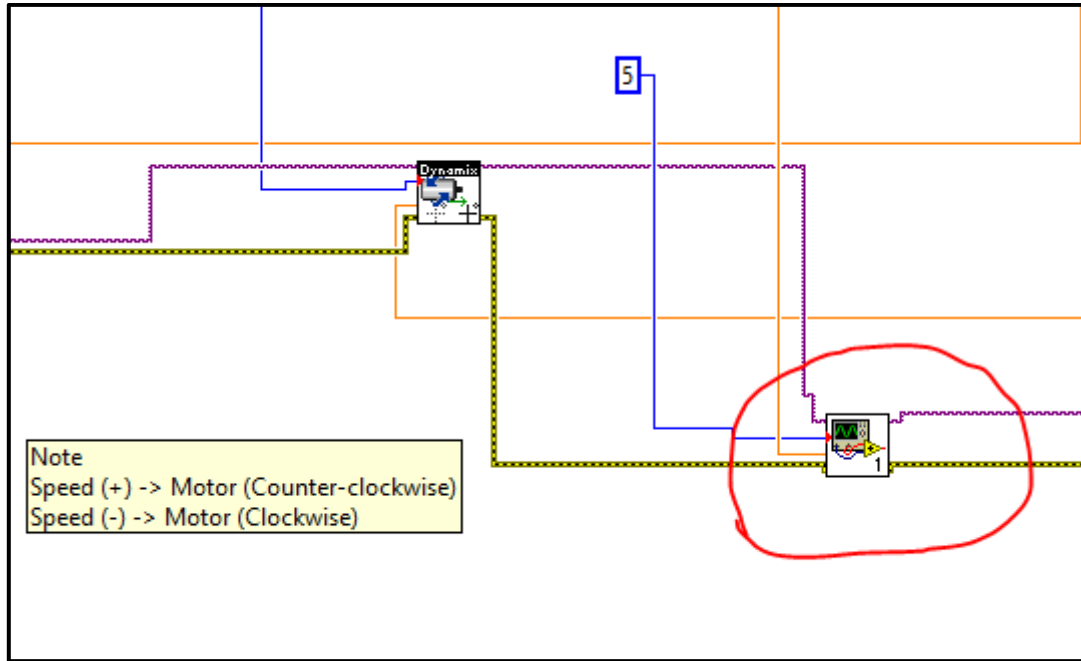


Figure 41 The subVI for the AX-18 Dynamixel actuator for the gripping mechanism

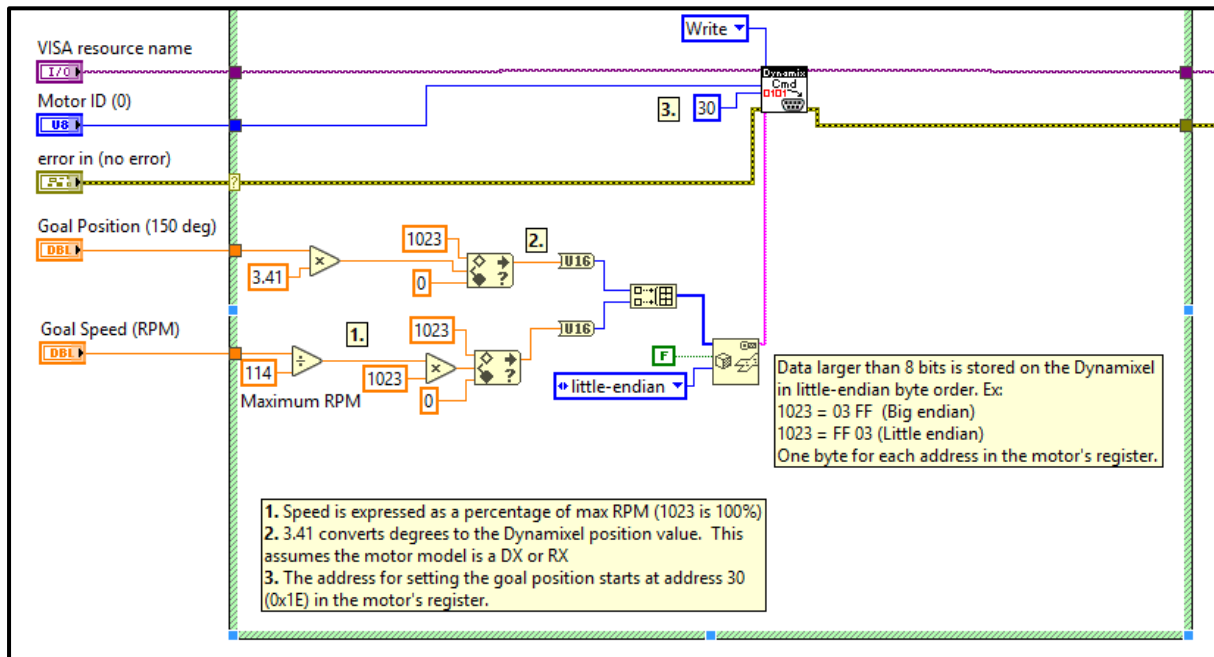


Figure 42 Detailed subVI for the AX-18 actuator; it has different RPM's and settings for this servo

As shown in the figure above, the AX-18 servo used in Dynamixel has different settings for the maximum speed and conversion ratio. Now, we are testing our car and tweaking the codes both on ROS and LabVIEW to effectively collect the balls.