**University of Waterloo**

Faculty of Engineering

LEGO Bluetooth Mario Kart Robot

MTE 100/GENE121

Group 5

Prepared by

Yongxuan Li 20570064

Michael Ru

Kevin Huo

Sachin Apiah

Prepared for

David Lau

Andy Cao

Table of Contents

Acknowledgements

For this project, there exist portions of final software that were created by others. The program used to map Wii remote input data to key presses on the computer was called GlovePIE [1], written by Carl Kenner.  Additionally, C# was also used in the final design of the robot due to its ability of having a built in communications library. The documentation for this library was found on MSDN [2], and the idea of using the SerialPort class from C# as a bluetooth communication framework was from stackoverflow user “absorr” [3]. This forum post also provided the group with the knowledge of bluetooth within RobotC, which was then enhanced through documentation on the RobotC wiki [4].

Additionally, for the second robot which used an EV3 brick instead of the NXT brick, the software to communicate with this was not written in RobotC. This was due to the fact that RobotC does not have the capability of bluetooth communication when running on the EV3 brick. Thus to get this robot working for demo day as well, the group used another library which is able to control the EV3 brick called the MonoBrick library [5].

The knowledge of C# was obtained by Yongxuan and Michael throughout their high school computer science courses. All other software written for this project was within the scope of things learned in GENE 121 by all group members.

Summary

Mario Kart is a fun video game that many people play, but yet real life examples of it are scarce. Using LEGO, a great building platform to create many different projects, this group created a LEGO Mario Kart game using Wii remotes as controllers. This report will explain the process in deciding the design of the vehicle as well as the design of the software to run it.  The main design of this robot was to create a movable motor vehicle controlled through Bluetooth to a Wii Remote a user would hold. This design was thoroughly tested by both creating multiple mechanical designs which were judged against selected constraints and criteria as well as continuously testing and debugging the software. Even though the demo for this project was a success, there still are many components to this design that could still be improved upon. Additionally, this group gained insight about the technicalities of creating a project like this, and the processes associated with it. Therefore, some recommendations for this project in the future include fixing up some mechanical stability as well as sensor placement issues. It would also probably be best if the software was able to be more efficient in the process of reading in Bluetooth.

**1.1 Design Problem Definition**

There are many people who play the popular video game Mario Kart and other racing type games on video game consoles, but real life examples of this are uncommon. Even though there are many RC (remote controlled) cars available, they do not provide the full experience of a video game like Mario Kart (i.e. do not give power ups, cannot shoot others with balls). More so, the number of LEGO robots which provide a racing game experience is even less. Additionally, previous LEGO project examples lack systems which provide a realistic full game experience. Therefore, a problem that needs to be solved is the lack of real world *Mario Kart*-esque experiences which provide entertainment.

**1.2 Goals and Objectives**

The aim of this project is to provide an entertaining and interactive, real-life experience that resembles the mechanics and race-like quality of the video game *Mario Kart*. The final product should satisfy the user as a fun game that incorporates intricate coding and aesthetically pleasing hardware, as well as a suitable racecourse for the user to follow. The basis of this project is to use Bluetooth communication with a Wii remote to control the movements of the car.

**1.3 Constraints**

The constraints involved with the actual robot include that the robot must use one or two touch sensors, one colour sensor, and motor encoders The movement of the robot must correspond to the inputs provided by the user through the Wii remote. It must turn left when remote is tilted to the left (and vice versa), drive forward when forward button is pressed, backwards when backward button is pressed, etc. Additionally, the robot must complete actions according to its environment/ input of sensor. For example, the touch sensor input causes the moving “kart” to become stunned for a couple seconds, and the colour sensor input makes the vehicle gain power ups or slowdowns. It must also be able to withstand multiple bumps and run-throughs without mechanical failure. Each vehicle must be less than 50cm long and 25cm wide in order for both karts to fit comfortably on the track. The project must include elements of the Mario-Kart video game, including: shell-shooting, inclusion of racetrack, power ups, interaction with other karts, Wii-remote controlled movement, and different characters. The player must also interact with the robot, either directly or indirectly. For example, the robot should prompt for user input and allow the user to select different game modes and characters.

**1.4 Criteria**

The robot vehicle should be visually appealing to the user. The overall Gameplay should be consistent (ie. inputs do not produce undesirable results).  Another criteria should be that this robot should be capable of using as many sensors and data as possible. This would allow for more functionality and expanded possibilities for the software. Additionally, any user should be able to easily interact with and understand the robot and game.  The Overall Mario Kart Experience could be enhanced by adding more character modes with different individual characteristics.

In addition, some constraints were decided to be too constraining, and were not able to be judged on a pass/fail scale. Therefore, some additional criteria that were added include: the fluidity of the vehicle should drive smoothly and that the project should provide an exceptional video game experience. This allows for the robot to be completely judged on a pass/fail scale and made it much easier to evaluate our designs.

**2.0 Mechanical Design and Implementation**

2.1 Motor Drive Design

The motors were chosen to be placed close to the front of the vehicle as it would be directly connected to the gear system and the wheels. They were also positioned upside down due to the fact that the bulge of the motor would provide an idea slant for the NXT brick to rest on (Figure 2.1.1).

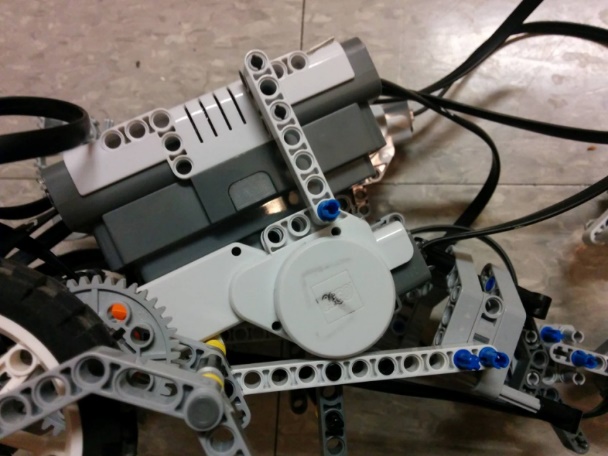


Figure 2.1.1 Motor Drive Design

In positioning the motors this way, it would have became a slight nuisance for the programming if the wheels were to be directly connected to the motor, as “forward” on the software would cause the vehicle to move backwards and vice versa.

Thus an even number of gears were needed on each side such train the gear train would cause the direction to be reverted to normal (“forward” would go forwards).

However, there were limitations on the number of gears which could be used. Too many gears would cause the gear train to take up a lot of space, thus it was decided that 2 gears in a 39:24 gear ratio (Figure 1.2) would be sufficient for the purpose of this vehicle. This gear train uses the largest gear with 39 teeth to be connected to the motor and the gear with a 24 gear ratio to be mounted on the axle of the wheels, allowing every one revolution of the motor to push out 1.625 revolutions of the wheel. A secondary reason why the 39:24 gear ratio was chosen was due to the fact that any other gear with less than 24 teeth would cause slippage between the gears when the motors were at high power.

Figure 2.1.2 Motor + Gear train

2.2 Chassis Design

The design of the chassis needed to be less than 50cm in length and 25cm in width such that 2 vehicles were able to be put on the track within the restricted space.

The configuration of the chassis required the motors close to the front of the vehicle such that the vehicle would be a forward drive robot. This wheel configuration was chosen due to previous prototypes with rear wheel drives and controlling such a vehicle in a constricted track/ obstacle course was very difficult, thus the forward wheel drive design was chosen. A secondary reason for having the forward wheel drive design was due to the fact that the centre of mass for the vehicle was further towards the front; therefore having a forward wheel drive would allow for better traction on the floor as well as increased manoeuvrability. Furthermore, many possible choices and combinations of wheels (Figure 2.2.1) were available to be used on this robot Mario Kart.



Figure 2.2.1 Possible Wheel sizes

Through many tests, the decision was made to choose the wheels with the largest diameter due to the fact that a racing game needed to have vehicles capable of relatively fast velocities. By having such wheels, every revolution of the motor would provide a larger displacement (25.76 cm) due to the large circumference.

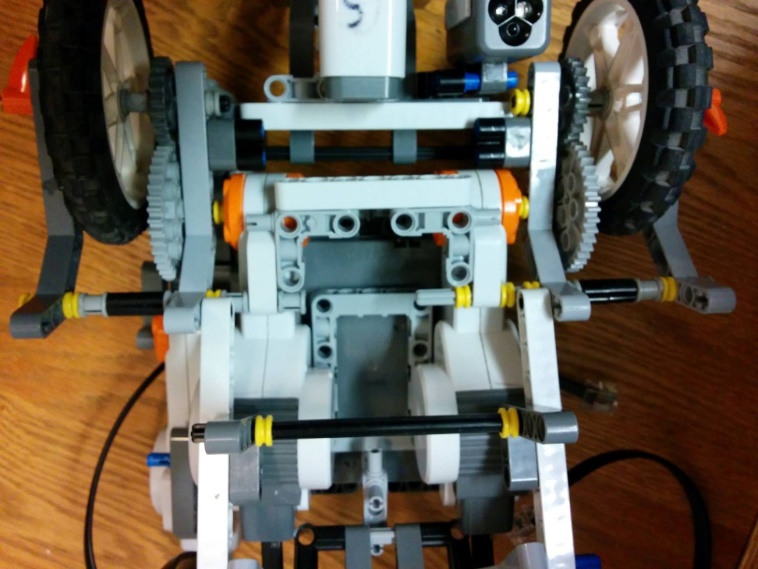
In order to maintain structural integrity while moving and colliding at such speeds, the chassis of the vehicle had multiple horizontal beams (Figure 2.2.2) to keep the rigidity of the vehicle to prevent the robot from falling apart if it were to collide with another object at full speed. Wherever possible, horizontal supports were placed so as to strengthen the vehicle without increasing the length or width of it.

Figure 2.2.2 Bottom View support structure

As the issue of maintaining fluid movement and manoeuvrability arose, the concept of utilizing a free spinning ball seemed like a suitable replacement for a set of back wheels and axle. The free spinning ball was chosen due to the fact that it was not restricted in terms of rotation and it was able to spin all 360° freely. This would also reduce the drag caused by a set of back wheels as they were not connected to the motor thus they were needed to by dragged along the surface of the floor by the front wheels. Having 2 wheels at the back increase the surface area touching the floor and thus increased the amount of friction and drag, which would ultimately led to having a slower vehicle. The free spinning ball, in theory, would provide minimal source of contact between the floor and vehicle, reducing the drag on the front wheels and the motors. Though this idea was theoretically superior, realistically it was difficult to acquire. With limited resources and time, the perfect ball was not able to be found. After testing with multiple free spinning ball holder designs, it was clear that doing a free spinning ball would require too much resources and hinder that progress of the group. Thus, an alternate way was thought of which incorporated some of the concepts of the free spinning ball. Since the vehicle was to be manoeuvred on a tiled surface, it was feasible to replace the ball altogether and just use a Lego support beam (Figure 2.2.3) as the point of contact between the floor and the robot. So long as this point was the only part of the rear section touching the floor, the ability to minimize drag still held true.

Figure 2.2.3 Rear Contact Point

Due to the sheer weight of the NXT brick in comparison to the frame of the vehicle, the NXT brick needed to be placed near the front of the vehicle where the motors and powered wheels were. As well, the NXT brick needed to be slanted such that the user of the game was able to access the buttons and screen of the NXT throughout the main menu and character selection process. These constraints meant that the NXT brick had to be slanted at an angle for optimal viewing and access. In order to do so, the frame itself needed to account for the tilt of the brick. The brick itself could not be placed directed in between the two motors as this would not only cause structural instability but also significantly increase the width of the vehicle which would directly affect the size and location of the track (since two robots were required for the race). The NXT brick was also restricted to being in the front section of the vehicle due to the fact that a forward wheel drive was previously decided to be superior. Therefore, with these constraints kept in mind, the NXT brick was placed directly on top of the motors as the motors were able to provide the optimal angle for the brick to be placed on, due to the build of the oddly shaped motors.

The final design of the vehicle was created with many constraints in mind. Thus the finished product was optimized to perform ideally during demo day. However, there were still many improvements that could have been put in place to further enhance the abilities of the robot vehicle. However many flaws of the design were in place simple because of limited time and the need for the prototype to be finished as quickly as possible. For example, due to the decision made to replace the free spinning ball, it was possible to completely remove the ball holder as it served no real purpose (Figure 2.2.4) and caused the vehicle to be longer and less versatile inside the track.



Figure 2.2.4 Ball Holder

Another problem with the design was the wheels. Although the selection of such large wheels allowed for a larger displacement per revolution of the motor, they were also very thin. This meant that the point of contact between the wheels and the floor was very minimal. Though a minimal point of contact with the floor was favourable for the rear end of the vehicle, this was not ideal for the front wheels as this mean that the traction onto the floors was significantly lowered. This was evident on the demo day when the wheels of the robot would occasionally skid when trying to move proving that there was not enough friction between the wheels and the floor to overcome the inertia of a vehicle at rest. However, such a problem could be easily fix with an added set of wheels onto each side such that there were two wheels on each side instead of one. This would mean that there would be twice the surface area touching the ground and would increase the amount of traction as well.

2.3 Track Design

The racetrack (Figure 2.3.1) was designed in an effort to demonstrate the capabilities of the LEGO kart. The track was composed of wooden boards that formed an L shaped map, and coloured strips of paper that mimicked various power ups and power downs that are found in the video game. The track began with a green strip representing the start line, and then a blue strip representing a speed boost. This speed boost was put in place so that players would not be able to cheat the game and go back and forth over the start line. Even so, the way the code was written prevented a speed boost from being placed behind the finish line, as then the kart would not be able to detect the start line. Only one ballshooter power up (red) was placed on the track, since the game requires the player to complete three laps, and each kart is only supplied with three balls.  At one point in the track, the player was given the choice to maneuver through one of two given paths, each with their own advantages and disadvantages. One path was wider, and therefore more accessible and easier to cross, but featured several yellow strips that could slow the kart down if the player wasn’t careful. The second path had a narrower opening, but included two blue speed boosts that could give the kart a large advantage. This design of allowing the player to choose between two paths was implemented in an effort to make the track more interesting. The planks of wood were taped upright, to minimize the chance of a kart accidentally running over a wall.



Figure 2.3.1 Track Design

2.4 Sensor Attachment Design

The sensors used in the program included touch, colour, and motor encoder. The touch sensors were mounted to the front and back of the vehicle, with free hanging flat surfaces attached providing more surface area for a ball or wall to hit the vehicle. The flat surfaces were designed to be heavy enough for the touch sensor to detect a positive signal, even if the ball only lightly strikes the surface, but light enough to reject signals that weren’t meant to be received. One tradeoff of this limited weight factor meant that the size of the surface was also limited, and so only a small section of the front and back could transmit a touch signal. The colour sensor was positioned in the front of the vehicle in order for the kart to react to a power up immediately when the vehicle passes over a coloured strip.

**3.0 Software Design and Implementation**

The goal of the software was to provide a real time communication link between the Wii remote and the NXT brick. To achieve this the code was split into multiple functions, each of which only perform one single task. The main function had a loop which called each of these tasks in succession. This loop allowed for continuous checking of the bluetooth buffer which is a requirement for accurate communication to work as well as continuously checking all the sensors on the vehicle. One problem with this software design is there was latency in the control of the robot. This stemmed from the fact that the code kept on looping through even when there was no data, and could have been in a different section of the code (e.g. colour sensing) when data was ready to be transmitted to the robot. This latency was minimized by not using any delays in the code other than a short 50 ms delay to debounce the colour sensor, all other timings used timer instead so that the code could continue to run.

The program consisted of the functions (without parameters):

bool initialize() //written by Yongxuan(bluetooth) and Sachin(menu)

char readBluetooth() //written by Yongxuan

void convertInput() //written by Michael

void setMotor() //written by Sachin

void launchBall() //written by Sachin

void checkColour() //written by Michael

void checkTouch() //written by Kevin

void outputResults() //written by Kevin

The initialization phase which consists of setting up bluetooth as well as the menu to select the character will run before the main game loop.  This phase can be tested by displaying a message to the screen when bluetooth connects and the character selection can be tested by using debug mode where the value of the character variable can be seen.

The next phase would consist of the main loop phase where the vehicle is actually moving. This phase would consist of reading in a bluetooth message sent from the computer (which got a signal from a Wii remote). This can be tested by outputting the received data to the NXT screen. The next part would be taking the message and mapping it to motor movement. This can be tested by giving a specific message and making sure that the motors are moving properly. After setting the motors, the brick will then check for input from the sensors. It would check from the touch sensors and the colour sensor. If a specific input is detected then the robot will act accordingly (e.g. a blue colour is detected and the robot would speed up, bump sensor is triggered and the robot would stop moving for a second). These can be tested by manually triggering the input and making sure that the appropriate action is taken. The colour sensor function would also increment the lap number when it the robot passes over green. A ball shooter function will also occur when the user has a shooter power-up and clicks the correct button. This can be tested by making sure that the shooter shoots only when both the conditions are satisfied. After three laps the main loop would stop and move onto the final phase.

The final phase will consist of using timers and encoders which were reset that the beginning to output the time taken as well as an approximate average speed. This phase can be tested by running the robot along a track which has a known distance as well as using a stopwatch. Then some math could be done to calculate the actual data which can then be compared.

The original design was to get the functions to do most of the processing, but in the final code, most of the processing was done by the main function and passed into the function either by reference or by value depending on what the variable was and that allowed all of the functions to be of return type void. This way was chosen because most of the data sent through functions was either timings for various events or incremental values. It made the functions easier because most of the data common to all the functions could just be calculated once, rather than by the functions each time they run. For example, rather than passing in the player to check what the base speed should be, the base speed was calculated in main and then passed into the function.

In order to organize the software, a flow chart was created (Figure 3.0.1)

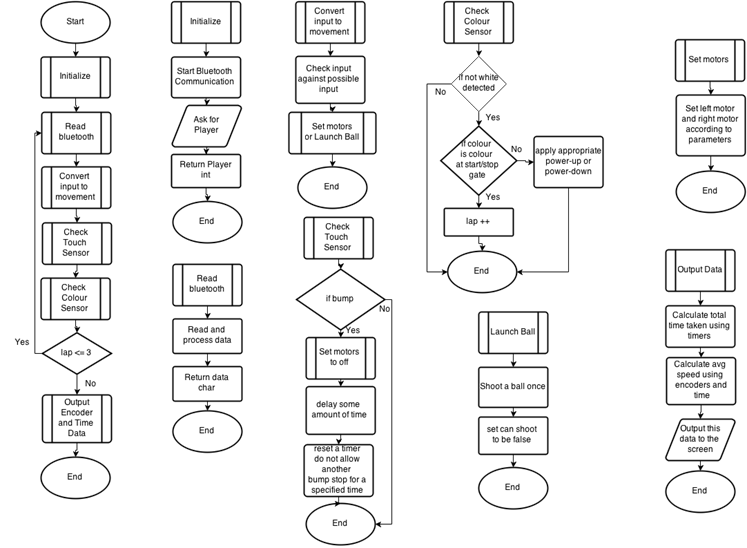
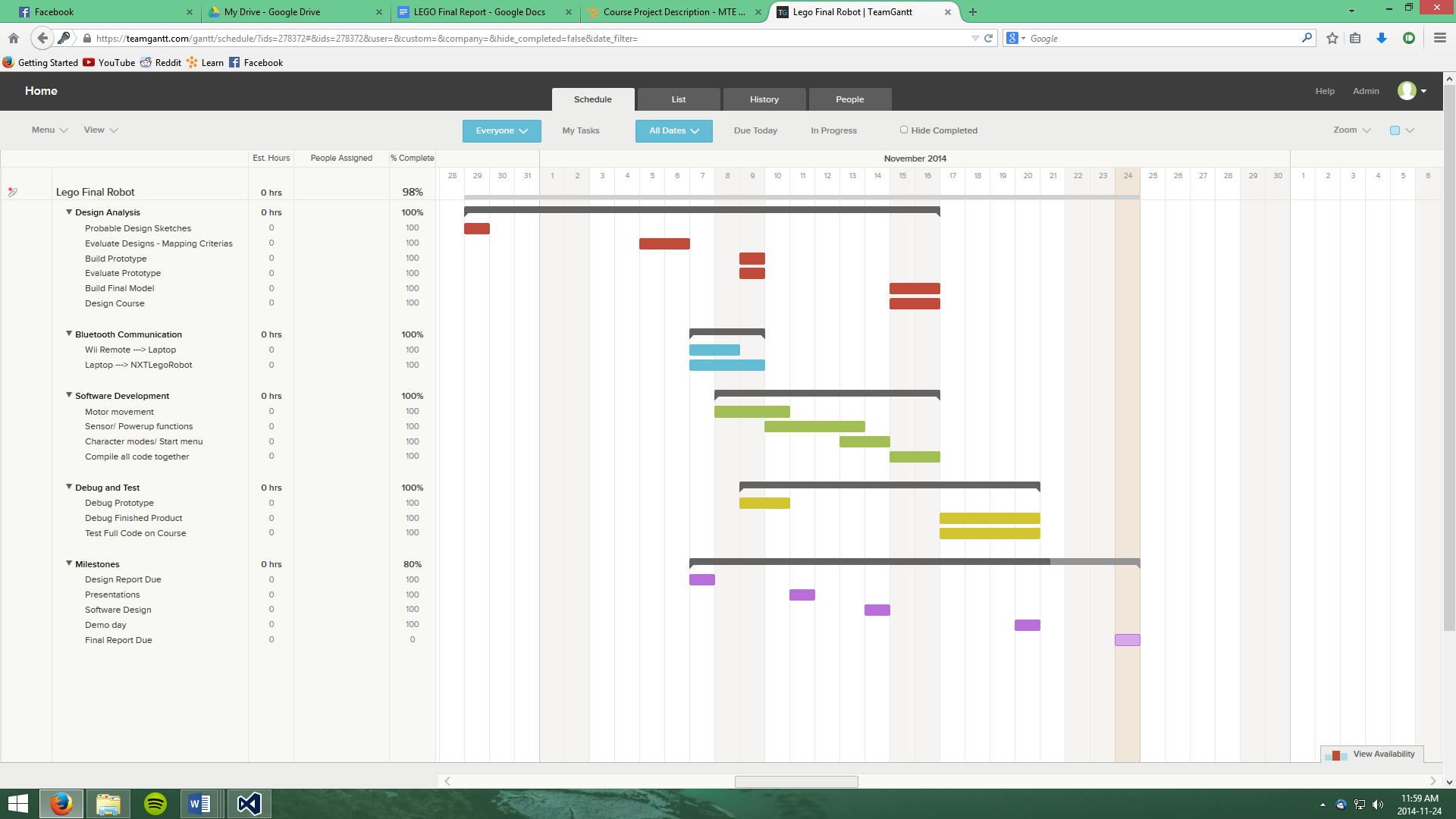


Figure 3.0.1 Flow Chart

**4.0 Project Management**

To organize this project, a Gantt Chart (Figure 4.0.1) was made to plan out when to complete each task. The group was consistent with completing most of the task by the mentioned deadlines apart from a few, of which were recovered from during later stages.



LEGEND

x sa – Design Analysis

\_\_\_\_\_\_ – Bluetooth Communication

\_\_\_\_\_\_ – Software Development

\_\_\_\_\_\_ – Debug and Test

\_\_\_\_\_\_ – Milestones

Figure 4.0.1 Proposed Gantt Chart

To achieve the most efficient use of the three week timeframe, the project was split up into

* Design Analysis by, Kevin, Michael and Sachin
* Bluetooth Communication by, Yongxuan
* Software Development by, the entire team.

The debugging and testing was mostly left for the last two days since we did not complete building the second robot until that time. The added effort required to complete the second robot caused some portions of the project to be completed slower than planned.

**5.0 Conclusion**

Many mechanical and software design decisions were considered for the making of this robot. In order to make the robot as closely related the online video game Mario Kart, the mechanical design had to allow the robot to be able to move corresponding to the inputs of the software. The objective to create a versatile robot fluid in its movement was fulfilled through the mechanical design as the forward wheel drive allowed for turning the front of the vehicle toward the intended direction instantly. The replacement for the free spinning ball allowed for fluidity in turns in any direction, thus it was a success in the sense that it was a suitable replacement. The NXT Brick provided an optimal angle for users/players to view and access the buttons, to enhance the game experience. Therefore, in terms of mechanical design, the robot did meet all of the constraints and criteria, but there are parts of the robot that could still have been improved. In terms of software design, the overall software did fulfill its objective. The robot was able to move with the Wii Remote and correspond to user input. However, the overall efficiency of the program could have still been improved.

**5.1 Recommendations**

The colour sensor was slightly slanted due to flaws in the mechanical design. This slant caused some issues during testing, sometimes rejecting colours and/or not reading the right colours (ie. sometimes reading blue when passing over gray duct tape). This design could have been better implemented if the colour sensor was positioned pointing directly at the ground, to prevent any possible errors. Due to limited time and resources, the colour sensor was only placed on the left side of the front of the robot, preventing a power up of the sensor does not directly pass over a coloured strip. This could be improved by having two colour sensors on either side of the robot. The track used for the project demonstration had several flaws. Some yellow pieces were positioned in such a way that a kart could get stuck in a loop of driving slowly forward, bumping into a wall, sensing the the yellow paper again, and repeating the loop. This can be improved by cutting thinner strips of paper that are positioned away from the walls so that such an infinite loop would not occur.

The software of the project can be implemented more efficiently. During the presentation day, the program ran repeatedly through the loop even though it was not performing any actions when there was no signal from the Wii remote. This can be changed so that the program only runs through certain function calls when a command is actually received from the player. Previously the program only featured a small amount of Mario-Kart components, like bananas, speed boosts, and ball shooting capabilities. The game can be improved by including more elements of the video game (eg. drifting, dropping bananas, random item blocks, more varying items, ramps).

References

[1] C. Kenner. (2010). GlovePIE [Online]. Available: <http://glovepie.org/>

[2] MSDN. (2014). SerialPort Class [Online]. Available: <http://msdn.microsoft.com/en-us/library/system.io.ports.serialport%28v=vs.110%29.aspx>

[3] Stackoverflow. (2014, Jan 30). Communicating with NXT by Bluetooth C# to RobotC [Online]. Available: <https://stackoverflow.com/questions/21457577/communicating-with-nxt-by-bluetooth-c-sharp-to-robotc>

[4] RobotC. (2012, June 5). NXT Bluetooth Overview [Online]. Available: <http://www.robotc.net/wiki/NXT_Bluetooth_Overview>

[5] Monobrick. (2014). Monobrick Communication Library [Online]. Available: <http://www.monobrick.dk/software/monobrick/>

Appendix

A - Source Code

/\*

LEGO Bluetooth Mario Kart Robot

Written by: Yongxuan Li, Michael Ru, Kevin Huo, Sachin Apiah

Written in RobotC

Group 5

MTE 100 and GENE 121

\*/

//prototypes

bool initialize(int & player);

void displayPlayer(int p);

char readBluetooth();

void convertInput(char input, int speed, bool & canShoot);

void setMotor(int mA, int mC);

void launchBall();

void checkColour(int & lap, bool & canShoot, int boostTime, int slowTime,

int & blue, int & yellow, int & red, int & lastValue);

void checkBump(int & numColl, int stunTime);

void outputScore(float dis, float time, float speed, int numColl, int b, int y, int r);

bool initialize(int & player) //Yongxuan and Sachin

{

setBluetoothRawDataMode(); //change bluetooth to read the raw data sent

while (!bBTRawMode) //loop while the mode has not changed

{

wait1Msec(1);

}

if (nBTCurrentStreamIndex >= 0)

//check that the bluetooth connection has occured

{

//menu stuff

nxtDisplayCenteredTextLine(3, "Insert Coin");

while (nNxtButtonPressed == -1);

while (nNxtButtonPressed != -1);

eraseDisplay();

do

{

if (nNxtButtonPressed == 2 && player > 0)

{

eraseDisplay();

player--;

}

else if (nNxtButtonPressed == 1 && player < 2)

{

eraseDisplay();

player++;

}

while (nNxtButtonPressed != -1);

displayPlayer(player);

} while (nNxtButtonPressed != 3);

while (nNxtButtonPressed != -1);

eraseDisplay();

return true;

}

else

return false;

}

/\*

Players:

0-Peach: Increased boost time, decreased banana slow time

1-Mario: Increased base speed

2-Bowser: Decreased bump stun time

\*/

void displayPlayer(int p) //Yongxuan

{

if (p == 0) //peach

{

nxtDisplayCenteredTextLine(0, "PEACH");

nxtDisplayCenteredTextLine(2, "+ mushroom boost");

nxtDisplayCenteredTextLine(3, "- banana slow");

///right arrow

nxtDrawLine(90, 20, 75, 20);

nxtDrawLine(90, 20, 85, 25);

nxtDrawLine(90, 20, 85, 15);

}

else if (p == 1) //mario

{

nxtDisplayCenteredTextLine(0, "MARIO");

nxtDisplayCenteredTextLine(2, "+ base speed");

//left arrow

nxtDrawLine(10, 20, 25, 20);

nxtDrawLine(10, 20, 15, 25);

nxtDrawLine(10, 20, 15, 15);

///right arrow

nxtDrawLine(90, 20, 75, 20);

nxtDrawLine(90, 20, 85, 25);

nxtDrawLine(90, 20, 85, 15);

}

else

{

nxtDisplayCenteredTextLine(0, "BOWSER");

nxtDisplayCenteredTextLine(2, "- stun time");

//left arrow

nxtDrawLine(10, 20, 25, 20);

nxtDrawLine(10, 20, 15, 25);

nxtDrawLine(10, 20, 15, 15);

}

}

char readBluetooth()// Yongxuan

{

ubyte receive[1]; //setup a buffer array to read data

nxtReadRawBluetooth(receive, 1); //read into the buffer

return (char)receive[0]; //return the sent character

}

void convertInput(char input, int speed, bool & canShoot) // Michael and Yongxuan

{

nxtDisplayString(0,"\*\*\*%c\*\*\*",input); //for debugging purposes

switch(input) //set the proper output

{

case 'z':

setMotor(0, 0);

break;

//left turns forward

case 'q':

setMotor(speed-40,speed +40);

break;

case 'w':

setMotor(speed-30,speed +30);

break;

case 'e':

setMotor(speed-20,speed+20);

break;

case 'r':

setMotor(speed-10,speed +10);

break;

//straight forwards

case 't':

setMotor(speed,speed);

break;

//right turns forward

case 'y':

setMotor(speed+10,speed-10);

break;

case 'u':

setMotor(speed+20,speed-20);

break;

case 'i':

setMotor(speed+30,speed-30);

break;

case 'o':

setMotor(speed+40,speed-40);

break;

//left backwards

case 'a':

setMotor(-(speed-40),-(speed +40));

break;

case 's':

setMotor(-(speed-30),-(speed +30));

break;

case 'd':

setMotor(-(speed-20),-(speed+20));

break;

case 'f':

setMotor(-(speed -10),-(speed +10));

break;

//straight back

case 'g':

setMotor(-speed, -speed);

break;

//right back

case 'h':

setMotor(-(speed+10),-(speed -10));

break;

case 'j':

setMotor(-(speed+20),-(speed -20));

break;

case 'k':

setMotor(-(speed+30),-(speed-30));

break;

case 'l':

setMotor(-(speed+40),-(speed -40));

break;

//ball shooting

case 'p':

if (canShoot)

{

launchBall();

canShoot = false;

}

break;

}

}

void setMotor(int mA, int mC) //Sachin

{

motor[motorA] = mA; //Left Motor

motor[motorC] = mC; //Right Motor

}

void launchBall() //Sachin

{

nMotorEncoder[motorB] = 0;

motor[motorB] = 50;

while (nMotorEncoder[motorB] < 350); //shooting a ball every revolution

//used 350 becuase during testing, a full rotation actually will actually turn //more due to timing delays

motor[motorB] = 0;

}

void checkColour(int & lap, bool &canShoot, int boostTime, int slowTime,

int & blue, int & yellow, int & red, int & lastValue) //Michael

{

int senValue = SensorValue[S1];

if (senValue != lastValue)

//proceed if the current colour is different from the last colour

{

lastValue = SensorValue[S1];

if (senValue != 1 && senValue != 6)

//proceed if the colour is not black or white

{

wait1Msec(50);

if (SensorValue[S1] == senValue) //debounce the colour sensor

{

if (senValue == 2) //blue - speedboost

{

nxtDisplayString(2, "BLUE");

blue++;

motor[motorA] = 100;

motor[motorC] = 100;

PlaySoundFile("speedup.rso");

wait1Msec(boostTime);

motor[motorA] = 0;

motor[motorC] = 0;

}

else if (SensorValue[S1] == 3) //green - finish line

{

lap += 1;

nxtDisplayString(3, "GREEN");

}

else if (SensorValue[S1] == 4) //yellow - banana

{

nxtDisplayString(4, "YELLOW");

yellow++;

motor[motorA] = 20;

motor[motorC] = 20;

PlaySoundFile("banana.rso");

wait1Msec(slowTime);

motor[motorA] = 0;

motor[motorC] = 0;

}

else //red - ballshooter

{

canShoot = true;

nxtDisplayString(5, "RED");

red++;

PlaySoundFile("! Attention.rso");

while (bSoundActive);

}

eraseDisplay();

}

}

}

}

void checkBump(int & numColl, int stunTime) //Kevin

{

if (time1[T4] > 2000) //give a grace period of 2 seconds after a bump

if (SensorValue[S3] == 1 || SensorValue[S4] == 1)

{

motor[motorA] = 0;

motor[motorC] = 0;

wait1Msec(stunTime);

time1[T4] = 0;

numColl++;

}

}

void outputScore(float dis, float time, float speed, int numColl, int b, int y, int r) // Kevin

{

PlaySoundFile("! Attention.rso");

while(bSoundActive);

nxtDisplayString(0, "Crashes: %d", numColl);

wait10Msec(50);

PlaySoundFile("! Attention.rso");

while(bSoundActive);

nxtDisplayString(1, "Bananas: %d", y);

wait10Msec(50);

PlaySoundFile("! Attention.rso");

while(bSoundActive);

nxtDisplayString(2, "Mushrooms: %d", b);

wait10Msec(50);

PlaySoundFile("! Attention.rso");

while(bSoundActive);

nxtDisplayString(3, "Green Shells: %d", r);

wait10Msec(50);

nxtDisplayString(7, "PRESS BUTTON...");

while (nNxtButtonPressed == -1);

while (nNxtButtonPressed != -1);

eraseDisplay();

PlaySoundFile("! Attention.rso");

while(bSoundActive);

nxtDisplayString(0, "Time Taken:");

nxtDisplayString(1, "%.2f s", time);

wait10Msec(50);

PlaySoundFile("! Attention.rso");

while(bSoundActive);

nxtDisplayString(2, "Total Distance:");

nxtDisplayString(3, "%.2f cm", dis);

wait10Msec(50);

PlaySoundFile("! Attention.rso");

while(bSoundActive);

nxtDisplayString(5,"Average Speed:");

nxtDisplayString(6,"%.2f cm/s",speed);

wait10Msec(50);

nxtDisplayString(7, "PRESS BUTTON...");

while (nNxtButtonPressed == -1);

while (nNxtButtonPressed != -1);

eraseDisplay();

nxtDisplayCenteredBigTextLine(4,"TALLYING SCORE...");

PlaySoundFile("! Startup.rso");

while(bSoundActive);

eraseDisplay();

nxtDisplayCenteredTextLine(1,"TOTAL SCORE:");

nxtDisplayCenteredBigTextLine(3,"%.2f", speed + b + r - y - numColl);

nxtDisplayCenteredTextLine(6, "THANKS 4 PLAYING");

nxtDisplayCenteredTextLine(7, "Group5Games");

while(nNxtButtonPressed == -1);

}

/\*

Players:

0-Peach: Increased boost time, decreased banana slow time

1-Mario: Increased base speed

2-Bowser: Decreased bump stun time

\*/

task main()

{

//initialize variables for game

int blue = 0, yellow = 0, red = 0, numColl = 0, ;

//keep track of how many colours and collisions hit

int lap = 1, player = 1;//default to Mario

int bananaSlow = 1500, speedBoost = 1000, stunTime = 1500;

//timings for special events

int defaultSpeed = 40;

int lastColour = 1; //last colour initialized at black

float totDis = 0, totTime = 0;

float avgSpeed = 0;

bool canShoot = false, canStun = true;

SensorType[S1] = sensorCOLORFULL;

SensorType[S3] = sensorTouch;

SensorType[S4] = sensorTouch;

nMotorEncoder[motorA] = 0;

nMotorEncoder[motorB] = 0;

time1[T1] = 0; //for total race length

time1[T4] = 0; //for bump

if (initialize(player))

{

//setup each player's abilities

if (player == 0)//peach

{

bananaSlow -= 500;

speedBoost += 500;

}

else if (player == 1)//mario

defaultSpeed += 10;

else //bowser

stunTime -= 500;

nxtDisplayCenteredBigTextLine(3, "3");

PlaySoundFile("! Click.rso");

wait1Msec(1000);

nxtDisplayCenteredBigTextLine(3, "2");

PlaySoundFile("! Click.rso");

wait1Msec(1000);

nxtDisplayCenteredBigTextLine(3, "1");

PlaySoundFile("! Click.rso");

wait1Msec(1000);

nxtDisplayCenteredBigTextLine(3, "Go");

PlaySoundFile("! Attention.rso");

eraseDisplay();

time1[T1] = 0;

do

{

if (time10[T1] > 32000)

//make sure the timer doesn't exceed bounds

{

totTime += time10[T1] / 100.0;

time10[T1] = 0;

}

if (nMotorEncoder[motorA] > 32000)

//make sure encoder doesn't exceed bounds

{

totDis += (42.37 / 360.0) \* nMotorEncoder[motorA];

nMotorEncoder[motorA] = 0;

}

//game loop

convertInput(readBluetooth(), defaultSpeed, canShoot);

checkColour(lap, canShoot, speedBoost, bananaSlow, blue, yellow,   
red, lastColour);

checkBump(numColl, stunTime);

nxtDisplayString(7, "lap %d", lap);

} while (lap <= 3); //loop for 3 laps in game

setMotor(0, 0);

//total up the data

totTime += time10[T1] / 100.0;

totDis += (42.37 / 360.0) \* nMotorEncoder[motorA];

//42.37 is the distance travelled per 1 rotation

//calculated using gear ratio and wheel radius

avgSpeed = totDis / totTime;

nxtDisplayString(0, "Press Button!");

while (nNxtButtonPressed == -1);

while (nNxtButtonPressed != -1);

eraseDisplay();

outputScore(totDis, totTime, avgSpeed, numColl, blue, yellow, red);

wait10Msec(100);

}

//if there is no bluetooth connection

else

{

nxtDisplayString(4, "No Connection");

PlaySoundFile("Woops.rso");

while (bSoundActive);

while (nNxtButtonPressed == -1);

while (nNxtButtonPressed != -1);

}

}

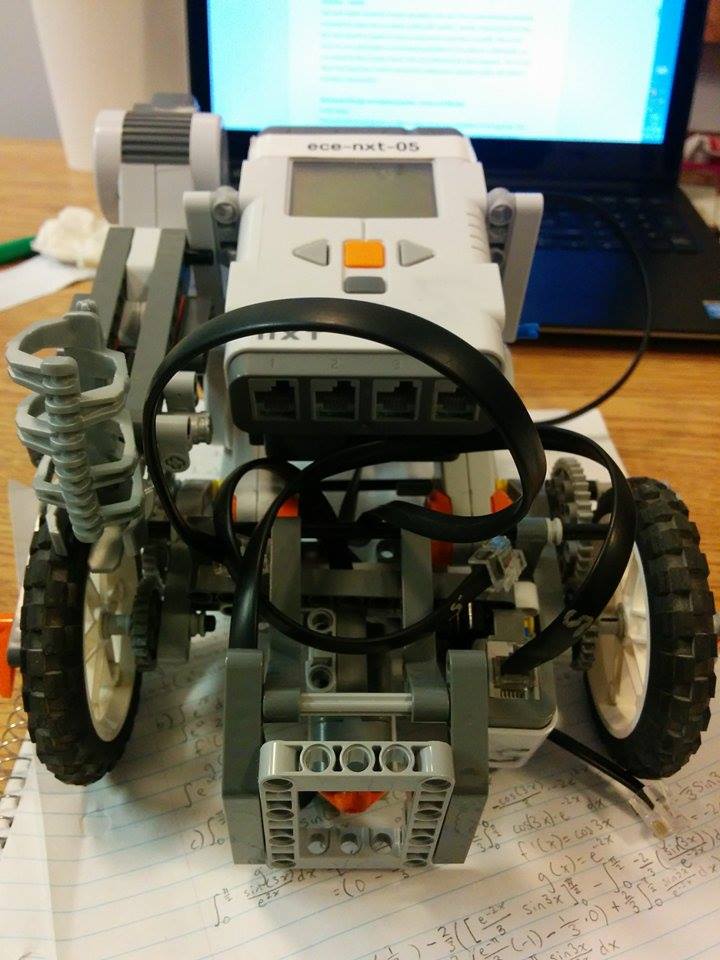
B - Robot Photos

Figure 2.1 Front View

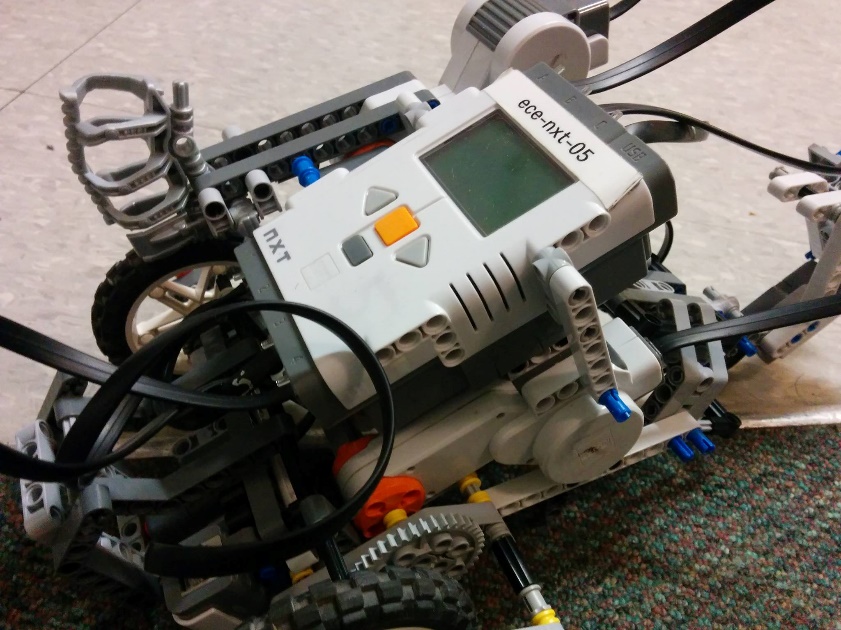


Figure 2.2 Angle View

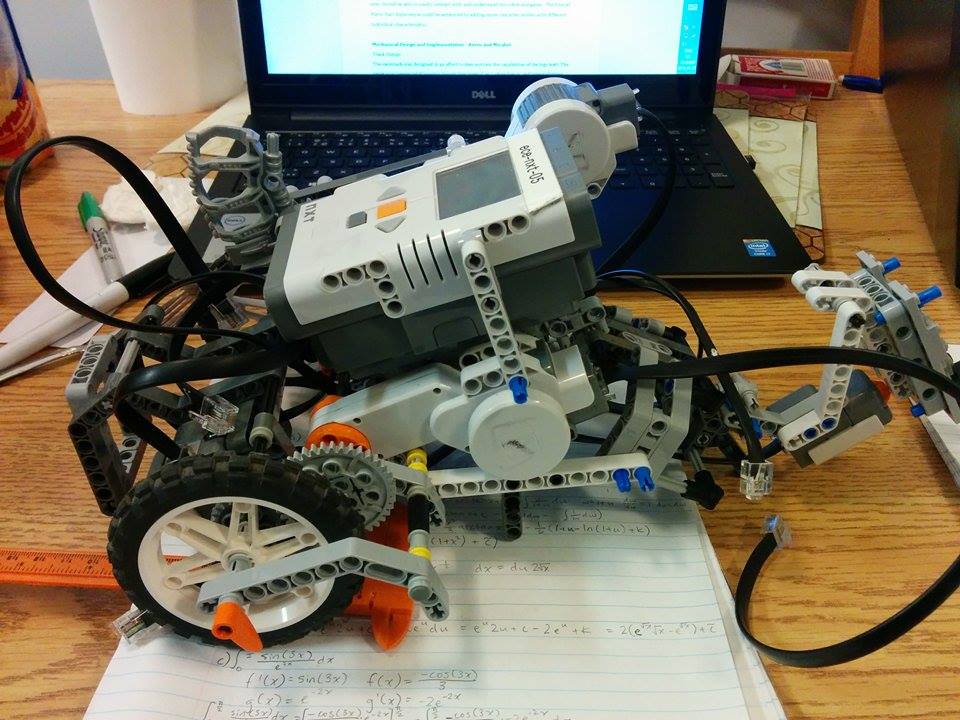
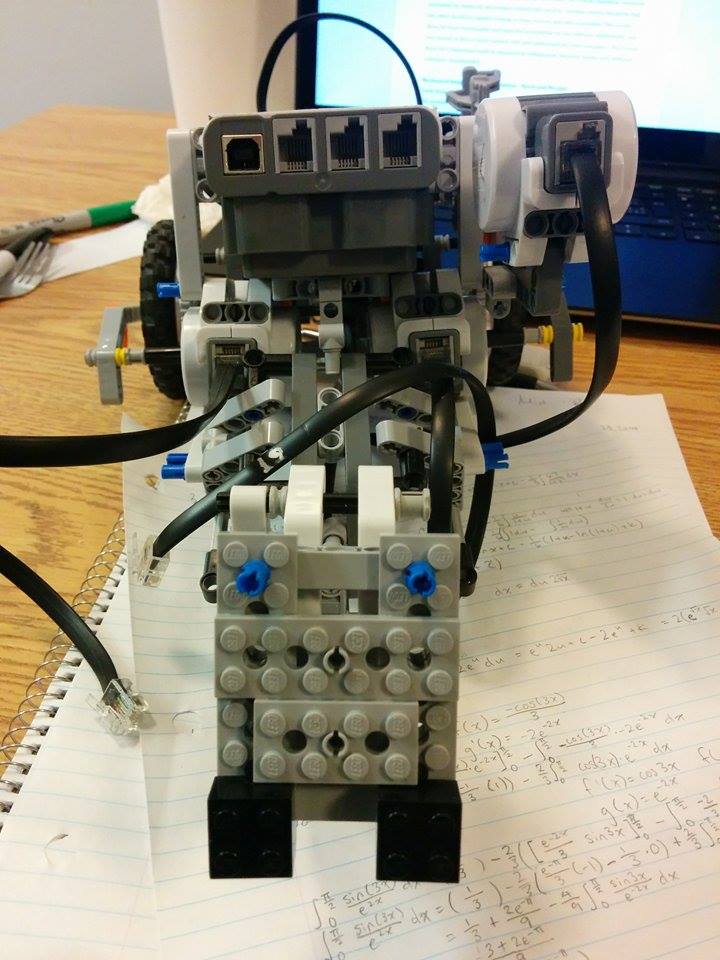


Figure 2.4 Back View

Figure 2.3 Side View

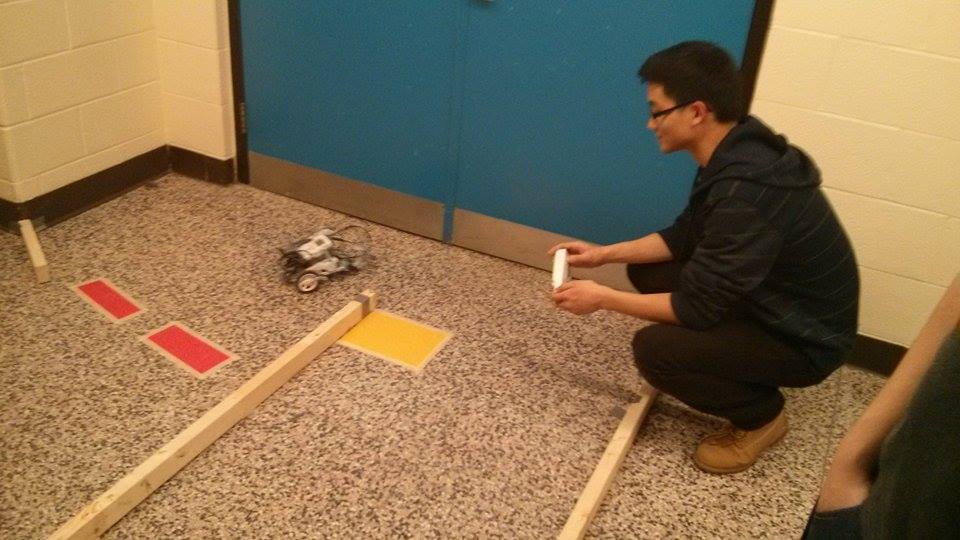
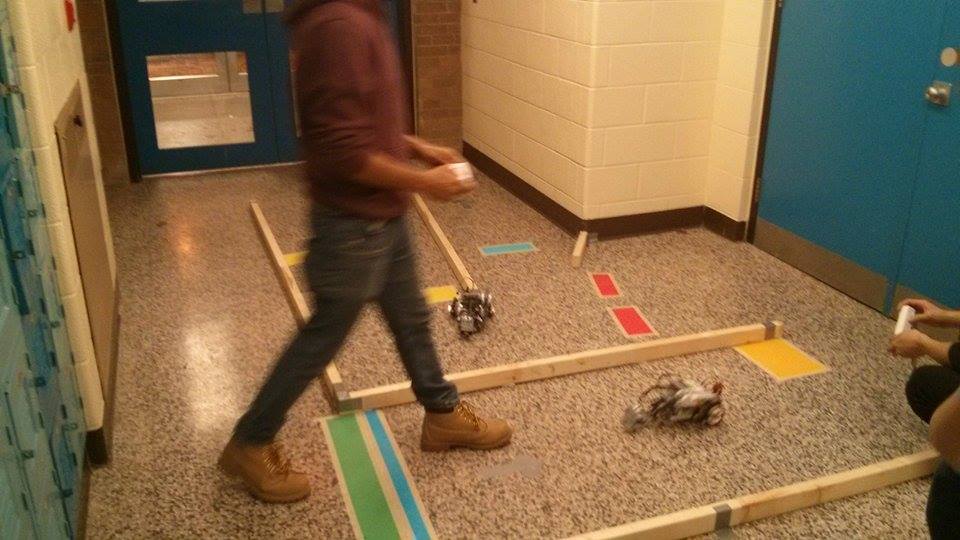


Figure 2.6 Demo Day

Figure 2.5 Demo Day

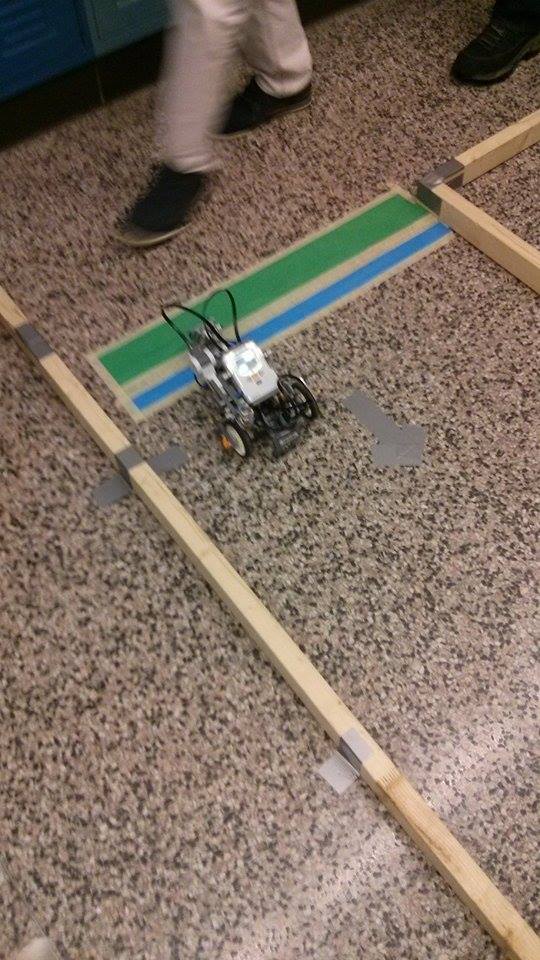


Figure 2.8 Demo Day, Robot running over start

Figure 2.7 Demo Day, Both Robots



Figure 2.9 Both motors and their motor drive



Figure 2.10 One single dear train

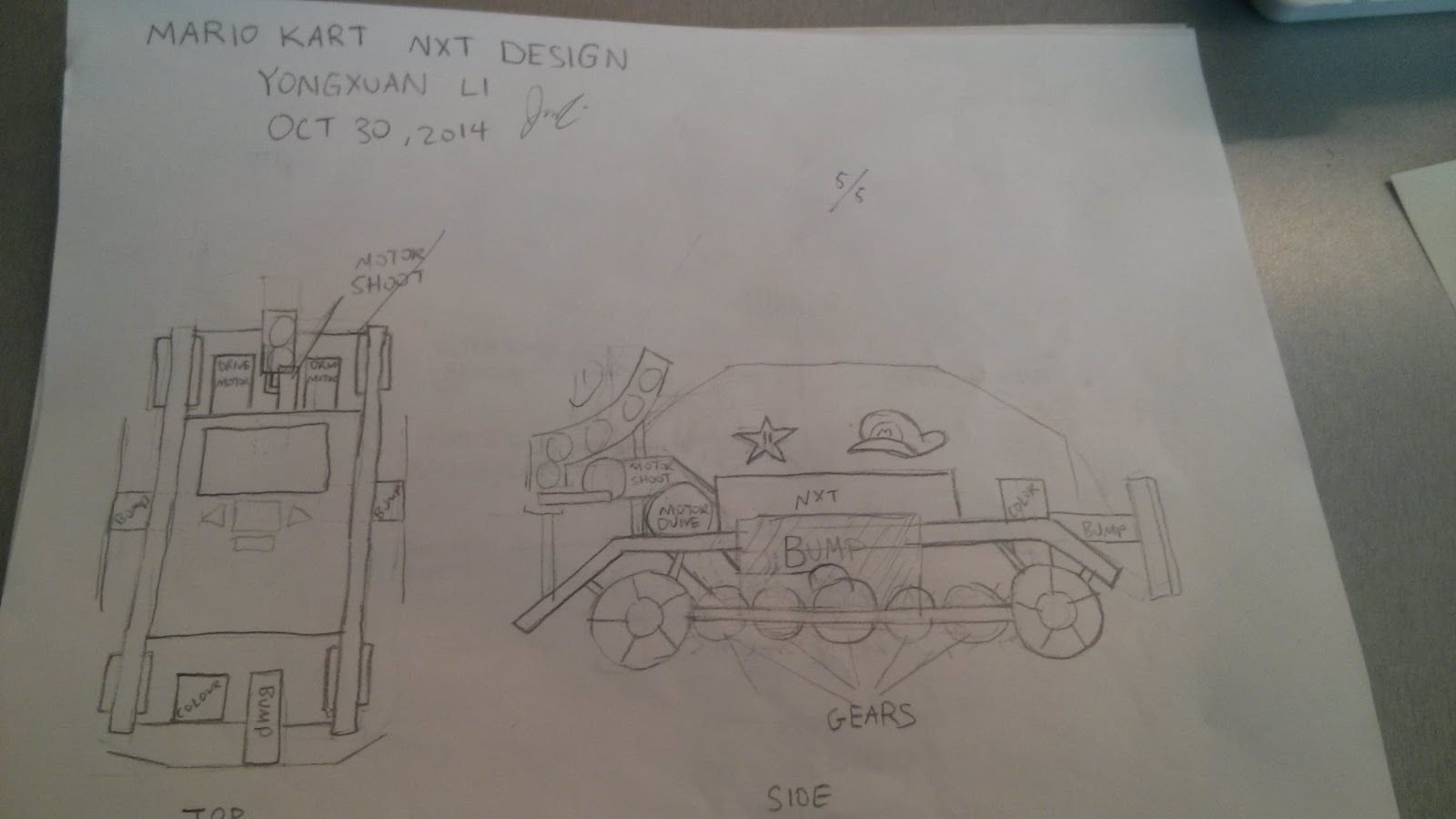
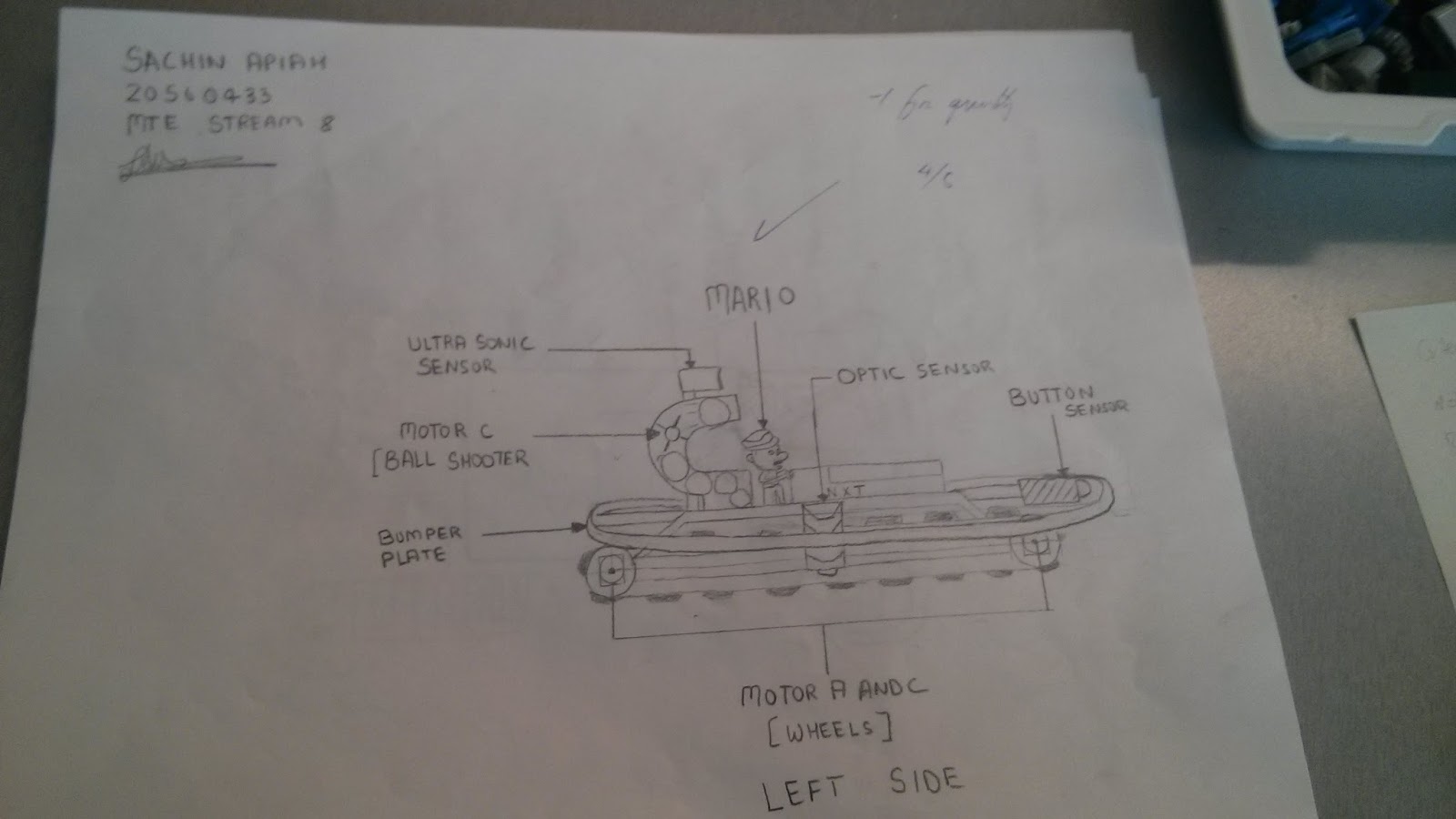
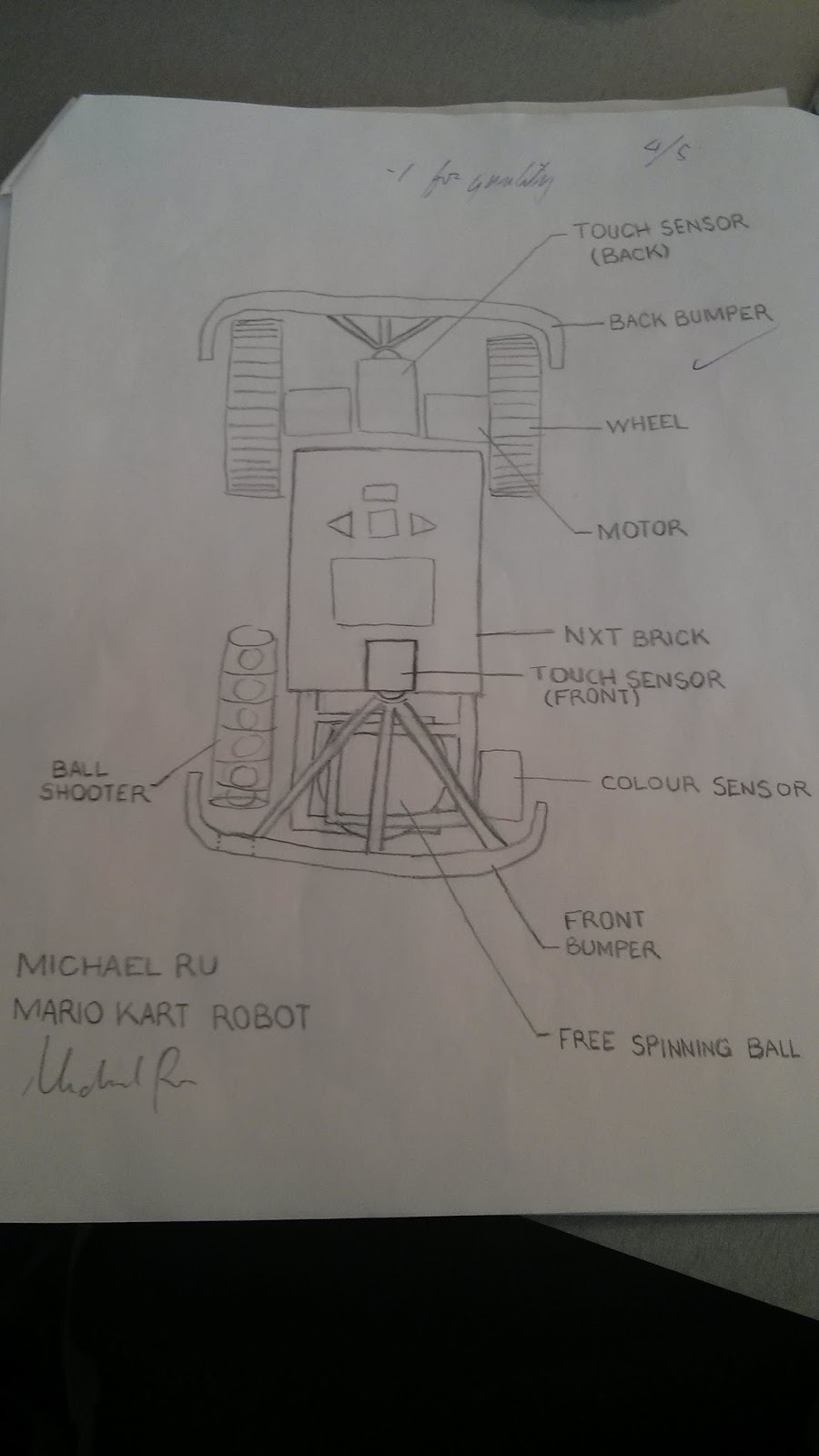


Figure 3.12 Possible Design 2

Figure 2.11 Possible Design 1



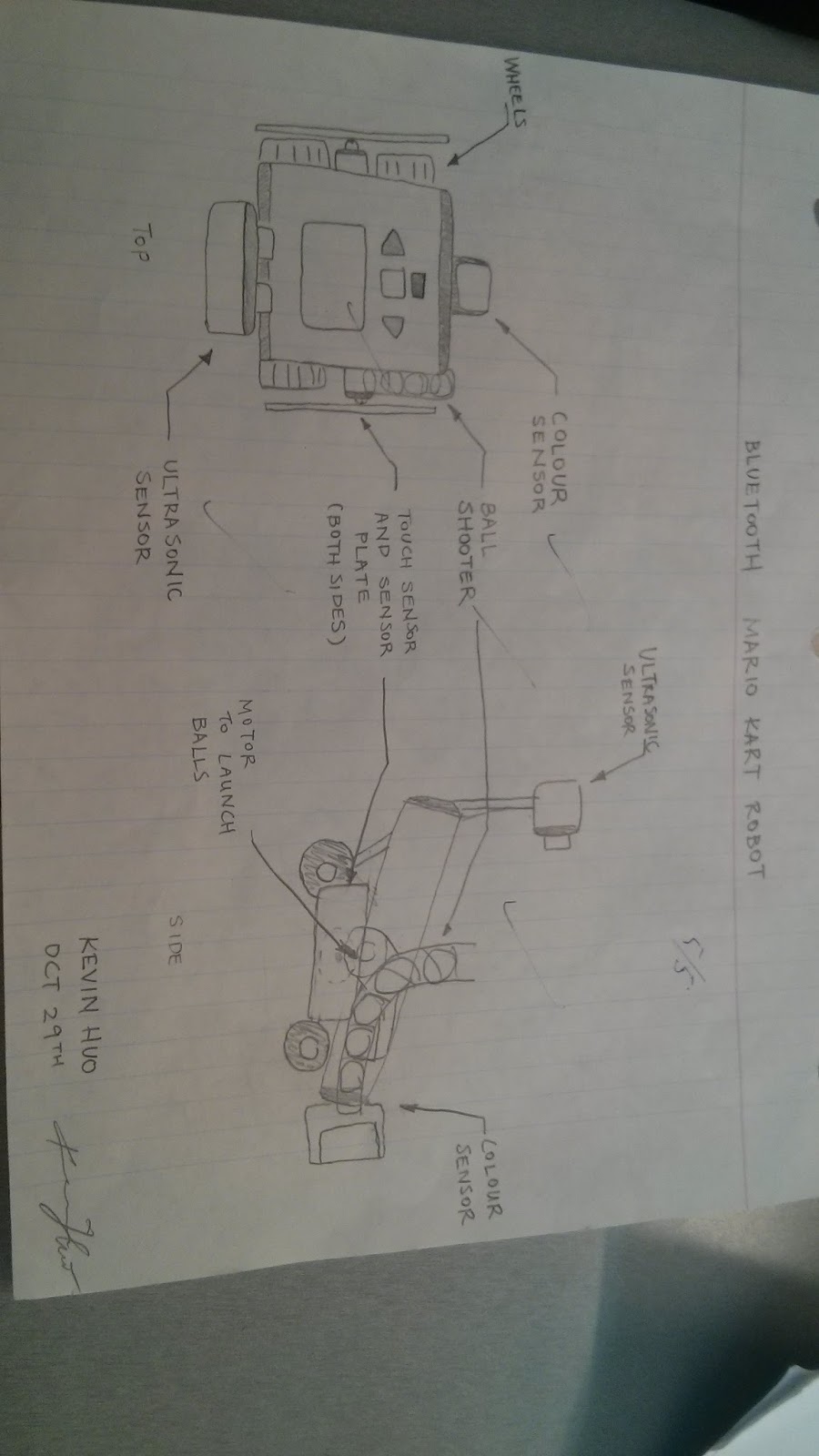


Figure 2.14 Possible Design 4

Figure 2.13 Possible Design 3

C - Design Matrix

Design Matrix used to compare 4 possible mechanical designs of the vehicle with each other using defined criteria.

Table 3.1 Tabulated Design Criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterion | Design 1 | Design 2 | Design 3 | Design 4 |
| Mobility | 4 wheels connected by gears | 2 wheels + 2 free wheels (using caterpillar tracks) | 2 wheels + free spinning ball | 2 wheels + 2 free wheels |
| Ease of build | Medium hard | Medium Hard | Medium | Easy |
| Appeal | Good | Good | Acceptable | Non-appealing |
| Additional Capabilities | Acceptable | Good | Acceptable | Good |

Table 3.2 Mapped Ratings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterion | Design 1 | Design 2 | Design 3 | Design 4 |
| Mobility | 0.6 | 0.5 | 0.8 | 0.5 |
| Ease of build | 0.4 | 0.4 | 0.5 | 0.8 |
| Appeal | 0.8 | 0.8 | 0.6 | 0.2 |
| Additional Capabilities | 0.4 | 0.6 | 0.4 | 0.6 |

Table 3.3 Relative Importance of Criteria

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Criterion | Mobility | Ease of Build | Appeal | Additional Capabilities | Row Total | Normal Rating |
| Mobility |  | 1 | 1 | 1 | 3 | 0.5 |
| Ease of Build | 0 | - | 0 | 1 | 1 | 0.166 |
| Appeal | 0 | 1 | - | 1 | 2 | 0.333 |
| Additional Capabilities | 0 | 0 | 0 | - | 0 | 0 |

Table 3.4 Scaled Ratings of Criteria

|  |  |  |
| --- | --- | --- |
| Criterion | Relative Rating | Fractional Rating |
| Mobility | 1.0 | 1.0 / 2.4 = 0.4167 |
| Ease of Build | 0.467 | 0.467 / 2.4 = 0.1946 |
| Appeal | 0.733 | 0.733 / 2.4 = 0.3054 |
| Additional Capabilities | 0.2 | 0.2 / 2.4 = 0.0833 |
|  | Total Rating: 2.4 |  |

3.5 Calculating Design Ratings

Design 1: (0.6)(0.4167) + (0.4)(0.1946) + (0.8)(0.3054) + (0.4)(0.083) = 0.6055

Design 2: (0.5)(0.4167) + (0.4)(0.1946) + (0.8)(0.3054) + (0.6)(0.083) = 0.58

Design 3: (0.8)(0.4167) + (0.5)(0.1946) + (0.6)(0.3054) + (0.4)(0.083) = 0.647

Design 4: (0.5)(0.4167) + (0.8)(0.1946) + (0.2)(0.3054) + (0.6)(0.083) = 0.475

Therefore from these calculations, Design 3 is the one that matches these criteria the best.