**Table of Contents**

1. Introduction 1
2. Software Design and Implementation 1
   1. Flowchart of Software Design 1
   2. Program Division 3
   3. Design Decisions 4
   4. Trade-offs 4
   5. Major Problems 6
3. Conclusions and Recommendations 6

**List of Figures**

**Figure 1:** Flowchart of Sotfware Design 3

**Figure 2:** Pictures of Mechanical Design 14

**List of Appendices**

**Appendix A** – Source Code 8

**Appendix B** – Robot Photos 13

**Acknowledgments**

We would like to thank the University of Waterloo, and the GENE 121 TA’s and professor for the use of the Lego Mindstorms NXT kit.

1. **Introduction**

The goal of the project was to create a robot with the capability to navigate a track without any external interference. The robot car navigates the track with the use of two colour sensors along the sides of the car to constantly align it with the track. Any obstacles in its path will be swept away with the help of the sonar sensor. If the robot is able to complete the predetermined number of laps it wins the race.

1. **Software Design and Implementation**
   1. **Flowchart of Software Design**

Start back motor at full speed

i < # of laps?

False

True

Start

Prompt for number of laps

Increases speed, waits half a second, then returns to normal speed (using speed boost function)

Activates sweeper arm to sweep object out of the way (usng sweep function)

Stop Robot Motors

Display:

-Total time

-all lap time individually

Either colour sensors detect green?

Either colour sensors detect yellow

Sonar sensor <=30 cm or button sensor pushed?

Steers robot back on track (using steer function)

Closed

Either colour sensors detect white

Display Lap Number

Either colour sensors detect green?

True

Increases speed, waits half a second, then returns to normal speed (using speed boost function)

False

Either colour sensors detect yellow

True

Steers robot back on track (using steer function)

False

Sonar sensor <=30 cm or button sensor pushed?

Activates sweeper arm to sweep object out of the way (usng sweep function)

True

False

Stores Lap Time

Either colour sensors detect white

Both sensors don’t detect white

True

True

False

False

End

Display total and lap times

Stop Robot Motors

**Figure 1:** Flowchart of Software Design

* 1. **Program Division**

The overall program was broken up for two reasons: simplicity and convenience. When developing the pseudo-code, it was established which actions in the code would be repeated. These actions were tagged as functions and written as such. The button select function and the time display function were written out of this selection process as they are both used more than once in the code. In addition, it was decided that the menu system would be an isolated task that only returned one variable. As such, the menu function was created in order to simplify the code.

The second course of action was to create functions for all of the tasks that robot was going to complete, such as activating the sweeper arm, going through a speed boost, and steering. This was also done due to simplification, allowing most the decisions to take place in the main program while the functions just carried out the tasks.

After testing the functions, the inaccuracy of the colour sensor caused the steering function to malfunction, which prompted the addition of a function whose purpose was to ensure the validity of the sensor when ending a turn. This function could not be used for every single colour, so as a result another function was added for sensing the white finish line.

* 1. **Design Decisions**

When designing the main program, the main issue that was considered was activating each function. In order to do this, a loop was created that constantly checked if the conditions were met, would activate each function when the condition was met. In order to avoid infinite looping, this loop would exit whenever a lap was completed. Due to the nature of lap counting, a for loop was created that would keep track of the laps, and once all were completed, it would exit into the end of the program.

When designing the steer function, it was initially considered to have a loop within the function that constantly checked if it needed to turn or not. However, this was not implemented, as this would result the robot getting stuck in the steer function and not be able to detect any other colours. Instead, the function will exit when a turn is completed, and will return to the function once a turn is required again. In addition, initially the steer function was separated into two sections, one section that would be run if the robot would need to turn right and one if it would turn left. However, a lot of the code was repeated, and therefore it was decided to have a variable that would store whether the robot was turning left or right, with a value of 1 or -1, which was used when activating the motor.

* 1. **Trade-offs**

One of the initial problems that were encountered was with the colour sensor, as it would not consistently give accurate values. This was most evident while the robot was turning. When the robot’s colour sensor was on a yellow border the intention was for the robot to turn. However, the robot would not necessarily turn when intended, but would turn at an inopportune moment which caused the robot to lose its way. The solution for this was to create a counter for the colour sensors. This counter kept track of the number of repetitions of a certain colour value that the sensor would detect. This method was effective because it allowed for the robot to have an erroneous reading. Instead of turning the vehicle immediately, the computer would now wait for a predetermined number of counts before turning. After some testing it was found that the optimal number of counts was five. The only problem that was discovered with the counter was that it caused a delay in the turning motion of the robot. We discovered that when the robot went off track that it would cause a vast increase in the time it takes for the robot to register that a turn is required. This resulted in more instances of the robot turning later and spending a lot of time along the border of the track. This method was implemented for the finish line detection, and was planned to be implemented for all detections. However, adding this method slowed down the processing of the robot, and consequently had to be removed, lowering the accuracy of the other tasks.

After the previously mentioned error was rectified it was found that the robot was travelling much to fast in relation to its ability to turn. This meant that it was routinely driving straight off the track. Initially the design was changed in order to create a smaller turning radius. However, this attempt was not successful and resulted in reducing the robot’s speed. Since the robot took time to register whether or not it needed to turn, it was found to be beneficial if the car was slower, thus allowing the car more time to turn around the corner. The downside to this change is that our car is labeled as a race car, this meant that reducing the speed decreased morale and the allure of our project. However, this was deemed a necessary change and was implemented with success.

The last trade off that was present was with the integrity of the robot. Due to time constraints, our track was built independent of some of the other components so there were a couple of faulty components. One of which was if the robot lost sight of the track and ended up in empty space. When in empty space the robot would continue aimlessly until one of the group members went about stopping it. This caused a problem as the goal of the robot is to complete a certain number of laps. Numerous ideas were bantered about such as having the robot reverse in order to try and restore its former position, or to have a race fail option in which the race would end if the car left the track. In the end, due to the propensity of the robot to have an error that left it off the track, it was decided that the optimal course of action was to create a crash function that allowed the robot to be redeployed at the nearest point on the track and to continue on the race as required. The trade-off was that the car lost its full autonomy and required a human to reset the robot. It was deemed necessary in order for the robot to complete its task.

* 1. **Major Problems**

There were a few significant problems with the code. The largest problem with the code was the implementation of error precautions in order to compensate for the unreliability of the colour sensors. As previously mentioned, a counter method was implemented in order to give an accurate reading. This method was chosen due to its simplicity, since it only involved two decisions (whether it detects the colour, and whether the counter was greater than the specified value) and an increment. In addition, this method was beneficial as the value of the increment could be passed through functions. Initially, there was no check for lap completion within the turn function, which caused the robot to often miss complete laps. Due to this simplicity of this method, it could be added in within the steer function, and could return the value of the increment when it was greater than the specified value.

The motor positioning proved to be a problem with direction of the motor having to be reinitialized multiple times. Because the forward direction of the motor may not correlate to the intended forward direction of our robot, any design change that it experienced resulted in the group needing to change the coding so all of the robots motors continued to go in the appropriate direction. This sometimes was the result of great confusion due to miscommunication within the group if the robot necessitated a design change.

1. **Conclusions And Recommendations**

There were many aspects of this project that could potentially be improved upon. The largest problem that was encountered with the robot had to do with the turning of the robot. The turning was faulty for a myriad of reasons however one distinctly possible solution to this problem is both mechanical and software in nature. Given the possibility of a smaller turning radius and 4 colour sensors it would be very beneficial to create a system of functions that would allow for a gradual turning process. With a system of four colour sensors – 2 on each side – this would allow for a soft turn when the outer sensors have been initiated by the appropriate colour and a hard turn if required by the inner colour sensor. This turning gradation would eliminate a lot of the problems that are experienced by our current rendition of the robot.

Another aspect that could have been improved upon is a complete predetermined setup of the code. An issue that was continually observed with the robot was that the variability of the course proved too much for the robot as small discrepancies from the beginning of the track would result in crashes throughout the course. This was also due to the unreliability of the colour sensors. To compensate for this hard coding the entire course and then writing functions for new features would have been a much more beneficial course of action, as it would have resulted in a much smoother

**Appendix A –** Source Code

/\*Stores constants for the speeds of the motors, TURN = Turn Speed, ANG = Turn Angle,

HS = Half Speed, FS = Full Speed, SS = Super Speed, SCHECK1 = # of colour sensor registers before registering that its out of a turn

SCHECK 2 = # of colour sensor registers before registering that its read a lap \*/

int const TURN = 10,TS = -20, FS = -40, SS = -70, SCHECK1 = 2, SCHECK2 = 10;

//Written by: Justin Langendoen

//Waits for a button press, and returns the pressed button

int buttonSelect ()

{

while (nNxtButtonPressed == -1){}

int b = nNxtButtonPressed;

while (nNxtButtonPressed != -1){}

return b;

}

//Written by: Cody Reading

//Displays menu and returns user input of number of laps

int menu()

{

int button[2];

do

{

nxtDisplayString(1, "Select number of");

nxtDisplayString(2, "laps:");

nxtDisplayString(4, "2 Laps (Right)");

nxtDisplayString(5, "3 Laps (Left)");

nxtDisplayString(6, "4 Laps (Middle)");

button[0] = buttonSelect() + 1;

eraseDisplay();

nxtDisplayString(0, "You have");

nxtDisplayString(1, "selected %d laps.", button[0]);

nxtDisplayString(3, "Is this correct?");

nxtDisplayString(4, "Yes(Middle)");

nxtDisplayString(5, "No (Left)");

button[1] = buttonSelect();

eraseDisplay();

} while (button[1] == 2);

eraseDisplay();

nxtDisplayString(0, "On your marks...");

wait10Msec(100);

nxtDisplayString(1, "Get set...");

wait10Msec(100);

nxtDisplayString(2, "Go!!!");

return button[0];

}

//Written by: Patrick Trzcinka

//Activates the sweeper arm

void sweep ()

{

motor[motorC] = 50;

while (nMotorEncoder[motorC] < 135){}

motor[motorC] = -50;

while(nMotorEncoder[motorC] > 25){}

motor[motorC] = 0;

}

//Written by: Nikhil Arora

//Converts and displays time from robots timer

void timeDisplay (int line, const string text, int time)

{

float sec = time /100.0;

if (sec > 0)

nxtDisplayString (line, "%s: %d:%.2f", text, time/6000, sec);

else

nxtDisplayString (line, "%s: %d:0%.2f", text, time/6000, sec);

}

//Written by: Patrick Trzcinka

//Incremenents the lapCheck variable for reading and coming off //white line

int lapUpdate(int lapCheck, bool startLine)

{

if ((SensorValue[S1] == 6 || SensorValue[S4] == 6) && startLine || SensorValue[S1] != 6 && SensorValue[S4] != 6 && !startLine)

lapCheck ++;

else

lapCheck = 0;

return lapCheck;

}

//Written by: Patrick Trzcinka

//Incremenents the turnCheck variable for both turns

int turnUpdate(int turnCheck, int dir)

{

if (dir == -1 && SensorValue[S4] != 4 || dir == 1 && SensorValue[S1] != 4)

turnCheck ++;

else

turnCheck = 0;

return turnCheck;

}

//Written by: Nikhil Arora

void crash()

{

eraseDisplay();

motor[motorA]=0;

motor[motorB]=0;

nxtDisplayString(0,"You have crashed");

nxtDisplayString(1,"Please place car ");

nxtDisplayString(2,"back on track");

nxtDisplayString(3,"and push button");

nxtDisplayString(4,"to continue race.");

while(nNxtButtonPressed == -1){}

while(nNxtButtonPressed != -1){}

//Resets steering back to straight

if (nMotorEncoder[motorB] < 0)

{

motor[motorB]=TURN;

while (nMotorEncoder[motorB] < 0) {}

}

else if (nMotorEncoder[motorB] > 0)

{

motor[motorB]=-TURN;

while (nMotorEncoder[motorB] > 0) {}

}

eraseDisplay();

motor[motorA] = FS;

}

//Written by: Justin Langendoen

//Speeds up the robot for short time, and then slows it down

void speedBoost()

{

motor[motorA] = SS;

wait10Msec(50);

motor[motorA] = FS;

}

//Written by: Cody Reading

//Controls steering of the robot

int steer(int lapCheck)

{

motor[motorA] = TS;

time10[T2] = 0;

int turnCheck = 0; // Increment variable for turning

int dir = 1; //Stores direction of turning, -1 is right, 1 is left

//Detects lane on left side, will turn right

if (SensorValue[S4] == 4)

{

motor[motorB]=TURN;

while (nMotorEncoder[motorB] < 20) {}

dir = -1;

}

//Detects lane of right side, will turn left

else if (SensorValue[S1] == 4)

{

motor[motorB]=-TURN;

while (nMotorEncoder[motorB] > -35) {}

}

motor[motorB]=0;

//Turns until reads not yellow multiple times

while (turnCheck < SCHECK1)

{

turnCheck = turnUpdate(turnCheck, dir);

lapCheck = lapUpdate(lapCheck, 1);

//Check for lap complete, crash, or speed boost while in //turn

if (lapCheck >= SCHECK2)

return lapCheck;

if (SensorValue[S1] == 4 && SensorValue[S4] == 4)

crash();

if (SensorValue[S1] == 2 || SensorValue[S4] == 2)

speedBoost();

}

motor[motorB] = dir \* TURN;

while (dir \* nMotorEncoder[motorB] < 0){}

motor[motorB] = 0;

motor[motorA] = FS;

return lapCheck;

}

//Written by: Cody Reading

task main()

{

int lap = menu();

int time[4];

int totTime = 0, lapCheck = 0;

SensorType[S1] = sensorCOLORFULL; // Right Colour

SensorType[S2] = sensorTouch;

SensorType[S3] = sensorSONAR;

SensorType[S4] = sensorCOLORFULL; //Left Colour

nMotorEncoder[motorB] = 0;

nMotorEncoder[motorC] = 0;

motor[motorA] = FS;

for (int i = 0; i < lap; i++)

{

time10[T1] = 0;

lapCheck = 0; //Increment variable for laps

nxtDisplayString(7, "Lap %d", i+1);

do

{

//Perform different tasks depending on colour passed

if (SensorValue[S1] == 2 || SensorValue[S4] == 2) //Blue

speedBoost();

else if (SensorValue[S1] == 4 || SensorValue[S4] == 4)

lapCheck = steer(lapCheck);

/\*Steer the robot if it detects yellow and return if white is passed through increment variable\*/

if (SensorValue[S3] <= 30 || SensorValue[S2] == 1)

//Detects object to sweep out of the way

sweep();

//Avoid update if detects white through steer function

if (lapCheck < SCHECK2)

lapCheck = lapUpdate(lapCheck,1);

}while (lapCheck < SCHECK2);

eraseDisplay();

time[i] = time10[T1]; //stores lap time

totTime += time[i];

lapCheck = 0;

//Waits for robot to come off white line to start a new lap

while (lapCheck < SCHECK2)

lapCheck = lapUpdate(lapCheck,0);

}

motor[motorA] = 0;

wait10Msec(300);

timeDisplay(0, "Total", totTime);

timeDisplay(2, "Lap 1", time[0]);

timeDisplay(3, "Lap 2", time[1]);

if (lap > 2)

timeDisplay(4, "Lap 3", time[2]);

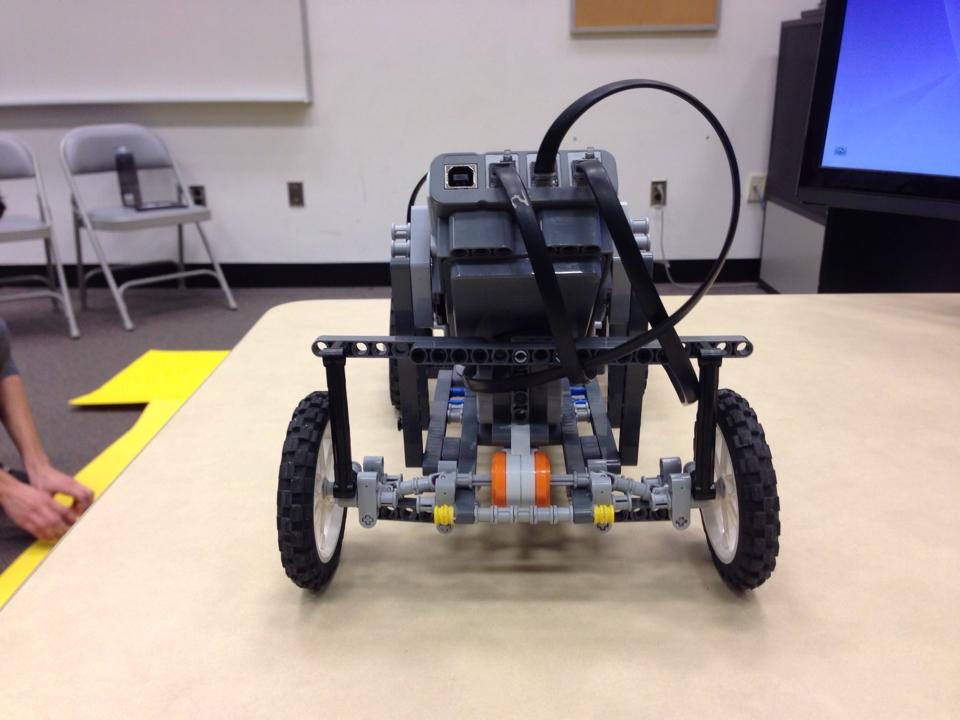
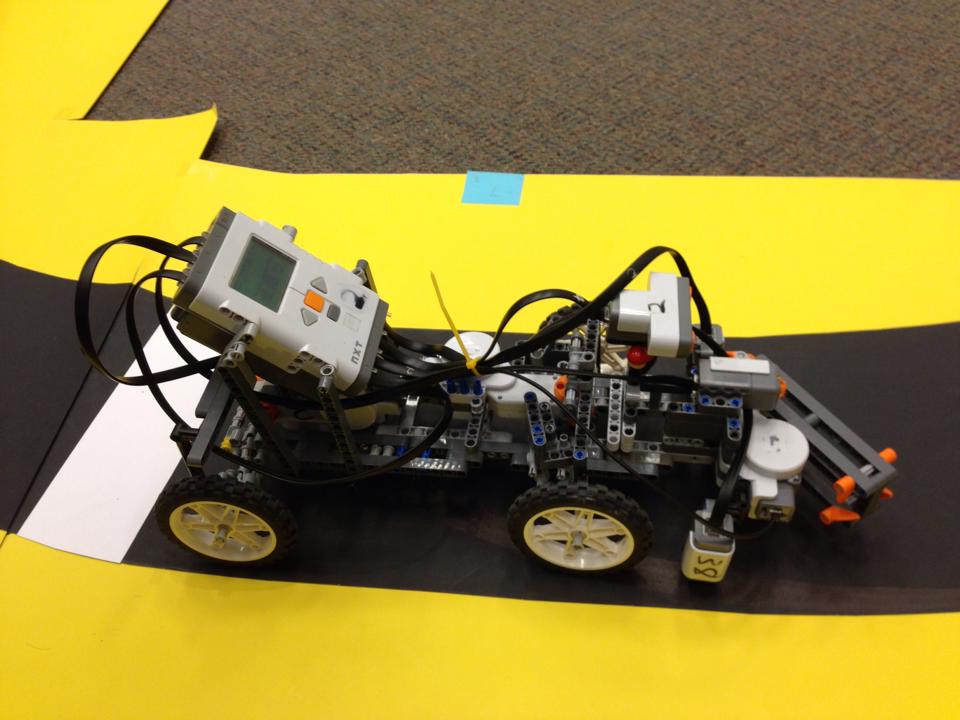
if (lap > 3)

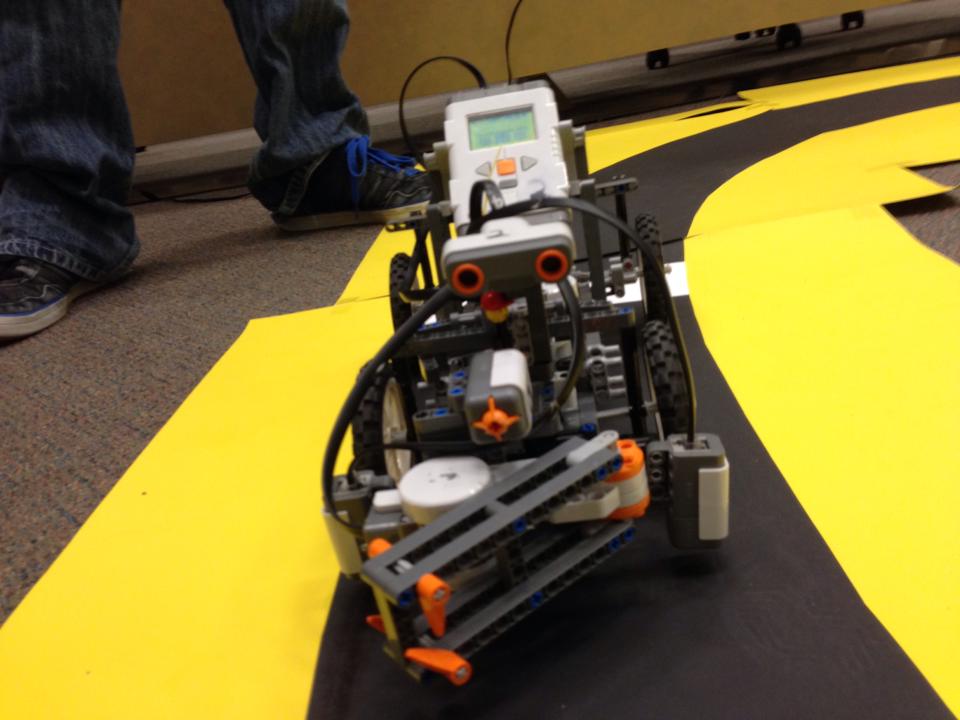
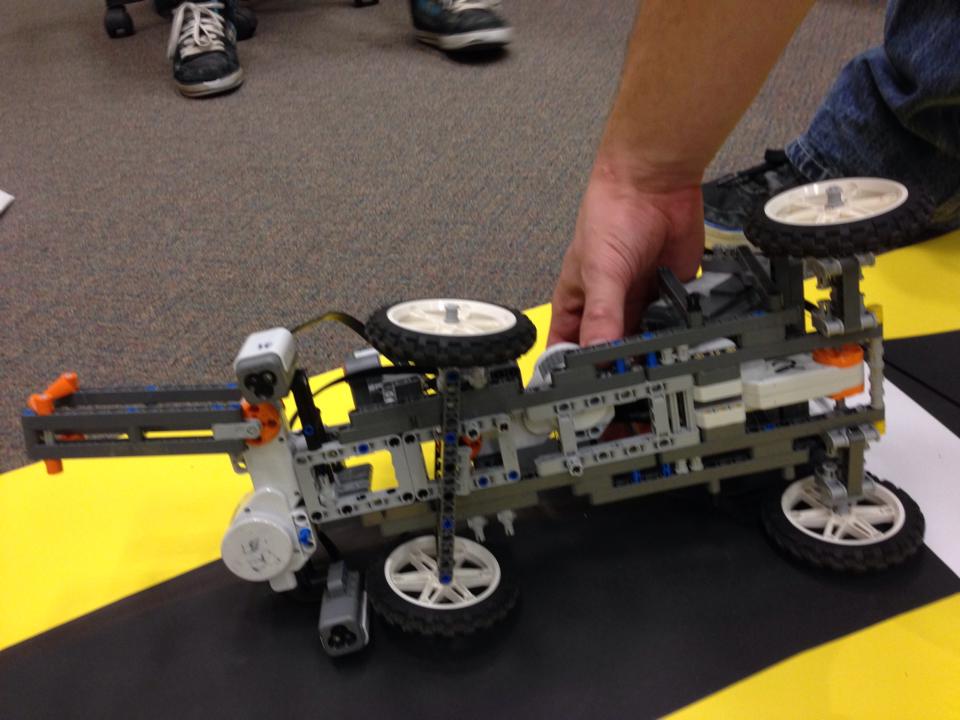
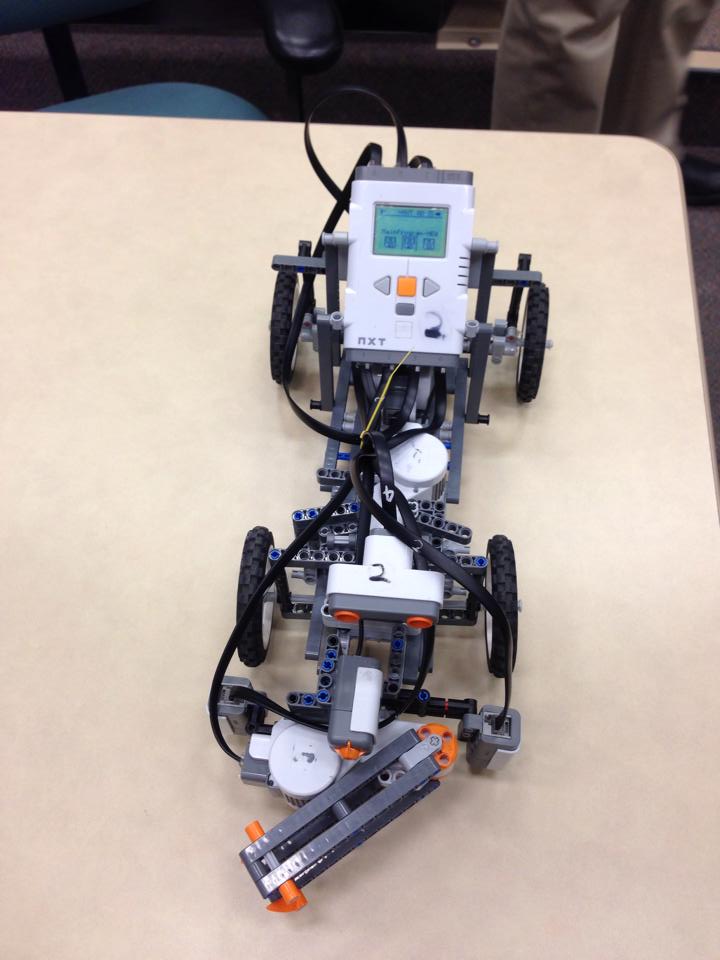
timeDisplay(5, "Lap 4", time[3]);

wait10Msec(1000);

}

**Appendix B –** Robot Photos





**Figure 2:** Pictures of Mechanical Design