|  |
| --- |
| https://uwaterloo.ca/brand-guidelines/sites/ca.brand-guidelines/files/resize/uploads/images/universityofwaterloo_logo_vert_rgb_0-300x195.png  FACULTY OF ENGINEERING |
| **Final Report on Spectaculum Lautus “Eyeglasses Cleaning Robot”** |
| A report prepared for Final Mechatronics 100 Project |
|  |
| by  Siddhant Chandgadkar 20605875 Rimpey Diaz Brandon Fong Himel Mondal  1A Mechatronics Engineering |
| 12/4/2015 |

Table of Contents

[List of Figures 2](#_Toc436759550)

[1.0 Project Summary 3](#_Toc436759551)

[2.0 Design Problem Definition 3](#_Toc436759552)

[3.0 Goals and Objectives 3](#_Toc436759553)

[4.0 Constraints and Criteria 4](#_Toc436759554)

[4.1 Constraints 4](#_Toc436759555)

[4.2 Criteria 4](#_Toc436759556)

[5.0 Mechanical Re-Design and Implementation 5](#_Toc436759557)

[5.1 Frame / Chassis Design 5](#_Toc436759558)

[5.2 Motor Drive Design 7](#_Toc436759559)

[5.3 Sensor Attachment Design 9](#_Toc436759560)

[6.0 Software Design and Implementation 11](#_Toc436759561)

[6.1 Software Design Flowchart 13](#_Toc436759562)

[7.0 Project Planning 14](#_Toc436759563)

[8.0 Conclusion 15](#_Toc436759564)

[9.0 Recommendations 15](#_Toc436759565)

[9.1 Improvements to Mechanical Design 15](#_Toc436759566)

[9.2 Improvements to Software Design 16](#_Toc436759567)

[10.0 References 17](#_Toc436759568)

[Appendix A: Source Code 18](#_Toc436759569)

[Appendix B: Spectaculum Lautus Photos 31](#_Toc436759570)

[Appendix C: Checklist / Task List on Project Demo Day 32](#_Toc436759571)

[Appendix D: Project Planning Gantt Chart from Design Report 33](#_Toc436759572)

# List of Figures

[**Figure 1: Overall Structure of Spectaculum Lautus** 5](#_Toc436759576)

[**Figure 2: Motors in Robotic System** 7](#_Toc436759577)

[**Figure 3: Initial X-Axis Motor Design** 8](#_Toc436759578)

[**Figure 4: Glasses Spraying Robot Car** 9](#_Toc436759579)

[**Figure 5: Software Flowchart** 13](#_Toc436759580)

[**Figure 6: Gantt Chart of Final Project Overview** 14](#_Toc436759581)

# 1.0 Project Summary

Spectaculum Lautus is the first step in solving the efficiency problem of cleaning large scale surfaces such as windows on a building. The implementation of this solution on a micro-scale occurred by sensing bounds of the glasses to allow for an accurate and autonomous cleaning of the surface. This solution was extensively tested as the project was being developed by testing each function of the software in correspondence to the functionality of Spectaculum Lautus in a chronological order. Given the functionalities of this project, there were a plethora of inefficiencies that could be improved with mechanical and software recommendations such as including an extra motor for even y-axis traversal, using a light intensity sensor to ditch the requirement of extraneous coloured pieces and integration of more advanced control systems to the code.

# 2.0 Design Problem Definition

Glass surfaces are embedded in many products used every day by large masses of people. These surfaces are found on products such as eyeglasses and screen technologies like smartphones. The rate at which these screens are smudged and get dirty is very high as dust and oil on fingers easily affect these glass surfaces. Manually cleaning these products constantly is a very meticulous process that isn’t always effective on the first iteration. The struggle of cleaning eyeglasses is a real one and should be addressed in modern society, as glasses have been a fundamental part of human vision clarity for centuries.

# 3.0 Goals and Objectives

The goal of the robot made was to clean the glass area of a pair of eyeglasses (prescription glasses, sunglasses, reading glasses), of any shape and length. Being able to clean both sides without any human assistance. The objective of the design made is to detect the surface of the glasses being cleaned, and determine its length and width. Afterwards the cleaning arm was designed to extend towards the pair of glasses and traverse its surface. The robot concluded its function by flipping the glasses and repeating the process for the other side.

# 4.0 Constraints and Criteria

## 4.1 Constraints

The first constraint that the robot had involved making the cleaning arm identify that there is a pair of eyeglasses placed in the glasses holder. To accomplish this task, a touch sensor was positioned in the base of the glasses holder. Once the glasses holder goes down in position to be cleaned, the pair of glasses placed in it would hit the touch sensor and communicate to the cleaning arm that there are a pair of glasses to be cleaned.

The next constraint is to position the cleaning arm towards the area where the precise area where the eyeglasses was located. Colour sensors were used to meet this constraint which detected red Lego pieces around the eyeglasses, indicating the boundaries that it was meant to traverse. This constraint was changed from just make arm reach the eyeglasses since another challenge came up in the design where even though the glasses cleaner reached the glasses, it was not aligned to its boundaries.

The third constraint was to make the cleaning arms make contact with the glasses at a point where it's not too close to it that it would damage the glasses or too far that it would barely touch them. For this constraint, a touch sensor was added to the cleaning arm to find the right point to clean it. This touch sensor is activated when the frame of the glasses hits the arm.

The last constraint was for the arms to maintain contact constantly with the eyeglasses as it traverses across its surface. Due to the sensor implementations mentioned above, both the cleaning arm and the glasses holder are positioned in an optimal state in which the cleaning cloths in the arms can traverse freely across the glasses without breaking contact. Since there isn't much friction between those two elements, the cloth easily moved around the glasses being cleaned.

## 4.2 Criteria

The first criterion for the robot was to transport the cleaning solution towards the glass from any distance. With it, the glasses were able to be sprayed and provide less friction for the cleaning arm to traverse across it.

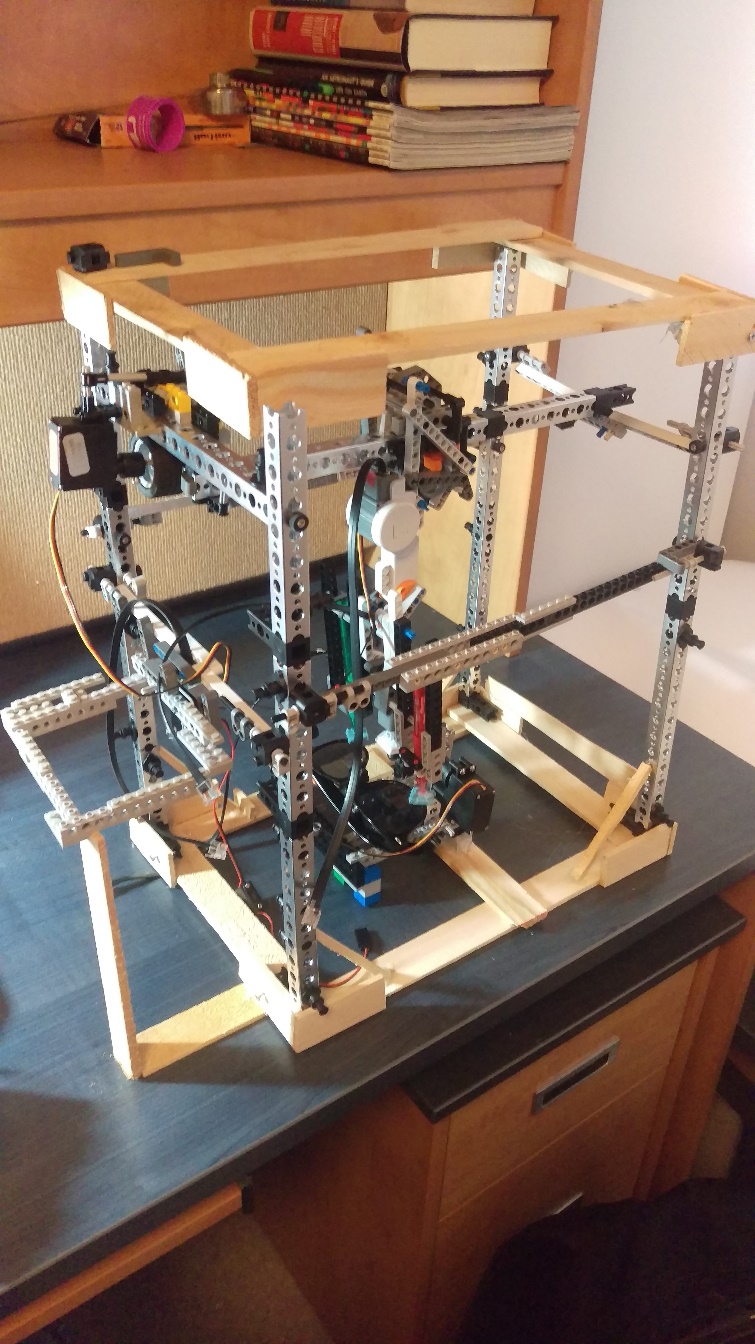
Additionally, both sides of the glasses needed to be cleaned. After flipping the eyeglasses to the uncleaned side, the arms needed to be positioned properly so they can accurately transverse it, following its boundaries.

Another criterion was to properly detect the boundaries that the cleaning arm needed to travel when wiping the glasses; being able to identify which area of the glasses is the glass surface, and which part of it is the frame.

A fourth criterion was to be able to flip the glasses being cleaned mechanically, without the aid of a person to adjust it or control it, while keeping it at the same altitude in which the cleaning arms were able to reach it and clean it.

# 5.0 Mechanical Re-Design and Implementation

## 5.1 Frame / Chassis Design



**Figure 1: Overall Structure of Spectaculum Lautus**

As observed in Figure 1, the main structure of the robot revolved around the outer chassis which gave it stability and eased the motors and sensors ability to perform its function efficiently.

The lower and the upper rectangular structures were mainly used to unite the four columns which are attached to the main structure, and gave it more stability when placed on a flat surface. Additionally a small wall was placed on the lower base, in front of the area where the glasses would remain when they were turned in the position to be sprayed. The cleaning solution enabled the surface to be smoother and have less friction when the cleaning cloths in the arms traversed across of it [1]. Adding a fluid which had a low viscosity was decided to be a practical solution to make the robot run smoother. This small wall gave the robot carrying the cleaning solution an indication of where to stop.

The bars which are parallel to the upper and lower rectangular structures each served a different purpose. The one on the side of the arm was used to hold the motor containing it, and provide a basis in which it can freely move on the left and right direction. The bar directly in front of it, on the opposite side of the robot, eased this movement. By positioning it close to the upper rectangular structure, the arm had an easier time moving without needing direct assistance from a human to help move it.

The bars perpendicular to the ones mentioned above, connected the four pillars containing the main structure. And the bar in between was used to hold the motor with the arm in it while enabling it to be able to move forward and backwards.

Another feature in the design included the structure placed on the side with the motor on it, where the NXT controller was placed. It was located outside of the main frame. Since most of the motors and sensors were closer to this side, they were all able to be connected to the main NXT controller, and it didn't get in the way of the main arm as it traversed around the glasses to clean it. A trade off of this design is that the movement of the arm in the x and y direction reached a certain limit since the placement of the Lego NXT controller added extra length to how far the wire could be extended.

The structure holding the glasses was designed to be adjustable depending on the thickness of the rim of the glasses. Additionally, 4 pillars were placed around it which indicated the color sensors the distance it needed to traverse in order to fully clean the glasses. An addition to the design that could have been implemented would be to attach the glasses holder structure to the base of the chassis holding the arm.

Another structure that the robot had included the car that carries the glasses cleaning solution. The frame around it provided the sprays stability so it didn't move as it travels across an area. A trade off of going with this design involved the fact that it makes it difficult to take the sprays off or put on if needed. An extension that would have been added if more time were given, would be to attach an arm to a motor which would press the spray when it reaches its destination in front of the glasses. With the current design, the glasses need to be sprayed manually.

## 5.2 Motor Drive Design

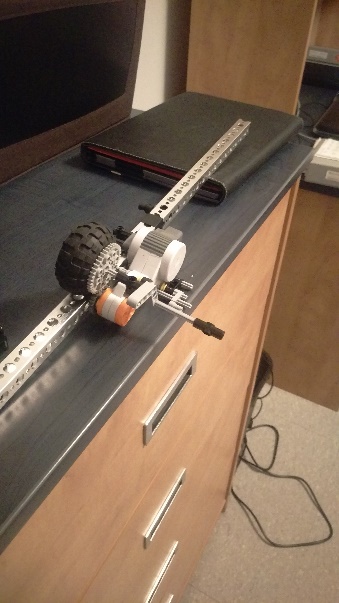
This project depended heavily on motor movement, as this movement essentially controlled the pattern in which the cleaning devices polished the lenses. As a brief overview of the functions of all the motors in the robotic system, there was an x-axis, y-axis and z-axis motor which helped in positioning the cleaning devices at optimal positions over the lenses. Then, a fourth motor was used to rotate the glasses on a clamp, as a double assurance mechanism to make sure the glasses touched the glasses lenses.

|  |  |  |
| --- | --- | --- |
| X-Axis Motor | Y-Axis and Z-Axis Motors | Glasses Rotator Motor |

**Figure 2: Motors in Robotic System**

As shown in Figure 2, each motor had an important functionality for the entire robotic system. Three Tetrix® motors were used (one for the x-axis motor, one for the y-axis movement and one for the rotation of the glasses), while the fourth was an NXT motor to control the initial movement of the cleaning devices. A fifth motor (NXT) was used for the car carrying the cleaning solution for the glasses; however, this robot car was a separate mechanism unrelated to the main portion of the project.

Analyzing the effectiveness of each one of these motors, initially, the x-axis motor consisted of a gear system attached to an NXT motor, as shown below in Figure 3.

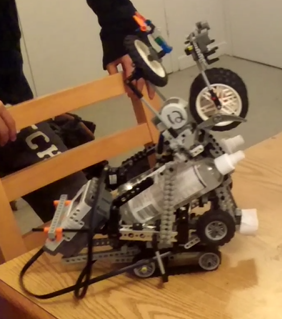


**Figure 3: Initial X-Axis Motor Design**

However, there were numerous problems associated with this gear system. The main area of concern was that the wheel did not have a tight grip on the bar. Therefore, this looseness caused the motor to slip off the bar. This resulted in problems with the positioning of the cleaning devices over the lenses. Referring back to Figure 2, an improvement made in this area was to use a servo motor instead of an NXT one, and to utilize only one wheel connected to the motor, instead of a gear system with two gears and a wheel. This was more effective, since all the power from the motor was instilled on the single wheel, and not distributed for the function of each gear.

With regards to the motor which rotated the glasses and the clamp, it bore resemblance to the purpose of a stepper motor. A stepper motor operates in pulses, with discrete steps that turn the motor to a specific position [2]. The motor used in the project was a 180 degree servo motor, which was coded to move within a fixed bound. However, the deadband (dead zone) of the motor was permanently set at a rotation exceeding ±180 degrees.

The lone NXT motor used as the z-axis motor, shown in Figure 2, was useful due to its ease of connection to the Lego parts. The Tetrix® motors were hard to interface with some of the Lego components; however, the ability to directly connect the structures of the cleaning devices to that of the Lego NXT motor made this construction process much easier. However, one problem with the NXT motor was the power required to hold the cleaning devices at a 90 degree angle, parallel to the ground, when the glasses were being sprayed by the cleaning solution. Since the cleaning devices had a low centre of mass, the NXT motor had trouble keeping the structure of the devices at a specific height. To solve the problem, pulse-width modulation was used to keep the motors in check, and hold it at this specific height.



**Figure 4: Glasses Spraying Robot Car**

Finally, as shown above in Figure 4, the glasses spraying robot car (GSRC) operated with the use of one NXT motor to drive forwards. The construction of a mechanism to spray the glasses automatically using the strength of an NXT motor was attempted; however, the NXT motor could not deliver enough power to press on the spray bottles. Perhaps another motor would have delivered double the power to initiate this action. This can be considered as a recommendation for better performance of the robotic system in the future.

## 5.3 Sensor Attachment Design

The glasses cleaning robot utilized a number of different sensors in order to accomplish the task at hand. To make the performance as ideal as possible, several modifications and trade-offs were made with the type and use of sensors to allow the process to run as smoothly as possible. The multiple sensors were integrated into the chassis and motor system, and included color, ultrasonic and touch sensors. The main change that was made was to use color sensors rather than light intensity sensors. Ideally, the light intensity would be different through the lenses and as such, the cleaning apparatus would have been able to centre over the lenses once it was detected. However, it became much more apparent that the use of a color sensor to detect the dark (black) color of the glasses frame could more easily set the boundaries for the cleaning pattern.

Running through the process from the start to the end, a touch sensor was used at the holding base where the glasses are placed. As the servo motor slowly rotates the glasses parallel to the ground, the frame hits a touch sensor when the glasses reach the ideal angle for cleaning. This was crucial in the design because it could signal to the cleaning apparatus to lift itself up and away from the spray mechanism, and through Bluetooth, would signal the cleaning car with the spray bottles to drive up towards the chassis. An ultrasonic sensor was used on the cleaning car to detect how far away from the frame of the robot it would need to drive to before it stopped. There was consideration of using a touch sensor, but it was scrapped because any excess force caused by bumping into the frame would cause it to potentially topple over. The cleaning apparatus waited for one minute as the glasses were sprayed by the robot car.

Once the glasses were sprayed, the cleaning apparatus was lowered down close to a vertical angle and went through a searching motion to detect where the glasses are located. The original placement of the color sensor was in the middle between the two cleaning tools, but it was shifted to the side of the left cleaning tool to detect the outside of the frame. A modification was made to allow the sensor to detect the glasses better, because the color black is what the color sensor is defaulted to. Therefore, bright red Lego bricks were added as part of the glasses stand base structure, and rose to the height of the glasses and surrounding it closely, as can be seen in the pictures. Since the distance from the glasses frame to red bricks was approximately the offset of the color sensor to where the cleaning tools would be centered, it was the most ideal modification to enhance the performance of the program. Once the sensor (now searching for red) detected the red, it would calibrate itself by going to the furthest right edge and centering itself over the glasses so that each cleaning tool was centered over the respective lens.

Once it was centered, the apparatus would wait for the servo motor on the base to rotate the glasses upwards until the bridge hit the touch sensor centered between the cleaning tools. This would indicate that the glasses are at the proper orientation and then the cleaning pattern would start. However, the original design was to have the touch sensor located at the tip of the cleaning tool because that would be the most accurate method of detection. But even after applying cleaning cloth to the tip, there was a potential of scratching the lens and it would also mean that one cleaning tool would be closer to the glasses than the other. The re-design was to put the touch sensor on the edge of the apparatus and it would detect the frame of the glasses. This did not work at all because as the glasses came up, when the touch sensor made contact, before the contact was registered (or the sensor was pushed deep enough in other words), the cleaning apparatus would slide sideways off the glasses. This was due to an uneven surface of the lens, and the inability of the cleaning apparatus to remain perfectly rigid.

The final decision was to put it in between the tools so that it could detect the bridge of the glasses. This was a good idea, but it was very difficult to extend the sensor properly so that it touched the bridge delicately. There was also a high chance of it missing, so the solution was the manually press it. If there was more time, an extension would be to try and aim for a working detection mechanism so that the cleaning apparatus can know when it has made contact with the glasses. A touch sensor making contact with the bridge of the glasses would be the best solution. The last instance for using a sensor was the touch sensor from the beginning. After the glasses are cleaned, the servo motor would rotate it back until the back of the frame hit the touch sensor on the holding base. This would signal the completion of cleaning and retract the cleaning apparatus back to the side of the frame, ending the program. Another touch sensor was located on the opposite side of the holding base, as detailed in the pictures, to allow for the supposed cleaning of the other side of the glasses when the servo motor carries out the same functions 180 degrees opposite on the other side.

# 6.0 Software Design and Implementation

The entire process of cleaning glasses is a structured system that requires a specific order of actions for the process to occur successfully and efficiently. The software was designed around the chronological order of cleaning a pair of glasses, but rethought to be created within the bounds and constraints of using a mechatronics system to clean glasses.

Using well named functions to describe each process, (as the hardware and design of the project was already finished development), each process was created one at a time in the proper order of its execution in the process of cleaning glasses. For example, calibration of the arms of the robot to be properly positioned at the right point within the space of the robot (movement in 3 axes) was the first block or task that was created as it was the first thing that was required to move onto the next step of cleaning glasses (which were to allow the glasses to be sprayed with cleaning solution). Each block or sub-process for the entire process were created and tested one by one before moving onto the next functionality. This allowed for a higher success rate of the functions of the project working with each other in unison.

Each block or task was chosen in coordination with its viability in being created with the given hardware and software as well as its practicality in contributing to the process of cleaning a pair of glasses.

The design of this project is the basis in which it would be functional or not. A great software design would translate to great functionality to give an easier and more straightforward foundation for code to be built off of. The initial design of the project that was decided among the team was slightly deviated in certain aspects of the project to accompany integration of the code with the given materials that were available to the team. This included the change of light intensity sensors to colour sensors to have a more accurate sensing of the glasses and fit within given material constraints.

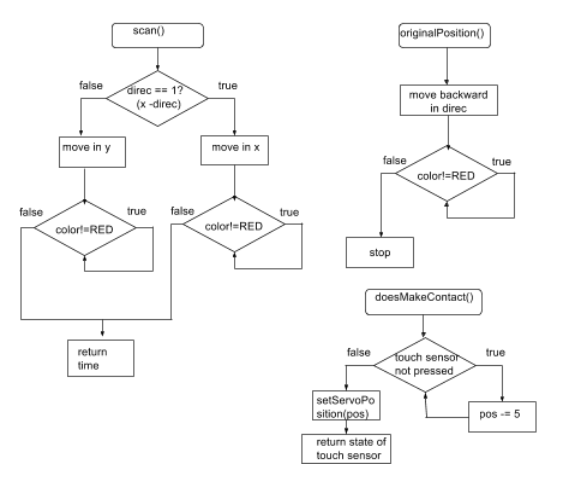
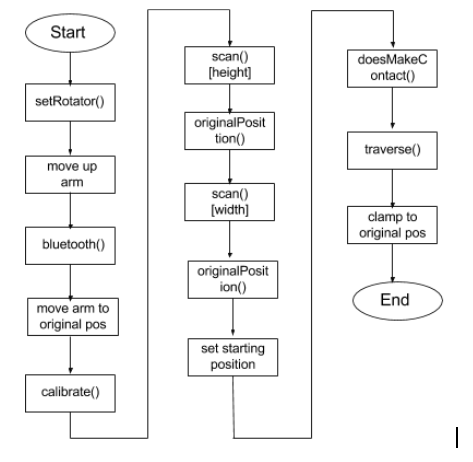
The method of creating this project that the team chose to follow in order to achieve an efficient and working final product was a chronological creation method. From this method of creating our project, testing also occurred in the same manner. It was decided that the mechanical design of the robot would be created first and foremost in accordance to the original design. After creation of most of the hardware, the software integration was created in chronological order as mentioned above and was then tested after each function/task was created until it was functional to satisfactory standards. After it was functional, the next aspect of the project that may require the initial process to be functional was worked on in the same manner until all of the tasks of the project were created and tested. This methodology lead to a successful end product that performed its intended task.

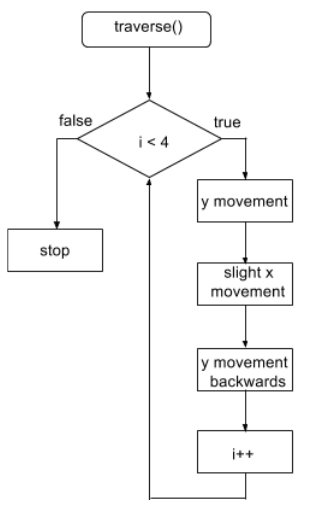
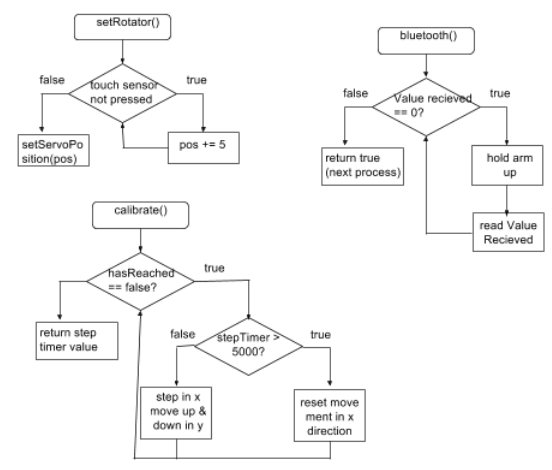
The most significant issue that was interfaced in this project was sensing, making contact with the glasses and moving the cleaning arms. This was difficult as the placement of the touch sensor being used to detect contact with glasses was not in conjunction with how the rotating base of the robot (which gripped the glasses) was designed and created. It took many iterations of problem solving methods that worked with the given code for the contact. In the end, the touch sensor was extended to a position on the arms that allows detection of contact with glasses while still being able to move the arms in the specified cleaning pattern. Tweaking of software values occurred until a good range values satisfied the intended function. This including fixing how high the rotating platform could move in a given restriction placed by the touch sensor.

In conclusion, the software design philosophy of this project was to be able to measure all aspects of cleaning glasses in order for actuation of the mechatronics system in conjunction with the measured values of the glasses to occur properly to satisfy the design problem.

## 6.1 Software Design Flowchart

Figure 5 below shows a flowchart of the software code used for the robotics system (found in Appendix A).

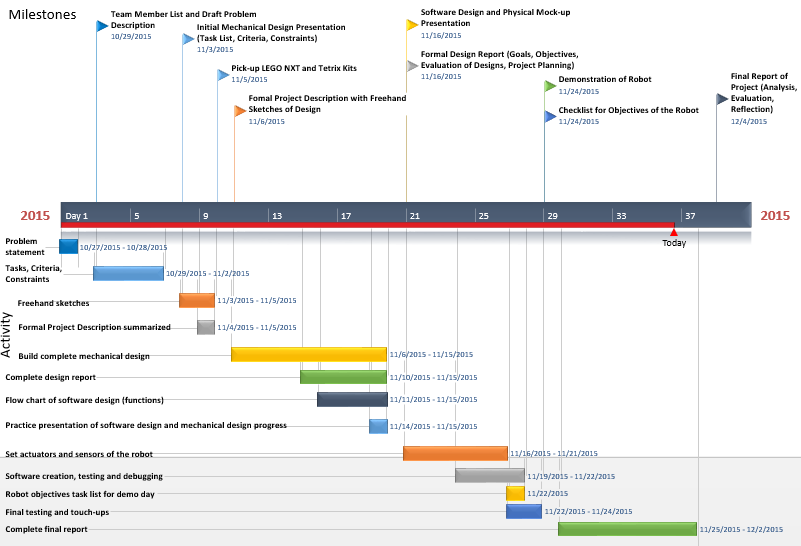


**Figure 5: Software Flowchart**

# 7.0 Project Planning

A new Gantt Chart was created after the completion of the project, as shown in Figure 6.



**Figure 6: Gantt Chart of Final Project Overview**

Changes in the timeline, compared to the design report, were observed during the construction of the prototype. In contrast to the previous Gantt Chart (also shown in Appendix C), the placement of the actuators and sensors of the robot took longer than expected. The software design and coding commenced later than planned; however, finished within a shorter timespan than earlier suggested. The final testing and touch-ups consisted of strengthening loose connections and fixing problems when interfacing the three different types of materials used in the project. This took less time than planned, since more time was spent on the mechanical design than previously predicted.

The entire group worked together on the mechanical design, while the software component was split evenly amongst group members (as shown in the software code in Appendix B). To make the work more productive, the task of setting the actuators (which was one of the most difficult portions of the project) was split between two group members at a time. Then, during occasional testing of the motors, the group members switched, in order to make it easier to point out areas of concern or flaws related to the motors.

# 8.0 Conclusion

In essence, the report details the constraints and criteria of the given problem at hand that the robot created was meant to solve. Looking through the underlying problems of the design that were formulated through the design process by pooling all of the ideas of each group member, the group took steps to modify the mechanical design before coding. This allowed more time to focus on perfecting the machine.

The final design that was put on display was able to fully meet the criteria put forth by the group. The constraints that were fulfilled include the detection of the glasses, and the extension of the cleaning utensils to make contact with the lenses. In addition, the contact was delicate, and did not break the glasses while being able to traverse surface while staying in contact. Constraints were mostly met, and would have been fully met if the conditions under which the robot were performing were ideal. The glasses were sprayed with a fair amount of accuracy, and would have been better if the wires connecting the motors allowed the holding base to have been placed closer to the area where the spraying mechanism would reach the frame. The entirety of both lenses were cleaned, but the right lens was not always cleaned perfectly on every trial. The glasses were easily flipped through the use of a servo motor to rotate them, and for the most part, the cleaning pattern could be utilized to the majority of glasses shapes on the market.

The most important feature of the design remains the x, y, z movement of the cleaning apparatus, because it mimics the motion of movement of a 3D printing machine [3]. The flexibility of developing this mechanism allowed for the cleaning apparatus to have access to any part of the surface of the lens. Creating a movement system that had good traction on the Tetrix® rods in combination with the servo and NXT motors as seen in the pictures quantified the apparatus’s ability to keep contact with the glasses as it was cleaned. The implications and success of this design are enormous, because it could potentially be implemented as a more efficient automated window cleaner for high-rise buildings under ideal conditions [4].

# 9.0 Recommendations

## 9.1 Improvements to Mechanical Design

A major issue with the mechanical design was the sturdiness and viability of the overall structure. There were many loose components and important pieces attached by a lone screw or a flimsy and bent Lego bar. Therefore, a recommended improvement could be to reduce the overall length of the structure, since the longer the structure was, the more the structure would lean to one side. The weight of the NXT brick and the motors all on one side of the robotic system caused the structure to lean to this particular side. Therefore, distributing the weight evenly across the robotic system, with the help of longer wires, would be useful for this design.

Another improvement that could be made to the mechanical design is to use a light intensity sensor as opposed to the colour sensor, which was used to verify and traverse the cleaning bounds. A light sensor would be more useful, since this would eliminate the use of the red blocks to measure these bounds. As well, it would recognize the difference between the frame of the glasses, and the lenses, since the lenses are transparent while the frame would be recognized as opaque.

Additionally, motors on both ends of the x-axis motor movement would be very useful for control purposes. During certain runs of the program, the end of the x-axis bar which had no motor attached seemed fixed upon a bar used to balance the structural weight. This hampered the overall movement of the motor system, since it made it harder for the sensors to detect the red blocks when traversing the lenses. Thus, a motor which would allow for a smoother movement of the other end of the bar would make it much easier to check the bounds of the lenses.

## 9.2 Improvements to Software Design

First and foremost, better developed concepts of control systems integrated with the software of this project would vastly improve the functionality of many aspects of Spectaculum Lautus. For example, the stiffness of the motors could have been improved to aid in a better cleaning of the surface of the glasses.

A more well developed sensor and software interface would aid in a more accurate mechatronics system to allow for the problem to be solved in a more efficient manner. For example, integration of ultrasonic sensors and light intensity sensors with a more refined pattern developing algorithm for cleaning the robot would significantly improve the robot’s functionality.

# 10.0 References

[1] Academia.edu, 'Ways on how to reduce friction A', 2015. [Online]. Available: http://www.academia.edu/4692549/Ways\_on\_how\_to\_reduce\_friction\_A. [Accessed: 01- Dec- 2015].

[2] Learn.adafruit.com, 'What is a Stepper Motor? | All About Stepper Motors | Adafruit Learning System', 2015. [Online]. Available: https://learn.adafruit.com/all-about-stepper-motors/what-is-a-stepper-motor. [Accessed: 01- Dec- 2015].

[3] 3dprintingforbeginners.com, '3D Printing Technology Explained | 3D Printing for Beginners', 2015. [Online]. Available: http://3dprintingforbeginners.com/3d-printing-technology/. [Accessed: 01- Dec- 2015].

[4] Singularity HUB, 'Window Cleaning Robots Making Their Way To Skyscraper Happy United Arab Emirates - Singularity HUB', 2013. [Online]. Available: http://singularityhub.com/2013/03/13/window-cleaning-robots-making-their-way-to-skyscraper-happy-united-arab-emirates/. [Accessed: 01- Dec- 2015].

# Appendix A: Source Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Cleaning Device (Main) Robot - Final Project Software Code

//Himel Mondal, Rimpey Diaz, Siddhant Chandgadkar, Brandon Fong

#include "NXTServo-lib-UW.c" //See library after source code (pages 24-31)

/\*Calibrate- Scans the bounds of the area of the robot

until a red starting point is found \*/

//Written by Himel Mondal

int calibrate(){

time1[T1] = 0;

bool hasReached = false;

int stepTimer = 0;

while(!hasReached){

//Motor movements looped in here until starting point found

if(stepTimer > 5000){

//When x-direction bounds reached in step timer

setServoSpeed(S1,2,-4,-15,12);

//go back to original position

wait1Msec(stepTimer);

setServoSpeed(S1,2,0,-15,12);

stepTimer = 0;

wait1Msec(200);

}

setServoSpeed(S1,2,4,-15,12); //steps in x-direc for 1000ms

wait1Msec(1000);

stepTimer += 1000;

setServoSpeed(S1,2,0,-15,12);

wait1Msec(300);

time1[T1] = 0;

setServoSpeed(S1,1,-7, -11, 19);

while(SensorValue[S3] != REDCOLOR && time1[T1] <= 2500){}

// move in y-direction until redblock is found or 2500 ms pass

setServoSpeed(S1,1,0, -11, 19);

if(SensorValue[S3] == REDCOLOR)

//calibrate ends if red is reached

{

hasReached = true;

}

else{

setServoSpeed(S1,1,6, -11, 19);

//moves back to original position in y-direction

wait1Msec(2500);

setServoSpeed(S1,1,0, -11, 19);

}

}

return stepTimer;

}

/\*Scans the x and y direction for dimensions of glasses indicated

by red blocks \*/

//Written by Siddhant Chandgadkar

int scan(int direc){ //takes in scan direction

time1[T2] = 0;

if(direc == 1){

setServoSpeed(S1,1,-7, -11, 19);

//will move forward until next red block reached in x - direc

wait1Msec(3000);

while(SensorValue[S3] != REDCOLOR){}

setServoSpeed(S1, 1, 0, -11,19);

}

else if(direc == 2){

//will move sideways until next red block reached in y-direc

setServoSpeed(S1,2,7, -15, 12);

wait1Msec(3000);

while(SensorValue[S3] != REDCOLOR){}

setServoSpeed(S1, 2, 0, -15,12);

}

return time1[T2];

//returns the time it takes to move in either x or y direction

}

/\* Communicates with the sprayer bot until it reaches the base \*/

//Written by Brandon Fong

bool bluetooth (){

ubyte valueReceived[1];

valueReceived[0] = 0;

while(valueReceived[0] == 0){

//will search until a value from sprayer bot returned

motor[motorB] = 4; //keeps arms lifted until sprayed

wait1Msec(800);

motor[motorB] = 0;

wait1Msec(5);

cCmdMessageRead (valueReceived, 1, 2);

//valueReceived indicates the current state of sprayer bot

displayString(0,"%d",valueReceived[0]);

//1 is the length of the array, 2 is the bluetooth channel

}

return true;

}

/\* Returns the arms to its original calibrated position \*/

//Written by Rimpey Diaz

void originalPosition(int spot, int speed, int negOff, int posOff){

//takes in different motors and their values

wait1Msec(1000);

setServoSpeed(S1,spot,speed,negOff,posOff);

wait1Msec(3000);

while(SensorValue[S3] != REDCOLOR){}

//Moves as specified in parameters until original red block reached

setServoSpeed(S1,spot,0,negOff,posOff);

}

/\* Sets the rotating arm in an optimal downward position for cleaning/scanning/etc. \*/

//Written by Brandon Fong

void setRotator(int &pos){

while(SensorValue[S2] == 0){

//will spin until touch sensor pressed

wait1Msec(1000);

setServoPosition(S1, 5, pos);

pos += 5;

}

setServoPosition(S1, 5, pos);

}

/\* Well-named function that returns when the arm makes contact with cleaning arms \*/

//Written by Rimpey Diaz

bool doesMakeContact(int &pos){

while(SensorValue[S4] == 0){

//moves glass clamp until touch sensor pressed

setServoPosition(S1, 5, pos);

wait1Msec(1000);

pos-=5;

}

setServoPosition(S1, 4, pos);

return (SensorValue[S4] == 1); //returns the state of the touch sensor

}

/\*main cleaning function (arm pattern movement) \*/

//Written by Siddhant Chandgadkar

void traverse(int heightTime, int widthTime){

//Takes in dimensions from the scan functions

for (int i = 0; i < 4; i++){

setServoSpeed(S1,1,-7, -11,19);

//for 4 iterations, moves the arm in a pattern in x, y direction

motor[motorB] = -1; //Movements based on scan dimension times

wait1Msec(heightTime-300);

setServoSpeed(S1,1,0, -11,19);

motor[motorB] = 0;

setServoSpeed(S1,2,-7, -15,12);

wait1Msec(widthTime/4);

setServoSpeed(S1,2,0, -15,12);

setServoSpeed(S1,1,7,-11,19);

motor[motorB] = 1;

wait1Msec(heightTime-300);

setServoSpeed(S1,1,0,-11,19);

motor[motorB] = 0;

wait1Msec(400);

}

}

//Written by Himel Mondal

task main()

{

/\* Declarations \*/

SensorType[S1] = sensorI2CCustom9V;

SensorType[S3] = sensorColorNxtFULL;

SensorType[S2] = sensorTouch;

SensorType[S4] = sensorTouch;

setServoSpeed(S1,2,0, -15, 12);

setServoSpeed(S1,1,0, -11, 19);

motor[motorB] = 0;

nMotorEncoder[motorB] = 0;

displayString(0,"%d", SensorValue[S2]);

/\* Sets rotators initial position \*/

int pos = 0;

setRotator(pos);

displayString(0,"%d",pos);

wait1Msec(1500);

/\* Moves arm up to be sprayed \*/

motor[motorB] = 20;

while(nMotorEncoder[motorB] < 90){}

motor[motorB] = 0;

/\* calls bluetooth function \*/

bool communication = bluetooth();

displayString(0,"%s", communication);

/\*sets the arm that moves the cleaning arms back to original position\*/

motor[motorB] = 0;

wait1Msec(2000);

motor[motorB] = -7;

wait1Msec(1000);

motor[motorB] = 0;

/\* Calibrate function called \*/

int step = calibrate();

/\* Height Scan \*/

int hTime = scan(1);

displayString(0,"%d", hTime);

originalPosition(1,5,-11,19);

/\* Width Scan \*/

int wTime = scan(2);

displayString(0,"%d", wTime);

originalPosition(2,-5,-15,12);

wait1Msec(1000);

/\*Sets appropriate starting position to clean\*/

setServoSpeed(S1,1,-7,-11,19);

wait1Msec(hTime/4);

setServoSpeed(S1,1,0,-11,19);

setServoSpeed(S1,2,7,-15,12);

wait1Msec(wTime/4);

setServoSpeed(S1,2,0,-15,12);

/\*Checks if rotating clamp makes contact with cleaning arms \*/

bool contact = doesMakeContact(pos);

displayString(0,"%s", contact);

wait1Msec(3000);

/\* Traverse function called \*/

if(contact){

traverse(hTime,wTime);

}

wait1Msec(500);

/\* sets the rotating clamp back to original position \*/

setRotator(pos);

/\* Moves x and y direction motors back to their orignal position \*/

setServoSpeed(S1,2,-4,-15,12);

wait1Msec(step - (wTime/2));

setServoSpeed(S1,2,0,-15,12);

setServoSpeed(S1,1,6, -11, 19);

wait1Msec(2000);

setServoSpeed(S1,1,0, -11, 19);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Cleaning Solution Robot - Final Project Software Code

//Himel Mondal, Rimpey Diaz, Siddhant Chandgadkar, Brandon Fong

//Written by Siddhant Chandgadkar

task main()

{

SensorType[S1] = sensorSONAR;

motor[motorA] = 40;

while(SensorValue[S1]>8){} //moves forward until 8cm away from base

motor[motorA] = 0;

wait1Msec(60000); //1 minute prep time for spraying glasses

ubyte valueToSend[1];

valueToSend[0] = 1;

cCmdMessageWriteToBluetooth(valueToSend,1,2);

//sends 1 to base to indicate it can continue cleaning

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Filename: NXTServo-lib-UW.c

Date: June 6, 2015

File Version: 1.3

A wrapper for the drivers provided by Mindsensor for use with the

NXTServo interface module.

History

Ver Date Comment

1.4 Jun 6/15 included code from NXTServo-lib.c (which was updated for

parameter passing in RobotC v4.30); removed fixed deadband

correction since it is different for each motor; reduced maximum

gripper angle by 5 degrees;

1.3 Jan 15/15 changed offset values and scaling

1.2 Jan 14/15 scaled speed setting by 4. previous range from -100 to 100.

updated range now -400 to 400 with the function call range

still -100 to 100.

1.1 Jan 12/15 restrict to valid servo numbers 1-7; added SetGripperPos

to limit gripper servo position; limit servo position

to 0 to 180; added constants, valid setting function,

parameter check function;

generalized to use any NXT sensor port;

1.0 Dec 18/14 original release

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

NXTServo-lib.c

This is function library to use with mindsensors.com NXTServo module.

For use with RobotC.

V1.0

2008(c) mindsensors.com

for more info visit www.mindsensors.com

History:

When Ahthor/Editor Comments

06/06/15 C. Hulls Convert I2C functions to pass pointers to message arrays;

Removed extra ; from empty loopsl

05/29/08 Deepak Patil Initial creation of the file.

03/21/11 support Added support for Port and i2c address in API parameters

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NXTServo-lib.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*===========================================================================

Register Summary

===================

Register Read Operation Write Operation

0x00 sw version control reg

0x01 Servo 1 speed Servo 1 speed

0x02 Servo 1 Position low byte in uS Servo 1 Position low byte in uS

0x03 Servo 1 Position hi byte in uS Servo 1 Position hi byte in uS

0x04 Servo 2 speed Servo 2 speed

0x05 Servo 2 Position low byte in uS Servo 2 Position low byte in uS

0x06 Servo 2 Position hi byte in uS Servo 2 Position hi byte in uS

0x07 Servo 3 speed Servo 3 speed

0x08 Servo 3 Position low byte in uS Servo 3 Position low byte in uS

0x09 Servo 3 Position hi byte in uS Servo 3 Position hi byte in uS

0x0a Servo 4 speed Servo 4 speed

0x0b Servo 4 Position low byte in uS Servo 4 Position low byte in uS

0x0c Servo 4 Position hi byte in uS Servo 4 Position hi byte in uS

0x0d Servo 5 speed Servo 5 speed

0x0e Servo 5 Position low byte in uS Servo 5 Position low byte in uS

0x0f Servo 5 Position hi byte in uS Servo 5 Position hi byte in uS

0x10 Servo 6 speed Servo 6 speed

0x11 Servo 6 Position low byte in uS Servo 6 Position low byte in uS

0x12 Servo 6 Position hi byte in uS Servo 6 Position hi byte in uS

0x13 Servo 7 speed Servo 7 speed

0x14 Servo 7 Position low byte in uS Servo 7 Position low byte in uS

0x15 Servo 7 Position hi byte in uS Servo 7 Position hi byte in uS

0x16 Servo 8 speed Servo 8 speed

0x17 Servo 8 Position low byte in uS Servo 8 Position low byte in uS

0x18 Servo 8 Position hi byte in uS Servo 8 Position hi byte in uS

0x19 battery voltage None

================

Psudo registers

================

0x1A servo position 1 nothing

0x1b servo position 2 nothing

0x1c servo position 3 nothing

0x1d servo position 4 nothing

0x1e servo position 5 nothing

0x1f servo position 6 nothing

0x20 servo position 7 nothing

0x21 servo position 8 nothing

\*/

#pragma SystemFile

const int kSc8CommandReg = 0x41;

const int kSc8\_speed = 0x52;

const int kSc8\_lowbyte = 0x42;

const int kSc8\_hibyte = 0x43;

const int kSc8\_Vbatt = 0x41;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UW code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// available servo numbers

const int MIN\_SERVO\_NUM = 1;

const int MAX\_SERVO\_NUM = 7; // assumes SV8 used for power

// continous servo

const int MAX\_SERVO\_SPEED = 100; // max at +/- 100

const int SERVO\_NEUTRAL = 1550; // 1550 us is neutral position (off)

// standard servo position limits

const int MIN\_SERVO\_ANGLE = 0; // numbers are from datasheets, verified w/ servo

const int MAX\_SERVO\_ANGLE = 180;

const int SERVO\_OFFSET = 90; // make zero to be the neutral position

const int SERVO\_ZERO = 600; // 600 us is 0 angle for servo

const int MIN\_GRIP\_ANGLE = 50; // numbers are based on tests with gripper

const int MAX\_GRIP\_ANGLE = 120;

// default I2C address (factory setting)

const int I2C\_ADDR = 0xb0;

// control register

const ubyte CONTROL\_REG\_ADDR = 0x41;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NXTServo-lib.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*===================================

\*\*

\*\* Read the battery voltage data from

\*\* NXTServo module (in mili-volts)

\*\*

===================================\*/

int NXTServo\_Get\_Batt\_V(tSensors kSc8Port, int kSc8ID)

{

ubyte sc8Msg[5];

const int kMsgSize = 0;

const int kSc8Address = 1;

const int kReadAddress = 2;

ubyte replyMsg[2];

// Build the I2C message

sc8Msg[kMsgSize] = 2;

sc8Msg[kSc8Address] = kSc8ID ;

sc8Msg[kReadAddress] = kSc8\_Vbatt ;

while (nI2CStatus[kSc8Port] == STAT\_COMM\_PENDING) // CH removed ;

{

// Wait for I2C bus to be ready

}

// when the I2C bus is ready, send the message you built

sendI2CMsg(kSc8Port, &sc8Msg[0], 1); // CH added &

while (nI2CStatus[kSc8Port] == STAT\_COMM\_PENDING) // CH removed ;

{

// Wait for I2C bus to be ready

}

// when the I2C bus is ready, send the message you built

readI2CReply(kSc8Port, &replyMsg[0], 1); // CH added &

return(37\*(0x00FF & replyMsg[0])); // 37 is calculated from

//supply from NXT =4700 mv /128

}

/\*===================================

\*\*

\*\* Quick setup for servo.

\*\* Valid values are from 50 to 250

\*\* uses pseudo registers from 0x1A to 0x21

\*\*

===================================\*/

void NXTServo\_Quick\_Servo\_Setup(tSensors kSc8Port, int kSc8ID, int servoNumber, int position)

{

ubyte sc8Msg[5];

const int kMsgSize = 0;

const int kSc8Address = 1;

const int kSc8Servo = 2;

const int kData = 3;

// Build the I2C message

sc8Msg[kMsgSize] = 3;

sc8Msg[kSc8Address] = kSc8ID;

sc8Msg[kSc8Servo] = 0x59+servoNumber ;

sc8Msg[kData] = (ubyte)position ;

while (nI2CStatus[kSc8Port] == STAT\_COMM\_PENDING) // CH removed ;

{

// Wait for I2C bus to be ready

}

// when the I2C bus is ready, send the message you built

sendI2CMsg(kSc8Port, &sc8Msg[0], 0); // CH added &

}

/\*===================================

\*\*

\*\* Set the speed of a servo to

\*\* given value.

\*\* highest speed -> 0

\*\* lowest speed -> 255

\*\*

===================================\*/

void NXTServo\_SetSpeed(tSensors kSc8Port, int kSc8ID, int servoNumber, int speed)

{

ubyte sc8Msg[5];

const int kMsgSize = 0;

const int kSc8Address = 1;

const int kSc8Servo = 2;

const int kData = 3;

// Build the I2C message

sc8Msg[kMsgSize] = 4;

sc8Msg[kSc8Address] = kSc8ID;

sc8Msg[kSc8Servo] = kSc8\_speed+servoNumber-1 ;

sc8Msg[kData] = speed ;

while (nI2CStatus[kSc8Port] == STAT\_COMM\_PENDING) // CH removed ;

{

// Wait for I2C bus to be ready

}

// when the I2C bus is ready, send the message you built

sendI2CMsg(kSc8Port, &sc8Msg[0], 0); // CH added &

}

/\*===================================

\*\*

\*\* Set the position of a servo to

\*\* given value.

\*\*

===================================\*/

void NXTServo\_SetPosition(tSensors kSc8Port, int kSc8ID, int servoNumber, int position)

{

NXTServo\_Quick\_Servo\_Setup(kSc8Port, kSc8ID, servoNumber, position/10);

}

/\*

\*

\*

\*

\*/

/\*

void NXTServo\_SetPosition(tSensors kSc8Port, int kSc8ID, int servoNumber, int position)

{

byte sc8Msg[5];

const int kMsgSize = 0;

const int kSc8Address = 1;

const int kSc8Servo = 2;

const int kData\_low = 3;

const int kData\_hi = 4;

// Build the I2C message

sc8Msg[kMsgSize] = 4;

sc8Msg[kSc8Address] = kSc8ID;

sc8Msg[kSc8Servo] = kSc8\_lowbyte+2\*servoNumber-2 ;

sc8Msg[kData\_low] = (ubyte)position ;

sc8Msg[kData\_hi] = position/0x100;

while (nI2CStatus[kSc8Port] == STAT\_COMM\_PENDING) // CH removed ;

{

// Wait for I2C bus to be ready

}

// when the I2C bus is ready, send the message you built

sendI2CMsg(kSc8Port, &sc8Msg[0], 0); // CH added &

}

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UW code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setValidSetting(int & setting, int minSetting, int maxSetting)

{

if (setting > maxSetting)

setting = maxSetting;

else if (setting < minSetting)

setting = minSetting;

}

bool paramIsValid(tSensors nxtPort, int servoNumber)

{

return (nxtPort == S1 || nxtPort == S2 || nxtPort == S3 || nxtPort == S4)

&& servoNumber >= MIN\_SERVO\_NUM && servoNumber <= MAX\_SERVO\_NUM;

}

void setServoSpeed(tSensors nxtPort, int servo\_number, int speed\_setting, int neg\_offset = 0, int pos\_offset = 0)

{

if (paramIsValid(nxtPort,servo\_number))

{

setValidSetting(speed\_setting, -MAX\_SERVO\_SPEED, MAX\_SERVO\_SPEED);

if (speed\_setting == 0)

NXTServo\_SetPosition(nxtPort, I2C\_ADDR, servo\_number, SERVO\_NEUTRAL);

else if (speed\_setting > 0)

NXTServo\_SetPosition(nxtPort, I2C\_ADDR, servo\_number, SERVO\_NEUTRAL + (speed\_setting + pos\_offset) \* 4);

else

NXTServo\_SetPosition(nxtPort, I2C\_ADDR, servo\_number, SERVO\_NEUTRAL + (speed\_setting + neg\_offset ) \* 4);

}

}

void setServoPosition(tSensors nxtPort, int servo\_number, int position)

{

if (paramIsValid(nxtPort,servo\_number))

{

// zero is neutral position

position = position + SERVO\_OFFSET;

setValidSetting(position, MIN\_SERVO\_ANGLE, MAX\_SERVO\_ANGLE);

//now convert to servo signals and send. The servo takes a signal from 600uS (-90 degrees) to 2400uS (90 degrees)

NXTServo\_SetPosition(nxtPort, I2C\_ADDR, servo\_number, SERVO\_ZERO + (position)\*10);

}

}

/\*

Set gripper position based on angle - normed to allow 0 to 70 as valid values

\*/

void setGripperPosition(tSensors nxtPort, int servo\_number, int position)

{

if (paramIsValid(nxtPort,servo\_number))

{

// zero commanded position is minimum angle

position = position + MIN\_GRIP\_ANGLE;

setValidSetting(position, MIN\_GRIP\_ANGLE, MAX\_GRIP\_ANGLE);

//now convert to servo signals and send. The servo takes a signal from 600uS (-90 degrees) to 2400uS (90 degrees)

NXTServo\_SetPosition(nxtPort, I2C\_ADDR, servo\_number, SERVO\_ZERO + position\*10);

}

}

void resetGripper(tSensors nxtPort, int servoNumber)

{

setGripperPosition(nxtPort, servoNumber, 90 - MIN\_GRIP\_ANGLE);

}

/\*

The two functions below have not been tested. They should reset the interface to its

factory settings. They can be tested after the first problems with an interface,

otherwise at the moment it is best to leave things as is.

// I2C message send is copied from the code in NXTServo-lib.c

void NXTServo\_SendCommand(tSensors port, ubyte cmd)

{

ubyte sc8Msg[5];

const int kMsgSize = 0;

const int kSc8Address = 1;

const int kSc8Register = 2;

const int kData = 3;

// Build the I2C message

sc8Msg[kMsgSize] = 3;

sc8Msg[kSc8Address] = I2C\_ADDR;

sc8Msg[kSc8Register] = CONTROL\_REG\_ADDR;

sc8Msg[kData] = cmd;

while (nI2CStatus[port] == STAT\_COMM\_PENDING)

{

// Wait for I2C bus to be ready

}

// when the I2C bus is ready, send the message you built

sendI2CMsg(port, &sc8Msg[0], 0);

}

void resetServoInterface(tSensors port)

{

NXTServo\_SendCommand(port, (ubyte)'S');

}

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright(c) 2014-2015 D. Lau, M. Stachowsky, and C.C.W. Hulls, P.Eng.

Students, staff, and faculty members at the University of Waterloo

are granted a non-exclusive right to copy, modify, or use this

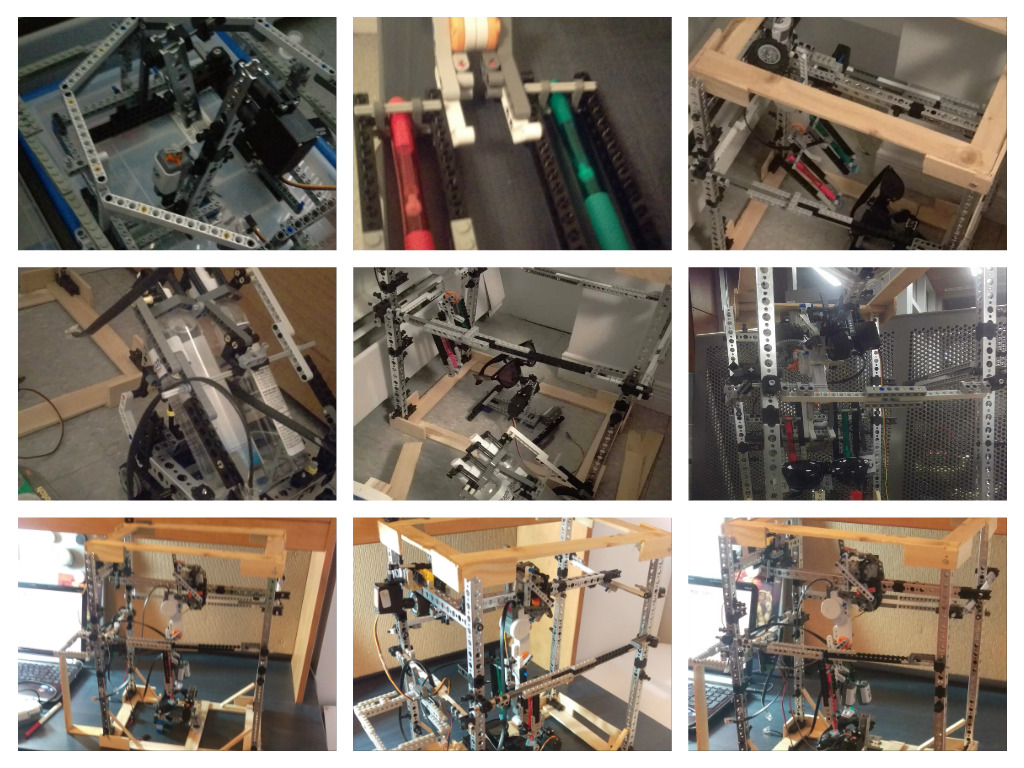
software for non-commercial teaching, learning, and research purposes

provided the author(s) are acknowledged except for as noted below.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# Appendix B: Spectaculum Lautus Photos

All the photos in this album were taken from the start of the construction of the project, until the finished project, in chronological order. Pictures were not repeated from the body of the report.



# Appendix C: Checklist / Task List on Project Demo Day

* Angular position of glasses set for most ideal cleaning method
* Can locate starting point for glasses
* Can determine dimensions of glasses (x, y distances)
* Can allow for an easy spray of glasses
* External robot communicates successfully to initiate cleaning glasses
* Cleaning devices can make contact with the glasses
* Cleaning devices clean glasses in a pattern based on height and width

# Appendix D: Project Planning Gantt Chart from Design Report

This figure is included to compare with the change in the Gantt Chart of the final report.

