

## Stair Climbing Robot

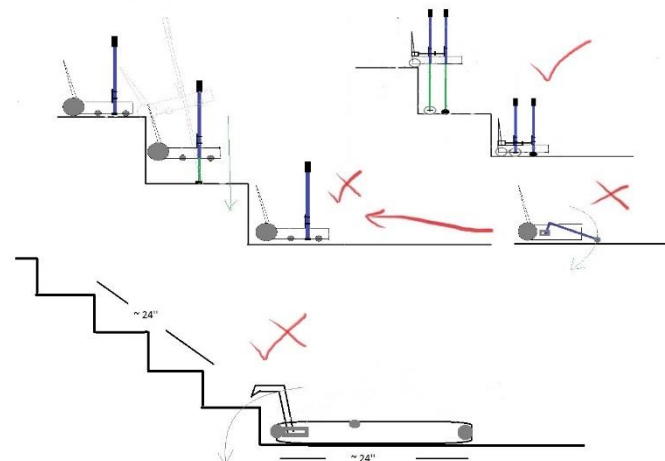
**Ayman Homadi, Electrical Engineering**

Sponsored by Dr. Blandford (UE)

Project advisor: Dr. Blandford

April 26, 2018

Evansville, Indiana



## **Acknowledgments**

I would like to give special thanks to Dr. Dick Blandford (project sponsor and adviser), Dr. Christina Howe (student adviser), Mr. Jeff Cron, and the rest of the EE faculty at the University of Evansville for the Guidance and Motivation.

## **Table of Contents**

I. Introduction

II. Statement of the Problem

III. Design Approach

A. Hardware Design

B. Software Design

C. Standards and Constraints

D. Costs

IV. Result

VI. Conclusion

## **List of Figures**

1. First design concept
2. First design
3. Second design concept
4. Third design concept

5. Fourth design concept
6. Fourth design
7. Pressure sensor used as a weight sensor
8. Circuit diagram of the weight sensor
9. PCB board
10. Fourth design (final robot)
11. Circuit diagram of the final robot
12. Android application for controlling the robot wirelessly
13. Code blocks of the Android application
14. 3D arms of the first design
15. Robot's paths to the second stairs

### **List of Tables**

1. Costs

## **I. Introduction**

Today's robots are becoming an essential aspect of our modern lives. After all, the number of robots used in many everyday applications has significantly increased in the last decade, and the variation of tasks and designs of these robots is high, making them suitable for many applications. In fact, the number of robots, both industrial and service robots, has increased from 4.5 million in 2006 to 8.6 million in 2008, as mentioned in [1]. Likewise, there is an enormous number of stair climbing robots, each of which has a unique design to accomplish certain tasks. Thus, there is usually at least one suitable design of stair climbing robots that can be used in a certain application involving climbing the stairs. Electric stair-climbing wheelchairs, for example, are used to help disabled people to climb the stairs, and many designs of stair climbing robots can be used for military and safety purposes. Moreover, designs of stair climbing robots can be used and implemented within the platforms of many other robots to make these robots capable of climbing the stairs. Despite the huge number of available designs of stair climbing robots, sufficient designs are limited. In fact, a sufficient design of a stair climbing robot is obtained after engineers study many other designs of stair climbing robots and learn from these designs. As a result, several creative designs of stair climbing robots were tested and compared in this senior project, adding new ideas to the field of stair climbing robots, and the finest design obtained was used to build a robot meeting the client's requirements of this senior project.

## **II. Statement of the Problem**

Creating the designs and selecting the right components for these designs were the core challenges of this project. In details, many elements had to be considered when creating the designs of stair climbing robots. First, all designs had to use similar components to reduce the total cost of this project. Secondly, before building any prototype, all designs had to pass theoretical tests for all cases, such as testing the performances of these designs theoretically when the robot was carrying an object. Otherwise, a lot of effort, time and money could have been wasted. Furthermore, when acquiring any component for the prototypes of the final designs, many factors had to be considered, such as the weight, price, and strength of this component. Similarly, high torque motors are usually used for such robots, but there are many high torque motors with slightly different RPMs and weights. Thus, many factors had to be accounted for when building the prototypes. In addition, the implementation of any design could acquire a designer to modify the design for several reasons like design inaccuracy or unavailability of a certain component, adding more challenges to this project. Therefore, creating reliable designs and choosing suitable components for these designs were the main challenges of this senior project.

#### Client Requirements:

1. The robot must be capable of climbing the stairs of the Koch Center all the way to the top floor.
2. The robot must be programmed using the STM32F407G-DISC1 microcontroller [2], and it must be self-operating.
3. The robot must carry a 1-lb object.

### III. Design Approach

To meet the client's requirements, several designs of stair climbing robots, which require similar components, were tested. Likewise, all proposed robots here theoretically should meet the client's main requirement, which is climbing the stairs while carrying a 1-lb object.

#### A. Hardware Design

The first proposed design was assembled, tested, and is shown in Figures 1 and 2.

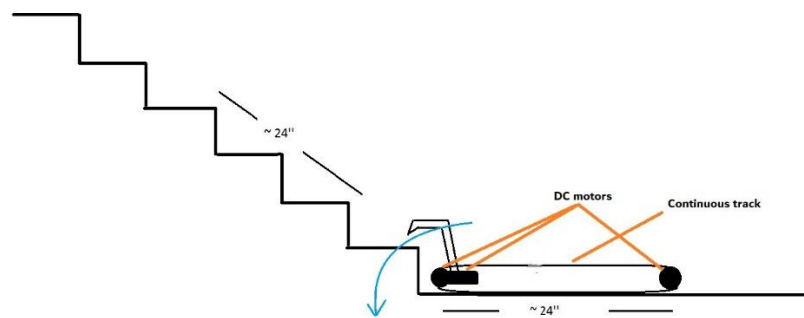


Figure 1: First design concept

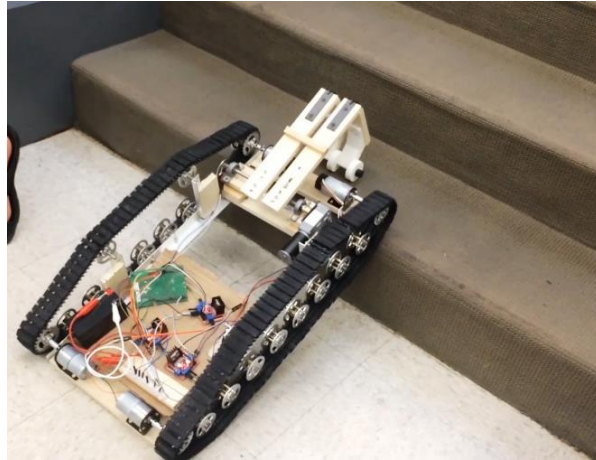


Figure 2: First design

In this design, two rotating arms, which are installed at the front end of the robot, help the robot climb the first few steps. After the first steps, the robot should be able to climb steadily all the way to the top since its length is about 24". This robot relies heavily on the torque produced by the motors and the attraction between the continuous tracks and stairs. Unfortunately, this design would require custom rubber tank wheels, which cost over \$300, and the weight of the robot need to be further reduced.

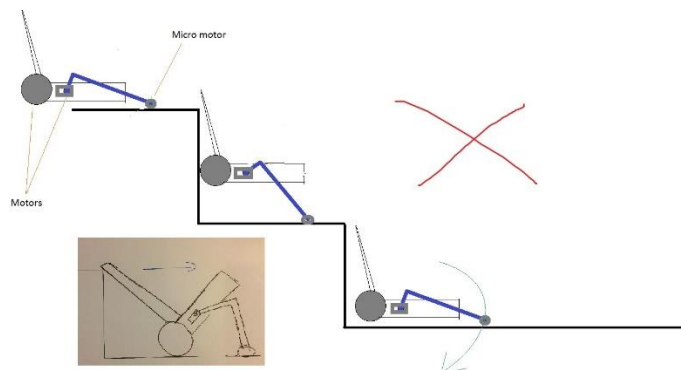


Figure 3: Second design concept



Moreover, the second design, which is shown in Figure 3, was also tested, and a critical problem was found in this design. In this design, the robot has one big arm that rotates downward and two micromotors, which are installed at the end of the rotating arm. The robot climbs each step using the rotating arm and the micromotors. That is, the rotating arm lifts the robot. Then, the micromotors move the robot toward the next step. However, after few experimental tests, it turned out that the rotating arm pushes the robot backward toward the previous step instead of upward because most of the weight of the robot is concentrated at the front. Thus, this design is flawed.

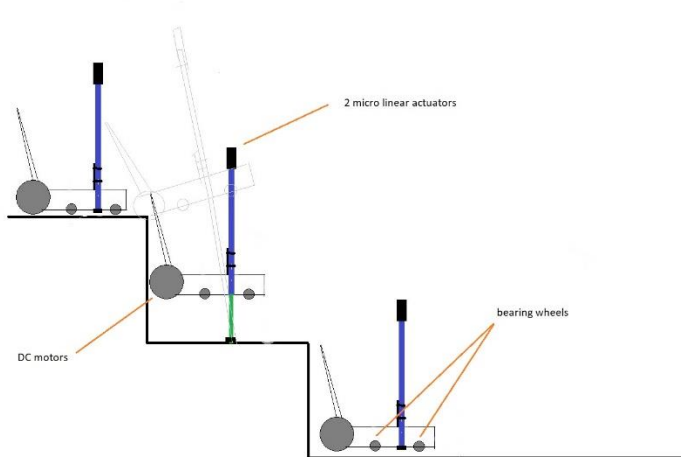


Figure 4: Third design concept

In addition, the third design is shown in Figure 4, and it uses two micro linear actuators. The linear DC motors here lift the robot to the next step while the robot is leaning on the step. This design was tested using 10" linear actuators, but it did not work. It appears that this design

requires at least 16” micro linear actuators, and the maximum length of standard micro linear actuators that are available today is 12”. Therefore, custom linear DC motors need to be built for this design to work.

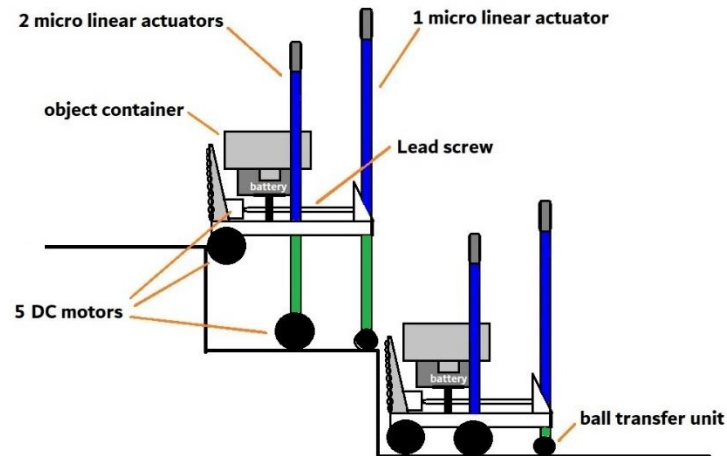


Figure 5: Fourth design concept

Furthermore, the fourth design is shown in Figure 5, and it uses three linear DC motors instead of two. The robot in this design can climb each step by using five DC motors, one of which is used to balance the robot, and three linear actuators. In this design, the robot lifts using the linear actuators while it is leaning on the next step until it reaches the top of the next step. Then, it uses the front DC motors to move forward and compresses the strokes of the linear DC motors in the middle. Finally, it moves forward again and compresses the stroke of the rear linear DC motor. Likewise, this design includes a DC motor connected to a lead screw to move the heaviest parts of the robot, battery and the carried object, from the front to the back ends of the

robot and vice versa. This design requires that the weight of the front and back ends of the robot to be changed twice each time the robot climbs a step. This design was tested on the stairs of the Koch Center, as shown in figure 6, and the robot successfully climbed the stairs. It was also observed that the robot was capable of climbing the stairs without using the weight shifter when the weight of the object was less than 0.5 lb. As a result, this robot was slightly modified and used as a final robot, and the DC motor that balances the weight of the robot was kept for safety enhancement.



Figure 6: Fourth design

Moreover, a pressure sensor was added to the robot to estimate the weight of the carried object, ensuring that the object does not exceed the allowed weight, as depicted in Figure 7. Also, Figure 8 shows a circuit diagram explaining how the pressure sensor was used to measure the weight of the carried object. Furthermore, a 4x20 LCD screen was installed at the back of the robot to update the user with robot's status and to indicate if the robot is overloaded.



Figure 7: Pressure sensor used as a weight sensor

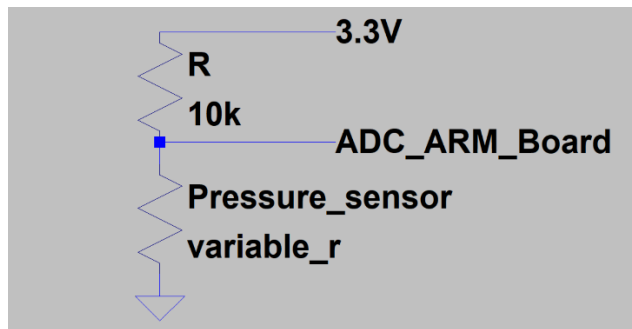


Figure 8: Circuit diagram of the weight sensor

A PCB board was also designed to hold the ARM board and organize the wires. As can be seen in figure 9, the PCB board also has five LEDs near the lower right corner. If the LCD screen is not working for any reason, the user can estimate the weight of the carried object by counting the number of LEDs that are on. In other words, if all five LEDs are on, the weight is

greater than 1 lb, but if two LEDs are on, the weight of the object is about 0.5 lb. Thus, there is a high chance the robot is overloaded if all LEDs are on. Moreover, this PCB board was ordered from ExpressPCB [3].

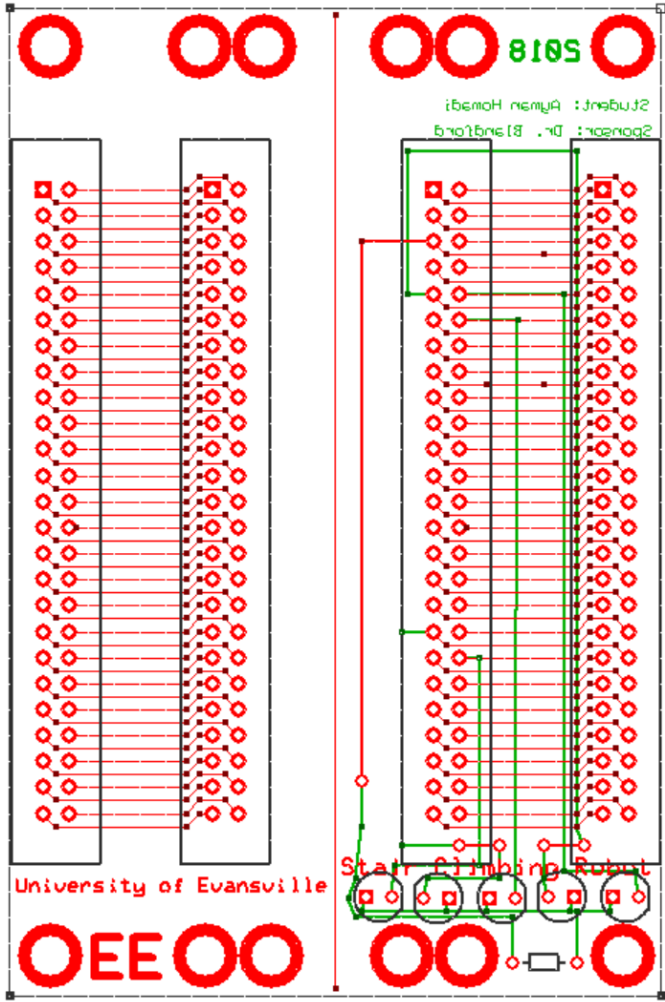


Figure 9: PCB board

Figure 10 shows pictures of the final robot after adding some components, including an LCD screen, a weight sensor, a Bluetooth module, and sharp sensors. Figure 11 presents a complete circuit diagram of the final robot. It is also important to mention that the robot is

powered using a 12 V lead-acid battery because all the DC motors on the robot need a 12 V source. The 12 V is also reduced to 5 V using the voltage regulator on the L298N Dual H-bridge board to power the microcontroller and the other small components.

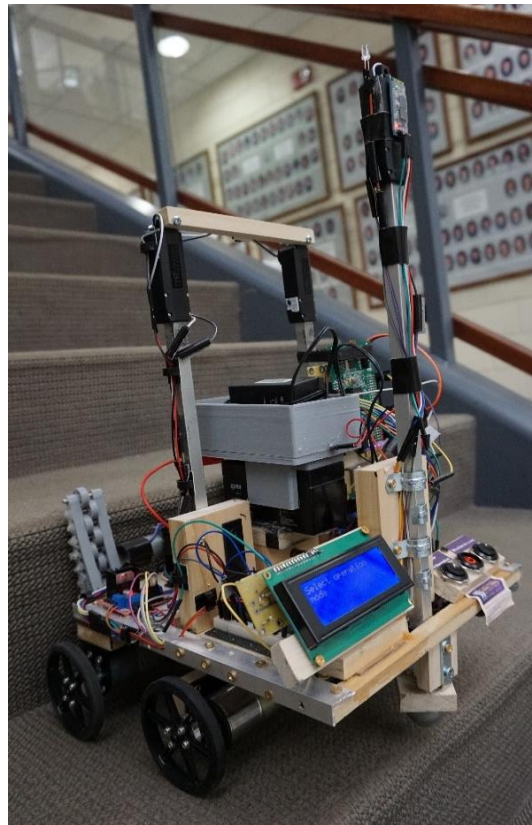
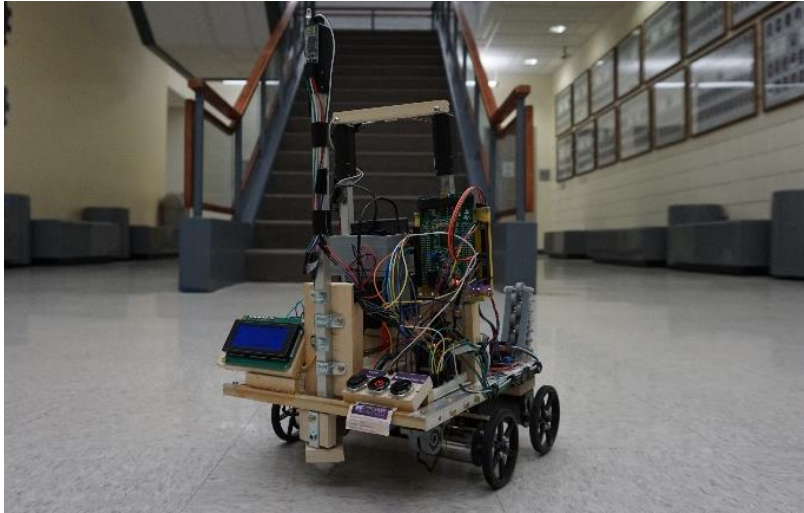


Figure 10: Fourth design (final robot)

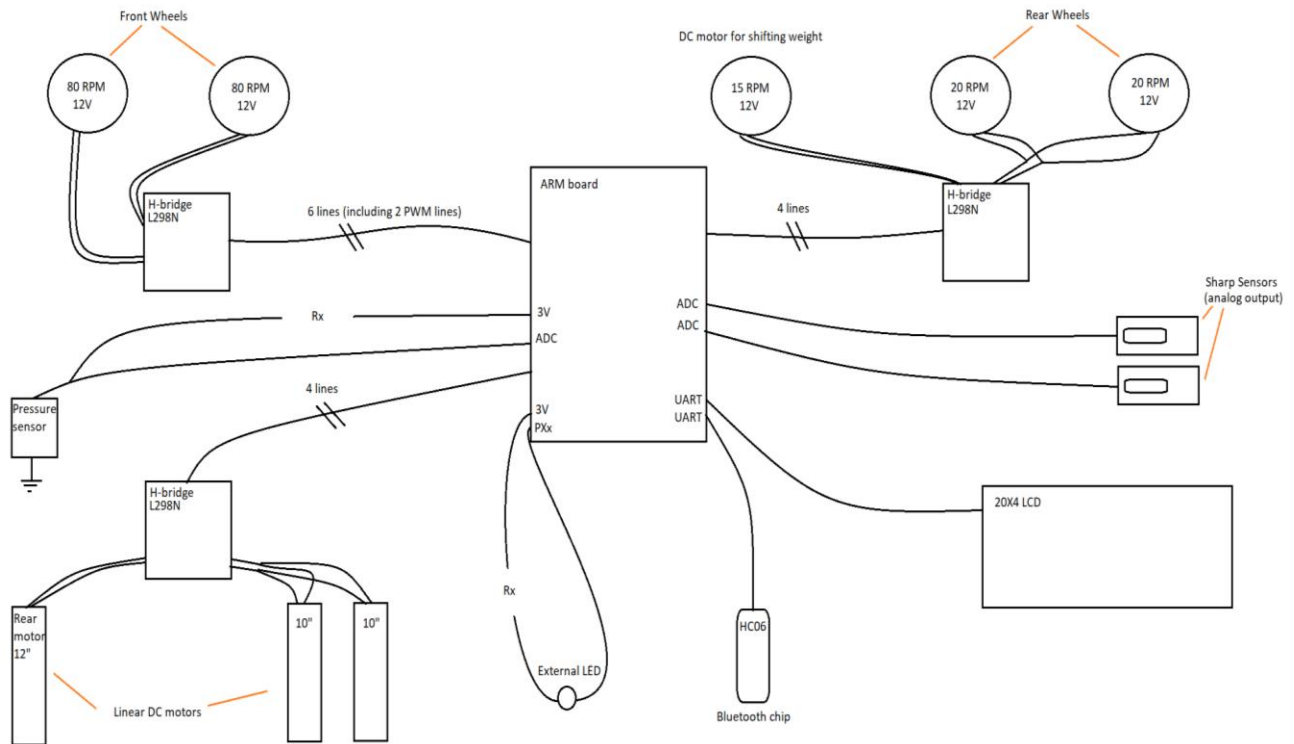


Figure 11: Circuit diagram of the final robot (in this diagram, all components are assumed to be powered)

### B. Software Design

The robot was programmed to let the user select an operation mode, either self-operating mode or wireless mode, and run the code for the selected mode. In details, after turning on the robot, the user must wait until the robot indicates all initializations are completed. Then, the user can select the desired mode using the labeled buttons at the back of the robot. If the selected mode is the self-operating mode, the robot will estimate the weight of the carried object to ensure that it is within the allowed weight and start climbing the stairs of the Koch Center all the way to the second floor. In case the weight is over 1 lb, the user will be asked to reduce the weight and

hit the Enter button. If the selected mode, on the other hand, is the wireless mode, the robot will display all the necessary details on the LCD screen to connect to the Bluetooth module installed on the robot. The user also has to use the android app that was created specifically for this robot using the MIT App Inventor website [4]. As it is shown in Figure 12, the app has one button to connect to the robot via Bluetooth networking, whose coverage can range to 100 meters, and the app has a list of commands that can be used to control the robot wirelessly. Figure 13 also illustrates the code blocks of the application, which explains how the application works.

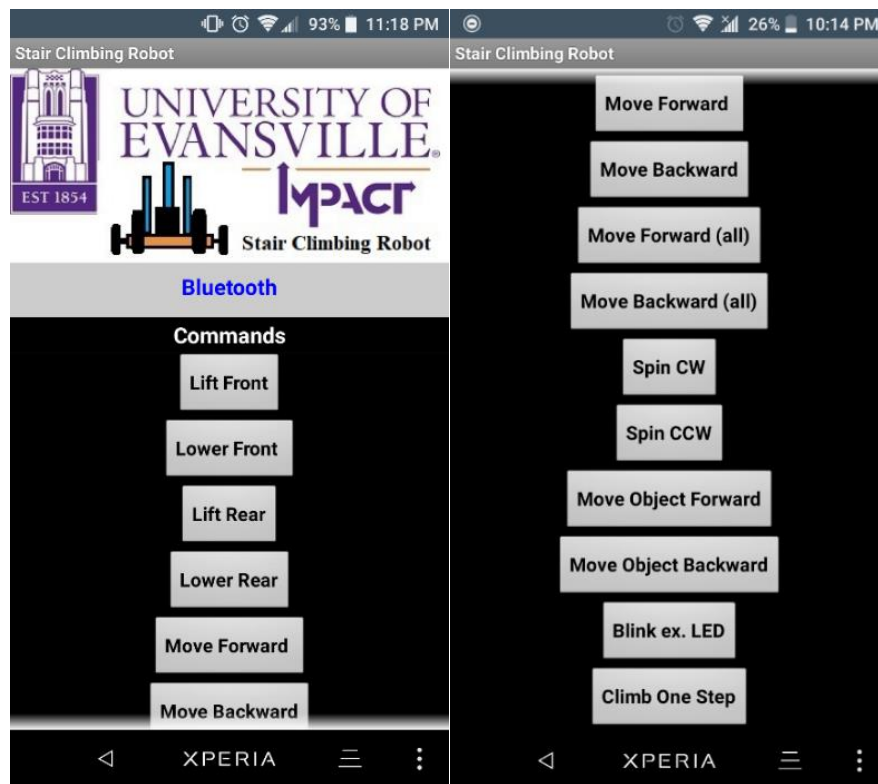


Figure 12: Android application for controlling the robot wirelessly



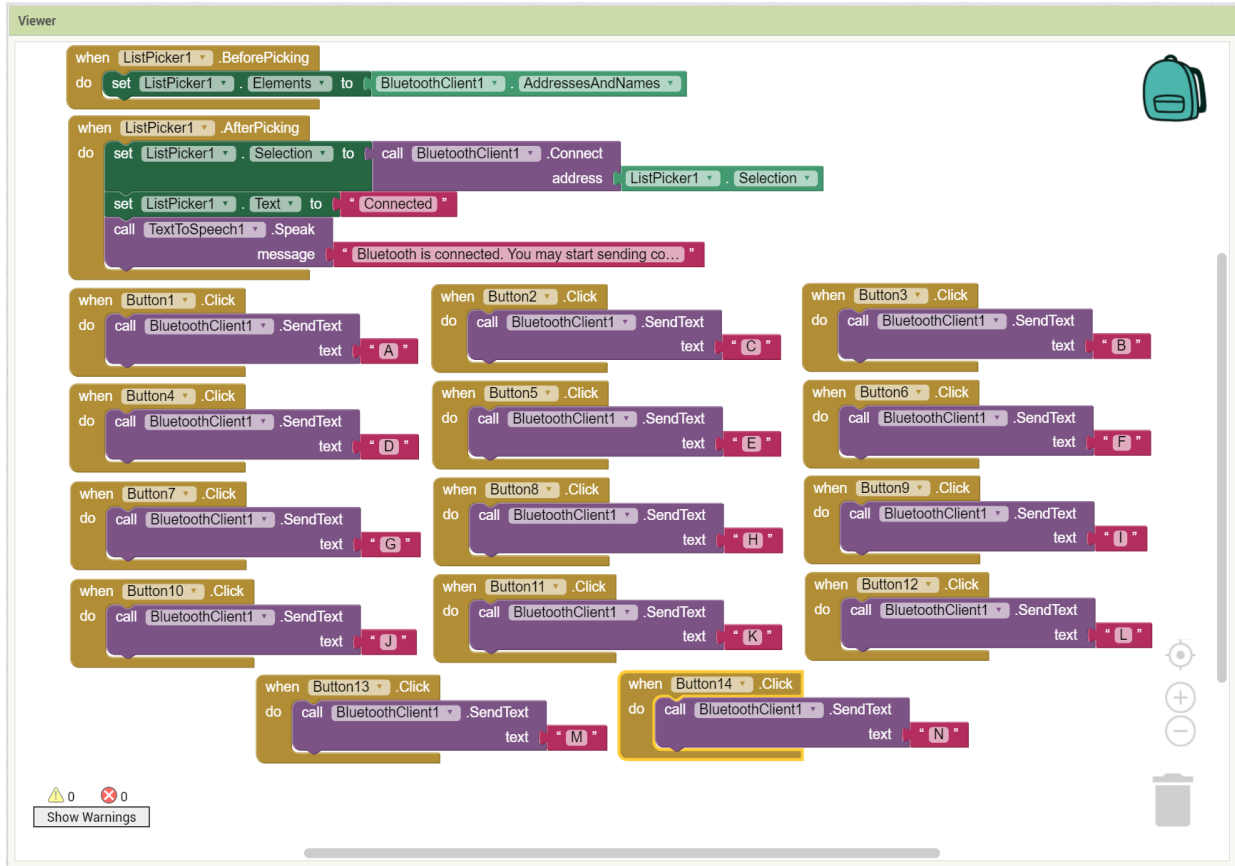


Figure 13: Code blocks of the Android application

The following is a brief pseudocode summarizing the final C-code that is in appendix A:

sub programs' prototypes

global variables

main ()

{

Initializations

display(instructions & info)

display(select operation mode)

while(button 1 selected) //first mode → self-operating mode

```

{
display(place the object inside the container and press <Enter>)
while(button 2 not selected);
read(ADC)
if (weight < 1 lb)
{display(wight)}
else
{
while (weight > 1lb)
{
display(weight)
display(you exceeded the limited weight ... reduce weight and hit <ENTER>)
while(button 2 not selected);
read(ADC)
}
display(weight)
display(weight is accepted)
}
display(robot will start climbing)
for(i = 0; i<14; i++) // 14 steps
{climb first stairs (see code in appendix A)}
for(i = 0; i<1; i++)
{climb last step of first stairs (see code in appendix A)}
make 180 degrees turn to second stairs (see code in appendix A)
for(i = 0; i<7; i++) // 7 steps
{climb second stairs (see code in appendix A)}
for(i = 0; i<1; i++)

```

```

{climb last step of second stairs (see code in appendix A)}
}
while(button 2 selected) //second mode → wireless mode
{
display(Bluetooth's name and password)
display(robot is waiting for commands)
while(1)
{
X = read(char received via UART from Bluetooth module)
if(X = something )
{display(command) and perform a certain command}
else if(X = something else)
{display(command) and perform a certain command}
.
.
.
}
}
}

```

## Subprograms

### *C. Standards and Constraints*

The combination of aluminum channels and lumbers, which were used in building the robot, forms a strong and light platform for the final robot, allowing the addition of extra weight to it. Thus, we can infer that the platform of the final robot is solid and reliable. In fact, the final

robot can carry any object weighing up to 2 lb and perform well, meaning it can carry 1 lb, which is the client's requirement, and perform even better.

To increase the safety of this product, the final robot does not have many bare or loose wires, which could generate an electric spark. Furthermore, because the lead-acid battery is covered by the container in the final robot, the chance of accidentally short-circuiting the battery terminals is very low, making the user safer. In addition to avoiding electric risks, the DC motor that balances the robot ensures that the robot does not fall down the stairs by moving the battery and the carried object forward and backward as the robot climbs the stairs.

The final robot, furthermore, uses few plastic components, so it does not increase significantly plastic pollution, which is one of the biggest issues in the world today. It also uses a rechargeable battery and does not emit any harmful gases, so it can be environment-friendly.

#### *D. Costs*

The items, listed in Table 1, are the most necessary components of this project. Likewise, several other components were printed using the 3D printer available in the EE department, such as the arms in Figure 14. Some components, such as the 20x4 LCD screen and ARM board, were already obtained by the student, reducing the final cost. Therefore, the total spending on this project was around \$ 795.00. However, the cost of the final robot is estimated to be around \$ 500.00.



Figure 14: 3D arms of the first design

Table 1: Costs

Item		Quantity	Cost
Five DC motors, two of which have built-in encoders		5	~ \$125
Items for robot's platform (e.g. Aluminum channels, screws, etc.)		NA	~ \$140
Sensors and Bluetooth chip (HC-06 RS232 Wireless Serial Bluetooth)		NA	~ \$40
Three Linear Actuators (10" and 12")		3	~ \$240
Tank wheels (from china)		2 (sets)	~ \$120
Two 12V batteries, 4 H-bridges (L298N), and 4 robot wheels		NA	~ \$80
PCB board		1	~ \$50
Obtained budget	= \$690	Total	~ \$795

\* Note: one of the five DC motors, which was used to rotate the arms in the first design, was returned. Thus, it is not included in this table.

## IV. Result

The overall outcome of this project is acceptable. After all, the final robot meets the main requirement of this project, which is climbing the stairs of the Koch Center while carrying a 1-lb object. The downside of this robot, however, is that it cannot make the 180-degree turn sometimes after climbing the first stairs, as can be seen in figure 15. Despite the fact that several methods were attempted to solve this issue, the robot is still not reliable when it comes to making the 180-degree turn. In the first method, the robot was programmed using the output values from the built-in encoders to make the turn, but about 50% of the time the robot does not hit the first step of the second stairs at 90 degrees angle. In the second method, the robot was programmed to follow the walls of the stairs to make the turn using two sharp sensors, but the signals of the sharp sensors interfered sometimes causing the robot to act oddly. To solve this issue, only one sharp sensor was used to make the robot follow the walls, but it is still not reliable. Some of the reasons for this odd behavior could be because the front and back wheels of the robot are not aligned, and the thin wheels of the robot get loose from time to time, causing the wheels to spin on the carpet floor of the stairs.

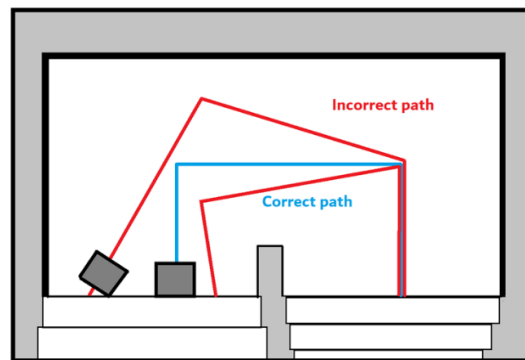


Figure 15: Robot's paths to the second stairs

## **VI. Conclusions and Recommendations**

To conclude, the final design satisfies the key concept of this senior project, which is about a robot climbing the stairs of the Koch Center while carrying a 1-lb object. It is also important to mention that this senior project proves that the final design works and that it could be modified slightly to make the robot climb not only the stairs of the Koch Center but also any standard stairs. Moreover, the robot built in this senior project cannot be considered as a final product because many of its components have been used in other designs. That is, many of the components are exhausted, such as the wheels and the main platform. Some of the robot's parts are also not perfectly built. As a result, replacing these components with newer ones or even better ones can improve significantly the robot's performance. For future work, the robot could be rebuilt using a better platform, and this platform should ensure that the wheels can be installed so that they are perfectly aligned. This new robot also should climb the stairs while detecting each step using distance or bumper sensors. Therefore, because of the improved platform and the ability to detect stair steps, this new robot should be able to climb any standard stairs and avoid falling down stairs. Equally important, this new robot can be used in many applications where small objects, such as medicines for disabled people and small military equipment, need to be carried and moved up or down the stairs using robots.

## References

- [1] E. Guizzo, "World Robot Population Reaches 8.6 Million", *IEEE Spectrum: Technology, Engineering, and Science News*, 2010. [Online]. Available: <https://spectrum.ieee.org/automaton/robotics/industrial-robots/041410-world-robot-population>. [Accessed: 02- Dec- 2017].
- [2] "STM32F4DISCOVERY - Discovery kit with STM32F407VG MCU \* New order code STM32F407G-DISC1 (replaces STM32F4DISCOVERY) - STMicroelectronics", *St.com*, 2017. [Online]. Available: <http://www.st.com/en/evaluation-tools/stm32f4discovery.html>. [Accessed: 02- Dec- 2017].
- [3] "MiniBoard Standard," ExpressPCB. [Online]. Available: <https://www.expresspcb.com/miniboard-standard/>. [Accessed: 24-Apr-2018].
- [4] "MIT App Inventor," MIT App Inventor | Explore MIT App Inventor, 02-Apr-2018. [Online]. Available: <http://appinventor.mit.edu/explore/>. [Accessed: 17-Apr-2018].



## Appendix A

The following is the C-code that was modified and used with all prototypes. The code was edited last time for the final robot, so all current subprograms in C-code are for the final robot. Moreover, this code lets the user choose between controlling the robot wirelessly via Bluetooth or letting the robot climb the stairs of the Koch center all the way to the second floor by itself.

```
1 //University of Evansville
2 //Senior Project(Stair Climbing Robot)
3 //Student: Ayman Homadi
4 //Sponser: Dr. Blandford
5
6 /*
7 NOTES:
8 H-Bridge1 - Front wheels
9 PA0 -> IN4
10 PA1 -> IN3
11 PA2 -> IN2
12 PA3 -> IN1
13 PA6 -> ENB
14 PA7 -> ENA
15 //////////////////////////////////////
16 Encoders
17 right motor -> PD0
18 left motor -> PD1
19 //////////////////////////////////////
20 H-Bridge2 - Linear Motors
21 PB0 -> IN4
22 PB1 -> IN3
23 PB2 -> IN2
24 PB3 -> IN1
25 //////////////////////////////////////
26 H-Bridge3 - Middle wheels/balance
27 PD2 -> IN4
28 PD3 -> IN3
29 PD4 -> IN2
30 PD5 -> IN1
31 */
32
33 #include "stm32f407vg.h"
34 #include "stdlib.h"
35
36 //Sub Programs
37 void InitPorts(void);
38 void InitUART(void);
39 void ConfigureUART(unsigned int baudDivisor);
```

```

40 void InitPWM(void);
41 void EXTI0_IRQHandler(void);
42 void EXTI1_IRQHandler(void);
43 void Forward(int L, int R, int D, int m, float Speed_PercL, float Speed_PercR);
44 void Backward(int L, int R, int D, int m, float Speed_PercL, float Speed_PercR);
45 void Linear( int UPDOWN, int MODE, int T);
46 void balance( int m, int T);
47 void MMOTORS( int m, int T);
48 int SHARPL(void);
49 int SHARPR(void);
50 int Load(void);
51 void exLED( int N, int B);
52 void UART_Send_String(char* msg2);
53 void UART_Receive_String(void);
54 void spin(int r, int t, float s);
55 void ToString( int num);
56 int Button1(void);
57 int Button2(void);
58 void LEDDelay(int x);
59 void PLEDs( int y);
60 void Delay(int x);
61
62 //Global Variables
63 int RightCount = 0;
64 int LeftCount = 0;
65 char c[32]; //char array for received Data
66 char str[10]; // converted number to string will be stored in this array
67
68 //Main program
69 int main()
70 {
71 InitPorts();
72 InitPWM();
73 InitUART(); //intialize UART
74 ConfigureUART(0xD05);
75 UART_Send_String("/");
76 LEDDelay(4);
77 UART_Send_String("ARM board is initializing...");
78 LEDDelay(4);
79 if( Button2() == 1) //pressing and holding the enter button during initializaion results in
compressing all linear actuators "hidden feature"
80 { Linear( 0, 1, 1500);}
81 int step = 0;
82 int R = 20;;
83 int L = 20;
84 // int SR = 0;
85 // int SL = 0;
86 balance(1, 1200);
87 UART_Send_String("/");
88 LEDDelay(1);
89 UART_Send_String("WARNING! if a wire is disconnected, turn off the robot immediately");
90 LEDDelay(10);
91 UART_Send_String("/");
92 LEDDelay(1);
93 UART_Send_String("Univ. of Evansville Stair Climbing RobotSponser Dr.BlandfordStudent Ayman Homadi");
94 LEDDelay(10);
95 UART_Send_String("/");
96 LEDDelay(1);

```

```
97 UART_Send_String("Intialization is complete");
98 for(int i =0; i <10;i++) //LEDs will blink to indicate intialization is complete
99 {GPIO_ODR = 0xF000;
100 Delay(400);
101 GPIO_ODR = 0x0000;
102 Delay(400);}
103 UART_Send_String("/");
104 Delay(300);
105 UART_Send_String("Select operation mode");
106
107 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
108 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
109 while(1)
110 {
111 while( Button2() == 1) //*****Self-Operation mode*****
112 {
113 UART_Send_String("/");
114 Delay(300);
115 UART_Send_String("Selected mode is Self-Operation mode");
116 Delay(15000);
117 UART_Send_String("/");
118 Delay(100);
119 UART_Send_String("Place the object inside the containerand press <Enter>");
120 while( Button2() != 1);
121 Delay(100);
122 UART_Send_String("/");
123 Delay(100);
124 ToString(Load());
125 if( Load() <= 500)
126 {
127 UART_Send_String("Estimated weight is ");
128 UART_Send_String(str);
129 UART_Send_String(" g");
130 Delay(15000);
131 UART_Send_String("/");
132 }
133 else
134 {
135 while(Load() > 500)
136 {
137 UART_Send_String("Estimated weight is ");
138 UART_Send_String(str);
139 UART_Send_String(" g");
140 Delay(15000);
141 UART_Send_String("/");
142 Delay(100);
143 UART_Send_String("You have exceeded the limited weight (500 g)");
144 Delay(20000);
145 UART_Send_String("/");
146 Delay(100);
147 UART_Send_String("Reduce the weight and press <Enter>");
148 while( Button2() != 1);
149 UART_Send_String("/");
150 Delay(100);
151 ToString(Load());
152 }
153 UART_Send_String("Estimated weight is ");
```

```

154 UART_Send_String(str);
155 UART_Send_String(" g");
156 Delay(15000);
157 UART_Send_String("/");
158 }
159 Delay(300);
160 UART_Send_String("The weight is within the allowable weight. The robot will start climbing shortly...");
161 Delay(30000);
162 UART_Send_String("/");
163 Delay(100);
164 LEDDelay(1);
165 UART_Send_String("Univ. of Evansville Stair Climbing Robot Sponser Dr. Blandford Student Ayman
Homadi");
166
167 ///----- FIRST STAIRS
168 for( step = 0; step < 14; step++)
169 {
170 Linear( 1, 1, 4200);
171 Linear( 1, 3, 300);
172 Linear( 1, 3, 700);
173 Linear( 1, 1, 800);
174 Linear( 1, 3, 1800);
175 Forward(1,1, 250, 1, 30, 30);
176 Linear( 1, 2, 2000);
177 Linear( 0, 3, 500);
178
179 Forward(1,1, 500, 0, 25, 25);
180 Backward(1,1, 150, 0, 25, 25);
181
182 Linear( 0, 3, 5600);
183 Forward(1,1, 1000, 0, 25, 25);
184 RightCount = 0;
185 LeftCount = 0;
186 while((RightCount + LeftCount)/2 < 200)
187 { if(RightCount > LeftCount && R > 0)
188 {R = R - 2;
189 L = L + 2;}
190 else if(RightCount < LeftCount && L > 0)
191 {R = R + 2;
192 L = L - 2;}
193 Forward(1,1, 1, 0, L, R); }
194 Linear( 1, 3, 600);
195 balance(0, 4000);
196 Linear( 0, 2, 5800);
197 Linear( 0, 3, 300);
198 Linear( 0, 1, 500);
199 Forward(1,1, 250, 1, 20, 20);
200 Linear( 0, 1, 300);
201 Linear( 0, 2, 450);
202 Forward(1,1, 20, 1, 7.5, 7.5);
203 balance(1, 4000);
204 LEDDelay(5);
205 }
206 ///-----
207 for( step = 0; step < 1; step++)
208 {
209 Linear( 1, 1, 4200);
210 Linear( 1, 3, 300);

```

```

211 Linear( 1, 3, 700);
212 Linear( 1, 1, 800);
213 Linear( 1, 3, 1800);
214 Forward(1,1, 250, 1, 30, 30);
215 Linear( 1, 2, 2000);
216 Linear( 0, 3, 500);
217
218 Forward(1,1, 500, 0, 25, 25);
219 Backward(1,1, 150, 0, 25, 25);
220
221 Linear( 0, 3, 5600);
222 RightCount = 0;
223 LeftCount = 0;
224 while((RightCount + LeftCount)/2 < 2700)
225 { if(RightCount > LeftCount && R > 0)
226 {R = R - 2;
227 L = L + 2;}
228 else if(RightCount < LeftCount && L > 0)
229 {R = R + 2;
230 L = L - 2;}
231 Forward(1,1, 1, 0, L, R); }
232 Linear( 1, 3, 600);
233 balance(0, 4000);
234 Backward(1,1, 200, 1, 20, 20);
235 Linear( 0, 2, 5850);
236 Linear( 0, 3, 300);
237 Linear( 0, 1, 500);
238 Forward(1,1, 250, 1, 20, 20);
239 balance(1, 4000);
240 LEDDelay(5);
241 }
242
243 ///----- TURN
244 Linear(1,2,800);
245 RightCount = 0;
246 LeftCount = 0;
247 while((RightCount + LeftCount)/2 < 12000)
248 { if(RightCount > LeftCount && R > 0)
249 {R = R - 2;
250 L = L + 2;}
251 else if(RightCount < LeftCount && L > 0)
252 {R = R + 2;
253 L = L - 2;}
254 Forward(1,1, 1, 0, L, R);
255 }
256 ///----
257 Linear(1,2,50);
258 spin(0,850,50);
259 Linear(0,2,50);
260 ///----
261 RightCount = 0;
262 LeftCount = 0;
263 while((RightCount + LeftCount)/2 < 34000)
264 { if(RightCount > LeftCount && R > 0)
265 {R = R - 2;
266 L = L + 2;}
267 else if(RightCount < LeftCount && L > 0)
268 {R = R + 2;

```

```

269 L = L - 2;}
270 Forward(1,1, 1, 0, L, R);
271 }
272 ///-----
273 Linear(1,2,50);
274 spin(0,750,50);
275 Linear(0,2,50);
276 ///-----
277 RightCount = 0;
278 LeftCount = 0;
279 while((RightCount + LeftCount)/2 < 18000)
280 { if(RightCount > LeftCount && R > 0)
281 {R = R - 2;
282 L = L + 2;}
283 else if(RightCount < LeftCount && L > 0)
284 {R = R + 2;
285 L = L - 2;}
286 Forward(1,1, 1, 0, L, R);
287 }
288 // SR = SHARPR() + 30; //THIS CODE WAS USED WHEN BOTH SHARP SENSORS WERE INSTALLED AT THE
FRONT
289 // SL = SHARPL(); //IT DID NOT WORK BECAUSE OF THE INTERFERENCE
290 //
291 // while( abs(SR - SL) > 50)
292 // {
293 // ToString(SR);
294 // UART_Send_String("R= ");
295 // UART_Send_String(str);
296 // ToString(SL);
297 // UART_Send_String(" L= ");
298 // UART_Send_String(str);
299 // Delay(20000);
300 // UART_Send_String("/");
301 // if(SL > SR)
302 // {Forward(0,1,200,0,20,30);}
303 // if(SL < SR)
304 // {Forward(1,0,200,0,30,20);}
305 // SR = SHARPR() + 30;
306 // SL = SHARPL();
307 // }
308 Forward(1,0, 200, 0, 60, 0);
309 Forward(0,1, 200, 0, 0, 60);
310 Linear(0,2,350);
311 Forward(1,1, 150, 1, 20, 20);
312 Linear(0,2,500);
313
314 ///----- SECOND STAIRS
315 for( step = 0; step < 7; step++)
316 {
317 Linear( 1, 1, 4200);
318 Linear( 1, 3, 300);
319 Linear( 1, 3, 700);
320 Linear( 1, 1, 800);
321 Linear( 1, 3, 1800);
322 Forward(1,1, 250, 1, 30, 30);
323 Linear( 1, 2, 2000);
324 Linear( 0, 3, 500);
325

```

```

326 Forward(1,1, 500, 0, 25, 25);
327 Backward(1,1, 150, 0, 25, 25);
328
329 Linear( 0, 3, 5600);
330 Forward(1,1, 1000, 0, 25, 25);
331 RightCount = 0;
332 LeftCount = 0;
333 while((RightCount + LeftCount)/2 < 200)
334 { if(RightCount > LeftCount && R > 0)
335 {R = R - 2;
336 L = L + 2;}
337 else if(RightCount < LeftCount && L > 0)
338 {R = R + 2;
339 L = L - 2;}
340 Forward(1,1, 1, 0, L, R); }
341 Linear( 1, 3, 600);
342 balance(0, 4000);
343 Linear( 0, 2, 5800);
344 Linear( 0, 3, 300);
345 Linear( 0, 1, 500);
346 Forward(1,1, 250, 1, 20, 20);
347 Linear( 0, 1, 300);
348 Linear( 0, 2, 450);
349 Forward(1,1, 20, 1, 7.5, 7.5);
350 balance(1, 4000);
351 LEDDelay(5);
352 }
353 ///-----
354 for( step = 0; step < 1; step++)
355 {
356 Linear( 1, 1, 4200);
357 Linear( 1, 3, 300);
358 Linear( 1, 3, 700);
359 Linear( 1, 1, 800);
360 Linear( 1, 3, 1800);
361 Forward(1,1, 250, 1, 30, 30);
362 Linear( 1, 2, 2000);
363 Linear( 0, 3, 500);
364
365 Forward(1,1, 500, 0, 25, 25);
366 Backward(1,1, 150, 0, 25, 25);
367
368 Linear( 0, 3, 5600);
369 RightCount = 0;
370 LeftCount = 0;
371 while((RightCount + LeftCount)/2 < 2700)
372 { if(RightCount > LeftCount && R > 0)
373 {R = R - 2;
374 L = L + 2;}
375 else if(RightCount < LeftCount && L > 0)
376 {R = R + 2;
377 L = L - 2;}
378 Forward(1,1, 1, 0, L, R); }
379 Linear( 1, 3, 600);
380 balance(0, 4000);
381 Backward(1,1, 200, 1, 20, 20);
382 Linear( 0, 2, 5850);
383 Linear( 0, 3, 300);

```

```

384 Linear( 0, 1, 500);
385 Forward(1,1, 250, 1, 20, 20);
386 balance(1, 4000);
387 LEDDelay(5);
388 }
389 while(1);
390 }
391
392 while( Button1() == 1) //*****Wireless mode*****
393 {
394 UART_Send_String("/");
395 Delay(300);
396 UART_Send_String("Selected mode is Wireless mode");
397 Delay(15000);
398 UART_Send_String("/");
399 Delay(300);
400 UART_Send_String("Bluetooth name: StairClimbingRobot Password: 1234");
401 Delay(40000);
402 UART_Send_String("/");
403 Delay(300);
404 UART_Send_String("Robot is wating for commands");
405 Delay(15000);
406 UART_Send_String("/");
407 while(1)
408 {
409 UART_Receive_String();
410 Delay(100);
411 if ( c[0] == 'A')
412 {UART_Send_String("Received command is Lift front");
413 Linear( 1, 3, 300);
414 UART_Send_String("/");}
415
416 else if ( c[0] == 'B')
417 {UART_Send_String("Received command is Lift rear");
418 Linear( 1, 2, 300);
419 UART_Send_String("/");}
420
421 else if ( c[0] == 'C')
422 {UART_Send_String("Received command is Lower front");
423 Linear( 0, 3, 300);
424 UART_Send_String("/");}
425
426 else if ( c[0] == 'D')
427 {UART_Send_String("Received command is Lower rear");
428 Linear( 0, 2, 300);
429 UART_Send_String("/");}
430
431 else if ( c[0] == 'E')
432 {UART_Send_String("Received command is Move Forward");
433 Linear( 1, 2, 800);
434 Forward(1,1, 300, 0, 50, 50);
435 Linear( 0, 2, 820);
436 UART_Send_String("/");}
437
438 else if ( c[0] == 'F')
439 {UART_Send_String("Received command is Move Backward");
440 Linear( 1, 2, 800);
441 Backward(1,1, 300, 0, 50, 50);

```



```
442 Linear( 0, 2, 820);
443 UART_Send_String("/");}
444
445 else if ( c[0] == 'G')
446 {UART_Send_String("Received command is All wheels forward");
447 Forward(1,1, 500, 1, 50, 50);
448 UART_Send_String("/");}
449
450 else if ( c[0] == 'H')
451 {UART_Send_String("Received command is All wheels backward");
452 Backward(1,1, 500, 1, 50, 50);
453 UART_Send_String("/");}
454
455 else if ( c[0] == 'I')
456 {UART_Send_String("Received command is Spin CW");
457 Linear( 1, 2, 900);
458 spin(1,200,90);
459 Linear( 0, 2, 920);
460 UART_Send_String("/");}
461
462 else if ( c[0] == 'J')
463 {UART_Send_String("Received command is Spin CCW");
464 Linear( 1, 2, 900);
465 spin(0,200,90);
466 Linear( 0, 2, 920);
467 UART_Send_String("/");}
468
469 else if ( c[0] == 'K')
470 {UART_Send_String("Received command is Move object forward");
471 balance(0, 1000);
472 UART_Send_String("/");}
473
474 else if ( c[0] == 'L')
475 {UART_Send_String("Received command is Move object backward");
476 balance(1, 1000);
477 UART_Send_String("/");}
478
479 else if ( c[0] == 'M')
480 {UART_Send_String("Received command is Blink LED");
481 exLED( 0, 1);
482 UART_Send_String("/");}
483
484 else if ( c[0] == 'N')
485 {UART_Send_String("Received command is Climb one step");
486 for( step = 0; step < 1; step++)
487 {
488 Linear( 1, 1, 4200);
489 Linear( 1, 3, 300);
490 Linear( 1, 3, 700);
491 Linear( 1, 1, 800);
492 Linear( 1, 3, 1800);
493 Forward(1,1, 250, 1, 30, 30);
494 Linear( 1, 2, 2000);
495 Linear( 0, 3, 500);
496
497 Forward(1,1, 500, 0, 25, 25);
498 Backward(1,1, 150, 0, 25, 25);
499
```



```

    balance motor //PD0-PD1 are input for ex. interrupt
555 GPIO_ODR |= 0xFFFF;
556
557 RCC_APB2ENR |= 0x4000; //SYSCFG clock enable
558 NVICISER0 |= (1<<6); //Enable EXTI0
559 NVICISER0 |= (1<<7); //Enable EXTI1
560 GPIO_ODR |= 0x05;
561 SYSCFG_EXTICR1 |= 0xFFFF00; // EXTI0&1 set to PD0 & PD1
562 SYSCFG_EXTICR1 |= 0x33;
563 EXTI_IMR |= 3; //don't mask interrupt line 0,1
564 EXTI_RTSR |= 3; //rising trigger enable line 0,1
565 }
566
567 void InitUART(void)
568 {
569 RCC_APB1ENR |= (1 << 4); //Enable peripheral timer for timer 6
570 RCC_APB2ENR |= (1 << 5); //Enable USART6 clock
571 NVICISER2 |= (1 << 5); //Bit 5 in ISER2 corresponds to int 71 (USART6)
572 GPIO_AFRL = 0x88000000; //Alternate Function on PC6 & PC7 to USART6
573 GPIO_MODER |= 0x0A000; //Alternate Function on PC6 & PC7
574 GPIO_OSPEEDER |= 0xF000; //high speed on PC6 & PC7
575 }
576
577 void ConfigureUART(unsigned int baudDivisor)
578 {
579 USART6_CR1 = 0;
580 USART6_BRR = baudDivisor;
581 USART6_CR2 = 0; //1 stop bit
582 USART6_CR1 = 0x200C;
583 USART6_CR3 = 0;
584 }
585
586 void InitPWM(void)
587 {
588 RCC_APB1ENR |= 0x02; //Peripheral timer clock
589 TIM3_CR1 |= (1<<7); //Enable Auto-reload
590 TIM3_PSC = 4; //Prescaler
591 TIM3_ARR = 64000; //Overflow at 20ms (16MHz/5)*20ms = 64000
592 TIM3_CCMR1 |= 0x6868; //Preload on CCR1 & CCR2 enabled, output forced high
593 TIM3_CCER |= 0x0011; //Signal outputted to output pin
594 TIM3_CCR1 = 0; //Initially not moving
595 TIM3_CCR2 = 0;
596 TIM3_EGR |= 1; //Load value to timer
597 TIM3_CR1 |= 1; //Enable Timer 3
598 }
599
600 void EXTI0_IRQHandler(void) //Interrupt ext line 0
601 {
602 RightCount++;
603 EXTI_PR |= 1;
604 }
605
606 void EXTI1_IRQHandler(void) //Interrupt ext line 1
607 {
608 LeftCount++;
609 EXTI_PR |= 2;
610 }
611

```

```

612 void Forward(int L, int R, int D, int m, float Speed_PercL, float Speed_PercR)
613 {
614 float ans1, ans2;
615 if ( m == 1)
616 {GPIOA_ODR = 0x8;
617 Speed_PercL = 15;
618 Speed_PercR = 15;
619 L= 1;
620 R =1;}
621 if ( L == 1 && R == 1) //Both left and right wheels are moving forward
622 {GPIOA_ODR = 0xA;}
623
624 else if ( L == 1 && R == 0) //Only left wheel is moving forward
625 {GPIOA_ODR = 0x2;}
626
627 else // L == 0 && R == 1 //Only right wheel is moving forward
628 {GPIOA_ODR = 0x8;}
629
630 ans1 = Speed_PercL/100; //Turn speed1 to percentage
631 ans2 = Speed_PercR/100; //Turn speed2 to percentage
632 TIM3_CCR1 = ans1 * TIM3_ARR; //New value for motor 1
633 TIM3_CCR2 = ans2 * TIM3_ARR; //New value for motor 2
634
635 int i, j;
636 for(i=0;i<D;i++)
637 for(j=0;j<10000;j++);
638 GPIOA_ODR &= 0xF0;
639 GPIOD_ODR = 0x0;
640 }
641
642 void Backward(int L, int R, int D, int m, float Speed_PercL, float Speed_PercR)
643 {
644 float ans1, ans2;
645 if ( m == 1)
646 {GPIOA_ODR = 0x4;
647 Speed_PercL = 15;
648 Speed_PercR = 15;
649 L= 1;
650 R =1;}
651 if ( L == 1 && R == 1) //Both left and right wheels are moving backward
652 {GPIOA_ODR = 0x5;}
653
654 else if ( L == 1 && R == 0) //Only left wheel is moving backward
655 {GPIOA_ODR = 0x1;}
656
657 else // L == 0 && R == 1 //Only right wheel is moving backward
658 {GPIOA_ODR = 0x4;}
659
660 ans1 = Speed_PercL/100; //Turn speed1 to percentage
661 ans2 = Speed_PercR/100; //Turn speed2 to percentage
662 TIM3_CCR1 = ans1 * TIM3_ARR; //New value for motor 1
663 TIM3_CCR2 = ans2 * TIM3_ARR; //New value for motor 2
664
665 int i, j;
666 for(i=0;i<D;i++)
667 for(j=0;j<10000;j++);
668 GPIOA_ODR &= 0xF0;
669 GPIOD_ODR = 0x0;

```

```

670 }
671
672 void Linear(int UPDOWN,int MODE, int T)
673 {
674 //MODE: 1 BOTH 2 BACK 3 FRONT
675 //UPDOWN: 0 DOWN 1 UP
676
677 exLED( 1, 0);
678 if ( UPDOWN == 1 && MODE == 1)
679 {GPIOB_ODR = 0x5;}
680 else if ( UPDOWN == 0 && MODE == 1)
681 {GPIOB_ODR = 0xA;}
682 else if ( UPDOWN == 1 && MODE == 2)
683 {GPIOB_ODR = 0x1;}
684 else if ( UPDOWN == 0 && MODE == 2)
685 {GPIOB_ODR = 0x2;}
686 else if ( UPDOWN == 1 && MODE == 3)
687 {GPIOB_ODR = 0x4;}
688 else if ( UPDOWN == 0 && MODE == 3)
689 {GPIOB_ODR = 0x8;}
690
691 int i, j;
692 for(i=0;i<T;i++)
693 for(j=0;j<10000;j++);
694
695 GPIOB_ODR &= 0xFFFF0;
696 exLED( 0, 0);
697 }
698
699 void balance( int m, int T)
700 {
701 // m = 0 forward m = 1 backward
702
703 exLED( 1, 1);
704 if ( m == 1)
705 {GPIOD_ODR = 0x10;}
706 else if ( m == 0)
707 {GPIOD_ODR = 0x20;}
708 int i, j;
709 for(i=0;i<T;i++)
710 for(j=0;j<10000;j++);
711 GPIOD_ODR = 0x0;
712 exLED( 0, 0);
713 }
714
715 void MMOTORS( int m, int T)
716 {
717 // m = 1 forward m = 0 backward
718
719 if ( m == 0)
720 {GPIOD_ODR = 0x4;}
721 else if ( m == 1)
722 {GPIOD_ODR = 0x8;}
723 int i, j;
724 for(i=0;i<T;i++)
725 for(j=0;j<10000;j++);
726 GPIOD_ODR = 0x0;
727 }

```

```

728
729 int SHARPL(void)
730 {
731   exLED( 0, 1);
732   RCC_APB2ENR |= 0x100; //ADC 1 clock is enabled
733   ADC1_CR2 |= 1;
734   ADC1_CR2 |= 0x400;
735   ADC_CCR |= 0x30000;
736   ADC1_SQR3 |= 0xC;
737   ADC1_CR2 |= 0x40000000; //Start conversion
738   while((ADC1_SR & 0x02) == 0); //Wait for bit 0 to be set
739   return ADC1_DR; //Load digital value to SHARPF
740 }
741
742 int SHARPR(void)
743 {
744   exLED( 0, 1);
745   RCC_APB2ENR |= 0x200; //ADC 2 clock is enabled
746   ADC2_CR2 |= 1;
747   ADC2_CR2 |= 0x400;
748   ADC_CCR |= 0x30000;
749   ADC2_SQR3 |= 0xB;
750   ADC2_CR2 |= 0x40000000; //Start conversion
751   while((ADC2_SR & 0x02) == 0); //Wait for bit 0 to be set
752   return ADC2_DR; //Load digital value to SHARPB
753 }
754
755 int Load(void)
756 { int x1, x2, x3;
757   RCC_APB2ENR |= 0x400; //ADC 3 clock is enabled
758   ADC3_CR2 |= 1;
759   ADC3_CR2 |= 0x400;
760   ADC_CCR |= 0x30000;
761   ADC3_SQR3 |= 0xD;
762   ADC3_CR2 |= 0x40000000; //Start conversion
763   while((ADC3_SR & 0x02) == 0); //Wait for bit 0 to be set
764   x1 = ADC3_DR;
765   x3 = abs(x1 - 3200);
766   if(x3 <= 350) //scalling & adjusting weight
767   { x2 = abs(x1 - 3200)*0.04;}
768   else if(350 < x3 && x3 <= 530)
769   {x2 = abs(x1 - 3200)*0.06;}
770   else if(530 < x3 && x3 <= 600)
771   {x2 = abs(x1 - 3200)*0.065;}
772   else if(600 < x3 && x3 <= 675)
773   {x2 = abs(x1 - 3200)*0.07;}
774   else if(675 < x3 && x3 <= 700)
775   {x2 = abs(x1 - 3200)*0.075;}
776   else if(700 < x3 && x3 <= 850)
777   {x2 = abs(x1 - 3200)*0.085;}
778   else if(850 < x3 && x3 <= 1100)
779   {x2 = abs(x1 - 3200)*0.11;}
780   else if(1100 < x3 && x3 <= 1400)
781   {x2 = abs(x1 - 3200)*0.125;}
782   else if(1400 < x3 && x3 <= 1850)
783   {x2 = abs(x1 - 3200)*0.17;}
784   else if(1850 < x3 && x3 <= 2200)
785   {x2 = abs(x1 - 3200)*0.23;}

```

```

786 else if(2200 < x3 && x3 <= 2350)
787 {x2 = abs(x1 - 3200)*0.25;}
788 else if(2350 < x3 && x3 <= 2550)
789 {x2 = abs(x1 - 3200)*0.345;}
790 else //if(2550 < x3)
791 {x2 = abs(x1 - 3200)*0.5;}
792
793 if(x2 <= 90) //Purple LEDs will indicate the weight of the carried object
794 {PLEDs(1);}
795 else if(90 < x2 && x2 <= 2*90)
796 {PLEDs(2);}
797 else if(2*90 < x2 && x2 <= 3*90)
798 {PLEDs(3);}
799 else if(3*90 < x2 && x2 <= 4*90)
800 {PLEDs(4);}
801 else if(4*90 < x2)
802 {PLEDs(5);}
803
804 return x2; //Load digital value to SHARPF
805 }
806
807 void exLED( int N, int B)
808 {
809 if ( B == 1)
810 {for ( int x = 0; x < 10; x++)
811 {if((GPIOC_ODR & 0x00001) == 0x00001)
812 {GPIOC_ODR = (GPIOC_ODR & 0xFFFFE);}
813 else
814 {GPIOC_ODR = (GPIOC_ODR | 0x00001);}
815 int i, j;
816 for(i=0;i<50;i++)
817 for(j=0;j<10000;j++);}
818 if ( N == 1)
819 {GPIOC_ODR = (GPIOC_ODR & 0xFFFFF0);}
820 else
821 {GPIOC_ODR = (GPIOC_ODR | 0x000001);}
822 }
823
824 void UART_Send_String(char* msg2)
825 {
826 unsigned char i = 0;
827 while (msg2[i] != 0)
828 {while((USART6_SR & 0x80) == 0); //loop till previous data is sent
829 USART6_DR = msg2[i];
830 i++;}
831 }
832
833 void UART_Receive_String(void)
834 {
835 int i = 0;
836 int ch;
837 while( i <1)
838 {while((USART6_SR & 0x20) == 0); //loop till data is ready to be read
839 ch = USART6_DR;
840 c[i] = (char)ch;
841 i++;}
842 }
843

```

```

844 void spin(int r, int t, float s)
845 {
846 float ans1, ans2;
847 if ( r == 1)
848 {GPIOA_ODR = 0x6 ; }
849 else
850 {GPIOA_ODR = 0x9;}
851 ans1 = s/100; //Turn speed1 to percentage
852 ans2 = s/100; //Turn speed2 to percentage
853 TIM3_CCR1 = ans1 * TIM3_ARR; //New value for motor 1
854 TIM3_CCR2 = ans2 * TIM3_ARR; //New value for motor 2
855
856 int i, j;
857 for(i=0;i<t;i++)
858 for(j=0;j<10000;j++);
859 GPIOA_ODR = 0x0;
860 }
861
862 void ToString(int num)
863 {
864 int i, rem, len = 0, n;
865 n = num;
866 while (n != 0)
867 {len++;
868 n /= 10;}
869 for (i = 0; i < len; i++)
870 {rem = num % 10;
871 num = num / 10;
872 str[len - (i + 1)] = rem + '0';}
873 str[len] = '\0';
874 }
875
876 int Button1(void)
877 {
878 int x1;
879 if((GPIOA_IDR & 0x10) != 0x10)
880 {x1 = 1;} // Button = 1 --> button is pushed
881 else
882 {x1 = 0;} // Button = 0 --> button is not pushed
883 return x1;
884 }
885
886 int Button2(void)
887 {
888 int x2;
889 if((GPIOA_IDR & 0x20) != 0x20)
890 {x2 = 1;} // Button = 1 --> button is pushed
891 else
892 {x2 = 0;} // Button = 0 --> button is not pushed
893 return x2;
894 }
895
896 void LEDDelay(int x)
897 {
898 for(int i = 0; i < x; i++)
899 {GPIOC_ODR = 0x1000;
900 Delay(700);
901 GPIOC_ODR = 0x2000;

```



```

902 Delay(700);
903 GPIOD_ODR = 0x4000;
904 Delay(700);
905 GPIOD_ODR = 0x8000;
906 Delay(700);
907 GPIOD_ODR = 0x0000;}
908 }
909
910 void PLEDs( int y)
911 {
912 if( y == 0)
913 {GPIOC_ODR = (GPIOC_ODR | 0xF800);}
914 else if( y == 1)
915 {GPIOC_ODR = (GPIOC_ODR | 0xF000);
916 GPIOC_ODR = (GPIOC_ODR & 0xF7FF);}
917 else if ( y == 2)
918 {GPIOC_ODR = (GPIOC_ODR | 0xE000);
919 GPIOC_ODR = (GPIOC_ODR & 0xE7FF);}
920 else if ( y == 3)
921 {GPIOC_ODR = (GPIOC_ODR | 0xC000);
922 GPIOC_ODR = (GPIOC_ODR & 0xC7FF);}
923 else if ( y == 4)
924 {GPIOC_ODR = (GPIOC_ODR | 0x8000);
925 GPIOC_ODR = (GPIOC_ODR & 0x87FF);}
926 else if ( y == 5)
927 {GPIOC_ODR = (GPIOC_ODR & 0x07FF);}
928 }
929
930 void Delay(int x)
931 {
932 int i, j;
933 for(i=0;i<x;i++)
934 for(j=0;j<1000;j++);
935 }
936
937
938 //Linear( 1, 2, 1000); //THESE CODES WERE USED FOR WALL FOLLOWING AND THE TURN
939 //L = 30;
940 //R = 30;
941 //RightCount = 0;
942 //LeftCount = 0;
943 //while((RightCount + LeftCount)/2 < 10000)
944 //if(SHARPL() <600)
945 // {Forward(1,0, 200, 0, 30, 0);}
946 // else
947 // {Forward(0,1, 200, 0, 0, 30);}
948 // UART_Send_String("/");
949 // Delay(100);
950 // ToString(SHARPL());
951 // UART_Send_String("ADC =");
952 // UART_Send_String(str);
953 // Delay(2000);
954 //
955 //}
956 //Linear( 0, 2, 1020);
957
958
959 // if(SHARPL() <600)

```

```
960 // {R = R - 3;
961 // L = L + 3;}
962 // else
963 // {R = R + 3;
964 // L = L - 3;}
965 // Forward(1,1, 200, 0, L, R);
966
967 //while(1)
968 //{ToString(SHARPR());
969 //UART_Send_String("R =");
970 //UART_Send_String(str);
971 //Delay(5000);
972 //UART_Send_String("/");
973 //Delay(100);
974 //
975 //ToString(SHARPL());
976 //UART_Send_String("L =");
977 //UART_Send_String(str);
978 //Delay(5000);
979 //UART_Send_String("/");
980 //Delay(100);
981 //}
982
```