ELEC6212 - Biologically Inspired Robots

Group H Report

Aran McConnell - ajm2g17 - 29131855

Cian Raychaudhuri - chr1g17 - 29330173

Alec Cavell-Taylor - amct1g17 - 09298467

Ryan Caby - rc5g17 - 29348714

Adebisi Akadu - ama1g20 - 32343566

Contents

1.1 Introduction

This project is aimed at the design and development of a Gecko robot. Gecko's are reptilians with microscopic hairs on their hands which gives them the ability to stick to surfaces using the Van Der Waals force. This project focuses on the Gecko's ability to stick to and navigate vertical surfaces. It was decided to use a suction system to adhere to surfaces rather than alternatives.

The main goals for the project were: to design a robot that is able to climb vertical surfaces, design a suction cup that can adhere reliably to most walls, and to have the robot controllable wirelessly by a user on the ground. Some additional stretch goals were to have the robot move laterally across the wall and to have the robot climb up the wall autonomously.

The Gecko Robot build is divided into phases; the mechanical, movement, suction, control and current monitoring systems.

1.2 Mechanical Design

1.2.1 Movement System Design

Many different designs were considered for achieving the desired motion for the robot. The design had to be capable of moving the robot up the wall while being achievable to implement within the scope of the project.

The first major design considered involved using four gecko-like arms each with suction cups attached to each "hand". The arms would be made up of two parts, an upper arm (bicep) and a lower arm (forearm) which the suction cups attach to (see Appendix figure A.1). To obtain this motion, the forearms would need to rotate inwards as the upper arms pull down, else a lateral force would be exerted on the suction cups which could detach them from the wall. This would have resulted in a complex design and each arm would require at least one motor, increasing the weight of the robot, so this idea was not developed any further.

The second design that was seriously considered involved using a rotating leg design shown in figure A.2. The tops of the legs would be attached to motor axles that rotate perpendicular to the wall, with suction cups on the ends of the legs such that they follow a similar circular path. This design had some advantages, being capable of driving all 4 legs from a single motor and capable of overcoming small obstacles on the wall, but this design removes the possibility of moving laterally across the wall as well as introducing many difficult design features.

The final design settled upon involved using a single arm that would move linearly, along a guide rail pulling the robot vertically up the wall. A suction cup would be attached to a head that moves down the arm, pulling the robot up the wall, and a suction cup on the body holds the robot in place while the head moves back to the top of the guide rails. The suction cup on the guide rails would also need to move horizontally away from the wall when moving back to its start position, to prevent dragging it across the surface. This design is simpler than the other designs as it only requires one arm to be built with fewer difficult mechanisms to construct. Additionally, extra arms could be added to the robot to obtain constant movement, and they could be placed on a rotating platform to achieve lateral movement across the wall.

1.2.2 Movement System Implementation

The movement system achieves 2 axes of motion, a vertical one that moves the robot up and down the wall, and a horizontal one which moves the top suction cup away or towards the wall. The design consists of 6 main components: a primary guide rail, which achieves the vertical axis of motion; a secondary guide rail which achieves the horizontal axis; an actuator which links the 2 guide rails and enables movement relative to one-another; motor mounts for the primary and

secondary motors; and a suction cup hold attached to the secondary guide rail. Images of the final components can be found in Appendix B.

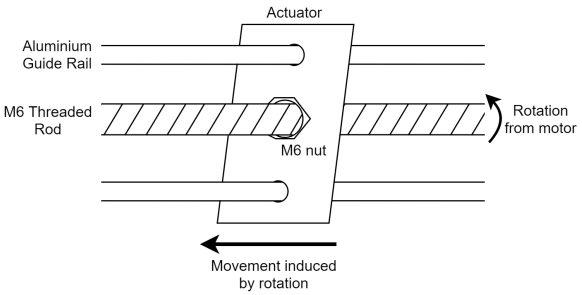


Figure 1.1: Actuator Diagram

Each guide rail consists of a threaded rod attached to the motor with a connector, and 2 aluminium rods parallel to the threaded rod such that they are stationary relative to the thread (figure 1.1). The actuator piece (figure B.1c) is free to slide up and down the aluminium rods while the threaded rod is threaded onto a nut press-fitted into the actuator. The aluminium rods restrict the actuator from rotating with the threaded rod, so it is forced to move up or down the threaded rod, achieving motion controlled by the motor.

The secondary guide rail (Figure B.3 provides a range of motion away from and towards the wall. This is necessary to prevent the robot from dragging against the wall as it moves. When attaching the moving suction cup to the wall, the secondary guide rail extends, moving the suction cup towards the wall until it comes into contact and the suction sticks it to the wall. The body suction cup then detaches from the wall and the secondary guide rail continues to extend, lifting the body away from the wall. The primary guide rail then pulls the body vertically without it dragging against the wall.

The moving suction cup is attached to the secondary guide rail through a bearing, reducing the forces being exerted onto the suction cup, particularly rotational "twisting" forces.

To prevent the actuator from moving too far, micro-switches acting as limit switches were placed on either side of the primary guide-rail. This also introduces feedback to the controller, allowing the robot to move the primary guide-rail automatically without the need for human-input. Although limit-switches could be added to the secondary guide rail to achieve total automated control, this wasn't completed before the end of the project.

To reduce wiring to the moving actuator, the secondary motor is powered through the primary guide rail rods. Since these aluminium rods have a fairly large cross-sectional area, the resistance up them is negligible. Thick copper loops are soldered to the secondary motor and looped onto the guide rails, held in place by the actuator piece such that they are in constant contact with the rods.

1.3 Suction System

One of the most crucial components in a robot designed to climb vertical walls, was a method of adhesion to such surfaces. For this a couple of different approaches were considered, either using a very high surface area material, or using pumped suction cups. It was decided to use suction cups for this project due their strength, reliability, and the difficulties associated with manufacturing a high surface area material.

The suction system had two main components, the suction cup itself, and a system of vacuum pumps and possibly valves to control the vacuum level in each cup. These would be connected by flexible tubes.

1.3.1 Suction Cups

Commercial suction cups designed to be used with a pump were found to be expensive, and uncommon. However they did have great documentation and, from reading this it was decided that it should be possible to manufacture independently. The ZP Series cups from SMC were a great reference, and from the datasheet [2], values for a cup diameter could be calculated, as it contains a lookup table for attachment force, and other recommendations.

The design for the DIY suction cups consisted firstly of a 3D printed rigid part, to give the suction cup shape and to contain the attachment point for the robot, and the tube connection. An example of the rigid component can be seen in figure C.2c. There would also be a skirt around the cup made of a flexible material to create a good seal on a surface. A number of materials were considered for this such as liquid silicone and latex. In the end a liquid latex was used, with the aim of casting it around the rigid component in moulds.
The initial suction cup design in figure C.1a involved a hemispherical bottom section to hold a vacuum. It had a hexagonal stem to serve as a strong attachment point from a few different angles, and a hexagonal hole at the top that an M6 nut could be press fitted into, in order for a tube barb to be screwed in. The planned moulding process was to pour latex over the cup and have it spill out over a smooth surface. Due to the steep angle that the cup hits the surface which it is on, this would give very little adhesion to the cup and so this design was modified to rectify this.
The next few variations C.1b of the rigid component flattened the base of the suction cup to allow better adhesion of latex, and to reduce the volume inside the base (so that a vacuum could be produced more quickly). The diameter was also increased in every iteration, up to 64mm C.1d, as this was calculated to roughly give enough force to hold 3-4kgs (much higher than expected for this application). Finally the stem was reinforced as this was a failure point when testing with heavier weights C.1d.
The casting process also had a number of iterations. At first it involved placing the cup flat on glass and letting latex spill over it out to an outer limit C.1f. This method was found to produce a skirt that was too thick to be "vacuumed" on to a surface, but also could not be compressed into the surface to create a seal. To fix this the rigid component was placed on a cone mould C.1e C.2d, and the latex was allowed to flow down vertically and then out to an outer wall. This moulding process can be seen in figures C.2 and C.2e.

1.3.2 Pump and other components

The second component to the vacuum system consisted of vacuum pumps and valves. The initial design considered was to use one vacuum pump for both suction cups, connected by solenoid valves that would allow the vacuum in each suction cup to be turned on or off by connecting/disconnecting from the pump. However after purchasing the solenoid valves, it was discovered that they were too leaky to hold a vacuum when disconnected. This led to a change in the design to have two vacuum pumps directly connected to each suction cup. The pumps used were Adafruit [1] 4.5V Vacuum pumps, which were rating for -55KPa. This pressure was more than enough to create the estimated force in each suction cup. In order to control these pumps with the ESP32, two simple MOSFET driver circuits were set up on two pins of the ESP, switching the main 5V rail on and off. This motor driver circuitry can be seen in figure F.3.

1.3.3 Testing

In order to test the different suction cups, a simple set up was used consisting of a vacuum pump powered by a bench top power supply, attached to the suction cup under test by a long tube. The suction cup was initially tested by trying to stick it to flat glass surfaces. If this initial test was passed, it underwent it's main testing by sticking it to a glass window, and then hanging increasing weights on the cup stem to assess it's strength. During this testing phase, a number of the early prototypes failed to hold even their own weight on the glass wall. However later prototypes worked extremely well, easily holding 2kgs (which was the largest weight that could be easily attached).

1.3.4 Problems and Commercial Cups

One major issue that was encountered with the DIY suction cups was reliability. After a while the latex on them seemed to dry out to a level where it was not as flexible, and so could not form such a good seal on surfaces. This led to the reliability of these suction cups dropping greatly, so much that it was decided they were too unreliable to be used on the main robot. As an alternative, small 40mm suction cups were purchased C.2g. A small hole was drilled in the center of the cup for the tube barb to fit in, and the cup was mounted using a custom 3D printed part D.1b. These cup did not take quite the same force as the DIY ones (due to a smaller diameter) but were much more reliable in creating a vacuum on a surface and so were used in the final robot.

1.4 Embedded Hardware and Software Design

For the software an RTOS (Real Time Operating System) was chosen to manage several tasks that would need to work simultaneously and to real time deadlines ensuring a responsive system. The diagram F.1 shows an overview of the system with the inputs on the left shown in red, processing electronics in the center, with tasks in orange, and the outputs on the right in green. The system has three tasks, firstly there is a Bluetooth parsing task that checks for new messages on Bluetooth and places the commands received on to a queue (In purple). These commands are then pulled off the queue by the actuation task and it works through the commands sequentially allowing up to 10 commands to be queued. Finally there is the current monitor task that is responsible for measuring the voltage from current monitoring circuitry and makes the decision whether the cup has made good contact and pulled a vacuum or not. This information is relayed to the actuation task for decision making. For controlling the robot a third-party app that allows you to bind messages to buttons is used and a simple numbering scheme is used to number commands and map buttons to them.

1.4.1 Current Monitoring

Initially limit switches were going to be used as are used on the primary guide rail to inform the control system when the robot is in contact with the wall. This could lead to the robot thinking that it has good connection with the wall, when in fact it has not made contact but is close to the wall and will simply fall off. To solve this, it was decided to use a current monitoring technique, as the current usage of the vacuum pumps is relative to the amount of load they are under and could allow for connection quality monitoring. The current increases rapidly when they have made contact with the wall as can be seen in F.2.2. This happens as the motor struggles to pump the remaining air out of the cup. To achieve this a simple circuit was designed (schematic in F.2.1) that uses a shunt resistance is used on the negative leg of the motor power line in-between the motor and ground. This configuration ideally forces all the current through the shunt resistance which only has half an ohm resistance which reduces the amount of voltage dropped. This produces a voltage relative to the current passing through as per ohms law, which is then read by an amplifier relative to ground. This amplifies this with a gain of 9 to get the signal in the range of the ADC. Since the high frequency variations in current produced by the motor are not needed, a low pass RC filter was added with low cutoff frequency to reduce the noise. This is then fed into the micro-controller for processing to determine whether the cup is in contact and has made a good connection with the wall.

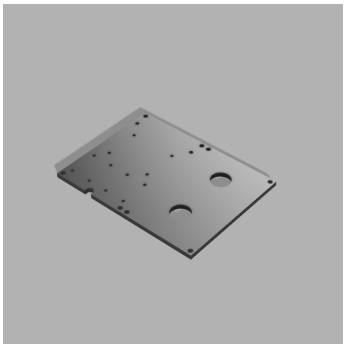
1.4.2 Motor Drivers

Due to the need to rotate the motors in both directions a H bridge is needed for each motor. Originally the H bridges were going to be made using power npn transistors. Two lines connected

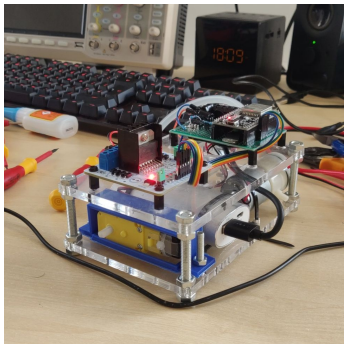
to a micro-controller would be used to control the direction of the motor. Each line is connected to the base of diagonal transistors. After designing and making the H bridges, a smaller premade board was found so this was used instead.

1.5 Chassis

In order to mount the majority of the components in one unit, a chassis was needed. The aim here was to create a simple and relatively lightweight chassis keeping the center of gravity close to the wall. Due to it's speed when producing larger parts, laser cutting was chosen for this, and a two level design was used, with heavy mechanical components on the bottom and light electronics on the upper layer. These layers would be connected by M8 bolts in each corner. The acrylic thickness chosen was 6mm. Whilst this was relatively heavy, it gave the chassis great rigidity and could be used as part of the suction cup bracket, keeping the cup close to the wall. In order to attach the individual components to these sheets, cable tie holes were cut out, and holes for bolts for some components. On the top layer M3 standoff were glued into holes in the acrylic, which the PCBs were screwed on to. Wires were routed through big cut out holes. A 3D render of the chassis can be seen in figure 1.2a and the assembled chassis in 1.2b. Whilst the body was rigid, weight could have been reduced but making a series of cutouts across each sheet.



(a) Render of the chassis plates



(b) Assembled Chassis

There were also a number of parts needed to mount certain components securely. These are displayed in appendix D.1, and include the motor mount, and mounts for both the DIY suction cups and commercial suction cups.

1.6 Conclusion

This project has resulted in a number of innovations to achieve the goals set out. The major goals of the project were achieved, successfully demonstrating a robot that utilises a novel method of vertical locomotion. Custom suction cup designs were investigated and were capable of holding upwards of 2kg on plaster walls, however some reliability issues need to be addressed before they are acceptable for use on the robot. With commercial suction cups, the robot is able to reliably stick to glass surfaces, and climb these surface vertically with no support. The robot can also be controlled wirelessly from a smartphone. It is also able to detect when a vacuum seal is achieved through current monitoring.

Other improvements that could be made to the robot would be: to increase the speed of the robot while it's climbing up walls (by increasing the power of the motor responsible); to add lateral movement by having the arm pivot at the body; and to allow the robot to autonomously climb up walls.

Bibliography

[1] Air pump and vacuum dc motor. URL: adafruit.com/product/4699.

[2] Smc zp series datasheet. URL: https://docs.rs-online.com/5bf3/0900766b80026045.pdf.

Appendix A

Movement System Designs

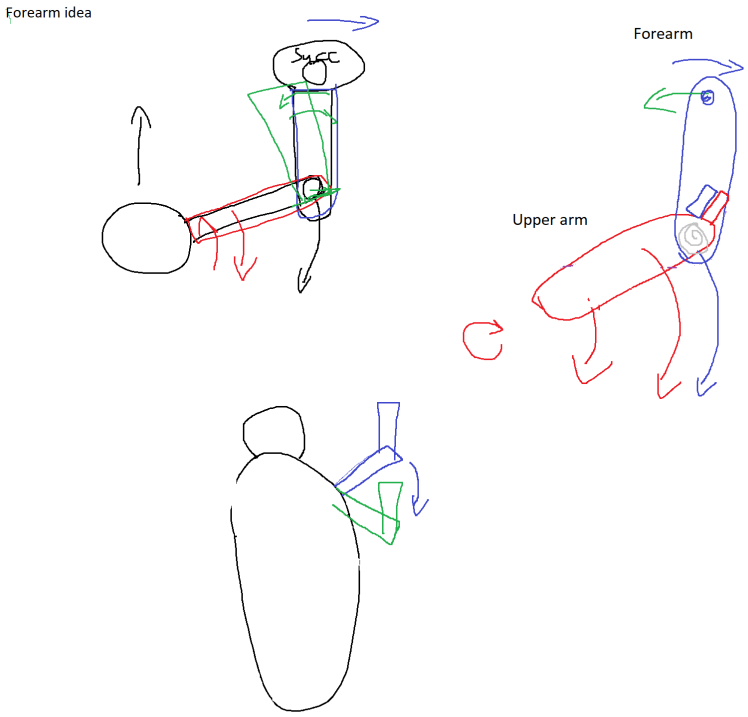


Figure A.1: Forearm Design Sketches

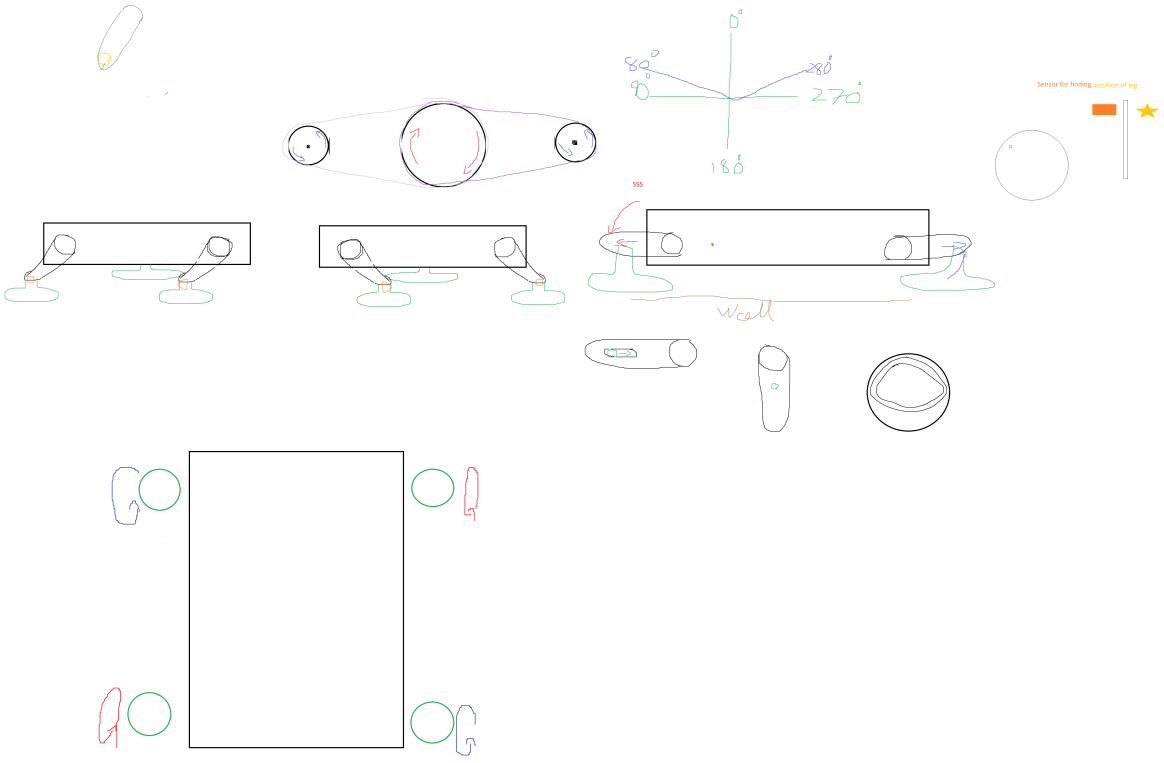
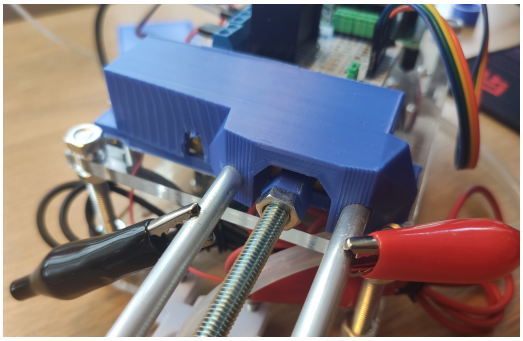


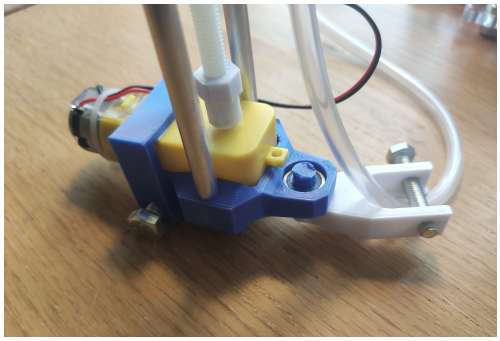
Figure A.2: Walker Design Sketches
https://www.overleaf.com/project/6086a19500f8f1067d0646a0

Appendix B

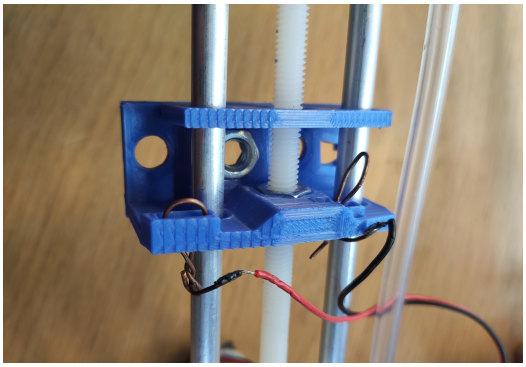
Movement System Components



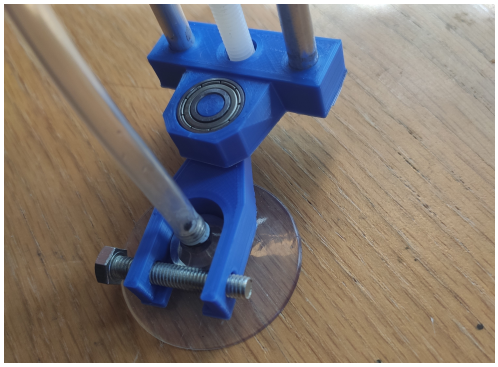
(a) Primary Motor Bracket



(b) Secondary Motor Bracket



(c) Final Actuator Piece



(d) Suction Cup hold

Figure B.1: Individual Movement system components

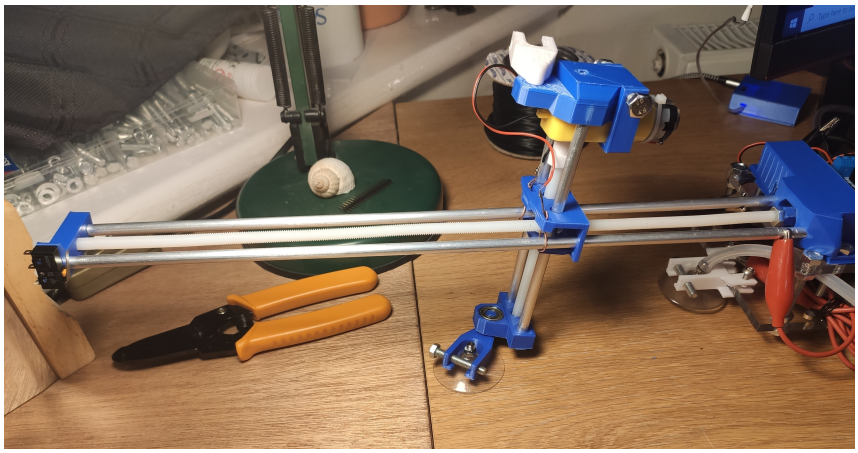


Figure B.2: Primary Guide Rail

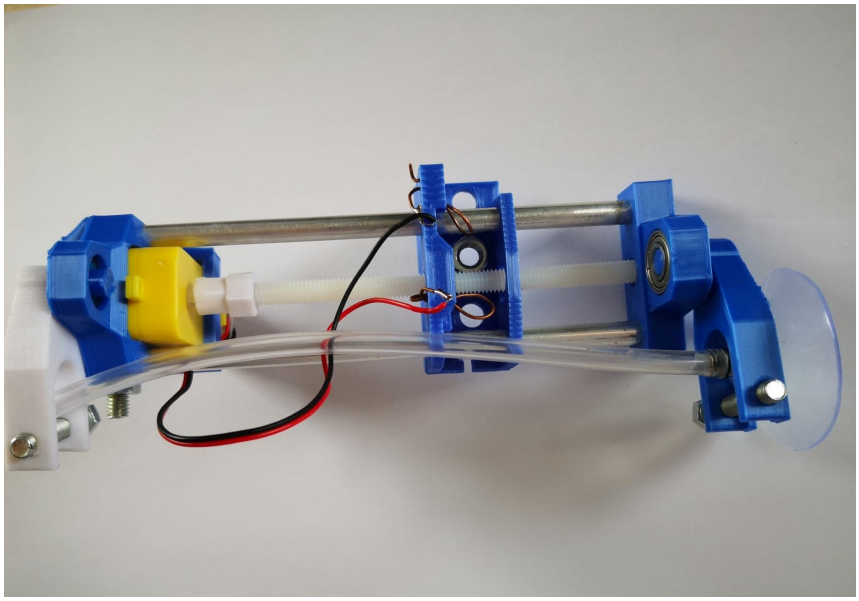
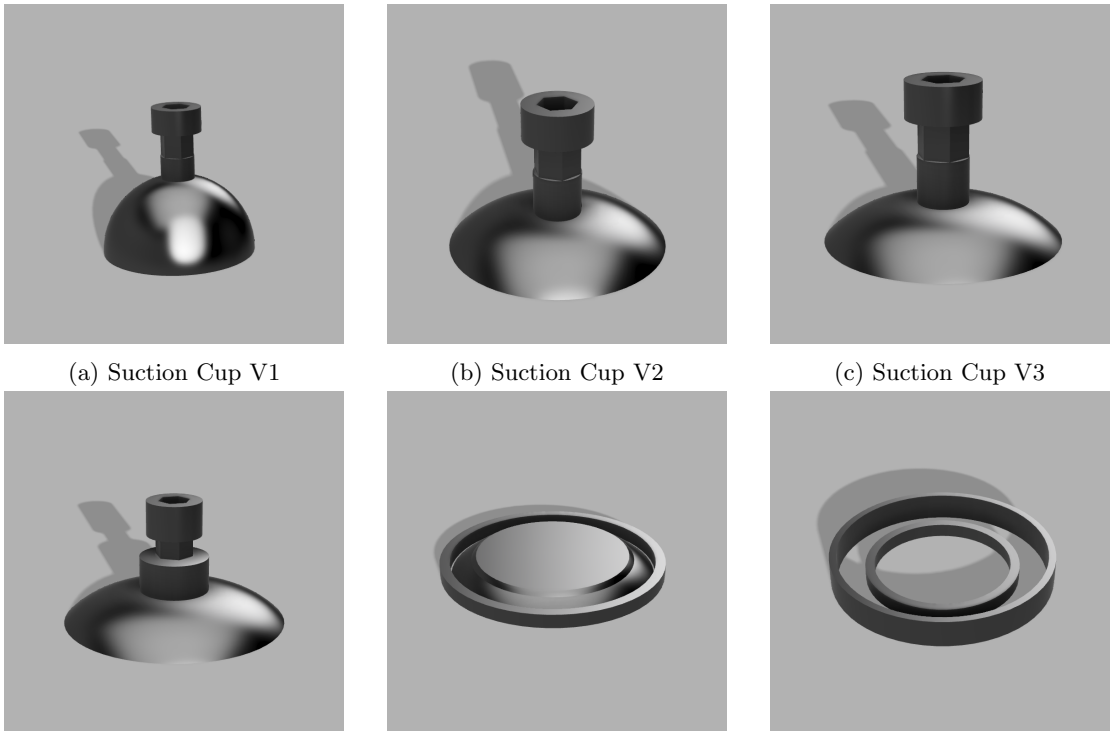


Figure B.3: Secondary Guide Rail

Appendix C

Suction Cup

C.1 Models



(a) Suction Cup V1 (b) Suction Cup V2 (c) Suction Cup V3

(d) Suction Cup V4 (e) Cone Cast (f) Ring Casts

Figure C.1: Renders of 3D models related to Suction Cup Construction

C.2 Images

(a) Suction Cup Prototype

(b) Improved Suction Cup

(c) Suction Cup Rigid Component

(d) Cone Cast Base

(e) Casting Latex

(f) Casting Latex

(g) Commercial Suction Cup

Figure C.2: Images of fabricated suction cup components

Appendix D

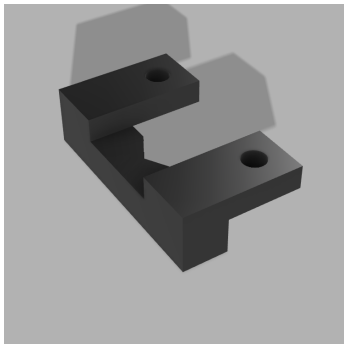
Chassis

D.1 Models and Images

(a) Render of Initial Commercial Suction Cup Bracket

(b) Render of Final Commercial Suction Cup Bracket

(c) Render of Motor Mount



(d) Render of Bracket For DIY Suction Cup

(e) Bracket For DIY Suction Cup

Figure D.1: Images and Renders of 3D models for mounting components

Appendix E

Code Listing

```

#include "BluetoothSerial.h"
#include "suctionCups.h"
#include "MotorControl.h"
#if CONFIG_FREERTOS_UNICORE
#define ARDUINO_RUNNING_CORE 0
#else
#define ARDUINO_RUNNING_CORE 1
#endif

//******************************************************************************
// Debug Messages
//******************************************************************************
#define DEBUG // toggle to enable/disable debug messages

#ifdef DEBUG
 #define DEBUG_PRINTln(x)  Serial.println (x)
 #define DEBUG_PRINT(x)  Serial.print (x)
#else
 #define DEBUG_PRINT(x)
 #define DEBUG_PRINTln(x)
#endif

//******************************************************************************
// Type Defines and Constants
//******************************************************************************
// just to make code more readable
// define what numbers sent from the app correspond to what commands
#define MOVE_FORWARD 0
#define BACK 1
#define LEFT 2
#define RIGHT 3
#define TOGGLE_VACCUM_FRONT 4
#define TOGGLE_VACCUM_BACK 5
#define STOP_MOTORS 6

//define the limit switch pins
#define LIM_SW_FRONT 0
#define LIM_SW_BACK 1
//dont need these ones yet
#define LIM_SW_LEFT 2
#define LIM_SW_RIGHT 3
#define LIM_SW_CENTER 4

//define limit switch for pulling front suction cup away from the wall
#define LIM_SW_FRONT_SUCTION 5

//define the motor & servo drive pins
#define MOTOR_THREADED_ROD_FWD 17
#define MOTOR_THREADED_ROD_REV 5
#define MOTOR_LIFT_SUCTION_FWD 4
#define MOTOR_LIFT_SUCTION_REV 16
#define MOTOR_LEFT_RIGHT 10

//define vacuum pump pins
#define PUMP_FRONT 33// can change
#define PUMP_BACK 32

```

```
59
60 // define current monitor input pins
61 #define CURRENT_MONITOR_FRONT 34
62 #define CURRENT_MONITOR_BACK
63
64 //define threshold that the current must pass to be classed as stuck to the wall
65 #define CURRENT_THRESHOLD 2600
66
67
68 //defines the number of commands allowed on the queue
69 #define QUEUE_SIZE 10
70
71 //*****************************************************************************
72 // Global Variables
73 //*****************************************************************************
74 QueueHandle_t xQueue; // define queue handle
75 BluetoothSerial SerialBT; // define serial bluetooth object
76
77 suctionCup suction_back(PUMP_BACK);
78 suctionCup suction_f(PUMP_FRONT);
79 motor suction_front_motor(MOTOR_LIFT_SUCTION_FWD,MOTOR_LIFT_SUCTION_REV);
80 motor threaded_rod_motor(MOTOR_THREADED_ROD_FWD,MOTOR_THREADED_ROD_REV);
81
82 bool vaccum_front = false;
83 bool vaccum_back = false;
84
85 //*****************************************************************************
86 // Task Prototypes
87 //*****************************************************************************
88 void BLE_RxTask( void *pvParameters );
89 void ActuatorTask( void *pvParameters );
90
91 //*****************************************************************************
92 // Setup
93 //*****************************************************************************
94 void setup() {
95   //*****************************************************************************
96   // BLE Setup
97   //*****************************************************************************
98   SerialBT.begin("GekoRobot"); //Bluetooth device name
99   //bluetooth has to be before tasks are created or causes it to crash
100  // initialize serial communication at 115200 bits per second:
101  Serial.begin(115200);
102
103
104
105  //*****************************************************************************
106  // RTOS Setup
107  //*****************************************************************************
108  xQueue = xQueueCreate( QUEUE_SIZE, sizeof( int ) );
109  // Now set up two tasks and pin them to a core.
110  xTaskCreatePinnedToCore(
111    BLE_RxTask
112    ,  "BLE Rx"   // A name just for humans kinda irrelevant unless we later want to
       check on the task
113    ,  1024  // This stack size can be checked & adjusted by reading the Stack
       Highwater
114    ,  NULL
115    ,  2  // set to 2 priority so we definitley receive messages, with 3 (
       configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
116    ,  NULL
117    ,  ARDUINO_RUNNING_CORE);
118
119  xTaskCreatePinnedToCore(
120    ActuatorTask
121    ,  "Actuator"
122    ,  1024  // Stack size
123    ,  NULL
124    ,  1  // Priority
125    ,  NULL
126    ,  ARDUINO_RUNNING_CORE);
```

```
127
128
129   xTaskCreatePinnedToCore(
130     CurrentMonitorTask
131     ,  "Current Monitor"
132     ,  1024  // Stack size
133     ,  NULL
134     ,  3  // Priority
135     ,  NULL
136     ,  ARDUINO_RUNNING_CORE);
137
138     pinMode(27,OUTPUT);
139
140   // Now the task scheduler, which takes over control of scheduling individual tasks
      , is automatically started.
141 }
142
143 void loop()
144 {
145   // Empty. Things are done in Tasks.
146 }
147
148 //*****************************************************************************
149 //                                  Tasks
150 //*****************************************************************************
151
152 void BLE_RxTask(void *pvParameters)  // This is a task.
153 {
154   (void) pvParameters;
155   int Command = 0;
156   const TickType_t xTicksToWait = ( TickType_t )0xff;
157   for (;;) // A Task shall never return or exit.
158   {
159     if (SerialBT.available())
160     {
161       Command = int(SerialBT.read())-48;
162       DEBUG_PRINT("Received: ");
163       DEBUG_PRINT(Command);
164       DEBUG_PRINT(" :: ");
165       if (xQueue!=0)
166       {
167           xQueueSend( xQueue, &Command, xTicksToWait );
168       }
169       else
170       {
171         DEBUG_PRINTln("Queue Broken");
172       }
173       vTaskDelay(15);
174     }
175   }
176 }
177
178 void ActuatorTask(void *pvParameters)  // This is a task.
179 {
180   (void) pvParameters;
181   for (;;)
182   {
183   int message;
184
185     if( xQueue != 0 )
186     {
187         // Receive a message on the created queue.  Block for 1000 ticks if a
188         // message is not immediately available.
189         //every action e.g FORWARD assumes that both suction cups are active before
      the command and should end with both active aswell
190         // and be fully extended to allow the next command to work properly
191         if( xQueueReceive( xQueue, &message, ( TickType_t ) 1000 ) )
192         {
193             switch(message){
194              case MOVE_FORWARD:
195                DEBUG_PRINTln("Forward Command received");
```

```
                threaded_rod_motor.FORWARD();
                //while()
                /*
                vTaskDelay(1000);
                threaded_rod_motor.STOP();
                vTaskDelay(1000);
                threaded_rod_motor.REVERSE();
                vTaskDelay(1000);
                threaded_rod_motor.STOP();
                */
                /*
                //let go of bottom suction cup
                suction_back.toggleSuction();
                //drive threaded rod motor until LIM_SW_BACK to pull up the wall
                threaded_rod_motor.REVERSE();
                while(digitalRead(LIM_SW_BACK)== LOW);
                threaded_rod_motor.STOP();
                //suction the bottom cup
                suction_back.toggleSuction();
                delay(500);
                //let go of top suction cup
                suction_f.toggleSuction();
                //pull top suction cup away from the wall
                suction_front_motor.REVERSE();
                while(digitalRead(LIM_SW_FRONT_SUCTION)== LOW);
                suction_front_motor.STOP();
                //drive motor until LIM_SW_FRONT
                threaded_rod_motor.FORWARD();
                while(digitalRead(LIM_SW_FRONT)== LOW);
                threaded_rod_motor.STOP();
                //push top suction cup back towards the wall
                suction_front_motor.FORWARD();
                while(suction_front == FALSE);
                delay(1000); //1 second delay for testing instead
                suction_front_motor.STOP();
                //suction top cup
                suction_f.toggleSuction();
                */
                break;
              case BACK:
              threaded_rod_motor.REVERSE();
              //while
              /*
                suction_front_motor.FORWARD();
                vTaskDelay(1000);
                suction_front_motor.STOP();
                vTaskDelay(1000);
                suction_front_motor.REVERSE();
                vTaskDelay(1000);
                suction_front_motor.STOP();

                DEBUG_PRINTln("Back Command received");
                //let go of top suction cup
                suction_f.toggleSuction();
                // pull top suction cup away from the wall
                suction_front_motor.REVERSE();
                while(digitalRead(LIM_SW_FRONT_SUCTION)== LOW);
                suction_front_motor.STOP();
                //drive threaded rod motor in reverse until LIM_SW_BACK
                threaded_rod_motor.REVERSE();
                while(digitalRead(LIM_SW_BACK)== LOW);
                threaded_rod_motor.STOP();
                //push suction cup back towards wall
                suction_front_motor.FORWARD();
                while(suction_front == FALSE);
                suction_front_motor.STOP();
                //suction top cup
                suction_f.toggleSuction();
                delay(500);//delay for half a second to allow vaccum to form
                //let go of bottom suction cup
                suction_back.toggleSuction();
```

```
                //drive threaded rod motor until LIM_SW_FRONT
                threaded_rod_motor.FORWARD();
                while(digitalRead(LIM_SW_FRONT)== LOW);
                threaded_rod_motor.STOP();
                //suction bottom cup
                suction_back.toggleSuction();
                */
                break;
              case LEFT:
                DEBUG_PRINTln("Left Command received");
                //let go of top suction cup
                //drive motor/servo until left limit switch triggered
                //suction top cup
                //let go of bottom suction cup
                //drive motor till center limit switch
                //suction bottom cup
                suction_front_motor.REVERSE();
                threaded_rod_motor.STOP();
                break;
              case RIGHT:
                DEBUG_PRINTln("Right Command received");
                //let go of top suction cup
                //drive motor/servo until right limit switch triggered
                //suction top cup
                //let go of bottom suction cup
                //drive motor till center limit switch
                //suction bottom cup
                suction_front_motor.FORWARD();

                break;
              case TOGGLE_VACCUM_FRONT:
                DEBUG_PRINTln("Toggle Vaccum Command received");
                //toggle power to the vaccum pump
                suction_f.toggleSuction();
                //suction_back.readPin();
                break;
              case TOGGLE_VACCUM_BACK:
                suction_back.toggleSuction();
                break;
              case STOP_MOTORS:
                suction_front_motor.STOP();
                threaded_rod_motor.STOP();
              default:
                //we have gotten here because the command is not one of the
    programmed options
                DEBUG_PRINT("ERROR :: -------> Command Not Recognised ::: ");
                DEBUG_PRINT("Value = ");
                DEBUG_PRINTln(message);
                break;
            }
            //vTaskDelay(1000);
        }
     }
  }
}

void CurrentMonitorTask(void *pvParameters)  // This is a task.
{
  (void) pvParameters;
  const TickType_t xTicksToWait = ( TickType_t )0xff;
  int num_above_threshold = 0;
  int val = 0;
  for (;;) // A Task shall never return or exit.
  {

      val = analogRead(CURRENT_MONITOR_FRONT);
      Serial.println(val);
      bool suction_front = false;
      if(val <= CURRENT_THRESHOLD and val > 2000)
      {
          num_above_threshold++;
```

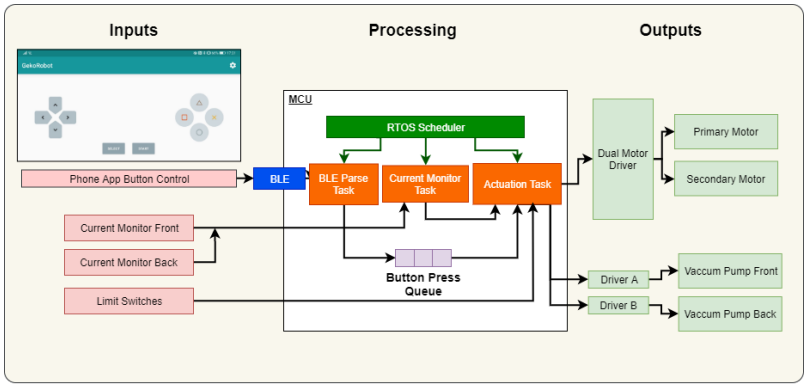
```
337       }
338       else if(num_above_threshold != 0)
339       {
340         num_above_threshold--;
341       }
342       if (num_above_threshold == 3)
343       {
344           Serial.print("STUCK---------------------------------------------->");
345           digitalWrite(27,HIGH);
346           suction_front = true;
347           num_above_threshold = 0;
348       }
349       else if(num_above_threshold == 0)
350       {
351         suction_front = false;
352         digitalWrite(27,LOW);
353       }
354       vTaskDelay(100);
355   }
356 }
```

Listing E.1: Main RTOS Code

Appendix F

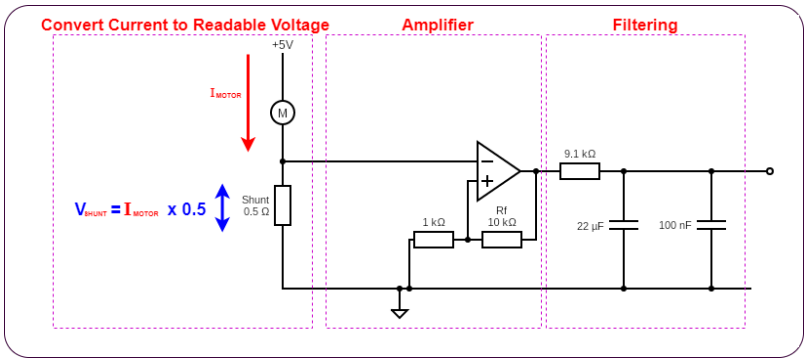
Embedded Electronics

F.1 RTOS System Overview

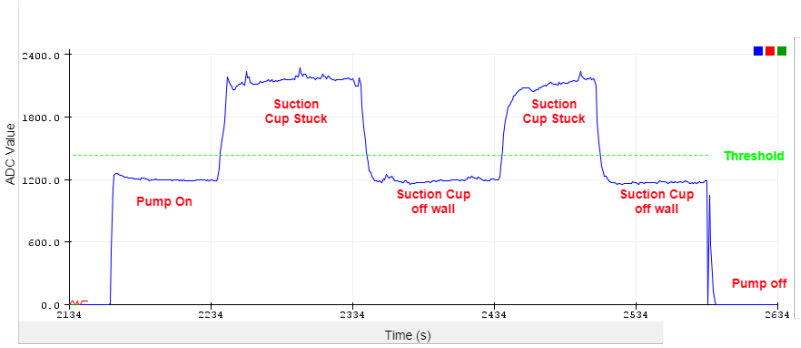


F.2 Current monitoring

F.2.1 Schematic



F.2.2 Example Current Output



F.3 Constructed Electronics