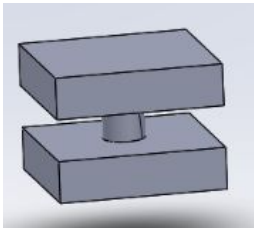
HAcK Group 7

Team Free Block-E

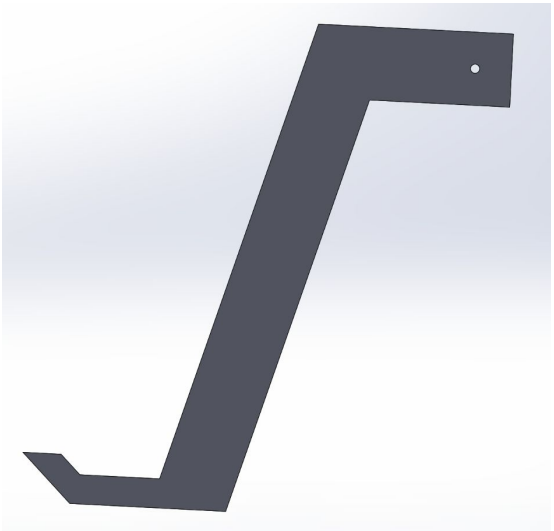
Jerry (Shengjie) Zhu, Julia Stoneburner, Phoenix Tsou

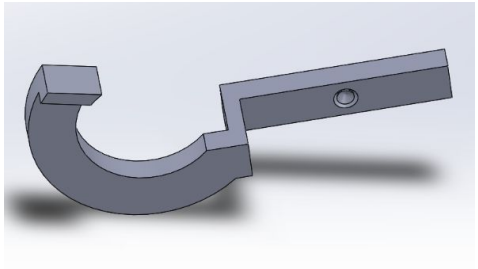
Our Physical Design: Arms

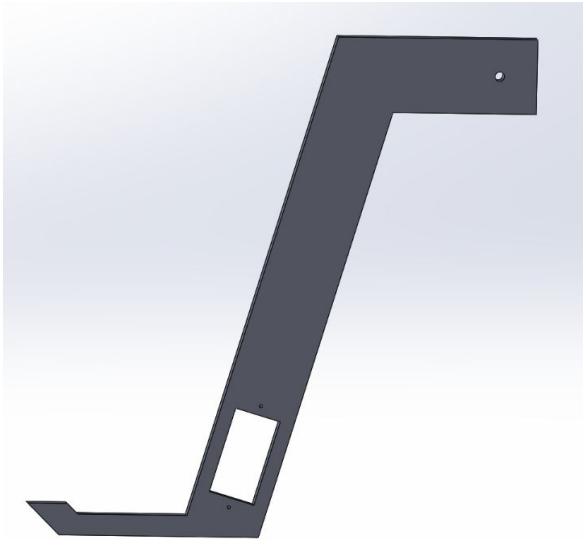


In our first design, our arms were almost identical in shape. The active arm was intended to have an additional smaller arm that functions in a clasping mechanism.

All of our prototypes were laser cut with the exception of one. This was due to laser cutting being much faster than 3D printing.



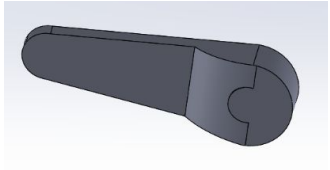




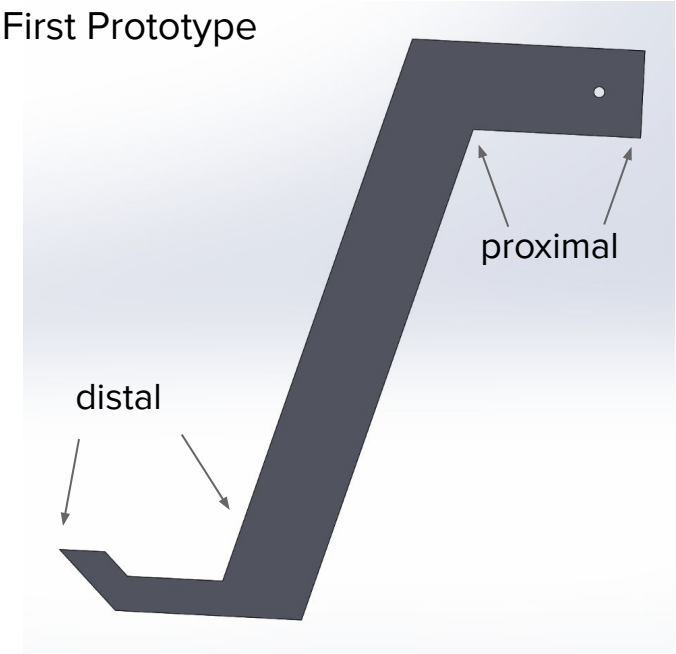
Passive Arm Design

- The proximal section made longer to fully extend past the wheels.
- The height of the distal section intended to do the actual picking up was decreased.
- The alignment holes were changed because the non-flat shape of the plastic clip that attaches to the Servo was not taken into consideration initially.

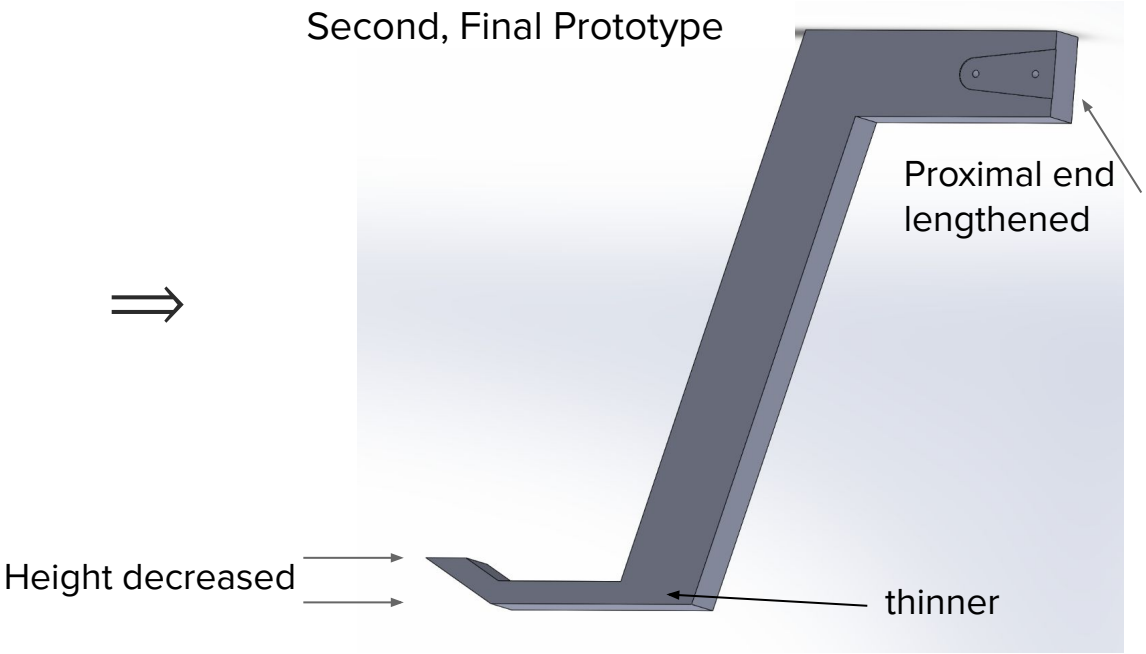
Crude CAD replica of the plastic Servo clip



First Prototype



Second, Final Prototype

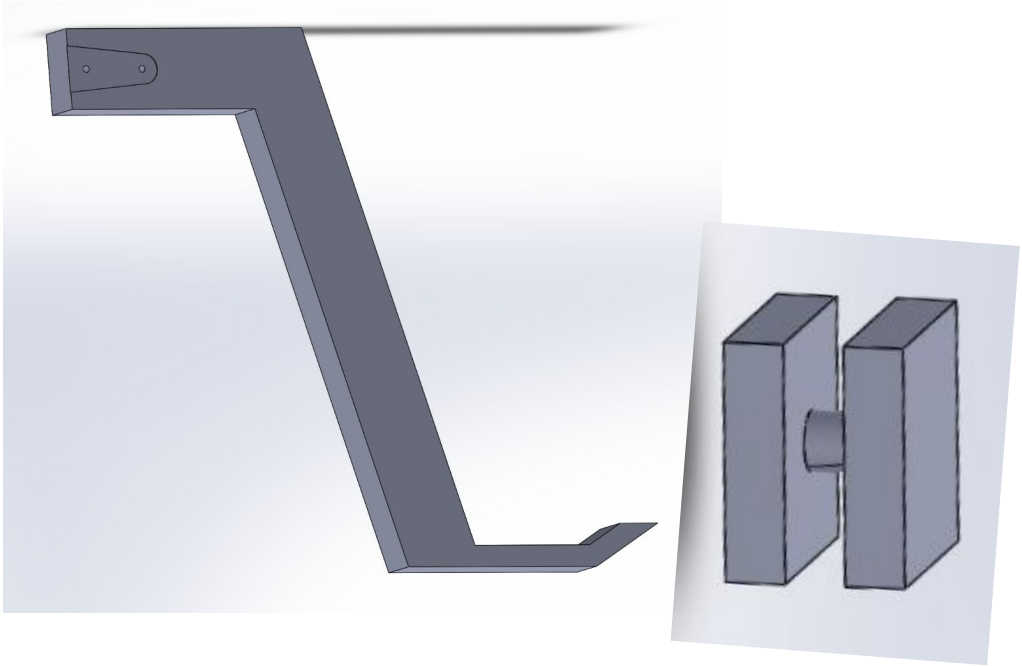


Passive Arm Mechanism

The passive arm suited picking up the *Tie Fighter*.

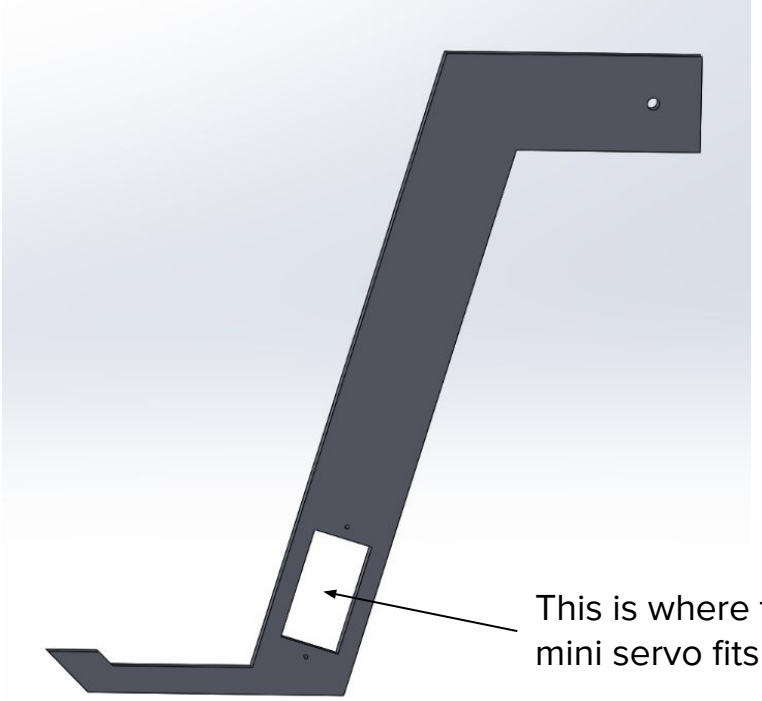
The intention was to hook underneath the center rod, lift it up, and allow it to slide down the arm into the backpack.

In testing this worked well. Unfortunately, we do not have video footage.

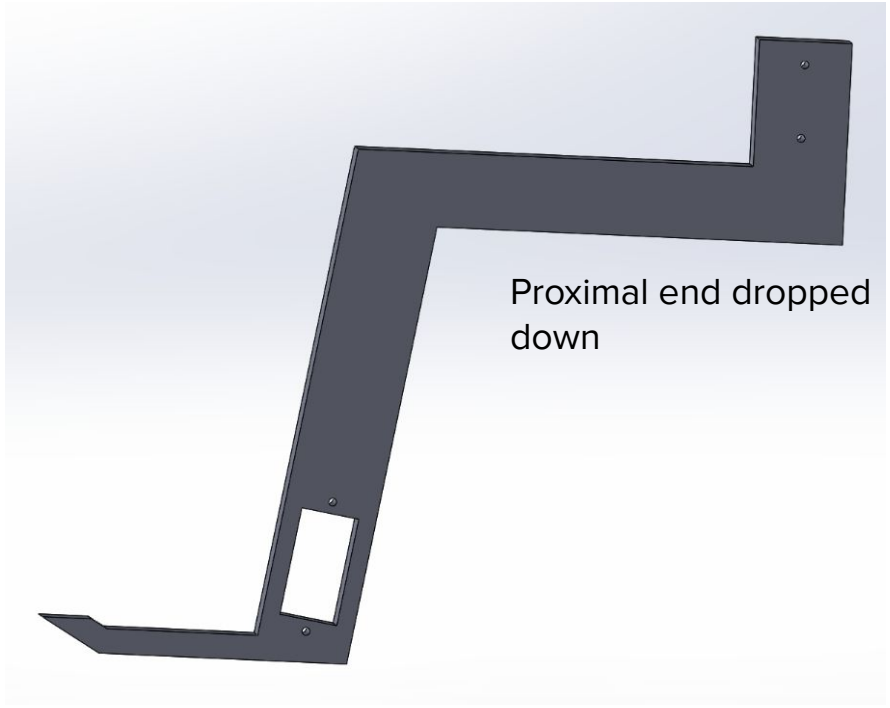


Active Arm Design

Similar adjustments were made to the active arm.



⟹



• The proximal section was altered to fully extend past the wheels while not hitting the backpack when the object is lifted up and dropped.

•The height of the distal section intended to do the actual picking up was decreased. It was also made thinner to fit the handles of the Atomic Death Star.

The Clasping Mechanism

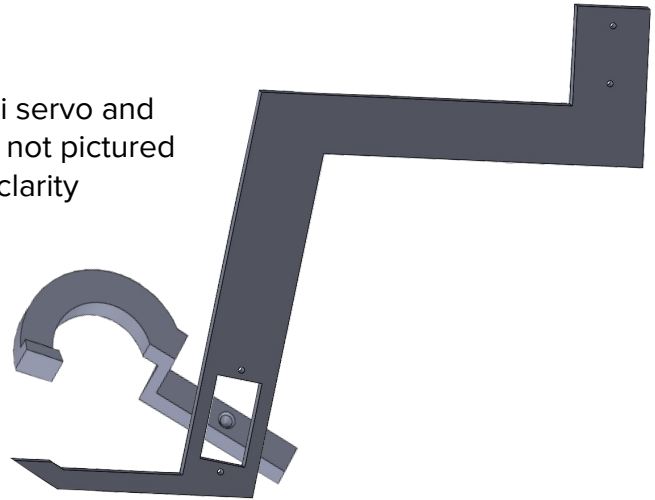
Not the best design.

•Mainly: The Atomic Death Star would get caught in the upper corner of the larger arm.

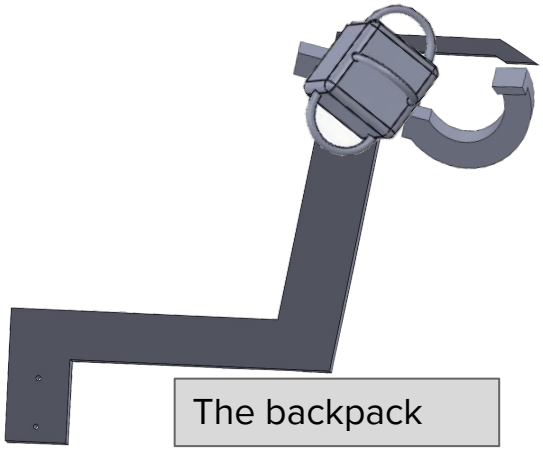
•The servo has to extend back really far to drop the Atomic Death Star into the box due to its curvature. Its curved this much so that the larger arm can be thin enough to slide into a handle of the Atomic Death Star.

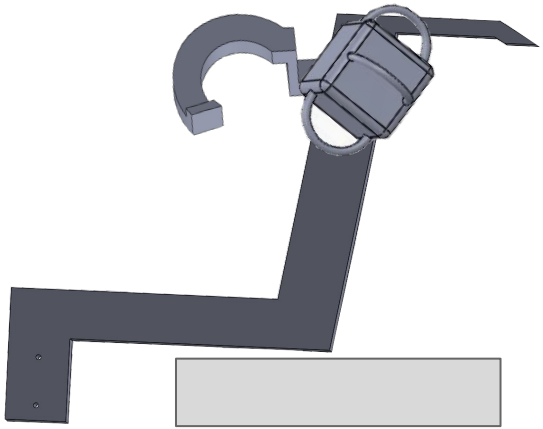
•There is a lip on the clasp I added in case both arms didn't make good contact. This would have hit the bigger arm which I didn't foresee until later.

Mini servo and clip not pictured for clarity



Object gets stuck!

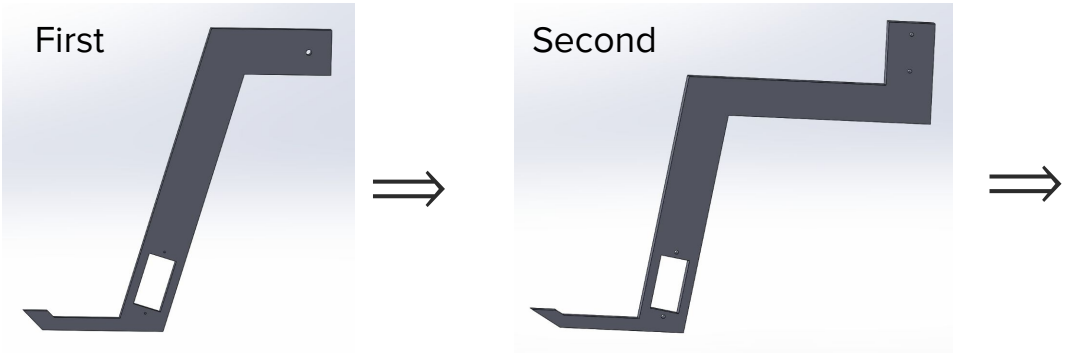


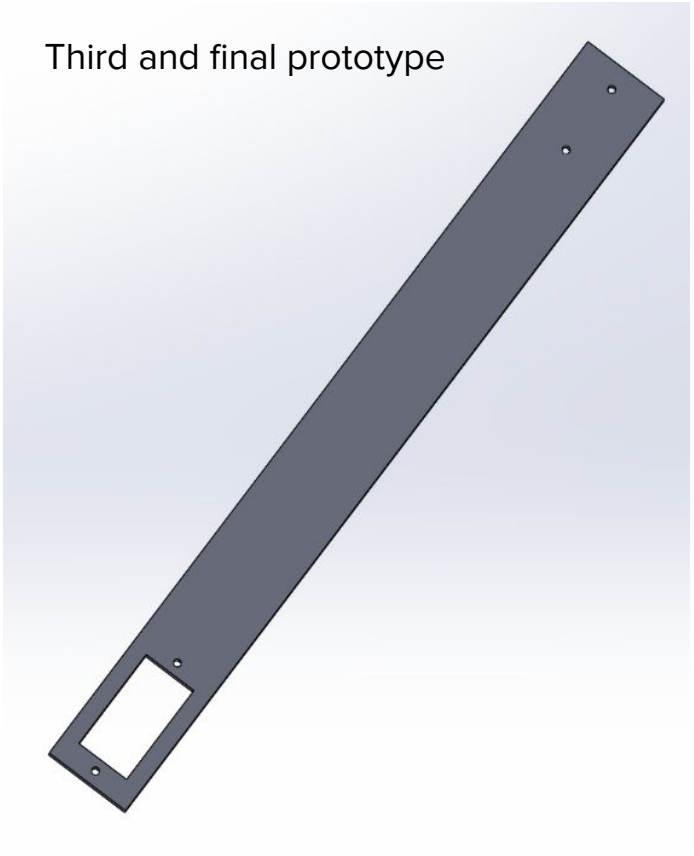


The larger arm rotates back about 180°—the maximum angle for the Servo. The Mini Servo does the same.

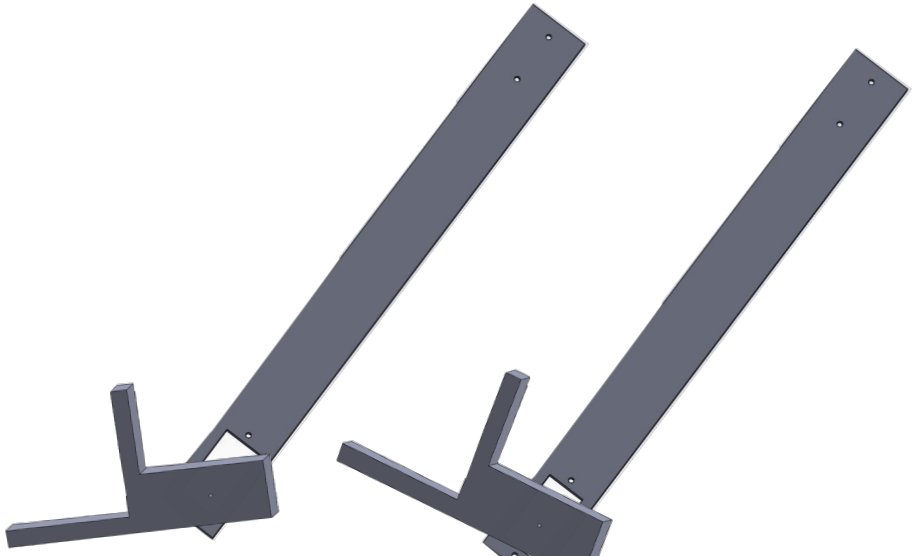
Active Arm Cont'd

The design in the third iteration was changed pretty drastically. While the clasping mechanism was able to pick up the Atomic Death Star by hooking and enclosing on of its 'handles', the object would get stuck on the inside corner in the process of being rotated up. We decided to change our approach.



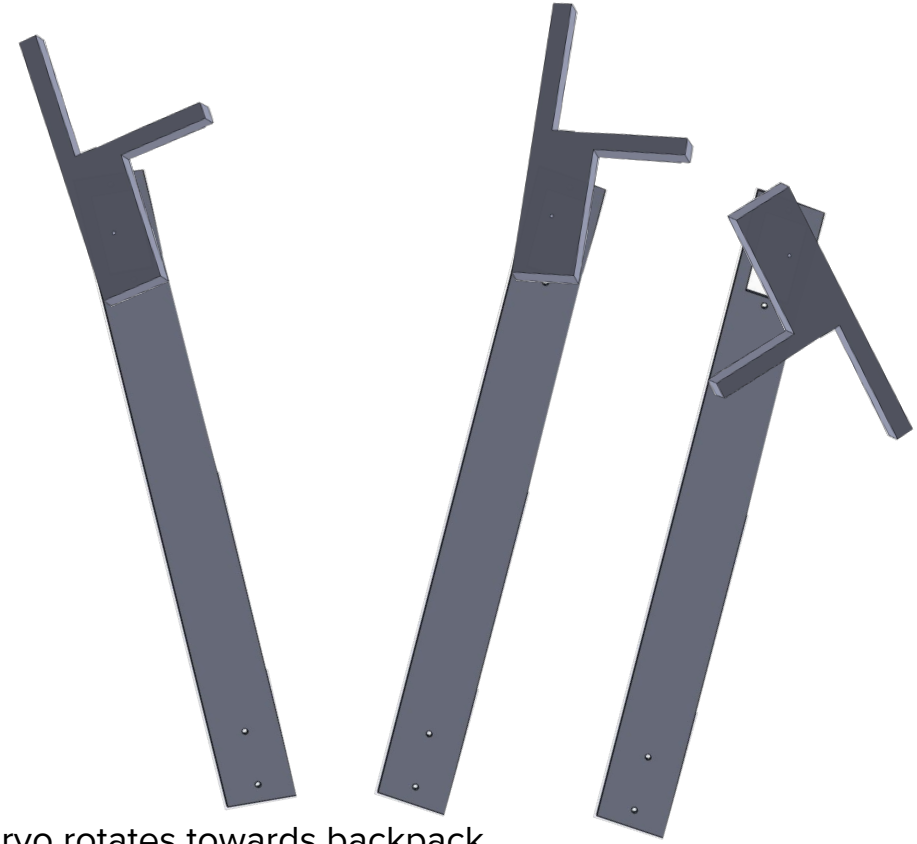


Active Arm Mechanism

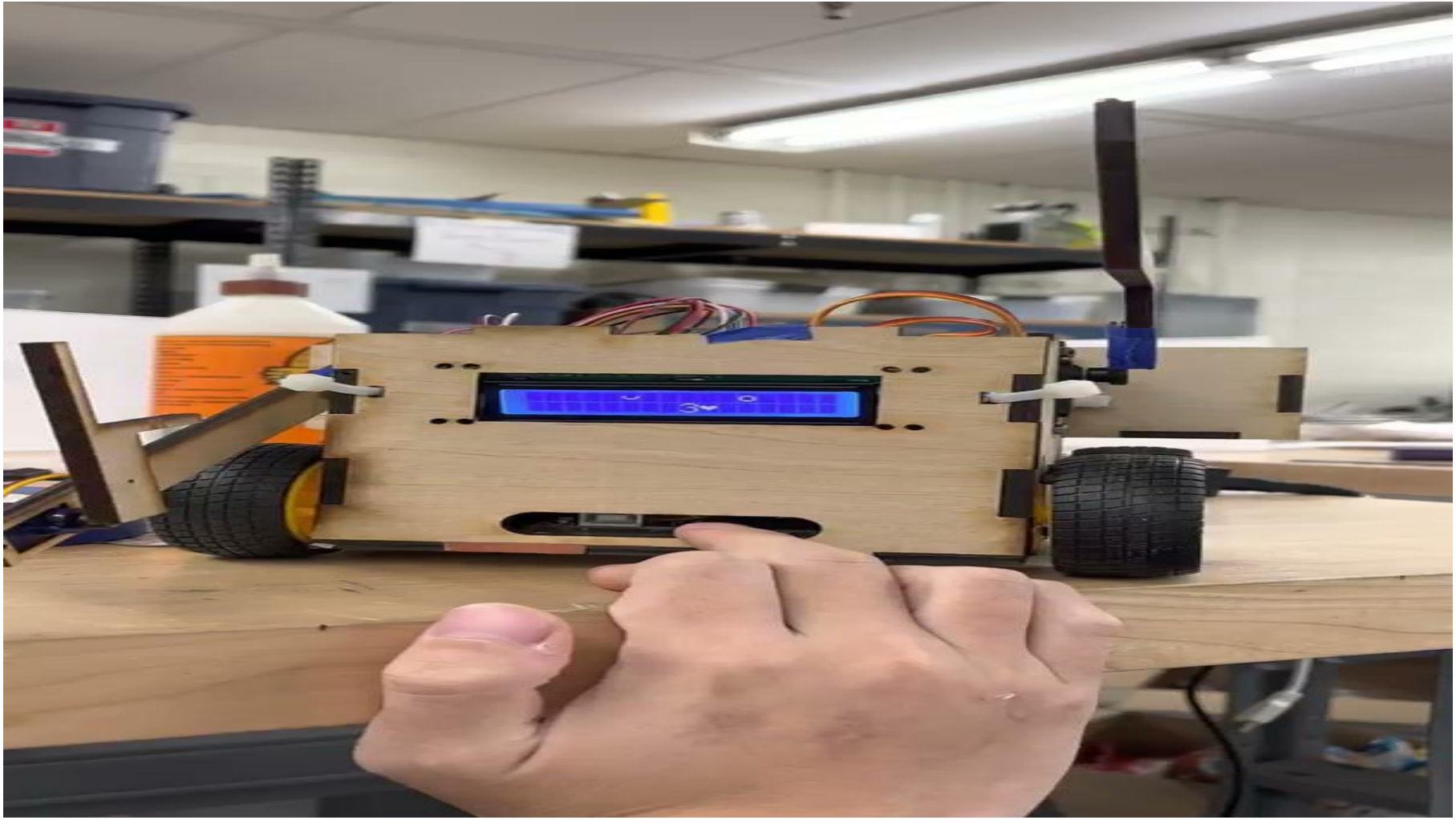


Block-E aligns and rolls into hook of

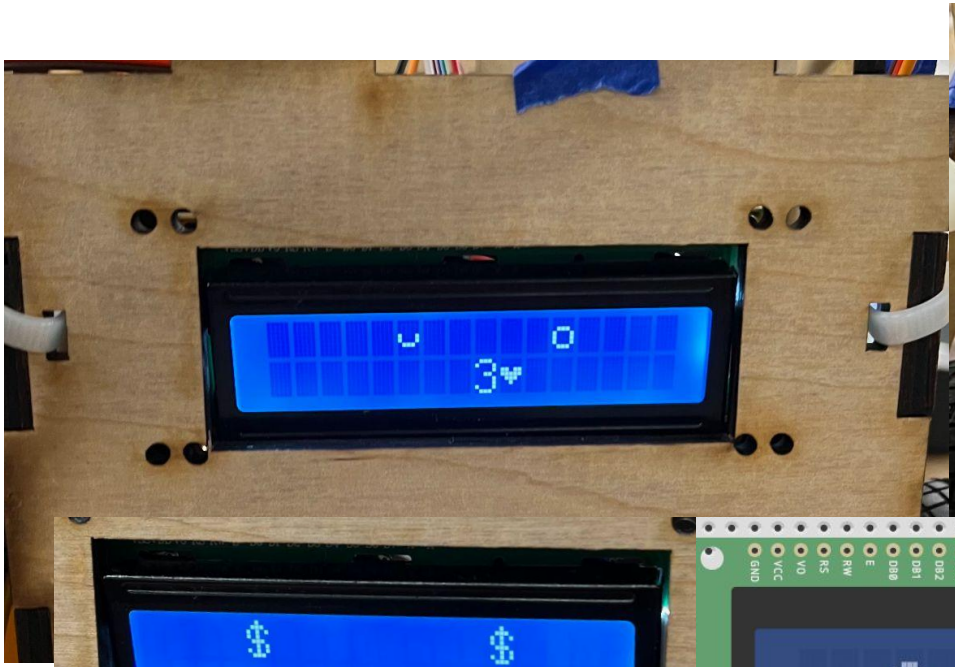
Mini Servo rotates the small arm to lift object

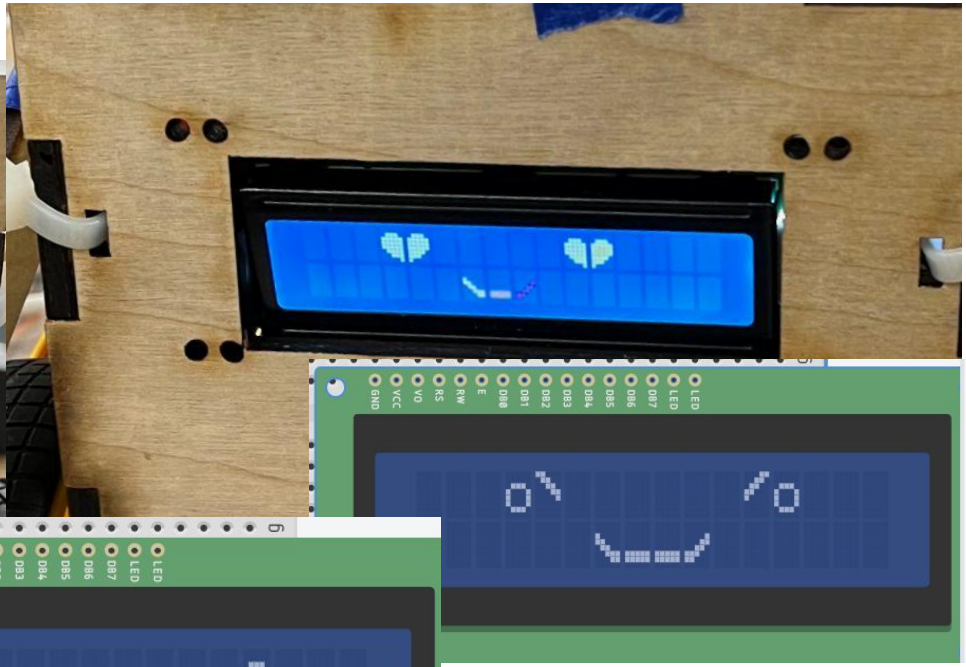


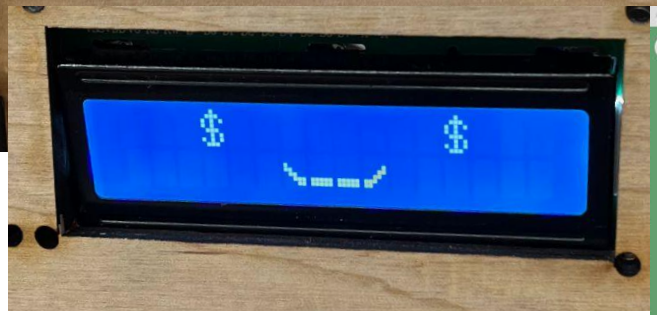
Big Servo rotates towards backpack

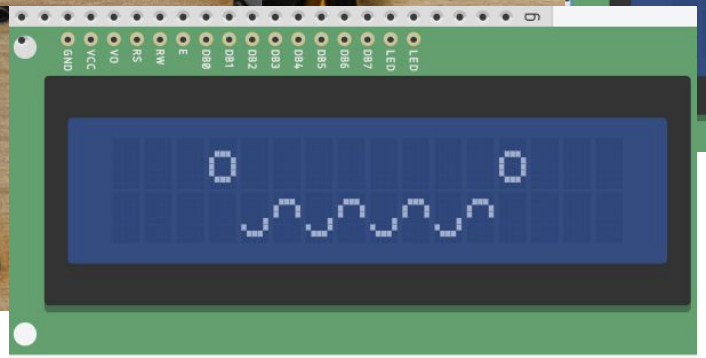


Block-E's Sentience—LCD Screen







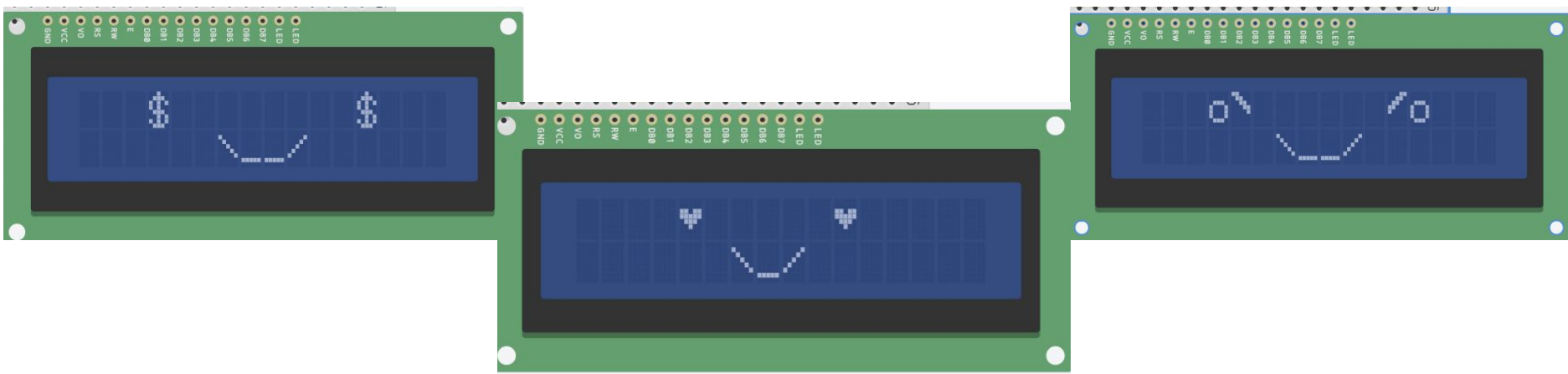


Bonus: Cool LCD During Testing

LCD Iterations

The numbers of customized characters were cut down from 13 to 8 while still making pretty similar faces. The smile was made with keyboard characters instead, and the smaller heart eyes were reused instead of the larger ones.

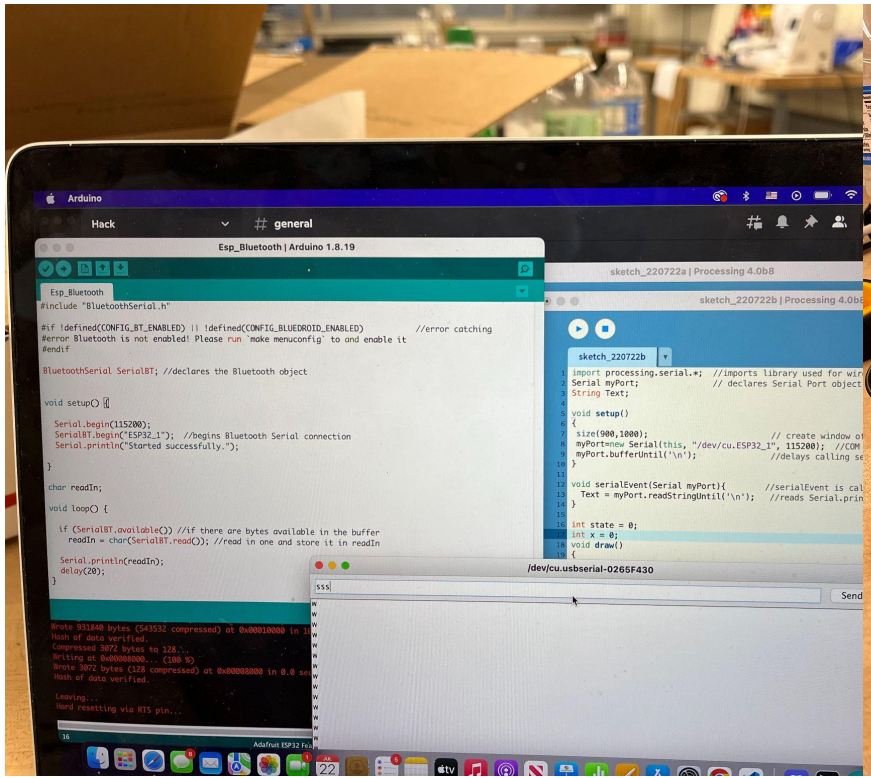
These weren't needed after all. It seemed to have resolved itself once the Arduino board was reset.

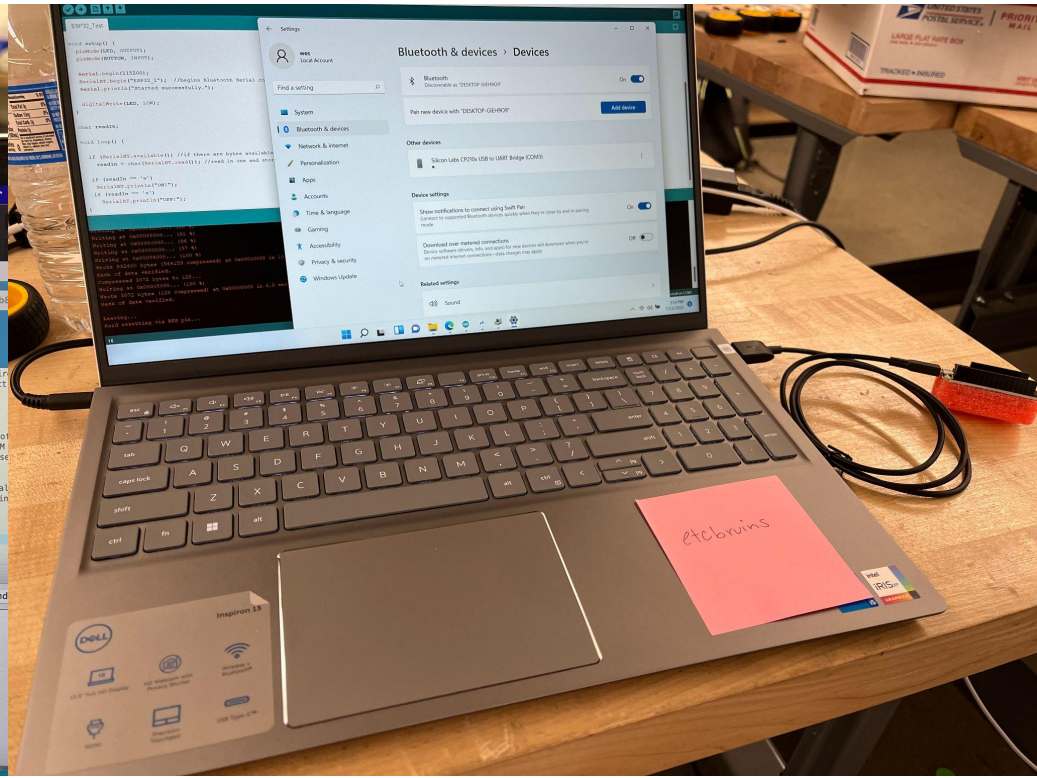


Our First Challenge: ESP32

- ESP32 being the foundation of making everything to work..
- Our laptops had problems connecting and controlling the ESP32 chip...
  - Jerry's laptop can't write... and can't upload codes to the chip (AMD cpu supporting?!)
  - Phoenix's laptop can't read... and can't transmit data between laptop and chip after bluetooth connection.
  - Tried, but time wasted
- Fix: Wes lended us a new laptop on the second day of competition
- The issue on Jerry's and Phoenix's laptop prove to be hardware related since the new laptop worked almost instantly :)

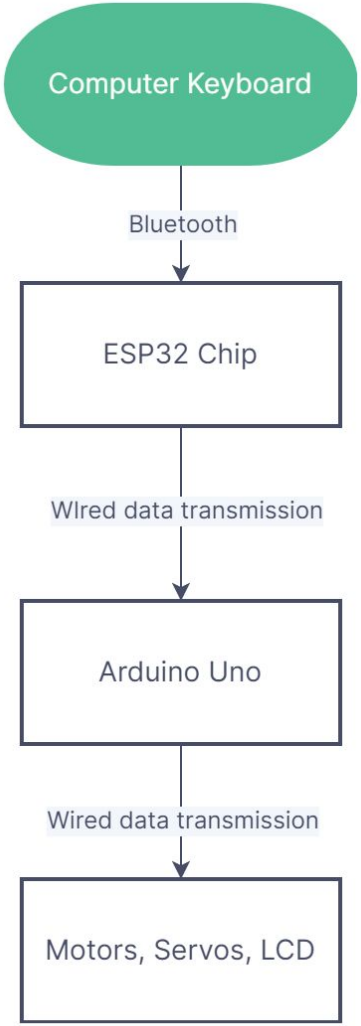
Trying to Debug the Data Transmission...





Our Design: Data Transmission / Work Flow

- The car needs to work remotely and wirelessly
- Data Flow Chart: (Basic)
-



Serial Connection Code

InputCode_Final

```
import processing.serial.*;  //imports library used for wireless comm with the HC-05
Serial myPort;               // declares Serial Port object from library
String Text = "";

void setup()
{
 size(900,1000);                          // create window of this size
 myPort=new Serial(this, "COM9", 115200);  //COM port will likely be different, try different ones until it works (usually one of COM3-COM7)
 myPort.bufferUntil('\n');                 //delays calling serialEvent unitl reaching '\n'
}

void serialEvent(Serial myPort){        //serialEvent is called whenever data is available
  Text = myPort.readStringUntil('\n');   //reads Serial.println() from Arduino
}

int state = 0;
int x = 0;
void draw()
{

  background(237,240,241);         // sets background to a color, parameters correspond to an RGB code
  fill(20,160,133);                // sets color we will use to fill shapes
  stroke(33);                      // sets color used for border around shapes
  strokeWeight(1);                 // sets width of stroke when drawing shapes

  rect(x,700,50,50,10);            //creates a rectangle at x,y with size width,height
  x+=1;
  if (x == 800)
    x=0;

  fill(0);
  textSize(32);
  text(Text, 400, 700);            //draws text at x,y


  if(state == 1){
    myPort.write('p');
  }
  if(state == 2){
    myPort.write(key);
  }
}

void keyReleased(){
  state = 1;
}

void keyPressed(){
  state = 2;
}
```

Hack_ESP32_Group_7 | Arduino 1.8.19

File Edit Sketch Tools Help

Hack_ESP32_Group_7

```
#include "BluetoothSerial.h"
BluetoothSerial SerialBT;
char readIn;

 void setup() {
  // put your setup code here, to run once:
  Serial1.begin(115200);
  SerialBT.begin("ESP32_7");
  while(!SerialBT.available()){
    ;
  }
}

void loop() {
  // put your main code here, to run repeatedly:
  if(SerialBT.available())
     {
      readIn = char(SerialBT.read());
      Serial1.println(readIn);
     }

}
```

1 ESP32 Dev Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on COM4

Our Design: Main Controlling Code (Arduino)

- Basic thought:
- Concise and Straightforward main loop code
- Things to check: Cars going forward? Backward? Left or Right?
- Servo Operations: Left, Right & Micro Servo going back/forth
- Bonus Function: All servo reset and micro servo reset
-

Main Controlling Code Continued

- Features:
  - Clear Pin Declaration, Key binding zones, and delay time.
  - No need to change every number in the document, easy to edit
  - Motor delay time was eventually not used.

```
//Servo Pins Declare!!! ALWAYS FIRST DO RESET SERVO AFTER APPLYING NEW POWER
int Micro_servo_attach = 10;  //The little servo on the top
int Micro_servo_pos = 0;
int Left_servo_attach = 11;
int Left_servo_pos = 0;
int Right_servo_attach = 12;
int Right_servo_pos = 0;
int Servo_delay_time = 7;
int Motor_delay_time = 15;
int Dance_delay_time = 500; //This controls the movement time for car's every dance
```

Main Controlling Code Continued

- Functions with clear names and nomenclature
- One separate function every move! (Car operations, servo operations)

```
int Check_all_reset(char Keyboard_input){
  if (Keyboard_input == All_servo_reset_key)
    return 1;
  else
    return 0;
  }

int Check_micro_reset(char Keyboard_input){
  if(Keyboard_input == Micro_servo_reset_key)
    return 1;
  else
    return 0;
  }


int Check_micro_forward(char Keyboard_input){
  if(Keyboard_input == Micro_servo_forward_key)
    return 1;
  else
    return 0;
  }
```

```
void Left_servo_forward(int *Left_pos){
  if(*Left_pos < 180){
    *Left_pos += 1;
    leftservo.write(*Left_pos);
    delay(Servo_delay_time);
    }

  }

int Check_left_backward(char Keyboard_in
  if(Keyboard_input == Left_servo_backwa
    return 1;
  else
    return 0;
  }

void Left_servo_backward(int *Left_pos){
  if(*Left_pos > 0){
    *Left_pos -= 1;
    leftservo.write(*Left_pos);
    delay(Servo_delay_time);

    }
```

```
int Check_car_forward(char Keyboard_inpu
  if(Keyboard_input == Car_forward_key)
    return 1;
  else
    return 0;

  }

int Check_car_backward(char Keyboard_inp
  if(Keyboard_input == Car_backward_key)
    return 1;
  else
    return 0;

  }

int Check_car_left(char Keyboard_input){
  if(Keyboard_input == Car_left_key)
    return 1;
  else
    return 0;

  }
```

Main Controlling Code Continued

- The self-defined functions make embedding all kinds of operations into the dancing possible.

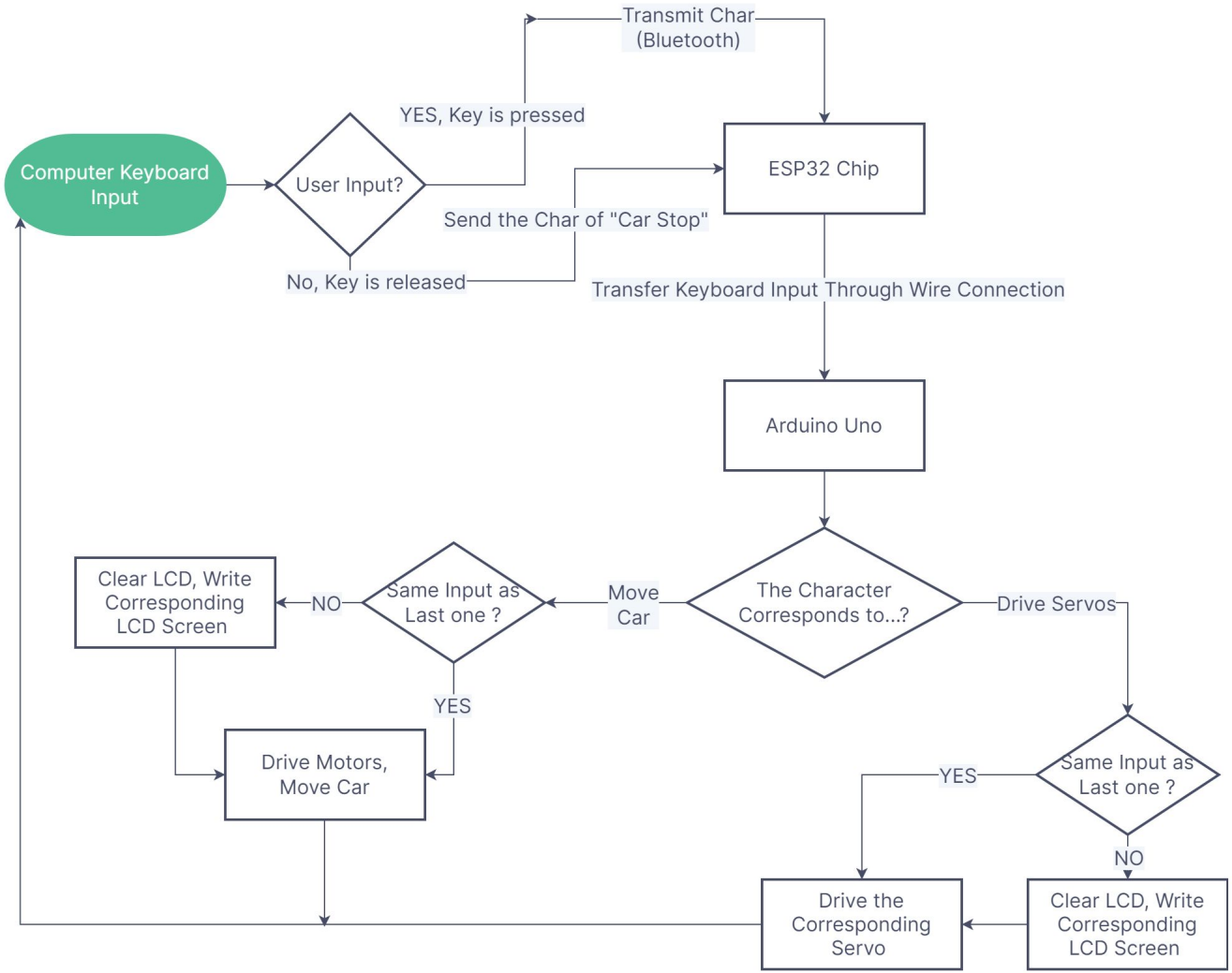
```
void Car_dance(int wheel1_1, int wheel1_2, int wheel2_1, int wheel2_2, int wheel3_1 ,
int wheel3_2, int wheel4_1, int wheel4_2, int *Micro_pos, int *Left_pos, int *Right_pos){
  Car_forward(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel
  delay(Dance_delay_time);
  Car_stop(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel4_2)
  delay(Dance_delay_time);
  Car_left(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel4_2)
  delay(Dance_delay_time);
  Car_stop(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel4_2)
  delay(Dance_delay_time);
  Car_backward(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel
  delay(Dance_delay_time);
  Car_stop(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel4_2)
  delay(Dance_delay_time);
  Car_right(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel4_
  delay(Dance_delay_time);
  Car_stop(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel4_2)
  delay(Dance_delay_time);
  Car_forward(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel
  delay(Dance_delay_time);
  Car_stop(wheel1_1, wheel1_2, wheel2_1, wheel2_2, wheel3_1, wheel3_2, wheel4_1, wheel4_2)
  delay(Dance_delay_time);
  int i = 0;
  for(i = 0; i < Dance_delay_time / Motor_delay_time; i ++){
```

```
for(i = 0; i < Dance_delay_time / Motor_delay_time; i ++
  Left_servo_forward(&(*Left_pos));
  Right_servo_forward(&(*Right_pos));
  }
delay(Dance_delay_time);
for(i = 0; i < Dance_delay_time / Motor_delay_time; i ++
  Left_servo_forward(&(*Left_pos));
  Right_servo_forward(&(*Right_pos));
  }
delay(Dance_delay_time);
for(i = 0; i < Dance_delay_time / Motor_delay_time; i ++
  Left_servo_forward(&(*Left_pos));
  Right_servo_forward(&(*Right_pos));
  }
delay(Dance_delay_time);
for(i = 0; i < Dance_delay_time / Motor_delay_time; i ++
  Left_servo_backward(&(*Left_pos));
  Right_servo_backward(&(*Right_pos));
  }
delay(Dance_delay_time);
Micro_servo_reset(&(*Micro_pos));
Micro_servo_half(&(*Micro_pos));
```

```
if(Check_all_reset(Keyboard_input) == 1)  // By pressing r, we reset all motors to 0.
  All_servo_reset(&Micro_servo_pos, &Left_servo_pos, &Right_servo_pos);



if(Check_micro_reset(Keyboard_input) == 1)
  Micro_servo_reset(&Micro_servo_pos);


if(Check_micro_forward(Keyboard_input) == 1)
  Micro_servo_forward(&Micro_servo_pos);  //Make the servo forward for 1 degree.

if(Check_micro_backward(Keyboard_input) == 1)
  Micro_servo_backward(&Micro_servo_pos);

if(Check_left_forward(Keyboard_input)== 1)
  Left_servo_forward(&Left_servo_pos);

if(Check_left_backward(Keyboard_input) == 1)
  Left_servo_backward(&Left_servo_pos);

if(Check_right_forward(Keyboard_input) == 1)
  Right_servo_forward(&Right_servo_pos);
```

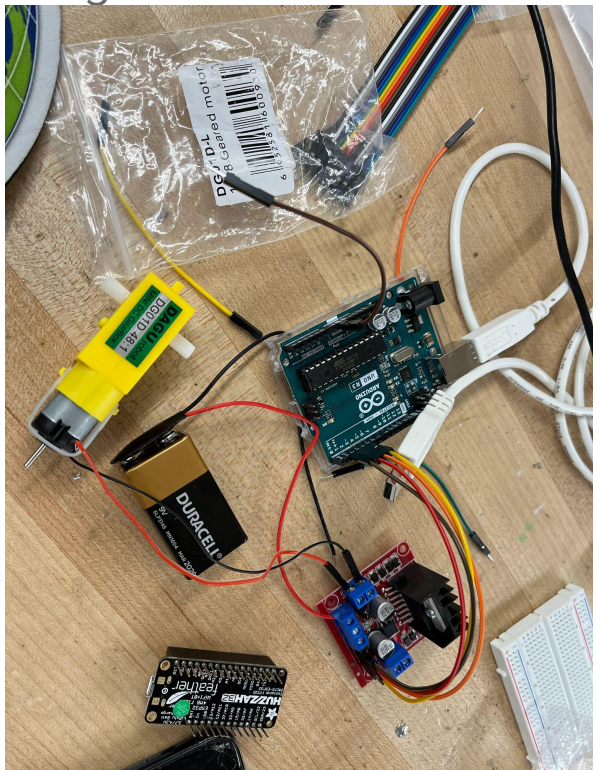
g code and
ed like this:
functions

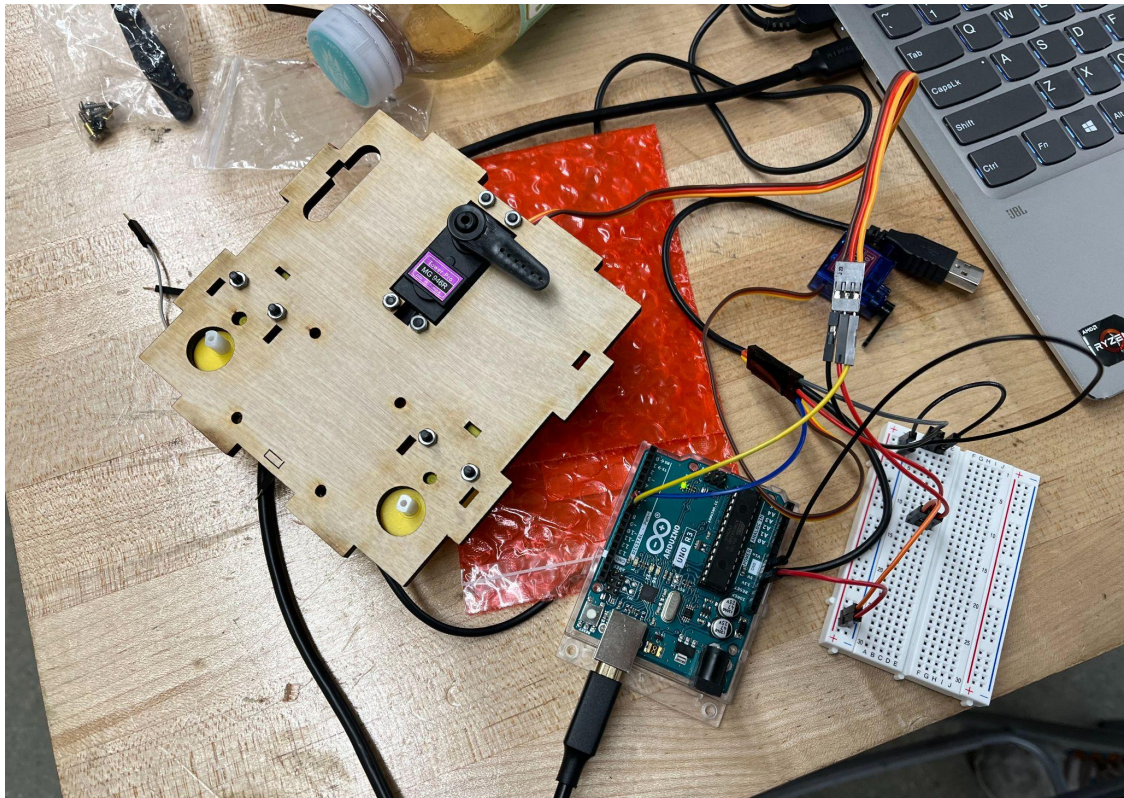
Th


Transmit Char
(Bluetooth)
YES, Key is pressed
Computer Keyboard Input
User Input?
ESP32 Chip
Send the Char of "Car Stop"
No, Key is released
Transfer Keyboard Input Through Wire Connection
Arduino Uno
Clear LCD, Write Corresponding LCD Screen
NO
Same Input as Last one ?
Move Car
The Character Corresponds to...?
Drive Servos
YES
Drive Motors, Move Car
YES
Same Input as Last one ?
NO
Drive the Corresponding Servo
Clear LCD, Write Corresponding LCD Screen

Our Process: Building & Testing the Car

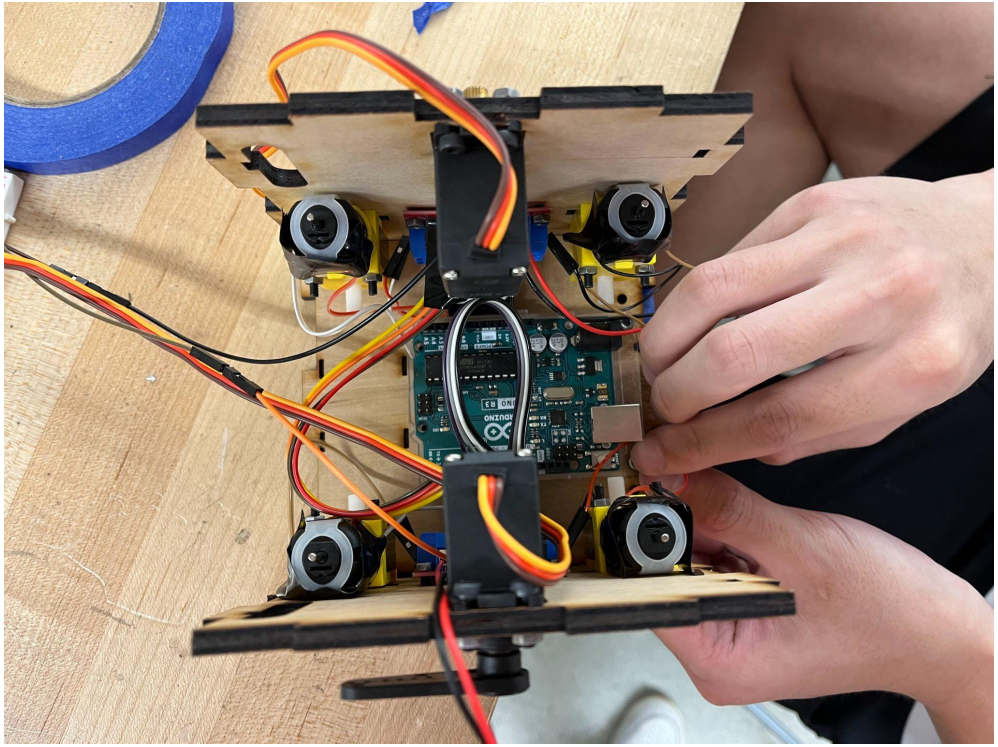
Testing motors and servos!

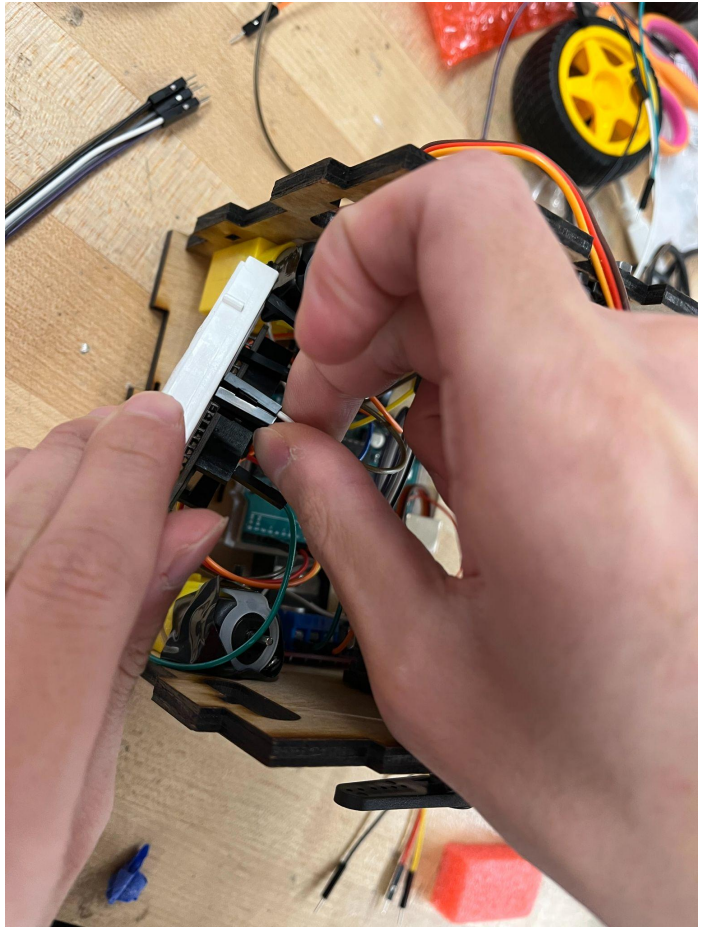




Our Process Continued

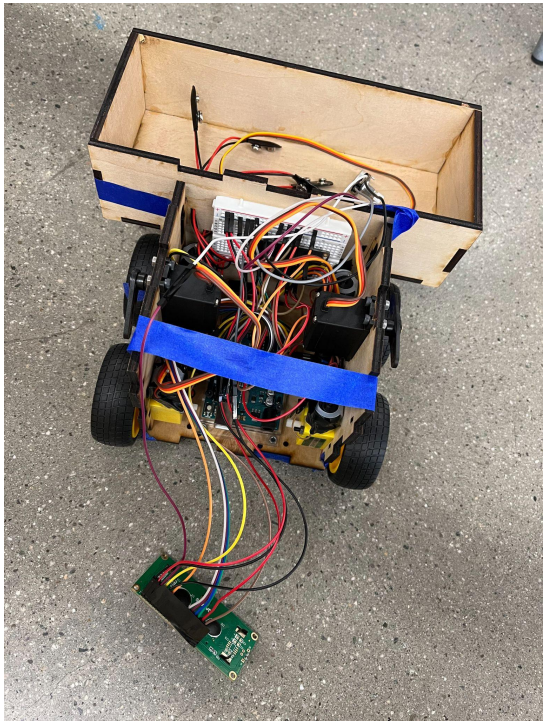
Putting things together...

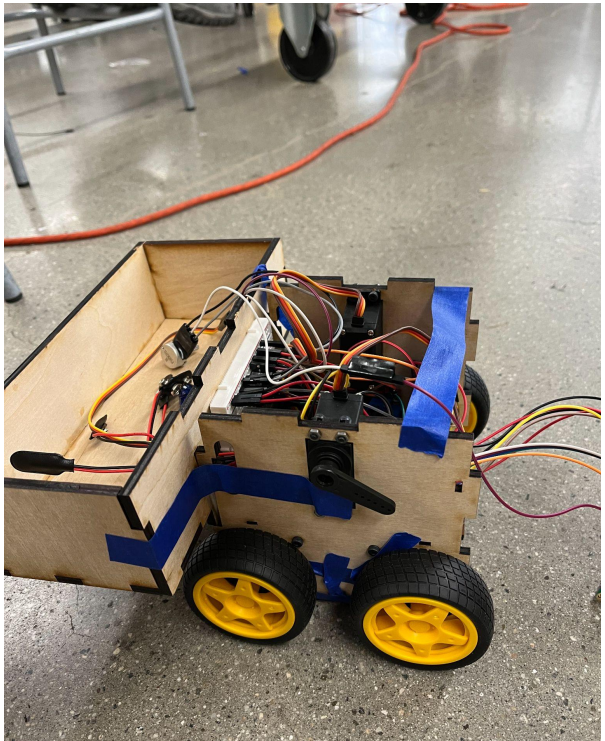




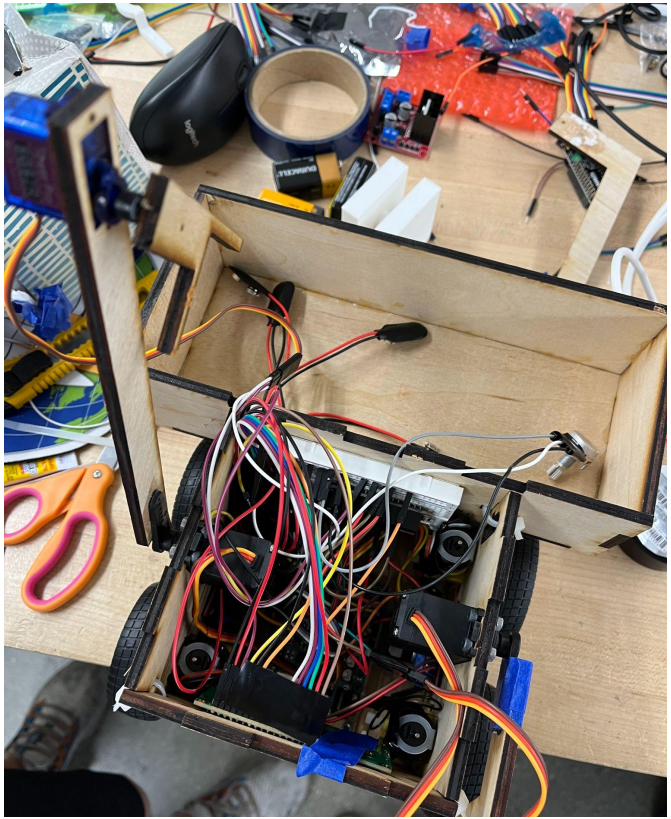
Our Process: Preliminary Testing

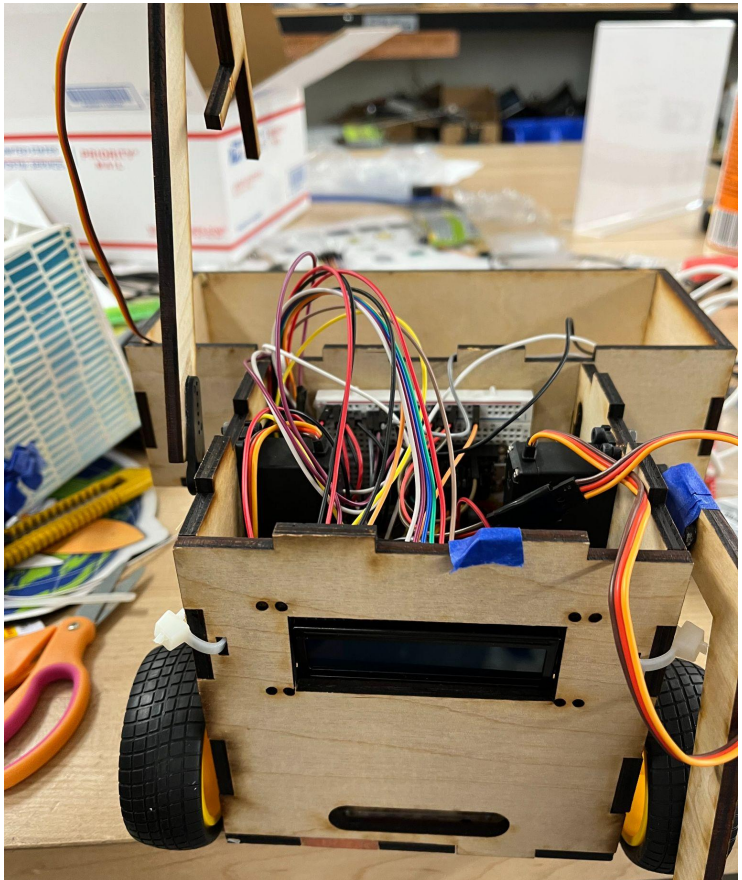
Block-E with his body taped and eyes waggling :)

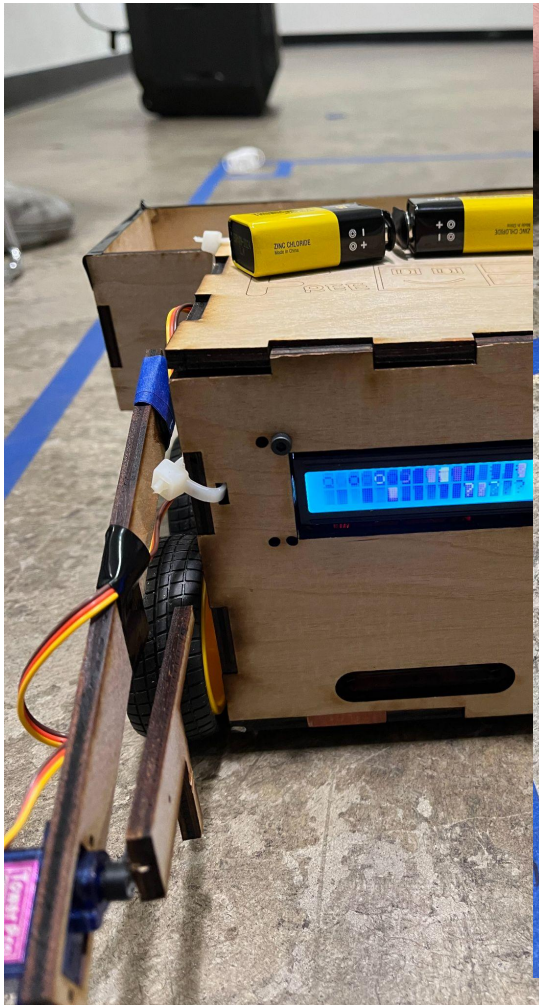


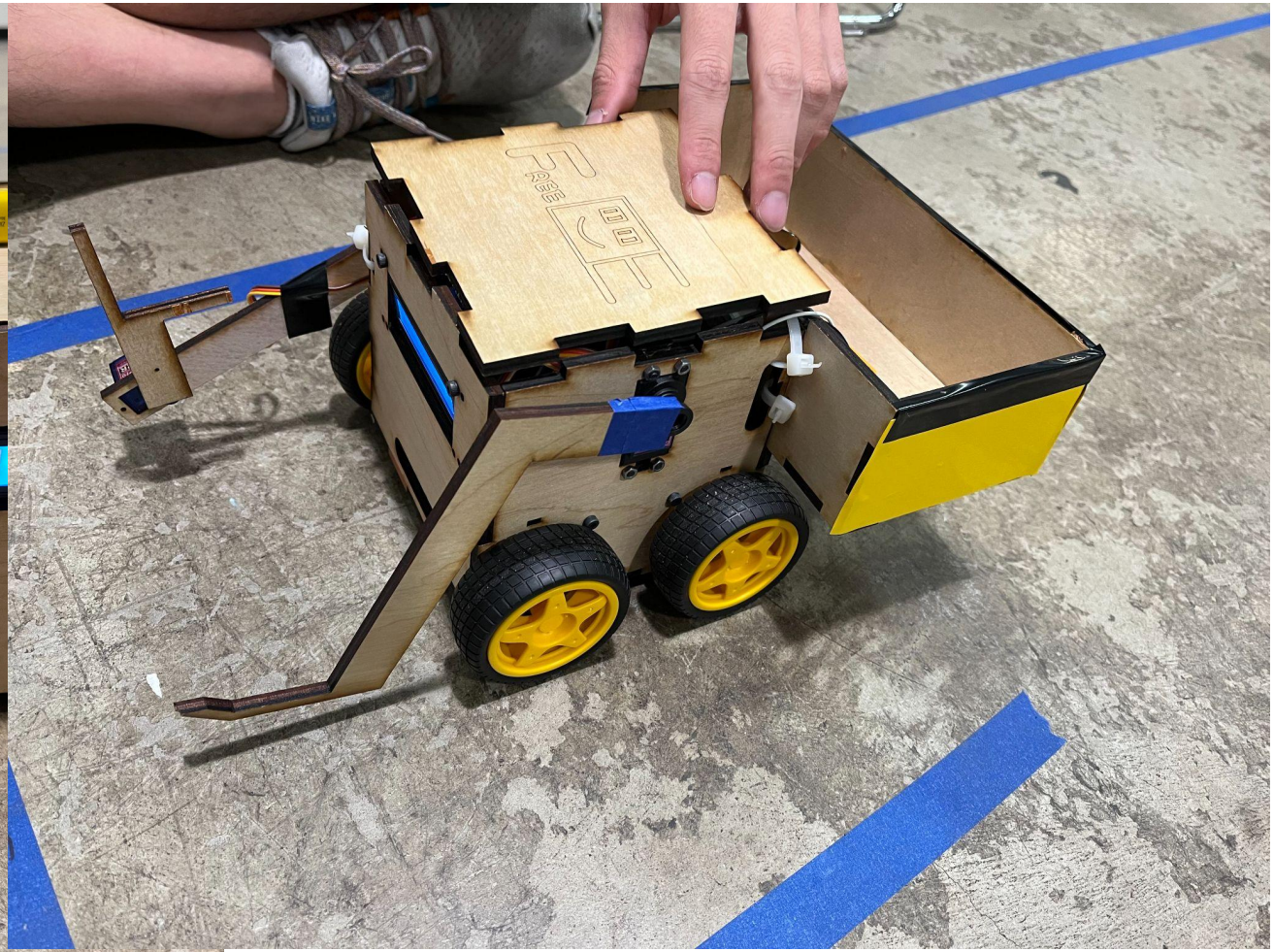


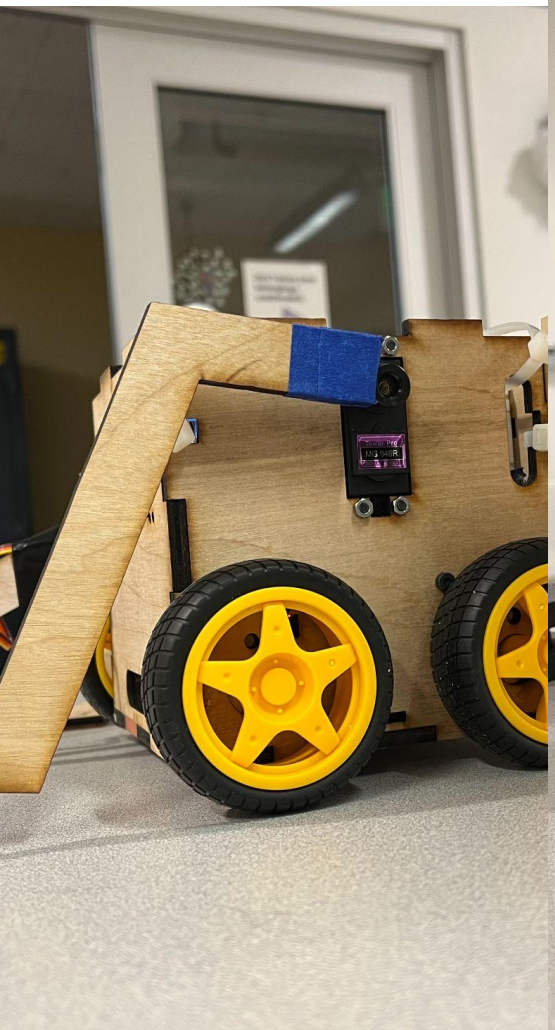
Our Process: Almost Everything Done






ZINC CHLORIDE




MG 946R


MG 946R

Maybe We Should Have Tried...

- We noticed that the electronic devices worked very unpredictably given insufficient voltage / unstable voltage, especially the big servos (they will refuse to work if given below ~4.9V, and we had to power it up using 5V connected to a breadboard)
- Should have checked more datasheets to make sure everything are given their desired voltage.

References:

https://www.quinapalus.com/hd44780udg.html

https://maxpromer.github.io/LCD-Character-Creator/

(Website used to make customized LCD figures)

Task Assigns

- Julia (Online): Solidworks, Arm Design, LCD Creativity
- Phoenix: Data Transmission, Car Assembly, Car Operator
- Jerry: Main Controlling Code, Wiring

For reference

Teams are required to submit their design review presentation by Tuesday, July 26th, at 10 AM. Presentations must be 10 minutes or less and given in recorded video format, where all team members present equally on their design process and demonstrate their finished robot's functionality via photos and videos. Teams are required to use Google Slides for their presentations and embed all photos and videos of the robot within the slides themselves. The guest judges will be evaluating each team's **Robot Functionality and Features** score as well as their **Presentation** score based solely on the content of the team's video presentation, so ensure that you showcase all of your robot's functionality within the presentation itself. Engineering communication practices demonstrated in the presentation, such as clear descriptions of the designs and code used, the iterative design process, and future improvements, will also be considered. For detailed information on presentation criteria, refer to the judging rubric.

Presentations will be submitted in three parts:

1) Video File (MP4)

2) YouTube Link (Instructions on how to upload a video to YouTube can be found here)

3) Link to Your Google Slides

Presentation Submission Form

all teams will create a **design review presentation**, a ten-minute video where they explain their design process from start to finish, showcase relevant CAD models and code snippets, and demonstrate their final robot's performance.