Ghostbusters PKE Meter with Espruino Pico

Table of Contents

1 Introduction

This PKE Meter is based on the „Ghostbusters PKE Meter for Arduino“ from CountDeM0net on Thingiverse (https://www.thingiverse.com/thing:2338494). Thingiverse is also the location where you can download the CAD files for 3D printing (read “Mechanics” prior to printing!). There is a discussion thread in the Fan Forum that has quite some details on how CountDeM0net did build his Arduino variant (https://www.gbfans.com/forum/viewtopic.php?f=5&t=43258).

Changing that to an Espruino Pico was quite a challenge for me, as I do not have much experience with soldering and wanted to avoid soldering the Espruino, thus I used an internal bread board. Also Arduino runs Java, while Espruino uses JavaScript, so I had to translate the code as well.

When you attempt this project, you need access to a workshop with some basic tools (like drills, saws, pliers and screwdrivers) and of course a soldering iron.

2 Parts

2.1 Electronics

- 1 Espruino Pico
- 2 micro buttons (for example from Espruino starter kit)
- 1 buzzer (for example from Espruino starter kit)
- 14 yellow LEDs 3mm (plus some spares)
- 1 micro servo (digital)
- 1 display module 0,96" 4-pin I2C OLED 128X64
- 4 rechargeable batteries AAA 1,2 V
- 2 battery holders for AAA
- 1 switch
- 1 bread board (Steckplatine) 170 tie/pin, 45 x 35 x 10 mm
- 1 soldering board (Lötplatine) 40 x 20 mm
- some normal wire (Schaltdraht) 0,14mm2 in 2+ colours
- some stranded wire (Schaltlitze) 0,08mm2 silicon isolation in 2+ colours

2.2 Mechanics

- 12 3D printed parts from “Ghostbusters PKE Meter for Arduino” (https://www.thingiverse.com/thing:2338494)
  Read “Mechanics” prior printing!
- 4 screws 2,5 x 16 mm
- 5 screws 2,5 x 10 mm (or shorter)
- 1 steel wire 1mm diameter, ~8 cm length
- some plastic glue and hat glue

3 Electronics

3.1 Espruino Links

- Getting Started https://www.espruino.com/Quick+Start+USB
- Web-Interface https://www.espruino.com/ide/
- Examples https://www.espruino.com/Espruino%20Kits
- Compare to Arduino https://www.espruino.com/Arduino%20Differences

3.2 Interfacing with Espruino

3.2.1 Basics

```
reset(); // stops program and clears RAM
save();  // saves RAM state into Flash
USB.setConsole(true); // frees up pins B6 and B7
```

3.2.2 Buttons

When pressed, the buttons connect those two legs, which are closer together. Connect one pin to 3.3 V and send signal from the other pin as input to Espruino.

```
//activate internal resistor to enable correct button read
pinMode(B3, "input_pulldown");
digitalRead(B3);
```

3.2.3 Buzzer

Top of housing and bottom view show which connector is plus.

```
analogWrite(A5, 0.5); // send sound signal
digitalWrite(A5,0);   // switch off sound
```

3.2.4 LEDs

Shorter leg is minus/GND.

```
analogWrite(B15, 1); // Switch LED on
analogWrite(B15, 0); // Switch LED off
```

3.2.5 Servo

Brown = Minus, Red = Plus, Yellow = Signal

Alternative 1: Direct control via pulse length

Every pulse moves the servo towards the position given by the length of the pulse. (0.5 ms most clockwise, 2.5 ms most counter clockwise).

```
digitalPulse(A8, 1, 0.5); // move clockwise
digitalPulse(A8, 1, 2.5); // move counter clockwise
```

Alternative 2: Use Espruino “servo” module

May cause problems if a new signal is send before the last position is achieved.

```
var SERVO = require("servo").connect(A8,{range:2});
var position = 0.5 // value between 0 and 1
SERVO.move(position, 3000); // move to position over 3 seconds
```

3.2.6 Display Module

Espruino has 14 output/input pins that are usable without soldering. Connecting 7 LEDs, 2 buttons, 1 buzzer and 1 servo needs 11 pins. Thus maximum 3 are available for the display. Typical SPI displays need 6 pins, so 4 pins in addition to ground and voltage. Thus I use a 4-pin I2C display which only needs 2 pins in addition to ground and voltage.

The 3D printed parts assume a 0.96 inch sized display, thus I used an 0.96 Zoll 128x64 LCD (White) SSD1306 I2C display module.

I2C is the bus protocol and SSD1306 is the graphic driver, available from the Espruino website (https://www.espruino.com/SSD1306). The “require” function automatically loads the latest version of that driver from the internet when the code is transferred from the web IDE to the board.

```
I2C1.setup({scl:B6,sda:B7}); // I2C Setup
var DISPLAY = require("SSD1306").connect(I2C1);
```

3.3 Voltage

Espruino Pico can use 3.3 V to 16 V.

Independent of input voltage, the board provides 3.3 V at the 3.3 pin, which can be used for the display.

The micro servo runs on 4.8 to 6 V.

Input is either USB (5 V), or 4x AAA battery, either rechargeable (4x 1.2 Volt = 4.8 V) or non-rechargeable (4x 1.5 Volt = 6 Volt).

So no voltage converter is needed.

3.4 Current

With acceptable battery life, each AAA delivers ~160 mA, so 4x AAA are ~640 mA.

Espruino Pico provides ~ 150 mA at the 3.3. pin. The display draws less than 50 mA, thus it can easily be driven from that pin, together with the two buttons.

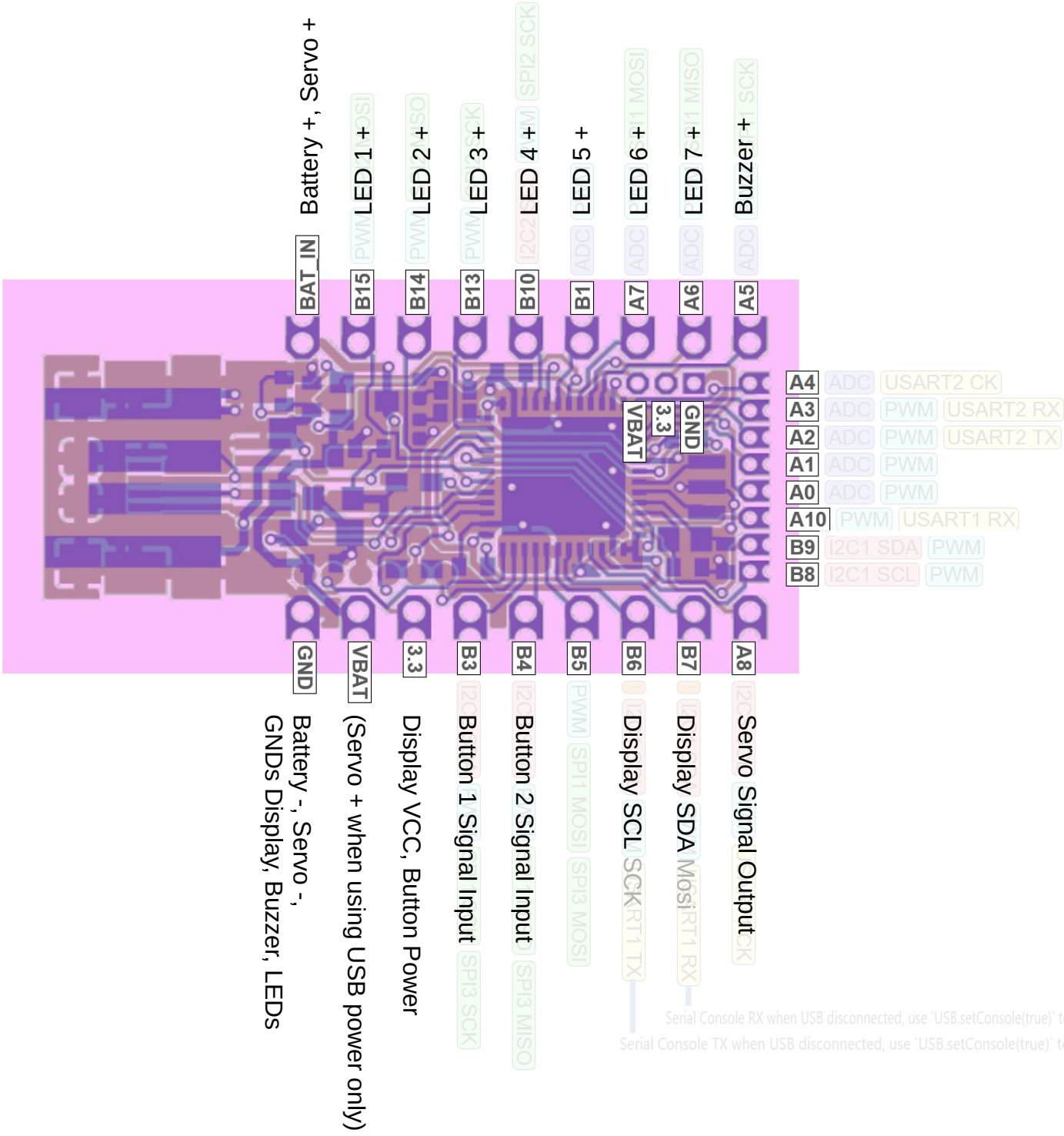
The servo however draws up to 550 mA, thus it should get its power directly from the battery (BAT_IN). However that pin has no power when you drive the Espruino Pico from USB only. You could use the VBAT pin instead, which has either the battery voltage from BAT_IN or the USB voltage. However it is recommended to not draw more than 500 mA from the VBAT pin, thus I did choose the BAT_IN.

That means that when you are working with USB, you either need to have the battery power available in addition or switch the connection from BAT_IN to VBAT. But do not forget to switch back to BAT_IN before you close the PKE Meter.

3.5 Pin Layout

One cool thing of the Espruino Pico is that you can wire everything directly to the pins, without need for any additional resistors.

View from above (looking down onto chip):



Note the comment on Servo+ on BAT_IN or VBAT in section Electronics/Current.

3.6 Code

Basically I converted the code from the “Ghostbusters PKE Meter for Arduino” from Java to JavaScript and did some minor changes, for example added the ability to adjust LED, buzzer and display timing independent of each other and changed the button input from 3 defined stages (low, med, high) to a 0-100 low to high range. In addition I changed the display content from “text+circle+bars” to “bars only”, similar to the display content of the Ghostbusters video game.

As I do not have experience with JavaScript, my code may not be optimal. It does work. However if you have ideas on how to improve it, please let me know.

```
// ------------------- Variables
// Map functions to pins
var BUTTON1 = B3; // left button
var BUTTON2 = B4; // right button
var BUZZER = A5;
var DISPLAYscl = B6;
var DISPLAYsda = B7;
var LED1 = B15;
var LED2 = B14;
var LED3 = B13;
var LED4 = B10;
var LED5 = B1;
var LED6 = A7;
var LED7 = A6;
var SERVO = A8;

// Counting variables
var button1status = 0;  //button1 depressed
var button2status = 0;  //button2 depressed
var buzzerSpeed = 0;    // delay for buzzer signal
var buzzerMillis = 0;   // store last time buzzer was sounded
var currentMillis = 0;  // time since startup in Milliseconds
var displayMillis = 0;  // store last time display was updated
var ectoLevel = 0;      // alert state between 0 (low) and 100 (high)
var ectoLevelOld = 0;   // last ecoLevel
var ledSpeed = 0;       // delay for LED walk
var ledMillis = 0;      // store last time LED was updated
var ledNum = 0;         // current LED that is lit
var servoPos = 0;       // pulse width for servo position

// Settings
var buzzerMin = 200;    // smallest buzzer delay in milliseconds
var buzzerMax = 1000;   // largest buzzer deplay in milliseconds
var ledBright = 1;      // brightness of LED from 0 to 1
var ledMin = 20;        // smallest led delay in milliseconds
var ledMax = 500;       // largest led delay in milliseconds
var servoMax = 1.85;    // servo position max (servo physical max 2.5)
var servoMin = 1.00;    // servo position min (servo physical min 0.5)

// Initialise
I2C1.setup({scl:DISPLAYscl,sda:DISPLAYsda}); // I2C pin Setup
var DISPLAY = require("SSD1306").connect(I2C1); // load "SSD1306" graphics
driver and set pins as stated in I2C1
```

```
// ------------------- Functions

function setup() {
     // free up pins B6 and B7
    USB.setConsole(true);

     // input button setup: activate internal resistor to enable correct
button read
    pinMode(BUTTON1, "input_pulldown");
    pinMode(BUTTON2, "input_pulldown");

     // define display font size
    DISPLAY.setFontVector(18);
}

function BuzzerStuff(inputVal) {
  buzzerSpeed = (100-inputVal)/100*(buzzerMax-buzzerMin)+buzzerMin; //
calculate buzzerSpeed from ectoLevel
  currentMillis = Date.now(); // reads milliseconds since startup

  if ((currentMillis -buzzerMillis >= buzzerSpeed)) { // check to see if it's
time to change the state
      buzzerMillis = currentMillis;
      analogWrite(BUZZER, 0.5); // send sound signal
      setTimeout(function(){    // wait for ...
      digitalWrite(BUZZER,0);   // switch off sound
        }, 100);                // ...100 milli seconds
    }
  }

function DisplayStuff(inputVal) {
  currentMillis = Date.now(); // reads milliseconds since startup

  if ((currentMillis -displayMillis >= 200)) { // check to see if it's time
to change the state
    displayMillis = currentMillis;

    let b1 = Math.random()*10; // random add to bar height
    let b2 = Math.random()*10;
    let b3 = Math.random()*7;
    let b4 = Math.random()*5;
    let b5 = Math.random()*5;
    let b6 = Math.random()*10;
    let b7 = Math.random()*7;
    let b8 = Math.random()*5;
    let b9 = Math.random()*5;

    DISPLAY.clear();
    // DISPLAY.drawString("Ghostbusters",5,55);
    DISPLAY.fillRect(58, inputVal*0.45+b1+5, 70, 0); // Center bar

    DISPLAY.fillRect(44, inputVal*0.33+b2+3, 56, 0); // Left bars
    DISPLAY.fillRect(30, inputVal*0.22+b3,   42, 0);
    DISPLAY.fillRect(16, inputVal*0.16+b4,   28, 0);
    DISPLAY.fillRect( 2, inputVal*0.05+b5,   14, 0);

    DISPLAY.fillRect( 72, inputVal*0.33+b6+3, 84, 0); // Right bars
    DISPLAY.fillRect( 86, inputVal*0.22+b7,   98, 0);
    DISPLAY.fillRect(100, inputVal*0.16+b8,  112, 0);
```

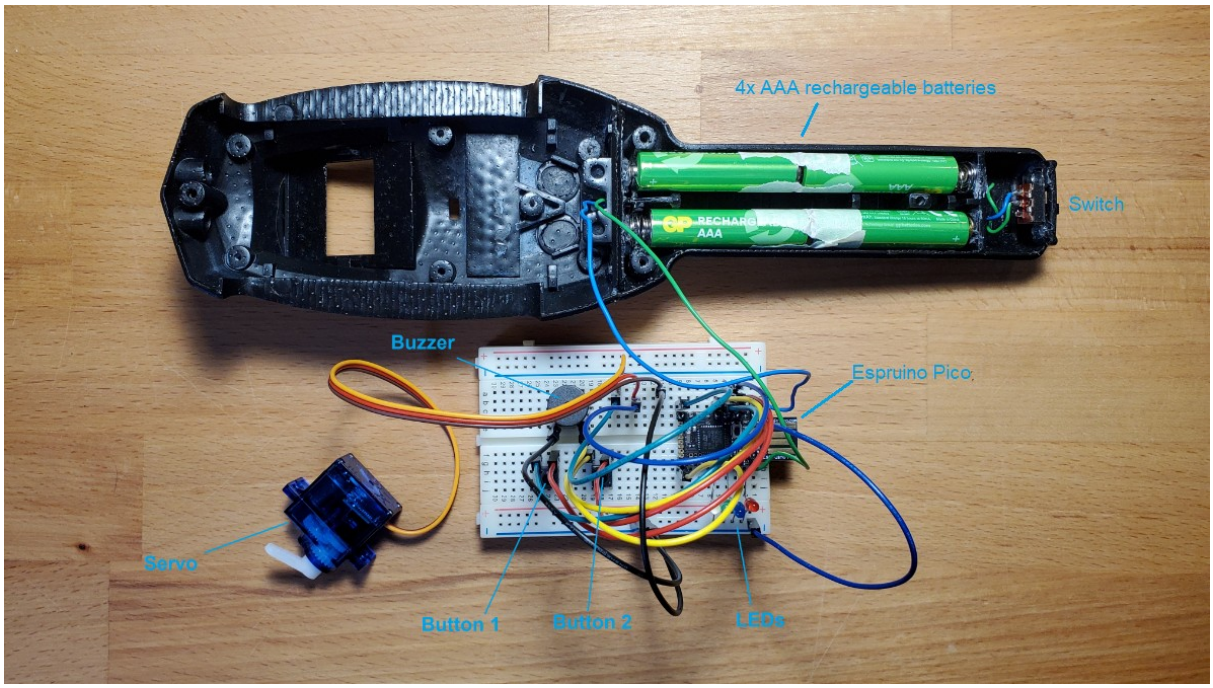
```
    DISPLAY.fillRect(114, inputVal*0.05+b9,  126, 0);

    DISPLAY.flip(); // write to the screen
    }
  }

function LedStuff(inputVal) {
  ledSpeed = (100-inputVal)/100*(ledMax-ledMin)+ledMin; // calculate ledSpeed
from ectoLevel
  currentMillis = Date.now(); // reads milliseconds since startup

  if ((currentMillis - ledMillis >= ledSpeed)) { // check to see if it's time
to change the state
    ledMillis = currentMillis;

  if ( ledNum == 0 ) {
      digitalWrite(LED1, ledBright);
      ledNum = 1;
    } else if ( ledNum == 1 ) {
      digitalWrite(LED1, 0);
      digitalWrite(LED2, ledBright);
      ledNum = 2;
    } else if ( ledNum == 2 ) {
      digitalWrite(LED2, 0);
      digitalWrite(LED3, ledBright);
      ledNum = 3;
    } else if ( ledNum == 3 ) {
      digitalWrite(LED3, 0);
      digitalWrite(LED4, ledBright);
      ledNum = 4;
    } else if ( ledNum == 4 ) {
      digitalWrite(LED4, 0);
      digitalWrite(LED5, ledBright);
      ledNum = 5;
    } else if ( ledNum == 5 ) {
      digitalWrite(LED5, 0);
      digitalWrite(LED6, ledBright);
      ledNum = 6;
    } else if ( ledNum == 6 ) {
      digitalWrite(LED6, 0);
      digitalWrite(LED7, ledBright);
      ledNum = 7;
    } else if ( ledNum == 7 ) {
      digitalWrite(LED7, 0);
      digitalWrite(LED1, ledBright);
      ledNum = 1;
    }
  }
}

function ServoStuff(inputVal) {
  servoPos = servoMax+0.02 - ((servoMax-servoMin) * (inputVal+1) /100); //
convert input to delta between min and max

  if( ectoLevel != ectoLevelOld){
    digitalPulse(SERVO, 1, servoPos); // position defined by pulse width from
0.5 to 2.5ms
    ectoLevelOld = ectoLevel;
    // console.log("servoPos "+servoPos);
  }
```

```
}

function loop() {
  button1status = digitalRead(BUTTON1); // 1 if depressed, 0 if not
  button2status = digitalRead(BUTTON2); // 1 if depressed, 0 if not

  if( button1status == 0 && button2status == 1 && ectoLevel<100){
    ectoLevel = ectoLevel+1;
    } else {
  if( button1status == 1 && button2status == 0 && ectoLevel>0){
    ectoLevel = ectoLevel-1;
     } else {}
  }

  BuzzerStuff(ectoLevel);
  DisplayStuff(ectoLevel);
  LedStuff(ectoLevel);
  ServoStuff(ectoLevel);
}

// ------------------- Program

setup();

setInterval(function() {
  loop();
}, 10); // repeat function (loop) with a delay of 10 milliseconds
```

3.7 Testing

Before integrating the electronics, I recommend testing the setup on a bread board. Make sure that you use the web IDE to load the code into flash memory (not only RAM), so it is available after power loss.

I did not have enough jumper cables to build the complete setup, so I did one test for servo, buzzer, buttons and 4 LEDs and a separate one for the display.



4 Mechanics

4.1 3D Printed Part Modifications

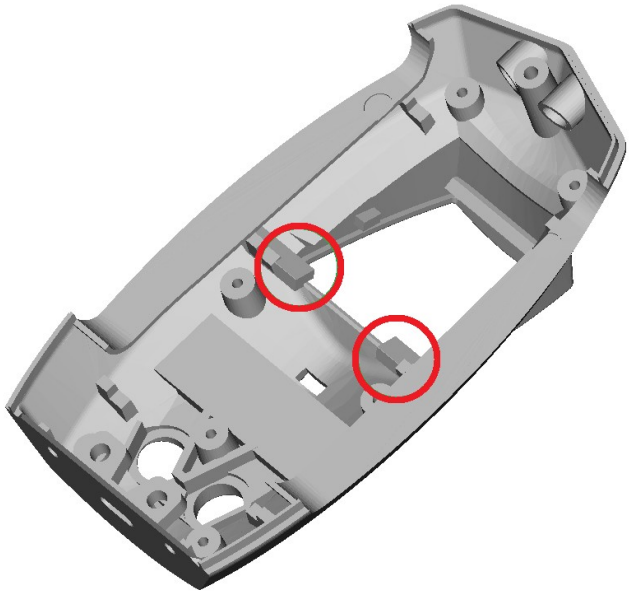
The 3D parts from “Ghostbuster PKE Meter for Arduino” (https://www.thingiverse.com/thing:2338494) need some modification to work with Espruino Pico. If you are capable of CAD work, you can do those changes prior printing. Otherwise you have to do them with drill, file and glue later.

I used black resin for the main parts and grey resin for the wings, which I spray painted silver later (but prior to integration).

4.1.1 Front Top

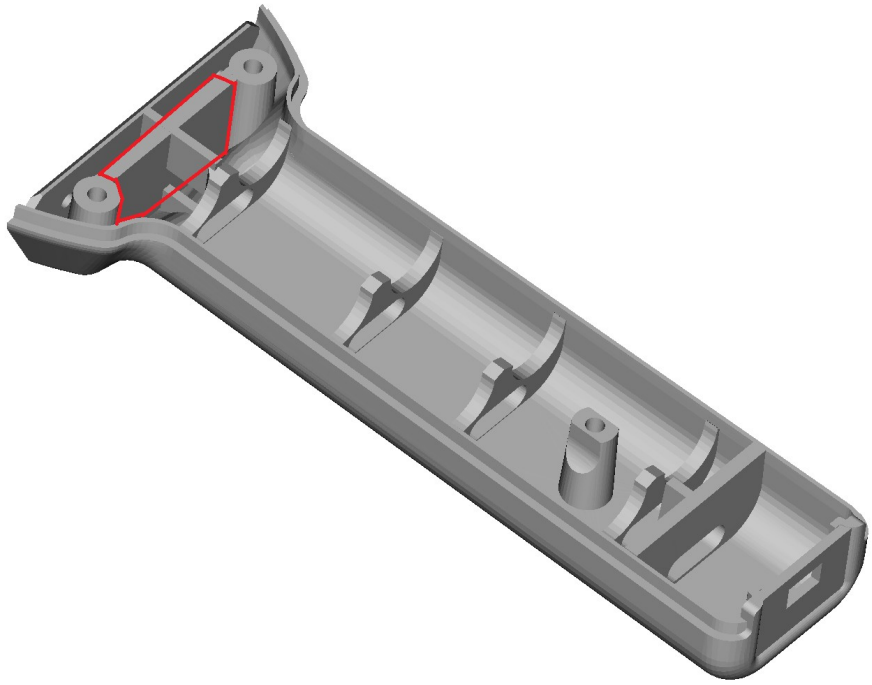
As the centre hole and the two at the top are not used, you could close them.

As I was not able to wiggle the screen holder into the guides, so I had to break away the lower ones. The screen holder did also hold nicely without them.



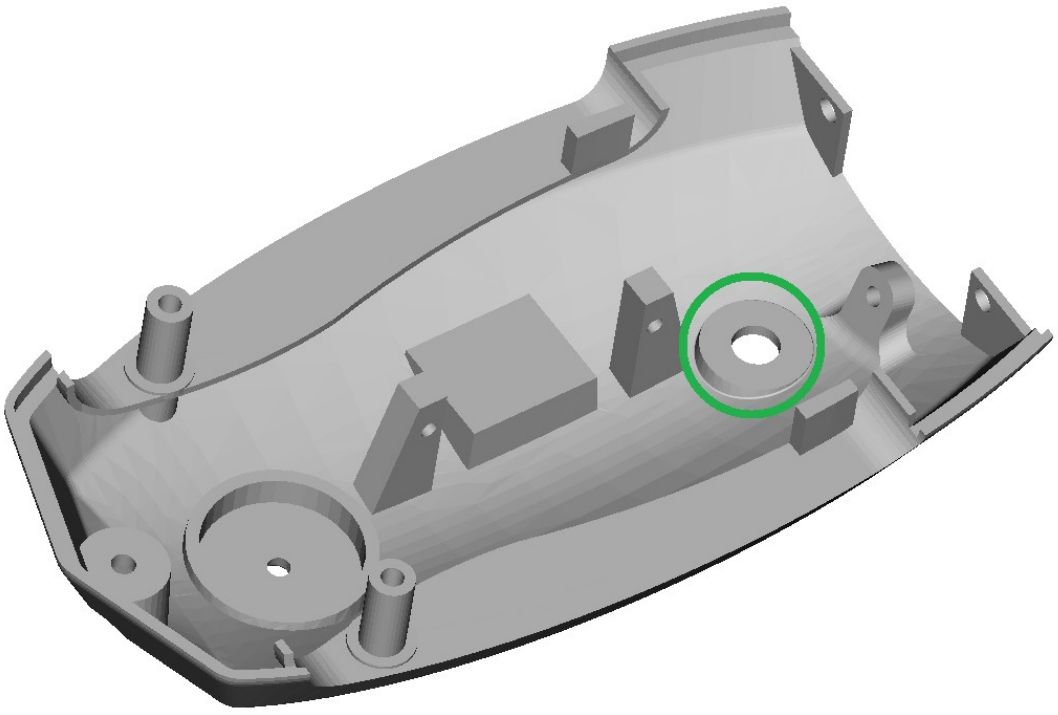
4.1.2 Front Bottom

There is a barrier which prevents wire routing to the top part and also blocks positioning of the AA batteries with contactor and springs. I had to remove most of that wall.

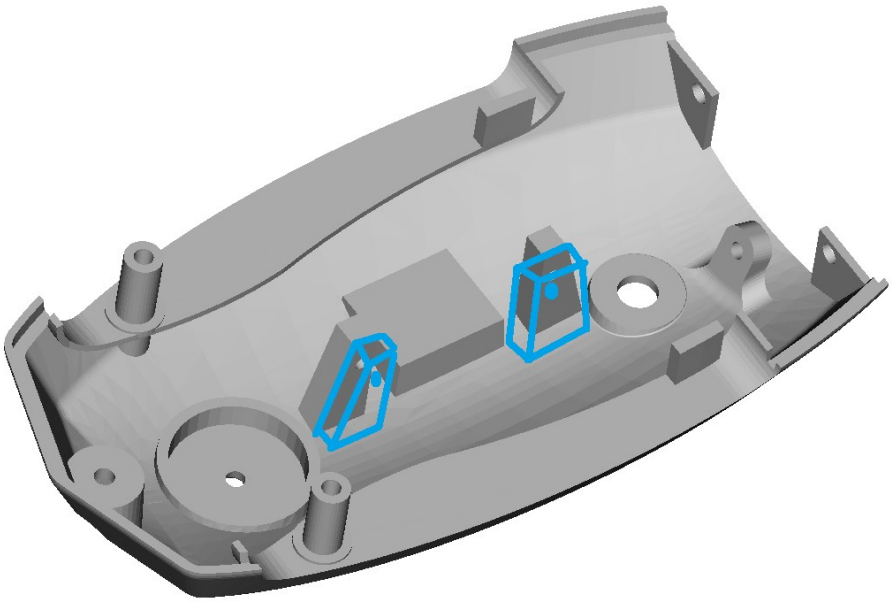


4.1.3 Back Top

As the centre hole is not used, you could close it.



Depending on the size of your servo, you may need to change the spacing and position of the fixing. In my case, I had to move the servo 5mm outboard and the fixings about 3mm closer together.



4.2 Assembly

Glue “front top” to “front bottom” and “back top” to “back bottom” using plastic glue. You can fix them with small screws while curing, but those should be removed again once the glue has bonded.

Insert the “screen holder” into “front top”.

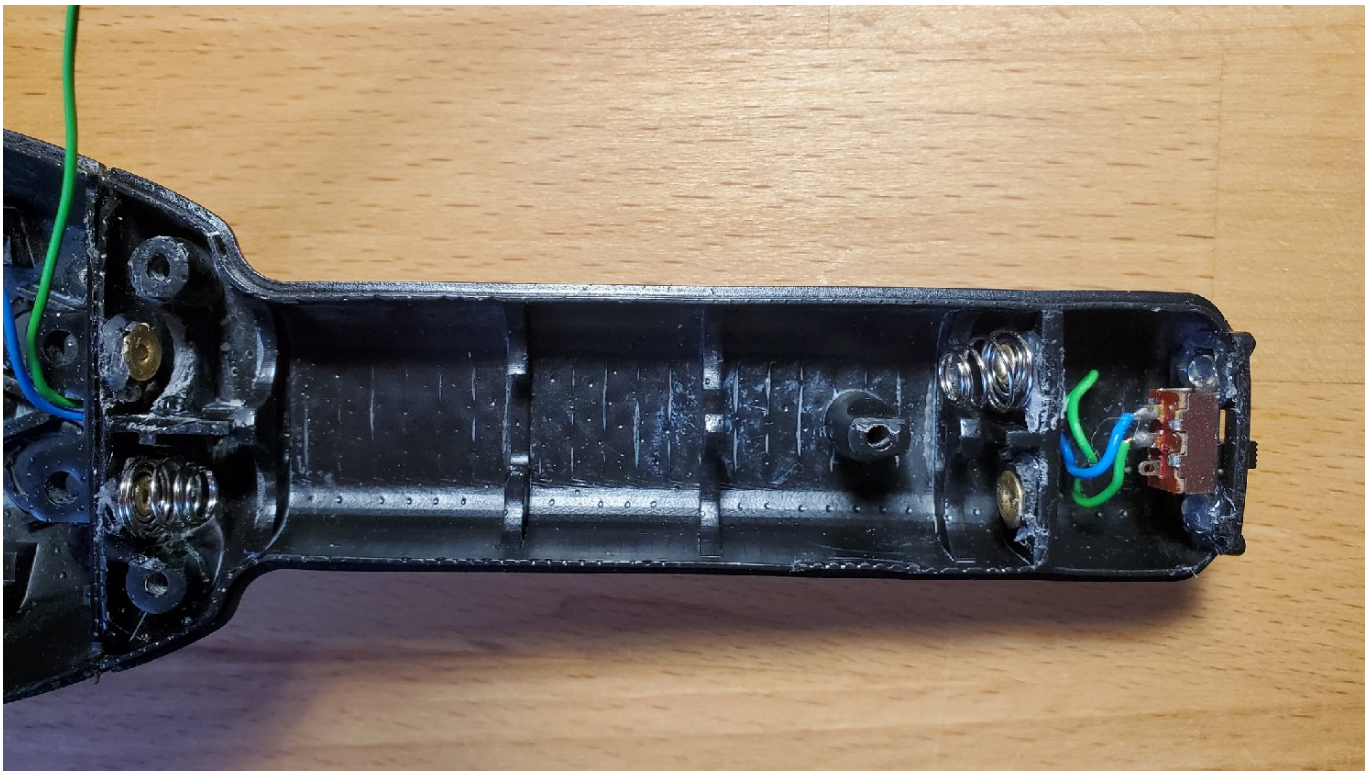
The wings are later assembled onto the pins in “back top”.

5 Integration

5.1 Battery Compartment

Cut off the contact plate and the spring from the two AAA battery holders. Solder on wires and glue contacts into the “front bottom” part. Glue a small switch into the opening at the rear and connect it between the two strands of batteries.

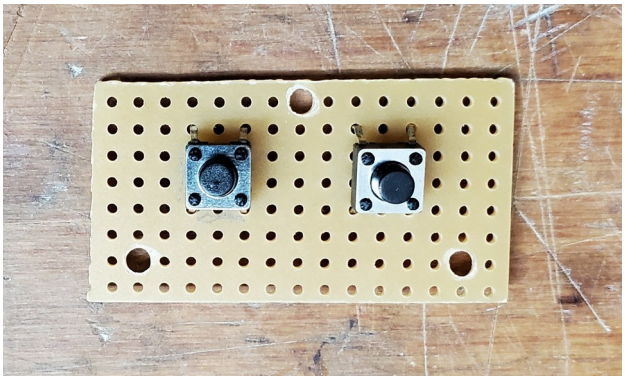
As I did not have black wire, I used blue for Minus.

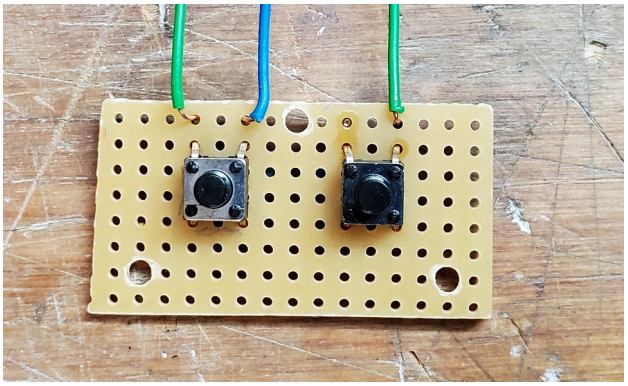


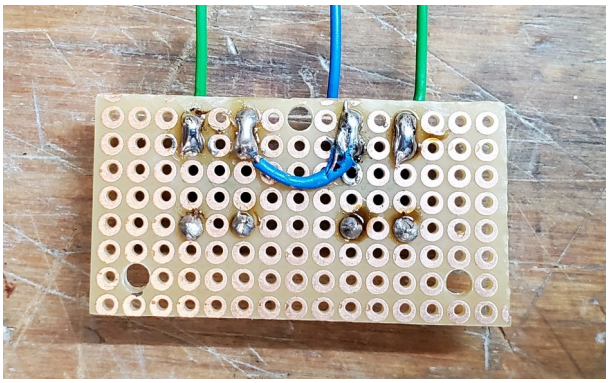
5.2 Buttons

Cut a 40 x 20 millimetre piece from a soldering board. Place in “front top” part, mark the screw position and drill three 2,5 mm holes. Place again and mark the button position through the openings in the part. Place buttons as marked.

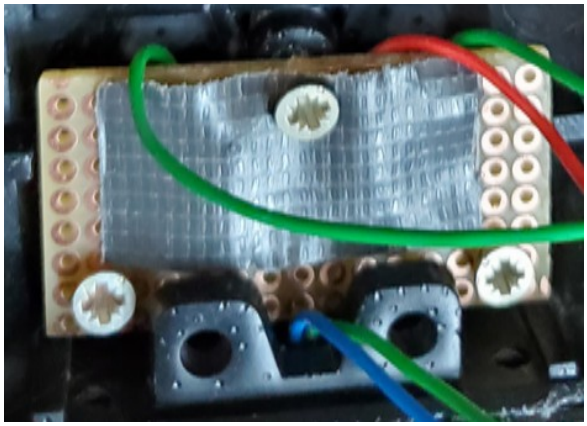
Solder buttons to board, connect one power wire (as I did not have red wire at that time, I used blue and later changed to red) and one signal wire each.





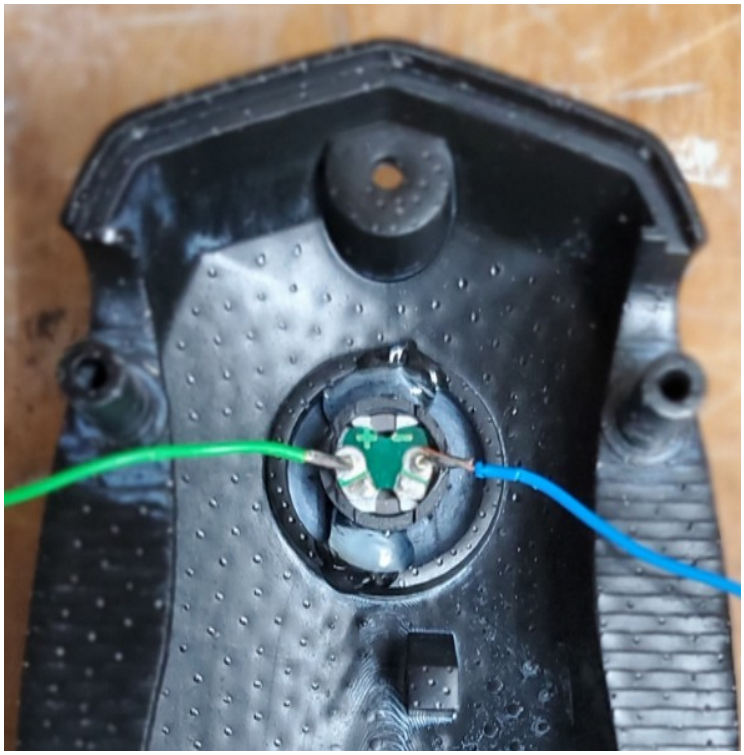


Place the 3D printed buttons, attach the board with three 2,5 x 10 mm screws and cover its rear with a bit of tape.



5.3 Buzzer

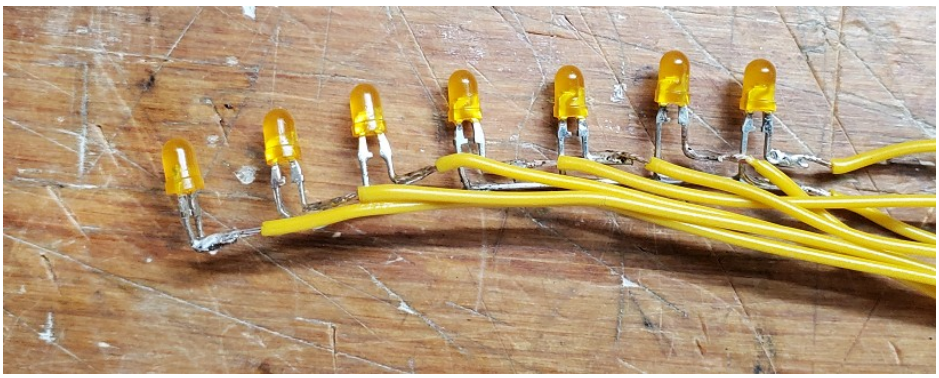
Solder on plus and minus wires to the buzzer and glue it into the round depression in the “back top” part.



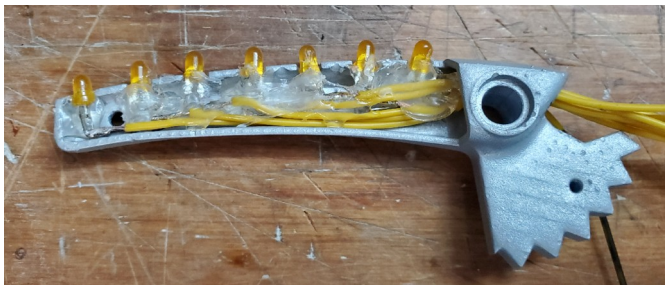
5.4 LEDs and Wings

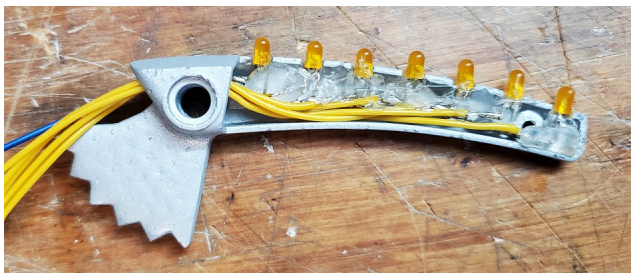
As proposed by CountDeM0net in the info for the Arduino variant, I soldered together the minus pins (shorter legs) of seven LEDs to a strip. Make sure to check the distance, so the strip later fits into the wing.

Next solder on the one minus and 7 signal wires. Make sure you use the flexible and thinner stranded wire, otherwise the servo will not be able to bend the 16 wires coming from the wings.



Place into the wing and fix with some glue. Repeat for the other side.



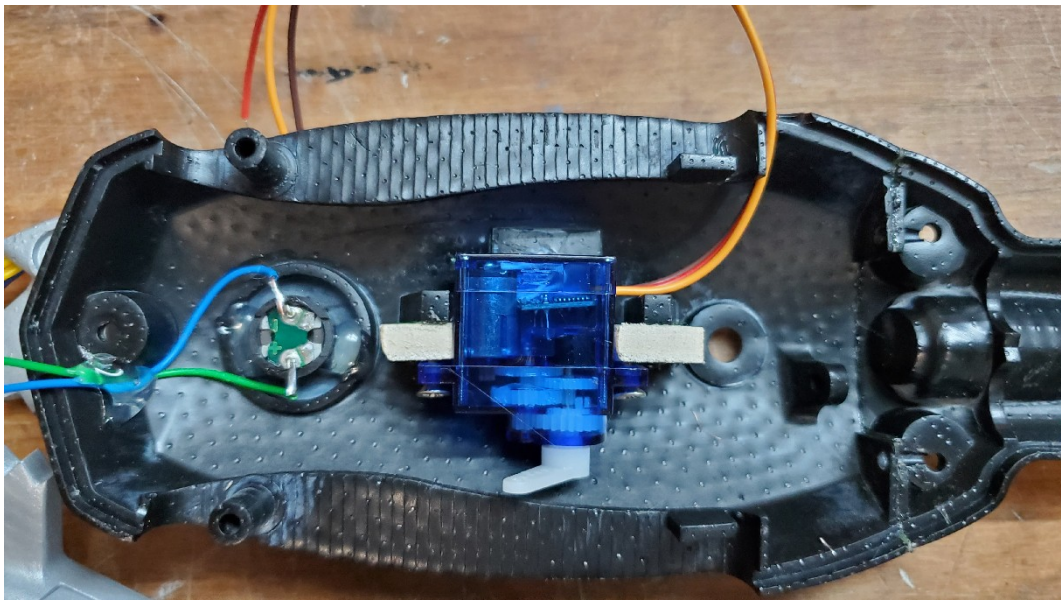


Close the wings with the covers and attach each with one 2,5 x 10 mm screw each. The 10mm screws still seemed a bit too long, but I could not get shorter ones, so I cut off the first ~2 mm of the screws prior to using them.

5.5 Servo

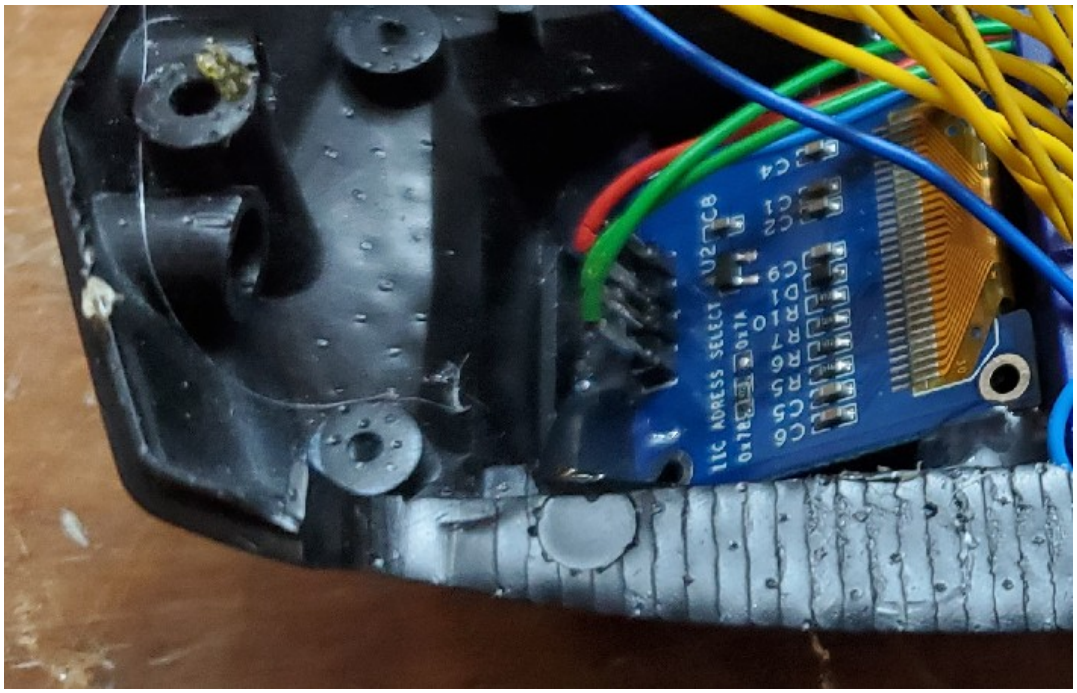
Place a single lever arm top onto the servo and attach it with the small screw provided with the servo.

If you have done the changes in the 3D CAD, you hopefully can screw in the servo directly, using the screws provided with the servo. I had to add two 5 mm thick wood pieces to place the servo far enough from the bread board (on the other half of the PKE meter) and also to match its screw pattern.



5.6 Display

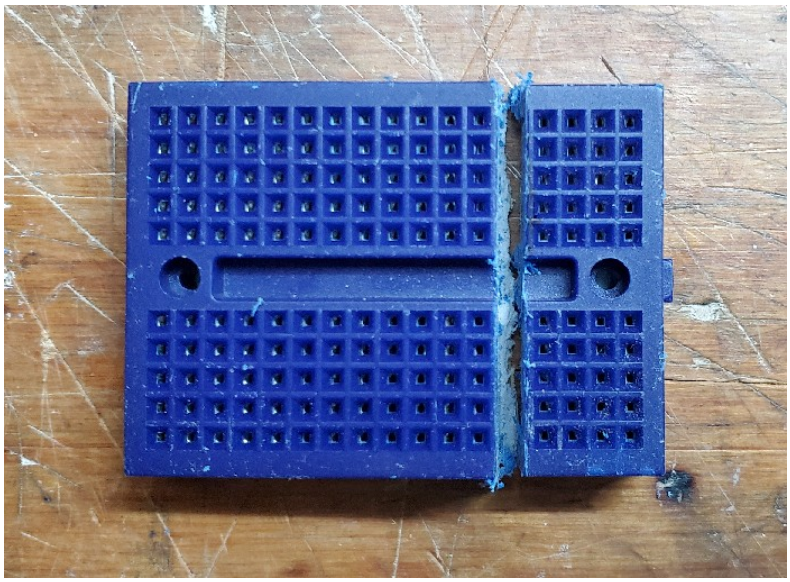
Solder on the four wires to the display pins. Place the display in the opening and attach it with a bit of glue.



5.7 Bread Board

Cut off a small section of the bread board such that you have 12 usable pin rows remaining. This is 9 pin rows for Espruino Pico plus 3 pin rows which I used to provide the many needed grounds without need to solder wires.

Glue it into the “front top” part, right above the board holding the buttons.

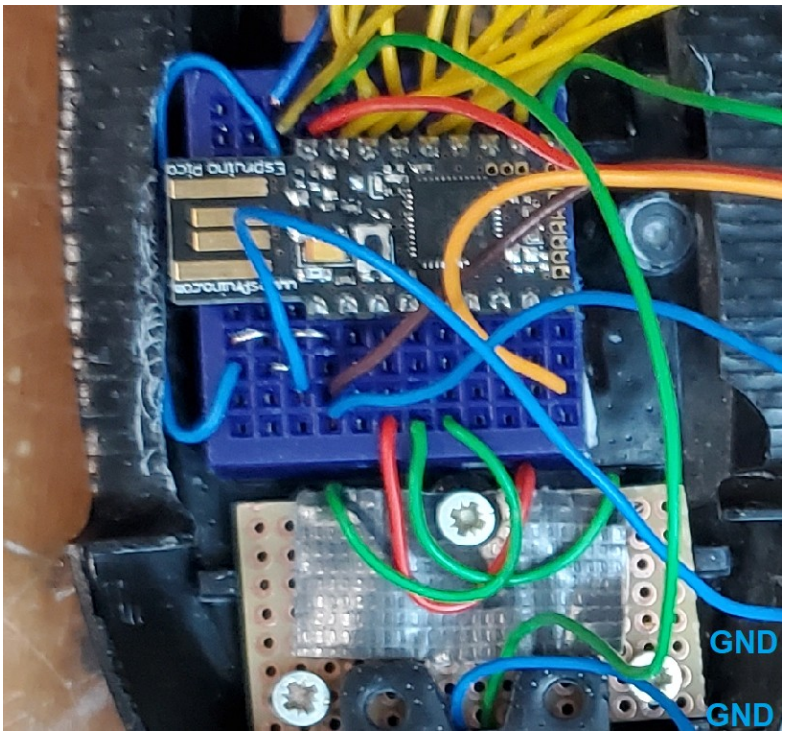


6 Putting It All Together

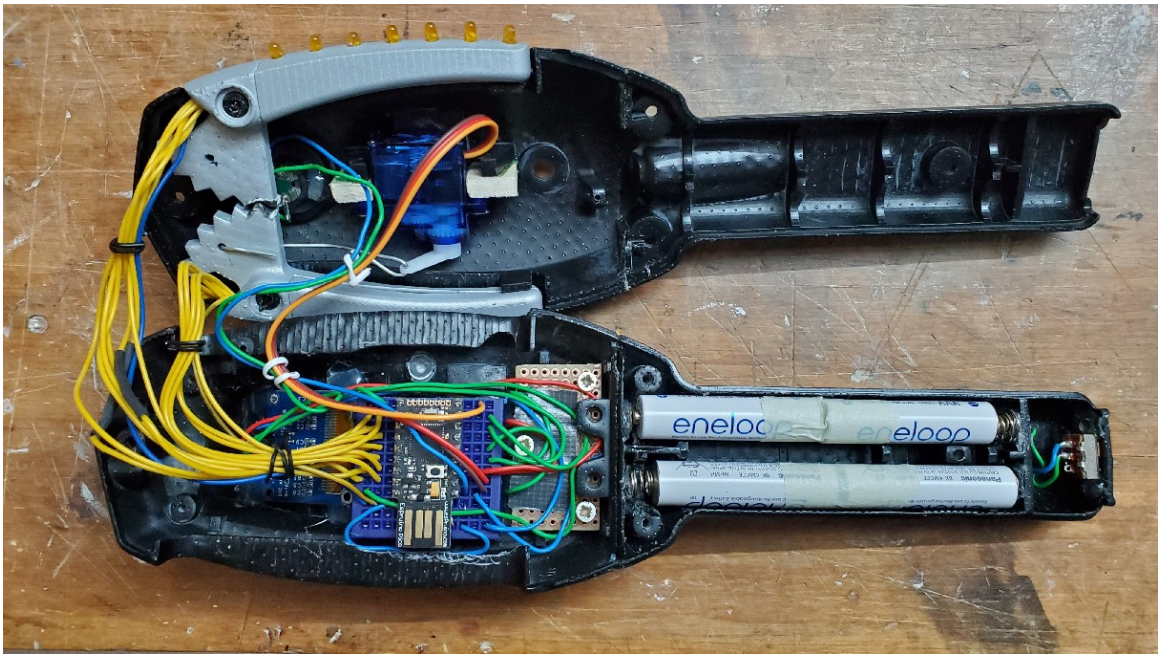
Place the Espruino Pico into the bread board, aligned to the right side.

Place each wire into the matching pin line, see Electronics – Pin Layout.

Note how I used bridges in the left most rows, to connect GND to enough pins to have spaces for each GND wire. I also have one wire taking GND to the upper side, so I did not have to cross the Espruino with the GND wires coming from top.



Connect the top of the servo lever arm to the hole in the left wing via the wire.



In this state you can still connect the Espruino Pico to USB and test all functions and update the code as necessary.

You may need to adjust the min and max servo positions to match your wing positions by adjusting the matching variables in the code.

```
var servoMax = 1.85;    // servo position max (servo physical max 2.5)
var servoMin = 1.00;    // servo position min (servo physical min 0.5)
```

When folding the two halves together, take care to make sure no wires are in the way of the servo movement space. Close it with the four 2,5 x 16 mm screws.

Done!