kerbal-control

Sean Donnellan, Oscar Donnellan, Fynn Donnellan

Table of Contents

1. About this document

HTML version

- https://donnels.github.io/kerbal-control/README.html

PDF version

- https://donnels.github.io/kerbal-control/README.pdf

Github source

- https://github.com/donnels/kerbal-control

2. Starting point

A project to add some more control to the PS5 Kerbal simulation.

Figure 1. Screenshot from www.kerbalspaceprogram.com

- https://www.kerbalspaceprogram.com/

After installing Kerbal on the PS5 and playing it for a while it quickly became clear that the controllers alone are not enough to enjoy the game. The search began to find some way of improving things. Kerbal allows control via the PS5 controllers AND via keyboard and mouse. This means it should be possible to add some easy toggles/ switches/ rotary encoders to pass information to the game. The first switch would be the space bar which kerbal uses to trigger stages.

While this is being developed for Kerbal it is generally a keyboard and can just as easily be used for many other applications, including video conferences.

2.1. kerbal key bindings

The following is a list of key bindings we can work with.

Key bindings for Kerbal

- https://wiki.kerbalspaceprogram.com/wiki/Key_bindings

The first key will be the space key to launch rockets.

2.2. Arduino - pro micro (5V)

Initial choice for a prototype is the AVR ATmega32u4 8-bit microcontroller which has a USB controller and can be used as both a keyboard and mouse if required.

> The Pro Micro is an Arduino-compatible microcontroller board developed under an open hardware license by Sparkfun. Clones of the Pro Micro are often used as a lower-cost alternative to a Teensy 2.0 as a basis for a DIY keyboard controller/converter when a lower number of pins would suffice.
>
> — https://deskthority.net/wiki/Arduino_Pro_Micro

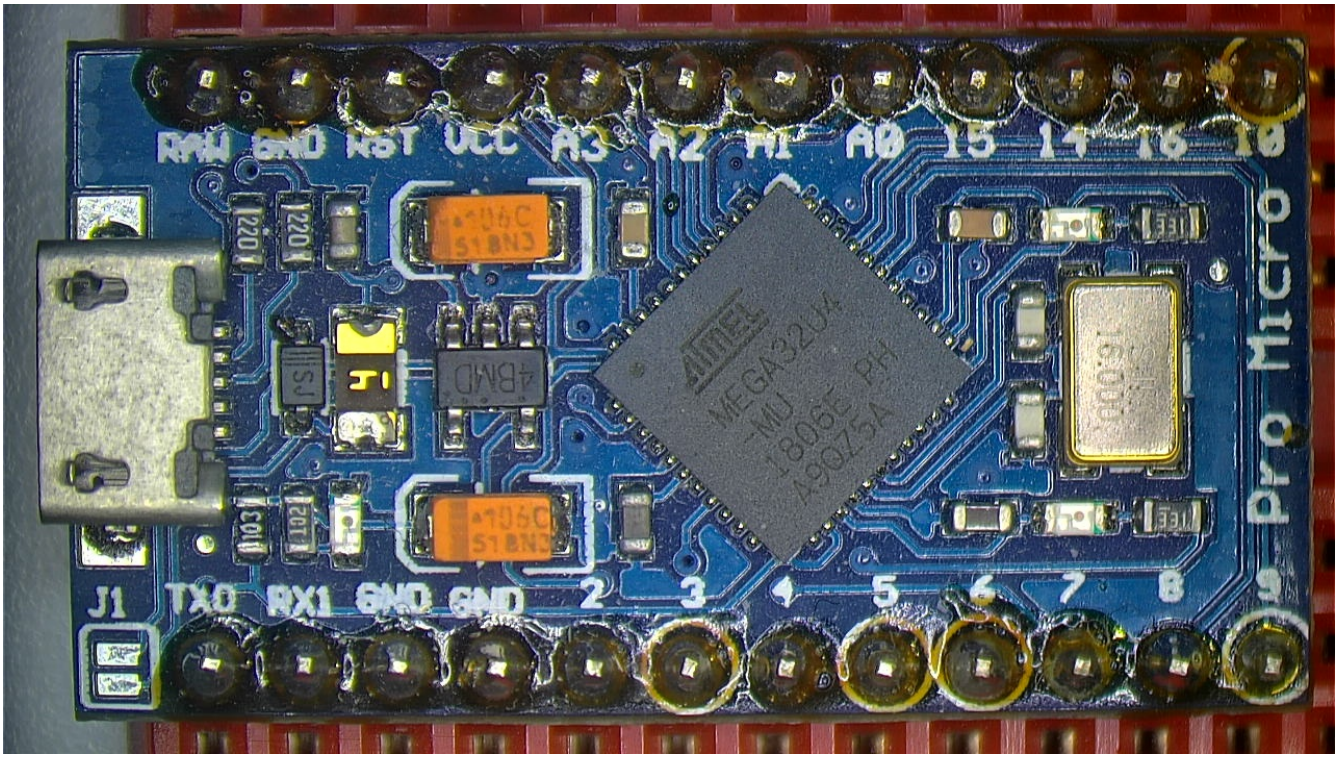

Figure 2. Pro micro 5v board

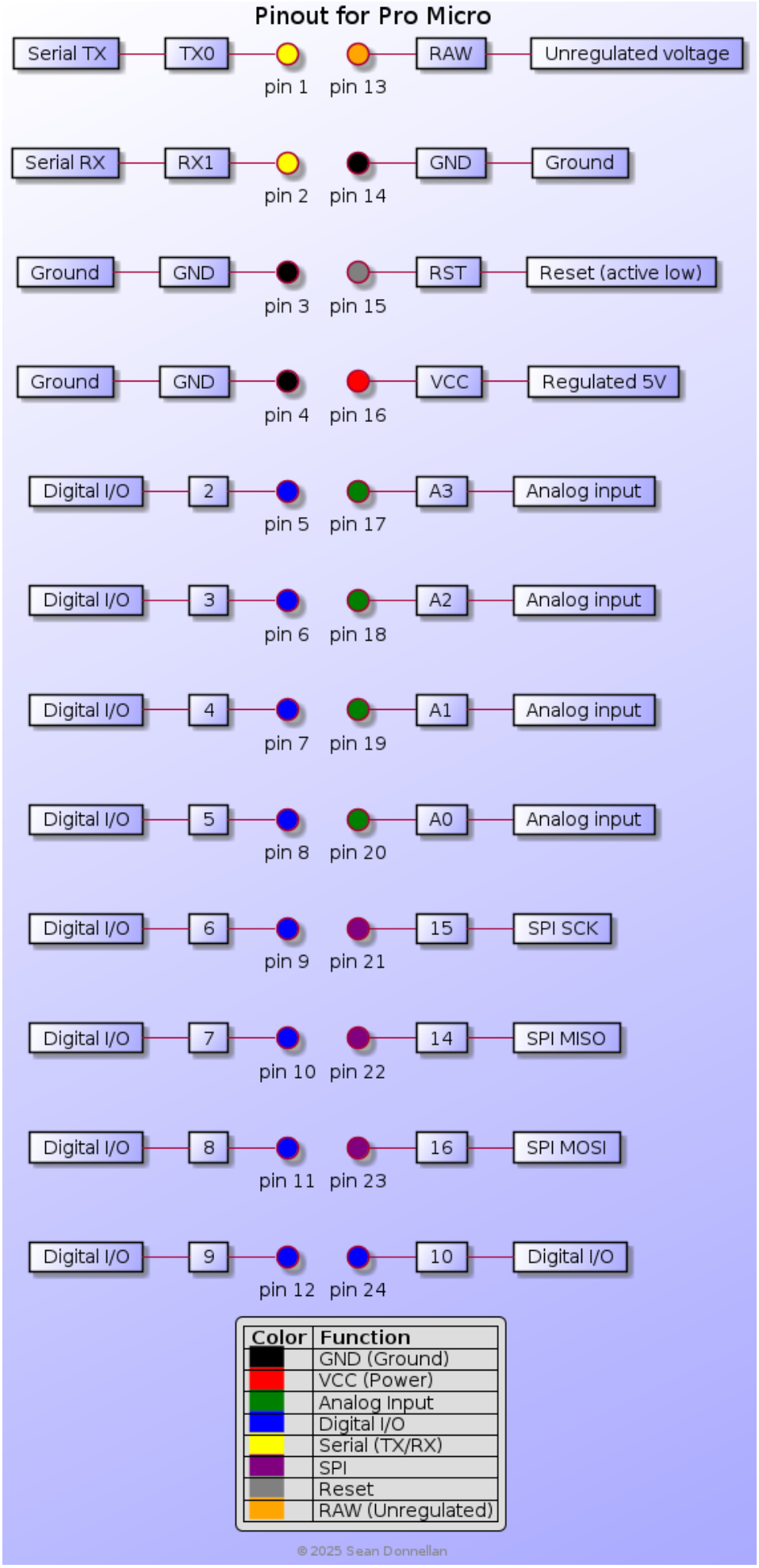

Figure 3. Pro Micro Pinout

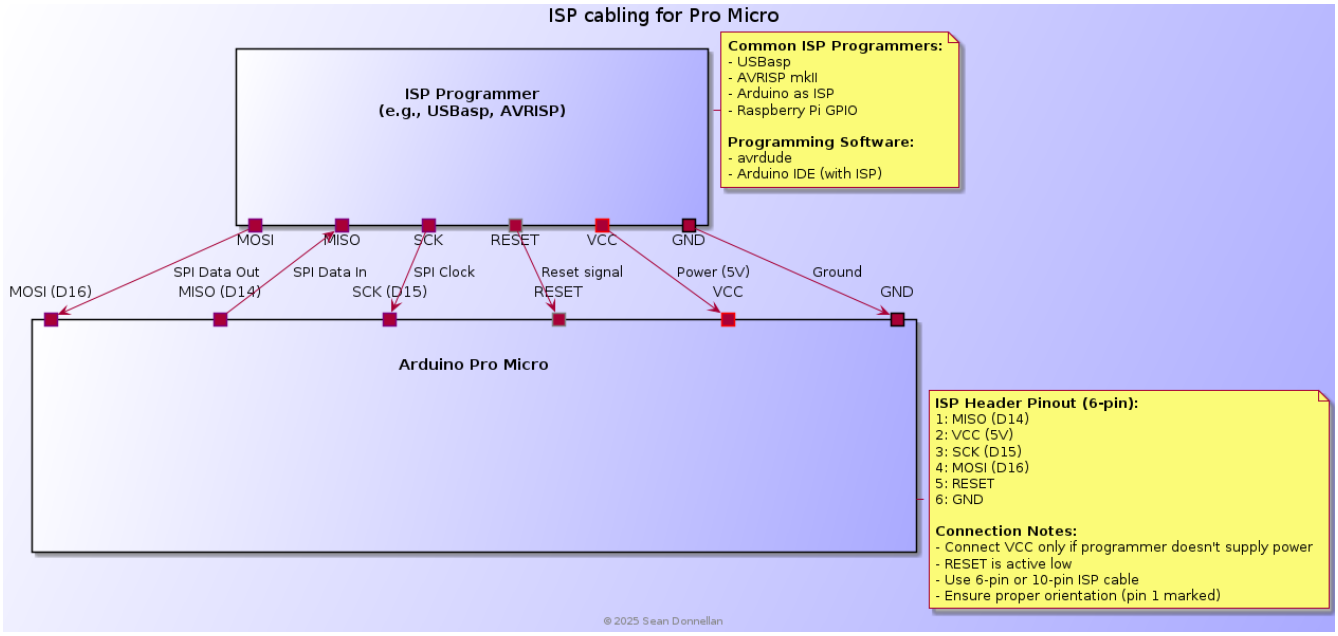

Figure 4. Pro Micro ISP connections

The above figure shows the connections required to program the pro micro via ISP if the usb programming is failing.

During ISP programming, the Pro Micro board requires external power. Connect the board to a USB power source (computer, wall adapter, etc.) as the Raspberry Pi Zero does not provide sufficient power through the ISP programmer alone. Without external power, you may see "device signature = 0x000000" errors.

When you have one or two different ones floating around it’s easy to forget which is which. Measure the voltage between GND and VCC, with USB powered up, to be sure you have the 5V version. Don’t let the magic smoke out by connecting the wrong voltage.

2.3. WS2812 LEDs

Because it’s always good to have status LEDs and the WS2812 is both easy to get and has good libraries, with fastled and adafruit, it’s the first choice.

Fastled

- https://github.com/FastLED/FastLED
- https://fastled.io/

The fastled library looks like a good choice. The WS2812 LED rings I have are not tightly packed with LEDs but will suffice. As I have a few WS2812 strips and a few WS2812 rings the choice went towards the WS2812 to be able to mix the strip and ring. In a first test I had accidentally used an RGBW ring which showed up that the fastled library can’t handle them well so WS2812 is the choice now.

3. Step 1 - First prototype

To get started I went to an example for LEDs to be able to later set a LED with a key/button.

First LED example (arduino IDE)

```
#include <FastLED.h>
#define NUM_LEDS 22
#define NUM_RING_LEDS 12
#define DATA_PIN 7

CRGB leds[NUM_LEDS];

void setup() {
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
    leds[12] = CHSV(0, 255, 16);
    leds[13] = CHSV(33, 255, 16);
    leds[14] = CHSV(65, 255, 16);
    leds[15] = CHSV(97, 255, 16);
    leds[16] = CHSV(129, 255, 16);
    leds[17] = CHSV(161, 255, 16);
    leds[18] = CHSV(193, 255, 16);
    leds[19] = CHSV(225, 255, 16);
    leds[20] = CHSV(255, 255, 16);
    leds[21] = CHSV(255, 255, 16);
    FastLED.show();
}

void loop() {
    for(int dot = 0; dot < NUM_RING_LEDS; dot++) {
        leds[dot] = CHSV(64, 255, 16);
        FastLED.show();
        // clear this led for the next time around the loop
        leds[dot] = CRGB::Black;
        delay(150);
    }
}
```

The first test looks good and was tested on a 5v pro micro. The prototype uses a strip with ten LEDs attached to a ring with 12 LEDs.

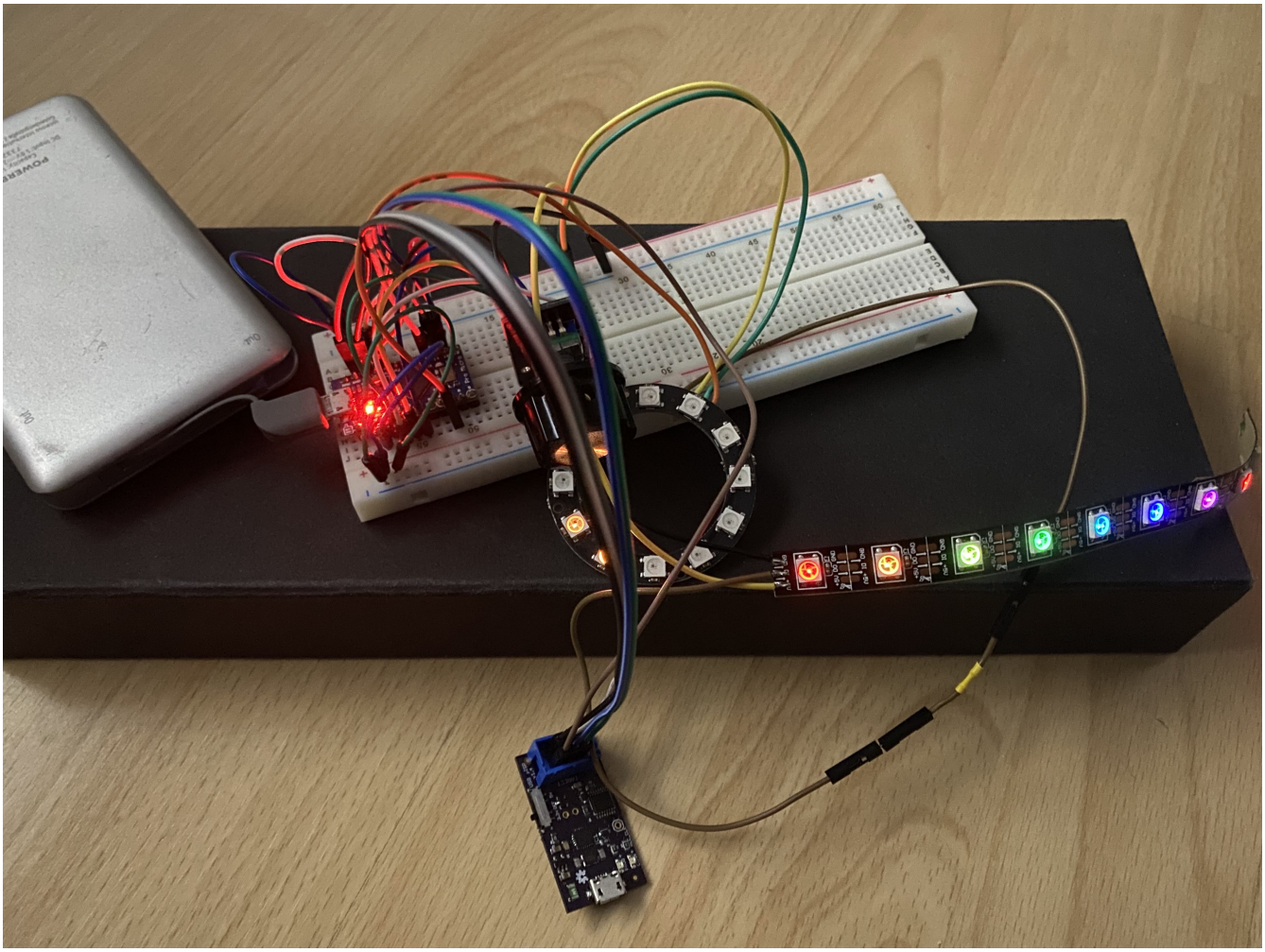

Figure 5. Initial breadboard prototype with WS2812 ring and strip

The above code runs a LED around the ring and sets static colours on the strip. A good start. The animation works with delay which is probably not a good solution but works fine for a first test.

4. Step 2 - Attach a button to trigger a stage

Since this will be starting rockets let's make it feel that way. Just the space bar and maybe a toggle to arm it.

Arduino reference

- keyboard
  - https://www.arduino.cc/reference/en/language/functions/usb/keyboard/
- Button
  - https://www.arduino.cc/en/Tutorial/BuiltInExamples/Button
- State change detection
  - https://www.arduino.cc/en/Tutorial/BuiltInExamples/StateChangeDetection

Figure 6. Resistor for pull down

We will need to pull down/up the pin for the button so the above resistor is included to show an example 10K Ohm resistor. The pro micro may have inbuilt pull up/down resistors but that has to be checked. Pulling a line up or down endures it is always in a defined state and that it reacts quickly so that it is highly recommended even for testing.

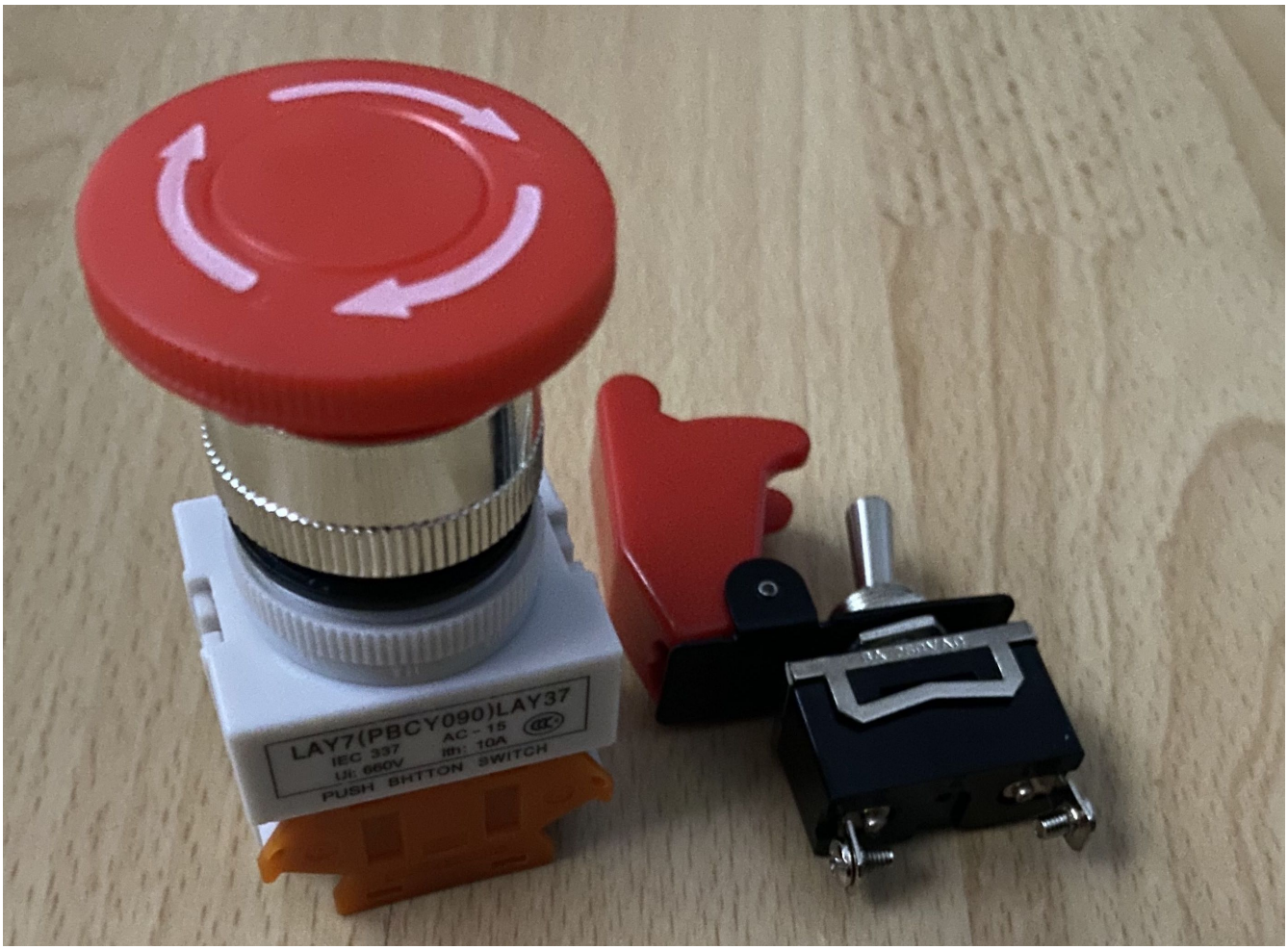

Figure 7. first buttons

The big red button is to trigger a stage and the toggle switch is to arm it. In this case we will soft arm the bitton as opposed to disconnecting it directly.

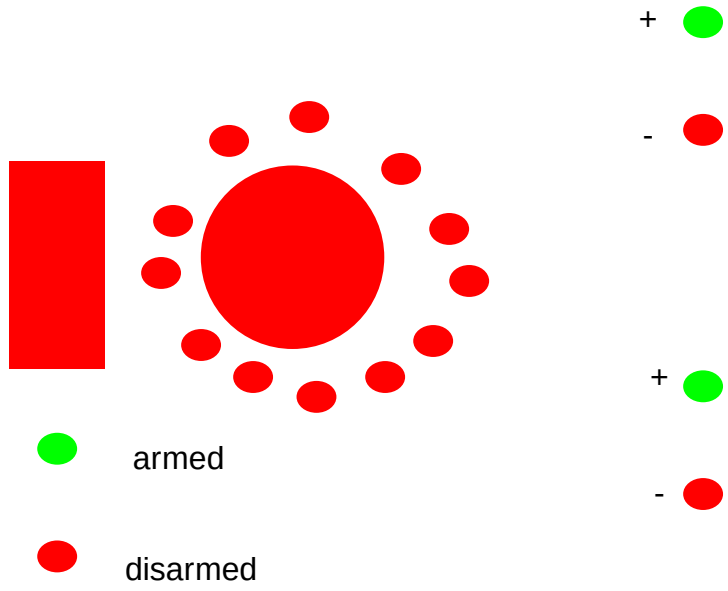

Figure 8. Very draft UI design

4.1. Bouncy Bouncy

Why all the fuss about bounce? And what is it?

- https://en.wikipedia.org/wiki/Switch

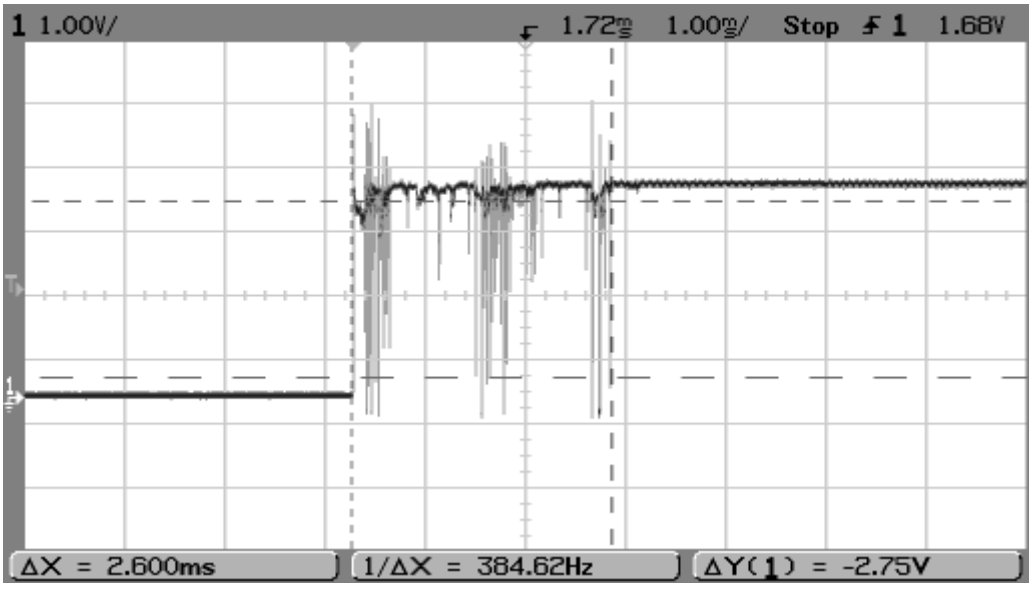

Figure 9. Example of bounce from the wikipedia

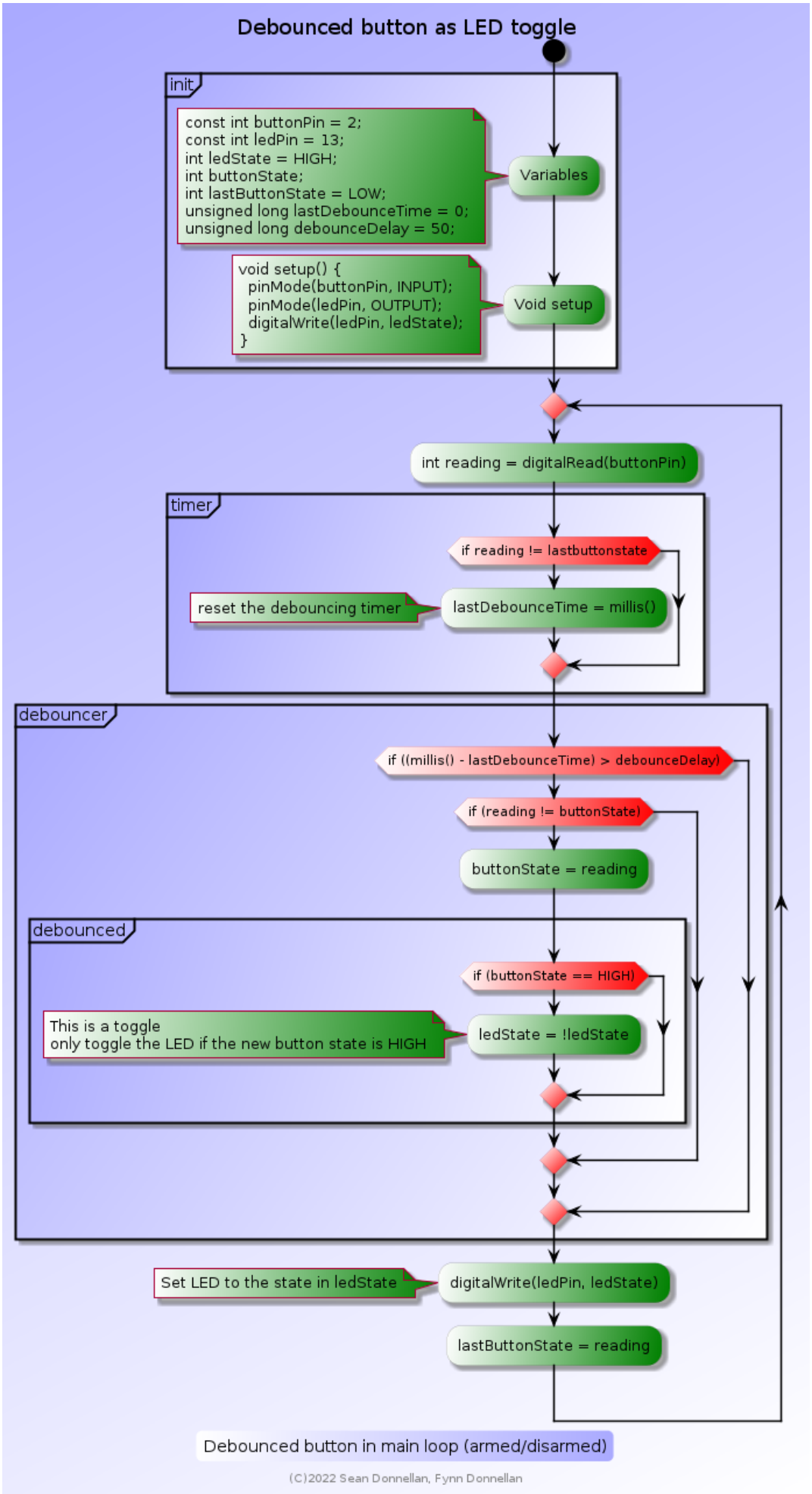

Figure 10. activity diagram to show the flow

4.1.1. Interrupts

> The Pro Micro has five external interrupts, which allow you to instantly trigger a function when a pin goes either high or low (or both). If you attach an interrupt to an interrupt-enabled pin, you'll need to know the specific interrupt that pin triggers: pin 3 maps to interrupt 0 (INT0), pin 2 is interrupt 1 (INT1), pin 0 is interrupt 2 (INT2), pin 1 is interrupt 3 (INT3), and pin 7 is interrupt 4 (INT6).
>
> — https://learn.sparkfun.com , pro micro hookup guide

Example interrupt driven routine debounced

```
void my_interrupt_handler()
{
  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();
  // If interrupts come faster than 200ms, assume it's a bounce and ignore
  if (interrupt_time - last_interrupt_time > 200)
  {
    ... do your thing
  }
  last_interrupt_time = interrupt_time;
}
```

The above short section shows a debounced interrupt that uses millis instead of delays. The important thing here is that if we debounce with delays we stall the whole loop so that if we have a LED animation or something else running it get's stalled. Using millis allows the rest to keep running.

Let's see if some of the above works.

4.2. first iteration

initial tests

```
const int buttonPin = 9;
const int ledPin = 13;
int ledState = HIGH;
int buttonState;
int lastButtonState = LOW;
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledState);
```

```
}

void loop() {
  int reading = digitalRead(buttonPin);
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }
  digitalWrite(ledPin, ledState);
  lastButtonState = reading;
}
```

The first test looks good and was tested on an arduino UNO.

The above code works as a debounced latching button. The aim though is to read a debounced button and to read it's state changes.

What we really need

- ONLY when the button is pressed
  - output state ONCE
- When button is released
  - Output state once

4.3. Second iteration

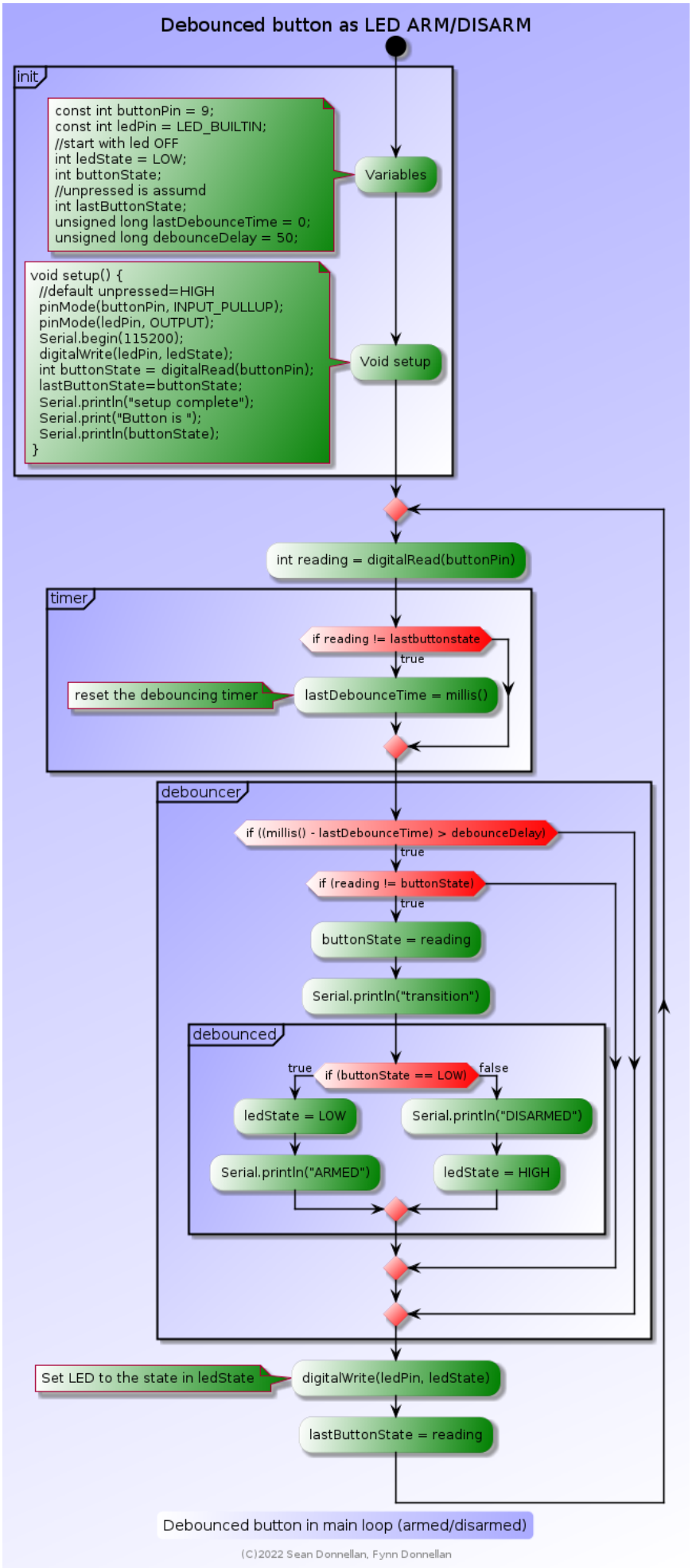

Figure 11. activity diagram to show the flow

Second button test non latching (armed/disarmed)

```
const int buttonPin = 9;
const int ledPin = LED_BUILTIN;
//start with led OFF
int ledState = LOW;
int buttonState;
//unpressed is assumd
int lastButtonState;
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

void setup() {
  //default unpressed=HIGH
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);
  digitalWrite(ledPin, ledState);
  int buttonState = digitalRead(buttonPin);
  lastButtonState=buttonState;
  Serial.println("setup complete");
  Serial.print("Button is ");
  Serial.println(buttonState);
}

void loop() {
  int reading = digitalRead(buttonPin);
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      Serial.println("transition");
      if (buttonState == LOW ) {
        Serial.println("ARMED");
        ledState = LOW;
      }
      else {
        Serial.println("DISARMED");
        ledState = HIGH;
      }
    }
  }
  digitalWrite(ledPin,ledState);
  lastButtonState = reading;
}
```

This works and it was tested on an arduino uno. Serial monitor was used to check the function.

5. Step 3 - bring things together to see how the loop runs with buttons

Integrated controller with LEDs, arm/disarm button, and stage button

```
#include <FastLED.h>
#include <Keyboard.h>

// Pin definitions
#define LED_DATA_PIN 7
#define ARM_BUTTON_PIN 8
#define STAGE_BUTTON_PIN 9

// LED setup
#define NUM_LEDS 22
#define RING_LEDS 12
CRGB leds[NUM_LEDS];

// Button states
int armButtonState = HIGH; // HIGH = unpressed (pullup)
int lastArmButtonState = HIGH;
unsigned long lastArmDebounceTime = 0;

int stageButtonState = HIGH;
int lastStageButtonState = HIGH;
unsigned long lastStageDebounceTime = 0;

unsigned long debounceDelay = 50;

// System state
bool armed = false;
bool startup = true;
bool stagePressed = false;

void setup() {
  // LEDs
  FastLED.addLeds<NEOPIXEL, LED_DATA_PIN>(leds, NUM_LEDS);
  FastLED.setBrightness(10); // Low brightness
  updateLEDs();

  // Buttons
  pinMode(ARM_BUTTON_PIN, INPUT_PULLUP);
  pinMode(STAGE_BUTTON_PIN, INPUT_PULLUP);

  // Keyboard
  Keyboard.begin();

  // Serial debugging
  Serial.begin(9600);
```

```
  Serial.println("Kerbal Controller Started - Disarmed");
}

void loop() {
  handleArmButton();
  handleStageButton();

  // LED animation when armed
  if (armed && !stagePressed) {
    static unsigned long lastLEDChange = 0;
    static int activeLED = 6;
    if (millis() - lastLEDChange > 100) {
      for(int i=6; i<=10; i++) leds[i] = CRGB::Black;
      leds[activeLED] = CRGB::Green;
      activeLED = (activeLED >= 10) ? 6 : activeLED + 1;
      lastLEDChange = millis();
    }
  }

  // Blink LEDs 6-10 orange while stage pressed
  if (stagePressed) {
    static unsigned long lastBlink = 0;
    static bool blinkState = false;
    if (millis() - lastBlink > 200) {
      CRGB color = blinkState ? CRGB::Orange : CRGB::Black;
      for(int i=6; i<=10; i++) leds[i] = color;
      blinkState = !blinkState;
      lastBlink = millis();
    }
  }

  // Debug output every second
  static unsigned long lastDebug = 0;
  if (millis() - lastDebug > 1000) {
    Serial.print("Armed: ");
    Serial.print(armed ? "YES" : "NO");
    Serial.print(" | ARM switch: ");
    Serial.print(armButtonState == LOW ? "ON" : "OFF");
    Serial.print(" | STAGE button: ");
    Serial.println(digitalRead(STAGE_BUTTON_PIN) == LOW ? "PRESSED" : "RELEASED");
    lastDebug = millis();
  }

  FastLED.show();
}

void handleArmButton() {
  int reading = digitalRead(ARM_BUTTON_PIN);
  if (reading != lastArmButtonState) {
    lastArmDebounceTime = millis();
  }
```

```
  if ((millis() - lastArmDebounceTime) > debounceDelay) {
    if (reading != armButtonState) {
      armButtonState = reading;
      if (startup) {
        startup = false;
      } else {
        armed = (armButtonState == LOW);
        updateLEDs();
      }
    }
  }
  lastArmButtonState = reading;
}

void handleStageButton() {
  int reading = digitalRead(STAGE_BUTTON_PIN);
  if (reading != lastStageButtonState) {
    lastStageDebounceTime = millis();
  }
  if ((millis() - lastStageDebounceTime) > debounceDelay) {
    if (reading != stageButtonState) {
      stageButtonState = reading;
      if (stageButtonState == LOW && armed) {
        if (!stagePressed) {
          Keyboard.press(' '); // Spacebar for stage
          delay(10);
          Keyboard.releaseAll();
          flashStageLED();
          stagePressed = true;
        }
      } else if (stageButtonState == HIGH) {
        stagePressed = false;
      }
    }
  }
  lastStageButtonState = reading;
}

void updateLEDs() {
  if (armed) {
    fill_solid(leds, NUM_LEDS, CRGB::Black);
    leds[0] = CRGB::Green; // Toggle indicator
  } else {
    fill_solid(leds, NUM_LEDS, CRGB::Red);
  }
}

void flashStageLED() {
  // Flash ring red briefly
  for (int i = 0; i < RING_LEDS; i++) {
    leds[i] = CRGB::Red;
```

```
147   }
148   FastLED.show();
149   delay(200);
150   updateLEDs();
151 }
```

This MVP integrates LEDs for status, debounced buttons, and keyboard emulation for staging.

6. Step 4 - rotary encoders

An initial example with interrupts

- https://gist.github.com/dkgrieshammer/66cce6ec92a6427c16804df84c22cc83

Appendix A: Requirements

Initial list of requirements

- Control Kerbal on PS5
  - Via USB(A) keyboard interface
    - Use big red button with latch for stages
  - Must show actions/button presses
    - WS2812 for key status changes red/green/etc
  - Add some safety toggles (arm/disarm)
    - A classic toggle with red cover
  - Control
    - Stage trigger (space bar)
    - SAS (on/off) (t)
    - gear (up/down) (g)
    - time warp (rotary +/-) (.,)
    - throttle (rotary +/-) (shift,cntrl)
    - motors (on/off) (x,k)
    - View (inside/outside) ???

This should cover the most required buttons and should be possible without multiplexing.

Appendix B: Interface design thoughts

Since the first button is a big red one with a latch it make sense to also show what state it's in. Adding a LED ring around it sounds like a good idea. Adding a LED ring around a rotary encoder also sounds like a good idea (optional).

B.1. Arm/disarm toggle

toggle disarmed

disArm led LED red(?), latch ring red blink(?)

toggle armed

ARM led green, latch ring green

B.2. Triger stage button

unlatched

ring green

press

ring red for 1 sec

latched

Ring orange

Appendix C: 3d printed test stand

The Aim here is to have a stand to mount the buttons and LEDs to while testing. After testing this can be used as a template for drilling. Also this can later be adapted to make a holder for the led ring and potentially the LEDs that can be mounted under the lid of the box.

OpenScad source

```
$fn=360;
// mm for slyrs box
topWidth=117;
topDepth=106;
topThick=2;
pillar=4;
height=46;
BigRedButtonD=22;
ToggleD=12;
WS2812D=4;
WS2812RingR=42/2;
numLEDs=12;
//versioning
letter_size = 10;
letter_height = topThick/2;
font = "Liberation Sans";
Version = "V3" ;

module letter(l) {
    linear_extrude(height = letter_height) {
        text(l, size = letter_size, font = font, halign = "center", valign =
```

```
   "center", $fn = 16);
22      }
23  }
24
25  module legs() {
26      //Legs are only needed during prototype phase
27      translate([0,0,0]) cube([pillar,pillar,height]);
28      translate([topWidth-pillar,0,0]) cube([pillar,pillar,height]);
29      translate([topWidth-pillar,topDepth-pillar,0]) cube([pillar,pillar,height]);
30      translate([0,topDepth-pillar,0]) cube([pillar,pillar,height]);
31  }
32  module boxTop() {
33      cube([topWidth,topDepth,topThick]);
34  }
35
36  module bigRedButton() {
37      // Big red Button
38      cylinder(h=topThick+2,d=BigRedButtonD);
39  }
40
41  module toggleSwitch() {
42      // toggle switch
43      cylinder(h=topThick+2,d=ToggleD);
44  }
45
46  module cherryKey() {
47      // toggle switch
48      cherryKeycap=18;
49      cherryClearance=cherryKeycap+1;
50      cylinder(h=topThick+2,d=cherryClearance);
51  }
52
53  module LED() {
54      // disarmed LED
55      cylinder(h=topThick+2,d=WS2812D);
56  }
57
58  module LEDRing() {
59      //LED ring
60      for ( i = [0 : 360/numLEDs : 360] ){
61              rotate([0, 0, i]) translate([0, WS2812RingR, -1]) LED();
62      }
63  }
64  module versioning() {
65      //text
66      letter(Version);
67  }
68
69  module testStand() {
70      legs();
71      translate([0,0,height])
```

```
72      difference() {
73          boxTop();
74          translate([topWidth/2,topDepth/2,-1]) bigRedButton();
75          translate([topWidth/6,topDepth/2-6,-1]) toggleSwitch();
76          translate([topWidth/6,topDepth/4,-1]) LED();
77          translate([topWidth/6,topDepth/4-10,-1]) LED();
78          //3 keys with circular keycaps chery MX clones
79          //keycaps have text on caps
80          translate([topWidth*.25,topDepth-(19/2)-7,-1]) cherryKey();
81          translate([topWidth*.5,topDepth-(19/2)-7,-1]) cherryKey();
82          translate([topWidth*.75,topDepth-(19/2)-7,-1]) cherryKey();
83          translate([topWidth/2,topDepth/2,0]) LEDRing();
84          translate([topWidth/2-10,+10,(topThick/2)+.5]) versioning();
85      }
86  }
87
88  testStand();
89
90
91  //add a right side with holes for rotary encoders or wait?
92  // wait?
93  // the side will have two rotary encoders
94  // one for Throttle and one for timewarp
95  // add Leds on the top ( + / toggle-Key / - )
96  // 3 LEDS for each encoder
97
98  module ringHolder() {
99      //module for led ring holder - DRAFT
100     holderH=2;
101     holderOutD=50;
102     holderInD=35;
103     holderFence=1;
104     holderFenceOutD=holderOutD+holderFence;
105     holderFenceInD=holderInD-holderFence;
106     difference() {
107         //holder ring
108         difference () {
109             cylinder(h=holderH+holderFence,d=holderFenceOutD);
110             translate([0,0,-1])
    cylinder(h=holderH+holderFence+2,d=holderFenceInD);
111         }
112         //led ring for subtraction
113         translate([0,0,holderFence+1]) difference () {
114             cylinder(h=holderH,d=holderOutD);
115             translate([0,0,-1]) cylinder(h=holderH+2,d=holderInD);
116         }
117     }
118 }
119 //draft for now
120 //ringHolder();
```

```
121 //LEDRing();
```

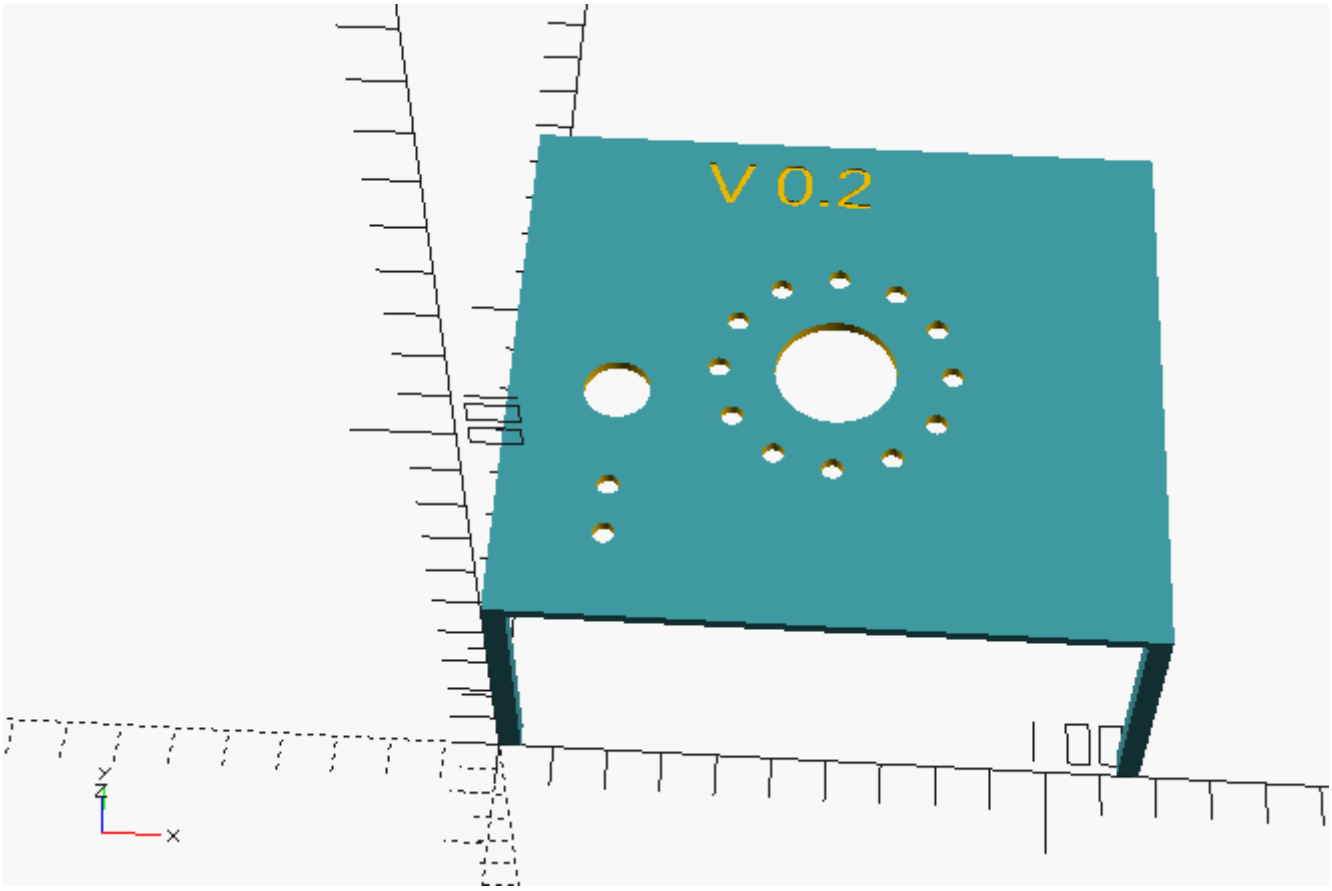

Figure 12. 3d stand STL (second iteration)

The first test fitting worked well to show up some room for improvement:

- Toggle switch moved left and down
- legs longer
- LED ring proper Radius

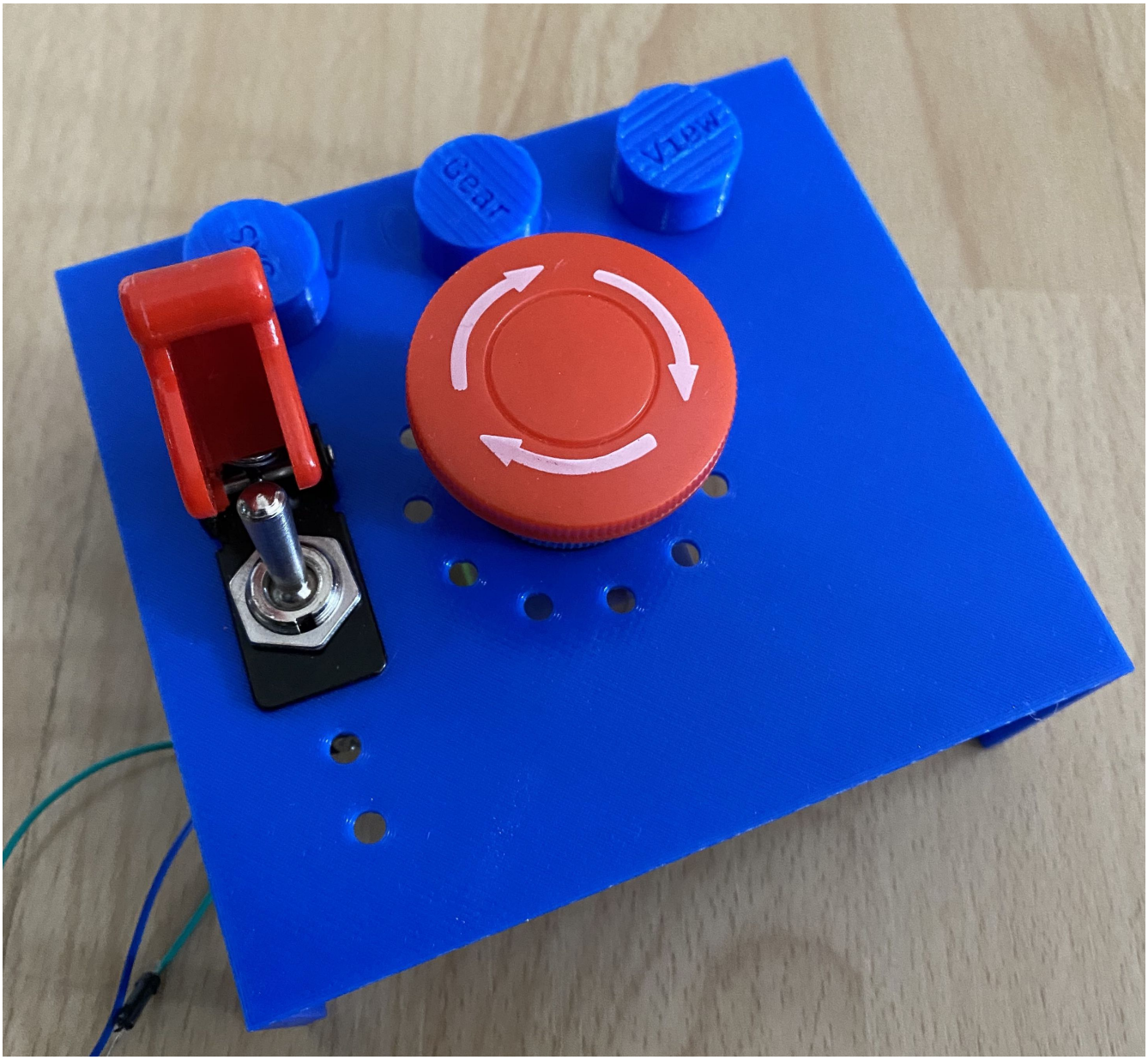

Figure 13. Prototype with v2

The intial V2 print populated with additional key caps placed to show approximate position of possible keys for later.

Appendix D: openscad library for custom keycaps

the following library allows the creation of keycaps in different sizes and shapes. since we'll be drilling the holes for any buttons round keycaps for cherry mx switches looks like a good option for keys.

- https://github.com/rsheldiii/KeyV2

As there is a broken backlit keyboard lying around it will probably find itself devoid of some switches for this if they fit.

D.1. Raw keycap print

Figure 14. draft keycaps with inlayed text for cherry mx (round)

Patch file for the above repo to make round keycaps

```
diff --git a/keys.scad b/keys.scad
index 768110e..aa6613f 100644
--- a/keys.scad
+++ b/keys.scad
@@ -9,7 +9,7 @@ include <./includes.scad>


 // example key
-dcs_row(5) legend("", size=9) key();
+//dcs_row(5) legend("", size=9) key();

 // example row
 /* for (x = [0:1:4]) {
@@ -17,4 +17,25 @@ dcs_row(5) legend("", size=9) key();
 } */

 // example layout
-/* preonic_default("dcs"); */
\ No newline at end of file
+/* preonic_default("dcs"); */
+
+union() {
+  // make the font smaller
+  $font_size = 3;
+  // top of keycap is the same size as the bottom
```

```
+  $width_difference = 0;
+  $height_difference = 0;
+  $key_shape_type="round";
+  $dish_type = "flat";
+  $height_slices = 10;
+  //$key_bump = "true";
+  // some keycap tops are slid backwards a little, and we don't want that
+  $top_skew = 0;
+  $support_type = "flared"; // [flared, bars, flat, disable]
+  $stem_support_type = "disable"; // [tines, brim, disabled]
+
+  legends = ["SAS", "Gear", "View"];
+  for(x=[0:len(legends)-1]) {
+    translate_u(x) cherry(0) legend(legends[x], size=4) key();
+  }
+}
\ No newline at end of file
diff --git a/src/shapes.scad b/src/shapes.scad
index 206727e..8fe8aa6 100644
--- a/src/shapes.scad
+++ b/src/shapes.scad
@@ -6,6 +6,7 @@ include <shapes/sculpted_square.scad>
 include <shapes/rounded_square.scad>
 include <shapes/square.scad>
 include <shapes/oblong.scad>
+include <shapes/round.scad>

 // size: at progress 0, the shape is supposed to be this size
 // delta: at progress 1, the keycap is supposed to be size - delta
@@ -25,6 +26,8 @@ module key_shape(size, delta, progress = 0) {
     square_shape(size, delta, progress);
   } else if ($key_shape_type == "oblong") {
     oblong_shape(size, delta, progress);
+  } else if ($key_shape_type == "round") {
+   round_shape(size, delta, progress);
   } else {
     echo("Warning: unsupported $key_shape_type");
   }
diff --git a/src/shapes/round.scad b/src/shapes/round.scad
new file mode 100644
index 0000000..62da4cd
--- /dev/null
+++ b/src/shapes/round.scad
@@ -0,0 +1,3 @@
+module round_shape(size, delta, progress){
+  rotate([0,0,22.5]) circle(d=size[0] - delta[0], $fn=360);
+}
\ No newline at end of file
```

The above keycaps fit the MX clone switches we have and will work for prototyping. To make them

look like really high quality switches some post processing will be required. This was a quick and dirty print in low resolution for initial testing.

Figure 15. Potential problem with clearance

Just to be sure the above mokup key was printed and it turns out there might be a clearance problem with the switch housing. It was not as apparent when test fitting to the switch as it was between other keys.

D.2. Keycap post processing

- One variant might be to print Caps with no inlay
  - Sand with 1000+ grit
  - Paint
  - Cover with tape
  - Laser inlay through tape
  - Paint inlay
  - Remove tape
- Another variant might be to print in the target colour with inlay
  - Paint inlay

- Sand with 1000+ grit

The above approaches need to be tested but since printing key caps is easy enough now this also should be easy enough. It might involve printing a positioning guide to help align any laser cutting and that can be done at a later stage if required.

D.2.1. Sand - Paint - Sand

Figure 16. Sand-Paint-Sand (without waiting for the paint to dry)

From this initial very fast test it looks like the laser option is getting more likely. Because the keycap is sloped the sanding uncovers the slope lines and, because we're hand sanding, unevenly. This leaves a slight artefact that would ideally beg for a coat of paint which would cover the inlay again so the simple sand - paint - sand option doesn't look that good. The painted part did look good though even if it didn't have time to dry. One question is what will the laser do with PLA. One question though is answered: The inlay looks good and sanding works well if combined with paint.

Figure 17. A closer look at the sanded surface (with a Wifi otoscope)

The above macro photo shows that PLA doesn't look that nice when sanded. There was an initial hope that heat might help smooth it again. The result of heat to the above key, while not shown, was uneven and unsatisfactory and probably requires more skill to do well so that the 2 minutes spent might have turned something better up if they had been 15 minutes. Still the result of the 2 minutes with a gas torch showed that the layer lines will be highlighted through heating so that it probably only makes good sense for a flat surface or one with finer layer lines more closely spaced. There is probably room for experiments with solvents here also as others have had success with that. The laser option looks to probably be the one with a higher degree of repeatability and probably a better surface finish also through sanding and painting and sanding and painting and then lasing through tape and filling in the inlay with the tape in place. The laser will require some help with positioning so a printed MX key stalk on a positioning plate for a wainlux K6 laser will probably be designed and printed to assist as otherwise the sloped keys are likely to be misaligned through rotation more than X/Y position. Maybe something like this: https://www.thingiverse.com/thing:4712402

D.2.2. Laser engraver approach

As described above using a laser to engrave the keycaps might be the road to a clean look.

The first step is to print a cherry keycap holder.

D.2.2.1. Cherry keycap holder

This is a requirement to be able to position the key caps repeatably. And this would have been good for the spray painting of the first cap too (maybe a bit higher). Since the keycap is slightly slanted this holder should probably account for that (V1 doesn't). The laser engraving should work fine none the less and if not the slant will be adapted to the holder also.

Openscad source

```
// Cherry MX keycap holder for laser engraving

cross_x = 4;
cross_y = 1.31;
cross_z = 3.6;
cross_tolerance = 0.2;
plate_z = 2 ;
key_slope = 5 ; // 5 degrees approx.

//versioning
letter_size = 8;
letter_height = 2;
font = "Liberation Sans";
Version = "V2" ;

module letter(l) {
    linear_extrude(height = letter_height) {
        text(l, size = letter_size, font = font, halign = "center", valign = "center",
$fn = 16);
    }
}
difference() {
    cube([20,20,plate_z],center=true);
    translate([0,5,plate_z/4]) letter(Version);
}
translate([0,0,plate_z/2+cross_z/2]) rotate([-5,0,0]) {
    cube([cross_y-cross_tolerance, cross_x-cross_tolerance, cross_z*2], center=true);
    cube([cross_x-cross_tolerance, cross_y-cross_tolerance, cross_z*2], center=true);
}
```

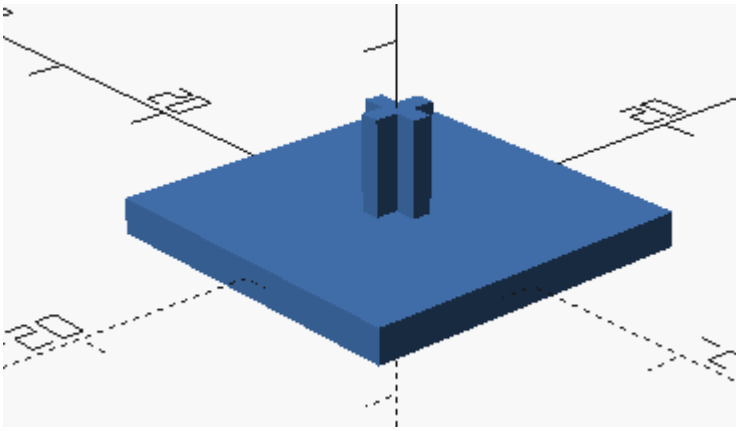

Figure 18. Screenshot of the keycap holder

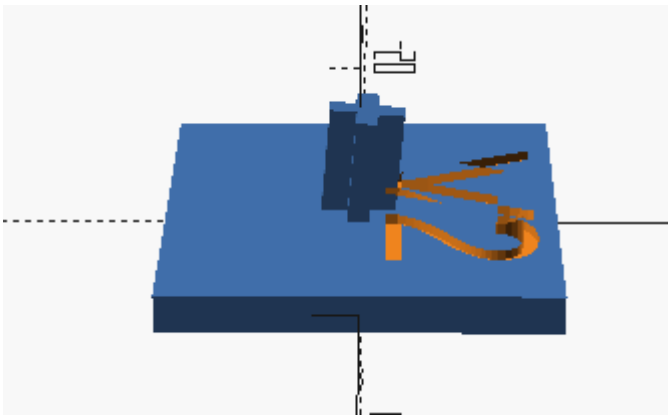

Figure 19. And the 5° tilted V2

The next steps (ToDo)

- Print a blank keycap
- Repeat as required:
  - Sand the keycap (1000 Grit or higher)
  - Spray paint the cap (1st coat with filler?)
- Mask the cap with tape
- Laser engrave
- Fill inlay
- Remove tape
- Optionally add clear coat

Appendix E: Current Progress and Next Steps

Code written. Compiled clean. USB fails. ISP wired. Ready to burn. Board waits. Test next.

- MVP controller code integrated in [`kerbal-controller.ino`](src/kerbal-controller/kerbal-controller.ino).
- Compiled successfully on Pi Zero using Arduino CLI.

- USB programming blocked by hardware issues; switched to ISP via avrdude.
- Wiring: Pi GPIOs to Pro Micro ISP header for SPI programming.
- Next: Upload via ISP, test in Kerbal Space Program.

7. Development Pipeline

This project uses a split development workflow between a host machine (Steamdeck, with VS Code) and a Raspberry Pi (used for compiling and uploading to the Arduino Pro Micro). This allows for convenient editing and version control on the host, while leveraging the Pi for hardware interfacing and programming.

7.1. Workflow Steps

1. Edit code on the host (Steamdeck) using VS Code or your preferred editor.
2. Save changes locally.
3. Transfer the updated code to the Raspberry Pi using `scp`:

```
scp /path/to/src/kerbal-controller/kerbal-controller.ino
sean@arduinoworkstation.fritz.box:/home/sean/kerbal-control/
```

4. SSH into the Raspberry Pi:

```
ssh sean@arduinoworkstation.fritz.box
```

5. On the Pi, compile the code using Arduino CLI:

```
cd /home/sean/kerbal-control
/home/sean/bin/arduino-cli compile --fqbn arduino:avr:leonardo src/kerbal-
controller/kerbal-controller.ino
```

6. Upload the compiled code to the Arduino Pro Micro:

```
/home/sean/bin/arduino-cli upload -p /dev/ttyACM0 --fqbn arduino:avr:leonardo
src/kerbal-controller/kerbal-controller.ino
```

7. (Optional) Monitor serial output for debugging:

```
screen /dev/ttyACM0 9600
```

This pipeline allows for rapid iteration: edit on the host, transfer, compile/upload on the Pi, and test on the Arduino.

7.2. Sequence Diagram

```
Unresolved directive in README.asciidoc - include::plantuml/dev-pipeline-
sequence.puml[]
```