

kerbal-control

Sean Donnellan, Fynn Donnellan

Table of Contents

1. About this document	1
2. Starting point	1
2.1. kerbal key bindings	2
2.2. Arduino - pro micro (5V)	2
2.3. WS2812 LEDs	2
3. Step 1 - First prototype	2
4. Step 2 - Attach a button to trigger a stage	4
4.1. Bouncy Bouncy	6
4.1.1. Interrupts	8
4.2. first iteration	8
4.3. Second iteration	9
5. Step 3 - bring things together to see how the loop runs with buttons	12
6. Step 4 - rotary encoders	13
Appendix A: Requirements	13
Appendix B: Interface design thoughts	14
B.1. Arm/disarm toggle	14
B.2. Triger stage button	14
Appendix C: 3d printed test stand	14
Appendix D: openscad library for custom keycaps	18
D.1. Raw keycap print	19
D.2. Keycap post processing	21
D.2.1. Sand - Paint - Sand	21
D.2.2. Laser engraver approach	23
D.2.2.1. Cherry keycap holder	24

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/](http://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

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)

```
1 #include <FastLED.h>
2 #define NUM_LEDS 22
3 #define NUM_RING_LEDS 12
4 #define DATA_PIN 7
5
6 CRGB leds[NUM_LEDS];
7
8 void setup() {
9     FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
10    leds[12] = CHSV(0, 255, 16);
11    leds[13] = CHSV(33, 255, 16);
12    leds[14] = CHSV(65, 255, 16);
13    leds[15] = CHSV(97, 255, 16);
14    leds[16] = CHSV(129, 255, 16);
15    leds[17] = CHSV(161, 255, 16);
16    leds[18] = CHSV(193, 255, 16);
17    leds[19] = CHSV(225, 255, 16);
18    leds[20] = CHSV(255, 255, 16);
19    leds[21] = CHSV(255, 255, 16);
20    FastLED.show();
21 }
22
23 void loop() {
24     for(int dot = 0; dot < NUM_RING_LEDS; dot++) {
25         leds[dot] = CHSV(64, 255, 16);
26         FastLED.show();
27         // clear this led for the next time around the loop
28         leds[dot] = CRGB::Black;
29         delay(150);
30     }
31 }
```



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 2. 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 3. 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 ensures it is always in a defined state and that it reacts quickly so that it is highly recommended even for testing.



Figure 4. 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 button as opposed to disconnecting it directly.



Figure 5. 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 6. Example of bounce from the wikipedia



Figure 7. 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

```
1 void my_interrupt_handler()
2 {
3     static unsigned long last_interrupt_time = 0;
4     unsigned long interrupt_time = millis();
5     // If interrupts come faster than 200ms, assume it's a bounce and ignore
6     if (interrupt_time - last_interrupt_time > 200)
7     {
8         ... do your thing
9     }
10    last_interrupt_time = interrupt_time;
11 }
```

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 gets stalled. Using millis allows the rest to keep running.

Let's see if some of the above works.

4.2. first iteration

initial tests

```
1 const int buttonPin = 9;
2 const int ledPin = 13;
3 int ledState = HIGH;
4 int buttonState;
5 int lastButtonState = LOW;
6 unsigned long lastDebounceTime = 0;
7 unsigned long debounceDelay = 50;
8
9 void setup() {
10  pinMode(buttonPin, INPUT_PULLUP);
11  pinMode(ledPin, OUTPUT);
12  digitalWrite(ledPin, ledState);
13 }
14
15 void loop() {
16  int reading = digitalRead(buttonPin);
17  if (reading != lastButtonState) {
18    lastDebounceTime = millis();
19  }
20  if ((millis() - lastDebounceTime) > debounceDelay) {
21    if (reading != buttonState) {
22      buttonState = reading;
23      if (buttonState == HIGH) {
24        ledState = !ledState;
25      }
26    }
27  }
28  digitalWrite(ledPin, ledState);
29  lastButtonState = reading;
30 }
```



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 8. activity diagram to show the flow

Second button test non latching (armed/disarmed)

```
1 const int buttonPin = 9;
2 const int ledPin = LED_BUILTIN;
3 //start with led OFF
4 int ledState = LOW;
5 int buttonState;
6 //unpressed is assumed
7 int lastButtonState;
8 unsigned long lastDebounceTime = 0;
9 unsigned long debounceDelay = 50;
10
11 void setup() {
12     //default unpressed=HIGH
13     pinMode(buttonPin, INPUT_PULLUP);
14     pinMode(ledPin, OUTPUT);
15     Serial.begin(115200);
16     digitalWrite(ledPin, ledState);
17     int buttonState = digitalRead(buttonPin);
18     lastButtonState=buttonState;
19     Serial.println("setup complete");
20     Serial.print("Button is ");
21     Serial.println(buttonState);
22 }
23
24 void loop() {
25     int reading = digitalRead(buttonPin);
26     if (reading != lastButtonState) {
27         lastDebounceTime = millis();
28     }
29     if ((millis() - lastDebounceTime) > debounceDelay) {
30         if (reading != buttonState) {
31             buttonState = reading;
32             Serial.println("transition");
33             if (buttonState == LOW ) {
34                 Serial.println("ARMED");
35                 ledState = LOW;
36             }
37             else {
38                 Serial.println("DISARMED");
39                 ledState = HIGH;
40             }
41         }
42     }
43     digitalWrite(ledPin,ledState);
44     lastButtonState = reading;
45 }
```



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



Work in Progress

Second test with LED ring and leds plus 1 button

```
1 #include <FastLED.h>
2 //fastled stuff
3 #define NUM_LEDS 22
4 #define NUM_RING_LEDS 12
5 #define DATA_PIN 7
6 #define disarmed_LED 12
7 #define armed_LED 13
8 #define SAS_LED 14
9 #define rot1p_LED 15
10 #define rot1m_LED 16
11 #define rot1b_LED 17
12 CRGB leds[NUM_LEDS];
13
14 //button stuff
15 const int buttonPin = 8;
16 const int ledPin = LED_BUILTIN;
17 //start with led OFF
18 int ledState = LOW;
19 int buttonState;
20 //unpressed is assumed
21 int lastButtonState;
22 unsigned long lastDebounceTime = 0;
23 unsigned long debounceDelay = 5;
24
25 void setup() {
26     //LED stuff
27     FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
28     leds[disarmed_LED] = CRGB::Red;
29     leds[armed_LED] = CRGB::Black;
30     leds[SAS_LED] = CRGB::Black
31     leds[rot1p_LED] = CRGB::Black
32     leds[rot1m_LED] = CRGB::Black;
33     leds[rot1b_LED] = CRGB::Black;
34     //spare
35     leds[18] = CHSV(0, 255, 16);
36     leds[19] = CHSV(63, 255, 16);
37     leds[20] = CHSV(127, 255, 16);
38     leds[21] = CHSV(255, 255, 16);
39     //END LEDs
40     FastLED.show();
41     //button stuff
42     //default unpressed=HIGH
```

```

43     pinMode(buttonPin, INPUT_PULLUP);
44     pinMode(ledPin, OUTPUT);
45     digitalWrite(ledPin, ledState);
46     int buttonState = digitalRead(buttonPin);
47     lastButtonState=buttonState;
48 //The rest
49 Serial.begin(115200);
50 Serial.println("setup complete");
51 }
52
53 void loop() {
54     for(int dot = 0; dot < NUM_RING_LEDS; dot++) {
55         leds[dot] = CHSV(64, 255, 16);
56         FastLED.show();
57         // clear this led for the next time around the loop
58         leds[dot] = CRGB::Black;
59         delay(150);
60     }
61 }
```

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

```

1 $fn=360;
2 // mm for slyrs box
3 topWidth=117;
4 topDepth=106;
5 topThick=2;
6 pillar=4;
7 height=46;
8 BigRedButtonD=22;
9 ToggleD=12;
10 WS2812D=4;
```

```

11 WS2812RingR=42/2;
12 numLEDs=12;
13 //versioning
14 letter_size = 10;
15 letter_height = topThick/2;
16 font = "Liberation Sans";
17 Version = "V3" ;
18
19 module letter(l) {
20     linear_extrude(height = letter_height) {
21         text(l, size = letter_size, font = font, halign = "center", valign =
22             "center", $fn = 16);
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])
111            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();

```

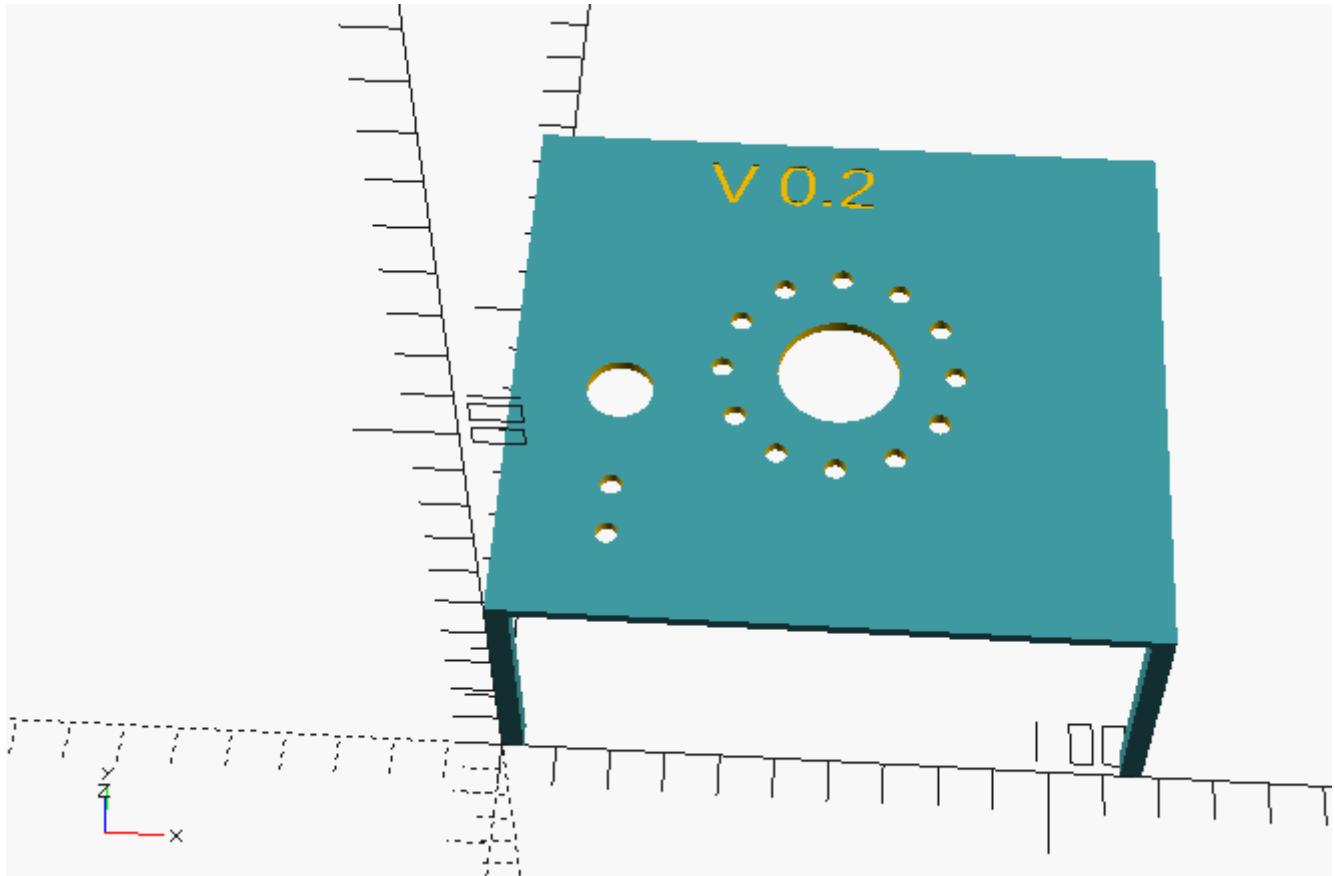


Figure 9. 3d stand STL (second iteration)

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

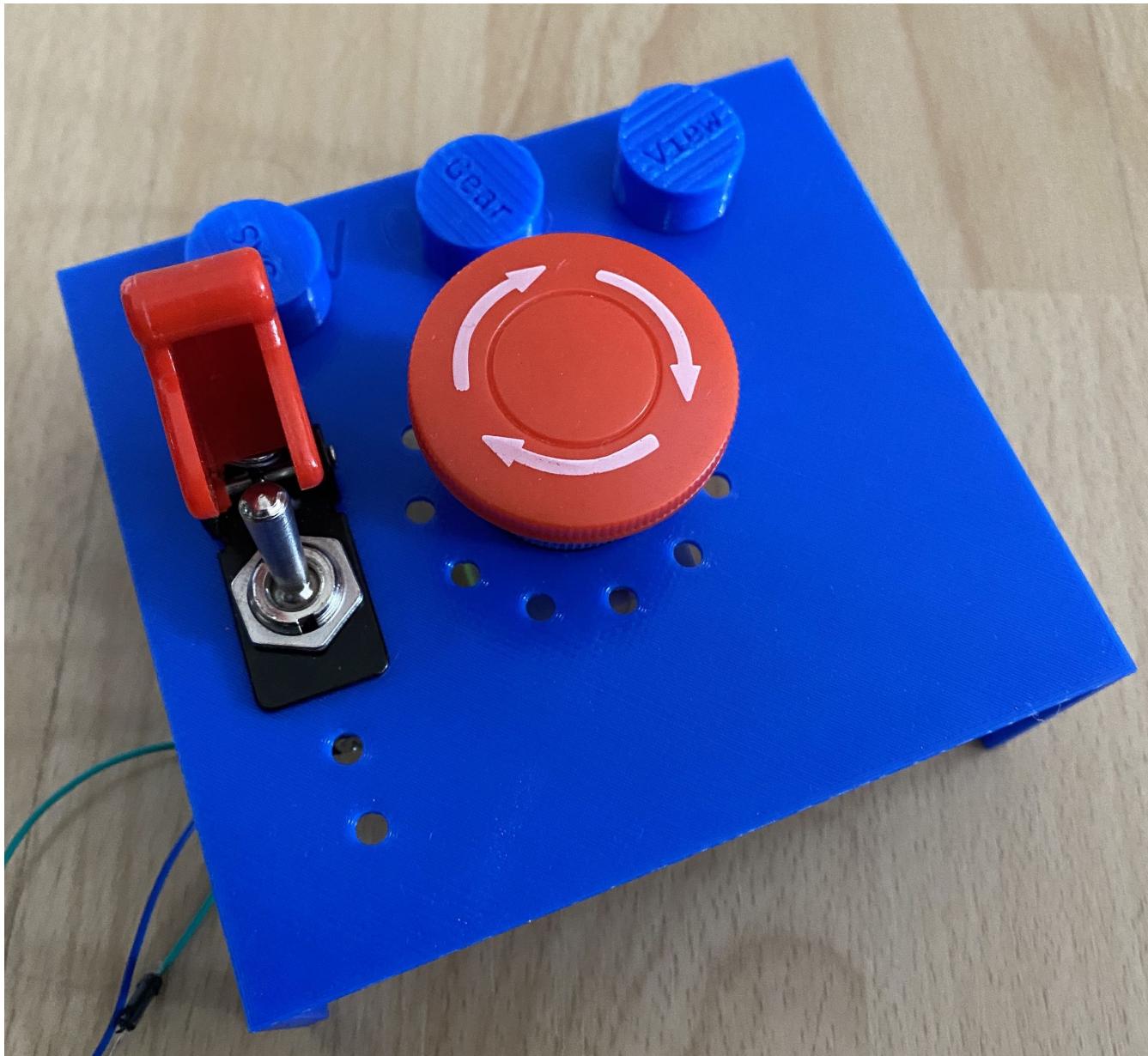


Figure 10. Prototype with v2

The initial 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 11. draft keycaps with inlaid 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.

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 12. 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 13. 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 lasering 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 ;

//versioning
letter_size = 8;
letter_height = 2;
font = "Liberation Sans";
Version = "V1" ;

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]){
    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);
}
```

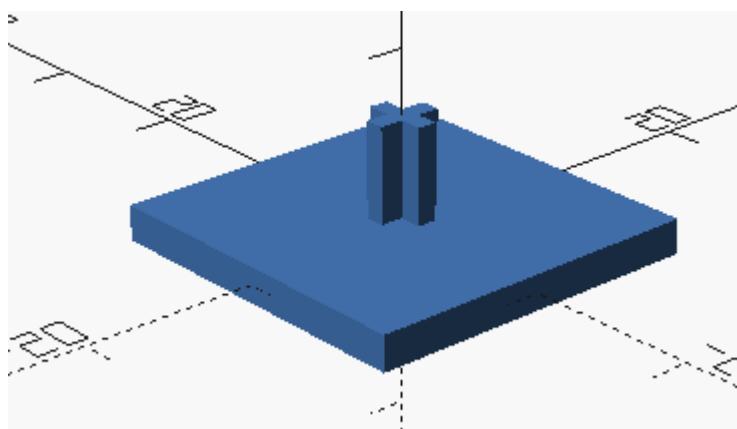


Figure 14. Screenshot of the keycap holder

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