kerbal-control

Sean Donnellan, Fynn Donnellan

Table of Contents

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

- https://www.kerbalspaceprogram.com/

Starting point

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 vi 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.

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

Arduino - pro micro (5V)

Initial choice for a prototype is the AVR ATmega32u4 8-bit microcontroller which has a USB controller and can therefore 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

WS2812 LEDs

Because it's always good to have status LEDs an the WS2812 is one that is both easy to get and has good libraries with fastled and adafruit.

Fastled

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

The fastled library looks like a good choice although it doesn't (yet) support RGBW LEDs for which I have a LED ring. The other LED rings I have are less tightly packed with LEDs. As I have a few WS2812 strips on top of the one RGBW ring the choice went towards the WS2812 to be able to mix the strip and ring. RGBW is better suited to lighting anyway so let's use it for that later.

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.

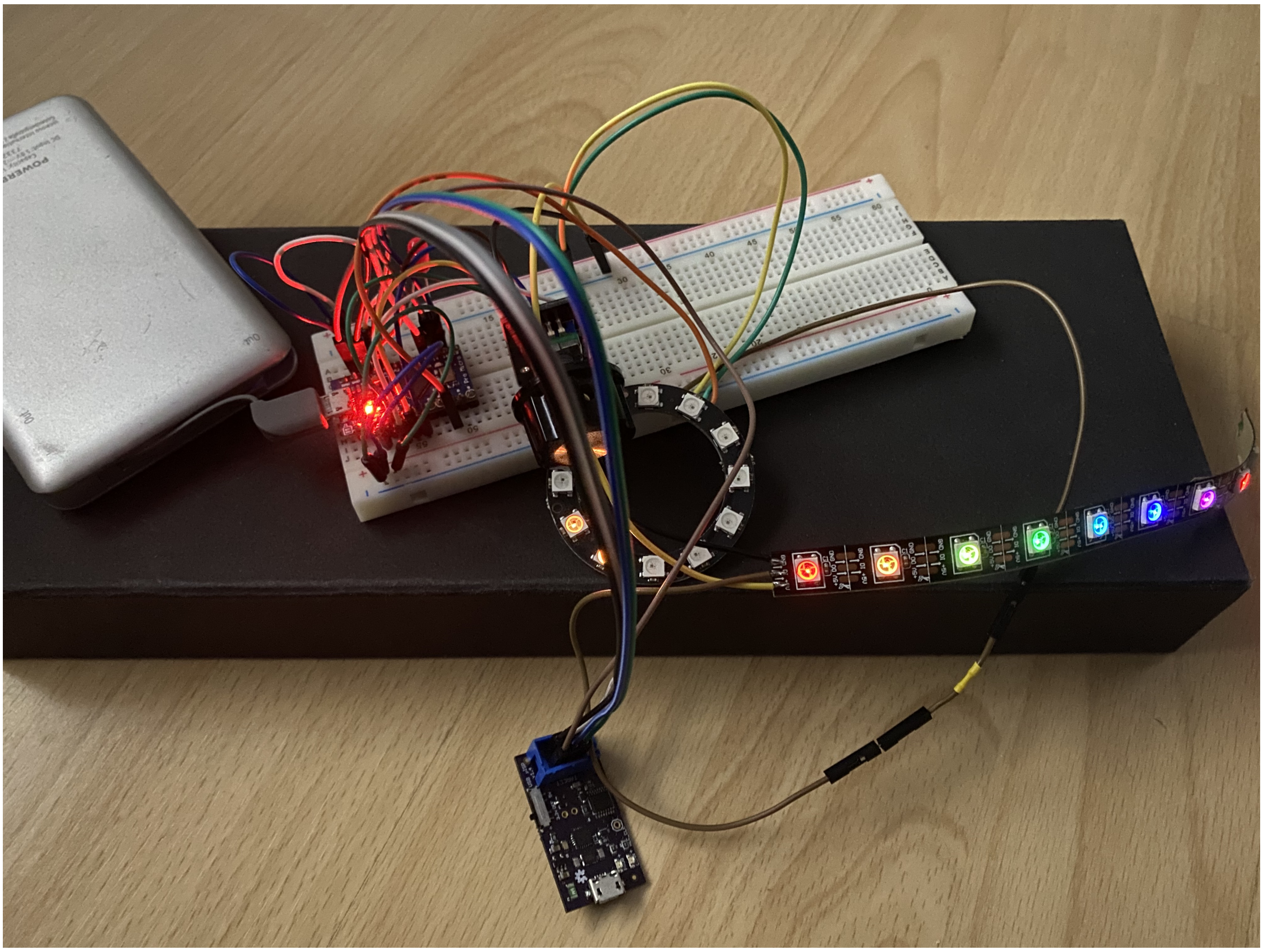

Figure 1. Initial breadboard prototype with WS2812 ring and strip

The above code runs a LED around the ring and sets static colours on the strip. A good start.

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 2. Resistor for pull down

We will need to pull down the pin for the button so the above resistor is included to show an example 10K Ohm resistor.

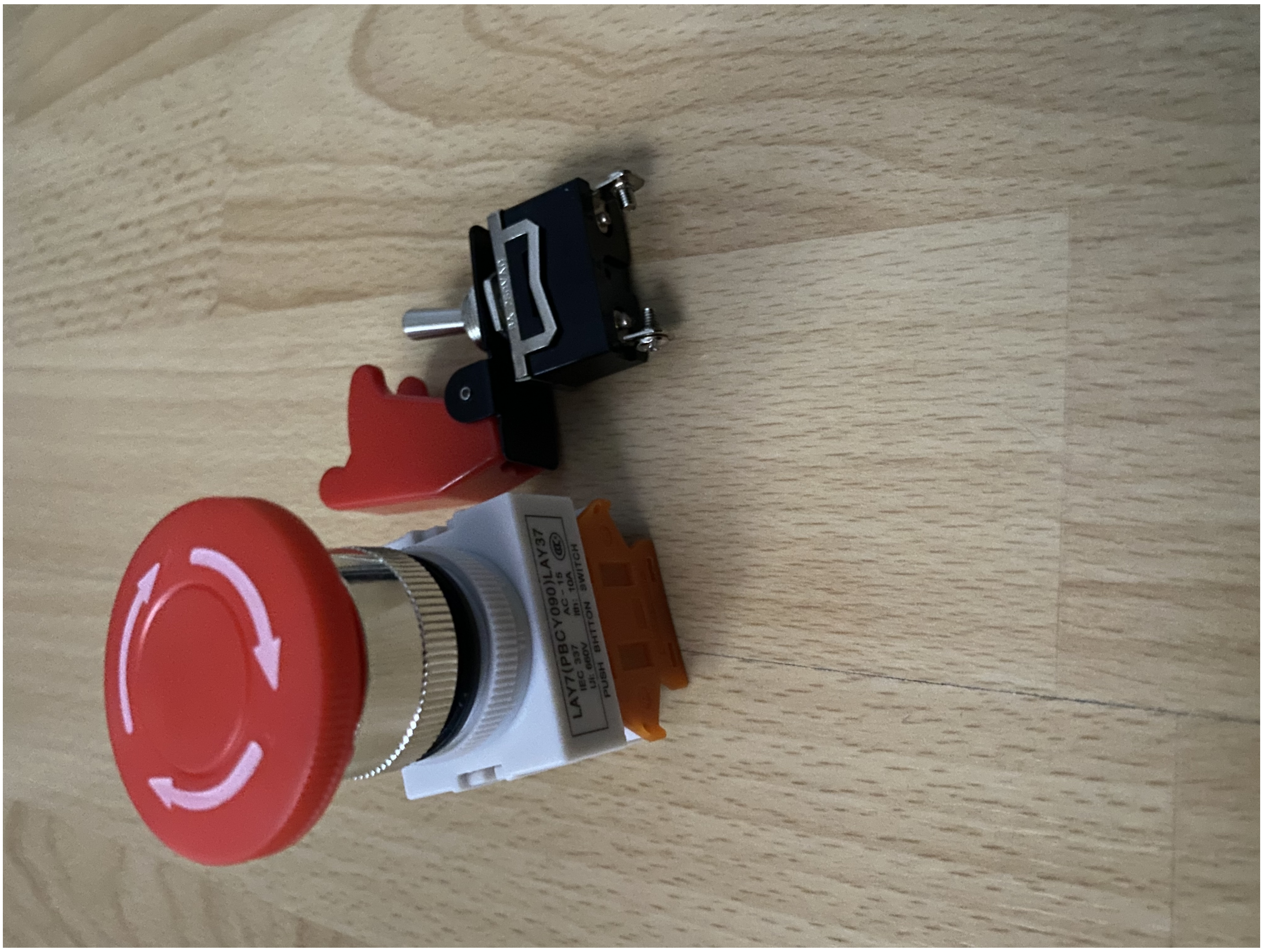

Figure 3. first buttons

The big red button is to trigger a stage and the toggle switch is to arm it.

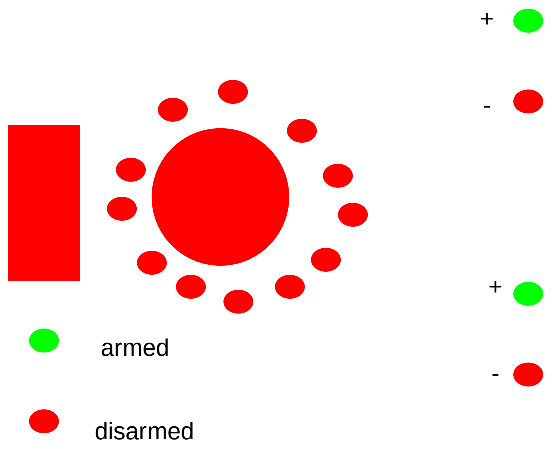

Figure 4. Very draft UI design

Let’s see if that works.

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

const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  //begins emulating a keyboard
  Keyboard.begin();
}

void BRB_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 you thing here
    last_interrupt_time = interrupt_time;
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
    //press spacebar on a keyboard
    Keyboard.press((char) 0x20);
    delay(10);
    Keyboard.releaseAll();
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

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.

Step 3 - 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).

Arm/disarm toggle

toggle disarmed
: Arm led LED green(?), latch ring red blink(?)

toggle armed
: ARM led red, latch ring green

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=40;
BigRedButtonD=22;
ToggleD=12;
WS2812D=4;
```

```
WS2812RingD=18;
numLEDs=12;
letter_size = 4;
letter_height = topThick/2;
font = "Liberation Sans";

module letter(l) {
    linear_extrude(height = letter_height) {
        text(l, size = letter_size, font = font, halign = "center", valign = "center",
$fn = 16);
    }
}

//Legs are only needed during prototype phase
translate([0,0,0]) cube([pillar,pillar,height]);
translate([topWidth-pillar,0,0]) cube([pillar,pillar,height]);
translate([topWidth-pillar,topDepth-pillar,0]) cube([pillar,pillar,height]);
translate([0,topDepth-pillar,0]) cube([pillar,pillar,height]);

//Top of the box for reference
translate([0,0,height])
difference() {
    cube([topWidth,topDepth,topThick]);
    // Big red Button
    translate([topWidth/2,topDepth/2,-1]) cylinder(h=topThick+2,d=BigRedButtonD);
    // toggle switch
    translate([topWidth/4,topDepth/2,-1]) cylinder(h=topThick+2,d=ToggleD);
    // disarmed LED
    translate([topWidth/4,topDepth/4,-1]) cylinder(h=topThick+2,d=WS2812D);
    // Armed LED
    translate([topWidth/4,topDepth/4-10,-1]) cylinder(h=topThick+2,d=WS2812D);
    //text
    translate([topWidth/4-15,topDepth/4-10,(topThick/2)+.5]) letter("Armed");
    translate([topWidth/4-15,topDepth/4,(topThick/2)+.5]) letter("DisArmed");
    //LED ring
    translate([topWidth/2,topDepth/2,0])
        for ( i = [0 : 360/numLEDs : 360] ){
            rotate([0, 0, i]) translate([0, WS2812RingD, -1])
cylinder(h=topThick+2,d=WS2812D);
    }
}
```

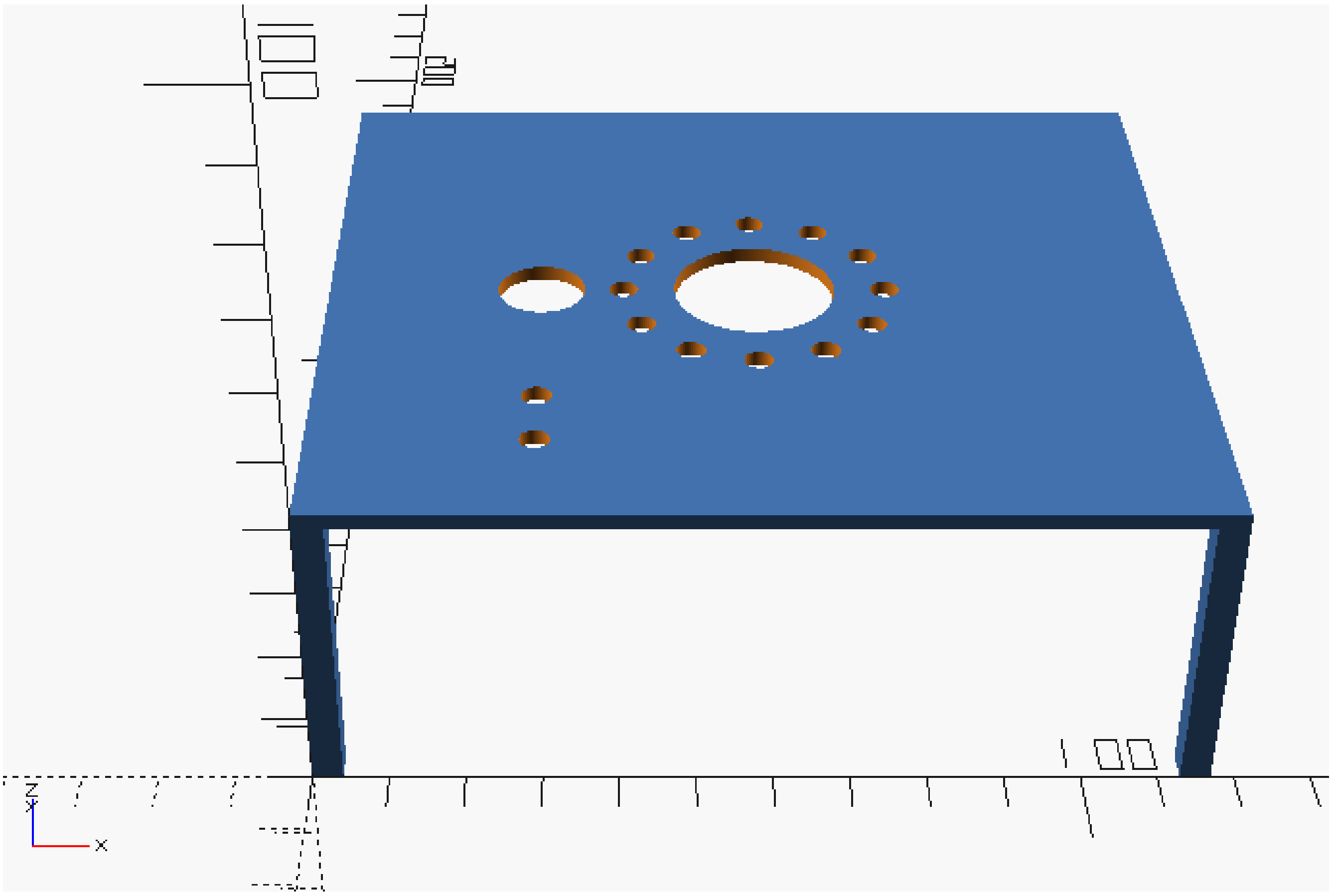

Figure 5. 3d stand STL (very first very draft iteration)