

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

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

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 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

2.3. 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.

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.



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.

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.



Figure 4. first buttons

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

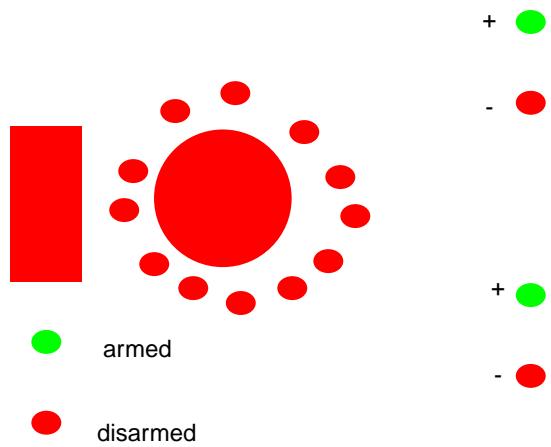


Figure 5. Very draft UI design

4.1. Bouncy Bouncy

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

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

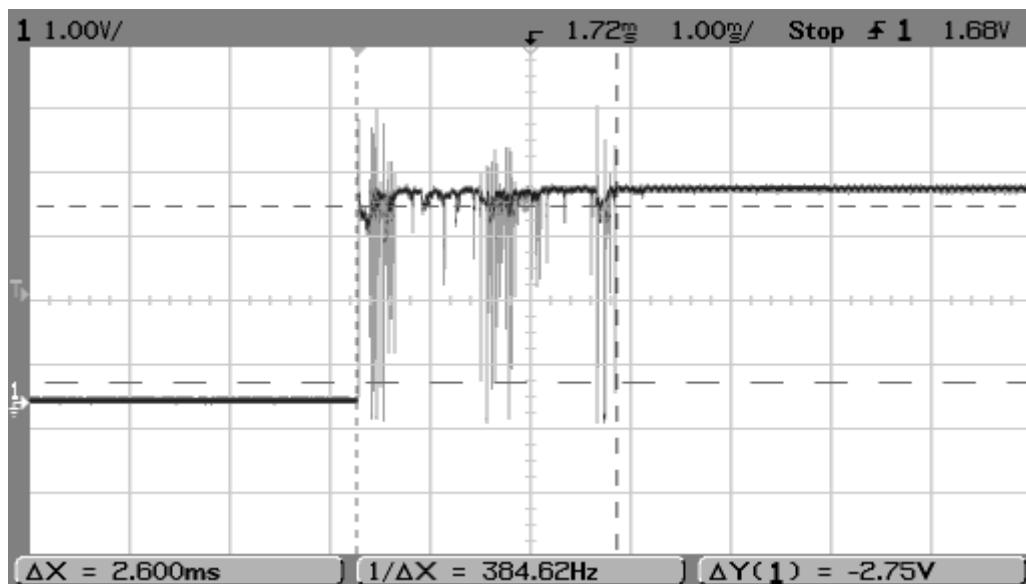


Figure 6. Example of bounce from the wikipedia



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

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

toggle armed

ARM led red, 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 = "V 0.3" ;
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 //Legs are only needed during prototype phase
26 translate([0,0,0]) cube([pillar,pillar,height]);
27 translate([topWidth-pillar,0,0]) cube([pillar,pillar,height]);
28 translate([topWidth-pillar,topDepth-pillar,0]) cube([pillar,pillar,height]);
29 translate([0,topDepth-pillar,0]) cube([pillar,pillar,height]);
30
31 //Top of the box for reference
32 translate([0,0,height])
33 difference() {
34     cube([topWidth,topDepth,topThick]);
35     // Big red Button
36     translate([topWidth/2,topDepth/2,-1]) cylinder(h=topThick+2,d=BigRedButtonD);
37     // toggle switch
38     translate([topWidth/6,topDepth/2-6,-1]) cylinder(h=topThick+2,d=ToggleD);
39     // disarmed LED
40     translate([topWidth/6,topDepth/4,-1]) cylinder(h=topThick+2,d=WS2812D);
41     // Armed LED
42     translate([topWidth/6,topDepth/4-10,-1]) cylinder(h=topThick+2,d=WS2812D);
43     //text
44     translate([topWidth/2-10,topDepth-10,(topThick/2)+.5]) letter(Version);
45     //LED ring
46     translate([topWidth/2,topDepth/2,0])
47         for ( i = [0 : 360/numLEDs : 360] ){
48             rotate([0, 0, i]) translate([0, WS2812RingR, -1])
49             cylinder(h=topThick+2,d=WS2812D);
50         }
51
52 //add a right side with holes for rotary encoders or wait?
53 // wait?
54 // the side will have two rotary encoders
55 // one for Throttle and one for timewarp
56 // add Leds on the top ( + / toggle-Key / - )
57 // 3 LEDS for each encoder
58
59 module ringHolder() {

```

```

60 //module for led ring holder - DRAFT
61 holderH=2;
62 holderOutD=42;
63 holderInD=32;
64 holderFence=1;
65 holderFenceOutD=holderOutD+holderFence;
66 holderFenceInD=holderInD-holderFence;
67 difference() {
68     //holder ring
69     difference () {
70         cylinder(h=holderH+holderFence,d=holderFenceOutD);
71         translate([0,0,-1]) cylinder(h=holderH+holderFence+2,d=holderFenceInD);
72     }
73     //led ring for subtraction
74     translate([0,0,holderFence+1]) difference () {
75         cylinder(h=holderH,d=holderOutD);
76         translate([0,0,-1]) cylinder(h=holderH+2,d=holderInD);
77     }
78 }
79 }
80 //draft for now
81 //ringHolder();

```

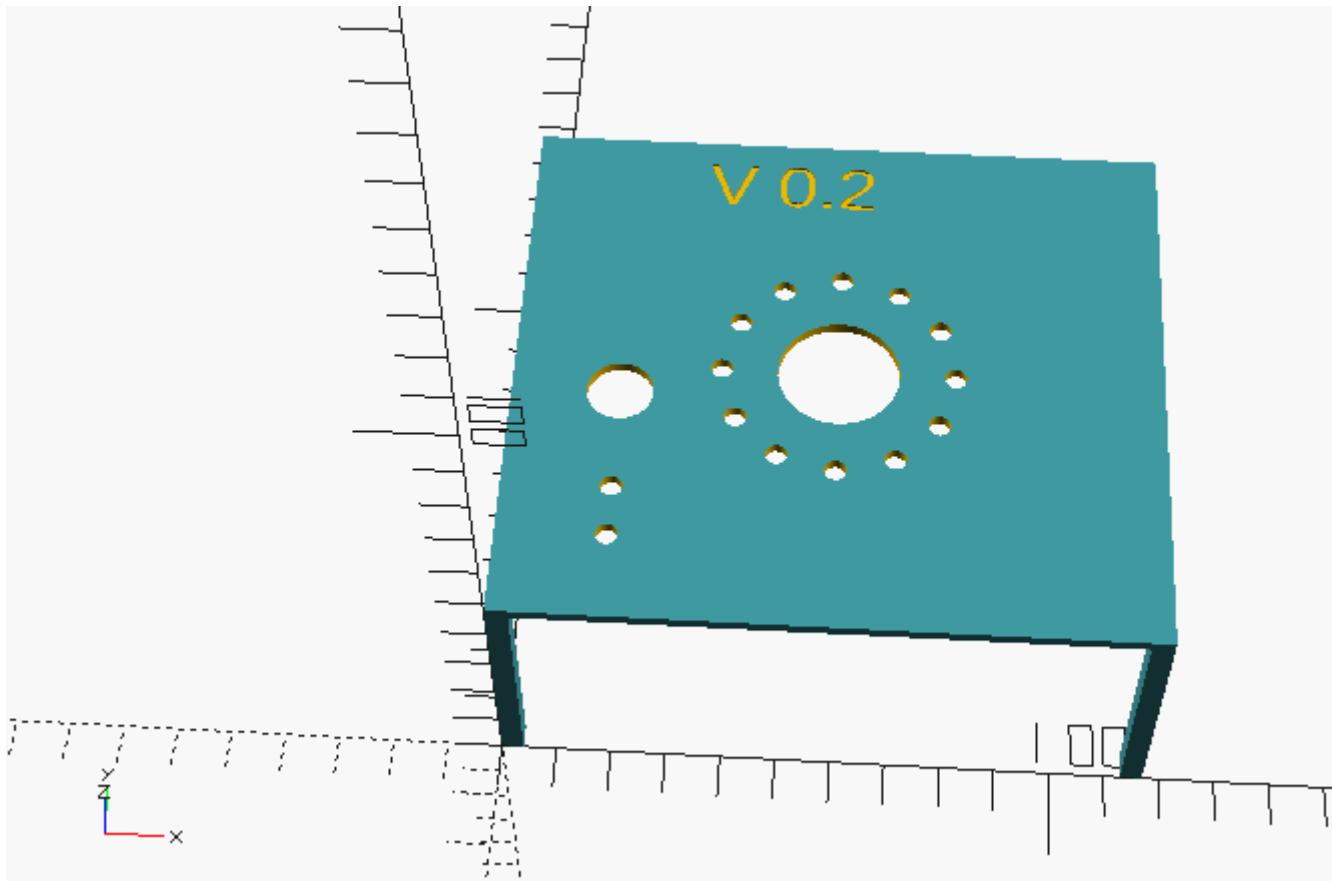


Figure 9. 3d stand STL (second iteration)

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