Breadboard Bend Layout (BBL)

Jeremy Huang, Griffin Zajac, Alex Gao

Opportunity:

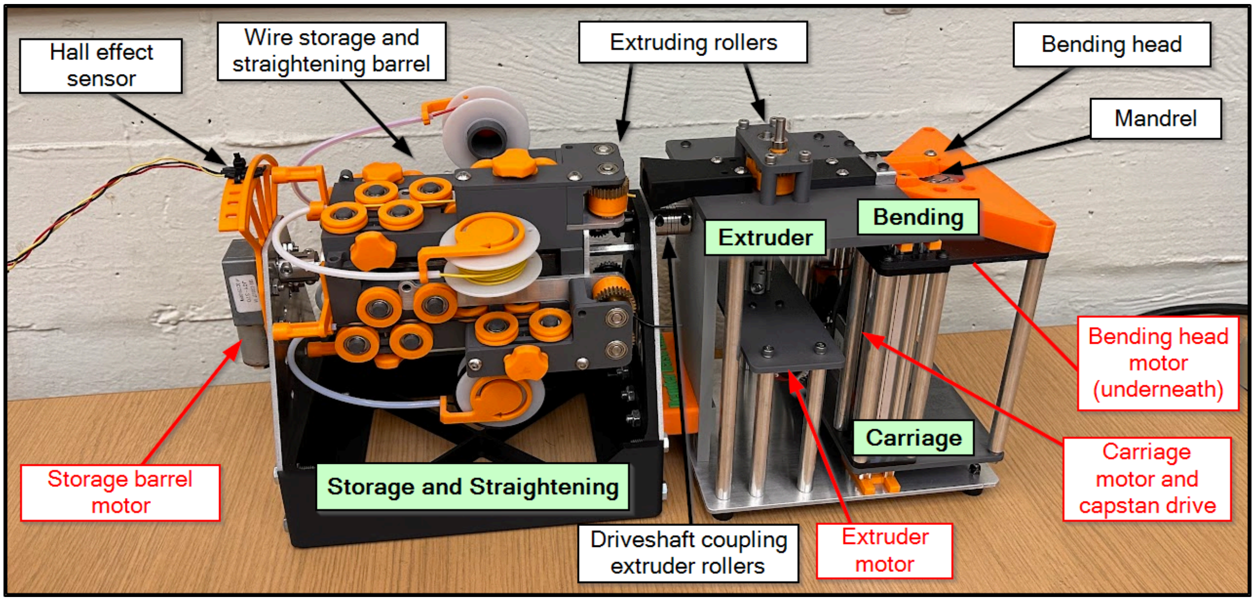
Neat wiring is a time consuming task, yet can be extremely helpful in debugging circuits. The task of manually creating neat wiring involves carefully bending and cutting wires to a desired shape and size. The alternative is to quickly wire something up, creating a messy and un-debuggable breadboard when issues inevitably arise in testing. We created a breadboard wire bender that takes custom inputs of color, length, and bend locations to produce the user's desired wire configuration, allowing you to relax and consistently produce aesthetically pleasing and easy to understand circuits.

High-Level Strategy:

Our high level strategy for the BBL was to separate each stage of the wire processing into independently functioning subsystems—storage/straightening, extruding, and bending/cutting. Each subsystem was designed to the requirements of adjacent subsystems, i.e. the extruder design assumed a straight wire with a consistent position, and the bending head assumed consistent extruding from a fixed point. Up to four different colors of wire can be loaded into the straightening modules, which use two sets of vertical and horizontal rollers to remove any curvature or kinks. To accomplish color switching without an independent motor for each straightener module, a stationary bevel gear on the extruder engages and disengages with the various colors as they rotate into position so that each module can actively feed wire.

There is an additional set of extruder rollers that feed the wire through an aluminum block to the bending head. The bending head and mandrel lie on a carriage that can move up and down. The carriage serves two purposes. Primarily, it allows for the bending head to be lowered so that it can be moved underneath the wire to do both left and right bends from either side of the wire. It also allows for automated wire cutting. Our strategy was to use the mandrel and the aluminum block on the extruder as shears in order to cut the wire with upward motion of the carriage. In implementation, the 3D printed mandrel and carriage deflected too much to fully cut the wire, although it would consistently strip the wire, and with enough repetitions, the wire was able to be fatigued off.

Photo of Integrated Physical Device:


Hall effect sensor
Wire storage and straightening barrel
Extruding rollers
Bending head
Mandrel
Extruder
Bending
Bending head motor (underneath)
Carriage
Carriage motor and capstan drive
Storage barrel motor
Storage and Straightening
Driveshaft coupling extruder rollers
Extruder motor

Function-Critical Decisions and Calculations:

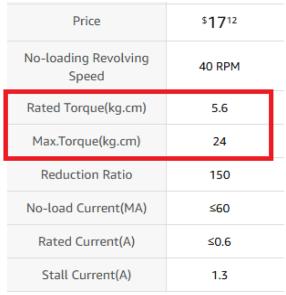
All motors have worm gearboxes, so they aren't backdrivable and flexible shaft couplings are not necessarily required.

Storage Motor Sizing:

The storage barrel is conservatively assumed to have a hoop mass distribution. An additional 1.5 factor of safety was applied and 5.25 kg was used in calculations. Our target acceleration for the barrel rotation was 4π rad/s^2, and the 60% stall torque rule was applied to find a required continuous torque.

$$I_{hoop} = M * R^2$$
$$= 5.25kg * (0.089m)^2$$
$$= 0.042\ kgm^2$$

$$\tau = (I_{hoop} * \alpha)/0.6$$
$$= (0.042\ kgm^2 * 4\pi/s^2)/0.6$$
$$= (0.528\ Nm)/0.6$$
$$\boxed{= 8.97\ kg\ cm}$$

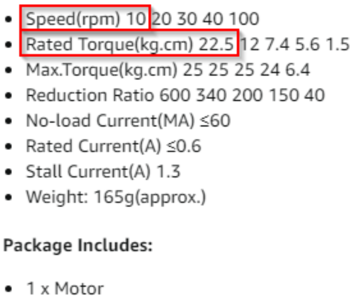


Price	$17^{12}
No-loading Revolving Speed	40 RPM
Rated Torque(kg.cm)	5.6
Max.Torque(kg.cm)	24
Reduction Ratio	150
No-load Current(MA)	≤60
Rated Current(A)	≤0.6
Stall Current(A)	1.3

Carriage Motor Sizing:

The maximum load case is assumed to be when attempting to cut the wire, calculated based on the cross sectional area of 22 AWG wire at 0.324mm^2, the maximum tensile strength of copper of 350 MPa corresponding to a shear strength of 175 MPa, and a winch pulley radius of 18mm. The 60% stall torque rule was applied to find the required continuous torque, leaving us with a 1.4 factor of safety.

$$F = \tau_{Cu}A = 175MPa * 0.324mm^2$$
$$= 56.7N = 5.78kgf$$
$$FR_{Winch} = \tau_{Motor}$$
$$= 5.78kgf * 1.8cm = 10.4kg\ cm$$
$$\tau_{Cont} = \tau/0.6 = \boxed{17.3kg\ cm}$$

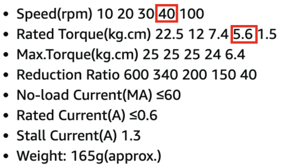


- Speed(rpm) 10 20 30 40 100
- Rated Torque(kg.cm) 22.5 12 7.4 5.6 1.5
- Max.Torque(kg.cm) 25 25 25 24 6.4
- Reduction Ratio 600 340 200 150 40
- No-load Current(MA) ≤60
- Rated Current(A) ≤0.6
- Stall Current(A) 1.3
- Weight: 165g(approx.)

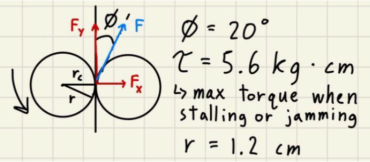
Package Includes:

- 1 x Motor

Forces on Extrusion Bearings: https://www.mcmaster.com/57155K476/, similar bearing properties

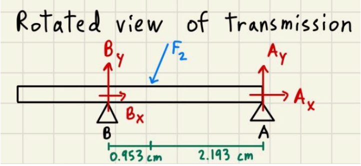


- Speed(rpm) 10 20 30 40 100
- Rated Torque(kg.cm) 22.5 12 7.4 5.6 1.5
- Max.Torque(kg.cm) 25 25 25 24 6.4
- Reduction Ratio 600 340 200 150 40
- No-load Current(MA) ≤60
- Rated Current(A) ≤0.6
- Stall Current(A) 1.3
- Weight: 165g(approx.)



$$\tau = F * r_c = F * r * \cos(20°)$$
$$F = 4.966\ kg$$
$$F_x = 4.966 * \sin(20°) = 1.7\ kg$$
$$F_y = 4.966 * \cos(20°) = 4.67\ kg$$

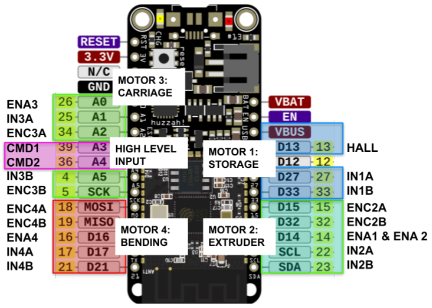
These forces would be applied on the wire and the other gear, so the bearings would experience $F_2 = -1.7\,\hat{x} - 4.7\hat{y}$



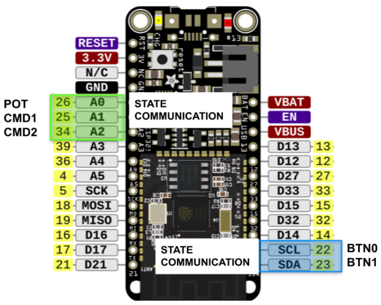
$$\Sigma F_x = 0 = -1.7 + \mathrm{B}_x + \mathrm{A}_x$$
$$\Sigma F_y = 0 = -4.67 + \mathrm{B}_y + \mathrm{A}_y$$
$$\Sigma M_x = 0$$
$$M_B = 0 = \mathrm{A}_x * 3.146\ cm - 1.7 * 0.953\ cm \rightarrow\ \mathrm{A}_x = 0.515\ kg$$
$$0 = -1.7 + \mathrm{B}_x + \mathrm{A}_x \rightarrow\ \mathrm{B}_x = 1.185\ kg$$
$$\Sigma M_y = 0$$
$$M_B = 0 = \mathrm{A}_y * 3.146\ cm - 4.67 * 0.953\ cm \rightarrow\ \mathrm{A}_x = 1.415\ kg$$
$$0 = -4.67 + \mathrm{B}_x + \mathrm{A}_x \rightarrow\ \mathrm{B}_x = 3.255\ kg$$

The greatest force experienced is 3.255 kg, similar bearings are rated for a static load of 43 kg. So in the worst case, the system is designed with an FOS = 13.2.

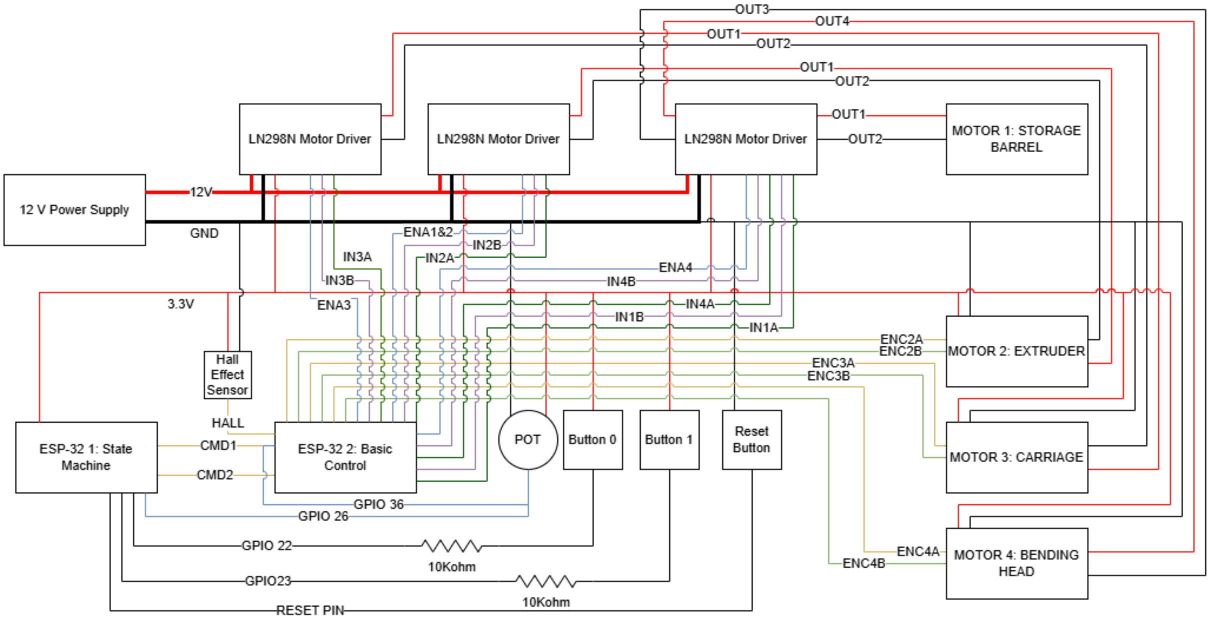
Circuit Diagrams:



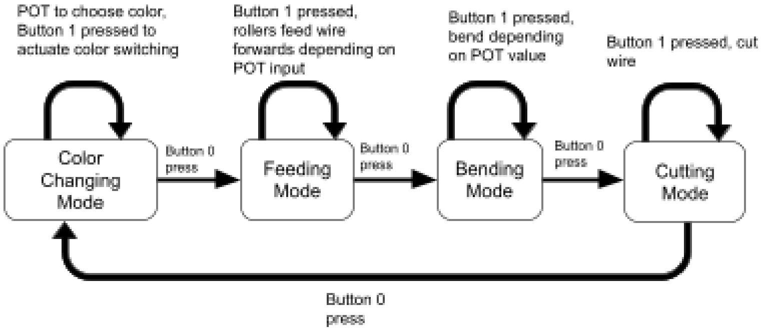
Low-Level Motor Control Breadboard



State Machine Breadboard



State Transition Diagram:



POT to choose color, Button 1 pressed to actuate color switching
Button 1 pressed, rollers feed wire forwards depending on POT input
Button 1 pressed, bend depending on POT value
Button 1 pressed, cut wire
Color Changing Mode
Button 0 press
Feeding Mode
Button 0 press
Bending Mode
Button 0 press
Cutting Mode
Button 0 press

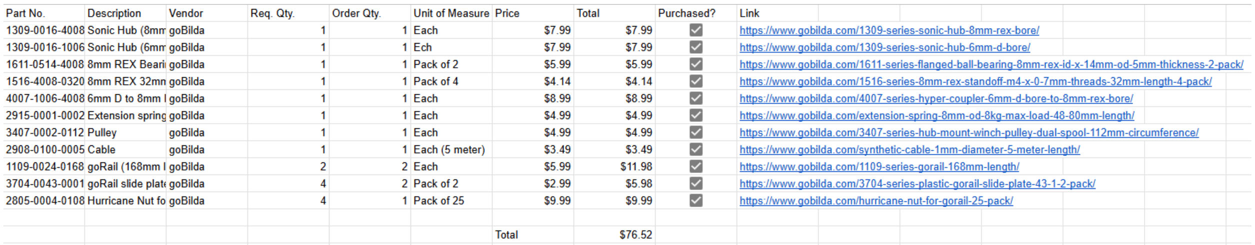
Reflection

It was helpful to work consistently on the project throughout the semester. Even though a large amount of work was done towards the end, we had time to include increased scope and avoid a stressful end of semester.

Appendix A

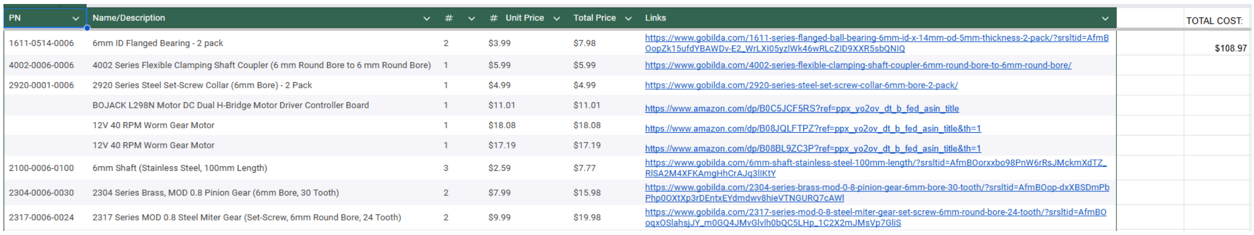
Final Product Bill of Materials: Final Deliverables BOM

Bending + Carriage:



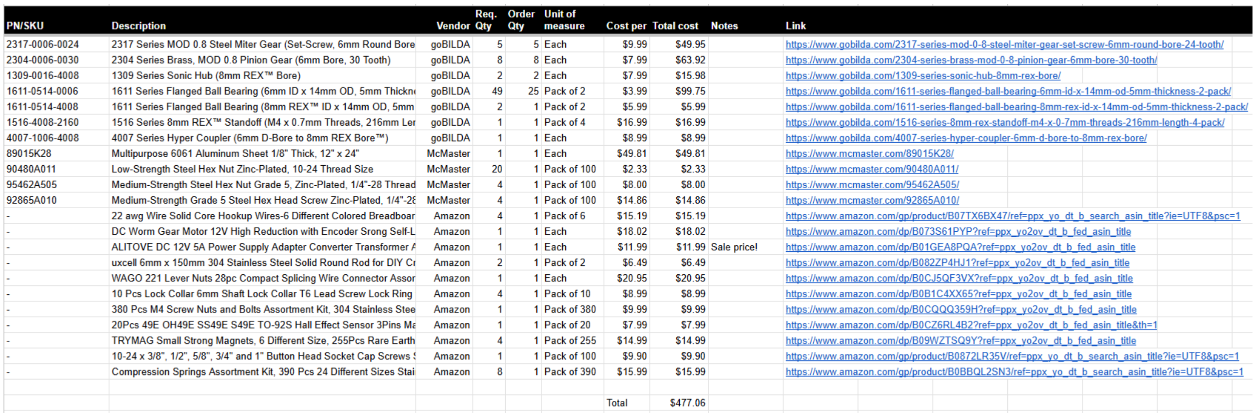
Part No.	Description	Vendor	Req. Qty.	Order Qty.	Unit of Measure	Price	Total	Purchased?	Link
1309-0016-4008	Sonic Hub (8mm	goBilda	1	1	Each	$7.99	$7.99	☑	https://www.gobilda.com/1309-series-sonic-hub-8mm-rex-bore/
1309-0016-1006	Sonic Hub (6mm	goBilda	1	1	Ech	$7.99	$7.99	☑	https://www.gobilda.com/1309-series-sonic-hub-6mm-d-bore/
1611-0514-4008	8mm REX Beari	goBilda	1	1	Pack of 2	$5.99	$5.99	☑	https://www.gobilda.com/1611-series-flanged-ball-bearing-8mm-rex-id-x-14mm-od-5mm-thickness-2-pack/
1516-4008-0320	8mm REX 32mn	goBilda	1	1	Pack of 4	$4.14	$4.14	☑	https://www.gobilda.com/1516-series-8mm-rex-standoff-m4-x-0-7mm-threads-32mm-length-4-pack/
4007-1006-4008	6mm D to 8mm I	goBilda	1	1	Each	$8.99	$8.99	☑	https://www.gobilda.com/4007-series-hyper-coupler-6mm-d-bore-to-8mm-rex-bore/
2915-0001-0002	Extension spring	goBilda	1	1	Each	$4.99	$4.99	☑	https://www.gobilda.com/extension-spring-8mm-od-8kg-max-load-48-80mm-length/
3407-0002-0112	Pulley	goBilda	1	1	Each	$4.99	$4.99	☑	https://www.gobilda.com/3407-series-hub-mount-winch-pulley-dual-spool-112mm-circumference/
2908-0100-0005	Cable	goBilda	1	1	Each (5 meter)	$3.49	$3.49	☑	https://www.gobilda.com/synthetic-cable-1mm-diameter-5-meter-length/
1109-0024-0168	goRail (168mm I	goBilda	2	2	Each	$5.99	$11.98	☑	https://www.gobilda.com/1109-series-gorail-168mm-length/
3704-0043-0001	goRail slide plat	goBilda	4	2	Pack of 2	$2.99	$5.98	☑	https://www.gobilda.com/3704-series-plastic-gorail-slide-plate-43-1-2-pack/
2805-0004-0108	Hurricane Nut fo	goBilda	4	1	Pack of 25	$9.99	$9.99	☑	https://www.gobilda.com/hurricane-nut-for-gorail-25-pack/
						Total	$76.52		

Extruding:



PN	Name/Description	#	Unit Price	Total Price	Links		TOTAL COST:
1611-0514-0006	6mm ID Flanged Bearing - 2 pack	2	$3.99	$7.98	https://www.gobilda.com/1611-series-flanged-ball-bearing-6mm-id-x-14mm-od-5mm-thickness-2-pack/?srsltid=AfmBOopZk15ufdYBAWDv-E2_WrLXI05yzlWk46wRLcZID9XXR5sbQNIQ		$108.97
4002-0006-0006	4002 Series Flexible Clamping Shaft Coupler (6 mm Round Bore to 6 mm Round Bore)	1	$5.99	$5.99	https://www.gobilda.com/4002-series-flexible-clamping-shaft-coupler-6mm-round-bore-to-6mm-round-bore/		
2920-0001-0006	2920 Series Steel Set-Screw Collar (6mm Bore) - 2 Pack	1	$4.99	$4.99	https://www.gobilda.com/2920-series-steel-set-screw-collar-6mm-bore-2-pack/		
	BOJACK L298N Motor DC Dual H-Bridge Motor Driver Controller Board	1	$11.01	$11.01	https://www.amazon.com/dp/B0C5JCF5RS?ref=ppx_yo2ov_dt_b_fed_asin_title		
	12V 40 RPM Worm Gear Motor	1	$18.08	$18.08	https://www.amazon.com/dp/B08JQLFTPZ?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1		
	12V 40 RPM Worm Gear Motor	1	$17.19	$17.19	https://www.amazon.com/dp/B08BL9ZC3P?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1		
2100-0006-0100	6mm Shaft (Stainless Steel, 100mm Length)	3	$2.59	$7.77	https://www.gobilda.com/6mm-shaft-stainless-steel-100mm-length/?srsltid=AfmBOorxxbo98PnW6rRsJMckmXdTZ_RlSA2M4XFKAmgHhCrAJq3lIKtY		
2304-0006-0030	2304 Series Brass, MOD 0.8 Pinion Gear (6mm Bore, 30 Tooth)	2	$7.99	$15.98	https://www.gobilda.com/2304-series-brass-mod-0-8-pinion-gear-6mm-bore-30-tooth/?srsltid=AfmBOop-dxXBSDmPbPhp0OXtXp3rDEntxEYdmdwv8hieVTNGURQ7cAWl		
2317-0006-0024	2317 Series MOD 0.8 Steel Miter Gear (Set-Screw, 6mm Round Bore, 24 Tooth)	2	$9.99	$19.98	https://www.gobilda.com/2317-series-mod-0-8-steel-miter-gear-set-screw-6mm-round-bore-24-tooth/?srsltid=AfmBOoqxOSlahsjJY_m0GQ4JMvGlvlh0bQC5LHp_1C2X2mJMsVp7GliS		

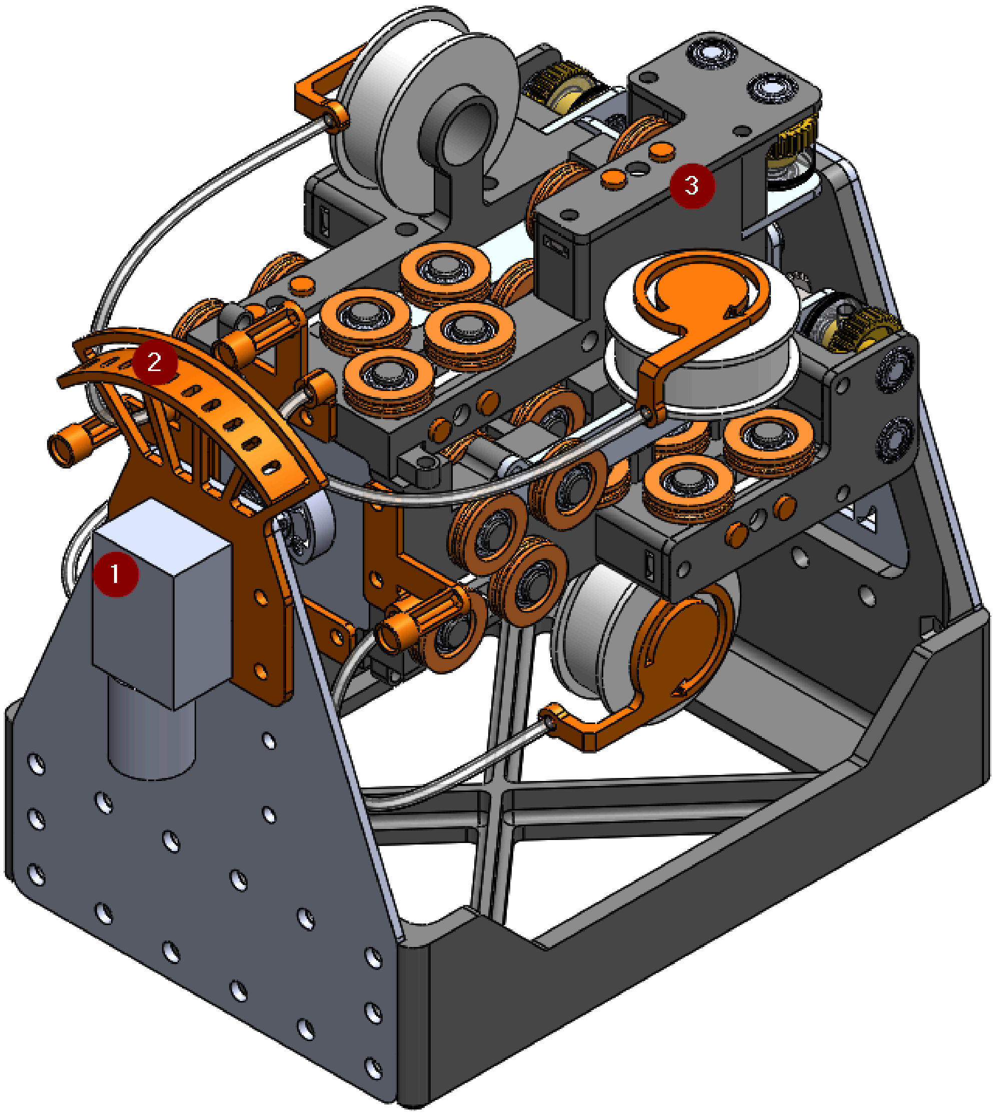
Color switching + Electronics & Hardware:



PN/SKU	Description	Vendor	Req. Qty	Order Qty	Unit of measure	Cost per	Total cost	Notes	Link
2317-0006-0024	2317 Series MOD 0.8 Steel Miter Gear (Set-Screw, 6mm Round Bore	goBILDA	5	5	Each	$9.99	$49.95		https://www.gobilda.com/2317-series-mod-0-8-steel-miter-gear-set-screw-6mm-round-bore-24-tooth/
2304-0006-0030	2304 Series Brass, MOD 0.8 Pinion Gear (6mm Bore, 30 Tooth)	goBILDA	8	8	Each	$7.99	$63.92		https://www.gobilda.com/2304-series-brass-mod-0-8-pinion-gear-6mm-bore-30-tooth/
1309-0016-4008	1309 Series Sonic Hub (8mm REX™ Bore)	goBILDA	2	2	Each	$7.99	$15.98		https://www.gobilda.com/1309-series-sonic-hub-8mm-rex-bore/
1611-0514-0006	1611 Series Flanged Ball Bearing (6mm ID x 14mm OD, 5mm Thickn	goBILDA	49	25	Pack of 2	$3.99	$99.75		https://www.gobilda.com/1611-series-flanged-ball-bearing-6mm-id-x-14mm-od-5mm-thickness-2-pack/
1611-0514-4008	1611 Series Flanged Ball Bearing (8mm REX™ ID x 14mm OD, 5mm	goBILDA	2	1	Pack of 2	$5.99	$5.99		https://www.gobilda.com/1611-series-flanged-ball-bearing-8mm-rex-id-x-14mm-od-5mm-thickness-2-pack/
1516-4008-2160	1516 Series 8mm REX™ Standoff (M4 x 0.7mm Threads, 216mm Ler	goBILDA	1	1	Pack of 4	$16.99	$16.99		https://www.gobilda.com/1516-series-8mm-rex-standoff-m4-x-0-7mm-threads-216mm-length-4-pack/
4007-1006-4008	4007 Series Hyper Coupler (6mm D-Bore to 8mm REX Bore™)	goBILDA	1	1	Each	$8.99	$8.99		https://www.gobilda.com/4007-series-hyper-coupler-6mm-d-bore-to-8mm-rex-bore/
89015K28	Multipurpose 6061 Aluminum Sheet 1/8" Thick, 12" x 24"	McMaster	1	1	Each	$49.81	$49.81		https://www.mcmaster.com/89015K28/
90480A011	Low-Strength Steel Hex Nut Zinc-Plated, 10-24 Thread Size	McMaster	20	1	Pack of 100	$2.33	$2.33		https://www.mcmaster.com/90480A011/
95462A505	Medium-Strength Steel Hex Nut Grade 5, Zinc-Plated, 1/4"-28 Thread	McMaster	4	1	Pack of 100	$8.00	$8.00		https://www.mcmaster.com/95462A505/
92865A010	Medium-Strength Grade 5 Steel Hex Head Screw Zinc-Plated, 1/4"-28	McMaster	4	1	Pack of 100	$14.86	$14.86		https://www.mcmaster.com/92865A010/
-	22 awg Wire Solid Core Hookup Wires-6 Different Colored Breadboar	Amazon	4	1	Pack of 6	$15.19	$15.19		https://www.amazon.com/gp/product/B07TX6BX47/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
-	DC Worm Gear Motor 12V High Reduction with Encoder Srong Self-L	Amazon	1	1	Each	$18.02	$18.02		https://www.amazon.com/dp/B073S61PYP?ref=ppx_yo2ov_dt_b_fed_asin_title
-	ALITOVE DC 12V 5A Power Supply Adapter Converter Transformer A	Amazon	1	1	Each	$11.99	$11.99	Sale price!	https://www.amazon.com/dp/B01GEA8PQA?ref=ppx_yo2ov_dt_b_fed_asin_title
-	uxcell 6mm x 150mm 304 Stainless Steel Solid Round Rod for DIY Cr	Amazon	2	1	Pack of 2	$6.49	$6.49		https://www.amazon.com/dp/B082ZP4HJ1?ref=ppx_yo2ov_dt_b_fed_asin_title
-	WAGO 221 Lever Nuts 28pc Compact Splicing Wire Connector Assor	Amazon	1	1	Each	$20.95	$20.95		https://www.amazon.com/dp/B0CJ5QF3VX?ref=ppx_yo2ov_dt_b_fed_asin_title
-	10 Pcs Lock Collar 6mm Shaft Lock Collar T6 Lead Screw Lock Ring	Amazon	4	1	Pack of 10	$8.99	$8.99		https://www.amazon.com/dp/B0B1C4XX65?ref=ppx_yo2ov_dt_b_fed_asin_title
-	380 Pcs M4 Screw Nuts and Bolts Assortment Kit, 304 Stainless Stee	Amazon	1	1	Pack of 380	$9.99	$9.99		https://www.amazon.com/dp/B0CQQQ359H?ref=ppx_yo2ov_dt_b_fed_asin_title
-	20Pcs 49E OH49E SS49E S49E TO-92S Hall Effect Sensor 3Pins Ma	Amazon	1	1	Pack of 20	$7.99	$7.99		https://www.amazon.com/dp/B0CZ6RL4B2?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1
-	TRYMAG Small Strong Magnets, 6 Different Size, 255Pcs Rare Earth	Amazon	4	1	Pack of 255	$14.99	$14.99		https://www.amazon.com/dp/B09WZTSQ9Y?ref=ppx_yo2ov_dt_b_fed_asin_title
-	10-24 x 3/8", 1/2", 5/8", 3/4" and 1" Button Head Socket Cap Screws S	Amazon	1	1	Pack of 100	$9.90	$9.90		https://www.amazon.com/gp/product/B0872LR35V/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
-	Compression Springs Assortment Kit, 390 Pcs 24 Different Sizes Stai	Amazon	8	1	Pack of 390	$15.99	$15.99		https://www.amazon.com/gp/product/B0BBQL2SN3/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
						Total	$477.06		

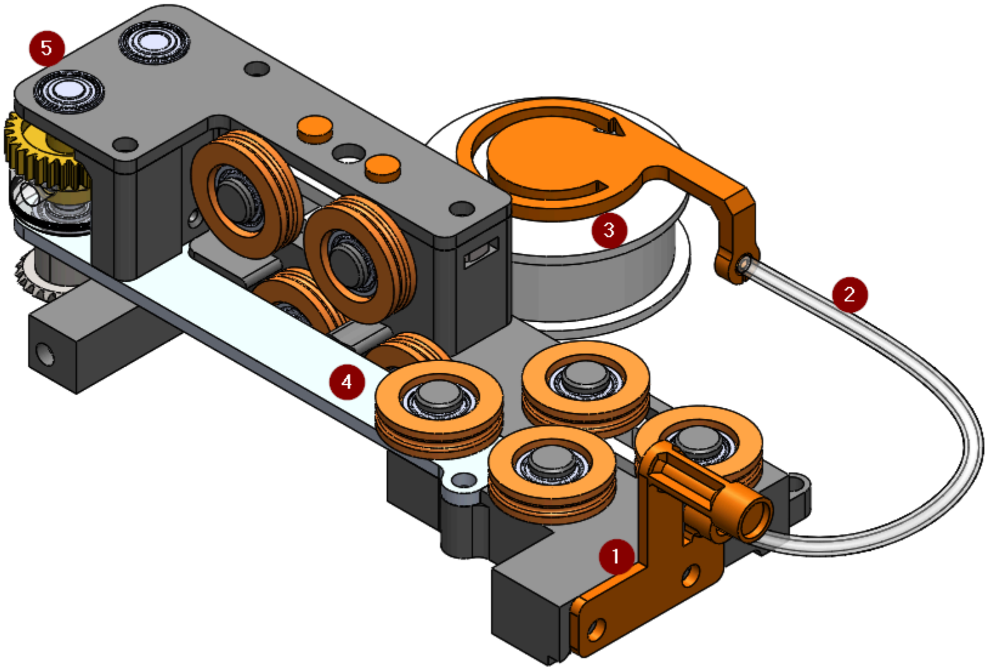
Appendix B

Color Switching:



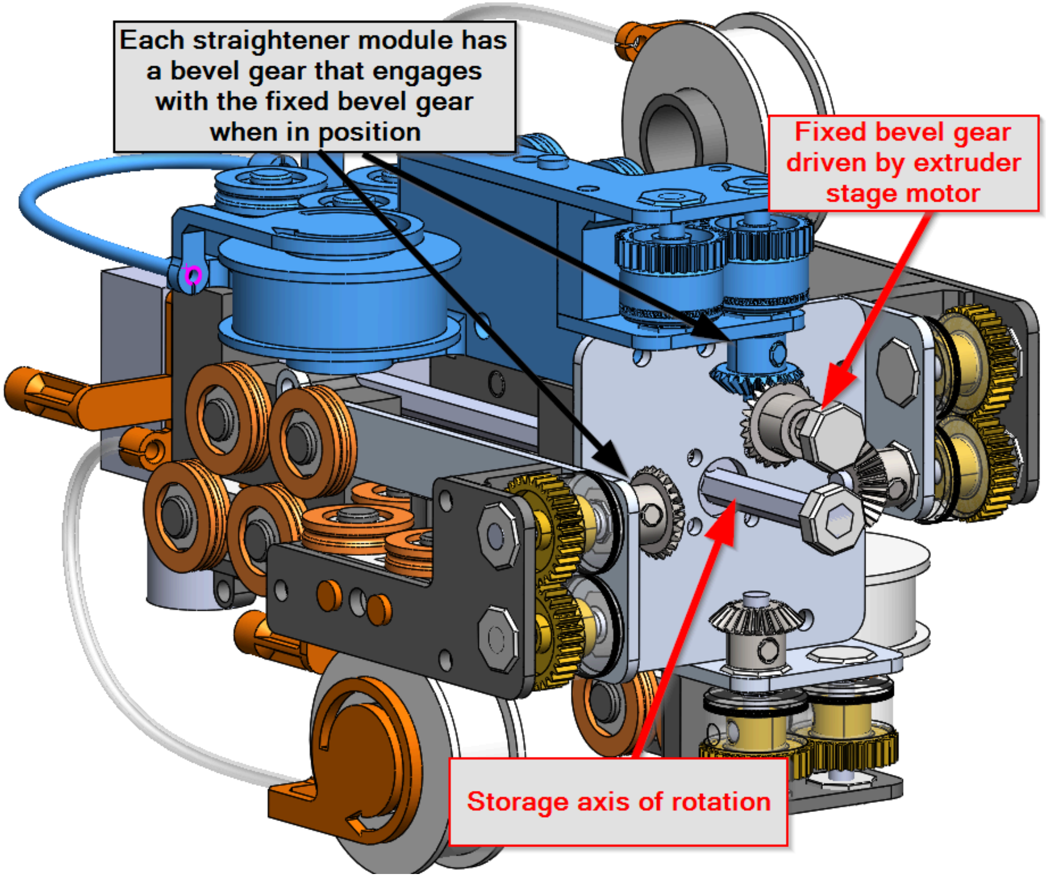
#	Description
1	Motor
2	Variable position hall effect sensor mount
3	Straightener module

Wire Straightener Module

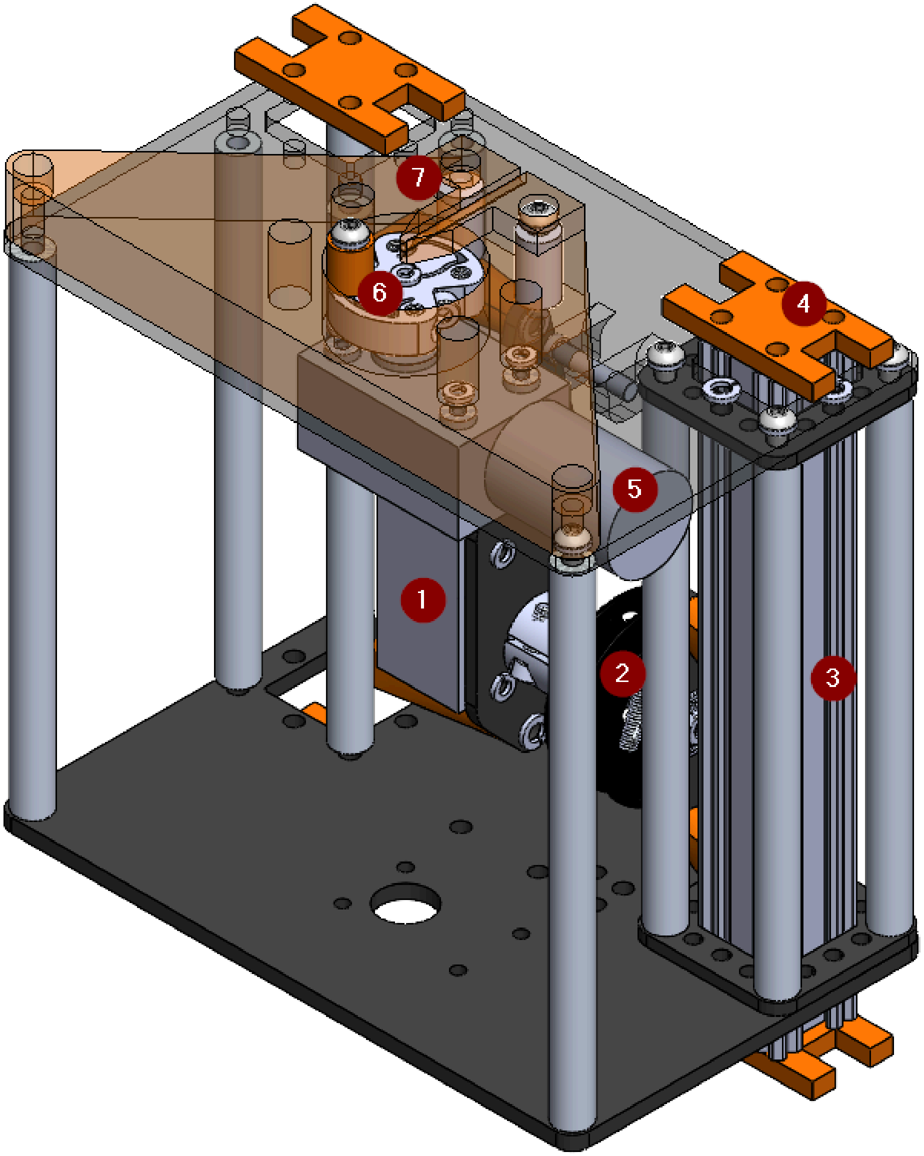


#	Description
1	Magnet mount for hall effect position sensing
2	PTFE tube for consistent wire routing and backfeeding into the spool
3	Wire spool holder
4	Variable height straightener rollers to accommodate different wire gages
5	Feeder rollers

Wire Straightener Transmission

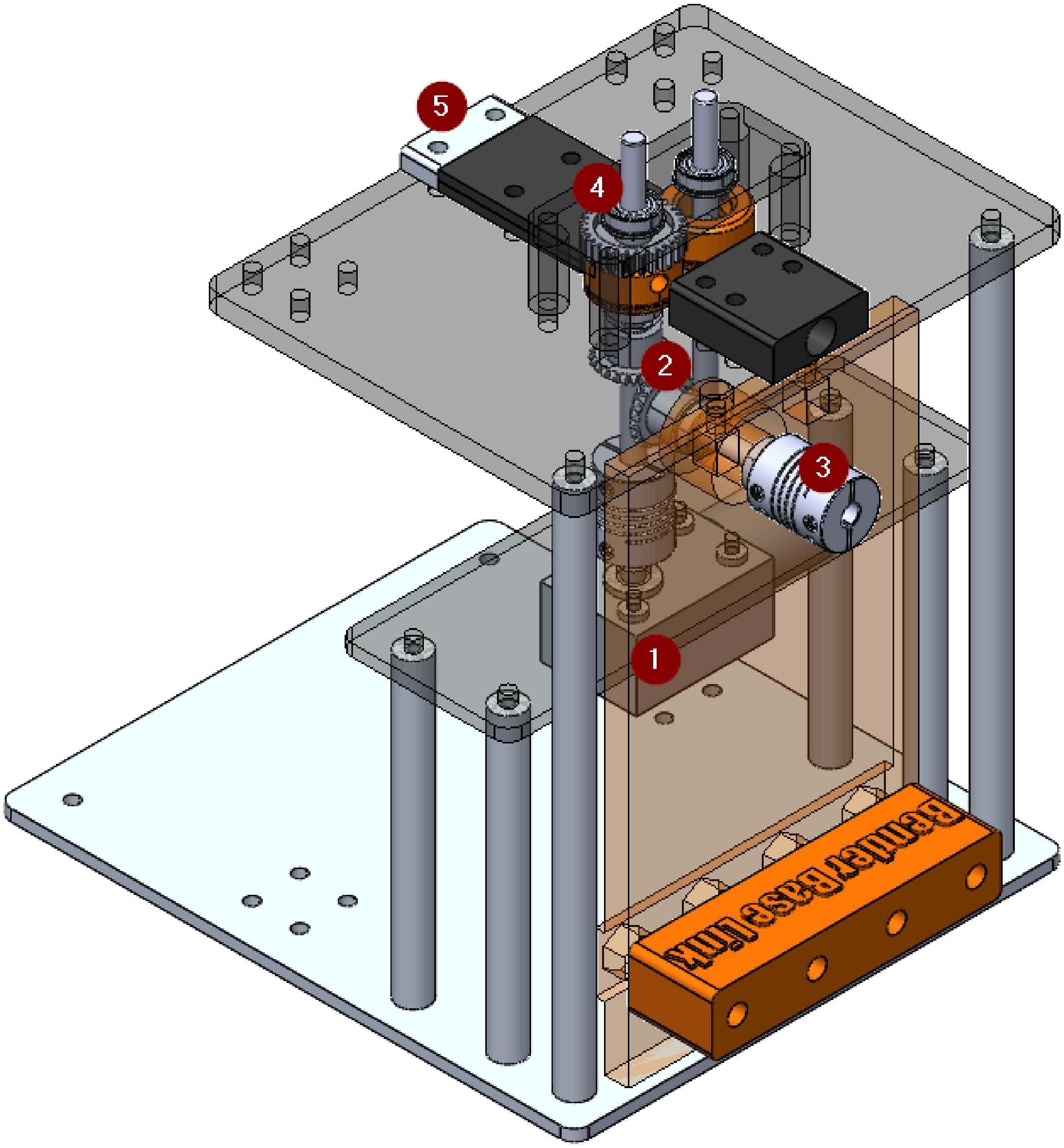

Each straightener module has a bevel gear that engages with the fixed bevel gear when in position
Fixed bevel gear driven by extruder stage motor
Storage axis of rotation

Carriage and Bending Head



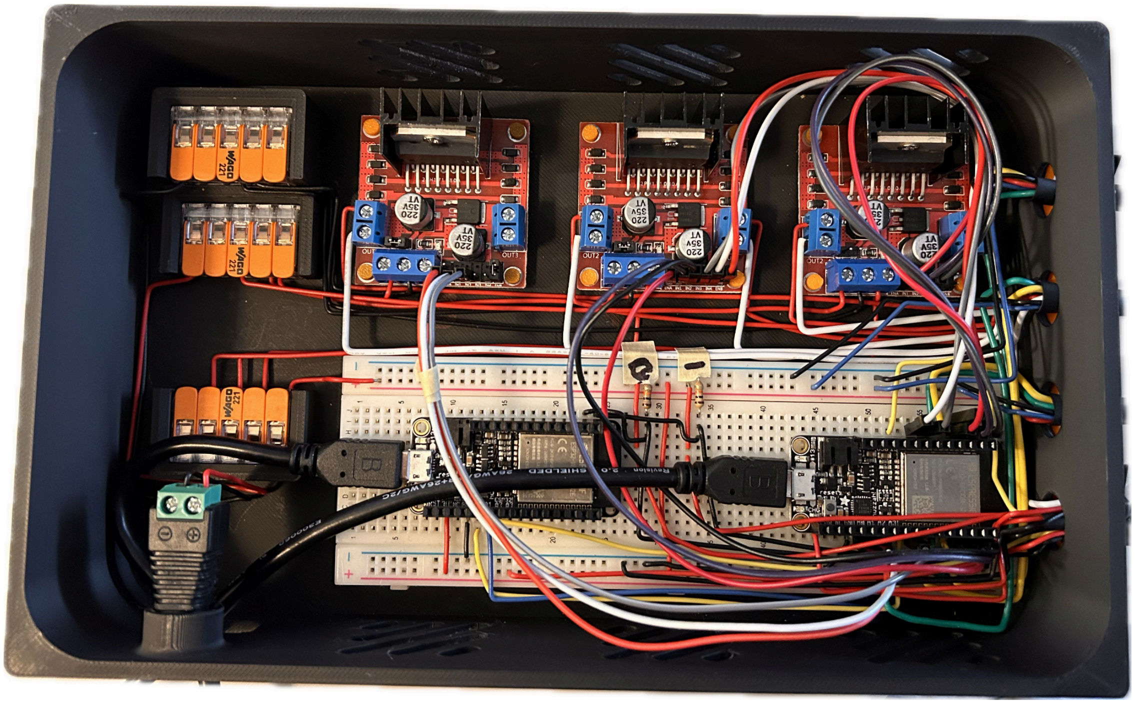
#	Description
1	Carriage motor and gearbox
2	Winch pulley for raising and lowering carriage
3	Aluminum extrusion fixed rail to interface with carriage slides
4	Hard stop
5	Bending head motor and gearbox
6	Bending head
7	Printed bending mandrel and wire rest

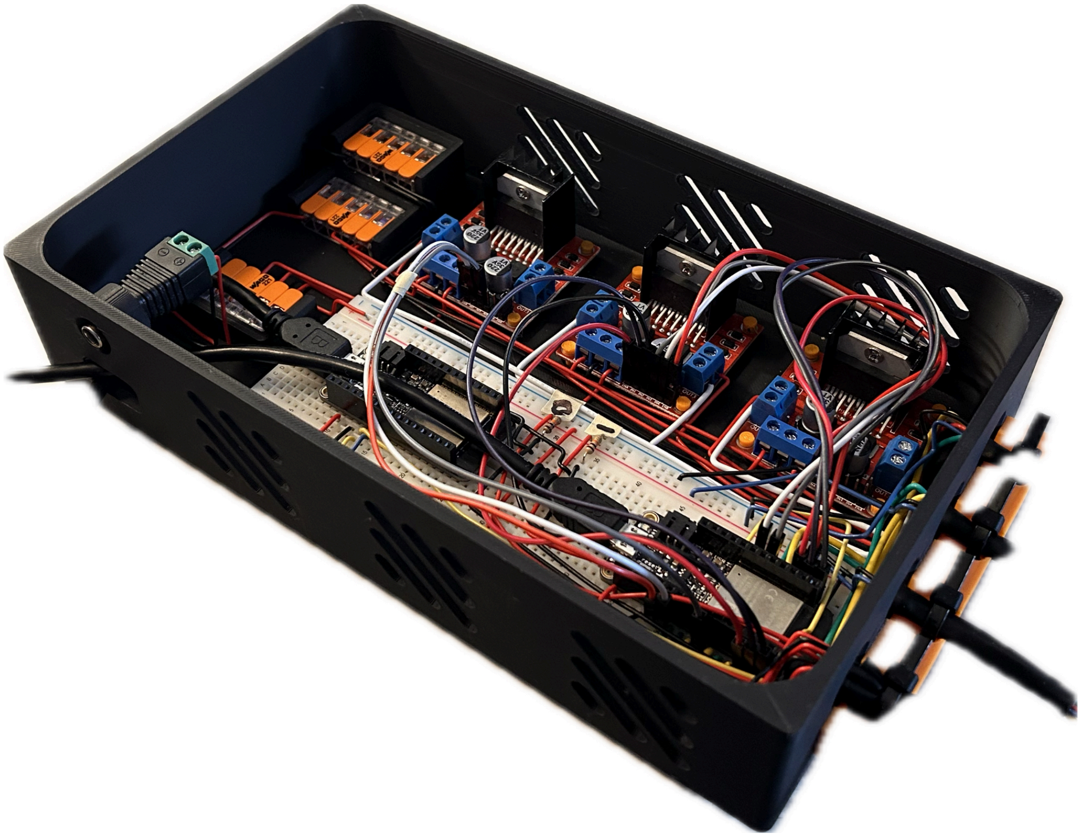
Extruder

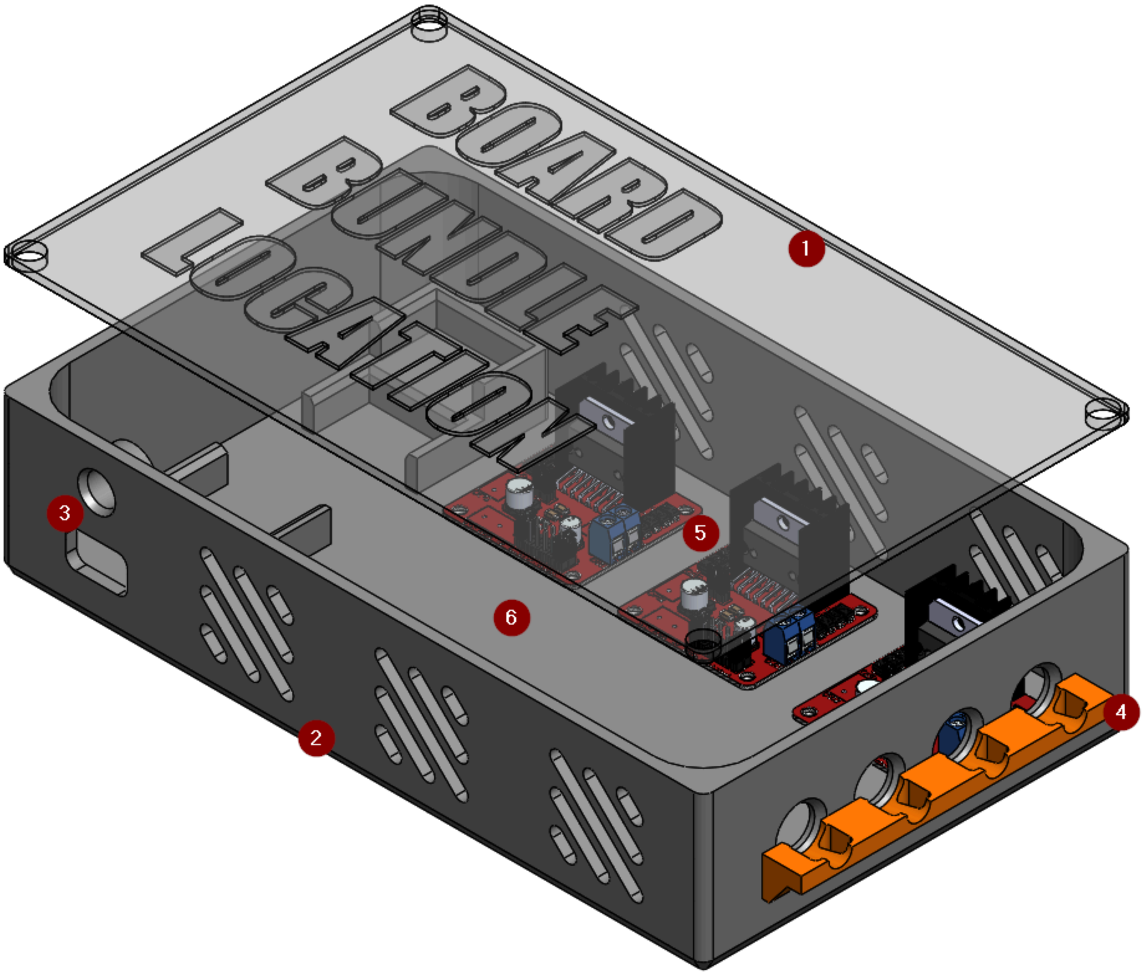


#	Description
1	Extruder motor and gearbox
2	Bevel gears between master extruder shaft and transmission to wire storage
3	Flexible shaft coupler at transmission
4	Geared transmission between extruder rollers
5	Aluminum block to shear wire against while cutting

Electronics Housing

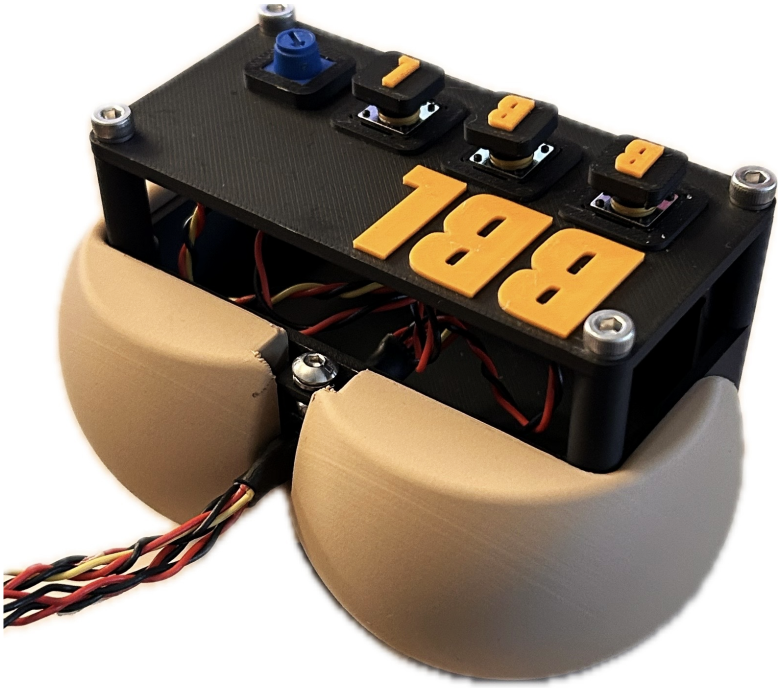


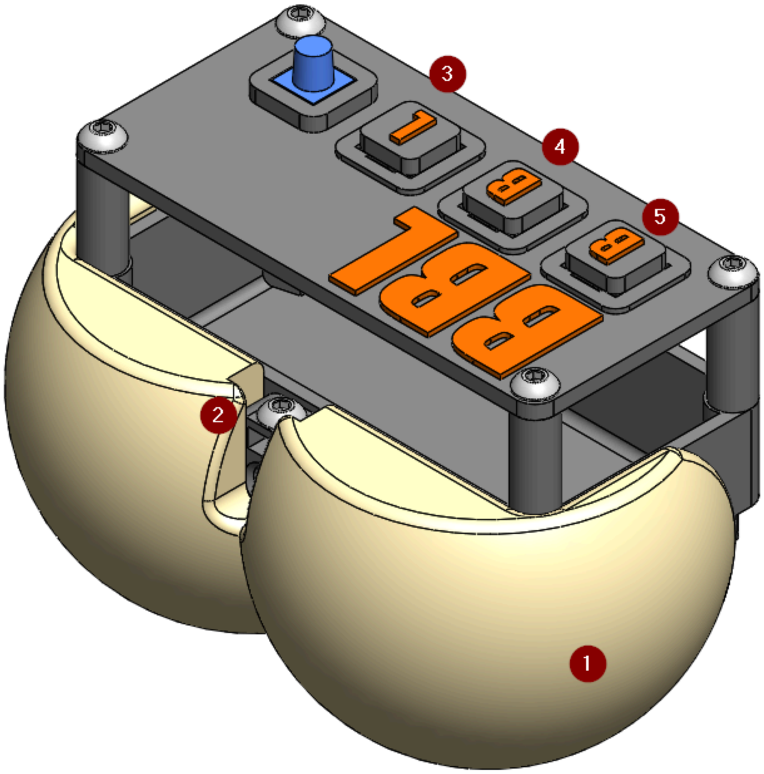




#	Description
1	Magnetic lid
2	Enclosure with embedded magnets
3	12V power plug and USB cable egress
4	Platform with zip tie holes for strain relieving motor cables
5	LN298N Motor drivers
6	Breadboard

Controller:





#	Description
1	Ergonomic grip for controller
2	Controller cable strain relief via clamping on conduit
3	Action button
4	State transition button
5	Basic_control breadboard reset button (for ease of access to run zeroing routine)

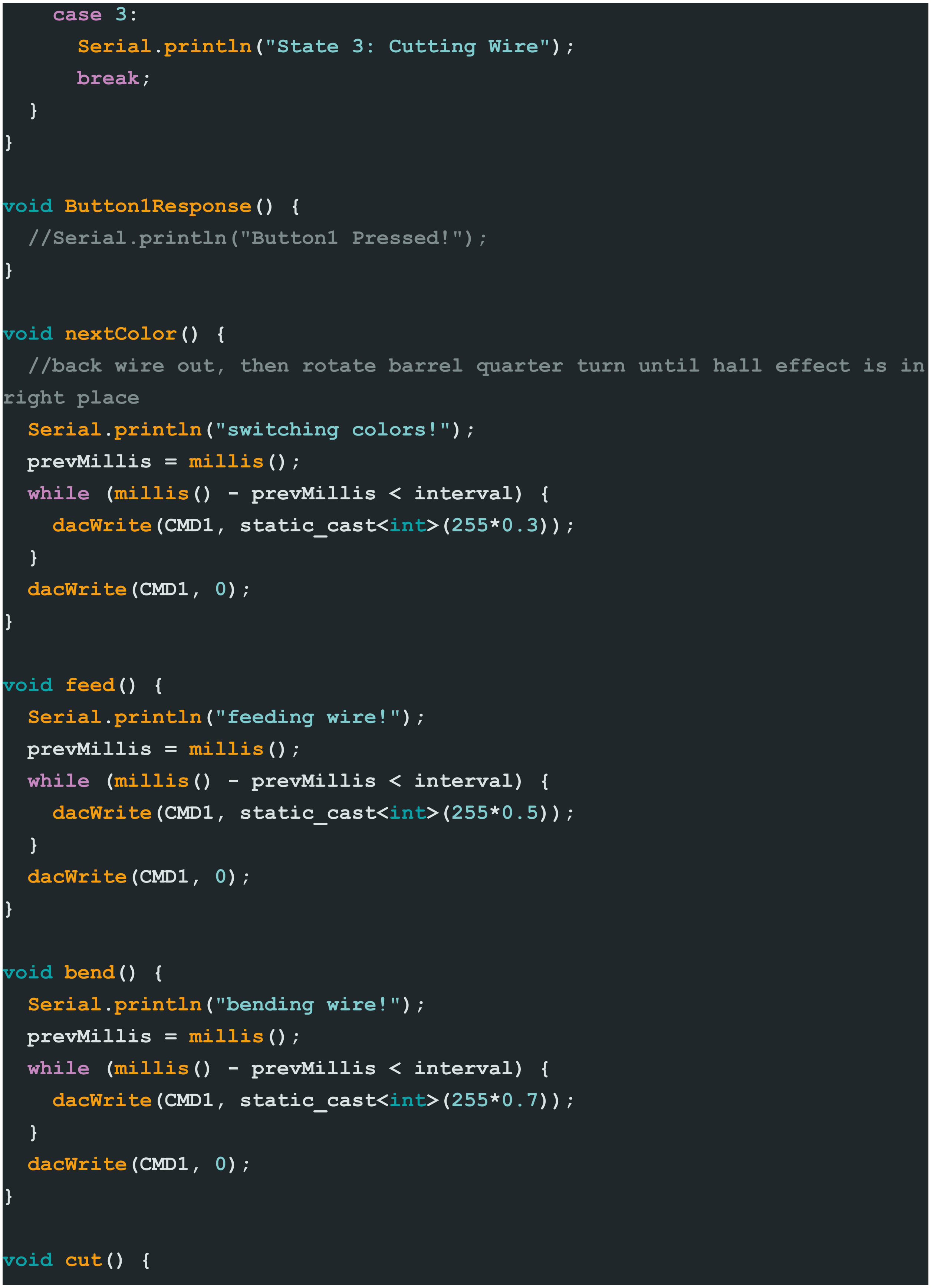
Appendix C

State Machine Board:

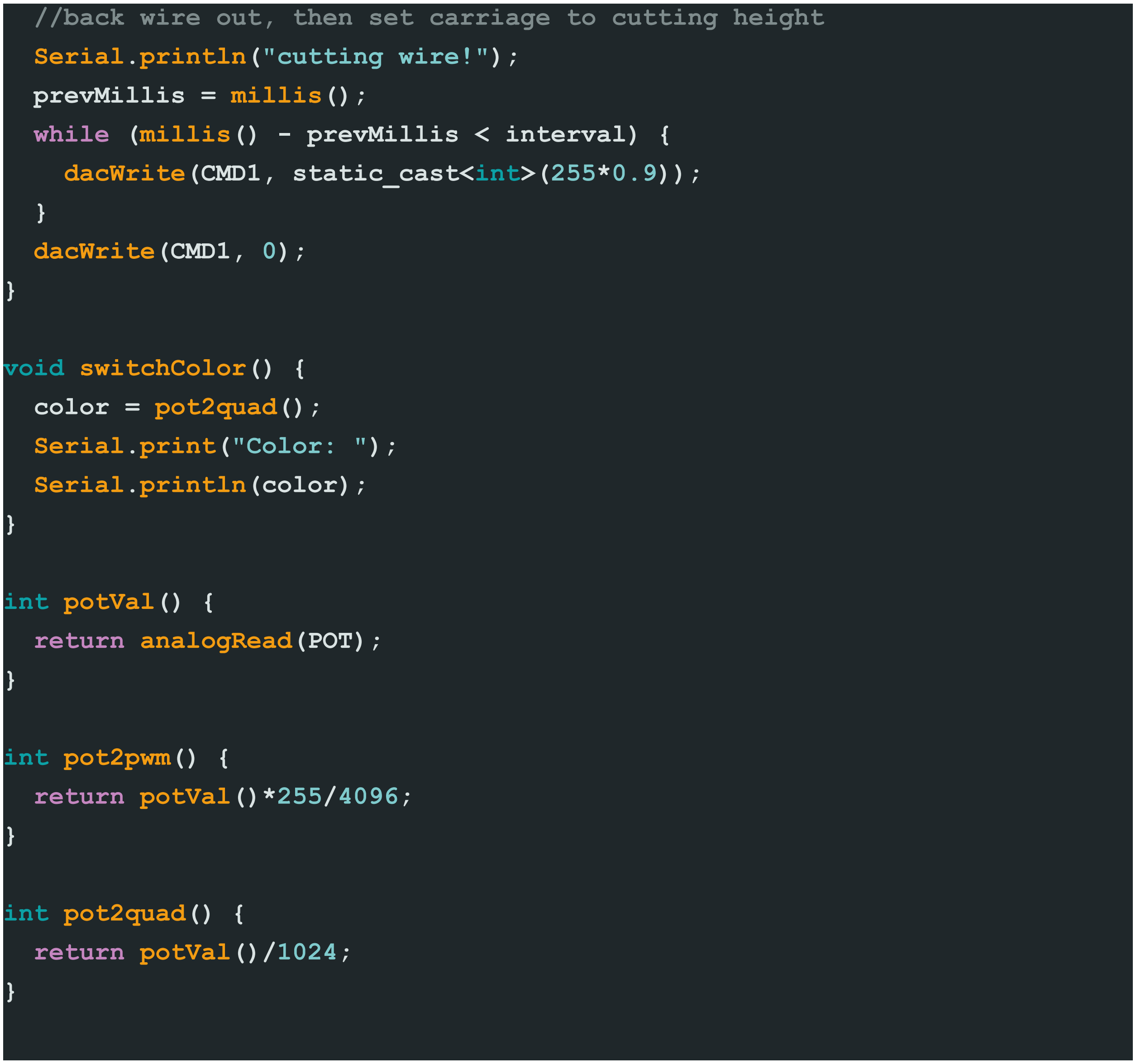
```
#include <ESP32Encoder.h>
ESP32Encoder encoder;
#define ENA 27
#define IN1 33
#define IN2 15
#define ENCA 32
#define ENCB 14
#define BTN0 22
#define BTN1 23
#define POT 26

#define CMD1 25
#define CMD2 34

int encval = 3260;
int quarterTurn = 805;
byte state = 0;
int color = 0;
int prevMillis = 0;
const int interval = 1000;
volatile bool DEBOUNCINGflag = false;
volatile bool button0IsPressed = false;
volatile bool button1IsPressed = false;
hw_timer_t* timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;

//INITIALIZING INTERRUPTS
void IRAM_ATTR BUTTON0isr() {  // the function to be called when interrupt is triggered
  button0IsPressed = true;
}

void IRAM_ATTR BUTTON1isr() {  // the function to be called when interrupt is triggered
  button1IsPressed = true;
}
```

```
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  DEBOUNCINGflag = false;
  portEXIT_CRITICAL_ISR(&timerMux0);
  timerStop(timer0);
  button0IsPressed = false;
  button1IsPressed = false;
}

void setup() {
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(BTN0, INPUT);
  pinMode(BTN1, INPUT);
  pinMode(POT, INPUT);

  Serial.begin(115200);

  attachInterrupt(BTN0, BUTTON0isr, RISING);
  attachInterrupt(BTN1, BUTTON1isr, RISING);
  timer0 = timerBegin(1000000);
  timerAttachInterrupt(timer0, &onTime0);
  timerAlarm(timer0, 500000, true, 0);
  // ESP32Encoder::useInternalWeakPullResistors = puType::up;
  // encoder.attachHalfQuad(ENCA, ENCB);
  // encoder.setCount(0);
  // Serial.println("Encoder Start = " +
String((int32_t)encoder.getCount()));
  // fwd();

  // put your setup code here, to run once:
}

void loop() { //HIGH LEVEL STATE MACHINE

  if (CheckForButton0Press()) {
    Button0Response();
  }
  analogWrite(CMD2, pot2pwm());
```

```
switch (state) {
  case 0: //CHANGE WIRE COLORS
    if (CheckForButton1Press()) {
      Button1Response();
      nextColor();
    }
    if (CheckForButton0Press()) {
      state = 1;
      Button0Response();
    }
    break;
  case 1: //FEED WIRE
    if (CheckForButton1Press()) {
      Button1Response();
      feed();
    }
    if (CheckForButton0Press()) {
      state = 2;
      Button0Response();
    }
    break;
  case 2: //BEND WIRE
    if (CheckForButton1Press()) {
      Button1Response();
      bend();
    }
    if (CheckForButton0Press()) {
      state = 3;
      Button0Response();
    }
    break;
  case 3: //CUT WIRE
    if (CheckForButton1Press()) {
      Button1Response();
      cut();
    }
    if (CheckForButton0Press()) {
      state = 0;
      Button0Response();
      }
      break;
  }
  // put your main code here, to run repeatedly:
}

//MEDIUM LEVEL EVENT CHECKERS AND SERVICES
bool CheckForButton0Press() {
  if (button0IsPressed && !DEBOUNCINGflag) {
    portENTER_CRITICAL_ISR(&timerMux0);
    DEBOUNCINGflag = true;
    portEXIT_CRITICAL_ISR(&timerMux0);
    timerStart(timer0);
    return true;
  } else
    return false;
}

bool CheckForButton1Press() {
  if (button1IsPressed && !DEBOUNCINGflag) {
    portENTER_CRITICAL_ISR(&timerMux0);
    DEBOUNCINGflag = true;
    portEXIT_CRITICAL_ISR(&timerMux0);
    timerStart(timer0);
    return true;
  } else
    return false;
}

void Button0Response() { //Cycle to next state
  switch (state) {
    case 0:
      Serial.println("State 0: Switching Wire Colors");
      break;
    case 1:
      Serial.println("State 1: Extruding Wire");
      break;
    case 2:
      Serial.println("State 2: Bending Wire");
      break;
```



```
    case 3:
      Serial.println("State 3: Cutting Wire");
      break;
  }
}

void Button1Response() {
  //Serial.println("Button1 Pressed!");
}

void nextColor() {
  //back wire out, then rotate barrel quarter turn until hall effect is in right place
  Serial.println("switching colors!");
  prevMillis = millis();
  while (millis() - prevMillis < interval) {
    dacWrite(CMD1, static_cast<int>(255*0.3));
  }
  dacWrite(CMD1, 0);
}

void feed() {
  Serial.println("feeding wire!");
  prevMillis = millis();
  while (millis() - prevMillis < interval) {
    dacWrite(CMD1, static_cast<int>(255*0.5));
  }
  dacWrite(CMD1, 0);
}

void bend() {
  Serial.println("bending wire!");
  prevMillis = millis();
  while (millis() - prevMillis < interval) {
    dacWrite(CMD1, static_cast<int>(255*0.7));
  }
  dacWrite(CMD1, 0);
}

void cut() {
```



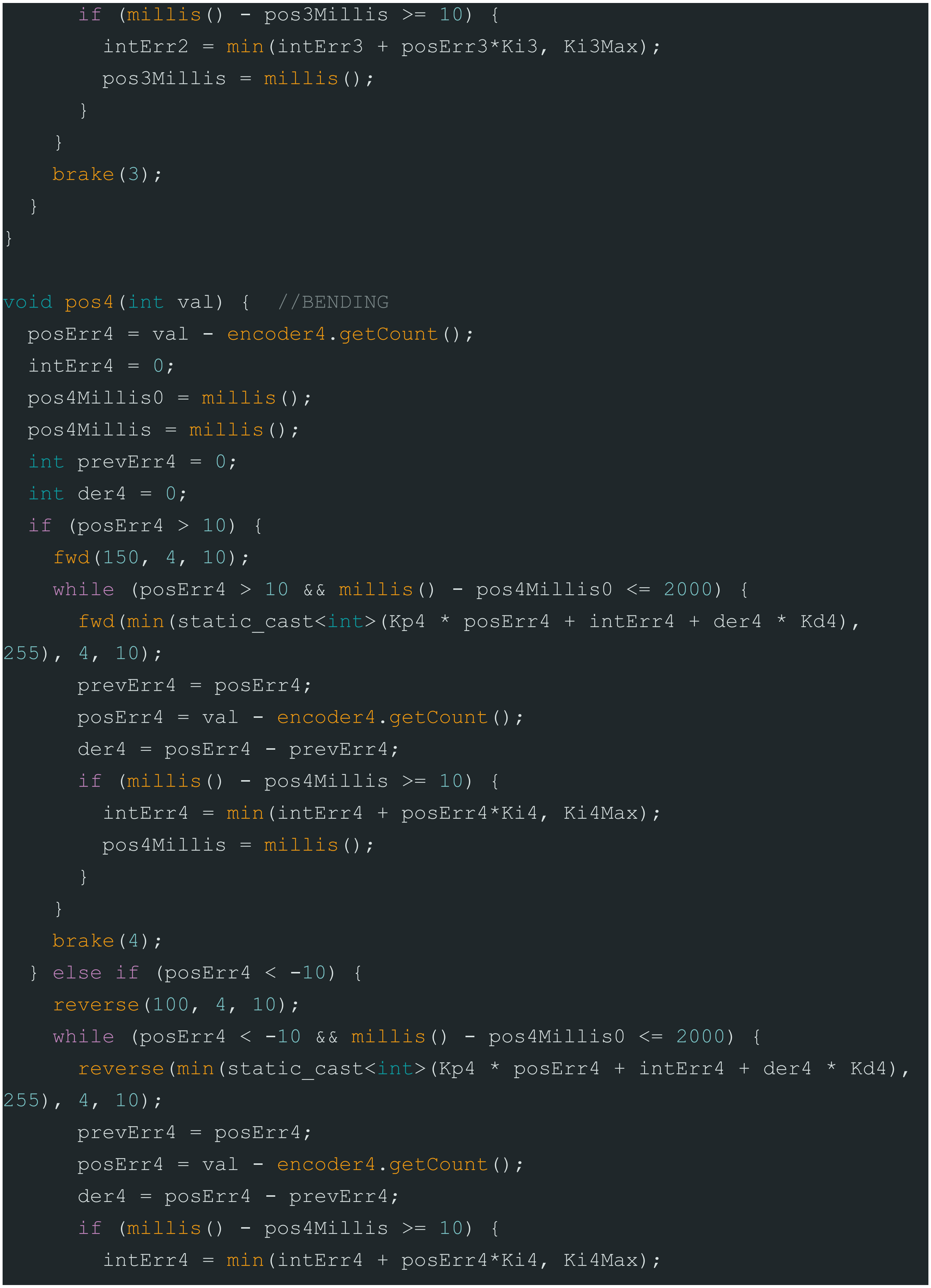
```
  //back wire out, then set carriage to cutting height
  Serial.println("cutting wire!");
  prevMillis = millis();
  while (millis() - prevMillis < interval) {
    dacWrite(CMD1, static_cast<int>(255*0.9));
  }
  dacWrite(CMD1, 0);
}

void switchColor() {
  color = pot2quad();
  Serial.print("Color: ");
  Serial.println(color);
}

int potVal() {
  return analogRead(POT);
}

int pot2pwm() {
  return potVal()*255/4096;
}

int pot2quad() {
  return potVal()/1024;
}
```

Basic Control Board

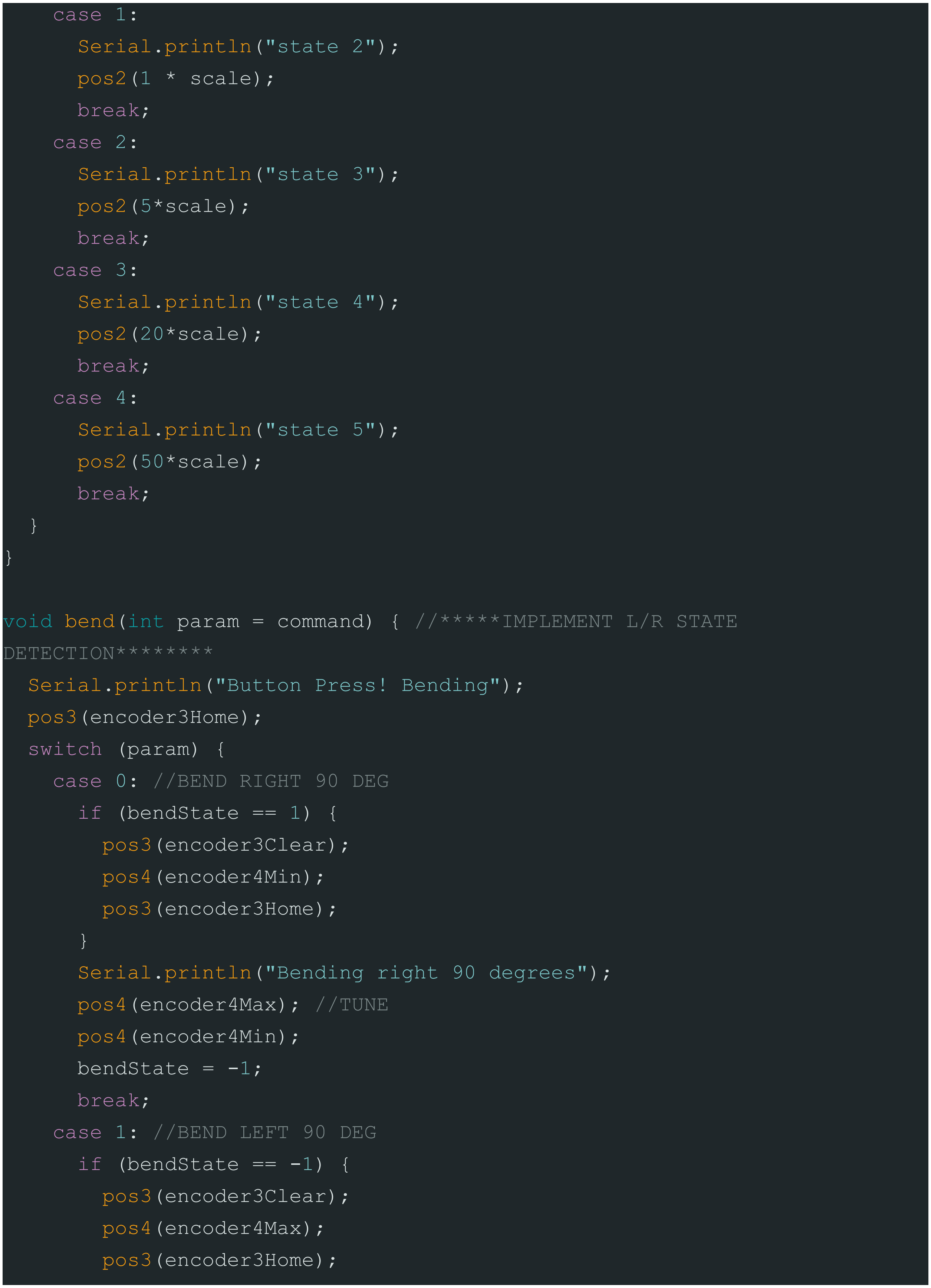
```
#include <ESP32Encoder.h>
#include <driver/adc.h>
#include <esp_timer.h>
ESP32Encoder encoder2;
ESP32Encoder encoder3;
ESP32Encoder encoder4;
//COMMAND PINS
#define CMD1 ADC1_CHANNEL_3
#define CMD2 ADC1_CHANNEL_0
//MOTOR 1 (STORAGE)
#define HALL 13
#define ENA1 14 //SAME AS ENA2!!!!!!!!!!!!!!
#define IN1A 27
#define IN1B 33
//MOTOR 2 (EXTRUDER)
#define ENC2A 15
#define ENC2B 32
#define ENA2 14
#define IN2A 22
#define IN2B 23
//MOTOR 3 (CARRIAGE)
#define ENC3A 34
#define ENC3B 5
#define ENA3 26
#define IN3A 25
#define IN3B 4
//MOTOR 4 (BENDING)
#define ENC4A 18
#define ENC4B 19
#define ENA4 16
#define IN4A 17
#define IN4B 21
int CMDSTATE1 = 0;
int CMDSTATE2 = 0;
//from 0-4 inclusive
int state = -1;
int command = -1;
int counter = 0;
//only here for reference, will replace with command logic
```

```
int analogMax = 4096;

//This is set to 4096/5 assuming that our OFF signal is 0 and our states
go from 0 - 819 - 1628 - 2442 - 3256 - 4096
const int threshold = analogMax * 0.2;

unsigned long pos2Millis = 0;
unsigned long pos2Millis0 = 0;
unsigned long pos3Millis = 0;
unsigned long pos3Millis0 = 0;
unsigned long pos4Millis = 0;
unsigned long pos4Millis0 = 0;
unsigned long fwdMillis = 0;
unsigned long revMillis = 0;
unsigned long brakeMillis = 0;
unsigned long colorMillis = 0;

int encoder3Max = 0;
int encoder3Home = 0;
int encoder3Clear = 0;
int encoder4Max = 0;
int encoder4Min = 0;

volatile int count2 = 0;
volatile int posErr2 = 0;
volatile int start2 = 0;
volatile float intErr2 = 0.0;

volatile int count3 = 0;
volatile int posErr3 = 0;
volatile float intErr3 = 0.0;

volatile int count4 = 0;
volatile int posErr4 = 0;
volatile float intErr4 = 0.0;
volatile int bendState = 0; //-1 if left side of wire, 1 if right side of
wire

volatile int hallState = 0;
volatile bool signalDetected1 = false;

volatile int prevC2 = 0;
volatile int prevC3 = 0;
volatile int prevC4 = 0;
volatile bool deltaT2 = false;
volatile bool deltaT3 = false;
volatile bool deltaT4 = false;

//TUNE THESE!!!!!!!!!!!!!!!!!!!!!!!
float Kp2 = 6;
float Kp3 = 8;
float Kp4 = 0.3;
float Ki2 = 0.3;
float Ki3 = 0.3;
float Ki4 = 0.4;
float Ki2Max = 100;
float Ki3Max = 100;
float Ki4Max = 100;
float Kd2 = 10;
float Kd3 = 20;
float Kd4 = 5;

hw_timer_t* timer2 = NULL;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;
hw_timer_t* timer3 = NULL;
portMUX_TYPE timerMux3 = portMUX_INITIALIZER_UNLOCKED;
hw_timer_t* timer4 = NULL;
portMUX_TYPE timerMux4 = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR signalInterrupt1() {
  // Your interrupt handler logic for ADC_PIN_1
  Serial.println("Signal detected on PIN 1!, state: " + String(state) + "
command is: " + String(command));

  logic();
}

//SPEED COUNTER TIMER INTERRUPT
void IRAM_ATTR onTime2() {
  portENTER_CRITICAL_ISR(&timerMux2);
  count2 = encoder2.getCount() - prevC2;
  deltaT2 = true;
  prevC2 = encoder2.getCount();
  portEXIT_CRITICAL_ISR(&timerMux2);
}
void IRAM_ATTR onTime3() {
  portENTER_CRITICAL_ISR(&timerMux3);
  count3 = encoder3.getCount() - prevC3;
  deltaT3 = true;
  prevC3 = encoder3.getCount();
  portEXIT_CRITICAL_ISR(&timerMux3);
}
void IRAM_ATTR onTime4() {
  portENTER_CRITICAL_ISR(&timerMux4);
  count4 = encoder4.getCount() - prevC4;
  deltaT4 = true;
  prevC4 = encoder4.getCount();
  portEXIT_CRITICAL_ISR(&timerMux4);
}

void setup() {
  Serial.begin(115200);
  pinMode(ENA1, OUTPUT);
  pinMode(IN1A, OUTPUT);
  pinMode(IN1B, OUTPUT);
  pinMode(HALL, INPUT);

  pinMode(ENA2, OUTPUT);
  pinMode(IN2A, OUTPUT);
  pinMode(IN2B, OUTPUT);
  pinMode(ENC2A, INPUT);
  pinMode(ENC2B, INPUT);

  pinMode(ENA3, OUTPUT);
  pinMode(IN3A, OUTPUT);
  pinMode(IN3B, OUTPUT);
  pinMode(ENC3A, INPUT);
  pinMode(ENC3B, INPUT);
  pinMode(ENA4, OUTPUT);
  pinMode(IN4A, OUTPUT);
  pinMode(IN4B, OUTPUT);
  pinMode(ENC4A, INPUT);
  pinMode(ENC4B, INPUT);

  pinMode(CMD1, INPUT);
  pinMode(CMD2, INPUT);

  //Initialize ADC
  adc1_config_width(ADC_WIDTH_BIT_12);
  adc1_config_channel_atten(CMD1, ADC_ATTEN_DB_11);
  adc1_config_channel_atten(CMD2, ADC_ATTEN_DB_11);
  // Timer for periodic ADC checks
  esp_timer_create_args_t timer_args;
  timer_args.callback = [](void*) {
    int val1 = adc1_get_raw(CMD1);
    int val2 = adc1_get_raw(CMD2);
    state = val1 / 819 - 1;
    command = val2 / 819 - 1;

    // Check thresholds and trigger interrupts
    if (val1 > threshold && !signalDetected1) {
      signalDetected1 = true;
      signalInterrupt1();
    } else if (val1 <= threshold) {
      signalDetected1 = false;
    }
  };

  esp_timer_handle_t adc_timer;
  esp_timer_create(&timer_args, &adc_timer);
  esp_timer_start_periodic(adc_timer, 10000);  // Check ADC every 10 ms

  //only here for reference/compiling, will replace

  ESP32Encoder::useInternalWeakPullResistors = puType::up;
  encoder2.attachHalfQuad(ENC2A, ENC2B);
  encoder2.setCount(0);
  encoder3.attachHalfQuad(ENC3A, ENC3B);
```

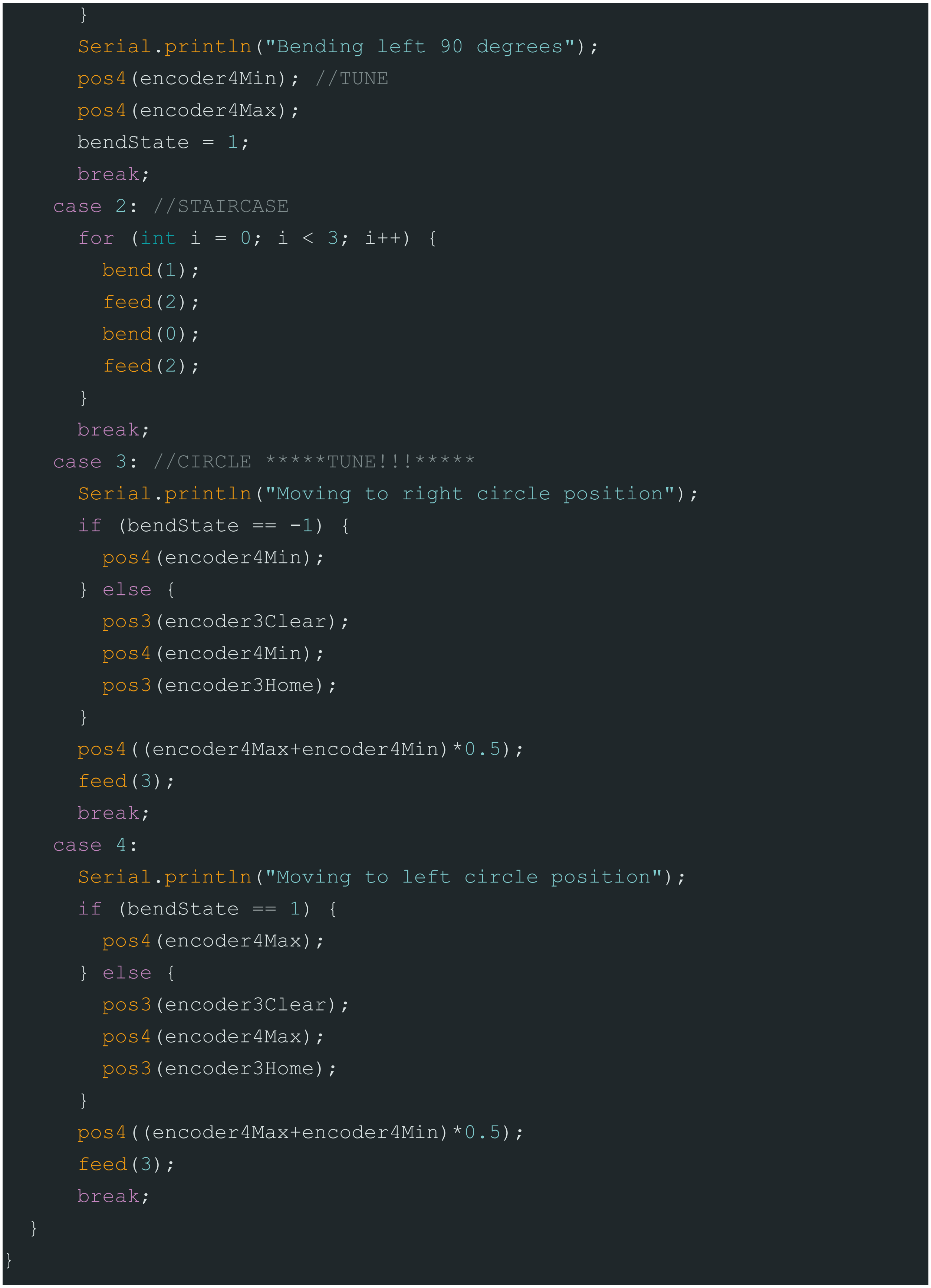
```
  encoder3.setCount(0);
  encoder4.attachHalfQuad(ENC4A, ENC4B);
  encoder4.setCount(0);

  timer2 = timerBegin(1000000);             // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer2, &onTime2);   // Attach onTimer1 function to our timer.
  timerAlarm(timer2, 10000, true, 0);       //10ms, autoreload true

  timer3 = timerBegin(1000000);             // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer3, &onTime3);   // Attach onTimer1 function to our timer.
  timerAlarm(timer3, 10000, true, 0);       //10ms, autoreload true

  timer4 = timerBegin(1000000);             // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer4, &onTime4);   // Attach onTimer1 function to our timer.
  timerAlarm(timer4, 10000, true, 0);       //10ms, autoreload true

  zero(); //CALIBRATE EVERYTHING

}

void fwd(int PWM, int motor, int duration) {
  switch (motor) {
    case 1:
      digitalWrite(IN1A, LOW);
      digitalWrite(IN1B, HIGH);
      analogWrite(ENA1, PWM);
      break;
    case 2:
      digitalWrite(IN2A, LOW);
      digitalWrite(IN2B, HIGH);
      analogWrite(ENA2, PWM);
      break;
    case 3:
      digitalWrite(IN3A, LOW);
      digitalWrite(IN3B, HIGH);
      analogWrite(ENA3, PWM);
      break;
    case 4:
      digitalWrite(IN4A, LOW);
      digitalWrite(IN4B, HIGH);
      analogWrite(ENA4, PWM);
      break;
  }
  delay(duration);
}

void reverse(int PWM, int motor, int duration) {
  switch (motor) {
    case 1:
      digitalWrite(IN1A, HIGH);
      digitalWrite(IN1B, LOW);
      analogWrite(ENA1, PWM);
      break;
    case 2:
      digitalWrite(IN2A, HIGH);
      digitalWrite(IN2B, LOW);
      analogWrite(ENA2, PWM);
      break;
    case 3:
      digitalWrite(IN3A, HIGH);
      digitalWrite(IN3B, LOW);
      analogWrite(ENA3, PWM);
      break;
    case 4:
      digitalWrite(IN4A, HIGH);
      digitalWrite(IN4B, LOW);
      analogWrite(ENA4, PWM);
      break;
  }
  delay(duration);
}

void brake(int motor) {
  switch (motor) {
  case 1:
    digitalWrite(IN1A, LOW);
    digitalWrite(IN1B, LOW);
    analogWrite(ENA1, LOW);
    break;
  case 2:
    digitalWrite(IN2A, LOW);
    digitalWrite(IN2A, LOW);
    analogWrite(ENA2, LOW);
    break;
  case 3:
    digitalWrite(IN3A, LOW);
    digitalWrite(IN3B, LOW);
    analogWrite(ENA3, LOW);
    break;
  case 4:
    digitalWrite(IN4A, LOW);
    digitalWrite(IN4B, LOW);
    analogWrite(ENA4, LOW);
    break;
  }
  delay(10);
}

void check2() {
  if (deltaT2) {
    portENTER_CRITICAL(&timerMux2);
    deltaT2 = false;
    portEXIT_CRITICAL(&timerMux2);
    Serial.println(count2);
  }
}

void check3() {
  if (deltaT3) {
    portENTER_CRITICAL(&timerMux3);
    deltaT3 = false;
    portEXIT_CRITICAL(&timerMux3);
    Serial.println(count3);
  }
}

void check4() {
  if (deltaT4) {
    portENTER_CRITICAL(&timerMux4);
    deltaT4 = false;
    portEXIT_CRITICAL(&timerMux4);
    Serial.println(count4);
  }
}

void pos2(int val) {  //EXTRUDER
  posErr2 = val;
  start2 = encoder2.getCount();
  intErr2 = 0;
  pos2Millis0 = millis();
  pos2Millis = millis();
  int prevErr2 = 0;
  int der2 = 0;
  if (posErr2 > 15) {
    while (posErr2 > 15 && millis() - pos2Millis0 <= val + 2000) { //time
out based on val + 2 seconds
      fwd(min(static_cast<int>(Kp2 * posErr2 + intErr2 + der2 * Kd2),
255), 2, 10);
      prevErr2 = posErr2;
      posErr2 = val + start2 - encoder2.getCount();
      der2 = posErr2 - prevErr2;
      if (millis() - pos2Millis >= 10) {
        intErr2 = min(intErr2 + posErr2*Ki2, Ki2Max);
        pos2Millis = millis();
      }
    }
    brake(2);
  } else if (posErr2 < -15) {
    while (posErr2 < -15 && millis() - pos2Millis0 <= abs(posErr2) + 2000)
{
      reverse(min(static_cast<int>(Kp2 * abs(posErr2) + abs(intErr2) +
der2 * Kd2), 255), 2, 10);
      prevErr2 = posErr2;
      posErr2 = val + start2 - encoder2.getCount();
      der2 = posErr2 - prevErr2;
      if (millis() - pos2Millis >= 10) {
        intErr2 = max(intErr2 + posErr2*Ki2, -Ki2Max);
        pos2Millis = millis();
      }
    }
    brake(2);
  }
}

void pos3(int val) {  //CARRIAGE
  posErr3 = val - encoder3.getCount();
  intErr3 = 0;
  pos3Millis0 = millis();
  pos3Millis = millis();
  int prevErr3 = 0;
  int der3 = 0;
  if (posErr3 > 50) {
    fwd(100, 3, 10);
    while (posErr3 > 50 && millis() - pos3Millis0 <= 2000) { //time out
after 2 seconds
      fwd(min(static_cast<int>(Kp3 * posErr3 + intErr3 + der3 * Kd3),
255), 3, 10);
      prevErr3 = posErr3;
      posErr3 = val - encoder3.getCount();
      der3 = posErr3 - prevErr3;
      Serial.println(encoder3.getCount());
      if (millis() - pos3Millis >= 10) {
        intErr2 = min(intErr3 + posErr3*Ki3, Ki3Max);
        pos3Millis = millis();
      }
    }
    brake(3);
  } else if (posErr3 < -50) {
    reverse(100, 3, 10);
    while (posErr3 < -50 && millis() - pos3Millis0 <= 2000) { //time out
after 2 seconds
      reverse(min(static_cast<int>(Kp3 * posErr3 + intErr3 + der3 * Kd3),
255), 3, 10);
      prevErr3 = posErr3;
      posErr3 = val - encoder3.getCount();
      der3 = posErr3 - prevErr3;
      Serial.println(encoder3.getCount());
```



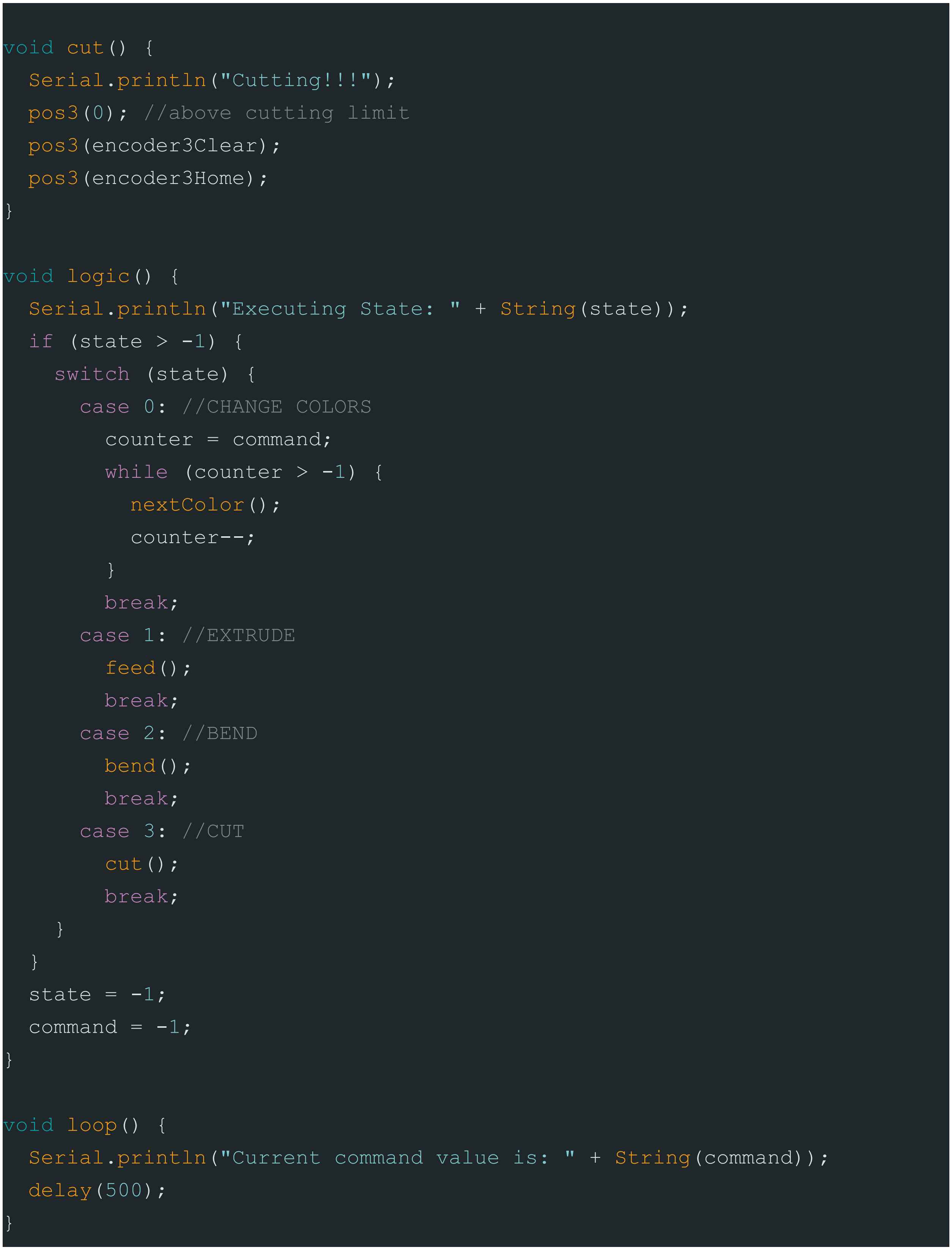
```
      if (millis() - pos3Millis >= 10) {
        intErr2 = min(intErr3 + posErr3*Ki3, Ki3Max);
        pos3Millis = millis();
      }
    }
    brake(3);
  }
}

void pos4(int val) {  //BENDING
  posErr4 = val - encoder4.getCount();
  intErr4 = 0;
  pos4Millis0 = millis();
  pos4Millis = millis();
  int prevErr4 = 0;
  int der4 = 0;
  if (posErr4 > 10) {
    fwd(150, 4, 10);
    while (posErr4 > 10 && millis() - pos4Millis0 <= 2000) {
      fwd(min(static_cast<int>(Kp4 * posErr4 + intErr4 + der4 * Kd4),
255), 4, 10);
      prevErr4 = posErr4;
      posErr4 = val - encoder4.getCount();
      der4 = posErr4 - prevErr4;
      if (millis() - pos4Millis >= 10) {
        intErr4 = min(intErr4 + posErr4*Ki4, Ki4Max);
        pos4Millis = millis();
      }
    }
    brake(4);
  } else if (posErr4 < -10) {
    reverse(100, 4, 10);
    while (posErr4 < -10 && millis() - pos4Millis0 <= 2000) {
      reverse(min(static_cast<int>(Kp4 * posErr4 + intErr4 + der4 * Kd4),
255), 4, 10);
      prevErr4 = posErr4;
      posErr4 = val - encoder4.getCount();
      der4 = posErr4 - prevErr4;
      if (millis() - pos4Millis >= 10) {
        intErr4 = min(intErr4 + posErr4*Ki4, Ki4Max);
        pos4Millis = millis();
      }
    }
    brake(4);
  }
}

//set motor to a desired speed, predetermined by an encoder calibration.
//actuate the motor until velocity dips below threshhold then brake. this
is the zero point

void zeroCarriage() { //Zero carriage then place at encoder3Max*0.3
(extruding/bending height)
  Serial.println("Zeroing carriage!");

  //LOWER CARRIAGE EVERY TIME, SAFE SINCE SPRING
  reverse(255, 3, 200);
  brake(3);

  fwd(200, 3, 50);

  fwd(120, 3, 100);  // At this speed, if encoder count drops to 1, we
have hit.
  check3();
  while (count3 > 4) {
    check3();
  }
  brake(3);
  encoder3.clearCount();

  reverse(120, 3, 200);
  check3();
  while (count3 < -6) {
    check3();
  }
  brake(3);
  Serial.println("Total encoder3 count = " +
String((int32_t)encoder3.getCount()));
  encoder3Max = encoder3.getCount();
  //fwd(100, 3, 100);
  encoder3Home = encoder3Max*0.38;
  encoder3Clear = encoder3Max;
  pos3(encoder3Home);
  Serial.println("Centered Carriage");
}

void zeroBendingHead() { //Zero bending head then place out of extrusion
path
  Serial.println("Zeroing bending head!");

  reverse(100, 4, 200);
  fwd(200, 4, 50);

  fwd(55, 4, 200);  // At this speed, if encoder count drops to 1, we have
hit.
  check4();
  while (count4 >= 1) {
    check4();
  }
  encoder4.clearCount();

  reverse(200, 4, 100);

  reverse(55, 4, 100);
  check4();
  while (count4 <= -1) {
    check4();
  }
  brake(4);
  Serial.println("Total encoder4 count = " +
String((int32_t)encoder4.getCount()));
  encoder4Max = encoder4.getCount()*0.75;
  encoder4Min = encoder4.getCount()*0.3;
  pos4(encoder4Min);
  bendState = -1;
  Serial.println("Bending Head Zeroed");
}

void zero() {
  //zero storage
  //nextColor();
  //zero carriage
  zeroCarriage();
  //zero bending head
  zeroBendingHead();
  //zero feeding (NOT SCOPED)
}

void nextColor() { //****IMPLEMENT CUT AND EXTRUDER REVERSE****
  Serial.println("Button press! moving");
  int fwd1i = 145;
  int fwd1f = 100;
  float ratio = 1.75;
  fwd(fwd1i, 1, 500);
  reverse(static_cast<int>(fwd1i*ratio), 2, 500);
  hallState = 0;
  while (hallState == 0) {
    hallState = digitalRead(HALL);
    fwd(fwd1f, 1, 10);
    reverse(static_cast<int>(fwd1f*ratio), 2, 10);
  }
  colorMillis = millis();
  while (millis() - colorMillis < 50) {
    brake(1);
  }
  Serial.println("MAGNET!!!");
  colorMillis = millis();
  while (millis() - colorMillis < 300) {
    brake(2);
  }
}

void feed(int param = command) {
  Serial.println("Button Press! Feeding");
  int scale = 200;
  switch (param) {
    case 0:
      Serial.println("state 1");
      pos2(-10 * scale);
      break;
```



```
    case 1:
      Serial.println("state 2");
      pos2(1 * scale);
      break;
    case 2:
      Serial.println("state 3");
      pos2(5*scale);
      break;
    case 3:
      Serial.println("state 4");
      pos2(20*scale);
      break;
    case 4:
      Serial.println("state 5");
      pos2(50*scale);
      break;
  }
}

void bend(int param = command) { //*****IMPLEMENT L/R STATE
DETECTION********
  Serial.println("Button Press! Bending");
  pos3(encoder3Home);
  switch (param) {
    case 0: //BEND RIGHT 90 DEG
      if (bendState == 1) {
        pos3(encoder3Clear);
        pos4(encoder4Min);
        pos3(encoder3Home);
      }
      Serial.println("Bending right 90 degrees");
      pos4(encoder4Max); //TUNE
      pos4(encoder4Min);
      bendState = -1;
      break;
    case 1: //BEND LEFT 90 DEG
      if (bendState == -1) {
        pos3(encoder3Clear);
        pos4(encoder4Max);
        pos3(encoder3Home);
```



```
      }
      Serial.println("Bending left 90 degrees");
      pos4(encoder4Min); //TUNE
      pos4(encoder4Max);
      bendState = 1;
      break;
    case 2: //STAIRCASE
      for (int i = 0; i < 3; i++) {
        bend(1);
        feed(2);
        bend(0);
        feed(2);
      }
      break;
    case 3: //CIRCLE *****TUNE!!!*****
      Serial.println("Moving to right circle position");
      if (bendState == -1) {
        pos4(encoder4Min);
      } else {
        pos3(encoder3Clear);
        pos4(encoder4Min);
        pos3(encoder3Home);
      }
      pos4((encoder4Max+encoder4Min)*0.5);
      feed(3);
      break;
    case 4:
      Serial.println("Moving to left circle position");
      if (bendState == 1) {
        pos4(encoder4Max);
      } else {
        pos3(encoder3Clear);
        pos4(encoder4Max);
        pos3(encoder3Home);
      }
      pos4((encoder4Max+encoder4Min)*0.5);
      feed(3);
      break;
  }
}
```



```
void cut() {
  Serial.println("Cutting!!!");
  pos3(0); //above cutting limit
  pos3(encoder3Clear);
  pos3(encoder3Home);
}

void logic() {
  Serial.println("Executing State: " + String(state));
  if (state > -1) {
    switch (state) {
      case 0: //CHANGE COLORS
        counter = command;
        while (counter > -1) {
          nextColor();
          counter--;
        }
        break;
      case 1: //EXTRUDE
        feed();
        break;
      case 2: //BEND
        bend();
        break;
      case 3: //CUT
        cut();
        break;
    }
  }
  state = -1;
  command = -1;
}

void loop() {
  Serial.println("Current command value is: " + String(command));
  delay(500);
}
```

Appendix D

Control Guide

Command **State**	**-1**	**0**	**1**	**2**	**3**	**4**
Changing colors	-	Cycle 1 color	Cycle 2 colors	Cycle 3 colors	Cycle 4 colors	Cycle 5 colors
Extruding wire	-	Reverse extrude	Extrude 1 unit	Extrude 5 units	Extrude 20 units	Extrude 50 units
Bending wire	-	Right bend 90 degrees	Left bend 90 degrees	Staircase routine	Right circle routine	Left circle routine
Cutting wire	-	Cut the wire				