ME 102B Project Final Report

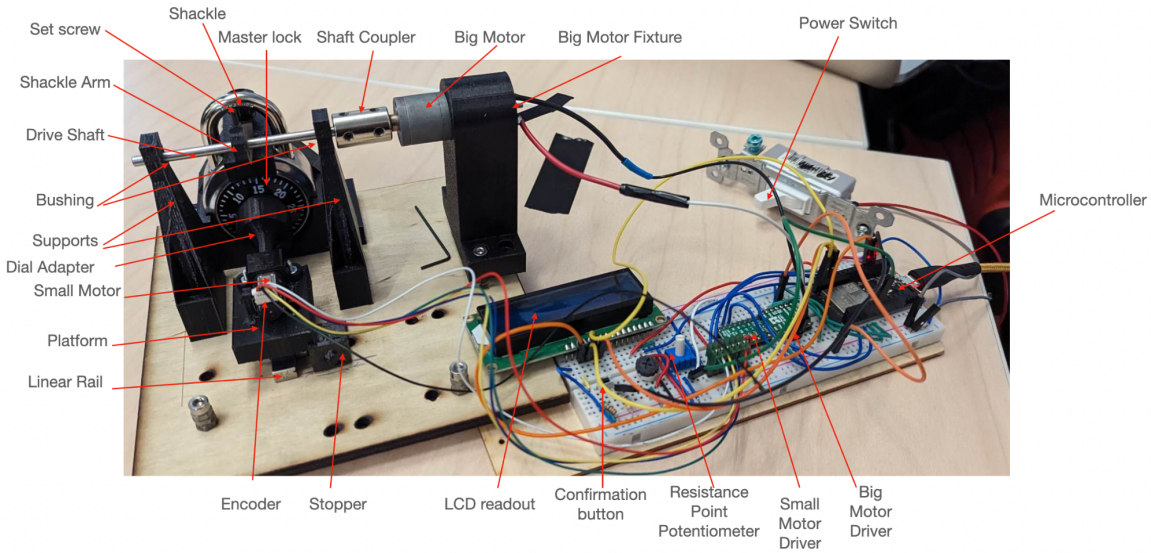
Opportunity: Create a device that helps individuals regain use of their old master locks, despite forgetting the password.

High level strategy:

1. User manually finds three resistant points
    a. The first is found by half-pulling up on the shackle and turning clockwise until resistance is felt
    b. The other two are found by full-pulling up on the shackle and turning counter clockwise. In finding these two locations, start at 0 and don't exceed 12. There will be more than 2 locations found, so only accept the locations where the resistance is found half way between 2 numbers.
2. Set lock to 0 and place in device
3. Individually enter the three resistant points using the potentiometer, pressing the button to confirm each entry
4. Microcontroller generates 16 potential combinations from user inputted numbers
5. Device tries all combinations, stopping when the user presses the button (indicating a successful combination)
6. Output combination to readout

Initially we wanted the device to automatically find the resistance locations via a force control loop (based on mapping current to PWM) for the big motor pulling up the shackle and a basic stop detection protocol for the motor turning the lock dial. While this scheme worked somewhat well, issues cropped up with finding the resistant locations accurately. Oftentimes the found locations would be 1 lock tick off, which would mess up the combinations the device would try in the brute force phase. Therefore we opted to manually determine the resistance points and input them to the device.

Photo of fully integrated device



Function critical decisions:

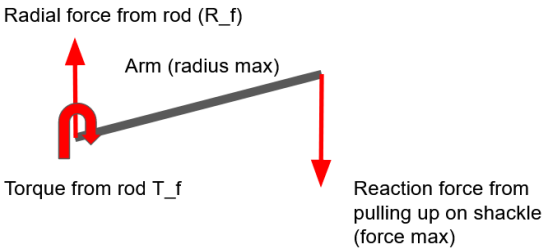
In selecting the motor and bushings that rotate the shaft which pulls up the master lock shackle the following calculations were carried out. Note that the numbers that follow were selected such that they are higher than our system would ever experience:

$$max\ force\ =\ 9.8N,\ max\ \ radius\ =\ 10cm$$

Note from these variables the **selected bushings are more than strong enough to withstand the experienced radial force of 9.8N, as they are rated for nearly 1650 Newtons of radial force.**

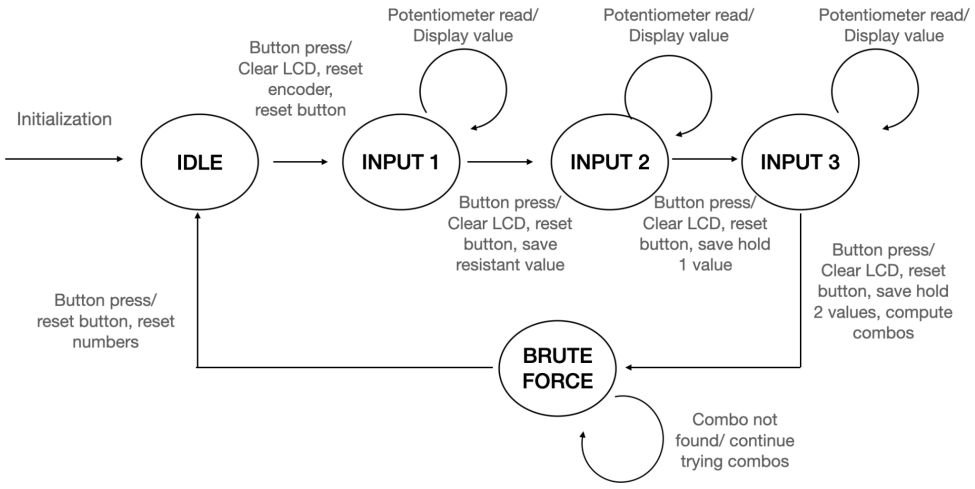
$$radius\ X\ max\ force\ \ =\ 98\ N\ *\ cm\ \ =\ \ 10\ kg\ *\ cm$$

We thus selected a motor with a stall torque higher than 16 kg*cm such that we would never expose it to torques above 60% of its stall torque (the actual motor has 21 kg*cm stall torque, so there is a decent safety margin as well). See the figure below for a basic outline of these calculations.

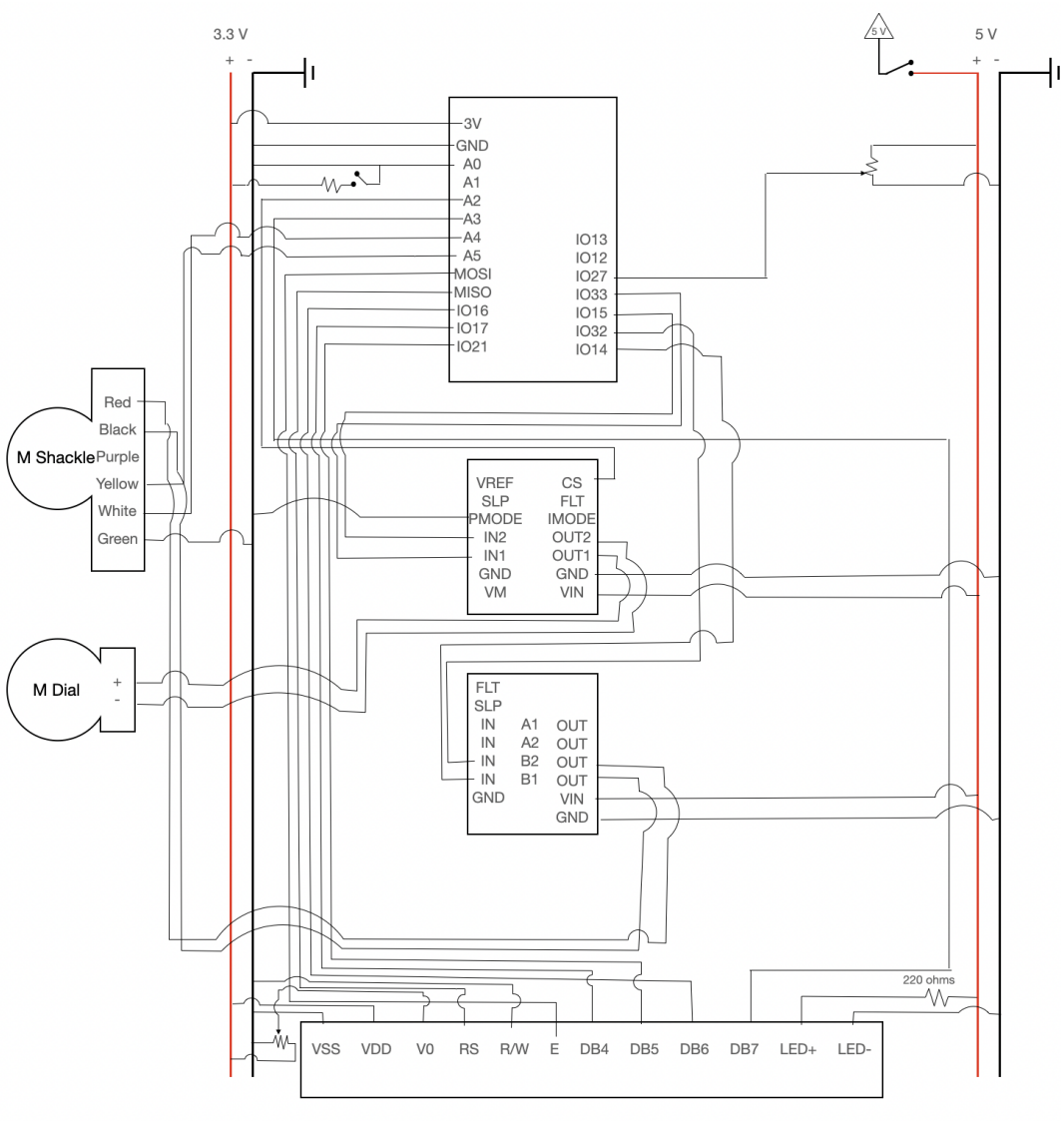


Force/moment balances (assuming static):
R_f = force max
T_f = radius max * force max

State Transition Diagram:



Circuit Diagram



Reflection:

Regular meetings and forming mini mechanical and electrical/software subteams were strategies that worked well for us. That's not to say if someone wanted to help with both subteams they couldn't, but that their main focus was well defined. We definitely recommend these strategies for future students. As for words of advice: don't let the thought of Thanksgiving break let you slow down progress around the start of November, and try to minimize the use of 3D printing where possible.

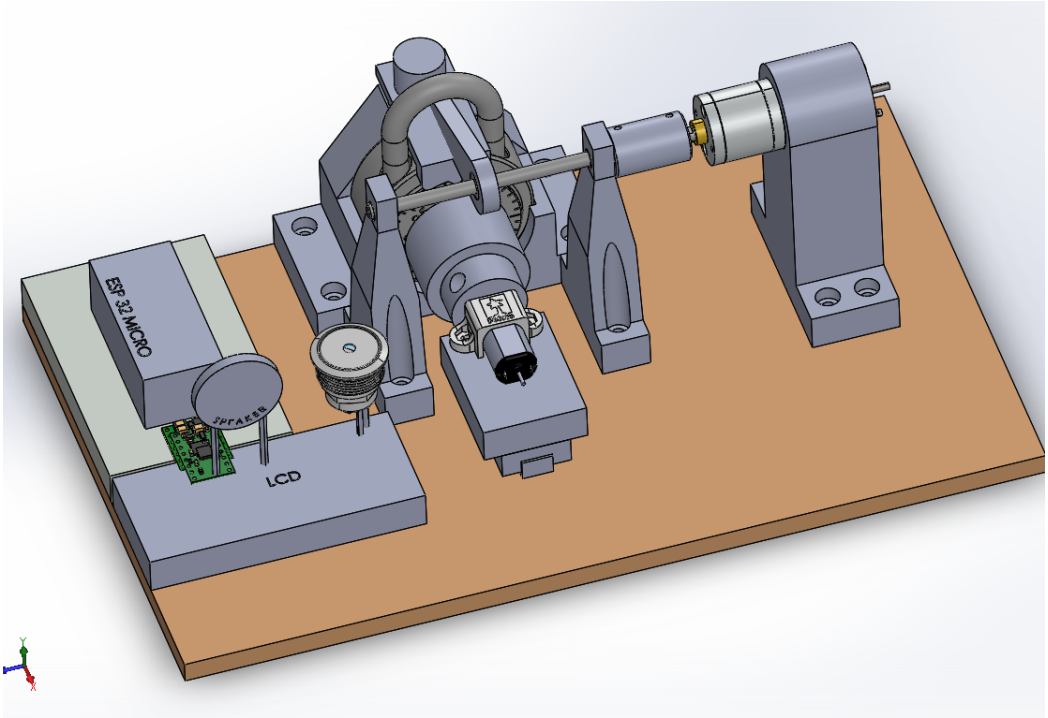
Appendix

Complete BOM:

	Lock Breaker BOM	
	Total:	$177.76

Item	Description	Supplier	Unit Cost	Number of Units	Total cost	Purchased?
Pololu 20mm motor	Motor for force arm	Pololu	$29.95	1	$29.95	Yes
DRV 8874 motor driver	H-Bridge motor driver	Pololu	$9.95	1	$9.95	Yes
DRV8833 Dual Motor Driver Carrier	H-bridge motor driver	Pololu	$0.00	1	$0.00	Yes
75:1 gear ratio micromotor	Lock turning motor	Pololu	$0.00	1	$0.00	Yes
M2 button head screws	screws for the screen readout	McMaster	$11.57	1	$11.57	Yes
Breadboard	Breadboard	Pololu	$0.00	1	$0.00	Yes
IK0 Linear Rail	Linear rail	IK0 Thompson	$0.00	1	$0.00	Yes
Master combination lock	lock	McMaster	$0.00	1	$0.00	Yes
Stock 1/4" Plywood	Wood	Jacobs Hall	$6.25	1	$6.25	Yes
High load dry running sleeve bearing	bushings for transmission	McMaster	$1.97	2	$3.94	Yes
Rotary shaft	rotary shaft for transmission	McMaster	$17.86	1	$17.86	Yes
4mm shaft coupling	shaft coupler for the transmission	Amazon	$9.99	1	$9.99	Yes
Set screw shaft collar	for applying force on shackle	McMaster	$1.77	1	$1.77	Yes
Long set screw	to touch shackle	McMaster	$5.71	1	$5.71	Yes
ESP32 Microcontroller	Microntroller	Adafruit	$0.00	1	$0.00	Yes
Jumper wire	Wires	Amazon	$0.00	1	$0.00	Yes
M4 Screw	Load cell fastner	McMaster	$8.72	1	$8.72	Yes
Button	Button	ITWSwitches	$0.00	1	$0.00	Yes
3D print filament	Misc components	N/A	$0.00	1	$0.00	Yes
Wing nut	Retain combo lock	McMaster	$13.05	1	$13.05	Yes
Power supply	power electronics	N/A	$0.00	1	$0.00	Yes
M4 bolt	Fastner for various components	McMaster	$7.30	1	$7.30	Yes
M4 nut	Fastner for various components	McMaster	$3.33	1	$3.33	Yes
Long M4 bolt	to fasten combo lock down	McMaster	$9.42	1	$9.42	Yes
Display	LCD	Amazon	$5.59	1	$5.59	Yes
M3 bolts/nuts	to mount linear rail and carriage	McMaster	$0.00	1	$0.00	Yes

CAD Image:



Code screenshots:

```
#include <ESP32Encoder.h>
#include <LiquidCrystal.h>

#define BIN_1 32
#define BIN_2 14
#define ENC_1 39
#define ENC_2 36
#define EN 15
#define PH 33
#define CS 34
#define LED_PIN 13
#define BTN 26
#define POT 27

#define RS 19
#define L_EN 18
#define D4 17
#define D5 21
#define D6 16
#define D7 25

// ---------------------------------------------------------------------
// Variable declarations -----------------------------------------------
// ---------------------------------------------------------------------

// Encoder object
ESP32Encoder encoder;

//Setup interrupt variables ----------------------------
volatile int count = 0; // encoder count variable
volatile bool deltaT = false; // position control check variable
volatile bool button_pressed = false; // debounce storage variable
volatile bool timeout = false; // timeout storage variable

// setting PWM properties ---------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int ledChannel_3 = 3;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 230;
const float MAX_VEL = 5.8;

// setting Timer info -------------------------------
const int time_count = 80;
const int tic_count = 1000;
const int clock_rate = 80000000;
```

```
48
49  hw_timer_t * timer0 = NULL; // timeout timer
50  portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
51
52  hw_timer_t * timer1 = NULL; // position control timer
53  portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
54
55  // setting Motor info ---------------------------------
56  const float gear_ratio = 75.81;
57  const int cpr = 12;
58
59  // combination computation info -----------------------
60  int res; int loc_1 = 98; int loc_2 = 99; int pos_1; int pos_2[10]; int pos_3[2];
61
62  // state variables ------------------------------------
63  int state = 0;
64
65  const int IDLE = 0;
66  const int INPUT1 = 1;
67  const int INPUT2 = 2;
68  const int INPUT3 = 3;
69  const int BRUTE_FORCE = 4;
70
71  // setting cascaded positional control information -----
72  const float kp = 0.3;
73  const float kip = 0.1;
74  const float kpp = 35;
75  const float kii = 0.01;
76  const int allowed_error = 9;
77  float goal = cpr * gear_ratio / 40.0;
78  bool spin_done = false;
79
80  // setting current PI control information -----
81  float prev_count = 0;
82  float current = 0;
83
84  // check for stop variable
85  int stop_count = 0;
86  bool start = false;
87  int pot = 0;
88
89  LiquidCrystal lcd(RS,L_EN, D4, D5, D6, D7);
90
91  // Interrupt callbacks ----------------------------------
92  // Position control timer callback
```

```
90
91  // Interrupt callbacks -----------------------------------
92  // Position control timer callback
93  void IRAM_ATTR onTime1() {
94    portENTER_CRITICAL_ISR(&timerMux1);
95    // cascaded position control data
96    count = encoder.getCount();
97    deltaT = true;
98    portEXIT_CRITICAL_ISR(&timerMux1);
99  }
100
101 // Button callback
102 void IRAM_ATTR isr() {
103   timeout_start();
104 }
105
106 // Timeout timer callback
107 void IRAM_ATTR onTime0() {
108   portENTER_CRITICAL_ISR(&timerMux0);
109   timeout = true;
110   portEXIT_CRITICAL_ISR(&timerMux0);
111   timerStop(timer0);
112 }
113
114 // ------------------------------------------------------------------
115 // Initialization ---------------------------------------------------
116 // ------------------------------------------------------------------
```

```
117
118 void setup() {
119   // sets the initial state of LED as turned-on
120   pinMode(LED_PIN, OUTPUT);
121   digitalWrite(LED_PIN, HIGH);
122
123   // ESP32 configuration
124   Serial.begin(115200);
125   ESP32Encoder::useInternalWeakPullResistors = UP;
126
127   // Encoder configuration
128   encoder.attachFullQuad(ENC_1, ENC_2);
129
130   // Button and shackle driver pin setup
131   pinMode(BTN, INPUT);  // configures the specified pin to behave either as an input or an output
132   attachInterrupt(BTN, isr, RISING);
133   pinMode(PH, OUTPUT);
134   pinMode(POT, INPUT);
135
136   // LED PWM configuration
137   ledcSetup(ledChannel_1, freq, resolution);
138   ledcSetup(ledChannel_2, freq, resolution);
139   ledcSetup(ledChannel_3, freq, resolution);
140   ledcAttachPin(BIN_1, ledChannel_1);
141   ledcAttachPin(BIN_2, ledChannel_2);
142   ledcAttachPin(EN, ledChannel_3);
143
144   // Timeout timer configuration
145   timer0 = timerBegin(0, time_count, true);
146   timerAttachInterrupt(timer0, &onTime0, true);
147   timerAlarmWrite(timer0, 6000, true);
148   timerAlarmEnable(timer0);
149   timerStop(timer0);
150
151   // Position control timer configuration
152   timer1 = timerBegin(1, time_count, true);
153   timerAttachInterrupt(timer1, &onTime1, true);
154   timerAlarmWrite(timer1, tic_count, true);
155   timerAlarmEnable(timer1);
156   timerStop(timer1);
157
158   // Setup LCD
159   lcd.begin(16, 2);
160   lcd.clear();
161   lcd.setCursor(0,0);
162   lcd.print("Lock Breaker");
163
164   shackle_turn(false);
165   delay(500);
166   ledcWrite(ledChannel_3, 0);
167 }
168
```

```
// ---------------------------------------------------------------
// Main loop -----------------------------------------------------
// ---------------------------------------------------------------

void loop() {
  switch(state) {

    // event checkers: check_button()
    // services: timeout_start(), shackle_turn(), encoder.setCount()
    case IDLE:
      if (check_button()) {
        state = INPUT1;
        lcd.clear();
        encoder.setCount(0);
        button_reset();
        Serial.println("Switching from IDLE to Resistant input");
      }
      break;

    case INPUT1:
      pot = analogRead(POT);
      pot = map(pot, 0, 4096, 0, 40);
      lcd_res(pot);
      delay(100);
      if (check_button()) {
        lcd.clear();
        button_reset();
        res = pot;

        state = INPUT2;
        Serial.println("Switching from Resistant input to Hold input 1");
      }
      break;

    case INPUT2:
      pot = analogRead(POT);
      pot = map(pot, 0, 4096, 0, 40);
      lcd_hold(pot, 99);
      delay(100);
      if (check_button()) {
        lcd.clear();
        button_reset();
        loc_1 = pot;

        state = INPUT3;
        Serial.println("Switching from Hold input 1 to Hold input 2");
      }
      break;

     case INPUT3:
      pot = analogRead(POT);
      pot = map(pot, 0, 4096, 0, 40);
      lcd_hold(loc_1, pot);
      delay(100);
      if (check_button()) {
        lcd.clear();
        button_reset();
        loc_2 = pot;

        combo_comp();
        state =  BRUTE_FORCE;
```

```
        combo_comp();
        state =  BRUTE_FORCE;

        Serial.println("Switching from Hold input 2 to Brute Force");
        Serial.println("Resistant location:");
        Serial.println(res);
        Serial.println("Hold 1:");
        Serial.println(loc_1);
        Serial.println("Hold 2:");
        Serial.println(loc_2);
      }
      break;



    // event checkers: brute_force()
    // services: disp_combo(), button_reset(), num_reset(), error()
    case BRUTE_FORCE:
      //brute force code
      if (brute_force()) {
        state = IDLE;

        button_reset();
        num_reset();

        Serial.println("Switching from BRUTE_FORCE to IDLE");
      }
      break;
  }
}

// ------------------------------------------------------------------
// Event checker functions ------------------------------------------
// ------------------------------------------------------------------

// Checks for start button press
bool check_button() {
  return timeout;
}
```

```
// carries out and verifies brute force was successful
bool brute_force() {
  bool success = false;
  lcd.clear();

  for(int w = 0; w < 10; w++) {
    for(int j = 0; j < 2; j++) {
      if((abs(pos_2[w] - pos_3[j]) > 2) && !check_button()) {
        lcd.clear();
        Serial.println("-------------------------");
        Serial.println(res + 5);
        Serial.println(pos_2[w]);
        Serial.println(pos_3[j]);
        lcd_bf_num(res + 5, pos_2[w], pos_3[j]);

        control_start();
        combo_spin(res + 5, pos_2[w], pos_3[j] + 1);
        control_stop();

        shackle_pull();
        unsigned long x = millis();
        while (millis() - x < 500) {
          if (check_button()) {
            lcd.clear();
            lcd_combo_found(res + 5, pos_2[w], pos_3[j]);
          }
        }
      }
    }
  }
  return check_button();
}

// ---------------------------------------------------------------
// Service functions ---------------------------------------------
// ---------------------------------------------------------------


// Turns the shackle motor at medium, constant PWM value at input direction
void shackle_turn(bool dir) {
  if (dir) {
    digitalWrite(PH, HIGH);
    ledcWrite(ledChannel_3, 255);
  } else {
    digitalWrite(PH, LOW);
    ledcWrite(ledChannel_3, 80);
  }
}


// Display error code on LCD
void error() {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Error, returning to idle");
}

void control_stop() {
  timerStop(timer1);
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount();
  deltaT = false;
  portEXIT_CRITICAL_ISR(&timerMux1);
}
```

```
// Creates list of possible combinations from resistant locations
// inputs:
// res - half force resistant location lock number
// loc_1 - full force resistant location lock number 1
// loc_2 - full force resistant location lock number 2
void combo_comp() {
  pos_1 = res + 5;


  int r = pos_1 % 4;
  int r1 = loc_1 % 10;
  int r2 = loc_2 % 10;

  int ind1 = 0;
  int ind2 = 0;
  int m = 0;
  int m1 = 0;
  int m2 = 0;
  for(int i = 0 ; i< 41; i++) {
    m = (i+2) % 4;
    m1 = i % 10;
    m2 = i % 4;
    if((m1 == r1 || m1 == r2) && m2 == r) {
     pos_3[ind1] = i;
     ind1 = ind1 + 1;
    }
    if(m == r) {
      pos_2[ind2] = i;
      ind2 = ind2 + 1;
    }
  }

}

// Executes one combination on the lock
// inputs:
// s1 - first lock number
// s2 - second lock number
// s3 - third lock number
void combo_spin(int s1, int s2, int s3) {
  int l1 = mapping(s1, 0, 0, 3, false);
  int l2 = mapping(s2, s1, l1, 1, true);
  int l3 = mapping(s3, s2, l2, 0, false);

  lock_spin(0);
  delay(500);
  lock_spin(l1);
  delay(500);
  lock_spin(l2);
  delay(500);
  lock_spin(l3);
  delay(500);
}
```

```
// Turns lock to specified lock number
void lock_spin(int g) {
  spin_done = false;

  g = g * goal;

  float e = 0; float e_p = 0; float prev_e = 0;
  float sum_e = 0;
  float p_count = count;

  while (!spin_done) {
    if (deltaT) {
      // Handle timer interupt
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      // compute position error
      e_p = g - count;

      // compute desired velocity
      float v_des = kp * e_p + kip * prev_e;

      // variable updates
      if (e_p < 40 * goal && e_p > -40 * goal) {
        if (v_des > MAX_VEL) {
          v_des = MAX_VEL;
        } else if (v_des < - MAX_VEL) {
          v_des = -MAX_VEL;
        }
      }

      prev_e = prev_e + e_p;
      if (prev_e > 75) {
        prev_e = 75;
      } else if (prev_e < -75) {
        prev_e = -75;
      }


      float vel = count - p_count;
      e = v_des - vel;

      // compute desired PWM
      int D = kpp * e + kii * sum_e;

      // variable updates
      if (D > MAX_PWM_VOLTAGE) {
        D = MAX_PWM_VOLTAGE;
      }
      else if (D < -MAX_PWM_VOLTAGE) {
        D = -MAX_PWM_VOLTAGE;
      }

      sum_e = sum_e + e;
      if (sum_e > 1000) {
        sum_e = 1000;
      } else if (sum_e < -1000) {
        sum_e = -1000;
      }
      p_count = count;
```

```
449
450         //Map the D value to motor directionality, or brake motor if close enough
451         if (e_p < allowed_error && e_p > -allowed_error) {
452           ledcWrite(ledChannel_2, 255);
453           ledcWrite(ledChannel_1, 255);
454           spin_done = true;
455         }
456         else if (D > 0) {
457           ledcWrite(ledChannel_1, LOW);
458           ledcWrite(ledChannel_2, D);
459         }
460         else if (D < 0) {
461           ledcWrite(ledChannel_2, LOW);
462           ledcWrite(ledChannel_1, -D);
463         }
464         else {
465           ledcWrite(ledChannel_2, LOW);
466           ledcWrite(ledChannel_1, LOW);
467         }
468       }
469     }
470   }
471
472
473
474   // Resets start button press variable
475   void button_reset() {
476     portENTER_CRITICAL_ISR(&timerMux0);
477     timeout = false;
478     portEXIT_CRITICAL_ISR(&timerMux0);
479   }
480
481   // Starts shackle motor timeout timer
482   void timeout_start() {
483     portENTER_CRITICAL_ISR(&timerMux0);
484     timeout = false;
485     portEXIT_CRITICAL_ISR(&timerMux0);
486
487     timerRestart(timer0);
488     timerStart(timer0);
489   }
490
491   // Starts shackle motor timeout timer
492   void control_start() {
493     timerRestart(timer1);
494     timerStart(timer1);
495   }
496
497   // Resets resistant location numbers
498   void num_reset() {
499     res = 0;
500     loc_1 = 0;
501     loc_2 = 0;
502   }
503
```

```
503
504 // ----------------------------------------------------------------
505 // Helper functions ------------------------------------------------
506 // ----------------------------------------------------------------
507
508 // Maps desired lock number to encoder count value
509 // inputs:
510 // s1 - desired lock number
511 // s2 - current lock number
512 // turns - number of turns to execute between two numbers
513 // dir - direction of rotation; true -> CCW, false -> CW
514 int mapping(int pos, int last_pos, int last_abs, int turns, bool dir) {
515   int a;
516   if (dir) {
517     a = pos - last_pos;
518     if (a < 0) {
519       a = a + 40;
520     }
521
522     return a + 40 * turns + last_abs;
523   }
524   else {
525     a = pos - last_pos;
526     if (a > 0) {
527       a = a - 40;
528     }
529
530     return a - 40 * turns + last_abs;
531   }
532 }
533
534 // Verifies if shackle is released or not (used by brute force to check success)
535 void shackle_pull() {
536   shackle_turn(true);
537   unsigned long x = millis();
538   while (millis() - x < 3500) {
539
540   }
541
542   x = millis();
543   shackle_turn(false);
544   while (millis() - x < 1500) {
545
546   }
547
548   ledcWrite(ledChannel_3, 0);
549 }
550
551 // LCD display functions -------------------
552 void lcd_res(int n1) { //looking for resistant location
553   if (n1 == 99) { //if still looking for res location, then n1 should be 99; if has found, then n1 will be 0-40 and it will print actual
554     int r1 = random(10,99); //generates random number for fuzz
555     n1 = r1;
556   }
557   String str1 = String(n1);
558   if (n1 < 10) {
559     str1 = "0" + str1;
560   }
561   String str = "Resistant: " + str1;
562   lcd.setCursor(0,0);
563   lcd.print(str);
564 }
565
```

```
void lcd_hold(int n1, int n2) { //if looking for hold1 then n1 == 99, n2 = 98; if looking for hold 2 then n2 = 99
  if (n1 == 99) {
    int r1 = random(10,99);
    n1 = r1;
  }
  String str1 = String(n1);
  if (n1 < 10) {
    str1 = "0" + str1;
  }
  if (n2 == 99) {
    int r2 = random(10,99);
    n2 = r2;
  }
  String str2 = String(n2);
  if (n2 < 10) {
    str2 = "0" + str2;
  }
  String str_a = "Hold 1: " + str1;
  String str_b = "Hold 2: " + str2;
  lcd.setCursor(0,0);
  lcd.print(str_a);
  lcd.setCursor(0,1);
  if (n2 == 98) {
    lcd.print("Hold 2: ");
  }
  else {
  lcd.print(str_b);
  }
}


void lcd_display3(int n1, int n2, int n3) { //display the 3 places it has found
  String str1 = String(n1);
  String str2 = String(n2);
  String str3 = String(n3);
  if (n1 < 10) {
    str1 = "0" + str1;
  }
  if (n2 < 10) {
    str2 = "0" + str2;
  }
  if (n3 < 10) {
    str3 = "0" + str3;
  }
  lcd.setCursor(0,0); lcd.print("Res");
  lcd.setCursor(0,1); lcd.print(str1);
  lcd.setCursor(7,0); lcd.print("H1");
  lcd.setCursor(7,1); lcd.print(str2);
  lcd.setCursor(14,0); lcd.print("H2");
  lcd.setCursor(14,1); lcd.print(str3);
}
```

```
void lcd_bf_num(int n1, int n2, int n3) {
  String str1 = String(n1);
  String str2 = String(n2);
  String str3 = String(n3);
  if (n1 < 10) {
    str1 = "0" + str1;
  }
  if (n2 < 10) {
    str2 = "0" + str2;
  }
  if (n3 < 10) {
    str3 = "0" + str3;
  }
  String str = str1 + " " + str2 + " " + str3;
  lcd.setCursor(0,0);
  lcd.print("Brute Forcing");
  lcd.setCursor(0,1);
  lcd.print(str);
}


void lcd_combo_found(int n1, int n2, int n3) { //display the final combo
  String str1 = String(n1);
  String str2 = String(n2);
  String str3 = String(n3);
  if (n1 < 10) {
    str1 = "0" + str1;
  }
  if (n2 < 10) {
    str2 = "0" + str2;
  }
  if (n3 < 10) {
    str3 = "0" + str3;
  }
  String str = str1 + " " + str2 + " " + str3;
  lcd.setCursor(0,0);
  lcd.print("COMBO");
  lcd.setCursor(0,1);
  lcd.print(str);
}

// Deprecated, used to trouble shoot control functions
void plotControlData(int c, int g, int p) {
  Serial.println("Position, Goal_Position, PWM");
  Serial.print(c);
  Serial.print(" ");
  Serial.print(g);
  Serial.print(" ");
  Serial.println(p);
}
```