Mobility Chair

Jacob Rey

```
/* Version Notes:
 3
         Features Implemented:
          PID control and encoder feedback of two motors
 4
 5
          8 Directional Control (orthogonal and diagonals)
 6
          Inputs are still sketched as physical buttons
 7
          Therapist Override Switch
 8
          LED outputs for control indicator/feedback
 9
          3 Mode Speed Selection
10
          Battery Voltage sensing / low battery indicator
11
          Alarms (blinks of different duration for different meanings for:
12
          - Gyroscope / Tilt Detection
13
          - IR distance sensing
14
          - Proximity of Enclosure
          - Audible Feedback for motion
15
16
        Features Not Yet Implemented:
17
18
          - Seat presence / weight detection
19
20
        Bugs Found/Fixed:
21
          - N/A
2.2
    */
2.3
24
25
    /*
26
      Code library for the MPU 9250 IMU was sourced from:
        Advanced_I2C.ino
27
28
        Brian R Taylor
29
        brian.taylor@bolderflight.com
30
        Copyright (c) 2017 Bolder Flight Systems
        https://github.com/bolderflight/MPU9250
31
32
        Permission is hereby granted, free of charge, to any person obtaining a copy of
33
    this software
        and associated documentation files (the "Software"), to deal in the Software
34
    without restriction,
        including without limitation the rights to use, copy, modify, merge, publish,
35
    distribute,
        sublicense, and/or sell copies of the Software, and to permit persons to whom
36
    the Software is
        furnished to do so, subject to the following conditions:
37
38
39
        The above copyright notice and this permission notice shall be included in all
    copies or
        substantial portions of the Software.
40
41
        THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
42
        BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
43
    PURPOSE AND
        NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
44
    FOR ANY CLAIM,
        DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
45
    OTHERWISE, ARISING FROM,
        OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.
47
    */
```

```
48
    //INCLUDE CODE
    49
     //Include the library for the MPU9250 and setup the accelerometer variables
50
       #include "MPU9250.h"
      //Setup variables for inertial measurment unit
51
       float Xdeg = 0;
52
53
       float Ydeg = 0;
      float Zdeg = 0;
54
55
       float Xrad = 0;
      float Yrad = 0;
56
       float Zmss = 0;
57
58
       float temp = 0;
       float Zmss limit = 8.80; // Limit of the tilt in the Z axis before
59
    shutoff
60
       const byte temp limit = 49;  // Limit of the temperature in °C before
    shutoff (~120°F)
61
     //Setup I2C Comms for MPU9250 object on bus 0 with address 0x68
62
       MPU9250 IMU(Wire,0x68); //NOTE: This sensor is physically connected to pins
    20 and 21 (SDA and SCL). It also requires 3.3V and GND.
63
        int status;
64
65
    //PINOUT
    //Control Inputs
66
    const byte forwardButtonPin = 5;
                                                    //Set the pin used for the
    input button
68
    const byte leftButtonPin = 6;
   const byte rightButtonPin = 7;
69
70
   const byte reverseButtonPin = 8;
    /* The direction inputs will all eventually be replaced by the I2C communication of
71
    the capacitive touch*/
72
    const byte modelPin = 32;
                                                     //Used for selecting mode 1 on
    rotary switch (switch dim 3 to 4, also 7 to 8)
73 const byte mode2Pin = 33;
                                                     //Used for selecting mode 2 on
    rotary switch (switch dim 1 to 2, also 5 to 6)
74
    const byte mode3Pin = 34;
                                                     //Used for selecting mode 3 on
    rotary switch
75
    //note, program needs to be simplified, switch only has 2 positions
76
    const byte enableButtonPin = 2;
                                                    //Used for button that enables
    motion control (controlled by supervisor)
77
    //Status Indicators (Outputs)
78
    const byte systemEnabledIndPin = 30;
                                                    //Used for LED lights that
    indicate motion control is activated
79
    const byte alarmIndPin = 35;
                                                     //Used for LED lights that
    indicate a low battery level
80
    const byte leftIndPin = 26;
                                                     //Used for LED lights that
    indicate left has been pressed
81
    const byte rightIndPin = 27;
                                                     //Used for LED lights that
    indicate right has been pressed
82
    const byte forwardIndPin = 28;
                                                     //Used for LED lights that
    indicate forward has been pressed
    const byte reverseIndPin = 29;
                                                     //Used for LED lights that
83
    indicate reverse has been pressed
                                                     //Reserved for audio feedback
84
    const byte audioIndPin = 51;
85
    //Motor Outputs
    const byte leftMotorPWMPin = 4;
                                                     //Used for left motor PWM
    constrol signal
    const byte leftMtrDirPin = 25;
                                                     //Pin used to signal rotation
87
    of left motor
    const byte rightMotorPWMPin = 3;
                                                     //Used for right motor PWM
88
    constrol signal
    const byte rightMtrDirPin = 24;
                                                     //Pin used to signal rotation
    of left motor
```

```
90
   //Encoder Inputs
91
    const byte encoderLeftAPin = 14;
                                                //Encoder channel A for Left
    Motor - Note: This will be accessed by the interrupt only
92
    const byte encoderLeftBPin = 15;
                                                //Encoder channel B for Left
    Motor
    const byte encoderRightAPin = 16;
                                                //Encoder channel A for Right
93
    Motor
94
    const byte encoderRightBPin = 17;
                                                //Encoder channel B for Right
    Motor
95
    //Sensor Inputs
96 const byte sclPin = 21;
                                                //Used for the gyroscope and
    the capacitive touch
97
    const byte sdaPin = 20;
                                                //Used for the gyroscope and
    the capacitive touch
98
    const byte distIR1Pin = A0;
                                                //Used for the IR sensor
99
    const byte distIR2Pin = A1;
                                                //Used for the second IR sensor
100
   const byte proxEncPin = A3;
                                                //Used for the prox sensor that
    monitors the enclosure
101
   const byte seatSensePin = A4;
                                                //Used for the sensor that
    detects weight of seat/passenger
                                               //Used for the voltage sensor
102
    const byte batteryVoltagePin = A5;
    that detects low battery level
103
    104
    //HARDWARE RELATED
105
    106
    const int encoderResolution = 360;
                                                //Number of pulses for a single
    resolution (used to calculate RPM for troubleshooting/reference)
107
    108
109
    //STATUS
    //Sum of All Control Input Variables
110
111 byte controlState_crnt = 0;
                                                    //Sum of all the control
    states, used to detect any change in input
112
    byte controlState prev = 0;
113
    //Values for individual controls
114 boolean fwdBtnState crnt = 1;
                                                    // current state of the
    forward button
115
    boolean fwdBtnState_prev = 1;
                                                    // previous state of the
    forward button
116
    boolean leftBtnState crnt = 1;
                                                    // These are all set to 1
    by default because they are pullup inputs (1 is off)
117
    boolean leftBtnState prev = 1;
118
    boolean rightBtnState crnt = 1;
119
    boolean rightBtnState_prev = 1;
120
    boolean revBtnState_crnt = 1;
121
    boolean revBtnState prev = 1;
122
    //Used in StopOverride
123
    boolean systemEnableState = HIGH;
                                                     // Used to track the
    state of the supervisor override button
    volatile boolean enableBtnState_crnt = LOW;
                                                    // Used to toggle the
    state of the enable button
    boolean enableBtnState prev = LOW;
                                                    // Used to toggle the
125
    state of the enable button
    volatile unsigned long enableTime crnt = 0;
                                                    // Used to debouce the
126
    enable button based on timing (compare current to previous to see duration in
127
    unsigned long enableTime_prev = 0;
                                                    // Used to debouce the
```

```
enable button based on timing
     //Battery Level Indication and Constants
128
129
     const int lowBatteryLevel = 125;
                                                          // This should be adjusted
     to the lowest acceptable battery voltage level x100 (2200 estimated for 22V)
130
     unsigned long lowBatteryTime_crnt = 0;
131
     unsigned long lowBatteryTime prev = 0;
132
133
     //Enclosure Status Indicators
134
     volatile boolean enclosureState crnt = 0;
                                                          // Used by the falling
     interrupt created by prox sensor to detect enclosure lid open/close
135
     volatile boolean enclosureState_prev = 0;
136
     volatile unsigned long enclosureTime crnt = 0;
                                                         //Used to debounce the
     enclosure prox sensor - bounce happens when voltage drops upon motor start
     unsigned long enclosureTime prev = 0;
137
     //IR Distance Constant
138
139
     const int IRdistMAX = 15;
140
     //General Alarm Status Variables
141
    byte alarmStatus = 0;
142
143
     //Motor and Encoder Status Variables
144
    boolean stateEncoderL B = 0;
                                                         // State of the quatrature
     channel for Left motor encoder (used to determin direction of rotation)
145
     boolean stateEncoderR B = 0;
                                                        // State of the quatrature
     channel for Left motor encoder (used to determin direction of rotation)
146
     volatile unsigned int encoderLeftValue_counter = 0;  //The current count of the
     left encoder (updated instantly by interrupt)
     volatile unsigned int encoderRightValue_counter = 0;  //The current count of the
147
     right encoder (updated instantly by interrupt)
148
     int encoderLeftValue crnt = 0;
                                                        //Count of left encoder
     after 50ms (updated on time basis)
149
     int encoderLeftValue prev = 0;
                                                        //Last count of the left
     encoder
150
     String encoderLeftDir txt = "";
     int encoderRightValue crnt = 0;
                                                        //Count of left encoder
     after 50ms (updated on time basis)
     int encoderRightValue prev = 0;
                                                        //Last count of the left
152
     encoder
153
     String encoderRightDir txt = "";
154
     155
156
     //TIMING
     157
    //time variables must use long type because they will overflow if other datatypes
158
    unsigned long fwdPressedTime = 0;
                                        // the moment when fwrd button was pressed
                                        // the moment when fwd button was released
159
    unsigned long fwdReleasedTime = 0;
     unsigned long fwdTimeActive = 0;
160
    unsigned long fwdTimeInactive = 0;
161
162
163
    unsigned long leftPressedTime = 0;
164
    unsigned long leftReleasedTime = 0;
165
     unsigned long leftTimeActive = 0;
     unsigned long leftTimeInactive = 0;
166
167
168
     unsigned long rightPressedTime = 0;
169
     unsigned long rightReleasedTime = 0;
170
     unsigned long rightTimeActive = 0;
171
     unsigned long rightTimeInactive = 0;
172
173
     unsigned long revPressedTime = 0;
174
     unsigned long revReleasedTime = 0;
175
     unsigned long revTimeActive = 0;
```

```
176
    unsigned long revTimeInactive = 0;
177
178
    //System timers
    unsigned long timeHeld = 0;
179
                                // the duration the button has currently been
    held (while active)
    unsigned long timeReleased = 0;
                                // the duration of time since the button was
180
    released (while inactive)
    unsigned long timeCurrent = 0;  // the length of time the button has been
181
182
    unsigned long shortTimer = 0;
    unsigned long mediumTimer = 0;
183
    unsigned long longTimer = 0;
184
185
186
    //Warning Indicator timers
187
    byte warningState = 0;
                                  // Set to an integer value by the various
    sensors to indicate which alarm is active
188
    byte warningShort = 100;
                                 // The off time of the alarm blink
                                 // The on time of the alarm blink
    byte warningLong = 300;
189
    unsigned long warningTimer_crnt = 0;
190
191
    unsigned long warningTimer prev = 0;
192
193
    194
195
    //TIMING
    196
    /*Timers used to determine how often different subroutines are executed*/
197
    ms (note if changed from 50 array values need to be recalculated)
198
    const int mediumInterval = 50;
199
    const int longInterval = 250;
                                // Refresh rate for items checked less often
200
    201
    //MOTOR SPEED and DIRECTION
202
    int leftTargetSpeed_crnt = 0;
                                        // This value is dynamically
    adjusted based on the time the control inputs have been held/released
204
    int leftTargetSpeed prev = 0;
205
    int rightTargetSpeed crnt = 0;
                                        // This value is dynamically
    adjusted based on the time the control inputs have been held/released
206
    int rightTargetSpeed prev = 0;
207
208
   const byte modelFactor = 1;
                                       //Full Speed
209
    const float mode2Factor = 0.67;
                                       //Medium Speed (default)
    const float mode3Factor = 0.50;
210
                                       //Slow Speed
    float modeFactorFinal = 1.00;
211
                                      //Set to 1 by default to indicate
212
    boolean modelStatus = 1;
    "off" (representing pullup inputs)
    boolean mode2Status = 1;
213
214
    boolean mode3Status = 1;
215
    //LeftMotor
216
    //String leftMotorDirection = ""; //Used for troubleshooting detected
    //String rightMotorDirection = ""; //Used for troubleshooting detected
218
    rotation
219
    220
221
    //MOTOR SPEED RELATED
```

```
222
   const int maxSpeed = 243;
                               //Maximum target speed in pulses per 50ms
    (based on expected pulses from encoder)
223
   const int accelTime = 4000;
                               //Target time to reach maximum speed in ms
    (controls rate of acceleration - note, if this changes array must be recreated)
   const int decelTime = 900;
                              //Target time to reach a stop in ms (rate of
224
   deceleration of the motor)
   const int AccelSpeedValues[80] = {1, 1, 1, 1, 1, 2, 2, 2, 3, 4, 4, 6, 7, 9, 10, 12,
225
    14, 17, 20, 22, 26, 29, 33, 36, 41, 45, 49, 54, 59, 64, 69, 75, 80, 86, 92, 98,
    103, 109, 116, 122, 128, 134, 140, 146, 151, 157, 163, 168, 174, 179, 184, 189,
    194, 198, 203, 207, 210, 214, 217, 221, 224, 226, 229, 231, 233, 235, 236, 237,
   //Number of values contained in
226
   const byte AccelArrayCount = 80;
   AccelSpeedValues[]
   const int DecelSpeedValues[18] = {238, 232, 221, 207, 190, 169, 146, 122, 98, 75,
227
   54, 37, 23, 12, 6, 2, 1, 0};
228
   const byte DecelArrayCount = 18;
                              //Number of values contained in
   DecelSpeedValues[]
229
   const byte revFactor = 2;
                              //This controls the speed of reverse relative
   to forward (divide by this value)
230
   const byte diagTurnFactor1 = 5;
                               //These constants control the tightness of a
   diagonal turning radius
231
   const byte diagTurnFactor2 = 2;
                              //Multiply by factor1, divide by factor 2 to
   achieve rounded factors (ex2.5)
232
    233
234
    //PID CONTROL
   float kp = 1.25; //These values will need some tuning
235
236
   float ki = 0.01;
237
   float kd = 0.1;
238
   float PID_Lp, PID_Li, PID_Ld;
   int PID Ltotal = 0;
239
    float PID_Rp, PID_Ri, PID_Rd;
240
    int PID Rtotal = 0;
241
242
    243
244
   //PID CONTROL
   ////////
245
   int errorLeft crnt;
246
   int errorLeft prev;
247
   int errorRight crnt;
248
   int errorRight_prev;
249
   int errorBetween crnt;
250
   int errorBetween prev;
251
    252
253
    //********BEGIN PROGRAM
   255
    *******
256
   //THIS SETS UP THE INPUT AND OUTPUTS AND SERIAL MONITOR COMMUNICATION
257
   void setup() {
258
     //Start the serial monitor for troubleshooting and displaying status
259
     Serial.begin(115200); // initialize serial communication
```

```
260
                                 // Uncomment to turn off serial comms
       //Serial.end();
       Serial.println("Mobility Chair Startup Routine");
261
262
263
       //Start the I2C communication for the MPU9250
264
       while(!Serial) {}
         //start communication with IMU
265
                                                    // NOTE: The sensor zeros its
266
         status = IMU.begin();
     orientation upon initialization.
267
         if (status < 0) {
           Serial.println("IMU initialization unsuccessful");
268
           Serial.println("Check IMU wiring or try cycling power");
269
           Serial.print("Status: ");
270
271
           Serial.println(status);
272
           while(1) {}
273
         }
274
         // setting the accelerometer full scale range to +/-8G
275
         IMU.setAccelRange(MPU9250::ACCEL_RANGE_8G);
276
         // setting the gyroscope full scale range to +/-500 deg/s
277
         IMU.setGyroRange(MPU9250::GYRO_RANGE_500DPS);
278
         // setting DLPF bandwidth to 20 Hz
279
         IMU.setDlpfBandwidth(MPU9250::DLPF BANDWIDTH 20HZ);
280
         // setting SRD to 19 for a 50 Hz update rate
         IMU.setSrd(19);
281
         Serial.println("IMU initialization success");
282
283
         Serial.println();
284
       //End MPU9250 Setup
285
286
       //Initialize the control input buttons
287
       pinMode(forwardButtonPin, INPUT PULLUP);
                                                  // initialize the button pin as a
     input
288
       pinMode(leftButtonPin, INPUT PULLUP);
                                                  // initialize the button pin as a
289
       pinMode(rightButtonPin, INPUT PULLUP);
                                                 // initialize the button pin as a
290
       pinMode(reverseButtonPin, INPUT PULLUP);
                                                 // initialize the button pin as a
291
       pinMode(enableButtonPin, INPUT PULLUP);
292
       //Initialize the mode selection inputs (rotary switch)
293
294
       pinMode(mode1Pin, INPUT_PULLUP);
                                                  //Used to set mode 1 (full speed)
295
       pinMode(mode2Pin, INPUT_PULLUP);
                                                  //Used to set mode 2 (medium speed)
       pinMode(mode3Pin, INPUT_PULLUP);
296
                                                 //Used to set mode 3 (slow speed)
297
298
       //Initialize the outputs for the motor controller
299
       pinMode(leftMotorPWMPin, OUTPUT);
                                                  //Used to control the speed of the
     left motor
300
       pinMode(leftMtrDirPin, OUTPUT);
                                                  //Used to control the direction of
     the left motor
301
       pinMode(rightMotorPWMPin, OUTPUT);
                                                 //Used to control the speed of the
     right motor
302
       pinMode(rightMtrDirPin, OUTPUT);
                                                  //Used to control the direction of
     the right motor
303
304
       //Initialize the inputs for the motor encoders
305
       encoder main channel (uses interrupt)
306
       pinMode(encoderLeftBPin, INPUT PULLUP);
                                                  //Used to detect state of left
     encoder quadrature channel to determine direction
307
       pinMode(encoderRightAPin, INPUT_PULLUP);
                                                 //Used to count pulses on the right
     encoder main channel (uses interrupt)
308
       pinMode(encoderRightBPin, INPUT_PULLUP); //Used to detect state of right
     encoder quadrature channel to determine direction
309
310
       //Initialize the system indicators outputs (lights/warnings)
311
       pinMode(systemEnabledIndPin, OUTPUT);
```

```
312
       pinMode(alarmIndPin, OUTPUT);
       pinMode(leftIndPin, OUTPUT);
313
314
       pinMode(rightIndPin, OUTPUT);
315
       pinMode(forwardIndPin, OUTPUT);
       pinMode(reverseIndPin, OUTPUT);
316
       pinMode(audioIndPin, OUTPUT);
317
318
319
       //Initialize the sensor related inputs
320
       pinMode(batteryVoltagePin, INPUT);
321
       pinMode(proxEncPin, INPUT);
322
       pinMode(distIR1Pin, INPUT);
323
324
       //SETUP INTERRUPTS
       attachInterrupt(digitalPinToInterrupt(encoderLeftAPin), readLeftMtrSpd, RISING);
325
     //Whenever the Left encoder has a rising pulse call the routine
326
       attachInterrupt(digitalPinToInterrupt(encoderRightAPin), readRightMtrSpd,
                //Whenever the Right encoder has a rising pulse call the routine
327
       attachInterrupt(digitalPinToInterrupt(enableButtonPin), StopOverrideISR,
     FALLING);
                 //Used for the therapist/supervisor enable/disable pushbutton
     momentary switch
328
       attachInterrupt(digitalPinToInterrupt(proxEncPin), enclosureSensorISR, CHANGE);
     //Used to detect a change in the prox sensor that monitors the enclosure (was
     falling)
329
330
       //Flash some lights so that you know the system started successfully.
331
       startupFlash();
332
333
     }
334
    void startupFlash() {
       //Flash the control lights so that the user knows the system has been turned on.
335
336
       digitalWrite(leftIndPin, HIGH);
337
       delay(250);
338
       digitalWrite(forwardIndPin, HIGH);
339
       delay(250);
340
       digitalWrite(rightIndPin, HIGH);
341
       delay(250);
342
       digitalWrite(reverseIndPin, HIGH);
343
       delay(750);
344
       digitalWrite(leftIndPin, LOW);
345
       digitalWrite(rightIndPin, LOW);
       digitalWrite(forwardIndPin, LOW);
346
347
       digitalWrite(reverseIndPin, LOW);
348
     }
349
     /*-----
350
     _____*/
351
     /*--MAIN PROGRAM LOOP---MAIN PROGRAM LOOP---MAIN PROGRAM LOOP---MAIN PROGRAM
     LOOP---MAIN PROGRAM LOOP---*/
352
     //THIS IS THE MAIN PROGRAM LOOP (it calls all the subroutines)
353
     void loop() {
354
       if (enableBtnState crnt != enableBtnState prev) {
                                                             // Check if a change
     has occurred to the enable/stop supervisor button
355
         StopOverrideHandler();
                                                              // If a change has
     occurred, go to the handler subroutine to enable or disable the system
356
       }
357
       if (enclosureState crnt != enclosureState prev) {
358
359
         enclosureProxHandler();
360
361
362
       if (alarmStatus > 0) {
363
         systemEnableState = LOW;
                                                              //Turn off the controls
```

```
if they are on
364
         digitalWrite(systemEnabledIndPin, systemEnableState);
                                                                   //Turn off the
     indicator for the controls
365
         warningIndicator();
                                                                    //Toggle the warning
     LED the appropriate number of times to indicate error
366
367
         switch (alarmStatus) {
                                                                   //Print the cause of
     the alarm to the serial monitor
368
             Serial.println("BATTERY ALARM - EC01");
369
370
             break;
371
           case 2:
372
             Serial.println("ENCLOSURE ALARM - EC02");
373
374
           case 3:
375
             Serial.println("DROPOFF ALARM - EC03");
376
             break:
377
           case 4:
             Serial.println("TILT ALARM - EC04");
378
379
380
           case 5:
             Serial.println("TEMPERATURE ALARM - EC05");
381
382
383
         }
       }
384
385
386
       if (systemEnableState == HIGH) {
                                                                        // Only allow the
     controls to be used if system is enabled
387
         Serial.println("CONTROLS ENABLED");
                                                                        // Print to the
     serial monitor for troubleshooting
388
         controlLoop();
                                                                        // Go to subroutine
     to update the control inputs as they occur
389
390
         //Update these functions only every 30ms
391
         if (millis() > shortTimer + shortInterval) {
           shortTimer = millis();
392
                                                                      //Update the system
     timer every 30ms
                                                                      //Duration controls
393
           updateTimerValue();
     pressed or released
394
                                                                      //Read the actual
           calcMtrSpd();
     encoder speed value
395
                                                                      //Motor PID control
           pidLoop();
396
397
398
         //Update these functions every 50ms
399
         if (millis() > mediumTimer + mediumInterval) {
400
           mediumTimer = millis();
                                                                      //Update the system
     timer every 50ms.
401
           updateTargetSpeed();
                                                                      //Set the target for
     the PID control based on current speed and duration of control inputs.
402
           IRdistance();
                                                                      //Monitor for sudden
     increase in distance that indicates a ledge is present.
403
           //Serial.println("MED INTERVAL");
404
         }
405
406
       } else {
407
         digitalWrite(leftMotorPWMPin, 0);
                                                                        // The system is
     disabled, turn off both motors
408
         digitalWrite(rightMotorPWMPin, 0);
409
         leftTargetSpeed crnt = 0;
                                                    //Ensure that the target system speed
     is reset if it was interrupted while in motion
410
         rightTargetSpeed crnt = 0;
411
         digitalWrite(forwardIndPin, LOW);
                                                     //Turn off the control LED indicator
     for visual feedback
412
         digitalWrite(leftIndPin, LOW);
                                                     //Turn off the control LED indicator
```

9 of 32

```
for visual feedback
413
       digitalWrite(rightIndPin, LOW);
                                              //Turn off the control LED indicator
     for visual feedback
414
        digitalWrite(reverseIndPin, LOW);
                                                 //Turn off the control LED
     indicator for visual feedback
       digitalWrite(audioIndPin, LOW);
415
                                            //Send a signal too the UNO board to
     turn off audio feedback
       Serial.println("SYSTEM DISABLED");
416
                                                               // Print to the
     serial monitor for troubleshooting.
417
        Serial.println();
418
      }
419
420
      //Update these functions every 250ms
421
      if (millis() > longTimer + longInterval) {
        longTimer = millis();
422
                                                               // Update the
     system timer every 250ms. Used for functions that do not need instant response.
423
        modeSelect();
                                                               // Determine which
     speed mode is active.
424
        batteryMonitor();
                                                               // Monitor the
     status of the battery for charge
425
       tiltSensor();
                                                               // Monitor tilt in
     the Z axis. This sensor is setup to refresh at a maximum of 20Hz or 0.05s.
426
        //Serial.println("LONG INTERVAL");
427
      }
     }
428
     /*-
429
     ----*/
430
     TSR-----
     -----*/
431
     /*----
    ISR-----
432
     /*INTERRUPT SERVICE FUNCTIONS HERE*/
    //These functions determine the encoder counts and handle the therapist override
433
     switch
434
435
    //Interrupt Routine for reading the left encoder
436
    void readLeftMtrSpd() {
437
      // Increment value for each pulse from encoder, this is triggered by an interrupt
438
      encoderLeftValue counter++;
439
      stateEncoderL_B = digitalRead(encoderLeftBPin);
440
     }
441
442
     //Interrupt Routine for reading the right encoder
443
    void readRightMtrSpd() {
444
      // Increment value for each pulse from encoder, this is triggered by an interrupt
445
      encoderRightValue_counter++;
446
      stateEncoderR B = digitalRead(encoderRightBPin);
447
    }
448
449
     //Interrupt Routine for the therapist stop button
450
    void StopOverrideISR() {
451
                                                  //Record the time the
      enableTime crnt = millis();
     stop/enable button was pressed
452
      enableBtnState crnt = !enableBtnState crnt;
453
     }
454
     //Interrupt Routine for the proximity sensor checking the enclosure lid
455
456
    void enclosureSensorISR() {
457
      enclosureTime_crnt = millis();
458
      enclosureState crnt = !enclosureState crnt;
459
      //Serial.println("ENCLOSURE STATE CHANGE");
460
      /*for (byte i = 0; i < 10; i++) {
```

```
Serial.println("ENCLOSURE STATE CHANGED
461
    462
      }
      */
463
464
    }
465
    /*-CONTROL
    LOOP-----
    _____*/
466
    /*----CONTROL
    I,00P-----
    -----*/
    /*----CONTROL
467
    LOOP-----
    */
468
    //This function reads the control inputs
469
    void controlLoop() {
      //Not timer dependent
470
471
472
      //Read the control input pins
      fwdBtnState_crnt = digitalRead(forwardButtonPin);
                                                       // read the forward
473
    button input
474
      leftBtnState crnt = digitalRead(leftButtonPin);
                                                       // read the left button
    input
475
      rightBtnState crnt = digitalRead(rightButtonPin);
                                                       // read the right
    button input
      revBtnState crnt = digitalRead(reverseButtonPin);
                                                       // read the reverse
476
    button input
477
478
      controlState_crnt = fwdBtnState_crnt + leftBtnState_crnt + rightBtnState_crnt +
    revBtnState crnt;
479
      //If a change in the control has occurred (first time pressed or released), keep
480
    track of the time
      /*NOTES: May be able to streamline this with math instead of reading each state*/
481
      if (controlState_crnt != controlState_prev) {
482
483
        updateStartStopTimes();
484
485
486
      fwdBtnState_prev = fwdBtnState_crnt;
                                                     // save fwdbtn state for
    next loop so that a change can be detected each iteration
487
      leftBtnState_prev = leftBtnState_crnt;
488
      rightBtnState prev = rightBtnState crnt;
489
      revBtnState prev = revBtnState crnt;
490
      controlState_prev = controlState_crnt;
491
492
      //Write the motor controller values
493
494
      //Left Motor Values
495
      byte leftMotorPWM_value = abs(PID_Ltotal);
                                                    //PWM value can only be
    positive (but direction is controlled by sign)
496
      if (PID_Ltotal > 0) {
                                                     //Set the motor direction
    based on sign of PWM value
497
        digitalWrite(leftMtrDirPin, HIGH);
                                                     //Set the motor direction
498
499
        digitalWrite(leftMtrDirPin, LOW);
500
501
      analogWrite(leftMotorPWMPin, leftMotorPWM_value);
                                                     //Set the motor speed
    based on the target calculated (first pass is zero)
502
503
      //Right Motor Values
504
      byte rightMotorPWM value = abs(PID Rtotal);
      if (PID_Rtotal > 0) {
                                                     //Set the motor direction
505
    based on sign of PWM value
        digitalWrite(rightMtrDirPin, HIGH);
506
                                                     //Set the motor direction
507
      } else {
```

```
508
      digitalWrite(rightMtrDirPin, LOW);
509
     }
510
     analogWrite(rightMotorPWMPin, rightMotorPWM value);
                                              //Set the motor speed
   based on the target calculated (first pass is zero)
511
    /*-MODE
512
   SELECTION-----
    _____*/
    /*----MODE
513
   SELECTION-----
    ----*/
    /*----MODE
514
   SELECTION-----
    ----*/
515
   void modeSelect() {
     //Timer dependent because it is not expected to change frequently
516
517
     //Mode inputs will be pullup so LOW will indicate the selection
518
     519
     //mode2Status = digitalRead(mode2Pin); //Medium Speed (This pin was not
   physically implemented because the switch is only 2 position.)
520
     mode3Status = digitalRead(mode3Pin);
                                //Full Speed
521
522
     if (mode1Status == LOW) {
      modeFactorFinal = mode3Factor;
523
      Serial.println("MODE 3 - Slow Speed");
524
525
     else if (mode3Status == LOW) {
526
      modeFactorFinal = modelFactor;
527
528
      Serial.println("MODE 1 - Full Speed");
529
     } else {
530
      modeFactorFinal = mode2Factor;
531
      Serial.println("MODE 2 - Medium Speed");
532
     }
533
    }
534
    /*-BATTERY VOLTAGE
   MONTTOR-----
    */
    /*----BATTERY VOLTAGE
535
   MONITOR-----
    ----*/
    /*----BATTERY VOLTAGE
536
   MONITOR-----
   ----*/
   void batteryMonitor() {
537
538
     float batteryPinADC = analogRead(batteryVoltagePin);
539
     unsigned int batteryVoltage = batteryPinADC * 2500 / 1024;
540
     Serial.println();
541
     Serial.print("Battery Voltage Level (x100) = ");
542
     Serial.print(batteryVoltage);
543
     Serial.println();
544
     if (batteryVoltage < lowBatteryLevel) {</pre>
545
      lowBatteryTime_crnt = millis();
546
      if (lowBatteryTime crnt - lowBatteryTime prev > 1000) {
547
548
        Serial.println("LOW BATTERY LEVEL");
549
        alarmStatus = 1;
550
      lowBatteryTime_prev = lowBatteryTime_crnt;
551
552
553
     Serial.println();
554
555
    /*-IR DISTANCE
    -----*/
556
    /*----IR DISTANCE
```

```
----*/
557
    /*----IR DISTANCE
    SENSOR-----
    ----*/
    void IRdistance() {
558
     float volts = analogRead(distIR1Pin) * 0.003125; // value from sensor * (5/1024)
559
      int distance = 13 * pow(volts, -1); // worked out from datasheet graph
560
561
      if (distance <= 100) {</pre>
562
       Serial.print("Distance: ");
563
564
       Serial.print(distance);
                            // print the distance
       Serial.println(" inches");
565
566
567
      if (distance > IRdistMAX) {
568
569
       Serial.println("DROPOFF DETECTED ERROR");
570
       alarmStatus = 3;
571
      }
572
      Serial.println();
573
    }
    /*-MPU9250 INERTIAL
574
    SENSOR-----
    ----*/
    /*----MPU9250 INERTIAL
575
    SENSOR-----
    ----*/
576
    /*----MPU9250 INERTIAL
    SENSOR-----
    ----*/
577
    void tiltSensor() {
578
      // Read the sensor from the MPU9250
579
      IMU.readSensor();
580
      // Display the data from the {\tt Z} axis
581
582
      Zmss = IMU.getAccelZ mss();
      temp = IMU.getTemperature_C();
583
584
585
      Serial.println("IMU Data");
      Serial.print("X: ");
586
587
      Serial.print(IMU.getAccelX_mss(),3); //3 decimal places
588
      Serial.print("\t");
589
      Serial.print("Y: ");
590
      Serial.print(IMU.getAccelY mss(),3);
591
      Serial.print("\t");
      Serial.print("Z: ");
592
593
      Serial.print(Zmss,3);
594
      Serial.print("\t");
      Serial.print("Temperature: ");
595
596
      Serial.print(temp, 3);
597
      Serial.println("°C");
598
599
      // If the tilt from the Z axis is greater than the limit, flag an error
    condition.
600
      if (abs(Zmss) < Zmss_limit) {</pre>
       Serial.println("TILT DETECTED ERROR");
601
602
       alarmStatus = 4;
603
      }
604
605
      // If the temperature inside the enclosure reaches the max temp shut motors off.
      if (temp > temp_limit) {
606
       Serial.println("OVER MAX TEMPERATURE ERROR");
607
608
       alarmStatus = 5;
609
610
      Serial.println();
```

```
611
    /*-
612
    START STOP TIMES-----
    _____*/
613
    START STOP TIMES-----
    _____*/
614
    START STOP TIMES-----
    _____*/
615
    //THIS UPDATES THE START AND STOP TIMES STORED AS VARIABLES IN THE GLOBAL SCOPE
    void updateStartStopTimes() {
616
      // Not timer dependent, is called by controlLoop()
617
      //Runs the instant any motion control is pressed or released, "PULLUP" so LOW
618
    means pressed
619
620
      //Forward control
621
      if (fwdBtnState crnt != fwdBtnState prev) {
622
        if (fwdBtnState_crnt == LOW) {
623
         fwdPressedTime = millis();
                                             //Stores the instant forward button
    was pressed
624
         fwdTimeInactive = 0;
                                             //Clears the inactivity timer
625
         digitalWrite(forwardIndPin, HIGH);
                                             //Turn on the control LED indicator
    for visual feedback
                                             //Send a signal too the UNO board
626
         digitalWrite(audioIndPin, HIGH);
    to play audio feedback
627
       } else {
                                             // the button was just released as
    the other state is HIGH
         fwdReleasedTime = millis();
628
                                             //Stores the instant the forward
    button was released
629
         fwdTimeActive = 0;
                                             //Clears the activity timer
         digitalWrite(forwardIndPin, LOW);
630
                                             //Turn off the control LED
    indicator for visual feedback
631
         turn off audio feedback
632
      }
633
634
635
      //Left Control
      if (leftBtnState_crnt != leftBtnState_prev) {
636
637
        if (leftBtnState_crnt == LOW) {
         leftPressedTime = millis();
                                             //Stores the instant forward button
638
    was pressed
639
         leftTimeInactive = 0;
                                             //Clear inactivity timer
640
         digitalWrite(leftIndPin, HIGH);
                                             //Turn on the control LED indicator
    for visual feedback
641
         digitalWrite(audioIndPin, HIGH);
                                             //Send a signal too the UNO board
    to play audio feedback
642
                                             // the button was just released as
        } else {
    the other state is HIGH
643
         leftReleasedTime = millis();
                                             //Stores the instant the button was
    released
644
         leftTimeActive = 0;
                                             //Clears the activity timer
         digitalWrite(leftIndPin, LOW);
645
                                             //Turn off the control LED
    indicator for visual feedback
646
         digitalWrite(audioIndPin, LOW);
                                            //Send a signal too the UNO board to
    turn off audio feedback
647
        }
648
      }
649
      //Right Control
650
651
      if (rightBtnState_crnt != rightBtnState_prev) {
652
        if (rightBtnState crnt == LOW) {
653
         rightPressedTime = millis();
                                             //Stores the instant forward button
    was pressed
```

```
654
         rightTimeInactive = 0;
                                              //Clear inactivity timer
655
         digitalWrite(rightIndPin, HIGH);
                                              //Turn on the control LED indicator
    for visual feedback
656
         digitalWrite(audioIndPin, HIGH);
                                              //Send a signal too the UNO board
    to play audio feedback
657
       } else {
                                              // the button was just released as
    the other state is HIGH
658
         rightReleasedTime = millis();
                                              //Stores the instant the button was
    released
659
                                              //Clears the activity timer
         rightTimeActive = 0;
         digitalWrite(rightIndPin, LOW);
                                             //Turn off the control LED
660
    indicator for visual feedback
                                      //Send a signal too the UNO board to
661
         digitalWrite(audioIndPin, LOW);
    turn off audio feedback
662
       }
663
664
665
      //Reverse Control
666
      if (revBtnState_crnt != revBtnState_prev) {
667
        if (revBtnState crnt == LOW) {
668
         revPressedTime = millis();
                                              //Stores the instant forward button
    was pressed
         revTimeInactive = 0;
669
                                              //Clear inactivity timer
                                              //Turn on the control LED indicator
670
         digitalWrite(reverseIndPin, HIGH);
    for visual feedback
671
         digitalWrite(audioIndPin, HIGH);
                                              //Send a signal too the UNO board
    to play audio feedback
672
                                              // the button was just released as
        } else {
    the other state is HIGH
         revReleasedTime = millis();
                                              //Stores the instant the forward
673
    button was released
674
         revTimeActive = 0;
                                             //Clears the activity timer
         digitalWrite(reverseIndPin, LOW);
                                             //Turn off the control LED
675
    indicator for visual feedback
676
         turn off audio feedback
677
       }
678
      }
679
    }
    /*--CONTROL TIME
680
    VALUES-----
    */
681
    /*----CONTROL TIME
    VALUES------
    -----*/
    /*----CONTROL TIME
682
    VALUES-----
    ----*/
683
    void updateTimerValue() {
684
      //Timer Dependent
685
      timeCurrent = millis();
686
      // This function runs ALL THE TIME but only calculates a time while the forward
    button is actively being pressed and for one cycle after its release
687
      if (fwdBtnState_crnt == fwdBtnState_prev && fwdBtnState_crnt == 0) { // FWD
688
    Button is active and loop has passed at least once
689
        fwdTimeActive = timeCurrent - fwdPressedTime;
        //fwdReleasedTime = 0;
690
691
        Serial.print("The FWD button has been held for");
        Serial.print('\t');
692
693
        Serial.print(fwdTimeActive);
694
        Serial.println(" ms.");
695
      if (fwdBtnState_crnt == fwdBtnState_prev && fwdBtnState_crnt == 1) { //FWD Button
    is inactive and loop has passed at least once
```

```
697
         fwdTimeInactive = timeCurrent - fwdReleasedTime;
         Serial.print("The FWD button has been released for");
Serial.print('\t');
698
699
700
         Serial.print(fwdTimeInactive);
701
         Serial.println(" ms.");
702
       }
703
       //LEFT
       if (leftBtnState crnt == leftBtnState prev && leftBtnState crnt == 0) { // LEFT
704
     Button is active and loop has passed at least once
705
         leftTimeActive = timeCurrent - leftPressedTime;
706
         //leftReleasedTime = 0;
707
         Serial.print("The LEFT button has been held for");
         Serial.print('\t');
708
709
         Serial.print(leftTimeActive);
710
         Serial.println(" ms.");
711
       }
712
       if (leftBtnState_crnt == leftBtnState_prev && leftBtnState_crnt == 1) { //LEFT
     Button is inactive and loop has passed at least once
713
         leftTimeInactive = timeCurrent - leftReleasedTime;
714
         Serial.print("The LEFT button has been released for");
         Serial.print('\t');
715
716
         Serial.print(leftTimeInactive);
717
         Serial.println(" ms.");
718
       }
       //RIGHT
719
       if (rightBtnState crnt == rightBtnState prev && rightBtnState crnt == 0) {
720
     //Right Button is active and loop has passed at least once
721
         rightTimeActive = timeCurrent - rightPressedTime;
         //rightReleasedTime = 0;
722
723
         Serial.print("The RIGHT button has been held for");
724
         Serial.print('\t');
725
         Serial.print(rightTimeActive);
726
         Serial.println(" ms.");
727
       }
       if (rightBtnState crnt == rightBtnState prev && rightBtnState crnt == 1) {
728
     //right Button is inactive and loop has passed at least once
729
         rightTimeInactive = timeCurrent - rightReleasedTime;
         Serial.print("The RIGHT button has been released for");
Serial.print('\t');
730
731
732
         Serial.print(rightTimeInactive);
733
         Serial.println(" ms.");
734
       }
735
       //REVERSE
736
       if (revBtnState crnt == revBtnState prev && revBtnState crnt == 0) { //Reverse
     Button is active and loop has passed at least once
737
         revTimeActive = timeCurrent - revPressedTime;
738
         //revReleasedTime = 0;
         Serial.print("The REV button has been held for");
739
         Serial.print('\t');
740
741
         Serial.print(revTimeActive);
         Serial.println(" ms.");
742
743
       }
744
       if (revBtnState_crnt == revBtnState_prev && revBtnState_crnt == 1) { //Reverse
     Button is inactive and loop has passed at least once
745
         revTimeInactive = timeCurrent - revReleasedTime;
         Serial.print("The REV button has been released for");
746
         Serial.print('\t');
747
748
         Serial.print(revTimeInactive);
749
         Serial.println(" ms.");
750
751
     }
752
     -----*/
753
```

```
/*______
        -____*/
754
    ----*/
755
    void calcMtrSpd() {
      //Motor/Encoder Speed
756
757
      encoderLeftValue prev = encoderLeftValue crnt;
                                                       //Update the previous
    value with the current value
      encoderLeftValue_crnt = encoderLeftValue_counter;
758
                                                       //Store how many
    pulses occurred for calculation
                                                       //Reset the encoder
      encoderLeftValue counter = 0;
759
    value each loop (after 50ms)
      //Right Encoder Speed
760
      encoderRightValue prev = encoderRightValue crnt;
761
                                                       //Update the previous
    value with the current value
762
      encoderRightValue_crnt = encoderRightValue_counter;
                                                      //Store how many
    pulses occurred for calculation
      encoderRightValue_counter = 0;
763
                                                       //Reset the encoder
    value each loop (after 50ms)
764
765
      //Motor/Encoder Direction
766
      if (stateEncoderL_B == LOW) {
                                                       //Determine the
    direction the LEFT motor is spining based on quadrature signal
       encoderLeftDir_txt = "CW ";
767
768
      } else {
       encoderLeftValue_crnt = encoderLeftValue_crnt * -1;  //Set the encoder value
769
    to negative if CCW rotation
770
       encoderLeftDir_txt = "CCW";
771
      }
772
                                                       //Determine the
773
      if (stateEncoderR B == LOW) {
    direction the RIGHT motor is spining based on quadrature signal
774
       encoderRightDir txt = "CW ";
775
       encoderRightValue crnt = encoderRightValue crnt * -1; //Set the encoder
776
    value to negative if CCW rotation
777
       encoderRightDir txt = "CCW";
778
779
      //Display the speed in the serial monitor for troubleshooting and tuning
      if (encoderLeftValue_prev != encoderLeftValue_crnt || encoderRightValue_prev !=
780
                           //Only update the serial monitor if there was a
    encoderRightValue crnt) {
    change
       Serial.print("PULSES per 50ms[L/R]: ");
781
       Serial.print('\t');
782
783
       Serial.print(encoderLeftValue crnt);
784
       Serial.print('\t');
785
       Serial.print(encoderLeftDir txt);
       Serial.print(" / ");
786
787
       Serial.print(encoderRightValue crnt);
       Serial.print('\t');
788
789
       Serial.println(encoderRightDir txt);
790
      }
791
    }
792
    /*-----
    _____*/
793
    /*_____
    _____*/
794
    _____*/
    void pidLoop() {
795
796
      //Set a maximum PWM value for this subroutine (normally 255, but for
```

```
demonstration purposes backed off to avoid battery fault)
797
       const unsigned char maxPWM = 130;
798
799
       //Get the error for each motor setpoint
800
       errorLeft_crnt = leftTargetSpeed_crnt - encoderLeftValue_crnt;
       errorRight crnt = rightTargetSpeed crnt - encoderRightValue crnt;
801
802
803
       //Calculate Proportional Term
804
       PID Lp = kp * (float)errorLeft crnt;
       PID_Rp = kp * (float)errorRight_crnt;
805
806
807
       //Calculate Derivative Term
808
       int errorLeft_diff = errorLeft_crnt - errorLeft_prev;
809
       PID_Ld = kd * (((float)errorLeft_crnt - (float)errorLeft_prev) /
      (float)shortInterval);
810
811
       int errorRight_diff = errorRight_crnt - errorRight_prev;
812
       PID Rd = kd * (((float)errorRight crnt - (float)errorRight prev) /
      (float)shortInterval);
813
814
       //Calculate Left Integral Term
815
       if (-3 < errorLeft_crnt && errorLeft_crnt < 3) {</pre>
         PID_Li = PID_Li + (ki * (float)errorLeft_crnt);
816
817
818
       } else {
819
820
         PID Li = 0; //When both motors are added, this is probably the place to
     compare their speed
821
822
823
       //Calculate Right Integral Term
824
       if (-3 < errorRight crnt && errorRight crnt < 3) {</pre>
825
         PID Ri = PID Ri + (ki * (float)errorRight crnt);
826
827
       } else {
828
829
         PID Ri = 0; //When both motors are added, this is probably the place to
     compare their speed
830
831
       }
832
833
       //Sum for total PID control signal
834
       PID_Ltotal = PID_Lp + PID_Li + PID_Ld;
835
       PID_Rtotal = PID_Rp + PID_Ri + PID_Rd;
836
837
       //The motor PWM value cannot exceed maxPWM (if it does it is maxed out)
838
       if (PID_Ltotal > maxPWM) {
         PID Ltotal = maxPWM;
839
840
841
       if (PID_Ltotal < -maxPWM) {</pre>
842
         PID Ltotal = -maxPWM;
843
844
845
       if (PID Rtotal > maxPWM) {
846
         PID_Rtotal = maxPWM;
847
       if (PID Rtotal < -maxPWM) {</pre>
848
849
         PID Rtotal = -maxPWM;
850
       }
851
       Serial.print("PID Values [L/R]: ");
852
       Serial.print('\t');
853
       Serial.print(PID_Ltotal);
854
855
       Serial.print(" / ");
856
       Serial.println(PID_Rtotal);
```

```
857
    /*--STOP BUTTON
858
    HANDLER-----
    ----*/
859
    /*----STOP BUTTON
    HANDLER-----
    */
    /*----STOP BUTTON
860
    HANDLER------
    ----*/
    void StopOverrideHandler() {
861
     if (enableTime crnt - enableTime prev > 350) {
862
                                            //If the interrupt has been
    toggled within 250ms its probably a switch bounce, ignore it
       systemEnableState = !systemEnableState;
863
                                             //Only change the system state
    if signals are recieved more than 350ms appart
       digitalWrite(systemEnabledIndPin, systemEnableState);
864
865
866
       //Clear any existing system alarms.
867
       alarmStatus = 0;
868
       digitalWrite(alarmIndPin, LOW);
869
870
       boolean proxValue = digitalRead(proxEncPin);
                                              //Read the proximity sensor
871
       if (proxValue == LOW) {
                                              //If the sensor output is
    high the lid is open
872
        enclosureState_prev = enclosureState_crnt;
                                              //Clear the enclosure alarm
    if it has been shut.
       } else {
873
         alarmStatus = 2;
874
875
       }
876
     }
877
     enableTime_prev = enableTime_crnt;
878
     enableBtnState prev = enableBtnState crnt;
879
     ///* Troubleshooting
880
     Serial.print("STOP");
     Serial.print('\t');
881
     Serial.print("STOP");
882
     Serial.print('\t');
883
     Serial.print("STOP");
884
     Serial.print('\t');
Serial.print("STOP");
885
886
     Serial.print('\t');
Serial.print("STOP");
887
888
     Serial.print('\t');
889
     Serial.print("STOP");
890
891
     Serial.print('\t');
     Serial.println("STOP - - - - - - - ");
892
893
     Serial.print("System state is: ");
894
     Serial.println(systemEnableState);
     //*/
895
896
    }
897
    /*--ENCLOSURE PROX
    HANDLER-----
    ----*/
    /*----ENCLOSURE PROX
898
    HANDLER-----
    ----*/
899
    /*----ENCLOSURE PROX
    HANDLER-----
    ----*/
900
    /*
901
      NOTES:
902
      The proximity sensor input and output is 5V which cannot be powered or directly
    read by the DUE.
903
      A logic level converter must be used to step 5V down to 3V.
904
```

```
905
    void enclosureProxHandler() {
906
907
      if (enclosureTime crnt - enclosureTime prev > 100) {
                                                          //If the interrupt has
     been toggled within 350ms its probably a ground bounce, ignore it
908
                                                          //enclosureTime crnt is
     updated by the enclosureISR when proxEncPin changes.
909
        boolean proxValue = digitalRead(proxEncPin);
                                                          //Read the proximity
     sensor
910
        if (proxValue == HIGH) {
911
                                                          //If the sensor output
     is high the lid is open
912
          Serial.println();
          Serial.println("ENCLOSURE OPEN");
913
          Serial.println("ENCLOSURE OPEN");
Serial.println("ENCLOSURE OPEN");
914
915
          Serial.println("ENCLOSURE OPEN");
916
917
          Serial.println("ENCLOSURE OPEN");
918
          Serial.println();
919
          alarmStatus = 2;
                                                         //Set the warning state
     flag to enclosure open
920
          enclosureTime prev = 0;
          enclosureTime_crnt = 1;
921
922
        if (proxValue == LOW) {
923
924
          Serial.println();
          Serial.println("ENCLOSURE CLOSED");
Serial.println("ENCLOSURE CLOSED");
925
926
          Serial.println("ENCLOSURE CLOSED");
927
928
          Serial.println("ENCLOSURE CLOSED");
          Serial.println("ENCLOSURE CLOSED");
929
930
          Serial.println();
931
        }
932
      }
933
      enclosureTime prev = enclosureTime crnt;
934
      //enclosureState_prev = enclosureState_crnt;
                                                       //Prevent re-executing
     this loop until the system has been reset
935
936
937
     /*--WARNING
     INDICATOR-----
     _____*/
938
     /*----WARNING
     INDICATOR-----
     */
939
     /*----WARNING
    INDICATOR-----
     ----*/
940
    void warningIndicator() {
941
      warningTimer_crnt = millis();
942
943
      switch (alarmStatus) {
                                                         //Blink the indicator a
     specific amount of times to indicate the alarm type
944
        case 1: //Battery Alarm
          if (warningTimer crnt > warningTimer prev && warningTimer crnt <
945
     warningTimer prev + 250) {
946
            digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
     level)
947
948
          if (warningTimer_crnt > warningTimer_prev + 250) {
            digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
949
     level)
950
951
          if (warningTimer crnt > warningTimer prev + 1500) {
952
            warningTimer prev = warningTimer crnt;
953
```

```
954
           break;
955
956
         case 2:
                   //Enclosure Alarm
957
           if (warningTimer_crnt > warningTimer_prev && warningTimer_crnt <</pre>
                                         // On if 0<timer<250
     warningTimer_prev + 250) {
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
958
     level)
959
           if (warningTimer_crnt > warningTimer_prev + 250 && warningTimer crnt <</pre>
960
     warningTimer prev + 500) { // Off if 250<timer<500</pre>
961
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
     level)
962
           if (warningTimer crnt > warningTimer prev + 500 && warningTimer crnt <
963
     warningTimer prev + 750) { // On if 500<timer<750</pre>
964
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
     level)
965
966
            if (warningTimer_crnt > warningTimer_prev + 750) {
967
             digitalWrite(alarmIndPin, LOW);
                                               // turn the LED on (HIGH is the voltage
     level)
                              // Off if timer>750
968
           }
969
           if (warningTimer_crnt > warningTimer_prev + 1750) {
     // Reset sequence after 1000ms delay
970
             warningTimer_prev = warningTimer_crnt;
971
972
           break;
973
974
                    //Drop off alarm (IR sensor)
         case 3:
           if (warningTimer crnt > warningTimer prev && warningTimer crnt <
975
                                        // On if 0<timer<250
     warningTimer prev + 250) {
976
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
     level)
977
           }
978
           if (warningTimer_crnt > warningTimer_prev + 250 && warningTimer_crnt <
     warningTimer prev + 500) { // Off if 250<timer<500</pre>
979
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
     level)
980
           if (warningTimer crnt > warningTimer prev + 500 && warningTimer crnt <
981
     warningTimer_prev + 750) { // On if 500<timer<750</pre>
982
             digitalWrite(alarmIndPin, HIGH);
                                                // turn the LED on (HIGH is the voltage
     level)
983
            }
984
           if (warningTimer crnt > warningTimer prev + 750 && warningTimer crnt <
     warningTimer prev + 1000) { // Off if 750<timer<1000</pre>
985
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
     level)
986
987
           if (warningTimer crnt > warningTimer prev + 1000 && warningTimer crnt <
     warningTimer prev + 1250) { // On if 1000<timer<1250
988
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
     level)
989
990
            if (warningTimer_crnt > warningTimer_prev + 1250) {
991
             digitalWrite(alarmIndPin, LOW);
                                               // turn the LED on (HIGH is the voltage
     level)
                              // Off if timer>1250
992
           if (warningTimer_crnt > warningTimer_prev + 2250) {
993
     // Reset sequence after 1000ms delay
994
             warningTimer_prev = warningTimer_crnt;
995
996
           break;
997
998
         case 4:
                    //Tilt alarm (accelerometer)
```

```
999
           if (warningTimer_crnt > warningTimer_prev && warningTimer_crnt <</pre>
     warningTimer_prev + 250) {
                                        // On if 0<timer<250
100
 0
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
     level)
100
 1
           }
100
           if (warningTimer crnt > warningTimer prev + 250 && warningTimer crnt <
     warningTimer prev + 500) { // Off if 250<timer<500</pre>
100
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
     level)
100
           }
  4
100
           if (warningTimer_crnt > warningTimer_prev + 500 && warningTimer_crnt <
 5
     warningTimer prev + 750) { // On if 500<timer<750</pre>
100
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
 6
     level)
100
           }
 7
100
           if (warningTimer_crnt > warningTimer_prev + 750 && warningTimer_crnt <
 8
     warningTimer prev + 1000) { // Off if 750<timer<1000</pre>
100
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
 9
     level)
101
           }
 0
101
           if (warningTimer crnt > warningTimer prev + 1000 && warningTimer crnt <
     warningTimer prev + 1250) {      // On if 1000<timer<1250</pre>
101
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
 2
     level)
101
           }
  3
101
           if (warningTimer_crnt > warningTimer_prev + 1250 && warningTimer_crnt <
  4
     warningTimer prev + 1500) { // Off if 1250<timer<1500</pre>
101
 5
              digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
     level)
101
           }
  6
101
 7
           if (warningTimer_crnt > warningTimer_prev + 1500 && warningTimer_crnt <
     warningTimer prev + 1750) { // On if 1500<timer<1750
101
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
 8
     level)
101
           }
 9
102
           if (warningTimer crnt > warningTimer prev + 1750) {
 0
102
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
 1
                              // Off if timer>1750
     level)
102
  2
           }
102
  3
           if (warningTimer_crnt > warningTimer_prev + 2750) {
     // Reset sequence after 1000ms delay
```

```
102
             warningTimer prev = warningTimer crnt;
 4
102
 5
           }
102
 6
           break;
102
 7
102
                   //Temperature sensor (included w/mpu9250)
 8
102
           if (warningTimer crnt > warningTimer prev && warningTimer crnt <
 9
                                        // On if 0<timer<250
     warningTimer_prev + 250) {
103
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
 0
     level)
103
           }
 1
103
 2
           if (warningTimer crnt > warningTimer prev + 250 && warningTimer crnt <
     warningTimer prev + 500) { // Off if 250<timer<500</pre>
103
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
 3
     level)
103
           }
103
           if (warningTimer_crnt > warningTimer_prev + 500 && warningTimer_crnt <
 5
     warningTimer_prev + 750) { // On if 500<timer<750</pre>
103
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
 6
     level)
103
 7
           }
103
 8
           if (warningTimer_crnt > warningTimer_prev + 750 && warningTimer_crnt <
     warningTimer prev + 1000) { // Off if 750<timer<1000</pre>
103
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
 9
     level)
104
           }
 0
104
 1
           if (warningTimer crnt > warningTimer prev + 1000 && warningTimer crnt <
     warningTimer prev + 1250) {      // On if 1000<timer<1250</pre>
104
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
 2
     level)
104
           }
104
           if (warningTimer_crnt > warningTimer_prev + 1250 && warningTimer crnt <</pre>
 4
     104
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
 5
     level)
104
  6
           }
104
           if (warningTimer crnt > warningTimer prev + 1500 && warningTimer crnt <
     warningTimer prev + 1750) {    // On if 1500<timer<1750</pre>
104
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
     level)
```

```
104
           }
105
           if (warningTimer_crnt > warningTimer_prev + 2000 && warningTimer_crnt <
     warningTimer_prev + 2250) { // Off if 2000<timer<2250</pre>
105
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
 1
     level)
105
           }
 2
105
           if (warningTimer crnt > warningTimer prev + 2250 && warningTimer crnt <
     warningTimer prev + 2500) {    // On if 2250<timer<2500</pre>
105
             digitalWrite(alarmIndPin, HIGH); // turn the LED on (HIGH is the voltage
     level)
105
           }
 5
105
  6
           if (warningTimer_crnt > warningTimer_prev + 2500) {
105
             digitalWrite(alarmIndPin, LOW); // turn the LED on (HIGH is the voltage
 7
     level)
                             // Off if timer>2500
105
           }
 8
105
           if (warningTimer crnt > warningTimer prev + 3500) {
 9
     // Reset sequence after 1000ms delay
106
             warningTimer prev = warningTimer crnt;
 0
106
 1
           }
106
 2
           break;
106
 3
       }
106
  4
106
 5
     }
106
     /*--TARGET
  6
106
     /*----TARGET
        ----*/
106
     /*----TARGET
106
     void updateTargetSpeed() {
 9
107
       //Timer dependent, called by main Void loop every 50ms
 0
107
 1
107
       //Store all the active timers into an array so that the maximum one can be
 2
     selected.
107
       unsigned long maxTimerArray[] = {fwdTimeActive, leftTimeActive, rightTimeActive,
     revTimeActive};
```

```
107
       const String indexedButtons[] = {"FWD", "LEFT", "RIGHT", "REV"};
 4
107
 5
       byte maxIndex = 0;
107
 6
107
 7
       //Iterate through each item in the array to select the maximum timer value
107
       for (byte i = 0; i < 4; i++) {
 8
107
         if (maxTimerArray[i] >= maxTimerArray[maxIndex]) {
 9
108
 0
           maxIndex = i;
108
 1
         }
108
 2
       }
108
                                                            //The very first iteration
 3
       Serial.print("The max active timer value is: ");
     will be irrelyant (always REV because it is last element of array) and the target
     speed should be set to zero
108
       Serial.print('\t');
                                                               //After controls are input
     the system will be able to determine which input was selected first.
108
       Serial.println(indexedButtons[maxIndex]);
                                                               //This is required to
     determine which direction to twist for holding left + right simultaneously.
108
 6
108
 7
       //Check control state to see how many control inputs are active
108
 8
       switch (controlState crnt) {
108
         case 4: // No controls are active, target should be zero or decelerate to
 9
     zero if needed
109
           // Left and right will be checked for both positive and negative motion to
     see if gradual decel is required
109
           //LEFT TARGET SPEED SETTING
 1
109
 2
           if (leftTargetSpeed crnt > 0) {
                                                                              // Only
     decelerate if one target has not already reached zero.
109
              for (int i = 0; i < DecelArrayCount; i++) {</pre>
                                                                              //i is the
 3
     number of elements in the decelSpeedValue array
109
                if (leftTargetSpeed_crnt > DecelSpeedValues[i]) {
  4
     value only if it is greater than the next index in the array, which is stepped
     through from largest to smallest
109
                  leftTargetSpeed crnt = DecelSpeedValues[i];
 5
109
                  break;
  6
109
 7
                }
109
              }
 8
109
           } else if (leftTargetSpeed_crnt < 0) {</pre>
                                                                             //The motor
 9
     must be moving backwards if negative target speed
110
  0
             for (int i = 0; i < DecelArrayCount; i++) {</pre>
                                                                             //i is the
```

```
number of elements in the decelSpeedValue array
110
                if (-leftTargetSpeed crnt > DecelSpeedValues[i]) {
  1
110
  2
                  leftTargetSpeed_crnt = -DecelSpeedValues[i];
110
  3
                  break;
110
                }
110
              }
  5
110
                                                                               // If value is
            } else {
  6
     not >0 or <0 must be =0
110
              leftTargetSpeed_crnt = 0;
110
            }
  8
110
  9
111
  0
            //RIGHT TARGET SPEED SETTING
111
                                                                               // Only
            if (rightTargetSpeed crnt > 0) {
  1
     decelerate if one target has not already reached zero.
111
                                                                               //i is the
              for (int i = 0; i < DecelArrayCount; i++) {</pre>
     number of elements in the decelSpeedValue array
111
                if (rightTargetSpeed crnt > DecelSpeedValues[i]) {
  3
111
                  rightTargetSpeed_crnt = DecelSpeedValues[i];
  4
111
  5
                  break;
111
  6
                }
111
              }
111
            } else if (rightTargetSpeed_crnt < 0) {</pre>
                                                                               // The motor
  8
     must be moving backwards if negative targetspeed
111
              for (int i = 0; i < DecelArrayCount; i++) {</pre>
                                                                               //i is the
  9
     number of elements in the decelSpeedValue array
112
                if (-rightTargetSpeed_crnt > DecelSpeedValues[i]) {
  0
112
                  rightTargetSpeed crnt = -DecelSpeedValues[i];
 1
112
                  break;
112
                }
  3
112
              }
  4
112
                                                                               // If value is
  5
            } else {
     not >0 or <0 must be =0
112
              rightTargetSpeed_crnt = 0;
112
            }
112
  8
            //Write the target value to the serial monitor
112
  9
            Serial.println();
```

```
113
           Serial.print("The target velocity is L/R:");
 0
113
 1
           Serial.print('\t');
113
           Serial.print(leftTargetSpeed crnt);
 2
113
           Serial.print(" / ");
 3
113
           Serial.println(rightTargetSpeed crnt);
  4
113
 5
           Serial.println();
113
  6
           break;
113
113
         case 3: // A single control input is active
 8
     ______
113
 9
114
           switch (maxIndex) {    //Sub switch case determined by which single control is
 0
     active
114
             case 0: // FORWARD only is active (drive both motors)
 1
114
               for (int i = 0; i < AccelArrayCount; i++) {</pre>
 2
114
                 if (leftTargetSpeed_crnt < AccelSpeedValues[i]*modeFactorFinal) { //</pre>
 3
     This checks if the system was already moving instead of accelerating from a stop
114
                   leftTargetSpeed_crnt = AccelSpeedValues[i] * modeFactorFinal;
  4
114
 5
                 }
114
                 if (rightTargetSpeed crnt < AccelSpeedValues[i]) {    // This checks if</pre>
     the system was already moving instead of accelerating from a stop
114
                   rightTargetSpeed_crnt = AccelSpeedValues[i];
 7
114
                   break;//forloop
 8
114
 9
                 }
115
 0
               }
115
 1
115
               break;
115
 3
115
             case 1: // LEFT only is active (drive right motor)
  4
115
 5
               for (int i = 0; i < AccelArrayCount; i++) {</pre>
115
                 if (rightTargetSpeed_crnt < AccelSpeedValues[i]*modeFactorFinal) { //</pre>
  6
     This checks if the system was already moving instead of accelerating from a stop
115
                   rightTargetSpeed crnt = AccelSpeedValues[i] * modeFactorFinal;
115
  8
                   break;
```

```
115
                  }
 9
116
  0
                }
116
 1
                break;
116
              case 2: // RIGHT only is active (drive left motor)
 2
116
                for (int i = 0; i < AccelArrayCount; i++) {</pre>
  3
116
                  if (leftTargetSpeed crnt < AccelSpeedValues[i]*modeFactorFinal) { //</pre>
     This checks if the system was already moving instead of accelerating from a stop
116
                    leftTargetSpeed crnt = AccelSpeedValues[i] * modeFactorFinal;
  5
116
                    break;
  6
116
                  }
 7
116
 8
                }
116
 9
                break;
117
              case 3: // REVERSE only is active (drive both motors, backwards and divided
  0
     by a factor)
117
                for (int i = 0; i < AccelArrayCount; i++) {</pre>
 1
117
                  if (abs(leftTargetSpeed crnt) < AccelSpeedValues[i] / revFactor *</pre>
 2
     modeFactorFinal) { // This checks if the system was already moving instead of
     accelerating from a stop
117
                    leftTargetSpeed crnt = -AccelSpeedValues[i] / revFactor *
     modeFactorFinal;
                              // abs() is used because target will be negative. Negative
     is used to designate reverse to the PID loop.
117
                  }
117
                  if (abs(rightTargetSpeed_crnt) < AccelSpeedValues[i] / revFactor *</pre>
  5
     modeFactorFinal) { // This checks if the system was already moving instead of
     accelerating from a stop
117
                    rightTargetSpeed crnt = -AccelSpeedValues[i] / revFactor *
  6
     modeFactorFinal;
117
 7
                    break;//for loop exit
117
                  }
 8
117
 9
                }
118
                break;
 0
118
            }
 1
118
  2
118
            //Write the target value to the serial monitor
 3
118
            Serial.println();
  4
118
  5
            Serial.print("The target velocity is L/R:");
118
  6
            Serial.print('\t');
```

```
118
 7
            Serial.print(leftTargetSpeed crnt);
118
 8
            Serial.print(" / ");
118
            Serial.println(rightTargetSpeed crnt);
 9
119
 0
            Serial.println();
119
           break;
 1
119
                    // Two control inputs are active simultaneously
 2
         case 2:
119
            //Diagonal Left Forward
119
           if (fwdBtnState_crnt == 0 && leftBtnState_crnt == 0) {
 4
119
 5
              for (int i = 0; i < AccelArrayCount; i++) {</pre>
119
                if (rightTargetSpeed crnt < AccelSpeedValues[i]*modeFactorFinal) {    //</pre>
  6
     This checks if the system was already moving instead of accelerating from a stop
119
                  rightTargetSpeed crnt = AccelSpeedValues[i] * modeFactorFinal;
 7
119
                  leftTargetSpeed crnt = rightTargetSpeed crnt * diagTurnFactor1 /
  8
     diagTurnFactor2; //Turn the left wheel proportionally slower than the right in
     order to turn left
119
                  break;//for loop exit
 9
120
  0
                }
120
 1
              }
120
 2
            }
120
  3
120
            //Diagonal Right Forward
  4
120
            if (fwdBtnState crnt == 0 && rightBtnState crnt == 0) {
 5
120
              for (int i = 0; i < AccelArrayCount; i++) {</pre>
  6
120
                if (leftTargetSpeed crnt < AccelSpeedValues[i]*modeFactorFinal) {</pre>
 7
     This checks if the system was already moving instead of accelerating from a stop
120
                  leftTargetSpeed crnt = AccelSpeedValues[i] * modeFactorFinal;
  8
120
  9
                  rightTargetSpeed crnt = leftTargetSpeed crnt * diagTurnFactor1 /
                        //Turn the right wheel proportionally slower than the left in
     diagTurnFactor2;
     order to turn right
121
                  break; //for loop exit
 0
121
                }
 1
121
  2
              }
121
            }
  3
121
            //Diagonal Left Reverse
  4
121
  5
            if (revBtnState_crnt == 0 && leftBtnState_crnt == 0) {
```

```
121
  6
              for (int i = 0; i < AccelArrayCount; i++) {</pre>
121
                if (abs(rightTargetSpeed_crnt) < AccelSpeedValues[i] / revFactor *</pre>
     modeFactorFinal) { // This checks if the system was already moving instead of
     accelerating from a stop
121
 8
                  rightTargetSpeed crnt = -AccelSpeedValues[i] / revFactor *
     modeFactorFinal;
121
                  leftTargetSpeed_crnt = rightTargetSpeed_crnt * diagTurnFactor1 /
                       //Turn the right wheel proportionally slower than the left in
     diagTurnFactor2;
     order to turn right
122
                  break; //for loop exit
  0
122
                }
 1
122
              }
 2
122
  3
            }
122
            //Diagonal Right Reverse
  4
122
            if (revBtnState_crnt == 0 && rightBtnState_crnt == 0) {
122
              for (int i = 0; i < AccelArrayCount; i++) {</pre>
  6
122
 7
                if (abs(leftTargetSpeed_crnt) < AccelSpeedValues[i] / revFactor *</pre>
     modeFactorFinal) { // This checks if the system was already moving instead of
     accelerating from a stop
122
                  leftTargetSpeed crnt = -AccelSpeedValues[i] / revFactor *
     modeFactorFinal;
122
                  rightTargetSpeed crnt = leftTargetSpeed crnt * diagTurnFactor1 /
  9
     diagTurnFactor2; //Turn the right wheel proportionally slower than the left in
     order to turn right
123
                  break; //for loop exit
  0
123
                }
 1
123
  2
              }
123
            }
  3
123
            //Twist Conditions (left and right control simultaneous)
  4
123
  5
            if (leftBtnState crnt == 0 && rightBtnState crnt == 0) {
123
              //Twist Left
  6
123
              if (maxIndex == 2) { //Right button was applied first
 7
123
                for (int i = 0; i < AccelArrayCount; i++) {</pre>
 8
123
                  if (rightTargetSpeed crnt < AccelSpeedValues[i] / revFactor *</pre>
     modeFactorFinal) {
                                   // This checks if the system was already moving instead
     of accelerating from a stop
124
                    rightTargetSpeed crnt = AccelSpeedValues[i] / revFactor *
     modeFactorFinal; //Set speed according to mode selected
124
 1
                    leftTargetSpeed_crnt = -rightTargetSpeed_crnt;
```

```
//Turn the left wheel the opposite direction but same speed as right wheel to pivot
     in place
124
 2
                    break; //for loop exit
124
 3
                  }
124
  4
                }
124
              } else {
                                  //Left button was applied first, maxIndex must be equal
  5
     to 1
124
                for (int i = 0; i < AccelArrayCount; i++) {</pre>
  6
124
                  if (leftTargetSpeed crnt < AccelSpeedValues[i] / revFactor *</pre>
     modeFactorFinal) {
                                   // This checks if the system was already moving
     instead of accelerating from a stop
124
                    leftTargetSpeed_crnt = AccelSpeedValues[i] / revFactor *
 8
     modeFactorFinal; //Set final speed according to mode selected
124
                    rightTargetSpeed crnt = -leftTargetSpeed crnt;
     //Turn the right wheel the opposite direction but same speed as left wheel to pivot
     in place
125
                    break;//for loop exit
125
 1
                  }
125
                }
 2
125
  3
             }
125
  4
            }
125
 5
125
            //Write the target value to the serial monitor
  6
125
           Serial.println();
 7
125
           Serial.print("The target velocity is L/R:");
 8
125
           Serial.print('\t');
 9
126
           Serial.print(leftTargetSpeed crnt);
 0
126
           Serial.print(" / ");
 1
126
           Serial.println(rightTargetSpeed crnt);
126
           Serial.println();
126
           break;
 4
126
  5
                    // Three control inputs are active simultaneously
126
                    // There is no forseeable condition in which the user could
  6
     intentionally do this, turn the motors off.
126
            leftTargetSpeed_crnt = 0;
126
            rightTargetSpeed_crnt = 0;
```

```
126
      break;
127
     case 0: // All 4 directional control inputs are active
127
          // There is no forseeable condition in which the user could
1
   intentionally do this, turn the motors off.
127
      leftTargetSpeed_crnt = 0;
127
      rightTargetSpeed_crnt = 0;
127
      break;
127
5
   }
127
6
   }
127
7
127
 8
   /*-----
   _____*/
127
   /*_____
   _____*/
```

32 of 32