

```

1  /*****
2  * FoamCutter
3  * This code is written for Design/Build/Fly CNC foamcutter.
4  * Written by Yuting Huang (ythuang96@gmail.com).
5  * Please report any bug to my email address.
6  *
7  * Last update: 6/30/2018
8  *
9  * Current Version: V 1.0.0
10 *****/
11 #define VERSION_A 1
12 #define VERSION_B 0
13 #define VERSION_C 0
14 #include "foamcutter_setup.h"
15
16 typedef enum state_t{
17     HOMED, GCODE, EXITING
18 } state_t;
19 typedef struct position_t{
20     int32_t LX, LY, RX, RY;
21 } position_t;
22 typedef struct speed_t{
23     float LX, LY, RX, RY;
24 } speed_t;
25 typedef struct coord_t{
26     float LX_old, LY_old, RX_old, RY_old;
27     float LX, LY, RX, RY;
28 } coord_t;
29 typedef struct coord_lim_t{
30     float LX_max, LY_max, RX_max, RY_max;
31     float LX_min, LY_min, RX_min, RY_min;
32 } coord_lim_t;
33
34 /*****
35 ***** GLOBAL VARIABLES *****
36 *****/
37 state_t state_;
38 position_t target_position_;
39 position_t current_position_;
40 position_t reached_position_;
41 position_t stop_;
42 speed_t set_speed_;
43 coord_t coord_;
44 coord_lim_t coord_lim_;
45
46 float coord_offset_x_;
47 float coord_offset_y_;
48 int gcode_menu_option_;
49 int ETA_;
50 struct timespec start_time_;
51 FILE *ptr_file_;
52
53 int state_STOP_ = 0;
54
55 // Threads
56 pthread_t LX_thread;
57 pthread_t LY_thread;
58 pthread_t RX_thread;
59 pthread_t RY_thread;
60 pthread_t printing_thread;
61 pthread_t cut_manager;
62 pthread_t switch_thread;
63 struct sched_param params_motor_thread;
64 struct sched_param params_print_thread;
65 struct sched_param params_cut_manager;
66 struct sched_param params_switch_thread;
67
68 /*****
69 ***** FUNCTION DECLARATIONS *****
70 *****/
71 // THREADS
72 void* LX_thread_func(void* ptr);
73 void* LY_thread_func(void* ptr);
74 void* RX_thread_func(void* ptr);
75 void* RY_thread_func(void* ptr);
76 void* cut_manager_func(void* ptr);
77 void* print_func(void* ptr);
78 void* switch_thread_func(void* ptr);
79
80 // SYSTEM FUNCTIONS
81 void initialize_pin();
82 void home();
83 int loadtext(char* filename);
84 int check_cord(char* str);
85 int allreached();
86 void stop_all();
87 float cut_length_func();

```

```

88 void drive(int pin_pul, int pin_dir, float speed, int32_t delta_pulse, int* ptr_current, int* ptr_stop, int polarity );
89 void cut_gcode(char* filename);
90 void moveto(float x, float y);
91
92 // MENU FUNCTIONS
93 void main_menu();
94 void gcode_menu();
95 void move_menu();
96 int menu(int numb_of_options);
97 int menu_enter();
98 int menu_yes();
99 int menu_enter_one(float* output, char* string);
100 int menu_enter_two(float* output1, float* output2, char* string);
101
102 // OTHER FUNCTIONS
103 void nsleep(uint64_t ns);
104 int file_filter(const struct dirent *entry);
105 void removespace(char* str);
106 void SigHandler(int dummy);
107 float max(float a, float b);
108 float min(float a, float b);
109 void print_time(int sec);
110 int str2f(char* str, float* output);
111
112
113 /*****
114 ***** MAIN *****
115 *****/
116
117 int main(){
118     // Setup GPIO pins
119     if (wiringPiSetupGpio () == -1) {
120         printf("Initialization failed. Most likely you are not root\n");
121         printf("Please remember to use 'sudo foamcutter'.\n");
122         return 1 ;
123     }
124     initialize_pin(); digitalWrite(PIN_RELAY, LOW);
125     // Setup signal handler for CTRL+C
126     signal(SIGINT, SigHandler);
127
128     // Start motor threads
129     params_motor_thread.sched_priority = 90;
130     pthread_setschedparam(LX_thread, SCHED_FIFO, &params_motor_thread);
131     pthread_create(&LX_thread, NULL, LX_thread_func, (void*) NULL);
132     pthread_setschedparam(LY_thread, SCHED_FIFO, &params_motor_thread);
133     pthread_create(&LY_thread, NULL, LY_thread_func, (void*) NULL);
134     pthread_setschedparam(RX_thread, SCHED_FIFO, &params_motor_thread);
135     pthread_create(&RX_thread, NULL, RX_thread_func, (void*) NULL);
136     pthread_setschedparam(RY_thread, SCHED_FIFO, &params_motor_thread);
137     pthread_create(&RY_thread, NULL, RY_thread_func, (void*) NULL);
138
139     stop_LX = stop_LY = stop_RX = stop_RY = 0;
140
141     // Print Header
142     printf("\n");
143     printf("-----+-----\n");
144     printf("| DBF Foamcutter Program by Yuting Huang | \n");
145     printf("| Current Version is V%d.%d.%d | \n" \
146         , VERSION_A, VERSION_B, VERSION_C);
147     printf("| Contact me at ythuang96@gmail.com to report bugs | \n");
148     printf("-----+-----\n");
149     printf("| Brief User Instructions | \n");
150     printf("| At any time in the program: | \n");
151     printf("| 1. Press CTRL+C to exit the prgram | \n");
152     printf("| 2. Long press EXIT button exit the prgram and shutdown the Pi | \n");
153     printf("| 3. Toggle PAUSE button to pause/resume all motor momevments | \n");
154     printf("-----+-----\n\n");
155
156     // Check Pause Switch
157     int counter1 = 0;
158     if (state_ != EXITING) {
159         for (int i = 1; i<= 21; i++) {
160             if (digitalRead(PIN_PAUSE)) counter1 ++;
161             nsleep(500000);
162         }
163         if (counter1 > 10) printf("Please toggle the PAUSE switch to resume\n\n");
164     }
165     while (state_ != EXITING && counter1 > 10) {
166         counter1 = 0;
167         for (int i = 1; i<= 21; i++) {
168             if (digitalRead(PIN_PAUSE)) counter1 ++;
169             nsleep(500000);
170         }
171     }
172 }
173
174 // Start Switch Thread

```

```

175 params_switch_thread.sched_priority = 50;
176 pthread_setschedparam(switch_thread, SCHED_FIFO, &params_switch_thread);
177 pthread_create(&switch_thread, NULL, switch_thread_func, (void*) NULL);
178
179 // Home the system
180 if (state_ != EXITING) home();
181
182 while (state_ != EXITING) main_menu();
183
184 // stop all motors
185 stop_all();
186 // end switch thread
187 pthread_join(switch_thread, NULL);
188 // end all motor threads
189 pthread_join(LX_thread, NULL); pthread_join(LY_thread, NULL);
190 pthread_join(RX_thread, NULL); pthread_join(RV_thread, NULL);
191 // print exit messages
192 printf("EXIT successful, Thank you for using the FoamCutter program.\n");
193 if (state_STOP_) {
194     printf("\nShuting down ... ..\n");
195     system("shutdown -P now");
196     return 0;
197 }
198 printf("\nIf you would like to shutdown the Pi now. Please press ENTER.\n");
199 printf("Otherwise, press 'n' then press ENTER:  "); fflush(stdout);
200
201 /***** SHUTDOWN MENU OPTIONS *****/
202 fd_set input_set; struct timeval timeout;
203 timeout.tv_sec = 10; timeout.tv_usec = 0;
204
205 // Listening for input stream for any activity
206 FD_ZERO(&input_set); FD_SET(0, &input_set);
207 while (!select(1, &input_set, NULL, NULL, &timeout)) {
208     timeout.tv_sec = 10;
209     FD_ZERO(&input_set ); FD_SET(0, &input_set);
210 }
211
212 // get input
213 char input_option[256]; fgets(input_option,256,stdin);
214 // determine length of input
215 int i; for(i=0; input_option[i]!='\0'; i++); i --;
216
217 if (i == 1 && (input_option[0] == 'n' || input_option[0] == 'N')) {
218     // if chose not to shutdown
219     printf("\nOk, Please remember to use 'sudo shutdown now'\n");
220     printf("to shutdown the Pi before unplugging the power.\n\n");
221 }
222 else { // chose to shutdown
223     printf("\nShuting down ... ..\n");
224     system("shutdown -P now");
225 }
226
227 return 0;
228 }
229
230 /***** THREADS *****/
231 void* LX_thread_func(void* ptr){
232     while (state_ != EXITING) {
233         if (set_speed_LX == 0 || reached_position_LX == 1 || stop_LX == 1) {
234             if (current_position_LX == target_position_LX){
235                 reached_position_LX = 1;
236             }
237             digitalWrite(PIN_LX_PUL, LOW);
238             nsleep(1000000);
239         }
240         else {
241             drive(PIN_LX_PUL, PIN_LX_DIR, set_speed_LX, \
242                 target_position_LX - current_position_LX, \
243                 &(current_position_LX), &(stop_LX), POLARITY_LX);
244             if (!stop_LX) reached_position_LX = 1;
245         }
246     }
247     return NULL;
248 }
249
250 void* LY_thread_func(void* ptr){
251     while (state_ != EXITING) {
252         if (set_speed_LY == 0 || reached_position_LY == 1 || stop_LY == 1) {
253             if (current_position_LY == target_position_LY){
254                 reached_position_LY = 1;
255             }
256             digitalWrite(PIN_LY_PUL, LOW);
257             nsleep(1000000);
258         }
259         else {
260

```

```

262         drive(PIN_LY_PUL, PIN_LY_DIR, set_speed_LY, \
263               target_position_LY - current_position_LY, \
264               &(current_position_LY), &(stop_LY), POLARITY_LY);
265         if (!stop_LY) reached_position_LY = 1;
266     }
267 }
268 return NULL;
269 }
270
271 void* RX_thread_func(void* ptr){
272     while (state_ != EXITING) {
273         if (set_speed_RX == 0 || reached_position_RX == 1 || stop_RX == 1) {
274             if (current_position_RX == target_position_RX){
275                 reached_position_RX = 1;
276             }
277             digitalWrite(PIN_RX_PUL, LOW);
278             nsleep(1000000);
279         }
280         else {
281             drive(PIN_RX_PUL, PIN_RX_DIR, set_speed_RX, \
282                   target_position_RX - current_position_RX, \
283                   &(current_position_RX), &(stop_RX), POLARITY_RX);
284             if (!stop_RX) reached_position_RX = 1;
285         }
286     }
287     return NULL;
288 }
289
290 void* RY_thread_func(void* ptr){
291     while (state_ != EXITING) {
292         if (set_speed_RY == 0 || reached_position_RY == 1 || stop_RY == 1) {
293             if (current_position_RY == target_position_RY){
294                 reached_position_RY = 1;
295             }
296             digitalWrite(PIN_RY_PUL, LOW);
297             nsleep(1000000);
298         }
299         else {
300             drive(PIN_RY_PUL, PIN_RY_DIR, set_speed_RY, \
301                   target_position_RY - current_position_RY, \
302                   &(current_position_RY), &(stop_RY), POLARITY_RY);
303             if (!stop_RY) reached_position_RY = 1;
304         }
305     }
306     return NULL;
307 }
308
309 void* cut_manager_func(void* ptr){
310     char buf[500];
311     while(state_ == GCODE && fgets(buf,500, ptr_file_)!=NULL){
312         while (state_ == GCODE && !allreached()) {
313             nsleep(1000000);
314         } // wait till last coord is reached
315         removespace(buf); // remove spaces
316         if (!strncmp(buf,"G4P",3)) { // check if is a pause statement
317             float temp; str2f(buf+3, &temp);
318             nsleep((uint64_t)(floor(temp*1.0E9)));
319         }
320         else if (!strncmp(buf,"G1",2)) { // if a cut statement
321             check_cord(buf); // read coordinates and update global coord_
322             // Update new target position
323             target_position_LX = (int32_t)floor((coord_LX + coord_offset_x_) * MM2PULSE);
324             target_position_LY = (int32_t)floor((coord_LY + coord_offset_y_) * MM2PULSE);
325             target_position_RX = (int32_t)floor((coord_RX + coord_offset_x_) * MM2PULSE);
326             target_position_RY = (int32_t)floor((coord_RY + coord_offset_y_) * MM2PULSE);
327             // Calculate time to move to the next coord
328             float dL = sqrt( pow(target_position_LX - current_position_LX,2.0) \
329                               + pow(target_position_LY - current_position_LY,2.0));
330             float dR = sqrt( pow(target_position_RX - current_position_RX,2.0) \
331                               + pow(target_position_RY - current_position_RY,2.0));
332             float time = (dL + dR)/2.0/FEEDRATE/MM2PULSE;
333             // Set speed for all 4 axis
334             set_speed_LX = (target_position_LX - current_position_LX)/time/MM2PULSE;
335             set_speed_LY = (target_position_LY - current_position_LY)/time/MM2PULSE;
336             set_speed_RX = (target_position_RX - current_position_RX)/time/MM2PULSE;
337             set_speed_RY = (target_position_RY - current_position_RY)/time/MM2PULSE;
338             if (state_ == GCODE) {
339                 // Start the cut by setting reached_position_ to 0
340                 reached_position_LX = reached_position_LY = 0;
341                 reached_position_RX = reached_position_RY = 0;
342                 // wait till new coord is reached
343                 nsleep((uint64_t)(floor(time*1.0E9)));
344             }
345         } // end while --- read line by line
346     }
347     // if the state_ is still GCODE, but the while loop ended;
348     // means the end of file is reached, and therefore cut is complete

```

```

349     if (state_ == GCODE) { state_ = HOMED;}
350     return NULL;
351 }
352
353 void* print_func(void* ptr){
354     struct timespec current_time;
355     int elapsed_time = 0;
356     int remain_time;
357     while(state_ == GCODE){
358         clock_gettime( CLOCK_REALTIME, &current_time);
359         elapsed_time = current_time.tv_sec - start_time_.tv_sec;
360         remain_time = ETA_ - elapsed_time;
361         printf("\n");
362         printf("%7.3f/%8.3f|%7.3f/%8.3f|%7.3f/%8.3f|%7.3f/%8.3f| ", \
363             current_position_.LX/MM2PULSE*MM2IN , current_position_.LX/MM2PULSE, \
364             current_position_.LY/MM2PULSE*MM2IN , current_position_.LY/MM2PULSE, \
365             current_position_.RX/MM2PULSE*MM2IN , current_position_.RX/MM2PULSE, \
366             current_position_.RY/MM2PULSE*MM2IN , current_position_.RY/MM2PULSE);
367         print_time(elapsed_time); printf(" | ");
368         print_time(remain_time); printf(" |");
369         fflush(stdout);
370         nsleep(1000000000); // run at 1 Hz
371     }
372     return NULL;
373 }
374
375 void* switch_thread_func(void* ptr){
376     int state_P = 0;
377     int counter_P, counter_S;
378     int counter_S2 = 0;
379     while (state_ != EXITING) {
380         counter_P = counter_S = 0;
381         for (int i = 1; i<= 41; i++) {
382             if (digitalRead(PIN_PAUSE)) counter_P ++;
383             if (digitalRead(PIN_STOP )) counter_S ++;
384             nsleep(500000);
385         }
386
387         if (counter_P > 25 && !state_P) {
388             stop_all();
389             state_P = 1;
390         }
391         else if (counter_P <= 15 && state_P && state_ == GCODE) {
392             state_P = 0;
393             stop_.LX = stop_.LY = stop_.RX = stop_.RY = 0;
394             digitalWrite(PIN_RELAY, HIGH);
395         }
396         else if (counter_P <= 15 && state_P && state_ == HOMED) {
397             state_P = 0;
398             stop_.LX = stop_.LY = stop_.RX = stop_.RY = 0;
399         }
400
401         if (counter_S > 25) counter_S2 ++;
402         else counter_S2 = 0;
403
404         if (counter_S2 == 100){
405             state_ = EXITING; stop_all(); state_STOP_ = 1;
406             printf("\n\n***** EXITING PROGRAM *****\n");
407         }
408     }
409     return NULL;
410 }
411
412 /*****
413 ***** SYSTEM FUNCTIONS *****
414 *****/
415
416 void initialize_pin(){
417     // Setup limit switch pins
418     pinMode(PIN_LX_LIM, INPUT); pullUpDnControl(PIN_LX_LIM, PUD_DOWN);
419     pinMode(PIN_LY_LIM, INPUT); pullUpDnControl(PIN_LY_LIM, PUD_DOWN);
420     pinMode(PIN_RX_LIM, INPUT); pullUpDnControl(PIN_RX_LIM, PUD_DOWN);
421     pinMode(PIN_RY_LIM, INPUT); pullUpDnControl(PIN_RY_LIM, PUD_DOWN);
422     // Setup motor drive pins
423     pinMode(PIN_LX_DIR, OUTPUT); pinMode(PIN_LX_PUL, OUTPUT);
424     pinMode(PIN_LY_DIR, OUTPUT); pinMode(PIN_LY_PUL, OUTPUT);
425     pinMode(PIN_RX_DIR, OUTPUT); pinMode(PIN_RX_PUL, OUTPUT);
426     pinMode(PIN_RY_DIR, OUTPUT); pinMode(PIN_RY_PUL, OUTPUT);
427     // Setup switches pins
428     pinMode(PIN_PAUSE, INPUT); pullUpDnControl(PIN_PAUSE, PUD_DOWN);
429     pinMode(PIN_STOP , INPUT); pullUpDnControl(PIN_STOP , PUD_DOWN);
430     // Setup relay control pin
431     pinMode(PIN_RELAY, OUTPUT);
432     return;
433 }
434
435 void home(){

```

```

436 // Print Header
437 printf("I see the system is not homed yet, please press ENTER to home the system:  ");
438 fflush(stdout);
439 if (menu_enter() == -2) return; // if EXITING state, end function
440 printf("Homing ...  "); fflush(stdout);
441
442 // Home the system
443 current_position_.LX = current_position_.LY = current_position_.RX = current_position_.RY = 0;
444 reached_position_.LX = reached_position_.LY = reached_position_.RX = reached_position_.RY = 0;
445 target_position_.LX = target_position_.LY = target_position_.RX = target_position_.RY = -640000;
446
447 // Home X axis
448 set_speed_.LY = set_speed_.RY = 0.0;
449 set_speed_.LX = set_speed_.RX = -1.0;
450
451 int counter1, counter2;
452 int state1 = 0; int state2 = 0;
453
454 while (state_ != EXITING && (!state1 || !state2)) {
455     counter1 = counter2 = 0;
456     for (int i = 1; i <= 41; i++) {
457         if (digitalRead(PIN_LX_LIM)) counter1 ++;
458         if (digitalRead(PIN_RX_LIM)) counter2 ++;
459         nsleep(20000);
460     }
461     if (counter1 > 30 && !state1) {
462         state1 = 1;
463         stop_.LX = 1;
464         set_speed_.LX = 0.0;
465         current_position_.LX = LIM2ORIGIN_LX;
466         reached_position_.LX = 1;
467     }
468     if (counter2 > 30 && !state2) {
469         state2 = 1;
470         stop_.RX = 1;
471         set_speed_.RX = 0.0;
472         current_position_.RX = LIM2ORIGIN_RX;
473         reached_position_.RX = 1;
474     }
475 }
476
477 if (state_ == EXITING) return;
478 nsleep(200000000);
479 target_position_.LX = LIM2ORIGIN_LX + 1000;
480 target_position_.RX = LIM2ORIGIN_RX + 1000;
481 set_speed_.LX = set_speed_.RX = +FEEDRATE;
482 stop_.LX = stop_.RX = 0;
483 reached_position_.LX = reached_position_.RX = 0;
484 while (state_ != EXITING && (!reached_position_.LX || !reached_position_.RX)) { nsleep(1000000); }
485
486 // Home Y axis
487 if (state_ == EXITING) return;
488 nsleep(500000000);
489 set_speed_.LX = set_speed_.RX = 0.0;
490 set_speed_.LY = set_speed_.RY = -1.0;
491
492 counter1 = counter2 = state1 = state2 = 0;
493 while (state_ != EXITING && (!state1 || !state2)) {
494     counter1 = counter2 = 0;
495     for (int i = 1; i <= 41; i++) {
496         if (digitalRead(PIN_LY_LIM)) counter1 ++;
497         if (digitalRead(PIN_RY_LIM)) counter2 ++;
498         nsleep(20000);
499     }
500     if (counter1 > 30 && !state1) {
501         state1 = 1;
502         stop_.LY = 1;
503         set_speed_.LY = 0.0;
504         current_position_.LY = LIM2ORIGIN_LY;
505         reached_position_.LY = 1;
506     }
507     if (counter2 > 30 && !state2) {
508         state2 = 1;
509         stop_.RY = 1;
510         set_speed_.RY = 0.0;
511         current_position_.RY = LIM2ORIGIN_RY;
512         reached_position_.RY = 1;
513     }
514 }
515 if (state_ == EXITING) return;
516 nsleep(500000000);
517
518 // Move to origin
519 if (state_ == EXITING) return;
520 printf("All limits reached, Moving to Origin ...  "); fflush(stdout);
521
522 target_position_.LX = target_position_.LY = target_position_.RX = target_position_.RY = 0;

```

```

523     set_speed_.LX      = set_speed_.LY      = set_speed_.RX      = set_speed_.RY      = +FEEDRATE;
524     stop_.LX = stop_.LY = stop_.RX = stop_.RY = 0;
525
526     reached_position_.LY = reached_position_.RY = reached_position_.LX = reached_position_.RX = 0;
527     while (state_ != EXITING && !allreached()) { nsleep(1000000);}
528
529     if (state_ == EXITING) return;
530     state_ = HOMED;
531     printf("Homing Complete! \n\n");
532
533     return;
534 }
535
536 int loadtext(char* filename){
537     float pause_time = 0.0;
538     float cut_length = 0.0;
539     int asym_cut = 0;
540     int error = 0; // set error to 1 will return to main menu
541     float span;
542     float coord_x_min;
543     float coord_y_min;
544     float width;
545     float height;
546
547     coord_lim_.LX_max = coord_lim_.LX_min = 0.0;
548     coord_lim_.LY_max = coord_lim_.LY_min = 0.0;
549     coord_lim_.RX_max = coord_lim_.RX_min = 0.0;
550     coord_lim_.RY_max = coord_lim_.RY_min = 0.0;
551
552     coord_.LX_old = coord_.LX = 0.0;
553     coord_.LY_old = coord_.LY = 0.0;
554     coord_.RX_old = coord_.RX = 0.0;
555     coord_.RY_old = coord_.RY = 0.0;
556
557     coord_offset_x_ = coord_offset_y_ = 0.0;
558     /***** READ FILE *****/
559     FILE *ptr_file; char buf[500];
560     ptr_file = fopen(filename, "r");
561
562     int line_numb = 1;
563     while (fgets(buf,500, ptr_file)!=NULL){ // get line by line
564         removespace(buf); // remove spaces
565         if (!strcmp(buf,"G4P",3)) { // check if is a pause statement
566             float temp;
567             if (str2f(buf+3, &temp)) {pause_time += temp;}
568             else {
569                 printf("G-code error at line %d. Returning to Main Menu.\n\n",line_numb);
570                 error = 1;
571                 return 1;
572             }
573         }
574         else if (!strcmp(buf,"G1",2)) { // if a cut statement
575             if (check_cord(buf)) { // check if statement is valid
576                 cut_length += cut_length_func();
577                 if (fabs(coord_.LX - coord_.RX) + fabs(coord_.LY - coord_.RY) > 0.0001 ) {
578                     asym_cut ++;
579                 }
580                 // update the max/min coordinates
581                 coord_lim_.LX_max = max(coord_lim_.LX_max, coord_.LX);
582                 coord_lim_.LX_min = min(coord_lim_.LX_min, coord_.LX);
583                 coord_lim_.LY_max = max(coord_lim_.LY_max, coord_.LY);
584                 coord_lim_.LY_min = min(coord_lim_.LY_min, coord_.LY);
585                 coord_lim_.RX_max = max(coord_lim_.RX_max, coord_.RX);
586                 coord_lim_.RX_min = min(coord_lim_.RX_min, coord_.RX);
587                 coord_lim_.RY_max = max(coord_lim_.RY_max, coord_.RY);
588                 coord_lim_.RY_min = min(coord_lim_.RY_min, coord_.RY);
589             }
590             // if not a valid line, print error message and stop reading
591             else {
592                 printf("G-code error at line %d. Returning to Main Menu.\n\n",line_numb);
593                 error = 1;
594                 return 1;
595             }
596         }
597         line_numb ++;
598     } // end while --- read line by line
599     fclose(ptr_file); // read complete
600
601     if (!error && state_ != EXITING) { // if no reading error occurred
602         coord_x_min = min(coord_lim_.LX_min,coord_lim_.RX_min);
603         coord_y_min = min(coord_lim_.LY_min,coord_lim_.RY_min);
604     } // end if --- check error
605     /***** FIRST: check if offset is needed *****/
606     // check x
607     if (coord_x_min < 0 && state_ != EXITING && !error){
608         printf("I see you have a min x coordinate of %7.3f in (%8.3f mm)\n", \
609             coord_x_min*MM2IN, coord_x_min);

```

```

610     printf("A negative value is not allowed\n");
611     printf("You can use the minimum offset, or enter one yourself\n");
612     printf("Would you like to use the minimum offset for x?\n");
613     if (!menu_yes()){
614         while(state_ != EXITING) {
615             int temp = menu_enter_one(&coord_offset_x_, "Please enter the x offset");
616             if (temp == -1) {printf("Invalid input, please enter again.\n\n");}
617             else if (temp == 1) {
618                 if (coord_offset_x_ < - coord_x_min) {
619                     printf("Insufficient x offset, please enter a bigger x offset\n\n");
620                 }
621                 else break;
622             }
623         }
624     }
625     else { coord_offset_x_ = - coord_x_min;}
626 }
627 // check y
628 if (coord_y_min < 0 && state_ != EXITING && !error){
629     printf("I see you have a min y coordinate of %7.3f in (%8.3f mm)\n", \
630         coord_y_min*MM2IN, coord_y_min);
631     printf("A negative value is not allowed\n");
632     printf("You can use the minimum offset, or enter one yourself\n");
633     printf("If you are cutting part of a 3-piece wing\n");
634     printf("I would recommend enter the same offset for all 3 pieces.\n");
635     printf("It will make the vaccum bagging easier\n\n");
636     printf("Would you like to use the minimum offset for y?\n");
637     if (!menu_yes()){
638         while(state_ != EXITING) {
639             int temp = menu_enter_one(&coord_offset_y_, "Please enter the y offset");
640             if (temp == -1) {printf("Invalid input, please enter again.\n\n");}
641             else if (temp == 1) {
642                 if (coord_offset_y_ < - coord_y_min) {
643                     printf("Insufficient y offset, please enter a bigger y offset\n\n");
644                 }
645                 else break;
646             }
647         }
648     }
649     else { coord_offset_y_ = - coord_y_min;}
650 }
651 coord_y_min += coord_offset_x_;
652 coord_y_min += coord_offset_y_;
653 coord_lim_.LX_max += coord_offset_x_; coord_lim_.LX_min += coord_offset_x_;
654 coord_lim_.LY_max += coord_offset_y_; coord_lim_.LY_min += coord_offset_y_;
655 coord_lim_.RX_max += coord_offset_x_; coord_lim_.RX_min += coord_offset_x_;
656 coord_lim_.RY_max += coord_offset_y_; coord_lim_.RY_min += coord_offset_y_;
657
658 if (coord_lim_.LX_max > X_MAX || coord_lim_.RX_max > X_MAX) {
659     printf("X axis out of bound, maximum X distance is 29 inches\n");
660     printf("Returning to G-code Menu\n\n");
661     error = 1; gcode_menu_option_ = -1;
662 }
663 if (coord_lim_.LY_max > Y_MAX || coord_lim_.RY_max > Y_MAX) {
664     printf("Y axis out of bound, maximum Y distance is 16 inches\n");
665     printf("Returning to G-code Menu\n\n");
666     error = 1; gcode_menu_option_ = -1;
667 }
668 /***** SECOND: determine and check foamsize *****/
669 if(!asym_cut && state_ != EXITING && !error){ // if not an asymetric cut
670     printf("I see this is a symmetric cut\n");
671     printf("The minimum require foam size is:\n");
672     width = coord_lim_.LX_max;
673     height = coord_lim_.LY_max;
674     printf("Width (x-direction): %7.3f in (%8.3f mm)\n", width*MM2IN, width);
675     printf("Thickness (y-direction): %7.3f in (%8.3f mm)\n", height*MM2IN, height);
676     printf("Please leave some extra space.\n");
677 }
678 else if (asym_cut && state_ != EXITING && !error){ // if an asymetric cut
679     printf("I see this is an asymmetric cut\n");
680     printf("The minimum require foam size depends on the span of the cut.\n");
681     printf("Please enter the span size of the cut.\n");
682     while(state_ != EXITING && menu_enter_one(&span, "Please enter the span size") == -1) {
683         printf("Invalid input, please enter again.\n\n");
684     }
685     width = min(coord_lim_.LX_max, coord_lim_.RX_max) + \
686         fabs(coord_lim_.LX_max - coord_lim_.RX_max)*(CUTTERWIDTH + span)/2.0/CUTTERWIDTH;
687     height = min(coord_lim_.LY_max, coord_lim_.RY_max) + \
688         fabs(coord_lim_.LY_max - coord_lim_.RY_max)*(CUTTERWIDTH + span)/2.0/CUTTERWIDTH;
689     if (state_ != EXITING){
690         printf("Width (x-direction): %7.3f in (%8.3f mm)\n", width*MM2IN, width);
691         printf("Thickness (y-direction): %7.3f in (%8.3f mm)\n", height*MM2IN, height);
692         printf("Please leave some extra space.\n");
693     }
694 }
695 if(state_ != EXITING && !error){
696     printf("Does this look correct and matches your foam size?\n"); fflush(stdout);

```



```

697     if (!menu_yes()){
698         printf("Looks like there's something wrong. Returning to G-code Menu.\n\n");
699         error = 1; gcode_menu_option_ = -1;
700     }
701 }
702 /***** THIRD: review cut settings *****/
703 if (state_ != EXITING && !error){
704     printf("***** CUT SETTINGS *****\n");
705     printf("G-code file: %s.\n", filename);
706     if (asym_cut) {
707         printf("Asymetric Cut of span %7.3f in (%8.3f mm)\n", span*MM2IN, span);
708     }
709     else {printf("Symmetric Cut\n");}
710     printf("Minimum Foam Size:\n");
711     printf("Width (x-direction): %7.3f in (%8.3f mm)\n", width*MM2IN, width);
712     printf("Thickness (y-direction): %7.3f in (%8.3f mm)\n", height*MM2IN, height);
713     if (coord_offset_x_) {
714         printf("x offset of %7.3f in (%8.3f mm)", coord_offset_x_*MM2IN, coord_offset_x_);
715     }
716     else {printf("No x offset");}
717     printf(" and ");
718     if (coord_offset_y_) {
719         printf("y offset of %7.3f in (%8.3f mm)", coord_offset_y_*MM2IN, coord_offset_y_);
720     }
721     else {printf("No y offset");}
722     printf("\n");
723     printf("Estimate total time of cut: ");
724     ETA_ = round(pause_time + cut_length/FEEDRATE );
725     print_time(ETA_); printf("\n");
726
727     if (asym_cut) {printf("Please make sure the foam is centered.\n");}
728     printf("\nWould you like to start the cut?\n");
729
730     if (!menu_yes()){
731         printf("OK, setting incorrect. Returning to G-code Menu.\n\n");
732         error = 1; gcode_menu_option_ = -1;
733     }
734 }
735 return error;
736 }
737
738 int check_cord(char* str){
739     coord_.LX_old = coord_.LX; coord_.LY_old = coord_.LY;
740     coord_.RX_old = coord_.RX; coord_.RY_old = coord_.RY;
741     char* ptr_X = strchr(str, 'X');
742     char* ptr_Y = strchr(str, 'Y');
743     char* ptr_Z = strchr(str, 'Z');
744     char* ptr_A = strchr(str, 'A');
745     if (ptr_X && ptr_Y && ptr_Z && ptr_A && \
746         (ptr_X < ptr_Y) && (ptr_Y < ptr_Z) && (ptr_Z < ptr_A) ){
747         char temp1[20], temp2[20], temp3[20], temp4[20];
748         float tempf1, tempf2, tempf3, tempf4;
749         strncpy(temp1, ptr_X+1, ptr_Y-ptr_X-1); temp1[(ptr_Y-ptr_X-1)] = '\0';
750         strncpy(temp2, ptr_Y+1, ptr_Z-ptr_Y-1); temp2[(ptr_Z-ptr_Y-1)] = '\0';
751         strncpy(temp3, ptr_Z+1, ptr_A-ptr_Z-1); temp3[(ptr_A-ptr_Z-1)] = '\0';
752         strncpy(temp4, ptr_A+1, 10);
753
754         if(str2f(temp1, &tempf1) && str2f(temp2, &tempf2) && \
755             str2f(temp3, &tempf3) && str2f(temp4, &tempf4)){
756             coord_.LX = tempf1; coord_.LY = tempf2;
757             coord_.RX = tempf3; coord_.RY = tempf4;
758             return 1;
759         }
760         else return 0;
761     }
762     else return 0;
763 }
764
765 int allreached() {
766     return reached_position_.LX * reached_position_.LY * reached_position_.RX * reached_position_.RY;
767 }
768
769 void stop_all() {
770     stop_.LX = stop_.LY = stop_.RX = stop_.RY = 1;
771     digitalWrite(PIN_RELAY, LOW);
772     return;
773 }
774
775 float cut_length_func(){
776     float L_length = sqrt(pow((coord_.LX - coord_.LX_old),2.0) + pow((coord_.LY - coord_.LY_old),2.0));
777     float R_length = sqrt(pow((coord_.RX - coord_.RX_old),2.0) + pow((coord_.RY - coord_.RY_old),2.0));
778     return (L_length + R_length)/2.0;
779 }
780
781 void drive(int pin_pul, int pin_dir, float speed, int32_t delta_pulse, int* ptr_current, int* ptr_stop, int polarity){
782     int inc;
783     if (speed*polarity > 0) { digitalWrite(pin_dir, HIGH); inc = 1*polarity;}

```

```

784     else if (speed*polarity < 0) { digitalWrite(pin_dir, LOW); inc = -1*polarity;}
785     // the time between pulses, calculated from speed
786     // 4000 is the sleep time in the loop;
787     // 100000 is the approximate code execution time;
788     uint64_t sleep_time = floor(100000000.0/MM2PULSE/fabs(speed)) - 4000 - 100000;
789     nsleep(5000); // ensure dir pin leads by at least 5 microsec
790     for (int i = 0; i < abs(delta_pulse); i++) { // send out desired number of pulses
791         digitalWrite(pin_pul, HIGH);
792         nsleep(4000); // ensure pulse width of at least 2 microsec
793         digitalWrite(pin_pul, LOW);
794         *ptr_current += inc;
795         if (*ptr_stop) break; // stop the motor if stop is a 1;
796         nsleep(sleep_time);
797     }
798     return;
799 }
800
801 void cut_gcode(char* filename){
802     state_ = GCODE;
803     if (state_ == EXITING) return;
804     moveto(-5.0,0.0);
805     while (state_ != EXITING && !allreached()){ nsleep(1000000);}
806     if (state_ == EXITING) return;
807     printf("Please connect and turn on the power supply for the hot wire\n");
808     printf("Please make sure the voltage is approximately 10V, and press ENTER to continue:  ");
809     fflush(stdout);
810     if (menu_enter() == -2) return;
811     digitalWrite(PIN_RELAY,HIGH);
812     printf("Please now adjust the power supply to the desired current.\n");
813     printf("Recommend 2.1 to 2.3 Amps, depending on the cut span.\n");
814     printf("Use higher current for wider cuts.\n");
815     printf("Increase current if wire bows significantly.\n");
816     printf("Press ENTER to start cutting:  ");
817     fflush(stdout);
818     if (menu_enter() == -2) return;
819     printf("Heating wire ... "); fflush(stdout);
820     if (state_ != EXITING) nsleep(500000000);
821     if (state_ == EXITING) return;
822     printf(" Cut Starting\n");
823     moveto(0.0,0.0);
824     while (state_ != EXITING && !allreached()) nsleep(1000000);
825     if (state_ == EXITING) return;
826
827     coord_.LX_old = coord_.LX = 0.0;
828     coord_.LY_old = coord_.LY = 0.0;
829     coord_.RX_old = coord_.RX = 0.0;
830     coord_.RY_old = coord_.RY = 0.0;
831     clock_gettime( CLOCK_REALTIME, &start_time_);
832
833     ptr_file_ =fopen(filename, "r");
834     printf("      LX      |      LY      |      RX      |      RY      |      TIME      |\n");
835     printf(" in / mm | in / mm | in / mm | in / mm | Elapsed | Remaining |\n");
836
837     params_print_thread.sched_priority = 40;
838     params_cut_manager.sched_priority = 99;
839     pthread_setschedparam(printing_thread, SCHED_FIFO, &params_print_thread);
840     pthread_create(&printing_thread, NULL, print_func, (void*) NULL);
841     pthread_setschedparam(cut_manager, SCHED_FIFO, &params_cut_manager);
842     pthread_create(&cut_manager, NULL, cut_manager_func, (void*) NULL);
843
844     while (state_ == GCODE) nsleep(100000);
845
846     fclose(ptr_file_); // read complete
847
848     if (state_ != EXITING) moveto(-5.0,0.0);
849     while (state_ != EXITING && !allreached()) nsleep(1000000);
850     digitalWrite(PIN_RELAY,LOW);
851     if (state_ != EXITING) moveto( 0.0,0.0);
852     while (state_ != EXITING && !allreached()) nsleep(1000000);
853     if (state_ != EXITING) printf("\nCut Complete, Returning to Main Menu.\n\n");
854
855     pthread_join(cut_manager, NULL);
856     pthread_join(printing_thread, NULL);
857     return;
858 }
859
860 void moveto(float x, float y){
861     target_position_.LX = target_position_.RX = (int32_t)floor(x * MM2PULSE);
862     target_position_.LY = target_position_.RY = (int32_t)floor(y * MM2PULSE);
863     // Set speed for all 4 axis
864     if (target_position_.LX > current_position_.LX) set_speed_.LX = +FEEDRATE;
865     else if (target_position_.LX < current_position_.LX) set_speed_.LX = -FEEDRATE;
866     else if (target_position_.LX == current_position_.LX) set_speed_.LX = 0.0;
867
868     if (target_position_.LY > current_position_.LY) set_speed_.LY = +FEEDRATE;
869     else if (target_position_.LY < current_position_.LY) set_speed_.LY = -FEEDRATE;
870     else if (target_position_.LY == current_position_.LY) set_speed_.LY = 0.0;

```

```

871
872     if (target_position_.RX > current_position_.RX) set_speed_.RX = +FEEDRATE;
873     else if (target_position_.RX < current_position_.RX) set_speed_.RX = -FEEDRATE;
874     else if (target_position_.RX == current_position_.RX) set_speed_.RX = 0.0;
875
876     if (target_position_.RY > current_position_.RY) set_speed_.RY = +FEEDRATE;
877     else if (target_position_.RY < current_position_.RY) set_speed_.RY = -FEEDRATE;
878     else if (target_position_.RY == current_position_.RY) set_speed_.RY = 0.0;
879
880     if (state_ != EXITING) {
881         // Start the cut by setting reached_position_ to 0
882         reached_position_.LX = reached_position_.LY = 0;
883         reached_position_.RX = reached_position_.RY = 0;
884     }
885     return;
886 }
887
888 /*****
889 ***** MENU FUNCTIONS *****
890 *****/
891
892 void main_menu() {
893     printf("***** MAIN MENU *****\n");
894     printf("Please choose from the following options:\n");
895     printf("a. Load and cut from G-Code;\n");
896     printf("b. Move wire to specified location;\n");
897     printf("c. Exit Program.\n");
898     printf("Please enter the corresponding letter and press ENTER key:  ");
899     fflush(stdout);
900
901     switch (menu(3)) {
902         case 0: {gcode_menu(); break; }
903         case 1: {move_menu(); break; }
904         case 2: {state_ = EXITING; break; }
905         case -1: {printf("Invalid option. Let's try again.\n\n"); break; }
906         case -2: { break; }
907     }
908     return;
909 }
910
911 void gcode_menu() {
912     if (state_ != EXITING && (current_position_.LX || \
913         current_position_.LY || current_position_.RX || current_position_.RY)){
914         printf("I see the wire is not at origin. Plesse press ENTER to move wire to origin:  ");
915         fflush(stdout);
916         if (menu_enter() != -2) {
917             moveto(0.0,0.0);
918             while (state_ != EXITING && !allreached()){ nsleep(1000000);}
919             printf("Origin Reached\n\n");
920         }
921     }
922
923     gcode_menu_option_ = -1;
924     int n = 10;
925     while (state_ != EXITING && gcode_menu_option_ == -1){
926         printf("***** GCODE MENU *****\n");
927         // keep looping when menu selection is invalid
928         struct dirent **namelist;
929         n = scandir("/home/pi/", &namelist, file_filter, alphasort);
930         // scan for files with .txt extension
931         if (n == 0) {
932             printf("I do not see any txt files in '/home/pi/' directory\n");
933             printf("Please put the desired gcode file in '/home/pi/' directory.\n");
934             printf("Returning to Main Menu.\n\n");
935             break;
936         }
937         else if (n > 9){
938             printf("Too many txt files in '/home/pi/' directory\n");
939             printf("Please clean it up.\n");
940             printf("Returning to Main Menu.\n\n");
941             break;
942         }
943         else{
944             printf("I see there are %d txt files listed below:\n\n",n);
945             int i = 0;
946             while (i++ < n){
947                 printf("%d:  %s\n", i, namelist[i-1]->d_name);
948             }
949             printf("0:  None of the above, or Return to Main Menu\n\n");
950             printf("Please select one by entering the corresponding number then press ENTER:  ");
951             fflush(stdout);
952             gcode_menu_option_ = menu(n+1);
953
954             if (gcode_menu_option_ >= 1) {
955                 printf("You selected: '%s' is that correct?\n",namelist[gcode_menu_option_-1]->d_name);
956                 switch (menu_yes()){
957                     case 1:

```

```

958         if (state_ != EXITING && !loadtext(namelist[gcode_menu_option_-1]->d_name) ) {
959             if (state_ != EXITING) cut_gcode(namelist[gcode_menu_option_-1]->d_name);
960             return;
961         }
962         break;
963     case 0:
964         printf("OK, Let try again\n\n");
965         gcode_menu_option_ = -1;
966         break;
967     }
968 }
969 else if (gcode_menu_option_ == 0) {
970     printf("Please put the desired gcode file in the working directory.\n");
971     printf("Returning to Main Menu.\n\n");
972     break;
973 }
974 else if (gcode_menu_option_ == -1) { // Invalid input
975     printf("Invalid Input, let's try again.\n\n");
976 }
977 else if (gcode_menu_option_ == -2) { // EXITING state
978     break;
979 } // end of if --- menu input check
980 } // end of if --- file number check
981 } // end of while
982 return;
983 } // end of gcode_menu
984
985 void move_menu(){
986     float x,y;
987     while (state_ != EXITING){
988         float current_x = (current_position_.LX + current_position_.RX)/2.0/MM2PULSE;
989         float current_y = (current_position_.LY + current_position_.RY)/2.0/MM2PULSE;
990         printf("***** MOVE MENU *****\n");
991         printf("The current wire position is (x,y) = (%.3f,%.3f) in = (%.3f,%.3f) mm\n", \
992             current_x*MM2IN,current_y*MM2IN,current_x,current_y);
993         printf("Please choose from the following options:\n");
994         printf("a. Move wire to a specific location relative to origin (Absolute Location);\n");
995         printf("b. Move wire to a specific location relative to current position (Increment);\n");
996         printf("c. Move wire to origin;\n");
997         printf("d. Return to Main Menu;\n");
998         printf("Please enter the corresponding letter and press ENTER key:   ");
999         fflush(stdout);
1000         switch (menu(4)) {
1001             case 0: { // Move to a specific location
1002                 while(state_ != EXITING && \
1003                     (menu_enter_two(&x,&y,"Please enter the destination x and y coordinates RELATIVE TO ORIGIN") == -1 \
1004                     || x < 0 || y < 0)) {
1005                     printf("Invalid input, please enter again. Please note that negative destination is not allowed.\n\n");
1006                 }
1007                 if (state_ != EXITING) {
1008                     printf("|      |      |      |      |      |      |      |      |      |      |\n");
1009                     printf("| Inch | %.3f, | %.3f | %.3f, | %.3f | %.3f, | %.3f | %.3f |\n", \
1010                         current_x*MM2IN, current_y*MM2IN, (x-current_x)*MM2IN, (y-current_y)*MM2IN, x*MM2IN, y*MM2IN);
1011                     printf("| MM   | %.3f, | %.3f | %.3f, | %.3f | %.3f, | %.3f | %.3f |\n", \
1012                         current_x, current_y, (x-current_x), (y-current_y), x, y);
1013                     printf("Continue?\n");
1014                     switch (menu_yes()){
1015                         case 1:
1016                             if (state_ != EXITING ) {
1017                                 printf("Moving ... .."); fflush(stdout);
1018                                 moveto(x,y);
1019                                 while (state_ != EXITING && !allreached()){ nsleep(1000000);}
1020                                 if (state_ != EXITING) printf(" Destination Reached\n\n");
1021                             }
1022                             break;
1023                         case 0:
1024                             printf("OK, Let try again\n\n");
1025                             break;
1026                     }
1027                 }
1028                 break; } // end case --- move to a specific location
1029
1030             case 1: { // Move to a specific location
1031                 while(state_ != EXITING && \
1032                     (menu_enter_two(&x,&y,"Please enter the destination x and y coordinates RELATIVE TO CURRENT POSITION") == -1 \
1033                     || x+current_x < 0 || y+current_y < 0)) {
1034                     printf("Invalid input, please enter again. Please note that negative destination is not allowed.\n\n");
1035                 }
1036                 if (state_ != EXITING) {
1037                     printf("|      |      |      |      |      |      |      |      |      |      |\n");
1038                     printf("| Inch | %.3f, | %.3f | %.3f, | %.3f | %.3f, | %.3f | %.3f |\n", \
1039                         current_x*MM2IN, current_y*MM2IN, x*MM2IN, y*MM2IN, (x+current_x)*MM2IN, (y+current_y)*MM2IN);
1040                     printf("| MM   | %.3f, | %.3f | %.3f, | %.3f | %.3f, | %.3f | %.3f |\n", \
1041                         current_x, current_y, x, y, (x+current_x), (y+current_y));
1042                     printf("Continue?\n");
1043                     switch (menu_yes()){
1044                         case 1:

```

```

1045         if (state_ != EXITING ) {
1046             printf("Moving ... "); fflush(stdout);
1047             moveto(x+current_x,y+current_y);
1048             while (state_ != EXITING && !allreached()){ nsleep(1000000);}
1049             if (state_ != EXITING) printf(" Destination Reached\n\n");
1050         }
1051         break;
1052     case 0:
1053         printf("OK, Let try again\n\n");
1054         break;
1055     }
1056     break; } // end case --- move to a specific location
1057
1058
1059     case 2: { //case --- move to origin
1060         if (!current_position_.LX && !current_position_.LY && \
1061             !current_position_.RX && !current_position_.RY) {
1062             printf("Already at Origin\n\n");
1063         }
1064         else{
1065             printf("Move to Orignin. Continue?\n");
1066             switch (menu_yes()){
1067                 case 1:
1068                     printf("Moving ... "); fflush(stdout);
1069                     moveto(0,0,0);
1070                     while (state_ != EXITING && !allreached()){ nsleep(1000000);}
1071                     if (state_ != EXITING) printf(" Origin Reached\n\n");
1072                     break;
1073                 case 0:
1074                     printf("OK, Let try again\n\n");
1075                     break;
1076             }
1077             break; } // end case --- move to origin
1078
1079     case -1: {printf("Invalid option. Let's try again.\n\n"); break; }
1080     default: {return; break; }
1081     } // end switch
1082 } // end while
1083 return;
1084 }
1085
1086 int menu(int numb_of_options){
1087     // pass in number of menu options, maximum of 10 options
1088     // return 0 to (numb_of_options - 1) if input is within the range
1089     // return -1 for invalid input
1090     // return -2 when state_ is exiting
1091
1092     if (state_ == EXITING) return -2;
1093     fd_set input_set;
1094     struct timeval timeout;
1095     timeout.tv_sec = 10;
1096     timeout.tv_usec = 0;
1097
1098     // Listening for input stream for any activity
1099     // If there
1100     FD_ZERO(&input_set );
1101     FD_SET(0, &input_set);
1102     while (state_ != EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
1103         timeout.tv_sec = 1;
1104         FD_ZERO(&input_set );
1105         FD_SET(0, &input_set);
1106     }
1107
1108     if (state_ == EXITING) return -2;
1109
1110     // get input
1111     char input_option[256]; fgets(input_option,256,stdin);
1112     // determine length of input
1113     int i; for(i=0; input_option[i]!='\0'; i++); i --;
1114     int result;
1115
1116     if (i == 1) {
1117         if (input_option[0] >= 48 && input_option[0] <= 57) {
1118             result = input_option[0]-48;
1119         }
1120         else if(input_option[0] >= 65 && input_option[0] <= 74) {
1121             result = input_option[0]-65;
1122         }
1123         else if(input_option[0] >= 97 && input_option[0] <= 106) {
1124             result = input_option[0]-97;
1125         }
1126         else result = -1;
1127     }
1128     else result = -1;
1129     if (result >= numb_of_options) result = -1;
1130
1131     printf("\n");

```

```

1132     return result;
1133 }
1134
1135 int menu_enter(){
1136     // return 1 if anything is entered, including ENTER key
1137     // return 0 if m is entered
1138     // return -2 when state_ is exiting
1139     if (state_ == EXITING) return -2;
1140     fd_set input_set;
1141     struct timeval timeout;
1142     timeout.tv_sec = 10;
1143     timeout.tv_usec = 0;
1144
1145     // Listening for input stream for any activity
1146     FD_ZERO(&input_set );
1147     FD_SET(0, &input_set);
1148     while (state_ != EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
1149         timeout.tv_sec = 1;
1150         FD_ZERO(&input_set );
1151         FD_SET(0, &input_set);
1152     }
1153
1154     if (state_ == EXITING) return -2;
1155
1156     // get input
1157     char input_option[256]; fgets(input_option,256,stdin);
1158     // determine length of input
1159     int i; for(i=0; input_option[i]!='\0'; i++); i --;
1160
1161     if (i == 1 && (input_option[0] == 'm' || input_option[0] == 'M')) {
1162         printf("\n"); return 0; // return 0 if input is 'm' or 'M'
1163     }
1164     else {printf("\n"); return 1; }
1165 }
1166
1167 int menu_yes(){
1168     // return 1 if anything is entered, including ENTER key
1169     // return 0 if n is entered
1170     // return -2 when state_ is exiting
1171     if (state_ == EXITING) return -2;
1172     printf("Press ENTER for YES, or 'n' then ENTER for NO: "); fflush(stdout);
1173     fd_set input_set;
1174     struct timeval timeout;
1175     timeout.tv_sec = 10;
1176     timeout.tv_usec = 0;
1177
1178     // Listening for input stream for any activity
1179     FD_ZERO(&input_set );
1180     FD_SET(0, &input_set);
1181     while (state_ != EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
1182         timeout.tv_sec = 1;
1183         FD_ZERO(&input_set );
1184         FD_SET(0, &input_set);
1185     }
1186
1187     if (state_ == EXITING) return -2;
1188
1189     // get input
1190     char input_option[256]; fgets(input_option,256,stdin);
1191     // determine length of input
1192     int i; for(i=0; input_option[i]!='\0'; i++); i --;
1193
1194     if (i == 1 && (input_option[0] == 'n' || input_option[0] == 'N')) {
1195         printf("\n"); return 0; // return 0 if input is 'm' or 'M'
1196     }
1197     else {printf("\n"); return 1;}
1198 }
1199
1200 int menu_enter_one(float* output, char* string){
1201     float scale = 1.0;
1202     if (state_ == EXITING) return -2;
1203     printf("Would you like to enter the coordinate in inches?\n");
1204     switch (menu_yes()) {
1205         case 1: scale = 25.4; printf("%s in inches then press ENTER: ", string); break;
1206         case 0: scale = 1.0; printf("%s in millimeters then press ENTER: ", string); break;
1207     }
1208     fflush(stdout);
1209     if (state_ == EXITING) return -2;
1210
1211     fd_set input_set;
1212     struct timeval timeout;
1213     timeout.tv_sec = 10;
1214     timeout.tv_usec = 0;
1215
1216     // Listening for input stream for any activity
1217     // If there
1218

```

```

1219     FD_ZERO(&input_set );
1220     FD_SET(0, &input_set);
1221     while (state_ != EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
1222         timeout.tv_sec = 1;
1223         FD_ZERO(&input_set );
1224         FD_SET(0, &input_set);
1225     }
1226
1227     if (state_ == EXITING) return -2;
1228
1229     // get input
1230     char input_option[256]; fgets(input_option,256,stdin);
1231     // remove white spaces
1232     removespace(input_option);
1233     float temp;
1234     if (str2f(input_option, &temp)) {
1235         *output = temp*scale;
1236         printf("\n"); return 1;
1237     }
1238     else {printf("\n"); return -1;}
1239 }
1240
1241 int menu_enter_two(float* output1, float* output2, char* string){
1242     float scale = 1.0;
1243     if (state_ == EXITING) return -2;
1244     printf("Would you like to enter the coordinate in inches?\n");
1245     switch (menu_yes()) {
1246         case 1:
1247             scale = 25.4;
1248             printf("%s in INCHES \nseparated with comma then press ENTER: ", string);
1249             break;
1250         case 0:
1251             scale = 1.0;
1252             printf("%s in MM \nseparated with comma then press ENTER: ", string);
1253             break;
1254     }
1255     fflush(stdout);
1256     if (state_ == EXITING) return -2;
1257
1258     fd_set      input_set;
1259     struct timeval timeout;
1260     timeout.tv_sec = 10;
1261     timeout.tv_usec = 0;
1262
1263     // Listening for input stream for any activity
1264     // If there
1265     FD_ZERO(&input_set );
1266     FD_SET(0, &input_set);
1267     while (state_ != EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
1268         timeout.tv_sec = 1;
1269         FD_ZERO(&input_set );
1270         FD_SET(0, &input_set);
1271     }
1272     if (state_ == EXITING) return -2;
1273
1274     // get input
1275     char input_option[256]; fgets(input_option,256,stdin);
1276     // remove white spaces
1277     removespace(input_option);
1278     // get the two
1279     char* ptr = strchr(input_option, ',');
1280     if (ptr == NULL) {return -1;}
1281     char temp1[20]; strncpy(temp1,input_option,ptr-input_option);
1282     temp1[(ptr-input_option)] = '\0';
1283     char temp2[20]; strncpy(temp2,ptr+1,20);
1284     float temp1f, temp2f;
1285     if (str2f(temp1, &temp1f) && str2f(temp2, &temp2f)) {
1286         *output1 = temp1f*scale;
1287         *output2 = temp2f*scale;
1288         printf("\n"); return 1;
1289     }
1290     else {return -1;}
1291 }
1292
1293 /*****
1294 ***** OTHER FUNCTIONS *****
1295 *****/
1296
1297 // Sleep for nanoseconds
1298 void nsleep(uint64_t ns){
1299     struct timespec req,rem;
1300     req.tv_sec = ns/1000000000;
1301     req.tv_nsec = ns%1000000000;
1302     // loop untill nanosleep sets an error or finishes successfully
1303     errno=0; // reset errno to avoid false detection
1304     while(nanosleep(&req, &rem) && errno==EINTR){
1305         req.tv_sec = rem.tv_sec;

```

```

1306     req.tv_nsec = rem.tv_nsec;
1307 }
1308 return;
1309 }
1310
1311 // Filter out file without .txt extension
1312 int file_filter(const struct dirent *entry){
1313     return !strcmp(entry->d_name + strlen(entry->d_name) -4, ".txt");
1314 }
1315
1316 // Remove space in a string
1317 void removespace(char* str) {
1318     int i,j=0;
1319     for(i=0;str[i]!='\0';i++) {
1320         if(str[i]!=' ' && str[i] != 10 && str[i] != 13)
1321             str[j++]=str[i];
1322     }
1323     str[j]='\0';
1324     return;
1325 }
1326
1327 // Signal Handler for CTRL+C
1328 void SigHandler(int dummy) {
1329     state_ = EXITING; stop_all();
1330     printf("\n\n***** EXITING PROGRAM *****\n");
1331     return;
1332 }
1333
1334 float max(float a, float b) {
1335     if(a >= b) return a;
1336     else return b;
1337 }
1338
1339 float min(float a, float b) {
1340     if(a >= b) return b;
1341     else return a;
1342 }
1343
1344 void print_time(int sec){
1345     if (sec <= 0) printf("00:00");
1346     else printf("%02d:%02d", (int)floor(sec /60.0),sec % 60);
1347     return;
1348 }
1349
1350 int str2f(char* str, float* output){
1351     removespace(str);
1352     float out = 0;
1353     int dec_location;
1354     int dec_numb = 0;
1355     int i; for(i=0; str[i] !='\0'; i++);
1356     int length = i;
1357     int j =1;
1358     for(int i = length-1; i >= 0; i--) {
1359         if (str[i]>=48 && str[i]<=57){out = out+j*(str[i]-48); j = j*10;}
1360         else if (str[i] == 46) {dec_numb ++; dec_location = i;}
1361         else if (i == 0 && str[i] == 43) {out = +out;}
1362         else if (i == 0 && str[i] == 45) {out = -out;}
1363         else return 0;
1364     }
1365     if (dec_numb == 0) {out = out;}
1366     else if (dec_numb == 1) {out = out/pow(10.0,length-1-dec_location);}
1367     else {return 0;}
1368     *output = out;
1369     return 1;
1370 }

```