```
/*********************************************************************
* FoamCutter
* This code is written for Design/Build/Fly CNC foamcutter.
* Written by Yuting Huang (ythuang96@gmail.com).
* Please report any bug to my email address.
*
* Last update: 6/30/2018
*
* Current Version: V 1.0.0
*********************************************************************/
#define VERSION_A 1
#define VERSION_B 0
#define VERSION_C 0
#include "foamcutter_setup.h"

typedef enum state_t{
    HOMED, GCODE, EXITING
} state_t;
typedef struct position_t{
    int32_t LX, LY, RX, RY;
} position_t;
typedef struct speed_t{
    float LX, LY, RX, RY;
} speed_t;
typedef struct coord_t{
    float LX_old, LY_old, RX_old, RY_old;
    float LX, LY, RX, RY;
} coord_t;
typedef struct coord_lim_t{
    float LX_max, LY_max, RX_max, RY_max;
    float LX_min, LY_min, RX_min, RY_min;
} coord_lim_t;

/*********************************************************************
************************ GLOBAL VARIABLES ****************************
*********************************************************************/
state_t state_;
position_t target_position_;
position_t current_position_;
position_t reached_position_;
position_t stop_;
speed_t set_speed_;
coord_t coord_;
coord_lim_t coord_lim_;

float coord_offset_x_;
float coord_offset_y_;
int gcode_menu_option_;
int ETA_;
struct timespec start_time_;
FILE *ptr_file_;

int state_STOP_ = 0;

// Threads
pthread_t LX_thread;
pthread_t LY_thread;
pthread_t RX_thread;
pthread_t RY_thread;
pthread_t printing_thread;
pthread_t cut_manager;
pthread_t switch_thread;
struct sched_param params_motor_thread;
struct sched_param params_print_thread;
struct sched_param params_cut_manager;
struct sched_param params_switch_thread;

/*********************************************************************
********************** FUNCTION DECLARATIONS *************************
*********************************************************************/
// THREADS
void* LX_thread_func(void* ptr);
void* LY_thread_func(void* ptr);
void* RX_thread_func(void* ptr);
void* RY_thread_func(void* ptr);
void* cut_manager_func(void* ptr);
void* print_func(void* ptr);
void* switch_thread_func(void* ptr);

// SYSTEM FUNCTIONS
void initialize_pin();
void home();
int loadtext(char* filename);
int check_cord(char* str);
int allreached();
void stop_all();
float cut_length_func();
```

```
void drive(int pin_pul, int pin_dir, float speed, int32_t delta_pulse, int* ptr_current, int* ptr_stop, int polarity );
void cut_gcode(char* filename);
void moveto(float x, float y);

// MENU FUNCTIONS
void main_menu();
void gcode_menu();
void move_menu();
int menu(int numb_of_options);
int menu_enter();
int menu_yes();
int menu_enter_one(float* output, char* string);
int menu_enter_two(float* output1, float* output2, char* string);

// OTHER FUNCTIONS
void nsleep(uint64_t ns);
int file_filter(const struct dirent *entry);
void removespace(char* str);
void SigHandler(int dummy);
float max(float a, float b);
float min(float a, float b);
void print_time(int sec);
int str2f(char* str, float* output);


/*****************************************************************
***************************** MAIN *******************************
*****************************************************************/

int main(){
    // Setup GPIO pins
    if (wiringPiSetupGpio () == -1) {
        printf("Initialization failed. Most likely you are not root\n");
        printf("Please remember to use 'sudo foamcutter'.\n");
        return 1 ;
    }
    initialize_pin(); digitalWrite(PIN_RELAY, LOW);
    // Setup signal handler for CTRL+C
    signal(SIGINT, SigHandler);

    // Start motor threads
    params_motor_thread.sched_priority = 90;
    pthread_setschedparam(LX_thread, SCHED_FIFO, &params_motor_thread);
    pthread_create(&LX_thread, NULL, LX_thread_func, (void*) NULL);
    pthread_setschedparam(LY_thread, SCHED_FIFO, &params_motor_thread);
    pthread_create(&LY_thread, NULL, LY_thread_func, (void*) NULL);
    pthread_setschedparam(RX_thread, SCHED_FIFO, &params_motor_thread);
    pthread_create(&RX_thread, NULL, RX_thread_func, (void*) NULL);
    pthread_setschedparam(RY_thread, SCHED_FIFO, &params_motor_thread);
    pthread_create(&RY_thread, NULL, RY_thread_func, (void*) NULL);

    stop_.LX = stop_.LY = stop_.RX = stop_.RY = 0;

    // Print Header
    printf("\n");
    printf("+-----------------------------------------------------------------+\n");
    printf("| DBF Foamcutter Program by Yuting Huang                          |\n");
    printf("| Current Version is V%d.%d.%d                                      |\n" \
        , VERSION_A, VERSION_B, VERSION_C);
    printf("| Contact me at ythuang96@gmail.com to report bugs                |\n");
    printf("|                                                                 |\n");
    printf("|-----------------------------------------------------------------|\n");
    printf("|                    Brief User Instructions                      |\n");
    printf("| At any time in the program:                                     |\n");
    printf("| 1. Press CTRL+C to exit the prgram                              |\n");
    printf("| 2. Long press EXIT button exit the prgram and shutdown the Pi   |\n");
    printf("| 3. Toggle PAUSE button to pause/resume all motor momevments     |\n");
    printf("+-----------------------------------------------------------------+\n\n");

    // Check Pause Switch
    int counter1 = 0;
    if (state_ != EXITING) {
        for (int i = 1; i<= 21; i++) {
            if (digitalRead(PIN_PAUSE)) counter1 ++;
            nsleep(500000);
        }
        if (counter1 > 10) printf("Please toggle the PAUSE switch to resume\n\n");
    }
    while (state_ != EXITING && counter1 > 10) {
        counter1 = 0;
        for (int i = 1; i<= 21; i++) {
            if (digitalRead(PIN_PAUSE)) counter1 ++;
            nsleep(500000);
        }
    }

    // Start Switch Thread
```

```
    params_switch_thread.sched_priority = 50;
    pthread_setschedparam(switch_thread, SCHED_FIFO, &params_switch_thread);
    pthread_create(&switch_thread, NULL, switch_thread_func, (void*) NULL);

    // Home the system
    if (state_ != EXITING) home();

    while (state_ != EXITING) main_menu();

    // stop all motors
    stop_all();
    // end swtich thread
    pthread_join(switch_thread, NULL);
    // end all motor threads
    pthread_join(LX_thread, NULL); pthread_join(LY_thread, NULL);
    pthread_join(RX_thread, NULL); pthread_join(RY_thread, NULL);
    // print exit messages
    printf("EXIT successful, Thank you for using the FoamCutter program.\n");
    if (state_STOP_) {
        printf("\nShuting down ... ...\n");
        system("shutdown -P now");
        return 0;
    }
    printf("\nIf you would like to shutdown the Pi now. Please press ENTER.\n");
    printf("Otherwise, press 'n' then press ENTER:    "); fflush(stdout);

/***************** SHUTDOWN MENU OPTIONS *************************/
    fd_set input_set; struct timeval timeout;
    timeout.tv_sec = 10; timeout.tv_usec = 0;

    // Listening for input stream for any activity
    FD_ZERO(&input_set); FD_SET(0, &input_set);
    while (!select(1, &input_set, NULL, NULL, &timeout)) {
        timeout.tv_sec = 10;
        FD_ZERO(&input_set ); FD_SET(0, &input_set);
    }

    // get input
    char input_option[256]; fgets(input_option,256,stdin);
    // determine length of input
    int i; for(i=0; input_option[i]!='\0'; i++); i --;

    if (i == 1 && (input_option[0] == 'n' || input_option[0] == 'N')) {
        // if chose not to shutdown
        printf("\nOk, Please remember to use 'sudo shutdown now'\n");
        printf("to shutdown the Pi before unpluging the power.\n\n");
    }
    else { // chose to shutdown
        printf("\nShuting down ... ...\n");
        system("shutdown -P now");
    }

    return 0;
}

/*********************************************************************
***************************** THREADS ********************************
*********************************************************************/
void* LX_thread_func(void* ptr){
    while (state_ != EXITING) {
        if (set_speed_.LX == 0 || reached_position_.LX == 1 || stop_.LX == 1) {
            if (current_position_.LX == target_position_.LX){
                reached_position_.LX = 1;
            }
            digitalWrite(PIN_LX_PUL, LOW);
            nsleep(1000000);
        }
        else {
            drive(PIN_LX_PUL, PIN_LX_DIR, set_speed_.LX, \
                target_position_.LX - current_position_.LX, \
                &(current_position_.LX), &(stop_.LX), POLARITY_LX);
            if (!stop_.LX) reached_position_.LX = 1;
        }
    }
    return NULL;
}

void* LY_thread_func(void* ptr){
    while (state_ != EXITING) {
        if (set_speed_.LY == 0 || reached_position_.LY == 1 || stop_.LY == 1) {
            if (current_position_.LY == target_position_.LY){
                reached_position_.LY = 1;
            }
            digitalWrite(PIN_LY_PUL, LOW);
            nsleep(1000000);
        }
        else {
```

```
            drive(PIN_LY_PUL, PIN_LY_DIR, set_speed_.LY, \
                target_position_.LY - current_position_.LY, \
                &(current_position_.LY), &(stop_.LY), POLARITY_LY);
            if (!stop_.LY) reached_position_.LY = 1;
        }
    }
    return NULL;
}

void* RX_thread_func(void* ptr){
    while (state_ != EXITING) {
        if (set_speed_.RX == 0 || reached_position_.RX == 1 || stop_.RX == 1) {
            if (current_position_.RX == target_position_.RX){
                reached_position_.RX = 1;
            }
            digitalWrite(PIN_RX_PUL, LOW);
            nsleep(1000000);
        }
        else {
            drive(PIN_RX_PUL, PIN_RX_DIR, set_speed_.RX, \
                target_position_.RX - current_position_.RX, \
                &(current_position_.RX), &(stop_.RX), POLARITY_RX);
            if (!stop_.RX) reached_position_.RX = 1;
        }
    }
    return NULL;
}

void* RY_thread_func(void* ptr){
    while (state_ != EXITING) {
        if (set_speed_.RY == 0 || reached_position_.RY == 1 || stop_.RY == 1) {
            if (current_position_.RY == target_position_.RY){
                reached_position_.RY = 1;
            }
            digitalWrite(PIN_RY_PUL, LOW);
            nsleep(1000000);
        }
        else {
            drive(PIN_RY_PUL, PIN_RY_DIR, set_speed_.RY, \
                target_position_.RY - current_position_.RY, \
                &(current_position_.RY), &(stop_.RY), POLARITY_RY);
            if (!stop_.RY) reached_position_.RY = 1;
        }
    }
    return NULL;
}

void* cut_manager_func(void* ptr){
    char buf[500];
    while(state_ == GCODE && fgets(buf,500, ptr_file_)!=NULL){
        while (state_ == GCODE && !allreached()) {
            nsleep(10000000);
        } // wait till last coord is reached
        removespace(buf); // remove spaces
        if (!strncmp(buf,"G4P",3)) { // check if is a pause statement
            float temp; str2f(buf+3, &temp);
            nsleep((uint64_t)(floor(temp*1.0E9)));
        }
        else if (!strncmp(buf,"G1",2)) { // if a cut statement
            check_cord(buf); // read coordinates and update global coord_
            // Update new target position
            target_position_.LX = (int32_t)floor((coord_.LX + coord_offset_x_) * MM2PULSE);
            target_position_.LY = (int32_t)floor((coord_.LY + coord_offset_y_) * MM2PULSE);
            target_position_.RX = (int32_t)floor((coord_.RX + coord_offset_x_) * MM2PULSE);
            target_position_.RY = (int32_t)floor((coord_.RY + coord_offset_y_) * MM2PULSE);
            // Calculate time to move to the next coord
            float dL = sqrt( pow(target_position_.LX - current_position_.LX,2.0) \
                           + pow(target_position_.LY - current_position_.LY,2.0));
            float dR = sqrt( pow(target_position_.RX - current_position_.RX,2.0) \
                           + pow(target_position_.RY - current_position_.RY,2.0));
            float time = (dL + dR)/2.0/FEEDRATE/MM2PULSE;
            // Set speed for all 4 axis
            set_speed_.LX = (target_position_.LX - current_position_.LX)/time/MM2PULSE;
            set_speed_.LY = (target_position_.LY - current_position_.LY)/time/MM2PULSE;
            set_speed_.RX = (target_position_.RX - current_position_.RX)/time/MM2PULSE;
            set_speed_.RY = (target_position_.RY - current_position_.RY)/time/MM2PULSE;
            if (state_ == GCODE) {
                // Start the cut by setting reached_position_ to 0
                reached_position_.LX = reached_position_.LY = 0;
                reached_position_.RX = reached_position_.RY = 0;
                // wait till new coord is reached
                nsleep((uint64_t)(floor(time*1.0E9)));
            }
        } // end while --- read line by line
    }
    // if the state_ is still GCODE, but the while loop ended;
    // means the end of file is reached, and therefore cut is complete
```

```
    if (state_ == GCODE) { state_ = HOMED;}
    return NULL;
}

void* print_func(void* ptr){
    struct timespec current_time;
    int elapsed_time = 0;
    int remain_time;
    while(state_ == GCODE){
        clock_gettime( CLOCK_REALTIME, &current_time);
        elapsed_time = current_time.tv_sec - start_time_.tv_sec;
        remain_time = ETA_ - elapsed_time;
        printf("\r");
        printf("%7.3f/%8.3f|%7.3f/%8.3f|%7.3f/%8.3f|%7.3f/%8.3f|    ", \
            current_position_.LX/MM2PULSE*MM2IN , current_position_.LX/MM2PULSE, \
            current_position_.LY/MM2PULSE*MM2IN , current_position_.LY/MM2PULSE, \
            current_position_.RX/MM2PULSE*MM2IN , current_position_.RX/MM2PULSE, \
            current_position_.RY/MM2PULSE*MM2IN , current_position_.RY/MM2PULSE);
        print_time(elapsed_time); printf("    |   ");
        print_time(remain_time);  printf("   |");
        fflush(stdout);
        nsleep(1000000000); // run at 1 Hz
    }
    return NULL;
}

void* switch_thread_func(void* ptr){
    int state_P = 0;
    int counter_P, counter_S;
    int counter_S2 = 0;
    while (state_ != EXITING) {
        counter_P = counter_S = 0;
        for (int i = 1; i<= 41; i++) {
            if (digitalRead(PIN_PAUSE)) counter_P ++;
            if (digitalRead(PIN_STOP )) counter_S ++;
            nsleep(500000);
        }

        if (counter_P > 25 && !state_P) {
            stop_all();
            state_P = 1;
        }
        else if (counter_P <= 15 && state_P && state_ == GCODE) {
            state_P = 0;
            stop_.LX = stop_.LY = stop_.RX = stop_.RY = 0;
            digitalWrite(PIN_RELAY, HIGH);
        }
        else if (counter_P <= 15 && state_P && state_ == HOMED) {
            state_P = 0;
            stop_.LX = stop_.LY = stop_.RX = stop_.RY = 0;
        }

        if (counter_S > 25) counter_S2 ++;
        else counter_S2 = 0;

        if (counter_S2 == 100){
            state_ = EXITING; stop_all(); state_STOP_ = 1;
            printf("\n\n************************** EXITING PROGRAM **************************\n");
        }
    }
    return NULL;
}

/*****************************************************************
********************** SYSTEM FUNCTIONS **************************
*****************************************************************/

void initialize_pin(){
    // Setup limit switch pins
    pinMode(PIN_LX_LIM,INPUT); pullUpDnControl(PIN_LX_LIM,PUD_DOWN);
    pinMode(PIN_LY_LIM,INPUT); pullUpDnControl(PIN_LY_LIM,PUD_DOWN);
    pinMode(PIN_RX_LIM,INPUT); pullUpDnControl(PIN_RX_LIM,PUD_DOWN);
    pinMode(PIN_RY_LIM,INPUT); pullUpDnControl(PIN_RY_LIM,PUD_DOWN);
    // Setup motor dive pins
    pinMode(PIN_LX_DIR,OUTPUT); pinMode(PIN_LX_PUL,OUTPUT);
    pinMode(PIN_LY_DIR,OUTPUT); pinMode(PIN_LY_PUL,OUTPUT);
    pinMode(PIN_RX_DIR,OUTPUT); pinMode(PIN_RX_PUL,OUTPUT);
    pinMode(PIN_RY_DIR,OUTPUT); pinMode(PIN_RY_PUL,OUTPUT);
    // Setup switches pins
    pinMode(PIN_PAUSE,INPUT); pullUpDnControl(PIN_PAUSE,PUD_DOWN);
    pinMode(PIN_STOP ,INPUT); pullUpDnControl(PIN_STOP ,PUD_DOWN);
    // Setup relay control pin
    pinMode(PIN_RELAY,OUTPUT);
    return;
}

void home(){
```

```
    // Print Header
    printf("I see the system is not homed yet, please press ENTER to home the system:    ");
    fflush(stdout);
    if (menu_enter() == -2) return; // if EXITING state, end function
    printf("Homing ...   "); fflush(stdout);

    // Home the system
    current_position_.LX = current_position_.LY = current_position_.RX = current_position_.RY = 0;
    reached_position_.LX = reached_position_.LY = reached_position_.RX = reached_position_.RY = 0;
    target_position_.LX  = target_position_.LY  = target_position_.RX  = target_position_.RY  = -640000;

    // Home X axis
    set_speed_.LY = set_speed_.RY = 0.0;
    set_speed_.LX = set_speed_.RX = -1.0;

    int counter1, counter2;
    int state1 = 0; int state2 = 0;

    while (state_ != EXITING && (!state1 || !state2)) {
        counter1 = counter2 = 0;
        for (int i = 1; i<= 41; i++) {
            if (digitalRead(PIN_LX_LIM)) counter1 ++;
            if (digitalRead(PIN_RX_LIM)) counter2 ++;
            nsleep(20000);
        }
        if (counter1 > 30 && !state1) {
            state1 = 1;
            stop_.LX = 1;
            set_speed_.LX = 0.0;
            current_position_.LX = LIM2ORIGIN_LX;
            reached_position_.LX = 1;
        }
        if (counter2 > 30 && !state2) {
            state2 = 1;
            stop_.RX = 1;
            set_speed_.RX = 0.0;
            current_position_.RX = LIM2ORIGIN_RX;
            reached_position_.RX = 1;
        }
    }

    if (state_ == EXITING) return;
    nsleep(200000000);
    target_position_.LX = LIM2ORIGIN_LX + 1000;
    target_position_.RX = LIM2ORIGIN_RX + 1000;
    set_speed_.LX       = set_speed_.RX       = +FEEDRATE;
    stop_.LX = stop_.RX = 0;
    reached_position_.LX = reached_position_.RX = 0;
    while (state_ != EXITING && (!reached_position_.LX || !reached_position_.RX)) { nsleep(1000000);}

    // Home Y axis
    if (state_ == EXITING) return;
    nsleep(500000000);
    set_speed_.LX = set_speed_.RX = 0.0;
    set_speed_.LY = set_speed_.RY = -1.0;

    counter1 = counter2 = state1 = state2 = 0;
    while (state_ != EXITING && (!state1 || !state2)) {
        counter1 = counter2 = 0;
        for (int i = 1; i<= 41; i++) {
            if (digitalRead(PIN_LY_LIM)) counter1 ++;
            if (digitalRead(PIN_RY_LIM)) counter2 ++;
            nsleep(20000);
        }
        if (counter1 > 30 && !state1) {
            state1 = 1;
            stop_.LY = 1;
            set_speed_.LY = 0.0;
            current_position_.LY = LIM2ORIGIN_LY;
            reached_position_.LY = 1;
        }
        if (counter2 > 30 && !state2) {
            state2 = 1;
            stop_.RY = 1;
            set_speed_.RY = 0.0;
            current_position_.RY = LIM2ORIGIN_RY;
            reached_position_.RY = 1;
        }
    }
    if (state_ == EXITING) return;
    nsleep(500000000);

    // Move to orgin
    if (state_ == EXITING) return;
    printf("All limits reached, Moving to Origin ...   "); fflush(stdout);

    target_position_.LX = target_position_.LY = target_position_.RX = target_position_.RY = 0;
```

```
    set_speed_.LX       = set_speed_.LY       = set_speed_.RX       = set_speed_.RY       = +FEEDRATE;
    stop_.LX = stop_.LY = stop_.RX = stop_.RY = 0;

    reached_position_.LY = reached_position_.RY = reached_position_.LX = reached_position_.RX = 0;
    while (state_ != EXITING && !allreached()) { nsleep(1000000);}

    if (state_ == EXITING) return;
    state_ = HOMED;
    printf("Homing Complete! \n\n");

    return;
}

int loadtext(char* filename){
    float pause_time = 0.0;
    float cut_length = 0.0;
    int asym_cut = 0;
    int error = 0; // set error to 1 will return to main menu
    float span;
    float coord_x_min;
    float coord_y_min;
    float width;
    float height;

    coord_lim_.LX_max = coord_lim_.LX_min = 0.0;
    coord_lim_.LY_max = coord_lim_.LY_min = 0.0;
    coord_lim_.RX_max = coord_lim_.RX_min = 0.0;
    coord_lim_.RY_max = coord_lim_.RY_min = 0.0;

    coord_.LX_old = coord_.LX = 0.0;
    coord_.LY_old = coord_.LY = 0.0;
    coord_.RX_old = coord_.RX = 0.0;
    coord_.RY_old = coord_.RY = 0.0;

    coord_offset_x_ = coord_offset_y_ = 0.0;
/************************** READ FILE ******************************/
    FILE *ptr_file; char buf[500];
    ptr_file =fopen(filename, "r");

    int line_numb = 1;
    while (fgets(buf,500, ptr_file)!=NULL){ // get line by line
        removespace(buf); // remove spaces
        if (!strncmp(buf,"G4P",3)) { // check if is a pause statement
            float temp;
            if (str2f(buf+3, &temp)) {pause_time += temp;}
            else {
                printf("G-code error at line %d. Returning to Main Menu.\n\n",line_numb);
                error = 1;
                return 1;
            }
        }
        else if (!strncmp(buf,"G1",2)) { // if a cut statement
            if (check_cord(buf)) { // check if statement is valid
                cut_length += cut_length_func();
                if (fabs(coord_.LX - coord_.RX) + fabs(coord_.LY - coord_.RY) > 0.0001 ) {
                    asym_cut ++;
                }
                // update the max/min coordinates
                coord_lim_.LX_max = max(coord_lim_.LX_max, coord_.LX);
                coord_lim_.LX_min = min(coord_lim_.LX_min, coord_.LX);
                coord_lim_.LY_max = max(coord_lim_.LY_max, coord_.LY);
                coord_lim_.LY_min = min(coord_lim_.LY_min, coord_.LY);
                coord_lim_.RX_max = max(coord_lim_.RX_max, coord_.RX);
                coord_lim_.RX_min = min(coord_lim_.RX_min, coord_.RX);
                coord_lim_.RY_max = max(coord_lim_.RY_max, coord_.RY);
                coord_lim_.RY_min = min(coord_lim_.RY_min, coord_.RY);
            }
            // if not a valid line, print error message and stop reading
            else {
                printf("G-code error at line %d. Returning to Main Menu.\n\n",line_numb);
                error = 1;
                return 1;
            }
        }
        line_numb ++;
    } // end while --- read line by line
    fclose(ptr_file); // read complete

    if (!error && state_ != EXITING) { // if no reading error occured
        coord_x_min = min(coord_lim_.LX_min,coord_lim_.RX_min);
        coord_y_min = min(coord_lim_.LY_min,coord_lim_.RY_min);
    } // end if --- check error
/*************** FIRST: check if offset is needed ******************/
    // check x
    if (coord_x_min < 0 && state_ != EXITING && !error){
        printf("I see you have a min x coordinate of %7.3f in (%8.3f mm)\n", \
            coord_x_min*MM2IN, coord_x_min);
```

```
        printf("A negative value is not allowed\n");
        printf("You can use the minimum offset, or enter one yourself\n");
        printf("Would you like to use the minimum offset for x?\n");
        if (!menu_yes()){
            while(state_ != EXITING) {
                int temp = menu_enter_one(&coord_offset_x_,"Please enter the x offset");
                if      (temp == -1) {printf("Invalid input, please enter again.\n\n");}
                else if (temp ==  1) {
                    if (coord_offset_x_ < - coord_x_min) {
                        printf("Insufficient x offset, please enter a bigger x offset\n\n");
                    }
                    else break;
                }
            }
        }
        else { coord_offset_x_ = - coord_x_min;}
    }
    // check y
    if (coord_y_min < 0 && state_ != EXITING && !error){
        printf("I see you have a min y coordinate of %7.3f in (%8.3f mm)\n", \
            coord_y_min*MM2IN, coord_y_min);
        printf("A negative value is not allowed\n");
        printf("You can use the minimum offset, or enter one yourself\n");
        printf("If you are cutting part of a 3-piece wing\n");
        printf("I would recommend enter the same offset for all 3 pieces.\n");
        printf("It will make the vaccum bagging easier\n\n");
        printf("Would you like to use the minimum offset for y?\n");
        if (!menu_yes()){
            while(state_ != EXITING) {
                int temp = menu_enter_one(&coord_offset_y_,"Please enter the y offset");
                if      (temp == -1) {printf("Invalid input, please enter again.\n\n");}
                else if (temp ==  1) {
                    if (coord_offset_y_ < - coord_y_min) {
                        printf("Insufficient y offset, please enter a bigger y offset\n\n");
                    }
                    else break;
                }
            }
        }
        else { coord_offset_y_ = - coord_y_min;}
    }
    coord_y_min += coord_offset_x_;
    coord_y_min += coord_offset_y_;
    coord_lim_.LX_max += coord_offset_x_; coord_lim_.LX_min += coord_offset_x_;
    coord_lim_.LY_max += coord_offset_y_; coord_lim_.LY_min += coord_offset_y_;
    coord_lim_.RX_max += coord_offset_x_; coord_lim_.RX_min += coord_offset_x_;
    coord_lim_.RY_max += coord_offset_y_; coord_lim_.RY_min += coord_offset_y_;

    if (coord_lim_.LX_max > X_MAX || coord_lim_.RX_max > X_MAX) {
        printf("X axis out of bound, maximum X distance is 29 inches\n");
        printf("Returning to G-code Menu\n\n");
        error = 1; gcode_menu_option_ = -1;
    }
    if (coord_lim_.LY_max > Y_MAX || coord_lim_.RY_max > Y_MAX) {
        printf("Y axis out of bound, maximum Y distance is 16 inches\n");
        printf("Returning to G-code Menu\n\n");
        error = 1; gcode_menu_option_ = -1;
    }
/************** SECOND: determine and check foamsize ****************/
    if(!asym_cut && state_ != EXITING && !error){ // if not an asymetric cut
        printf("I see this is a symmetric cut\n");
        printf("The minimum require foam size is:\n");
        width  = coord_lim_.LX_max;
        height = coord_lim_.LY_max;
        printf("Width     (x-direction): %7.3f in (%8.3f mm)\n", width*MM2IN, width);
        printf("Thickness (y-direction): %7.3f in (%8.3f mm)\n", height*MM2IN, height);
        printf("Please leave some extra space.\n");
    }
    else if (asym_cut && state_ != EXITING && !error){ // if an asymetric cut
        printf("I see this is an asymmetric cut\n");
        printf("The minimum require foam size depends on the span of the cut.\n");
        printf("Please enter the span size of the cut.\n");
        while(state_ != EXITING && menu_enter_one(&span,"Please enter the span size") == -1) {
                printf("Invalid input, please enter again.\n\n");
        }
        width  = min(coord_lim_.LX_max,coord_lim_.RX_max) + \
            fabs(coord_lim_.LX_max-coord_lim_.RX_max)*(CUTTERWIDTH + span)/2.0/CUTTERWIDTH;
        height = min(coord_lim_.LY_max,coord_lim_.RY_max) + \
            fabs(coord_lim_.LY_max-coord_lim_.RY_max)*(CUTTERWIDTH + span)/2.0/CUTTERWIDTH;
        if (state_ != EXITING){
            printf("Width     (x-direction): %7.3f in (%8.3f mm)\n", width*MM2IN, width);
            printf("Thickness (y-direction): %7.3f in (%8.3f mm)\n", height*MM2IN, height);
            printf("Please leave some extra space.\n");
        }
    }
    if(state_ != EXITING && !error){
        printf("Does this look correct and matches your foam size?\n"); fflush(stdout);
```

```
        if (!menu_yes()){
            printf("Looks like there's something wrong. Returning to G-code Menu.\n\n");
            error = 1; gcode_menu_option_ = -1;
        }
    }
/****************** THIRD: review cut settings *********************/
    if (state_ != EXITING && !error){
        printf("************************** CUT SETTINGS ****************************\n");
        printf("G-code file: %s.\n", filename);
        if (asym_cut) {
            printf("Asymetric Cut of span %7.3f in (%8.3f mm)\n", span*MM2IN, span);
        }
        else {printf("Symmetric Cut\n");}
        printf("Minimum Foam Size:\n");
        printf("Width     (x-direction): %7.3f in (%8.3f mm)\n", width*MM2IN, width);
        printf("Thickness (y-direction): %7.3f in (%8.3f mm)\n", height*MM2IN, height);
        if (coord_offset_x_) {
            printf("x offset of %7.3f in (%8.3f mm)", coord_offset_x_*MM2IN, coord_offset_x_);
        }
        else {printf("No x offset");}
        printf(" and ");
        if (coord_offset_y_) {
            printf("y offset of %7.3f in (%8.3f mm)", coord_offset_y_*MM2IN, coord_offset_y_);
        }
        else {printf("No y offset");}
        printf("\n");
        printf("Estimate total time of cut: ");
        ETA_ = round(pause_time + cut_length/FEEDRATE );
        print_time(ETA_); printf("\n");

        if (asym_cut) {printf("Please make sure the foam is centered.\n");}
        printf("\nWould you like to start the cut?\n");

        if (!menu_yes()){
            printf("OK, setting incorrect. Returning to G-code Menu.\n\n");
            error = 1; gcode_menu_option_ = -1;
        }
    }
    return error;
}

int check_cord(char* str){
    coord_.LX_old = coord_.LX; coord_.LY_old = coord_.LY;
    coord_.RX_old = coord_.RX; coord_.RY_old = coord_.RY;
    char* ptr_X = strchr(str, 'X');
    char* ptr_Y = strchr(str, 'Y');
    char* ptr_Z = strchr(str, 'Z');
    char* ptr_A = strchr(str, 'A');
    if (ptr_X && ptr_Y && ptr_Z && ptr_A &&  \
        (ptr_X < ptr_Y) && (ptr_Y < ptr_Z) &&(ptr_Z < ptr_A) ){
        char temp1[20], temp2[20], temp3[20], temp4[20];
        float tempf1, tempf2, tempf3, tempf4;
        strncpy(temp1,ptr_X+1,ptr_Y-ptr_X-1); temp1[(ptr_Y-ptr_X-1)] = '\0';
        strncpy(temp2,ptr_Y+1,ptr_Z-ptr_Y-1); temp2[(ptr_Z-ptr_Y-1)] = '\0';
        strncpy(temp3,ptr_Z+1,ptr_A-ptr_Z-1); temp3[(ptr_A-ptr_Z-1)] = '\0';
        strncpy(temp4,ptr_A+1,10           );

        if(str2f(temp1, &tempf1) && str2f(temp2, &tempf2) && \
            str2f(temp3, &tempf3) && str2f(temp4, &tempf4)){
            coord_.LX = tempf1; coord_.LY = tempf2;
            coord_.RX = tempf3; coord_.RY = tempf4;
            return 1;
        }
        else return 0;
    }
    else return 0;
}

int allreached() {
    return reached_position_.LX * reached_position_.LY * reached_position_.RX * reached_position_.RY;
}

void stop_all() {
    stop_.LX = stop_.LY = stop_.RX = stop_.RY = 1;
    digitalWrite(PIN_RELAY,LOW);
    return;
}

float cut_length_func(){
    float L_length = sqrt(pow((coord_.LX - coord_.LX_old),2.0) + pow((coord_.LY - coord_.LY_old),2.0));
    float R_length = sqrt(pow((coord_.RX - coord_.RX_old),2.0) + pow((coord_.RY - coord_.RY_old),2.0));
    return (L_length + R_length)/2.0;
}

void drive(int pin_pul, int pin_dir, float speed, int32_t delta_pulse, int* ptr_current, int* ptr_stop, int polarity ){
    int inc;
    if      (speed*polarity > 0) { digitalWrite(pin_dir, HIGH); inc =  1*polarity;}
```

```
    else if (speed*polarity < 0) { digitalWrite(pin_dir, LOW ); inc = -1*polarity;}
    // the time between pulses, calculated from speed
    // 4000 is the sleep time in the loop;
    // 100000 is the approximate code execution time;
    uint64_t sleep_time = floor(1000000000.0/MM2PULSE/fabs(speed)) - 4000 - 100000;
    nsleep(5000); // ensure dir pin leads by at least 5 microsec
    for (int i = 0; i < abs(delta_pulse); i++) { // send out desired number of pulses
        digitalWrite(pin_pul, HIGH);
        nsleep(4000); // ensure pulse width of at least 2 microsec
        digitalWrite(pin_pul, LOW );
        *ptr_current += inc;
        if (*ptr_stop) break; // stop the motor if stop is a 1;
        nsleep(sleep_time);
    }
    return;
}

void cut_gcode(char* filename){
    state_ = GCODE;
    if (state_ == EXITING) return;
    moveto(-5.0,0.0);
    while (state_ != EXITING && !allreached()){ nsleep(1000000);}
    if (state_ == EXITING) return;
    printf("Please connect and turn on the power supply for the hot wire\n");
    printf("Please make sure the voltage is approximately 10V, and press ENTER to continue:    ");
    fflush(stdout);
    if (menu_enter() == -2) return;
    digitalWrite(PIN_RELAY,HIGH);
    printf("Please now adjust the power supply to the desired current.\n");
    printf("Recommend 2.1 to 2.3 Amps, depending on the cut span.\n");
    printf("Use higher current for wider cuts.\n");
    printf("Increase current if wire bows significantly.\n");
    printf("Press ENTER to start cutting:    ");
    fflush(stdout);
    if (menu_enter() == -2) return;
    printf("Heating wire ... ..."); fflush(stdout);
    if (state_ != EXITING) nsleep(5000000000);
    if (state_ == EXITING) return;
    printf(" Cut Starting\n");
    moveto(0.0,0.0);
    while (state_ != EXITING && !allreached()) nsleep(1000000);
    if (state_ == EXITING) return;

    coord_.LX_old = coord_.LX = 0.0;
    coord_.LY_old = coord_.LY = 0.0;
    coord_.RX_old = coord_.RX = 0.0;
    coord_.RY_old = coord_.RY = 0.0;
    clock_gettime( CLOCK_REALTIME, &start_time_);

    ptr_file_ =fopen(filename, "r");
    printf("       LX       |       LY       |       RX       |       RY       |          TIME          |\n");
    printf("   in  /   mm   |   in  /   mm   |   in  /   mm   |   in  /   mm   |  Elapsed  | Remaining |\n");

    params_print_thread.sched_priority = 40;
    params_cut_manager.sched_priority = 99;
    pthread_setschedparam(printing_thread, SCHED_FIFO, &params_print_thread);
    pthread_create(&printing_thread, NULL, print_func, (void*) NULL);
    pthread_setschedparam(cut_manager, SCHED_FIFO, &params_cut_manager);
    pthread_create(&cut_manager, NULL, cut_manager_func, (void*) NULL);

    while (state_ == GCODE) nsleep(100000);

    fclose(ptr_file_); // read complete

    if (state_ != EXITING) moveto(-5.0,0.0);
    while (state_ != EXITING && !allreached()) nsleep(1000000);
    digitalWrite(PIN_RELAY,LOW);
    if (state_ != EXITING) moveto( 0.0,0.0);
    while (state_ != EXITING && !allreached()) nsleep(1000000);
    if (state_ != EXITING) printf("\nCut Complete, Returning to Main Menu.\n\n");

    pthread_join(cut_manager, NULL);
    pthread_join(printing_thread, NULL);
    return;
}

void moveto(float x, float y){
    target_position_.LX = target_position_.RX = (int32_t)floor(x * MM2PULSE);
    target_position_.LY = target_position_.RY = (int32_t)floor(y * MM2PULSE);
    // Set speed for all 4 axis
    if      (target_position_.LX  > current_position_.LX) set_speed_.LX = +FEEDRATE;
    else if (target_position_.LX  < current_position_.LX) set_speed_.LX = -FEEDRATE;
    else if (target_position_.LX == current_position_.LX) set_speed_.LX = 0.0;

    if      (target_position_.LY  > current_position_.LY) set_speed_.LY = +FEEDRATE;
    else if (target_position_.LY  < current_position_.LY) set_speed_.LY = -FEEDRATE;
    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:
```

```
                    if (state_ != EXITING && !loadtext(namelist[gcode_menu_option_-1]->d_name) ) {
                        if (state_ != EXITING) cut_gcode(namelist[gcode_menu_option_-1]->d_name);
                        return;
                    }
                    break;
                case 0:
                    printf("OK, Let try again\n\n");
                    gcode_menu_option_ = -1;
                    break;
                }
            }
            else if (gcode_menu_option_ == 0 ) {
                printf("Please put the desired gcode file in the working directory.\n");
                printf("Returning to Main Menu.\n\n");
                break;
            }
            else if (gcode_menu_option_ == -1 ) { // Invalid input
                printf("Invalid Input, let's try again.\n\n");
            }
            else if (gcode_menu_option_ == -2 ) { // EXITING state
                break;
            } // end of if --- menu input check
        } // end of if --- file number check
    } // end of while
    return;
} // end of gcode_menu

void move_menu(){
    float x,y;
    while (state_ != EXITING){
        float current_x = (current_position_.LX + current_position_.RX)/2.0/MM2PULSE;
        float current_y = (current_position_.LY + current_position_.RY)/2.0/MM2PULSE;
        printf("****************************** MOVE MENU ******************************\n");
        printf("The current wire position is (x,y) = (%.3f,%.3f) in = (%.3f,%.3f) mm\n", \
            current_x*MM2IN,current_y*MM2IN,current_x,current_y);
        printf("Please choose from the following options:\n");
        printf("a. Move wire to a specific location relative to origin (Absolute Location);\n");
        printf("b. Move wire to a specific location relative to current position (Increment);\n");
        printf("c. Move wire to origin;\n");
        printf("d. Return to Main Menu;\n");
        printf("Please enter the corresponding letter and press ENTER key:    ");
        fflush(stdout);
        switch (menu(4)) {
            case  0: { // Move to a specific location
                while(state_ != EXITING && \
                    (menu_enter_two(&x,&y,"Please enter the destination x and y coordinates RELATIVE TO ORIGIN") == -1 \
                        || x < 0 || y <0)) {
                    printf("Invalid input, please enter again. Please note that negative destination is not allowed.\n\n");
                }
                if (state_ != EXITING) {
                    printf("|      |       Current      |      Increment     |     Destination    |\n");
                    printf("| Inch |  %7.3f,  %7.3f |  %7.3f,  %7.3f |  %7.3f,  %7.3f |\n", \
                        current_x*MM2IN, current_y*MM2IN, (x-current_x)*MM2IN, (y-current_y)*MM2IN, x*MM2IN, y*MM2IN);
                    printf("|  MM  | %8.3f, %8.3f | %8.3f, %8.3f | %8.3f, %8.3f |\n", \
                        current_x, current_y, (x-current_x), (y-current_y), x, y);
                    printf("Continue?\n");
                    switch (menu_yes()){
                    case 1:
                        if (state_ != EXITING ) {
                            printf("Moving ... ..."); fflush(stdout);
                            moveto(x,y);
                            while (state_ != EXITING && !allreached()){ nsleep(1000000);}
                            if (state_ != EXITING) printf(" Destination Reached\n\n");
                        }
                        break;
                    case 0:
                        printf("OK, Let try again\n\n");
                        break;
                    }
                }
                break; } // end case --- move to a specific location

            case  1: { // Move to a specific location
                while(state_ != EXITING && \
                    (menu_enter_two(&x,&y,"Please enter the destination x and y coordinates RELATIVE TO CURRENT POSITION") == -1 \
                        || x+current_x < 0 || y+current_y <0)) {
                    printf("Invalid input, please enter again. Please note that negative destination is not allowed.\n\n");
                }
                if (state_ != EXITING) {
                    printf("|      |       Current      |      Increment     |     Destination    |\n");
                    printf("| Inch |  %7.3f,  %7.3f |  %7.3f,  %7.3f |  %7.3f,  %7.3f |\n", \
                        current_x*MM2IN, current_y*MM2IN, x*MM2IN, y*MM2IN, (x+current_x)*MM2IN, (y+current_y)*MM2IN);
                    printf("|  MM  | %8.3f, %8.3f | %8.3f, %8.3f | %8.3f, %8.3f |\n", \
                        current_x, current_y, x, y, (x+current_x), (y+current_y));
                    printf("Continue?\n");
                    switch (menu_yes()){
                    case 1:
```

```
                        if (state_ != EXITING ) {
                            printf("Moving ... ..."); fflush(stdout);
                            moveto(x+current_x,y+current_y);
                            while (state_ != EXITING && !allreached()){ nsleep(1000000);}
                            if (state_ != EXITING) printf(" Destination Reached\n\n");
                        }
                        break;
                    case 0:
                        printf("OK, Let try again\n\n");
                        break;
                    }
                }
                break; } // end case --- move to a specific location

            case  2: { //case --- move to origin
                if (!current_position_.LX && !current_position_.LY && \
                    !current_position_.RX && !current_position_.RY) {
                    printf("Already at Origin\n\n");
                }
                else{
                    printf("Move to Orignin. Continue?\n");
                    switch (menu_yes()){
                    case 1:
                        printf("Moving ... ..."); fflush(stdout);
                        moveto(0.0,0.0);
                        while (state_ != EXITING && !allreached()){ nsleep(1000000);}
                        if (state_ != EXITING) printf(" Origin Reached\n\n");
                        break;
                    case 0:
                        printf("OK, Let try again\n\n");
                        break;
                    }
                }
                break; } // end case --- move to origin
            case -1: {printf("Invalid option. Let's try again.\n\n"); break; }
            default: {return;                                        break; }
        } // end switch
    } // end while
    return;
}

int menu(int numb_of_options){
    // pass in number of menu options, maximum of 10 options
    // return 0 to (numb_of_options - 1) if input is within the range
    // return -1 for invalid input
    // return -2 when state_ is exiting

    if (state_ ==  EXITING) return -2;
    fd_set          input_set;
    struct timeval  timeout;
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    // Listening for input stream for any activity
    // If there
    FD_ZERO(&input_set );
    FD_SET(0, &input_set);
    while (state_ !=  EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
        timeout.tv_sec = 1;
        FD_ZERO(&input_set );
        FD_SET(0, &input_set);
    }

    if (state_ ==  EXITING) return -2;

    // get input
    char input_option[256]; fgets(input_option,256,stdin);
    // determine length of input
    int i; for(i=0; input_option[i]!='\0'; i++); i --;
    int result;

    if (i == 1) {
        if     (input_option[0] >= 48 && input_option[0] <= 57) {
            result = input_option[0]-48;
        }
        else if(input_option[0] >= 65 && input_option[0] <= 74) {
            result = input_option[0]-65;
        }
        else if(input_option[0] >= 97 && input_option[0] <= 106) {
            result = input_option[0]-97;
        }
        else result = -1;
    }
    else result = -1;
    if (result >= numb_of_options) result = -1;

    printf("\n");
```

```
    return result;
}

int menu_enter(){
    // return 1 if anthing is entered, including ENTER key
    // return 0 if m is entered
    // return -2 when state_ is exiting
    if (state_ ==  EXITING) return -2;
    fd_set          input_set;
    struct timeval  timeout;
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    // Listening for input stream for any activity
    FD_ZERO(&input_set );
    FD_SET(0, &input_set);
    while (state_ !=  EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
        timeout.tv_sec = 1;
        FD_ZERO(&input_set );
        FD_SET(0, &input_set);
    }

    if (state_ ==  EXITING) return -2;

    // get input
    char input_option[256]; fgets(input_option,256,stdin);
    // determine length of input
    int i; for(i=0; input_option[i]!='\0'; i++); i --;

    if (i == 1 && (input_option[0] == 'm' || input_option[0] == 'M')) {
        printf("\n"); return 0; // return 0 if input is 'm' or 'M'
    }
    else {printf("\n"); return 1; }
}

int menu_yes(){
    // return 1 if anthing is entered, including ENTER key
    // return 0 if n is entered
    // return -2 when state_ is exiting
    if (state_ ==  EXITING) return -2;
    printf("Press ENTER for YES, or 'n' then ENTER for NO:    "); fflush(stdout);
    fd_set          input_set;
    struct timeval  timeout;
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    // Listening for input stream for any activity
    FD_ZERO(&input_set );
    FD_SET(0, &input_set);
    while (state_ !=  EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
        timeout.tv_sec = 1;
        FD_ZERO(&input_set );
        FD_SET(0, &input_set);
    }

    if (state_ ==  EXITING) return -2;

    // get input
    char input_option[256]; fgets(input_option,256,stdin);
    // determine length of input
    int i; for(i=0; input_option[i]!='\0'; i++); i --;

    if (i == 1 && (input_option[0] == 'n' || input_option[0] == 'N')) {
        printf("\n"); return 0; // return 0 if input is 'm' or 'M'
    }
    else {printf("\n"); return 1;}
}

int menu_enter_one(float* output, char* string){
    float scale = 1.0;
    if (state_ ==  EXITING) return -2;
    printf("Would you like to enter the coordinate in inches?\n");
    switch (menu_yes()) {
        case 1: scale = 25.4; printf("%s in inches then press ENTER:    ", string); break;
        case 0: scale =  1.0; printf("%s in millimeters then press ENTER:    ", string); break;
    }
    fflush(stdout);
    if (state_ ==  EXITING) return -2;


    fd_set          input_set;
    struct timeval  timeout;
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    // Listening for input stream for any activity
    // If there
```

```
    FD_ZERO(&input_set );
    FD_SET(0, &input_set);
    while (state_ !=  EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
        timeout.tv_sec = 1;
        FD_ZERO(&input_set );
        FD_SET(0, &input_set);
    }

    if (state_ ==  EXITING) return -2;

    // get input
    char input_option[256]; fgets(input_option,256,stdin);
    // remove white spaces
    removespace(input_option);
    float temp;
    if (str2f(input_option, &temp)) {
        *output = temp*scale;
        printf("\n"); return 1;
    }
    else {printf("\n"); return -1;}
}

int menu_enter_two(float* output1, float* output2, char* string){
    float scale = 1.0;
    if (state_ ==  EXITING) return -2;
    printf("Would you like to enter the coordinate in inches?\n");
    switch (menu_yes()) {
        case 1:
        scale = 25.4;
        printf("%s in INCHES \nseparated with comma then press ENTER:    ", string);
        break;
        case 0:
        scale =  1.0;
        printf("%s in MM \nseparated with comma then press ENTER:    ", string);
        break;
    }
    fflush(stdout);
    if (state_ ==  EXITING) return -2;

    fd_set          input_set;
    struct timeval  timeout;
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    // Listening for input stream for any activity
    // If there
    FD_ZERO(&input_set );
    FD_SET(0, &input_set);
    while (state_ !=  EXITING && !select(1, &input_set, NULL, NULL, &timeout)) {
        timeout.tv_sec = 1;
        FD_ZERO(&input_set );
        FD_SET(0, &input_set);
    }
    if (state_ ==  EXITING) return -2;

    // get input
    char input_option[256]; fgets(input_option,256,stdin);
    // remove white spaces
    removespace(input_option);
    // get the two
    char* ptr = strchr(input_option, ',');
    if (ptr == NULL) {return -1;}
    char temp1[20]; strncpy(temp1,input_option,ptr-input_option);
    temp1[(ptr-input_option)] = '\0';
    char temp2[20]; strncpy(temp2,ptr+1,20);
    float temp1f, temp2f;
    if (str2f(temp1, &temp1f) && str2f(temp2, &temp2f)) {
        *output1 = temp1f*scale;
        *output2 = temp2f*scale;
        printf("\n"); return 1;
    }
    else {return -1;}
}

/*****************************************************************
************************* OTHER FUNCTIONS *************************
*****************************************************************/

// Sleep for nanoseconds
void nsleep(uint64_t ns){
    struct timespec req,rem;
    req.tv_sec = ns/1000000000;
    req.tv_nsec = ns%1000000000;
    // loop untill nanosleep sets an error or finishes successfully
    errno=0; // reset errno to avoid false detection
    while(nanosleep(&req, &rem) && errno==EINTR){
        req.tv_sec = rem.tv_sec;
```

```
        req.tv_nsec = rem.tv_nsec;
    }
    return;
}

// Filter out file without .txt extension
int file_filter(const struct dirent *entry){
    return !strcmp(entry->d_name + strlen(entry->d_name) -4, ".txt");
}

// Remove space in a string
void removespace(char* str) {
    int i,j=0;
    for(i=0;str[i]!='\0';i++) {
        if(str[i]!=' ' && str[i] != 10 && str[i] != 13)
            str[j++]=str[i];
    }
    str[j]='\0';
    return;
}

// Signal Handler for CTRL+C
void SigHandler(int dummy) {
    state_ = EXITING; stop_all();
    printf("\n\n************************** EXITING PROGRAM **************************\n");
    return;
}

float max(float a, float b) {
    if(a >= b) return a;
    else return b;
}

float min(float a, float b) {
    if(a >= b) return b;
    else return a;
}

void print_time(int sec){
    if (sec <=  0) printf("00:00");
    else printf("%02d:%02d", (int)floor(sec /60.0),sec % 60);
    return;
}

int str2f(char* str, float* output){
    removespace(str);
    float out = 0;
    int dec_location;
    int dec_numb = 0;
    int i; for(i=0; str[i] !='\0'; i++);
    int length = i;
    int j =1;
    for(int i = length-1; i >= 0; i--) {
        if (str[i]>=48 && str[i]<=57){out = out+j*(str[i]-48); j = j*10;}
        else if (str[i] == 46) {dec_numb ++; dec_location = i;}
        else if (i == 0 && str[i] == 43) {out = +out;}
        else if (i == 0 && str[i] == 45) {out = -out;}
        else return 0;
    }
    if (dec_numb == 0) {out = out;}
    else if (dec_numb == 1) {out = out/pow(10.0,length-1-dec_location);}
    else {return 0;}
    *output = out;
    return 1;
}
```