```c
/********************************************************************************
* FINAL PROJECT
*
* This program estimates the angle of the MiP in relation to the horizontal floor.
* It uses the gyro to remain accurate while moving and the accelerometer to provide
* an accurate steady state estimate. They are combined via complementary filtering.
********************************************************************************/

// usefulincludes is a collection of common system includes for the lazy
// This is not necessary for roboticscape projects but here for convenience
#include <rc_usefulincludes.h>
// main roboticscape API header
#include <roboticscape.h>
#include <time.h>

// function declarations
void on_pause_pressed();
void on_pause_released();

void arm_controller();
void disarm_controller();

int waiting_to_start();

void balance_controller();
void* setpoint_controller();
void* printdata();

//set the timestep (inverse of frequency).
#define INNERSAMPLERATE 100
#define OUTERSAMPLERATE 20

#define timestepin      .01
#define timestepout     .05

#define D1_GAIN         1.0
#define D1_ORDER        2
#define D1_SATURATION_TIMEOUT 0.5
#define PREFACTOR       1/1.4


#define D2_GAIN         0.55
#define D2_ORDER        1

#define STEERING_GAIN   .2

#define ANGLE_OFFSET    0.39
#define START_ANGLE_RANGE 0.5
#define START_WAIT      0.25
#define MAX_LEAN        0.4


//Physical things
#define L_MOTOR_POLARITY 1
#define R_MOTOR_POLARITY -1

#define R_MOTOR_CHANNEL 2
```

```c
#define L_MOTOR_CHANNEL 3

#define R_ENCODER_CHANNEL 2
#define L_ENCODER_CHANNEL 3

#define L_ENCODER_POLARITY 1
#define R_ENCODER_POLARITY -1

#define GEARBOX 4*15*35.577


//make imudata struct global to allow ISR funct. to access it as well.
rc_imu_data_t imudata;

//define an armstate enum, used to turn various loops and checks on/off.
typedef enum arm_state_t{
    DISARMED,
    ARMED
}arm_state_t;

arm_state_t armstate=DISARMED;

/*WARNING: THE NEXT 50 LINES ARE NOT FOR PROGRAMMERS FAINT OF HEART*/

/* Needs to be global so arm_controller can zero out vars on restart...*/

//setpoints and steering controller
float phi_setpoint=0;
float phi_diff=0;
float steeringinput=0;


//Initialize innerloop (body angle) variables
float theta_g_raw=0;
float theta_g_raw_last=0;
float theta_g=0;
float theta_a=0;
float theta_a_raw=0;
float theta_f=0;

//initialize angle error stores
float theta_e_2last=0;
float theta_e_last=0;
float theta_e=0;

//Initialize duty cycle storage and steering correction variable
float u=0;
float u_last=0;
float u_2last=0;

//initialize outer loop (setpoint) variables
float rightphi=0;
float leftphi=0;
float avgphi=0;
float theta_ref_last=0;

float phierror=0;
float phierror_last=0;

float theta_ref=0;
```

```


/*******************************************************************************
* int main()
*
* This template main function contains these critical components
* - call to rc_initialize() at the beginning
* - main while loop that checks for EXITING condition
* - rc_cleanup() at the end
*******************************************************************************/
int main(){
    // always initialize cape library first
    if(rc_initialize()){
        fprintf(stderr,"ERROR: failed to initialize rc_initialize(), are you root?\n");
        return -1;
    }

    rc_set_pause_pressed_func(&on_pause_pressed);
    rc_set_pause_released_func(&on_pause_released);

    //initialize an imu_config struct with default values
    rc_imu_config_t imuconfig = rc_default_imu_config();
    //initialize the IMU itself passing it our data and config structs.
    if(rc_initialize_imu_dmp(&imudata, imuconfig)){
        fprintf(stderr,"ERROR: failed to initialize IMU, exiting\n");
        return -1;
    }

    //Set our filtering function as the IMU interrupt function.
    rc_set_imu_interrupt_func(&balance_controller);

    //get a rough estimate to initalize theta_a with;
    if(rc_read_accel_data(&imudata)){
            fprintf(stderr,"ERROR: failed to read sensor data from IMU for initialization.\n");
        }
    theta_a=atan2(-imudata.accel[2],imudata.accel[1]);


    //make it super obivious that the the user must push the pause button to start
    printf("\n##############################################\nPRESS PAUSE TO START BALANCING!!\n############################################\n\n");

    //Print header for data and setup pthread to start printing values.
    printf(" theta_e |   u   | theta_f | theta_ref | avgphi | phi_diff |(1)A/D(0)|\n");

    //Initialize and start running the parallel printing, setpoint and battery
    //checking threads.
    pthread_t print_thread;
    pthread_create(&print_thread, NULL, printdata, (void*) NULL);
    struct sched_param params;
    params.sched_priority=1;
    pthread_setschedparam(print_thread, SCHED_FIFO, &params);

    pthread_t setpoint_thread;
    pthread_create(&setpoint_thread, NULL, setpoint_controller, (void*) NULL);
    struct sched_param sp_params;
    sp_params.sched_priority=1;
    pthread_setschedparam(setpoint_thread, SCHED_FIFO, &sp_params);

```

```c


    // done initializing so set state to RUNNING
    rc_set_state(RUNNING);

    // Keep looping until state changes to EXITING
    while(rc_get_state()!=EXITING){
        // always sleep at some point
        if(armstate==ARMED) {
            rc_set_led(GREEN,ON);
            rc_set_led(RED,OFF);
        }

        else if(armstate==DISARMED) {
            rc_set_led(GREEN,OFF);
            rc_set_led(RED,ON);
        }
        rc_usleep(1000000*timestepin);
    }

    // exit cleanly, join pthreads

    printf("Trying to join printing and setpoint threads\n");
    //try to join print thread within timeout limit
    timespec pthreadtimeout;
    clock_gettime(CLOCK_REALTIME, &pthreadtimeout);
    rc_timespec_add(&pthreadtimeout, 0.5);
    int thread_err = 0;
    thread_err = pthread_timedjoin_np(print_thread, NULL, &pthreadtimeout);
    if(thread_err == ETIMEDOUT){
        printf("WARNING: Print thread exit timeout\n");
        return -1;
    }
    else printf("print thread joined\n");

    //try to join setpoint thread within timeout limit
    clock_gettime(CLOCK_REALTIME, &pthreadtimeout);
    rc_timespec_add(&pthreadtimeout, 0.5);
    thread_err = 0;
    thread_err = pthread_timedjoin_np(setpoint_thread, NULL, &pthreadtimeout);
    if(thread_err == ETIMEDOUT){
        printf("WARNING: setpoint thread exit timeout\n");
        return -1;
    }
    else printf("setpoint thread joined\n");

    //puts imu in low power state
    rc_power_off_imu();
    rc_cleanup();
    return 0;
}


/*******************************************************************************
* void on_pause_pressed()
*
* If the user holds the pause button for 2 seconds, set state to exiting which
* triggers the rest of the program to exit cleanly.
*******************************************************************************/
void on_pause_pressed(){
    int i=0;
    const int samples = 100;    // check for release 100 times in this period
    const int us_wait = 2000000; // 2 seconds

    // now keep checking to see if the button is still held down
    for(i=0;i<samples;i++){
        rc_usleep(us_wait/samples);
        if(rc_get_pause_button() == RELEASED) return;
    }
    printf("long press detected, shutting down\n");
    rc_set_state(EXITING);
    return;
}

void on_pause_released(){
    // toggle betewen paused and running modes
    if(rc_get_state()==RUNNING)     rc_set_state(PAUSED);
    else if(rc_get_state()==PAUSED) rc_set_state(RUNNING);
    if (armstate==ARMED) disarm_controller();
    else if (armstate==DISARMED) arm_controller();
    return;
}
/*******************************************************************************
* void balance_controller()
*
* Using incoming IMU data and theta_ref from outer loop, this calculates an
* appropriate duty cylce to pass to the motors
*******************************************************************************/
void balance_controller(){
    static int inner_sat_timer=0;
    // define crossover frequency for complementary filter.
    static float wc=.5;

    //filter accel data and then calculate the angle in relation to the floor with accelerometer
    theta_a_raw=atan2(-imudata.accel[2],imudata.accel[1]);
    theta_a=-theta_a*(wc*timestepin-1)+wc*timestepin*theta_a_raw;

    //process gyro data
    theta_g_raw_last=theta_g_raw;
    theta_g_raw+=imudata.gyro[0]*DEG_TO_RAD*timestepin;
    theta_g=theta_g_raw-theta_g_raw_last-theta_g*(wc*timestepin-1);

    // combine for final angle estimate
    theta_f=theta_a+theta_g+ANGLE_OFFSET;

    /** INNER LOOP CONTROLLER*/
    theta_e_2last=theta_e_last;
    theta_e_last=theta_e;
    theta_e=(theta_ref*PREFACTOR)-theta_f;

    //evaluate inner loop difference equation
    u_2last=u_last;
    u_last=u;
    u=D1_GAIN*(1.675*u_last-0.675*u_2last-2.48*theta_e+4.256*theta_e_last-1.807*theta_e_2last);

    //prevent windup by setting max and min values output can take.
    if(u>=1) u=1;
    if(u<=-1) u=-1;

    /**************************
    Perform checks on status of system, and reacts accordingly
    **************************/

    if(rc_get_state()==EXITING){
    rc_disable_motors();
    return;
    }

    //if disarmed, nothing needs to be done.
    if(armstate==DISARMED){
    return;
    }

    //if robot has tipped past point of no return, stop trying...
    if(abs(theta_f)>MAX_LEAN){
        printf("\rCould you help me up please?            \n");
        disarm_controller();
    }

    if(fabs(u)>0.95) inner_sat_timer++;
    else inner_sat_timer=0;

    if(inner_sat_timer > (INNERSAMPLERATE*D1_SATURATION_TIMEOUT)){
        printf("\rI'm winded, taking a rest.            \n");

        disarm_controller();
        inner_sat_timer = 0;
        return;
    }

    if(abs(theta_f)<=START_ANGLE_RANGE && rc_get_state()==RUNNING && armstate==DISARMED){
        arm_controller();
    }

    //proportional steering controller to keep MiP pointing in approximately the right direction!
    steeringinput=STEERING_GAIN*phi_diff;

    rc_set_motor(L_MOTOR_CHANNEL, L_MOTOR_POLARITY*(u-steeringinput));
    rc_set_motor(R_MOTOR_CHANNEL, R_MOTOR_POLARITY*(u+steeringinput));
}

void arm_controller(){

    rc_set_encoder_pos(R_MOTOR_CHANNEL,0);
    rc_set_encoder_pos(L_MOTOR_CHANNEL,0);

    //Zero out all previous values to reset controller
    phierror=0;
    phierror_last=0;

    theta_ref=0;
    theta_ref_last=0;

    u=0;
    u_last=0;
    u_2last=0;
```

```c

    theta_e_2last=0;
    theta_e_last=0;
    theta_e=0;

    rc_enable_motors();
    armstate=ARMED;
}

void disarm_controller(){
    rc_disable_motors();
    armstate=DISARMED;
}

void* setpoint_controller(){
    /*******OUTER LOOP CONTROLLER********/
    while(rc_get_state()!=EXITING){
        //Get raw wheel positions from encoders
        rightphi=(rc_get_encoder_pos(R_MOTOR_CHANNEL)*2*M_PI)/(GEARBOX*R_ENCODER_POLARITY);
        leftphi=(rc_get_encoder_pos(L_MOTOR_CHANNEL)*2*M_PI)/(GEARBOX*L_ENCODER_POLARITY);
        avgphi=(rightphi+leftphi)/2+theta_f;

        //find diff between wheel positions to use later in steering controller
        phi_diff=leftphi-rightphi;

        //calculate phi error for outer controller
        phierror_last=phierror;
        phierror=phi_setpoint-avgphi;

        theta_ref_last=theta_ref;

        theta_ref=D2_GAIN*(0.1785*phierror-0.1698*phierror_last+0.6014*theta_ref_last);

        rc_usleep(1000000/OUTERSAMPLERATE);
    }
    printf("setpoint_controller thread returning\n");
    return NULL;
}

/*******************************************************************************
* void* printdata()
*
* Print the angle estimates to the terminal.
*******************************************************************************/
void* printdata(){

    while(rc_get_state()!=EXITING){
            //Show current angle estimates combing from filtered  accelerometer,
            //gyro and combined estimate

            printf("\r");
            printf("%6.3f     %6.3f  %6.3f   %6.3f      %6.3f     %6.3f %d",\
            theta_e, u, theta_f,theta_ref,avgphi,phi_diff,armstate);

            fflush(stdout);
            rc_usleep(100000);
        }

        printf("printdata is returning\n");
        return NULL;
}

```