```c
/**************************************************************************
* File: balance.c
* Author: Parker Brown
* Date: 12/15/2017
* Course: MAE 144, Fall 2017
* Description: Balance program estimates MIP state, evaluates D1 and D2
* controllers, checks out of bounds conditions to disarm.
**************************************************************************/

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

// Controller arming enumerated type
typedef enum arm_state_t{
    ARMED,
    DISARMED
}arm_state_t;

// Struct for angles
typedef struct angles_t{
    float theta_a_raw[2];
    float theta_g_raw[2];
    float theta_a[2];
    float theta_g[2];
    float theta;
    float theta_error[3];
    float theta_ref;

    float phi;
    float phi_error[3];
    float phi_ref;
}angles_t;

// Struct for filters
typedef struct filter_t{
    float lp_num[2];
    float lp_den[2];
    float hp_num[2];
    float hp_den[2];
} filter_t;

// Struct for controller
typedef struct controllers_t{
    float d1_u[3];
    float d2_u[3];
    arm_state_t arm_state;
} controllers_t;

// Function declarations
void angle_mananger(); // MIP state estimation
float tfn(int order, float a[], float b[], float u[], float y[], float gain, float sat);
void d1_ctrl(); // D1 Controller
void inner_loop(); // IMU ISR func with arming checks and motor driving
void d2_ctrl(); // D2 Controller
void* outer_loop(void* ptr); // outer_thread func for D2 Controller
int arm_controller(); // Ser controller state to ARMED
int disarm_controller(); // Set contrller state to DISARMED
int start_condition(); // Start with upright condition
void zero_out(); // Zero out controllers and filters
void* printf_data(void* ptr); // printf_thread func to print data
void on_pause_pressed(); // do stuff when paused button is pressed
void on_pause_released(); // do stuff when paused button is released

// Global Variables
```

```
70    filter_t filter; // filter struct to hold filter coefficients
71    angles_t angles; // angles struct to hold theta angles
72    controllers_t ctrl; // d1, d2 controller struct
73    rc_imu_data_t data; // imu struct to hold new data
74
75    // Initialize Controller coefficients
76    float d1_num[] = D1_NUM;
77    float d1_den[] = D1_DEN;
78
79    /*****************************************************************************
80     * int main()
81     *
82     * balance main function contains these critical components
83     * - call to rc_initialize() at the beginning
84     * - UNINITIALIZED and DISARMED
85     * - Initialize DMP for interrupt
86     * - Start and schedule outer_thread
87     * - Start and schedule printf_thread
88     * - Initialize filters
89     *   - Set RUNNING and start IMU ISR
90     * - main while loop that checks for EXITING condition
91     *       - checks start condition that arms controllers
92     * - shutdown procedures
93     * - rc_cleanup() at the end
94     *****************************************************************************/
95    int main(){
96
97        // initialize hardware first
98        if(rc_initialize()){
99            fprintf(stderr,"ERROR: failed to initialize rc_initialize(), are you root?\n");
100           return -1;
101       }
102
103       // Set UNINITIALIZED while setting up
104       rc_set_led(RED,1);
105       rc_set_led(GREEN,0);
106       rc_set_state(UNINITIALIZED);
107
108       // make sure controller state starts DISARMED
109       ctrl.arm_state = DISARMED;
110
111       // Initialize imu
112       rc_imu_config_t conf = rc_default_imu_config(); // imu config to defaults
113       conf.dmp_sample_rate = INNER_RATE;
114       // Initialize imu for dmp interrupt operation
115       if(rc_initialize_imu_dmp(&data, conf)){
116           printf("rc_initialize_imu_failed\n");
117           return -1;
118       }
119
120       // initialize pause functions
121       rc_set_pause_pressed_func(&on_pause_pressed);
122       rc_set_pause_released_func(&on_pause_released);
123
124       // Check min/max sched_priority
125       printf("Valid priority range for SCHED_FIFO: %d - %d\n",
126                   sched_get_priority_min(SCHED_FIFO),
127                   sched_get_priority_max(SCHED_FIFO));
128
129       // Start printf_thread
130       pthread_t outer_thread;
131       struct sched_param outer_params;
132       outer_params.sched_priority = 60; // Reasonably low priority
133       pthread_create(&outer_thread, NULL, outer_loop, (void*) NULL);
134       pthread_setschedparam(outer_thread, SCHED_FIFO, &outer_params);
135
136       // Start printf_thread
137       pthread_t printf_thread;
138       struct sched_param printf_params;
```

```c
139         printf_params.sched_priority = 20; // Reasonably low priority
140         pthread_create(&printf_thread, NULL, printf_data, (void*) NULL);
141         pthread_setschedparam(printf_thread, SCHED_FIFO, &printf_params);
142
143         // Initialize Low Pass filter variables
144         filter.lp_num[0] = 0;
145         filter.lp_num[1] = WC*DT_INNER;
146         filter.lp_den[0] = 1;
147         filter.lp_den[1] = WC*DT_INNER-1;
148         // Initialize High Pass filter variables
149         filter.hp_num[0] = 1;
150         filter.hp_num[1] = -1;
151         filter.hp_den[0] = 1;
152         filter.hp_den[1] = WC*DT_INNER-1;
153
154         // done initializing so set state to RUNNING
155         rc_set_state(RUNNING);
156         rc_set_led(GREEN, ON);  // GREEN when running
157         rc_set_led(RED, OFF);   // RED when paused
158
159         rc_set_imu_interrupt_func(&inner_loop); // IMU ISR for D1 Controller
160
161     // Keep looping until state changes to EXITING
162         while(rc_get_state()!=EXITING){
163             rc_usleep(1000000 / 100); // 100hz
164
165             // nothing to do if paused, go back to beginning of loop
166             if(rc_get_state() != RUNNING) continue;
167
168             // Wait for start condition (upright) to pass, then arm controllers
169             if(ctrl.arm_state == DISARMED){
170                 if(start_condition()==0){
171                     zero_out();
172                     arm_controller();
173                 }
174                 else continue; // do nothing if start condition fails
175             }
176
177         }
178
179         // Shutdown procedures
180         printf("Joining printf_thread... ");
181         pthread_join(printf_thread, NULL);
182         printf("joined.\n");
183         printf("Joining outer_thread... ");
184         pthread_join(outer_thread, NULL);
185         printf("joined.\n");
186         disarm_controller(); // Disarm controller after closing all threads
187         rc_power_off_imu();
188
189         // exit cleanly
190         rc_cleanup();
191         return 0;
192 }
193
194 /***********************************************************************
195 * float tfn(int order, float a[], float b[], float u[], float y[], float gain, float sat)
196 *
197 * Computes difference equation for nth order transfer function.
198 * Input: denominator and numerator coefficients a[] and b[], old inputs uk's
199 * u[], old outputs yk's y[], tf order, tf gain, and saturation value.
200 * Output: float y_new from evaluation of tf.
201 ***********************************************************************/
202 float tfn(int order, float a[], float b[], float u[], float y[], float gain, float sat){
203     float y_new;
204     y[0] = 0;
205     // Assume a nd b are normalized by a[0]
206     // Compute y_new for numerator coefficients b[i]
207     int i;
```

```
208        for(i = 0; i <= order; i++){
209          y[0] += b[i] * u[i];
210        }
211        // Compute y_new for denominator coefficients a[j]
212          int j;
213        for(j = 1; j <= order; j++){
214          y[0] -= a[j] * y[j];
215        }
216
217          // scale by gain
218          y[0] = y[0] * gain;
219
220          // Saturate output
221          if (y[0] > sat){
222              y[0] = sat;
223          }   else if (y[0] < -sat){
224              y[0] = -sat;
225          }
226
227          // Update values
228          int k;
229          for(k = order; k > 0; k--){
230              u[k] = u[k-1];
231              y[k] = y[k-1];
232          }
233
234          y_new = y[0];
235        return y_new;
236    }
237
238    /***********************************************************************
239    * void angle_mananger()
240    *
241    * MIP state estimation, theta, phi, and their outputs calculated.
242    ***********************************************************************/
243    void angle_mananger(){
244          float wheelAngleL = 0;
245          float wheelAngleR = 0;
246
247          // Complimentary Filter start
248          angles.theta_a_raw[0] = -1.0 * atan2(data.accel[2], data.accel[1]); // theta [rad]
249          angles.theta_g_raw[0] = angles.theta_g_raw[0] \
250                                                  + (data.gyro[0] * DT_INNER *
                                                    DEG_TO_RAD); // theta [rad]
251
252          // Run high and low pass filter
253          tfn(1, filter.lp_den, filter.lp_num, angles.theta_a_raw, angles.theta_a, 1, 100);
254        tfn(1, filter.hp_den, filter.hp_num, angles.theta_g_raw, angles.theta_g, 1, 100);
255
256        // Sum of Low and High pass filters of theta
257          angles.theta = angles.theta_a[0] + angles.theta_g[0];
258          // Correct for BBBlue mount angle
259          angles.theta += CAPE_MOUNT_ANGLE;
260          // Complimentary Filter end
261
262          // Theta error for D1
263          angles.theta_error[0] = angles.theta_ref - angles.theta;
264
265          // Get phi [rad] for D2 Controller
266          wheelAngleL = ((rc_get_encoder_pos(ENCODER_CHANNEL_L) * TWO_PI) \
267                                    / (ENCODER_POLARITY_L * GEAR_RATIO * ENCODER_RES));
268          wheelAngleR = ((rc_get_encoder_pos(ENCODER_CHANNEL_R) * TWO_PI) \
269                                    / (ENCODER_POLARITY_R * GEAR_RATIO * ENCODER_RES));
270          angles.phi = ((wheelAngleL + wheelAngleR)/2) + angles.theta;
271          angles.phi_error[0] = angles.phi_ref - angles.phi;
272    }
273
274    /***********************************************************************
275    * void inner_loop()
```

```
276     *
277     * Inner (fast) loop run in interrupt service routine. Gets data, angles, errors.
278     * Checks tipping and loop saturation, disarms on failure. Runs D1 Controller.
279     * Drives motors if everyting passes.
280     *********************************************************************/
281    void inner_loop(){
282        static int sat_counter = 0;
283        float duty = 0;
284
285        /*************************************************************
286         * MIP state estimation: phi and theta angles
287         *************************************************************/
288        angle_mananger();
289
290        /*************************************************************
291         * check for various exit conditions AFTER state estimate
292         *************************************************************/
293        //DISARM if EXITING
294        if(rc_get_state() == EXITING){
295            rc_disable_motors();
296            return;
297        }
298        // DISARM if not RUNNING (i.e. PAUSED)
299        if((rc_get_state() != RUNNING) && (ctrl.arm_state == ARMED)){
300            disarm_controller();
301            return;
302        }
303        // Return out of loop if DISARMED
304        if(ctrl.arm_state == DISARMED){
305            return;
306        }
307        // DISARM if tip over detected
308        if(fabs(angles.theta) > TIP_ANGLE){
309            disarm_controller();
310            printf("tip detected \n");
311            return;
312        }
313
314        /*************************************************************
315         * Run inner loop if checks pass.
316         *************************************************************/
317        // Second order tf for D1 Controller
318        tfn(D1_ORDER, d1_den, d1_num, angles.theta_error, ctrl.d1_u, D1_GAIN, D1_SAT);
319
320        /*************************************************************
321         * Check if the inner loop saturated. If it saturates for over
322         * the timout, DISARM the controller.
323         *************************************************************/
324        if(fabs(ctrl.d1_u[0]) > 0.95) sat_counter++;
325        else sat_counter = 0;
326        // if saturate for a second, disarm for safety
327        if(sat_counter > (INNER_RATE * D1_SATURATION_TIMEOUT)){
328            printf("inner loop controller saturated\n");
329            disarm_controller();
330            sat_counter = 0;
331            return;
332        }
333
334        /*********************************************************
335         * Drive motors.
336         *********************************************************/
337        duty = ctrl.d1_u[0]; // Set duty cycle to write to motors
338        rc_set_motor(MOTOR_CHANNEL_L, MOTOR_POLARITY_L * duty);
339        rc_set_motor(MOTOR_CHANNEL_R, MOTOR_POLARITY_R * duty);
340
341        return;
342    }
343
344    /********************************************************************************
```

```c
345    * void outer_loop()
346    *
347    * Runs D2 controller in outer_loop thread.
348    *****************************************************************************/
349    void* outer_loop(void* ptr){
350        float d2_num[] = D2_NUM;
351        float d2_den[] = D2_DEN;
352        while(rc_get_state()!=EXITING){
353            // Just run D2 Controller and wait
354            // Second order tf for D2 Controller
355            tfn(D2_ORDER, d2_den, d2_num, angles.phi_error, ctrl.d2_u, D2_GAIN, D2_SAT);
356            angles.theta_ref = ctrl.d2_u[0]; // theta ref passed to inner controller
357            rc_usleep(1000000 / OUTER_RATE); // Sleep to set outer loop rate
358        }
359        return NULL;
360    }
361
362    /*****************************************************************************
363    * int disarm_controller()
364    *
365    * Disable motors and set arming state to DISARMED
366    *****************************************************************************/
367    int disarm_controller(){
368        rc_disable_motors();
369        ctrl.arm_state = DISARMED;
370        return 0;
371    }
372
373    /*****************************************************************************
374    * int arm_controller()
375    *
376    * Zero out the controllers and encoders. Enable motors and arm the controllers.
377    *****************************************************************************/
378    int arm_controller(){
379        zero_out();
380        rc_set_encoder_pos(ENCODER_CHANNEL_L,0);
381        rc_set_encoder_pos(ENCODER_CHANNEL_R,0);
382        ctrl.arm_state = ARMED;
383        rc_enable_motors();
384        return 0;
385    }
386
387    /*****************************************************************************
388    * int start_condition()
389    *
390    * Wait for MiP to be held upright long enough to initiate arming.
391    * Returns -1 on fail.
392    *****************************************************************************/
393    int start_condition(){
394        int count = 0;
395        const int count_hz = 20;     // check 20 times per second
396        int count_needed = round(START_DELAY*count_hz);
397        int wait_us = 1000000/count_hz;
398
399        // Wait for MIP to be tipped out of START_ANGLE range
400        while(rc_get_state() == RUNNING){
401            // if within range, start counting
402            if(fabs(angles.theta) > START_ANGLE) count++;
403            // fell out of range, restart counter
404            else count = 0;
405            // waited long enough, return
406            if(count >= count_needed) break;
407            rc_usleep(wait_us);
408        }
409        // Wait for MIP to be within START_ANGLE range
410        count = 0;
411        while(rc_get_state() == RUNNING){
412            // If within range, start counting
413            if(fabs(angles.theta) < START_ANGLE) count++;
```

```
414              // Else out of range and restart count
415              else count = 0;
416              // Return if waited long enough
417              if(count >= count_needed) return 0;
418              rc_usleep(wait_us);
419          }
420          return -1;
421      }
422
423      /****************************************************************************
424       * void zero_out()
425       *
426       * Zero out filter inputs nad integration values.
427       ****************************************************************************/
428      void zero_out(){
429          // Complimentary filter values
430          angles.theta_a_raw[0] = 0;
431          angles.theta_a_raw[1] = 0;
432          angles.theta_g_raw[0] = 0;
433          angles.theta_g_raw[1] = 0;
434          angles.theta_a[0] = 0;
435          angles.theta_a[1] = 0;
436          angles.theta_g[0] = 0;
437          angles.theta_g[1] = 0;
438          // D1 Controller feedback
439          angles.theta = 0;
440          // D1 Controller inputs
441          angles.theta_ref = 0;
442          angles.theta_error[0] = 0;
443          angles.theta_error[1] = 0;
444          angles.theta_error[2] = 0;
445          // D1 Controller outputs
446          ctrl.d1_u[0] = 0;
447          ctrl.d1_u[1] = 0;
448          ctrl.d1_u[2] = 0;
449          // D2 Controller feedback
450          angles.phi = 0;
451          // D2 Controller inputs
452          angles.phi_ref = 0;
453          angles.phi_error[0] = 0;
454          angles.phi_error[1] = 0;
455          angles.phi_error[2] = 0;
456          // D2 Controller outputs
457          ctrl.d2_u[0] = 0;
458          ctrl.d2_u[1] = 0;
459          ctrl.d2_u[2] = 0;
460      }
461
462      /****************************************************************************
463       * void* printf_data(void* ptr)
464       *
465       * printf_thread function prints data.
466       ****************************************************************************/
467      void* printf_data(void* ptr){
468          rc_state_t last_rc_state, new_rc_state; // keep track of last state
469          last_rc_state = rc_get_state();
470          while(rc_get_state()!=EXITING){
471              new_rc_state = rc_get_state();
472              // First time in RUNNING, print header
473              if((new_rc_state == RUNNING) && (last_rc_state != RUNNING)){
474                  printf("\nRUNNING: Hold upright to balance.\n|");
475                  printf("     θ     |");
476                  printf("  θ_ref  |");
477                  printf("     φ     |");
478                  printf("  φ_ref  |");
479                  printf("  d1_u   |");
480                  printf("  d2_u   |");
481                  printf(" theta_a |");
482                  printf(" theta_g |");
```

```
483                     printf("arm_state|");
484                     printf("\n");
485                 }
486             else if(new_rc_state==PAUSED && last_rc_state!=PAUSED){
487                     // First time being PAUSED, print pause statement
488                     printf("\nPAUSED: Press pause again to start.\n");
489             }
490             last_rc_state = new_rc_state; // update last_rc_state
491
492             // Print data while RUNNING
493             if(new_rc_state == RUNNING){
494                     // Print raw angles
495                     printf("\r|"); // carriage return because it looks pretty
496                     printf(" %7.3f |", angles.theta);
497                     printf(" %7.3f |", angles.theta_ref);
498                     printf(" %7.3f |", angles.phi);
499                     printf(" %7.3f |", angles.phi_ref);
500                     printf(" %7.3f |", ctrl.d1_u[0]);
501                     printf(" %7.3f |", ctrl.d2_u[0]);
502                     printf(" %7.3f |", angles.theta_a[0]);
503                     printf(" %7.3f |", angles.theta_g[0]);
504
505                     if(ctrl.arm_state == ARMED) printf("  ARMED  |");
506                     else printf("DISARMED |");
507                     fflush(stdout);
508             }
509
510             rc_usleep(1000000 / PRINTF_HZ); // Sleep to set print rate
511     }
512     return NULL;
513 }
514
515 /*******************************************************************************
516 * void on_pause_released()
517 *
518 * Make the Pause button toggle between paused and running states.
519 *******************************************************************************/
520 void on_pause_released(){
521     // toggle betewen paused and running modes
522     if(rc_get_state()==RUNNING){
523             rc_set_state(PAUSED);
524             disarm_controller(); // Always set DISARMED on PAUSE change
525             rc_set_led(GREEN, OFF); // GREEN when running
526             rc_set_led(RED, ON); // RED when paused
527     }
528     else if(rc_get_state()==PAUSED){
529             rc_set_state(RUNNING);
530             disarm_controller(); // Always set DISARMED on PAUSE change
531             rc_set_led(GREEN, ON); // GREEN when running
532             rc_set_led(RED, OFF); // RED when paused
533     }
534     return;
535 }
536
537 /*******************************************************************************
538 * void on_pause_pressed()
539 *
540 * If the user holds the pause button for 2 seconds, set state to exiting which
541 * triggers the rest of the program to exit cleanly.
542 *******************************************************************************/
543 void on_pause_pressed(){
544     int i=0;
545     const int samples = 100;    // check for release 100 times in this period
546     const int us_wait = 2000000; // 2 seconds
547
548     // now keep checking to see if the button is still held down
549     for(i=0;i<samples;i++){
550             rc_usleep(us_wait/samples);
551             if(rc_get_pause_button() == RELEASED) return;
```

```
552          }
553      printf("long press detected, shutting down\n");
554      rc_set_state(EXITING);
555      return;
556  }
557
```