Parker Brown
A11961591
MAE 144

Complimentary Filter Assignment

Note: Code is provided for Parts 2 and 3 together since Part 3 builds on Part 2. However, separate code is provided for Part 5 with significant changes in order to make use of threading and the IMU's ISR.

Part 1

What are the units of the gyroscope and accelerometer readings displayed with the rc_test_imu program?

- The gyroscope is in units of radians per second and the accelerometer is in units of meters per second squared.

Referencing the API documentation, what are the available full scale ranges of the gyroscope and accelerometer?

- The gyroscope has full scale ranges of 250 deg/s, 500 deg/s, 1000 deg/s, and 2000 deg/s. The accelerometer has full scale ranges of 2G, 4G, 8G, and 16G.

For the default full scale ranges used by test_imu, calculate the conversion rates from raw ADC to m/s2 and degrees/s for the accelerometer and gyroscope respectively.

- Since the imu has a 16 bit ADC built in, the conversion factor for the accel is accel_FSR * 9.80665 / 32768.0 to get acceleration in meters per second squared.

2G FSR: 0.00059855041

4G FSR: 0.00119710083

8G FSR: 0.00239420166

16G FSR: 0.00478840332

- Also, the gyro conversion factor is gyro_FSR /32768.0 to get angular rate in degrees per second

250DPS FSR: 0.00762939453

500DPS FSR: 0.01525878906

1000DPS FSR: 0.03051757812

2000DPS FSR: 0.06103515625

Parts 2-3

Note: Generic rc_template Makefile used with TARGET = hw2. Values of tau = 0.5 s and w_c = 2 rad/s were used to get a quicker settling time, leaning more heavily on the accelerometer data since the maneuver for this code was performed slowly and the accelerometer could respond quickly enough with little to no drift.

Code: hw2_config.h

```
/*******************************************************************************
* hw2_config.h
*
* Contains the settings for configuration of hw2.c
*******************************************************************************/

#ifndef HW2_CONFIG
#define HW2_CONFIG

// Set constants
#define		SAMPLE_RATE_HZ		100	// Loop rate
#define		DT		0.01	// 1/SAMPLE_RATE_HZ
#define	TAU	0.5	// Filter time constant
#define	WC	2	// 1/TAU

#endif	//HW2_CONFIG
```

Code: hw2.c

```
/*******************************************************************************
* File: hw2.c
* Author: Parker Brown
* Date: 12/3/2017
* Course: MAE 144, Fall 2017
* Description: Program calculates MIP body angle theta from raw accelerometer
* and gyroscope data and runs the raw angels through a complimentary filter.
* hw2.c uses rc_usleep for loop timing and does not use threading.
*******************************************************************************/

// 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 "hw2_config.h"

// Struct for angles
typedef struct angles_t{
	float theta_a_raw;
```

```
	float theta_g_raw;
	float last_theta_g_raw;
	float theta_a;
	float theta_g;
	float theta_f;
}angles_t;

// Struct for filters
typedef struct filter_t{
	float lp_coeff[2];
	float hp_coeff[3];
} filter_t;

// function declarations
void on_pause_pressed(); // do stuff when paused button is pressed
void on_pause_released(); // do stuff when paused button is released
void complimentary_filter(); // Complimentary filter
void zero_filers(); // Zero out filters

// Global Variables
filter_t filter;
angles_t angles;

/*******************************************************************************
* int main()
*
* hw1 main function contains these critical components
* - call to rc_initialize() at the beginning
* - Initialize filters
* - Initialize IMU
* - Print header for data
* - main while loop that checks for EXITING condition
*		- get raw data and convert to angles
*		- run raw accel and gyro angles through complimentary filter
*		- print filtered angle
* - rc_cleanup() at the end
*******************************************************************************/
int main(){

	// initialize hardware first
	if(rc_initialize()){
		fprintf(stderr,"ERROR: failed to initialize rc_initialize(), are you root?\n");
		return -1;
	}

	// initialize stuff here
	rc_set_pause_pressed_func(&on_pause_pressed);
	rc_set_pause_released_func(&on_pause_released);
```

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

	// Initialize variables used in the while loop
	int sleep_time = DT * 1e6; // Sleep time to set rough loop rate
	rc_imu_data_t data; // imu struct to hold new data
	rc_imu_config_t conf = rc_default_imu_config(); // config to defaults

	// Initialize filters
	zero_filers(); // Initialize angles to zero
	filter.lp_coeff[0] = -(WC*DT-1);
	filter.lp_coeff[1] =  WC*DT;
	filter.hp_coeff[0] = -(WC*DT-1);
	filter.hp_coeff[1] = 1;
	filter.hp_coeff[2] = -1;

	// Initialize imu
	if(rc_initialize_imu(&data, conf)){
		fprintf(stderr,"rc_initialize_imu_failed\n");
		return -1;
	}

	// Print header for standard output
	printf("Data Output\n");
	printf("| theta_a_raw |");
	printf(" theta_g_raw |");
	printf(" theta_a |");
	printf(" theta_g |");
	printf(" theta_f |");
	printf(" \n");

// Keep looping until state changes to EXITING
	while(rc_get_state()!=EXITING){
		// If RUNNING, run complimentary filter
		if(rc_get_state()==RUNNING){
			rc_set_led(GREEN, ON); // GREEN when on
			rc_set_led(RED, OFF); // RED when paused

			// Get accel data and convert to angle
			if(rc_read_accel_data(&data)<0){
				printf("read accel data failed\n");
			}
			angles.theta_a_raw = -1 * atan2(data.accel[2], data.accel[1]); // [rad]

			// Get gyro data and integrate to angle
			if(rc_read_gyro_data(&data)<0){
				printf("read gyro data failed\n");
```

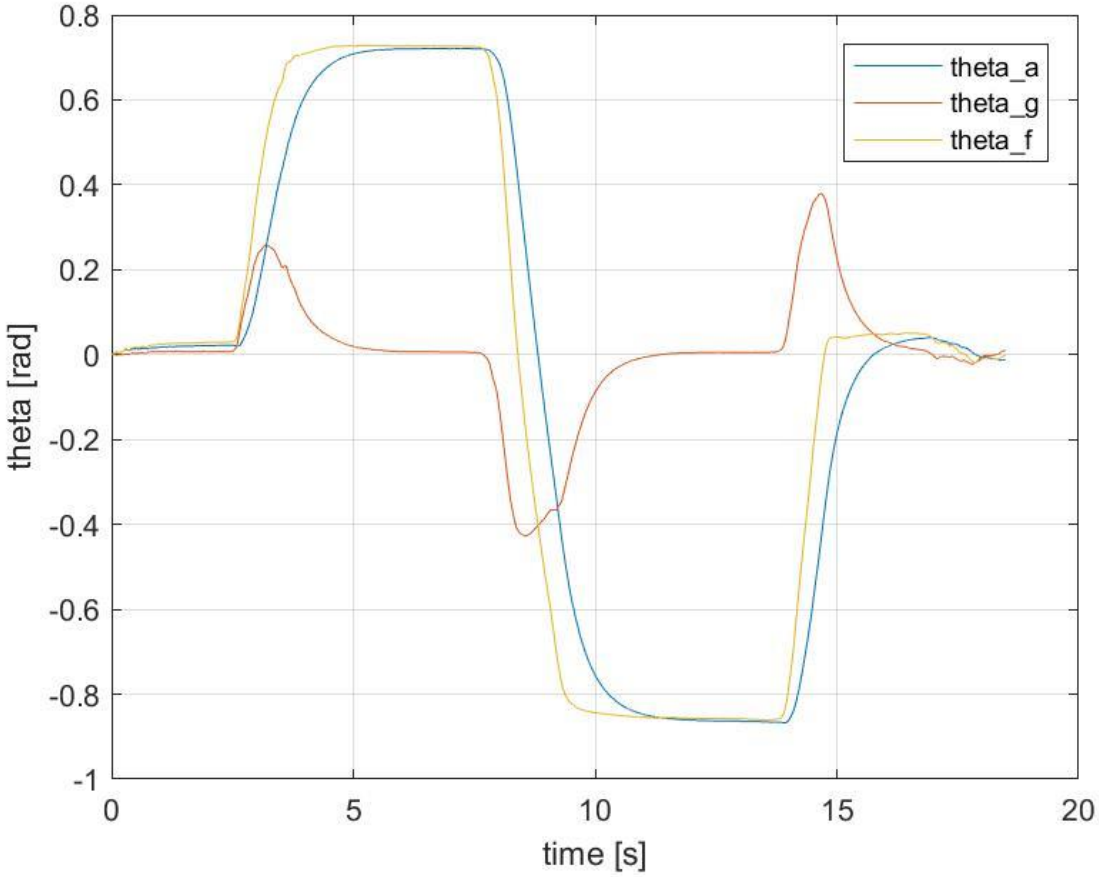
```
                        }
                        angles.theta_g_raw = angles.last_theta_g_raw \
                                        + (data.gyro[0] * DT * DEG_TO_RAD); // [rad]

                        complimentary_filter(); // Complimentary Filter

                        // Update integration value
                        angles.last_theta_g_raw = angles.theta_g_raw;

                        // Print raw angles
                        printf("\r|"); // carriage return because it looks pretty
                        printf(" %11.3f |", angles.theta_a_raw);
                        printf(" %11.3f |", angles.theta_g_raw);
                        printf(" %7.3f |", angles.theta_a);
                        printf(" %7.3f |", angles.theta_g);
                        printf(" %7.3f |", angles.theta_f);
                        fflush(stdout);

                }
                else if(rc_get_state()==PAUSED){
                        // Set everything to an off state when paused
                        rc_set_led(GREEN, OFF); // GREEN when on
                        rc_set_led(RED, ON); // RED when paused
                        zero_filers(); // Reset filters when paused
                }

                rc_usleep(sleep_time); // Sleep for DT in microseconds
        }

        // Shutdown procedures
        rc_power_off_imu();

        // exit cleanly
        rc_cleanup();
        return 0;
}

/*******************************************************************************
* void zero_filers()
*
* Zero out filter inputs nad integration values.
*******************************************************************************/
void zero_filers(){
        angles.last_theta_g_raw = 0; // Zero out for gyro integration
        angles.theta_a = 0;
        angles.theta_g = 0;
        angles.theta_f = 0;
}
```

```
/*******************************************************************************
* void complimentary_filter()
*
* Complimentary filter built by summing high and low pass fitlers applied to
* raw theta values from accel and gyro data, respectively.
*******************************************************************************/
void complimentary_filter(){
	// First order Low Pass filter of theta from raw accel data
	angles.theta_a = (filter.lp_coeff[0] * angles.theta_a) \
			+ (filter.lp_coeff[1] * angles.theta_a_raw);
 // First order high pass filter of theta from raw gyro data
	angles.theta_g = (filter.hp_coeff[0] * angles.theta_g) \
			+ (filter.hp_coeff[1] * angles.theta_g_raw) \
			+ (filter.hp_coeff[2] * angles.last_theta_g_raw);
 // Sum of Low and High pass filters of theta
	angles.theta_f = angles.theta_a + angles.theta_g;
}

/*******************************************************************************
* void on_pause_released()
*
* Make the Pause button toggle between paused and running states.
*******************************************************************************/
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);
	return;
}

/*******************************************************************************
* 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;
}
```

Part 4

Plot: Maneuver, Upright → pi/4 Forward → pi/4 backwards → Upright



Part 5

Note: Generic rc_template Makefile used with TARGET = hw2_isr

Code: hw2_isr_config.h

```
/*******************************************************************************
* hw2_isr_config.h
*
* Contains the settings for configuration of hw2_isr.c
*******************************************************************************/

#ifndef HW2_ISR_CONFIG
#define HW2_ISR_CONFIG

// Set constants
#define		SAMPLE_RATE_HZ		100	// Loop rate
#define		DT			0.01	// 1/SAMPLE_RATE_HZ
#define	TAU	0.5	// Filter time constant
#define	WC	2	// 1/TAU
#define	PRINTF_HZ	10	// printf_data rate

#endif	//HW2_ISR_CONFIG
```

Code: hw2_isr.c

```
/*******************************************************************************
* File: hw2_isr.c
* Author: Parker Brown
* Date: 12/3/2017
* Course: MAE 144, Fall 2017
* Description: Program calculates MIP body angle theta from raw accelerometer
* and gyroscope data and runs the raw angels through a complimentary filter.
* hw2_isr.c uses the IMU's interrupt service for loop timing and threads the
* print statements.
*******************************************************************************/

// 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 "hw2_isr_config.h"

// Struct for angles
typedef struct angles_t{
	float theta_a_raw;
	float theta_g_raw;
```

```
	float last_theta_g_raw;
	float theta_a;
	float theta_g;
	float theta_f;
}angles_t;

// Struct for filters
typedef struct filter_t{
	float lp_coeff[2];
	float hp_coeff[3];
} filter_t;

// function declarations
void on_pause_pressed(); // do stuff when paused button is pressed
void on_pause_released(); // do stuff when paused button is released
void complimentary_filter(); // Complimentary filter
void zero_filers(); // Zero out filters
void get_data(); // IMU interrupt routine
void* printf_data(void* ptr); // printf_thread function ot print data

// Global Variables
filter_t filter; // filter struct to hold filter coefficients
angles_t angles; // angle filter to hold new angle data
rc_imu_data_t data; // imu struct to hold new data

/*******************************************************************************
* int main()
*
* hw1 main function contains these critical components
* - call to rc_initialize() at the beginning
* - Initialize IMU
* - Initialize DMP for interrupt
* - Start and schedule printf_thread
* - Initialize filters
*	- Set RUNNING and start IMU isr
* - main while loop that checks for EXITING condition
*		- do nothing, just sleep
* - shutdown procedures
* - rc_cleanup() at the end
*******************************************************************************/
int main(){

	// initialize hardware first
	if(rc_initialize()){
		fprintf(stderr,"ERROR: failed to initialize rc_initialize(), are you root?\n");
		return -1;
	}
```

```
	// Initialize imu
	rc_imu_config_t conf = rc_default_imu_config(); // imu config to defaults
	if(rc_initialize_imu(&data, conf)){
		fprintf(stderr,"rc_initialize_imu_failed\n");
		return -1;
	}

	// Initialize imu for dmp interrupt operation
	if(rc_initialize_imu_dmp(&data, conf)){
		printf("rc_initialize_imu_failed\n");
		return -1;
	}

	// initialize pause functions
	rc_set_pause_pressed_func(&on_pause_pressed);
	rc_set_pause_released_func(&on_pause_released);

	// Check min/max sched_priority
	printf("Valid priority range for SCHED_FIFO: %d - %d\n",
		sched_get_priority_min(SCHED_FIFO),
		sched_get_priority_max(SCHED_FIFO));

	// Start printf_thread
	pthread_t printf_thread;
	struct sched_param params;
	params.sched_priority = 10; // Reasonably low priority
	pthread_create(&printf_thread, NULL, printf_data, (void*) NULL);
	pthread_setschedparam(printf_thread, SCHED_FIFO, &params);

	// Initialize filter variables
	zero_filers(); // Initialize angles to zero
	filter.lp_coeff[0] = -(WC*DT-1);
	filter.lp_coeff[1] =  WC*DT;
	filter.hp_coeff[0] = -(WC*DT-1);
	filter.hp_coeff[1] = 1;
	filter.hp_coeff[2] = -1;

	// done initializing so set state to RUNNING
	rc_set_state(RUNNING);
	rc_set_led(GREEN, ON);  // GREEN when running
	rc_set_led(RED, OFF);  // RED when paused

	rc_set_imu_interrupt_func(&get_data); // IMU isr to get data

// Keep looping until state changes to EXITING
	while(rc_get_state()!=EXITING){
		rc_usleep(100000); // Sleep ocassionally
	}
```

```
	// Shutdown procedures
	pthread_join(printf_thread, NULL);
	rc_power_off_imu();

	// exit cleanly
	rc_cleanup();
	return 0;
}

/*******************************************************************************
* void zero_filers()
*
* Zero out filter inputs nad integration values.
*******************************************************************************/
void zero_filers(){
	angles.last_theta_g_raw = 0; // Zero out for gyro integration
	angles.theta_a = 0;
	angles.theta_g = 0;
	angles.theta_f = 0;
}

/*******************************************************************************
* void complimentary_filter()
*
* Complimentary filter built by summing high and low pass fitlers applied to
* raw theta values from accel and gyro data, respectively.
*******************************************************************************/
void complimentary_filter(){
	// First order Low Pass filter of theta from raw accel data
	angles.theta_a = (filter.lp_coeff[0] * angles.theta_a) \
			+ (filter.lp_coeff[1] * angles.theta_a_raw);
 // First order high pass filter of theta from raw gyro data
	angles.theta_g = (filter.hp_coeff[0] * angles.theta_g) \
			+ (filter.hp_coeff[1] * angles.theta_g_raw) \
			+ (filter.hp_coeff[2] * angles.last_theta_g_raw);
 // Sum of Low and High pass filters of theta
	angles.theta_f = angles.theta_a + angles.theta_g;
}

/*******************************************************************************
* void get_data()
*
* Gets imu data using rc_set_imu_interrupt_func(&get_data)
*******************************************************************************/
void get_data(){
	// If RUNNING, run Complimentary Filter
	if(rc_get_state()==RUNNING){
```

```
                // Get accel data and convert to angle
                if(rc_read_accel_data(&data)<0){
                        printf("Read accel data failed.\n");
                }
                angles.theta_a_raw = -1 * atan2(data.accel[2], data.accel[1]); // theta [rad]

                // Get gyro data and integrate to angle
                if(rc_read_gyro_data(&data)<0){
                        printf("Read gyro data failed.\n");
                }
                angles.theta_g_raw = angles.last_theta_g_raw \
                                + (data.gyro[0] * DT * DEG_TO_RAD); // theta [rad]

                complimentary_filter(); // Complimentary Filter

                // Update integration value
                angles.last_theta_g_raw = angles.theta_g_raw;
        }
        else if(rc_get_state()==PAUSED){
                zero_filers(); // Reset filters when paused
        }
}

/*******************************************************************************
* void* printf_data(void* ptr)
*
* printf_thread function prints data.
*******************************************************************************/
void* printf_data(void* ptr){
        rc_state_t last_rc_state, new_rc_state; // keep track of last state
        last_rc_state = rc_get_state();
        while(rc_get_state()!=EXITING){
                new_rc_state = rc_get_state();
                // check if this is the first time since being paused
                if(new_rc_state==RUNNING && last_rc_state!=RUNNING){
                        printf("\nRUNNING: Complimentary Filter, theta in [rad].\n");
                        // Print header for standard output
                        printf("| theta_a_raw |");
                        printf(" theta_g_raw |");
                        printf(" theta_a |");
                        printf(" theta_g |");
                        printf(" theta_f |");
                        printf(" \n");
                }
                else if(new_rc_state==PAUSED && last_rc_state!=PAUSED){
                        // first time sonce being paused
                        printf("\nPAUSED: Press pause again to start.\n");
                }
```

```
                last_rc_state = new_rc_state; // update last_rc_state

                if(new_rc_state == RUNNING){
                        // Print raw angles
                        printf("\r|"); // carriage return because it looks pretty
                        printf(" %11.3f |", angles.theta_a_raw);
                        printf(" %11.3f |", angles.theta_g_raw);
                        printf(" %7.3f |", angles.theta_a);
                        printf(" %7.3f |", angles.theta_g);
                        printf(" %7.3f |", angles.theta_f);
                        fflush(stdout);
                }

                rc_usleep(1000000 / PRINTF_HZ); // Sleep to set 10HZ print rate
        }
        return NULL;
}

/*******************************************************************************
* void on_pause_released()
*
* Make the Pause button toggle between paused and running states.
*******************************************************************************/
void on_pause_released(){
        // toggle betewen paused and running modes
        if(rc_get_state()==RUNNING){
                rc_set_state(PAUSED);
                rc_set_led(GREEN, OFF); // GREEN when running
                rc_set_led(RED, ON); // RED when paused
        }
        else if(rc_get_state()==PAUSED){
                rc_set_state(RUNNING);
                rc_set_led(GREEN, ON); // GREEN when running
                rc_set_led(RED, OFF); // RED when paused
        }
        return;
}

/*******************************************************************************
* 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;
}
```