**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 wc = 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

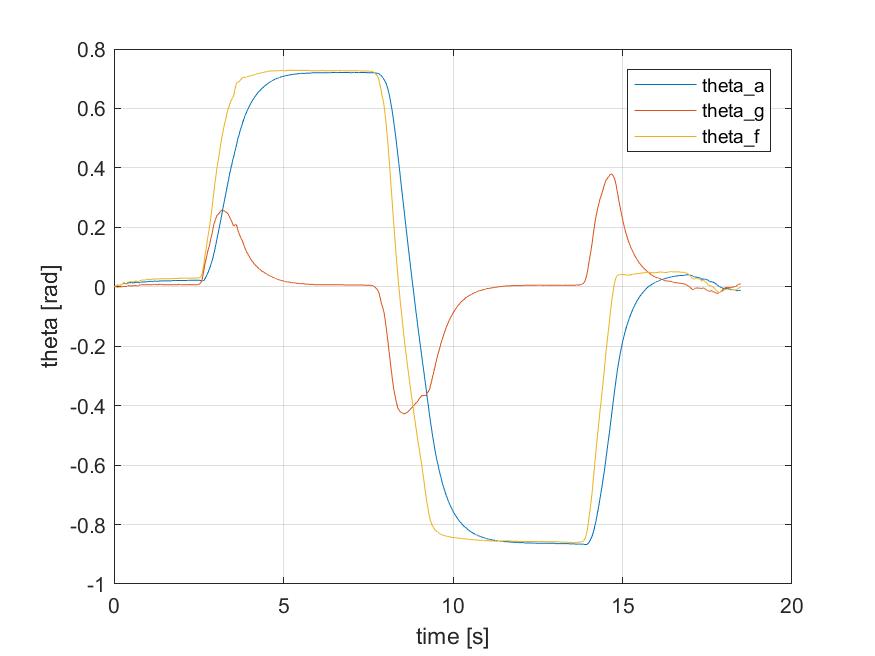
|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* hw2\_config.h  \*  \* Contains the settings for configuration of hw2.c  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #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;  } |