```
/*
 * File: PSET5_Question 2
 * --------------
 * Erick Blankenberg, Beck Goodloe
 * ME327
 * 5/7/2019
 * PSET 5 Question 2
 *
 * Description:
 *   Renders teleoperated system.
 */

// Includes
#include <math.h>
#include <Wire.h>

//#define POSITIONSCALING
//#define FORCESCALING

// System Parameters
float positionProportion   = 25;
float derivativeProportion = 0.1;

// Pin declares
// > System 1
int pwmPin_1       = 5;  // PWM output pin for motor 1
int dirPin_1       = 8;  // direction output pin for motor 1
int sensorPosPin_1 = A2; // input pin for MR sensor
int fsrPin_1       = A3; // input pin for FSR sensor
// > System 2
int pwmPin_2       = 6;  // PWM output pin for motor 2
int dirPin_2       = 7;  // direction output pin for motor 2
int sensorPosPin_2 = A4; // input pin for MR sensor on other system

// Position tracking variables
// > System 1
int updatedPos_1       = 0; // keeps track of the latest updated value of the MR
sensor reading
int rawPos_1           = 0; // current raw reading from MR sensor
int lastRawPos_1       = 0; // last raw reading from MR sensor
int lastLastRawPos_1   = 0; // last last raw reading from MR sensor
int flipNumber_1       = 0; // keeps track of the number of flips over the 180deg
mark
int tempOffset_1       = 0;
int rawDiff_1          = 0;
```

```
int lastRawDiff_1      = 0;
int rawOffset_1        = 0;
int lastRawOffset_1    = 0;
const int flipThresh_1 = 700;  // threshold to determine whether or not a flip
over the 180 degree mark occurred
boolean flipped_1      = false;
// > System 2
int updatedPos_2       = 0; // keeps track of the latest updated value of the MR
sensor reading
int rawPos_2           = 0; // current raw reading from MR sensor
int lastRawPos_2       = 0; // last raw reading from MR sensor
int lastLastRawPos_2   = 0; // last last raw reading from MR sensor
int flipNumber_2       = 0; // keeps track of the number of flips over the 180deg
mark
int tempOffset_2       = 0;
int rawDiff_2          = 0;
int lastRawDiff_2      = 0;
int rawOffset_2        = 0;
int lastRawOffset_2    = 0;
const int flipThresh_2 = 700;  // threshold to determine whether or not a flip
over the 180 degree mark occurred
boolean flipped_2      = false;

// Kinematics variables
// > System 1
double xh_1      = 0;      // position of the handle [m]
double theta_s_1 = 0;      // Angle of the sector pulley in deg
double xh_prev_1;          // Distance of the handle at previous time step
double xh_prev2_1;
double dxh_1;              // Velocity of the handle
double dxh_prev_1;
double dxh_prev2_1;
double dxh_filt_1;         // Filtered velocity of the handle
double dxh_filt_prev_1;
double dxh_filt_prev2_1;
// > System 2
double xh_2      = 0;      // position of the handle [m]
double theta_s_2 = 0;      // Angle of the sector pulley in deg
double xh_prev_2;          // Distance of the handle at previous time step
double xh_prev2_2;
double dxh_2;              // Velocity of the handle
double dxh_prev_2;
double dxh_prev2_2;
double dxh_filt_2;         // Filtered velocity of the handle
double dxh_filt_prev_2;
```

```
double dxh_filt_prev2_2;

// Force output variables
// > System 1
double force_1         = 0;    // force at the handle
double Tp_1            = 0;    // torque of the motor pulley
double duty_1          = 0;    // duty cylce (between 0 and 255)
unsigned int output_1 = 0;    // output command to the motor
// > System 2
double force_2         = 0;    // force at the handle
double Tp_2            = 0;    // torque of the motor pulley
double duty_2          = 0;    // duty cylce (between 0 and 255)
unsigned int output_2 = 0;    // output command to the motor

// I added a sign function
#define signum(x) ((x > 0)?(1):(-1))

// --------------------------------------------------------------
// Setup function -- NO NEED TO EDIT
// --------------------------------------------------------------
void setup() {
  // Set up serial communication
  Serial.begin(115200);

  // Set PWM frequency
  setPwmFrequency(pwmPin_1, 1);
  setPwmFrequency(pwmPin_2, 1);

  // Input pins
  pinMode(sensorPosPin_1, INPUT); // set MR sensor pin to be an input
  pinMode(fsrPin_1, INPUT);       // set FSR sensor pin to be an input
  pinMode(sensorPosPin_2, INPUT); // set MR sensor pin to be an input

  // Output pins
  pinMode(pwmPin_1, OUTPUT);  // PWM pin for motor A
  pinMode(dirPin_1, OUTPUT);  // dir pin for motor A
  pinMode(pwmPin_2, OUTPUT);  // PWM pin for motor A
  pinMode(dirPin_2, OUTPUT);  // dir pin for motor A

  // Initialize motor
  analogWrite(pwmPin_1, 0);     // set to not be spinning (0/255)
  digitalWrite(dirPin_1, LOW);  // set direction
  analogWrite(pwmPin_2, 0);     // set to not be spinning (0/255)
  digitalWrite(dirPin_2, LOW);  // set direction
```

```
  // Initialize position valiables
  lastLastRawPos_1 = analogRead(sensorPosPin_1);
  lastRawPos_1     = analogRead(sensorPosPin_1);
  lastLastRawPos_2 = analogRead(sensorPosPin_1);
  lastRawPos_2     = analogRead(sensorPosPin_1);
}


// --------------------------------------------------------------
// Main Loop
// --------------------------------------------------------------
void loop()
{

  //***********************************************************
  //*** Section 1. Compute position in counts (do not change) ***
  //***********************************************************

  // > Device 1
  // Get voltage output by MR sensor
  rawPos_1 = analogRead(sensorPosPin_1);  //current raw position from MR sensor
  // Calculate differences between subsequent MR sensor readings
  rawDiff_1 = rawPos_1 - lastRawPos_1;          //difference btwn current raw
position and last raw position
  lastRawDiff_1 = rawPos_1 - lastLastRawPos_1;  //difference btwn current raw
position and last last raw position
  rawOffset_1 = abs(rawDiff_1);
  lastRawOffset_1 = abs(lastRawDiff_1);
  // Update position record-keeping vairables
  lastLastRawPos_1 = lastRawPos_1;
  lastRawPos_1 = rawPos_1;
  // Keep track of flips over 180 degrees
  if((lastRawOffset_1 > flipThresh_1) && (!flipped_1)) { // enter this anytime
the last offset is greater than the flip threshold AND it has not just flipped
    if(lastRawDiff_1 > 0) {        // check to see which direction the drive
wheel was turning
      flipNumber_1--;              // cw rotation
    } else {                     // if(rawDiff < 0)
      flipNumber_1++;              // ccw rotation
    }
    if(rawOffset_1 > flipThresh_1) { // check to see if the data was good and the
most current offset is above the threshold
      updatedPos_1 = rawPos_1 + flipNumber_1*rawOffset_1; // update the pos value
to account for flips over 180deg using the most current offset
      tempOffset_1 = rawOffset_1;
```

```
    } else {                          // in this case there was a blip in the data and
we want to use lastactualOffset instead
      updatedPos_1 = rawPos_1 + flipNumber_1*lastRawOffset_1;  // update the pos
value to account for any flips over 180deg using the LAST offset
      tempOffset_1 = lastRawOffset_1;
    }
    flipped_1 = true;                 // set boolean so that the next time through the
loop won't trigger a flip
  } else {                            // anytime no flip has occurred
    updatedPos_1 = rawPos_1 + flipNumber_1*tempOffset_1; // need to update pos
based on what most recent offset is
    flipped_1 = false;
  }

  // > Device 2
  // Get voltage output by MR sensor
  rawPos_2 = analogRead(sensorPosPin_2);  //current raw position from MR sensor
  // Calculate differences between subsequent MR sensor readings
  rawDiff_2 = rawPos_2 - lastRawPos_2;          //difference btwn current raw
position and last raw position
  lastRawDiff_2 = rawPos_2 - lastLastRawPos_2;  //difference btwn current raw
position and last last raw position
  rawOffset_2 = abs(rawDiff_2);
  lastRawOffset_2 = abs(lastRawDiff_2);
  // Update position record-keeping vairables
  lastLastRawPos_2 = lastRawPos_2;
  lastRawPos_2 = rawPos_2;
  // Keep track of flips over 180 degrees
  if((lastRawOffset_2 > flipThresh_2) && (!flipped_2)) { // enter this anytime
the last offset is greater than the flip threshold AND it has not just flipped
    if(lastRawDiff_2 > 0) {        // check to see which direction the drive
wheel was turning
      flipNumber_2--;              // cw rotation
    } else {                      // if(rawDiff < 0)
      flipNumber_2++;              // ccw rotation
    }
    if(rawOffset_2 > flipThresh_2) { // check to see if the data was good and the
most current offset is above the threshold
      updatedPos_2 = rawPos_2 + flipNumber_2*rawOffset_2; // update the pos value
to account for flips over 180deg using the most current offset
      tempOffset_2 = rawOffset_2;
    } else {                          // in this case there was a blip in the data and
we want to use lastactualOffset instead
      updatedPos_2 = rawPos_2 + flipNumber_2*lastRawOffset_2;  // update the pos
value to account for any flips over 180deg using the LAST offset
```

```
      tempOffset_2 = lastRawOffset_2;
    }
    flipped_2 = true;             // set boolean so that the next time through the
loop won't trigger a flip
  } else {                         // anytime no flip has occurred
    updatedPos_2 = rawPos_2 + flipNumber_2*tempOffset_2; // need to update pos
based on what most recent offset is
    flipped_2 = false;
  }

  //*************************************************************
  //*** Section 2. Compute position in meters *******************
  //*************************************************************

  // > System 1
  // Compute the angle of the sector pulley (ts) in degrees based on updatedPos
  double ts_1 = 0.0124 * updatedPos_1 - 9.3517;
  // Compute the position of the handle (in meters) based on ts (in radians)
  xh_1 = 0.07 * ((ts_1 * 2 * PI) / (double) 360);
  // Calculate velocity with loop time estimation
  dxh_1 = (double)(xh_1 - xh_prev_1) / 0.001;
  // Calculate the filtered velocity of the handle using an infinite impulse
response filter
  dxh_filt_1 = .9*dxh_1 + 0.1*dxh_prev_1;
  // Record the position and velocity
  xh_prev2_1 = xh_prev_1;
  xh_prev_1 = xh_1;
  dxh_prev2_1 = dxh_prev_1;
  dxh_prev_1 = dxh_1;
  dxh_filt_prev2_1 = dxh_filt_prev_1;
  dxh_filt_prev_1 = dxh_filt_1;

  // > System 2
  // Compute the angle of the sector pulley (ts) in degrees based on updatedPos
  double ts_2 = 0.0124 * updatedPos_2 - 9.3517;
  ts_2 = -ts_2; // Readings from second device are inverted.
  // Compute the position of the handle (in meters) based on ts (in radians)
  xh_2 = 0.07 * ((ts_2 * 2 * PI) / (double) 360);
  // Calculate velocity with loop time estimation
  dxh_2 = (double)(xh_2 - xh_prev_2) / 0.001;
  // Calculate the filtered velocity of the handle using an infinite impulse
response filter
  dxh_filt_2 = .9*dxh_2 + 0.1*dxh_prev_2;
  // Record the position and velocity
  xh_prev2_2 = xh_prev_2;
```

```
xh_prev_2 = xh_2;
dxh_prev2_2 = dxh_prev_2;
dxh_prev_2 = dxh_2;
dxh_filt_prev2_2 = dxh_filt_prev_2;
dxh_filt_prev_2 = dxh_filt_2;

//*************************************************************
//*** Section 3. Assign a motor output force in Newtons *******
//*************************************************************

Serial.print(xh_1, 4);
Serial.print(" ");
Serial.println(xh_2, 4);

#ifdef POSITIONSCALING
xh_2 = 0.1 * xh_2; // Basically just scales down what controller sees beforehand
#endif

float proportionalForce = positionProportion * (xh_1 - xh_2);
float derivativeForce   = derivativeProportion * (dxh_filt_1  - dxh_filt_2);

force_1 = proportionalForce + derivativeForce;
Tp_1 = ((0.07 * 0.0075)/(0.075)) * force_1;
force_2 = -force_1;
Tp_2 = -Tp_1;

#ifdef FORCESCALING
// Literarily just scales up results from calculation
Tp_2 = 10 * Tp_2;
force_2 = 10 * force_2;
#endif

//*************************************************************
//*** Section 4. Force output (do not change) *****************
//*************************************************************

// > System 1
// Determine correct direction for motor torque
if(force_1 > 0) {
  digitalWrite(dirPin_1, HIGH);
} else {
  digitalWrite(dirPin_1, LOW);
}
// Compute the duty cycle required to generate Tp (torque at the motor pulley)
duty_1 = sqrt(abs(Tp_1)/0.03);
```

```
  // Make sure the duty cycle is between 0 and 100%
  if (duty_1 > 1) {
    duty_1 = 1;
  } else if (duty_1 < 0) {
    duty_1 = 0;
  }
  output_1 = (int)(duty_1 * 255);   // convert duty cycle to output signal
  analogWrite(pwmPin_1, output_1);  // output the signal

  // > System 2
  // Determine correct direction for motor torque
  if(force_2 > 0) {
    digitalWrite(dirPin_2, HIGH);
  } else {
    digitalWrite(dirPin_2, LOW);
  }
  // Compute the duty cycle required to generate Tp (torque at the motor pulley)
  duty_2 = sqrt(abs(Tp_2)/0.03);
  // Make sure the duty cycle is between 0 and 100%
  if (duty_2 > 1) {
    duty_2 = 1;
  } else if (duty_2 < 0) {
    duty_2 = 0;
  }
  output_2 = (int)(duty_2 * 255);   // convert duty cycle to output signal
  analogWrite(pwmPin_2, output_2);  // output the signal
}

// --------------------------------------------------------------
// Function to set PWM Freq -- DO NOT EDIT
// --------------------------------------------------------------
void setPwmFrequency(int pin, int divisor) {
  byte mode;
  if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 64: mode = 0x03; break;
      case 256: mode = 0x04; break;
      case 1024: mode = 0x05; break;
      default: return;
    }
    if(pin == 5 || pin == 6) {
      TCCR0B = TCCR0B & 0b11111000 | mode;
    } else {
```

```
      TCCR1B = TCCR1B & 0b11111000 | mode;
    }
  } else if(pin == 3 || pin == 11) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 32: mode = 0x03; break;
      case 64: mode = 0x04; break;
      case 128: mode = 0x05; break;
      case 256: mode = 0x06; break;
      case 1024: mode = 0x7; break;
      default: return;
    }
    TCCR2B = TCCR2B & 0b11111000 | mode;
  }
}
```