// Gaby, Andy, and Ruqayya's Code!
// 11.21.13
// Code to create Teleoperated Device
// Includes
#include <math.h>

// Pin declares
int pwmAPin = 5; // to send PWM signal to motor A
int pwmBPin = 6; // to send PWM signal to motor B
int dirAPin = 8; // to send direction signal to motor A
int dirBPin = 7; // to send direction signal to motor B
int sensorPosPin = A2; // to read the MR sensor values
int sensorPosPin2 = A4; // to read the MR sensor values from dumb hapkit

// Position tracking variables for master hapkit
int updatedPos = 0; // keeps track of the latest updated value of the MR sensor
int rawPos = 0; // current raw reading from MR sensor
int lastRawPos = 0; // last raw reading from MR sensor
int lastLastRawPos = 0; // last last raw reading from MR sensor
int flipNumber = 0; // keeps track of the number of flips over the 180deg mark
int tempOffset = 0;
int rawDiff = 0;
int lastRawDiff = 0;
int rawOffset = 0;
int lastRawOffset = 0;
const int flipThresh = 700; // threshold to determine whether or not a flip occurred
boolean flipped = false;

// Position tracking variables for dumb hapkit
int updatedPos2 = 0; // keeps track of the latest updated value of the MR sensor
int rawPos2 = 0; // current raw reading from MR sensor
int lastRawPos2 = 0; // last raw reading from MR sensor
int lastLastRawPos2 = 0; // last last raw reading from MR sensor
int flipNumber2 = 0; // keeps track of the number of flips over the 180deg mark
int tempOffset2 = 0;
int rawDiff2 = 0;
int lastRawDiff2 = 0;
int rawOffset2 = 0;
int lastRawOffset2 = 0;
const int flipThresh2 = 700; // threshold to determine whether or not a flip occurred
boolean flipped2 = false;

// Kinematics variables for master hapkit
double x = 0; // position of the handle [m]
double lastx = 0; // last x position of the handle
double vel    =  0; //velocity of the handle
double lastVel =  0; //last velocity of the handle
double lastLastVel =  0; //last last velocity of the handle

// Kinematics variables for dumb hapkit
double   x2    =  0; //position of the handle [m]
double  lastx2  =  0; //last x position of the handle
double   vel2   =  0; //velocity of the handle
double  lastVel2 =  0; //velocity of the handle
double lastLastVel2 =  0; //last last velocity of the handle

// Force output variables for master hapkit
double force    =  0; //force at the handle
double Tm       =  0; //torque of the motor
double duty     =  0; //duty cycle (between 0 and 255)
unsigned int output =  0; //output command to the motor

// Force output variables for dumb hapkit
double force2   =  0; //force at the handle
double Tm2      =  0; //torque of the motor
double duty2    =  0; //duty cycle (between 0 and 255)
unsigned int output2 =  0; //output command to the motor

//-----------------------------------
//Initialize
//-----------------------------------
void setup()
{
    // Set Up Serial
    Serial.begin(9600);

    // Set PWM frequency
    setPwmFrequency(pwmAPin, 1);

    // Set PWM frequency dumb hapkit
    setPwmFrequency(pwmBPin, 1);

    // Input Pins
    pinMode(sensorPosPin, INPUT); // set MR sensor pin to be an input
    pinMode(sensorPosPin2, INPUT); // set MR sensor pin for dumb hapkit to be an input

    // Output Pins
    pinMode(pwmAPin, OUTPUT); // PWM pin for motor A
    pinMode(dirAPin, OUTPUT); // dir pin for motor A

    // Output Pins dumb hapkit
}
void loop() {

    // Get voltage output by MR sensor
    rawPos = analogRead(sensorPosPin); //current raw position from MR sensor
    //calculate differences between MR sensor readings
    rawDiff = rawPos - lastRawPos; //difference btwn current raw position and last
    lastRawDiff = rawPos - lastLastRawPos; //difference btwn current raw position and last
    rawOffset = abs(rawDiff);
    lastRawOffset = abs(lastRawDiff);

    // Update position record-keeping variables
    lastLastRawPos = lastRawPos;
    lastRawPos = rawPos;

    // Keep track of flips over 180 degrees
    if ((lastRawOffset > flipThresh) && (!flipped)) // enter this anytime the last
    {
        if (lastRawDiff > 0) // check to see which direction the drive wheel was
        {
            flipNumber--; // cw rotation
        }
        else // if (rawDiff < 0)
        {
    

}
flipNumber++; // ccw rotation

if (rawOffset > flipThresh) // check to see if the data was good and the most
{
    updatedPos = rawPos + flipNumber * rawOffset; // update the pos value to account
    tempOffset = rawOffset;
} else 
    // in this case there was a blip in the data and update
    { // the pos value to account
        updatedPos = rawPos + flipNumber * lastRawOffset; // update the pos value to account
        tempOffset = lastRawOffset;
    }
    flipped = true; // set boolean so that the next time through the loop
} else 
    // anytime no flip has occurred
    {
        updatedPos = rawPos + flipNumber * tempOffset; // need to update pos based on
        flipped = false;
    }

    // Get voltage output by MR sensor
    rawPos2 = analogRead(sensorPosPin2); // current raw position from MR sensor
    // calculate differences between MR sensor readings
    rawDiff2 = rawPos2 - lastRawPos2; // difference between current raw position
    lastRawDiff2 = rawPos2 - lastLastRawPos2; // difference between current raw position
    rawOffset2 = abs(rawDiff2);
    lastRawOffset2 = abs(lastRawDiff2);

    // Update position record-keeping variable
    lastLastRawPos2 = lastRawPos2;
    lastRawPos2 = rawPos2;

    // Keep track of flips over 180 degrees
    if ((lastRawOffset2 > flipThresh2) && (!flipped2)) // enter this anytime the last
    { // move
        if (lastRawDiff2 > 0) // check to see which direction the drive wheel
            {
                flipNumber2--; // cw rotation
            }
        else // if (rawDiff < 0)
            {
                flipNumber2++; // ccw rotation
            }
    }
    if (rawOffset2 > flipThresh2) // check to see if the data was good and the most
    { // move
        updatedPos2 = rawPos2 + flipNumber2 * rawOffset2; // update the pos value to account
    }
tempOffset2 = rawOffset2;
}
else // in this case there was a blip in the data and we
{
  updatedPos2 = rawPos2 + flipNumber2*lastRawOffset2; // update the pos value
  tempOffset2 = lastRawOffset2;
}
flipped2 = true; // set boolean so that the next time through the
else // anytime no flip has occurred
{
  updatedPos2 = rawPos2 + flipNumber2*tempOffset2; // need to update pos based
  flipped2 = false;
}

//********************END CODING VIRTUAL ENVIRONMENTS HERE*******************
//***********************************************************************

//NOTE: Some of the variables you will need have already been
//created for you and are indicated below with quotes ("variableName").
//All of these variables are declared at the top of the program. You
//will need to create the rest of the variables on your own. Suggested
//variable names are in parentheses.*/

//--------Enter parameter values to calculate kinematics--------
//parameter names from slides: theta sector (ts), r handle (rh), r pulley (rh),

//--------Enter MR sensor calibration code------------------------
//hint: based on your calibration you should have a linear fit
//that allows you to calculate the sector angle (in degrees, then
//convert to radians) based on the updatedPos you found above
double posrad = .0002*updatedPos + 0.1826;
double posrad2 = -.0002*updatedPos2 + 0.0983;
Serial.println(posrad);

//---Enter calculations to find "x" and "vel" of handle---------
x = posrad*.077;
x2 = posrad2*.077;

//Serial.print(x, 4);
//Serial.print(" ");
//Serial.println(x2, 4);
//Serial.println(x);
---Enter parameters for virtual environments-------------------
//such as spring constant (k)
double k=200;
//force=0.5;
//force2=0.5;
double b =-0.01 ;

//to set virtual damper to smoothe hapkit movement
//force= (-b*vel);
vel = -(.95*.95)*lastLastVel + 2*.95*lastVel + (1-.95)*(x-lastx)/.0001;

//force2= (-b*vel2);
vel2 = -(.95*.95)*lastLastVel2 + 2*.95*lastVel2 + (1-.95)*(x2-lastx2)/.0001;

force2=k*(x+x2)+b*(vel2+vel);
force=-force2;

//Serial.println(force);

//---Enter code to determine "force" for virtual environment------

//****************************************************************************
******STOP CODING VIRTUAL ENVIRONMENTS HERE**************
****************************************************************************

//--------Forceoutput----------
//Determine correct direction
if(force > 0) // [N]
{
    digitalWrite(dirAPin, HIGH);
} else
{
    digitalWrite(dirAPin, LOW);
}

//--------Force output 2 dumb hapkit----------
//Determine correct direction
if(force2 > 0) // [N]
{
    digitalWrite(dirBPin, HIGH);
} else
{
    digitalWrite(dirBPin, LOW);
}
/*******************************************************

---Enter Motor calibration code----
//(given the desired "force", calculate the
//corresponding "Tm" (torque of the motor)
//and "duty" (duty cycle of the motor)
double rPulley=.004;
double rSector=.072;
double rHandle=.077;
duty=sqrt(abs(force)*rHandle*rPulley/(rSector*0.03));
duty2=sqrt(abs(force2)*rHandle*rPulley/(rSector*0.03));
//dumb hapkit
//Serial.println(duty);

if (duty > .80)
{
    duty = .80; // output a maximum of 80% duty cycle
}
if (duty < 0)
{
    duty = 0;
}
output = (int)(duty*255); // convert duty cycle to output signal
analogWrite(pwmAPin,output); // output the signal

//dumb hapkit
if (duty2 > .80)
{
    duty2 = .80; // output a maximum of 80% duty cycle
}
if (duty2 < 0)
{
    duty2 = 0;
}
output2 = (int)(duty2*255); // convert duty cycle to output signal
analogWrite(pwmBPin,output2); // output the signal

//------Update variables------
lastx = x;
lastLastVel = lastVel;
lastVel = vel;

lastx2 = x2;
lastLastVel2 = lastVel2;
lastVel2 = vel2;
}

// 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 & 0b00000000 | mode;
        } else {
            TCCR1B = TCCR1B & 0b00000000 | 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 = 0x07; break;
            default: return;
        }
        TCCR2B = TCCR2B & 0b00000000 | mode;
    }
}