```
#include <math.h>

// Pin declares
int pwmPin = 5; // PWM output pin for motor 1
int dirPin = 8; // direction output pin for motor 1
int sensorPosPin = A2; // input pin for MR sensor
int fsrPin = A3; // input pin for FSR sensor

// Position tracking variables
int updatedPos = 0;     // keeps track of the latest updated value
of the MR sensor reading
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 over the 180 degree mark occurred
boolean flipped = false;

// Kinematics variables
double xh = 0;           // position of the handle [m]
double theta_s = 0;      // Angle of the sector pulley in deg
double xh_prev;          // Distance of the handle at previous time
step
double xh_prev2;
double dxh;              // Velocity of the handle
double dxh_prev;
double dxh_prev2;
double dxh_filt;         // Filtered velocity of the handle
double dxh_filt_prev;
double dxh_filt_prev2;

double ddxh;
double ddxh_prev;
double ddxh_filt;
double ddxh_filt_prev;
```

```
// Initialize Variables
int joystickPin = A0;
int buttonPin = 6;
int k_stiff = 500;
int k_land = 100;
double b_object = 0.006;
double b_v = 0.1;
bool button_status;
bool last_button_status;
bool current_state;
double final_angle;
double current_angle;
double lengthRate = 1.5;
double angles_object_max[9] = {99.46232, 170.54993, 22.95518, 38.
97205, 70.85657, 148.95654, 125.86484, 92.38594, 55.03925};  // the
right most angle that can catch each object
double angles_object_min[9] = {80.53768, 152.58017, 13.91472, 28.
40809, 65.54061, 143.66332, 116.06267, 87.61406, 51.22095}; //the
left most angle that can catch each object
double weights[9] = {0.548, 0.548, 0.826, 1.2, 0.3, 0.3, 1.1, 0.826,
0.826};  //////change later
double lengths[9] = {250.0, 266.22777, 291.22777, 490.83269, 513.
51648, 515.83269, 533.09519, 575.0, 725.0};
double path_length;
bool object_found; // if the final_angle can catch stone or gold
bool caught; // check if reel in or out

double width_image = 1.2;
double height_image = 0.700;
double air_weight = 0.01;
double g_earth = 9.81;

double current_weight;
int current_index;



//************************************************************
//****************** Initialize Variables (END) ***************
//************************************************************

// Force output variables
double force = 0;           // force at the handle
```

```
double Tp = 0;              // torque of the motor pulley
double duty = 0;            // duty cylce (between 0 and 255)
unsigned int output = 0;    // output command to the motor


// ---------------------------------------------------------------
// Setup function
// ---------------------------------------------------------------
void setup()
{
  // Set up serial communication
  Serial.begin(115200);
  // Serial.begin(9600);

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

  // Input pins
  pinMode(sensorPosPin, INPUT); // set MR sensor pin to be an input
  pinMode(fsrPin, INPUT);       // set FSR sensor pin to be an input
  pinMode(joystickPin, INPUT);
  pinMode(buttonPin, INPUT);

  // Output pins
  pinMode(pwmPin, OUTPUT);  // PWM pin for motor A
  pinMode(dirPin, OUTPUT);  // dir pin for motor A

  // Initialize motor
  analogWrite(pwmPin, 0);     // set to not be spinning (0/255)
  digitalWrite(dirPin, LOW);  // set direction

  // Initialize position valiables
  lastLastRawPos = analogRead(sensorPosPin);
  lastRawPos = analogRead(sensorPosPin);

  current_state = 0;
  object_found = 0;
  current_index = -1;

  // scale variables
  for (int i = 0; i < 9; i++) {
    angles_object_max[i] = angles_object_max[i] * PI / 180.0;
    angles_object_min[i] = angles_object_min[i] * PI / 180.0;
```

```
    lengths[i] = lengths[i] / 1000.0;
    weights[i] = weights[i] * 3;
  }

}


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

  //*********************************************************
  //*** Section 1. Compute position in counts ***************
  //*********************************************************

  // Get voltage output by MR sensor
  rawPos = analogRead(sensorPosPin);  //current raw position from MR
sensor

  // Calculate differences between subsequent MR sensor readings
  rawDiff = rawPos - lastRawPos;          //difference btwn current
raw position and last raw position
  lastRawDiff = rawPos - lastLastRawPos;  //difference btwn current
raw position and last last raw position
  rawOffset = abs(rawDiff);
  lastRawOffset = abs(lastRawDiff);

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

  // Keep track of flips over 180 degrees
  if ((lastRawOffset > flipThresh) && (!flipped)) { // enter this
anytime the last offset is greater than the flip threshold AND it
has not just flipped
    if (lastRawDiff > 0) {       // check to see which direction the
drive wheel was turning
      flipNumber--;              // cw rotation
    } else {                     // if(rawDiff < 0)
      flipNumber++;              // ccw rotation
    }
```

```
    if (rawOffset > flipThresh) { // check to see if the data was
good and the most current offset is above the threshold
      updatedPos = rawPos + flipNumber * rawOffset; // update the
pos value to account for flips over 180deg using the most current
offset
      tempOffset = rawOffset;
    } else {                      // in this case there was a blip in
the data and we want to use lastactualOffset instead
      updatedPos = rawPos + flipNumber * lastRawOffset; // update
the pos value to account for any flips over 180deg using the LAST
offset
      tempOffset = lastRawOffset;
    }
    flipped = true;            // set boolean so that the next time
through the loop won't trigger a flip
  } else {                        // anytime no flip has occurred
    updatedPos = rawPos + flipNumber * tempOffset; // need to update
pos based on what most recent offset is
    flipped = false;
  }

  // Compute Position in Meters
  double rh = 0.031;  //[m]
  double ts = 0.0408 * updatedPos - 25.738; // Compute the angle of
the sector pulley (ts) in degrees based on updatedPos
  xh = lengthRate * rh * ts * PI / 180;



  // Calculate velocity with loop time estimation
  dxh = (double)(xh - xh_prev) / 0.001;
  ddxh = double(dxh - dxh_prev) / 0.001;

  // Calculate the filtered velocity of the handle using an infinite
impulse response filter
  dxh_filt = .9 * dxh + 0.1 * dxh_prev;
  ddxh_filt = .9 * ddxh + 0.1 * ddxh_prev;



  // Record the position and velocity
  xh_prev2 = xh_prev;
  xh_prev = xh;

  dxh_prev2 = dxh_prev;
```

```
  dxh_prev = dxh;

  dxh_filt_prev2 = dxh_filt_prev;
  dxh_filt_prev = dxh_filt;

  ddxh_prev = ddxh;
  ddxh_filt_prev = ddxh_filt;

  //*************************************************************
  //****** Assign a Motor Output Force in Newtons ***************
  //*************************************************************
  //******************** Rendering Algorithms ********************
  //*************************************************************

  button_status = digitalRead(buttonPin);
  if (button_status != last_button_status) {
    if (button_status == HIGH) {
      current_state = 1;
      final_angle = analogRead(joystickPin) * PI / 1023.0; // set
final angle for each press of button , i.e. each run of game
      object_found = 0; // re-initialize for each press of button ,
i.e. each run of game
      current_index = -1; // re-initialize for each press of button
, i.e. each run of game
      caught = 0; // new run, re-initialize indicator
    }
  }

  last_button_status = button_status;
  current_angle = analogRead(joystickPin) * PI / 1023.0; // map
0~1023 to 0~pi
  if (current_state == 0 ) {
    if (xh < -0.01) {
      force = k_land * abs(xh + 0.01);
    } else if (xh > 0.01) {
      force = -k_land * (xh - 0.01);
    } else {
      force = 0;
    }

    Serial.print(current_state);
    Serial.print(" ");
    Serial.println(current_angle, 5);
```

```
  } else {  //current_state == 1

    if (current_index == -1) {  // only go to for loop to check if
any object on this path when current_index = -1
      // when current_index = -1 means first time in the loop for
this run of game
      // when current_index >=0 means found targeted object,
      // when current_index=-2 means air for this run

      for (int i = 0; i < 9; i++) {
        if (angles_object_min[i] < final_angle && final_angle <
angles_object_max[i] && angles_object_min[i] != 0 ) {
          object_found = 1;
          path_length = lengths[i];
          current_weight = weights[i]; //////add to global
          current_index = i; // initialize = -1, need to initialize
after each run of game
          angles_object_min[i] = 0; // remove this object from the
checking list
          angles_object_max[i] = 0;
          break; // once found, stop searching for others
        }
      }

      // no object found in the for loop for this run of game
      if (!object_found) {
        current_index = -2; // means catch air for this run of game,
don't need to check in for loop again before next run
        if (final_angle <= atan2(height_image, width_image / 2)) {
          path_length = (width_image / 2) / cos(final_angle);
        } else if (final_angle > atan2(height_image, width_image /
2) && final_angle < (PI - atan2(height_image, width_image / 2))) {
          path_length = height_image / sin(final_angle);
        } else {
          path_length = (width_image / 2) / cos(PI - final_angle);
        }
        current_weight = air_weight;
      }

    }

    if (xh < 0) {
      force = k_land * abs(xh);
```

```
    } else if (xh >= path_length) {
      force = -k_stiff * (xh - path_length);
      caught = 1;
    }

    if (caught == 0) {
      if ( xh > 0 && xh < path_length) {
//        force = b_gripper * dxh_filt;
        force = 0;

      }
    } else { //caught ==1
      if (xh > 0.01 && xh < path_length) {
        if (dxh_filt < 0) {
//          force = current_weight*sin(final_angle) - current_weight
* ddxh_filt/g_earth + (b_gripper + b_object) * dxh_filt;  // check
for direction in the future depends on the wheel design
        force = current_weight*sin(final_angle) + ( b_object) *
dxh_filt;
        } else {
          force = abs(current_weight*sin(final_angle) - b_v *
dxh_filt) ;
          if (force > 0.2){
            force = 0.2;
          }
        }
      }
      if (xh <= 0.007) {
        current_state = 0;
        final_angle = 0;

      }
    }

    Serial.print(current_state);
    Serial.print(" ");
    Serial.println((xh), 5);

  }



  // Calculate the motor torque
  double rp = 0.012;    //[m]
```

```
  double rs = 0.047;    //[m]
  Tp = force * rh * rp / rs; // Compute the require motor pulley
torque (Tp) to generate that force using kinematics
  // rs = r of large gear, rh = r of handle, rp = r of small gear

  //*************************************************************
  //******* Assign a Motor Output Force in Newtons (END) ********
  //*************************************************************
  //*************************************************************




  //*************************************************************
  //************ Force output ***********************************
  //*************************************************************

  // Determine correct direction for motor torque
  if (force > 0) {
    digitalWrite(dirPin, HIGH);
  } else {
    digitalWrite(dirPin, LOW);
  }

  // Compute the duty cycle required to generate Tp (torque at the
motor pulley)
  duty = sqrt(abs(Tp) / 0.03);

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

// -----------------------------------------------------------
// Function to set PWM Freq
// -----------------------------------------------------------
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;
  }
}
```