/\*

 \* GROUP 21

 \* Coded by: Hunter Burnett & Jaskin Kabir

 \* Date: 02/22/2024

 \* Class: ECGR 4161 | Intro to Robotics Spring 2024

 \* Lab04

\*/

#include "SimpleRSLK.h"

#define DELAY\_MS            250   // delay in milliseconds

// Default pwm signals (percentage-% of power 0-100) for both RSLK motor.

// Change these values as needed

#define LEFT\_MOTOR\_SPEED    15    // Speed percentage

#define RIGHT\_MOTOR\_SPEED   15    // Speed percentage

#define LEFT\_TURN\_SPEED     12    // Speed percentage

#define RIGHT\_TURN\_SPEED    12    // Speed percentage

// Value for turning directions (do not change)

#define CCW                  1     // rotate robot counter clockwise

#define CW                   2     // rotate robot clockwise

// RSLK specified mechanics (do not change)

#define wheelDiameter       6.999      // in centimeters

#define cntPerRevolution    360        // Number of encoder (rising) pulses every time the wheel turns completely

/\* Place code here to only run once \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setup() {

  //Do not edit this setup function

  Serial.begin(9600);         // Start serial com at 9600 baud rate

  setupRSLK();                // Initialize RSLK functions and classes

  setupWaitBtn(LP\_LEFT\_BTN);  // Setup left button on Launchpad

  setupLed(RED\_LED);          // Red led in rgb led

  setupLed(GREEN\_LED);        // Red led in rgb led

}

/\* Place code here to run forever/loop/repeat \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void loop() {

  startProgram();   // wait until left MSP 432 button is pressed to start program

  /\*\*\*\*\*\*\*\*\*\*\*\* after reading, you may delete this for your testing and use \*\*\*\*\*\*\*\*\*\*\*

    Test your algorithm and/or functions by placing them here just like the example below.

    It does not matter what is in this loop when you submit. It will be replaced by the

    instructor or TA with a specific algorithm of their choosing.

    forward(10);

    rotate(CCW, 360);

    forward(90);

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

  forward(500);

}

/\* Function Name: rotate

   Input: 2 int inputs

   Return: void

   Details:  Will rotate the robot the specified number of degrees in the specified direction

   Uses the encoders to ensure this number of degrees is reached as closely as possible

\*/

void rotate(int rotate\_dir, int rotate\_deg) {

  //Put your code here that will rely on motor encoders for accurately rotating in place

  uint32\_t robot\_base\_diam = 14.0;

  uint32\_t distance = (robot\_base\_diam \* PI) \* (rotate\_deg / 360.0);

  uint32\_t targetCount = countForDistance(wheelDiameter, cntPerRevolution, distance);

  resetLeftEncoderCnt();

  resetRightEncoderCnt();

  int leftCount = 0;

  int rightCount = 0;

  switch (rotate\_dir) {

    case CCW:

      setMotorDirection(RIGHT\_MOTOR, MOTOR\_DIR\_FORWARD);

      setMotorDirection(LEFT\_MOTOR, MOTOR\_DIR\_BACKWARD);

    case CW:

      setMotorDirection(RIGHT\_MOTOR, MOTOR\_DIR\_BACKWARD);

      setMotorDirection(LEFT\_MOTOR, MOTOR\_DIR\_FORWARD);

  }

  enableMotor(BOTH\_MOTORS);

  setMotorSpeed(LEFT\_MOTOR, LEFT\_TURN\_SPEED);

  setMotorSpeed(RIGHT\_MOTOR, RIGHT\_TURN\_SPEED);

  while (leftCount < targetCount || rightCount < targetCount) {

    leftCount = getEncoderLeftCnt();

    rightCount = getEncoderRightCnt();

  }

  disableMotor(BOTH\_MOTORS);

  delay(2000);

}

/\* Function Name: pid

   Input: 2 int inputs, 3 double inputs

   Return: int value

   Details: Implements PID Control to calculate an

   adjustment to be added to the input of a system based

   on the error detected at the output of the system.

   Requires 3 input parameters to be tuned for the system.

\*/

int pid(int setpoint, int input, double kp, double ki, double kd) {

  static unsigned long lastTime = millis();

  static int lastErrorCalculated = input - setpoint;

  static double cumulativeError = 0;

  unsigned long currentTime = millis();

  double deltaTime = (double)(currentTime - lastTime);

  lastTime = currentTime;

  int error = input - setpoint;

  float prop = kp \* error;

  cumulativeError += error \* deltaTime;

  float integral = cumulativeError \* ki;

  float derivative = ((error - lastErrorCalculated) / deltaTime) \* kd;

  lastErrorCalculated = error;

  int adjustment = prop + integral + derivative;

  //Serial.println("e: " + String(error) + " p: " + String(prop) + " i: " + String(integral) +  " d: " + String(derivative) + " sum: " +  String(adjustment));

  return adjustment;

}

/\* Function Name: forward

   Input: 1 float input

   Return: void

   Details: Function called to command robot to move forward for the distance--in centimeters-- specified by the user.

   Uses PID Control to ensure robot drives straight

\*/

void forward(float travel\_dist) {

  int minSpeed = 5;

  int maxSpeed = 65;

  float kp = 0.396;

  float ki = 0.0005;

  float kd = 49.8;

  //Put your code here that will rely on motor encoders to drive your robot straight for a specified distance

  resetLeftEncoderCnt();

  resetRightEncoderCnt();

  int targetCount = countForDistance(wheelDiameter, cntPerRevolution, travel\_dist);

  int leftSpeed = LEFT\_MOTOR\_SPEED;

  int rightSpeed = RIGHT\_MOTOR\_SPEED;

  setMotorDirection(RIGHT\_MOTOR, MOTOR\_DIR\_FORWARD);

  setMotorDirection(LEFT\_MOTOR, MOTOR\_DIR\_FORWARD);

  int rightCnt = 0;

  int leftCnt = 0;

  enableMotor(BOTH\_MOTORS);

  while ((rightCnt <= targetCount || leftCnt <= targetCount)) {

    rightCnt = getEncoderRightCnt();

    leftCnt = getEncoderLeftCnt();

    int error = rightCnt - leftCnt;

    // Difference between counts should be 0, so setpoint is 0

    int adjustment = pid(0, error, kp, ki, kd);

    leftSpeed += adjustment;

    if (leftSpeed < minSpeed) {

      //blinkRedLED(50);

      leftSpeed = minSpeed;

    }

    if (leftSpeed > maxSpeed) {

      leftSpeed = maxSpeed;

      //blinkLED(50);

    }

    setMotorSpeed(LEFT\_MOTOR, leftSpeed);

    setMotorSpeed(RIGHT\_MOTOR, rightSpeed);

    delay(50);

  }

  disableMotor(BOTH\_MOTORS);

  delay(2000);

}

/\* Function Name: countForDistance

   Input: 2 int input variables, 1 float input

   Return: int value

   Details: Function called to calculate the number of pulses need to travel a specified

            distance by the user input variable "distance."

\*/

uint32\_t countForDistance(float wheel\_diam, uint16\_t cnt\_per\_rev, uint32\_t distance) {

  float temp = (wheel\_diam \* PI) / cnt\_per\_rev;

  temp = distance / temp;

  return int(temp);

}

/\* Function Name: startProgram

   Input: void

   Details: Function called to wait for a button to be pressed

            in order to start the robot program

\*/

void startProgram() {

  /\* Setup message to print to serial montor \*/

  String btnMsg = "Push left button on Launchpad to start lab program.\n";

  //String btnMsg = "";

  waitBtnPressed(LP\_LEFT\_BTN, btnMsg, RED\_LED); //Setup button, msg, LED

  /\* Using an LED as a DELAY \*/

  blinkLED(1000); //Cause LED to blink for a period of one second

  blinkLED(1000); //Cause LED to blink for a period of one second

}

/\* Function Name: blinkLED

   Input: integer (period) in milliseconds

   Details: Function call that will blink a colored LED for a period specified.

\*/

int blinkLED(int period) {

  int pause = period / 2;         // Determine on/off time

  digitalWrite(GREEN\_LED, HIGH);  // Turn LED on

  delay(pause);                   // Time the LED is on

  digitalWrite(GREEN\_LED, LOW);   // Turn LED off

  delay(pause);                   // Time LED is off

}

/\* Function Name: blinkRedLED

   Input: integer (period) in milliseconds

   Details: Function call that will blink a red LED for a period specified.

\*/

int blinkRedLED(int period) {

  int pause = period / 2;         // Determine on/off time

  digitalWrite(RED\_LED, HIGH);  // Turn LED on

  delay(pause);                   // Time the LED is on

  digitalWrite(RED\_LED, LOW);   // Turn LED off

  delay(pause);                   // Time LED is off

}