Aryan Gupta
Partner: Anthony Samagaio

Video Link: https://youtu.be/_FC7VgYZEkI

Objective: The video will demonstrate a program to drive a motorized two-wheeled vehicle in a square then an equilateral triangle. The vehicle will use on-board encoders to ensure the vehicle travels in the correct direction.

Reflection: One of the biggest issues that I faced was that at certain speeds, the abrupt stop would rotate the vehicle forward and move the vehicle in an undesired orientation. This undesired behavior would be magnified by each stop the vehicle took. In order to remedy this, a speed transform was implemented. The first iteration of the speed transform was a simple step. For the first 25% of the journey, the vehicle would travel slowly, for the next 50% of the journey, it would travel at the default speed, and lastly would slow back down to the slow speed, this helped tremendously during turns. Another speed transform was created which ramped up the speed for the first 10% to the max speed then slowed down the vehicle after 40% of the journey was completed, the effect of this was largely seen during the straight travel portion of the journey.

Code:

```
//=============================================================================
// File:  DriveSquare
//
// 2020-06-22 - James Conrad, from code borrowed from TI
//    (some of this original code by Franklin S. Cooper Jr.)
// 2020-06-29 - Aryan Gupta
// Summary:
//  This example will demonstrate the various features of the Encoder library.
//  The robot will go forward by a specified amount in inches. A robot naturally
//  will not go straight due to factors such as wheel wobble or differences in
//  behavior of the left and right motor. Incorporating PID with the encoder count
//
//=============================================================================

#include "SimpleRSLK.h"

// Various constants defining the physical characteristics
// of the vehicle
#define WHEEL_DIAM 0.06959 // in meters
#define WHEEL_BASE 0.143 // in meters
```

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

// default, max, and min speed of the wheels
// the correction speed is used if the encoder
// values are out of sync and a wheel needs to sped up
#define WHEEL_SPEED 15 // 15%
#define MAX_SPEED 25 // 25%
#define MIN_SPEED 6 // 6%
#define CORRECTION_SPEED_DELTA 5 // 5%

// Percent of journey to stop acceleration
// and start deceleration
#define SPEED_RAMP_ACCEL 0.10 // 0% to 10% accelerate
#define SPEED_RAMP_DECEL 0.40 // 40% to 100% decelerate
#define SPEED_STEP_ACCEL 0.25
#define SPEED_STEP_DECEL 0.75

/// Calculates the speed (in percent) the vehicle
/// should travel at using the ramp function
///
/// @param progress The progress of the vehicle in a percentage
/// @param SPEED_RAMP_ACCEL A constant defined var containing the
///        at what percent to stop acceleration
/// @param SPEED_RAMP_DECEL A constant defined var containing the
///        at what percent to stat deceleration
/// @return The speed of the vehicle
uint8_t calc_speed_ramp(float progress) {
   Serial.println();
   Serial.print("progress: ");
   Serial.print(progress);

   if (progress < SPEED_RAMP_ACCEL) { // acceleration ramp
       float slope = float(MAX_SPEED - MIN_SPEED) / SPEED_RAMP_ACCEL;

       Serial.print("\t slope: ");
       Serial.print(slope);

       return slope * progress + MIN_SPEED;
   } else if (progress > SPEED_RAMP_DECEL) { // deceleration ramp
       progress -= (1 - SPEED_RAMP_DECEL);
       float slope = float(MAX_SPEED - MIN_SPEED) / -(SPEED_RAMP_DECEL);
```

```
        Serial.print("\t slope: ");
        Serial.print(slope);

        return slope * progress + MAX_SPEED;
    } else { // plauteu section of ramp
        Serial.print("\t slope: ");
        Serial.print("platue");

        return MAX_SPEED;
    }
}

/// Calculates the speed (in percent) the vehicle
/// should travel at using a simple step function.
/// Will travel vehicle slowly for some time, jump to
/// normal speed, then jump back to slow speed for the
/// last leg of the journey
///
/// @param progress The progress of the vehicle in a percentage
/// @param SPEED_STEP_ACCEL A constant defined var containing the
///        at what percent to resume normal speed
/// @param SPEED_STEP_DECEL A constant defined var containing the
///        at what percent to go back to slow speed
/// @return The speed of the vehicle
uint8_t calc_speed_step(float progress) {
    if (progress < SPEED_STEP_ACCEL) {
        return MIN_SPEED;
    } else if (progress > SPEED_STEP_DECEL) {
        return MIN_SPEED;
    } else {
        return WHEEL_SPEED;
    }
}

/// Converts the an angle of rotation to the amount the
/// wheels will travel. Assumes vehicle will use both wheels
/// to rotate
///
/// @param degrees The degrees of rotation to convert
/// @param WHEEL_BASE The distance between the 2 wheels
///        Should be renamed to WHEEL_TRACK
/// @return The distance one wheel travels in meters
float calc_degrees_to_distance_spin (float degrees) {
```

```
    float wheel_base_circum = WHEEL_BASE * PI;
    float angle_ratio = degrees / 360.0;
    return angle_ratio * wheel_base_circum;
}

/// Converts wheel travel distance to an encoder count
/// Useful for calculateing how many encoder counts needed
/// to travel a distance
///
/// @param distance The distance to convert
/// @param WHEEL_DIAM The diameter of the wheel
/// @param CNT_PER_REV The number of encoder counts per
///        one revolution of the wheel
/// @return The number of encoder counts it should take
///         to travel \param distance distance
uint16_t calc_enoder_count (float distance) {
    float wheel_circum = WHEEL_DIAM * PI;
    float wheel_rotations = distance / wheel_circum;
    return uint16_t( wheel_rotations * CNT_PER_REV );
}

/// Keeps the motor in sync using the encoders until both
/// encoders have reached \param cnt count. Also will
/// run a speed control function to adjust the speed as it
/// travels
///
/// @todo Make speed control function optional
///
/// @param cnt The number of encoder counts to keep sync until
/// @param speed_ctrl_func A functional pointer used to determine the
///        base speed of the motors, used to transform the speed thoughout
///        the journey of the vehicle
/// @param WHEEL_SPEED The default wheel speed when no transform is used
void sync_motors_until_cnt(uint16_t cnt, uint8_t (*speed_ctrl_func)(float)) {
    uint16_t leftCount = getEncoderLeftCnt();
    uint16_t rightCount = getEncoderRightCnt();
    uint8_t baseSpeed = WHEEL_SPEED;
    uint8_t leftMotorSpeed = baseSpeed;
    uint8_t rightMotorSpeed = baseSpeed;

    while (leftCount < cnt and rightCount < cnt) {
        uint16_t avgCount = (leftCount + rightCount) / 2;
        float progress = float(avgCount) / cnt;
```

```
// first get base speed using the transform function
baseSpeed = speed_ctrl_func(progress);

Serial.print(baseSpeed);
Serial.print("\t");

leftCount = getEncoderLeftCnt();
rightCount = getEncoderRightCnt();

Serial.print(leftCount);
Serial.print("\t");
Serial.print(rightCount);
Serial.print("\t");

// then adjust the left and right wheel speeds to sync up
// the wheel. only gets synced up if the encoder counts there
// is more than 10 counts of difference otherwise reset the
// wheel speeds back to default
int abs_diff = abs(int(leftCount) - rightCount);
if (abs_diff > 10) {
    Serial.print("off_");

    if (leftCount < rightCount) { // left wheel lagging
        leftMotorSpeed = baseSpeed + CORRECTION_SPEED_DELTA;
        rightMotorSpeed = baseSpeed - CORRECTION_SPEED_DELTA;

        Serial.print("inc_left");
    }

    if (rightCount < leftCount) { // right wheel lagging
        leftMotorSpeed = baseSpeed - CORRECTION_SPEED_DELTA;
        rightMotorSpeed = baseSpeed + CORRECTION_SPEED_DELTA;

        Serial.print("inc_right");
    }

    // Check bounds for under/overflow
    // if (leftMotorSpeed <= speed_inc) {
    //  leftMotorSpeed = speed_inc;
    // }

    // if (rightMotorSpeed <= speed_inc) {
```

```
            //  rightMotorSpeed = speed_inc;
            // }

            // if (leftMotorSpeed >= 100) {
            //  leftMotorSpeed = 100;
            // }

            // if (rightMotorSpeed >= 100) {
            //  rightMotorSpeed = 100;
            // }

            Serial.print("\t");
            Serial.print(leftMotorSpeed);
            Serial.print("\t");
            Serial.print(rightMotorSpeed);
        } else {
            leftMotorSpeed = baseSpeed;
            rightMotorSpeed = baseSpeed;

            Serial.print("in_sync");
        }

        setMotorSpeed(LEFT_MOTOR,leftMotorSpeed);
        setMotorSpeed(RIGHT_MOTOR,rightMotorSpeed);

        Serial.println();
    }
}

/// Drives the vehicle straight \param meters number of meters
///
/// @param meters The number of meters to travel straight. Use
///           a negative value to travel backwards
void drive_straight(float meters) {
   uint16_t encoder_cnt_needed = calc_enoder_count(abs(meters));

   resetLeftEncoderCnt();
   resetRightEncoderCnt();

   if (meters > 0) {
       setMotorDirection(BOTH_MOTORS, MOTOR_DIR_FORWARD);
   } else {
       setMotorDirection(BOTH_MOTORS, MOTOR_DIR_BACKWARD);
```

```
    }

    enableMotor(BOTH_MOTORS);
    setMotorSpeed(BOTH_MOTORS, WHEEL_SPEED);

    sync_motors_until_cnt(encoder_cnt_needed, calc_speed_ramp);

    disableMotor(BOTH_MOTORS);
}

/// Spins the vehicele using both wheels \param degrees number
/// of degrees.
///
/// @param degrees The number of degrees to spin the vehicle, positive
///        numbers mean CCW, negative numbers mean CW
void spin(float degrees) {
    float distance_to_turn = calc_degrees_to_distance_spin(abs(degrees));
    uint16_t encoder_cnt_needed = calc_enoder_count(distance_to_turn);

    resetLeftEncoderCnt();
    resetRightEncoderCnt();

    if (degrees > 0) {
        setMotorDirection(RIGHT_MOTOR, MOTOR_DIR_FORWARD);
        setMotorDirection(LEFT_MOTOR, MOTOR_DIR_BACKWARD);
    } else {
        setMotorDirection(RIGHT_MOTOR, MOTOR_DIR_BACKWARD);
        setMotorDirection(LEFT_MOTOR, MOTOR_DIR_FORWARD);
    }

    enableMotor(BOTH_MOTORS);
    setMotorSpeed(BOTH_MOTORS, WHEEL_SPEED);

    sync_motors_until_cnt(encoder_cnt_needed, calc_speed_step);

    disableMotor(BOTH_MOTORS);
}

/// Draws a polygon with \param num_sides number of sides
///
/// @param num_sides The number of sides in the polygon to draw
void draw_polygon(uint8_t num_sides) {
    uint8_t i = 0;
```

```
    for(; i < num_sides; ++i) {
        drive_straight(0.5);
        delay(1000);

        spin(360.0 / num_sides);
        delay(1000);
    }
}

///===============================================================================
/// The setup() funtion runs one time at the beginning of the Energia program
///===============================================================================
void setup() {
    Serial.begin(115200);
    setupRSLK(); // Set up all of the pins & functions needed to
                 //   be used by the TI bot
    setupWaitBtn(LP_LEFT_BTN);   // Left button on Launchpad
    setupLed(RED_LED);           // Red LED of the RGB LED
}

///===============================================================================
/// The loop() function runs after the setup() function completes in an
/// Energia program and will continue to run in a repeating loop until the
/// LaunchPad is reset or powered off
///===============================================================================
void loop() {
    String btnMsg = "Push left button on Launchpad to start demo.\n";
    waitBtnPressed(LP_LEFT_BTN,btnMsg,RED_LED);
    delay(2000); // wait 2 seconds for user to run away

    // draw a square
    draw_polygon(4);

    // spin 30 deg so the vehicle doesnt run into my
    // parents rug
    spin(30);

    // draw a triangle
    draw_polygon(3);

    // spin back 30 deg to counter the earlier 30 deg turn
    spin(-30);
}
```