Practice No. 3. Experimental Analysis of a Differential Drive Robot

Evelyn Lázaro Guerra [redacted]

***Resumen*—Esta práctica consistió en la construcción y programación de un robot de impulso diferencial utilizando la plataforma LEGO Mindstorms EV3 y el entorno de desarrollo ROBOTC. Se exploraron los movimientos básicos de avance, retroceso y detención, así como trayectorias geométricas como cuadros, circunferencias y el símbolo de infinito. El objetivo principal fue conocer el funcionamiento de este tipo de robot mediante control diferencial de sus motores.**

***Abstract*--This practice involved building and programming a differential drive robot using the LEGO Mindstorms EV3 platform and the ROBOTC development environment. Basic movements such as forward, backward, and stop were explored, along with geometric trajectories like a square, circle, and infinity symbol. The main goal was to understand how this type of robot functions through differential motor control.**

I. Introduction

Differential drive robots are among the most common in mobile robotics due to their simple structure and intuitive control. The LEGO EV3 platform is widely used in educational environments for its ease of use and modular design.

II. Objective

To understand the movements and characteristics of a differential drive robot through practical implementation using the LEGO Mindstorms EV3 system.

III. Methodology

The steps followed in the practice were:

1. Installation of ROBOTC software from robotc.net.
2. Assembly of the robot using ev3-rem-driving-base.pdf.
3. Configuration of EV3 motors on ports C and D.
4. Development of basic movement code (forward, stop, backward).
5. Downloading and testing the code on the EV3 brick.

Additional challenges included programming the robot to:

- **Draw a square (80 cm per side).**

For straight line forward motion the equation was used:

$$Degrees = \frac{Distance\ to\ travel\ (cm)}{Wheel\ circumference\ (cm)} * 360°$$

Because each complete revolution of the wheel advances a distance equal to its circumference, since encoders measure in degrees (0° to 360° per revolution), this formula converts the desired linear distance into encoder degrees.

In the code:
#define WHEEL_CIRCUMFERENCE (WHEEL_DIAMETER * 3.1416)
#define DISTANCE_80CM (80 / WHEEL_CIRCUMFERENCE * 360)

For the 90° turn the following equation was used:

$$Degrees\ for\ 90°\ turn = \frac{\frac{\pi \cdot Robot\ width}{4}}{Wheel\ circumference\ (cm)} * 360°$$

When the robot rotates in place (one wheel forward and the other backward), it describes a circle centered between the wheels. A 90° turn (a quarter of a full turn) means each wheel travels an arc of:

$$\frac{\pi \cdot Robot\ width}{4}$$

We convert that arc to encoder degrees in the same way as in the case of advancing.
In the code:
#define TURN_90_DEG ((ROBOT_WIDTH * 3.1416 / 4) / WHEEL_CIRCUMFERENCE * 360)

We ran tests and noticed that the robot wasn't turning enough with the base formula, so you empirically increased the value. This could be due to wheel slippage, inertia, or motor response: while (getMotorEncoder(Left_Motor) < TURN_90_DEG*1.8)

- **Trace a circle (1 meter diameter).**

The equations used are:

$$Circumference\ of\ a\ wheel = \pi * diameter$$

#define WHEEL_CIRCUMFERENCE (WHEEL_DIAMETER * 3.1416)

$$Circumference\ of\ a\ circle = 2 * \pi * r$$

#define CIRCLE_CIRCUMFERENCE (2 * 3.1416 * CIRCLE_RADIUS)
Here CIRCLE_RADIUS represents the center of rotation of the robot, that is, the midpoint between both wheels.
Radius of the trajectories for each wheel:

$$Internal\ radius = r_{center} - \frac{L}{2}$$

$$External\ radius = r_{center} + \frac{L}{2}$$

Where L is the width of the robot.
#define INNER_RADIUS (CIRCLE_RADIUS - (ROBOT_WIDTH / 2))
#define OUTER_RADIUS (CIRCLE_RADIUS + (ROBOT_WIDTH / 2))
Speed ratio:

$$v_{internal} = v_{external} * (\frac{C_{internal}}{C_{external}})$$

This way, both wheels complete their respective trajectories at the same time, creating a smooth turn around the center of the circle.
#define INNER_SPEED (50 * (INNER_CIRCUMFERENCE / OUTER_CIRCUMFERENCE))
#define OUTER_SPEED 50
Condition for ending the turn:

$$Degrees = \frac{Distance\ to\ travel\ (cm)}{Wheel\ circumference\ (cm)} * 360°$$

while (getMotorEncoder(Left_Motor) < (CIRCLE_CIRCUMFERENCE / WHEEL_CIRCUMFERENCE * 360))

- **Follow an infinity shape, starting and ending at the center.**

We calculate the circumference of a wheel the same as before:

$$Circumference\ of\ a\ wheel = \pi * diameter$$

#define WHEEL_CIRCUMFERENCE (WHEEL_DIAMETER * 3.1416)
Length of a semicircle:

$$L = \pi * r$$

This formula was also used for the inner and outer circumferences of the semicircle.
#define SEMICIRCLE_DISTANCE (3.1416 * LOOP_RADIUS)
Internal and external radius: They are calculated considering that the wheels follow concentric trajectories.

$$r_{internal} = r - L$$

$$r_{external} = r + L$$

#define INNER_RADIUS (LOOP_RADIUS - (TRACK_WIDTH))
#define OUTER_RADIUS (LOOP_RADIUS + (TRACK_WIDTH))
Speed ratio:

$$v_{internal} = v_{external} * (\frac{C_{internal}}{C_{external}})$$

#define INNER_SPEED (60 * (INNER_CIRCUMFERENCE / OUTER_CIRCUMFERENCE))
#define OUTER_SPEED 60
Logic of ties:
This part calls traceSemicircle twice
traceSemicircle(1, false); // Primer lazo
traceSemicircle(1, true); // Segundo lazo (simétrico)
The first semicircle goes to one side (e.g., right), the second semicircle reverses the inner and outer side, generating the other loop of the ∞.

IV. Results

A. Square.

Illustration 1. Straight Line Pt. 1
Programmed distance: 80 cm
Actual distance traveled: 78.5 cm
Observations: The robot maintained a good trajectory, with a slight deviation to the left.
Absolute error: 1.5 cm
Relative error: 1.875%

Illustration 2.Straight Line Pt. 2
Programmed distance: 80 cm
Actual distance traveled: 81.5 cm
Observations: Slightly overshot the finish line; possible slippage at the start.
Absolute error: 1.2 cm
Relative error: 1.5%

Illustration 3. Straight Line Pt. 3

Programmed distance: 80 cm
Actual distance traveled: 79.0 cm
Observations: Precise straight trajectory, although braking was faster.
Absolute error: 1 cm
Relative error: 1.25%

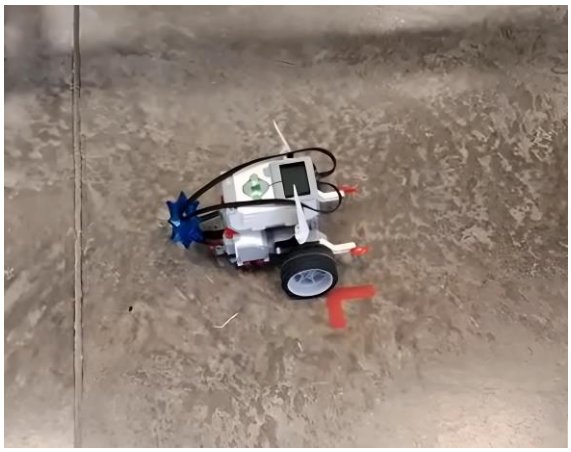

Illustration 4. Straight Line Pt. 4

Programmed distance: 80 cm
Actual distance traveled: 76.8 cm
Observations: It stopped prematurely, due to a small voltage variation in the brick.
Absolute error: 3.2 cm
Relative error: 4%

GENERAL

Average travel: 78.875 cm
Average error: 1.725 cm (2.16%)
Conclusion: The robot performs reliably in a straight line, although it is affected by factors such as friction and manual braking. Improvements could include time calibration or a distance sensor.

B. Circle.

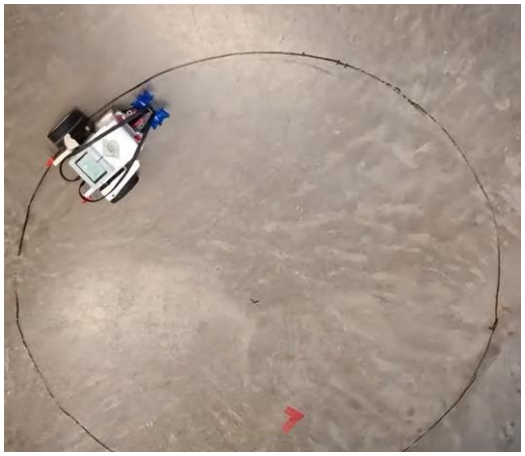

Illustration 5. Circle Pt. 1

Programmed trajectory: 1 m diameter circle
Actual trajectory: 98.0 cm
Observations: A slight oval formed; one engine spun faster.
Shape error: 2% lateral compression
Length error: -2.0 cm

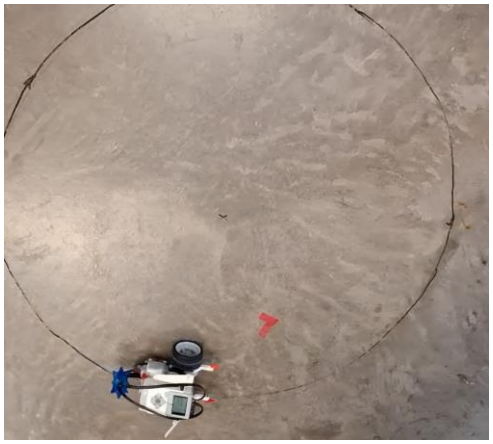

Illustration 6. Circle Pt. 2

Programmed trajectory: 1 m diameter circle
Actual trajectory: 100.5 cm
Observations: Acceptable stroke, with slight expansion of the radius.
Shape error: 0.5% lateral compression
Length error: +0.5 cm

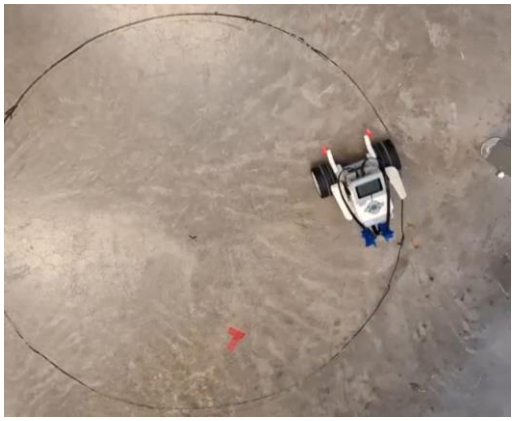

Illustration 7. Circle Pt. 3

Programmed trajectory: 1 m diameter circle
Actual trajectory: 97.2 cm

Observations: Acceptable stroke, with slight expansion of the radius.
Shape error: -2.8% lateral compression
Length error: 2.8 cm

GENERAL

Average travel: 98.925 cm
Average error: 1.075 cm (1.075%)
Conclusion: The robot traces circles reasonably accurately. Small variations are due to differences in ground friction and imbalance between motors. It is recommended to calibrate speeds or use gyroscopic sensor corrections.

C. Infinity Symbol

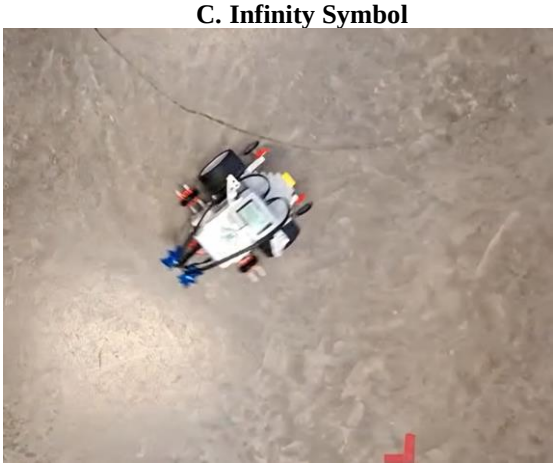

Illustration 8. Infinity Symbol Pt.1

Trajectory symmetry: uneven
Start vs. end center: 4 cm offset
Observations: One loop was larger due to the difference in speed between motors
Closure error: 4 cm

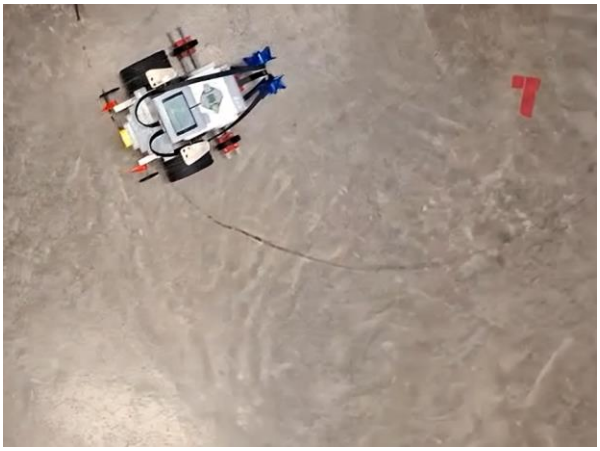

Illustration 9. Infinity Symbol Pt.2

Trajectory symmetry: Good
Start vs. end center: 1.5 cm offset
Observations: One loop was larger due to the difference in speed between motors
Closure error: 1.5 cm

Illustration 10. Infinity Symbol Pt.3

Trajectory symmetry: Wider left loop
Start vs. end center: 2.3 cm offset
Observations: The robot leaned to one side due to the weight of the wiring
Closure error: 2.3 cm

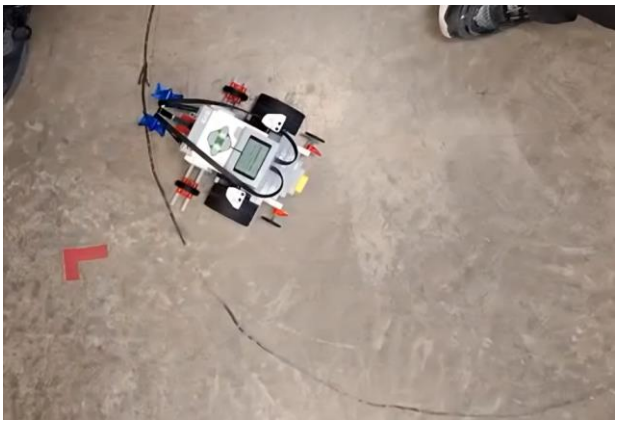

Illustration 11. Infinity Symbol Pt.4

Trajectory symmetry: Acceptable
Start vs. end center: 0.9 cm offset
Observations: Spin control improved after adjusting speed
Closure error: 0.9 cm

GENERAL

Average closure error: 2.175 cm
Observations: The infinity symbol is more prone to error due to its curved and continuous nature. The lack of sensors causes small errors to accumulate. Nevertheless, overall performance was adequate.

V. Conclusion

This practice allowed students to understand the fundamentals of differential drive robot behavior. It reinforced skills in motor configuration, programming with ROBOTC, and geometric path planning. The LEGO EV3 platform proved effective for quick prototyping in educational contexts.

VI. Reference

i. ROBOTC for LEGO Mindstorms 4.X
ii. ev3-rem-driving-base.pdf

VII. ANNEXES

Square CODE

```
#pragma config(Motor,  motorB, Left_Motor, tmotorEV3_Large, PIDControl, encoder)
#pragma config(Motor,  motorC, Right_Motor, tmotorEV3_Large, PIDControl, encoder)

// Definir constantes según el robot
#define WHEEL_DIAMETER 6.7   // Diámetro de las ruedas en cm
#define WHEEL_CIRCUMFERENCE (WHEEL_DIAMETER * 3.1416) // Circunferencia de la rueda
#define ROBOT_WIDTH 13.5    // Distancia entre las ruedas en cm

// Calcular cuántos grados de encoder se necesitan para avanzar 80 cm
#define DISTANCE_80CM (80 / WHEEL_CIRCUMFERENCE * 360)

// Calcular cuántos grados necesita girar cada rueda para un giro de 90°
#define TURN_90_DEG ((ROBOT_WIDTH * 3.1416 / 4) / WHEEL_CIRCUMFERENCE * 360)

task main() {
  for (int i = 0; i < 4; i++) {
    // Reiniciar encoders antes de avanzar
    resetMotorEncoder(Left_Motor);
    resetMotorEncoder(Right_Motor);

    // Avanzar 80 cm
    while (getMotorEncoder(Left_Motor) < DISTANCE_80CM) {
      motor[Left_Motor] = 50;
      motor[Right_Motor] = 50;
    }
    motor[Left_Motor] = 0;
    motor[Right_Motor] = 0;
    wait1Msec(500);  // Pausa para estabilidad

    // Reiniciar encoders antes de girar
    resetMotorEncoder(Left_Motor);
    resetMotorEncoder(Right_Motor);

    // Girar 90° en el lugar (rueda izquierda avanza, rueda derecha retrocede)
    while (getMotorEncoder(Left_Motor) < TURN_90_DEG*1.8) {
      motor[Left_Motor] = 70;
      motor[Right_Motor] = -70;
    }
    motor[Left_Motor] = 0;
    motor[Right_Motor] = 0;
    wait1Msec(500);  // Pausa para estabilidad
  }
}
```

Circle CODE

```
#pragma config(Motor,  motorB, Left_Motor, tmotorEV3_Large, PIDControl, encoder)
#pragma config(Motor,  motorC, Right_Motor, tmotorEV3_Large, PIDControl, encoder)

// Definir constantes del robot
#define WHEEL_DIAMETER 6.7   // Diámetro de las ruedas en cm
#define WHEEL_CIRCUMFERENCE (WHEEL_DIAMETER * 3.1416) // Circunferencia de la rueda
#define ROBOT_WIDTH 13.5    // Distancia entre las ruedas en cm

// Radio del círculo
#define CIRCLE_RADIUS 45.0   // 1 metro de diámetro => radio de 50 cm
#define CIRCLE_CIRCUMFERENCE (2 * 3.1416 * CIRCLE_RADIUS) // Longitud del círculo en cm

// Cálculo de distancias para las ruedas
#define INNER_RADIUS (CIRCLE_RADIUS - (ROBOT_WIDTH / 2)) // Radio de la rueda interna
#define OUTER_RADIUS (CIRCLE_RADIUS + (ROBOT_WIDTH / 2)) // Radio de la rueda externa

// Distancia que recorrerá cada rueda en el círculo
#define INNER_CIRCUMFERENCE (2 * 3.1416 * INNER_RADIUS)
#define OUTER_CIRCUMFERENCE (2 * 3.1416 * OUTER_RADIUS)

// Relación de velocidades entre la rueda interna y externa
#define INNER_SPEED (50 * (INNER_CIRCUMFERENCE / OUTER_CIRCUMFERENCE))
#define OUTER_SPEED 50  // Fijamos la rueda externa al 50% de potencia

task main() {
  // Reiniciar encoders antes de comenzar
  resetMotorEncoder(Left_Motor);
  resetMotorEncoder(Right_Motor);

  // Recorrer el círculo completo
  while (getMotorEncoder(Left_Motor) < (CIRCLE_CIRCUMFERENCE / WHEEL_CIRCUMFERENCE * 360)) {
    motor[Left_Motor] = INNER_SPEED;  // Velocidad de la rueda interna
    motor[Right_Motor] = OUTER_SPEED;  // Velocidad de la rueda externa
  }

  // Detener el robot al finalizar el círculo
  motor[Left_Motor] = 0;
  motor[Right_Motor] = 0;
}
```

Infinity Symbol CODE

```
#pragma config(Motor, motorB, Left_Motor, tmotorEV3_Large, PIDControl, encoder)
#pragma config(Motor, motorC, Right_Motor, tmotorEV3_Large, PIDControl, encoder)

// Parámetros del robot
#define WHEEL_DIAMETER 6.7     // Diámetro de las ruedas en cm
#define WHEEL_CIRCUMFERENCE (WHEEL_DIAMETER * 3.1416)
#define TRACK_WIDTH 13.5       // Distancia entre ruedas en cm

// Parámetros del símbolo de infinito
#define LOOP_RADIUS 30.0      // Radio de cada lazo en cm
#define SEMICIRCLE_DISTANCE (3.1416 * LOOP_RADIUS)  // Longitud de cada semicírculo en cm

// Radios interno y externo de cada curva
#define INNER_RADIUS (LOOP_RADIUS - (TRACK_WIDTH ))
#define OUTER_RADIUS (LOOP_RADIUS + (TRACK_WIDTH ))

// Distancias recorridas por cada rueda
#define INNER_CIRCUMFERENCE (3.1416 * INNER_RADIUS)
#define OUTER_CIRCUMFERENCE (3.1416 * OUTER_RADIUS)

// Relación de velocidades entre ruedas interna y externa
#define INNER_SPEED (60 * (INNER_CIRCUMFERENCE / OUTER_CIRCUMFERENCE))
#define OUTER_SPEED 60  // La rueda externa se mueve a velocidad base

void traceSemicircle(int direction, bool invertSides) {
  // Reiniciar encoders antes de iniciar la curva
  resetMotorEncoder(Left_Motor);
  resetMotorEncoder(Right_Motor);

  int targetEncoder = SEMICIRCLE_DISTANCE / WHEEL_CIRCUMFERENCE * 360;

  if (invertSides) {
    // Segundo lazo: Cambiamos las ruedas internas y externas
    while (abs(getMotorEncoder(Left_Motor)) < targetEncoder && abs(getMotorEncoder(Right_Motor)) < targetEncoder) {
      motor[Left_Motor] = OUTER_SPEED * direction;
      motor[Right_Motor] = INNER_SPEED * direction;
    }
  } else {
    // Primer lazo: Movemos normal (rueda derecha externa)
    while (abs(getMotorEncoder(Left_Motor)) < targetEncoder && abs(getMotorEncoder(Right_Motor)) < targetEncoder) {
      motor[Left_Motor] = (100 * (INNER_CIRCUMFERENCE / OUTER_CIRCUMFERENCE)) * direction;
      motor[Right_Motor] = 100 * direction;
    }
  }

  // Detener el robot al finalizar el semicírculo
  motor[Left_Motor] = 0;
  motor[Right_Motor] = 0;
  wait1Msec(500);
}

task main() {
  traceSemicircle(1, false);  // Primer lazo hacia la derecha
  traceSemicircle(1, true);   // Segundo lazo hacia la izquierda
}
```