Colorado School of Mines

Memo

To: Gary Nave, Chukwuebuka Okwor, and Ramon Chavez

From: Gordon Dina and Coleson Oliver

Team #: N-423

Date: 16 February 2023

Re: Lab #2: Open-Loop Control

Problem Statement:

In this lab, we were tasked with building and characterizing a robot capable of navigating a pre-planned course using open-loop control. The error in the robot's course navigation was measured at given setpoints using the Euclidean distance from the target point to a fixed point on the robot.

Methods:

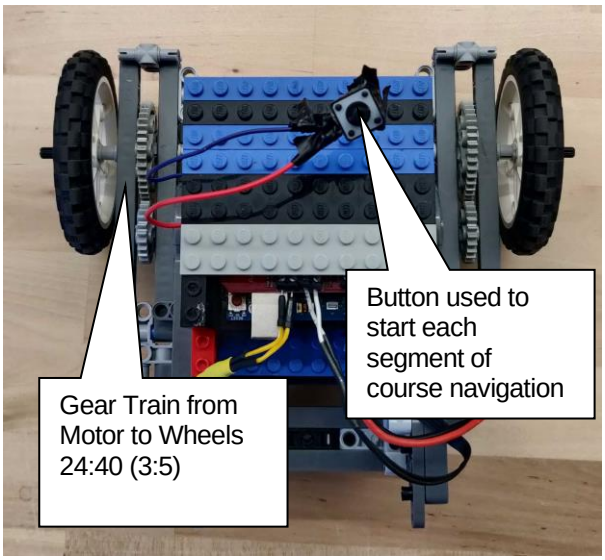


Figure 1: Robot Setup

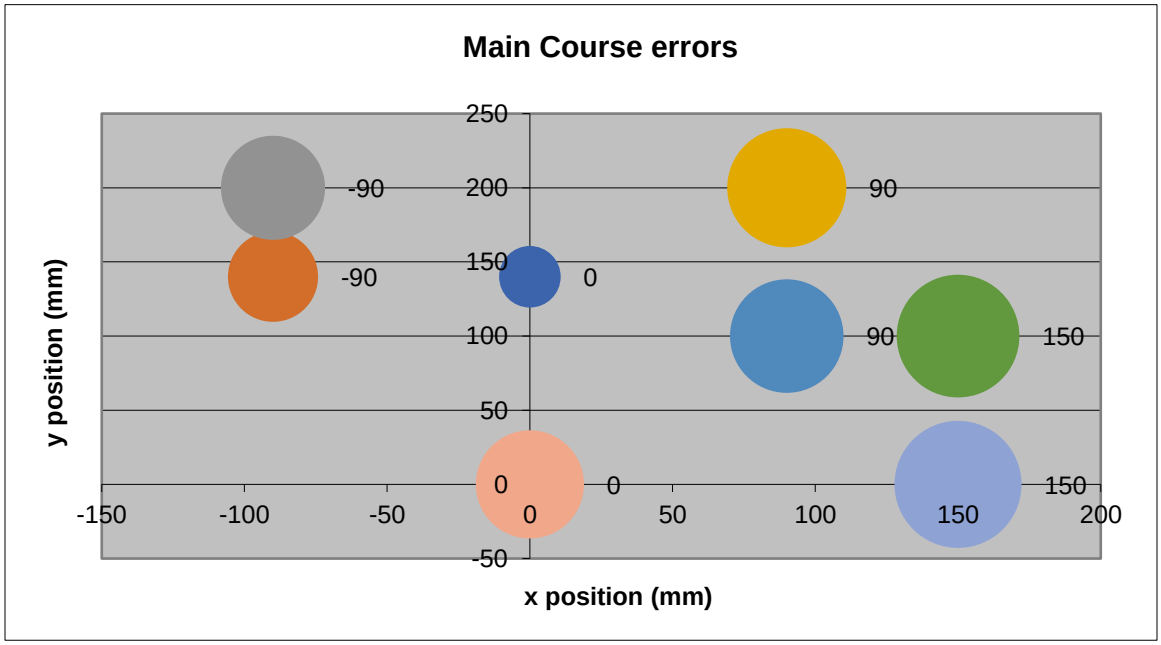
For this lab, we chose a two-wheeled robot with a skid platform at the rear to create a stable tripod. Initially we designed and implemented a caster wheel at the back but accomplishing a low friction axle on which the caster could rotate turned out to be infeasible, as any slight error in the initial angular position of the caster wheel caused the robot to turn unexpectedly, making it very difficult to characterize. When we pivoted to the skid platform, the functionality did not change significantly as the robot was still a tripod that turned about the center of the drive axles but did not suffer from the issue of uncommanded random turning.

The test procedure we used to measure the robot's error in navigating the course was to have it be hand placed as close to the start point designated by tape on the floor in the lab, and then at the press of the indicated button in Figure 1, traverse one segment of the course and then wait. During this pause, we measured the absolute error from the target point as designated again by the tape on the floor of the lab using a tape measure in centimeters. Once the button was again pressed, the robot would traverse the following segment of the course and this pattern was repeated until the course was complete. We did this measurement of the course 4 separate times and recorded the data in a spreadsheet.

The code we used to do the path following was mostly provided in the lab assignment but the basic functionality is captured in an array of commands, specifically a distance (in centimeters) followed by a direction to turn. The distance to travel was measured by first characterizing the robot in terms of the average distance traveled for a given PWM signal to both motors over a fixed time interval, and similarly in terms of the time taken for the same given PWM signal to both motors to traverse 90 degrees. Once these constants were measured for our robot, we could then use it to calculate the average milliseconds per 90 degrees and milliseconds per centimeter which were used in the path following controller to instruct the robot to traverse a certain linear or angular distance by modulating the amount of time for which the motors were run.

Results:

	Pt1	Pt2	Pt3	Pt4	Pt5	Pt6	Pt7	Pt8
Avg error mm	87.5	187.5	250	327.5	297.5	347.5	372.5	270
%	6.25%	20.83%	41.67%	18.19%	29.75%	57.92%	37.25%	18%
SD	25	39.48	73.94	94.30	38.62	69.46	87.70	119.16



Overall average error: 267.5

Overall SD: 93.38

Conclusions:

In terms of the results of the experiment, the errors we encountered in running the course were fairly significant. The errors also largely propagated through the experiment and increased over the breadth of the course. In our estimation, this is due to the difficulty of accurately characterizing the robot's performance, as using an average velocity as a benchmark for future performance ignores random error in each run of the course. As a result, some runs were much more accurate overall then others due to these random errors. Specifically referring to the robot's design, we believe that outside of the error inherent to the environment of the test, the subsystems of the robot performed as well as could be expected in an open-loop control design. In control theory, the effect of environmental errors on the robot system are mitigated by the use of sensors to perceive the robot's state and react to changes in said state to more accurately approach target points. Open-Loop control may be much easier to design and implement but it suffers significantly from inaccuracy where Closed-Loop controllers do not.

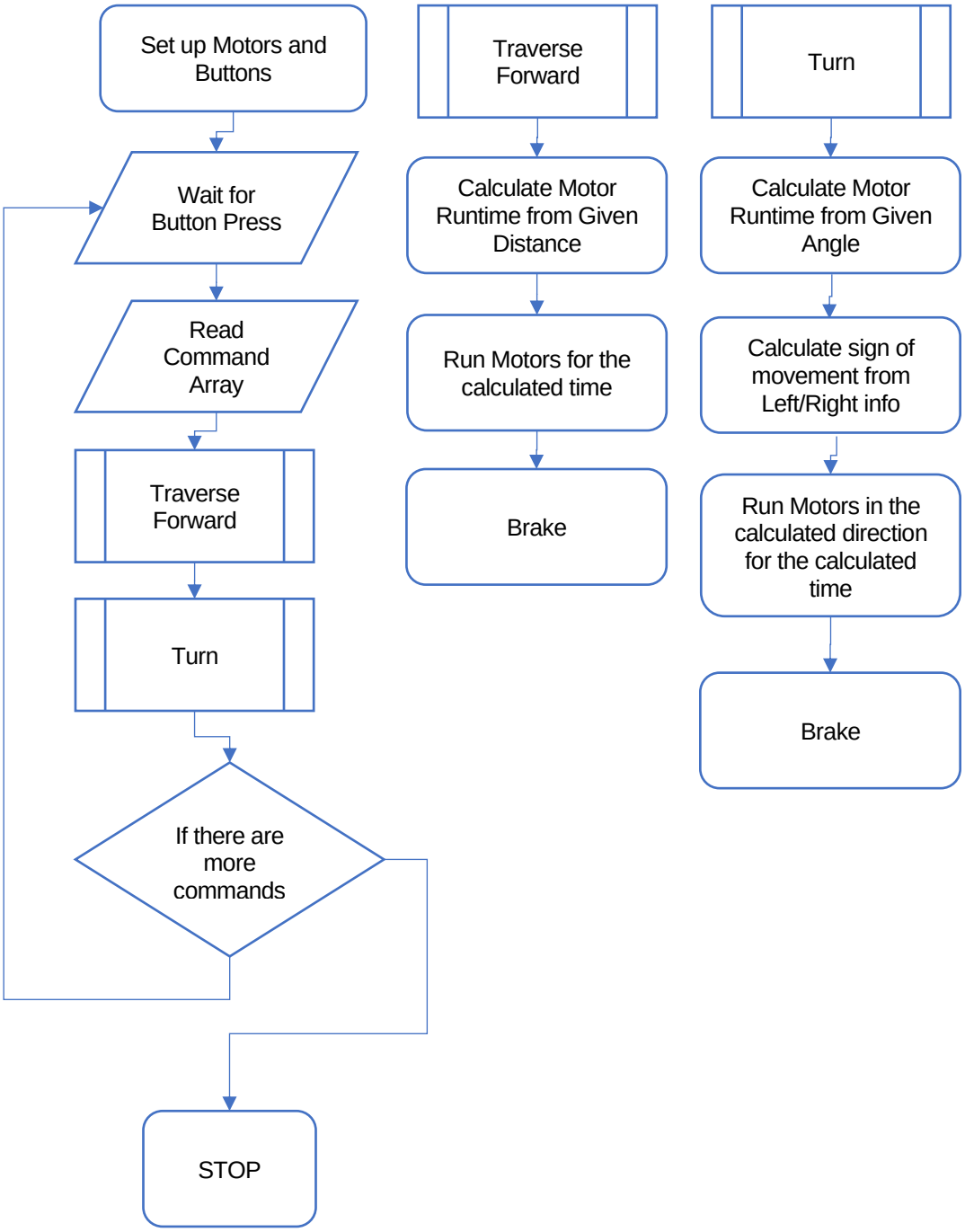
References

FTC Game Design Committee. "2016-17 FIRST Tech Challenge VELOCITY VORTEX Game Manual Part 2". For Inspiration and Recognition of Science and Technology (FIRST), Manchester NH. Retrieved 15 February 2013.

*NOTE for the above: the game manual is cited in place of the practical experience gathered while in the program, this was cleared and discussed with Dr. Nave.

Appendices:

Flowchart of logic:



Raw code used in the lab:

```
/* Dead Reckoning
  20220127 - gknave
  20220210 - coliver
*/

// If you have a kit with the moto shield, set this to true
// If you have the Dual H-Bridge controller w/o the shield, set to false
#define SHIELD true

// Defining these allows us to use letters in place of binary when
// controlling our motor(s)
#define A 0
#define B 1

//SHIELD Pin varables - cannot be changed
#define motorApwm 3
#define motorAdir 12
#define motorBpwm 11
#define motorBdir 13

// Dual H-Bridge motor controller pin variables - can be any 4 analog pins
(marked with ~ on your board)
// Only used if SHIELD is false.
#define IN1 9
#define IN2 10
#define IN3 5
#define IN4 6


// Preprocessor Definitions
#define FORWARD    0
#define LEFT       1
#define RIGHT     -1
#define pushButton 10

// the following converts centimeters into milliseconds as long datatype
#define milliSecondsPerCM    31  //Constant derived from Robot
Characterization
#define milliSecondsPer90Deg 505 //Constant derived from Robot
Characterization
#define PWM_A 205 //Constant derived from Robot Characterization
#define PWM_B 230 //Constant derived from Robot Characterization

// the itemized list of moves for the robot as a 1D array
```

```
// this setup assumes that all the turns are 90 degrees and that all
motions are pairs of drives and turns.
int moves[] = {140, LEFT, 90, RIGHT, 60, RIGHT, 180, RIGHT, 100, LEFT, 60,
RIGHT, 100, RIGHT, 150, LEFT};

void setup() {
  // set up the motor drive ports
  motor_setup();
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT_PULLUP);
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
  int i, dist, dir;
  long time;

  //This for loop steps (or iterates) through the array 'moves'
  for (i = 0; i < sizeof(moves) / 2; i = i + 2) {

    // Pause before each segment to allow setup / error measurement
    while(digitalRead(pushButton) == 1);

    delay(250);
    //Forward Leg of each step
    Serial.print("Step #:");
    Serial.println(i);
    dist = moves[i];
    Serial.print("Forward for");
    time = Forward(dist);
    Serial.print(time);
    Serial.println(" ms");
    delay(1000);

    //Turn Leg of each step
    Serial.print("Step #:");
    Serial.println(i + 1);
    dir = moves[i + 1];
    if (dir == LEFT) {
      time = Turn(90);
      Serial.print("turning LEFT ");
      Serial.print(time);
      Serial.println(" ms");
    }
```

```
    else {
      time = Turn(-90);
      Serial.println("turning RIGHT ");

      Serial.print(time);
      Serial.println(" ms");
    } // end of else motions conditional
    delay(1000);

  } // end of for loop
  Serial.println("That's All Folks!");
  delay(1000);
  exit(i);
} // the end

////////////////////////////////////////////
unsigned long Forward(int distance) {
  unsigned long t;
  t = distance * milliSecondsPerCM; //Time to keep motors on

  //To drive forward, motors go in the same direction
  run_motor(A, PWM_A); //change PWM to your calibrations
  run_motor(B, PWM_B); //change PWM to your calibrations
  delay(t);
  // software brake
  run_motor(A, -200);
  run_motor(B, -200);
  delay(50);
  run_motor(A, 0);
  run_motor(B, 0);
  return (t);
}

////////////////////////////////////////////
unsigned long Turn(int degrees) {
  unsigned long t;
  int sign = degrees / abs(degrees); //Find if left or right
  t = (abs(degrees) / 90) * milliSecondsPer90Deg; //Time to keep motors on

  // The run motor command takes in a PWM value from -255 (full reverse)
to 255 (full forward)
  /* Using the Forward function as a guide,
   * Write commands in this Turn function to power the
   * motors in opposite directions for the calculated time
   * and then shut off
```

```
   */

  //To drive forward, motors go in the same direction
  run_motor(A, sign * PWM_A); //change PWM to your calibrations
  run_motor(B, -sign * PWM_B); //change PWM to your calibrations
  delay(t);
  // software brake
  run_motor(A, -sign * 200);
  run_motor(B, sign * 200);
  delay(50);
  run_motor(A, 0);
  run_motor(B, 0);
  return (t);
}
```