**MEGN 441**
**Christian Prather**

# Lab 2 Open Loop Control

**Problem Statement:**

A robotic platform that is built to operate in the real world is only as good as its ability to sense the real word. Variations in terrain and conditions often lead to discrepancies in what a bot is programmed to do and what it actually does. This lab aims to discover how optimal an open loop robot can perform in real world conditions.

**Methods:**

a)

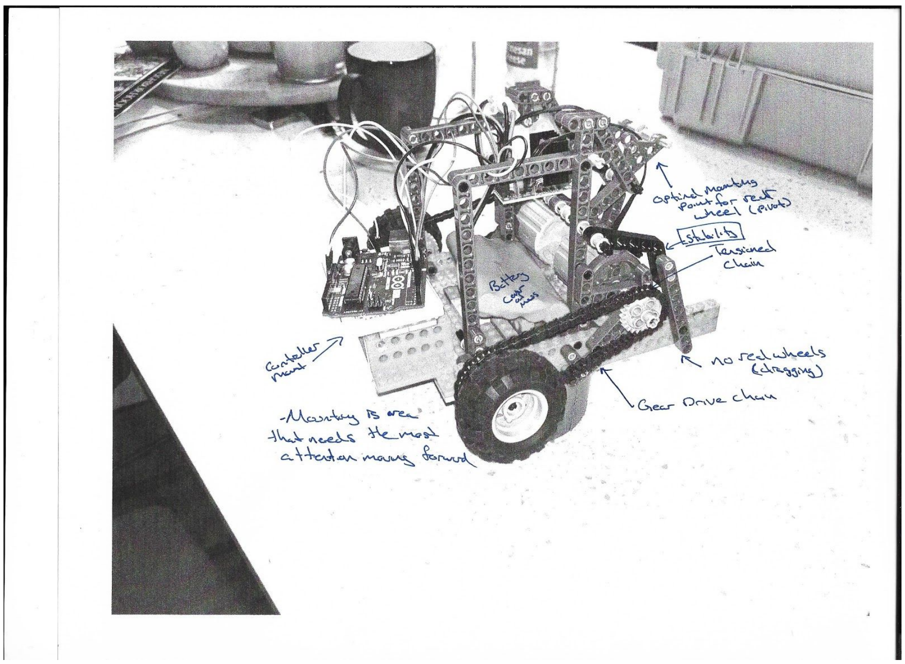



Fig 1 (Photo of Robot)

b)

There were many considerations that had to be taken into account when choosing a mechanical configuration. I focused my attention on exploring two primary key sections of my robot to explore different options: drive method and rear wheel layout. I knew from the results of lab 1 that the Lego gears do not offer great resilience from slipping unless well built, as I was not confident in my ability to design a well constructed Lego gearbox I choose to use a chain as contrary to common belief when done in a chain sprocket configuration and properly tensioned are far less susceptible to slipping than direct drive connections. This did present a question as to if I would need a method of dynamically tensioning the chain. I explored the ability to simply use an idler sprocket to wrap around but as it was not spring tensioned this had no more dynamic tensioning effect than the direct two gear configurations have. I then looked at, and constructed a spring tension system utilizing a shock found in the lego sets. I was able to construct a test that worked in applying the proper force for tension but I could not find an appropriate mounting method for the shock and soon scrapped the idea, opting for a careful static tensioning. This is likely a great cause in my variance. I spent a significant amount of time looking at my rear wheel configurations. I knew that I wanted to do a dragged third wheel that was free to pivot and was not driven. I constructed such a mechanism however once again mounting to my base became problematic and after about an hour I chose not to spend more time on it. I then explored a simple 2 free wheel configuration that would behave well when driving linearly but as the friction was too great when turning I realized that I would be better off minimizing friction and doing a simple drag system, which is what I used for the remainder of this lab. In summary I had the majority of my issues with using Legos as the platform, I was not able to construct a sufficient base to give me the freedom I needed in the mechanisms I wanted to attach and in future labs this may require adjusting.

c)

For testing I first had to go through some experimentation to understand the dynamics of my platform. I began by utilizing the inintal_testing code provided with some minor adjustments to get my motor calibrated for driving in a linear path on my given floor. I knew from Lab 1 my motors have different characteristics and that a single PWM value for both would not be sufficient. I quickly found that motor A needed to be held back from motor B by about 10%. I then used a simple trial method of finding how long it took my robot to travel 50 cm by adjusting a delay driven program. Changing the delay accordingly to over and under shoot. Once these values were set I was able to use them for the primary function code. Testing then became straight forward, I began by selecting a single measurement point on my robot. This was the center front and proved to be an easy repeatable measurement point. I then selected a starting point to base all future measurements on the track with. I chose the back line of the tape used to account for the thickness of the tape. I then let the robot run one section at a time including its respective turn and measured the resulting point from the new relative starting point on the track section. I then measured drift relative to x, y coordinates that are relative to the bot as in positive y is forward and positive x is right.

d)

Most of the code was provided and used as is, however I did do some simple adjustments for convenience and to work with my motor configurations. I set up two turning functions to allow for simple calls when running. The run time for both linear and turning was calculated from a given distance and travel time found in the calibration step. The primary flow of the code is that it waits for button input to begin running the next step in an array representation of the path. If it's linear then delay time is calculated and the motor run code is executed. If a turn it is determined whether it's a left or right turn. This primary function then calls its respective turn function. This is then repeated for each step on the path delaying in execution until user interaction through the push button.

**Results**:

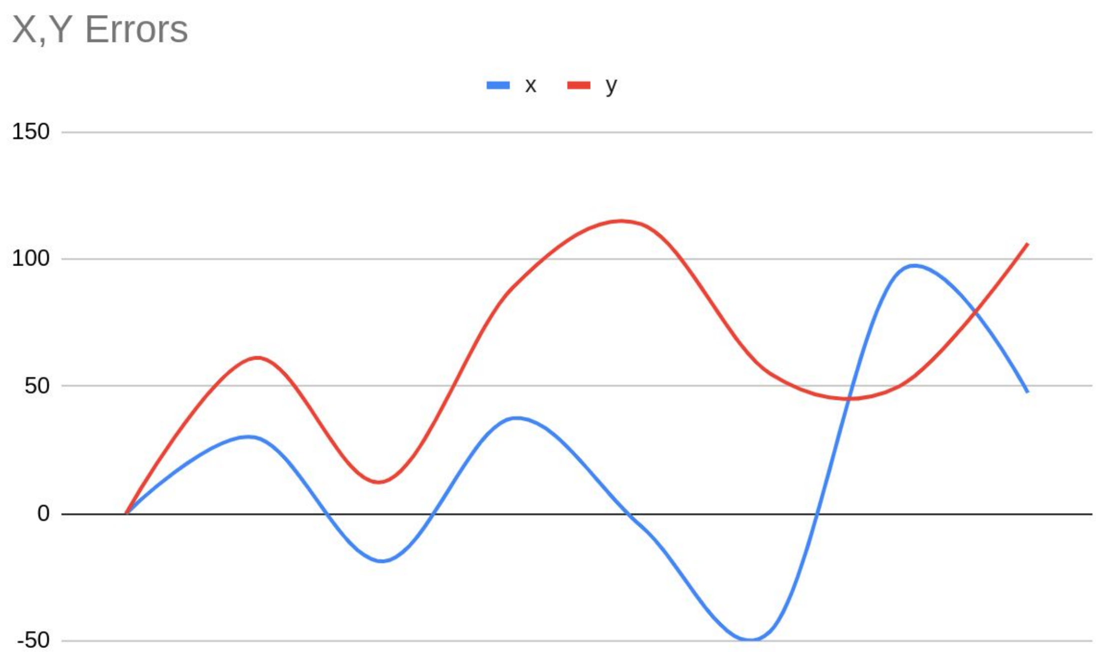
a)

Average Error

| Average Error | | | |
|---|---|---|---|
| X mm | Y mm | % X | % Y |
| 0 | 0 | 0 | 0 |
| 30 | 61.25 | 3.333333333 | 6.805555556 |
| -18.75 | 12.5 | -3.125 | 2.083333333 |
| 37.5 | 88.75 | 2.083333333 | 4.930555556 |
| -5 | 113.75 | -0.5 | 11.375 |
| -46.25 | 55 | -7.708333333 | 9.166666667 |
| 95 | 50 | 9.5 | 5 |
| 47.5 | 106.25 | 3.166666667 | 7.083333333 |

Standard Deviation

| SD: | |
|---|---|
| x | y |
| 18.70828693 | 7.071067812 |
| 18.70828693 | 25.58686186 |
| 24.07670036 | 25.86020108 |
| 43.80353867 | 73.34635301 |
| 43.87482194 | 59.51627929 |
| 66.08469944 | 66.8954408 |
| 18.02775638 | 47.4341649 |
| 29.47456531 | 103.6445247 |




| | | | | |
|---|---|---|---|---|
| Averages average | | | Averaged SD | |
| x | y | | x | y |
| 17.5 | 60.9375 | | 44.16385885 | 41.11042403 |

**Conclusion**:

a)

The results indicate that there is a problem with consistency across time of the performance of my robot. I can see from the average errors that while my platform is capable of doing fairly consistent in run per run basis however it is not an accurate robot. Elements that contributed to the error are an inconsistent floor surface, my living room while wood is not even and this definitely resulted in enough change in terrain to affect precision. Other sources of error include the tilt of my drive wheels as their axis proved to not be supported well enough to handle the chain tension. Locations that produced the largest error in terms of percentage of distance traveled was from point 5-6, this area seems to have the worst ability to stay straight and travel the appropriate distance out of all sections.

b)

The positives of the robot design are in its chain drive as this proved to be very low slip, so much so that I need to now be careful about stalling the motor under load as there is no way to relieve the force. There are also benefits in its openness as a platform with minimalist components I have a great deal of space for mounting. The code is also very strong having been mostly provided for us. Problems with the bot however include the rigidity of the drive axis, and the rear wheel drives, both vary too much during drive and offer harder conditions for repeatability in testing. I had a significantly harder time in construction of this robot using Legos than I would have simply modeling and printing it. I am not a fan of the limitations it has imposed that are not indicative of my robotics ability but rather of my ability to use a toy.

c)

The advantages of open loop control are that it is simple, there is no need for advanced sensors or complexities in code such as real time calculations or sensor data fusing.

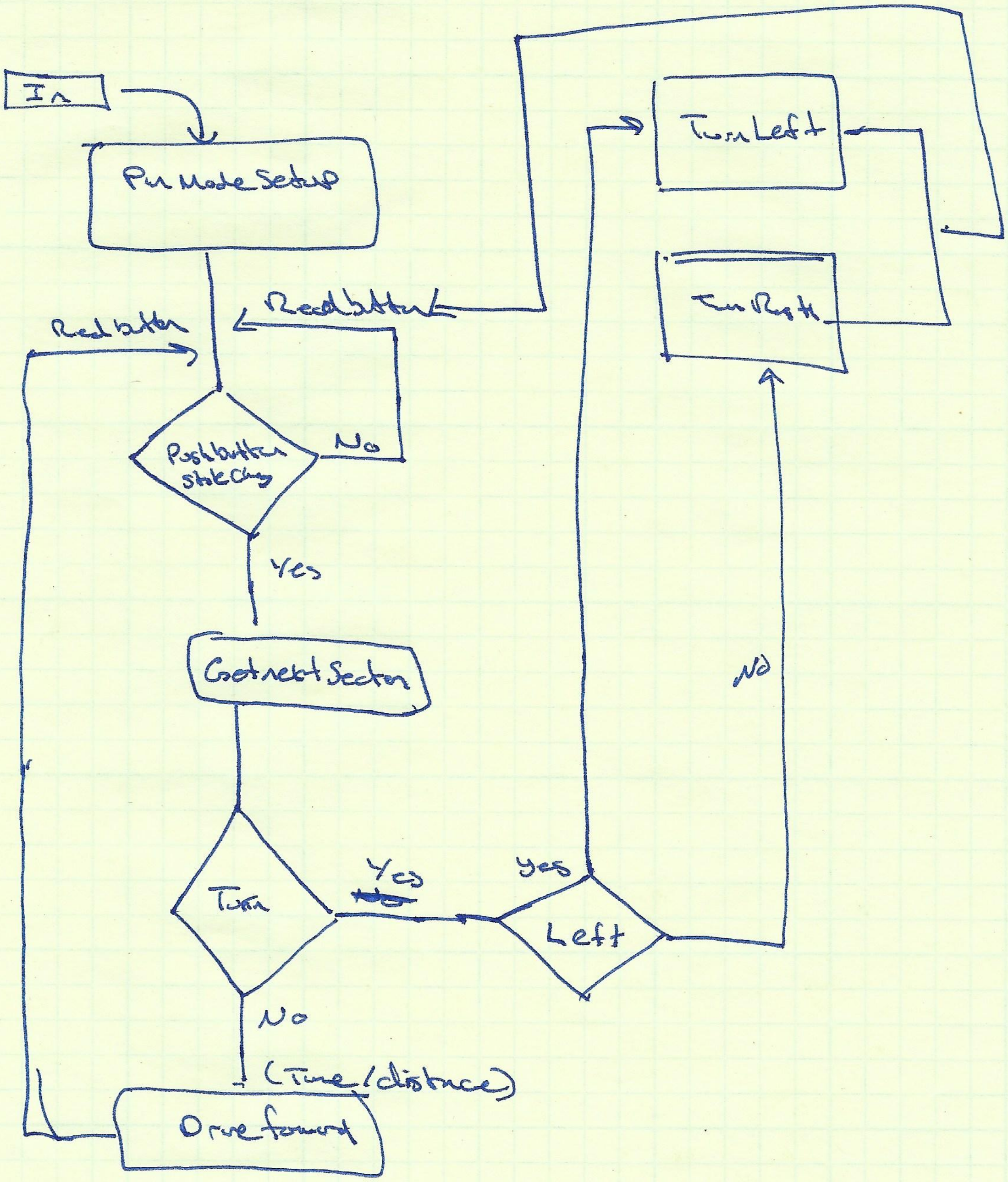
Reference:

N/A

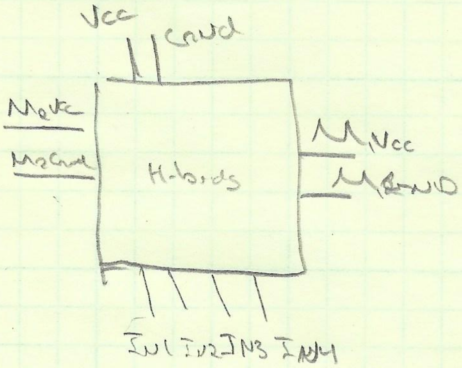
Appendix:

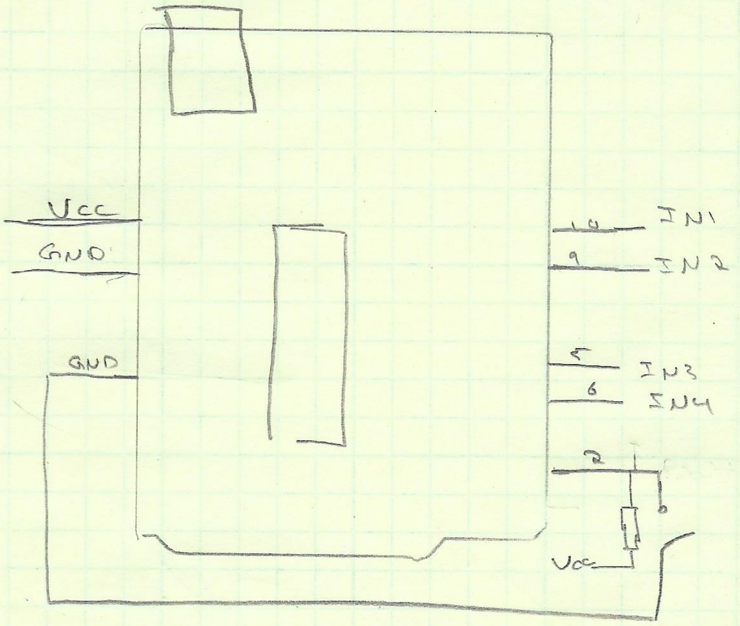
See attached pages

Flow diagram:

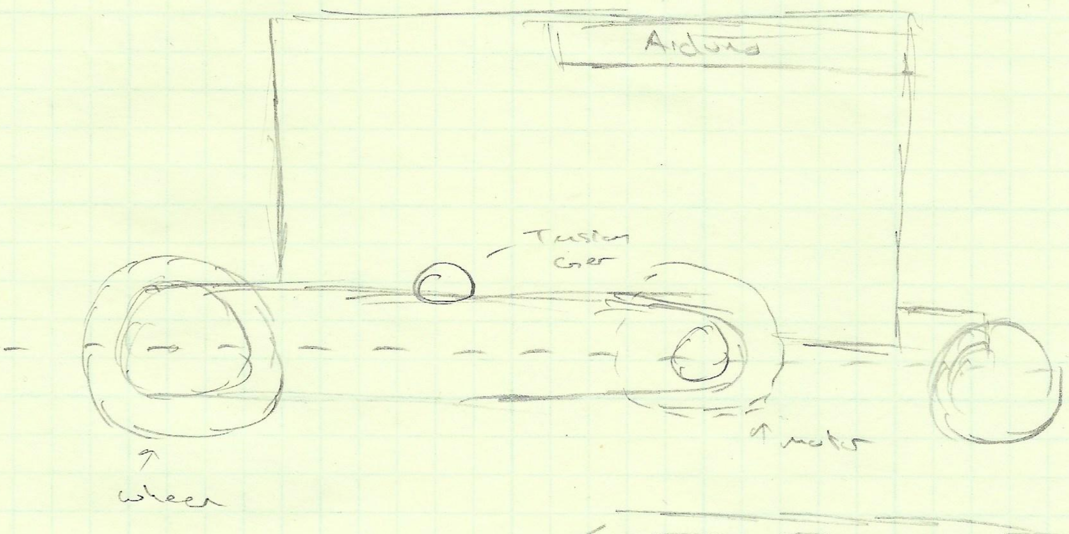


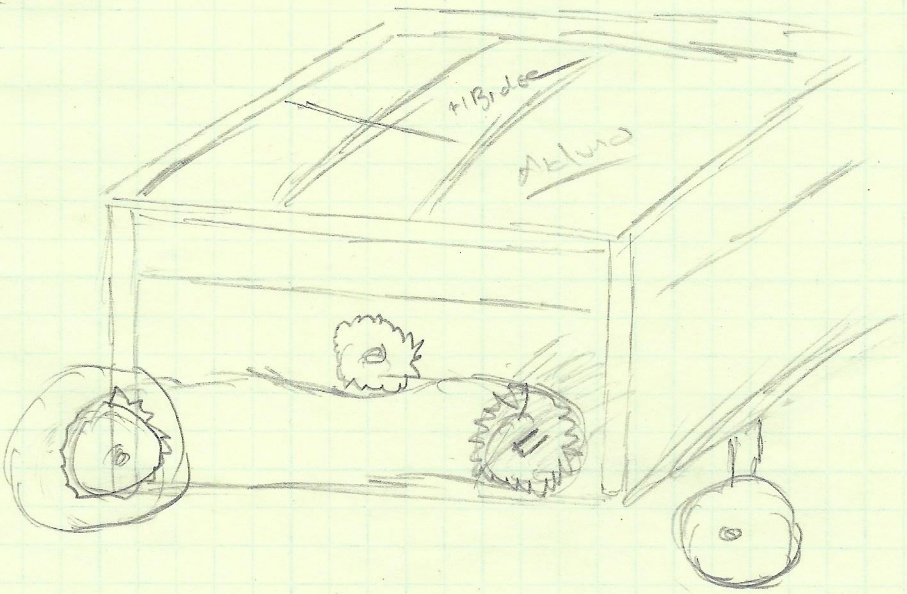
Electrical:


Vcc
Gnd
M2Vcc
M2Gnd
H-bridge
M1Vcc
M1GND
IN1 IN2 IN3 IN4


Vcc
GND
GND
10
IN1
9
IN2
5
IN3
6
IN4
2
Vcc

Mechanical



```
/* Initial Robot Testing
 *  20200117
 *  Program to get your robot's unique driving constants
 */

/* Program TODO LIST
  1) Write code for the button pause functionality
  2) Experiment with the wire connections to your motors until
     positive motor commands cause both motors to drive forward.
     Switch the two wires of a motor to flip the direction it spins
  3) Find the relative power settings to drive completely straight A 200 B 210
     What pwms do you put to each motor so they spin at the same speed?
  4) Find the time in ms to drive 1 cm 22ms/ cm
     Change this program to drive straight for a set amount of time. Measure
     the distance traveled, and divide. Use the average of multiple trials.
  5) Find the time in ms to turn 90 degrees
     Change this program to spin in place for a set amount of time. Measure
     the total degrees traveled, and divide. Use the average of multiple trials. 273 ms
*/

#define pushButton 2
// 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 false

// 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
#define motorApwm 3
#define motorAdir 12
#define motorBpwm 11
#define motorBdir 13

//Driver Pin variable
#define IN1 9
#define IN2 10
#define IN3 5
#define IN4 6
bool RUN = false;

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

void loop()
{

    /* Write code here to make the program pause until
     * the button has been pressed. Remember that getting a value
```

```
     * from the board is digitalRead(pinNumber) and if statements
     * need a double equals sign (==) for comparisons
     */
    if (!digitalRead(pushButton))
    {
        /*
    * Change your wiring so that positive numbers drive forward
    * After wiring has been fixed:
    * To drive forward, motors should have positive commands
    * To turn Left, the left motor should have a negative number,
    *              the right motor should have a positive number
    * To turn Right, the left motor should have a positive number,
    *               the right motor should have a negative number
    */

        // Run once to calibrate the motors driving straight
        for (int i = 0; i < 1; i++)
        {
            // Give me time to get my hand away from the robot
            delay(1000);
            // Found Motor A needs to be lower
            run_motor(A, 185); //set this to a number between -255 and 255
            run_motor(B, 200); //set this to a number between -255 and 255
            delay(2700);       //set this to a time in ms for the motors to run
            run_motor(A, 0);   //motors stop
            run_motor(B, 0);
            delay(750);
        }

        // RIGHT
        {
            run_motor(A, 200);  //set this to a number between -255 and 255
            run_motor(B, -200); //set this to a number between -255 and 255
            delay(900);         //set this to a time in ms for the motors to run
            run_motor(A, 0);    //motors stop
            run_motor(B, 0);
        }
        delay(2000);

        // LEFT
        {
            run_motor(A, -200); //set this to a number between -255 and 255
            run_motor(B, 200);  //set this to a number between -255 and 255
            delay(900);         //set this to a time in ms for the motors to run
            run_motor(A, 0);    //motors stop
            run_motor(B, 0);
        }
    }
}
```

```
/* Dead Reckoning
  20200117
  Program to get the ardbot to drive a predefined path
  jsteele, mshapiro, klarsen
*/

// Complete Tasks in Initial_Robot_Testing.ino first!!

/* Program TODO LIST
  1) Change the milliSecondsPerCM constant to the value you found in testing
  2) Change the milliSecondsPer90Deg constant to the value you found in testing
  3) Change the PWM of the forward function to the values you found in testing
  4) Write code for the button pause functionality
  5) Write your own turn function

  Note: type // to make a single line comment
        Comments are for the future you to understand how you wrote your code
*/

// Preprocessor Definitions

// 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 false

// 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
#define motorApwm 3
#define motorAdir 12
#define motorBpwm 11
#define motorBdir 13

//Driver Pin variable
#define IN1 9
#define IN2 10
#define IN3 5
#define IN4 6

#define FORWARD 0
#define LEFT 1
#define RIGHT -1
#define pushButton 2
#define A 1
#define B 2
#define pwmA 3
#define dirA 12
#define pwmB 11
#define dirB 13

// PWM values for the motors
#define motorA_PWM 185
#define motorB_PWM 200

```

```
#define SHIELD 0

// the following converts centimeters into milliseconds as long datatype
#define milliSecondsPerCM 54     //CHANGE THIS ACCORDING TO YOUR BOT
#define milliSecondsPer90Deg 900 //CHANGE THIS ACCORDNING TO YOUR BOT

// 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, RIGHT};

// RIGHT param t is delay time calculated from dist and speed ratio
void turnRight(int t)
{
  run_motor(A, -motorA_PWM); //set this to a number between -255 and 255
  run_motor(B, motorB_PWM);  //set this to a number between -255 and 255
  delay(t);                  //set this to a time in ms for the motors to run
  run_motor(A, 0);           //motors stop
  run_motor(B, 0);
}

// LEFT param t is delay time calculated from dist and speed ratio
void turnLeft(int t)
{
  run_motor(A, motorA_PWM);  //set this to a number between -255 and 255
  run_motor(B, -motorA_PWM); //set this to a number between -255 and 255
  delay(t);                  //set this to a time in ms for the motors to run
  run_motor(A, 0);           //motors stop
  run_motor(B, 0);
}

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

void loop()
{
  int i, dist, dir;
  long time;
  while (digitalRead(pushButton) == 1)
    ;
  while (digitalRead(pushButton) == 0)
    ;
  //This for loop steps (or iterates) through the array 'moves'
  for (i = 0; i < sizeof(moves) / 2; i = i + 2)
  {

    while (digitalRead(pushButton))
    {
      // Do nothing but wait
```

```
      //Serial.println("Waiting");
    }

    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, motorA_PWM); //change PWM to your calibrations
  run_motor(B, motorB_PWM); //change PWM to your calibrations
  delay(t);
  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

  if (sign == -1)
  {
    turnLeft(t);
  }
  else
  {
    turnRight(t);
  }

  return (t);
}
```

```
void motor_setup()
{
  if (SHIELD)
  { //if you're using the motoshield
    //define Sheild pins as outputs
    pinMode(motorApwm, OUTPUT);
    pinMode(motorAdir, OUTPUT);
    pinMode(motorBpwm, OUTPUT);
    pinMode(motorBdir, OUTPUT);
    // initialize all pins to zero
    digitalWrite(motorApwm, 0);
    digitalWrite(motorAdir, 0);
    digitalWrite(motorBpwm, 0);
    digitalWrite(motorBdir, 0);
  }
  else
  { // if using dual motor driver
    // define driver pins as outputs
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    // initialize all pins to zero
    digitalWrite(IN1, 0);
    digitalWrite(IN2, 0);
    digitalWrite(IN3, 0);
    digitalWrite(IN4, 0);
  } // end if
  return;
} // end function

// int motor is the defined A or B
// pwm = the power cycle you want to use
void run_motor(int motor, int pwm)
{
  int dir = (pwm / abs(pwm)) > 0; // returns if direction is forward (1) or reverse (0)
  pwm = abs(pwm);                 // only positive values can be sent to the motor

  if (SHIELD)
  { // if using motor shield
    switch (motor)
    {                               // find which motor to control
    case A:                         // if A, write A pins
      digitalWrite(motorAdir, dir); // dir is either 1 (forward) or 0 (reverse)
      analogWrite(motorApwm, pwm);  // pwm is an analog value 0-255
      break;                        // end case A
    case B:                         // if B, write B pins
      digitalWrite(motorBdir, dir); // dir is either 1 (forward) or 0 (reverse)
      analogWrite(motorBpwm, pwm);  // pwm is an analog value 0-255
      break;                        // end case A
    }                               // end switch statement
  }                                 // end if
  else
  { // if using dual motor drivers
    switch (motor)
    {       // find which motor to control
```

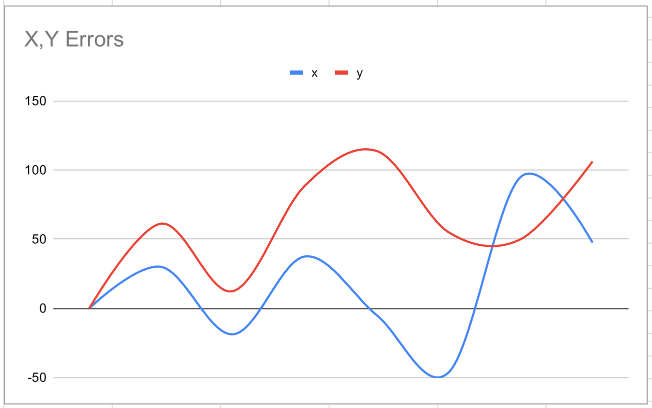
```
    case A: // if A, write A pins
      if (dir)
      {                         // If dir is forward
        analogWrite(IN1, pwm);  // IN1 is the forward pwm pin
        digitalWrite(IN2, LOW); // IN2 is low
      }
      else
      {
        digitalWrite(IN1, LOW); // IN1 is low
        analogWrite(IN2, pwm);  // IN2 is the reverse pwm pin
      }                         // end if
      break;                    // end case A
    case B:                     // if B, write B pins
      if (dir)
      {                         // if dir is forward
        analogWrite(IN3, pwm);  // IN3 is the forward pwm pin
        digitalWrite(IN4, LOW); // IN4 is low
      }
      else
      {
        digitalWrite(IN3, LOW); //IN3 is low
        analogWrite(IN4, pwm);  // IN4 is the reverse pwm pin
      }                         // end if
      break;                    // end case B
    }                           // end switch case
  }                             //end if
  return;
} // end function

```

| Trial 1 | | Trial 2 | | Trial 3 | | Trial 4 | |
|---|---|---|---|---|---|---|---|
| x | y | x | y | x | y | x | y |
| 20 | 0 | 10 | 0 | 0 | 10 | -30 | -10 |
| 30 | 100 | 50 | 65 | 0 | 30 | 40 | 50 |
| -40 | 0 | 0 | 20 | -45 | 50 | 10 | -20 |
| 20 | 200 | -20 | 100 | 100 | 55 | 50 | 0 |
| 30 | 170 | 40 | 160 | -20 | 20 | -70 | 105 |
| -100 | 140 | -110 | 100 | 55 | -20 | -30 | 0 |
| 120 | 40 | 100 | 130 | 70 | 20 | 90 | 10 |
| 0 | 200 | 80 | 200 | 60 | 75 | 50 | -50 |

| Average Error | | | | Distances: |
|---|---|---|---|---|
| x | y | % X | % Y | 1400 |
| 0 | 0 | 0 | 0 | 900 |
| 30 | 61.25 | 3.333333333 | 6.805555556 | 600 |
| -18.75 | 12.5 | -3.125 | 2.083333333 | 1800 |
| 37.5 | 88.75 | 2.083333333 | 4.930555556 | 1000 |
| -5 | 113.75 | -0.5 | 11.375 | 600 |
| -46.25 | 55 | -7.708333333 | 9.166666667 | 1000 |
| 95 | 50 | 9.5 | 5 | 1500 |
| 47.5 | 106.25 | 3.166666667 | 7.083333333 | |

| Mean differences Run 1 | | Run 2 | | Run 3 | | Run 4 | |
|---|---|---|---|---|---|---|---|
| x | y | x | y | x | y | x | y |
| 400 | 0 | 100 | 0 | 0 | 100 | 900 | 100 |
| 0 | 1501.5625 | 400 | 14.0625 | 900 | 976.5625 | 100 | 126.5625 |
| 451.5625 | 156.25 | 351.5625 | 56.25 | 689.0625 | 1406.25 | 826.5625 | 1056.25 |
| 306.25 | 12376.5625 | 3306.25 | 126.5625 | 3906.25 | 1139.0625 | 156.25 | 7876.5625 |
| 1225 | 3164.0625 | 2025 | 2139.0625 | 225 | 8789.0625 | 4225 | 76.5625 |
| 2889.0625 | 7225 | 4064.0625 | 2025 | 10251.5625 | 5625 | 264.0625 | 3025 |
| 625 | 100 | 25 | 6400 | 625 | 900 | 25 | 1600 |
| 2256.25 | 8789.0625 | 1056.25 | 8789.0625 | 156.25 | 976.5625 | 6.25 | 24414.0625 |




| SD: | | Averages average | | Averaged SD | |
|---|---|---|---|---|---|
| x | y | x | y | x | y |
| 18.70828693 | 7.071067812 | 17.5 | 60.9375 | 44.16385885 | 41.11042403 |
| 18.70828693 | 25.58686186 | | | | |
| 24.07670036 | 25.86020108 | | | | |
| 43.80353867 | 73.34635301 | | | | |
| 43.87482194 | 59.51627929 | | | | |
| 66.08469944 | 66.8954408 | | | | |
| 18.02775638 | 47.4341649 | | | | |
| 29.47456531 | 103.6445247 | | | | |

To: TA
From: Christian Prather

Team #: NA
Date: Sep 18
Re: Lab 2 Memo

Mechanical: A square chassis will be constructed with two motors each geared to up utilizing a chain mechanism to avoid gear slipping as I saw in the motor characteristic lab. The center point will be used as the measurement point. This will allow for me to measure the movement using “Zero point turning”. Turning will be accomplished with the tank style turning (motors turned in opposite) direction.

Electrical:
See attached PDF

Flow Chart:
See attached PDF