Memo

To:	Gary Nave, Chukwuebuka Okwor, and Ramon Chavez
From:	Gordon Dina and Coleson Oliver
Team #:	N-423
Date:	04 May 2023
Re:	Lab #5: Student Design Showcase

Problem Statement

This lab was an open design project on a problem of our choosing. We chose to extend the car platform developed in the previous labs to create a drawing robot that can maneuver a pen on a piece of paper to draw specific shapes. This involved the use of new mechanical, electrical, and software features to create and actuate a carriage for the pen to move up and down to be placed on or removed from the paper.

Methods

Since the new features of the robot for this lab were the pen carriage and its associated actuation, we focused on the physical construction and integration of the carriage first. For the new mechanical feature, we have a four-bar linkage attached to the carriage that holds the pen and allows semi-linear actuation (the pen tip moves through an arc technically, but the pen is constantly held normal to the paper). For the new electrical and software features, we used a servo motor to actuate the carriage and linkage which required understanding how a servo motor operates and what its strengths and limitations are. Most servo motors are restricted to about 180 degrees of travel but have high position accuracy and are very small compared to our drive motors. Also since the pen only moves through a small arc between the up and down positions, the limited angular range of the servo motor was not a problem. The servo motor is mechanically attached to the linkage with a small length of single core steel wire which we chose for its stiffness, allowing it to securely hold the pen to the paper, but also the ease of forming it to the shape required. Electrically, the servo has 3 pins: 5V input, Ground, and signal in the form of PWM. On the software side we used the included Servo.h library that is packaged with Arduino to drive the servo motor, and empirical testing to determine the appropriate angles for pen up and pen down. For further clarity, there is a photo of the mechanical setup in the appendices as well.

Results

Due to the simplicity of our design no data was collected. When the robot ran it was able to perform the correct driving and turning actions for drawing the shape given to it, as well the servo correctly positioned the pen when driving, turning, and completing/starting an action.

Conclusions

Overall, our robot performed well for this lab. The biggest surprise for us was the fact that the pen was able to stay in place when the robot was driving and turning, as building a pen holder out of Legos was the most challenging part. In the end our robot drove and turned relative well compared to previous labs, the servos itself performed the correct angles allowing the pen to make contact with the paper. While the bot did perform expected results when driving on the table, we did not consider the friction of the paper which in turn messed with the driving and turning of the robot. And while the servo did allow the pen to make contact with the paper it did not have enough pressure to actual draw the shape when moving. If we had more time on this lab, we would fix the issue of the pen not having enough pressure to draw the shape, as well adding more shapes or other drawing options.

References

Benne de Bakker. "How to control servo motors with Arduino". Makerguides. Retrieved 01 May 2023

Appendices

Pen Carriage Picture:

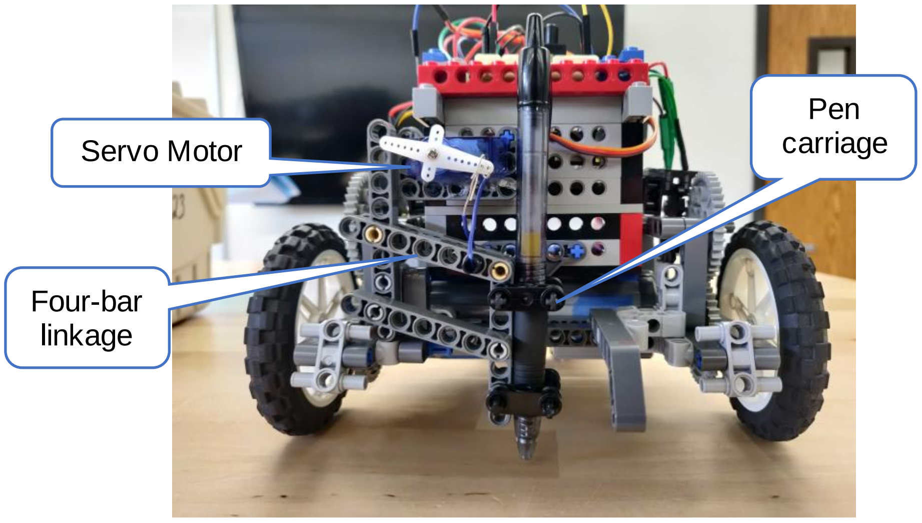
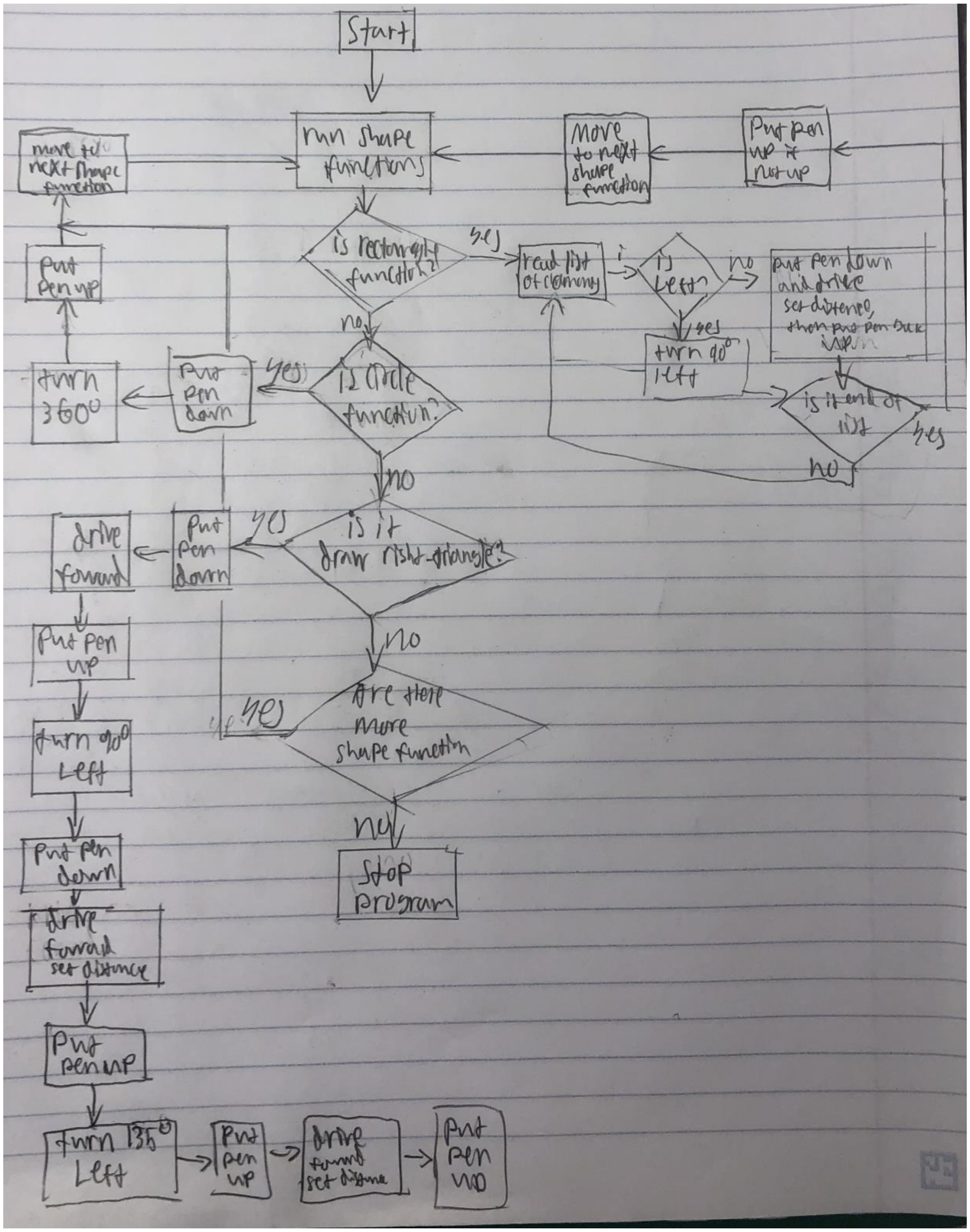


Figure 1: A front view of the carriage and linkage to hold and actuate the pen

Test/Calibration Data: Not much testing data is applicable for our project, but in order to use the servo, we had to determine what the appropriate angles for pen up and pen down were, given our mechanical configuration. We accomplished this by creating a separate servo testing script that moved the servo between defined angles at the press of a button and dialed in the appropriate values iteratively.

Flow Chart:


Start
run shape functions
move to next shape function
Put pen up
turn 360°
Put pen down
yes
is rectangle function?
yes
no
is circle function?
no
read list of commands
is Left?
no
put pen down and drive set distance, then put pen back up
yes
turn 90° left
is it end of list
yes
no
Put pen up if not up
Move to next shape function
is it draw right-triangle?
yes
Put pen down
drive forward
Put pen up
turn 90° Left
Put pen down
drive forward set distance
Put pen up
turn 135° Left
Put pen up
drive forward set distance
Put pen up
no
Are there more shape function
yes
no
Stop program

Arduino Code:

```
/* Lab 5: student design showcase
   ms 20200926
   gkn 20230309
   co 20230414
*/

#include <PinChangeInterrupt.h>
#include <HCSR04.h> // If using any Ultrasonic Distance Sensors
#include <Servo.h>

// Copy constants and definitions from Lab 3

// Motor Pin & Variable definitions
#define SHIELD true

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

//Driver Pin variable - any 4 analog pins (marked with ~ on your board)
#define IN1 9
#define IN2 10
#define IN3 5
#define IN4 6

#define A 0
#define B 1

// Move array definitions
#define FORWARD             5
#define LEFT                1.0
#define RIGHT              -1.0

// Pin definitions
#define pushButton 10
#define EncoderMotorLeft  8
#define EncoderMotorRight 7

volatile unsigned int leftEncoderCount = 0;
volatile unsigned int rightEncoderCount = 0;
```

```
// Encoder and hardware configuration constants
#define EncoderCountsPerRev 120.0
#define DistancePerRev      25.5 // TODO: Measure wheel diameter (8.1
cm)         (Calculated Value 25.44)
#define DegreesPerRev       165 // TODO: Measure distance between wheels (18.8
cm) (Calculated Value 155.58)

// Proportional Control constants
#define kP_A 1.8
#define kP_B 2.0

// Derivative Control constants
#define kD_A 5.0
#define kD_B 1.0

// Controller Distance Tolerance (in encoder ticks)
#define distTolerance 3

// Deadband power settings
// The min PWM required for your robot's wheels to still move
// May be different for each motor
#define minPWM_A 75
#define minPWM_B 75

#define maxPWM_A 202
#define maxPWM_B 230

// servo angle definitions
#define up_angle 135
#define down_angle 105


int rect[] = {FORWARD, LEFT, FORWARD, LEFT, FORWARD, LEFT, FORWARD};



#define buttonPin 10
Servo arm;

void setup() {
  Serial.begin(9600);

  // Set up motors
  Serial.println("Setting up the Motors");
  motor_setup();

  // Space for push button
  Serial.print("Setting up the Push Button: Pin ");
  Serial.print(EncoderMotorLeft);
  Serial.println();
  pinMode(pushButton, INPUT_PULLUP);

  // Valid interrupt modes are: RISING, FALLING or CHANGE
  Serial.print("Setting up the Left Encoder: Pin ");
  Serial.print(EncoderMotorLeft);
  Serial.println();
  pinMode(EncoderMotorLeft, INPUT_PULLUP); //set the pin to input
  attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(EncoderMotorLeft),
indexLeftEncoderCount, CHANGE);
  Serial.print("Setting up the Right Encoder: Pin ");
  Serial.print(EncoderMotorRight);
  Serial.println();
  pinMode(EncoderMotorRight, INPUT_PULLUP);     //set the pin to input
  attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(EncoderMotorRight),
indexRightEncoderCount, CHANGE);

  pinMode(buttonPin, INPUT_PULLUP);
  arm.attach(5);
}

void loop()
{
  while (digitalRead(pushButton) == 1); // wait for button push
  while (digitalRead(pushButton) == 0); // wait for button release
  draw_rectangle();
  run_motor(A, 0);
  run_motor(B, 0);

  while (digitalRead(pushButton) == 1); // wait for button push
  while (digitalRead(pushButton) == 0); // wait for button release
  arm.write(up_angle);
  draw_circle();
  run_motor(A, 0);
  run_motor(B, 0);
  while (digitalRead(pushButton) == 1); // wait for button push
  while (digitalRead(pushButton) == 0); // wait for button release
  arm.write(up_angle);
  draw_right_triangle();
  run_motor(A, 0);
  run_motor(B, 0);

}

void draw_rectangle()
{
  for(int i = 0; i < sizeof(rect)/2; i++) {
      if(rect[i] == LEFT){
        turn(LEFT,90);
      }
      else{
        arm.write(down_angle);
        drive(rect[i]);
        arm.write(up_angle);
      }

  }
}

void draw_circle() {
  arm.write(down_angle);
  turn(LEFT,360);
}

// 45-45-90
void draw_right_triangle(){
  arm.write(down_angle);
  drive(FORWARD);

  arm.write(up_angle);
  turn(LEFT,90);

  arm.write(down_angle);
  drive(FORWARD);

  arm.write(up_angle);
  turn(LEFT,135);
  arm.write(down_angle);
  drive(7.07);
}


// Copy any necessary functions from Lab 3

//////////////////////////////////////////////////////////
int PD_Controller(float kP, float kD, int xerr, int dxerr, float dt, int minPWM,
int maxPWM)
//  gain, deadband, and error, both are integer values
{
  if (xerr <= distTolerance) { // if error is acceptable, PWM = 0
    return (0);
  }

  // Proportional and Derivative control
  int pwm = int((kP * xerr) + (kD * (dxerr/dt)));
  pwm = constrain(pwm,minPWM,maxPWM); // Bind value between motor's min and max
  return(pwm);
}
//////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////
int drive(float distance)
{
  // create variables needed for this function
  int countsDesired, cmdLeft, cmdRight, xerr_L, xerr_R, dxerr_L, dxerr_R;
  float dt;

  // Find the number of encoder counts based on the distance given, and the
  // configuration of your encoders and wheels
  countsDesired = distance / DistancePerRev * EncoderCountsPerRev;

  // reset the current encoder counts
  leftEncoderCount = 0;
  rightEncoderCount = 0;

  // we make the errors greater than our tolerance so our first test gets us into
the loop
  xerr_L = distTolerance + 1;
  xerr_R = distTolerance + 1;

  dxerr_L = 0;
  dxerr_R = 0;

  unsigned long t0 = millis();

  // Begin PID control until move is complete
  while (xerr_L > distTolerance || xerr_R > distTolerance)
  {
    dt = float(millis() - t0);
    t0 = millis();

    // Get PWM values from the controller function
    cmdLeft  = PD_Controller(kP_A, kD_A, xerr_L, dxerr_L, dt, minPWM_A,
maxPWM_A);
    cmdRight = PD_Controller(kP_B, kD_B, xerr_R, dxerr_L, dt, minPWM_B,
maxPWM_B);

    // Run motors with calculated PWM values
    run_motor(A, cmdLeft);
    run_motor(B, cmdRight);

    // Update encoder error
    // Error is the number of encoder counts between here and the destination
    dxerr_L = (countsDesired - leftEncoderCount ) - xerr_L;
    dxerr_R = (countsDesired - rightEncoderCount) - xerr_R;

    xerr_L = countsDesired - leftEncoderCount;
    xerr_R = countsDesired - rightEncoderCount;

    // Some print statements, for debugging
    Serial.print(xerr_L);
    Serial.print(" ");
    Serial.print(dxerr_L);
    Serial.print(" ");
    Serial.print(cmdLeft);
    Serial.print("\t");
    Serial.print(xerr_R);
    Serial.print(" ");
    Serial.print(dxerr_R);
    Serial.print(" ");
    Serial.println(cmdRight);
  }
  run_motor(A, 0);
  run_motor(B, 0);

}
////////////////////////////////////////////////////////////////////////////////

// Write a function for turning with PID control, similar to the drive function
int turn(float direction, float degrees)
{
  // create variables needed for this function
  int countsDesired, cmdLeft, cmdRight, xerr_L, xerr_R, dxerr_L, dxerr_R;
  float dt;

  // Find the number of encoder counts based on the distance given, and the
  // configuration of your encoders and wheels
  countsDesired = degrees / DegreesPerRev * EncoderCountsPerRev;

  // reset the current encoder counts
  leftEncoderCount = 0;
  rightEncoderCount = 0;

  // we make the errors greater than our tolerance so our first test gets us into
the loop
  xerr_L = distTolerance + 1;
  xerr_R = distTolerance + 1;

  dxerr_L = 0;
  dxerr_R = 0;

  unsigned long t0 = millis();

  // Begin PID control until move is complete
  while (xerr_L > distTolerance || xerr_R > distTolerance)
  {
    dt = float(millis() - t0);
    t0 = millis();

    // Get PWM values from controller function
    cmdLeft  = PD_Controller(kP_A, kD_A, xerr_L, dxerr_L, dt, minPWM_A,
maxPWM_A);
```

```
    cmdRight = PD_Controller(kP_B, kD_B, xerr_R, dxerr_L, dt, minPWM_B,
maxPWM_B);

    // Run motors with calculated PWM values in the indicated direction
    run_motor(A, direction * cmdLeft);
    run_motor(B, -direction * cmdRight);

    // Update encoder error
    // Error is the number of encoder counts between here and the destination
    dxerr_L = (countsDesired - leftEncoderCount ) - xerr_L;
    dxerr_R = (countsDesired - rightEncoderCount) - xerr_R;

    xerr_L = countsDesired - leftEncoderCount;
    xerr_R = countsDesired - rightEncoderCount;

    // Some print statements, for debugging
    Serial.print(xerr_L);
    Serial.print(" ");
    Serial.print(dxerr_L);
    Serial.print(" ");
    Serial.print(cmdLeft);
    Serial.print("\t");
    Serial.print(xerr_R);
    Serial.print(" ");
    Serial.print(dxerr_R);
    Serial.print(" ");
    Serial.println(cmdRight);
  }
  run_motor(A, 0);
  run_motor(B, 0);
}

// These are the encoder interrupt funcitons, don't need to edit.
void indexLeftEncoderCount()
{
  leftEncoderCount++;
}
void indexRightEncoderCount()
{
  rightEncoderCount++;
}
```