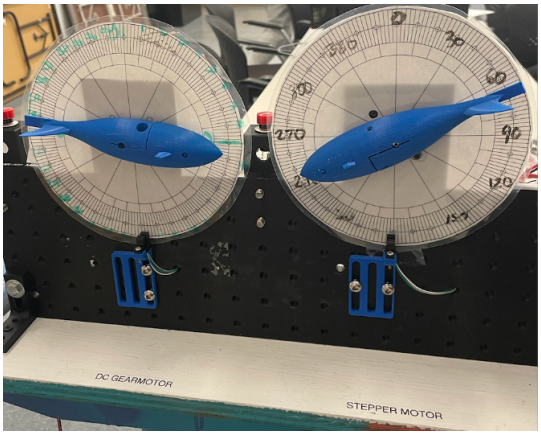
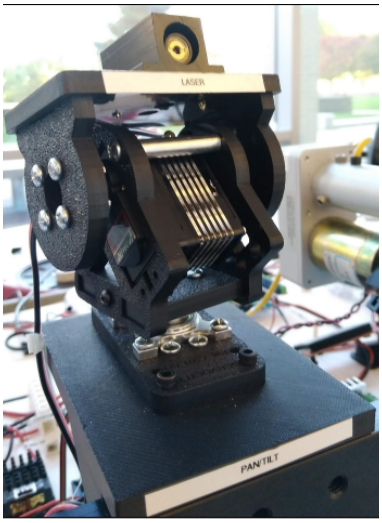
ENGR3390: Fundamentals of Robotics Report 1
ACT Lab





DC Gearmotor, Stepper Motor, Differential Drive or Dual DC Drive Motors, and Pan Tilt Laser (from top left to bottom middle)

Octopus Team: Alexis Wu, Braden Vaccari, Dowling Wong, Eliyahu Suskind, and Mahderekal Regassa

Report Author: Mahderekal Regassa

March 8, 2023

Table of Contents

Pan-Tilt Laser Turret

For the electrical wiring setup we primarily have the pan and tilt servos, link port, and power relay going into the spark fun motor shield –on the rightmost first row of pins– which is in turn connected to an Arduino microcontroller. The two servos move the laser in two axis's and are connected to the PWM pins 'D9' and 'D11' respectively. The relay makes sure the laser only turns on when it receives a command from Arduino, and it is connected to the row following the servos. The relay connects the laser to the link port; the latter converts 12V DC e-stoppable power to 5V as the motors being used require 5V.

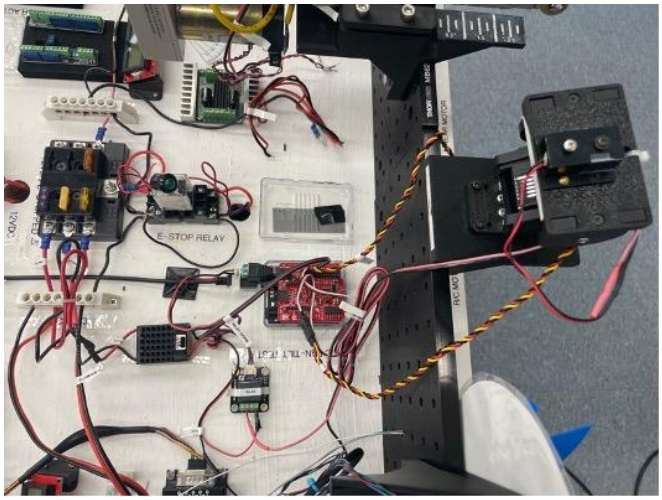


Fig.1.0 Pan-tilt laser turret

As the electrical wiring was already done when we started the lab, we made sure everything was connected to the expected pins. Additionally, we checked if the switches at the center of the motor shield are switched to VS and VMotor in order to get the motors powered from the jack and ensure that the jack is connected to the digital pins (PWM).

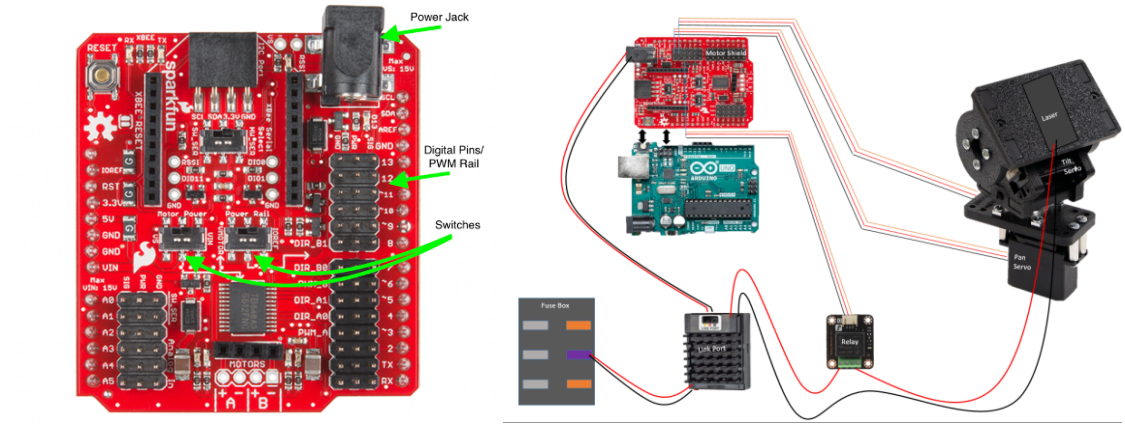


Fig.1.1 Indicating pins used on the Sparkfun board (left) and wiring diagram (left)

Code

Calibration

- Started off by calibrating the data. We first loaded the data, set it to position and angle variables (for both servos) and centered the angles to 0. We then fit the data to a curve and saved it.

```
load('rc_calibration_data.mat') % loading calibration data

panAngle = calibrationData(:,1);
panPos = calibrationData(:,2);
tiltAngle = calibrationData(:,3);
tiltPos = calibrationData(:,4);

% shifting the data to center it on 0 rad
tiltAngle_shifted = tiltAngle - (max(tiltAngle)+min(tiltAngle))/2;
panAngle_shifted = panAngle - (max(panAngle)+min(panAngle))/2;
```

fitting a curve (first order polynomial line) to match the data

```
fit_pan = fit(panAngle_shifted,panPos,'poly1')
fit_tilt = fit(tiltAngle_shifted,tiltPos,'poly1')
```

save fitted curve

```
save fit_pan fit_pan
save fit_tilt fit_tilt
```

```
    0.9599    0.2000    3.8397    0.2000
    1.0472    0.3000    3.6652    0.3000
    1.2217    0.4000    3.5779    0.4000
    1.3090    0.5000    3.3161    0.5000
    1.5359    0.6000    3.4034    0.6000
    1.6232    0.7000    3.1416    0.7000
    :         :         :         :
```

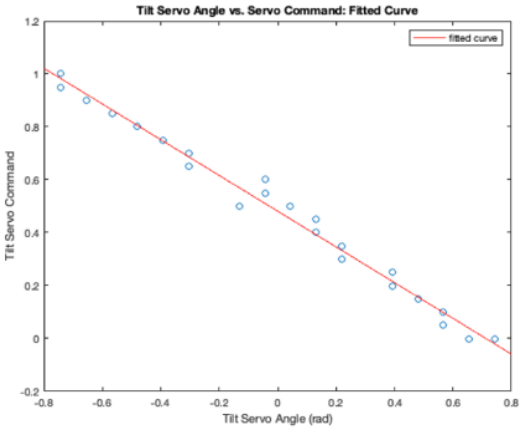
```
fit_pan =
     Linear model Poly1:
     fit_pan(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      0.7265  (0.6787, 0.7742)
       p2 =      0.4839  (0.4635, 0.5043)
fit_tilt =
     Linear model Poly1:
     fit_tilt(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =     -0.6756  (-0.7092, -0.642)
       p2 =      0.4814  (0.4659, 0.4969)
```

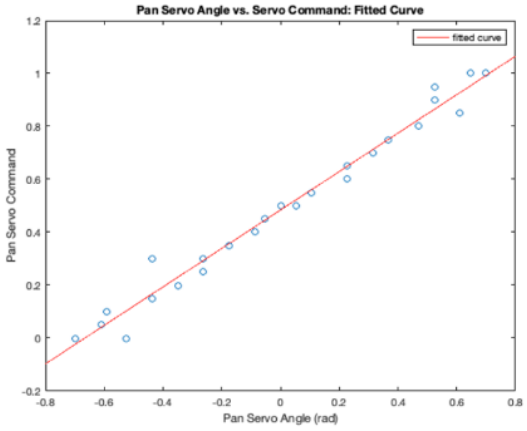
- We then plotted the tilt and pan angles with the servo command angles (0 to 1). I'll paste the code for the tilt angle vs servo command as the code for both servos is the same (only need to change the tilt data to pan data).

plotting the tilt vs command angle

```
figure()
plot(tiltAngle_shifted,tiltPos,"o")
hold on;
plot(fit_tilt)
xlabel('Tilt Servo Angle (rad)')
ylabel('Tilt Servo Command')
title('Tilt Servo Angle vs. Servo Command: Fitted Curve')
```

- Here are the calibration plots





Set up

- In the set up code, we set up the Arduino environment by calling the global variables in the functions section of the code (more on the functions section)

Set up robot control system (code that runs once)

```
% create arduino and arduino-velocity servo ref-objects
disp('note: It takes a 10 seconds or so to download code to Arduino ')


port = '/dev/tty.usbmodem141201'; % leave empty for change in port
[robotArduino, panServo, tiltServo, blinkLed] = SETUPARDUINO(port);


% turn on board LED on and off to signal program has started
Blink(robotArduino, blinkLed,3);
disp('warning! Pan and tilt servos active!');
```

Robot control loop

- The first chunk of code tells the laser where to go, second chunk commands the laser to draw a circle (see the comments for details)

```
writeDigitalPin(robotArduino,'D8',0);

% toTarget(d,x,y,panServo,tiltServo,fit_pan,fit_tilt)
% d is the distance from laser to wall
% x, y are points in coordinate system: range from -1 to 1
toTarget(1.5,0,1,panServo,tiltServo,fit_pan,fit_tilt)
```

To draw a circle:

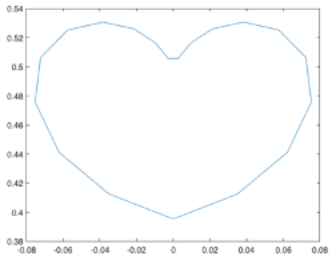
```
% creating the x and y for the circle
circle_step = 50
radius=0.1;
theta=linspace(0,2*pi,circle_step)
circle_x=radius*cos(theta);
circle_y=radius*sin(theta)+1;

writeDigitalPin(robotArduino,'D8',1); %turn laser on
% loop through x and y values to make the circle
for i = 1:circle_step*10
     i = rem(i,circle_step);
        if (i > 0)
            toTarget(1.5,circle_x(i),circle_y(i),panServo,tiltServo,fit_pan,fit_tilt)
        end
end

writeDigitalPin(robotArduino,'D8',0); % turn laser off
```

- Here's the code for drawing the shape of a heart.

To draw a heart

```
% making x and y values
step = 20
t = linspace(-pi,pi,step);
x = t .* sin( pi * .872*sin(t)./t)
y = -abs(t) .* cos(pi * sin(t)./t)

% rescaling
x = x/25
y = y/30 + 0.5

writeDigitalPin(robotArduino,'D8',1); % turns laser on

% loops through i steps for x and y to create the heart
for i = 1:step*20
    i = rem(i,step);
    if (i > 0)

        toTarget(1.5,x(i),y(i),panServo,tiltServo,fit_pan,fit_tilt)
    end
end
writeDigitalPin(robotArduino,'D8',0); % turns laser off

plot(x,y)
```

```
step = 20
x = 1×20
    -0.0000   -0.8747   -1.5565   -1.8824 ...

y = 1×20
    -3.1416   -2.6278   -1.7667   -0.7315 ...

x = 1×20
    -0.0000   -0.0350   -0.0623   -0.0753 ...

y = 1×20
     0.3953    0.4124    0.4411    0.4756 ...
```



Robot Functions

- In our robot functions, we have set a couple of global variables where each one either centers the Arduino or sets the two servo objects (pan and tilt) to their respective pins.

```
function [robotArduino, panServo, tiltServo, blinkLED]= SETUPARDUINO(COMPORT)
% SETUPARDUINO creates and configures an arduino to be a simple robot
% controller. It requires which COM port your Arduino is attached to
% as its input and returns an Arduino object called robotArduino
% D. Barrett 2021 Rev A
% Create a global arduino object so that it can be used in functions
% a - arduino('setToYourComNumber', 'Uno', 'Libraries', 'Servo');
    robotArduino = arduino(COMPORT, 'Uno', 'Libraries', 'Servo') ;

% configure pin 13 as a digitial-out LED
    blinkLED='D13' ;
    configurePin(robotArduino,blinkLED,'DigitalOutput');

% create two servo objects driving PWM pin 9 and pin 11
% MinPulseDuration: 1120 (microseconds) center 1520 (microseconds)
% MaxPulseDuration: 1920 (microseconds)

    panServo=servo(robotArduino,'D9','MinPulseDuration', 10*10^-6, ...
'MaxPulseDuration', 1925*10^-6) ;
    tiltServo = servo(robotArduino,'D11', 'MinPulseDuration', 10*10^-6, ...
'MaxPulseDuration', 1925*10^-6);
% R-C Mode expects to start up with joystick centered
% In MATLAB function servo position is 8-1 so 8.5 (1520ms) is centered
    writePosition (panServo, 0.5); % always start servo-command at 0.5
    writePosition(tiltServo, 0.5); % always start servo-command at 0.5
    pause(5.0);
% wait for Arduino to send stable pwm
end
```

- Our SENSE function prompts the user to give a desired x and y position while the ACT function commands the two servos where to go.

Sense Functions (store all Snese related local functions here)

```
function x = getX()
    x = input('put desired target horizontal position(-1 to 1): ');
end

function y =getY()
    y = input('put desired target vertical position(-1 to 1): ');
end
```

Think Function (store all Think related local functions here)

```
function THINK()
    % null function, not much thinking to do here
end
```

Act Functions (store all Act related local functions here)

```
% telling the pan and tilt servo where to go
function ACTRCServoArduino(panServo, tiltServo, panAngle, tiltAngle)
    writePosition(panServo,panAngle);
    writePosition(tiltServo,tiltAngle);
end
```

Brushed DC Gearmotor/ encoder with RoboClaw Drive

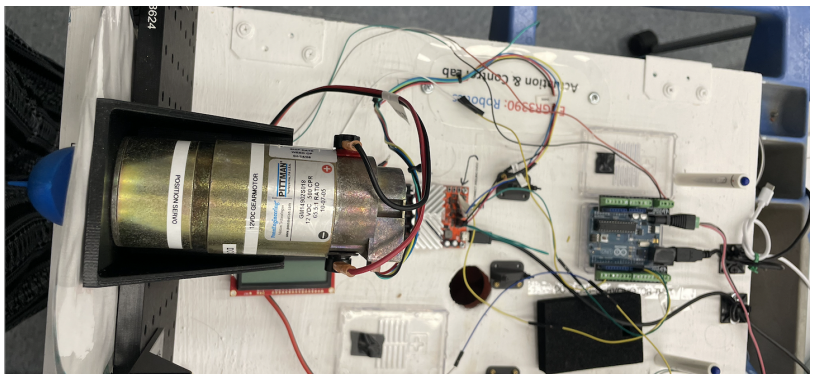


Fig 1.2 Brushed DC gearmotor

A brushed DC motor works by moving a coil inside a magnetic field (surrounding magnets). For this lab, our brushed Pittman DC gearmotor has a gear ratio of 65.5:1 and an encoder attached to it that determines the speed and position of the motor. This encoder is quadrature, or in other words has two steady pulses going in opposite directions. A motor control driver we used in this lab is called RoboClaw and it controls the motion of our DC gearmotor by processing the encoder values.

Once the RoboClaw controller driver is installed, we set the first channel to Motor1 Setpoint and second channel to Motor1 Speed on the PWM setting tab. Due to the relative incremental encoder on the motor, the location on power up will be zero and

then the position will count up and down from there. A PWM position signal is being sent from the arduino (pin 9) to S1 pin on the RoboClaw. Check the wiring below:

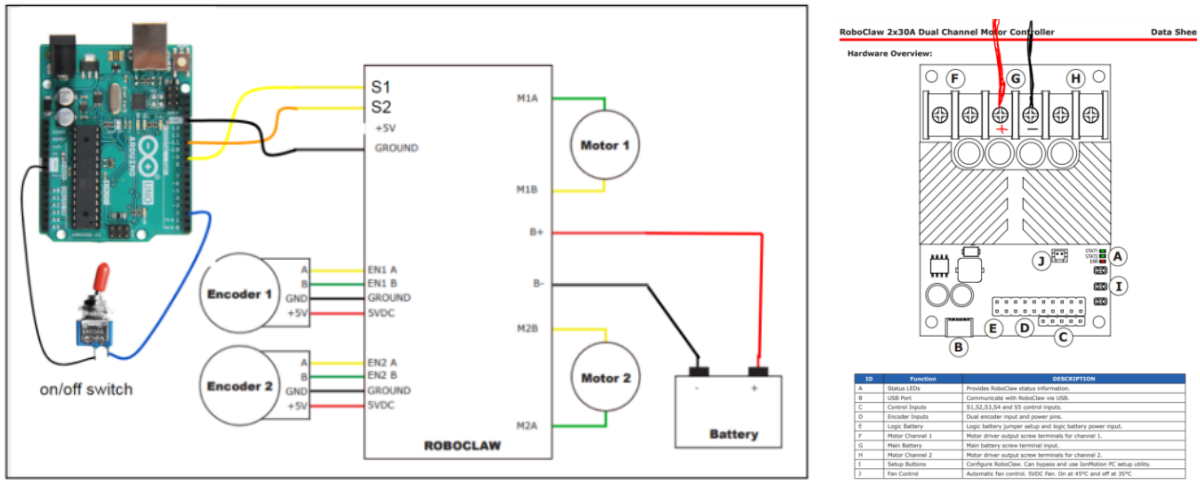


Fig 1.3 Electrical wiring

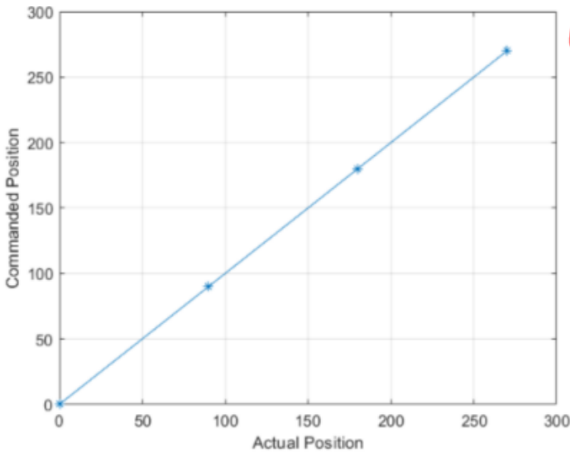
Code

The code was not entirely functional so I will only include the main ACT function below

Act Functions

```
function ACT(myServo, commandedAngle)
% A function to move the motor to a desired angle
% Inputs: myServo is servo obj to be moved, commandedAngle is desired angle in int

% Scaled desired angle from degree range to number between 0 and 1 for
% motor readability
scaledAngle = commandedAngle/360;
writePosition(myServo, scaledAngle)

end
```

The command vs actual position was plotted in calibrating the the DC gearmotor, see below:



DC Stepper Motor

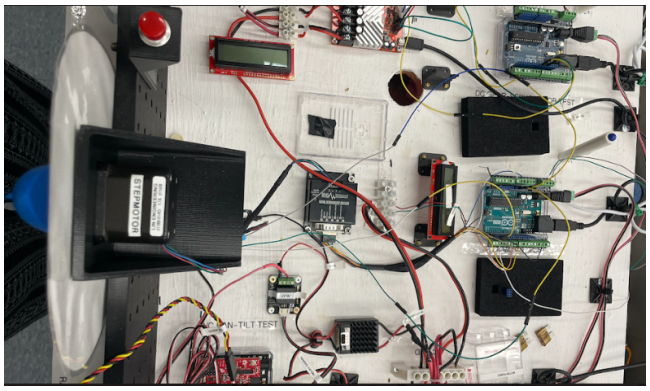


Fig. 1.4 DC Stepper Motor

Stepper motors, similar to the gear motors explained above, have electromagnetic coils but are organized in groups called “phases” around a central magnetically stepped rotor. When these coils are energized in sequence with a pulse, it gives stepper motors the discrete steps as they rotate.

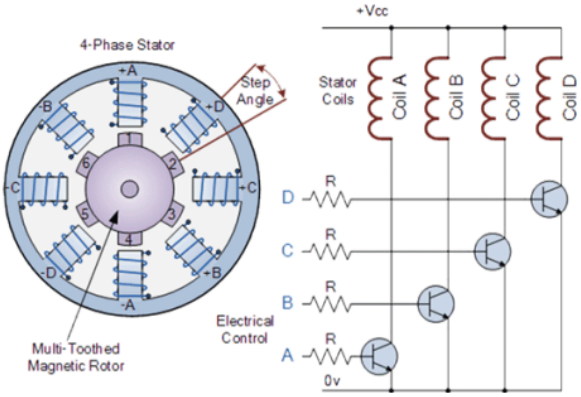


Fig 1.5 Phases of electromagnetic coils inside a stepper motor and electrical powering set up

Look at the electrical wiring diagram below for the entire setup:

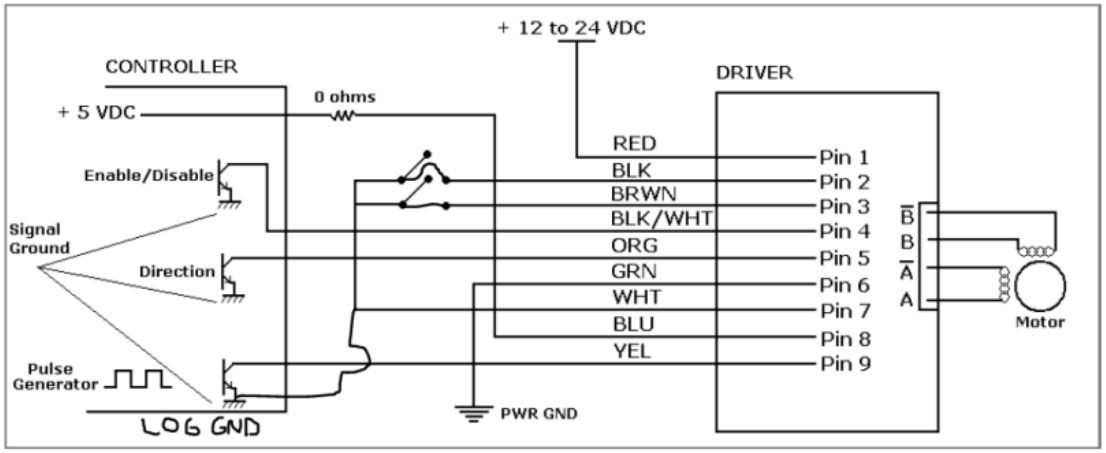


Fig. 1.6 Electrical wiring diagram for the DC stepper motor, driver and controller

Code

After clearing the workspace, we set up the control system.

Set up Robot control system(code runs once)

```
%Create arduino and arduino-pin ref objects
[robotArduino, blinkLED]=SETUPARDUINO('/dev/cu.usbmodem14101', 'Mega2560');
%ACTmovesStepMotor(arduinoName, numberOfSteps, direction, speed)
pause(1.0);
beep;


ACTmoveStepMotor(robotArduino, 50, 0, 0.01)
Blink(robotArduino, blinkLED, 1);
beep;
ACTmoveStepMotor(robotArduino, 50, 0, 0.01)
Blink(robotArduino, blinkLED, 1);
beep;

%configure test loop to collect n data points.
nTest = input(['Enter number of servo test positions,' ...
    'followed by enter: ']);
%create a variable to hold experimental position data
    positionData = zeros(nTest,2);
r = rateControl(0.25);
reset(r);
```

Then the loop is set up. This loop calls different functions in the code (SENSE, THINK etc) and sets the position of the step motor to the angle the user puts in. Afterwards, I will copy functions.

Run robot control loop(code that runs over and over)

```
controlFlag = 1;
oldAngle = 0;
while(controlFlag < nTest+1)
    commandAngle = SENSE();
    [steps, direction] = THINK(oldAngle, commandAngle);
    ACTmoveStepMotor(robotArduino, steps, direction, 0.1);
    oldAngle = commandAngle;

    positionData(controlFlag, 1)= input('Enter actual position degrees: ');
    positionData(controlFlag, 2)= commandAngle;
    Blink(robotArduino, blinkLED, 1);
    waitfor(r);
    controlFlag = controlFlag+1;
end
```

Robot Function(store this code local functions here)

```
function [robotArduino, blinkLED] = SETUPARDUINO(COMPORT, MODEL)
    robotArduino = arduino(COMPORT, MODEL, 'Libraries', 'I2C');
    %create a global arduino object
    disp('Warning! Downloading may take a seconds');

    blinkLED = 'D13';
    configurePin(robotArduino, blinkLED, 'DigitalOutput');
    %pin13 as LED

    stepPin = 'D6';
    configurePin(robotArduino, stepPin, 'DigitalOutput');
    %pin6 as step pulse

    dirPin = 'D7';
    configurePin(robotArduino, dirPin, 'DigitalOutput');
    %pin7 as direction control
    Blink(robotArduino, blinkLED, 3);
    disp('Warning! Stepper moving!');
    pause(3.0);
end

function [] = Blink(a, LED, n)
    for bIndex = 1:n
        writeDigitalPin(a, LED, 0);
        pause(0.2);
        writeDigitalPin(a, LED, 1);
        pause(0.2);
    end
end
```

ACT Function(store all sense related local functions here)

```
function [] = ACTmoveStepMotor(arduinoName, numberOfSteps, direction, speed)
%rotate stepper by numberOfSteps in direction, with puase in between
% #of steps are(0 -200) direction is (0 clockwise)(1 counterclockwise)
    stepPin = 'D6';
    dirPin = 'D7';

    writeDigitalPin(arduinoName, dirPin, direction);
    %run stepper for number of steps required at speed requested
    for mIndex = 1:numberOfSteps
        writeDigitalPin(arduinoName, stepPin, 0);
        pause(speed);
        writeDigitalPin(arduinoName, stepPin, 1);
        pause(speed);
    end
end
```

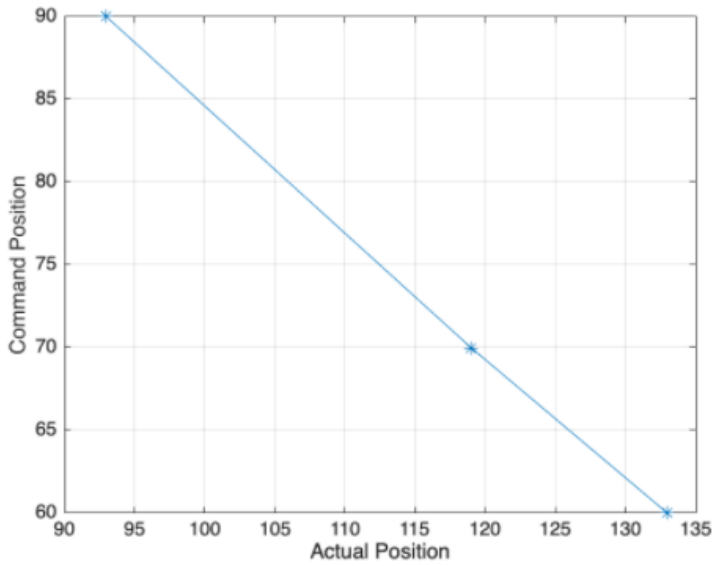
Sense Function(store all sense related local functions here)

```
function commandAngle = SENSE()
    commandAngle = input('Enter desired servo angle in degrees: ');
end

function readAngle = SENSE2(arduinoName, readPin)
    print(readVoltage(arduinoName,readPin))

end
```

Think Function(store all think related local functions here)

```
function [steps, direction]= THINK(oldAngle, commandAngle)
    differenceAngle = oldAngle-commandAngle;
    steps = abs(differenceAngle*0.55555);
    if (commandAngle >=oldAngle)
        direction = 1;
    else
        direction=0;
    end
end
```

The last chunk of code run after that will close everything and clean the workspace. In the mission data, the actual position vs command position is plotted and it shows that they are inversely proportional to one another.



Dual DC Drive Wheels with RoboClaw Control

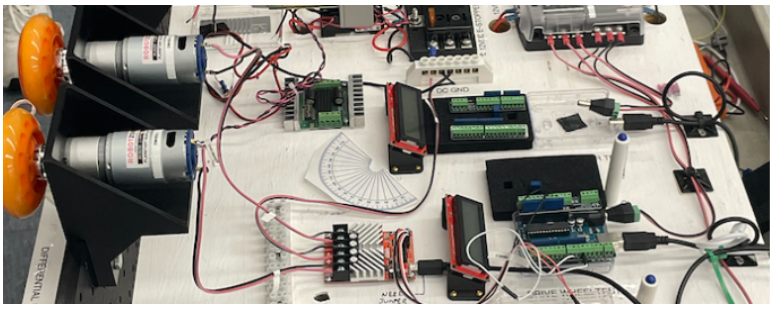


Fig. 1.7 Dual DC drive wheels

Dual DC drive motors provide propulsion by acting as a pair to control the robot motion. These motors are usually open loop and velocity controlled. When both motors are moving in the same speed and direction, the robot will move forward or if they move in the same speed but opposite direction, the robot will reverse. The robot will arc at a radius if both motors go at different speeds in either direction or the robot will rotate if the motors turn with the same speed but in opposite directions.

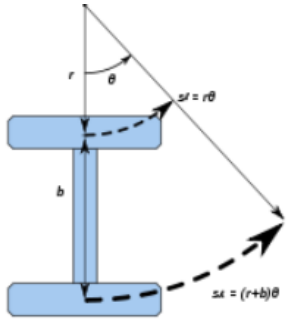


Fig. 1.8 Path of the wheels

Code

Same as before we set up the COM port and number of times we test and have our control loop.

Run robot control loop(code that runs over and over)

```
controlFlag = 1;        %create a loop control
while (controlFlag < nTests+1)%This part of the code is actually just ask
to input
    commandSpeed = SENSE();
    THINK();
    ACTRoboClawServoArduino(leftServo, rightServo, commandSpeed,
commandSpeed);
    leftVelocityData(controlFlag,1)=input('Enter actual left velocity
(Rev/Sec: ');
    leftVelocityData(controlFlag,2)=commandSpeed;
    rightVelocityData(controlFlag,1)=input('Enter actual right velocity
(Rev/Sec: ');
    rightVelocityData(controlFlag,2)=commandSpeed;
    Blink(robotArduino, blinkLED,1);
    waitfor(r);
    controlFlag = controlFlag+1;
end
```

Robot Functions

```
function [robotArduino, leftServo, rightServo,
blinkLED]=SETUPARDUINO(COMPORT, MODEL)
    robotArduino = arduino(COMPORT, MODEL, 'libraries', 'Servo');
    disp('Warning, RoboClaw must be OFF before running');
    blinkLED='D13';
    configurePin(robotArduino, blinkLED, 'DigitalOutput');
    %setting up the left and right servo pins, ratio of operation
    leftServo = servo(robotArduino, 'D9', 'MinPulseDuration', 10*10^-
6, ...
        'MaxPulseDuration', 1925*10^-6);
    rightServo = servo(robotArduino, 'D11', 'MinPulseDuration',
10*10^6, ...'MaxPulseDuration', 1925*10^-6);
    writePosition(leftServo, 0.5);
    writePosition(leftServo, 0.5);
    pause(5.0);
    disp('Please poswer up RoboClaw now');
    pause(2.0);
end
```

This code sets up the motors.

ACT Functions

```
function ACTRoboClawServoArduino(leftServo, rightServo, leftVelocity,
rightVelocity)
writePosition(leftServo, leftVelocity);
writePosition(rightServo, rightVelocity);


end
```

```
function right(leftServo, rightServo)
writePosition(leftServo, 1);
writePosition(rightServo, 0.5);
end


function left(leftServo, rightServo)
writePosition(leftServo, 0.5);
writePosition(rightServo, 1);
end
```

```
function forward(leftServo, rightServo)
writePosition(leftServo, 1);
writePosition(rightServo, 1);
end


function reverse(leftServo, rightServo)
writePosition(leftServo, 0);
writePosition(rightServo, 0);
end
```

This code (ACT function) is setting commands for the motor to move forward, backward, turn left and right.

The calibration line for left and right motors were plotted and are shown below. They are parallel to each other indicating that the hardware has similar output.