Methodologies for the development of PID motor controllers

Control Systems, EGR2006-2324

Lab Report Coursework

Joseph Ashton

27047440

UNIVERSITY OF
LINCOLN

The University of Lincoln, School of Engineering

Brayford Pool, Lincoln, LN6 7TS, United Kingdom

2023-12-13

Contents

List of Figures

List of Tables

Nomenclature

The lists below define all the abbreviations, characters and symbols used throughout this document. For ease of reference all abbreviations, characters, symbols, citations, figures, tables and equations within the text are linked.

Abbreviations

Characters & Symbols

The motor torque constant (K_t) and the back electromotive force constant (K_b) will be treated as the motor constant (K).

1 Introduction

This report aims to demonstrate 2 methods for modelling the behaviour of a brushed DC motor to produce an effective PID controller. The first method requires no physical interaction with the motor provided that sufficient specifications are known in advance. The second method requires no knowledge of the motors specifications and only 10 seconds of test data.

1.1 Theoretical Background

1.1.1 DC Motors

DC motors are a key component for a broad range of industrial & commercial applications especially where the key concerns are simplicity, speed control with further advantages to battery powered systems. The speed of a brushless motor is linearly related to the input voltage this combined with torque staying relatively constant makes speed control particularly easy to implement. The fact that they run form a DC source makes them suitable for battery powered systems as no inverter or high-power DAC is necessary. [1]

The motor modelled & tested in this report is a brushed DC motor. Brushed DC motors are constructed from a rotor with a series of windings that terminate in contacts these are joined electrically to the stator by brushes this is called a commutator. The contacts are arranged such that the windings generate an electric field that is slightly out of phase with that of the stators permanent magnets inducing a torque. As this torque spins the rotor the brushes move over the contacts such that the field generated always stays ahead of that of the stator.

Brushed DC motor do have several drawbacks primarily to do with the brushes themselves as they wear during use. Not only do the brushes need to be replaced once worn out but also they produce a conductive dust that can damage sensitive electrical components if not managed effectively. This means high use appliances that use Brushed DC motors need regular maintenance or regularly replaced.

Brushless DC motors solve this issue by make the commutator and therefore the brushes redundant by using an electronic motor controller to modulate the power sent into each coil and thereby rotate the electric field.

1.1.2 PID Control

PID controller is a type of feedback control system this means it is a function that maintain a desired output by continuously adjusting the control input based on the difference between the desired output and the actual output of the system called the error. PID is an acronym that stands for stands for Proportional, Integral, and Derivative the 3 domains in which it monitors this difference between desired output and actual output. Proportional refers to the current error, integral is the accumulated sum of past errors over time while the derivative component is proportional to the rate of change of the error. For each of these domains the gain can be adjusted this will affect how the system reacts to a change in the given domain.

2 Motor Modelling & Simulation

This part of the report outlines 2 different ways of modelling a dynamic system such as the motor used. The first method is Theory Based Transfer Function explored in section 2.1 where the model is derived from known mathematical relationships & physical principles. In the following section 2.2 Data driven Transfer Function a model is derived from a set of experiential data from the motor. In the final section 2.3 Theoretical & data driven model comparison the accuracy & performance of the two models are compared.

2.1 Theory Based Transfer Function

In this section a transfer function that predicts rotor position based on input voltage is found based on general well documented mathematical relationships and factoring in some given values specific to the motor used

The rotor position can be considered a 3rd order system starting with a function for the angular acceleration of the rotor. The angular acceleration of the rotor shaft ($\ddot{\theta}$) is related to torque (T), inertia (J) & damping (b). [2]

$$T = J\ddot{\theta} + b\dot{\theta}$$

Substituting for the motor constant (K) with $T = Ki$, and finding current (i) in terms of angular velocity using $V = Ri + e$ & $e = K\dot{\theta}$ gives:

$$K\frac{(V - K\dot{\theta})}{R} = J\ddot{\theta} + b\dot{\theta}$$

Which can be simplified & rearranged to:

$$\frac{\dot{\theta}}{V}(s) = \frac{k}{(Js + b)(Ls + R) + k^2}$$

and taking a Laplace transform of both sides gives position.

$$\frac{\theta}{V}(s) = \frac{k}{(Js + b)(Ls + R) + k^2} \times \frac{1}{s}$$

This transfer function is added to a Simulink model and all known values are substituted in apart from damping (b) which was estimated for best fit with experimental data similar to the values found in the Data driven below. For the full code & description see appendix.

Table 1 Given values for theory-based transfer function

Name	Value	Unit
Motor Torque Constant	$5.02E-02$	$N.m/A$
Motor Armature Resistance	10.6	Ω
Motor Armature Inductance	0.82	mH
Motor Maximum Continuous Torque	$3.50E-02$	$N.m$
Motor Power Rating	18	W
Moment Of Inertia Of Rotor & load	$2.21E-05$	$kg.m2$
Motor Mechanical Time Constant	$5.00E-03$	s
Viscous friction constant	$3.30E-05$	$N.m.s$

**Note that J accounts for the sum of the motor's rotor inertia & the inertial Load disk but does not account the weights used part*

The theoretical model was used to predict rotor velocity and position and then compared with provided lab data for the motor's performance. The accuracy of the model seen in Figure 1 gives sufficient confidence for its use in developing a PID controller.

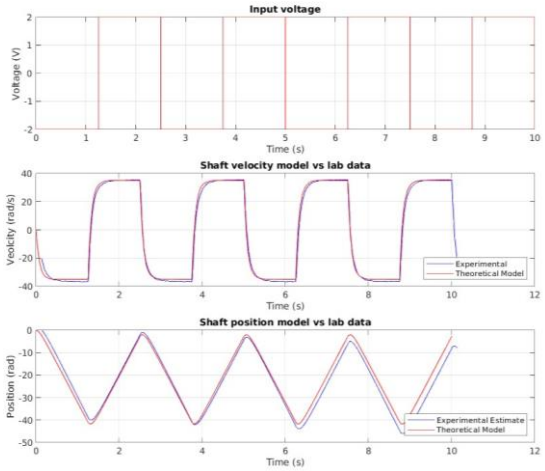


Figure 1 Theory model against provided motor data

2.2 Data driven Transfer Function

In this section a second much simpler transfer function that predicts rotor position based on input voltage is found by varying the gain (K), and time constant (τ) in the general formula $G(s) = \frac{K}{\tau s+1}$ to fit existing data.

A data set was provided which recorded the angular velocity of the motor shaft over time while it was powered by a square wave of $2\ A$ at $0.4\ Hz$. The time series was shifted to start at the first full wavelength and plotted against the Simulink model below left.

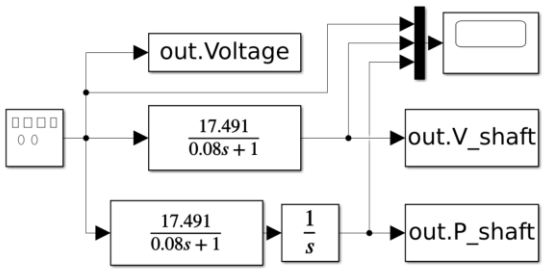


Figure 2 Data based model Simulink diagram

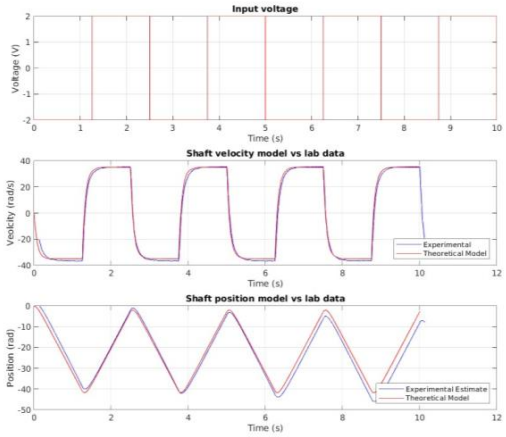


Figure 3 Data Model against provided motor data

Through iterative trial & error the gain (K), and time constant (τ) values are optimised to fit the general 1st order transfer function $G(s) = \frac{K}{\tau s+1}$ to match the behaviour of the motor under the same conditions above right.

2.3 Theoretical & data driven model comparison

The theoretical & data driven models are compared in a simulation from the same signal generator as seen in Figure 4 and are indistinguishable in Figure 5.

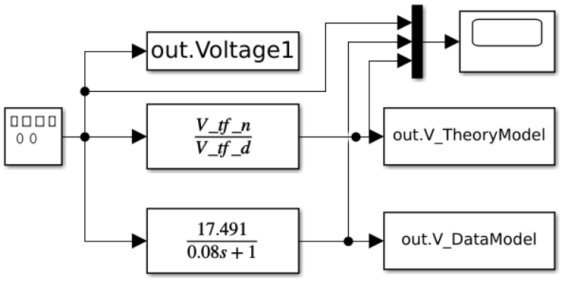


Figure 4 Theory vs Data based model Simulink diagram

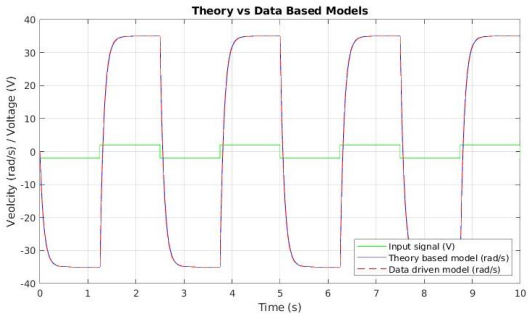


Figure 5 Theory vs Data based model comparison plot

This carries over to the numerical analysis where the average deviation of both models from the lab data is less than 0.1%. Both models are accurate enough to base the development of PID controller from.

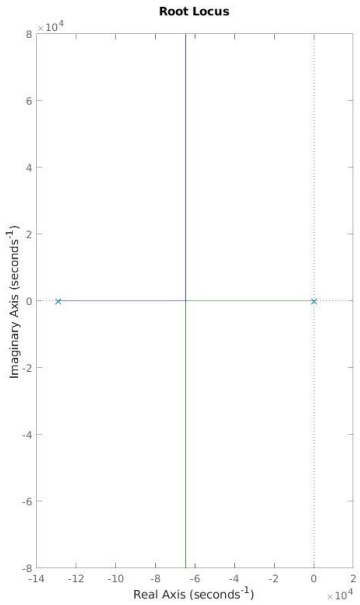


Figure 6 Root Locus plot of Theory Model

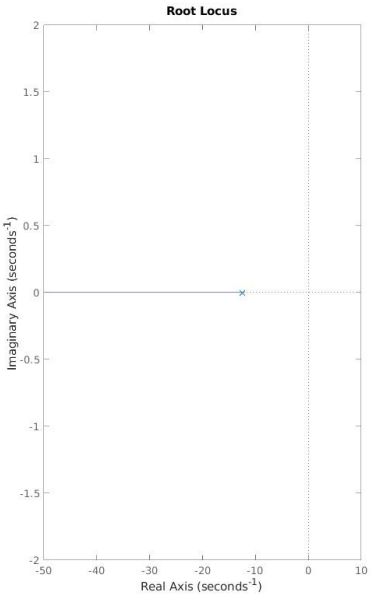


Figure 7 Root Locus of Data Model

In terms of reliability the data-based model is more stable as it's a simpler first order system with a single real pole at -12.5 compared to the theory-based model with 2 poles at -10 and -129260, the second is at such a high frequency its unlikely to be reached. It is however worth noting that the theory-based model did consistently fail to solve in fixed time step ode4 as it would hit a singularity.

3 PID Controller design

This part of the report describes how the process of PID controller design the first section 3.1 PID Controller Tuning focuses on tuning a PID controller in MATLAB & Simulink based on a representative model. section 3.2 Testing Setup describes the testing methodology and finally 3.3 PID Controller Testing explores the results.

3.1 PID Controller Tuning

In this section a PID controller is tuned to fit the response of the motor to a desired input based on the Data driven Transfer Function

A Simulink model, **Error! Reference source not found. Error! Reference source not found.**, is made by integrating the Data driven Transfer Function for rotor velocity into a closed loop PID controller. The signal generator is set to square wave $1\ A$ @ $0.4\ Hz$ and the PID controller is limited to $\pm\ 15\ V$ based on the maximum range of the power supply used in 3.2 Testing Setup.

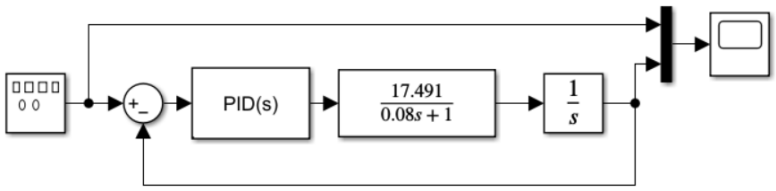


Figure 8 Simulink PID block diagram

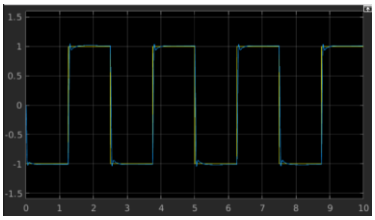


Figure 9 Scope output from Simulink PID controller

The proportional, integral & derivative gain are initially set to 1, 0, & 0 respectively and then tuned manually to maximise response time and minimise overshoot & steady state error. The final values used where; $k_p = 5$, $k_d = 0.15$, and $k_i = 0.2$, these settings produced the blue trace seen **Error! Reference source not found.** in **Error! Reference source not found.**. These values also worked well for controlling velocity, see appendix.

3.2 Testing Setup

The Quanser DC Motor Control Trainer was used along with its accompanying QET interactive interface (QETii) software on a windows machine, for further information see appendix A of reference [3]. In the QETii software the signal generator is set to output a square wave of amplitude 1 at 0.4 Hz. The b_{sp} & b_{sd} values are both set to 1 while transfer function (tf) is set to 0.006. As determined in section 3 PID $k_p = 5$, $k_d = 0.15$, and $k_i = 0.2$. [4]

Measurements are taken by waiting for at least 10 seconds after the motor behaviour has settled and then saving time series, input signal & rotor position values to a suitably named .mat file.

The following experiments where undertaken:

Table 2 List of experiments

Test name	Independent variable
no_weights.mat	Control, no variables changed
_1a_weights.mat	1 aluminium weight added
_1b_weights.mat	1 brass weight added
_2a_weights.mat	2 aluminium weights added
_2b_weights.mat	2 brass weights added
no_weights_A01.mat	Signal generator amplitude set to 0.1
no_weights_A10.mat	Signal generator amplitude set to 10
no_weights_F4.mat	Signal generator frequency set to 4 Hz
no_weights_F004.mat	Signal generator frequency set to 0.04 Hz
_2a_weights_A01.mat	2 aluminium weights added & Signal generator amplitude set to 0.1
_2a_weights_A10.mat	2 aluminium weights added & Signal generator amplitude set to 10

For those that wish to reproduce these tests it is advised that your file names should **not begin with numbers, underscores or contain symbols such as "#", this will make the files & variables generated within unreadable, see appendix for more details.*

3.3 PID Controller Testing

In this section, the PID values found previously in 3.1 PID Controller Tuning are tested on a physical motor monitoring performance in ideal & varying conditions with the setup described above in 3.2 Testing Setup.

3.3.1 Effective Amplitude & frequency range

The system's ability to make fine adjustments as well as rapid dynamic adjustments is tested by changing the amplitude of the square wave generator that determines the target position. In Figure 10 System response in 3 orders of amplitude below are the systems response at 3 levels. The top plot is considered nominal with the square wave generator set to 1 radian at 0.4 Hz meaning the rotor position should rotate by 2 radians as quickly & accurately as possible. The bottom left plot benchmarks the rotor must adjust by 0.2 radians this is considered a fine adjustment. The third tests the course adjustments where the rotor must rotate 20 radians.

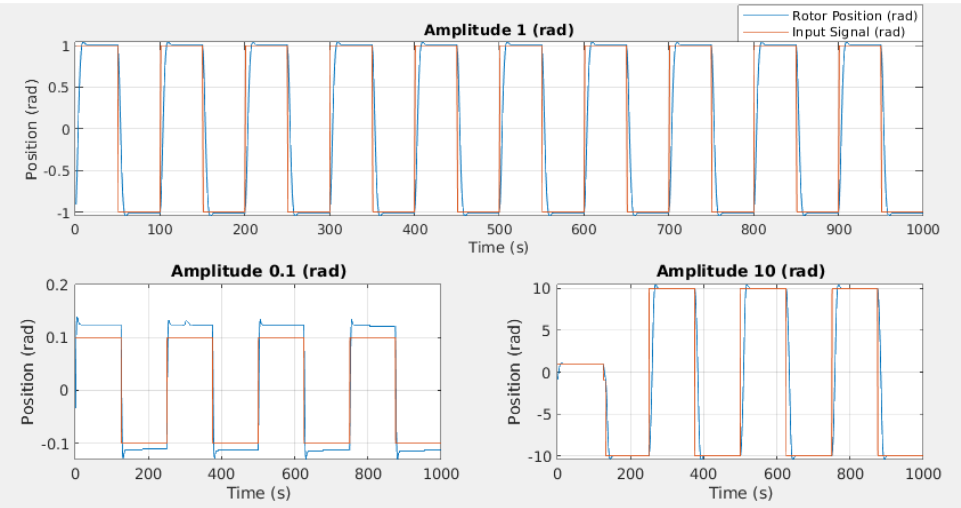


Figure 10 System response in 3 orders of amplitude

It should be noted that the steady state error will appear larger in the fine adjustment benchmark & smaller in the course, relative to the target movement they are however in terms of absolute value they were similar.

During a nominal response it takes the system 200 ms to respond and then correct for overshoot, this can be observed in the top plot of Figure 10 above.

Based on this increasing the frequency an order of magnitude from 0.4 to 4 Hz should approach the systems upper limit for the nominal 2 radian rotations. This was tested and the response can be seen below in Figure 11.

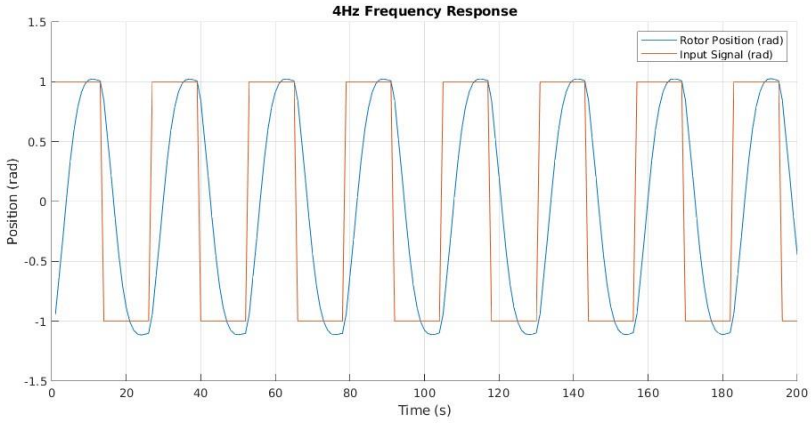


Figure 11 maximum frequency response

3.3.2 Resilience

The system's ability to deal with disturbances is tested by attaching additional weights to one or both points on the inertial load disk [3], these weights will be referred to as either brass or aluminium, these are 20g and 6.5g respectively. When one weight is attached it adds an off-axis load however when both are attached these balance out.

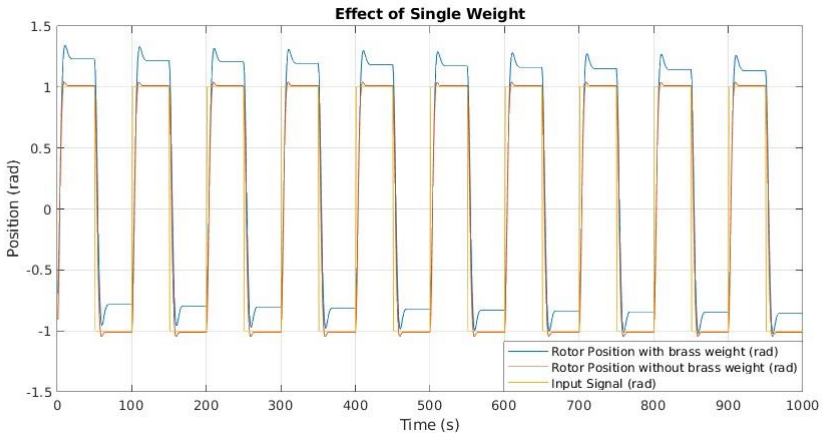


Figure 12 effect of an off-axis mass added to the rotor

With a single weight, the response of the system is displaced settling above or below its target depending on which side the mass is added.

The addition of pairs of weights further affects the systems overshoot, steady state error, and impulse response, as can be seen below in Figure 13.

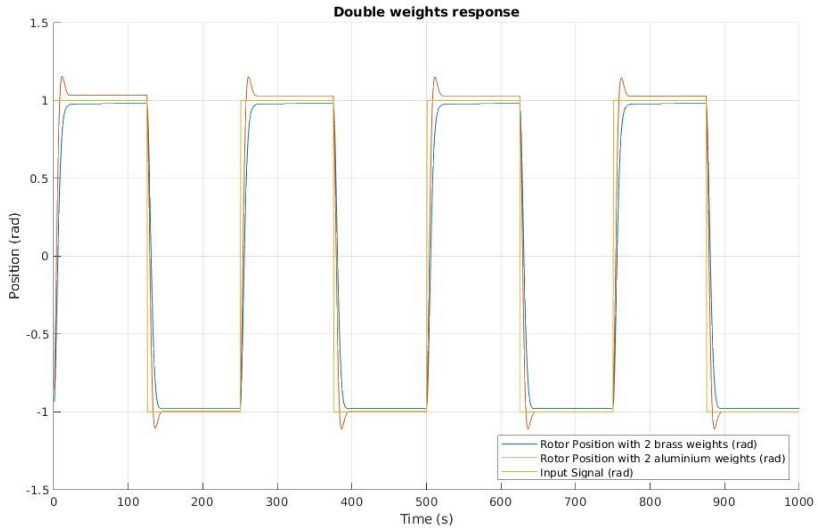


Figure 13 effect of different balanced masses on rotor position response

The reason these specific weights where chosen is they highlight 2 different effects on the system. The aluminium weights increase the systems overshoot from nominal Aswell as add positive steady state error, whereas the brass weights do the opposite slowing the impulse response & adding negative steady state error.

4 Conclusion

This report has demonstrated 2 methods for modelling the behaviour of a brushed DC motor and established that both methods are capable of producing a model sufficiently accurate to base the development of a PID controller. And validated this with experimental evidence.

5 References

Bishop, Mechatronic Systems, Sensors, and Actuators, Fundamentals and Modeling, Taylor & Francis Group, 2007.

ersity of Michigan, “University of Michigan Control Tutorials,” [Online]. Available: https://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition&section=SystemModeling.

H. L. Karl Johan Åström, “USB QICii Laboratory Workboo,” Quanser.

Bernstein, “https://umich.edu,” June 2005. [Online]. Available: https://public.websites.umich.edu/~dsbaero/others/35-QuanserDCMotor.pdf.

Works, “MATLAB Documentation,” 12 06 2022. [Online]. Available: https://uk.mathworks.com/help/matlab/. [Accessed 14 01 2023].

6 Appendices

Data based PID model used for velocity control

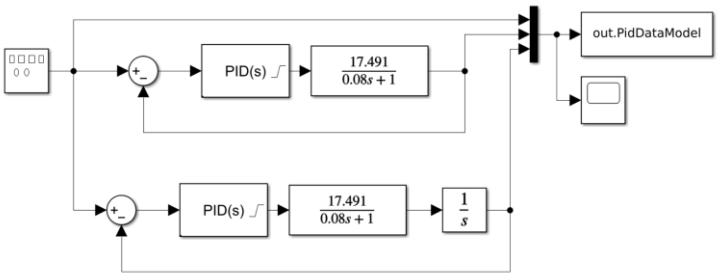


Figure 14 Data based PID controller Simulink diagram for position & velocity control

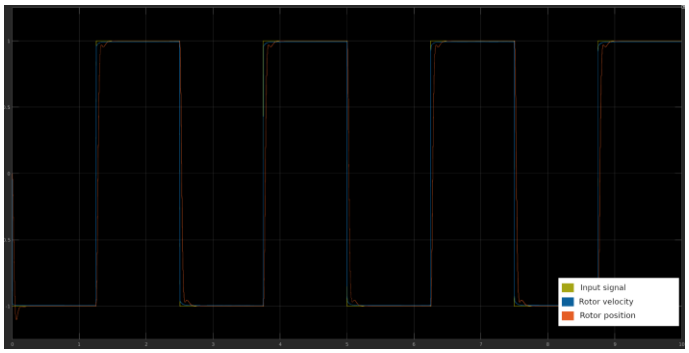
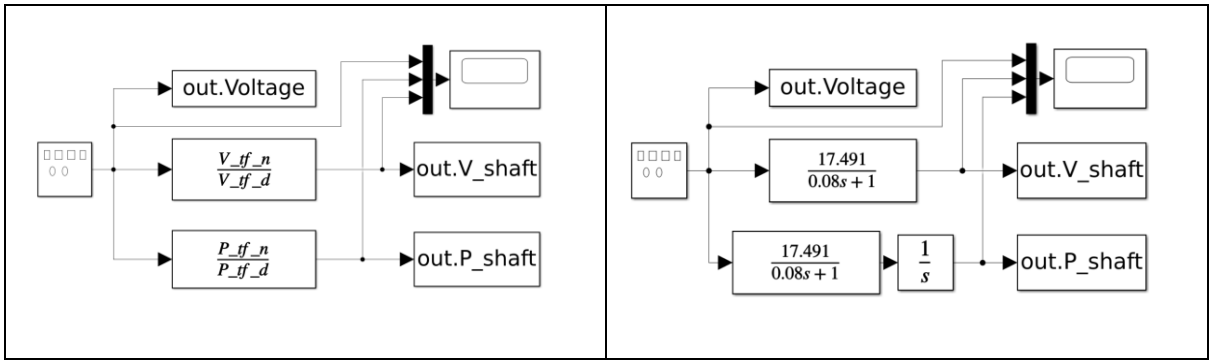


Figure 15 Scope output of Figure 14

Simulink Models used



Main Working document :

Control Systems Coursework (Ctrl sys CW)

Control systems coursework brief.pdf
DC Motor Control Lab Sheet 2023.pdf
University of Michigan Control Tutorials

Testing

Room: 3236
Login: students3236
Password: 3236

Technical team room: 3231
Christopher Phillopson room: 3215

Report Plan

1. INTRODUCTION, BACKGROUND, ENGINEERING PRINCIPLES: 20%

In this part, include a review of the application of DC motors in consumer products or in industry.
In what circumstances are DC motors preferred to other types?
Advantages

- Easy speed control
- High torque at low speeds
- Easy to power with batteries

In this coursework the motor under consideration is a brushed DC motor. How does this differ from a brushless DC motor?
How is the motor modelled?

Using your own words, summarise and explain the physical principles behind a brushed DC motor. You may refer to the DC motor modelling section of the University of Michigan Control Tutorials [3]:
https://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition§ion=SystemModeling
A transfer function is given in this reference together with a set of parameters. It is suggested that you familiarise yourself with this transfer function, as it will be used (Model 1) in Part 2.
There is no need to present any simulation results in this part of the coursework, but you may want to comment on the effect different physical parameters have on the open-loop step response.

2. MODELLING AND SIMULATION: 20%

Plot loaded data

```
%% Lab data plot
load LabData.mat Time Motor Model;
Time = Time-Time(1);
figure;
    plot(Time, Motor, 'r-','DisplayName', 'Experimental');
```

```
hold on
plot(Time, Model, '-','DisplayName', 'Modle');
hold off
title('Experimental Shaft velocity data');
xlabel('Time (s)');
ylabel('Veolcity (rad/s)');
grid on;
legend('Location', 'southeast');
```

Model 1

"*Use the transfer function from Part 1, together with the data below in Table 2 to create "Model 1" of the DC motor plant. Where parameters are missing you should estimate them (e.g., no viscous damping constant is given) – stating clearly what you have done.*"

Transfer function:

$$\frac{\theta}{V}(s) = \frac{k}{s(Js+b)(Ls+R)+k^2}$$

Explanation

- Rotor position = (rotor velocity / s)
- Rotor velocity

$$\frac{\dot{\theta}}{V}(s) = \frac{k}{(Js+b)(Ls+R)+k^2}$$

The angular acceleration of the rotor shaft ($\ddot{\theta}$) is related to torque (T), inertia(J) & damping (b).

$$T = J\ddot{\theta} + b\dot{\theta}$$

The motor torque constant (K_t) and the back electromotive force constant (K_b) will be treated as the motor constant(K).
Substituting for the motor constant (K) with $T = Ki$, and finding current (i) in terms of angular velocity using $V = Ri + e$ & $e = K\dot{\theta}$ gives:

$$K\frac{(V - K\dot{\theta})}{R} = J\ddot{\theta} + b\dot{\theta}$$

Which can be simplified & rearranged to:

$$\frac{\dot{\theta}}{V}(s) = \frac{k}{(Js+b)(Ls+R)+k^2}$$

and taking a Laplace transform of both sides gives position

$$\frac{\theta}{V}(s) = \frac{k}{(Js+b)(Ls+R)+k^2} \times \frac{1}{s}$$

Given parameters:

symbol	description	value	unit
------	**Motor**	-------	----------

symbol	description	value	unit
J_m	Rotor moment of inertia	1.16E-6	$kg \cdot m^2$
b_m	Viscous friction constant		
K_b	Electromotive force constant	0.0502	$N \cdot m/A$
K_t	Motor torque constant	0.0502	$N \cdot m/A$
R_m	Electrical resistance	10.6	$\dot{U}$
L_m	Electrical inductance	0.82	mH
	Max continuous Torque	0.035	$N \cdot m$
	Power rating	18	W
M_1	Inertial load disk mass	0.068	kg
r_1	Inertial load disk radius	0.0248	m
------	**Linear Amplifier**	-------	----------
V_{max}	max output voltage	15	V
	max output current	1.5	A
	max output power	22	W
	Gain	3	V/V

Matlab notes

2023-11-20 @ 23:25

- Added relevant parameters
- Added transfer function for motor position
  $\frac{\dot{\theta}}{V}(s) = \frac{k}{(Js+b)(Ls+R)+k^2}$

```
J = 1.16E-6;
b = ?; %%example:3.5077E-6
K = 0.0502;
R = 10.6;
L = 8.2E-5;
s = tf('s');
P_motor = K/(s*((J*s+b)*(L*s+R)+K^2))
```

2023-12-07 @ 05:03

- I accounted for the inertia of the loads

```
M1 = 0.068;  % Inertial load disk mass (kg)
r1 = 0.0248; % Inertial load disk radius (m)

J1 = 0.5 * M1 * r1^2;  % Inertia of the load disk
```

- I got the script to output numerators & denominators for the Simulink transfer function blocks

```
[V_tf_n, V_tf_d] = tfdata(V_shaft, 'v');
[P_tf_n, P_tf_d] = tfdata(P_shaft, 'v');
```

- Imported the lab data
- Plotted the lab data
- added transfer function for angle as well as position

```
V_shaft = K/( (J_total*s+b)*(L*s+R)+K^2 );
P_shaft = K / (s * ((J_total * s + b) * (L * s + R) + K^2));
```

- messed about for ages trying to get Simulink to use the time variable form labData until I realised that didn't solve my problem anyway so just set the labData to start from 0.14s to match. `Time = Time-Time(1)+0.14;`
- Made a figure compare my model with the lab values
- predicted rotor position based on lab values for motor angle
  `position = cumtrapz(Time, Motor);`
  not sure why its cumtrapz instead of just trapz but it works

```
%clear, close all

%% defining variables
J = 1.16E-6; % Rotor moment of inertia
b = 3.3E-5; % Viscous friction constant
K = 0.0502; % Motor torque / back emf constant
R = 10.6;
L = 8.2E-5;
s = tf('s');

% Transfer functions
V_shaft = K/( (J*s+b)*(L*s+R)+K^2 );
P_shaft = K/(s*((J*s+b)*(L*s+R)+K^2));

%% Account for the load
M1 = 0.068;  % Inertial load disk mass (kg)
r1 = 0.0248; % Inertial load disk radius (m)

J1 = 0.5 * M1 * r1^2;  % Inertia of the load disk
J_total = J + J1;      % Total inertia

% Updated transfer functions
V_shaft = K/( (J_total*s+b)*(L*s+R)+K^2 );
P_shaft = K / (s * ((J_total * s + b) * (L * s + R) + K^2));

%% Simulink variables
[V_tf_n, V_tf_d] = tfdata(V_shaft, 'v');
[P_tf_n, P_tf_d] = tfdata(P_shaft, 'v');
load LabData.mat Time;
Time = Time-Time(1); %+0.13998
timeMatrix = [linspace(0, (length(Time)-1)*0.01, length(Time))', double(Time)];
sim MotorSim1;

load LabData.mat Motor Model;
position = cumtrapz(Time, Motor);
%% Lab data plot
figure;
    plot(Time, Motor, 'r-','DisplayName', 'Experimental');
```

```
    hold on
    plot(Time, Model, 'b-','DisplayName', 'Modle');
    hold off
    title('Experimental Shaft velocity data');
    xlabel('Time (s)');
    ylabel('Veolcity (rad/s)');
    grid on;
    legend('Location', 'southeast');
%% Model plots
figure;
        % Input voltage
        subplot(3, 1, 1);
                plot(out.Voltage, 'r-');
                title('Input voltage');
                xlabel('Time (s)');
                ylabel('Voltage (V)');
                grid on;
        % Velocity
        subplot(3, 1, 2);
                plot(Time, Motor, 'b-','DisplayName', 'Experimental');
        hold on
        plot(out.V_shaft, 'r-','DisplayName', 'Modle');
        hold off
                title('Shaft velocity model vs lab data');
                xlabel('Time (s)');
                ylabel('Veolcity (rad/s)');
                grid on;
        legend('Location', 'southeast');
        % Position
        subplot(3, 1, 3);
                plot(Time, position, 'b-','DisplayName', 'Experimental estimate');
        hold on
        plot(out.P_shaft, 'r-','DisplayName', 'Modle');
        hold off
                title('Shaft position model vs lab data');
                xlabel('Time (s)');
                ylabel('Position (rad)');
                grid on;
        legend('Location', 'southeast');
```

Basic transfer function

"Now create a Simulink version of the simple motor model using the signal generator and transfer function blocks. Figure 4 shows how this may look. The plant transfer function is $G(s) = \frac{K}{\tau s+1}$ and the parameters K and τ need to be adjusted.
Confirm the model gives the same or similar results as the loaded 'Model' data. Show a plot comparing the two signals.**
Tune the parameters to give a good representation of the actual motor result.
Present your results and note the values of K and τ, as well as including a brief summary of your method"

My summary

- tune a basic transfer function block to look like labData.motor
- Plot your model against labData.motor

- Record your method

Compare the two models

"Now add the transfer function of Model 1 to the Simulink diagram, so you can compare the two, using the same input voltage.
Compare angular velocity and angle responses. You will need to make some adjustments since Model 1 predicts angle and Model 2 predicts speed!
Also compare pole locations. Comment on similarities and differences. In any case you will use Model 2 for the remainder of the assignment, since we have confidence that it represents the actual plant dynamics."

My summary

- Compare basic tf to tf based on $\frac{\dot{\theta}}{V}(s) = \frac{k}{(Js+b)(Ls+R)+k^2}$
- Find pole locations for both & compare

PART 3. PID CONTROLLER DESIGN AND SIMULATION: 20%

- Expand your Simulink model (Model 2) to include a PID controller for rotor angle,
- using the signal generator to create a square-wave reference signal, with amplitude = 1 radian and frequency 0.4 Hz.
- Selecting the PID gains kp = 1, kd = 0, ki = 0, you should obtain the result shown in figure 6.
- manually tune the PID controller to improve tracking of the square-wave.

kp = 5
kd = 0.15
ki = 0.2

Bsp = 1
bsd = 0
tf = 0.0060

- Describe the method you use to tune the three PID gains, and how each affects control performance.
- After you obtain your best result, compare with the initial result similar to Figure 6 and point out which performance metrics have improved and by how much.
- The actual motor has a limited input voltage of 15V. use the PID Advanced tab to mimic this (set the upper and lower saturation limits to 15 and -15). How does this affect your results?
- Test your controller with another type of reference signal, e.g. a sine or What limitations are there, if any, when you the reference at
  - (i) high frequency (5 Hz say)
  - (ii) low frequency (0.2 Hz say)?
- Include plots and brief descriptions to explain your results.

PART 4. PID CONTROL EXPERIMENT (LAB TESTING): 20%

2023-12-07 @ 05:49

1. test the PID controller that was designed in simulation,
2. see how real-world performance matches simulation for step changes in the reference,
3. test robustness when weights are added to the rotor.

PID controller in the DCMCT takes the expanded form:

$$U(s) = k_p(b_{sp}R(s) - Y(s)) + k_{ds}s(b_{sd}R(s) - Y(s)) + k_i\frac{1}{s}(R(s) - Y(s))$$

Reference weights: b_{sp} & b_{sd}
The reference weights include additional direct response to the reference signal. The term bsp can potentially be used to offset any steady-state error, but where integral control is used this is not required, so please always select bsp = 1. The term bsd would also normally also be set to 1, but there may be an advantage of setting it to zero – test this out and try to explain any difference you see. You should perform step and sinusoid tests when looking at this part.

Recovering my lab data

2023-12-11 @ 21:59
of the 10 experiments run 6 where corrupted as the file names & automatically generated variables began with a number.

```
joeashton@pop-os:~/Sync/Obsidian/SuperValult/Projects/Uni Projects/Courseworks/Control
systems/myLabData$ ls
'1a weights.mat'      '2a weights.mat'      'no weights F4.mat'
'1b weight.mat'       '2b weights.mat'      'no weights.mat'
'2a weights A0.mat'   'no weights A0.mat'
'2a weights A10.mat'  'no weights A10.mat'
```

For further information see this thread on the Matlab forum.

Python script

2023-12-12 @ 22:15
From the thread I copied this python code that appears to solve the issue:

```python
import scipy.io as sio
import pathlib
import string

def fix_name(name):
    name="".join(c for c in name if c in string.ascii_letters+string.digits+"_")
    if name[0] not in string.ascii_letters:#name must start with a letter
        name="X"+name
    name=name[:63]#names may not be longer than 63 chars
    return name

ignore_names=["__header__","__version__","__globals__"]

def fix_struct(s):
    out_struct=dict()
    for name,value in s.items():
        if name not in ignore_names:
            out_struct[fix_name(name)]=value
    return out_struct

def struct_needs_fixing(s):
    for name in s.keys():
        if name not in ignore_names:
```

```
            if fix_name(name)!=name:
                return True
    return False

root=r"C:\someplace\folder_with_the_files"
root=pathlib.Path(root)
files=list(root.glob("*.mat"))#get all the .mat data files in the directory

Overwrite=False #overwrite files in place

for f in files:
    new_f=f.parent/("fixed_"+f.name)#save the modified file with a prefix
    if Overwrite:new_f=f
    struct=sio.loadmat(str(f))
    if struct_needs_fixing(struct):
        print("fixing",str(f))
        struct=fix_struct(struct)
        sio.savemat(new_f,struct)
    else:
        print("skipping",str(f))
```

Instead of fixing the issue it just deleted all the corrupted lab data without creating any of the fixed files.
Luckily I do all my work in a Syncthing directory that is constantly synchronised across 3 devices with staggered versioning making retrieving the lost data a non issue.
I thought this error could be to do with the way I have my file permissions set so I temporarily `sudo chmod 777` all the relevant files & the script itself moving everything into the same directory and tried again with no luck.
Its not spitting any compile errors but I think the script is failing to import its directories through pip as it's not loading in the correct conda environment. it still fails when launched with `python3` & when launched as a bare executable with `#!/usr/bin/env python` as first line.
Made a brand new conda environment with the required dependencies and ran it in there still nothing: `(FixMatlab) joeashton@pop-os:~/Sync/Obsidian/SuperValult/Projects/Uni Projects/Courseworks/Control systems/myLabData$ python3 invalidMatFix.py`
This isn't working I will ask some advice form a physicist friend who's more experienced in both Matlab & python.

MEX script

2023-12-12 @ 02:22
There was also a C++ script on the above mentioned thread. Matlab allows the use of C++ scripts as MEX functions (See documentation).

Method:

terminal
`touch RetrieveData.cpp`
`chmod 777 RetrieveData.cpp`
`gedit RetrieveData.cpp`

- paste:

```
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
```

```
  char *Name;
  mxArray *Var;

    Name = mxArrayToString(prhs[0]);
    Var  = mexGetVariable("caller", Name);
    if (Var != NULL) {
       plhs[0] = mxDuplicateArray(Var);
    } else {
       plhs[0] = mxCreateDoubleMatrix(0, 0, mxREAL);
    }
    mxFree(Name);
    return;
  }
```

- save & close;
  Matlab live script

```
clear
mex -setup c++
mex RetrieveData.cpp
A = RetrieveData('a2 weights.mat')
```

- Output: `A = []`

```
MEX configured to use **'g++'** for C++ language compilation.

Building with 'g++'.
MEX completed successfully.

A =

     []
```

No useful output tried using underscores & backslash for space, tried some sensible edits to the function code but no luck.

Second Mex attempt

Found this fairly reputable submission on a series of threads and gave it a try but it no longer compiles. With the least helpful compiler error I've seen all day:

```
Error using loadfixnames
Unable to compile loadfixnames.c
```

Makes sense it was last updated on Tue, 01 Aug 2017.

Second python attempt

2023-12-12 @ 04:59
Tried using the same python script again on a different machine with the hopes I was just missing something by doing it in miniconda on Linux. This time I went full Microsoft using jupyter notebook inside VS code on windows 10.
Finally success!

Results

2023-12-10 @ 16:38

Comparing model 1&2

Plots I should do

- Plot model 1 & 2 against labData.motor
- Plot avdev of model 1, 2 from labData.motor
- Do Bode & Rlocus plots for both & compare

Real world data

Plots I should do

- Compare model 2 w PID against real world w PID
  - show effects of weights
    - plot both graphs of 1 weight against no weights
    - plot both 2 weights against no weights
  - Show effect of amplitude
    - Find difference in avdev
  - Show effect of frequency
    - Find difference in avdev
  - Show compounding effects of weights & frequency
    - Find difference in avdev

Normalising lab data

2023-12-12 @ 02:02
I can't seem to find a Matlab function that automatically finds recurring patterns an allows you to truncate data to fit it like the screen of an oscilloscope dose automatically.
I'm going to try and figure out how to do it myself, though it could be a colossal waste of time compared to doing it manually.

Brainstorming

How is this going to work?

- run the signal generator column of the `_value` arrays by a comparator and index the array at points where it switches.
- Write a C++ script to Regex match the signal generator column of the `_value` arrays with an array of ones and import as a MEX function.
- Look at each dataset and manually find the second instance of a negative followed by a positive in each and record this as an offset associated with each respective dataset.

What do these have in common?

- Need to find the index of full wavelengths
  With an array of indexes for every full wavelength I can pick a set near the end of the sample to get the best data.
  One way of creating the index:
  `full_waves_index = strfind(no_weights_Value(:,2)', ones(1,49));`
  I don't like this as some have 49 1s in a row and others have 50.
  Improvement
  I need something more like a comparitor,
  `full_waves_index = find(diff(no_weights_Value(:, 2) > 0) > 0) + 1;`
  now I want an array with 4 full wavelengths near the end
  `Last_4_waves = no_weights_Value(full_waves_index(end-5):full_waves_index(end)-1,1);`
  2023-12-12 @ 02:52 damn that took pretty much a whole hour, I'm really not sure about this coding for a living thing anymore.
  now as a function

```
function L4w = Last_4_waves(Value)
        full_waves_index = find(diff(Value(:, 2) > 0) > 0) + 1;
        L4w = Value(full_waves_index(end-5):full_waves_index(end)-1,1);
end
```

can't plot without time-series may as-well add that so its consistent

```
function L4w = Last_4_waves(Value)
full_waves_index = find(diff(Value(:, 2) > 0) > 0) + 1;
L4w = [linspace(0, 10, 1000)',Value(full_waves_index(end-11):full_waves_index(end-11)+999,:)];
end
```

never mind terns out you don't need a time series scratch that.

Conclusion

2023-12-13 @ 18:35

- It is possible to model motors easily to a high degree of accuracy based on their specs or a sample of test data.

Main MATLAB code

```
%% clear
clear
close all

%% defining variables
J = 1.16E-6; % Rotor moment of inertia
b = 3.3E-5; % Viscous friction constant
K = 0.0502; % Motor torque / back emf constant
R = 10.6;
L = 8.2E-5;
s = tf('s');

% Transfer functions
```

```
V_shaft = K/( (J*s+b)*(L*s+R)+K^2 );
P_shaft = K/(s*((J*s+b)*(L*s+R)+K^2));

%% Account for the load
M1 = 0.068;  % Inertial load disk mass (kg)
r1 = 0.0248; % Inertial load disk radius (m)

J1 = 0.5 * M1 * r1^2;  % Inertia of the load disk
J_total = J + J1;      % Total inertia

% Updated transfer functions
V_shaft = K/( (J_total*s+b)*(L*s+R)+K^2 );
P_shaft = K / (s * ((J_total * s + b) * (L * s + R) + K^2));

%% Simulink variables
[V_tf_n, V_tf_d] = tfdata(V_shaft, 'v');
[P_tf_n, P_tf_d] = tfdata(P_shaft, 'v');

%% Load lab data
Time = linspace(0.14,10.14,1000); %+0.13998 shift timeseries to fit labData
%timeMatrix = [linspace(0, (length(Time)-1)*0.01, length(Time))', double(Time)];
sim MotorSim1;
sim PIDmodel.slx;
load LabData.mat Motor Model;
position = cumtrapz(Time, Motor);

%% provided Lab data plot
figure(3);
    plot(Time, Motor, 'b-','DisplayName', 'Experimental');
    hold on
    plot(Time, Model, 'r-','DisplayName', 'Provided Modle');
    hold off
    title('Experimental Shaft velocity data');
    xlabel('Time (s)');
    ylabel('Veolcity (rad/s)');
    grid on;
    legend('Location', 'southeast');

%% Model plots
figure(2);
        % Input voltage
        subplot(3, 1, 1);
                plot(out.Voltage, 'r-');
                title('Input voltage');
                xlabel('Time (s)');
                ylabel('Voltage (V)');
                grid on;
        % Velocity
        subplot(3, 1, 2);
                plot(Time, Motor, 'b-','DisplayName', 'Experimental');
        hold on
        plot(out.V_shaft, 'r-','DisplayName', 'Theoretical Model');
        hold off
                title('Shaft velocity model vs lab data');
                xlabel('Time (s)');
```

```
                ylabel('Veolcity (rad/s)');
                grid on;
        legend('Location', 'southeast');
        % Position
        subplot(3, 1, 3);
                plot(Time, position, 'b-','DisplayName', 'Experimental Estimate');
        hold on
        plot(out.P_shaft, 'r-','DisplayName', 'Theoretical Model');
        hold off
                title('Shaft position model vs lab data');
        labels()
                % xlabel('Time (s)');
                % ylabel('Position (rad)');
                % grid on;
        % legend('Location', 'southeast');

%% Compare performance
V_shaftD = 17.491/(0.08*s+1);
figure(1);
    plot(out.Voltage1, 'g','DisplayName', 'Input signal (V)');
    hold on
    plot(out.V_TheoryModel, 'b-','DisplayName', 'Theory based model (rad/s)');
    hold on
    plot(out.V_DataModel, 'r--','DisplayName', 'Data driven model (rad/s)');
    hold off
    title('Theory vs Data Based Models');
    xlabel('Time (s)');
    ylabel('Veolcity (rad/s) / Voltage (V)');
    grid on;
    legend('Location', 'southeast');


%% load my lab data
load myLabData/'no weights.mat'
load myLabData/'no weights A0.mat'
load myLabData/'no weights A10.mat'
load myLabData/'no weights F4.mat'

load myLabData/'fixed_a1 weights.mat'
load myLabData/'fixed_b1 weight.mat'

load myLabData/'fixed_a2 weights.mat'
load myLabData/'fixed_b2 weights.mat'

load myLabData/'fixed_a2 weights A0.mat'
load myLabData/'fixed_a2 weights A10.mat'

%% My lab data plots
%% Varying amplitude
figure;
    subplot(2,2,1:2);
        plotme(no_weights_Value);
        title("Amplitude 1 (rad)");
    subplot(2,2,3);
        plotme(no_weights_A0_Value);
```

```
        title("Amplitude 0.1 (rad)");
    subplot(2,2,4);
        plotme(no_weights_A10_Value);
        title("Amplitude 10 (rad)");
    legend({'Rotor Position (rad)','Input Signal (rad)'},  'Location', 'northeastoutside');
%% Single weights
figure;
    plot(Last_10_wavesD(X_1b_weight_Value));
    hold on
    plotme(no_weights_Value);
    title("Effect of Single Weight");
    legend({'Rotor Position with brass weight (rad)','Rotor Position without brass weight
(rad)','Input Signal (rad)'},  'Location', 'southeast');
%% double weights
figure;
    hold on
    plot(Last_10_wavesD(X_2b_weights_Value));
    plot(Last_10_waves(X_2a_weights_Value));
    title('Double weights response');
    labels();
    legend({'Rotor Position with 2 brass weights (rad)','Rotor Position with 2 aluminium weights
(rad)','Input Signal (rad)'},  'Location', 'southeast');
%% frequency
figure;
    hold on
    plot(Last_n_waves(no_weights_F4_Value,40,199));
    title("4Hz Frequency Response");
    legend({'Rotor Position (rad)','Input Signal (rad)'},  'Location', 'northeast');
    labels();
%% Funks
function labels()
    xlabel('Time (s)');
    ylabel('Position (rad)');
        grid on;
    % legend('Location', 'southeast');
end

function Ltw = Last_10_waves(Value)
    full_waves_index = find(diff(Value(:, 2) > 0) > 0) + 1;
    Ltw = [Value(full_waves_index(end-11):full_waves_index(end-11)+999,:)]; %linspace(0, 10,
1000)',
end
function Ltw = Last_10_wavesD(Value)
    full_waves_index = find(diff(Value(:, 2) > 0) > 0) + 1;
    Ltw = [Value(full_waves_index(end-11):full_waves_index(end-11)+999,1)]; %linspace(0, 10,
1000)',
end

function Ltw = Last_n_waves(Value,n,t)
    full_waves_index = find(diff(Value(:, 2) > 0) > 0) + 1;
    Ltw = [Value(full_waves_index(end-n-1):full_waves_index(end-n-1)+t,:)]; %linspace(0, 10,
1000)',
end

function plotme(Value)
```

```
        plot(Last_10_waves(Value));
        labels();
end
```