Machine Learning assisted tuning of PID Controller

Ishan Prakash - 121

Keshav Singh - 134

1/06/2021

—

Control Systems

—

Prof. Bhavnesh Jaint

INDEX

INTRODUCTION

The aim of our project is to explore new methods and technologies that are being used in research as well as in the industry and integrate them with our working knowledge of the subject of control systems.

The objective is to utilize concepts learnt from both theory and practical lectures and integrate them with new technologies in the field of Control. We do so to improve our own knowledge and also to provide use cases for these technologies.

We have incorporated concepts like modelling transfer function, working with various systems like SISO, MIMO and MISO. We have also gone deeper to understand how complex mathematical modelling of such systems are done using state spaces, etc.

OUR CONTRIBUTION

How to Tune a PID? This is a question that doesn’t have a definitive answer yet. Usually, methods which involve such uncertainty resolve to brute-forcing/trial and error approaches, but with the advent of Machine Learning these methods can be replaced.

In our project we use machine learning algorithms to tune our PID controller. We present the Theory, Working and Simulation as proof of concept in later sections.

TUNING PID FOR CLOSED LOOP RESPONSE OF DC MOTOR

THEORY

Control theory deals with the control of **dynamical systems** in engineered processes and machines. The objective is to **develop a model or algorithm** governing the application of system inputs to drive the system to a desired state, while minimizing any *delay*, *overshoot*, or *steady-state error* and ensuring a level of control stability; often with the aim to achieve a degree of optimality.

To achieve this, we use a controller which promotes corrective behavior, this is done by monitoring the controlled variable and comparing it with a reference point. In this project we use a **PID controller** to achieve this task.

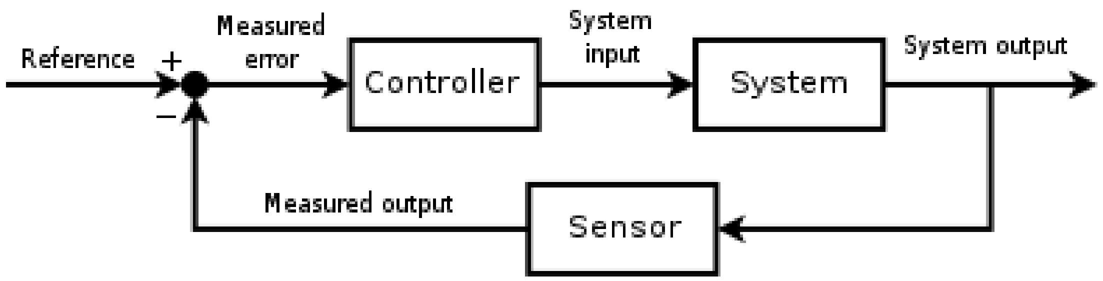


Figure1: Block Diagram of a Dynamic System

A proportional–integral–derivative controller is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an *error value* as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted *P*, *I*, and *D* respectively), hence the name.

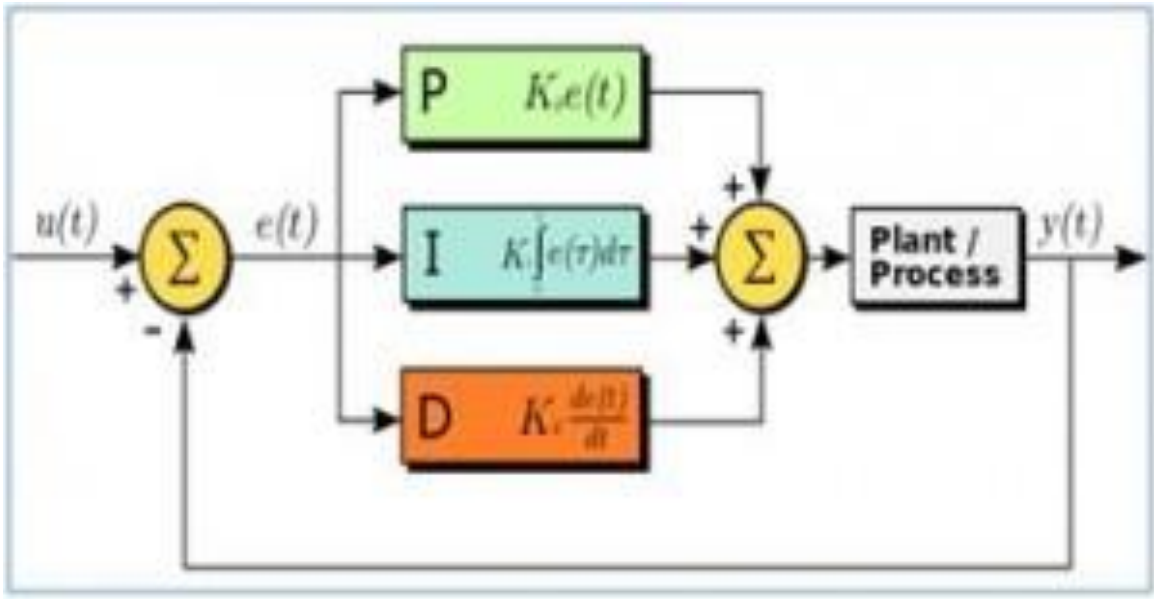


Figure2: PID controller block diagram

Here ***u(t)*** is the input signal/reference signal and ***y(t)*** the output signal. ***e(t)*** is the error signal which is fed into the controller.

$$c(t) = K_p e(t) + K_i \int edt + K_d \frac{de}{dt}$$

c(t) - control signal
K_p - Proportional gain
K_i - Integral gain
K_d – Differential gain

There are several techniques on how to tune these gains for optimal performance, the most common is listed below:

1. Set all gains to zero.
2. Increase the P gain until the response to a disturbance is steady oscillation.

3. Increase the D gain until the oscillations go away (i.e., it's critically damped).
4. Repeat steps 2 and 3 until increasing the D gain does not stop the oscillations.
5. Set P and D to the last stable values.
6. Increase the I gain until it brings you to the setpoint with the number of oscillations desired (normally zero but a quicker response can be had if you don't mind a couple oscillations of overshoot).

This method is based on Trial and Error, and though it works in most cases, this cannot be applied to fairly complex dynamical system.

This is where we would like to introduce MACHINE LEARNING BASED OPTIMIZATION OF PID CONTROLLER.

ML-Based Control Law for Dynamic Systems

Machine learning is a branch of Artificial Intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

Machine-learning algorithms use statistics to find patterns in massive amounts of data. And data, here, encompasses a lot of things—numbers, words, images, clicks, what have you. If it can be digitally stored, it can be fed into a machine-learning algorithm.

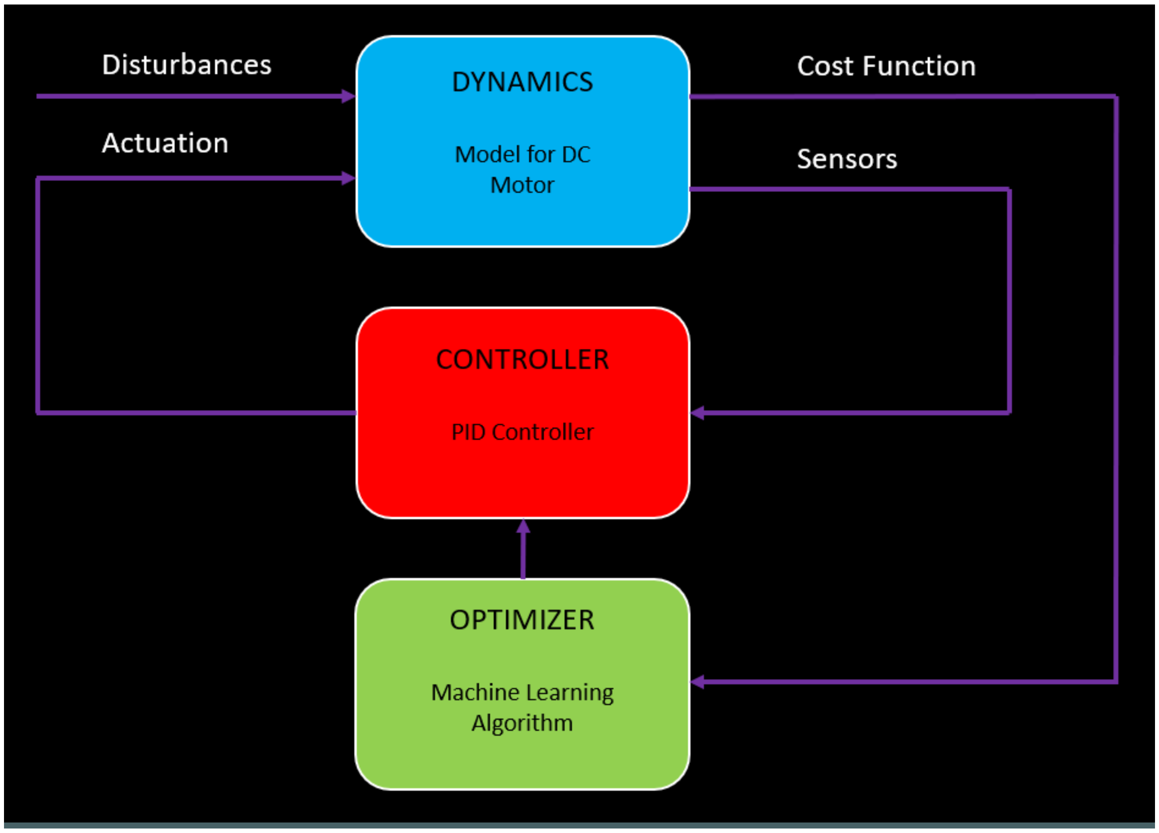
Machine learning is the process that powers many of the services we use today—recommendation systems like those on Netflix, YouTube, and Spotify; search engines like Google and Baidu; social-media feeds like Facebook and Twitter; voice assistants like Siri and Alexa. The list goes on.

We explain our approach using a simple block diagram shown below:

The blocks are explained below:

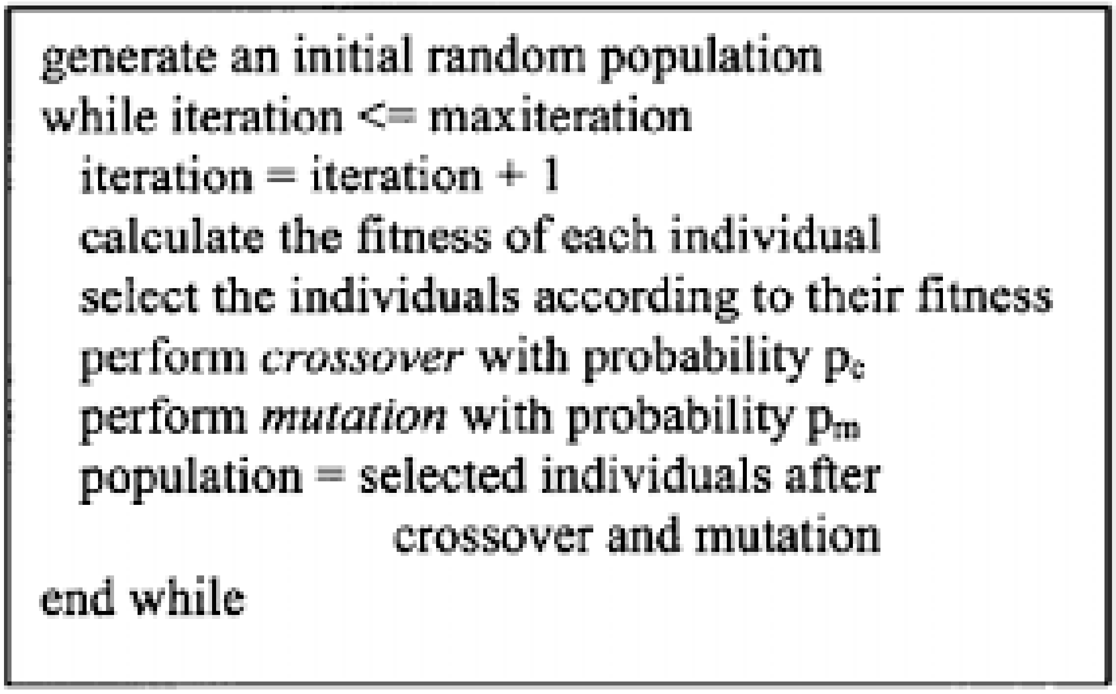
- Dynamical system maybe complex non-linear, multi-scale, high dimensional.
- Disturbances to the systems can be a gust of wind, break (reverse torque), something unmeasurable parameters that change the dynamics.
- There is ahigh dimensional cost function that we want to min/max.
- The Optimizer change the values of the individual gain for better performance on the cost function.
- Then the idea is to design a control law with the information acquired from the sensor and the cost function which can then lead to actuation.

Figure3: Generalized feedback control block diagram



GENETIC ALGORITHM

Genetic Algorithms (GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. **They are commonly used to generate high-quality solutions for optimization problems and search problems.**



```
generate an initial random population
while iteration <= maxiteration
    iteration = iteration + 1
    calculate the fitness of each individual
    select the individuals according to their fitness
    perform *crossover* with probability $p_c$
    perform *mutation* with probability $p_m$
    population = selected individuals after
                        crossover and mutation
end while
```

Figure4: Pseudocode for Genetic Algorithm

MATLAB SIMULATION

For our experiment we have simulated A DC motor system using the concept of Multiple Input Single Output. The details of the experiment are as follows:

- We have modelled the dynamic system of a DC Motor using its intrinsic properties.
- Input to the system is a step input with certain disturbances which we aim to stabalize.
- For optimization we use the Genetic Algorithm package provided by MATLAB.
- The LQR funciton is used as a Cost Funciton for the genetic algorithm.

PROCEDURE:

We first create a function to define the motor dynamics and inputs.

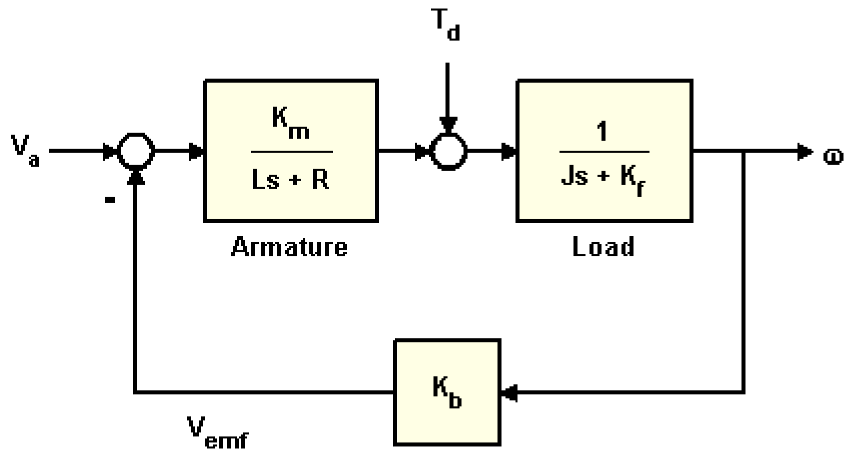


Figure5: A simplified model of a DC Motor

The code is as follows:

1. We define the parameters of the motors; Resistance, Inductance, Torque constant, Back EMF constant, Damping constant, Moment of Inertia.
2. As seen in the Figure5 we create 2 transfer functions namely h1 and h2.
3. We then use the state space function of MATLAB to create a transfer function as we also have disturbances in the system.
4. We then add the Back EMF and use the feedback function to complete the loop.
5. Now we add the PID controller using the output as the feedback.
6. Now we use the lsim function to give a step input for 20sec in steps of 0.001sec. The disturbances are modelled to be present in the system for 5-10 secs.
7. The response of the system is passes through the LQR function.
8. Our function return the Cost value, which is then used by the Genetic Algorithm.

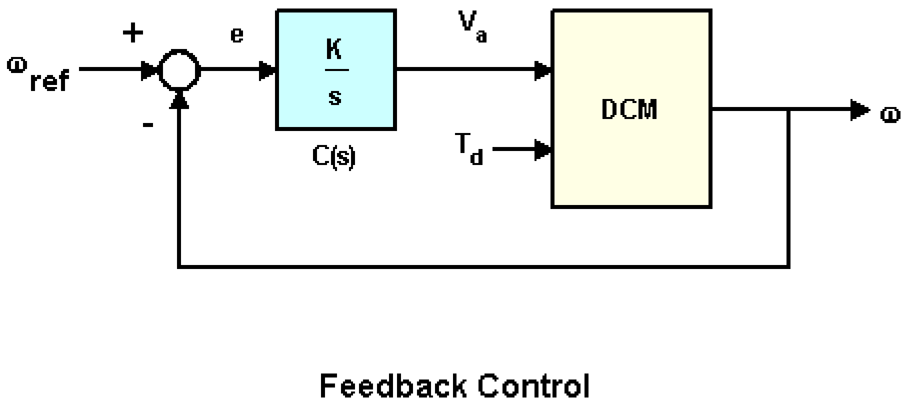
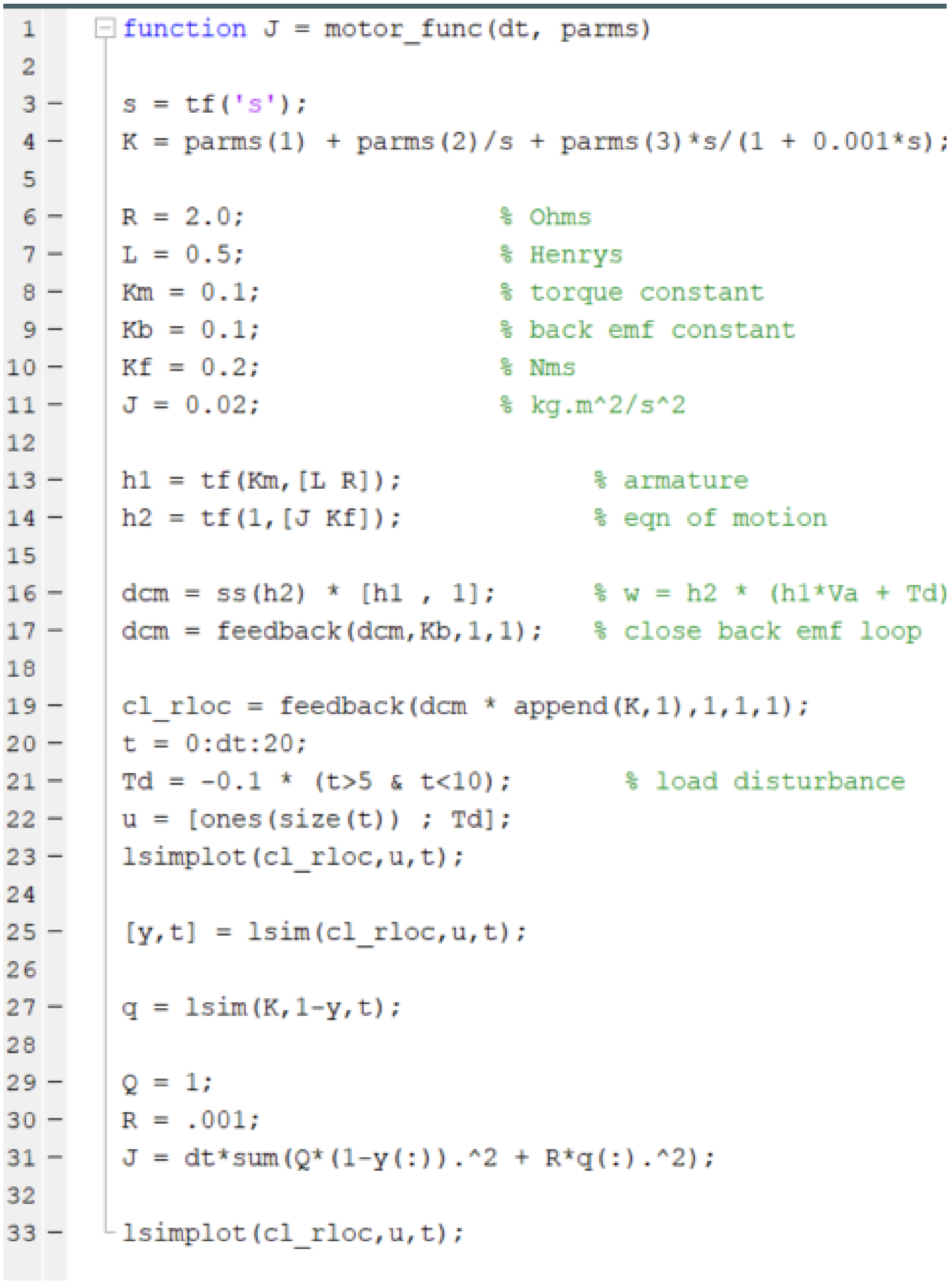


Figure6: Feedback control block diagram



```
function J = motor_func(dt, parms)

s = tf('s');
K = parms(1) + parms(2)/s + parms(3)*s/(1 + 0.001*s);

R = 2.0;                    % Ohms
L = 0.5;                    % Henrys
Km = 0.1;                   % torque constant
Kb = 0.1;                   % back emf constant
Kf = 0.2;                   % Nms
J = 0.02;                   % kg.m^2/s^2

h1 = tf(Km,[L R]);                % armature
h2 = tf(1,[J Kf]);                % eqn of motion

dcm = ss(h2) * [h1 , 1];          % w = h2 * (h1*Va + Td)
dcm = feedback(dcm,Kb,1,1);       % close back emf loop

cl_rloc = feedback(dcm * append(K,1),1,1,1);
t = 0:dt:20;
Td = -0.1 * (t>5 & t<10);          % load disturbance
u = [ones(size(t)) ; Td];
lsimplot(cl_rloc,u,t);

[y,t] = lsim(cl_rloc,u,t);

q = lsim(K,1-y,t);

Q = 1;
R = .001;
J = dt*sum(Q*(1-y(:)).^2 + R*q(:).^2);

lsimplot(cl_rloc,u,t);
```

Figure7: Motor Function code.

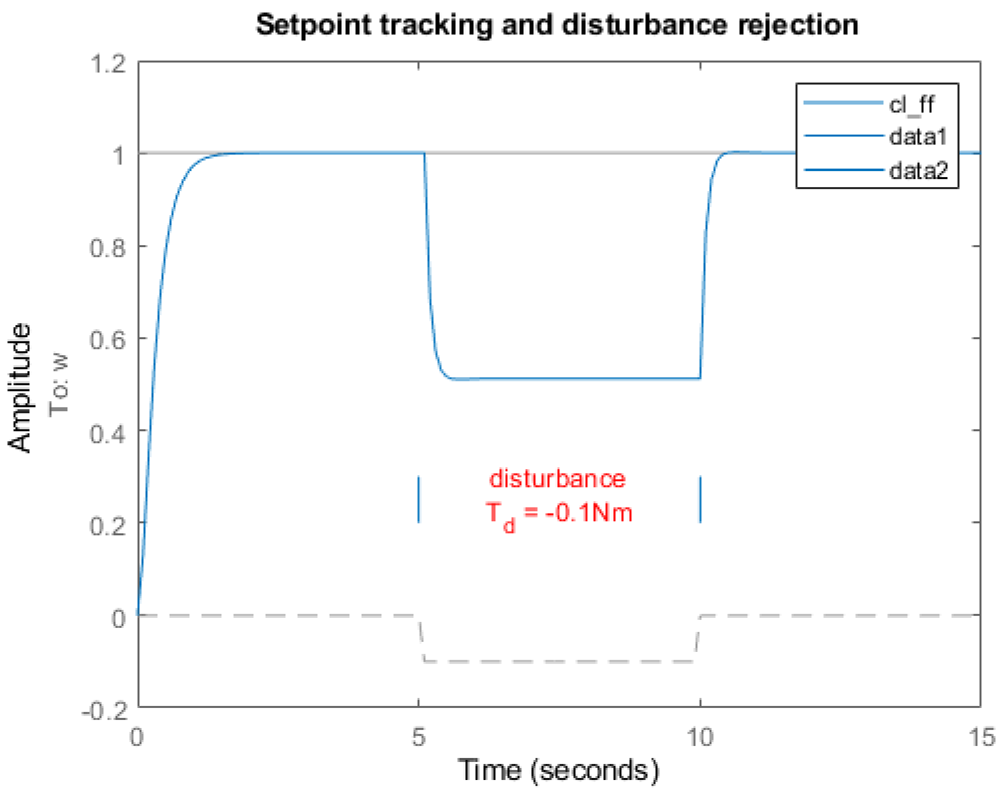
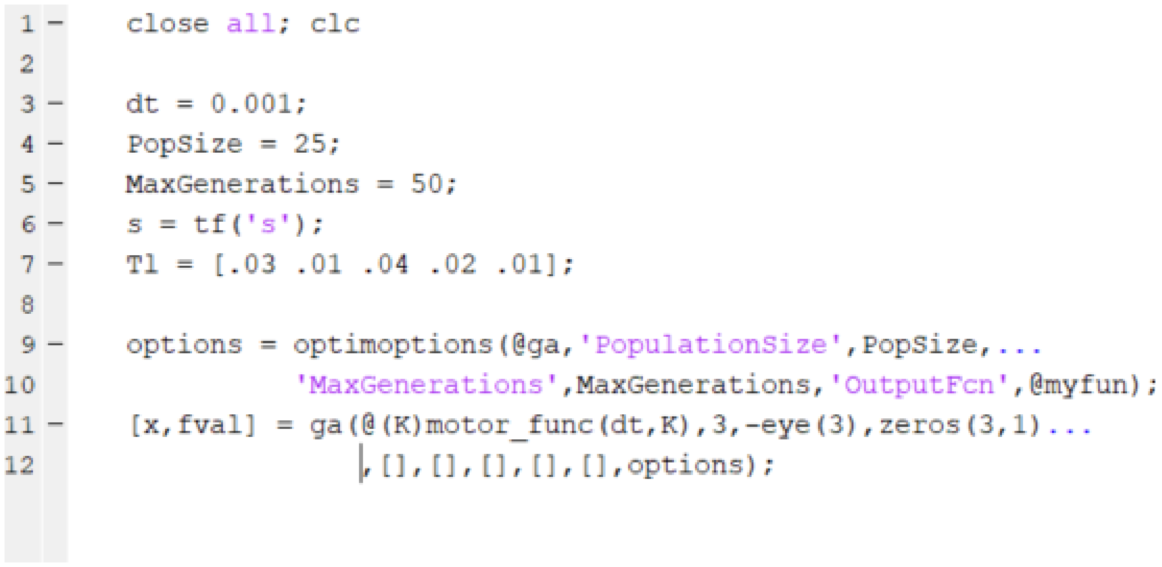


Figure8: The step input given to the motor

Now that we have modelled the Motor dynamics, we can focus on the genetic algorithm. We create another file , here the following steps are taken:

1. We first define the time step which we want , here it is 0.001sec.
2. We then define the parameters for the Genetic Algorithm, naemly population size (25) and maximum number of generations (50).
3. We set the options for the genetic algorithm using optimoptions here we pass the variable mentioned in the above point.
4. Now we use the GA function in MATLAB and pass our motor_funciton as the input parameter and wrap it around the PID gain K which we wish to optimize.
5. Originally the PID gain is set to -I(3).

The code is as follows:



```
close all; clc

dt = 0.001;
PopSize = 25;
MaxGenerations = 50;
s = tf('s');
Tl = [.03 .01 .04 .02 .01];

options = optimoptions(@ga,'PopulationSize',PopSize,...
            'MaxGenerations',MaxGenerations,'OutputFcn',@myfun);
[x,fval] = ga(@(K)motor_func(dt,K),3,-eye(3),zeros(3,1)...
                 ,[],[],[],[],[],options);
```

Figure9: Genetic algorithm execution

RESULTS

We initialized our PID gain value to $\begin{matrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{matrix}$. Here for simplification we have used defined the PID using a matrix where row1 represent Prportional gain, row2: Integral gain, row3: Differential gain.

Before simulating the uncontrolled response of the system is as follows:

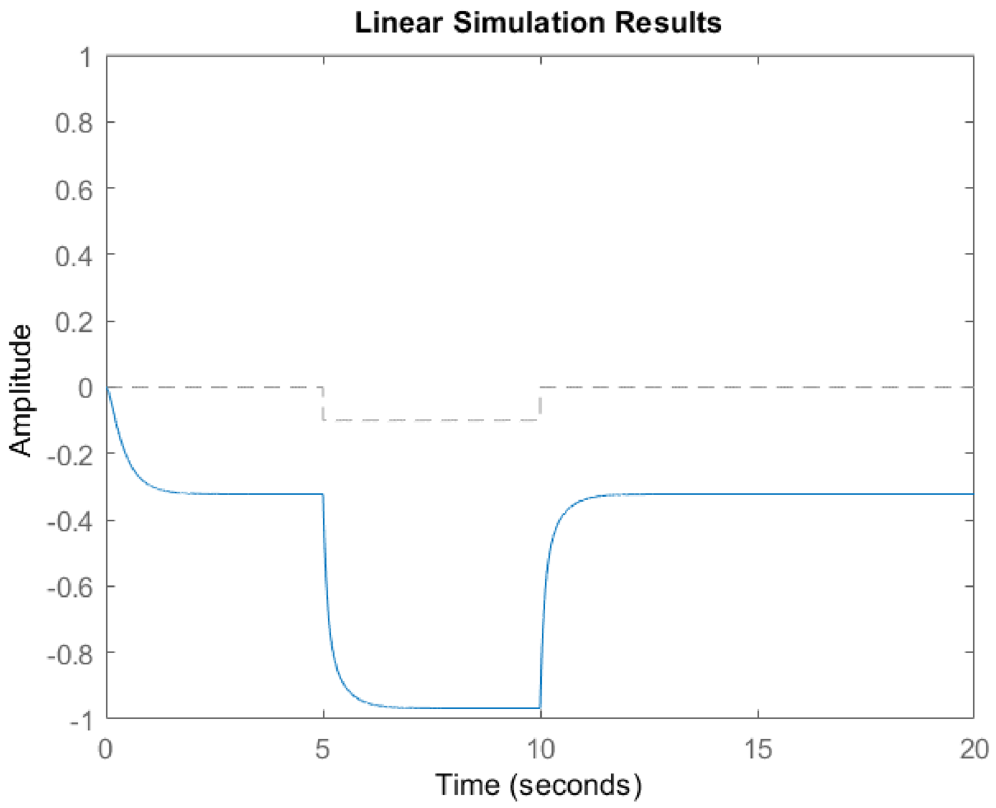


Figure9: Initial respose of the system

The initial value of the cost function is **45.1839.**

After running the simulation we get the following result.

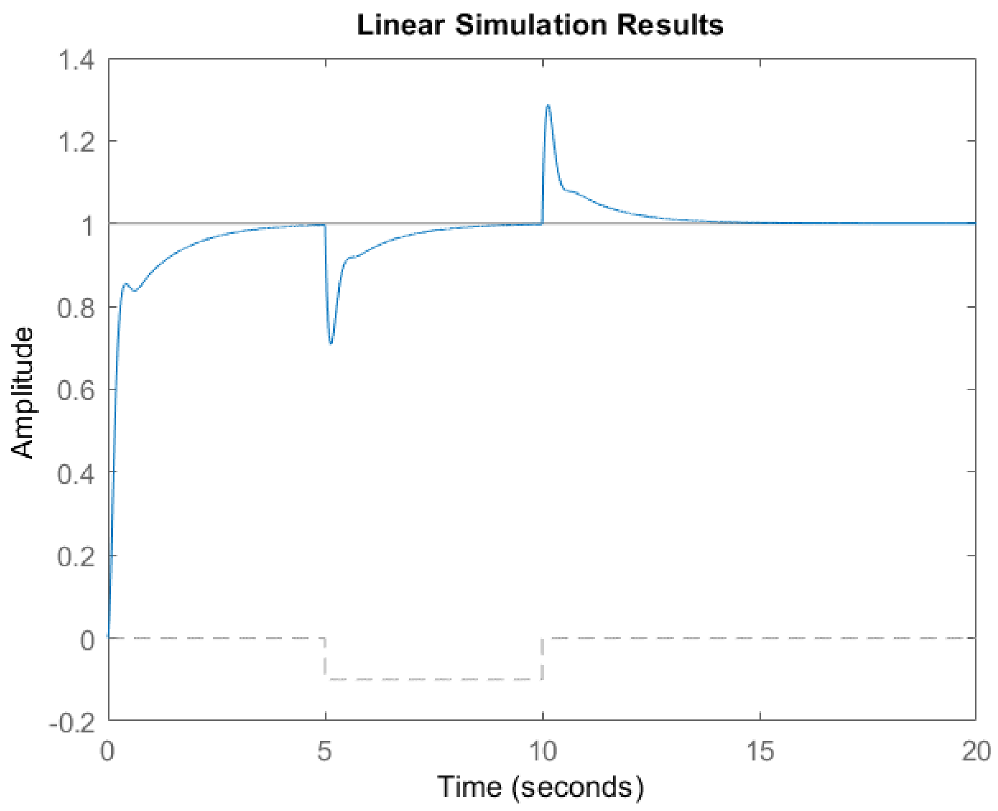


Figure10: ML-based tuned PID response of the motor

The final values after optimization as follows:
K_p - 8.4893
K_i - 10.1158
K_d – 0.0251
Cost function – **0.6283**

We can observe that the value of the cost function (J) has tremendously decreased from **45.1839** to **0.6283**.
This shows that the optimization algorithm was able to converge effectively.

Conclusion

In this project we were able to successfully able to prove that Machine Learning methods can perform better for tuning of a PID controller rather than calculated bruteforcing. This is also valid for other control laws. It observed that when modelling complex systems like disease models, stock analysis it is easier to simulate the conditions sing Machine Learning simlations.

Therefore Machine Learning can not only be used for modelling an appropirate control law but can also help s to design complex dynamical systems. Futher we will try to expand the work done in this report to more complex systems like Quadrotor control.

Submitted by:
Ishan Prakash - 2K19/EE/121
Keshav Singh – 2K19/EE/134