Department of Information Engineering and Computer Science (DISI)
University of Trento, Italy

Applied Robotics Lab

Winter Semester 2015-2016

Lego Motor Controller

Authors:
Pratheeksha Mankude Lambodara(181617)
Varun Gowtham (181832)
Vijay Krishna Chalamalasetty (181608)

Mentors:
Luigi Palopoli
Daniele Fontanelli

January 25, 2016

UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

Abstract

In this report of the course, we explain the procedure for identifying the parameters of a DC motor. The identified parameters are used to design and implement a discrete controller able to satisfy certain design criteria required by the course for a given reference angular speed. Further a wheeled robot was implemented using the controller.

1 Introduction

Lego Mindstorm kit provides a DC motor which can be interfaced to a Lego Brick. The motor can be controlled by feeding various levels of powers ranging from -100 to 100. The motor is also equipped with a counter with each increment or decrement of counter corresponding to a change of angle in 1 degree or -1 degree respectively. Figure 1a shows a picture of the motor supplied in the Lego Motor kit. Figure 1b shows the mathematical representation of a DC motor. In Figure 1b **R** and **L** are resistance and inductance of the armature of the circuit respectively, **J** is the inertia of the motor shaft, τ_m is the generated torque and τ_r is the is the torque load and **b** is the viscous friction of the motor shaft. The response of the motor in frequency domain for a step input with amplitude A is given by Equation (1).

$$\Omega_1(s) = \frac{k}{(JLs^2 + (bL - RJ)s + Rb + k^2)}\frac{A}{s} = \frac{q}{s(\frac{s^2}{\omega_n^2} + \frac{2\xi}{\omega_n}s + 1)} \tag{1}$$

Equation (1) shows that the DC motor is a second order system where

$$\omega_n = \sqrt{\frac{Rb + k^2}{JL}}$$

is the natural frequency of the motor in $rad\, s^{-1}$,

$$\xi = \frac{\sqrt{\frac{Rb + k^2}{JL}}(bL - RJ)}{2(Rb + k^2)}$$

is the damping factor and

$$q = \frac{kA}{Rb + k^2}$$

is the gain of the system for the step input with amplitude A.

Ω_1 has three poles, which are

$$p_1 = 0$$

and

$$p_{2,3} = -\xi\omega_n \pm j\sqrt{1 - \xi^2}\omega_n$$

Equation (1) can be used to obtain the time domain response of the motor. Equation (2) gives the equation of the time domain response of the motor.

$$\omega_1(t) = [q - qN \exp^{-\xi\omega_n t} \sin(\omega_n\sqrt{1 - \xi^2}t + \phi)]1(t) \tag{2}$$

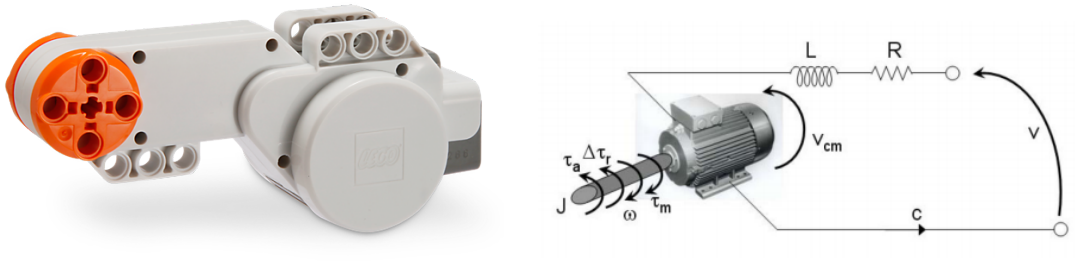
where :

$$N = \frac{1}{\sqrt{1 - \xi^2}}$$

and

$$\phi = \arctan\frac{\sqrt{1 - \xi^2}}{\xi}$$

In this report we explain the procedure used to identify the parameters ω_n ξ and k of the motor used to represent the motor mathematically in Equation (1). The identified parameters are used to design a discrete controller used in a closed loop to maintain a constant reference angular velocity. Using the designed controller, we implement a wheeled robot required to travel in a straight path at a given velocity.



(a) Lego Motor

(b) Model of Motor

Figure 1: DC Motor and Model

2 Paramter Identification

2.1 Aim

A parameter identification procedure is followed to identify the parameters derived from actual step response obtained from the motor. Further it is required to perform error analysis between the actual step response from the DC motor to the step response obtained from the estimated parameters. The parameters that we are going to analyze are the ω_n, ξ and k for different levels of input step values ranging from power percents 20 to 100.

2.2 Analysis Methodology

The important values that characterizes a second order system obtained from the time domain response Equation (2) are the overshoot (O) and the settling time (T_s). It is enough if O and T_s are found from the time domain response of a DC motor, we can deduce the the required values of the parameter ω_n ξ and k.

$$O = \frac{|\omega_{max} - \omega(\infty)|}{|\omega(0) - \omega(\infty)|} = \exp^{-\frac{\pi\xi}{\sqrt{1-\xi^2}}} \tag{3}$$

$$T_s \approx \frac{\log(\frac{\alpha}{100}) - \log(\frac{1}{\sqrt{1-\xi^2}})}{-\xi\omega_n} \tag{4}$$

$$k = \frac{q}{A} \tag{5}$$

Equation (3) gives the relationship of O with system parameters and relation to with the system output. In our case, the output of the system is considered to be angular velocity in $rad\,s^{-1}$ and α is the percent of steady state value. Equation (4) gives a relationship to obtain the value of ω_n from the system response. Equation (5) gives the gain of the system where q is the output after the settling time and A is the amplitude of the applied step input.

2.3 Analysis Procedure and Implementation

Figure 2 shows the block diagram representation of the experimental setup containing a computer as a server and Lego Brick as the client, connected via Bluetooth. The Brick is connected to a motor, which could be actuated from the brick with the help of a program. Figure 3a shows the flowchart of the client side program implementation. The client is programmed to set a motor speed, sample the motor count and the time difference and send it to server. Figure 11b shows the flowchart of the server side program implementation. The server is programmed to receive the motor count and computer the elapsed time based on the time difference and store it in a Comma Separated Variable (CSV) file. Each

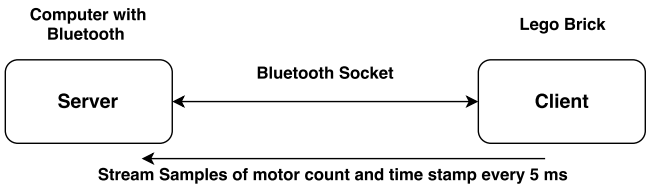
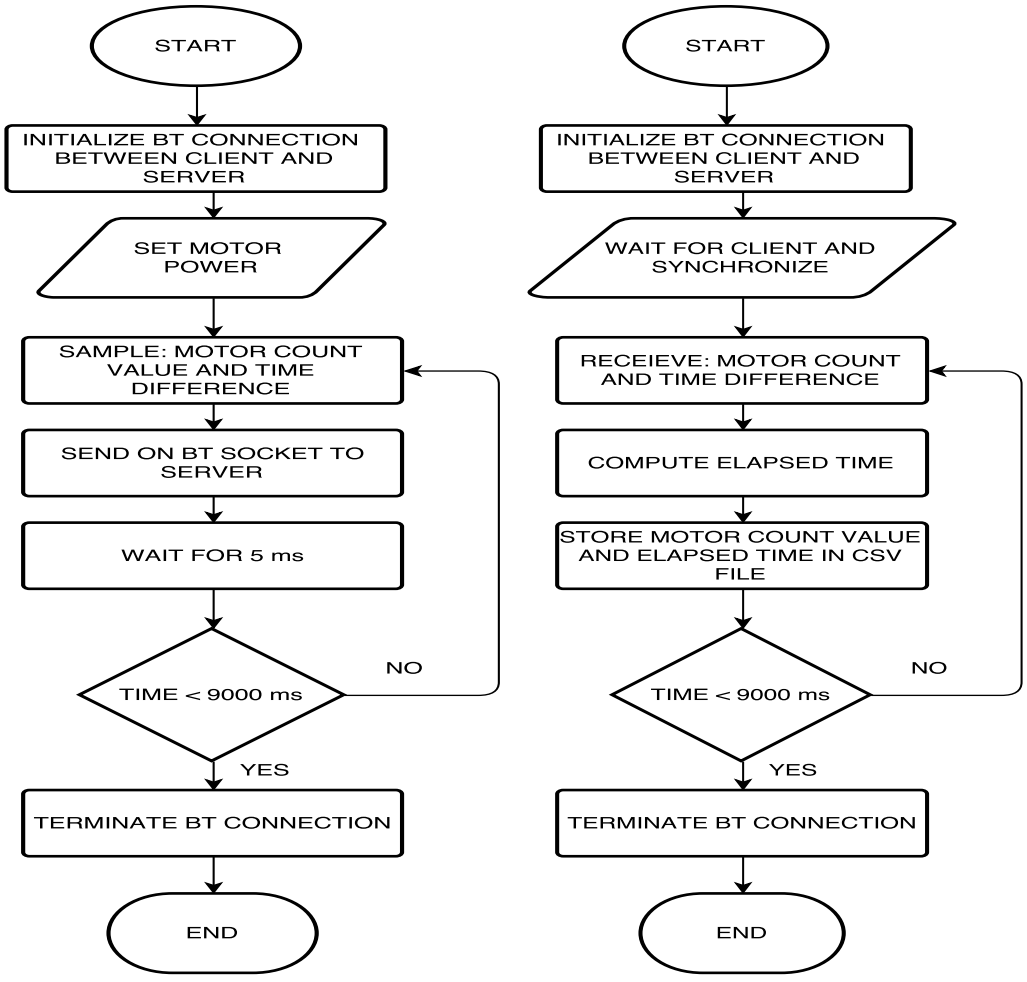


Figure 2: Client-Server experimental setup



(a) Client Flowchart

(b) Server Flowchart

Figure 3: System Flowchart for parameter identification

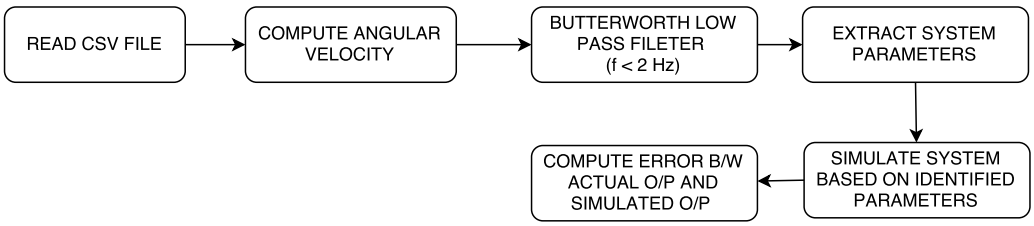


Figure 4: Analysis flow at the server

power level set at the client produces a corresponding CSV file on the server used for analysis. Figure 4 shows the block diagram of the analysis procedure followed at the server. We used Scilab for analysis. Following texts show the implementation details with code snippets used in analysis on Scilab.

2.3.1 Read CSV File

Read the CSV file into a matrix **M**.

```
M = csvRead(strcat(["CSV/5m_P",string(data),".csv"]))
```

2.3.2 Compute angular velocity

Vector **Omega** contains the angular velocity in $rad\ s^{-1}$ vs. milli seconds.

```
t = M(2:$,2)
for i = 2:(size(M,1)-1)
    prescount = M(i,3)
    presms = M(i,2)
    if initcount < prescount then
        calcOmega = ((prescount - initcount)*%pi*1000)/((presms-initms)*180)
        initms = presms
        initcount = prescount
        presOmega = calcOmega
    end
    Omega(i) = presOmega
end
```

2.3.3 Butterworth Low Pass filter

Pass Omega through Low Pass Filter Butterworth filter given by **iir** of Scilab and filter **Omega** to obtain filtered angular velocity Omega_out. Butterworth filter uses a **cut off frequency** of **2 Hz**. **Fs** is the sampling frequency used at the client side. The waveform shown Figure 5a belongs to actual system output for **motor power level set at 20**.

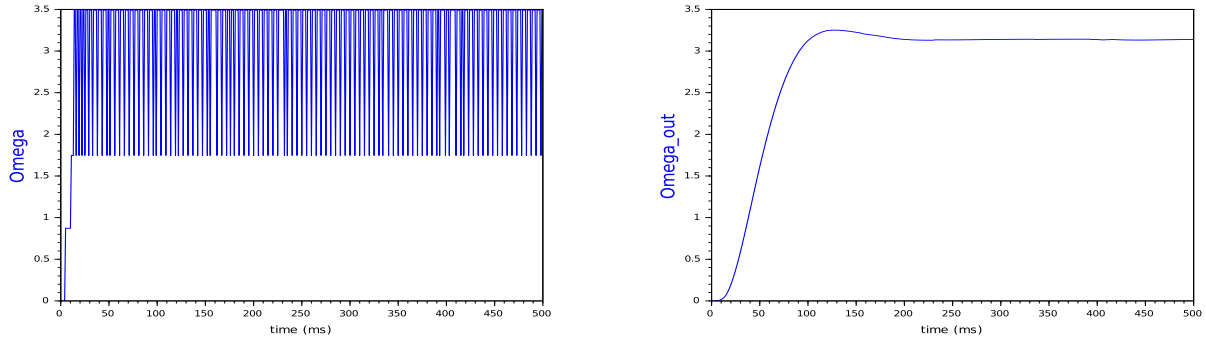
```
Fs = 200
hz = iir(2,'lp','butt',[2/Fs],[])
Omega_out = flts(Omega,hz)
```

Figure 5a shows a plot of the vector Omega derived directly from the precious step. Figure 5b shows the plot of vector Omega_out after passing Omega through the low pass filter.

2.3.4 Extract system parameters

From the vector of Omega_out it is now possible to estimate required system parameters using Equation (3) and Equation (4). We now calculate estimated values of ξ, ω_n and k in the form of ξ_{est}, ω_{n_est} and k_{est}.

$$\xi_{est} = \sqrt{\frac{\log(O)^2}{\pi^2 + \log(O)^2}} \tag{6}$$



(a) Plot of Omega: ω with noise

(b) Plot of Omega_out: ω filtered

Figure 5: ω before and after filtering through Butterworth lowpass filter

Solving for ξ from Equation (3), we get Equation (6). This can be accomplished from the following code snippet in Scilab. q_{est} is the last value of the vector Omega_out.

```
q_est = Omega_out($)
y_0 = Omega_out(1)
y_inf = q_est
y_max = max(Omega_out)
Overshoot = abs(y_max - y_inf)/abs(y_0 - y_inf)
xi_est = sqrt(log(Overshoot)^2/(%pi^2 + log(Overshoot)^2))
```

$$\omega_{n\_est} = \frac{\log(\frac{\alpha}{100}) - \log(\frac{1}{\sqrt{1-\xi^2}})}{\xi_{est} T_s} \tag{7}$$

Ts can be calculated from the Omega_out by sweeping through the vector from the last value towards the first sample. The value is considered settling time when, the value is beyond $\pm\alpha$ percent of the final value of Omega_out which is q_est. T_s can be found out from the following code snippet.

```
alpha = 5
Ts = -1
Omega_at_Ts = -1
for i = 250:-1:1
    if abs(q_est - Omega_out(i)) < alpha/100*q_est
        Ts = t(i)
        Omega_at_Ts = Omega_out(i)
    else
        break
    end
end
```

Solving for ω_n from Equation (4), we get Equation (7).

```
N_bar = 1/sqrt(1 - xi_est^2)
o_n_est = (log(alpha/100) - log(N_bar))/(-xi_est*Ts)*1000
```

$$k_{est} = \frac{q_{est}}{A} \tag{8}$$

Equation (8) can be used to estimate the system gain k_{est}, where A is the power percent value set in program at client ranging between 1 to 100.

```
k_est = q_est/A
```

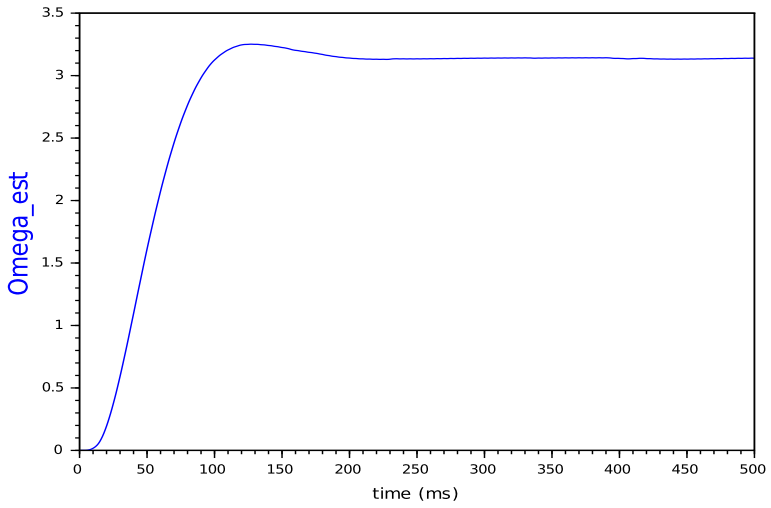


Figure 6: Plot of Omega_est: Simulated system output

2.3.5 Simulate system

The estimated values can be substituted into Equation (1) and used to simulate the system on using the **csim** function in Scilab.

```
// Simulation
time = 0:0.001:size(Omega,2)/Fs
s = poly(0, 's');
G_est = k_est/(s^2/o_n_est^2 + 2*xi_est/o_n_est*s + 1);
G_est = syslin('c', G_est);
Omega_est = csim(ones(1,size(time,2))*A, time, G_est);
```

Omega_est now contains the simulated output of the estimated system. Figure 6 shows a plot of the simulated output from the estimated parameters.

2.3.6 Model Verification

With the availability of Omega_out and Omega_est it is now possible to verify the system model by calculating performance indices. Following equations give the method to compute the performance indices.

$$(IntegralSquareError)ISE = \int_0^T (y_m(t) - y(t))^2\, dt$$

$$(IntegralAbsoluteError)IAE = \int_0^T |y_m(t) - y(t)|\, dt$$

$$(IntegralTimeSquareError)ITSE = \int_0^T t(y_m(t) - y(t))^2\, dt$$

$$(IntegralTimeAbsoluteError)ITAE = \int_0^T t|y_m(t) - y(t)|\, dt$$

where T is the upper bound to evaluate the integrals, $y_m(t)$ and $y(t)$ are the simulated and actual response of the system respectively. Following code snippet shows the implementation to calculate performance indices in Scilab.

Table 1: Estimated system parameters

power	k_{est}	$\omega_{n\_est} rad\, s^{-1}$	ξ_{est}
20	0.156	10.387	0.718
30	0.139	6.980	0.688
40	0.142	10.144	0.720
50	0.141	10.147	0.708
60	0.142	9.954	0.717
70	0.141	9.860	0.716
80	0.142	9.592	0.723
90	0.141	9.554	0.727
100	0.126	9.553	0.727

Table 2: Performance indices

power	ISE	IAE	ITSE	ITAE
20	1789	1338.16	4560.35	4964.67
30	6034	2506.23	17811.10	9551.27
40	7973	3319.66	24116.71	13278.65
50	5323	1371.26	5604.89	2395.96
60	11963	3845.87	27220.11	14338.66
70	13110	3092.94	23396.65	9659.96
80	16354	3715.09	24899.62	11672.89
90	19872	4263.08	29474.89	13720.49
100	19854	4273.36	29550.27	13779.94

```
// Error Calculation
ISE = 0
IAE = 0
ITSE = 0
ITAE = 0
for i = 1:size(t,1)
        E(i) = Omega_est(i) - Omega(i)
        ISE = ISE + (E(i)^2)
         IAE = IAE + abs(E(i))
         ITSE = ITSE + i*0.005*(E(i)^2)
         ITAE = ITAE + i*0.005*abs(E(i))
end
```

The files used for analysis are attached to the report.

2.4 Results

Table 1 contains the tabulated values of the estimated parameters of the system obtained from analysis of data obtained for power levels 20 to 100. Table 2 contains the tabulated values of performance indices for power levels 20 to 100.

2.5 Identified system model

Taking average values of identified parameters in Table 1 over the different power levels used in the analysis, we can now define the system model with $\omega_{n\_est} = 9.585 rad\, s^{-1}$, $\xi_{est} = 0.7166$ and $k_{est} = 0.141$. Substituting the identified values in Equation (1) we get the following system model given

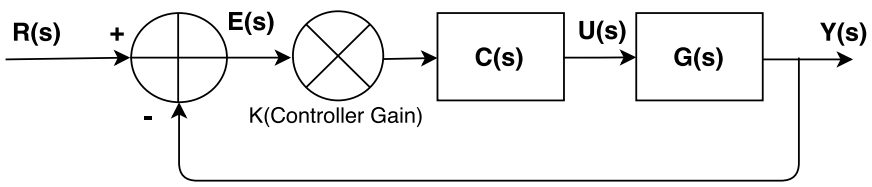


Figure 7: Closed loop system with dynamic controller

by Equation (9).

$$\Omega_1(s) = \frac{0.141}{\frac{s^2}{9.585^2} + 2s\frac{0.7166}{9.585} + 1} \tag{9}$$

3 Controller Design and Implementation

3.1 Aim

The Controller is an entity that helps in regulating the behavior of a system. It is expected that the controller acts to maintain a given settings of the system. In this assignment we are required to design a controller for the Lego DC motor. The controller is part of a closed loop system. The input to the controller is the error between a reference input (angular velocity) given in speed and the feedback from output which is the speed of the motor at a given instant. The output of the controller is power percent which is fed to the motor. The controller is responsible to maintain the behavior of the motor according to given specifications.

Figure 7 shows the block diagram of a closed loop system. Using the system model G(s), we can design a dynamic controller C(s) with K (denoted as k_c from now), the gain of the controller to get better control of the system and meet our design specifications.

3.2 Design Specifications

The response of the closed loop system with the controller for a step input is expected to follow the following design specifications.

1. Zero steady state tracking error.
2. The settling time for 5 percent of the reference speed is less than 400 ms.
3. The maximum overshoot is less than 20 percent.

3.3 Design Methodology

In this section we describe the mathematical approach we followed to come up with the controller design. Taking the design specifications into account we describe the following methods to define the Region Of Acceptance (ROA) for closed-loop system pole location.

3.3.1 Zero steady state tracking error

In Section 3.1 the tracking error is E(s). Equation (10) gives the equation for the tracking error.

$$E(s) = \frac{R(s)}{1 + k_c C(s) G(s)} \tag{10}$$

For the system to have steady state error of zero for a step input of $R(s) = \frac{1}{s}$ (in time domain r(t)=1(t)), it is evident that there needs to be a pole at origin. This can be considered as a required component of

the controller.
Substituting $C(s) = \frac{1}{s}$ in Equation (10) we get

$$E(s) = \frac{R(s)}{1 + k_c \frac{1}{s} G(s)}$$

For a step input and by application of final value theorem we get:

$$E(s) = \lim_{s \to 0} \frac{s}{s(1 + k_c \frac{1}{s} G(s))} = 0$$

We can see that a pole at the origin is necessary for E(s) to be zero.

3.3.2 Settling Time and Maximum Overshoot

The motor is expected to track and maintain the reference angular velocity given by R(s).
Equation (3) gives the overshoot of a second order system. It is required that, the $O \leq 0.2$ to meet the specifications. The angle θ of the open loop poles of the system is given by

$$\theta = \arctan \frac{\sqrt{1 - \xi^2}}{\xi}$$

Solving to obtain the value of θ from Equation (3) we obtain Equation (11) and a bound on the θ which is 62.8°.

$$\theta = \arctan \frac{\pi}{\log(O)} \leq [\arctan(1.951) = 62.8^\circ] \tag{11}$$

Solving for ξ in Equation (3), we get the value of $\xi \geq 0.456$. Equation (4) gives the equation for T_s. We are required to design a system having $T_s \leq 0.4s$. Given the value of bound of T_s we are now able to calculate $\xi\omega_n$ value for which given specification on T_s is met. It can be noted that the value $\xi\omega_n$ is the real part of the conjugate poles of the system. This means that we are finding a bound on the real part of closed loop conjugate poles. Substituting values for required T_s and ξ we are able to get a range on the value $-\xi\omega_n$ shown in the following equation.

$$-\xi\omega_n \leq [\frac{\log(\frac{\alpha}{100}) - \log(\frac{1}{\sqrt{1 - \xi^2}})}{T_s} = -7.78]$$

From the methodology followed, we are now able to draw a ROA on the s-plane, where the closed loop poles of the system should lie so that closed loop system meets the given design specifications. The ROA is

$$-\xi\omega_n \leq -7.78$$

$$\xi \geq 0.456$$

$$\theta \leq 62.8^\circ$$

The ROA can be visualized on s-plane as shown in Section 3.3.2 shaded in red. If the closed loop poles lie in the shaded ROA, then the system specifications can be met.

3.4 Proposed Controller Design

We experimented with controller designs on Root Locus plots (Evans Plots) in Scicoslab to bring the closed loop poles in ROA. To meet the design goals considering the ROA, we came up with the controller design having the following components.

1. Zero steady state error requires pole at origin $s = 0$
2. Zeros at $s = -6.5 \pm 6.5i$

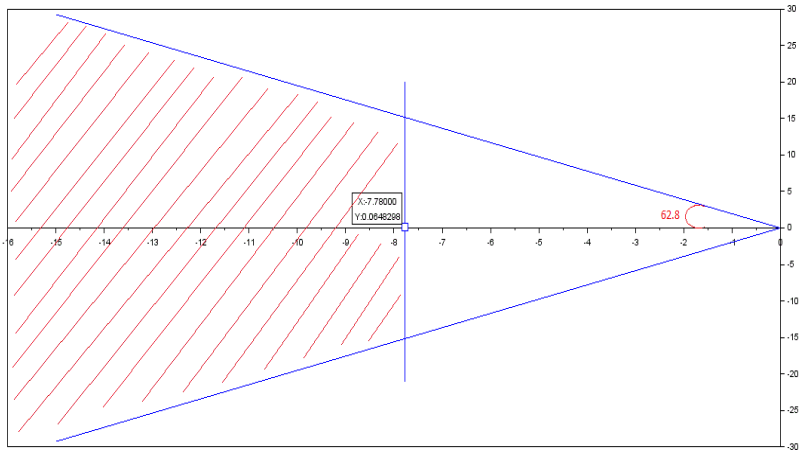


Figure 8: ROA, x-axis denoting the real axis and y axis denoting imaginary axis in s-plane

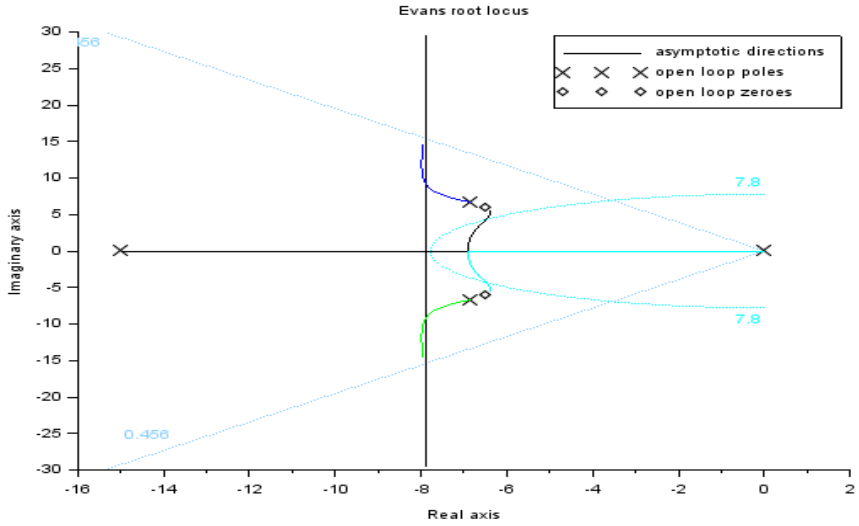


Figure 9: Root locus of the proposed controller design

3. A pole at $s = -15$
4. A suitable k_c chosen to be 20.

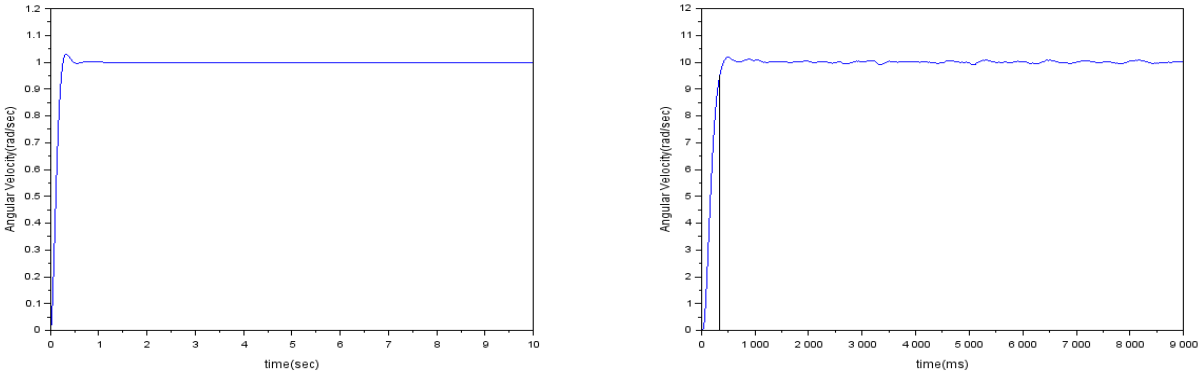
Our proposed controller design for the closed loop system to meet design specifications is

$$C^{'}(s) = 20\frac{(s + 6.5 + 6i)(s + 6.5 - 6i)}{s(s + 15)}$$

Figure 9 shows the root locus of closed loop system with the proposed controller design. It can be seen that the closed loop system poles are within the ROA. Figure 10a shows the simulated response of the closed loop system with proposed controller. It can be seen that the system satisfies the given specification requirements of $O \leq 0.2$ and $T_s \leq 0.4s$.

3.5 Controller implementation

The controller C(s) was first discretized and implemented using a simple C program on the Lego Brick. Input to controller is E(s) and output is power in P(s). Therefore P(s) = E(s)C(s)



(a) Simulated response

(b) Actual response

Figure 10: Simulated and actual response of the closed loop system with controller

3.5.1 Discretization using Trapezoidal rule

For Discretization of controller we used Trapezoidal rule which is the preferred discretization algorithm. The discretization of controller is given by

$$s \leftarrow \frac{2}{T}\frac{q-1}{q+1}$$

The resulting equation for discretization with T=0.005s (sampling interval at client, interval at which we sample the angular velocity of motor for feedback in closed loop system), E(s) and P(s) transform to e(k) and p(k) respectively. k is the discrete time used in the implementation.
After discretization the controller transforms into discrete time controller given by

$$p(k) = \frac{((413.1956e(k)) - (799.608e(k-1)) + (387.195e(k-2))) \times 20 + (800p(k-1)) - (385p(k-2))}{415}$$

3.5.2 Controller Implementation on Lego Brick and Analysis

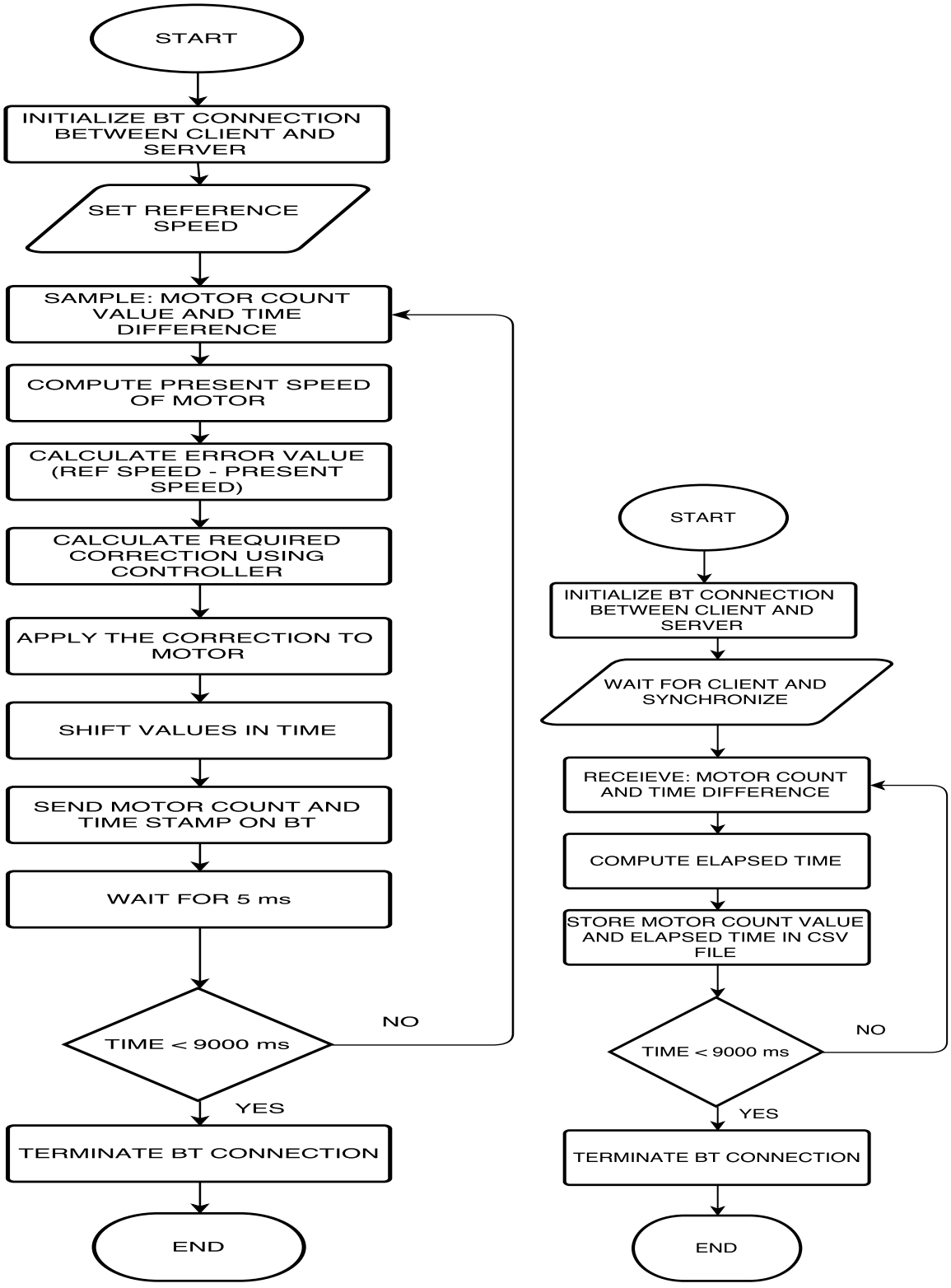
Figure 11a shows the block diagram of the implementation used in the controller on Lego Brick. The sampled motor count and time stamps values are used to compute the present motor speed. Based on the present motor speed, required corrections are taken using the discrete controller. The corrections are further applied to maintain the reference speed considering the saturation of the motor. If

$$-100 \leq \textit{correction value} \leq 100$$

then the correction values are applied to the motor. If *correction value* falls out of the range ±100, then it is considered a saturation and either -100 or 100 is applied to motor based on $correction0.1cmvalue$ is negative or positive respectively. This cycle is repeated every 5 ms. At the server side, we receive the motor count and time stamp values and compute T_s and overshoot to verify if the controller performed according to given design specifications. The procedure is similar to the one followed in parameter identification for extracting T_s and overshoot values from the data obtained by client. The approach is summarized by the block diagram shown in Figure 4.

3.6 Results

Table 3 shows the tabulated values of the overshoot and T_s for reference speeds from 3 $rad\ s^{-1}$ and 14 $rad\ s^{-1}$. In the table, final speed shows the last last recorded value in Omega_out which can be considered as the angular speed at ∞ time.
We can see that values of overshoot and T_s in Table 3 are under 20% and 400 ms respectively, confirming that the required design specifications were met by the closed loop system employing the controller.



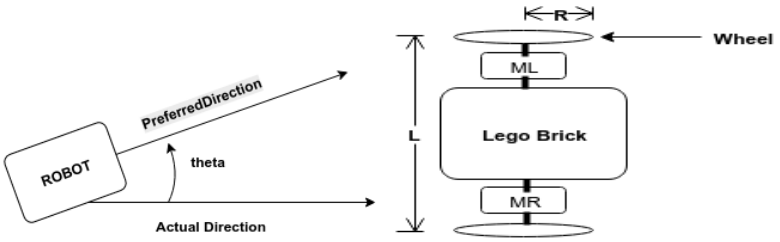
(a) Controller Implementation on Lego Brick

(b) Server Flowchart

Figure 11: System Flowchart for analysis on controller implementation

Table 3: Closed System Performance

Reference Speed ($rad\ s^{-1}$)	Final Speed ($rad\ s^{-1}$)	Overshoot(%)	T_s(ms)
3	2.95	3.67	335
4	4.01	4.99	345
5	4.91	5.90	325
6	6.07	1.60	370
7	6.96	1.77	340
8	7.94	3.69	335
9	8.97	2.28	340
10	10.00	1.89	345
11	10.98	3.15	340
12	12.02	2.21	335
13	13.03	1.18	335
14	14.13	4.63	345



(a) Block Diagram of robot (b) Specifications of the robot

Figure 12: Wheeled robot

4 Wheeled robot using controller

4.1 Aim

The task of the assignment is to design and implement a wheeled robot able to travel at a given linear velocity in a straight path and vary the linear speeds during its transit. The given parameters for design are linear velocity and the direction in phase. The phase however is 0 (at all times) since the robot is required to travel in a straight path which requires no direction changes.

4.2 Introduction

A robot is an entity which can be programmed to perform specific actions. In our assignment we are required to design and implement a wheeled robot able to travel in a straight line with a given linear velocity. This can be accomplished by help of two motors, one placed on left and one on right side of motor. The idea is to make both motors rotate at a constant angular velocity and track each other to realize the purpose. The robot is implemented using the LEGO Mindstorm kit and NXT OSEK.

4.3 Robot Design

Figure 12b shows a block diagram of the robot. The robot is made up of LEGO Brick, two LEGO motors, two wheels. The LEGO Brick controls the robot behavior. The motors act in command of the brick and perform movement by moving the wheels attached to the motor. The radius of the wheel is R and the width of the robot is given by L. The left motor is denoted by ML and the right motor denoted by MR.
Figure 12a shows that in order to change the direction to preferred direction we need to change by an angle theta (θ). The value of θ sets the direction of the robot at a given moment. In our case $\theta = 0$ since robot is required to travel in a straight path. For the robot shown in Figure 12b we can

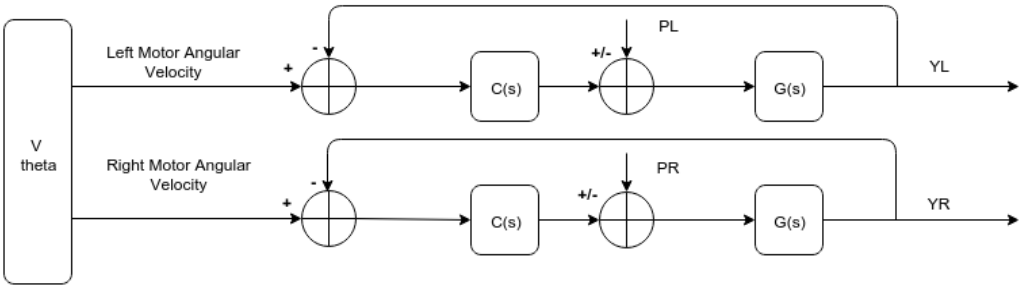


Figure 13: Dual control design

analytically formalize the required angular velocities to be given to the motor for a given linear velocity and direction of travel.
Given the angular velocity of left motor is ω_l and angular velocity of right motor is ω_r during the robot operation, we can derive the current linear velocity (V) and direction of travel (θ) of the motor with respect to current position.
Formula for linear velocity is given by

$$V = (\omega_l + \omega_r)\frac{R}{2}$$

and equation for θ is given by

$$\theta = (\omega_r - \omega_l)\frac{R}{L}$$

Give velocity V and direction of travel is along a straight path, we have $\theta = 0$. We can obtain the values ω_l and ω_r by solving the above equations as

$$\omega_l = \omega_r = \frac{V}{R}$$

Figure 13 shows the block diagram of the dual control design of our robot. In the figure YL and YR are the output velocities of the left and right motor respectively. C(s) and G(s) are the controller designed for the LEGO motor as described in Section 2 and G(s) is the second order system function of the LEGO motor obtained in Section 3 of the course respectively. The required angular velocities of the motor are calculated and fed to the system. The top flow belongs to the left motor and bottom flow belongs to the right motor. With ability to correctly track and maintain a given angular velocity using the controller, we are able to apply same speed on both motors and achieve the required linear velocity. PL and PR are final correction and tuning on the power before being fed to the motor.

4.4 Robot Implementation

Figure 14 shows the implementation flowchart of dual control on the Lego Brick. The robot works autonomously to move in a given speed and in straight path. We have programmed the robot to stop and take up several speeds on its journey to show its performance. Figure 15 shows the actual robot we built using the Lego Mindstorms kit.

5 Conculsion

It was possible from the parameter identification and controller design phases to come up with a controller for the given DC motor satisfying design specifications. We were integrate to integrate the controller for both the motors and achieve the required objective of traveling on a straight path and with a given linear velocity.

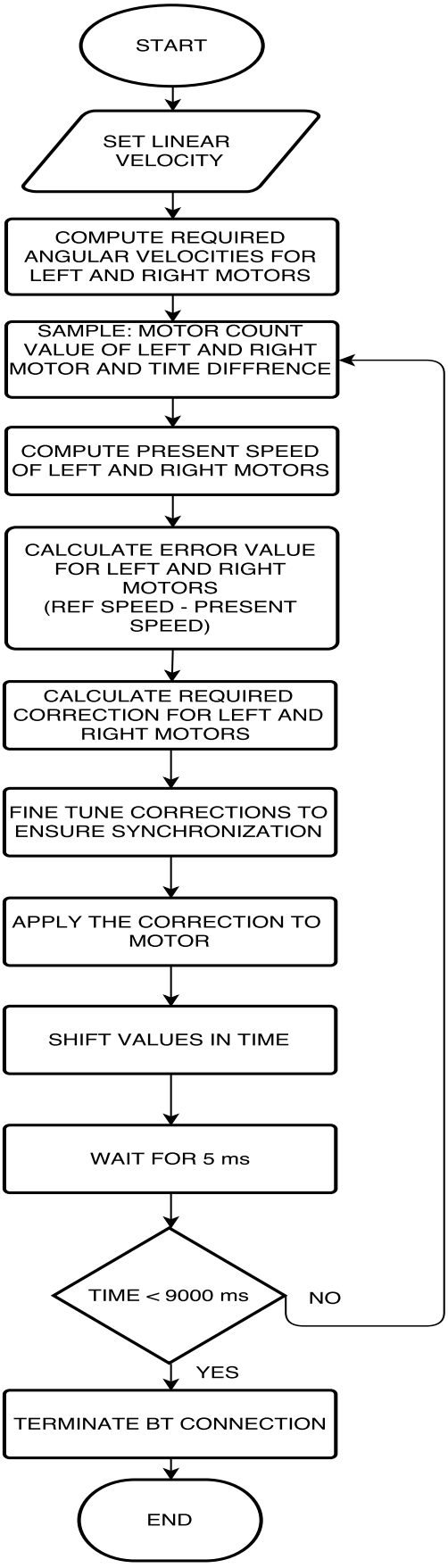


Figure 14: Flowchart of dual control implementation on Lego Brick

Figure 15: Actual wheeled robot built using the Lego Mindstorms kit