ASSIGNMENT REPORT ON

"Autonomous Balancing of 2-wheeled Segway Robot"

BY

GROUP NO. - 14

Jash Shah	2018A8PS0507P
Akshit Patel	2018A8PS0094P
Rishabh Jain	2018A8PS0430P

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

SUBMITTED TO

Dr Puneet Mishra

Assistant Professor, Department of Electrical and Electronics Engineering

SUBMITTED ON

April 13, 2021

1. Motivation and Abstract

The Segway robot is an electric, two-wheeled self-balancing human transporter with a computer-controlled gyroscopic stabilization and control system. This new vehicle is a complex and hybrid machine that requires engineering competence in many fields; vehicle dynamics, automatic control, battery technology, power electronics, software engineering, microcomputer programming, network and communication engineering.

The project is the implementation of the below-mentioned paper. Further, we also delve into the dynamics of the controller and also the mechanical aspects of model development. We also extend the discussion by implementing various other optimization algorithms and comparing the performance with the one given in the paper. The abstract for the paper is as follows :

”””
A two-wheeled single-seat Segway robot is a special kind of wheeled mobile robot, using it as a human transporter system needs applying a robust control system to overcome its inherent unstable problem. The mathematical model of the system dynamics is derived and then state-space formulation for the system is presented to enable the design of a state feedback controller scheme. In this research, an optimal control system based on the linear quadratic regulator (LQR) technique is proposed to stabilize the mobile robot.

The LQR controller is designed to control the position and yaw rotation of the two-wheeled vehicle. The proposed balancing robot system is validated by simulating the LQR using Matlab software. Two tuning methods, genetic algorithm (GA) and bacteria foraging optimization algorithm (BFOA) are used to obtain optimal values for controller parameters. A comparison between the performance of both controllers GA-LQR and BFO-LQR is achieved based on the standard control criteria, including rising time, maximum overshoot, settling time, and control input of the system. Simulation results suggest that the BFOA-LQR controller can be adopted to balance the Segway robot with minimal overshoot and oscillation frequency.
“”

2 Contribution and Acknowledgement

All the 3 members bearing ID 2018A8PS0507P, 2018A8PS0430P and 2018A8PS0094P acknowledge to have contributed to this project with **fair and equal** contribution and the same has been mentioned below as well :

Akshit Patel	33.333 %
Rishabh Jain	33.333 %
Jash Shah	33.333 %

Table of Contents

1. Motivation and Abstract
2. Contribution and Acknowledgement
3. Problem definition
   - 3.1. Segway's servo and regulatory models
   - 3.2. Soft Computing and Optimization
4. Detailed Solutions
   - 4.1. Development of Segway Model
     - 4.1.1. Electrical Subsystem
     - 4.1.2. Chassis Subsystem
     - 4.1.3. Mechanical Subsystem
   - 4.2. Methodology
     - 4.2.1. Controllability of Servo and Regulatory Action
     - 4.2.2. LQR Controller
     - 4.2.3. Optimization Algorithm
5. Comparison and Simulations
   - 5.1. Disclaimer and changes
   - 5.2. GA_LQR
   - 5.3. BFO - LQR
6. Improvements, Scope and Future Works
7. Results and Conclusions
   - 7.1. Observation and Discussion
   - 7.2. Conclusion
8. References

Appendix A: MATLAB code
Appendix B: Simulink Models

Other Related Documents

-> Project Presentation view link -

https://docs.google.com/presentation/d/1yOReIAvVy_GWijMy9ZSYenYkto2RVBrg3X-Kbd3mIC8/edit?usp=sharing

3. Problem Definition

Please note that the models developed in this section are intuitive and idealistic models and that specific details have been shared in the subsequent sections.

3.1. Segway's Servo and Regulatory Models

The Segway is an extraordinary robot with very little input constraint and a high degree of freedom. This makes it very convenient and fast for use even for even law enforcement. A good amount of control is required to balance the vehicle as it is a combination of a **Unicycle and an Inverted Pendulum. It now becomes very intuitive that the inverted pendulum needs to be balanced.**

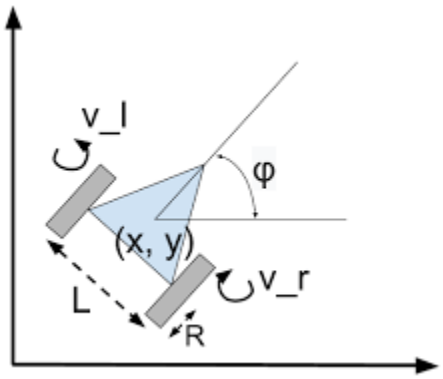


Fig. 1: segway robot

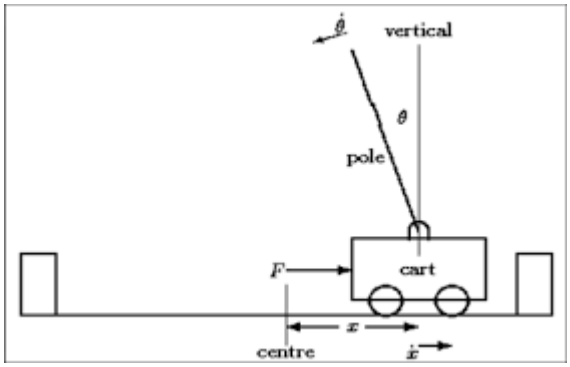


Fig. 2: inverter pendulum attached to a cart

The Uicycle's generalized model includes the state equations of velocity and angular velocity, and the inputs to the system would be forced or the torques applied on the 2 wheels. The Control of the Inverted Pendulum involves the tilt angle and the cart's velocity. The variables as used in the picture above.

All the state matrix is as follows: $X = [x\ y\ v\ v^*\ \phi\ \phi^*\ \theta\ \theta^*]$

But, once we try to find the controllability of the model, we find that the **rank is 6 !!! i.e.** we need to eliminate some of the states. The model does not have enough degrees of freedom.

Now, a very obvious elimination is the elimination of x and y coordinates because the unicycle can not move sideways. Also, the inclusion of **V and φ covers the entire workspace**.

The Servo action requires the model to be linearized and the best way to linearize a system is to linearize it about the origin. This would create a problem specifically when we want to move the device. Although it is advised to use direct variables in a multistep process, over here, we use deviational variables to tackle the dynamic linearization of the system.

$$\mathbf{X\_new = X - \delta = [\ V\ \ \phi\ \ \theta\ \ \theta^*\ ] - [\ } V_{req}\ \ \phi_{req}\ \ 0\ \ 0\ ]$$

,where, V_{req} and ϕ_{req} are the desired velocity and orientation at any time t .

3.2. **Soft Computing and Optimization**

Soft computing is a technique that is used to solve problems that are very hard to analyse or whose equations are very complex. Soft computing includes neural networks, fuzzy logic and some algorithms inspired by the behaviours of various species in nature. Soft computing can be used to find very efficient solutions that are cost-effective in a short amount of time.

3.2.1 Genetic algorithm

Genetic algorithms or GA is a type of soft computing optimization technique. This technique is based on the process of natural selection. We start by taking a random pool of population which is composed of various chromosomes or candidate solutions. These solutions are basically binary strings like human DNA.

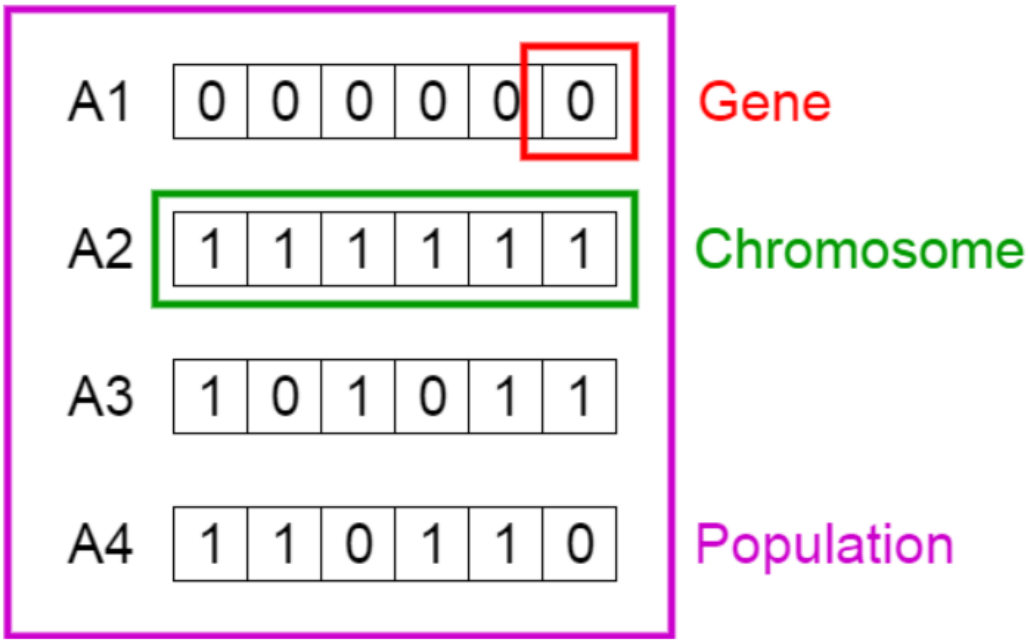


Fig. 3: Random generated population

For every generation we find the fitness of all the possible solutions in our pool and then we remove inefficient solutions by using a natural selection method. A crossover point is chosen at random from within the genes. Reproduction is performed by exchanging parental genes between them until they reach a crossover point.

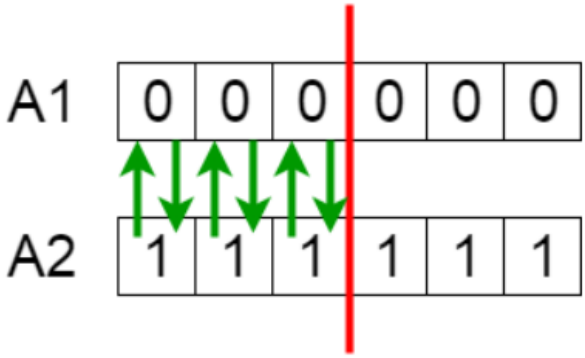


Fig. 4: crossover between two chromosome

We then choose a random point and flip the bit and this is termed as mutation.
The solutions left are then broken, recombined and mutated giving our next generation. This procedure is repeated several times until our population saturates and the best solution is then extracted and used for our problem.

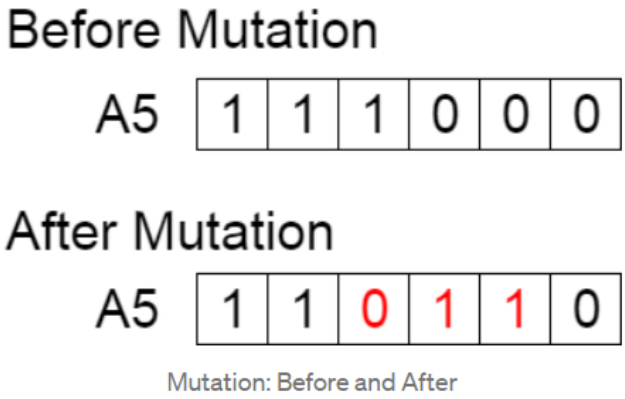


Fig. 5: mutation on a chromosome

The algorithm ends when the population has changed (it does not produce very different fitness from previous generations). This final population is considered as the set of solutions to our problem and the best solution can be extracted for best performance.

The pseudo code is given below:

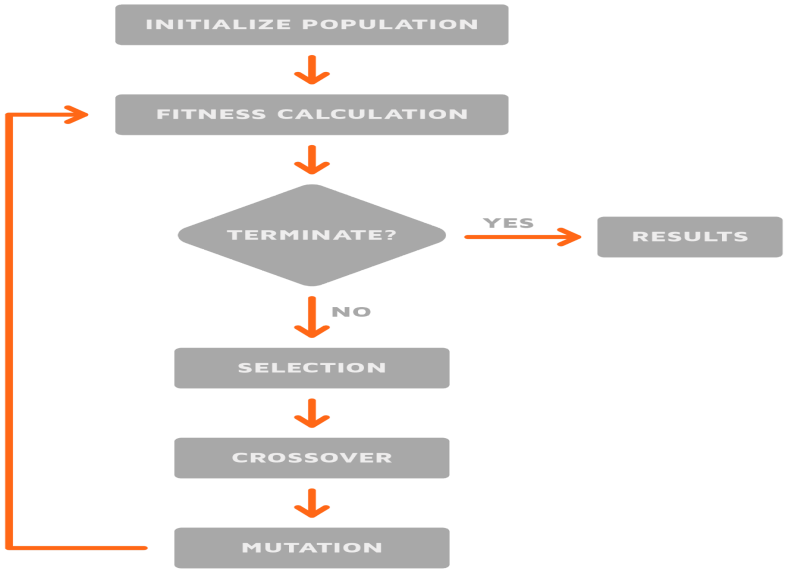


Fig. 6: GA logic diagram

3.2.2 Bacteria Foraging Optimization

Natural selection methods eliminate bacterias with poor "foraging strategies".Forging strategies are methods for locating, handling, and ingesting food. This method favours those bacterias that have successful foraging strategies or the bacterias with enough food s.t they can reproduce.After many generations, poor foraging strategies are either eliminated or shaped into good ones i.e. redesigned .Such evolutionary principles have led scientists in the field of "foraging theory" to hypothesize .It is appropriate to model the activity of foraging as an optimization

A foraging animal's main focus is to maximize the energy obtained per unit time spent foraging .In the face of constraints presented by its own physiology . e.g. includes, sensing and cognitive capabilities, environment ,etc. Evolution has balanced these constraints .Sometimes referred to as optimal foraging policy. Such terminology is especially justified in cases where the models and policies have been ecologically validated.

Optimal foraging theory formulates the foraging problemas an optimization problem. Optimization models are valid for social foraging where groups of animals cooperatively forage .Here, we explain the biology and physics underlying the chemotactic (foraging) behavior of E.coli bacteria for optimization named as Bacterial Foraging Optimization. Individual bacterium also communicates with others by sending signals .During foraging of a real bacteria two basic operations swim and tumble is performed by a bacterium at the time of foraging by a set of tensile flagella.

Bacterial foraging algorithm is inspired by an activity called “chemotaxis” exhibited by bacterial foraging behaviors. Motile bacteria such as E. coli and salmonella propel themselves by rotation of the flagella. To move forward, the flagella rotates counterclockwise and the organism “swims” or “runs” while a clockwise rotation of the flagellum causes the bacterium to randomly “tumble” itself in a new direction and swim again 19. Alternation between “swim” and “tumble” enables the bacterium to search for nutrients in random directions. Swimming is more frequent as the bacterium approaches a nutrient gradient. Tumbling, hence direction changes, is more frequent as the bacterium moves away from some food to search for more. Basically, bacterial chemotaxis is a complex combination of swimming and tumbling that keeps bacteria in places of higher concentrations of nutrients.

BFO foraging strategy

`

- Chemotaxis: This process is related to movement of bacterium during search for food. The E-coli bacteria can move in two ways namely, swimming and tumbling, and they are able to alternate between these two movement styles for the whole of their lifetime. In the swimming mode, the bacteria walk in a certain direction for gathering food, while in the tumbling mode, they move with random directions.
- Swarming: The swarming action means the bacteria with a good fitness value, try to attract others to form groups, so that they all can arrive at the desired location. These groups of E-coli cells arrange themselves, in which they can move as concentric patterns for food searching.
- Reproduction: In this stage, all the bacteria population are classified based on health status. The healthier bacteria, which have had sufficient nutrients will be reproduced, while the less healthy bacteria will die. The surviving bacteria will split into an identical replica of itself placed in the same locations with a that number equals to the number of the dead ones.
- Elimination and Dispersal: During this evolutionary step, gradual or sudden events or attacks in the local living environment of the bacteria, may occur due to significant rising of the temperature caused by occupancy of a high density of bacteria for a specific area. This high temperature may kill a group of bacteria and dispersal of others into some new locations.

4. Detailed Solution

4.1.Development of Segway model

4.1.1 Electrical subsystems

The electrical subsystems contain a DC motor which is used for accelerating and thus changing the position of our robot. When we apply Voltage Va (V) current I (t) (A) is generated in the armature of our motor and this generates a torque(Nm). The following equations are generated.

$$C = K_m i \tag{1}$$

In equation 1 the Km is constant (Nm/A). Due to the rotation of the coil in the DC motor, a back electromagnetic force(emf) voltage Ve (V) is generated. This function can be approximated as a linear function of motor angular velocity (rad/s) as follows.

$$V_e = K_e \dot{\theta} \tag{2}$$

Here Ke is torque. Using Kirchoff's voltage law on our motor circuit as shown in figure x we get the following equation.

$$V_a = Ri + L\frac{di}{dt} + V_e \tag{3}$$

As we know the dynamics of a mechanical system is slower as compared to the electrical system so the transient of the system can be omitted. By solving (3) we get this equation.

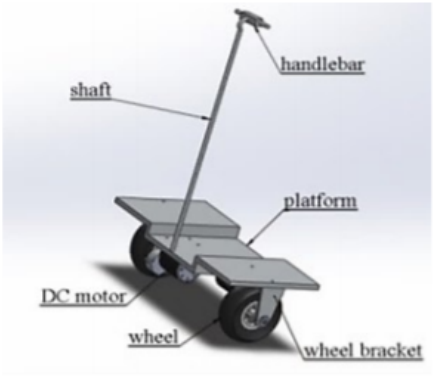
$$\mathbf{i} = \frac{\mathbf{V_a - V_e}}{\mathbf{R}} \tag{4}$$

based on (2), (4) can be written as follows:

$$i = \frac{V_a}{R} - \frac{K_e}{R}\dot{\theta} \tag{5}$$

After using (5) and (1) we get the following equation for torque

$$C = \frac{K_m}{R}V_a - \frac{K_m K_e}{R}\dot{\theta} \tag{6}$$



. Fig. 7: Model of a two-wheeled robot

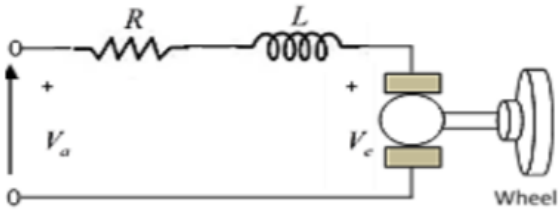


Fig. 8: . Electric model circuit of DC motor

4.1.2 Mechanical system modelling

The mechanical system consists of a left and a right wheel attached to our chassis which behave like an inverted pendulum. For simplicity purposes the left and the right wheel are assumed to be the same i.e they have the same mass, radius and same moment of inertia. We will derive equations using the right wheel. We will derive the equations of motion and the dynamics of our system by considering the wheels and the inverted pendulum separated.

The free-body diagram for the left and right wheel is shown in figure 3. The sum of the external forces, which governs its translation motion in the horizontal x-direction which is derived using Newton laws of motion is given by.

$$\sum F_x = M_w a \tag{7}$$

$$M_w \ddot{x} = H_f - H \tag{8}$$

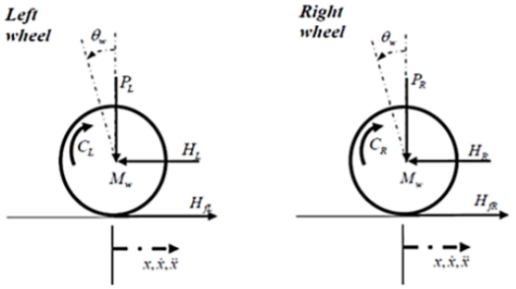


Fig. 9: Free body diagram of the robot wheels

the wheel mass of the robot is *Mw*(*Kg*), a is the gravity acceleration (*m*/*s* 2), the and*Hf*is the friction force between ground and wheels(*N*) and the rotational motion *Mo* of the wheel is given by go

$$\sum M_o = I_w \ddot{\theta}_w \tag{9}$$

$$I_w \ddot{\theta}_w = C - H_f r \tag{10}$$

Here the moment of inertia and acceleration of the wheels and radius of the wheel(m) is given by where *Iw* (*Kgm*2), *θw* (*m*/*s* 2) and r respectively. Using equation (6) we get

$$I_w \ddot{\theta}_w = \frac{K_m}{R} V_a - \frac{K_m K_e}{R} \dot{\theta}_w - H_f r \tag{11}$$

$$H_f = \frac{K_m}{Rr} V_a - \frac{K_m K_e}{Rr} \dot{\theta}_w - \frac{I_w}{r} \ddot{\theta} \tag{12}$$

where *θw*is the angular velocity of the wheel (*m*/*s*). Substituting (12) into (8) yields (13) and (14) for the left and right wheels respectively.

Here angular velocity is given as *θw*. Using (12),(8),(13) and (14) for both the wheels we get

$$M_w \ddot{x} = \frac{K_m}{Rr} V_a - \frac{K_m K_e}{Rr} \dot{\theta}_w - \frac{I_w}{r} \ddot{\theta}_w - H_L \tag{13}$$

$$M_w \ddot{x} = \frac{K_m}{Rr} V_a - \frac{K_m K_e}{Rr} \dot{\theta}_w - \frac{I_w}{r} \ddot{\theta}_w - H_R \tag{14}$$

As we are not considering the y-axis i.e turning of our chassis the centre of the robot wheel is following a linear motion, Using the simple transformation $\theta wr = \ddot{x} \rightarrow \theta = \ddot{x}/r$, $\theta wr = \dot{x} \rightarrow \theta = \dot{x}/r$ the wheel can be transformed into these simple linear equation. By the linear transformation, (13) and (14) become as follows:

$$M_w \ddot{x} = \frac{K_m}{Rr} V_a - \frac{K_m K_e}{Rr^2} \dot{x}_w - \frac{I_w}{r^2} \ddot{x} - H_L \tag{15}$$

$$M_w \ddot{x} = \frac{K_m}{Rr} V_a - \frac{K_m K_e}{Rr^2} \dot{x}_w - \frac{I_w}{r^2} \ddot{x} - H_R \tag{16}$$

Using (15) and (16) we get

$$2(M_w + \frac{I_w}{r^2})\ddot{x} = 2\frac{K_m}{Rr} V_a - 2\frac{K_m K_e}{Rr^2} \dot{x}_w - (H_L + H_R) \tag{17}$$

4.1.3 Chassis Model

We can assume the chassis to act like an inverted pendulum. The free-body diagram (FBD) of the chassis acting as an inverted pendulum is shown in figure 4. Using Newton's law of motion, the sum of forces acting on the chassis in the horizontal x-direction is given by [20]

$$\sum F_x = M_p \ddot{x} \tag{18}$$

$$M_p \ddot{x} = H_L + H_R - M_p l \ddot{\theta}_p \cos\theta_p + M_p l \dot{\theta}^2{}_p \sin\theta_p \tag{19}$$

The mass of our robot (Kg) is Mp and θp is the rotational angle of the chassis (*rad*). After rearranging the above equation we get:

$$H_L + H_R = M_p \ddot{x} + M_p l \ddot{\theta}_p \cos\theta_p - M_p l \dot{\theta}^2{}_p \sin\theta_p \tag{20}$$

The sum of perpendicular forces acting on the pendulum is:

$$\sum F_p = M_p \ddot{x} \cos\theta_p \tag{21}$$

$$(H_L + H_R)\cos\theta_p + (P_L + P_R)\sin\theta_p - M_p g \sin\theta_p - M_p l \ddot{\theta}_{\ddot{p}} = M_p \ddot{x} \cos\theta_p \tag{22}$$

where *PL*and *PR* are the reaction forces between left and right wheel and chassis (*N*) respectively and $\dot{\theta} p$ is chassis angular acceleration(*m*/*s* 2), the sum of moments around the centre of pendulum mass is given by:

$$\sum M_o = I_p \ddot{\theta}_p \tag{23}$$

$$I_p \ddot{\theta}_p = -(H_L + H_R) l \cos\theta_p - (P_L + P_R) l \sin\theta_p - (C_L + C_R) \tag{24}$$

$$C_L + C_R = 2\frac{K_m}{R} V_a - 2\frac{K_m K_e}{R}\frac{\dot{x}}{r} \tag{25}$$

Using (24) and (25) gives

$$I_p \ddot{\theta}_p - 2\frac{K_m K_e}{R}\frac{\dot{x}}{r} + 2\frac{K_m}{R} V_a = -(H_L + H_R) l \cos\theta_p \, (P_L + P_R) l \sin\theta_p \tag{26}$$

Using (22) and (26) gives

$$I_p\ddot{\theta}_p - 2\frac{K_mK_e}{R}\frac{\dot{x}}{r} + 2\frac{K_m}{R}V_a + M_pgl\sin\theta_p + M_pl^2\ddot{\theta}_{\ddot{p}} = -M_pl\ddot{x}\cos\theta_p \quad (27)$$

The Sum of moments around the centre of pendulum mass is given as:

$$-M_pl\ddot{x}\cos\theta_p = (I_p + M_pl^2)\ddot{\theta}_p - 2\frac{K_mK_e}{Rr^2}\dot{x} + 2\frac{K_m}{Rr}V_a + M_pgl\sin\theta_p \quad (28)$$

For eliminating (*HL* +*HR*) we use (17) and (20) which gives us:

$$2\frac{K_m}{Rr}V_a = (2M_w + \frac{I_w}{r^2} + M_p)\ddot{x} + 2\frac{K_mK_e}{Rr^2}\dot{x} + M_pl\ddot{\theta}_p\cos\theta_p - M_pl\dot{\theta}^2_{\ddot{p}}\sin\theta_p \quad (29)$$

For purpose of design a linear control system, the system dynamic (28) and (29) are linearized about an operating point based on the assumption $\theta w = \pi + \phi$ where denotes an angle measured from the vertically upward direction. Therefore, $sin\ \theta p = -\phi$, $cos\ \theta p = -1$ and $(d\theta p/dt)\ 2 = 0$ for purpose of state-space representation, the dynamic (28) and (29) are rewritten as follows:

$$\ddot{\phi} = \frac{M_pI}{(I_p+M_pI^2)}\ddot{x} + \frac{2K_mK_e}{Rr(I_p+M_pI^2)}\dot{x} + \frac{M_pgI}{Rr(I_p+M_pI^2)}\phi - \frac{2K_m}{(I_p+M_pI^2)}V_a \quad (30)$$

$$\ddot{x} = \frac{2K_m}{RrK_w}V_a - \frac{2K_mK_e}{Rr^2K_w}\dot{x} + \frac{M_pI}{Rr^2K_w}\ddot{\phi} \quad (31)$$

where $Kw = 2Mw + 2Iw/r\ 2$. After a series of algebraic manipulation, the state equation becomes:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\phi}\\ \ddot{\phi}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{K_1(M_pIr - I_p - M_pI^2)}{Rr^2\alpha} & \frac{M^2pgI^2}{\alpha} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{K_1(r\beta - M_pI)}{Rr^2\alpha} & \frac{M_pgI\beta}{\alpha} & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \phi\\ \dot{\phi}\end{bmatrix} + \begin{bmatrix}0\\ \frac{2K_m(I_p+M_pI^2-M_pIr)}{Rr\alpha}\\ 0\\ \frac{2K_m(M_pI-\beta)}{Rr\alpha}\end{bmatrix}V_a \quad (32)$$

where, $K1 = 2KmKe$, $\beta = 2Mw + 2Iw\ r\ 2 + Mp$ and $\alpha = [Ip\beta + 2MpI\ 2\ (Mw + Iw\ r\ 2\ )]$. It is worth considering that the system modelling is based on the supposition that both wheels of the walking robot are assumed in a state of contact with the ground and without sliding. Cornering forces produced by vehicle wheels during cornering are also considered negligible [19].

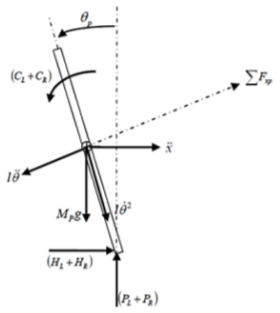


Fig. 10: Free body diagram of the chassis

4.2 Methodology

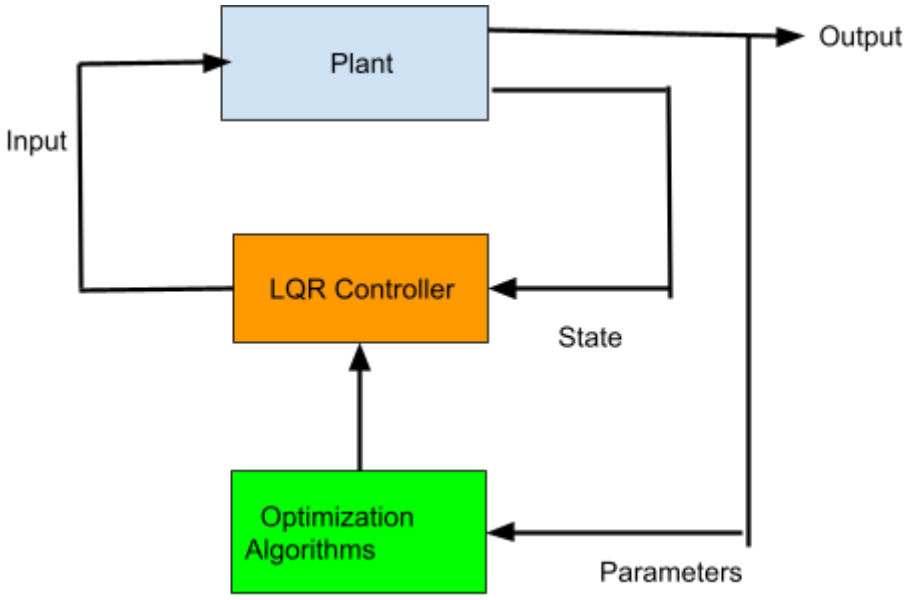


Fig. 11: model diagram

4.2.1 Controllability of Servo and Regulatory Action

Controllability is an important property of a control system, and the controllability property plays a crucial role in many control problems, such as stabilization of unstable systems by feedback, or optimal control. Controllability and observability are dual aspects of the same problem.

Out of the many possible states, we choose only 4states that make the system controllable and observable. Also, the choice of these states was based on the fact that linearization of the equations was possible.

After defining the state space models, we used the MATLAB command ctrb(A,B) to find the rank of the controllability matrix.

```
Co =

         0    0.0575   -0.0018   -0.2725
    0.0575   -0.0018   -0.2725    0.0172
         0   -0.7141    0.0226  -35.3866
   -0.7141    0.0226  -35.3866    1.0111
```

Finally, unco = length(A) - rank(Co), gave the following result :

```
>> unco

unco =

     0
```

Fig. 12: Controllablity matrix

4.2.2 LQR Controller

LQR is a state feedback controller that uses the model based cost function to optimize the control action. It can be thought of as an improved pole placement controller.

Our goal is to make the state controller depend only on X and not on U i.e. U = -KX. This would ensure that poles are never placed on the RHS of the pole-zero axis. Here, $K = [K1\ K2\ K3\ K4]$ is an optimal feedback gain matrix of the controller used to track the input command while the following performance index:

$$J = \int_0^\infty (X^T(t)Q(t)X(t) + u^T(t)R(t)u(t))dt$$

This cost function is developed by taking the controller effort and the input cost into account. This is integrated to the entire time domain to incorporate the total Integral Absolute cost.

From the above equation, we can derive the value of K can be found using the Riccati equation. The feedback gain matrix K can be determined by using the following:

$$K = R^{-1}B\ TP$$

where P denotes the solution of the following Riccati:

$$A\ TP + PA - PBR - 1B\ TP + Q = 0 \qquad (\text{Keeping N} = 0)$$

After getting the Q and R matrix, we used the MATLAB command lqr(Q,R) to get the value of K. Also, we used the deviation model to move the segway to different positions.

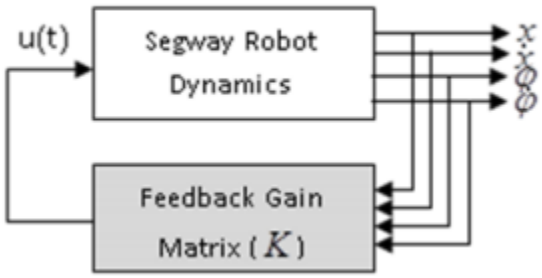


Fig. 13: LQR model

4.2.3 Optimization Algorithms

2 biological inspired algorithms have been experimented with and to obtain the optimized Q and R matrix for LQR.

- Fitness Function

Fitness function in the paper has not been defined properly, first, we took fitness function to be variation from desired values which is mentioned in the paper but as rising time and settling time were not varying much with each population, only small changes in overshoot we directly added all three values which seemed to provide better results. So our fitness function was the addition of rising time, settling time, and overshoot of position vs time graph in each iteration.

The proposed controller design is validated using Matlab programming. Based on step input, the control system is designed for the following requirements: rise time less than 10 (ms), settling time less than 30 (ms), maximum overshoot percentage less than 5%.

Fitness Function = |Rise Time| + |Settling Time| + |Overshoot |

- Genetic Algorithm
  In GA first it selects a random population of q11,q22,q33,q44 and R. In our case a population of 25 is generated. For each population, K is generated and position vs time is plotted. After that cost value is found. Members having the least value of cost function are mutated and crossover to generate another set of population. At the end of the last generation, the best member of the generation is computed.

Fig. 14: Genetic model applied to LQR parameters

- BFO

We used a random population of 5 bacteria swarms, each with 5 bacteria (for q11, q22, q33, q44, r). After simulating the model, and tuning the other parameters of BFA using Brute force, we achieved the final population that optimizes the model to a steady state error state. The model had reached a global minima due to the small population sizes.
We also incorporated constraint bounds so that the bacteria values don't go beyond a certain limit. These were taken into account by using the results of genetic algorithm

5. Comparison and Simulation

This section discusses all the results obtained from the simulation done by us. 6.1 presents some key points and notes due to which there were major changes in the obtained results and one mentioned in paper. 6.2 presents results with simple LQR with random values of Q and R. 6.3 and 6.4 shows results for Q and R obtained with genetic algorithm optimization and BFO optimization.

5.1 Disclaimer

a.) Value of Q,R,K mentioned in the paper for GA optimization

[q11 q22 q33 q44 R] = [8.969 0.308 0.121 0.0085 2.123*10−5]

K = [-6.4988 -3.8592 -5.1982 -0.7348]

But according to the values of Q and R following K should be obtained which provides correct results

K = 1.0e+02 * [-6.499755421319758 -3.859528575379667 -5.198431774768298 -0.734822500368363]

b.) Value of Q and R mentioned in paper for BFO algorithm is following

[q11 q22 q33 q44 R] = [289.104 -5.15e-5 -5e-5 -5.1e-5 0.00028]

K = [-1022 -260.2 886.72 77.7]

But according to the values mentioned of Q and R value of K should be,

K = 1.0e03 *[-1.016127101161195 -0.462096381004273 -0.527358592608302 -0.072462682693461]

In this magnitude is same but signs are different

And according to the values of K found by us results come this which are wrong according to us. Hence BFO is not providing providing better results according to the values mentioned in the paper

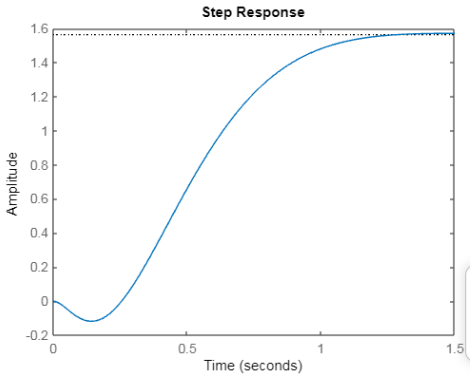


Fig. 15: distance vs time (paper)

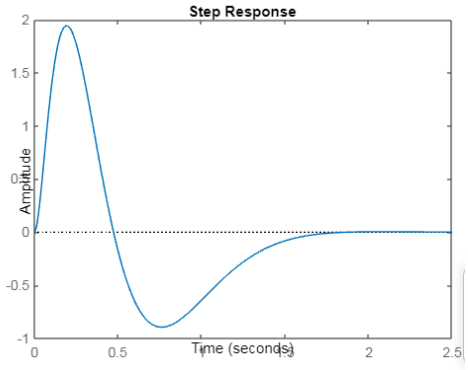


Fig. 15: angle vs time (paper)

c.) Due to limited computational power, we have taken only 10 generations instead of 100. So the total number of iterations in our case is 150.

5.2 GA LQR Results

5.2.1 Parameters

Population Size = 15
Generation = 10

Best Q Obtained =
q11=7.056311630519287
q22=0.040608399663108
q33=0.150562417774016
q44=0.030452791998497
Best R obtained = 0.000016648572929

K = 1.0e+02 *[-6.510290527654438 -3.928285136135124 -5.775516991619111 -0.891476241249279]

Fitness Function value for most optimised one = 2.170198213995964

History = Appendix C

5.2.2 Graphs

- Voltage Input vs time graph made in Simulink

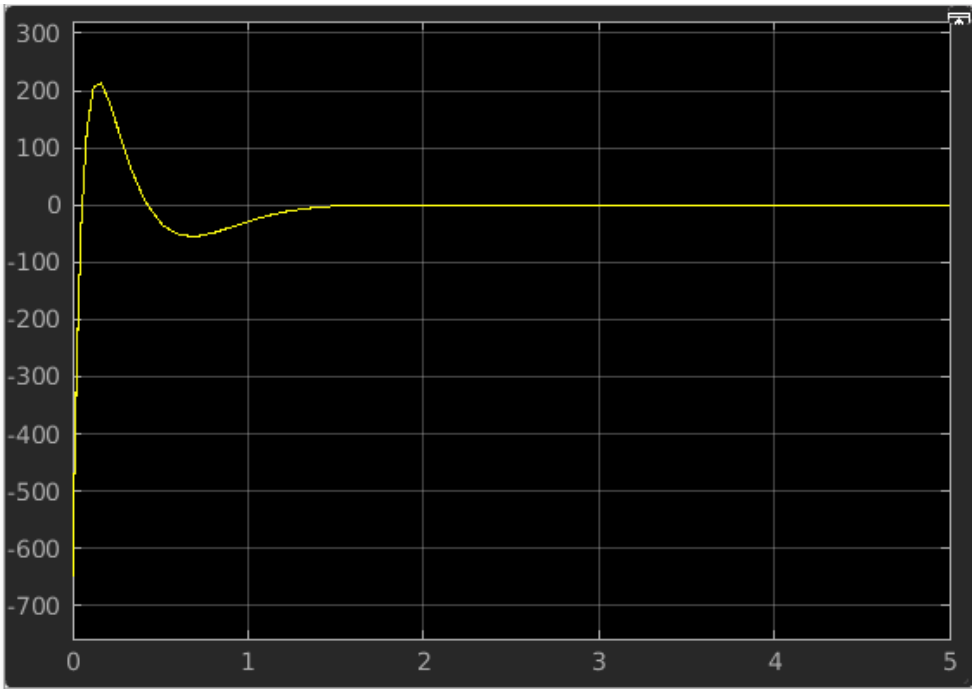


Fig. 17: Voltage vs time graph

- Position vs Time using transfer function model in MATLAB

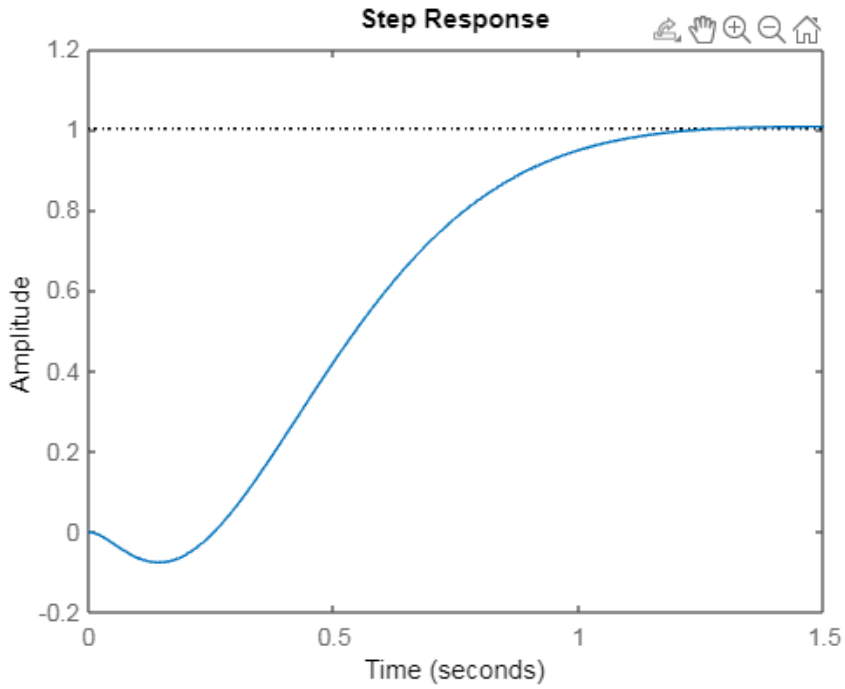


Fig. 18: Position vs time graph

- Position vs Time using State Space model in simulink

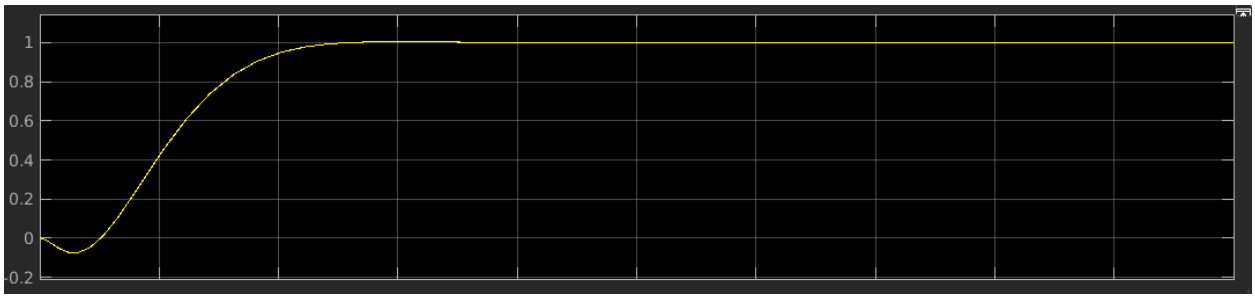


Fig. 19: Position vs time using state space model

- Angle vs Time using State Space model in simulink

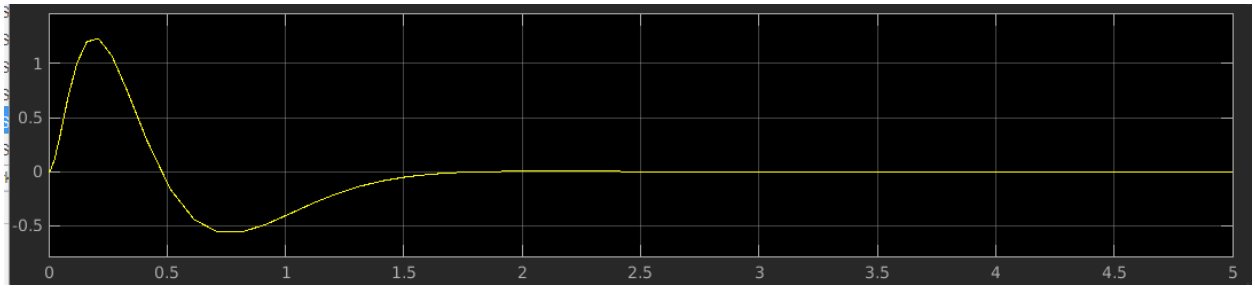


Fig. 20: Angle vs time using state space model

- Angle vs Time using Transfer Function model in MATLAB

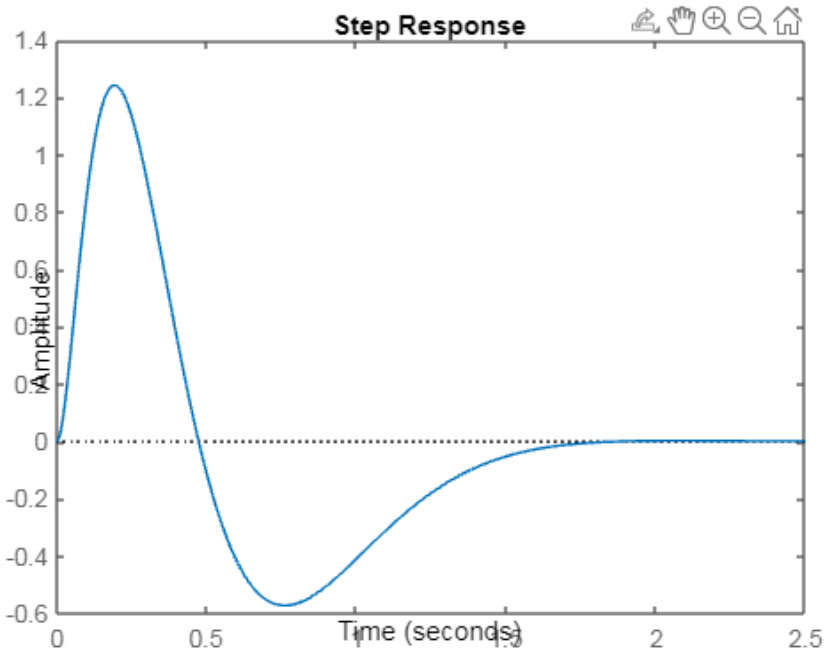


Fig. 21: Angle vs Time using transfer function model

- Graph for getting q11 vs iteration using Genetic algorithm

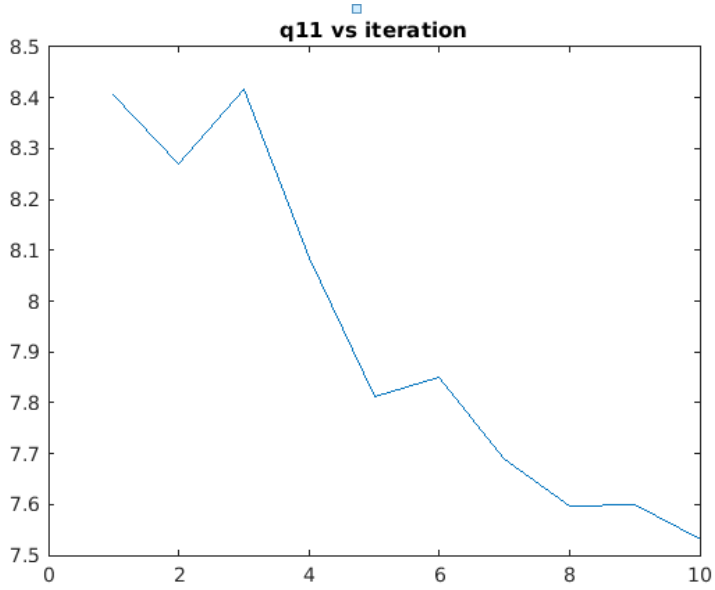


Fig. 22: q11 vs iteration graph

- Graph for getting q22 vs iteration using Genetic algorithm

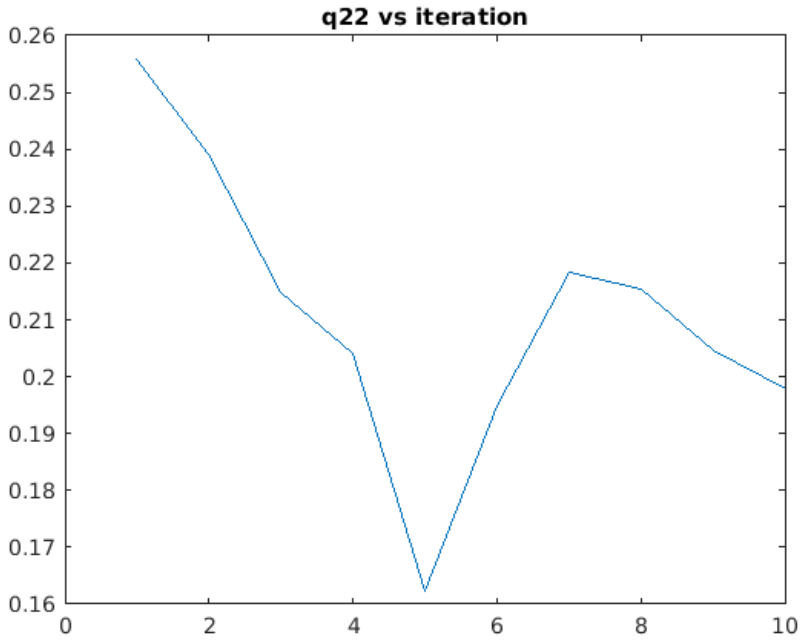


Fig. 23: q22 vs iteration graph

- Graph for getting q33 vs iteration using Genetic algorithm

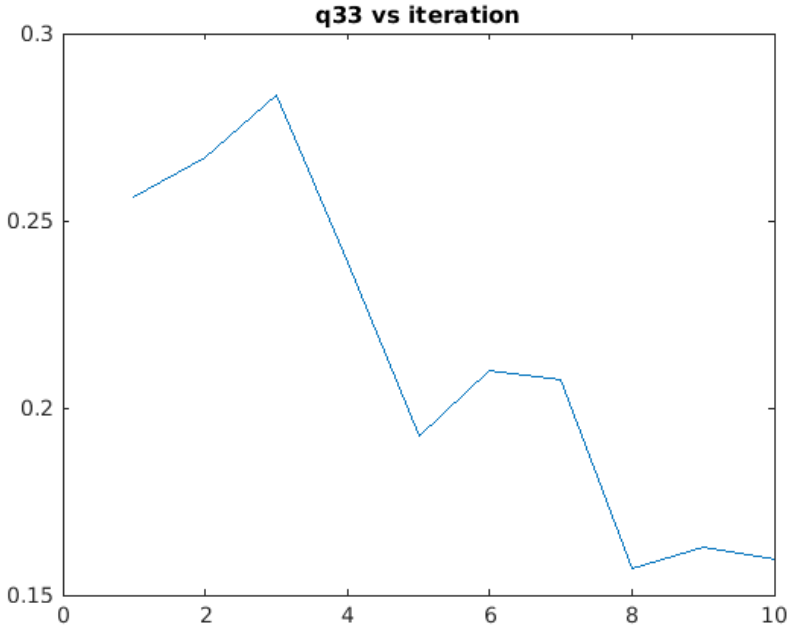


Fig. 24: q33 vs iteration graph

- Graph for getting q44 vs iteration using Genetic algorithm

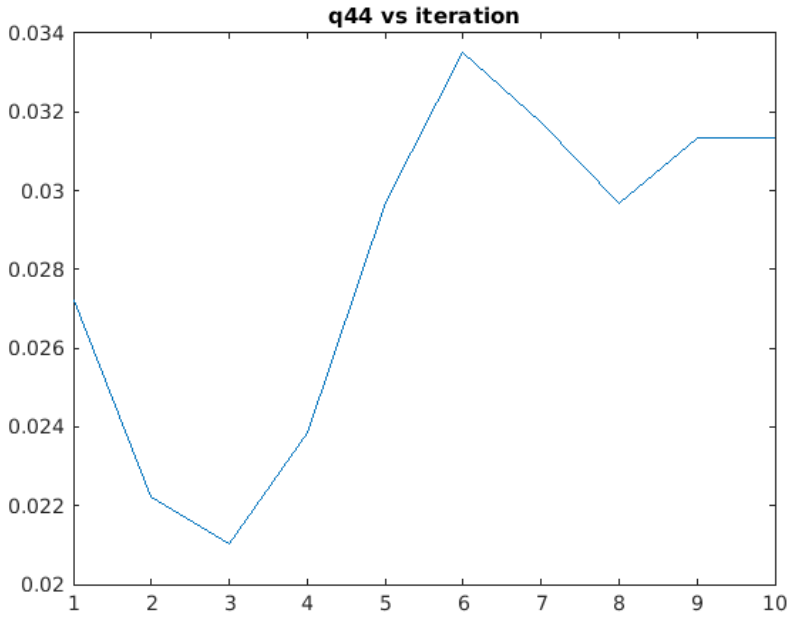


Fig. 25: q44 vs iteration graph

- Graph for getting r vs iteration using Genetic algorithm

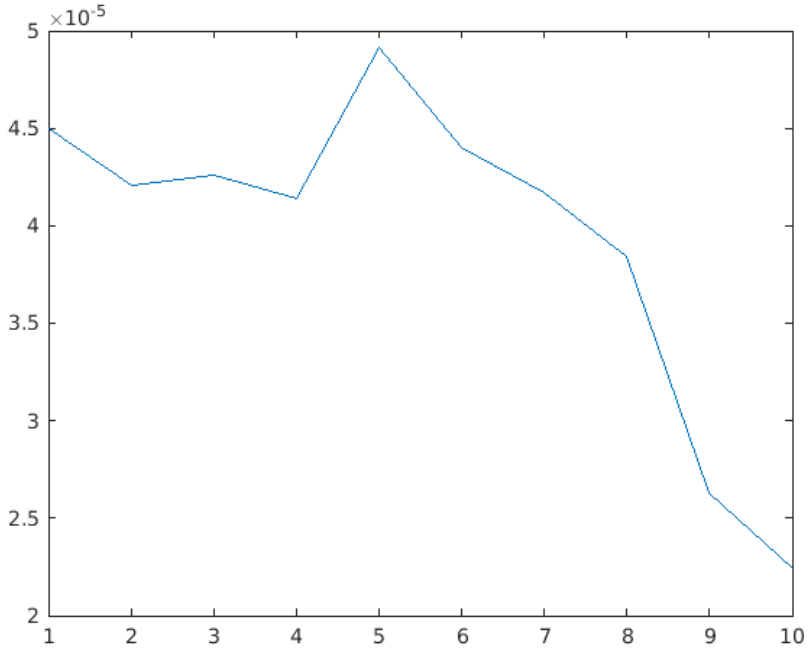


Fig. 26: r vs iteration graph

5.3 BFA-LQR Results

5.3.1 Parameters

Fitness Function = Rise Time + Settling Time + Overshoot

Number of elimination and dispersal steps (Ne)=10;
Reproductive Steps (Nr)=10;
Chemotactic Steps (Nc)=10;
Population Size (Np)=5;
Swim Steps (Ns)=8;
Swarm Size (D)=5;
Probability of elimination (Ped)=0.9;
The run-length unit i.e., the chemotactic step size during each run or tumble (C)=0.01;

Fitness Function Value = 2.170198213995942

Value of Q and R obtained =
[q11 q22 q33 q44 R] = [q = [72471.87066219141 -0490.898006878771 -04750.45905426314
-05.18027662623499 0.401824162185799];]

Value of K obtained from it =
1.0e+02 * [-4.246847946535302 -2.057666744004326 -2.923004678200982 -0.430108120238977]

5.3.2 Graphs

- Voltage Input vs time graph made in Simulink

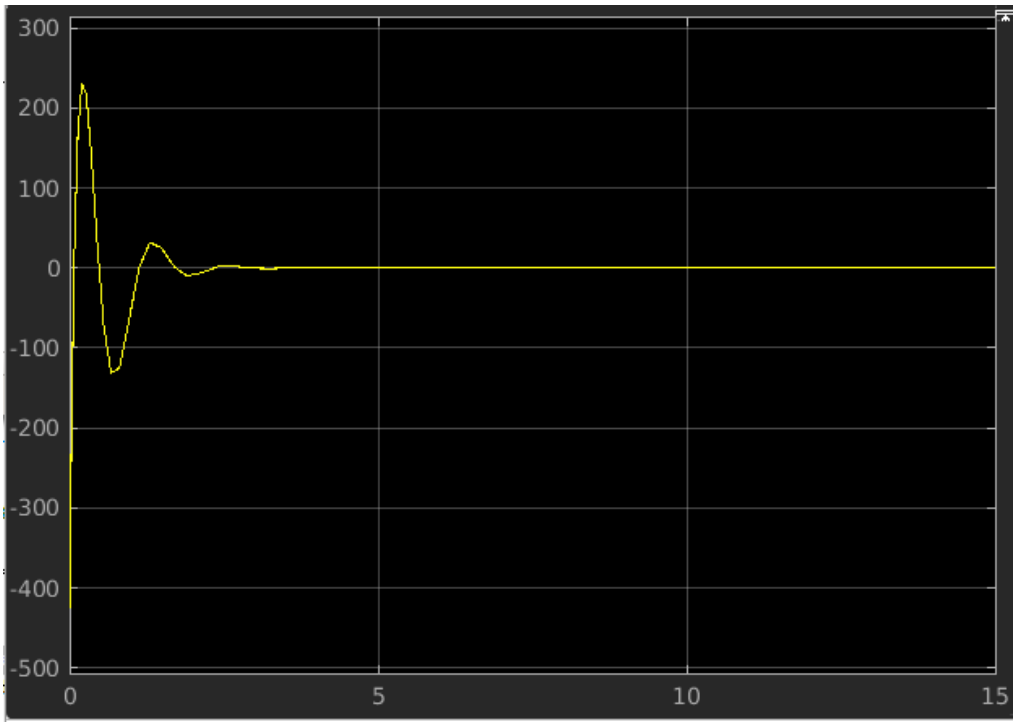


Fig. 27: Voltage vs time graph

- Position vs Time using transfer function model in MATLAB

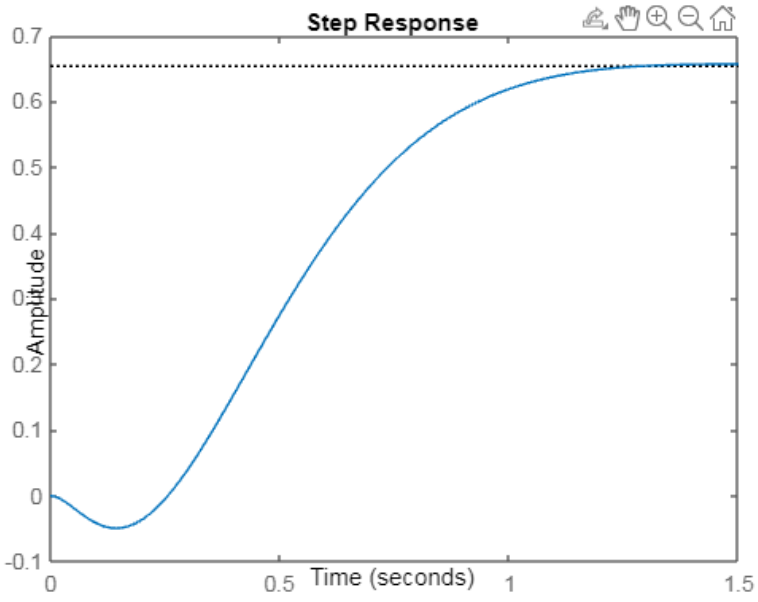


Fig. 28: Position vs time graph

- Position vs Time using State Space model in simulink

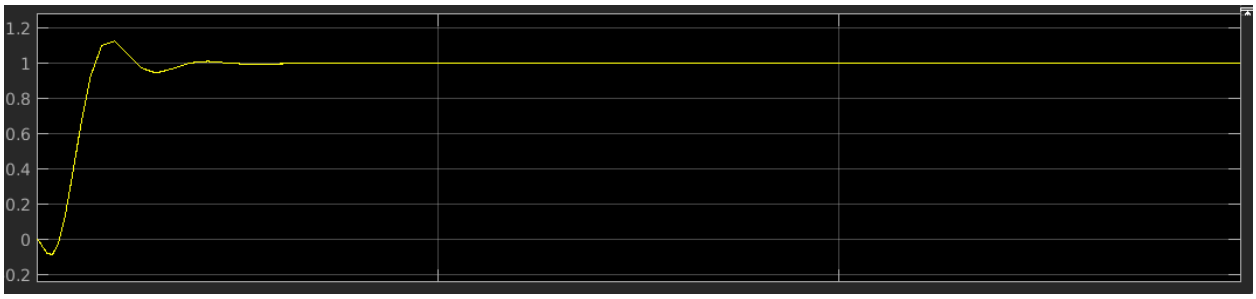


Fig. 29: Position vs Time using Space model

- Angle vs Time using State Space model in simulink

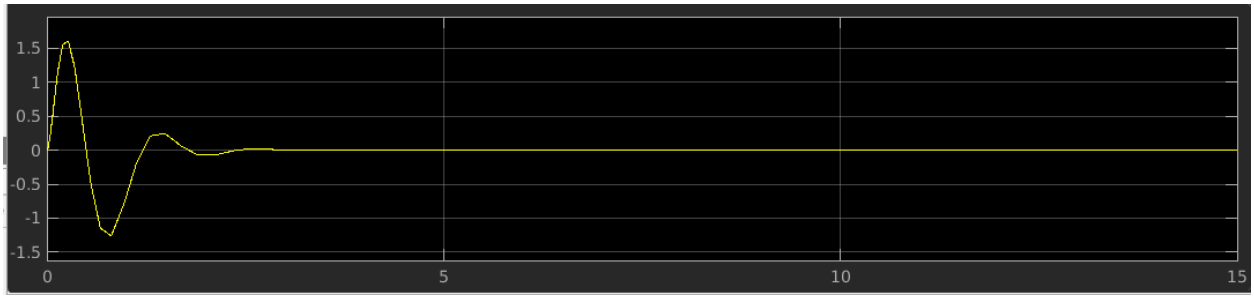


Fig. 30: Angle vs Time using Space model

- Angle vs Time using transfer Function model in MATLAB

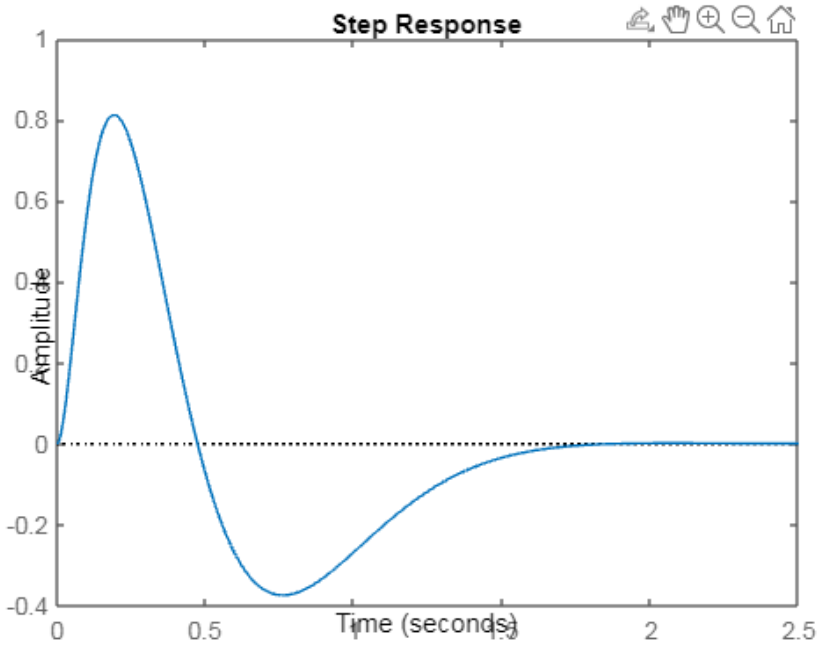


Fig. 31: Step response vs amplitude

6. Future work and scope

- The developed model is a baseline for many other research works. The use of a simple inverted pendulum controller scheme has been extended to our model. We have not incorporated the turning scenario which would involve non-uniform voltage inputs to both the wheels.

- We have only used a state-feedback LQR for tuning the controller gains. Instead many other controllers like PID and its variants(like Fuzzy and FOPID) could be used. Even a simple Q-learning based approach could also be incorporated.

- Similarly, for tuning the controller parameters, we could have used any other data-based(DL) and data-free methods. We had planned to use PSO and ACO along with BFO and GA, but were not able to complete it on time.

- Further, the distance-speed graphs obtained can be sampled and provided to any navigation algorithm using Artificial Potential field and be used for autonomous navigation of the segway. This hardware model can be easily applied to any microcontroller.

- State estimation Kalman filters could have been used to find the accurate values of the output variables, and this could have made the model more robust and real-world. In our output graphs, we do not see ripples, which would have been manifested by using a state estimation filter.

7. Results and Conclusion and Observations

7.1 Observations and Discussion

A. GA LQR seems to perform less better and requires less computational power. It took around 1 sec for execution of GA and 4min for complete execution of BFO. Therefore the conclusion obtained by us is different from that mentioned in paper. Also the value of the fitness function for the most optimized result is the same for GA and BFO whereas that should have been different for both. This is due to the fact that steady state is not an error and is not included in the fitness function.

B. According to our modelling we simulated the results using both Steady state and Transfer function models for a state feedback control scheme. This was different from the standard controller because of the fact that modelling a TF model for such a controller needs heavy mathematical manipulations.

C. For our model, we incorporated a deviational set-point scheme (not present in paper) and provided it with a step input to see the results. The graphs were obtained as expected after mathematical manipulations and tunings.

D. We also tried using an animation model, made by Columbia University professors, into our model to see the results of the segway model. But, we could not get the running animation, but only the final figure :

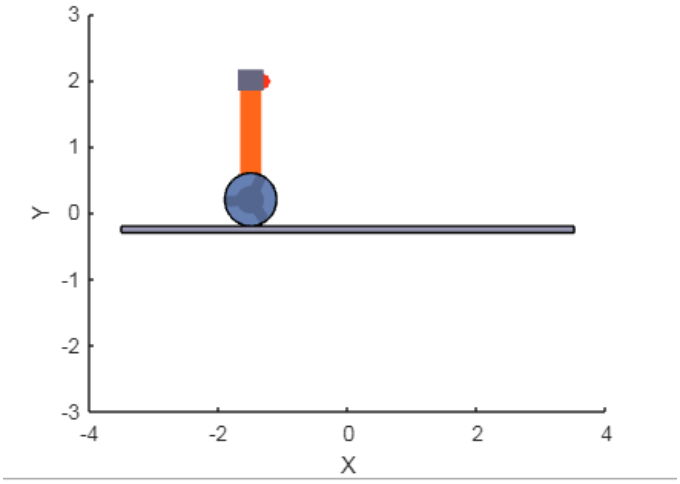


Fig. 32: Segway Animation

7.2 Conclusions

In this work, an optimal linear control system was adopted to balance a Segway two-wheeled mobile robot. First state space model of the plant was derived and the model was simulated both in MATLAB and SIMULINK for a step input to position. To improve the results for LQR two biologically inspired algorithms were implemented, GA and BFO, to find optimal values of Q and R. According to the obtained results it was found that GA LQR had better performance than BFO. Though the mentioned conclusion is different from one that is mentioned in the paper but values provided by them also provide simulation results similar to that done by us.

8. References

- Mohammed, I. K., & Abdulla, A. I. (2020). Balancing a Segway robot using LQR controller based on genetic and bacteria foraging optimization algorithms. TELKOMNIKA (Telecommunication Computing Electronics and Control), 18(5), 2642. https://doi.org/10.12928/telkomnika.v18i5.14717
- EIT TUK. (2018, February 7). State Space Control for the Pendulum-Cart System: A short tutorial on using Matlab® and Simulink®. YouTube
- mouhknowsbest. (2013, June 24). 4.4 Segway Robots. YouTube
- Bhanot, S. (2021). Process Control: Principles And Application. OXFORD UNIVERSITY PRESS.
- EIT TUK. (2018, February 7). State Space Control for the Pendulum-Cart System: A short tutorial on using Matlab® and Simulink®. YouTube
- T. (2018). turnwald/CAE_Exercise. GitHub
- NCTEL. (2016, May 19). Bacterial Foraging Optimization by Er Neha Sharma. YouTube
- Chen, H., Zhu, Y., & Hu, K. (2011). Adaptive Bacterial Foraging Optimization. Abstract and Applied Analysis, 2011, 1/27. https://doi.org/10.1155/2011/108269
- A. (2017). avandekleut/bacterial-foraging. Biomimicry of Bacterial Foraging for Optimization and Control.
- Chen, H., Zhu, Y., & Hu, K. (2011). Adaptive Bacterial Foraging Optimization. Abstract and Applied Analysis, 2011, 1–27. https://doi.org/10.1155/2011/108269
- MathWorks. (2021). Linear-Quadratic Regulator (LQR) design - MATLAB lqr.Design a LQR controller in MATLAB.
- Steve Brunton. (2017, January 29). Linear Quadratic Regulator (LQR) Control for the Inverted Pendulum on a Cart [Control Bootcamp]. YouTube
- MIT Robotics. (2021). Course | edX. Introduction to Control System Design-Computational State Space Approaches.
- S. Chantarachit, "Development and Control Segway by LQR adjustable Gain," 2019 International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 2019, pp. 649-653, doi: 10.1109/ICOIACT46704.2019.8938489.

APPENDIX

Appendix A: MATLAB Codes

1. For Model Development

```
%% Version 3 deals with the calculations and parameter estimation intelligent
control.
%%      We deal with a MIMO system here with state feedback topology. We have
derived the transfer functions of individual interacting
%%      elements and the result obtained in Simulink is thus, verified.
clear all;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Km = 1;
K1 = 2*Km*Ke;
beta = 2*Mw + Mp + 2*Iw/(r.^2);
alpha = Ip*beta + 2*Mp*(I.^2)*(Mw + Iw/(r.^2));
Kw = 2*Mw + 2*Iw/(r.^2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
R = 1.6;
L = 1.2e-3;
l = 0.16;
Mw = 0.02;
Mp = 0.52;
Iw = 0.0032;
Ip = 0.0038;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % X = [x , x_dot , theta , theta_dot]
    % X_dot = AX + BU
   % Y = CX + DU

   % Here, x is a state variable and X is the state matrix itself.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A22 = -0.0316;
A23 = 0.3817;
A42 = 0.3927;
A43 = 49.5531;
B2 = 0.05746;
B4 = -0.7141;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A = [0  1  0  0;
    0 A22 A23 0;
    0 0 0 1;
    0 A42 A43 0;];
B = [0 ; B2 ; 0 ; B4;];
C = [1 0 0 0;
   0 0 1 0];             % We want the distance and the angle as the output.
```

```
C_temp = [1 0 0 0];
D = [0; 0;];
X0 = [0 ;0 ;0 ;0;];              % For initializing the integrator.
sys = ss(A, B, C, D);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
q11 = 8.969;
q22 =0.308;
q33 =0.121;
q44 =0.0085;
R = 2.123e-5;
Q = [
   q11 0 0 0;
   0 q22 0 0;
   0 0 q33 0;
   0 0 0 q44;
];
[K,S,e] = lqr(A,B,Q,R);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
states = {'distance' 'speed' 'angle' 'angular velocity'};
inputs = {'Voltage'};
outputs = {'distance' 'angle'};
system = ss(A-B*K, B, C, D,
'statename',states,'inputname',inputs,'outputname',outputs);
%% In the command window, type tf(system)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sts_dist = tf([K(1)*0.05746, K(1)*7.145e-16, -K(1)*3.12],[1, 30.33, 284.3,
1202, 2028]);     % For distance
sts_angle = tf([K(1)*-0.7141,K(1)*- 1.018e-06,K(1)*-
5.936e-15],[1,30.33,284.3,1202,2028]);       % For angle.
step(sts_angle);
step(sts_dist);
```

2. For finding the Cost function for optimization

```
function J = Cost(parms)
   A22 = -0.0316;
   A23 = 0.3817;
   A42 = 0.3927;
   A43 = 49.5531;

   B2 = 0.05746;
   B4 = -0.7141;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   A = [0  1  0  0;
        0 A22 A23 0;
```

```
        0 0 0 1;
        0 A42 A43 0;];

   B = [0 ; B2 ; 0 ; B4;];

   C = [1 0 0 0;
       0 0 1 0];             % We want the distance and the angle as the
output.

   C_temp = [1 0 0 0];

   D = [0; 0;];

   X0 = [0 ;0 ;0 ;0;];             % For initializing the integrator.

   sys = ss(A, B, C, D);

   Q = [parms(1) 0 0 0;
       0 parms(2) 0 0;
       0 0 parms(3) 0;
       0 0 0 parms(4);];
   R = parms(5);

   [K,S,e] = lqr(A,B,Q,R);
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   states = {'distance' 'speed' 'angle' 'angular velocity'};
   inputs = {'Voltage'};
   outputs = {'distance' 'angle'};

   % system = ss(A-B*K, B, C, D,
'statename',states,'inputname',inputs,'outputname',outputs);

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   sts = tf([K(1)*0.05746, K(1)*7.145e-16, -K(1)*3.12],[1, 30.33, 284.3,
1202, 2028]);

   s = stepinfo(sts);
   J = abs(s.RiseTime)+ abs(s.SettlingTime) + abs(s.Overshoot);
   % riseTime = s.RiseTime
   % settlingTime = s.SettlingTime
   % overshoot = s.Overshoot

   step(sts);
   h = findobj(gcf,'type','line');
   set(h,'linewidth',2);
   drawnow
end
```

3. For GA-LQR

```
% Main code to run GA  and obtain Q and R. Might show some error for few
% times as randomly as value of initial Q and R might not be solvable for
% riccati equation
clear all, close all, clc
PopSize = 15;               %Population size
MaxGenerations = 10;        %Generation limit
options =
optimoptions(@ga,'PopulationSize',PopSize,'MaxGenerations',MaxGenerations
,'OutputFcn',@myfun);
format long %As R value is very small
[x,fval] = ga(@(K)Cost(K),5,[],[],[],[],[7 0 0 0 0],[10 0.5 0.5 0.05
1e-4],[],options)
```

4. For BFO-LQR

```
clear all;
close all;
clc;
% Bacteria Foraging Optimization %
% ------- initialisation ----------%
Ne=10;
Nr=10;
Nc=10;
Np=5;
Ns=8;
D=5;
C=0.01;
Ped=0.9; % elimination dispersion probability
% initializing bacteria with proper of x for which ricatti equation is
% solvable
x=  [ 7.000000000000000   0.500000000000000                   0
0   0.000100000000000;
  7.788751273517272   0.347898695070861   0.163385655339712
0.038252544147685   0.000007336042891;
  7.126858683930900   0.040608399663108   0.150562417774016
0.049294271749241   0.000093622850008;
  8.112320210505949   0.368102915584891   0.480736836721369
0.025368836870055   0.000076636604052;
  8.628307695765541   0.199041455630806   0.188079154272970
0.029127853009886   0.000016648572929;];

J=zeros(Np,1);
for k=1:Np
       J(k,1) = Cost([x(k,1) x(k,2) x(k,3) x(k,4) x(k,5)]); % initial
fitness calculation
end
```

```
Jlast=J;
for l=1:Ne
   for k=1:Nr
       Jchem=J;
       for j=1:Nc
           % Chemotaxis Loop %
           for i=1:Np-1
               del=(rand(1,D)-0.5)*2;
               x(i,:)=x(i,:)+(C/sqrt(del*del'))*del;
               J(i) = Cost([x(i,1) x(i,2) x(i,3) x(i,4) x(i,5)]);

               for m=1:Ns
                   if J(i)<Jlast(i)
                       Jlast(i)=J(i);
                       x(i,:)=x(i,:)+C*(del/sqrt(del*del'));
                       J(i) = Cost([x(i,1) x(i,2) x(i,3) x(i,4) x(i,5)]);
                   else
                       del=(rand(1,D)-0.5)*2;
                       x(i,:)=x(i,:)+C*(del/sqrt(del*del'));
                       J(i) = Cost([x(i,1) x(i,2) x(i,3) x(i,4) x(i,5)]);
                   end
               end

           end

           Jchem=[Jchem J];
       end  % End of Chemotaxis %


           for i=1:Np-1
               Jhealth(i)=sum(Jchem(i,:)); % sum of cost function of all
chemotactic loops for a given k & l
           end
           [Jhealth1,I]=sort(Jhealth,'ascend');
           x=[x(I(1:Np/2),:);x(I(1:Np/2),:)];
           J=[J(I(1:Np/2),:);J(I(1:Np/2),:)];
           xmin=x(I(1),:);
   end
   Jmin(l)=min(J);
   % random elimination dispersion

 for i=1:Np
       r=rand;
       if r>=Ped
           x(i,:)=(rand(1,D)-0.5);
           J(i) = Cost([x(i,1) x(i,2) x(i,3) x(i,4) x(i,5)]);
       end
   end
```

```
end
plot(Jmin);
```

Appendix B: Simulink Models

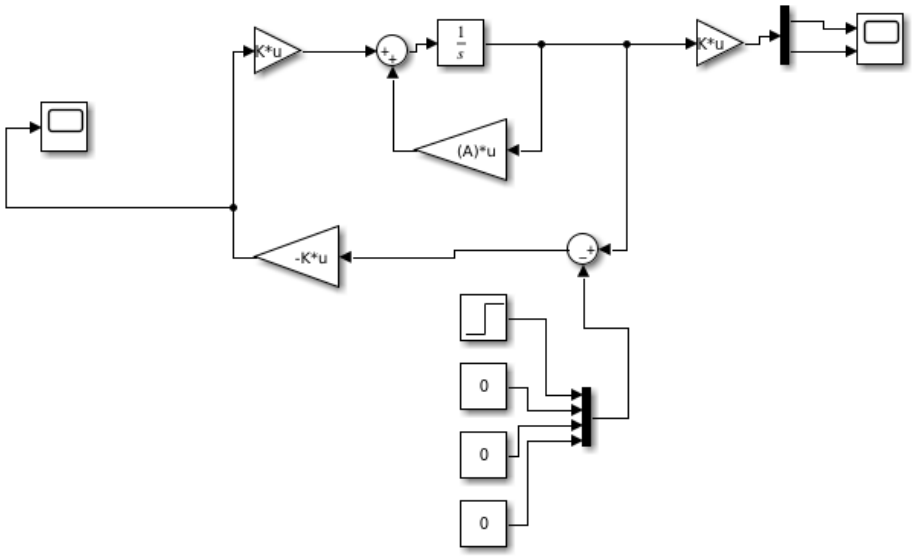


Fig. 33: Circuit diagram for simulink model

Appendix C: History of Genetic Algorithm

Similar values of cost are mentioned for each population is present in 'history.MAT' the code file submitted.

val(:,:,1) =

7.0000	0.5000	0	0	0.0001
7.7888	0.3479	0.1634	0.0383	0.0000
7.1269	0.0406	0.1506	0.0493	0.0001
8.1123	0.3681	0.4807	0.0254	0.0001
8.6283	0.1990	0.1881	0.0291	0.0000
9.5080	0.2921	0.3671	0.0425	0.0000
9.7243	0.0298	0.0336	0.0260	0.0000
8.8002	0.4137	0.4215	0.0019	0.0000
7.2131	0.4917	0.3957	0.0450	0.0000
7.0563	0.0611	0.1705	0.0277	0.0001
8.2670	0.2753	0.3011	0.0010	0.0001
9.5994	0.4782	0.4856	0.0466	0.0000
8.9572	0.0809	0.4509	0.0344	0.0000
9.3783	0.1332	0.1246	0.0305	0.0000
8.9395	0.1246	0.1114	0.0107	0.0000

val(:,:,2) =

8.2670	0.2753	0.3011	0.0010	0.0001
7.7888	0.1990	0.1881	0.0383	0.0000
7.2131	0.4917	0.3957	0.0254	0.0001
8.2670	0.4137	0.3011	0.0010	0.0000
9.5994	0.1332	0.4856	0.0466	0.0000
7.1269	0.0406	0.1506	0.0305	0.0001
9.5080	0.4137	0.4215	0.0019	0.0000
8.9395	0.1246	0.1881	0.0107	0.0000
7.0563	0.1990	0.1705	0.0291	0.0000
7.0000	0.2753	0.3011	0	0.0001
9.7243	0.2753	0.0336	0.0260	0.0000
9.3783	0.0809	0.4509	0.0344	0.0000
8.7888	0.3479	0.1634	0.0383	0.0000
7.2670	0.2753	0.3011	0.0010	0.0001
8.1269	0.0406	0.1506	0.0493	0.0001

val(:,:,3) =

8.2670	0.2753	0.3011	0.0010	0.0001
7.0563	0.0406	0.1506	0.0305	0.0000

7.2131	0.2753	0.3957	0.0010	0.0001
8.2670	0.2753	0.3957	0.0254	0.0001
9.5080	0.1332	0.4856	0.0466	0.0000
8.7888	0.3479	0.1634	0.0383	0.0000
8.2670	0.2753	0.1881	0.0107	0.0001
9.7243	0.0809	0.0336	0.0260	0.0000
8.1269	0.0406	0.1506	0.0493	0.0001
8.1269	0.0406	0.1506	0.0493	0.0001
7.7888	0.1990	0.4509	0.0344	0.0000
9.5994	0.2753	0.4856	0.0010	0.0000
9.2670	0.4137	0.3011	0.0010	0.0000
8.0000	0.2753	0.3011	0	0.0001
8.2670	0.2753	0.3011	0.0010	0.0001

val(:,:,4) =

7.0563	0.0406	0.1506	0.0305	0.0000
8.1269	0.0406	0.1506	0.0493	0.0001
7.2131	0.2753	0.1506	0.0010	0.0001
8.2670	0.1990	0.3011	0.0344	0.0000
7.0563	0.0406	0.1506	0.0305	0.0000
9.5080	0.1332	0.4856	0.0466	0.0000
7.0563	0.3479	0.1506	0.0383	0.0000
8.2670	0.4137	0.3011	0.0010	0.0001
8.2670	0.2753	0.3011	0	0.0001
8.2670	0.2753	0.3957	0.0107	0.0001
8.2670	0.2753	0.3957	0.0010	0.0001
8.2670	0.2753	0.3011	0.0010	0.0000
7.1269	0.0406	0.1506	0.0493	0.0001
8.7888	0.3479	0.1634	0.0383	0.0000
9.7243	0.0809	0.0336	0.0260	0.0000

val(:,:,5) =

7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.3479	0.1506	0.0383	0.0000
7.0563	0.0406	0.1506	0.0305	0.0000
8.2670	0.0406	0.3011	0.0305	0.0000
7.1269	0.0406	0.1506	0.0493	0.0000
9.5080	0.1332	0.4856	0.0493	0.0001
7.0563	0.0406	0.1506	0.0305	0.0001
7.0563	0.0406	0.1506	0.0305	0.0001
8.2670	0.2753	0.1506	0.0010	0.0001
8.2670	0.4137	0.3957	0.0107	0.0001
8.2670	0.2753	0.1506	0.0305	0.0001
7.0563	0.2753	0.1506	0.0010	0.0000
7.6269	0.0406	0.1506	0.0493	0.0001
9.2243	0.0809	0.0336	0.0260	0.0000
8.2888	0.3479	0.1634	0.0383	0.0000

val(:,:,6) =

7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.0406	0.1506	0.0305	0.0001
8.2670	0.3479	0.3957	0.0383	0.0001
8.2670	0.2753	0.1506	0.0493	0.0001
7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.0406	0.1506	0.0305	0.0001
8.2888	0.3479	0.1634	0.0493	0.0000
8.2888	0.3479	0.1634	0.0383	0.0000
7.0563	0.2753	0.1506	0.0010	0.0000
8.2670	0.0406	0.3011	0.0305	0.0000
7.0563	0.3479	0.1506	0.0383	0.0000
9.5080	0.1332	0.4856	0.0493	0.0000
9.2243	0.0809	0.0336	0.0260	0.0000
8.2670	0.2753	0.1506	0.0305	0.0001
7.0563	0.2906	0.4006	0.0305	0.0001

val(:,:,7) =

7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.2906	0.1506	0.0305	0.0000
7.0563	0.2753	0.1506	0.0305	0.0000
9.5080	0.0406	0.4856	0.0305	0.0001
8.2670	0.3479	0.3957	0.0383	0.0000
9.2243	0.0406	0.1506	0.0260	0.0000
7.0563	0.3479	0.1506	0.0493	0.0000
8.2888	0.3479	0.1634	0.0305	0.0000
8.2670	0.0406	0.1506	0.0305	0.0001
7.0563	0.2753	0.1506	0.0493	0.0001
7.0563	0.0406	0.1506	0.0305	0.0001
8.2888	0.2753	0.1634	0.0305	0.0000
7.0563	0.3479	0.1506	0.0383	0.0000
7.0563	0.2753	0.1506	0.0010	0.0000
7.0563	0.2906	0.4006	0.0305	0.0001

val(:,:,8) =

7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.2753	0.1506	0.0305	0.0001
7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.2906	0.1634	0.0305	0.0001
7.0563	0.3479	0.1506	0.0383	0.0000
7.0563	0.0406	0.1506	0.0305	0.0001
8.2670	0.3479	0.1506	0.0010	0.0000
7.0563	0.2906	0.1634	0.0305	0.0000
8.2888	0.2753	0.1634	0.0305	0.0000
9.2243	0.2753	0.1506	0.0260	0.0000
7.0563	0.3479	0.1506	0.0305	0.0000
7.0563	0.2906	0.1506	0.0305	0.0000
9.2868	0.0406	0.1506	0.0260	0.0000
8.2670	0.0406	0.2131	0.0305	0.0001
7.1071	0.2854	0.1506	0.0493	0.0000

val(:,:,9) =

7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.2753	0.1506	0.0260	0.0000
7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.2906	0.1634	0.0305	0.0000
9.2868	0.0406	0.1506	0.0305	0.0000
8.2888	0.2753	0.1634	0.0305	0.0000
7.0563	0.3479	0.1506	0.0260	0.0000
8.2670	0.2753	0.2131	0.0305	0.0001
7.0563	0.3479	0.1506	0.0305	0.0000
8.2670	0.3479	0.2131	0.0305	0.0000
7.1071	0.2854	0.1506	0.0493	0.0000
7.0563	0.0406	0.1506	0.0305	0.0001
7.0563	0.3479	0.1506	0.0383	0.0000
9.2868	0.0719	0.1506	0.0260	0.0000
7.0563	0.0406	0.1818	0.0305	0.0000

val(:,:,10) =

7.0563	0.0406	0.1506	0.0305	0.0000
7.0563	0.3479	0.1506	0.0305	0.0000
7.1071	0.3479	0.1506	0.0493	0.0000
8.2670	0.3479	0.2131	0.0305	0.0000
9.2868	0.0406	0.1506	0.0260	0.0000
7.0563	0.3479	0.1506	0.0305	0.0000
8.2670	0.3479	0.2131	0.0305	0.0000
7.0563	0.0719	0.1506	0.0260	0.0000
8.2670	0.0406	0.1506	0.0305	0.0000

8.2888	0.2753	0.1506	0.0260	0.0000
7.0563	0.3479	0.1506	0.0383	0.0000
7.0563	0.0406	0.1506	0.0305	0.0001
7.0563	0.2906	0.1634	0.0305	0.0000
7.0563	0.0406	0.1506	0.0148	0.0000
7.0563	0.0406	0.1506	0.0461	0.0000