ID3802 : OPEN ENDED LAB PROJECT

Reinforcement Learning (RL) control of a two-wheeled self-balancing robot

Team : - Manje Sushanth,
Electrical Department
122001025

M Anish Goud
Computer Science Department
112001020

Mentor : - Dr. Shaikshavali Chitraganti
Assistant Professor,
Electrical Department,
IIT Palakkad

Abstract— This project aims to implement optimal control for a two-wheeled self-balancing robot (TWSBR) based on Reinforcement Learning (RL). The need for this method arose as the already existing methods require a precise model of the system .In this report, first an introduction is given about the Two wheeled Self Balancing Robot. Then it is physically modelled using Newtonian Mechanics. From the mathematical model a state space representation of the system is obtained. Then the system is decoupled into two sub systems. For the sub systems, Q Learning based control methods are proposed. Q Learning Value Iteration (VI) and Q Learning Policy Iteration (PI) are implemented and the results are obtained. Further efforts are made to implement control based on Multi Step Q Learning.

Key Terms— Two Wheeled Self Balancing Robot (TWSBR), Q Learning based control, Value Iteration, Policy Iteration, Multi Step Q Learning.

I. INTRODUCTION

THE Two wheeled self balancing robot (TWSBR) has a pair of identical wheels, along with their actuators, a chassis, and an inverted pendulum. The chassis sustains the inverted pendulum and the pair of wheels. Basically, Two wheeled self balancing robot exhibits three types of motion: - Linear, tilt and yaw motion. The robot is able to execute linear motion along the X-axis, rotate around the Z-axis to execute tilt motion, and rotate around the Y-axis to execute yaw motion.

Now, we will see the need for the RL based control for the proposed system.

Need for Reinforcement Learning Based Control

In earlier publications, conventional control techniques including PID control, fuzzy control, and sliding mode control were proposed. While some of these techniques can control the TWSBR without exact knowledge of the system parameters, they do not always result in optimal control. In real applications, it is frequently desirable to go beyond simple stabilization to attain optimality.

Traditionally optimal control can be obtained by Linear Quadratic Regulation (LQR). This is based on the solution of algebraic riccati equation. But to solve this equation we require precise model of system parameters. We propose a scheme using RL which is completely online and which doesn't require the precise model of the system. Now we will see about the physical modelling of the system.

II. PHYSICAL MODELLING OF THE SYSTEM

The Two-Wheeled Self-Balancing robot is modelled using Newtonian mechanics.

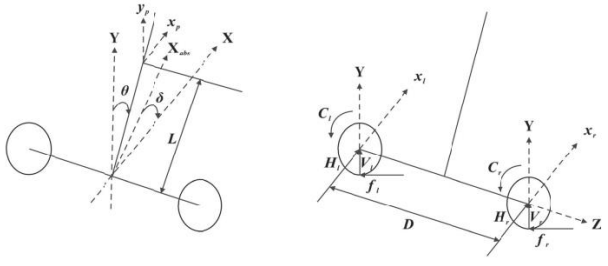

Fig 1. FBD of the TWSBR

The figure 1 shows the free body diagram (FBD) of the TWSBR. The dynamics of the left wheel and the right wheel are written and then the system is linearized around

$\Theta = 0$. Then the following linear state space representation of the system is obtained.

$$\begin{bmatrix} \dot{x} \\ \dot{v} \\ \dot{\theta} \\ \dot{\omega} \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & a_{43} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ v \\ \theta \\ \omega \\ \delta \\ \dot{\delta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ b_{21} & b_{22} \\ 0 & 0 \\ b_{41} & b_{42} \\ 0 & 0 \\ b_{61} & b_{62} \end{bmatrix} \begin{bmatrix} C_l \\ C_r \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ v \\ \theta \\ \omega \\ \delta \\ \dot{\delta} \end{bmatrix},$$

Where :- $\begin{bmatrix} x & v & \theta & \omega & \delta & \dot{\delta} \end{bmatrix}^T$ is the state vector and $\begin{bmatrix} C_l & C_r \end{bmatrix}^T$ is the input velocity vector $y = \begin{bmatrix} x & \theta & \delta \end{bmatrix}^T$ is the output vector. The parameters in the above equations are described below: -

$$a_{23} = \frac{-M^2L^2g}{MJ_p + 2\left(J_p + ML^2\right)\left(m + J_\omega/R^2\right)}$$

$$a_{43} = \frac{M^2gL + 2MgL\left(m + J_\omega/R^2\right)}{MJ_p + 2\left(J_p + ML^2\right)\left(m + J_\omega/R^2\right)}$$

$$b_{21} = b_{22} = \frac{\left(J_p + ML^2\right)/R + ML}{MJ_p + 2\left(J_p + ML^2\right)\left(m + J_\omega/R^2\right)}$$

$$b_{41} = b_{42} = \frac{-(R+L)M/R - 2\left(m + J_\omega/R^2\right)}{MJ_p + 2\left(J_p + ML^2\right)\left(m + J_\omega/R^2\right)}$$

$$b_{61} = -b_{62} = \frac{D/2R}{J_\delta + \frac{D^2}{2R}\left(mR + \frac{J_\omega}{R}\right)}.$$

The parameters in the above equation are given in the below table.

Symbol and Unit	Definition
M [kg]	Mass of the chassis (with the inverted pendulum part)
D [m]	Distance between the two wheels
R [m]	Radius of the wheel
L [m]	Distance between the center of gravity of the robot and the Z-axis
Jδ [kg·m2] -	Moment of inertia of the chassis with respect to the Y-axis
Jp [kg·m2]-	Moment of inertia of the chassis with respect to the Z-axis
Jω [kg·m2] -	Moment of inertia of the left (or right) wheel with respect to the Z-axis
m [kg] -	Mass of each wheel

Table-1: symbols and units , Definations

Now, the obtained system is of high order. So, decoupling is applied to the system to decouple the system into two subsystems. In order to make our algorithm converge faster a pre-feedback gain is also introduced.

III. Q LEARNING BASED CONTROL

Linear Quadratic Regulation (LQR) problem: -

Here, we are using Q - Learning reinforcement learning algorithm as this method is model free. For a discrete time linear invariant system: -

$$x(k+1) = Ax(k) + Bu(k),$$

A,B are controllable.

Now, in this project we aim to solve the Linear Quadratic Regulation(LQR) problem. The following is cost function for the LQR problem: -

$$J = \sum_{i=0}^{\infty} r\left(x_i, u_i\right) = \sum_{i=0}^{\infty} \left(x_i^T Q x_i + u_i^T R u_i\right),$$

where r(xi,ui) is the reward function or one step utility function. It's the reward you get when you are in state xi and perform action ui.

From the LQR control the optimal control law is: -

$$u_k^* = K^* x_k = -\left(R + B^T P^* B\right)^{-1} B^T P^* A x_k,$$

In the above equation for the optimal control law, P is the solution of Algebraic Riccatti Equation (ARE).

$$A^T PA - P + Q - A^T PB\left(R + B^T PB\right)^{-1} B^T PA = 0.$$

So, to get the solution for the optimal control we need P, and to get P we need to solve Riccati equation. Solving Riccati equation needs the knowledge of system parameters in the matrices A and B. So, we now see about Q Learning method to realize optimal control without the precise knowledge of system parameters.

Q Learning Based Control

The cost function (Value Function) is defined as: -

$$V_K\left(x_k\right) = \sum_{i=k}^{\infty} r\left(x_i, u_i\right),$$

So, the cost of following a control policy u(k) = Kx(k): -

$$V_K\left(x_k\right) = x_k^T P x_k,\ P = P^T > 0.$$

Applying Bellman's principle: -

$$V_K\left(x_k\right) = r\left(x_k, Kx_k\right) + V_K\left(x_{k+1}\right).$$

Now, we define the Q function: -

$$Q_K\left(x_k, u_k\right) = r\left(x_k, u_k\right) + V_K\left(x_{k+1}\right),$$

Q Function: - It is defined as the sum of single step cost of following a arbitrary control policy u(k) from state x(k) and the total cost of implementing a policy K from state of x(k+1) and all future states.

Further Q function can be written as: -

$$\begin{aligned} Q_K(x_k, u_k) &= x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P x_{k+1} \\ &= x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T P (Ax_k + Bu_k) \\ &= \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H_{xx} & H_{xu} \\ H_{ux} & H_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \\ &\triangleq z_k^T H z_k, \end{aligned}$$

Where: - $z_k = \begin{bmatrix} x_k^T & u_k^T \end{bmatrix}^T$.

The submatrices in the above equation are: -

$$H_{xx} = Q + A^T P A \in \mathbb{R}^{n \times n}$$
$$H_{ux} = B^T P A \in \mathbb{R}^{m \times n}$$
$$H_{xu} = A^T P B \in \mathbb{R}^{n \times m}$$
$$H_{uu} = R + B^T P B \in \mathbb{R}^{m \times m}.$$

So, to get improved policy we have to make: -(∂/∂u(k))(Q(K)) = 0.
Defining the optimal Q function: -

$$Q^*(x_k, u_k) = r(x_k, u_k) + V^*(x_{k+1}).$$

The optimal control policy K* can be obtained as: -

$$K^* x_k = \arg\min_u (Q^*(x_k, u_k)),$$

Making (∂/∂u(k))(Q*(K)) = 0, we get: -

$$u_k^* = -(H_{uu}^*)^{-1} H_{ux}^* x_k.$$

Recursive form of Q function: -

$$Q_K(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k + Q_K(x_{k+1}, K x_{k+1})$$

$$z_k^T H z_k = x_k^T Q x_k + u_k^T R u_k + z_{k+1}^T H z_{k+1}.$$

The matrix H in the Q-function is related to the system dynamics and the cost function parameters, and is unknown.
Parametrizing the matrix H: -
Where : - H_bar = vec(H) z_bar
=zk ⊗ zk

$$Q_K = \overline{H}^T \bar{z}_k,$$

Final Bellman Equation:

$$\overline{H}^T \bar{z}_k = x_k^T Q x_k + u_k^T R u_k + \overline{H}^T \bar{z}_{k+1}.$$

The matrix H is unknown and needs to be learned.

Now, to implement to get Optimal Control Policy, we implemented two algorithms:
1)Q-Learning Policy Iteration(PI)
2)Q-Learning Value Iteration(VI)

A. Q-Learning Policy Iteration(PI)

Step – 1 : -
Initialization: - First, start with a stabilizing control policy: -
Step – 2 :-
Policy Evaluation: - Evaluate the policy by finding the least square solution of:

$$\left(\overline{H}^j\right)^T (\bar{z}_k - \bar{z}_{k+1}) = x_k^T Q x_k + u_k^T R u_k$$

Step – 3 :-
Policy Update: - Finding the improved control policy: -

Step – 4: -

$$u_k^{j+1} = -\left(H_{uu}^j\right)^{-1} H_{ux}^j x_k$$

Step – 5 :-

Check for convergence: - where ε is a small positive scalar that can be set according to desired convergence accuracy.

$$\| \overline{H}^j - \overline{H}^{j-1} \| < \epsilon$$

Repeat until the convergence criterion is met.

B. Q-Learning Value Iteration(VI)

Step – 1 : -
Initialization: - First, start with a arbitrary control policy and initialize H0 >= 0
Step – 2 : -
Value Update: Determine the least square solution of:

$$\left(\overline{H}^{j+1}\right)^T \bar{z}_k = x_k^T Q x_k + u_k^T R u_k + \left(\overline{H}^j\right)^T \bar{z}_{k+1}$$

Step – 3: -
Policy Update: Finding the improved control policy:

$$u_k^{j+1} = -\left(H_{uu}^{j+1}\right)^{-1} H_{ux}^{j+1} x_k$$

Step - 4 : -
Check for convergence: -
where ε is a small positive scalar that can be set according to desired convergence accuracy

$$\| \overline{H}^{j+1} - \overline{H}^j \| < \epsilon$$

Repeat until the convergence criterion is met

The above two policies are implemented and the results are obtained. The optimal H obtained in the above two are compared with the H obtained from LQR. The states of the system with each time step are also obtained.

IV. RESULTS

A. Q-Learning Policy Iteration(PI)

H_optimal obtained by solving ARE (LQR): -

```
H_optimal =

   1.0e+03 *

    0.1267    0.1292    0.0619    0.0575   -0.0040
    0.1292    0.2961    0.0939    0.1319   -0.0098
    0.0619    0.0939    1.4111    0.0166    0.0361
    0.0575    0.1319    0.0166    0.0708   -0.0072
   -0.0040   -0.0098    0.0361   -0.0072    0.0032
```

H_optimal obtained from PI: -

```
H =

   1.0e+03 *

    0.1267    0.1292    0.0619    0.0575   -0.0040
    0.1292    0.2961    0.0939    0.1319   -0.0098
    0.0619    0.0939    1.4111    0.0166    0.0361
    0.0575    0.1319    0.0166    0.0708   -0.0072
   -0.0040   -0.0098    0.0361   -0.0072    0.0032
```

The State Trajectories: -

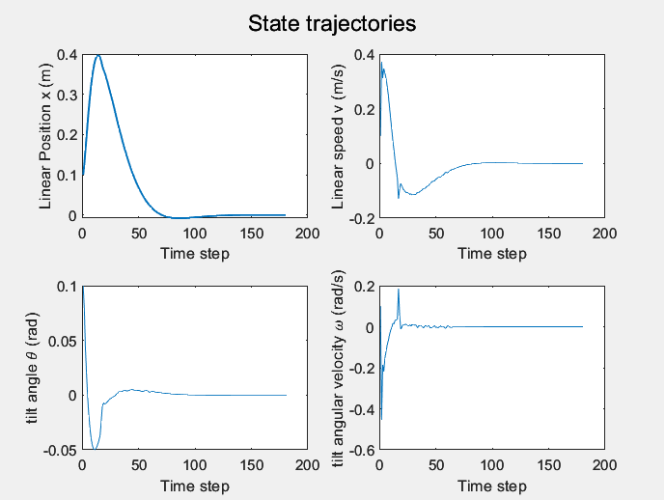

Fig 2: Plot of the state trajectories form PI

They are being stabilized.

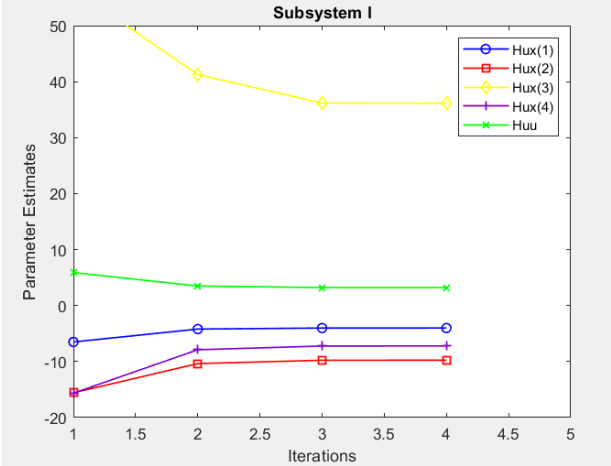

Fig 3 : - Plot of H matrix parameters for PI

B. Q-Learning Value Iteration(VI)

H_optimal obtained from VI: -

```
H =

   1.0e+03 *

    0.1250    0.1275    0.0610    0.0567   -0.0040
    0.1275    0.2935    0.0930    0.1307   -0.0097
    0.0610    0.0930    1.4107    0.0162    0.0362
    0.0567    0.1307    0.0162    0.0702   -0.0072
   -0.0040   -0.0097    0.0362   -0.0072    0.0032
```

The State Trajectories: -

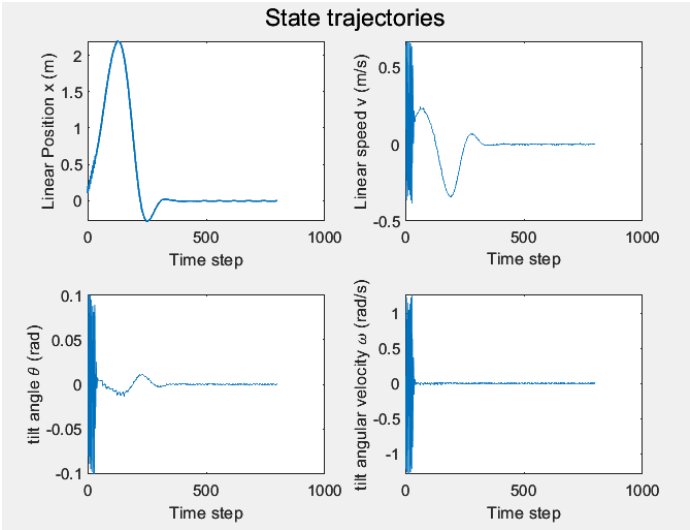

Fig 4: Plot of the state trajectories form VI

We can see that they are being stabilized.

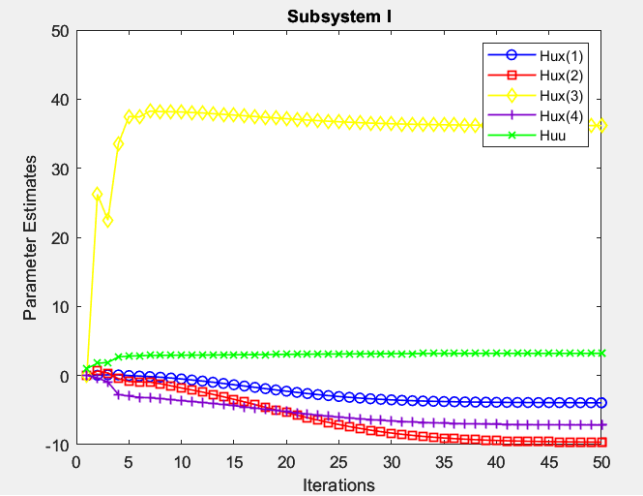

Fig 4 : - Plot of H matrix parameters for VI

V. MULTI STEP Q LEARNING – EXTENSION TO Q LEARNING (FUTURE PERSPECTIVE)

In this project we have also attempted to extend the Q Learning algorithm to Multistep Q Learning. In Q learning for calculating Q function we go one step ahead and add the reward we get from that state if we take action uk and the value function of that state. In Multistep Q Learning instead of ahead 1 step we go multiple steps and add the rewards we get from each of the state with the value function of the final state to calculate Q function. This is expected to converge more in less time than the Q learning algorithm. We tried to implement this multistep Q Learning algorithm for Quanser Helicopter. The results are yet to be obtained and can be carried out for future. This Multi Step Q Learning can also be implemented for our Two-wheeled self balancing robot. The results from both the methods can be compared with each other.

VI. CONCLUSION

The optimal control method for Two-Wheeled Self Balancing Robot (TWSBR) has been implemented using Q Learning Value Iteration and Q Learning Policy Iteration methods. It has been observed that the in both the algorithms the state trajectories are being stabilized. Comparing both, PI algorithm has a better state response. Further, attempts have been made to extend Q Learning to Multi Step Q Learning (MsQL). The results are yet to be obtained and this can be implemented in future. It is expected that the MsQL will converge faster than Q Learning algorithm.

VII. REFERNCES

[1] Guo L, Rizvi SAA, Lin Z. Optimal control of a two-wheeled self-balancing robot by reinforcement learning.
[2] B. Luo, H. -N. Wu and T. Huang, "Optimal Output Regulation for Model-Free Quanser Helicopter With Multistep Q-Learning," in IEEE Transactions on Industrial Electronics, vol. 65, no. 6, pp. 4953-4961, June 2018, doi: 10.1109/TIE.2017.2772162.
[3] Slides - Introduction to Reinforcement Learning with David Silver.