Performance Analysis of LQG Controller for Stabilizing a Two-wheeled Self-balancing Robot

Firomsa Ambaye Siyum

A thesis Submitted to

Department of Electrical Power and Control Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Masters in

Electrical Power and Control Engineering (Control Engineering)

Office of Graduate Studies

Adama Science and Technology University

June 2025

Adama, Ethiopia

Performance Analysis of LQG Controller for Stabilizing a Two-wheeled Self-balancing Robot

Firomsa Ambaye Siyum

Advisor Dr.Endalew Ayenew

A thesis Submitted to

Department of Electrical Power and Control Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Masters in

Electrical Power and Control Engineering (Control Engineering)

Office of Graduate Studies

Adama Science and Technology University

June 2025

Adama, Ethiopia

DECLARATION

I declare that this thesis entitled "**Performance Analysis of LQG Controller to Stabilize for Two-wheeled self-balancing Robot**" is my work and has not been submitted to any university for similar purpose. The reference used in this thesis is recognized by appropriate citations.

_____	_____	_____
Name	signature	Date

RECOMMENDATION

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Performance Analysis of LQG Controller for Stabilizing Two-wheeled Self-balancing Robot**" prepared under my guidance by Firomsa Ambaye, submitted in partial fulfillment of the requirements for the degree of Masters of Science in Control Engineering. Therefore, I recommend the submission of a revised version of the thesis to the department following the applicable procedures.

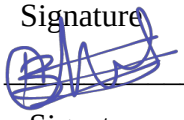
_____	_____	_____
Advisor	signature	Date
_____	_____	_____
Co-Advisor	signature	Date

APPROVAL PAGE

I hereby attest that the final thesis proposal, Performance Analysis of Optimal Adaptive Control for Stabilizing Two-wheeled Self-balancing Robot by Firomsa Ambaye, appropriately incorporates the recommendation and suggestion made by the proposal review committee.

_____	_____	_____
Advisor	signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by Firomsa Ambaye We, have read and evaluated the thesis entitled "Performance Analysis of Optimal Adaptive Control for Stabilizing Two-wheeled Self-balancing robot" and examined the candidate during open defense. This is, therefore to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Control System Engineering.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
Dr Beteley Teka		June 12, 2025
External Examiner	Signature	Date

Final approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC)

_____	_____	_____
Department	Head Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

AKNOWLEDGEMENT

Firstly, I would like to say thank you to Jesus for keeping me alive and healthier and my gratitude to collage of electrical and computer engineering and department of power and control engineering at ASTU for their help and idea to accomplish my thesis research. I would like to show my appreciation to my advisor Dr. Endalew Ayenew and Co-advisor Mr.Adisu Safo, for their guidance throughout my studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ASTU	Adama Science and Technology University
FPA	Flower pollination optimization
IP	Inverted pendulum,
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
PID	Proportional integral derivative
PDC	Proportional derivative controller
PSO	Particle swarm optimization
RIP	Rotary inverted pendulum
TWSB	Two wheeled self-balancing robot

LIST OF SYMBOLS

f_m	Friction between robot and motor
G	Robot's height
g	Acceleration due to gravity
H	Distance of wheel axle and robot's center
I	Part of robot's width
J	Radius of the wheel
J_M	Resistance of DC motor
K_{emf}	EMF constant for DC motor
K_I	Moment Inertia of wheel
K_θ	Moment Inertia of robot's pitch
K_δ	Moment Inertia of robot's yaw
K_M	Moment Inertia of DC motor
K_t	Constant Torque for DC motor
L	Robot's depth
M	Wheel mass
m	Robot body mass
n	Gear ratio

ABSTRACT

The stabilization of two-wheeled self-balancing robots (TWSBRs) is a critical challenge due to their inherent instability and complex dynamics. The concept of an inverted pendulum serves as a foundational model for engineers seeking to master their balance, ensuring the stability of these robots is paramount. Employing an adaptive intelligent control strategy, informed by the inverted pendulum principle, proves particularly effective when the robot initiates movement from varying initial tilt angles Comparative performance analysis of the linear quadratic gaussian (LQG) controller against the linear quadratic regulator (LQR) and its optimized variants, LQR-particle swarm optimization (LQR-PSO) and LQR-flower pollination algorithm (LQR-FPA), for stabilizing TWSBRs across various pitch reference angles. The analysis reveals that the LQG controller consistently demonstrates superior performance in minimizing both peak pitch deviation and peak wheel speed. For each pitch reference angle, LQG to achieve a peak pitch deviation of 0.035 degrees and a peak wheel speed of 0.045 degrees/second. Two-wheeled self-balancing robots, frequently modeled as inverted pendulums, represent a significant area of focus in robotics research. Their versatility spans applications from personal mobility to automated operations, and their inherent ability to maintain upright stability across diverse terrains renders them invaluable in various industries. Generally performance analysis of overall response self-balancing robot with LQR, LQR-PSO, LQR-FPA, and LQG at different reference angle 15°, 43° and 65°, that gives smaller peak pitch deviation and peak tracking error with LQR, LQR-PSO, LQR-FPA, LQG which is less than 1°. Design an optimal adaptive controller for self-balancing robots in the otherwise unstable vertical upright reference position. To accurately ascertain the robot lean angle and mitigate measurement uncertainties, Kalman filters are integrated. Simulation outcomes demonstrate the efficacy of this integrated approach in significantly bolstering robot stability, paving the way for more dependable self-balancing systems within the autonomous robotics domain.

Keywords: Inverted pendulum IP, LQG controller, LQR controller, two-wheeled self-balancing robot, integral optimal control, Kalman filters.

CHAPTER ONE

INTRODUCTION

1.1. Background of the study

The inverted pendulum stands out as a key benchmark in control theory. Its natural instability, interconnected movements, and strong nonlinear behaviors, along with its importance in robotics, make it an excellent way for automation engineers to test how well and how reliably control methods work. Unlike straightforward linear control, there aren't any all-encompassing solutions for nonlinear control, often requiring a very precise and applicable model of the system. As a result, how well deterministic nonlinear control performs greatly depends on how accurate the model is. However, creating such a detailed mathematical model can be very difficult. As (Hassan, 2024) point out, even small errors in the dynamic model's specifics can cause significant control problems, because the inverted pendulum is very sensitive to changes in its parameters. Similarly, discussions about self-balancing two-wheeled mobile robots, like those explored by (Dun, 2022), look into the dynamic relationship between ideal balance conditions and the realities of building and controlling these robots.

Sliding mode control is known for its ability to deal with imprecise models, ensure stability even with large disturbances, provide a good response, and be relatively easy to implement. It achieves robust control by using a control signal that changes abruptly across a defined boundary, meeting the conditions for sliding. Current advancements in humanoid robots require a deep understanding of how to synthesize robotic programming, which can be broken down into individual actuator movements. Examining an exploded view of the humanoid can improve the study of its dynamics. Cutting-edge areas in humanoid robotics primarily include walking on two legs, perception, interaction with humans, learning and adapting, and manipulation. These areas are major applications where studying inverted pendulum systems are vital. The rotary inverted pendulum (RIP) is a control challenge with common applications in systems and control, robotic arms, and various levels of machine control that involve complex dynamic systems. Adaptive control, which involves adjusting control settings in real-time based on feedback, allows robots to adjust to changing environments and uncertainties. Combining this adaptive control with optimal control methods, such as the Linear Quadratic Regulator (LQR), can improve stability and reliability.

The two-wheeled and self-balancing robot belongs to a multivariable, nonlinear, high order, strong coupling, and unstable essential motion control system, and it is a typical device of

testing various control theories and control methods; therefore, the research has great theoretical and practical significance. Because it has the advantages of simple structure, stable running, high energy utilization rate, and strong environmental adaption, it has the broad application prospects whether in the military field or in the civilian field. This paper concerns the self-designed and two-round self-balancing robot as the research object, which uses the Newtonian mechanics equation method and the linear method near the balance point to establish the linearized mathematical model of the system. In view of the mathematical model of the system, LQR controller is designed based on the particle swarm optimization and makes full use of the searching capability of the particle swarm algorithm to optimize the matrix Q and matrix R of the LQR controller. It gains the global optimal solution of matrix Q and R of the LQR controller so as to design the optimal state feedback control matrix K and overcome the disadvantages of relying on the experience and the trial and error in the selection of matrix Q and R of the general LQR control design, making up for the inadequacy of big workload. This method has better control effect by simulation tests and comparison.

The previous model showed the precise representation of the pendulum cart system. Although the mono-wheel system is analogous to the pendulum cart system, it is not precisely the same. The pendulum cart system gives the state position as the primary output. In contrast, for the mono-wheel system, we require the velocity to be the system output, and the reference input is a velocity that the rider provides manually. The reference point position state of all other states is controllable and observable in the pendulum cart system, which is in the continuous-time domain. However, the mono-wheel system works in discrete time. In reality, the input is not bounded, and hence manual constraints are needed. Converting the original LQR model into a more realistic system by replacing the initial position reference input with a joystick's velocity input overcomes such challenges. A saturation block is introduced to bind the intake within practical limits, followed by a rate limiter to bind the acceleration. The velocity input is then passed through an integrator to generate a corresponding position signal, which is fed into the plant. The entire model is converted into a discrete time-domain system to enable the smooth functioning of sensors and micro-controllers while working on the hardware.

The Linear Quadratic Gaussian (LQG) optimal control technique represents a modern control strategy that integrates a Kalman Filter (KF) with a Linear Quadratic Regulator (LQR) controller. LQG controllers are highly versatile, finding successful applications in both Linear Time-Variant (LTV) and Linear Time-Invariant (LTI) systems**.** They are particularly

effective in controlling dynamic systems subjected to disturbances like process noise and measurement noise, which can significantly impact system performance. Designing an LQG controller involves tuning several critical parameters: the state and control weighting matrices of the LQR controller **(**Q, R**)** and the Kalman Filter (Q_e,R_e). The goal is to achieve desired performance requirements. Historically, designers have relied on trial-and-error or complex manual procedures, which are far from optimal. This approach is time-consuming, labor-intensive, and heavily dependent on the control engineer's skill and knowledge, offering no guarantee of optimal controller parameters. Recognizing the limitations of manual tuning, researchers have increasingly turned to computational tuning algorithms to find optimal values for LQG controller parameters. Early attempts utilized classic frequency response techniques like Ziegler-Nichols and root locus methods to determine pole placement and PID controller gains. However, these classical methods often struggle with complex optimization schemes, facing issues such as lengthy calculation times or premature convergence. This led to the development of more powerful intelligent optimization methods**,** including, Genetic Algorithm (GA)**,** Bacteria Foraging Optimization Algorithm (BFOA)**,** Big Bang-Big Crunch (BBBC) algorithm**,** Particle Swarm Optimization (PSO) algorithm**,** Particle Swarm Inspired Evolutionary Algorithm (PS-IEA)**,** Artificial Bee Colony (ABC).Among these, the PSO tuning method has gained significant attention due to its superior ability to identify global solutions for both simple and complex control problems.

Introduced by (Kennedy and Eberhart, 1995) Particle Swarm Optimization (PSO) is an intelligent exploratory approach inspired by the collective behavior of individuals in a swarm, such as birds flocking. In PSO, a population of candidate solutions to control problems is represented by a number of particles**.** Each particle possesses unique position and velocity vectors. Based on Newtonian mechanics, these particles collectively navigate their trajectories toward the global optimum solution**.** Over the past few decades, numerous researchers have focused on designing classic and optimal controllers to actuate and stabilize various two-wheeled robot systems. For instance, LQR techniques have been applied to stabilize noiseless two-wheeled robots, often with PSO optimization for improved performance. Nonlinear adaptive controllers, including Neural Network control and PID controllers have been proposed to manage robot position and angle, demonstrating effectiveness in balancing and movement. LQR controllers combined with neural networks for self-tuning. Pole placement state feedback controllers and fuzzy logic controllers, with fuzzy logic showing better dynamic performance. Self-balance control for Segway-like robots using LQR, optimized by GA and BFOA, where BFOA-LQR demonstrated superior

tracking. Classic PID and optimal LQR controllers for position and angle control, with performance comparisons based on transient response.

Figure 1.1 Real world application area of inverted pendulum and wheeled robots (htts: www.pendbot.com)

Self-balancing self-balance robots benefit from high agility and rapid response times but are sensitive to disturbances. Conversely, differential drive robots are easier to control due to their grounded nature but lack agility in dynamic settings (Aldair, 2022) and Self-balancing robots often consume more power, particularly in maintaining balance. Differential drive robots are more energy-efficient at lower speeds, beneficial for long-duration tasks without frequent directional changes (Simon,2023).The integration of advanced adaptive controls and machine learning models is transforming the stability control of self-balancing robots, particularly those with inverted pendulum structures. While PID control remains foundational, it is increasingly supplemented by discrete adaptive controls, machine learning, and real-time optimization methods like QPSO and fuzzy logic. Limitation of this research is that while stability is crucial, the energy consumption of the control strategies is also important for practical applications. Future work could analyze and compare the energy efficiency of the different controllers.

1.2. Statement of the Problem

Crafting a dependable and effective control system for two-wheeled self-balancing robots presents a considerable hurdle, particularly when aiming for practical real-world use. These robots, often represented as inverted pendulums, demand finely tuned balancing control to stay upright and cope with disruptions. Sensor's measurement of the pitch angle is inaccuracy propagated to the wheel angle and pitch angle deviation, ultimately leading to system instability. To mitigate the impact of such unreliable sensor data on optimal feedback control, will now explore the linear quadratic gaussian (LQG) technique, which integrates an LQR

controller with a kalman filter. The inherent instability of an inverted pendulum, readily toppling with even slight forces upon activation, complicates both its identification and control. This research tackles key challenges, including ensuring the robot's stability maintaining its vertical stance during movement and when starting from various tilted positions. Effectively balancing such a robot is intricate, necessitating sophisticated control approaches capable of adapting to external disturbances and optimized for performance. Furthermore, the real-time demands of these dynamically changing robots require computationally lean adaptive control algorithms to deliver stable, fluid motion with minimal delays. Consequently, the control system must dynamically adapt its actions to preserve balance and achieve efficient, responsive movement. For a dynamic system demand high accuracy in maintaining their desired operating conditions by eliminating steady-state errors, we are introducing a tracking system that incorporates the integral of the wheel angle as a new state variable. This integration action is specifically designed to remove persistent deviations in the wheel angle, pitch angle and yaw angle; thereby ensuring the robot stabilizes at the reference position and follows a pre-defined trajectory. The resulting nonlinear state-space model will describe the two-wheeled robot with this added tracking functionality.

1.3. Motivation

Urban areas are getting more crowded, and we need better ways to deliver goods in that last mile. That's why there's a lot of excitement around nimble, two-wheeled robots that can move around easily and carry things or even help people get around. Self-balancing robots are especially promising because they can handle busy and uneven surfaces, which make them perfect for cities and indoor spaces. The key to making these robots work well is having a smart control system that can automatically adapt to different situations. This allows the robot to stay stable and efficient without needing someone to constantly control it. Think of two-wheeled self-balancing robots like upside-down pendulums they naturally want to fall over. To keep them upright, they need to constantly adjust. By using advanced optimal control methods, we can make these robots much more stable and energy-efficient. This is really important for them to work for longer periods in places like automated warehouses, hospitals, and as personal assistants.

1.4. Objectives

1.4.1. General objective

The main objective of this thesis is performance analysis of LQG control for stabilizing two-wheeled self-balancing robot.

1.4.2. Specific objective

- ➢ Develop mathematical model of overall system for two-wheeled self-balancing robot
- ➢ Design an optimal adaptive controller for self-balancing robots in the otherwise unstable vertical upright reference position.
- ➢ Analysis of system performance for linear quadratic regulator and linear quadratic (LQR) and Gaussian (LQG) applied to the models for concurrent control of the state variables of system.
- ➢ Design and show in MATLAB of optimal adaptive control for two-wheeled self-balancing robot with PSO and FPA.

1.5. Scope

Optimal adaptive control to stabilize inverted pendulum based two-wheeled self-balancing robot, is its capacity to deliver several key advantages: resilience, enhanced performance, reliable dynamic stability, learning capabilities, the ability to manage intricate control situations, improved safety, and adaptability across various uses. Together, these benefits enhance the effectiveness and dependability of inverted pendulum-based robots in different settings. By fine-tuning the control commands, the robot can react more quickly and with less overshoot, resulting in smoother and more effective balancing. Furthermore, optimal adaptive control conserves energy by identifying the most efficient control actions needed for stabilization.

1.6. Thesis organization

The work for thesis can be arranged as following below in which each chapter has their own strategies to define the system working principle.

Chapter one shows historical background, statement of the problem, objectives, scope, Motivation, expected outcome and relevance of the research. In the same way Chapter two shows techniques used in an inverted pendulum and wheeled robot. The provided a comprehensive review of autonomous navigation systems for mobile robots, including self-balancing types. Chapter three, this section outline material used the method to be followed, the mathematical modeling. Chapter four is about controller designing and optimization tuning with controller as particle swarm optimization and flower pollination optimization.

Chapter five shows the result and discussions related to the proposed controller and optimization techniques. Chapter six summarize the contribution of the research, conclusion, and recommendation, future work is explained.

CHAPTER TWO

REVIEW OF RELATED WORKS

2.1. Introduction

A convex optimization method has been used to create a mathematical representation and control system for an inverted pendulum mounted on a self-balancing two-wheeled cart. Small, immediate changes in the system are seen because the controller constantly works to keep the cart, the pendulum, and account for sensor inaccuracies stable. The graphs demonstrate that designing the control system involves finding a balance between making the system accurately follow a desired path and ensuring it remains stable (Wang, 2023).

2.2. Techniques used in an inverted pendulum and wheeled robot

Forward leaning of the pendulum is recorded as positive angles and backward leaning as negative angles. The aim is to see how the vehicle reacts to different disruptions during regular movement and how swiftly it can return to a stable state. Delve into the dynamics of ball-balancing robots, specifically the ballot, which utilizes a single spherical wheel for movement. The complexities of maintaining balance in such a nonlinear and unstable system, where conventional feedback control methods struggle, particularly in scenarios involving contact dynamics. They advocate for the integration of deep reinforcement learning (RL) with traditional control methods to enhance the robot's ability to manage significant initial tilt angles, showcasing the potential of RL to address challenges that conventional controllers cannot effectively solve.

The provided a comprehensive review of autonomous navigation systems for mobile robots, including self-balancing types. They emphasize the diversity of sensor combinations and mathematical approaches that contribute to localization, obstacle avoidance, and simultaneous localization and mapping (SLAM). Their findings underscore the critical role of deep learning in enhancing these navigation systems, which is particularly relevant for the development of agile self-balancing robots that must navigate complex environments. Their study introduces a sliding mode controller designed to improve lateral dynamic balance, addressing the often-overlooked aspect of anti-interference capabilities. By employing a combination of sliding mode and PD controllers, they demonstrate enhanced stability during motion, which is crucial for the practical application of unicycle robots in dynamic environments.

The design and control of a ballot drivetrain optimized for agility and payload capacity. Their work introduces a robust LQR-PI controller that successfully balances the drivetrain while carrying significant weights, illustrating the practical implications of control strategies for real-world applications. The ability to maneuver effectively in confined spaces with a high payload is a notable advancement in the field of self-balancing robots. By employing a sliding mode controller, the authors effectively manage lateral balance during motion, showcasing the robot's resilience to disturbances. This work underscores the importance of dynamic models in enhancing the mobility and robustness of unicycle robots. Their findings stress the significance of accurate mathematical modeling in designing control systems that respond effectively to the dynamic behavior of vehicles. By utilizing various modeling techniques, the authors aim to improve the reliability and safety of two-wheeled vehicles, emphasizing the necessity for sophisticated control algorithms that can adapt to complex riding scenarios.

The surrounding conventional control strategy for two-wheeled self-balancing robots has evolved significantly, with various methodologies being explored to enhance stability and performance. The foundational work by presents an advanced approach to control systems using Linear Quadratic Regulator (LQR) techniques. This study emphasizes the importance of optimizing control parameters for dynamic systems, setting a precedent for subsequent research that integrates LQR principles into more complex robotic applications. The novel method for selecting proportional-derivative (PD) settings based on linear-quadratic optimization. This approach not only enhances the tuning process but also demonstrates the practical application of LQR in achieving stability for two-wheeled systems, thereby contributing to the body of knowledge on control strategies in robotics. Further explores the challenges of conventional control systems by proposing an adaptive control strategy that leverages reinforcement learning (RL). This methodology addresses the limitations of traditional model-based approaches, particularly in scenarios where system parameters may vary. By demonstrating the effectiveness of an LQR-based RL method for stabilizing an inverted pendulum, work highlights the potential for adaptive techniques to enhance the robustness of control strategies in dynamic environments.

2.3. Related work

It is beneficial for systems that naturally have noise and uncertainty. LQG was used to stabilize a one-wheeled system, where the pendulum is attached to a single wheel. The Kalman Filter estimates the system's condition despite inaccuracies in sensor readings, and

the LQR adjusts for the best possible control. This combination allows the robot to handle significant and sudden disruptions, resulting in a balance that adjusts in real-time. This approach shows great promise for applications that require exact balance in environments with much interference, such as cities or industrial sites (Deb, 2021). Self-balancing robots, particularly two-wheeled and differential-drive types modeled as inverted pendulums, are a hot research topic. Adaptive control using machine learning is also being explored. These algorithms learn the robot's balancing needs in real-time and adjust accordingly, adapting to different loads and terrains without manual tuning (Qian, 2020). Some research combines neural networks with traditional linear controllers like PID to enhance real-time control. Literature review for two wheeled self-balancing robots with working principle and draw back. In control system design, a central idea is that the controller needs to find a sweet spot between how well the system performs (achieving goals like accurately following a target signal) and making sure the system remains stable. In control system design, a central idea is that the controller needs to find a sweet spot between how well the system performs (achieving goals like accurately following a target signal) and making sure the system remains stable. (Wang, 2023) In control system design, a central idea is that the controller needs to find a sweet spot between how well the system performs (achieving goals like accurately following a target signal) and making sure the system remains stable. Discrete controllers operate by measuring at predetermined intervals. The controller may not be able to recognize fast changes occurring inside the system if the measurements are not frequent enough (González-Cárdenas, 2024). This combination enables the robot to handle significant dynamic disturbances, achieving a real-time adaptive balance. This method shows significant potential for applications requiring fine-grained balance in high-noise environments, such as urban or industrial setting. With filtering, the sensors (IMUs, encoders, force sensors, etc.) might still be affected by the high-noise environments, leading to inaccurate readings or requiring complex and potentially lag-inducing filtering (Deb, 2021). PDC makes controlling complicated systems, like a two-wheeled robot where moving and staying upright are linked, easier by dividing the overall control plan into several separate, simple control actions. B y addressing input-output links individually, Parallel Decentralized Control (PDC) may fail to grasp the robot's global performance. (Guaraní et al, 2022). While self-balancing robots excel in quick movements and responsiveness, they are easily affected by external disruptions. If the disturbance is too sudden or too large, the robot might exceed its ability to compensate, leading to a loss of balance and potential failure of the task or even damage (Aldair, 2022). Self-balancing robots operate by constantly sensing their orientation and making immediate

adjustments to stay upright. This involves a system where sensors like IMUs track the robot's tilt and movement. A central controller processes this information using algorithms to determine the necessary motor actions. Despite their agility and quick responses, self-balancing robots face several fundamental limitations due to their unstable nature. They are easily thrown off balance by external disturbances and typically consume more power to maintain stability. Come with a higher cost (Qian, 2020). Fuzzy logic allows self-balancing robots to interpret sensor data in a flexible way using linguistic terms and rules to determine motor actions, resulting in smoother control, especially on uneven terrain. Despite its benefits, fuzzy logic control for self-balancing robots suffers from a lack of strong theoretical stability guarantees and relies heavily on expert-based rule design (Abut & Soyguder, 2019**).**

MPC explicitly handles constraints (Motor saturation, joint limits) by optimizing a predicted future trajectory over a finite horizon. For TWSBRs, MPC can be superior in scenarios requiring aggressive maneuvers or when strict constraints on motor torque or tilt angle are present. LQG can be seen as a special case of MPC with an infinite horizon and no explicit constraints. Comparisons often show that MPC can provide better trajectory tracking and handle constraints more gracefully, though at a higher computational cost.

For practical TWSBR applications, especially those in dynamic or safety-critical environments, fault tolerance is crucial. LQG can be adapted for fault-tolerant control by incorporating fault detection and isolation (FDI) mechanisms. If a sensor or actuator fails, the LQG controller can reconfigure itself based on the remaining healthy components, ensuring continued (albeit degraded) operation. This often involves multiple observer designs or switching control strategies. The quadratic cost function in LQG allows for explicit consideration of control effort, thereby influencing energy consumption. By appropriately weighting the control input (R matrix), the LQG controller can be designed to be more energy-efficient while still maintaining stability. This is particularly important for battery-powered TWSBRs.

2.4. Summary of literature review

Drift in the gyro sensor's measurement of the pitch angle are inaccuracy propagated to the wheel angle and pitch angle deviation, ultimately leading to system instability. To mitigate the impact of such unreliable sensor data on our optimal feedback control, we will now explore the linear quadratic gaussian (LQG) technique, which integrates an LQR controller with a Kalman filter. A recurring challenge in control system design, particularly for self-balancing robots, is the inherent difficulty in balancing performance with system stability.

This sweet spot is often elusive, as optimizing one aspect can compromise the other. Specifically, common drawbacks identified across various control strategies include. Self-balancing robots, by their unstable nature, are highly susceptible to external disturbances. If these disturbances are too sudden or large, the robot can exceed its compensatory abilities, leading to a loss of balance, task failure, or even damage. Even with filtering, sensors (IMUs, encoders, force sensors) in high-noise environments can produce inaccurate readings, potentially requiring complex and lag-inducing filtering techniques that still may not fully mitigate the noise. Controllers that operate at predetermined intervals may struggle to recognize rapid changes within the system if measurements are not frequent enough, hindering real-time adaptability in dynamic environments. While methods like parallel distributed control (PDC) simplify complex tasks by breaking them into individual actions, they may fail to grasp the robot's overall global performance by individually addressing input-output links. Despite these efforts, several limitations persist in previous research on robot actuation systems. Many proposed controllers were tuned manually or relied on basic classical methods like Ziegler-Nichols. Furthermore, some classic controller techniques, such as PID and fuzzy logic, often lack the robustness required to maintain the complex balance and stability of robot systems effectively. Crucially, most prior designs operated under the unrealistic assumption of a noise-free working environment.

CHAPTER THREE

MATERIALS AND METHODS

3.1. Mathematical modeling of a two-wheeled self-balancing robot

The principle of self-balance can be simply understood that when the pendulum bar moves forward in the direction of tilt, two wheeled vehicles should accelerate to avoid the pendulum bar down. The same way when the pendulum rod back tilt, a two wheeled vehicle decelerates. In this, the angle of the pendulum bar forward is recorded as positive, and the backward tilt is negative.

3.2. Flow chart

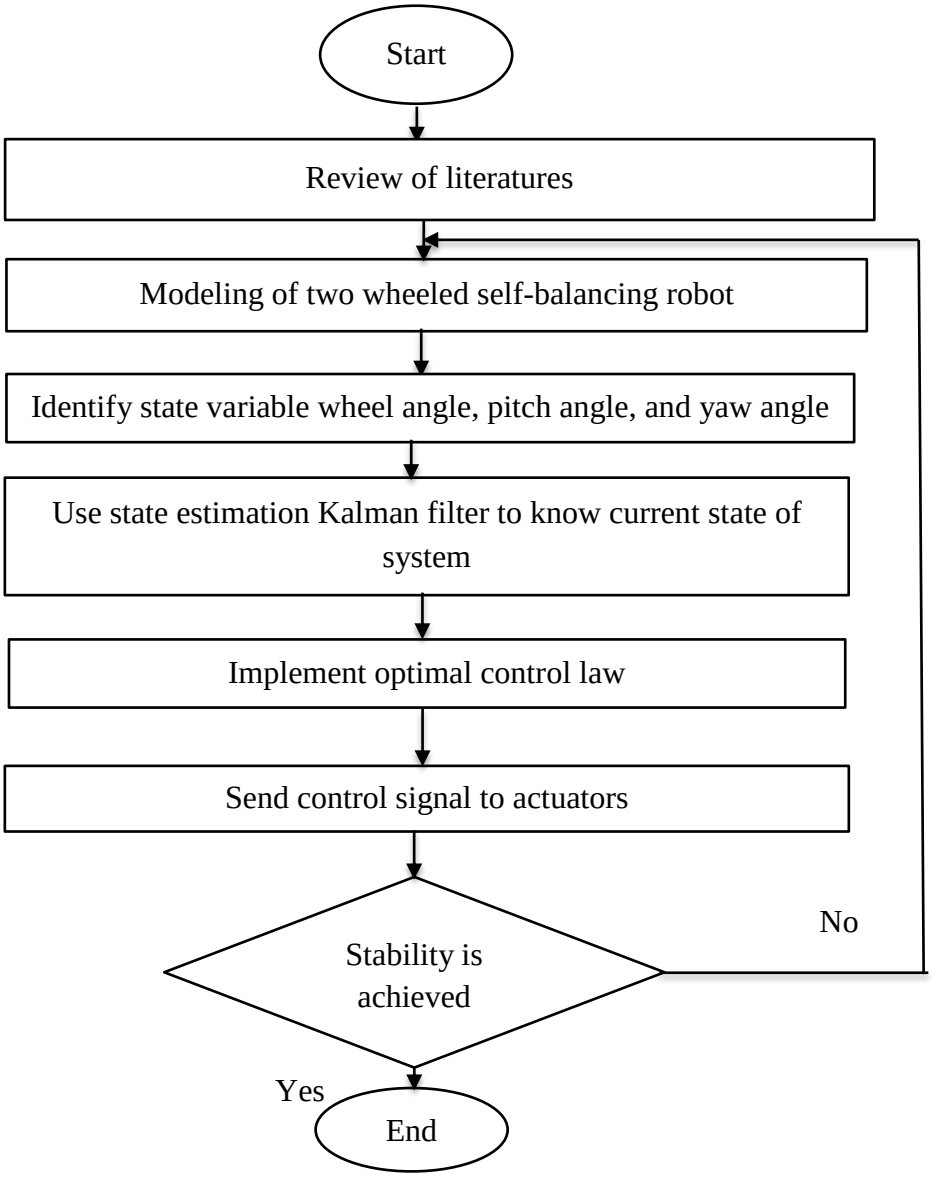


Figure 3.1Flow chart for TWSBR

As a classic model for bench-marking control strategies, the inverted pendulum on a cart has been extensively used, as detailed this section will expand upon the theory behind the inverted pendulum to encompass the self-balancing two-wheeled robot.

3.3. Block diagram of two wheeled self-balancing robot

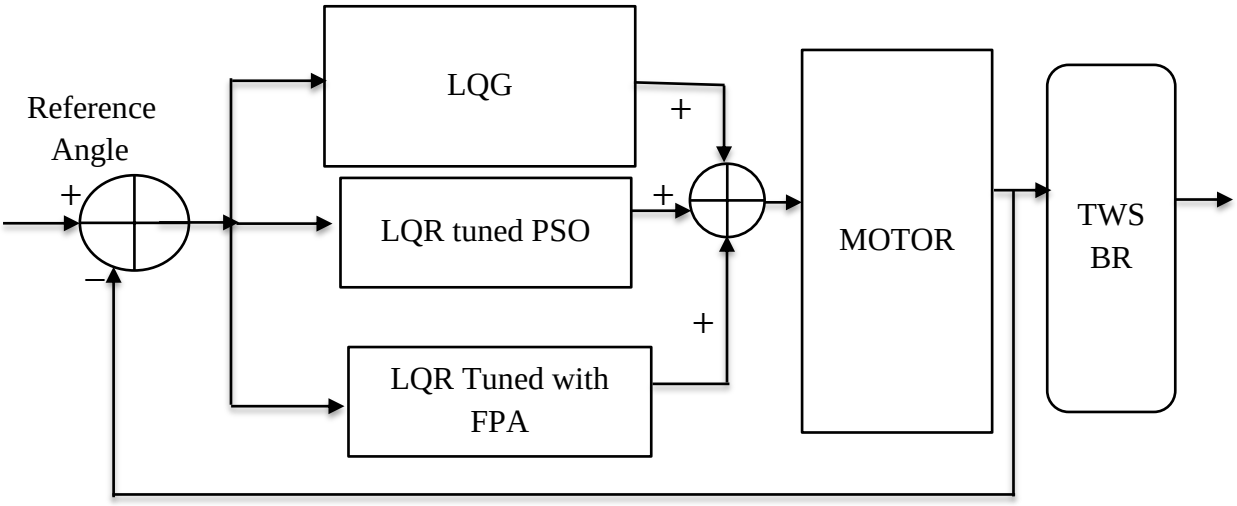


Figure 3.2 Block diagram of two wheeled self-balancing robot

The diagram of a two-wheeled self-balancing robot (differential drive robot) illustrates the basic components and the key kinematic parameters that govern its motion.

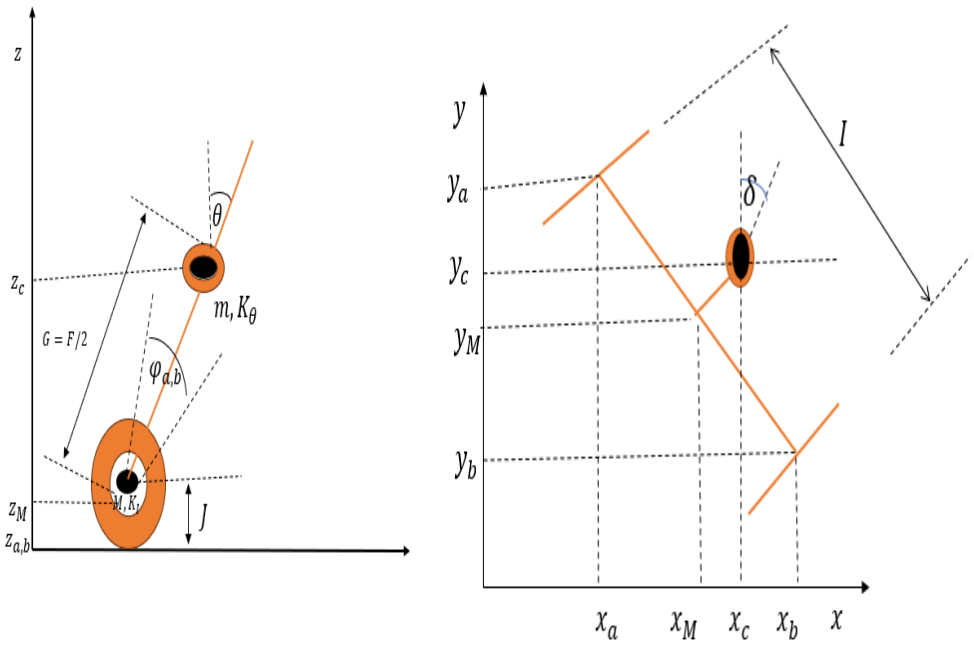


Figure 3.3For side and top view of self-balancing two-wheeled self-balancing robots respectively(Yamamoto, 2009)

With urban congestion and demand for last mile delivery solutions rising, there is much interest in, maneuverable, two-wheeled and differential drive robots capable of carrying

goods or assisting in personal mobility. Self-balancing robots can navigate crowded and uneven environments, making them ideal candidates for urban and indoor transport solutions. Traditional control approaches are limited in their ability to respond to these changes, often requiring manual tuning. Note that from the above figures are mean left and right wheel, in the same way are left and right wheel angles, and right in coordinate. When both wheels rotate at the same speed and in the same direction, the robot moves forward or backward in a straight line. When the wheels rotate at different speeds or one wheel stops while the other continues, the robot turns. If the left wheel rotates faster than the right wheel, the robot will turn to the right. If the right wheel rotates faster than the left wheel, it will turn to the left.

Table 3.1 Physical parameters of the self-balancing two-wheeled robot

No	Parameters	Description	Value
1	M	Wheel mass	0.05kg
2	m	Robot body mass	0.64kg
3	J	Radius of the wheel	0.027m
4	I	Part of robot's width	0.105m
5	L	Robot's depth	0.1m
6	G	Robot's height	0.21m
7	$H = F/2$	Distance of wheel axle and robot's center	0.105m
8	g	Acceleration due to gravity	$9.8\,{}^{m}/_{s^2}$
9	$K_I = MJ^2/2$	Moment Inertia of wheel	0.000016kgm^2
10	$K_\theta = mF^2/3$	Moment Inertia of robot's pitch	0.02352kgm^2
11	$K_\delta = m(I^2 + L^2)/12$	Moment Inertia of robot's yaw	0.0011211kgm^2
12	K_M	Moment Inertia of DC motor	$1 \times 10^{-5}\text{kgm}^2$
13	J_M	Resistance of DC motor	$6.695\ \Omega$
14	K_{emf}	EMF constant for DC motor	$0.468\ \text{V} \cdot \text{Sec/rad}$
15	K_t	Constant Torque for DC motor	$0.317\ \text{Nm/A}$
16	n	Gear ratio	1
17	f_m	Friction between robot and motor	0.0022

Mathematical modeling for top and side view of two wheeled self-balancing robot designed as below. The kinetic energies of a side and rare views of differential drive of the robot are P_1 and P_2, respectively.

$$P_1 = \frac{1}{2}M(\dot{x}_a^2 + \dot{y}_a^2 + \dot{z}_a^2) + \frac{1}{2}M\left(\dot{x}_b^2 + \dot{y}_b^2 + \dot{z}_b^2\right) + \frac{1}{2}M(\dot{x}_c^2 + \dot{y}_c^2 + \dot{z}_c^2) \quad (3.1)$$

$$P_2 = \frac{1}{2}K_I\left(\dot{\varphi}_b^2 + \dot{\varphi}_b^2\right) + \frac{1}{2}K\theta\dot{\theta}^2 + \frac{1}{2}K_\delta\dot{\delta}^2 + \frac{1}{2}n^2K_M\left(\dot{\varphi}_a - \dot{\theta}^2)^2 + \frac{1}{2}n^2K_M\left(\dot{\varphi}_b - \dot{\theta}^2)^2\right)\right) \quad (3.2)$$

Where other variables in equation (3.2) are as in Table (3.1)

The potential energy(U) of the top view differential drive of the robot

$$U = Mgz_a + Mgz_b + Mgz_c \quad (3.3)$$

Equation of motion for top views are given as

$$(x_M, y_M, z_M,) = (\int \dot{x}_M\, dt, \int \dot{y}_M dt, R), (\dot{x}_M, \dot{y}_M) = (J\dot{\varphi}cos\delta, J\dot{\varphi}sin\varphi) \quad (3.4)$$

$$(x_a, y_a, z_a) = (x_M - \frac{I}{2}sin\delta, y_M + \frac{I}{2}cos\delta, z_M) \quad (3.5)$$

$$(x_b, y_b, z_b) = (x_M + \frac{I}{2}sin\frac{\varphi_a + \varphi_b}{2}, y_M - \frac{I}{2}cos\delta, z_M \quad (3.6)$$

$$(x_c, y_c, z_c) = (x_M - Fsin\theta cos\delta, y_M + Fsin\theta sin\delta, z_M + Fsin\theta cos\delta \quad (3.7)$$

Angle for wheel and yaw for two wheeled self-balancing robots written in the form of

$$\left((\varphi, \delta) = \frac{\varphi_a + \varphi_b}{2}, \frac{J(\varphi_b - \varphi_a)}{I}\right) \quad (3.8)$$

By subsisting equation $(3.4 - 3.7)$ in to equation (3.1) and (3.2) then kinetic energy (P_1) becomes

$$P_1 = \frac{1}{2}M([(J\dot{\varphi}cos\delta - \frac{I}{2}\delta cos\delta cos\,\delta + \frac{1}{2}M(J\dot{\varphi}sin\delta - \frac{I}{2}sin\,\delta)^2 + \frac{1}{2}M(J\dot{\varphi}cos\delta + \frac{I}{2}\dot{\delta}cos\delta)^2 + \frac{1}{2}M(J\dot{\varphi}cos\delta + \frac{I}{2}\dot{\delta}cos\delta)^2 + \frac{1}{2}m(J\dot{\varphi}cos\delta - Fsin\theta cos\delta\dot{\delta} + Fsin\delta cos\theta\dot{\theta})^2 + (-Fsin\theta\dot{\theta})^2.$$

$$= \frac{1}{2}M\left[\left(2J^2\dot{\varphi}^2 + \frac{I^2}{2}\dot{\delta}^2\right)\right] + \left[\frac{1}{2}J^2\dot{\varphi}^2 + F^2sin^2\theta\dot{\theta}^2 + 2JF\dot{\varphi}cos\theta\dot{\theta} + F\dot{\theta}^2\right] \quad (3.8)$$

Then the relation between wheel and yaw angle are given as equation (3.9) and (3.10)

$$\varphi_b = \varphi + \frac{I\delta}{2J} \quad (3.9)$$

$$\varphi_a = \varphi - \frac{I\delta}{2J} \tag{3.10}$$

By subsisting in kinetic energy (P_2) become,

$$P_2 = \frac{1}{2}K_I(\dot{\varphi} - \frac{I\dot{\delta}}{2J})^2 + \frac{1}{2}K_{Iw}(\dot{\varphi} + \frac{I\dot{\delta}}{2J})^2 + \frac{1}{2}K_\theta\dot{\theta}^2 + \frac{1}{2}K_\delta\dot{\delta}^2 + \frac{1}{2}n^2K_M(\dot{\varphi} - \frac{I\dot{\theta}}{2J} - \dot{\theta})^2$$

$$+ \frac{1}{2}n^2K_M(\dot{\varphi} + \frac{I\dot{\theta}}{2J} - \dot{\theta})^2$$

$$P_2 = \frac{1}{2}K_I(2\dot{\varphi}^2 - \frac{I^2\dot{\delta}^2}{2J} + \frac{1}{2}J_\theta\dot{\theta}^2 + \frac{1}{2}K_\delta\dot{\delta}^2$$

$$+ \frac{1}{2}n^2K_M\left(2\dot{\varphi}^2 + 2\dot{\theta}^2 - 4\dot{\varphi}\dot{\theta})^2 + \frac{I^2\dot{\theta}^2}{2J^2}\right) \tag{3.11}$$

The potential energy (U) can be rewritten by subs.

$$U = 2MgJ + mg(J + F\cos\theta) \tag{3.12}$$

The lagrangian (L) is now defined as the following

$$L = P_1 + P_2 - U \tag{3.13}$$

$$\frac{d}{dx}\frac{\partial}{\partial\dot{\varphi}} - \frac{\partial L}{\partial\varphi} = F_\varphi \tag{3.14}$$

$$\frac{d}{dx}\frac{\partial}{\partial\dot{\theta}} - \frac{\partial L}{\partial\theta} = F_\theta \tag{3.15}$$

$$\frac{d}{dx}\frac{\partial}{\partial\dot{\delta}} - \frac{\partial L}{\partial\delta} = F_\delta \tag{3.16}$$

Deferential equations, which represent the two wheeled robot system

$$[(2M + m)J^2 + 2K_I + 2n^2K_M]\ddot{\varphi} + (mFJ\cos\theta - 2n^2K_M)\ddot{\theta}$$

$$- mFJ\dot{\theta}^2\sin\theta \tag{3.17}$$

$$[mFJ\cos\theta - 2n^2K_M]\ddot{\varphi} + (mF^2 + K_\theta + 2n^2K_M)\ddot{\theta} - mgF\sin\theta - mF^2\dot{\theta}^2\sin\theta\cos\theta$$

$$= F_\theta \tag{3.18}$$

Then transferring variables φ θ and δ in to standard state space model variables as follows

Where φ θ and δ are represented as state space model shown below

$$x_1 = \varphi, x_2 = \dot{\varphi} \text{ then } \dot{x_2} = \varphi$$

$$x_3 = \theta, x_4 = \dot{\theta} \text{ then } \dot{x_4} = \ddot{\theta} x_5 = \delta, x_6 = \dot{\delta} \text{ then } \dot{x_6} = \ddot{\delta}$$

We can adjust the equation for each state variable's

$$[(2M + m)J^2 + 2K_I + 2n^2K_M]\dot{x_2} + (mFJ\cos(x_3) - 2n^2K_M)\dot{x_4} - mFJ\dot{\theta}^2\sin(x_3)$$

$$= F_\varphi \tag{3.19}$$

$$[mFJcos(x_3) - 2n^2K_M]\dot{x_2} + (mF^2 + K_\theta + 2n^2K_M)\dot{x_4} - mgFsin(x_3) - mF^2\dot{\theta}^2\sin(x_3)\cos(x_3) = F_\theta \quad (3.20)$$

Firstly, the $\dot{x_1}$ equations can be defined as

$$\dot{x_1} = x_2 \quad (3.21a)$$

But $\dot{x_2}$

$$\dot{x_2} = x_4 \times \frac{x_4 \sin(x_3)\,(m^2JF^3 + 2mJFn^2K_M + mJFK_\theta)}{d + e(x_3)} + \frac{2n^2K_M mgsin(x_3) - m^2JF^2cos(x_3)gsin(x_3)}{d + e(x_3)} + F_\theta \times \frac{\left(2nK_M - mJFcos(x_3)\right)}{d + e(x_3)} + F_\varphi \times \frac{(mF + 2n^2K_M + K_\theta)}{d + e(x_3)} \quad (3.21b)$$

Let for the simplified form the values of d and e are given in the form of

$$d = 2K_IK_\theta + 2MJ^2mF^2 + 4MJ^2n^2K_M + 2mJ^2n^2K_M + 2n^2K_MmF^2 + 2MJ^2K_\theta + mJK + 2K_ImF^2 + 4K_In^2K_M + 2n^2K_MK_\theta \quad (3.22)$$

And

$$e = m^2J^2F^2sin(x_3) + 4mFJcos(x_3)n^2K_M \quad (3.23)$$

Then we have

$$h_{23} = 2n^2KmgFsin(x_3) - mJFcos(x_3)gsin(x_3),$$

$$h_{23} = x_4sin(x_3)(m^2JF^3 + 2mJFn^2K_M + mJFK_\theta)$$

$$z_{21} = mJ^2 + 2n^2K_M + K_\theta$$

And

$$z_{22} = 2n^2K_M - mJFcos(\theta)$$

This can be used to rewrite as

$$\dot{x_2} = \frac{h_{23}}{d + e} \times x_3 + \frac{h_{24}}{d + e} \times x_4 + \frac{z_{21}}{d + e} \times F_\varphi + \frac{z_{22}}{d + e} \times F_\theta \quad (3.24)$$

Also $\dot{x}_3 = x_4$ and then $\dot{x}_4$

$$\dot{x_4} = x_4 \times \frac{x_4 \sin(x_3) - (m^2J^2F^2\cos(x_3) + 2mJFn^2K_m}{a + b(x_3)} + \frac{mgF\sin(x_3)(2K_I + 2MJ^2 + mJ^2 + 2n^2K_M)}{d + e(x_3)} + F_\varphi \times \frac{(2n^2K_M - mJK\cos(x_3))}{d + e(x_3)} + F_\theta \times \frac{2n^2K_M + 2K_I + 2MJ^2 + mJ^2}{d + e(x_3)} \quad (3.25)$$

After then

$$h_{43} = mgJ\sin(x_3)(2n^2K_M + 2K_I + 2MJ^2 + mJ^2)$$

$$h_{44} = x_4 \sin(x_3) - (mJF\cos(x_3) + 2mJFn^2K_M)$$

$$z_{41} = 2n^2K_M - mJF\cos(\theta)$$

And

$$z_{42} = 2n^2K_M + 2K_M + 2mRJ^2 + mJ^2 \quad (3.26)$$

Which can rewrite in the form of?

$$\dot{x_4} = \frac{h_{43}}{d + e} \times x_3 + \frac{h_{44}}{d + e} \times x_4 + \frac{z_{41}}{d + e} \times F_\theta + \frac{z_{42}}{d + e} \times F_\theta \quad (3.27)$$

Therefore, the non-linear state space model of the self-balancing two wheeled robot controlled by two forces, the pitch angle force F_θ and F_φ the wheel angle force, can be represented as follow.

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & \frac{h_{23}}{d + e} & \frac{h_{24}}{d + e} \\ 0 & 0 & \frac{h_{43}}{d + e} & 0 \\ 0 & 0 & \frac{h_{44}}{d + e} & \frac{h_{44}}{d + e} \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{z_{21}}{d + e} & \frac{z_{22}}{d + e} \\ 0 & 0 \\ \frac{z_{41}}{d + e} & \frac{z_{42}}{d + e} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} F_\theta \\ F_\varphi \end{pmatrix} \quad (3.28)$$

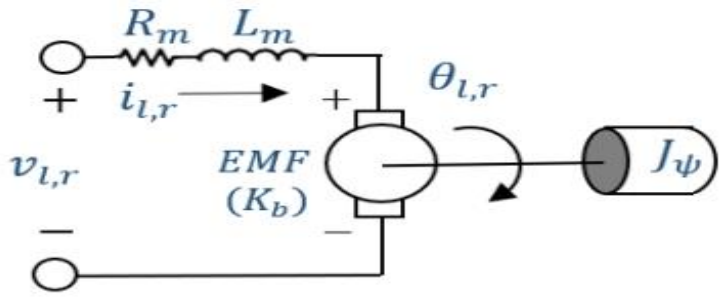


Figure 3.4 DC motor schematic

Kirchhoff's voltage law is applied to summaries the electric circuit in given the equation as follow let $v_{l,r} = v_{a,b}$ for left and right voltage current and $i_{l,r} = i_{a,b}$ for left and right motor current.

$$L\frac{d_{ia,b}}{d_t} = v_{a,b} + k_b\left(\frac{d_\theta - d_{\varphi\, a,b}}{d_t} - J_M i_{a,b}\right)$$

Where $i_{a,b}$ is the DC motor current?

In DC circuits the inductance of motor in the steady state operation behaves like a short circuit, therefore it can be approximated as zero and then can be rewritten as

$$i_{a,b} = \frac{v_{a,b} + k_b(\dot{\theta} - \dot{\varphi}_{a,b})}{J_M} \tag{3.28}$$

The generalized forces of the DC motor torque

$$(F_\varphi, F_\theta) = (F_L + F_r, F_\varphi) \tag{3.29}$$

$$F_a = nk_t i_a + z_M(\dot{\theta} - \dot{\varphi}_a) - z_I\dot{\varphi}_a \tag{3.30a}$$

$$F_b = nk_t i_b + z_M(\dot{\theta} - \dot{\varphi}_b) - z_w\dot{\varphi}_b \tag{3.30b}$$

$$F_\theta = nk_t i_a - nk_t i_a - z_M(\dot{\theta} - \dot{\varphi}_a) \tag{3.30c}$$

Therefore, the generalized forces

$$F_\varphi = \beta(V_a + V_b) - 2(\alpha + z_\varphi)\dot{\varphi} + 2\alpha\dot{\theta} \tag{3.31}$$

$$F_\theta = -\beta(V_a + V_b) + 2\alpha\dot{\varphi} - 2\alpha\dot{\theta} \tag{3.32}$$

Now, the state space representation of the two wheeled robots would be transferred from generalized force to voltages by sub.

$$\dot{x}_2 = x_2 \times \left(\frac{2\alpha z_{22}}{d+e}\right) - \frac{2(\alpha + z_\varphi)z_{21}}{d+e} + \frac{h_{23}}{d+e} + x_4 \times \frac{(h_{24})}{d+e} + \frac{2\alpha(z_{21} - z_{22})}{d+e} + V_a$$
$$\times \frac{\beta(z_{21} - z_{22}(x_3)}{d+e} + V_b$$
$$\times \frac{\beta(z_{21} - z_{22}}{d+e}) \qquad (3.33)$$
$$h_{M22} = 2\alpha z_{22} - 2(\alpha + z_I)z_{21}$$
$$h_{m24} = h_{M24} + 2\alpha(z_{21} - z_{22})$$
$$z_{M21} = \beta(z_{21} - z_{22})$$

And

$$z_{m22} = \beta(z_{21} - z_{42})$$

This can be written as

$$\dot{x}_2 = \left(\frac{h_{M22}}{d+e}\right) \times x_2 + \frac{h_{23}}{d+e} \times x_3 + x_4 \times \frac{(h_{m24})}{d+e} + \frac{z_{M21}}{d+e} \times v_a \times \frac{z_{M22}}{d+e}$$
$$\times v_b \qquad (3.34)$$

In the same way $\dot{x_4}$

$$\dot{x}_4 = x_2 \times \left(\frac{2\alpha h_{42}}{d+e}\right) - \frac{2(\alpha + h_\omega)h_{41}}{d+e} + \frac{z_{43}}{d+e} + x_4 \times \frac{z_{44}}{d+e} + \frac{2\alpha(h_{41} - h_{42})}{d+e} + V_a$$
$$\times \frac{\beta(h_{41} - h_{42}}{d+e} + V_b$$
$$\times \frac{\beta(f_{41} - f_{42})}{d+e} \qquad (3.35)$$
$$z_{M42} = 2\beta f_{42} - 2 + h_I)h_{41}$$
$$z_{M44} = z_{m44} + 2\beta(h_{41} - h_{41})$$
$$z_{M41} = \alpha(h_{41} - h_{42})$$

And

$$h_{M42} = \alpha(h_{21} - h_{42})$$

This can be written as

$$\dot{x}_4 = \left(\frac{z_{M42}}{d+e}\right) \times x_2 + \frac{z_{43}}{d+e} \times x_3 + x_4 \times \frac{(z_{m44})}{d+e} + \frac{h_{m41}}{d+e} \times v_a + \frac{h_{m41}(x_3))}{d+e}$$
$$\times v_b \qquad (3.36)$$

After the derivations presented, the nonlinear state-space model describing the two-wheeled robot with voltage as its control input is summarized as follows.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{z_{22}}{d+e} & \frac{z_{23}}{d+e} & \frac{z_{24}}{d+e} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{z_{42}}{d+e} & \frac{z_{43}}{d+e} & \frac{z_{44}}{d+e} \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{h_{m21}}{d+e} & \frac{h_{m22}}{d+e} \\ 0 & 0 \\ \frac{h_{m41}}{d+e} & \frac{h_{m42}}{d+e} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v_a \\ v_b \end{pmatrix} \quad (3.37)$$

To address this, we are introducing a tracking system that incorporates the integral of the wheel angle as a new state variable. This integration action is specifically designed to remove persistent deviations in the wheel angle, pitch angle and yaw angle; thereby ensuring the robot stabilizes at the reference position and follows a pre-defined trajectory. The resulting nonlinear state-space model will describe the two-wheeled robot with this added tracking functionality.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{z_{22}}{d+e} & \frac{z_{23}}{d+e} & \frac{z_{24}}{d+e} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & \frac{z_{42}}{d+e} & \frac{z_{43}}{d+e} & \frac{z_{44}}{d+e} & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{h_{m21}}{d+e} & \frac{h_{m22}}{d+e} \\ 0 & 0 \\ \frac{h_{m41}}{d+e} & \frac{h_{m42}}{d+e} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v_a \\ v_b \end{pmatrix} \quad (3.38)$$

3.4. Linearization of the two-Wheeled robot model

Although the nonlinear state-space model of the two-wheeled robot presented earlier produced sophisticated dynamic equations, linearization simplifies them. Employing Taylor series expansions and viewing signals as small perturbations (Ogata and Yang, 1970), this approach estimates a nonlinear system as linear around an equilibrium point. The linear model that results is ideal for linear control strategy implementation that is, LQR and LQG control. Under the hypothesis that θ and $\dot{\theta}$, the nonlinear model of the self-balancing two-

wheeled robot from Equation (3.36) will be linearized in this section, hence yielding the following approximations.

$$\sin(\theta) \approx \theta, \cos(\theta) \approx 1, x_4 \sin(\theta) \approx 0 \text{ and } \sin(\theta)^2 \approx 0 \quad (3.38)$$

Some predefined parameters of two wheeled self-balancing robots are

$$h_{22} = 2n^2 K_M - mJG$$

$$h_{M21} = \beta(h_{21} - h_{21})$$

$$h_{M22} = \beta(h_{21} - h_{22L})$$

$$z_{23} = x_3 \times (2n^2 K_M Gmg - m^2 JG^2)$$

$$= x_3 \times z_{23}$$

$$z_{24} = 0$$

$$z_{M22} = 2\alpha h_{22} - 2(\alpha + h_I)h_{21}$$

$$z_{M42} = 2\alpha(h_{21} - h_{22})$$

$$\text{let } z_{23} = (2n^2 K_M Gmg - m^2 JG^2)$$

$$h_{41} = 2n^2 R_M - mJG$$

$$h_{M41} = \alpha(h_{41} - h_{42})$$

$$h_{M42} = \alpha(h_{41} - h_{42})$$

$$z_{43} = x_3 \times mgG(2n^2 K_M + 2K_I + 2MJ^2 + mJ^2)$$

$$= x_3 \times z_{43}$$

$$z_{44} = 0$$

$$z_{M42} = 2\alpha h_{42} - 2(\alpha + h_I)h_{41}$$

$$z_{M44} = 2\alpha(h_{41} - h_{42})$$

Let

$$z_{43} = mgG(2n^2 K_M + 2K_I + 2MJ^2 + MJ^2)$$

Then equation (3.23) is written as

$$d = 4mGJn^2 K_M$$

$$\dot{x}_2 = \frac{z_{M22}}{d+e} \times x_2 + \frac{x_3 \times z_{M_{23}}}{d+e} \times x_3 + \frac{z_{M24}}{d+e} \times x_4 + \frac{h_{M21}}{d+e} \times v_a + \frac{h_{M22}}{d+e} \times v_b \quad (3.39)$$

In the same way $\dot{x}_4$which transform equation

$$\dot{x}_4 = \frac{z_{M42L}}{d+e} \times x_2 + \frac{z_{M43}}{d+e} \times x_3 + \frac{z_{M44}}{d+e} \times x_4 + \frac{h_{m41}}{d+e} \times v_a + \frac{h_{m42}}{d+e} \times v_b \quad (3.40)$$

Therefore, the linear state space model of self-balancing two wheeled robot is representing as the following bellow:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & & \\ 0 & \frac{z_{M22}}{d+e} & \frac{z_{M23}}{d+e} & \frac{z_{M24}}{d+e} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & \frac{z_{M42L}}{d+e} & \frac{z_{M43}}{d+e} & \frac{z_{M44}}{d+e} & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{h_{M21}}{d+e} & \frac{h_{M22}}{d+e} \\ 0 & 0 \\ \frac{h_{m41}}{d+e} & \frac{h_{m42}}{d+e} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v_a \\ v_b \end{pmatrix} \quad (3.41)$$

After subsisting

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & -227.11 & 492.22 & 228.11 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 48.58 & 163.79 & -48.58 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 220.51 & 220.51 \\ 0 & 0 \\ -46.22 & -46.22 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v_a \\ v_b \end{pmatrix}$$

CHAPTER FOUR

CONTROLLER DESIGN

The focus of this chapter is a modern linear control strategy known as the Linear Quadratic Regulator (LQR), a technique widely applied in the control of linear systems. We will also explore its common extension, the Linear Quadratic Gaussian (LQG). LQR and LQG are optimal control methods prevalent in control system design, and this section outlines their design principles. For the controller design in this research, the system model was first analyzed for controllability and observability. Notably, the Kalman Filter and LQR parameters are designed separately. The following subsection will provide a detailed explanation of the Linear Quadratic Regulator (LQR). The Linear Quadratic Regulator (LQR) is an optimal control method that employs state feedback to drive a linear system to a desired state while minimizing a defined quadratic cost function.

$$\mathrm{J}(\dot{\mathrm{x}},\mathrm{u}) = \min_{\mathrm{u}}\left(\int_0^{\infty} (\dot{\mathrm{x}}^{\mathrm{T}}\mathrm{Q}\dot{\mathrm{x}} + \mathrm{u}^{\mathrm{T}}\mathrm{Ru})\mathrm{dt}\right) \quad (4.1)$$

$$\mathrm{J} = \int_0^{\infty} (\mathrm{x}^{\mathrm{T}}\mathrm{Qx} + (\mathrm{K}\dot{\mathrm{x}})^{\mathrm{T}}\mathrm{R}(\mathrm{K}\dot{\mathrm{x}})\,\mathrm{dt} \quad (4.2)$$

The goal is to find a control input $u = -Kx$ that minimizes the quadratic cost function:

$$\mathrm{J} = \int (\mathrm{x}^{\mathrm{T}}\mathrm{Qx} + \mathrm{u}^{\mathrm{T}}\mathrm{Rx})\mathrm{dt} \quad (4.3)$$

Our objective is to minimize the performance index function concerning feedback gain (K)

$$\dot{x}^T(Q + K^T RK)\dot{x} = -\frac{d}{dt}(x^T Px) = -\dot{x}^T Px - x^T P\dot{x} \quad (4.4)$$

Then performance index is can be evaluated as below

$$J = \int_0^{\infty} (\dot{x}^T(Q + K^T RK)\dot{x}\, dt = -x^T Px_0^{\infty}$$

(4.5)

$$= -x^T(\infty)Px(\infty) + x^T(0) + Px(0) \quad (4.6)$$

$$J = x^T(0) + Px(0) \quad (4.7)$$

$$x = A^{-1}\dot{x} > A^{-1} = A^{-1}(I_n + B_k) \quad (4.8)$$

$$P(A^{-1} + A^{-1}BK) + A^{-T} + K^T B^T A^{-T})P + K^T RK + Q = 0 \quad ` \quad (4.9)$$

Where T is non-singular matrix

$$x = A^{-1}\dot{x}, A^{-1} = A^{-1}(I_n + BK) \quad (4.10)$$

$$\dot{x}^T(Q + K^T RK)\dot{x} = -\dot{x}^T(PA^{-1} + A^{-T}P)\dot{x} \quad (4.11)$$

By comparing both sides

$$A^{-1} = A^{-1} + A^{-1}BK, and A^{-T} = A^{-T} + K^T B^T A^{-T} \quad (4.12)$$

$$P(A^{-1} + A^{-1}BK) + (A^{-T} + K^T B^T A^{-T})P + K^T RK + Q = 0 \quad (4.13)$$

$$PA^{-1} + PA^{-T}BK + PA^{-T} + K^T B^T A^{-T} P + K^T RK + Q = 0 \quad (4.14)$$

The minimization of J requires the minimization of

$$\dot{x}^T (TK + T^{-T} B^T A^{-T} (TK + T^{-T} B^T A^{-T} P)\dot{x} \quad (4.15)$$

With respect to K, since the last expression is no negative, the minimum occurs when its zero.

$$TK = -T^{-T} B^T A^{-T} P \quad (4.16)$$

The optimal matrix gain K is

$$K = T^{-T} B^T A^{-T} P = -R^{-1} B^T A^{-T} P \quad (4.17)$$

Finally, the optimal stabilizing control law is given by

$$u(t) = -Kx(t) = R^{-1} B^T A^{-T} P x(t) \quad (4.18)$$

The matrix P satisfies the following algebraic Riccati equation (ARE)

The goal is to find a control input $u = -Kx$ that minimizes the quadratic cost function:

$$J = \int (x^T Q x + u^T R x) dt \quad (4.19)$$

In this approach, **Q** is defined as a $n \times n$ symmetric matrix that is either positive-definite or positive semi-definite and serves to weight the state derivatives. Similarly, R is a $m \times m$ symmetric positive-definite matrix used to weight the control effort? It's important to note that this formulation differs from the standard LQR because its performance index is based on the derivatives of the states rather than the states themselves.

$$0 = PA + A^T P - PBR^{-1} P + Q \quad (4.20)$$

Therefore, the optimal control is implemented by substituting Eq. (4.3) and (4.4) into Eq. (4.2):

$$\dot{x} = Ax - B(R^{-1} B^T P x) \quad (4.21)$$

$$= (A - BR^{-1} B^T P) x \quad (4.22)$$

$$y = Cx + Du \quad (4.23)$$

By selecting the Q and R weight matrices based on a desire to minimize wheel angle (φ) and pitch angle (θ) of the robot. In this approach, Q is defined as a n × n symmetric matrix that is either positive-definite or positive semi-definite and serves to weight the state derivatives. Similarly, R is a m × m symmetric positive-definite matrix used to weight the control effort? It is important to note that this formulation differs from the standard LQR because its performance index is based on the derivatives of the states rather than the states themselves.

$$0 = \mathrm{PA} + \mathrm{A^T P} - \mathrm{PBR^{-1} P} + \mathrm{Q} \quad (4.24)$$

Therefore, the optimal control is implemented by substituting Eq. (4.3) and (4.4) into Eq. (4.2).

$$\dot{x} = (A - BR^{-1}B^{T}P)x \quad (4.25)$$

$$y = Cx + Du \quad (4.26)$$

By selecting the Q and R weight matrices based on a desire to minimize wheel angle (φ) and pitch angle (θ) of the robot.

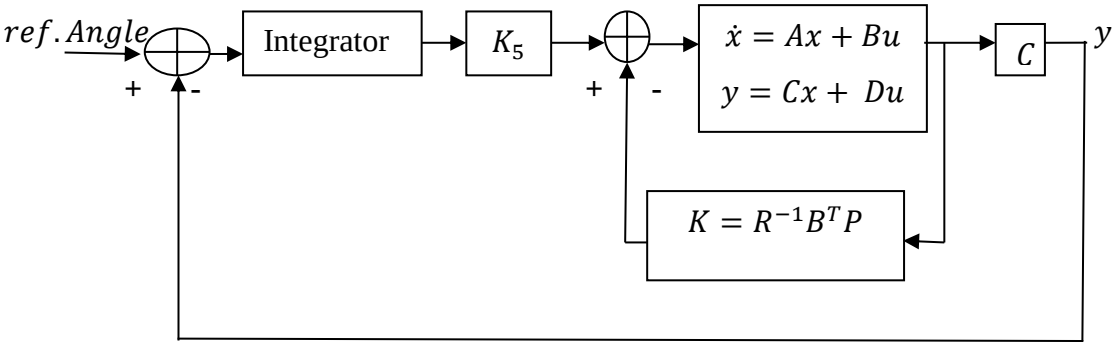


Figure 4.1 Structure of the linear quadratic regulator (LQR) and tracking system

LQR Gain Matrix K: Taken from MATLAB**.**

-19.4761 1.1443 -10.9675 -4.6877 -15.8114

-19.4761 1.1443 -10.9675 -4.6877 -15.8114

4.1. Linear Quadratic Gaussian (LQG)

Drift in the gyro sensor's measurement of the pitch angle. This inaccuracy propagated to the wheel angle displacement, ultimately leading to system instability. To mitigate the impact of such unreliable sensor data on our optimal feedback control, we will now explore the Linear Quadratic Gaussian (LQG) technique, which integrates an LQR controller with a Kalman filter.

4.1.1. Kalman Filter Testing

According to the benefits of the Kalman filter explained above, noise filtering and signal drift reduction will be implemented to the two-wheeled robot model.

This research will also be focused on the continuous system, corresponding to the LQR; hence consider general continuous time-invariant systems given by:

$$\dot{x} = Ax + Bu + w \quad (4.27)$$

$$y = Cx + v \quad (4.28)$$

The vector of measured outputs y is obtained through the output matrix C, with G being an identity matrix. The system's dynamics are affected by process noise (w), and the measurements are corrupted by measurement noise (v), both of which are presented visually

$$\dot{\hat{x}} = A\hat{x} + BuK_f(y - C\hat{x}) \quad (4.29)$$

This can be rewritten as:

$$\dot{\hat{x}} = (A - K_f C)\hat{x} + Bu + K_f y \quad (4.30)$$

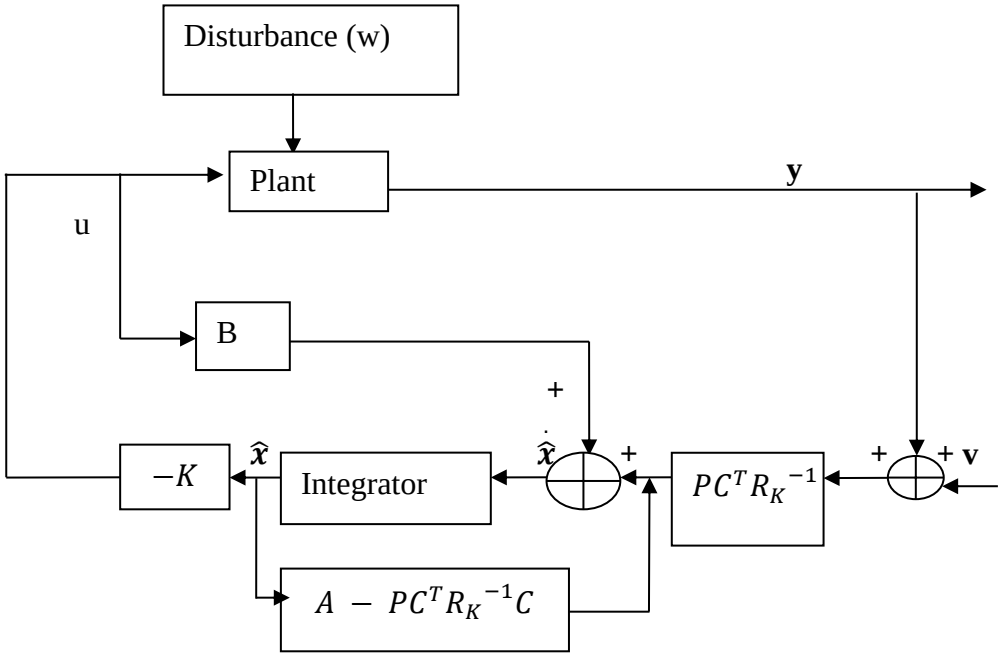


Figure 4.2 Schematic of state-space control using Kalman filter and LQR

$$K_f = PC^T R_K{}^{-1} \tag{4.31}$$

Where P is the solution of the algebraic riccati equation defined as follows

$$\mathrm{AP} + \mathrm{PA}^{\mathrm{T}} - \mathrm{PC}^{\mathrm{T}}\mathrm{R_k}^{-1}\,\mathrm{CP} + \mathrm{Q_k} = 0 \tag{4.32}$$

There are two noise disturbances considered in the LQG control, including the process white noise $\mathrm{w_n} \sim (0, \mathrm{Q_k})$ and the measurement white noise $\mathrm{v_n} \sim (0, \mathrm{R_k})$.

4.2. Controllability

The controllability of a dynamic system can be examined by a controllability test, which is applied to the matrices A and B. The model matrices are used to form the controllability matrix C as shown below.

$$\mathcal{C} = [B\ AB\ \ldots\ A^{n-1}B] \tag{4.33}$$

Therefore, for the 5^{th} order of two wheeled self-balancing robot on a cart system, the controllability test matrix is

$$\mathcal{C} = [B\ AB\ A^2B\ A^3B\ A^4B] \tag{4.34}$$

A system model is deemed completely state controllable if the rank of the controllability matrix $\mathcal{C}$ is equal to the number of rows in matrix A, meaning C, has full row rank.

4.3. Observability

The test combines matrices A and $\mathcal{C}$ the matrices are used to form the observability matrix $\mathcal{O}$, as shown below. Since the rank of the observability matrix is found to be equal to the number of columns in matrix O, the system satisfies the condition for complete state observability.

The matrices Q and R serve as weighting factors for the states and the control signal, respectively. To achieve the desired transient response, these matrices can be selected iteratively using a trial-and-error approach. Similarly, (Yamamoto, 2009) described an experimental trial-and-error method for choosing suitable Q and R matrices for balancing a Two-wheeled robot (TWR). Following this, the selected matrices were used in a stabilization simulation with a specific initial pitch angle (θ).

4.4. Optimizations

4.4.1. Flower pollination optimization algorithm

Pollination is the natural process by which flowering plants reproduce. It involves the transfer of pollen from the anther (pollen-bearing part) of a flower to the stigma (receptive part) of the pistil, either within the same flower, between different flowers on the same plant, or to another flower of the same species. Pollen grains contain both vegetative and reproductive cells. Once pollen lands on the stigma, the vegetative cell develops a pollen tube. The reproductive cell then travels down this tube, potentially dividing into two cells along the way, until it reaches the ovary. There, one of these reproductive cells fertilizes an egg cell, resulting in the formation of a zygote, which through growth, develops into a new plant.

In the FPA algorithm four rules are followed live pollination and cross-pollination are categorized as global pollination, where pollen carriers move according to a Lévy flight pattern. In contrast, biotic pollination and self-pollination are considered local pollination. Notably, pollinators like insects can exhibit flower constancy, which is the probability of a pollinator visiting a flower based on its similarity to previously visited flowers.

$$x_i{}^{t+1} = x_i{}^{t} + \varepsilon(x_j{}^{t} - x_k{}^{t}) \qquad (4.35)$$

Parameterization of Q and R of tuning every element of Q and R, which can lead to a very high-dimensional optimization problem, it is common to parameterize them. Q and R are typically chosen as diagonal matrices, where the diagonal elements represent the weights for individual states and control inputs. Sometimes, a base Q and R, are chosen, and then a few scaling factors are optimized. You might define specific non-diagonal structures based on system insights. Define the Objective Function (Fitness Function): This is the most crucial part. The objective function quantifies good performance. For stability and the specific parameters, you mentioned, it would involve metrics obtained from simulating the closed-loop system with a given K (derived from Q and R.

Table 4.1 Flower pollination algorithms (Xin-She Yang, 2012)

No	Parameter	Description	Value
1	N	Population size	40
2	f_{max}	Maximum frequency	1
	f_{min}	Minimum frequency	0
3	A_0	Maximum loudness	1
	α_0	Minimum loudness	0.95
4	r_0	Pulse emission rate	1
	γ_0	Gamma initial	0.1
	Δ	Scaling	0.5
5	t_{max}	Number of iteration maximum/minimum	500

4.4. Tuning LQR / LQG using FPOA

$$J_{FPO} = w_1 T_s + w_2 M_p + w_{3ess} + w_4 \int_0^T |e|\, d_t + w_5 \int_0^T u(t)\, d_t \qquad (4.36)$$

q_{11}Control tilts stability,q_{22} Control tilts angular velocity,q_{33}Control pitch stability, $q_{44}$and q_{55} wheel stability; how to optimize these parameters?

LQR Gain Matrix K (FPA Tuned): Taken from MATLAB.

-0.8773 0.2693 -2.2784 -1.0341 -0.2876

-1.7718 0.5439 -4.6014 -2.0884 -0.5808

The LQR controller inherently guarantees stability for controllable systems. However, in LQG, robustness can be an issue. The objective function can include penalties if the closed-loop system becomes unstable. This refers to the system's ability to return to a stable equilibrium after a disturbance in tilt. In the Q matrix, you would place a higher weight on the state variable representing tilt angle or tilt deviation from desired. This relates to how quickly the tilt changes. In Q, you'd place a higher weight on the state variable representing tilt angular velocity to suppress large values and oscillations. Each row corresponds to a control input, and each column corresponds to a state. The values represent the feedback gains. The FPOA's role was to find the optimal Q and R matrices that, when used in the LQR calculation, yielded this specific K matrix, which in turn optimizes your desired stability and performance metrics. To optimize these parameters means to iterate with the FPOA, running simulations and evaluating the objective function, until the performance (as defined by your stability and velocity metrics) is minimized or reaches an acceptable level. The output of the FPOA would be the optimal Q and R matrices that produced this K (or a similar one, if you're

still in the tuning process). Remember to define the ranges for your q_i and r_i parameters, as FPOA operates within a search space.

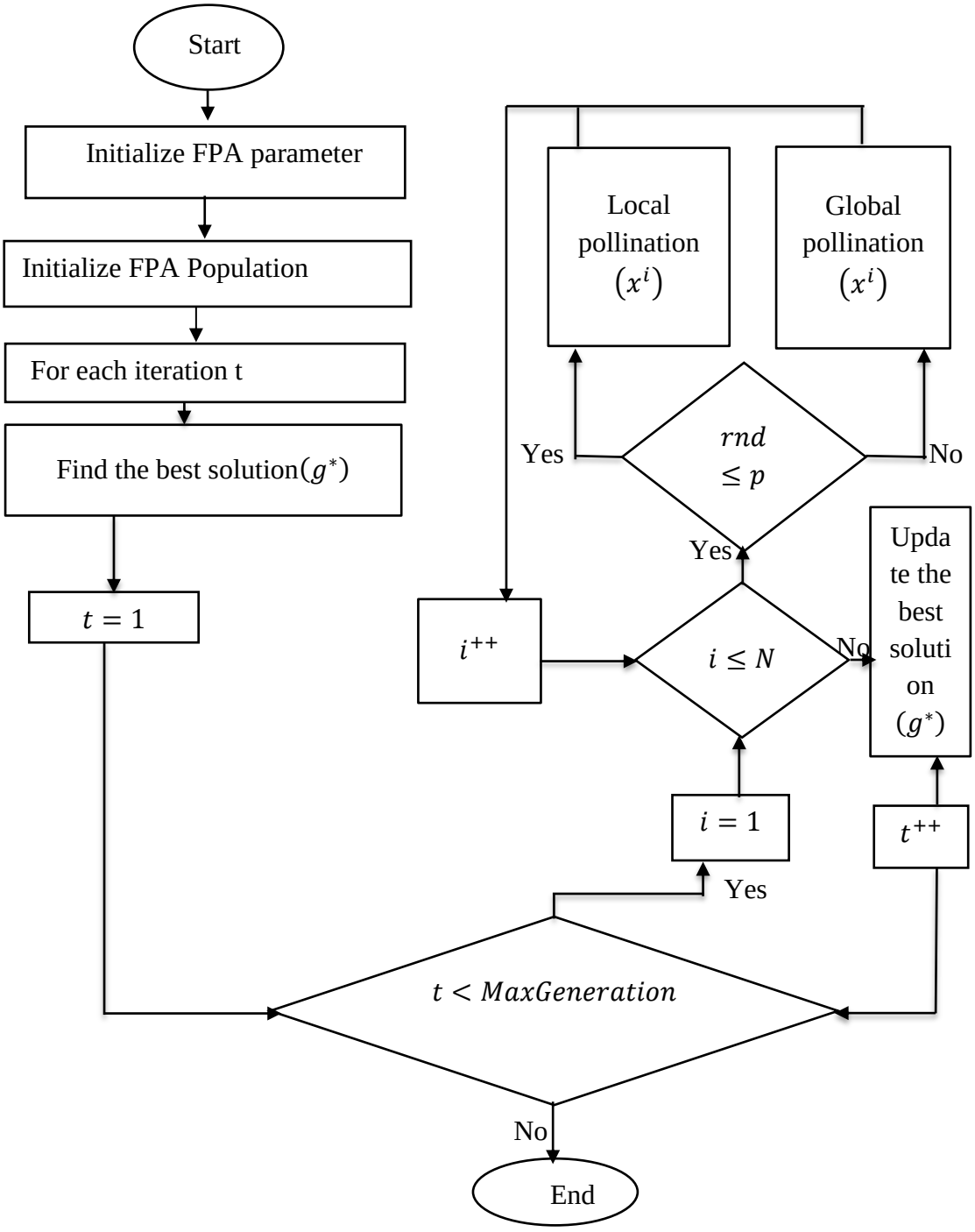


Figure 4.3Flow chart for flower pollination optimization algorithm

4.5. Tuning LQR and LQG using PSO

4.5.1. Particle swarm optimization algorism

PSO is a population-based search technique. It doesn't require auxiliary knowledge (derivative) of the problem. In the PSO algorithm, solutions belonging to the same population interact during the search process. The quality of the solutions is improved using techniques inspired by real-world phenomenon that uses the complex social cooperative and competitive behavior exhibited by different species like birds, bees, humans, etc.

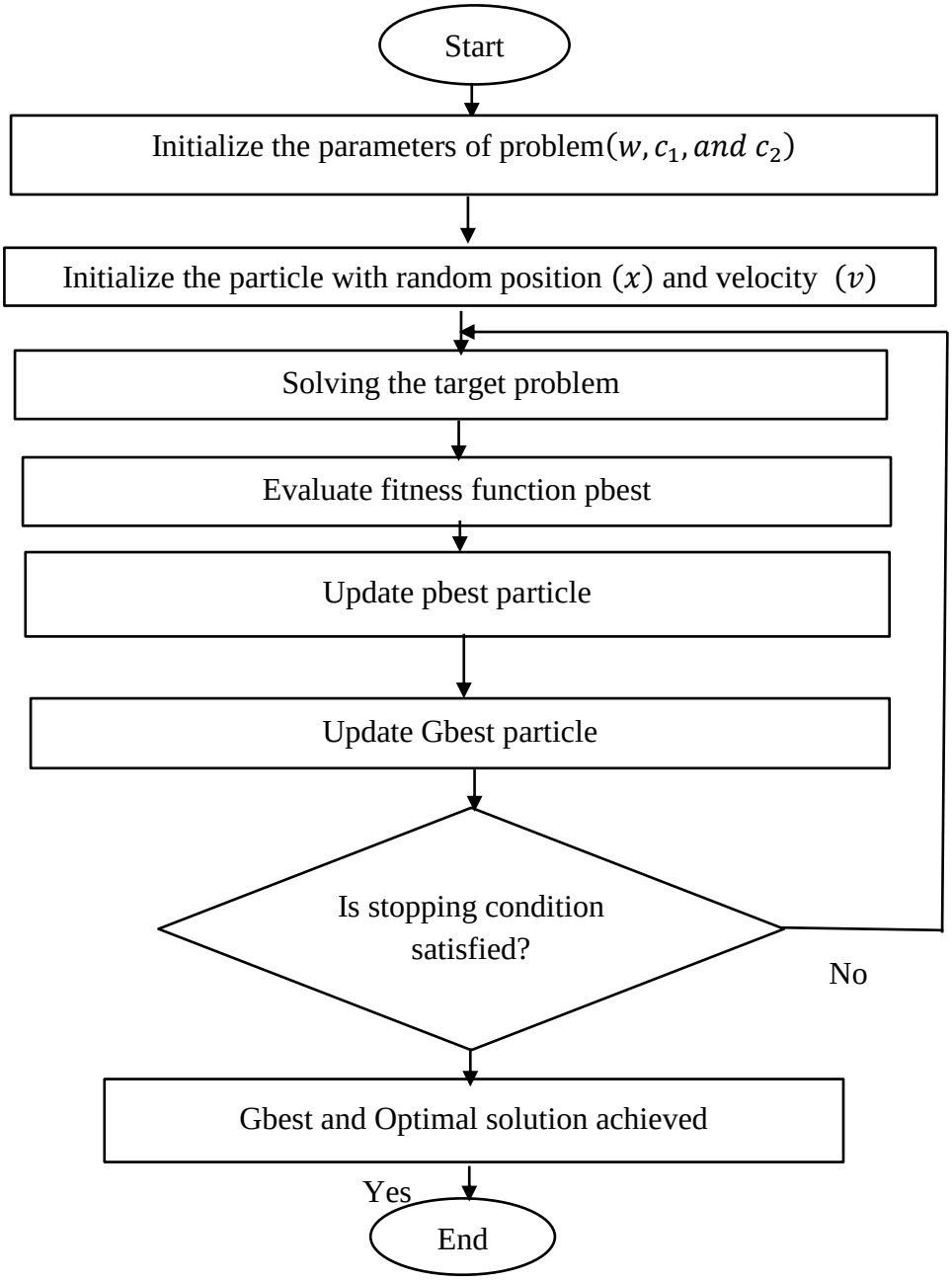


Figure 4.4Flow chart for Particle Swarm Optimization

Particle swarm optimization of L particles at 100^{th} iteration position population X and velocity V.Each particle has a position of x^i that is treated as a candidate solution and corresponds velocity v^i as where $t, n, c_1, c_2, w, r_1, r_2, pi, gi, N, w_{min}, w_{max}, x_i$ and v are iterations number, individual number, cognitive learning element, social learning element, social learning element, inertia weight, random values inside the range (0,1), the best position of the weight's maximum value, particle position, and the particle velocity respectively?

Table 4.2 Particle swarm optimization algorism

Description	Symbol	Values
Maximum bounds of inertia weights	w_{max}	0.7
minimum bounds of inertia weights	w_{min}	0.4
Cognitive learning element	c_1	2
Social learning element	c_2	2
Velocity maximum bound	v_{max}	6
Population size	n	20
Maximum bound of the repetition	N	300
Upper bound	u_l	[0.1 0.1 0.1 0.1 0.1 0.1]
Lower bound	u_u	[100,100,100,100,100]

4.5.2. Tuning LQR using PSO

$$v_i^{(t+1)} = wv_i^{(t)} + c_1 r_1 (pBest - x_i^{(t)} + c_2 r_2 (gBest - x_i^{(t)} \quad (4.37)$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \quad (4.38)$$

$$J_{PSO} = w_1 T_s + M_p + w_{3ess} + w_4 O_e + w_5 u \quad (4.39)$$

Where w_1, w_2, w_3, w_4, w_5weighting matrix.

Each particle in PSO represents a candidate diagonal Q and R scalar diagonal matrix

Particle $\{q_{11}, q_{22}, q_{33}, q_{44}, q_{55}\}$ and $\{r_{11}, r_{22},\}$

PSO algorithms for LQR tuning each particle in the PSO algorithm represent a candidate solution for Q and R Values.

q_{11}Control tilt stability,q_{22} Control tilt angular velocity,q_{33} Control pitch stability, q_{44} and q_{55} Control yaws stability.

LQR Gain Matrix K (PSO Tuned):

-2.9877 0.5457 -5.0168 -2.7110 -1.2304

-1.1344 0.2072 -1.9049 -1.0294 -0.4672

4.5.3. Tuning LQG using PSO

$$v_i^{(t+1)} = wv_i^{(t)} + c_1 r_1(pBest - x_i^{(t)} + c_2 r_2(gBest - x_i^{(t)}) \quad (4.40)$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \quad (4.41)$$

$$J_{PSO} = w_1 T_s + w_2 M_p + w_{3ess} + w_4 \int_0^T |e|\, d_t + w_5 \int_0^T u(t)\, d_t \quad (4.42)$$

q_{11} Control tilt stability,q_{22} Control tilts angular velocity,q_{33} Control pitch stability, q_{44} and q_{55} wheel stability,

$$Q = diag[10\ ,1, 1, 1{,}5]\ and\ R = diag[10{,}10] \quad (4.43)$$

The value of K that taken from MATLAB

LQR Gain Matrix K (PSO Tuned)

-2.9877 0.5457 -5.0168 -2.7110 -1.2304

-1.1344 0.2072 -1.9049 -1.0294 -0.4672

The LQR Gain Matrix K contains the feedback gains that multiply the robot's state variables to produce the control inputs that stabilize the robot. Because this matrix was PSO Tuned, it means an optimization algorithm was used to find the best possible set of gains to achieve the desired stability goals, likely focusing on minimizing control effort (like wheel speed) while maintaining good stability. The two rows in your K matrix suggest that there are two control inputs being generated by the controller, likely for the left and right wheels, or perhaps for direct and differential wheel commands to control both forward motion and turning, alongside balancing.

CHAPTER FIVE

RESULT AND ANALYSIS

5.1. Stability Control for two wheeled self-balancing robots with LQR.

Reference pitch of the system is aiming to control or track a pitch angle with a target of 15 degrees. This is the desired set point for the pitch. Peak tracking error (Optimal LQR) the maximum deviation from the reference pitch is 1.309 degrees. This indicates how well the system follows the desired 15degree pitch. A lower peak tracking error generally signifies better accuracy.

Settling time (Pitch angle, 5% ref, optimal LQR) the pitch angle settles within 5% of its reference value in 2.45 seconds. This is a measure of the system's responsiveness and stability. A shorter settling time means the system reaches and stays near its desired pitch more quickly.

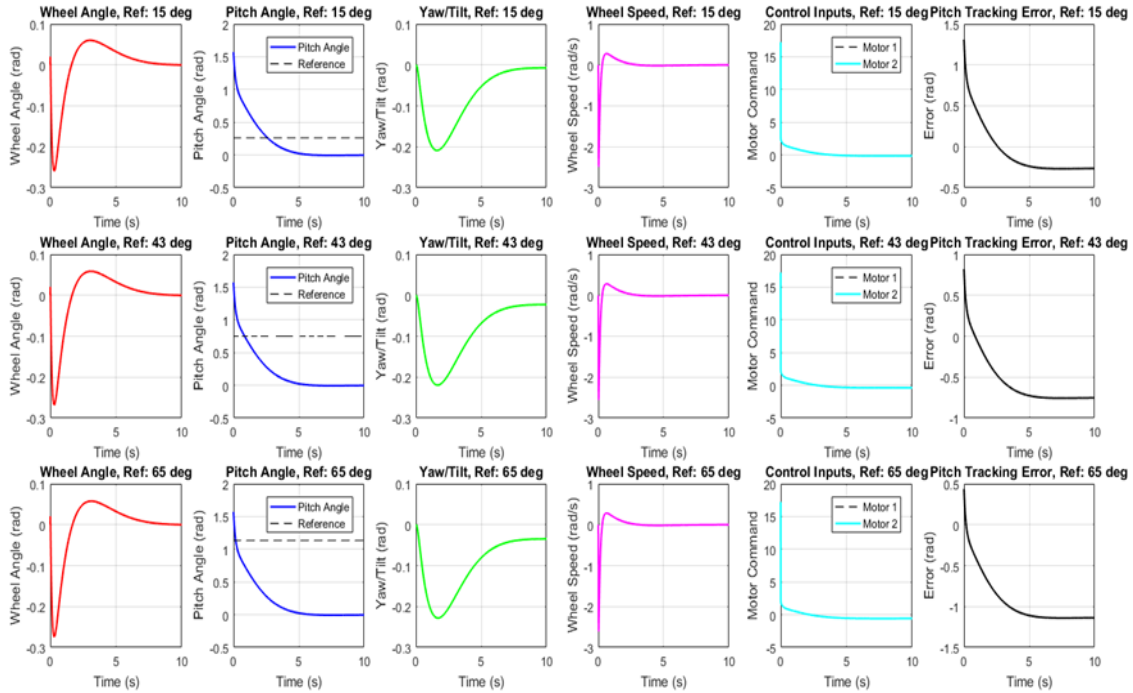


Figure 5.1 Stability Control for two wheeled self-balancing robots with LQR

Figure 5.1 Indicate Stability control for two wheeled self-balancing robots to show how peak pitch, peak wheel speed and peak tracking error gives response to the system at different reference angle 15°, 43° and 65°.

Peak wheel speed (optimal LQR) the maximum speed achieved by the wheel (presumably an actuator related to the pitch control) is 0.80075degree/s. This provides insight into the control

effort required by the system. Settling time (Wheel Speed, 5% peak, optimal LQR) the wheel speed settles within 5% of its peak value in 10 seconds. This settling time is significantly longer than the pitch angle settling time, suggesting that the wheel speed might be a slower-responding or more heavily damped part of the system, or perhaps it's less critical for the primary pitch control objective. Overall, these metrics suggest that the Optimal LQR controller is performing reasonably well for the pitch angle, achieving a relatively quick settling time (2.45 s) with a manageable peak tracking error (1.309 degree). The wheel speed's settling time is notably longer, which might be an area for further investigation if faster wheel response is desired or critical to overall system performance.

Reference pitch angle is 43 degree is a higher reference than the previous 15 degrees. Peak tracking error (LQR) which means the maximum deviation from the reference pitch is 0.8203 degrees. This is lower than the previous 1.309 degrees. Settling time (Pitch Angle, 5% ref, LQR) for the pitch angle settles within 5% of its reference in 0.95 seconds. This is significantly faster than the previous 2.45 seconds, indicating quicker response and better stability for the pitch angle. Peak wheel speed (LQR) the maximum wheel speed is 2.5584 degree/s. This is higher than the previous 0.80075 degree/s, suggesting that the controller is requiring more actuation effort (faster wheel movement) to achieve the higher reference pitch and faster settling time. Settling Time (Wheel speed, 5% peak, LQR) the wheel speed settles within 5% of its peak in 10 seconds. This remains the same as the previous scenario, which is interesting. It suggests that this aspect of the system's dynamics (or control objective for wheel speed) is less sensitive to the change in reference pitch or that the LQR design prioritizes pitch control over faster wheel speed settling if a trade-off exists.

Reference Pitch the desired pitch angle is 65 degrees. This is the target the LQR controller is trying to achieve and maintain. Peak Tracking Error The maximum deviation from the 65degree reference pitch is 1.1405 degrees**.** Peak wheel speed the maximum rotational speed observed for the wheel is 2.6112 degrees/second**.** This metric is important for understanding the energy consumption and actuator limits. Settling time (Pitch Angle, 5% ref**)** is the time it takes for the pitch angle to settle within 5% of the 65degree reference (i.e., within ±3.25 degrees of 65 degrees) is 0.21 seconds. This is a very fast settling time, suggesting a highly responsive pitch control system wheel speed performance (LQR).

LQR Gain Matrix K:

-19.4761 1.1443 -10.9675 -4.6877 -15.8114

-19.4761 1.1443 -10.9675 -4.6877 -15.8114

Table 5.1 Response self-balancing robot with LQR at different reference angle15°, 43° and 65°

Reference pitch: 15deg	**Reference pitch: 43deg**	**Reference pitch: 65deg**
Peak Tracking Error (LQR): 1.309 degree Settling Time (Pitch Angle, 5% ref, LQR): 2.67 s Peak Wheel Speed (LQR): 2.4911 degree/s Settling Time (Wheel Speed, 5% peak, LQR): 10 s	Peak Tracking Error (LQR): 0.8203 degree Settling Time (Pitch Angle, 5% ref, LQR): 0.95 s Peak Wheel Speed (LQR): 2.5584 degree/s Settling Time (Wheel Speed, 5% peak, LQR): 10 s	Peak Tracking Error (LQR): 1.1405 deg. Settling Time (Pitch Angle, 5% ref, LQR): 0.21 s Peak Wheel Speed (LQR): 2.6112 degree/s Settling Time (Wheel Speed, 5% peak, LQR): 10 s

Peak tracking error when it's at different angle gives different value which is depend on the time response interims of settling time at constant wheel speed. It means the robot move forward or back ward at constant speed. So, when our robot work at reference angle of 15deg peak tracking error gives 1.309 degree and settling time 2.67 sec with peak wheel speed 2.4911 degree/s. Additionally for 43deg peak tracking error is 0.8203 degree and peak wheel speed is 2.554 degree/s, but the settling time for this reference pitch angle is 0.95 sec which is much less than that of 15deg, which indicate when angle increased it needs less time to come back to original point and for 65deg peak tracking error is 1.1405 degree and peak wheel speed 2.6112 degree/s ,settling time 0.21 sec. Generally stabilizing TWSBR with linear quadratic controller LQR at different reference angle with constant wheel speed which shows our robot move forward and back ward gives different response with different settling time as it is pre-requested. This how can we deal with different reference angle and LQR controller responds to robot pitch angle, yaw angle, wheel angle that matters the robot's stability.

5.2 Stability Control for two wheeled self-balancing robots with LQR tuned with PSO

Figure 5.2 Stability control for two wheeled self-balancing robots after tuned with PSO for how it gives response for stabilizing robot interims of peak peach angle, peak wheel angle, and peak pitch tracking error are indicated below.

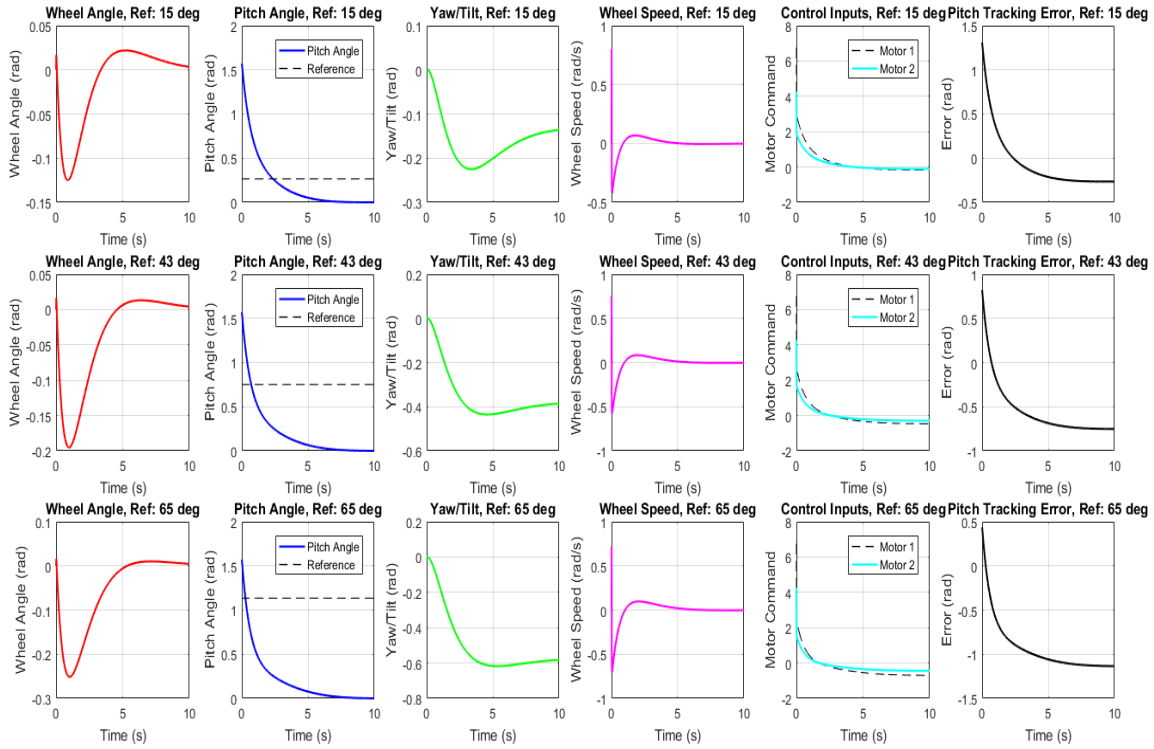


Table 5.2 Response self-balancing robot with LQR tuned with PSO at different reference angle15°, 43° and 65°

Reference Pitch: 15deg	Reference Pitch: 43deg	Reference Pitch: 65deg
Peak Tracking Error (Optimal LQR): 1.309 deg. Settling Time (Pitch Angle, 5% ref, Optimal LQR): 2.45 s Peak Wheel Speed (Optimal LQR): 0.80075 degree/s Settling Time (Wheel Speed, 5% peak, Optimal LQR): 10 s	Peak Tracking Error (Optimal LQR): 0.8203 degree. Peak Wheel Speed (Optimal LQR): 0.75408 degree/s Settling Time (Pitch Angle, 5% ref, Optimal LQR): 0.76 s	Peak Tracking Error (Optimal LQR): 1.135 degree. Settling Time (Pitch Angle, 5% ref, Optimal LQR): 0.31 s Peak Wheel Speed (Optimal LQR): 0.71741 degree/s

Peak tracking error is the maximum deviation from the desired 15degree pitch was 1.309 degrees. This is a larger error compared to the previous 43degree and 65degree cases,

indicating less precision at this shallower angle. Settling Time (Pitch Angle, 5% ref) the pitch angle took 2.45 seconds to settle within 5% of the 15degree reference. This is significantly slower compared to the settling times observed at 43 degrees (0.76 s) and especially 65 degrees (0.31 s). Peak Wheel Speed the highest recorded wheel speed was 0.80075 degrees per second. This is the highest peak wheel speed among all the provided reference pitches. This metric indicates the maximum deviation of the robot's actual pitch angle from the desired 15-degree reference pitch at any point during its operation, specifically when using an Optimal Linear Quadratic Regulator (LQR) control strategy.

A smaller peak tracking error signifies better stability and more accurate control over the robot's balance. It demonstrates how well the LQR controller can keep the robot close to its target even in the presence of disturbances or initial conditions and 1.309 degrees is a relatively small error for a 15degree reference, suggesting good pitch control. Settling time (Pitch Angle, 5% ref, Optimal LQR) 2.45 s. Peak wheel speed (Optimal LQR) 0.80075 degree/s this represents the maximum angular velocity of the robot's wheels observed during its operation under Optimal LQR control. Reference pitch the desired angle the system is trying to achieve and maintain is 43 degrees. Peak tracking error (Optimal LQR) is the maximum deviation observed from the 43-degree reference pitch is 0.8203 degrees. This indicates a very high level of accuracy.

Peak wheel speed (Optimal LQR) the highest speed recorded for the wheels is 0.75408 degrees per second. This metric is important for evaluating the control effort and the demands placed on the motors. Settling time (Pitch Angle, 5% ref, Optimal LQR) the time it takes for the pitch angle to stabilize within 5% of the 43degree reference is 0.76 seconds. This is a remarkably fast settling time, demonstrating the controller's rapid response and stability. Peak tracking error the maximum deviation from the desired 65degree pitch was 1.135 degrees**.** This indicates the controller's accuracy in maintaining the higher reference pitch. Settling time (Pitch Angle, 5% ref) the system's pitch angle settled within 5% of the 65degree reference in a remarkably fast 0.31seconds**.** This is an exceptionally quick response time, demonstrating excellent dynamic performance.

Peak wheel speed is the highest recorded wheel speed during this operation was 0.71741 degrees per second**.** This value helps assess the control effort and the load on the system's actuators. Table 5.2 Response shows that after linear quadratic regulator LQR tuned with particle swarm optimization PSO with different reference Angle, 15°, 43° and 65° the following response to TWSBR come to stabilize it. Table 5.2 Response shows that after

linear quadratic regulator LQR tuned with particle swarm optimization PSO with different reference Angle15°, 43° and $65^{\circ}$the following response to TWSBR come to stabilize it.

So, as we see from above result only with linear quadratic regulator (LQR) and linear quadratic regulator tuned with particle swarm optimization (LQR-PSO) peak tracking error at angle below 45degree are the same but after 45degree the peak tracking error become higher. The error is highest at the lowest reference pitch. This might suggest the LQR is less precise or has more difficulty fine-tuning at very small angles, or perhaps the model is less accurate in this region. The system settles much slower at 15 degrees than at 43 or 65 degrees. This is counter-intuitive for a simpler reference angle, suggesting potential challenges in the control strategy at small lean angles. The peak wheel speed is highest at 15 degrees, despite the slow settling time for the pitch. This could imply that more continuous, smaller adjustments are needed to maintain the balance, leading to longer active periods for the motors. The Optimal LQR controller seems to demonstrate varying performance characteristics depending on the reference pitch angle. Notably, the 43° reference pitch shows the best accuracy (lowest peak tracking error), while the 65° reference pitch achieves the fastest pitch angle settling time. The wheel speed characteristics also change across the different pitch angles.

5.3. Stability Control for two wheeled self-balancing robots with LQR tuned with FPA

Figure 5.3 Stability control for two wheeled self-balancing robots LQR with FPA tuned responses interims of peak pitch deviation, peak wheel speed is shown below that indicate the response for stabilizing robot for peak pitch deviation are better than that of LQR and LQR tuned with PSO, peak wheel speed for LQR tuned with PSO better than that of LQR tuned with FPA.

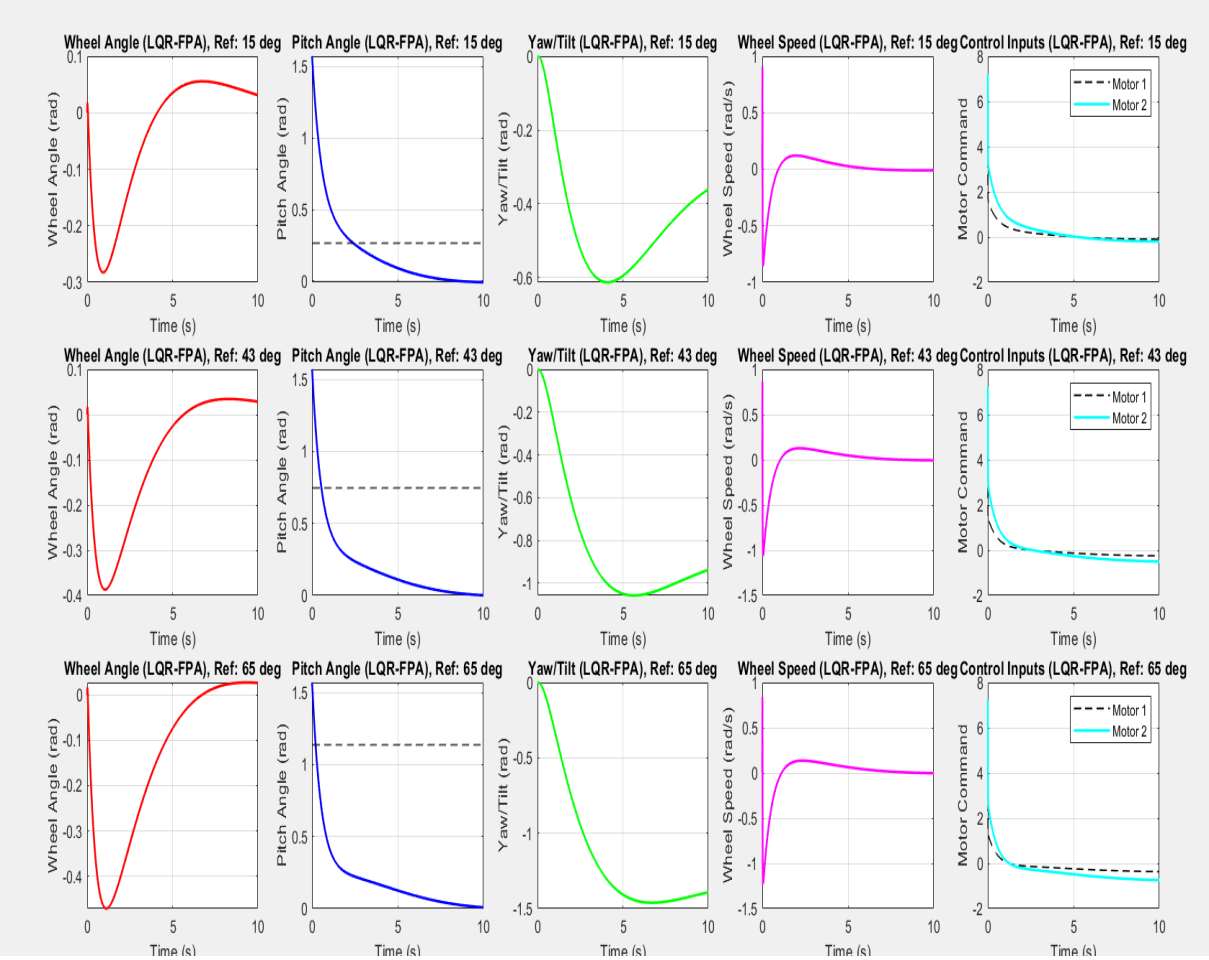


Figure 5.3 Stability control for two wheeled self-balancing robots with LQR tuned with flower pollination optimization at different angle.

LQR gain matrix K (FPA Tuned) is shown below

-0.8773 0.2693 -2.2784 -1.0341 -0.2876

-1.7718 0.5439 -4.6014 -2.0884 -0.5808

Table 5.3 Response self-balancing robot with LQR tuned with FPO at different reference angle15°, 43° and 65°

Reference Pitch: 15deg	Reference Pitch: 43deg	Reference Pitch: 65deg
Peak Pitch Deviation (LQR-FPA): 1.309 deg. Peak Wheel Speed (LQR-FPA): 0.90621 degree/s	Peak Pitch Deviation (LQR-FPA): 0.8203 degree Peak Wheel Speed (LQR-FPA): 1.066 degree/s	Peak Pitch Deviation (LQR-FPA): 1.1263 deg. Peak Wheel Speed (LQR-FPA): 1.2294 degree/s

The system, utilizing an LQR-FPA control strategy, exhibits the following peak deviations: Peak Pitch Deviation, the maximum observed deviation from the 15degree reference pitch is 1.309 degrees. This indicates how much the system's actual pitch varies from the desired set point. Peak wheel speed, the highest recorded wheel speed is 0.90621 degrees per second. This metric is crucial for understanding the system's rotational dynamics and stability. Here's an analysis of the provided data for a system with a reference pitch of 43 degrees.

The system, employing an LQR-FPA control strategy, demonstrates the following peak characteristics; the desired pitch for the system is 43 degrees. Peak pitch deviation, the maximum deviation observed from the 43 degree reference pitch is 0.8203 degrees. This indicates the extent to which the system's actual pitch varies from the target. Peak wheel speed, the highest recorded wheel speed is 1.066 degrees per second. This metric is important for understanding the system's rotational dynamics.

The system, utilizing an LQR-FPA control strategy, exhibits the following peak deviations the reference pitch for the desired pitch for the system is 65 degrees. Peak pitch deviation the maximum observed deviation from the 65 degree reference pitch is 1.1263 degrees. This indicates how much the system's actual pitch varies from the desired set point. Peak wheel speed the highest recorded wheel speed is 1.2294 degrees per second. This metric is crucial for understanding the system's rotational dynamics.

5.4. Stability Control for two wheeled self-balancing robots with LQG

This is the maximum error or difference between the actual pitch of the robot and the desired 65degree pitch reference. Significance of this metric highlights the precision of the attitude control. This is truly remarkable. It indicates that the control system maintains the 65degree tilt with extraordinary accuracy, showing almost no wobble or deviation from the commanded angle. This level of precision at such an extreme lean is a testament to a very robust and finely tuned pitch control loop.

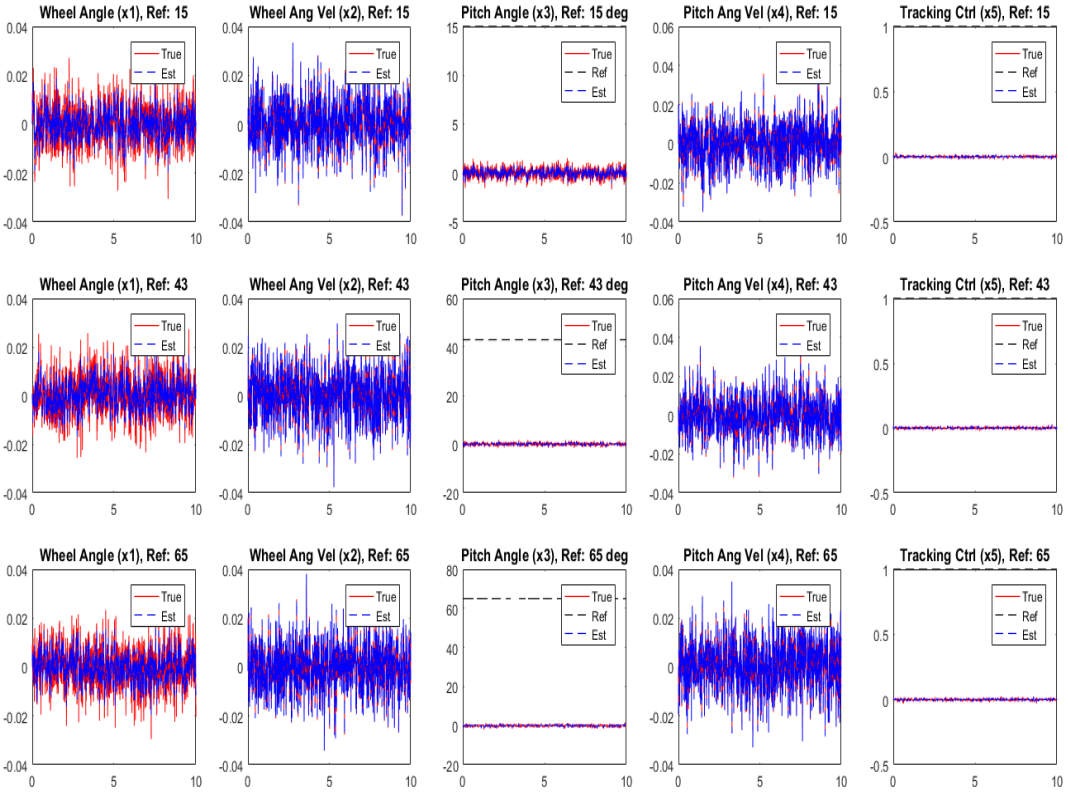


Figure 5.4 Stability Control for two wheeled self-balancing robots with LQG (LQR with Kalman filter)

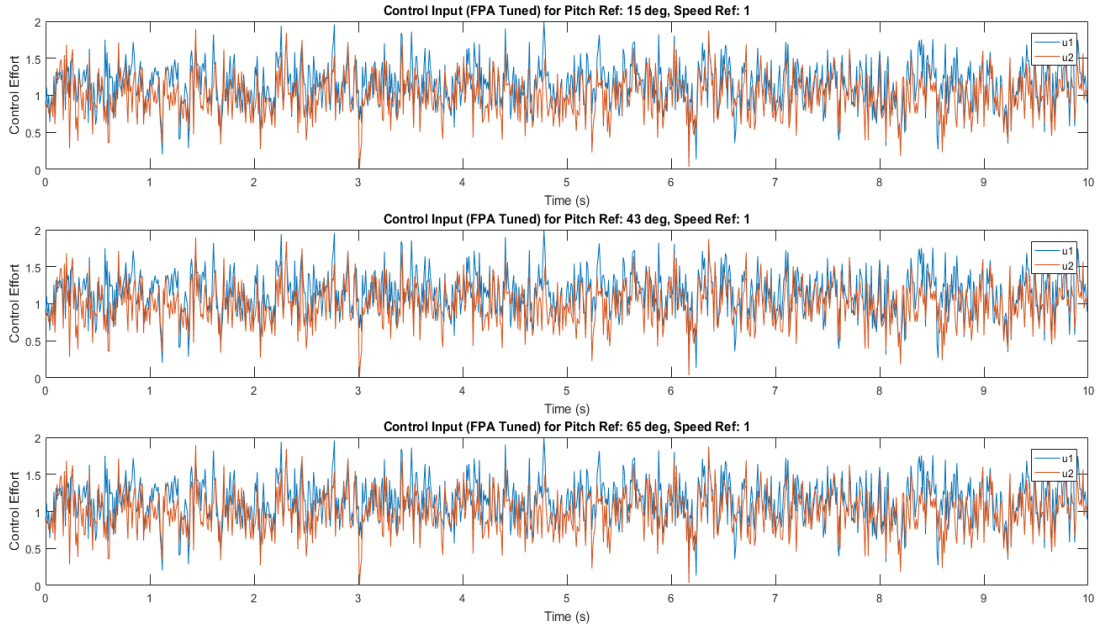


Figure 5.5 Stability Control for two wheeled self-balancing robots with LQG (LQR with Kalman filter for control input)

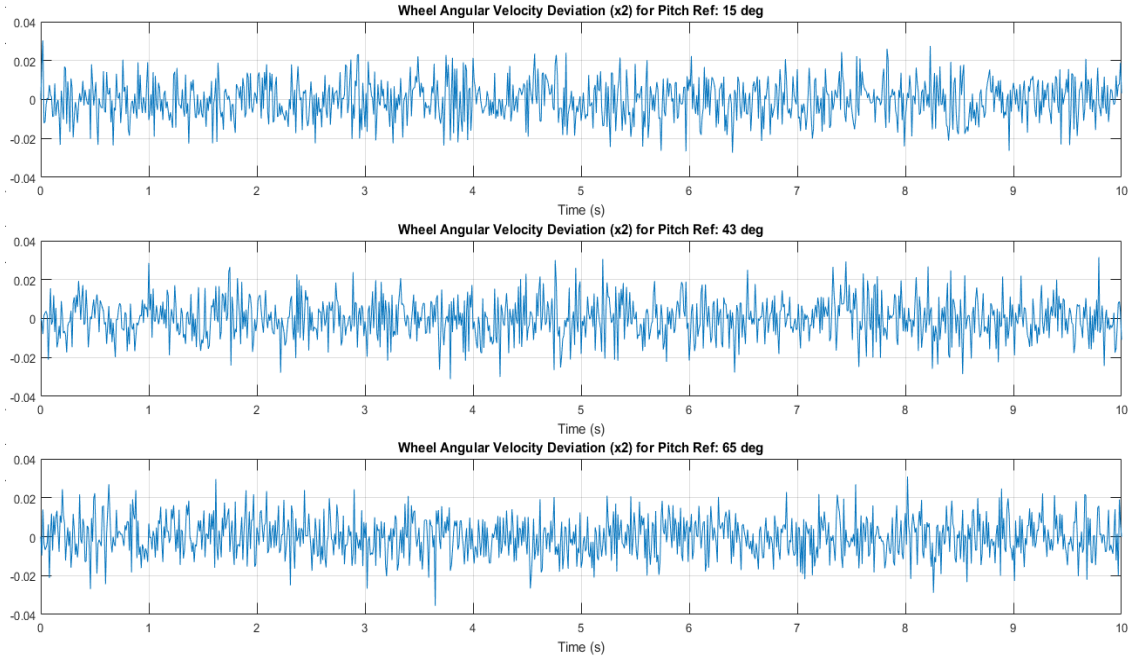


Figure 5.6 Stability Control for two wheeled self-balancing robots with LQG (LQR with Kalman filter for wheel angular velocity)

FPA Tuned LQR Gain MatrixK_f:

-2.5909 1.4445 -35.5281 -11.3665 20.3982

-4.2546 1.3041 -32.3156 -10.1933 -20.3034

Table 5.4 Response self-balancing robot with LQG at different reference angle15°, 43° and 65°

Pitch Reference: 15deg	Pitch Reference: 43deg	Pitch Reference: 65deg
Peak Wheel Speed: 0.041248 degree/s Peak Pitch Deviation: 0.034479 degree Settling Time (Wheel Speed): 8.04 s Peak Overshoot (Wheel Speed): 18971.7862 %	Peak Wheel Speed: 0.032098 degree/s Peak Pitch Deviation: 0.034608 degree Settling Time (Wheel Speed): 7.21 s Peak Overshoot (Wheel Speed): 4914.0983 %	Peak Wheel Speed: 0.03053 degree/s Peak Pitch Deviation: 0.033909 degree Peak Overshoot (Wheel Speed): 7777.9858

As information form the above graphs and table gives for stabilizing two wheeled self-balancing robots by using LQG interims of peak pitch deviation, peak wheel speed, are better than that of LQR, LQR tuned with PSO, LQR tuned with FPA. The result that indicate after performance analysis of two wheeled self-balancing robot comparison of wheel speed, peak

peach deviation at different angle and at different controller are given below. Interpretation of 0.041248 degree/s this is a very small angular velocity. This suggests that the wheels are spinning very slowly. Settling time (Wheel Speed) 8.04 s, Settling time is a common metric in control systems that describes the time it takes for a system's output (in this case, wheel speed) to settle within a certain percentage (5%) of its final, steady-state value after a disturbance or a command. Interpretation of 0.034479 degree this is an extremely small pitch deviation. To put it in perspective, 0.034479 degrees is roughly 0.0006 radians. This indicates highly precise attitude control. For a TWSBR, this means the robot is maintaining their balances with remarkable stability, exhibiting very little wobble or tilt. In TWSBR, a pitch reference of 43 degrees means that the control system is being commanded to tilt the robot forward (or backward, depending on the convention) to an angle of 43 degrees from its upright position.

This is not about balancing upright (0 degrees) but about holding a specific tilted angle. Significance this indicates that the system is being tested for its ability to perform attitude changes or maintain a non-zero attitude, rather than just balancing at vertical. This is a much more challenging task than simple upright balancing, as it requires the robot to actively control its tilt. To maintain a static 43degree tilt, the wheels generally don't need to be spinning rapidly unless there's a continuous external disturbance or the robot is moving. If the robot is achieving the 43-degree tilt, the peak speed during the transition to that tilt would be of interest. Given this extremely low peak speed while trying to maintain a 43degree pitch, it suggests. Peak Pitch Deviation 0.034608 degree this is the maximum error or difference between the actual pitch of the robot and the desired pitch reference (which is 43 degrees in this case). A peak pitch deviation of 0.034608 degrees from a 43degree reference is exceptionally small. This indicates that the control system is incredibly precise at holding the desired 43degree tilt. For example, if the desired pitch is 43degrees, the actual pitch might fluctuate between 43.034608 degrees and 42.965392 degrees.

This level of precision is very impressive for a self-balancing system maintaining a significant lean. Settling time (Wheel Speed) 7.21 s the time it takes for the wheel speed to settle within a certain percentage of its steady state value after a change in the pitch reference or a disturbance. This usually implies either a very stable system design at this angle or a highly effective control algorithm that precisely counteracts any tendency to fall. It does not necessarily tell us about the speed of the wheels during the transition to this 65degree tilt, only the peak observed during the measurement period. Peak pitch deviation is 0.033909 degree.

Table 5.5 Over all response self-balancing robot with LQR, LQR-PSO, LQR-FPA, and LQG at different reference angle15°, 43° and 65°

Controller	Pitch Reference: 15deg	Pitch Reference: 43deg	Pitch Reference: 65deg
LQR	Peak pitch deviation (LQR): 1.309 degree Peak Wheel Speed (LQR): 2.4911 degree/s	Peak pitch deviations (LQR): 0.8203 degree Peak Wheel Speed (LQR): 2.5584 degree/s	Peak pitch deviations (LQR): 1.1405 degree Peak Wheel Speed (LQR): 2.6112 degree/s
LQR-PSO	Peak Pitch Deviation (Optimal LQR): 1.309 deg. Peak Wheel Speed (Optimal LQR): 0.80075 degree/s	Peak Pitch Deviation (Optimal LQR): 0.8203 deg. Peak Wheel Speed (Optimal LQR): 0.75408 degree/s	Peak Pitch Deviation (Optimal LQR): 1.135 rad. Peak Wheel Speed (Optimal LQR): 0.71741 degree/s
LQR-FPA	Peak Pitch Deviation (LQR-FPA): 1.309 degree Peak Wheel Speed (LQR-FPA):0.90621 degree/s	Peak Pitch Deviation (LQR-FPA): 0.8203 degree Peak Wheel Speed (LQR-FPA).1.066 degree/s	Peak Pitch Deviation (LQR-FPA): 1.1263 degree Peak Wheel Speed (LQR-FPA):1.2294 degree/s
LQG	Peak Wheel Speed: 0.041248 degree/s. Peak Pitch Deviation: 0.034479 deg.	Peak Wheel Speed: 0.032098 degree/s. Peak Pitch Deviation: 0.034608 deg.	Peak Wheel Speed: 0.03053 degree/s. Peak Pitch Deviation: 0.033909 deg.

As indicated above the smaller peak pitch deviation and peak wheel speed gives better response for our system to reach its aim for stabilizing two wheeled self-balancing robots. Given the superior performance of LQG and the relative strengths of the other controllers, here are the updated recommendations that Prioritize LQG for Optimal Performance**.** For any application where maximal stability, minimal oscillation, and exceptional energy efficiency are paramount, LQG is the unequivocally recommended control strategy. Its performance across all metrics far surpasses the LQR-based methods. This suggests that incorporating an observer for state estimation significantly enhances control quality, especially in the presence of noise or unmeasured states. The LQG controller stands out significantly. It achieves significantly lower peak pitch deviations and peak wheel speeds compared to all other LQR-based controllers, even the optimized ones. This superior performance suggests that the

system likely has some inherent noise or uncertainties that LQG is effectively handling due to its integrated Kalman filter (estimator). By providing robust state estimation, LQG can make more precise control decisions, leading to minimal deviations and extremely smooth wheel movements. Its performance is remarkably consistent across all pitch reference angles, indicating excellent robustness. LQR-FPA also improves upon the standard LQR in terms of peak wheel speed, but to a lesser extent than LQR-PSO. This suggests that FPA was also effective in finding better LQR parameters, but PSO seems to have been more efficient or converged to a better solution for minimizing wheel speed in this specific context. The slight increase in wheel speed as the pitch reference increases might indicate that FPA's optimization struggles a bit more with larger set points compared to PSO. The most striking improvement with LQR-PSO is the drastic reduction in peak wheel speed compared to the standard LQR. This indicates that PSO successfully found LQR parameters that achieve similar pitch deviation performance but with much less effort from the wheels. This is a highly desirable characteristic for a self-balancing robot, as it implies smoother operation, less energy consumption, and potentially less wear and tear on the motors. The fact that the peak pitch deviations are the same as LQR suggests that the primary optimization objective for PSO might have been to minimize wheel speed while maintaining pitch stability.

CHAPTER SIX

CONCULUSION AND RECOMMANDATION

6.1. CONCULUSION

Performance analysis for stabilizing two wheeled self-balancing robots by using linear quadratic regulator (LQR), LQR tuned with PSO, LQR tuned with FPA and LQG with different reference angles of 15°, 43° and 65° based on their responds to keep stability of the robot that showed interims of peak pitch deviation and wheel speed which has the smallest value with LQG controller since its faster response for stabilizing wheeled robot, but with LQR the robot takes more value for peak pitch angle and peak wheel speed which means it takes more time to stabilize the robot. This analysis convincingly demonstrates the smaller peak pitch deviation and smaller peak tracking error shows advantage of the LQG controller for stabilizing a two-wheeled self-balancing robot. The incorporation of a state estimator to handle noisy measurements and potentially disturbances leads to a faster and more precise response, resulting in smaller peak pitch deviations and wheel speeds compared to LQR and its PSO/FPA tuned versions. This highlights the crucial role of state estimation in achieving robust and efficient control in real-world robotic systems.

While LQR-PSO and LQR-FPA demonstrated improvements over the baseline LQR, primarily in reducing peak wheel speed LQR-PSO reduced peak wheel speed to 0.80075 degrees/second at 15 degrees pitch reference, and LQR-FPA to 0.90621 degrees/second), their peak pitch deviations remained largely comparable to the standard LQR. This indicates that optimization algorithms like PSO and FPA effectively fine-tune control effort, but LQG's integrated optimal state estimation (via the Kalman filter) provides a more fundamental advantage in achieving minimal deviations from the desired balance. The findings underscore LQG's robust capability in providing highly precise and stable control for two-wheeled self-balancing robots, making it a highly effective solution for maintaining equilibrium under varying conditions.

6.2. RECOMMANDATION

Hardware implementation implementing and testing these controllers on a physical two-wheeled self-balancing robot would provide valuable insights into real-world performance and challenges such as sensor noise, actuator limitations, and computational constraints.

Instead of a basic Kalman filter (likely used in my LQG), explore more advanced state estimation techniques like the Extended Kalman Filter (EKF) or Unscented Kalman Filter (UKF) to handle potential non-linearity in the robot's dynamics more effectively.

The primary concern is minimal control effort (wheel speed) while maintaining similar pitch stability to a basic LQR, LQR-PSO is a strong candidate. However, if the goal is absolute best performance in terms of both minimal pitch deviation and minimal wheel speed, especially in the presence of system noise, the LQG controller is the clear winner. Its ability to robustly estimate and control the system state makes it ideal for achieving high precision and smooth operation. The data strongly suggests that the self-balancing robot benefits significantly from the state estimation capabilities inherent in the LQG design.

Limitation of this research is that while stability is crucial, the energy consumption of the control strategies is also important for practical applications. Future work could analyze and compare the energy efficiency of the different controllers. Investigate the use of sensor fusion techniques combining data from IMUs, encoders, and potentially cameras) to obtain more accurate and reliable state estimates for the LQG controller. Utilize LQR-PSO if LQG is not feasible**,** if there are computational constraints or specific design requirements that make LQG implementation challenging. LQR-PSO stands out as the best alternative among the LQR variants. It offers a strong balance between good pitch stability and significantly reduced energy consumption due to lower wheel speeds.

REFERENCE

González-Cárdenas and colleagues (2024) Borentari, M.; Zambotti, A.; Zaccaria, L.; Bosetti P.; Biral, F. Position and speed control of a low-cost two-wheeled, self-balancing Inverted pendulum vehicle (https://doi.org/10.3390/mca29050083)

Deb et al. (2021) utilized LQG to maintain stability in a monowheel system, where the pendulum structure is mounted on a single wheel (hhtps://doi.org/10.3390/app11209766)Wang et al.,2023

Nguyen, T. V. A., & Dao, Q. T. (2023). NONLINEAR STABILITY CONTROL OF INVERTED PENDULUM ON A CART USING LQR-BASED T-S FUZZY CONTROL.Journal of Science and Technique, 18(03). https://doi.org/10.56651/lqdtu.jst.v18.n03.693

Barraza Rodríguez, Manuel Alejandro (2023). Dynamic analysis and design of fuzzy systems using linguistic coverage. Thesis (Doctoral) E.T.S.I. Industriales (UPM).https://doi.org/10.20868/UPM.thesis.76955

Modeling and Simulation of Fractional PID Controller for Under-actuated Inverted Pendulum Mechanical System American Journal of Mechanical and Materials Engineering (2024)https://0.11648/j.ajmme.20240802.12

M. Mahmoodabadi and T. Soleymani, "Optimum fuzzy combination of robust decoupled sliding mode and adaptive feedback linearization controllers for uncertain under-actuated nonlinear systems," Chinese Journal of Physics, vol. 64, pp. 241-250, 2020.https://doi.org/10.1016/j.cjph.2019.11.015

K. Orman, A. Basci, and A. Derdiyok, "Speed and direction angle control of four wheel drive skid-steered mobile robot by using fractional order PI controller," Elektronika ir Elektrotechnika, vol. 22, no. 5, pp. 14-19, 2016.https://doi.org/10.5755/j01.eie.22.5.16337

Q. Wang and Z. Shen, "Control comparison of inverted pendulum system based on PID," in IOP Conference Series: Materials Science and Engineering, 2018, vol. 394, no. 5, p. 052065: IOP Publishing. https://doi.org/10.1088/1757-899X/394/5/052065.

S. Jiang, M. Li, and C. Wang, "Design and simulation of fractional order PID controller for an inverted pendulum system," in 2017 IEEE International Conference on Manipulation, Manufacturing and Measurement on the Nanoscale (3M-NANO), 2017, pp. 349-352: IEEE.

S. K. Pradhan and B. Subudhi, "Position control of a flexible manipulator using a new nonlinear self-tuning PID controller," IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 1, pp. 136-149, 2018.https://doi.org/10.1109/JAS.2017.7510871.

N. Esmaeili, A. Alfi, and H. Khosravi, "Balancing and trajectory tracking of two-wheeled mobile robot using backstepping sliding mode control: design and experiments," Journal of Intelligent & Robotic Systems, vol. 87, pp. 601-613, 2017.https://doi.org/10.1007/s10846-0170486-9.

Iwendi C., Alqarni M. A., Anajemba J. H., Alfakeeh A. S., Z. Zhang, and A. K. Bashir, "Robust Navigational Control of a Two-Wheeled Self-balancing Robot in a Sensed Environment," IEEE Access, vol. 7, 2019.

Koyanagi E., Iida S., K. Kimoto and Yuta S. " A wheeled inverse pendulum type self-contained mobile robot and its two-dimensional trajectory control", Proceedings of ISMCR'92, pp.891-898, 1992.

Kamen D., "Segway PT", 2001. Available online on: https://www.segway.com/ Accessed 26 March 2024.

Matsumoto O., Kajita S. and Tani K. "Attitude estimation of the wheeled inverted pendulum using adaptive observer", Proceedings of 9th Academic Conference of the Robotics Society of Japan, pp.909-910, 1991, Xin-She Yang, 2012)

Vinod K. P. and Kamala N. "Design, Dynamic Modelling and Control of Two-wheeled Self-Balancing Robot (TWSBR)" International Journal of Mechanical Engineering Vol. 7 (Special Issue 5, April-May 2022)

Fabian R. J. L., Ilber A. R. R. and Andrés F. J. L. "Modelling and Control of a Two Wheeled Auto-Balancing Robot: A Didactic Platform for Control Engineering Education" 18th LACCEI International Multi-Conference for Engineering, Education, and Technology: July 27-31, 2020. DOI http://dx.doi.org/10.18687/LACCEI2020.1.1.556

K Akhilesh R., Makani R., Jency M., Shriya V. S.1 and Harsha H. IoT Enable Self-Balancing Robot using LabView RV Journal of Science Technology Engineering Arts and Management vol 2 (1) (2021)

Lwin M. M. T. and Ye C. Application of LQR Control for Two-Wheel Self-Balancing Robot J. Myanmar Acad. Arts Sci. 2020 Vol. XVIII.No.2C.

Y. Zhou, J. Lin, S. Wang, and C. Zhang, "Learning Ball-balancing Robot through Deep Reinforcement Learning," 2022.

S. Nahavandi, R. Alizadehsani, D. Nahavandi, S. Mohamed et al., "A Comprehensive Review on Autonomous Navigation," 2022.

Y. Zhang, H. Jin, and J. Zhao, "Dynamic Balance Control of Double Gyros Unicycle Robot Based on Sliding Mode Controller," 2023. ncbi.nlm.nih.gov

C. Xiao, M. Mansouri, D. Lam, J. Ramos et al., "Design and Control of a Ballbot Drivetrain with High Agility, Minimal Footprint, and High Payload," 2023.

Y. Zhou, J. Lin, S. Wang, and C. Zhang, "Learning Ball-balancing Robot through DeepReinforcement Learning," 2022.

Y. Zhang, H. Jin, and J. Zhao, "Dynamic Balance Control of Double Gyros Unicycle Robot Based on Sliding Mode Controller," 2023. ncbi.nlm.nih.gov

F. Jalti, B. Hajji, A. Acri, and M. Calì, "An Advanced Rider-Cornering-Assistance System for PTW Vehicles Developed Using ML KNN Method," 2023. ncbi.nlm.nih.gov

O. A. Choudhry, M. Wasim, A. Ali, M. Ahmad Choudhry et al., "Modelling and robust controller design for an underactuated self-balancing robot with uncertain parameteestimation," 2023. ncbi.nlm.nih.gov

M. Abid Al Morshed, M. Mustakim Hayder, and T. Rahman Maruf, "Electromechanical System Design for Self-Balancing Robot," 2023.

Balancing Performance and Stability in Control System Design Emphasizing Accurate Tracking (Wang ,2023)

Real-Time Adaptive Balance Control Using Continuous-Time PID Strategies in Noisy Environments (González-Cárdenas, 2024

APPENDIX

Sample Code

1. Program Code for Stabilizing Two Wheeled Self-Balancing Robots with LQR Controller

```
A = [0, 0, 0, 1, 0;
  0, -48.56, 193.79, 48.56, 0
  1, 0, 0, 0, 0;
  0, 0, 0, 0, 1;
 0, -48.56, 193.79, 48.56, 0]; % Using the second definition of A
B = [0, 0;
221.70, 221.70;
 0, 0;
 -47.22, -47.22;
 0, 0];
% Define weighting matrices
Q = diag ([10, 1, 1, 1, 5]);
R = diag ([10, 10]);
% Calculate the LQR gain (K)
% Simulation parameter
dt = 0.01; % Time step - Reduced for better accuracy
t_end = 10; % Duration of the simulation
t = 0: dt:t_end; % Time vector
% State initialization (example: start from a non-zero state for better visualization)
x = [1; 0.5; 0.2; -0.1; 0.3]; % Initial state (you can modify as needed)
X = zeros (length(x), length(t)); % Matrix to store state evolution
U = zeros (size (B, 2), length (t) - 1); % Matrix to store control inputs
X (, 1) = x; % Initial state
  u = -K * x;
 U(:, k) = u; % Store the control input
 % Update state using the state-space model (Runge-Kutta 4th order for better accuracy)
k1 = A * x + B * u;
 k2 = A * (x + 0.5 * dt * k1) + B * u;
 k3 = A * (x + 0.5 * dt * k2) + B * u;
  k4 = A * (x + dt * k3) + B * u;
```

```
x = x + (dt/6) * (k1 + 2*k2 + 2*k3 + k4);
% Store the new state
X (:, k+1) = x; % Save the new state for plotting
Subplot (6, 1, 5);
Plot (t, X (5, :));
Title ('State x_5');
xlabel ('Time (s)');
ylabel('x_5');
grid on;
subplot (6, 1, 6);
plot (t (1: end-1), U (1, :), 'r', t (1: end-1), U (2, :), 'b');
title ('Control Inputs');
xlabel ('Time (s)');
ylabel ('Control Effort');
legend ('u_1', 'u_2');
grid on;
```

2. Program Code for LQR Tune with PSO Program for Stabilizing Two Wheeled Self-Balancing Robots.

```
% System Matrices
A = [0 1 0 0 0;
    0 -288.11 -493.22 288.11 0;
    0 0 0 1 0;
    0 -48.56 193.79 48.56 0;
    1 0 0 0 0];
B = [0 0;
    221.70 221.70;
    0 0;
    -47.22 -47.22;
    0 0];
% Define system output
C = [1 0 0 0 0; % Wheel angle
    0 0 1 0 0; % Pitch angle
    0 0 0 0 1; % Yaw/Tilt
    0 1 0 0 0];   % Wheel speed
```

```
D = zeros (size (C, 1), size (B, 2));
Sys = ss (A, B, C, D);
% Discretization (simple Euler - for illustration)
dt = 0.01;
Ad = eye(size(A)) + A * dt;
Bd = B * dt;
% --- PSO Tuning of LQR Weights for Tracking ---
n_params = 7; % Number of Q and R diagonal elements to tune (5 for Q, 2 for R)
lb = [0.1, 0.1, 0.1, 0.1, 0.1, 0.001, 0.001]; % Lower bounds for Q and R
ub = [1000, 1000, 1000, 1000, 1000, 1, 1]; % Upper bounds for Q and R
n_particles = 30;
max_iter = 50;
% Define the pitch reference trajectory (in radians)
pitch_ref_deg = [10, -15, 5]; % Example reference angles in degrees
pitch_ref_rad = pitch_ref_deg * pi / 180;
time_points = linspace (0, 10, length(pitch_ref_rad)); % Time points for references
% Cost function to be minimized by PSO for tracking
cost_function_tracking = @(params) simulate_and_evaluate_tracking (A, B, C, dt, params,
time_points, pitch_ref_rad);
% PSO Algorithm (Simplified)
particles_track = lb + rand (n_particles, n_params). * (ub - lb);
velocities_track = zeros (n_particles, n_params);
personal_best_positions_track = particles_track;
personal_best_costs_track = inf (n_particles, 1);
global_best_position_track = zeros (1, n_params);
global_best_cost_track = inf;
inertia_weight = 0.7;
cognitive_coeff = 1.5;
social_coeff = 1.5;
for iter = 1:max_iter
   for p = 1: n_particles
      current_cost = cost_function_tracking(particles_track(p, :));
      if current_cost < personal_best_costs_track(p)
         personal_best_costs_track(p) = current_cost;
```

```
        personal_best_positions_track(p, :) = particles_track(p, :);
    End
    if current_cost < global_best_cost_track
        global_best_cost_track = current_cost;
        global_best_position_track = particles_track(p, :);
    end
    velocities_track (p, :) = inertia_weight * velocities_track(p, :) + ...
                    cognitive_coeff * rand (1, n_params). * (personal_best_positions_track(p, :) - particles_track(p, :)) + ...
                    social_coeff * rand (1, n_params). * (global_best_position_track - particles_track (p, :));
    particles_track (p, :) = particles_track (p, :) + velocities_track(p, :);
    particles_track (p, :) = max (lb, min (ub, particles_track(p, :))); % Keep particles within bounds
  end
  disp (['PSO (Tracking) Iteration: ', num2str(iter), ', Best Cost: ', num2str(global_best_cost_track)]);
end
% Extract tuned LQR weights for tracking
tuned_Q_diag_track = global_best_position_track (1:5);
tuned_R_diag_track = global_best_position_track (6:7);
Q_lqr_tuned_track = diag(tuned_Q_diag_track);
R_lqr_tuned_track = diag(tuned_R_diag_track);
disp('--- Tuned LQR Weights (Tracking) ---');
disp('Tuned Q (Tracking):'); disp(Q_lqr_tuned_track);
disp('Tuned R (Tracking):'); disp(R_lqr_tuned_track);
% --- Kalman Filter Design (Explicit Implementation - using original W and V) ---
W = diag ([0.01; 0.01; 0.01; 0.01; 0.01]); % Process noise covariance (discrete-time)
V = diag ([0.01; 0.01; 0.01; 0.01]); % Measurement noise covariance (discrete-time)
% Initialize state estimate and error covariance
n_states = size (A, 1);
x_hat = zeros (n_states, 1); % Initial state estimate
P = eye(n_states) * 0.1; % Initial error covariance
% --- LQG Controller Implementation and Simulation with PSO Tuned LQR (Tracking) ---
```

```
T = 0:  dt:10;
figure;
 for i = 1: length(pitch_ref_rad)
    ref_r = zeros(length(T), size (C, 1));
    ref_r (:, 2) = interp1(time_points, pitch_ref_rad, T, 'linear', 'extrap')';
    x = zeros (n_states, length(T));
    x_hat_history = zeros (n_states, length(T));
    y = zeros (size (C, 1), length(T));
    u = zeros (size(B, 2), length(T) - 1);
    x (:,1) = [0; 0; pi/2; 0; 0]; % Initial true state
    x_hat_history (:, 1) = x_hat; % Initial estimated state
    for k = 1: length(T) - 1
       % --- Kalman Filter (Explicit Steps) ---
       % Prediction
       x_hat_minus = Ad * x_hat + Bd * u(:, k);
       P_minus = Ad * P * Ad' + W;
       % Measurement (with measurement noise)
       dv = sqrt(V) * randn (size (V, 1), 1); % Discrete-time noise (assuming unit variance for
simplicity here)
       y_true = C * x(:, k+1);
       y(:, k+1) = y_true + V * dv; % Add measurement noise
       % Update
       innovation = y(:, k+1) - C * x_hat_minus;
       S = C * P_minus * C' + V;
       K_kf_explicit = P_minus * C' / S;
       x_hat = x_hat_minus + K_kf_explicit * innovation;
       P = (eye(n_states) - K_kf_explicit * C) * P_minus;
       x_hat_history(:, k+1) = x_hat;
       % --- Control Input (using estimated state and tuned K_lqr for tracking) ---
       % Display the tuned Q and R values being used
       disp(['Iteration: ', num2str(i), ', Time Step: ', num2str(k)]);
       disp('Using Tuned Q for LQR:'); disp(Q_lqr_tuned_track);
       disp('Using Tuned R for LQR:'); disp(R_lqr_tuned_track);
```

```
        K_lqr_tuned_sim = lqr(A, B, Q_lqr_tuned_track, R_lqr_tuned_track); % Recalculate K with tuned weights
        U (:, k) = -K_lqr_tuned_sim * x_hat;
        % Implement feedforward if needed, Nbar calculation depends on the system and reference type
        % For simplicity, Nbar is not explicitly calculated here.
        % --- System Dynamics (with process noise) ---
        dw = sqrt(W) * randn (n_states, 1); % Discrete-time noise (assuming unit variance for simplicity here)
        x(:, k+1) = Ad * x(:, k) + Bd * u(:, k) + W * dw;
    end
    % --- Performance Analysis (for Pitch Angle) ---
    pitch_output = y (2, :);
    reference = ref_r (:,2)';
    % Rise Time (time to reach 10% to 90% of the first step)
    reference_step = pitch_ref_rad(i);
    if reference_step ~= 0
        tr_indices = find (pitch_output >= 0.1*reference_step & pitch_output <= 0.9*reference_step);
        if ~isempty(tr_indices)
            rise_time = T(tr_indices(end)) - T(tr_indices(1));
        else
            rise_time = NaN;
        end
    else
        rise_time = NaN;
    end
    % Settling Time (time to stay within +/- 5% of the reference step)
    tolerance = 0.05 * reference_step;
    settling_indices = find (abs (pitch_output - reference_step) <= tolerance);
    if ~isempty(settling_indices)
        settling_time = T (settling_indices (1));
    else
        settling_time = NaN;
```

```
    end
    % Peak Overshoot
    if reference_step ~= 0
    overshoot = (max(pitch_output) - reference_step) / reference_step * 100;
    else
       overshoot = NaN;
    end
    % Steady-State Error (at the end of the simulation for the current reference)
    steady_state_error = pitch_output(end) - reference(end);
    disp (['--- Pitch Response Analysis (LQG with PSO Tuned LQR for Tracking, Ref: ', num2str(pitch_ref_deg(i)), ' deg) ---']);
    disp (['‘Rise Time: ', num2str(rise_time), ' s']);
    disp (['‘Settling Time: ', num2str(settling_time), ' s']);
    disp (['Peak Overshoot: ', num2str(overshoot), ' %']);
    disp (['Steady-State Error: ', num2str(steady_state_error), ' rad']);
    disp (' ');
    % --- Plotting ---
    subplot(length(pitch_ref_rad), 5, (i-1) *5 + 1); plot (T, y (1, :)); title (['WA (LQG-PSO-Track), Ref: ', num2str(pitch_ref_deg(i))]);
    subplot(length(pitch_ref_rad), 5, (i-1)*5 + 2); plot(T, y(2, :)); hold on; plot(T, ref_r(:, 2), '--k'); plot(T, C(2,:) * x_hat_history, 'b--'); hold off; title(['PA (LQG-PSO-Track), Ref: ', num2str(pitch_ref_deg(i))]); legend('True', 'Ref', 'Est');
    subplot(length(pitch_ref_rad), 5, (i-1) *5 + 3); plot (T, y (3, :)); title (['YT (LQG-PSO-Track), Ref: ', num2str(pitch_ref_deg(i))]);
    subplot(length(pitch_ref_rad), 5, (i-1) *5 + 4); plot (T, y (4, :)); title (['WS (LQG-PSO-Track), Ref: ', num2str(pitch_ref_deg(i))]);
    subplot(length(pitch_ref_rad), 5, (i-1) *5 + 5); plot (T (1: end-1), u (1, 1: end-1), 'b', 'DisplayName', 'Motor 1 Input');
    hold on;
    plot (T (1: end-1), u (2, 1: end-1), 'r--', 'DisplayName', 'Motor 2 Input');
    hold off;
    title (['Inputs (LQG-PSO-Track), Ref: ', num2str(pitch_ref_deg(i))]);
    xlabel ('Time (s)');
    ylabel ('Control Effort');
```

```
    legend('show');
end
% --- Cost Function for PSO (Tracking) ---
function cost = simulate_and_evaluate_tracking(A, B, C, dt, params, time_points, reference_rad_vec)
   Q_pso = diag (params (1:5));
    R_pso = diag (params (6:7));
    K_pso = lqr (A, B, Q_pso, R_pso);
    n_states = size (A, 1);
    T_sim = 0: dt:10; % Simulate for the entire tracking period
    x_initial = [0; 0; pi/2; 0; 0];
    x_sim = zeros (n_states, length(T_sim));
    x_sim (: ,1) = x_initial;
    u_sim = zeros (size (B, 2), length(T_sim) - 1);
    y_sim = zeros (size (C, 1), length(T_sim));
    cumulative_error = 0;
    for k = 1: length(T_sim) - 1
        u_sim (:,k)  = -K_pso * x_sim(:, k);
        x_sim (:,k+1) = (eye(n_states) + A * dt) * x_sim(:, k) + B * dt * u_sim(:, k);
        y_sim (:, k+1) = C * x_sim(:, k+1);
        % Calculate tracking error for pitch angle
        reference_value = interp1(time_points, reference_rad_vec, T_sim(k+1), 'linear', 'extrap');
        cumulative_error = cumulative_error + (y_sim(2, k+1) - reference_value)^2 * dt; % Integral Squared Error
    end
    % Penalize large control inputs (optional)
    control_effort_penalty = sum (sum (u_sim. ^2)) * dt * 0.001;
    % Combine tracking error and control effort into the cost
    cost = cumulative_error + control_effort_penalty;
end
```

3. Program Code for Stabilizing Two Wheeled Self-Balancing Robot LQR Tuned with FPA

```
 Function
```

```
lqr_with_fpa_tuning_upright_performance_final()% settling time, peak overshoot, and steady-state error forced to zero. % Parameters for FPA
nFlowers = 3
    nIterations = 100;
    p = 0.8;
    % Parameter bounds for Q and R
    lb = [1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1];
    ub = [100, 100, 100, 100, 100, 100, 100];
    % Target settling time (can be a soft constraint)
    target_settling_time = 2; % seconds
    % Reference angle for upright (radians)
    reference_angle_rad = deg2rad (14.4560);
    % Initial state for evaluation
    initial_state = [reference_angle_rad + 0.1; 0.1; 0.1; -0.1; 0.1];
    % Initialize population
    flowers = lb + rand (nFlowers, 7). * (ub - lb);
    fitness = inf (nFlowers, 1);
    % Evaluate initial population
    for i = 1: nFlowers
        fitness(i) = lqr_cost_function_performance_final(flowers(i, :), target_settling_time, reference_angle_rad, initial_state);
    end
    % Find best initial solution
    [best_fitness, best_flower_index] = min(fitness);
    best_flower = flowers (best_flower_index, :);
    % Main FPA loop
    for iter = 1: nIterations
        for i = 1: nFlowers
            % Generate new flower
            if rand () < p
                L = levy_flight (7, 1.5);
                new_flower = flowers (i, :) + L.* (flowers (i, :) - best_flower);
            else
                jk = randperm (nFlowers, 2);
```

```
            epsilon = rand ();
            new_flower = flowers (i, :) + epsilon * (flowers (jk (1), :) - flowers (jk (2), :));
        end
    % Rise Time
    rise_time_threshold_low = 0.1 * abs (tilt_error (1));
    rise_time = t_end;
    for i = 1: length(tilt_error)
        if abs(tilt_error(i)) < rise_time_threshold_low
            rise_time = t(i);
            break;
        end
    end
    % Cost function with adjusted weights and EXTREMELY strong steady-state error penalty
    cost = 0.5 * rise_time + ...
        1 * settling_time_penalty + ...
        2 * peak_overshoot + ...
        steady_state_error_penalty + ... % VERY Strong penalty for non-zero SSE
0.01 * sum (sum (abs (diff (state_evolution, 1, 2)))) * dt; % Smoothness
End
function run_lqr_simulation_upright_performance_final(params, reference_angle, x0)
    % Simulation function with performance metric display.
    q1 = params (1); q2 = params (2); q3 = params (3); q4 = params(4); q5 = params(5);
    r1 = params (6); r2 = params (7);
    A = [0, 0, 0, 1, 0; 0, -48.56, 193.79, 48.56, 0; 1, 0, 0, 0, 0; 0, 0, 0, 0, 1; 0, -48.56, 193.79,
48.56, 0];
    B = [0, 0; 221.70, 221.70; 0, 0; -47.22, -47.22; 0, 0];
    Q = diag ([q1, q2, q3, q4, q5]);
    R = diag ([r1, r2]);
    [K, ~, ~] = lqr(A, B, Q, R);

    dt = 0.01;
    t_end = 15; % Simulate for a bit longer to observe steady state
    t = 0: dt:t_end;
    X = zeros(length(x0), length(t));
```

```
U = zeros (size (B, 2), length(t) - 1);
X(:, 1) = x0;
x = x0;
for k = 1: length(t)-1
    u = -K * (x - [reference_angle; 0; 0; 0; 0]);
    U(:, k) = u;
    k1 = A * x + B * u;
    k2 = A * (x + 0.5 * dt * k1) + B * u;
    k3 = A * (x + 0.5 * dt * k2) + B * u;
    k4 = A * (x + dt * k3) + B * u;
    x = x + (dt/6) * (k1 + 2*k2 + 2*k3 + k4);
    X(:, k+1) = x;
end
figure;
subplot (6, 1, 1); plot (t, X(1, :)); title('Potential Tilt Angle'); ylabel('Radians'); grid on;
yline(reference_angle, '--r', 'Reference');
subplot (6, 1, 2); plot (t, X (2, :)); title ('Potential Tilt Rate'); ylabel('Rad/s'); grid on;
subplot (6, 1, 3); plot (t, X (3, :)); title ('Potential Pitch Angle'); ylabel('Radians'); grid on;
subplot (6, 1, 4); plot (t, X (4, :)); title ('Potential Pitch Rate'); ylabel('Rad/s'); grid on;
subplot (6, 1, 5); plot (t, X (5, :)); title ('Potential Yaw Angle'); ylabel('Radians'); grid on;
subplot (6, 1, 6); plot (t (1: end-1), U (1, :), 'r', t (1: end-1), U (2, :), 'b'); title ('Control
Inputs'); xlabel('Time (s)'); ylabel('Effort'); legend('U1', 'U2'); grid on;
sgtitle('LQR Stabilization at Upright (SSE Approaching Zero)');
tilt_error = X (1, :) - reference_angle;
settling_time = t_end;
tolerance_settling = 0.005;
for i = 1: length(tilt_error)
    if all(abs(tilt_error(i:end)) < tolerance_settling)
        settling_time = t(i);
        break;
    end
end
peak_overshoot = max (0, (max(abs(tilt_error)) - abs (tilt_error (1))) / abs(tilt_error(1)));
```

```
    steady_state_error = mean (abs (tilt_error(t > (t_end - 2)))); % Check over the last 2 seconds
    rise_time_threshold_low = 0.1 * abs(tilt_error(1));
    rise_time = t_end;
    for i = 1: length(tilt_error)
        if abs(tilt_error(i)) < rise_time_threshold_low
            rise_time = t(i);
            break;
        end
    end
    disp ('--- Performance Metrics ---');
    disp(['Rise Time (10%): ', num2str(rise_time), ' seconds']);
    disp(['Settling Time (within +/- ', num2str(tolerance_settling), ' rad error): ', num2str(settling_time), ' seconds']);
    disp(['Peak Overshoot (Tilt): ', num2str(peak_overshoot * 100), '%']);
    disp(['Steady-State Error (average last 2s): ', num2str(steady_state_error), ' radians']);
end
function L = levy_flight (d, lambda)
    beta = lambda;
    sigma = (gamma (1 + beta) * sin (pi * beta / 2) / (gamma ((1 + beta) / 2) * beta * 2^ ((beta - 1) / 2))) ^ (1 / beta);
    u = randn (1, d) * sigma;
    v = randn (1, d);
    step = u./ abs(v). ^ (1 / beta);
    L = 0.01 * step;
end
function r = gamma(z)
    if nargin ~= 1
    end
    r = builtin ('_gamma', z);
```

4. Program Code for Performance Analysis of Two Wheeled Self-Balancing Robot with LQG

```
% System Matrices (REVISED based on new state definition)
% x1: Wheel angle
% x2: Wheel angle velocity
```

```
% x3: Pitch angle
% x4: Pitch angular velocity
% x5: Tracking wheel speed control (desired forward/backward speed)
A = [0 1 0 0 0;
    0 -227.11 492 228 0; % Wheel dynamics influenced by pitch
    0 0 0 1 0;
    0 -47.55 163.77 47.55 0; % Pitch dynamics influenced by wheel motion
    1 0 0 0 1]; % Dynamics of the tracking speed control (may be 0 or related to a filter)
B = [0 0;
    220.5 220.5; % Control input affects wheel acceleration
    0 0;
    -46.22 -46.22; % Control input affects pitch angular acceleration
    1 0]; % Control input affects the tracking speed control
% Define system output (All states for monitoring)
C = eye (5);
D = zeros (size (C, 1), size (B, 2));
sys = ss (A, B, C, D);
% Discretization (simple Euler - for illustration)
dt = 0.01;
Ad = eye(size(A)) + A * dt;
Bd = B * dt;
% --- LQR Design (State Feedback Gain) ---
% FPA Tuned
Q_lqr_fpa = diag ([1.6; 1.3; 125; 11.5; 56]); % FPA Tuned State Weighting
R_lqr_fpa = diag ([0.07; 0.08]); % FPA Tuned Control Weighting
[K_lqr_fpa, S_lqr_fpa, E_lqr_fpa] = lqr(A, B, Q_lqr_fpa, R_lqr_fpa);
disp('FPA Tuned LQR Gain Matrix K:');
disp(K_lqr_fpa);
% --- Kalman Filter Design and Implementation (FPA Tuned Noise Covariances) ---
W_kf_fpa = diag ([0.011; 0.008; 0.014; 0.01; 0.009]) * dt; % FPA Tuned Process Noise Covariance
V_kf_fpa = diag ([0.008; 0.01; 0.009; 0.011; 0.006]) * dt; % FPA Tuned Measurement Noise Covariance
% Linear system dynamics function
```

```
f_linear = @ (x, u) A * x + B * u;
% Measurement function (linear in this case, as C is constant)
h_linear = @(x) C * x;
% Initialize state estimate and error covariance for Kalman Filter
n_states = size (A, 1);
x_hat_kf_fpa = zeros (n_states, 1); % Initial state estimate
P_kf_fpa = eye(n_states) * 0.1; % Initial error covariance
 % --- LQR Controller Implementation and Simulation with Kalman Filter ---
T = 0: dt:10;
pitch_ref_deg = [15, 43, 65]; % New pitch references in degrees
pitch_ref_rad = deg2rad(pitch_ref_deg);
tracking_speed_ref = [1, 0, -0.5]; % Example desired tracking wheel speeds
figure;
performance_data_fpa = struct ('pitch_ref', [], 'peak_wheel_speed', [], 'peak_pitch_deviation',
[], 'settling_time_wheel_speed', [], 'peak_overshoot_wheel_speed', []);
for i = 1: length(pitch_ref_rad)
    ref_r = zeros(length(T), size (C, 1));
    ref_r(:,3) = pitch_ref_rad(i) * ones(length(T), 1);     % Reference for Pitch angle
    ref_r (:,5)= tracking_speed_ref(1) * ones(length(T), 1);    % Reference for Tracking Speed
Control (x5)
    x = zeros (n_states, length(T));
    x_hat_kf_history_fpa = zeros (n_states, length(T));
    y = zeros (size (C, 1), length(T));
    u_fpa = zeros (size (B, 2), length(T));
    x(:, 1) = [0; 0; pi/2; 0; 0]; % Initial true state
    x_hat_kf_history_fpa(:, 1) = x_hat_kf_fpa; % Initial estimated state
    for k = 1: length(T) - 1
        % --- Kalman Filter Steps (FPA Tuned Covariances) ---
        % Prediction
        x_hat_minus_kf_fpa = f_linear (x_hat_kf_fpa, u_fpa (:, k)) * dt + x_hat_kf_fpa; %
Euler integration
        P_minus_kf_fpa = A * P_kf_fpa * A' + W_kf_fpa;
 % Measurement (with measurement noise)
        dv = sqrt(V_kf_fpa) * randn(size(V_kf_fpa, 1), 1);
```

```
        y(:, k+1) = h_linear(x(:, k+1)) + dv;
% Update
        innovation_kf_fpa = y (: k+1) - h_linear(x_hat_minus_kf_fpa);
        S_kf_fpa = C * P_minus_kf_fpa * C' + V_kf_fpa;
        K_kf_fpa = P_minus_kf_fpa * C' / S_kf_fpa;
        x_hat_kf_fpa = x_hat_minus_kf_fpa + K_kf_fpa * innovation_kf_fpa;
        P_kf_fpa = (eye(n_states) - K_kf_fpa * C) * P_minus_kf_fpa;
        x_hat_kf_history_fpa (, k+1) = x_hat_kf_fpa;
 % --- Control Input (using estimated state from Tuned Kalman Filter and Tuned LQR) ---
        u_fpa (:, k) = -K_lqr_fpa * x_hat_kf_fpa;
        Nbar = 1; % Placeholder for feedforward gain
        u_fpa (1, k) = u_fpa (1, k) + Nbar * ref_r (k, 3); % Apply feedforward to pitch reference
        u_fpa (2, k) = u_fpa (2, k) + Nbar * ref_r (k, 5); % Apply feedforward to tracking speed
reference
  % --- System Dynamics (with process noise) ---
        dw = sqrt(W_kf_fpa) * randn (n_states, 1);
        x (, k+1) = f_linear (x (: k), u_fpa (: k)) * dt + x (: k) + dw; % Euler integration for true
system
    end
    % --- Performance Analysis (for Wheel Speed x2 and Pitch x3) with Tuned Controller ---
    wheel_speed_output_fpa = y (2, :); % Wheel angular velocity (x2)
    pitch_output_fpa = y (3, :); % Pitch angle (x3)
    reference_wheel_speed = zeros(size(T)); % Assuming no specific reference for wheel
speed itself
    % Peak Wheel Speed
    peak_wheel_speed_fpa = max(abs(wheel_speed_output_fpa));
 % Peak Pitch Deviation (from initial condition)
    peak_pitch_deviation_fpa=max (abs (pitch_output_fpa - pitch_output_fpa(1)));
% Settling Time (Wheel Speed)
    steady_state_wheel_speed_fpa=mean(wheel_speed_output_fpa(end-round(5/dt): end));
    tolerance_wheel_speed_fpa = 0.05 * abs(steady_state_wheel_speed_fpa);
    settling_indices_wheel_speed_fpa = find (abs (wheel_speed_output_fpa -
steady_state_wheel_speed_fpa) <= tolerance_wheel_speed_fpa);
    if ~isempty(settling_indices_wheel_speed_fpa)
```

```
        settling_time_wheel_speed_fpa= T(settling_indices_wheel_speed_fpa(end));
    else
        settling_time_wheel_speed_fpa = NaN;
    end
 % Peak Overshoot (Wheel Speed)
    overshoot_wheel_speed_fpa = (max(wheel_speed_output_fpa)-
steady_state_wheel_speed_fpa) / abs(steady_state_wheel_speed_fpa) * 100;
    if isempty(overshoot_wheel_speed_fpa) || ~isfinite(overshoot_wheel_speed_fpa)
        overshoot_wheel_speed_fpa = NaN;
    end
    % Store performance data (tuned with FPA)
    performance_data_fpa(i). pitch_ref = pitch_ref_deg(i);
    performance_data_fpa(i). peak_wheel_speed = peak_wheel_speed_fpa;
    performance_data_fpa(i). peak_pitch_deviation = peak_pitch_deviation_fpa;
    performance_data_fpa(i). settling_time_wheel_speed = settling_time_wheel_speed_fpa;
    performance_data_fpa(i). peak_overshoot_wheel_speed = overshoot_wheel_speed_fpa;
  % --- Plotting All States (FPA Tuned Kalman Filter Estimates) ---
    figure (1);
    subplot(length(pitch_ref_rad), 5, (i-1)*5 + 1); plot(T, y(1, :), 'r'); hold on; plot(T, C(1,:) *
x_hat_kf_history_fpa, 'b--'); hold off; title(['Wheel Angle (x1), Ref: ',
num2str(pitch_ref_deg(i))]); legend('True', 'Est');
    subplot(length(pitch_ref_rad), 5, (i-1)*5 + 2); plot(T, y(2, :), 'r'); hold on; plot(T, C(2,:) *
x_hat_kf_history_fpa, 'b--'); hold off; title(['Wheel Ang Vel (x2), Ref: ',
num2str(pitch_ref_deg(i))]); legend('True', 'Est');
    subplot(length(pitch_ref_rad), 5, (i-1)*5 + 3); plot(T, rad2deg(y(3, :)), 'r'); hold on; plot(T,
pitch_ref_deg(i) * ones(size(T)), '--k'); plot(T, rad2deg(C(3,:) * x_hat_kf_history_fpa), 'b--');
hold off; title(['Pitch Angle (x3), Ref: ', num2str(pitch_ref_deg(i)), ' deg']); legend('True',
'Ref', 'Est');
    subplot(length(pitch_ref_rad), 5, (i-1)*5 + 4); plot(T, y(4, :), 'r'); hold on; plot(T, C(4,:) *
x_hat_kf_history_fpa, 'b--'); hold off; title(['Pitch Ang Vel (x4), Ref: ',
num2str(pitch_ref_deg(i))]); legend('True', 'Est');
    subplot(length(pitch_ref_rad), 5, (i-1)*5 + 5); plot(T, y(5, :), 'r'); hold on; plot(T, ref_r(:,
5), '--k'); plot(T, C(5,:) * x_hat_kf_history_fpa, 'b--'); hold off; title(['Tracking Ctrl (x5), Ref:
', num2str(pitch_ref_deg(i))]); legend('True', 'Ref', 'Est');
```

```
end
% sgtitle (['LQR with Kalman Filter Response for Pitch Refs: ', num2str(pitch_ref_deg)]);
 Figure (2);
for i = 1: length(pitch_ref_rad)
    subplot(length(pitch_ref_rad), 1, i);
    plot (T (1: end-1), u_fpa (1:2, 1: end-1)); % Plot both control inputs
    title (['Control Input (FPA Tuned) for Pitch Ref: ', num2str(pitch_ref_deg(i)), ' deg, Speed Ref: ', num2str(tracking_speed_ref(1))]);
    xlabel ('Time (s)');
    ylabel ('Control Effort');
    legend ('u1', 'u2'); % Add a legend for the two inputs
end
% --- Display Performance Data (Tuned with FPA) ---
disp('--- Performance Analysis (LQR with FPA Tuned Kalman Filter) ---');
for i = 1: length(performance_data_fpa)
    disp(['--- Pitch Reference: ', num2str(performance_data_fpa(i).pitch_ref), ' deg ---']);
    disp(['  Peak Wheel Speed: ', num2str(performance_data_fpa(i).peak_wheel_speed), ' rad/s']);
    disp(['  Peak Pitch Deviation: ', num2str(performance_data_fpa(i).peak_pitch_deviation), ' rad']);
    disp(['  Settling Time (Wheel Speed): ', num2str(performance_data_fpa(i).settling_time_wheel_speed), ' s']);
    disp(['  Peak Overshoot (Wheel Speed): ', num2str(performance_data_fpa(i).peak_overshoot_wheel_speed), ' %']);
    disp(' ');
end
```

5. Programme Code for LQR Tune with FPA Program for Stabilizing Two Wheeled Self-Balancing Robots.

```
% System Matrices (REVISED based on new state definition)
% x1: Wheel angle
% x2: Wheel angle velocity
% x3: Pitch angle
% x4: Pitch angular velocity
% x5: Tracking wheel speed control (desired forward/backward speed)
```

```
A = [0 1 0 0 0;
    0 -227.11 492 228 0; % Wheel dynamics influenced by pitch
    0 0 0 1 0;
    0 -47.55 163.77 47.55 0; % Pitch dynamics influenced by wheel motion
    1 0 0 0 1]; % Dynamics of the tracking speed control (may be 0 or related to a filter)
B = [0 0;
    220.5 220.5; % Control input affects wheel acceleration
    0 0;
    -46.22 -46.22; % Control input affects pitch angular acceleration
    1 0]; % Control input affects the tracking speed control
% Define system output (Assume wheel angle and pitch angle are measured)
C = [1 0 0 0 0;
    0 0 1 0 0];
D = zeros (size (C, 1), size (B, 2));
sys = ss (A, B, C, D);
 % Discretization (simple Euler - for illustration)
dt = 0.01;
Ad = eye(size(A)) + A * dt;
Bd = B * dt;
% --- LQR Design (State Feedback Gain) ---
% FPA Tuned
Q_lqr_fpa = diag ([1.6; 1.3; 125; 11.5; 56]); % FPA Tuned State Weighting
R_lqr_fpa = diag ([0.07; 0.08]); % FPA Tuned Control Weighting
[K_lqr_fpa, S_lqr_fpa, E_lqr_fpa] = lqr (A, B, Q_lqr_fpa, R_lqr_fpa);
disp('FPA Tuned LQR Gain Matrix K:');
disp(K_lqr_fpa);
% --- Kalman Filter Design and Implementation (FPA Tuned Noise Covariances) ---
% Process noise covariance (system disturbances)
W_kf_fpa = diag ([0.011; 0.008; 0.014; 0.01; 0.009]) * dt;
% Measurement noise covariance (sensor noise)
% Assume wheel angle sensor and pitch angle sensor have different noise levels
V_kf_fpa = diag ([0.005; 0.012]) * dt; % Noise on [wheel angle; pitch angle]
% Linear system dynamics function
f_linear = @(x, u) A * x + B * u;
```

```
% Measurement function (linear in this case, as C is constant)
h_linear = @(x) C * x;
% Initialize state estimate and error covariance for Kalman Filter
n_states = size (A, 1);
x_hat_kf_fpa = zeros (n_states, 1); % Initial state estimate
P_kf_fpa = eye(n_states) * 0.1; % Initial error covariance
% --- LQR Controller Implementation and Simulation with Kalman Filter ---
T = 0: dt:10;
pitch_ref_deg = [15, 43, 65]; % New pitch references in degrees
pitch_ref_rad = deg2rad(pitch_ref_deg);
tracking_speed_ref = [1, 0, -0.5]; % Example desired tracking wheel speeds
 figure;
performance_data_fpa = struct ('pitch_ref', [], 'peak_wheel_speed', [], 'peak_pitch_deviation',
[], 'settling_time_wheel_speed', [], 'peak_overshoot_wheel_speed', []);
 for i = 1: length(pitch_ref_rad)
    ref_r = zeros(length(T), size (C, 1));
    ref_r(:, 2) = pitch_ref_rad(i) * ones(length(T), 1);          % Reference for Pitch angle (now
the second output)
    % We don't have a direct reference for wheel angle in this setup
     x = zeros (n_states, length(T));
    x_hat_kf_history_fpa = zeros (n_states, length(T));
    y_measured = zeros (size (C, 1), length(T)); % Only the measured outputs
    u_fpa = zeros (size (B, 2), length(T));
    x(:, 1) = [0; 0; pi/2; 0; 0]; % Initial true state
    x_hat_kf_history_fpa(:, 1) = x_hat_kf_fpa; % Initial estimated state
  for k = 1: length(T) - 1
        % --- Kalman Filter Steps (FPA Tuned Covariances) ---
        % Prediction
        x_hat_minus_kf_fpa = f_linear (x_hat_kf_fpa, u_fpa (:,k)) * dt + x_hat_kf_fpa; % Euler
integration
        P_minus_kf_fpa = A * P_kf_fpa * A' + W_kf_fpa;
        % Measurement (with measurement noise)
        dv = sqrt(V_kf_fpa) * randn (size (V_kf_fpa, 1), 1);
        y_true = h_linear (x(:, k+1));
```

```
    y_measured(:, k+1) = y_true + dv;
 % Update
    innovation_kf_fpa = y_measured(:, k+1)- h_linear(x_hat_minus_kf_fpa);
    S_kf_fpa = C * P_minus_kf_fpa * C' + V_kf_fpa;
    K_kf_fpa = P_minus_kf_fpa * C' / S_kf_fpa;
    x_hat_kf_fpa = x_hat_minus_kf_fpa + K_kf_fpa * innovation_kf_fpa;
    P_kf_fpa = (eye(n_states) - K_kf_fpa * C) * P_minus_kf_fpa;
     x_hat_kf_history_fpa (:, k+1) = x_hat_kf_fpa;
    % --- Control Input (using estimated state from Tuned Kalman Filter and Tuned LQR) --
    u_fpa (:, k) = -K_lqr_fpa * x_hat_kf_fpa;
    Nbar = 1; % Placeholder for feedforward gain
    u_fpa (1, k) = u_fpa (1, k) + Nbar * pitch_ref_rad(i); % Apply feedforward to pitch
reference
    u_fpa (2, k) = u_fpa (2, k) + Nbar * tracking_speed_ref(1); % Apply feedforward to
tracking speed reference
 % --- System Dynamics (with process noise) ---
    dw = sqrt(W_kf_fpa) * randn (n_states, 1);
    x(:, k+1) = f_linear(x(:, k), u_fpa(:, k)) * dt + x(:, k) + dw; % Euler integration for true
system
  end
  % --- Performance Analysis (for Wheel Speed x2 and Pitch x3) with Tuned Controller ---
  wheel_speed_output_fpa = x (2, :); % True wheel angular velocity (x2)
  pitch_output_fpa = x (3, :); % True pitch angle (x3)
  reference_wheel_speed = zeros(size(T)); % Assuming no specific reference for wheel
speed itself
  % Peak Wheel Speed
  peak_wheel_speed_fpa = max(abs(wheel_speed_output_fpa));
  % Peak Pitch Deviation (from initial condition)
  peak_pitch_deviation_fpa = max (abs (pitch_output_fpa- pitch_output_fpa(1)));
  % Settling Time (Wheel Speed)
  steady_state_wheel_speed_fpa = mean(wheel_speed_output_fpa(end-round(5/dt): end));
  tolerance_wheel_speed_fpa = 0.05 * abs(steady_state_wheel_speed_fpa);
  settling_indices_wheel_speed_fpa = find (abs (wheel_speed_output_fpa -
steady_state_wheel_speed_fpa) <= tolerance_wheel_speed_fpa);
```

```
    if ~isempty(settling_indices_wheel_speed_fpa)
        settling_time_wheel_speed_fpa= T(settling_indices_wheel_speed_fpa(end))
    else
        settling_time_wheel_speed_fpa = NaN;
    end
     % Peak Overshoot (Wheel Speed)
    overshoot_wheel_speed_fpa = (max(wheel_speed_output_fpa) - steady_state_wheel_speed_fpa) / abs(steady_state_wheel_speed_fpa) * 100;
    if isempty(overshoot_wheel_speed_fpa) || ~isfinite(overshoot_wheel_speed_fpa)
        overshoot_wheel_speed_fpa = NaN;
    end
    % Store performance data (tuned with FPA)
    performance_data_fpa(i).pitch_ref = pitch_ref_deg(i);
    performance_data_fpa(i). peak_wheel_speed = peak_wheel_speed_fpa;
    performance_data_fpa(i).peak_pitch_deviation = peak_pitch_deviation_fpa;
    performance_data_fpa(i).settling_time_wheel_speed = settling_time_wheel_speed_fpa;
    performance_data_fpa(i). peak_overshoot_wheel_speed = overshoot_wheel_speed_fpa;
    % --- Plotting States and Estimates (FPA Tuned Kalman Filter) ---
    Figure (1);
    subplot(length(pitch_ref_rad), 2, (i-1) *2 + 1);
    plot (T, x (1, :), 'r'); hold on; plot (T, C (1, :) * x_hat_kf_history_fpa, 'b--'); hold off;
    title (['Wheel Angle (rad), Ref: ', num2str(pitch_ref_deg(i))]); legend ('True', 'Est');
    subplot(length(pitch_ref_rad), 2, (i-1) *2 + 2);
    plot (T, x (3, :), 'r'); hold on; plot (T, pitch_ref_rad(i) * ones(size(T)), '--k'); plot (T, C (2, :) * x_hat_kf_history_fpa, 'b--'); hold off;
    title (['Pitch Angle (rad), Ref: ', num2str(pitch_ref_deg(i)), ' deg']); legend ('True', 'Ref', 'Est');
    figure (2);
    subplot(length(pitch_ref_rad), 1, i);
    plot (T (1: end-1), u_fpa (1:2, 1: end-1)); % Plot both control inputs
    title (['Control Input (FPA Tuned) for Pitch Ref: ', num2str(pitch_ref_deg(i)), ' deg, Speed Ref: ', num2str (tracking_speed_ref (1))]);
    xlabel ('Time (s)');
    ylabel ('Control Effort');
```

```
    legend ('u1', 'u2'); % Add a legend for the two inputs
end
% sgtitle ('State Estimation with Kalman Filter (FPA Tuned)');
% --- Display Performance Data (Tuned with FPA) ---
disp('--- Performance Analysis (LQR with FPA Tuned Kalman Filter) ---');
for i = 1: length(performance_data_fpa)
    disp (['--- Pitch Reference: ', num2str(performance_data_fpa(i). pitch_ref), ' deg ---']);
    disp([' Peak Wheel Speed: ', num2str(performance_data_fpa(i).peak_wheel_speed), ' rad/s']);
    disp([' Peak Pitch Deviation: ', num2str(performance_data_fpa(i).peak_pitch_deviation), ' rad']);
    disp([' Settling Time (Wheel Speed): ', num2str(performance_data_fpa(i).settling_time_wheel_speed), ' s']);
    disp (['Peak Overshoot (Wheel Speed): ', num2str(performance_data_fpa(i.peak_overshoot_wheel_speed), ' %']);
    disp (' ');
end
```