Nonlinear Optimal Control and Its Application to a Two-Wheeled Robot

Surapong Kokkrathoke

A thesis submitted in partial fulfilment of the requirements of

Sheffield Hallam University

for the degree of Doctor of Philosophy

November 2021

Abstract

This research studies two advanced nonlinear optimal control techniques, i.e., the freezing control and the iteration scheme, and their associated applications, such as a single inverted pendulum (IP) on a cart system and a two-wheeled robot (TWR) system. These techniques are applied to stabilise the highly unstable nonlinear systems in the vertical upright position when facing different initial pitch angles. Different linear optimal controllers (linear quadratic regulator and linear quadratic Gaussian) and nonlinear optimal controllers are designed and applied to the models for concurrent control of all state variables. The controlled systems are tested in simulation and the best performing control design is eventually implemented on a robot prototype built with an educational kit – the LEGO EV3, after practical factors such as motor voltage limitation, gyro sensor drift and model uncertainties have been considered, analysed and dealt with.

Simulations and experiments on the TWR robot prototype demonstrate the superiority of the nonlinear freezing optimal control technique, showing larger operation ranges of the robot pitch angle and better response performances (i.e., shorter rise time, less overshoot and reduced settling time) than the linear optimal control methods. In particular, a novel mixing method to create a new nonlinear model (Model AB) from two different models on the same physical prototype with an increased controllable region of the TWR system is introduced, for the first time, for the calculations of optimal feedback gains for the system. Significantly, the utilisation of this mixed model, combined with the nonlinear freezing controller, achieves true global control of the TWR, even from an initial pitch angle of 90° (i.e., the horizontal position), when a motor with a saturated voltage of 48V and nominal torque of 298 mNm is adopted in simulation tests. This is wider than the angle achievable from the primary model (Model A) and any other single feedback control method on TWR reported in the literature. Robustness tests when introducing model uncertainties by adding mass and height on the TWR also illustrate excellent control performances from the nonlinear optimal control in both simulations and hardware implementations.

Declaration

I hereby declare that:

1. I have not been enrolled for another award of the University, or other academic or professional organisation, whilst undertaking my research degree.
2. None of the material contained in the thesis has been used in any other submission for an academic award.
3. I am aware of and understand the University's policy on plagiarism and certify that this thesis is my own work. The use of all published or other sources of material consulted have been properly and fully acknowledged.
4. The work undertaken towards the thesis has been conducted in accordance with the SHU Principles of Integrity in Research and the SHU Research Ethics Policy.
5. The word count of the thesis is 32,693.

Name	*Surapong Kokkrathoke*
Date	*November 2021*
Award	*PhD*
Faculty	*Department of Engineering and Mathematics*
Director(s) of Studies	*Dr Xu Xu(DoS), Dr Alex Shenfield and Prof. Ian Halliday*

Acknowledgement

First and foremost, I would like to express my great thanks to my principal supervisor, Dr Xu Xu, for her invaluable advice and enthusiastic encouragement during the research study. Also, I am thankful for her kindness, concerns and suggestions while living out of my home country. Moreover, I would like to thank other supervisors, Dr Alex Shenfield and Professor Ian Halliday, and the university staff for their support.

Furthermore, I would like to thank the Royal Thai Government and the Synchrotron Light Research Institute (public organization), my workplace, for the doctoral's degree scholarship. This research would have been impossible without their support.

Special thanks to my beloved wife, Tipsuda, who is alongside me on this journey and my life. Thank you for your encouragement. I would like to extend my thanks to our families for their support and for looking after us. Finally, I also would like to thank my grandfather for spiration and waiting for me back to our home country after finishing my doctoral study.

Table of Contents

List of Tables

List of Figures

Nomenclature

Variable	Description	Units
	IP system	
m_1	Mass of cart	kg
m_2	Mass of pendulum	kg
r	Length of the pendulum	m
x_1	Cart displacement	m
$\dot{x}_1$ and x_2	Cart velocity	m/s
$\ddot{x}_1$and $\dot{x}_2$	Cart acceleration	m^2/s
x_3 and θ	Pendulum angle	rad
$\dot{x}_3$ and x_4	Pendulum angular velocity	rad/s
$\dot{x}_4$ and $\ddot{\theta}$	Pendulum angular acceleration	rad^2/s
u	Control input	N
f	Generalised force	N
	TWR system	
PWM	Pulse-width modulation	%
$V_{battery}$	Battery voltage	V
V_{cal}	Voltage coefficient for converting battery voltage to pulse-width modulation	V
m	Mass of robot's wheel	m
M	Mass of robot's body	m
R	Robot's wheel radius	m
W	Robot's body width	m
D	Robot's body depth	m
H	Robot's body height	m

L	Distance between wheel axle and centre of robot	m
J_w	Inertia moment of robot's wheel	kgm^2
J_ψ	Inertia moment of robot pitch	kgm^2
J_ϕ	Inertia moment of robot yaw	kgm^2
J_m	Inertia moment of DC motor	kgm^2
R_m	Resistance of DC motor	Ω
K_b	Back EMF constant of DC motor	$V \cdot sec/rad$
K_t	Torque constant of DC motor	$N.m/A$
n	Gear ratio	-
f_m	Coefficient of friction between robot and DC motor	-
f_w	Coefficient of friction between wheel and floor	-
ψ and x_3	Robot pitch angle	rad
$\dot{\psi}$ and x_4	Robot pitch angular velocity	rad/s
$\ddot{\psi}$ and $\dot{x}_4$	Robot pitch angular accelerator	rad^2/s
ϕ and x_5	Robot yaw angle	rad
$\dot{\phi}$ and x_6	Robot yaw angular velocity	rad/s
$\ddot{\phi}$ and $\dot{x}_6$	Robot yaw angular accelerator	rad^2/s
θ and x_1	Wheel angle	rad
$\dot{\theta}$ and x_2	Wheel angular velocity	rad/s
$\ddot{\theta}$ and $\dot{x}_2$	Wheel angular accelerator	rad^2/s
θ_l	Left wheel angle	rad
θ_r	Right wheel angle	rad
z_b	Z-axis of robot's body	m
z_m	Z-axis of robot's wheel	m
y_l	Y-axis of robot's left wheel	m

y_r	Y-axis of robot's left wheel	m
y_b	Y-axis of robot's body	m
y_m	Z-axis of between two wheels	m
x_l	X-axis of robot's left wheel	m
x_r	X-axis of robot's left wheel	m
x_b	X-axis of robot's body	m
x_m	X-axis of between two wheels	m
F_θ	Force of wheel angle	N
F_ϕ	Force of yaw angle	N
F_ψ	Force of pitch angle	N
$v_{l,r}$	Right and left voltages of DC motor	V
v_l	Left voltage of DC motor	V
v_r	Right voltage of DC motor	V
$i_{l,r}$	Current of DC motor	A
L_m	Inductance of DC motor	H
α and β	Coefficients of motor voltage and force	-

Others

L	The Lagrangian	$N.m$
V	Total potential energy	$N.m$
T	Total kinetic energy	$N.m$
g	Acceleration due to gravity	m^2/s
λ	Limitation of control signal saturation	V
v and $\phi(x_n + 1)$	Voltage input saturation	V
t_f	The final time	Sec
J	Cost function value	-
Q	States weight matrix	-
R	Control weight matrix	-

A	System matrix	-
B	Control matrix	-
C	Output matrix	-
K	Optimal feedback gains	-
P	The solution of the algebraic matrix Riccati equation	-
$\mathcal{C}$	Controllability matrix	-
$\mathcal{O}$	Observability matrix	-
I and G	Identity matrix	
w_n	Process noise of the Kalman filter	-
v_n	Measurement noise of the Kalman filter	-
Q_K	Process noise matrix	-
R_K	Measurement noise matrix	-
K_f	The Kalman filter gain	-
$\hat{x}$	State estimation of state variable x	-
w	Artificial control signal	-
Q_a	State weight matrix of voltage saturation input	-
F	The final time penalty matrix	-
i	Iteration number	-

Abbreviations

ANN	Artificial neural network
CAD	Computer-aided design
CMG	Control moment gyroscope
CPU	Central processing unit
DBD	Delta-bar-delta algorithm
DC	Direct currrent
DOF	Degrees of freedom
EKF	Extended Kalman filter
EMF	Electric and magnetic fields
Eq	Equation
FC-PDPS	Fuzzy controller with parallel distributed pole assignment scheme
FSUC	Fuzzy swing-up controller
GPC	Generalized predictive control
GPS	Global positioning system
INS	Inertial navigation system
IP	Inverted pendulum
LQG	Lnear quadratic Gaussian
LQR	Linear quadratic regulator
LTV	Linear, time-varying
MEMS	Micro-electromechanical system
MPC	Model predictive control
NASA	The National Aeronautics and Space Administration
ODEs	Ordinary differential equations
PD	Proportional derivative
PID	Proportional-integral-derivative
PWM	Pulse-width modulation
RBFNNs	Radial basis function neural networks
RCX	Robotic Command eXplorers
RIS	Robotics invention system
SDRE	State-dependent Riccati equation

SISO	Single-input single-out
TWD	Threshold with delay
TWR	Two-wheeled robot
T-S	Takagi–Sugeno
UAV	Unmanned aerial vehicle
USB	Universal serial bus
WFLC	Weighted-frequency Fourier Linear Combiner

Chapter 1

Introduction

1.1 Background

During the past decade, various existing linear controls have been applied to regulate linear and nonlinear systems. Some of the linear control methods are well-known, for instance, proportional-integral-derivative (PID) control, Linear Quadratic Regulator (LQR) and linear quadratic Gaussian (LQG). These techniques use local linearisation around the small neighbourhood of an equilibrium of the system, which limits their operational range and therefore their applications and performances. By contrast, there are some existing nonlinear techniques that illustrate outstanding control outcomes for very nonlinear systems, because the system can be controlled globally using its nonlinear system models**.**

To begin with, Banks and Mhana (1992) first introduced a nonlinear freezing control technique by extending the LQR theory to control nonlinear systems in the form of

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x}, \boldsymbol{u})\boldsymbol{x} + \boldsymbol{B}(\boldsymbol{x}, \boldsymbol{u})\boldsymbol{u} \qquad (1.1)$$

where $\boldsymbol{x}$ is a state variable vector, $\boldsymbol{A}(\boldsymbol{x})$ and $\boldsymbol{B}(\boldsymbol{x})$ represent the nonlinear system matrices which form controllability matrices and $\boldsymbol{u}$ is the nonlinear optimal control.

This method can be generalised to more complicated nonlinear systems, e.g. optimal altitude control for a single inverted pendulum on a cart (Harrison, 2003),

F-8 crusader (Çimen & Banks, 2004a), and a double inverted pendulum on a cart (Xu, Zhang, & Carbone, 2017) etc.

Furthermore, another nonlinear control method, an iteration scheme, was created by Banks and McCaffrey (1998), which introduced linear, time-varying (LTV) approximations to the infinite-time horizon nonlinear optimal affine control problem. There has been a wide range of applications using this technique, for example, super-tankers autopilot design (ÇImen & Banks, 2004b), optimal altitude control for spacecraft (Zheng, Banks, & Alleyne, 2005), optimal drug therapy control in cancer treatment (Itik, Salamci, & Banks, 2009), velocity tracking in a hydraulic press (Du, Xu, Banks, & Wu, 2009) and the dynamics of a tunnel diode oscillator (Itik, 2016).

Of interest here are the characteristics of nonlinear systems and how different control techniques can be applied to nonlinear systems for control design in software simulation and hardware implementation. Moreover, these techniques can be applied to nonlinear systems (for instance, synchrotron light orbit stability) within the Synchrotron Light Research Institute (Thailand), which provided the scholarship for this PhD research study.

1.2 Aims and Objectives

The project aims to study advanced nonlinear control techniques, mathematical modelling and signal filtering estimation of various sensors, applied to a self-balancing two-wheeled robot (a classical system based on inverted pendulum theory, centred on system stability). In particular, these techniques can be applied

to any nonlinear systems directly, without the need to linearise systems locally around the equilibrium points.

The objectives of this project and the steps in achieving them are as follows:

1. Develop nonlinear models of a two-wheeled robot from the principle of the single inverted pendulum methods and analyse the non-unique mathematical models in the correct pseudo-linear (nonlinear) forms.

2. Apply classical linear control (e.g., LQR and LQG) and advanced nonlinear controls (e.g., freezing control technique and the iteration scheme) theoretically to the two-wheeled robot models for optimal control designs and develop simulations of the control systems in MATLAB to verify the theoretical results.

3. Conduct controllability tests on different nonlinear models of the two-wheeled robot to create a larger controllable range and new capability.

4. Implement suitable control techniques to a two-wheeled robot prototype (built with Lego Mindstorms EV3), taking into account of physical factors and practical conditions. Then, summarise advantages and disadvantages of the advanced nonlinear control techniques against the linear control strategies.

5. Design effective signal filtering estimation (e.g., Kalman filter estimation) to support the self-balancing control of the two-wheeled robot.

1.3 Publications and Presentations Resulted from This PhD Study

- Journal paper

- Kokkrathoke, S., Rawsthorne, A., Zhang, H., and Xu, X., (in press). Nonlinear Optimal Stabilising Control of a Two-wheel Robot. *International Journal of Modelling, Identification and Control*.

- Conference papers

- Kokkrathoke, S., & Xu, X. (2021). Implementation of Nonlinear Optimal Control of Two-wheel Robot with Extended Kalman Filter. *2021 IEEE International Conference on Automatic Control & Intelligent Systems (I2CACIS).* Shah Alam, Malaysia. (pp.19-25). IEEE.
- Kokkrathoke, S., & Xu, X. Controllability Study of Two-Wheel Robot for Nonlinear Optimal Control and Implementation [Manuscript accepted for publication and presentation]. *2021 IEEE Conference on Systems, Process & Control (ICSPC2021).* Shah Alam, Malaysia.

- Presentations

- Kokkrathoke, S., (2018, December). Supervisors: Xu, X., Halliday, I. and Shenfield, A. Nonlinear Freezing Control of Inverted Pendulum. [Poster presentation]. Winter Poster Event 2018, Sheffield Hallam University.
- Kokkrathoke, S., (2019, May). Supervisors: Xu, X., Halliday, I. and Shenfield, A. Nonlinear Freezing Control of Inverted Pendulum and Cart System. [Poster presentation]. Materials and Engineering Research Institute (MERI) Research Symposium 2019, Sheffield Hallam University.
- Kokkrathoke, S., (2019, December). Supervisors: Xu, X., Halliday, I. and Shenfield, A. Nonlinear Control Design and Implementation of Self-Balancing Lego Robot. [Poster presentation]. BMRC & MERI Winter Poster Event 2019, Sheffield Hallam University.
- Kokkrathoke, S., (2020, December). Supervisors: Xu, X., Halliday, I. and Shenfield, A. Nonlinear Optimal Control of Self-Balancing Two-Wheel Robot. [Poster presentation]. I2RI Winter Poster Event 2019, Sheffield Hallam University.
- Kokkrathoke, S., (2021, June). Supervisors: Xu, X., Halliday, I. and Shenfield, A. Tracking and Balancing Control of the LEGO Two-Wheel Robot with Extended Kalman Filter. [Oral presentation]. Materials and Engineering Research Institute (MERI) Research Symposium 2021, Sheffield Hallam University.

1.4 Thesis Structure

Chapter 1: Introduction

This chapter presents an overview of the PhD research, of which the aim is to study advanced nonlinear control methods in theoretical design, simulation and implementation, and compare them against linear ones, by applying all methods to the single inverted pendulum and self-balancing two-wheeled robot systems. Furthermore, the research outcomes, such as paper publications and presentations, are listed in this chapter.

Chapter 2: Literature Reviews

This chapter reviews existing research work, relating to inverted pendulum systems, such as single and multiple inverted pendulums on a cart, rotary inverted pendulums, two-wheeled scooters, and self-balancing two-wheeled robots. Furthermore, the results of applying various linear controllers, e.g., PID, LQR, LQG, Fuzzy logic and model predictive control (MPC) to inverted pendulum systems are investigated. Additionally, prominent research work and development in nonlinear controls, e.g., freezing control, iteration scheme, sliding-mode control and neural network, applied to inverted pendulum and two-wheeled robot, are also examined in this chapter.

Chapter 3: Hardware and Software Descriptions of a Self-Balancing Robot

This chapter illustrates the history of LEGO robots hardware and software. LEGO Mindstorms EV3 is used as a prototype of the self-balancing robot to test different control designs, in this research. Moreover, the specification of LEGO Mindstorms EV3 is provided.

Chapter 4: Modelling of Inverted Pendulum and Two-Wheeled Robot Systems

This chapter analyses the single inverted pendulum equations of motion, converting them to mathematical models using the Lagrangian method. Likewise, the dynamic system of the LEGO Mindstorms EV3 robot is transformed into state-space representation. Significantly, for the LEGO EV3 model, the control input in terms of force is converted to motor voltages for practical considerations.

Chapter 5: Linear Control Designs and Implementations

This chapter presents and analyses the capability of two linear control techniques. Firstly, a LQR controller is applied to stabilise an inverted pendulum on a cart model, a two-wheeled robot model and a robot prototype. Secondly, the LQG controller is utilised on the two-wheeled robot model and prototype to provide state estimation and deal with a challenging sensor drift issue, from the LEGO EV3 robot prototype.

Chapter 6: Nonlinear Control Designs and Implementations

This chapter introduces three nonlinear control methods: an iteration (or LTV approximation) scheme, a freezing optimal control, and the freezing control with extended Kalman filter, applied to the inverted pendulum and two-wheeled robot. Before simulating these models, controllability tests are performed and analysed, which leads to an investigation of the effect of different state-space models on controllability; especially, when these models are created from the same set of 1st order dynamical ordinary differential equations (ODEs). Furthermore, the experimentations on a practical two-wheeled robot are examined and discussed, where the results of stabilising the system are obtained by linear and nonlinear techniques.

Chapter 7: Conclusion

This chapter summarises the comparison of performances between linear and nonlinear controls approach on the classical benchmark models, the single inverted pendulum, and the self-balancing two-wheeled robot. Furthermore, the contributions to existing knowledge are presented in this chapter.

Chapter 2

Literature Review

2.1 Introduction

The literature review for this project can be divided into four categories. The first section illustrates the various applications based on inverted pendulum control theory. Secondly, different types of system modelling used to simulate the self-balancing robots are introduced. Then, a representative selection of linear control techniques for self-balancing robot and others are discussed. Finally, the widely used nonlinear control methods and their applications are reviewed.

2.2 Inverted Pendulum System and Applications

To begin with, an inverted pendulum is a classical benchmarking tool for studying feedback control by mounting the pendulum on a cart, which can move horizontally to balance the pendulum in the vertical upright position, as presented in Figure 2.1. However, the pendulum is unstable without control, it will fall over; therefore, the feedback controller is needed. Furthermore, the advantage of studying the inverted pendulum model is that it is convenient to design the system modelling in several forms, such as, Newton's equation, Lagrangian method, State-Space Modelling and physical CAD modelling, etc. Moreover, many applications are built from the inverted pendulum model, e.g., rotatory inverted pendulums, two-wheeled robots, two-wheeled vehicles and self-balancing

bicycles. As the inverted pendulum is a nonlinear and unstable system, controlling it outside the traditionally linearised range is a challenge.

The single inverted pendulum on a cart is a basic model to research, as shown in Figure 2.2 (left). Additionally, a more complicated model than the single inverted pendulum is known as multi-link inverted pendulum (i.e., double link or high-link), as presented in Figure 2.2 (right). Both models are stabilised by force u in the horizontal axis to maintain all rods in the vertical upright position, which means all pitch angles θ equal to 0°.

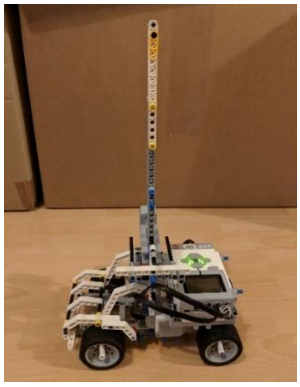


Figure 2.1: The single inverted pendulum on a cart (LEGO Mindstorms EV3)

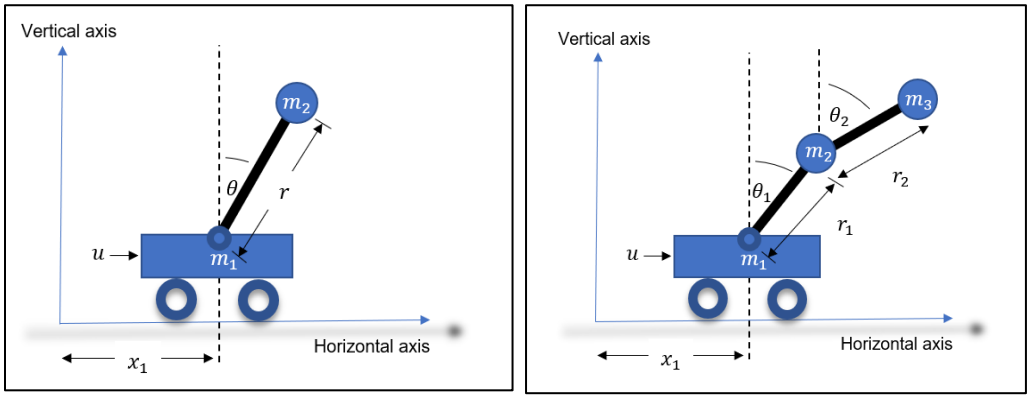


Figure 2.2: Single (left) and multi-link (right) inverted pendulum on cart (Xu, Zhang, & Carbone, 2017).

For instance, the simulation of single inverted pendulum on a cart control was presented in Alkamachi (2020), Tao et al. (2008) and Banks & Dinesh (2000) and Harrison (2003) with different control techniques. Moreover, a double inverted pendulum on a cart was modelled and analysed by Xu, Zhang, & Carbone (2017). In particular, the physical implementation of the single inverted pendulum on a cart was realised using the LEGO EV3 robot introduced by Xu, Zhang, & Carbone (2017) with a nonlinear freezing control technique.

Furthermore, the balancing theory of inverted pendulum has been applied to the Furuta pendulum (or known as the rotatory inverted pendulum), as shown in Figure 2.3. However, the cart is transformed into a fixed base and is stabilising the inverted pendulum by a driven arm in the horizontal axis.

Figure 2.3: Rotary Inverted Pendulum (Quanser, 2020)

The simulation results of the rotatory inverted pendulum can be found in Zabihifar et al. (2020). The authors applied the adaptive neural network control to the Furuta pendulum CAD model (the performance will be discussed in Section 2.3). Moreover, the implementations of rotatory inverted pendulum are presented by many researchers, for instance, by Seman et al. (2013), and Aranda-Escolástico (2016) (the approaches will be detailed in Section 2.4).

Additionally, the inverted pendulum theory has been investigated and analysed for two-wheeled robot systems, as shown in Figure 2.4; therefore, the pendulum with four-wheeled cart system has been transformed to two-wheeled instead. The torque of motors from two wheels is used to balance the pendulum or the robot's body. The angle between the robot's body and the robot's balancing point in the vertical upright position is called the pitch angle. Likewise, the robot's rotation angle in the horizontal axis is known as the yaw angle. For instance, Grasser et al. (2002) introduced the prototype of a two-wheeled mobile inverted pendulum known as JOE. The two state-space controllers with pole placement were applied to stabilise the system. Moreover, Yamamoto (2009) presented the self-balancing two-wheeled robot from the LEGO NXT, which the LQR technique was selected to balance the system. Yamamoto (2009) presented better performance with LQR than in Grasser et al. (2002) with pole placement when implementing on the two-wheeled robot for position tracking. The robot pitch angle of LEGO NXT with LQR control had slight oscillation between $\pm 6°$, using the linear optimal control technique, before moving to track the reference; by contrast, the JOE robot showed more than doubling of pitch angle swing, between $\pm 14°$, for the same distance of tracking implementation. Similarly, other researchers studying the self-balancing two-wheeled robot also applied the inverted pendulum theory; for example, in Ahn & Jung (2014) and da Silva & Sup (2017) with linear control, and also Jung & Kim (2008) and Cruz, García, & Bandala (2016) with nonlinear control. Analysis and comparisons of the techniques used and the associated performances will be given in Sections 2.4 and 2.5.

Figure 2.4: Self-balancing two-wheeled robot (LEGO Mindstorms EV3) (LEGO, 2021)

Moreover, the self-balancing two-wheeled robot has been developed to use in personal transportation, as shown in Figure 2.5. This is because these vehicles are compact, easy to manoeuvre and comfortable for travel on either smooth or bumpy surfaces. The system models of two-wheeled vehicles are similar to the two-wheeled robot which are also based on the inverted pendulum theory. For instance, Tsai et al. (2010) presented a self-balancing two-wheeled scooter and applied adaptive neural network control to stabilise the system. Similarly, a seated transportation two-wheel vehicle was introduced by Kim & Jung (2016). The vehicle was controlled by one PD (Proportional-derivative) control, and two PID (proportional-integral-derivative) controls. More details will be given in Section 2.4.

Figure 2.5: Segway PT (Personal Transporter) (Segway, 2021)

Likewise, the theory of inverted pendulum has been applied to self-balancing bicycles. The gyroscopic effect or control moment gyroscope (CMG) method is utilised to stabilise bicycles in the upright position, as presented in Figure 2.6, using one or more flywheels, depending on the control design. The simulation of self-balancing bicycle was introduced by Chu & Chen (2017). The system model combined a bicycle with two flywheels and the model predictive control (MPC) was selected to stabilise the system. The control design and performance of this application will be discussed in Section 2.4.

Figure 2.6: Jyrobike - Auto Balance Bicycle (Kickstarter, 2021)

2.3 System Modelling

Mathematical models of single and multiple inverted pendulum on a cart were considered in Xu, Zhang, & Carbone (2017), and a self-balancing two-wheeled robot was investigated in Yamamoto (2009). These models have been used in almost all research work in this area. They can be created from system analysis, based on the First Principles modelling method, to derive mathematical equations with linear or nonlinear characteristics (according to the specific system considered). In particular, mathematical equations represent physical

phenomena and can be obtained using Newton's law of motion (as shown in Banks & Dinesh (2000) and Grasser et al. (2002)). Additionally, there is another method for creating dynamic models: the Euler–Lagrange formulation, based on a system's kinetic and potential energy, see e.g. Harrison (2003), Yamamoto (2009) and Xu, Zhang, & Carbone (2017). The benefit of the Euler–Lagrange approach is its simplicity in summarising energy terms; therefore, force analysis and the associated force directions in the Newton method is not necessary. On the other hand, the complicated partial derivative calculation of Lagrange's equation when the system is complex and of high order is a disadvantage. These dynamical equations from both techniques can be transformed into a state-space form, which is useful for simulation and internal monitoring purposes. The state space approach also generally has the advantages of being applicable to nonlinear and time-variant, or multiple-input-multiple-output, or multivariable systems, over the traditional frequency domain models.

In other literature, such as Kharola & Patil (2017a), appropriate dynamic equations are obtained and then simulated using Simulink block functions instead of a state-space model to stabilise one wheeled mobile robot, because the Simulink block functions can be applied for the dynamic equations directly. The linearisation process is not needed because nonlinear equations can be utilised straightforwardly in implementation. In particular, a nonlinear control method was used in this research, which is a neural network control system. Therefore, this technique is suitable for researchers who require a nonlinear model but do not require coding using programming language such as MATLAB script file or C-programming language etc. However, this technique has not been widely used,

as it is an inconvenience to wire Simulink block functions when there are many variables in the nonlinear system modelling equations. The block functions are complexly wired and therefore this process is not as neat and efficient as the alternatives.

On the other hand, some researchers have not established mathematical dynamical equations, rather, they have used physical software model instead; for instance, the single inverted pendulum based on track model presented by Alkamachi (2020). The author applied SolidWorks CAD design imported to Simulink Simscape tool in the MATLAB program. This extension tool was formerly known as SimMechanics, which was used to simulate mechanical systems. The inverted pendulum model was investigated with the state feedback control to balance the system. Moreover, another physical software model, namely MSC Nastran was introduced by Ahmad & Siddique (2011). The authors presented a fuzzy logic control to stabilise the visual model of the two-wheelchair in Simulink. The software modelling is applied to analyse the structure of applications, known as Visual Nastran 4D. It has been developed by the MSC software (MacNeal-Schwendler Corporation) company, which was previously named as Nastran (NASA Structural Analysis) software. The MSC company granted the first contract in this software to NASA in 1965 (MSCsoftware, 2020). The advantage of Visual Nastran 4D programme over the SolidWorks CAD design, presented by Alkamachi (2020), is that the Simscape transformation is not needed as researchers can import the dynamic model from Visual Nastran 4D to Simulink directly. Furthermore, Zabihifar et al. (2020) also presented an adaptive neural network to stabilise a rotary inverted pendulum model, using the multibody

dynamics software, namely ADAMS, which is a product of MSC software company, similar to the one used in Ahmad & Siddique (2011). The ADAMS software is similar to Visual Nastran 4D in term of multibody simulation; however, there are a few differences between the two software: Visual Nastran 4D is can model more structure details, e.g., strength and stiffness. The systems which have already created 3D CAD models can gain advantage from this physical software model by combining with the dynamic equations directly; in particular, the linearisation is not necessary for these system modelling.

2.4 Linear Control Techniques

Although almost all systems in reality are nonlinear, linear control techniques are often used to regulate both linear and nonlinear systems. This is because linear control theory is more mature and much more well understood. Before being able to apply linear control techniques to a nonlinear system, linearisation is performed to approximate the nonlinear system to a linear one. After the systems are modelled using dynamic differential equations, they can be linearised around an equilibrium (operating point). Around this operating point, Taylor series are often applied to expand the equations with small deviations that gives the required linear system approximation (Dutton, Thompson, & Barraclough, 1997). This subsection reviews different linear control methods, including PID control, LQR control, LQG control, fuzzy control and model predictive control.

2.4.1 PID Controller

Firstly, the well-known classical linear control method is the three-term controller or Proportional–Integral–Derivative (PID) controller. It has been used in many industrial applications for approximately a century because of its simplicity for understanding and implementation. It is also used as a standard or fundamental linear control technique to benchmark control performances when compared against others. PID control has been used to control many self-balancing two-wheeled robots, e.g. a service two-wheel robot with two arms (Ahn & Jung, 2014) and a two-wheel chair robot (Kim & Jung, 2016).

Ahn & Jung (2014) presented two PID controllers, controlling the orientation angle (yaw angle) and the robot position (displacement), whilst another PD controller is used to balance the robot's pitch angle. The authors applied the three (2 PID + 1 PD) controllers to stabilise the system separately, but the three variables are coupled in the dynamics and therefore affect the control design. In tuning the PID gain parameters, when one parameter was changed, the performance of other variables was also affected meaning all other PID parameters also need to be re-tuned. That is one disadvantage of the PID control technique. Moreover, the disturbing external force at robot's body was limited to a small range, within approximately ± 10N, as linear controllers were applied to stabilise the robot; therefore, the pitch angle of balancing robot was restricted, which was presented at a narrow-angle range of $\pm 2^{\circ}$.

Regarding the two-wheel chair robot (Kim & Jung, 2016), the authors applied the three linear PID controllers from Ahn & Jung (2014) to the personal vehicle

transportation, namely TransBOT. The vehicle presented self-balancing with the passenger on board. Nevertheless, the pitch angle was balanced within approximately $\pm 5^{\circ}$, which was not too far from the two-wheel robot (Ahn & Jung, 2014). Noticeably, different weights of the riders would affect the PID gain parameters to stabilise the overall system. Therefore, the authors prepared various PID gains in the two PID controllers (pitch angle and robot position) to take into account of passengers' weight range. In the next section, the use of LQR control technique, where one controller can control multiple variables in the generalised coordinates of the system, will be discussed.

2.4.2 Linear Quadratic Regulator

Secondly, a more complex linear controller applied to self-balancing two-wheeled robots is the linear optimal control. This technique (also known as linear quadratic regulator (LQR)) optimises the system to achieve the best possible performance using a mathematical algorithm which minimises a quadratic performance index (cost function) with feedback controller (Dutton, Thompson, & Barraclough, 1997).

For instance, the linear optimal control was applied to a self-balancing two-wheeled robot, namely NXTway-GS robot (Yamamoto, 2009). The LEGO robot was controlled in three generalised coordinates, including the wheel angle, the body pitch angle and yaw angle. These coordinates were analysed to create a nonlinear system model. Then, linearisation was applied to the model for generating feedback gains in the LQR control system. The pitch and yaw angles were controlled by the same LQR feedback gain, but the wheel angle was

controlled through a separate feedback gain in order to check the robot's position. Furthermore, a tracking system was designed to enhance the control performance to include robot displacement. The results of both simulation and implementation in Yamamoto (2009) showed the maximum pitch angle of the two-wheeled robot to be balanced was at approximately $\pm 6^{\circ}$, and the robot position tracking design functioned well.

Another research of linear LQR control related to self-balancing robot was conducted by da Silva & Sup (2017). The authors developed an adult walking aid with handlebar based on the two-wheeled robot. The LQR control design was similar to Yamamoto (2009); however, there were two tracking systems used to control the robot's displacement and robot's heading angle (acting at the handlebar angle in the yaw axis). The authors designed an LQR feedback controller for both the pitch angle and the robot displacement, and the yaw angle was controlled by a separate LQR feedback controller. The maximum pitch angle for stabilisation achieved in da Silva & Sup (2017) was similar to other linear controls, at approximately $\pm 6^{\circ}$. Significantly, the researches of Yamamoto (2009) and da Silva & Sup (2017) were based on the LQR method. They applied an LQR feedback controller from multiple outputs; on the other hand, the PID control technique presented by Ahn & Jung (2014) and Kim & Jung (2016) was only designed to control one variable at a time.

2.4.3 Linear Quadratic Gaussian

Furthermore, many researchers have experienced issues such as noise, vibration, or signals drift of measurement tools during physical implementations

of their control designs. To deal with these issues, a state-space feedback control system can be developed to supervise the inaccurate signal outputs by combining with a Kalman filter (Kalman & Bucy, 1961). The Kalman filter is used to estimate the state variables for the purpose of feedback, utilising measurements made at the inputs and outputs, and the plant is required to be observable.

In Hanselmann & Engelke (1988), the authors illustrated the implementation of a hard disk head position, controlled by the linear quadratic Gaussian technique (LQG). The method combined the classical LQR control with the Kalman filter; therefore, the Kalman technique was applied to estimate the position of magnetic disk heads on desired tracks at the high frequency.

Similarly, Chang & Liu (2007) introduced a vibration absorber control in optical disk drive using the LQG method. The active vibration absorber was controlled by a state feedback control method, namely LQR, with the assistance from the Kalman filter, presenting vibration amplitudes reduction of up to 50% in the implementation.

Moreover, the Kalman filter has been applied to overcome sensor drift issues. For instance, gyro drift correction was implemented in a head controlled mouse for disabled people who were unable to use a standard computer mouse, as presented by Du et al. (2017). The authors applied the Kalman filter with the Weighted-frequency Fourier Linear Combiner (WFLC) and the threshold with delay (TWD) control. Note, this review is to concentrate on gyro drift issue, which is in the MEMS-gyroscope sensor, based on advantages of the Kalman filter; therefore, control methods used in the research of Du et al. (2017) are neglected.

It was shown that the gyro drift problem in gyroscopic head-borne mouse was significantly reduced once the signals were processed by the Kalman filter. For example, the implementation of filtering signal in the micro-electromechanical system(MEMS)-gyroscope model MLX90609 reduced inaccurate angular position measurements. Before the filtering process, the signal was drifting over -20° from the reference position at the 80th second; however, when the Kalman filter was applied, the sensor drift problem was eliminated during the same period of time. The authors suggested that one Kalman filter on its own could only reduce the noise and sensor drift, but not eliminate them. However, if the other signal processors, such as WFLC and TWD, were combined with the Kalman filter, results showed that both the signal noise and drift issues were completely removed and therefore the performance was much improved.

2.4.4 Fuzzy Logic Control

In this section, Fuzzy Logic Control is considered as a linear control method because various studies in the literature applied this technique to linearised systems for control. However, it is worth noting that the fuzzy logic control method can be applied to nonlinear systems too, due to the flexibility of its rule base.

In 1985, Takagi & Sugeno developed the fuzzy logic control technique known as Takagi–Sugeno (T-S) fuzzy model (Takagi & Sugeno, 1985) and nonlinear systems can be transformed into a T–S fuzzy model linearised in each fuzzy region on its if-then rules (fuzzy rules). This method has been widely used and applied to swinging up the inverted pendulum on the limited rail presented by Tao et al. (2008), with two fuzzy hybrid controls. To begin with, the first fuzzy swing-

up controller (FSUC) with fewer fuzzy rules was applied to make the inverted pendulum swing from an upside-down position upwards to reach the upright position, by moving the cart in the horizontal axis. Secondly, the Takagi-Sugeno (T-S) Fuzzy Model, controlled by the second fuzzy controller with parallel distributed pole assignment scheme (FC-PDPS), was used to balance the pendulum and cart on the rail at the equilibrium point in a small range, operating at approximately $\pm 17^{\circ}$($\pm$0.3 rad). In terms of balancing control in the upright position, the FC-PDPS was compared to the LQR control in the simulation test. Noticeably, the pendulum swinging up process was conducted with the same technique, which was the FSUC. The results showed that the performance using the FC-PDPS method was better than LQR as no oscillations in the pitch angle appeared around the equilibrium point. In particular, the cart was almost frozen in the horizontal axis for stabilising the system when implementing the FC-PDPS method; by contrast, the LQR technique presented significant movement of the cart between -0.3 m and 0.3 m.

Similarly, Aranda-Escolástico et al. (2016) presented the practical single and double rotary pendulum, stabilised by the Takagi–Sugeno (T-S) fuzzy model controller in the equilibrium position as well. This technique was compared to the full state feedback control and LQR method. In the case of swinging up the pendulum, an energy control method (Åström & Furuta, 2000) was applied. The authors introduced the region of attraction at the equilibrium point, which was the maximum angle α when the pendulum switched from swinging up control to stabilisation control. The implementation showed that the fuzzy model presented

significantly the widest angle at $+40.2^{\circ}$ over the full state feedback control ($+31.5^{\circ}$) and LQR method ($+34.2^{\circ}$).

2.4.5 Model Predictive Control.

In Chu & Chen (2017), the authors illustrated the linear control method of model predictive control for a self-balancing bicycle model. The gyroscopic effect (control moment gyroscope) was utilised for the system to maintain the vehicle in the upright vertical position using torque from two flywheels. Subsequently, the mathematic model of the system was linearised for the model predictive control scheme, which was used to predict horizon behaviour of the model for preparing the input parameters balancing the bicycle in the equilibrium point. The simulation demonstrated that the roll angle reached the desired angle at 20° and gimbal angles of the flywheels were 49.7° to maintain the desired angle.

Other applications of the model predictive control include, for instance: the implementation of a Furuta pendulum or inverted rotary pendulum by Seman et al. (2013). The authors introduced the two-step control for swinging-up the pendulum. Firstly, the classical energy control was compared with an exponentiation operation for swing-up the pendulum. After swinging the pendulum up by the two controllers, model predictive control was used to balance the pendulum in the vertically upright position. Note, the dynamic equations of the system needed to be linearised before be able to apply the predictive model. The results of both techniques to swing-up pendulum were similar; however, the computational time of the exponentiation control was slightly shorter as pre-calculation of pendulum energy was not needed. Because of this, the

exponentiation control was selected to operate with the MPC by Seman et al. (2013) and they presented satisfactory result for stabilising the inverted pendulum.

2.5 Nonlinear Control Techniques

Nonlinear theories have been demonstrated to generate excellent control results for highly nonlinear systems. Linearisation around the equilibrium is not necessary when using nonlinear methods, because the system can be controlled globally using pseudo-linear (nonlinear) system equation with advanced control techniques discussed below.

2.5.1 Freezing Control Technique

First of all, the extension of the LQR theory to control nonlinear systems, known as the freezing control technique (Banks & Mhana, 1992), showed superior control performance over all linear control techniques. Theoretical details of the nonlinear freezing technique will be presented in Chapter 6, but here a review of previous work using this method is conducted.

Harrison (2003) applied the nonlinear freezing control on an inverted pendulum on a cart model in simulation and demonstrate a number of benefits comparing against the LQR method. Most importantly, it was demonstrated that the stabilised system from an unconstrained nonlinear control provided a more comprehensive operation range than the linear control, at the initial pitch angle over 55.2°. The linearly controlled system was unstable over this angle, but the

nonlinear method could stabilise the system from an initial pitch angle up to 73.8° as well as between 96° and 179°.

Similarly, Xu, Zhang, & Carbone (2017) presented simulation results of a single and a double inverted pendulum on a cart model using the freezing control technique. Both models were stabilised from much wider initial pitch angles than using any linear control methods. For instance, the single pendulum was demonstrated to balance from initial pitch angles 60° ,180° and 240°, and the double pendulum was stabilised from initial pitch angles $x_3 = 45^{\circ}$ and $x_5 = 30^{\circ}$ (where x_3 and x_5 are lower and upper pendulum angle), respectively. Furthermore, Xu, Zhang, & Carbone (2017) applied the freezing control technique to the prototype of a single inverted pendulum on a cart using a built LEGO Mindstorms EV3 robot. The pendulum was well balanced from the initial pitch angle 10° and maintained at the upright position. These results demonstrated that the nonlinear freezing control technique was able to control the highly nonlinear inverted pendulum systems away from the usual, linearised region.

The freezing control technique was also referred to as a state-dependent Riccati equation (SDRE) control (Çimen & Banks, 2004a), which was applied to various other systems. For example, Çimen & Banks (2004a) presented this nonlinear method, compared with the LQR control, in flight control system of an F-8 aircraft model. The SDRE technique produced better performance than the LQR method in terms of stronger stabilisation when subject to different disturbances in the angle of attack (the angle between the chord line and flight path), in which the SDRE displayed capability of rejecting a wider disturbance angle, of up to 32.08° ;

by contrast, the LQR, which was itself an optimal technique, was restricted at 29.79° of disturbance rejection.

Based on the literature review and advantages discussed above, it is decided that the nonlinear freezing control would be one of the primary nonlinear control methods to be investigated and applied in this PhD research.

2.5.2 Freezing Control Technique with Extended Kalman Filter

Recall that in the cases of linear control in Section 2.4.3, the LQG method consists of the LQR control and the Kalman filter. Similarly, in nonlinear control, the extended Kalman filter (EKF), suitable for nonlinear systems, can be combined with the freezing control technique as well.

This technique has been applied in many applications: for instance, missile guidance simulation presented by Çimen & Merttopçuoğlu, (2008). The freezing control technique was applied to tracking systems used in military and the benefit was the ability to optimise the nonlinear tracking system globally. Furthermore, EKF was applied to estimate the state variables so they are available for feedback to control the missile trajectory, including the relative position vector, the relative velocity vector, and the target acceleration vector. Çimen & Merttopçuoğlu, (2008) presented the advantage of the combination of this nonlinear control and estimation technique, leading the missile (with a velocity of 500 m/s) to attack the target at a very distant position (initially 10 km away) in 20 s, in which the target had a speed velocity of 250 m/s.

Furthermore, the SDRE with EKF was applied to drug regimens in cancer treatment (Batmani & Khaloozadeh, 2013). The system model of tumour growth,

which was the cause of cancer, included four state variables: the tumour cells size, the number of normal cells, the drug concentration and the immune cells. The authors decided to only estimate the number of normal cells by EKF as the measurements were corrupted with noise but the size measurement of tumour cells where accurate from the medically professional equipment. Significantly, the simulation presented estimations of the number of normal cells with much reduced noise; moreover, the combination of the SDRE and EKF techniques showed efficiency in calculation the drug regimen, which eliminated the cancer cells completely.

Additionally, Nemra & Aouf (2010) introduced the INS/GPS sensor fusion technique for Unmanned Aerial Vehicle (UAV) by using the state-dependent Riccati equation (SDRE) with nonlinear filtering. Inside the UAV, the gyroscope and accelerator sensors are used to measure the dynamic system. The location is then calculated by INS/GPS sensor. In any real-world applications, these sensors come with noise. Therefore, an estimation of the state feedback system was performed using an extended Kalman filter. In terms of the flight control system, the SDRE was utilised in this implementation. The SDRE with extended Kalman filter (SDRE-EKF) control was compared with the linear Kalman filter control (KF) and a standalone EKF. SDRE-EKF presented the best performance on the UAV position estimated with smooth trajectory as well as strong nonlinearities. Then, the EKF control came 2nd with decent tracking trajectory, but not strong enough nonlinearities which caused undesirable oscillations. The linear KF produced the worst performance with a fair estimation trajectory, showing very oscillatory signals.

2.5.3 Iteration Scheme

Banks & McCaffrey (1998) introduced another nonlinear control technique, named iteration scheme, using linear, time-varying (LTV) approximations, which are arbitrarily close to the exact solution. This approximation solves the infinite-time horizon of pseudo-linear optimal control problem.

This control theory was applied to the inverted pendulum on a cart, for instance, using a sequence of time-varying iterative approximations with the optimal control (Banks & Dinesh, 2000). The simulations showed an increase of state sequences approximation to stabilise the inverted pendulum with initial pitch angles 20° and 50°. The results showed that the pitch angle converged rapidly to the reference position and the settling time was shorter when implementing with system using a higher order of state sequences; in this case, the simulation compared the 3rd order sequence with the 1st order one.

Furthermore, the nonlinear iteration control theory can be extended to complex nonlinear systems and there are a wide range of applications. For example, Çimen & Banks (2004b) presented the iteration scheme with linear-quadratic control for super-tankers autopilot. The model of oil-tank ship was simulated by increasing the state sequences of approximation with 1st, 5th and 20th orders to track the desired course heading of the ship. The best performance of tracking system was illustrated using the highest state sequences (order 20th) with the lowest error of heading angle.

Moreover, Itik, Salamci, & Banks (2009) introduced an optimal drug therapy control in cancer treatment using the iteration scheme combined with the LQR

optimal control. The order (10^{th}) of state sequences approximation simulated the model of tumour growth accurately, where the number of tumour is the cause of cancer. This combined control method was shown to lead to reduced tumour cells and chemotherapy; moreover, the simulation results suggested that cancer could be eliminated by providing an appropriate amount of drug at optimised times.

Additionally, in Du et al. (2009), the authors presented velocity tracking of a hydraulic press model by using a sequence of linear time-varying systems with a sliding mode controller, disturbed with reaction force. The simulation showed that the error of the velocity tracking was reduced significantly at a higher-order sequence (i.e., 5^{th}) rather than the 1^{st} order sequence, which was demonstrated to be globally convergent under a certain force.

Another application is the dynamics control of tunnel diode oscillator. Itik (2016) demonstrated the advantage of the iteration scheme technique, approximating the mathematical model of an electric current at the 5^{th}, 10^{th} and 20^{th} order sequences. The highest sequence (20^{th}) presented a convergence graph closer to the reference position than the rest, obtaining a more stabilised system for the electric circuits.

2.5.4 Sliding-Mode Control

The sliding-mode control method has been known as one of the complicated nonlinear control designs. Mun-Soo Park & Dongkyoung (2009) demonstrated swinging up and stabilisation of an inverted pendulum system (including inverted pendulum on cart and Furuta-pendulum) using a coupled sliding-mode surface, which were composed of actuated and unactuated systems. They demonstrated

in simulation semi-global asymptotic stabilisation using this control technique over the upper half-plane of the inverted pendulum systems, which could balance the pendulum at a large initial pitch angle at 88° rather than the classical linear controls, such as the simulation result of LQR controller (55.2°) in Harrison (2003). In the case of the Furuta pendulum, sliding control was applied and implemented to both the swing-up and stabilisation controls of the rotary pendulum. This method contrasted other linear controls using hybrid strategy, e.g., Seman et al. (2013) and Aranda-Escolástico et al. (2016). Moreover, the results showed asymptotic stabilisation of rotary pendulum with impulse disturbances, and also demonstrated aggressive swinging-up of the pendulum from an unactuated status to the vertically upright position. There has been recent advancement in sliding mode control - a new technique introduced by Zhu (2021). The author demonstrated that the novel complete model-free sliding mode control (CMFSMC) technique could be used to control and observe systems without the need of obtaining plant models, which could be applied to control more complex robotic systems in the future.

2.5.5 Neural Network Control

Neural network control is an intelligence nonlinear control technique which can learn to create appropriate parameters to control the system. It can be trained and can learn how to optimise output parameters depending on layers, weights, etc., to control the nonlinear systems.

In Cruz, García, & Bandala (2016), the implementation of a self-balancing two-wheeled robot was demonstrated using artificial neural network (ANN) control

with the extended delta-bar-delta algorithm (DBD) and the DBD algorithm was applied to increase the speed of convergence in ANNs weight. For instance, in terms of the pitch angle, the ANN control structure was designed as four feedback error inputs of desired pitch angle (including a recent error and three past errors), three neurons hidden layers, and two output layers (left and right motor voltages). After the ANN learning was completed, the implementation showed largely stabilised result of the robot's pitch angle with oscillations of approximately $\pm 1.7^{\circ}$ when the initial pitch angle was set at approximately 5°; moreover, the yaw angle was retained at the reference angle. In cases of displacement, the robot deviated from the reference position of 0 m and settled at 0.055 m, which was caused by oscillation.

Furthermore, the ANN could also be integrated with another control method, such as the model predictive control (MPC). For instance, Kokkrathoke (2018) implemented this combination to the self-balancing two-wheeled robot model. The author utilised ANN for the plant model which predicted future evolutions of the model by optimising a cost function, called Generalized Predictive Control (GPC). This mixed control method was compared against a classical PID controller with the same impulse disturbance, and the results demonstrated that the ANN method presented overshoots of lower amplitudes in the pitch angle than the PID controller approximately 2° after facing disturbance. However, both techniques could not control the robot displacement - they diverged from the reference position. This is because both MPC and the single PID controller could only be applied to control single-input single-out (SISO) systems.

Nevertheless, many researchers combined the neural network with linear controls. Therefore, the system model was linearised and applied to the neural network model; for instance, Jung & Kim (2008) introduced the neural network with PID controller for stabilising a two-wheel mobile inverted pendulum. The control system is similar to research presented by Cruz, García, & Bandala (2016); moreover, the PID controller was enhanced in the output of ANN to achieve real-time control. The structure of ANN is composed of six inputs (three variables of pitch angle and three variables of robot position), six outputs and nine hidden layers, which the outputs connect to the 6 feedback gains of 2 PID controllers. Note that the yaw angle control was neglected. The result of this technique was compared with the one using classical linear PID controller. When using PID control, the practical robot was balancing in the upright position, but the displacement diverged from the reference position. After that, the robot was disturbed by force, causing the pitch angle to go over 5° and the balancing robot toppled over. In terms of the ANN with PID controller implementation, the robot pitch angle was disturbed to approximately 8°, but the system stabilised well without a crash and maintained the desired position with a slight deviation after numerous impacts.

In another approach similar to the ANN with linear controller, Tsai et al. (2010) applied two adaptive controls with the radial basis function neural networks (RBFNNs) for balancing a two-wheeled scooter, compared with a state-feedback controller. The controllers were divided into the yaw angle control and pitch angle monitor. Similar to other linear controls, the mathematical model of system was linearised before being combined with the neural network method for control. The

simulation presented that both controllers stabilised the system from the initial pitch angle up to, approximately 17°. Moreover, the reference position tracking of yaw and pitch angles using the two adaptive controls with RBFNNs presented faster settling time than the state-feedback controller method. When implemented on the two-wheeled scooter, the two adaptive control methods were selected to control the practical vehicle and the results matched well to the simulations.

In conclusion, the first section of this chapter described the inverted pendulum on a cart system and its applications, e.g., rotatory inverted pendulum, self-balancing two-wheeled robot, two-wheeled personal transportation, and self-balancing bicycle. In the next section, methods to create mathematical models of the inverted pendulum were demonstrated, such as the Newton's law of motion and the Euler–Lagrange formulation (with the advantage force analysis is not necessary for obtaining the dynamic equations of the system. In addition, some researchers presented CAD modelling instead of the mathematical model. In cases of control strategy sections, various linear and nonlinear control methods were illustrated. Many linear controllers showed smaller operation ranges than nonlinear controllers as linearisation was needed. In particular, the freezing control, a nonlinear optimal control technique, demonstrated its capability of stabilising an inverted pendulum system with the broadest operation ranges over the linear and nonlinear controllers in this review.

Chapter 3

Hardware and Software Descriptions of a Self-Balancing Robot

3.1 Introduction

This chapter introduces the specification of the self-balancing two-wheeled robot and its programme, of which LEGO EV3 robot is selected as an application, for analysis, in this research. The chapter is organised as follows: In section 3.2, the evolution of the LEGO robotic and hardware specification are introduced. Then, the various software of LEGO EV3 are presented in section 3.3. The use of readily available hardware enhances the reproducibility of this work.

3.2 Hardware

This section provides information on hardware used in this research, including LEGO hardware history, LEGO EV3 specifications and their various sensors and an actuator.

3.2.1 History

To begin with, the 1st LEGO robotic tools set has been known as Robotics Invention System (RIS), based on LEGO building blocks. It can be programmed using an intelligent brick inside and was introduced by the LEGO group company in 1998 (LEGO, LEGO® Mindstorms, 2020a). The LEGO robotic kit has been named as RCX (Robotic Command eXplorers) brick, including a few sensors and executing programs by an infrared interface with a computer. For example,

LegWay balancing robots were built using the LEGO RCX (Hassenplug, 2003), as shown in Figure 3.1.

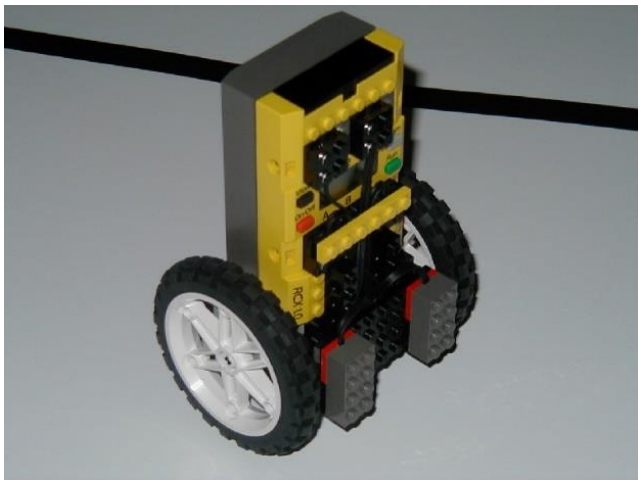


Figure 3.1: LegWay Balancing robots based on LEGO RCX. Reprinted from Steve's LegWay by Steve Hassenplug, 2003 (http://www.teamhassenplug.org/robots/legway). Copyright 2003 by Steve Hassenplug. Reprinted with permission.

The 2nd generation of LEGO Mindstorms product known as NXT was released in 2006. The interface was also upgraded using USB and Bluetooth connection with a computer. This set includes much hardware, e.g. servo motors, an intelligent brick, an ultrasonic sensor, a sound sensor, a touch sensor, and a light sensor; moreover, in 2009, the LEGO Mindstorms NXT was upgraded to version 2.0 by adding a colour sensor and remote control applications (Ford, 2011). For instance, one interesting application of LEGO Mindstorms NXT is presented as an NXTway self-balancing robot (Yamamoto, 2009), as shown in Figure 3.2.

Figure 3.2: NXTway self-balancing robot by using LEGO Mindstorms NXT (Yamamoto, 2009)

In addition, the 3rd generation of LEGO Mindstorms, namely EV3, was launched in 2013, as shown in Figure 3.3. This version is the current robotic kit of LEGO Mindstorms. In particular, the PC interface is upgraded to Wi-Fi connectivity, which is convenient for the user when implementing with high-speed wireless connection. Moreover, various sensors and hardware are introduced as follows (LEGO, 2020b):

- An Intelligent Brick, compatible with 32-bit CPU ARM9 processor, Wi-Fi USB port, Micro SD card reader, 4 motor ports and 5 buttons.
- Other hardware, including 3 servo motors, a gyro sensor, an ultrasonic sensor, a colour sensor and 2 touch sensors etc.

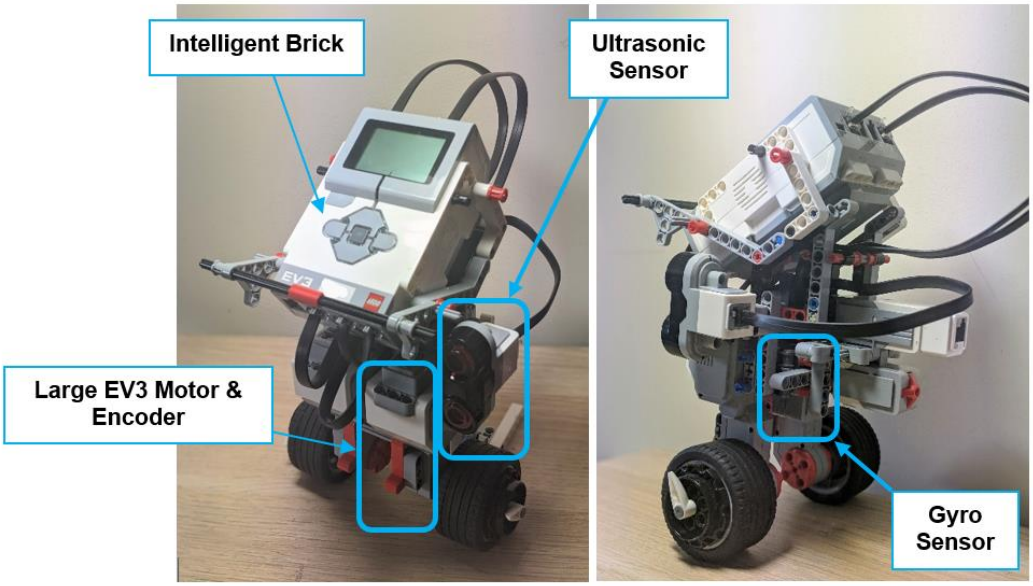


Figure 3.3: Self-balancing robot using LEGO Mindstorms EV3

In this research, the LEGO Mindstorms EV3 has been selected as the newest version of LEGO robotic kit, and also the latest version of EV3 firmware has been updated, in 2020. Furthermore, the EV3 robot has been widely used by researchers in many educational institutes as it is flexible to reconfigure and reprogramme and readily available.

3.2.2 LEGO Mindstorms EV3 Specifications

In this subsection, the hardware of LEGO Mindstorms EV3 is demonstrated, in the context of technical specifications used in this research, as follows:

EV3 Intelligent Brick with Education Version

- CPU ARM9 processor 32-bit with Linux operating system
- Four input ports with 1,000 samples/sec
- Four output ports for actuators
- FLASH memory 16 MB (5 MB left for creating or downloading a file)
- RAM 64 MB
- 178x128 pixel display
- USB port supported Wi-Fi dongle
- SD Card reader port
- Bluetooth communication
- Powered by lithium-ion 2200 mAh rechargeable DC battery or 6 AA batteries

Note that the EV3 brick firmware used is the developer edition version 1.09D, which is an alternative version from the LEGO company for high-level developers to connect the Wi-Fi network with MATLAB & Simulink. This is because the official firmware version for Home Edition and Education Edition (1.09H and 1.09E) does not allow remote Telnet access and is protected by non-administrative rights, controlling LEGO EV3 on Wi-Fi connection for beginner level developers (LEGO, 2020c).

Rechargeable DC Battery

The rechargeable DC lithium-ion battery of LEGO Mindstorms EV3 has a capacity of 2200 mAh 7.4V on its label from manufacturer, as shown in Figure 3.4. However, the voltage can sometimes present a higher value, for example, when being tested by a multimeter as shown in Figure 3.5. Therefore, the maximum voltage supply of LEGO EV3 battery will be examined before being used as a parameter for voltage input saturation. Details about control input saturation will be explained for both simulation and hardware implementation in Chapters 5 and 6.

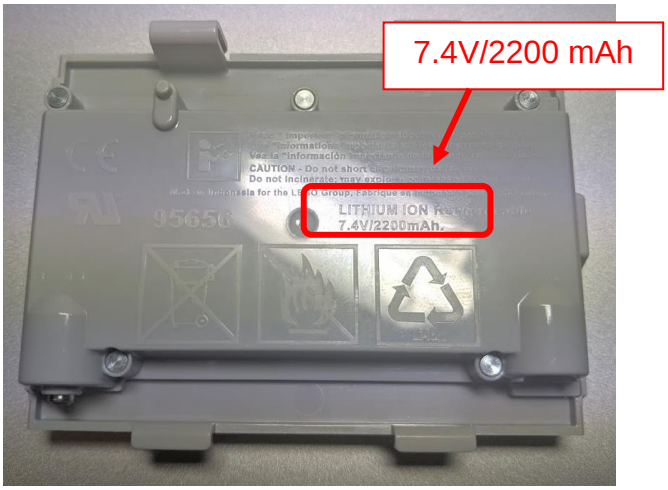


Figure 3.4: Rechargeable DC lithium-ion battery of LEGO Mindstorms EV3

Firstly, the maximum voltage of the power supply can be measured directly by a multimeter when the battery is charged fully, as shown in Figure 3.5.

Figure 3.5: Battery voltage measured by the multimeter

It can be seen in Figure 3.5 that the voltage is not 7.4V, the value given by the manufacturer; instead, the maximum voltage measured here is approximately 8.3V.

Alternatively, the Simulink block program can be used to measure the voltage of LEGO EV3 from the battery meter block diagram, as shown in Figure 3.6.

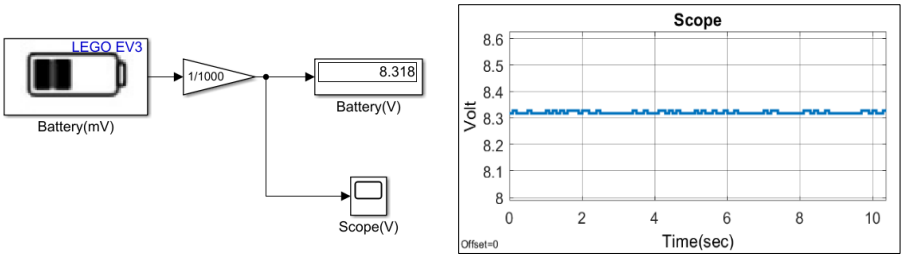


Figure 3.6: Voltage measured using Simulink block diagram

As can be seen from Figure 3.6, the voltage value read off the Simulink block diagram, at approximately 8.3 V, matches that measured by the multimeter. This voltage measurement made using the Simulink block diagram in the control system programme during implementation is very useful, as the programme could convert the percentage of PWM accurately. Furthermore, it is possible that the actual battery voltage may reduce because the power is dropped from implementation. Thus, the battery meter has been applied to the balancing control system.

The equation used to convert the input voltage of DC motor to PWM is given by,

$$PWM(\%) = V_{cal} \times \frac{100}{V_{battery}}, \tag{3.1}$$

where $V_{battery}$ is the voltage from the EV3 rechargeable DC battery, which is read by the battery block diagram in Simulink, as shown in Figure 3.6, and V_{cal} is the voltage supplied to the EV3 DC motor by the control system optimisation.

Sensors and Actuator

In terms of sensors and actuator description, specifications are provided in Tables 3.1-3.2.

Table 3.1: Sensors specification

Sensor	Output	Unit	Accuracy	Data Type	Maximum Sample Rate
Gyro Sensor	Angular velocity (-440 to 440 deg/s)	deg/s	± 3 deg	int32	1 kHz
Rotary Encoders	Angle	deg	± 1 deg	int32	1 kHz
Ultrasonic Sensor	Distance (5-255cm)	cm	±1 cm	uint8	100 Hz

Note: The output and accuracy are given in (LEGO, 2020b); moreover, the data type and maximum sample rate are provided in (Mathworks, 2020).

Table 3.2: Actuator specification

Actuator	Input	Unit	Input Range (%)	Speed	Running Torque (N.cm)	Stall Torque (N.cm)
Large EV3 Motor	PWM	%	-100 to 100	160-170 RPM or 960-1,020 deg/s	20	40

Note: The input range is stated in (Mathworks, 2020) whilst the speed and torque are given in (LEGO, 2020b).

As shown in Table 3.1, a gyro sensor is used to measure and calculate the robot body pitch angle in this research; however, it could generate a sensor drift when using only one sensor type for computing the pitch angle. Therefore, a combination of accelerometer and gyro sensor is an option to improve the accuracy of pitch angle calculation. Alternatively, in the case of only one available sensor in LEGO Mindstorm EV3, a state-estimation technique, namely, the Kalman filter, can be used to reduce the sensor drift and noise in feedback systems. This is because the signal output from the gyroscope is provided to the state-observer, which is used to estimate a more accurate signal before transferring the data to the controller. Secondly, the wheel angles of a LEGO robot are detected by the rotary encoders, which are attached inside the Large EV3 Motor. Next, an ultrasonic sensor is used as an non-touching starting button by the user. The advantage of this is to avoid touching any physical button, which would affect the initial pitch angle before running the programme.

In the case of the actuator, there are two types in LEGO Mindstorms EV3: medium and large motors. In this research, the two large motors with higher torques are chosen with description given in Table 3.2.

USB Wi-Fi dongle

Wi-Fi communication between the LEGO brick and the computer is necessary as it provides a long-range networking comparing with Bluetooth, and the USB cable is not needed to avoid physical movement restrictions. Therefore, an external USB Wi-Fi dongle is required. The Wi-Fi module supported by LEGO

is NetGear N150 (WNA1100) Wireless USB Adapter (LEGO, LEGO® Mindstorms, 2020a). However, there is an alternative USB Wi-Fi dongle for developers, which is Edimax N150 Wi-Fi Nano USB Adapter, shown in Figure 3.7. Moreover, the Edimax adapter has a smaller size, and less weight than the NetGear N150. Hence, the Edimax N150 is chosen as being more appropriate for this research.

(a) *(b)*

Figure 3.7: (a) Edimax N150 Wi-Fi Nano USB Adapter and (b) NetGear N150 (WNA1100) Wi-Fi USB Adapter (Netgear, 2020)

3.3 Software

The computation programmes widely used in LEGO robotics will be presented in this section; for instance, RIS, brickOS, NXT-G, LEGO Mindstorms EV3 Software, leJOS, Python and MATLAB and Simulink. More details of software are provided in the following sections:

3.3.1 Robotics Invention System (RIS)

The official programme RIS is compatible with LEGO RCX which is the 1st graphical programming environment design tool for the LEGO set. It was released

in 1998. This LEGO programming is also called RCX code blocks, which programmes the LEGO RCX by building blocks functions; next, the Infrared transmitter is applied to transfer data between the RCX and computer (LEGO, 1999).

3.3.2 BrickOS

BrickOS is a popular programme for LEGO RCX used in Legway, of which the previous name is LegOS, developed by Markus L. Noga (Hassenplug, 2003). Although it is an alternative software for LEGO RCX, the software is suitable for advanced developers, as it provides a C/C++ programming language environment.

3.3.3 NXT-G

NXT-G is an official software that comes bundled with the LEGO Mindstorms NXT 1.0 and 2.0, which is a graphical programming environment with a design similar to the RIS; however, the programming tools of NXT-G are simpler to the users than the RIS. Furthermore, the data connection between the NXT programme and the LEGO robot is upgraded by using a USB cable and Bluetooth connections, which provide a wider range and is more stable than the Infrared transmitter.

3.3.4 LEGO Mindstorms EV3 Software

Lego Mindstorms EV3 is the latest official graphical programming software of the LEGO Mindstorms products, for the EV3 version. The programming interface is similar to RIS and NXT-G as it uses the building blocks' functions to programme

the LEGO robot. Furthermore, there are 43 programming blocks containing more function than the NXT-G software (35 blocks). Moreover, the Wi-Fi connection is utilised to transfer data between the robot and computer, obtaining a broader transferring data range than the previous official LEGO software.

3.3.5 leJOS

The java programming language leJOS is used for all LEGO Mindstorms. The first leJOS used for the LEGO Mindstorms RCX was divided from TinyVM project in 2000. Then, the leJOS was ported to LEGO Mindstorms NXT in 2006, known as leJOS NXJ. Finally, the leJOS for the EV3 was released in 2013 (LeJOS, 2009).

3.3.6 Python

This high level and open-source programming language is applied to LEGO Mindstorms products, including NXT and EV3. There are official and alternative softwares used for LEGO Mindstorms EV3. Firstly, the Debian Linux-based operating system, namely, ev3dev, is an alternative software for high-level developers (ev3dev, 2020). Next, a collaboration of LEGO and ev3dev (2020) released an official software for LEGO Mindstorms and has been known as EV3 MicroPython (ev3dev, 2020), which is suitable for beginner-to-intermediate level developers. In particular, both software need to install a memory card inside the LEGO EV3 brick, used for the Python programming language.

3.3.7 MATLAB & Simulink

The well-known programme MATLAB, together with its powerful Simulink graphical programming environment, have been used for many types of research. The programme also has support packages for the LEGO Mindstorms NXT and EV3. There are a number of applications coded in MATLAB and Simulink programme on LEGO Mindstorms NXT and EV3, e.g. NXTway (Yamamoto, 2009), Rotary Inverted Pendulum (Masakatsu, 2015) and Gyroboy (Roslovets, 2020). Moreover, the MATLAB programme is a high-level programming language to control the LEGO Mindstorms NXT and EV3 via USB, Bluetooth and Wi-Fi (only EV3) connection, utilising MATLAB script and function files. Furthermore, developers can also design a control system using Simulink block diagrams, combining with MATLAB script and function files. The version used is MATLAB & Simulink R2019b with an academic license.

To summarise, in this research, the LEGO EV3 robot is selected for implementation, which is widely used in various other research; furthermore, it is compatible with MATLAB & Simulink programmes, and there are numerous tools for monitoring and controlling real-time systems, provided by MATLAB & Simulink, which are very useful and effective for researchers. However, the battery voltage and memory capacity of LEGO EV3 have limitations and therefore need to be considered carefully during the design and programming stages.

Chapter 4

Modelling of Inverted Pendulum and Two-Wheeled Robot Systems

4.1 Introduction

In this chapter, mathematical representations of an inverted pendulum on a cart and a self-balancing two-wheeled robot are introduced and studied as the mathematical model of the TWR is applied to robot prototype, LEGO EV3, in the experimentation sections in Chapters 5 and 6. This chapter can be divided into three parts. Firstly, the mathematical models are analysed in Section 4.2. Then, in Section 4.3, a revised mathematical representation, where the control inputs are transferred from forces to the motor voltages is presented. Finally, linearisation of the model is demonstrated in Section 4.4.

4.2 Mathematical Model

4.2.1 Inverted Pendulum on a Cart Model

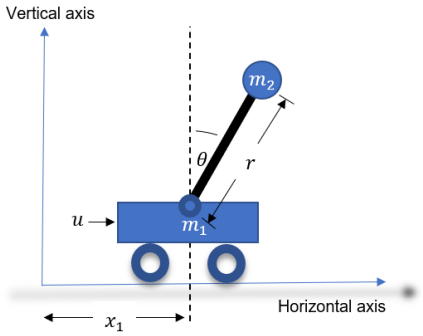


Figure 4.1: An inverted pendulum on a cart (Xu, Zhang, & Carbone, 2017)

The self-balancing two-wheeled robot has been developed from the classical benchmarking system known as an inverted pendulum on a cart, shown in Figure 4.1. The control objective is to balance the pendulum in the vertically upright but unstable position.

The mathematical models of an inverted pendulum describe equations of motions and are often obtained by analysing forces, using Newton's laws. There are also other methods for obtaining the mathematical models of these systems, for example, using the Lagrangian approach based on the system's potential energy and kinetic energy. Xu, Zhang, and Carbone (2017) presented nonlinear models of the inverted pendulum system shown in Figure 4.1, using the Lagrangian method. This is because the Lagrangian technique utilises only two terms energy and the calculations are simpler than using the Newton's method, which require a number of force component equations.

In terms of potential energy (V), the amount of energy in the vertical displacement of an inverted pendulum is analysed and given by (Xu, Zhang, & Carbone, 2017)

$$V = m_2 g(r + r\,cos\,\theta), \tag{4.1}$$

where

m_2 is the mass of pendulum $(0.1\ kg)$,

g is the acceleration due to gravity $(9.8\ m^2/s)$,

and r is the length of the pendulum $(0.5\ m)$.

In the case of kinetic energy (T), the amount of energy along the horizontal axis can be written as (Xu, Zhang, & Carbone, 2017)

$$\begin{aligned} T &= \frac{1}{2}m_1\dot{x}_1^2 + \frac{1}{2}m_2\left[\frac{d}{dt}x_1 + r\sin\theta\right]^2 + \frac{1}{2}m_2\left[\frac{d}{dt}(r\cos\theta)\right]^2 \\ &= \frac{1}{2}m_1\dot{x}_1^2 + \frac{1}{2}m_2(\dot{x}_1 + r\dot{\theta}\cos\theta)^2 + \frac{1}{2}m_2(-r\dot{\theta}\sin\theta)^2 \\ &= \frac{1}{2}(m_1 + m_2)\dot{x}_1^2 + m_2 r\dot{x}_1\dot{\theta}\cos\theta + \frac{1}{2}m_2 r^2\dot{\theta}^2, \end{aligned} \tag{4.2}$$

where m_1 is the mass of cart $(2\ kg)$, x_1 is the cart displacement (m) and θ is the pendulum angle (rad).

The Lagrangian (L) is given by the difference between total kinetic energy (T) and the total potential energy (V) as follows:

$$L = T - V. \tag{4.3}$$

Thus, substituting (4.1) and (4.2) into (4.3), the following Lagrangian is obtained

$$L = \frac{1}{2}(m_1 + m_2)\dot{x}_1^2 + m_2 r\dot{x}_1\dot{\theta}\cos\theta + \frac{1}{2}m_2 r^2\dot{\theta}^2 - m_2 g(r + r\cos\theta) \tag{4.4}$$

Lagrange's equation is

$$\frac{d}{dx}\frac{\partial L}{\partial \dot{x}_i} - \frac{\partial L}{\partial x_i} = f_i, \qquad 1 \le i \le n, \tag{4.5}$$

where x_i is the ith generalised coordinate, f_i is the ith generalised force and n is the number of the degrees of freedom (DOF).

There are two generalised coordinates in the inverted pendulum on a cart system presented in Figure 4.1, which are the cart replacement x_1 and the pitch angle θ as given by

$$f_1 = u, \text{ and } f_2 = 0. \tag{4.6}$$

Then the Lagrange's equation becomes

$$\frac{d}{dx}\frac{\partial L}{\partial \dot{x}_1} - \frac{\partial L}{\partial x_1} = u, \qquad and \qquad \frac{d}{dx}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0 \tag{4.7}$$

Substituting Eq.(4.4) into (4.7) (Xu, Zhang, & Carbone, 2017) for the single inverted pendulum on a cart system, the result is as follows:

$$\begin{aligned} \ddot{x}_1 &= \frac{m_2 r\dot{\theta}^2 \sin\theta - m_2 g \sin\theta \cos\theta + u}{m_1 + m_2 \sin^2\theta} \\ \ddot{\theta} &= \frac{-m_2 r\dot{\theta}^2 \sin\theta \cos\theta + (m_1 + m_2) g \sin\theta - u\cos\theta}{r(m_1 + m_2 \sin^2\theta)}, \end{aligned} \tag{4.8}$$

which can be transferred into standard state-space model equations by defining variables $x_2 = \dot{x}_1,\ x_3 = \theta$ and $x_4 = \dot{x}_3$. Therefore, a state-space representation of the inverted pendulum on a cart is given by

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{m_2 r x_4^2 \sin x_3 - m_2 g \sin x_3 \cos x_3 + u}{m_1 + m_2 \sin^2 x_3} \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= \frac{-m_2 r x_4^2 \sin x_3 \cos x_3 + (m_1 + m_2) g \sin x_3 - u\cos x_3}{r(m_1 + m_2 \sin^2 x_3)}, \end{aligned} \tag{4.9}$$

which can be rewritten into a non-unique nonlinear state-space matrix form as

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ 0 & 0 & \dfrac{-m_2 g \sin x_3 \cos x_3}{(m_1 + m_2 \sin^2 x_3)x_3} & \dfrac{m_2 r x_4 \sin x_3}{m_1 + m_2 \sin^2 x_3}\\ 0 & 0 & 0 & 1\\ 0 & 0 & \dfrac{(m_1 + m_2) g \sin x_3}{r(m_1 + m_2 \sin^2 x_3)x_3} & \dfrac{-m_2 r x_4 \sin x_3 \cos x_3}{r(m_1 + m_2 \sin^2 x_3)}\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\\ x_4\end{pmatrix} + \begin{pmatrix}0\\ \dfrac{1}{m_1 + m_2 \sin^2 x_3}\\ 0\\ \dfrac{-\cos x_3}{r(m_1 + m_2 \sin^2 x_3)}\end{pmatrix} u. \tag{4.10}$$

Additionally, the linear model of an inverted pendulum on a cart can be expressed by limiting x_3 and x_4 as 'small' quantities in Eq.(4.10) and therefore several approximations follows:

$$sin(x_3) \approx x_3,\ cos(x_3) \approx 1,\ x_4\, sin(x_3) \approx 0 \quad \text{and} \quad sin\,(x_3)^2 \approx 0 \tag{4.11}$$

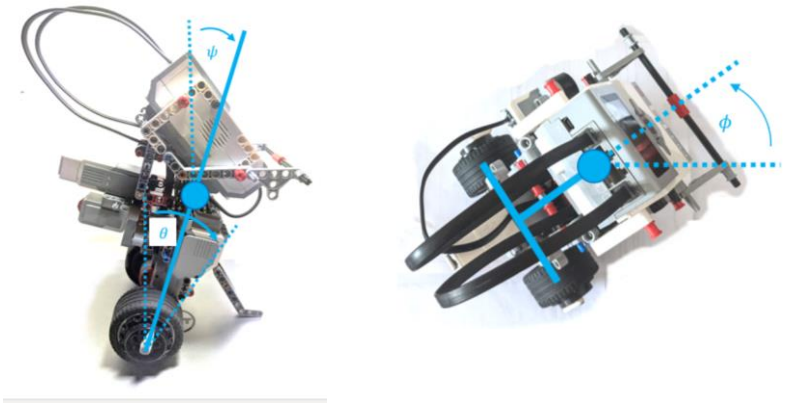
Thus, a linear state-space model of the inverted pendulum on a cart system can be represented in the form of

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ 0 & 0 & \dfrac{-m_2 g}{m_1} & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & \dfrac{(m_1 + m_2) g}{r m_1} & 0\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\\ x_4\end{pmatrix} + \begin{pmatrix}0\\ \dfrac{1}{m_1}\\ 0\\ \dfrac{-1}{r m_1}\end{pmatrix} u. \tag{4.12}$$

4.2.2 Self-Balancing Two-Wheeled Robot Model

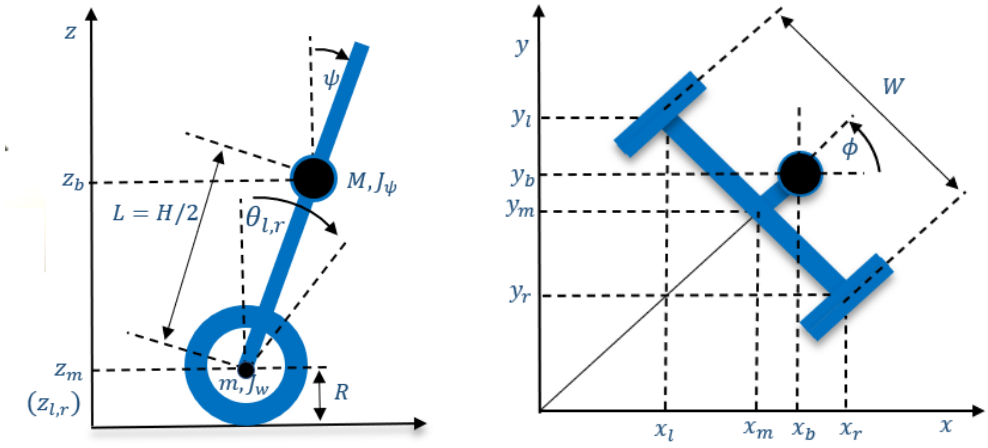
As a classical benchmarking model, an inverted pendulum on a cart is widely used to test control strategies, the model of which was described in Section 4.2.1. In this subsection, the inverted pendulum theory will be extended to the self-balancing two-wheeled robot, of which the LEGO EV3 robot is selected as a prototype, for the analysis and investigation. A LEGO self-balancing two-wheeled

robot is shown in Figures 4.2-4.3. Figure 4.2 presents the prototype TWR with the three generalised coordinates: the robot pitch angle ψ, the robot yaw angle ϕ and the wheel angle θ (Yamamoto, 2009). More details of the TWR on the X,Y and Z coordinates for designing the motion equation can be seen in Figure 4.3.



(a) *(b)*

Figure 4.2: Self-balancing two-wheeled robot (LEGO EV3), (a) side view and (b) top view.



(a) *(b)*

Figure 4.3: Self-balancing two-wheeled robot diagram, (a) side view and (b) top view (Yamamoto, 2009).

Note that l and r mean left and right. For instance, θ_l and θ_r are the left and right wheel angles; similarly, Z_l and Z_r are the height of left and right wheel in Z coordinate.

Physical parameters of the self-balancing two-wheeled robot are given in Table 4.1. Note that the values of parameters1-11 were measured and calculated from the LEGO EV3 for this research and the parameters12-18 are presented in Yamamoto (2009).

Table 4.1: Physical parameters of the self-balancing two-wheeled robot

No.	Parameters	Description	Value
1	m	Mass of wheel	$0.05\ kg$
2	M	Mass of robot body	$0.64\ kg$
3	R	Wheel radius	$0.027\ m$
4	W	Robot's body width	$0.105\ m$
5	D	Robot's body depth	$0.1\ m$
6	H	Robot's body height	$0.21\ m$
7	$L = H/2$	Distance between wheel axle and centre of robot	$0.105\ m$
8	g	Acceleration due to gravity	$9.81\ m^2/s$
9	$J_w = mR^2/2$	Inertia moment of wheel	$0.0000162\ kgm^2$
10	$J_\psi = ML^2/3$	Inertia moment of robot pitch	$0.002352\ kgm^2$
11	$J_\phi = M(W^2 + D^2)/12$	Inertia moment of robot yaw	$0.001121 kgm^2$
12	J_m	Inertia moment of DC motor	$1 \times 10^{-5}\ kgm^2$
13	R_m	Resistance of DC motor	$6.69\ \Omega$
14	K_b	Back EMF constant of DC motor	$0.468\ V \cdot Sec/rad$
15	K_t	Torque constant of DC motor	$0.317\ Nm/A$
16	n	Gear ratio	1
17	f_m	Coefficient of friction between robot and DC motor	0.0022
18	f_w	Coefficient of friction between wheel and floor	0

Table 4.1, Yamamoto (2009) provided the physical parameters for the LEGO NXT model, which is a different version to the robot (LEGO EV3) used in this research. However, Roslovets (2020) verified that these parameters could be utilised for the LEGO EV3 as he presented a self-balancing two-wheeled robot from the LEGO EV3, namely the Gyroboy, in his Simulink simulation and the implementation sections.

For the two-wheeled robot, the Lagrangian method is also applied to analyse the mathematical model based on the system's potential and kinetic energy, as indicated in Figure 4.3. Yamamoto (2009) produced the total kinetic energy and potential energy equations from the LEGO two-wheeled robot using the Lagrangian technique, which can be expressed as

The kinetic energy (T)

$$T_1 = \frac{1}{2}m\left(\dot{x}_l^2 + \dot{y}_l^2 + \dot{z}_l^2\right) + \frac{1}{2}m(\dot{x}_r^2 + \dot{y}_r^2 + \dot{z}_r^2) + \frac{1}{2}M\left(\dot{x}_b^2 + \dot{y}_b^2 + \dot{z}_b^2\right) \tag{4.13}$$

$$T_2 = \frac{1}{2}J_w\dot{\theta}_l^2 + \frac{1}{2}J_w\dot{\theta}_r^2 + \frac{1}{2}J_\psi\dot{\psi}^2 + \frac{1}{2}J_\phi\dot{\phi}^2 + \frac{1}{2}n^2J_m(\dot{\theta}_l - \dot{\psi})^2 + \frac{1}{2}n^2J_m(\dot{\theta}_r - \dot{\psi})^2. \tag{4.14}$$

The potential energy (V)

$$V = mgz_l + mgz_r + Mgz_b. \tag{4.15}$$

The motion equations are given by

$$(x_m,\ y_m,\ z_m) = \left(\int \dot{x}_m\, dt, \int \dot{y}_m\, dt, R\right), (\dot{x}_m, \dot{y}_m) = \left(R\dot{\theta}cos\,\phi\,, R\dot{\theta}\,sin\,\phi\,\right) \tag{4.16}$$

$$(x_l,\ y_l,\ z_l) = \left(x_m - \frac{W}{2}sin\,\phi\,,\ \ y_m + \frac{W}{2}cos\,\phi\,,\ z_m\right) \tag{4.17}$$

$$(x_r,\ y_r,\ z_r) = \left(x_m + \frac{W}{2}sin\,\phi\,,\ \ y_m - \frac{W}{2}cos\,\phi\,,\ z_m\right) \tag{4.18}$$

$$(x_b,\ y_b,\ z_b) = (x_m + L\,sin\,\psi\,cos\,\phi\,,\ \ y_m + L\,sin\,\psi\,sin\,\phi\,,\ \ z_m + Lcos\,\psi\,) \tag{4.19}$$

$$(\theta, \phi) = \left(\frac{(\theta_l + \theta_r)}{2}, \frac{R(\theta_r - \theta_l)}{W}\right). \tag{4.20}$$

The kinetic energy (T_1) can be rewritten by substituting Eqs.(4.16)-(4.19) into Eq. (4.13):

$$\begin{aligned} T_1 = {} & \frac{1}{2}m\left[\left(R\dot{\theta}\cos\phi - \frac{W}{2}\dot{\phi}\cos\phi\right)^2 + \left(R\dot{\theta}\sin\phi - \frac{W}{2}\dot{\phi}\sin\phi\right)^2\right] \\ & + \frac{1}{2}m\left[(R\dot{\theta}\cos\phi + \frac{W}{2}\dot{\phi}\cos\phi)^2 + (R\dot{\theta}\sin\phi + \frac{W}{2}\dot{\phi}\sin\phi)^2\right] \\ & + \frac{1}{2}M\left[\left(R\dot{\theta}\cos\phi - L\sin\psi\sin\phi\dot{\phi} + L\cos\phi\cos\psi\dot{\psi}\right)^2\right. \\ & \qquad + \left(R\dot{\theta}\sin\phi + L\sin\psi\cos\phi\dot{\phi} + L\sin\phi\cos\psi\dot{\psi}\right)^2 \\ & \qquad \left. + \left(-L\sin\psi\dot{\psi}\right)^2\right] \\ = {} & \frac{1}{2}m\left[2R^2\dot{\theta}^2 + \frac{W^2}{2}\dot{\phi}^2\right] \\ & \qquad + \frac{1}{2}M\left[R^2\dot{\theta}^2 + L^2\sin^2\psi\,\dot{\phi}^2 + 2RL\dot{\theta}\cos\psi\,\dot{\psi} + L\dot{\psi}^2\right]. \end{aligned} \tag{4.21}$$

Equation (4.20) can be transformed as

$$\theta_r = \theta + \frac{W\phi}{2R} \tag{4.22}$$

$$\theta_l = \theta - \frac{W\phi}{2R}. \tag{4.23}$$

Rewriting the kinetic energy (T_2) by substituting Eqs.(4.22)-(4.23) into Eq.(4.14):

$$\begin{aligned} T_2 = {} & \frac{1}{2}J_w\left(\dot{\theta} - \frac{W\dot{\phi}}{2R}\right)^2 + \frac{1}{2}J_w\left(\dot{\theta} + \frac{W\dot{\phi}}{2R}\right)^2 + \frac{1}{2}J_\psi\dot{\psi}^2 + \frac{1}{2}J_\phi\dot{\phi}^2 \\ & \qquad + \frac{1}{2}n^2J_m(\dot{\theta} - \frac{W\dot{\phi}}{2R} - \dot{\psi})^2 + \frac{1}{2}n^2J_m\left(\dot{\theta} + \frac{W\dot{\phi}}{2R} - \dot{\psi}\right)^2 \\ = {} & \frac{1}{2}J_w\left(2\dot{\theta}^2 - \frac{W^2\dot{\phi}^2}{2R^2}\right) + \frac{1}{2}J_\psi\dot{\psi}^2 + \frac{1}{2}J_\phi\dot{\phi}^2 \\ & \qquad + \frac{1}{2}n^2J_m\left[2\dot{\theta}^2 + 2\dot{\psi}^2 - 4\dot{\theta}\dot{\psi} + \frac{W^2\dot{\phi}^2}{2R^2}\right]. \end{aligned} \tag{4.24}$$

The potential energy (V) can be rewritten by substituting Eqs.(4.16)-(4.19) into Eq.(4.15):

$$V = mgR + mgR + Mg(R + Lcos\,\psi) \tag{4.25}$$
$$= 2mgR + MgR + MgLcos\,\psi.$$

The Lagrangian (L) is now defined as the following:

$$L = T_1 + T_2 - V. \tag{4.26}$$

Therefore, the Lagrange equations are given by

$$\frac{d}{dx}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = F_\theta, \tag{4.27}$$

$$\frac{d}{dx}\frac{\partial L}{\partial \dot{\psi}} - \frac{\partial L}{\partial \psi} = F_\psi, \tag{4.28}$$

$$\frac{d}{dx}\frac{\partial L}{\partial \dot{\phi}} - \frac{\partial L}{\partial \phi} = F_\phi. \tag{4.29}$$

Evaluating Eqs. (4.27) - (4.29) produces the following (Yamamoto, 2009) differential equations, which represent the two-wheel robot system:

$$[(2m + M)R^2 + 2J_w + 2n^2J_m]\ddot{\theta} + (MLRcos\psi - 2n^2J_m)\ddot{\psi} \tag{4.30}$$
$$-MLR\dot{\psi}^2\,sin\,\psi = F_\theta,$$

$$(MLRcos\psi - 2n^2J_m)\ddot{\theta} + \left(ML^2 + J_\psi + 2n^2J_m\right)\ddot{\psi} - MgL\,sin\,\psi \tag{4.31}$$
$$-ML^2\dot{\phi}^2\,sin\,\psi\,cos\psi = F_\psi,$$

$$\left[\frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}(J_w + n^2J_m) + ML^2sin^2\psi\right]\ddot{\phi} \tag{4.32}$$
$$+2ML^2\dot{\psi}\,\dot{\phi}sin\,\psi cos\psi = F_\phi.$$

Then, transferring variables θ, ψ, and ϕ into standard state-space model variables as follows:

$x_1 = \theta,\ \ x_2 = \dot{\theta},$ then $\dot{x}_2 = \ddot{\theta},$

$x_3 = \psi,\ \ x_4 = \dot{\psi},$ then $\dot{x}_4 = \ddot{\psi},$

$x_5 = \phi,\ \ x_6 = \dot{\phi},$ then $\dot{x}_6 = \ddot{\phi},$

we re-write Eqs. as (4.*30*) - (4.*32*) as

$$[(2m+M)R^2 + 2J_w + 2n^2J_m]\dot{x}_2 + (MLRcos(x_3) - 2n^2J_m)\dot{x}_4 - MLRx_4^2\, sin(x_3) = F_\theta, \tag{4.33}$$

$$(MLRcos(x_3) - 2n^2J_m)\dot{x}_2 + (ML^2 + J_\psi + 2n^2J_m)\dot{x}_4 - MgL\, sin(x_3) - ML^2x_6^2\, sin(x_3)\, cos(x_3) = F_\psi, \tag{4.34}$$

$$\left[\frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}(J_w + n^2J_m) + ML^2sin^2(x_3)\right]\dot{x}_6 + 2ML^2x_4\, x_6\, sin(x_3)\, cos(x_3) = F_\phi \tag{4.35}$$

This PhD research work focuses on the stabilisation in the robot pitch angle ψ. Therefore, the robot yaw angle ϕ is not considered. Moreover, the computation of three generalised coordinates requires a large amount of memory capacity when the robot's CPU is processing with nonlinear freezing optimal control; thus, the associated equation for the yaw angle ϕ is neglected and no yaw control is provided. For more details see in Chapter 6, Section 6.6: Experimental Results.

Therefore, the nonlinear system equations of the two-wheeled robot model with two generalised coordinates (the robot pitch angle ψ and the wheel angle θ) are written by extending Eqs.(4.33)-(4.34), which can be used as the state-space representation of the system, as the following:

Firstly, the $\dot{x}_1$ equation can be defined as

$$\dot{x}_1 = x_2 \tag{4.36}$$

Next, the $\dot{x}_2$ equation can be written as

$$\begin{aligned}\dot{x}_2 = {} & x_4 \times \frac{x_4 sin(x_3)(M^2RL^3 + 2MRLn^2J_m + MRLJ_\psi)}{a + b(x_3)} \\ & + \frac{2n^2J_mMgLsin(x_3) - M^2RL^2cos(x_3)g\ sin(x_3)}{a + b(x_3)} \\ & + F_\psi \times \frac{(2n^2J_m - MRLcos(x_3))}{a + b(x_3)} \\ & + F_\theta \times \frac{(ML^2 + 2n^2J_m + J_\psi)}{a + b(x_3)}\end{aligned} \tag{4.37}$$

where a and b are given by

$$\begin{aligned}a = {} & 2J_wJ_\psi + 2mR^2ML^2 + 4mR^2n^2J_m + 2MR^2n^2J_m + 2n^2J_mML^2 \\ & + 2mR^2J_\psi + MR^2J_\psi + 2J_wML^2 + 4J_wn^2J_m + 2n^2J_mJ_\psi,\end{aligned} \tag{4.38}$$

$$b(x_3) = M^2R^2L^2\ sin(x_3)^2 + 4MLRcos(x_3)n^2J_m \tag{4.39}$$

Let

$$e_{23}(x_3) = 2n^2J_mMgLsin(x_3) - M^2RL^2cos(x_3)g\ sin(x_3),$$

$$e_{24}(x_3, x_4) = x_4 sin(x_3)(M^2RL^3 + 2MRLn^2J_m + MRLJ_\psi),$$

$$f_{21} = ML^2 + 2n^2J_m + J_\psi$$

and

$$f_{22}(x_3) = 2n^2J_m - MRLcos(x_3),$$

which can be used to rewrite Eq.(4.37) as follows:

$$\dot{x}_2 = \frac{e_{23}(x_3)}{(a+b(x_3))x_3} \times x_3 + \frac{e_{24}(x_3,x_4)}{(a+b(x_3))} \times x_4 + \frac{f_{21}}{(a+b(x_3))} \times F_\theta + \frac{f_{22}(x_3)}{(a+b(x_3))} \times F_\psi \tag{4.40}$$

Then, define the $\dot{x}_3$ equation as

$$\dot{x}_3 = x_4 \tag{4.41}$$

Next, the $\dot{x}_4$ equation is represented as

$$\dot{x}_4 = x_4 \times \frac{x_4 sin(x_3)(-M^2R^2L^2cos(x_3) + 2MRLn^2J_m)}{a+b(x_3)} + \frac{MgLsin(x_3)(2J_w + 2mR^2 + MR^2 + 2n^2J_m)}{a+b(x_3)} + F_\theta \times \frac{(2n^2J_m - MRLcos(x_3))}{a+b(x_3)} + F_\psi \times \frac{(2n^2J_m + 2J_w + 2mR^2 + MR^2)}{a+b(x_3)}. \tag{4.42}$$

Also let

$$e_{43}(x_3) = MgLsin(x_3)(2n^2J_m + 2J_w + 2mR^2 + MR^2),$$
$$e_{44}(x_3,x_4) = x_4 sin(x_3)(-M^2R^2L^2cos(x_3) + 2MRLn^2J_m),$$
$$f_{41}(x_3) = 2n^2J_m - MRLcos(x_3)$$

and

$$f_{42} = 2n^2J_m + 2J_w + 2mR^2 + MR^2$$

which can be rewritten from Eq.(4.42) as follows:

$$\dot{x}_4 = \frac{e_{43}(x_3)}{(a+b(x_3))x_3} \times x_3 + \frac{e_{44}(x_3,x_4)}{(a+b(x_3))} \times x_4 + \frac{f_{41}(x_3)}{(a+b(x_3))} \times F_\theta + \frac{f_{42}}{(a+b(x_3))} \times F_\psi \tag{4.43}$$

Therefore, the nonlinear state-space model of the self-balancing two-wheeled robot controlled by two forces: the pitch angle force F_ψ and the wheel angle force F_θ, can be represented as follows:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{e_{23}(x_3)}{(a+b(x_3))x_3} & \frac{e_{24}(x_3,x_4)}{a+b(x_3)} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{e_{43}(x_3)}{(a+b(x_3))x_3} & \frac{e_{44}(x_3,x_4)}{a+b(x_3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{f_{21}}{a+b(x_3)} & \frac{f_{22}(x_3)}{a+b(x_3)} \\ 0 & 0 \\ \frac{f_{41}(x_3)}{a+b(x_3)} & \frac{f_{42}}{a+b(x_3)} \end{pmatrix} \begin{pmatrix} F_\theta \\ F_\psi \end{pmatrix}. \tag{4.44}$$

4.3 Converting Control Inputs from Forces to Voltages

The system dynamics of a self-balancing two-wheeled robot, controlled by forces were presented in the previous section. In real-world applications, forces are generated from the hardware; for instance, the LEGO EV3 robot is controlled by forces, produced from the DC motors. This section, therefore, presents voltage conversion of the LEGO EV3 robot from forces, and then applies to the state-

space model, which is used to balance the two-wheel robot in the subsequent chapters.

The standard DC motor schematic is presented in Figure 4.4.

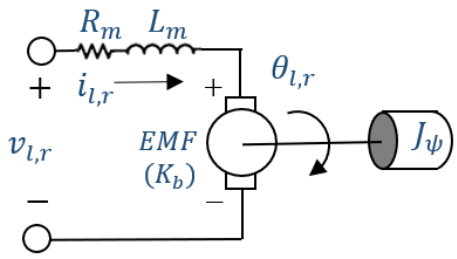


Figure 4.4: DC motor schematic (Chiasson, 2005)

Kirchhoff's voltage law is applied to summarise the electrical circuits in Figure 4.4, given the equation as follows (Yamamoto, 2009):

$$L_m \frac{di_{l,r}}{dt} = v_{l,r} + K_b\left(\frac{d\psi}{dt} - \frac{d\theta_{l,r}}{dt}\right) - R_m i_{l,r}, \tag{4.45}$$

where $i_{l,r}$ is the DC motor current.

In DC circuits, the inductance of motor in steady-state operation behaves like a short circuit; therefore, it can be approximated as zero, and then Eq. (4.45) can be rewritten as:

$$i_{l,r} = \frac{v_{l,r} + K_b(\dot{\psi} - \dot{\theta}_{l,r})}{R_m}. \tag{4.46}$$

Yamamoto (2009) demonstrated the generalised forces of the DC motor torque and viscous friction as follows:

$$(F_{\theta,}\, F_\psi) = \left(F_l + F_r\,,\, F_\psi\right) \tag{4.47}$$

$$F_l = nK_t i_l + f_m(\dot{\psi} - \dot{\theta}_l) - f_w\dot{\theta}_l \tag{4.48}$$

$$F_r = nK_t i_r + f_m(\dot{\psi} - \dot{\theta}_r) - f_w\dot{\theta}_r \tag{4.49}$$

$$F_\psi = nK_t i_l - nK_t i_r - f_m(\dot{\psi} - \dot{\theta}_l) - f_m(\dot{\psi} - \dot{\theta}_r) \tag{4.50}$$

Therefore, the generalised forces F_θ and F_ψ in terms of motor voltage from Eq.(4.46*)* are given by

$$F_\theta = \alpha(v_l + v_r) - 2(\beta + f_w)\dot{\theta} + 2\beta\dot{\psi} \tag{4.51}$$

$$F_\psi = -\alpha(v_l + v_r) + 2\beta\dot{\theta} - 2\beta\dot{\psi} \tag{4.52}$$

where

$$\alpha = \frac{nK_t}{R_m}, \beta = \frac{nK_t K_b}{R_m} + f_m. \tag{4.53}$$

Then Eqs.(4.51)-(4.52) can be rewritten in the standard state-space model variables, given by

$$F_\theta = \alpha(v_l + v_r) - 2(\beta + f_w)x_2 + 2\beta x_4 \tag{4.54}$$

$$F_\psi = -\alpha(v_l + v_r) + 2\beta x_2 - 2\beta x_4 \tag{4.55}$$

Now, the state-space representation of the two-wheeled robot would be transferred from generalised forces to voltages by substituting Eqs. (4.54)-(4.55) to Eq.(4.37) as follows:

$$\begin{aligned}\dot{x}_2 = \; & x_2 \times \left(\frac{2\beta f_{22}(x_3)}{a + b(x_3)} - \frac{2(\beta + f_w)f_{21}}{a + b(x_3)}\right) + \frac{e_{23}(x_3)}{a + b(x_3)} \\ & + x_4 \times \left(\frac{e_{24}(x_3, x_4)}{a + b(x_3)} + \frac{2\beta(f_{21} - f_{22}(x_3))}{a + b(x_3)}\right) \\ & + v_l \times \frac{\alpha(f_{21} - f_{22}(x_3))}{a + b(x_3)} + v_r \times \frac{\alpha(f_{21} - f_{22}(x_3))}{a + b(x_3)}\end{aligned} \tag{4.56}$$

Let

$$e_{m22}(x_3) = 2\beta f_{22}(x_3) - 2(\beta + f_w)f_{21},$$
$$e_{m24}(x_3, x_4) = e_{24}(x_3, x_4) + 2\beta(f_{21} - f_{22}(x_3)),$$
$$f_{m21}(x_3) = \alpha(f_{21} - f_{22}(x_3))$$

and

$$f_{m22}(x_3) = \alpha(f_{21} - f_{22}(x_3)),$$

which can be rewrite Eq.(4.56) as below:

$$\dot{x}_2 = \frac{e_{m22}(x_3)}{(a + b(x_3))} \times x_2 + \frac{e_{23}(x_3)}{(a + b(x_3))x_3} \times x_3 + \frac{e_{m24}(x_3, x_4)}{(a + b(x_3))} \times x_4 + \frac{f_{m21}}{(a + b(x_3))} \times v_l + \frac{f_{m22}(x_3)}{(a + b(x_3))} \times v_r \quad (4.57)$$

Next, for the $\dot{x}_4$ equation, substituting Eqs. (4.54)-(4.55) into Eq.(4.42):

$$\dot{x}_4 = x_2 \times \left(\frac{2\beta f_{42}}{a + b(x_3)} - \frac{2(\beta + f_w)f_{41}(x_3)}{a + b(x_3)}\right) + \frac{e_{43}(x_3)}{a + b(x_3)} + x_4 \times \left(\frac{e_{44}(x_3, x_4)}{a + b(x_3)} + \frac{2\beta(f_{41}(x_3) - f_{42})}{a + b(x_3)}\right) + v_l \times \frac{\alpha(f_{41}(x_3) - f_{42})}{a + b(x_3)} + v_r \times \frac{\alpha(f_{41}(x_3) - f_{42})}{a + b(x_3)} \quad (4.58)$$

Also let

$$e_{m42}(x_3) = 2\beta f_{42} - 2(\beta + f_w)f_{41}(x_3),$$
$$e_{m44}(x_3, x_4) = e_{44}(x_3, x_4) + 2\beta(f_{41}(x_3) - f_{42}),$$
$$f_{m41}(x_3) = \alpha(f_{41}(x_3) - f_{42})$$

and

$$f_{m42}(x_3) = \alpha(f_{41}(x_3) - f_{42}),$$

which can simplify Eq.(4.58) as follows:

$$\dot{x}_4 = \frac{e_{m42}(x_3)}{(a + b(x_3))} \times x_2 + \frac{e_{43}(x_3)}{(a + b(x_3))x_3} \times x_3 + \frac{e_{m44}(x_3, x_4)}{(a + b(x_3))} \times x_4 + \frac{f_{m41}(x_3)}{(a + b(x_3))} \times v_l + \frac{f_{m42}(x_3)}{(a + b(x_3))} \times v_r. \tag{4.59}$$

In summary, from the above derivations, the nonlinear state-space model of the two-wheeled robot with voltage input control is represented as the following:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{e_{m22}(x_3)}{a + b(x_3)} & \frac{e_{23}(x_3)}{(a + b(x_3))x_3} & \frac{e_{m24}(x_3, x_4)}{a + b(x_3)} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{e_{m42}(x_3)}{a + b(x_3)} & \frac{e_{43}(x_3)}{(a + b(x_3))x_3} & \frac{e_{m44}(x_3, x_4)}{a + b(x_3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{f_{m21}(x_3)}{a + b(x_3)} & \frac{f_{m22}(x_3)}{a + b(x_3)} \\ 0 & 0 \\ \frac{f_{m41}(x_3)}{a + b(x_3)} & \frac{f_{m42}(x_3)}{a + b(x_3)} \end{pmatrix} \begin{pmatrix} v_l \\ v_r \end{pmatrix}. \tag{4.60}$$

Many dynamical systems require high accuracy to maintain their setpoints, by removing steady-state errors. Therefore, we now proceed to a tracking system, which can be designed to remove the steady-state error in state variable x_1 (wheel angle), to ensure the robot is stabilised at the reference position (tracking a pre-defined trajectory) by adding an integrator of state variable x_1 as

the 5th state-variable of system. Hence, the nonlinear state-space model of the two-wheeled robot with tracking system is represented in the form:

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0 & 0\\ 0 & \frac{e_{m22}(x_3)}{a+b(x_3)} & \frac{e_{23}(x_3)}{(a+b(x_3))x_3} & \frac{e_{m24}(x_3,x_4)}{a+b(x_3)} & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & \frac{e_{m42}(x_3)}{a+b(x_3)} & \frac{e_{43}(x_3)}{(a+b(x_3))x_3} & \frac{e_{m44}(x_3,x_4)}{a+b(x_3)} & 0\\ 1 & 0 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{pmatrix} \quad (4.61)$$

$$+\begin{pmatrix}0 & 0\\ \frac{f_{m21}(x_3)}{a+b(x_3)} & \frac{f_{m22}(x_3)}{a+b(x_3)}\\ 0 & 0\\ \frac{f_{m41}(x_3)}{a+b(x_3)} & \frac{f_{m42}(x_3)}{a+b(x_3)}\\ 0 & 0\end{pmatrix}\begin{pmatrix}v_l\\ v_r\end{pmatrix}.$$

4.4 Linearisation of the Two-Wheeled Robot Model

A suitable nonlinear state-space model of the two-wheeled robot was deployed in the previous section and the dynamic equations were complicated. Thus, it is possible to simplify the mathematical model of a nonlinear system by applying the well-known method: linearisation. Linearisation is a technique to approximate a nonlinear system into a linear form, (where in the system is operating around an equilibrium point) by applying Taylor series expensions and the signals are considered as small quantities (Ogata & Yang, 1970). The linearised model is suitable for the applications of linear controllers, e.g. PID, LQR and LQG controller.

In this section, the nonlinear model of the self-balancing two-wheeled robot given in Eq.(4.61) is linearised by assuming x_3 and x_4 are small quantities. Then, approximations can be taken as follows:

$$sin(x_3) \approx x_3,\ cos(x_3) \approx 1,\ x_4\, sin(x_3) \approx 0 \ \text{ and } \ sin\,(x_3)^2 \approx 0 \qquad (4.62)$$

Some defined parameters of the $\dot{x}_2$ equations are rewritten accordingly as follows:

$$f_{22L} = 2n^2 J_m - MRL,$$
$$f_{m21L} = \alpha(f_{21} - f_{22L}),$$
$$f_{m22L} = \alpha(f_{21} - f_{22L}).$$
$$e_{23L}(x_3) = x_3 \times (2n^2 J_m LMg - M^2 R L^2 g)$$
$$= x_3 \times e_{23L}\,,$$
$$e_{24L} = 0,$$
$$e_{m22L} = 2\beta f_{22L} - 2(\beta + f_w) f_{21},$$
$$e_{m24L} = 2\beta(f_{21} - f_{22L}),$$

Let $e_{23L} = 2n^2 J_m LMg - M^2 R L^2$

Similarly, the revision of parameters of the $\dot{x}_4$ equation is carried out to obtain

$$f_{41L} = 2n^2 J_m - MRL,$$
$$f_{m41L} = \alpha(f_{41L} - f_{42}),$$
$$f_{m42L} = \alpha(f_{41L} - f_{42}).$$
$$e_{43L}(x_3) = x_3 \times MgL(2n^2 J_m + 2J_w + 2mR^2 + MR^2)$$
$$= x_3 \times e_{43L}\,,$$
$$e_{44L} = 0,$$
$$e_{m42L} = 2\beta f_{42} - 2(\beta + f_w) f_{41L}\,,$$
$$e_{m44L} = 2\beta(f_{41L} - f_{42}),$$

Let $e_{43L} = MgL(2n^2 J_m + 2J_w + 2mR^2 + MR^2)$

Then Eq.(4.39) is rewritten as

$$b_L = 4MLRn^2J_m, \tag{4.63}$$

which transforms Eq.(4.57) to the following:

$$\dot{x}_2 = \frac{e_{m22L}}{(a+b_L)} \times x_2 + \frac{(x_3 \times e_{23L})}{(a+b_L)x_3} \times x_3 + \frac{e_{m24L}}{(a+b_L)} \times x_4 + \frac{f_{m21L}}{(a+b_L)} \times v_l + \frac{f_{m22L}}{(a+b_L)} \times v_r. \tag{4.64}$$

Also, Equation(4.59) can be rewritten as follows:

$$\dot{x}_4 = \frac{e_{m42L}}{(a+b_L)} \times x_2 + \frac{(x_3 \times e_{43L})}{(a+b_L)x_3} \times x_3 + \frac{e_{m44L}}{(a+b_L)} \times x_4 + \frac{f_{m41L}}{(a+b_L)} \times v_l + \frac{f_{m42L}}{(a+b_L)} \times v_r \tag{4.65}$$

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

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{e_{m22L}}{a+b_L} & \frac{e_{23L}}{a+b_L} & \frac{e_{m24L}}{a+b_L} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & \frac{e_{m42L}}{a+b_L} & \frac{e_{43L}}{a+b_L} & \frac{e_{m44L}}{a+b_L} & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{f_{m21L}}{a+b_L} & \frac{f_{m22L}}{a+b_L} \\ 0 & 0 \\ \frac{f_{m41L}}{a+b_L} & \frac{f_{m42L}}{a+b_L} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v_l \\ v_r \end{pmatrix}. \tag{4.66}$$

Chapter 5

Linear Control Designs and Implementations

5.1 Introduction

This chapter illustrates a modern linear control strategy widely used in controlling linear systems, known as linear quadratic regulator (LQR) and its widespread application extension, named linear quadratic Gaussian (LQG). The chapter is organised as follows: the LQR is applied to control an inverted pendulum on a cart and a two-wheeled robot will be presented in Section 5.2, and the addition of the LQG with the two-wheeled robot system will be shown in Section 5.3. Each section presents the control strategy, controllability and observability analysis, simulation, and experimental results, respectively.

5.2 Linear Quadratic Regulator (LQR)

5.2.1 Linear Quadratic Regulator (LQR) Theory

LQR is a traditional solution of optimal control formulation. The theory is considered in terms of continuous or discrete problem. This research, however, will be concentrated on the continuous version. The further extensions applied the LQR techniques to inverted pendulum systems can be found in Yamamoto (2009), Fang (2014), Prasad, Tyagi, & Gupta (2014),and da Silva & Sup (2017).

The cost function of the quadratic is defined by

$$J = \int_0^\infty (\boldsymbol{x^T Q x} + \boldsymbol{u^T R u}) dt \tag{5.1}$$

where the matrix $\boldsymbol{Q}$ is positive semi-definite, and the matrix $\boldsymbol{R}$ is positive definite applied to discipline states and applied to control objective, respectively; moreover, $\boldsymbol{x}$ is an $n-$ state variable vector, $\boldsymbol{u}$ is an $m-$ control variable vector (Harrison, 2003).

The general linear time invariant system is in the form of

$$\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu}, \tag{5.2}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ form the pair of controllability matrix.

The control defined above leads to the linear optimal feedback control given by

$$\boldsymbol{u} = -\boldsymbol{Kx} \tag{5.3}$$

$$\boldsymbol{K} = \boldsymbol{R^{-1} B^T P} \tag{5.4}$$

where the matrix $\boldsymbol{K}$ provides optimal feedback gains to the system and the matrix $\boldsymbol{P}$ is the solution of the algebraic matrix Riccati equation:

$$0 = \boldsymbol{PA} + \boldsymbol{A^T P} - \boldsymbol{PBR^{-1} P} + \boldsymbol{Q}. \tag{5.5}$$

Computationally, Eq. (5.5) can be solved by applying the linear quadratic regulator function in MATLAB function as shown in Appendix A.5.1.

Therefore, the optimal control is implemented by substituting Eqs. (5.3) and (5.4) into Eq. (5.2):

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{Ax} - \boldsymbol{B}(\boldsymbol{R^{-1} B^T P x}) \\ &= (\boldsymbol{A} - \boldsymbol{BR^{-1} B^T P})\boldsymbol{x}, \end{aligned} \tag{5.6}$$

which obtains stabilisation of the control system in the linearisation term subject to the condition of observability $(\boldsymbol{Q}^{0.5}, \boldsymbol{A})$ (Xu, Zhang, & Carbone, 2017).

In terms of the single inverted pendulum on a cart model analysed in Chapter 4, the 4th order linearised state-space model from Eq. (4.12), which contains the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, can be calculated the state feedback gain matrix $\boldsymbol{K}$ shown in Figure 5.1 by selecting the $\boldsymbol{Q}$ and $\boldsymbol{R}$ weight matrices based on a desire to minimize wheel angle (x_1) and pitch angle (x_3) of the robot.

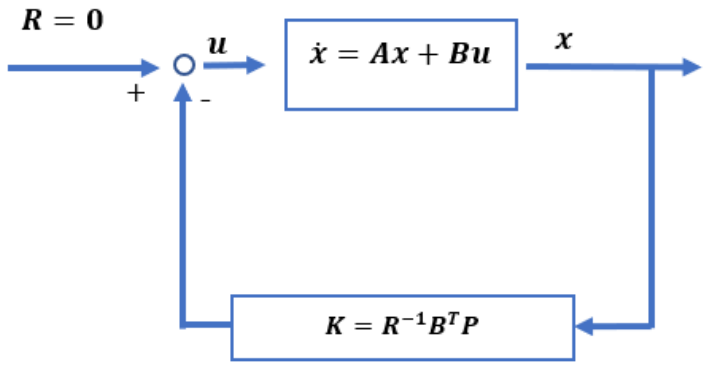


Figure 5.1: Structure of the linear quadratic regulator (LQR) (Burns, 2001)

In the case of the two-wheeled balancing robot, the dynamical model with 5 state variables from Eq. (4.66) is investigated in this research by combining it with a tracking system, as demonstrated in Figure 5.2.

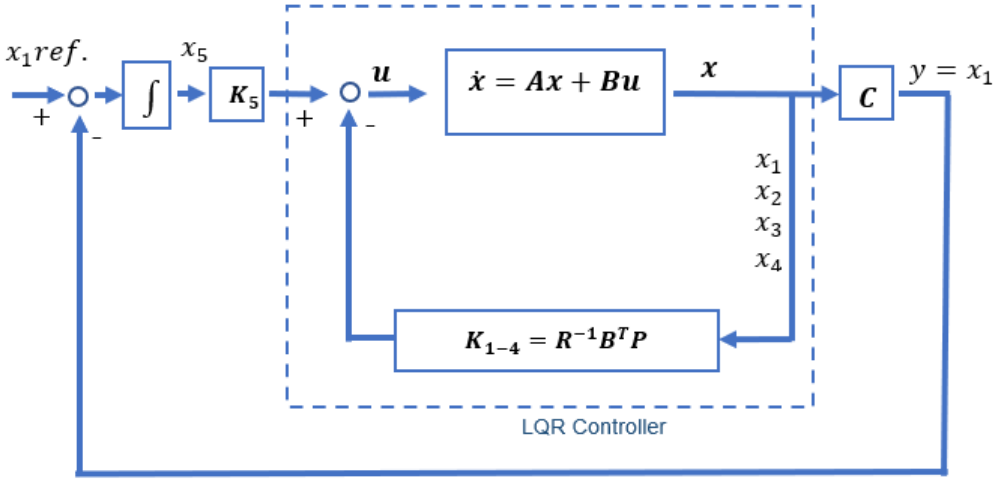


Figure 5.2: Structure of the linear quadratic regulator (LQR) and tracking system (Burns, 2001)

5.2.2 Controllability

The controllability of a dynamic system can be examined by a controllability test, which is applied to the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. The model matrices are used to form the controllability matrix $\mathcal{C}$ as shown below (Dutton, Thompson, & Barraclough, 1997):

$$\mathcal{C} = [\boldsymbol{B}\ \boldsymbol{AB}\ \ldots\ \boldsymbol{A^{n-1}B}] \tag{5.7}$$

where $\boldsymbol{n}$ represents the number of state variables of the system.

Therefore, for the 4th order inverted pendulum on a cart system, the controllability test matrix is

$$\mathcal{C} = [\ \boldsymbol{B}\ \boldsymbol{AB}\ \boldsymbol{A^2B}\ \boldsymbol{A^3B}\ ]. \tag{5.8}$$

The required system model is said to be completely state controllable if the rank of $\mathcal{C}$ is equal to the number of rows in matrix $\boldsymbol{B}$ and thus $\mathcal{C}$ is full rank (Dutton,

Thompson, & Barraclough, 1997). Hence, the controllability is directly affected by the system's state-space representations.

The rank of the controllability matrix was determined using MATLAB (see in Appendix A.5.2). The result given is Rank ($\boldsymbol{\mathcal{C}}$) = 4, equal to the number of rows in matrix $\boldsymbol{B}$; therefore, the system is said to be completely state controllable.

Likewise, the controllability test of a two-wheeled robot model with 5-state variables is given by

$$\boldsymbol{\mathcal{C}} = [\, \boldsymbol{B}\ \boldsymbol{AB}\ \boldsymbol{A^2B}\ \ \boldsymbol{A^3B}\ \boldsymbol{A^4B}\,]. \tag{5.9}$$

Hence, the system is completely controllable when Rank ($\boldsymbol{\mathcal{C}}$) = 5.

5.2.3 Simulation Results

In this section, the LQR will be used to control two linearised systems: the inverted pendulum model and the two-wheeled robot model. The control objective is to balance both the inverted pendulum and the robot in the otherwise unstable vertical upright reference position.

The simulation is conducted using MATLAB script files rather than in Simulink, for faster computation. The MATLAB programme demonstrated in Appendix A.5.3 for an inverted pendulum system and Appendix A.5.4 for the TWR system, which the structure of LQR feedback control in Figure 5.2 was applied.

5.2.3.1 The Effect of Matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$

The matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are weighting matrices for the states and the control signal, respectively. The trial and error method can be applied to select $\boldsymbol{Q}$ and $\boldsymbol{R}$ until the

desired transient response is achieved (Van De Vegte, 1990). Likewise, Yamamoto (2009) stated the experimental trial and error method, which was used to select the suitable matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ for balancing a TWR. Next, the matrices were implemented in the simulation of stabilisation with an initial pitch angle (x_3) of 15° performed in MATLAB.

- **Inverted Pendulum on a Cart System**

In the case of the single inverted pendulum model, the stabilising system is investigated by varying the state weighting matrix $\boldsymbol{Q}$ and control weighting quantity R (Remark: R is a scalar quantity in the single inverted pendulum model as a scalar control is used). The results are presented in Figures 5.3-5.7.

Firstly, it can be seen in Figure 5.3 that an increase of Q_{11} affects the x_1 graph significantly by reducing the settling time and also decreasing the cart movement for stabilising the pendulum. However, the undershoot of pendulum angle x_3 is larger when Q_{11} is raised.

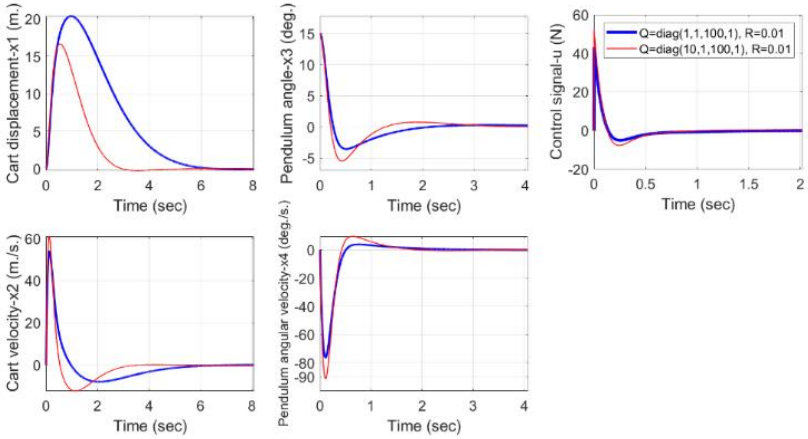


Figure 5.3: Dynamical evolution of state variable x_1-x_4 and control signal u over time, with varying Q_{11} values

Next, there are slight effects when Q_{22} is raised, as shown in Figure 5.4. For instance, the cart takes a longer time, travelling to the original position in the x_1 graph.

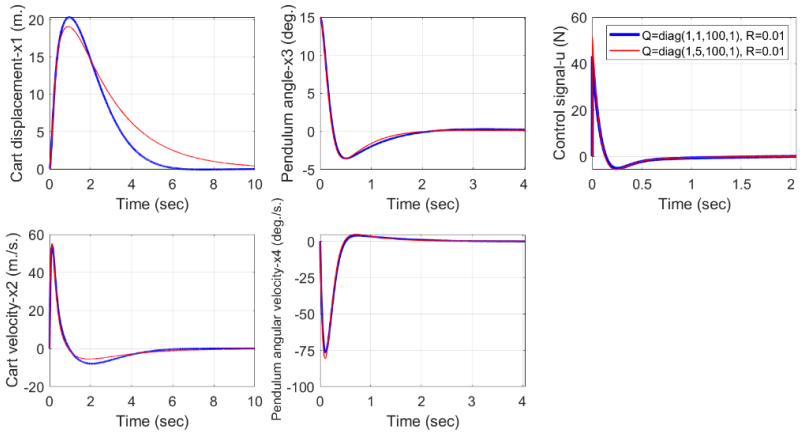


Figure 5.4: Dynamical evolution of state variable x_1-x_4 and control signal u over time, with varying Q_{22} values

Then, in Figure 5.5, when the Q_{33} value is set to be 10 times higher, the undershoot of pendulum angle x_3 is decreased. Similarly, the overshoot of cart displacement x_1 is dropped.

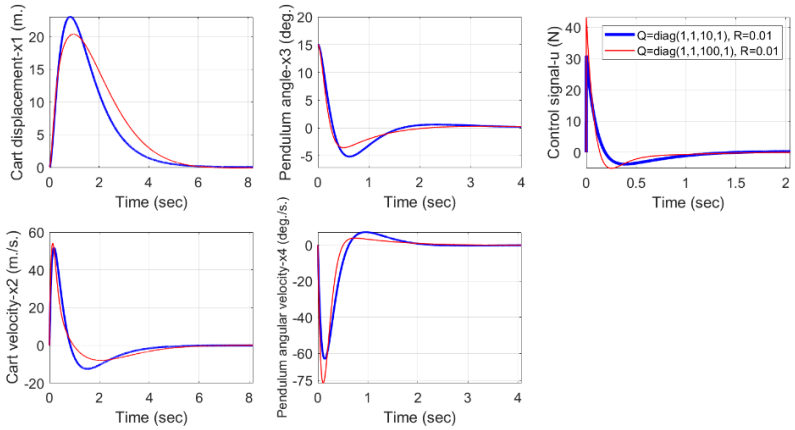


Figure 5.5: Dynamical evolution of state variable x_1-x_4 and control signal u over time, with varying Q_{33} values

Moreover, in Figure 5.6, the settling time in the pendulum angle graph takes longer to settle to the reference position and the amplitude of x_1 graph is larger when the Q_{44} is raised.

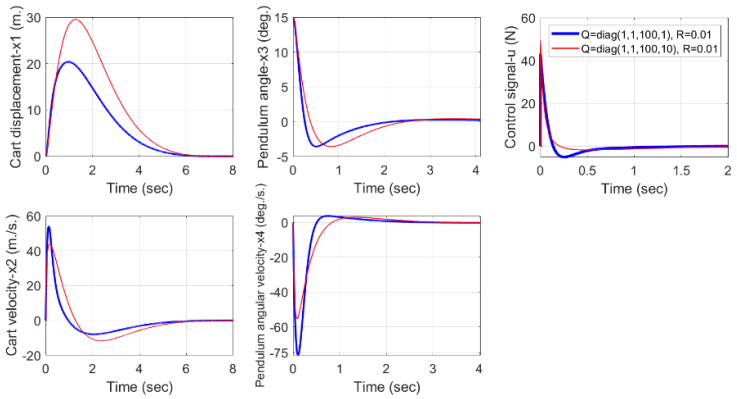


Figure 5.6: Dynamical evolution of state variable x_1-x_4 and control signal u over time, with varying Q_{44} values

Additionally, an increase of control weighting R leads to significant decrease of the cart displacement amplitude, as well as reduction of the settling time in the x_3 graph. Furthermore, the magnitude of control signal is doubled, as shown in Figure 5.7.

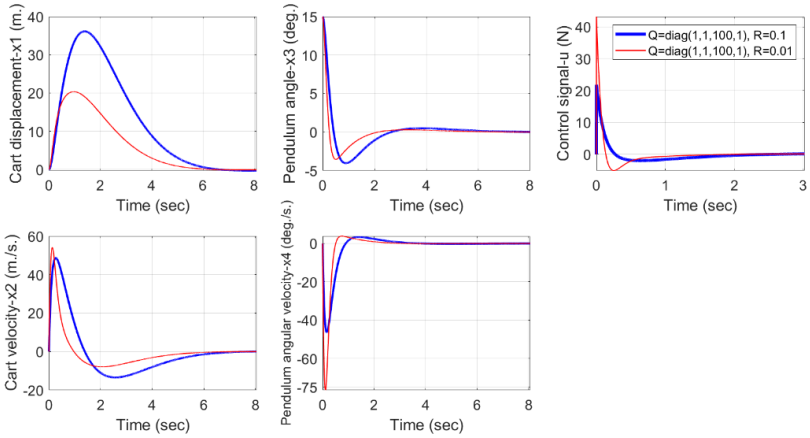


Figure 5.7: Dynamical evolution of state variable x_1-x_5 and control signal u over time, with varying R values

To summarise, when increasing certain parameters in the $\boldsymbol{Q}$ matrix, the cart displacement decreases; by contrast, it obtains a larger pendulum swing angle before stabilising the system, such as Q_{11}. However, the advantage of increasing Q_{33} is to reduce an undershoot in pendulum angle x_3 and also to improve an overshoot in cart displacement x_1 in the same way. Moreover, a smaller amount of R demonstrates a decrease in cart displacement amplitude and a decrease of the pendulum angle undershoot. Therefore, the $\boldsymbol{Q}$ = diag {1,1,100,1} and R =0.01 are chosen to simulate the single inverted pendulum model in the implementation step next.

- **Two-Wheeled Robot System**

In terms of the two-wheeled robot model, the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ will be selected to apply to the prototype LEGO EV3 two-wheeled robot. Other physical parameters are given in Table 4.1. Therefore, the effect of changing weight matrices will be investigated more intensively than the inverted pendulum case. The outcomes are given in Figures 5.8-5.15. (Note that the control signal graph presents merely one signal from two motors, which have the same values).

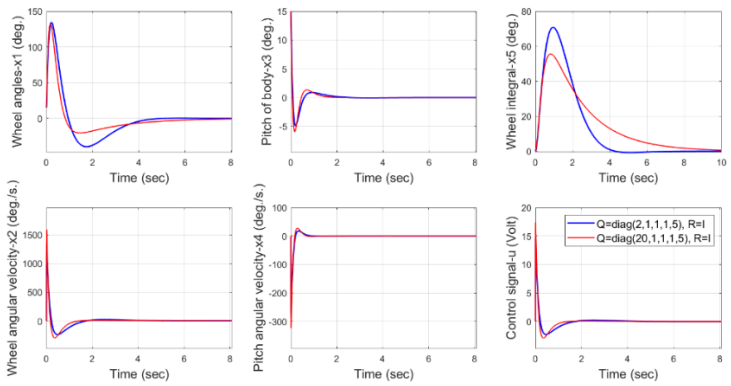


Figure 5.8: Dynamical evolution of state variables x_1-x_5 and control signal u over time, with varying Q_{11} values

Firstly, Figure 5.8 presents the effect of varying Q_{11} values when the rest of matrix $\boldsymbol{Q}$ ($Q_{22} - Q_{55}$) are fixed. Q_{11} corresponds to the first state variable, which is the wheel angle (x_1). The blue plot represents $Q_{11} = 2$ and the red plot represents $Q_{11} = 20$. It can be seen that by increasing the value of Q_{11} (weighting on x_1), the wheel angle x_1 and the integral of wheel angle x_5 display reduced magnitude of deviation from the required values; on the other hand, the time taken to reach the steady-states is marginally longer. Additionally, the pitch angle (x_3) is slightly affected, showing increased oscillation when the value of Q_{11} rises. It is obvious that both peak magnitudes of the wheel angular velocity x_2 ($\sim 1500^{\circ}/s$) in the simulations shown in Figure 5.8 exceed the hardware specification (LEGO EV3 motor) at approximately $1{,}000^{\circ}/s$. Similarly, the control signals from both Q_{11} peak at around 18 V which are beyond the Lego EV3 motor voltage limit at 8.3 V (see in Chapter 3 for details about hardware specification). In these simulations, the input motor voltage is not constrained to any range, therefore the generated magnitudes of the state variables could also exceed physically realistic ranges. The effect of limiting the input motor voltages in simulation and in Lego EV3 robot implementation will be discussed in later sections.

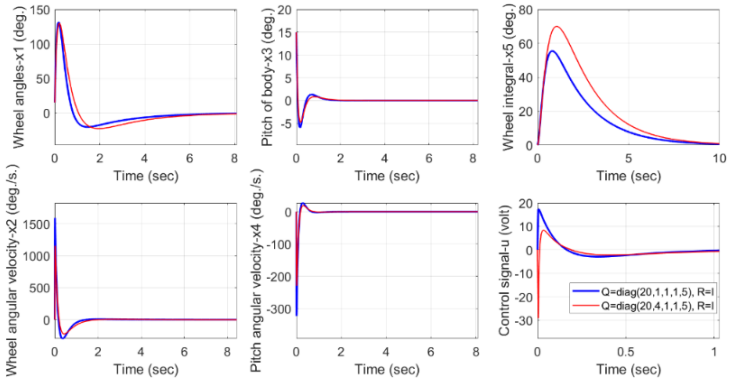


Figure 5.9: Dynamical evolution of state variable x_1-x_5 and control signal u over time, with varying Q_{22} values

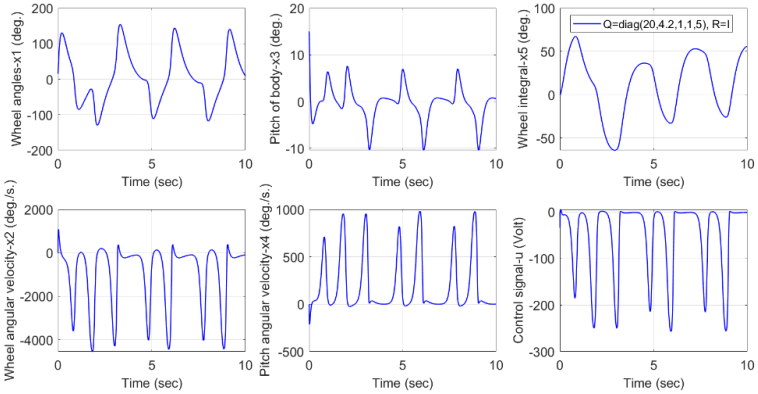


Figure 5.10: Unstable system when the Q_{22}=4.2

Secondly, increasing Q_{22} parameter produces some minor effects to the state variables x_1 and x_3 in Figure 5.9. However, there is a more significant effect to state variable x_5 which could be seen as an increase in the peak amplitude, and the undershoot of control signal u appears when the Q_{22} is increased. As observed from simulation experiments, when Q_{22} varies within the range of [1, 4], the controlled system's response is stable. However, when Q_{22} is outside this range, for example, as shown in Figure 5.10, when Q_{22} = 4.2, the closed-loop system appears to be unstable.

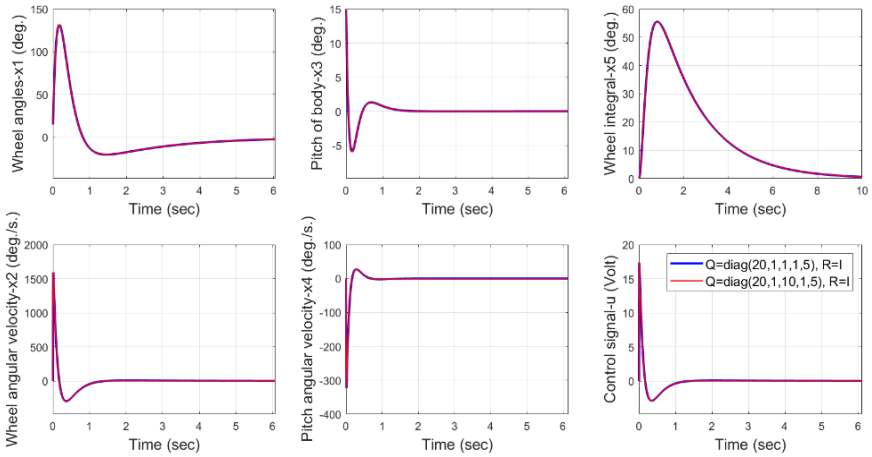


Figure 5.11: Dynamical evolution of state variable x_1-x_5 and control signal u over time, with varying Q_{33} values

Moreover, Figure 5.11 presents slight differences on all state variables and the control, when Q_{33} varies as shown.

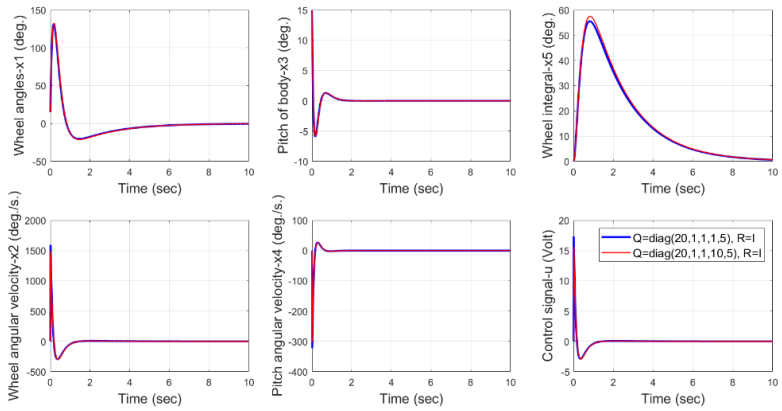


Figure 5.12: Dynamical evolution of state variable x_1-x_5 and control signal u over time, with varying Q_{44} values

Likewise, the changing of Q_{44} parameter does not influence the state variables and control signal, as shown in Figure 5.12.

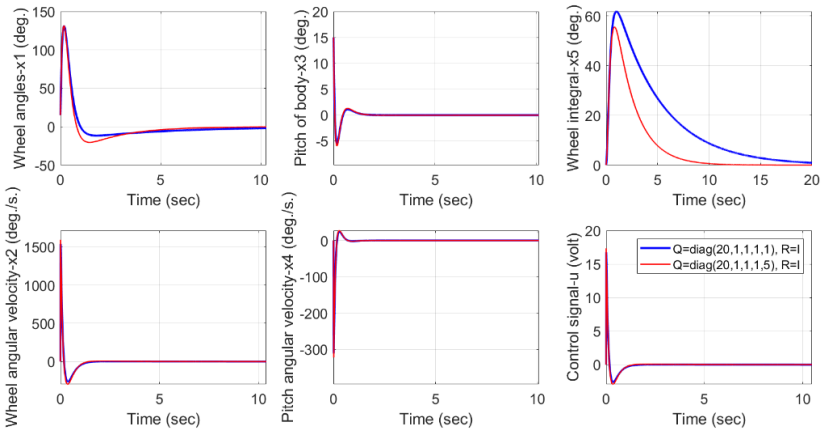


Figure 5.13: Dynamical evolution of state variable x_1-x_5 and control signal u over time, with varying Q_{55} values

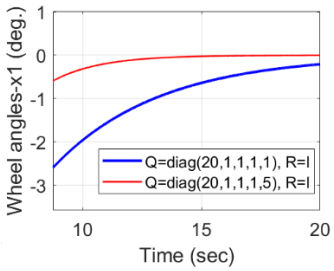


Figure 5.14: Dynamical evolution of state variable x_1 between 10 - 20 s., with varying Q_{55} values

Additionally, the Q_{55} factor relates to the state variable x_5 (the integral of wheel angle, used to track horizontal distance of the robot from starting point) is changed with results shown in Figure 5.13. The noticeable difference when increasing increasing Q_{55} illustrates that steady-state error and magnitude are reduced, which is the main purpose of introducing the state variable x_5 to monitor the time spent by robot moving back to the reference position. Moreover, the wheel angle x_1 is influenced by increasing Q_{55}. It can be seen that the robot is taken a time faster back to the centre position in state variable x_1 shown in Figure 5.14.

In terms of the effect of modifying matrix $\boldsymbol{R}$, by contrast, the results are shown in Figure 5.15, when $\boldsymbol{Q}$ is fixed.

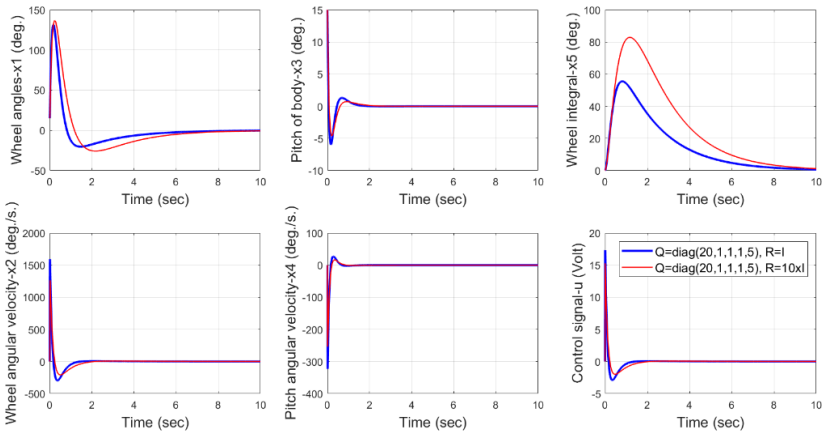


Figure 5.15: Dynamical evolution of state variable x_1-x_5 and control signal u over time, with varying $\boldsymbol{R}$ values

The primary function of $\boldsymbol{R}$ is to supervise the control signal u**.** When the value of $\boldsymbol{R}$ is increased ten times from $\boldsymbol{R} = \boldsymbol{I_{2\times 2}}$ to $\boldsymbol{R} = \boldsymbol{10 I_{2\times 2}}$**,** the reduction in the amplitude of control u is observed. Moreover, the oscillation of wheel angular velocity x_2 and the pitch angle x_3 are slightly reduced, as shown in Figure 5.15. Noticeably, the displacement configuration x_1 in the horizontal axis takes longer to reach steady state, as the voltage of the motor (control signal) driving the robot is dropped. The increase in matrix $\boldsymbol{R}$ achieved some advantages to the system with reduced voltage demand.

Therefore, after conducting trial and error tests on matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, the $\boldsymbol{Q}$ =diag{20,1,1,1,5} and $\boldsymbol{R} = 10\boldsymbol{I_{2\times 2}}$ were selected to implement in the simulation. This parameter set requires low voltage to control system, appropriate to the battery of LEGO EV3; moreover, simulation results using these factors Q_{11} and

Q_{55} present satisfying performance for system stabilisation, in terms of requiring shorter wheel displacement before reaching the reference position.

5.2.3.2 Simulations of IP and TWR without Input Saturations

- **Inverted Pendulum Simulation Results**

Now, the single inverted pendulum model under unconstrained control input condition, represented by Eq. (4.12) is stabilised starting from several different initial pendulum angles x_3. The simulation results are presented in Figures 5.16-5.17.

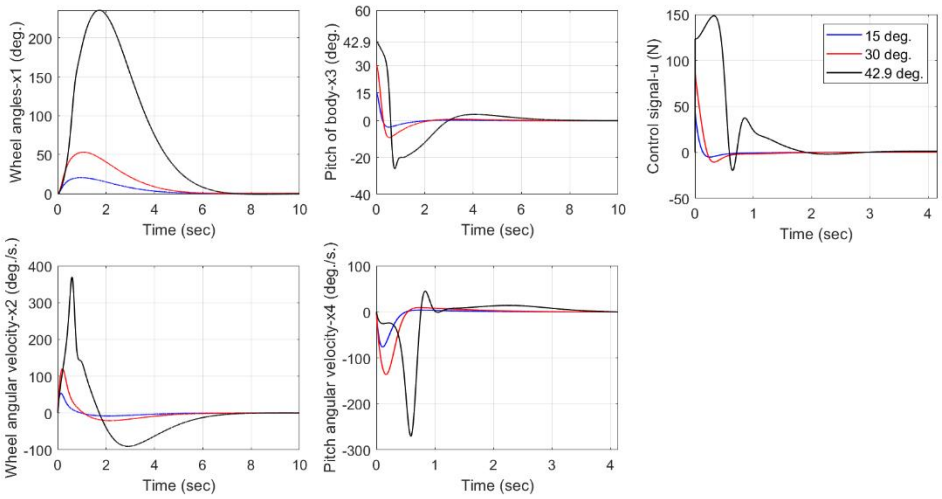


Figure 5.16: The stabilisation on different initial pendulum angles (x_3)

Figure 5.16 demonstrates the ability of the LQR technique to control and stabilise a single inverted pendulum on a cart from three initial pendulum angles: $15^{\circ}, 30^{\circ}$ and 42.9°. It can be seen from the cart displacement (x_1) graph that the amplitude rises when initial pendulum angle becomes wider; moreover, the

undershoot magnitude in the pendulum angles graph x_3 also becomes larger. Furthermore, there are strong oscillations in the other state variable and the control graphs when the initial pendulum angle is 42.9° (because the control system reaches the limit for stabilisation). Beyond this angle, the inverted pendulum model will crash as the linear optimal controller fails to stabilise it, as shown in Figure 5.17.

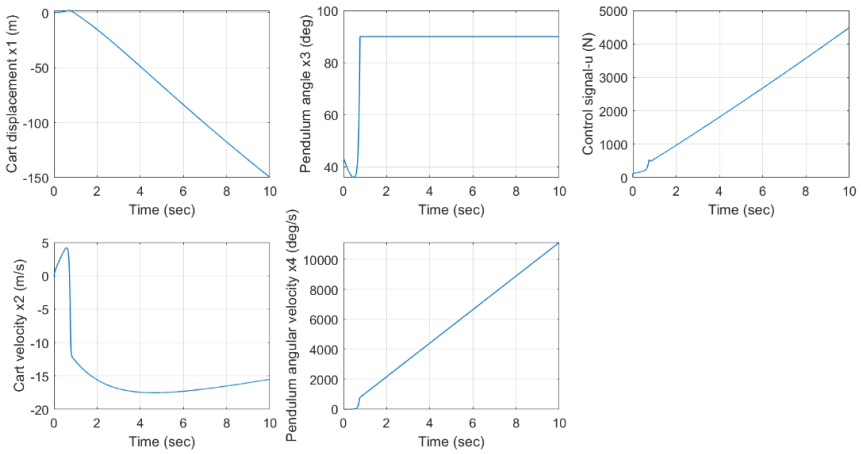
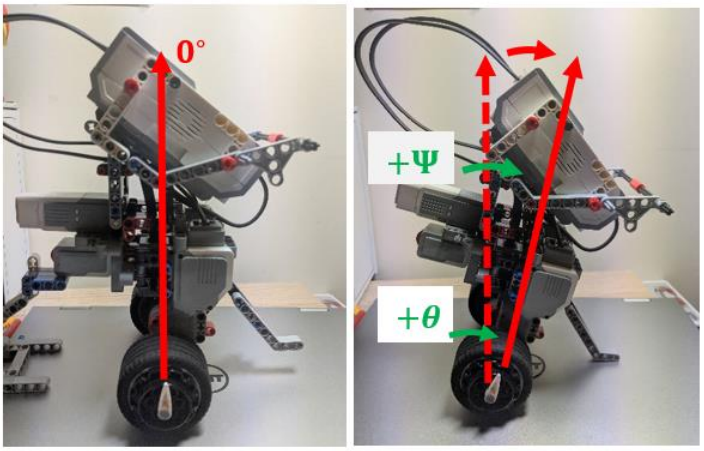


Figure 5.17: Unstable system responses from initial pendulum angle $x_3 = 43°$.

- **Two-Wheeled Robot Simulation Result**

This subsection demonstrates the simulation of the balancing two-wheeled robot model represented by Eq.(4.66) with different initial pitch angles (x_3). The robot in Figure 5.18 (a) shows the robot's reference position before setting the initial pitch angle $(\Psi = x_3)$ to non-zero and Figure 5.18 (b) presents the initial pitch angle shifted for simulation. It can be seen that it is not merely the initial pitch angle (Ψ) shifted, but the wheel angle (θ) is also shifted with the same value together in the hardware implementation. Therefore, the wheel angle will be

initialised at the same value as the initial pitch angle (Ψ) in the simulation. The direction of the wheel and pitch angles are presented in Figure 5.18. When the robot moves forward (towards right-hand side in the figure), the direction of wheel and pitch angles are positive.



(a) *(b)*

Figure 5.18: (a) Robot's reference position before setting the initial pitch angle (Ψ), (b) Wheel angle (θ) shifted after the initial pitch angle set.

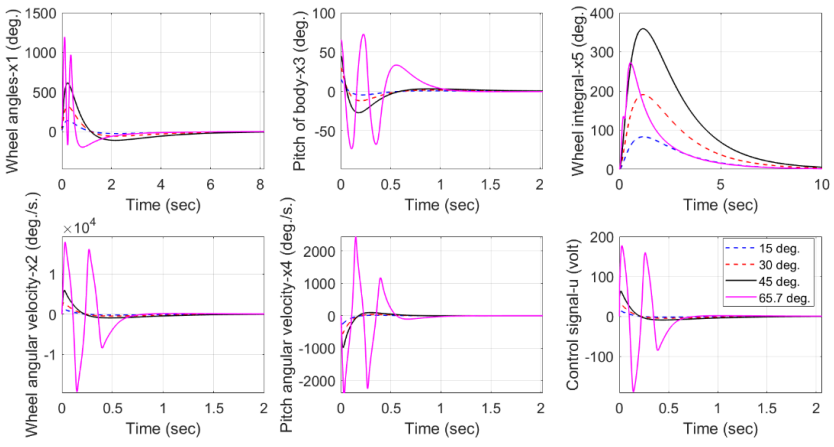


Figure 5.19: The stabilisation from different initial pitch angles (x_3)

In Figure 5.19, the blue curve represents the responses of the state variables and the control signal when starting from a narrow initial pitch angle at 15° followed by increases of the initial angle to 30°, 45° and 65.7° shown by red, black and pink curves, respectively. The maximum deviation of all state variables and the control signal magnitude are raised when the initial pitch angle is increased. In particular, note the oscillations on the pitch angle and the pitch angular velocity from the initial pitch angle 65.7° as the capability of the LQR technique to stabilise this nonlinear system reaches its limit. Furthermore, the overshoot of wheel angle reaches approximately $1{,}200^{\circ}$ at the maximum initial pitch angle, which means that the TWR diverges from the reference position by approximately 0.56 m.

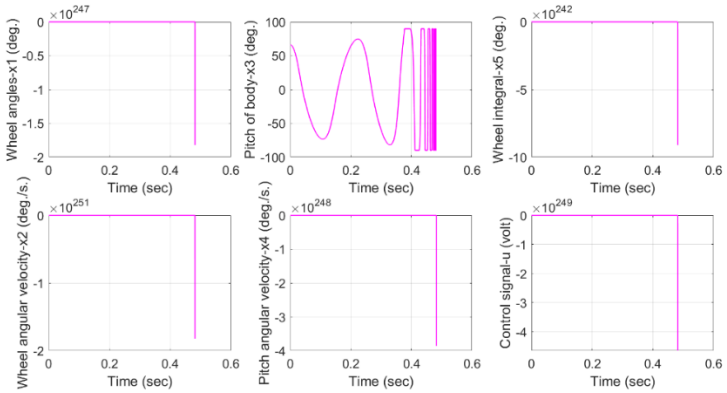


Figure 5.20: The unstable system response at initial pitch angles $x_3 = 65.8°$

Figure 5.20 exhibits the failure of system control at the initial pitch angle 65.8°. The magnitudes of other state variables x_1, x_2, x_4, x_5 and control signal are at unrealistically high levels (approaching infinity), and the pitch angle oscillated continually; moreover, the simulation stopped after only 0.5 seconds as the instability caused the simulation to crash.

5.2.3.3 Simulations of TWR with Input Saturations

In the section 5.2.3.2, it can be seen that the control signal needed voltage over the limitation of any real-world implementation, for example, when the LEGO EV3 robot was used. Therefore, before implementing using hardware, simulations including input saturations need to be conducted first.

This section illustrates the hard constraints applied as saturations on the control signal using the LEGO EV3 motor range: -8.3 to 8.3 V on each motor with the following function:

$$u = \begin{cases} \lambda, & u \geq \lambda \\ u, & |u| \leq \lambda \\ -\lambda, & u \leq -\lambda, \end{cases} \tag{5.10}$$

where $\boldsymbol{\lambda}$ is the control signal saturation limited at 8.3 volts.

The stabilisation simulations of the two-wheel model with control constraints, starting from initial pitch angles 15° and 20° are presented in Figures 5.21-5.23.

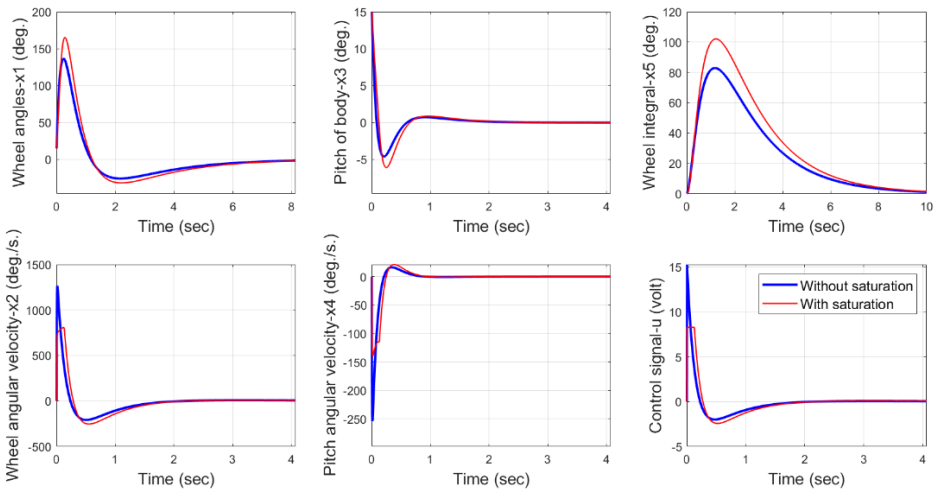


Figure 5.21: Stabilisation from the initial pitch angles (x_3) at 15°, with and without saturation.

At the initial pitch angle 15° in Figure 5.21, the magnitudes in the state variables x_1, x_3 and x_5 increase, compared with the unconstrained signals; however, the derivative in signals x_2 and x_4 , and also the control input u present noticeable cut-off magnitudes as the input saturation is provided. For instance, the state variable x_4 presents an undershoot reaching -250°/s when simulated with the unconstrained control input; however, the undershoot is cut-off at approximately -140°/s when an input saturation is applied (Note, the maximum angular velocity provided by the LEGO gyroscope is ±440°/s as shown in Table 4.1).

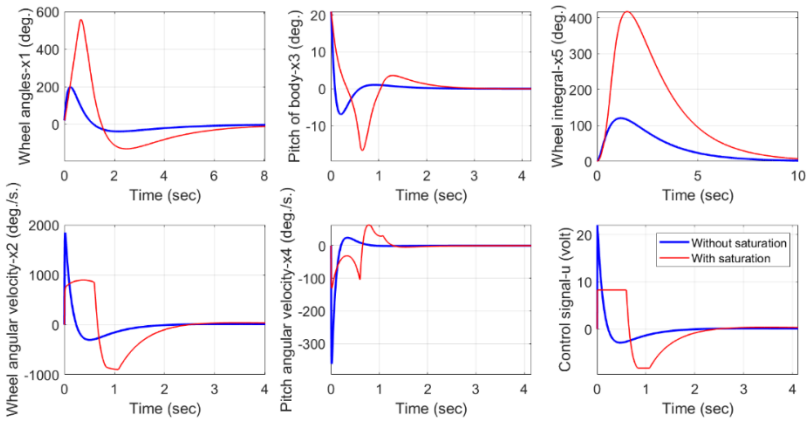


Figure 5.22: Stabilisation of the initial pitch angles (x_3) at 20.9°with and without saturation.

Figure 5.22 presents the maximum initial pitch angle of saturation input implementation at 20.9°. The increase of the initial angle demonstrates different significant outcomes of all constrained signals. Firstly, the magnitude of x_1, x_3 and x_5 in the constrained graphs are more than doubled. Moreover, the obvious oscillations are also present in the pitch angular velocity (x_4)**.** Furthermore, the wheel velocity and control signal noticeably show cut-off features of constraint at

both the maximum and minimum when reaching the saturation limits. After this point (when the initial pitch angle goes above 20.9°), unstable behaviour will appear, as shown in Figure 5.23.

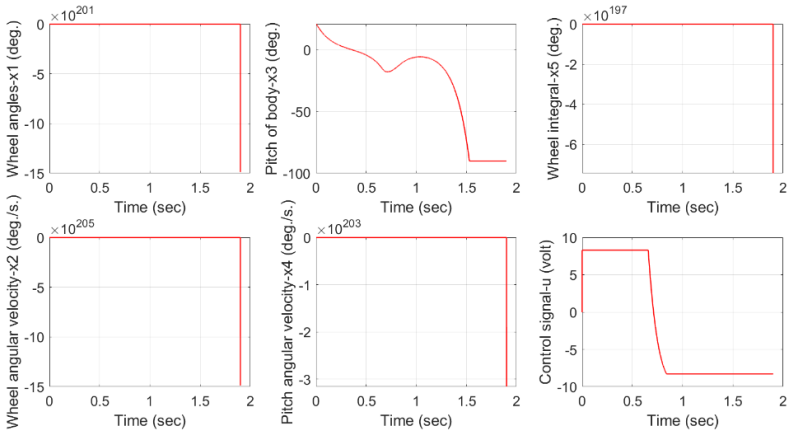


Figure 5.23: Unstable system with input saturation at the initial pitch angles 21°

5.2.4 Experimental Results

With the simulation tests of the self-balancing two-wheeled robot model with the linear-quadratic regulator are completed, hardware implementation of the controlled system using LEGO EV3 robot is conducted. The results from running the Simulink program are detailed in Appendix B (Figure B1.1), including the robot's sensors and motors, LQR feedback control and tracking system block diagrams. Moreover, the experiment is set up in a closed environment with no wind disturbance when implementing the TWR, and the floor is covered by a carpet. Therefore, the outcomes were presented in Figures 5.24-5.26 as follows.

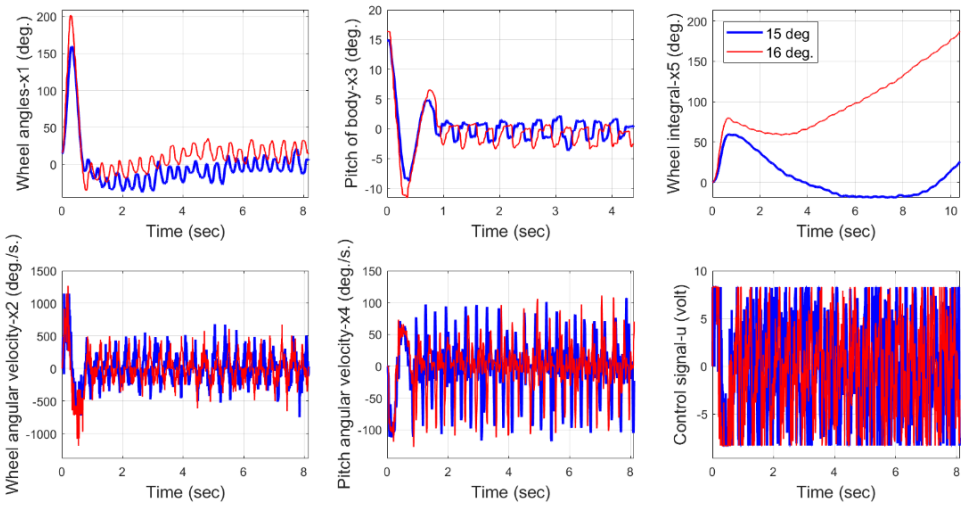


Figure 5.24: The stabilisation from different initial pitch angles $x_3 = 15°$ and 16° implemented on LEGO EV3 robot

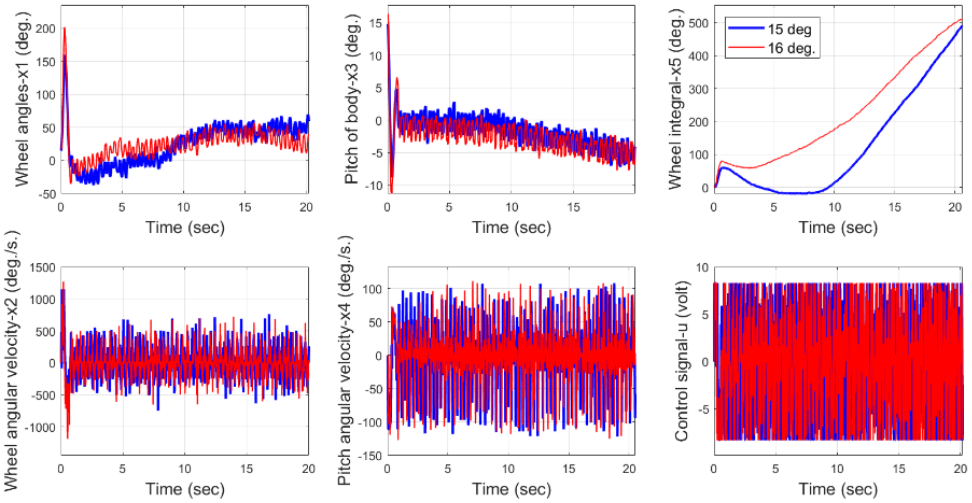


Figure 5.25: The stabilisation on different initial pitch angles x_3 at 15° and 16° implemented on LEGO EV3 robot over 20 seconds

Figure 5.24 illustrates the result of state variables and control signal on two different initial situations. The blue and red graphs present the initial pitch angle at $15^{\circ}$and 16°, respectively. The initial pitch angle 16° demonstrate the limit of LQR method to balance the system in the upright position in the hardware

implementation, which is similar to the simulation at initial pitch angle 20° under input saturation. Furthermore, the magnitude of wheel angle and pitch angle at the initial pitch angle 16° presented higher overshoot signals than 15° as it needed more mobility, when increasing the initial angle for stabilisation. Moreover, the state variable x_5 in the two plots present noticeable divergence caused by gyro sensor error. More details will be described with the other information in Figure 5.25.

In terms of Figure 5.25, the graph shows a longer period of time than Figure 5.24 to present the significant sensor error which caused system instability. The plots of wheel angle x_1 and wheel angle integral x_5 produce upward signal drifts; by contrast, the pitch angle x_3 drifted downwards. This issue is caused by the measurement of gyroscope sensor drift, generating pitch angle error and affecting the wheel angle and wheel angle integral, which is compensating the error. This challenge can be overcome by using a state observer or state estimator. Related theory and subsequent implementation will be given in the next section.

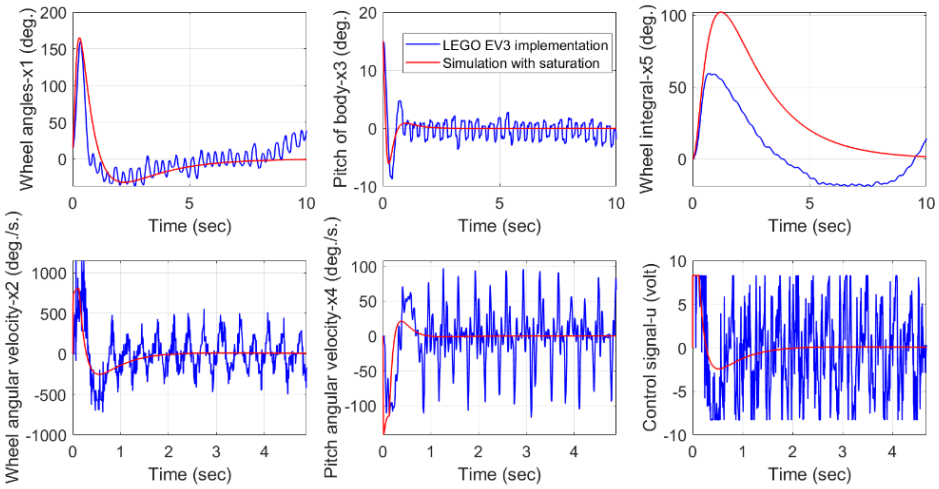


Figure 5.26: The stabilisation implemented by LEGO EV3 robot compared to simulation at the initial pitch angles x_3 15°

Furthermore, the average magnitude of each state variable' response of the Lego EV3 robot is very similar to the simulation result, as in Figure 5.26; for instance, the peak magnitudes of the wheel angles in simulation and experimentation are approximately 160° and 150°, respectively. Likewise, the maximum deviation of pitch angle in the simulation and the experimentation are also similar, at approximately -5° and -8°, respectively. Furthermore, the state variable trends are also comparable during the two methods. Therefore, this hardware experiment can be analysed by the simulation, although there were some differences as the hardware implementation has many interfering factors.

5.3 Linear Quadratic Gaussian (LQG)

5.3.1 Linear Quadratic Gaussian (LQG) Strategy

In Section 5.2, the capability of the LQR was demonstrated. It can be seen in the experimentations that a gyro sensor drift was discovered in the pitch angle, which also affected the wheel angle displacement, causing the system to be unstable. This section will illustrate the strategy that handles the inaccurate sensor measurement used for optimal feedback control. This technique is known as linear quadratic Gaussian (LQG), which is a combination of an LQR and a Kalman filter shown in Figure 5.27. The additional applications of the Kalman filter can be found in Kalman (1960), Hanselmann & Engelke (1988), Chang & Liu (2007) and Du et al. (2017).

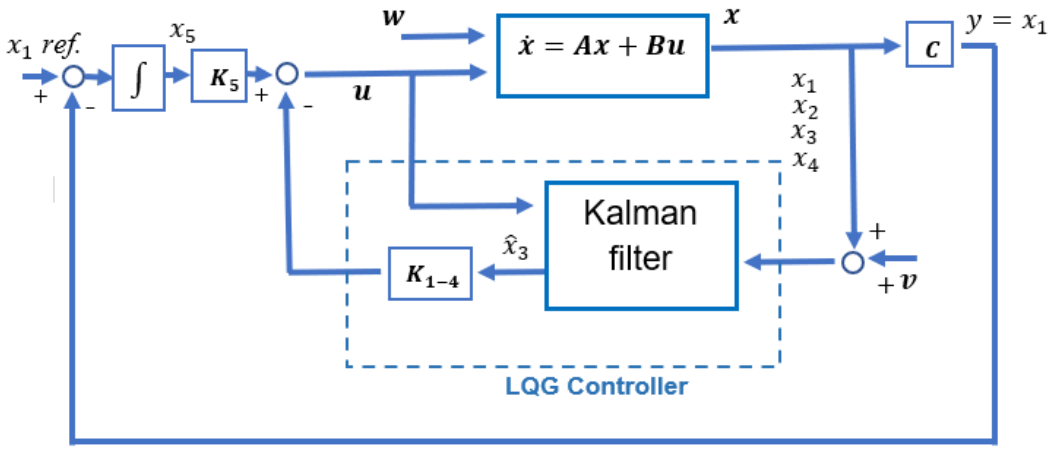


Figure 5.27: LQG Controller and tracking system block diagram (Anderson & Moore, 1989)

The Kalman filter is a mathematical algorithm which achieves minimum state estimation errors, utilising output measurements (Burns, 2001). It was

developed by Kalman and Bucy (Kalman & Bucy, 1961). The advantage of the Kalman filter is the ability to estimate a single state variable instead of full state variables interfered with noise. In order to achieve this, the $\boldsymbol{A}$ and $\boldsymbol{C}$ matrices are required to be observable (Brunton & Kutz, 2019). This research will also be focused on the continuous system, corresponding to the LQR; hence consider a general continuous time-invariant systems given by (Lewis, Xie, & Popa, 2007) as follows:

$$\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu} + \boldsymbol{Gw_n}, \tag{5.11}$$

$$\boldsymbol{y} = \boldsymbol{Cx} + \boldsymbol{v_n}, \tag{5.12}$$

where $\boldsymbol{y}$ is the vector of measured outputs, $\boldsymbol{C}$ is the output matrix, and $\boldsymbol{G}$ is an identity matrix. The $\boldsymbol{w_n}$ and $\boldsymbol{v_n}$ are supposed process noise and measurement noise, respectively, as presented in Figure 5.27.

The Kalman filter supposes that both white noise signals $\boldsymbol{w_n}$ and $\boldsymbol{v_n}$ have zero mean $(\overline{\boldsymbol{w}}_{\boldsymbol{n}} = 0, \overline{\boldsymbol{v}}_{\boldsymbol{n}} = 0)$ and also satisfy covariance functions ($\overline{\boldsymbol{w_n w_n^T}} = \boldsymbol{Q_k}$, $\overline{\boldsymbol{v_n v_n^T}} = \boldsymbol{R_k}$), which are symbolised as $\boldsymbol{w_n} \sim (0,\ \boldsymbol{Q_k})$ and $\boldsymbol{v_n} \sim (0,\ \boldsymbol{R_k})$; moreover, $\boldsymbol{Q_k}$, and $\boldsymbol{R_k}$ are symmetric and positive semi-definite matrices (Lewis, Xie, & Popa, 2007).

The state estimator of dynamic systems is defined as follows (Lewis, Xie, & Popa, 2007):

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A\hat{x}} + \boldsymbol{Bu} + \boldsymbol{K_f}(\boldsymbol{y} - \boldsymbol{C\hat{x}}), \tag{5.13}$$

which can be rewritten as:

$$\dot{\hat{\boldsymbol{x}}} = (\boldsymbol{A} - \boldsymbol{K_f C})\hat{\boldsymbol{x}} + \boldsymbol{Bu} + \boldsymbol{K_f y}, \tag{5.14}$$

where $\boldsymbol{K_f}$ is the Kalman filter gain, which is similar to the feedback gain $\boldsymbol{K}$ of linear quadratic regulator. The schematics of the system with the Kalman filter is shown in Figure 5.28. The Kalman filter gain K_f is given by:

$$\boldsymbol{K_f} = \boldsymbol{PC^T R_k^{-1}} \tag{5.15}$$

where $\boldsymbol{P}$ is the solution of the algebraic Riccati equation defined as follows (Anderson & Moore, 1989):

$$\boldsymbol{AP} + \boldsymbol{PA^T} - \boldsymbol{PC^T R_k^{-1} CP} + \boldsymbol{Q_k} = 0 \tag{5.16}$$

Equation (5.16) can be solved by applying the linear quadratic regulator in a MATLAB function shown in Appendix A.5.5.

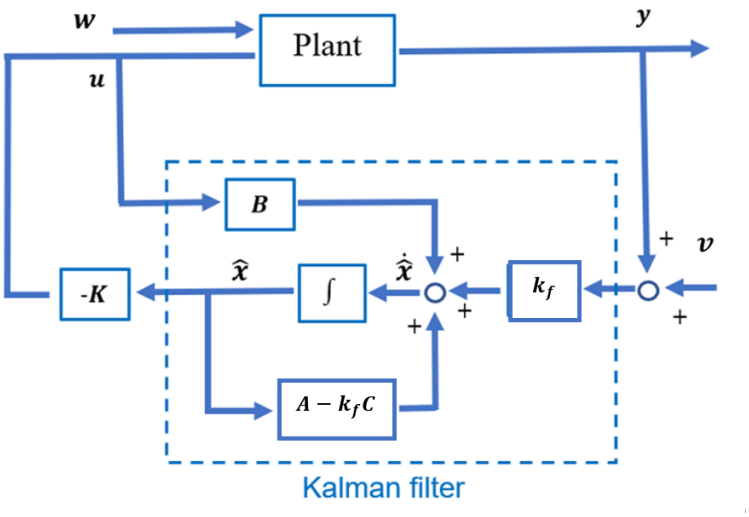


Figure 5.28: Schematic of state-space control using Kalman filter and LQR (Anderson & Moore, 1989)

5.3.2 Observability

As mentioned above, the system must be observable, in order for a Kalman filter design to be possible. In this section, the observability test, which is a test to determine whether the state variables of a system can be estimated by using measurements made at the outputs, will be conducted. The test combines matrices $\boldsymbol{A}$ and $\boldsymbol{C}$, and the matrices are used to form the observability matrix $\mathcal{O}$, as shown below:

$$\mathcal{O} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \vdots \\ \boldsymbol{CA^{n-1}} \end{bmatrix} \tag{5.17}$$

where $\boldsymbol{n}$ represents the number of state variables of the system.

Therefore, for the 4th order single inverted pendulum on a cart system, the observability matrix is

$$\mathcal{O} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \boldsymbol{CA^2} \\ \boldsymbol{CA^3} \end{bmatrix} \tag{5.18}$$

The required system model is said to be completely state observable if the rank of $\mathcal{O}$ equal to the number of columns in the matrix $\boldsymbol{C}$ and thus $\mathcal{O}$ is of full rank (Dutton, Thompson, & Barraclough, 1997).

The rank of the observability matrix can be calculated in MATLAB (see Appendix A.5.6) and the result shows Rank($\mathcal{O}$) = 4, which equals the number of columns in matrix $\boldsymbol{C} = \boldsymbol{I_{4\times4}}$; therefore, the system is said to be completely state observable.

That means the state variables are able to be reconstructed from the system outputs ($\boldsymbol{y}$). This test demonstrates that the linear quadratic Gaussian is applicable for this system.

Similarly, the observability matrix of the two-wheel self-balancing robot with 5-state variables is shown as

$$\mathcal{O} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \boldsymbol{CA^2} \\ \boldsymbol{CA^3} \\ \boldsymbol{CA^4} \end{bmatrix} \tag{5.19}$$

Thus, the system is completely state observable when the Rank($\mathcal{O}$) = 5.

5.3.3 Simulation Results

In this section, LQG will be applied to the control problem of the self-balancing two-wheeled robot model in MATLAB, compared to the LQR. The control objective is to balance the robot in the otherwise unstable vertical upright position; moreover, reduction on drift signal error of the gyro sensor will be considered. The simulation of the LQG is conducted in MATLAB and the script files are included in Appendix A.5.7. (Noticeably, only the state variable x_3 known as the pitch angle will be selected to have the Kalman filter estimation technique applied on.)

5.3.3.1 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

conducted in MATLAB in this section. There are two noise disturbances considered in the LQG control, including the process white noise $\boldsymbol{w_n} \sim(\mathbf{0},\ \mathbf{Q}_k)$ and the measurement white noise $\boldsymbol{v_n} \sim(\mathbf{0},\ \mathbf{R}_k)$. Therefore, only the measurement noise will be implemented with varying $\boldsymbol{R_k}$ values for eliminating the gyro sensor error; by doing this, the process noise will be fixed at $\boldsymbol{Q_k} = \boldsymbol{I_{4\times4}}$.

Firstly, the noise filtering in a gyro sensor will be demonstrated by applying the different $\boldsymbol{R_k}$ factors to reduce the random white noise from MATLAB programme in the state variable x_3. The outcomes of state estimation $\hat{x}_3$ is given in Figures 5.29-5.30.

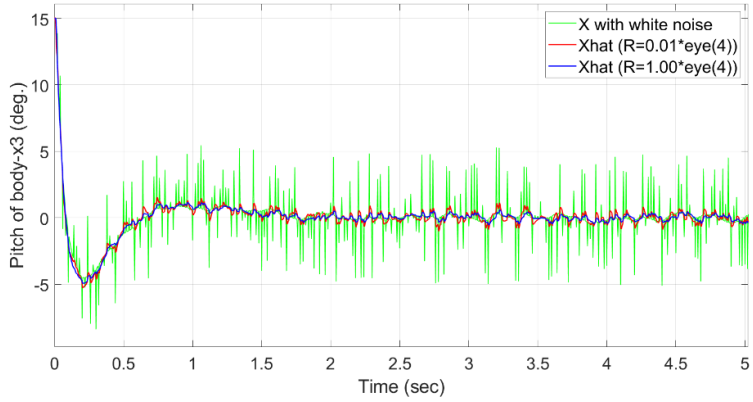


Figure 5.29: Simulation results of noise filtering on a gyro sensor.

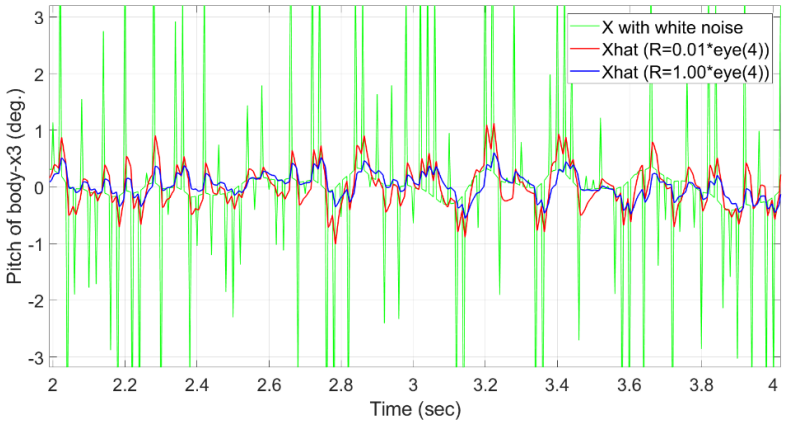


Figure 5.30: The magnified simulation results of noise filtering on a gyro sensor.

In Figure 5.29, the white noise interfered with the state variable x_3 , which is shown in green. The state estimations $\hat{x}_3$ named as **"**Xhat" (Circumflex), are generated using the values of $\boldsymbol{R_k}$ parameters of $0.01\boldsymbol{I_{4\times4}}$ and $\boldsymbol{I_{4\times4}}$ and plotted in red and blue, respectively. The results of noise filtering in the central region are magnified and demonstrated in Figure 5.30. It can be seen that the white noise is decreased by Kalman filter described above. Moreover, the noise signals are reduced by increasing the $\boldsymbol{R_k}$ parameters from $\boldsymbol{R_k = 0.01I_{4\times4}}$ to $\boldsymbol{I_{4\times4}}$. This method has the benefit of stabilising the system interfered with noise.

Secondly, an essential advantage of the linear quadratic Gaussian in this research is that the sensor drift is reduced. Figure 5.31 presents the outcomes of state estimation in a gyro sensor drift condition.

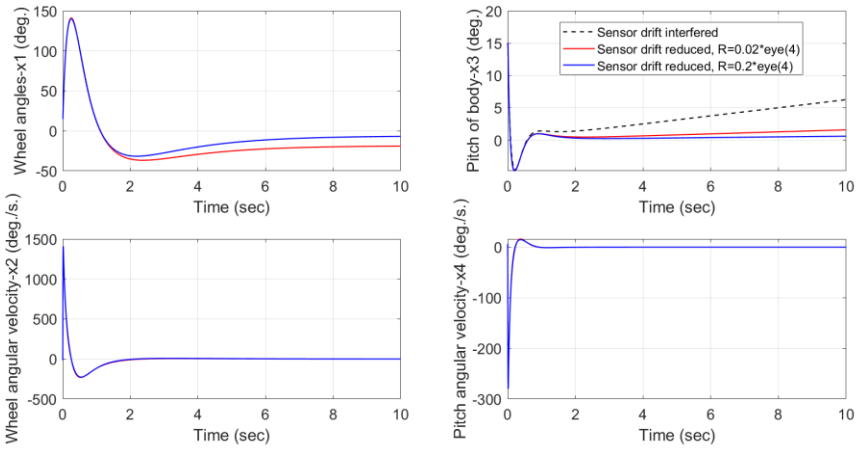


Figure 5.31: The simulation of sensor drift reduction.

Generally, the LQG controller can be applied to full state feedback; by contrast, the state variable x_3 was the only variable selected to have state estimation applied in this simulation. The black dashed curve represents the sensor drift;

moreover, the red and blue plots represent the stabilisation of the system when the state estimation is activated by two $\boldsymbol{R_k}$ parameters in Figure 5.31. Noticeably, the signal drift introduced by the gyro sensor is divergent from the graph of pitch angle (x_3)***;*** on the other hand, the Kalman filter reduces the signal drift and shows signal convergence. The increased $\boldsymbol{R_k}$ factor reduces the signal drift more effectively and also decreases the settled wheel displacement (x_1) from the reference position. This advantage will be applied to the LEGO EV3 robot in the experiment section.

5.3.3.2 Simulations of TWR without Input Saturations

This section demonstrates the simulation of balancing two-wheeled robot model from different initial pitch angles (x_3) between the LQG and the LQR without input saturation. The $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices of LQG controller were selected similar to the LQR in Section 5.2.3 by using $\boldsymbol{Q}$ = diag{20,1,1,1,5} and $\boldsymbol{R} = 10\boldsymbol{I_{2\times2}}$; moreover, the white noise matrices $\boldsymbol{Q_k}$ and $\boldsymbol{R_k}$ of the Kalman filter method were applied when $\boldsymbol{Q_k} = \boldsymbol{I_{4\times4}}$ and $\boldsymbol{R_k} = 0.2\boldsymbol{I_{4\times4}}$**.** The stabilisation of the system without input constraints is presented in Figures 5.32-5.35. (Recall that the state variable x_3 was the only variable applied to state estimation in the LQG controller.)

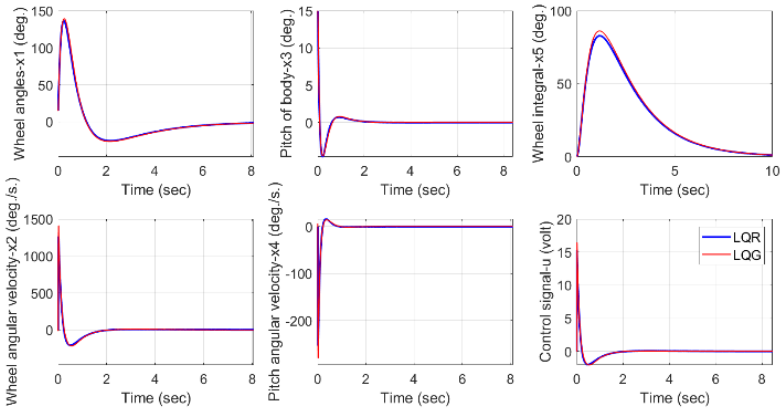


Figure 5.32: The stabilisation between LQR and LQG controller at the initial pitch angles (x_3) 15° without saturation.

To begin with, in Figure 5.32, the outcomes of LQR and LQG controllers are very similar to each other at the initial narrow pitch angle at 15°.

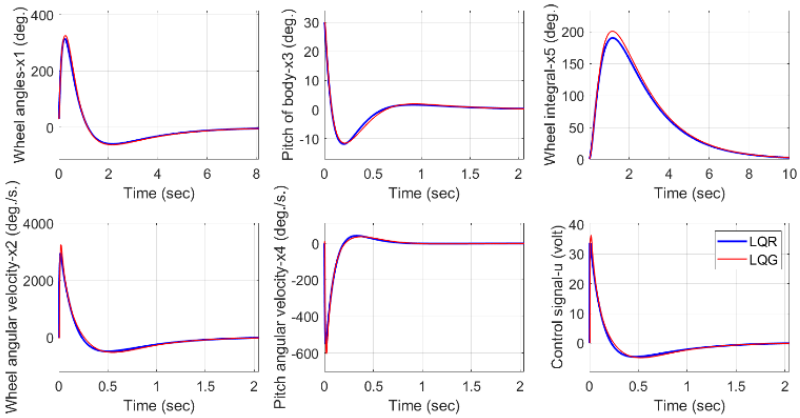


Figure 5.33: The stabilisation between LQR and LQG controller at the initial pitch angles (x_3) 30° without saturation.

Secondly, the two controllers present slightly different magnitudes, when the initial pitch angle is modified to a wider angle at 30° in Figure 5.33.

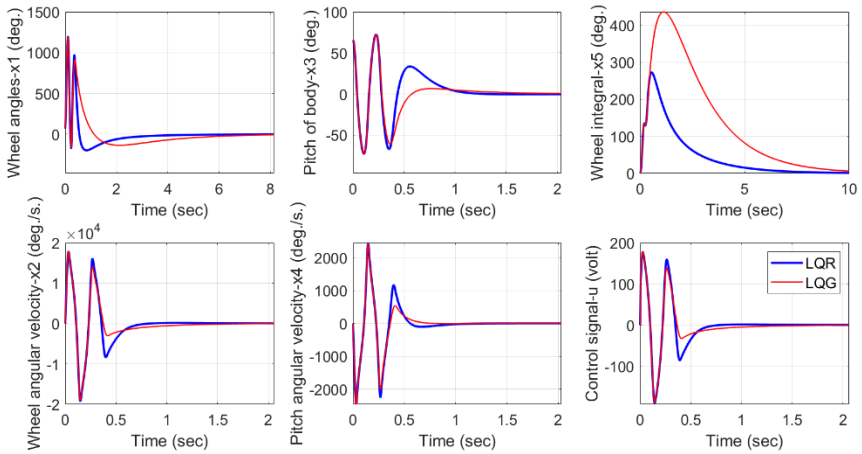


Figure 5.34: The stabilisation between LQR and LQG controller at the initial pitch angles (x_3) 65.7° without saturation.

Moreover, the maximum initial pitch angle (65.7°) of both controllers are similar in Figure 5.34. Significantly, state variables $x_1 - x_4$ and control signal of the LQG are less oscillatory than the LQR control as the estimation method is applied. Furthermore, the wheel integral's magnitude x_5 of LQG control is noticeably higher than the LQR, because the wheel angle x_1 of the LQG takes a longer time to reduce steady state error than the LQR. Above this initial angle, the LQG control results in an unstable system, shown in Figure 5.35, which is similar to the LQR method.

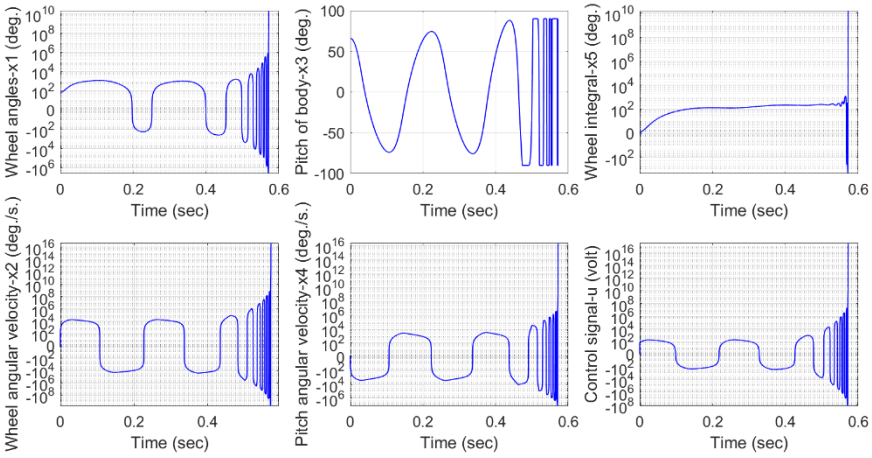


Figure 5.35: Unstable responses of the LQG control at the initial pitch angles (x_3) 65.8°

5.3.3.3 Simulations of TWR with Input Saturations

This section illustrates the simulation of the two-wheeled balancing robot model with different initial pitch angles (x_3) between the LQG and the LQR control with input saturation. The stabilisation of the system with input constraints are presented in Figures 5.36-5.37.

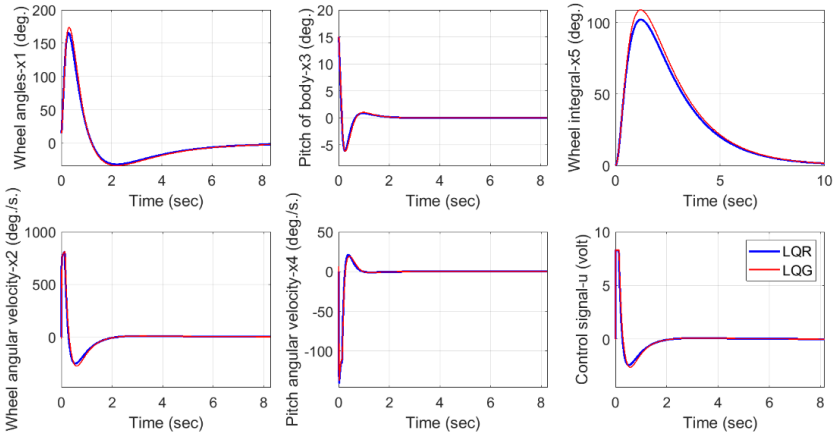


Figure 5.36: The stabilisation using LQR and LQG controllers at the initial pitch angles $x_3 = 15°$ with saturation.

It can be seen from Figure 5.36, the outcomes are largely similar between the two controllers at the narrow initial pitch angle of 15°, with control constraint.

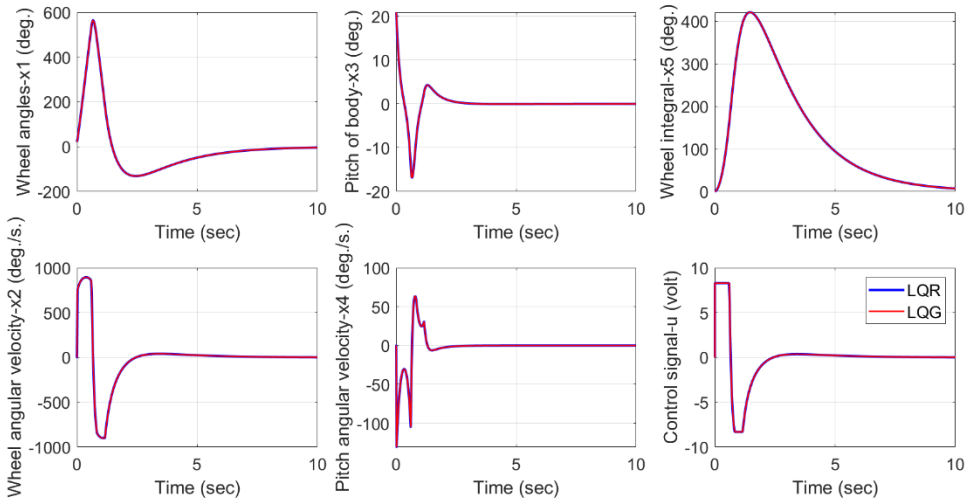


Figure 5.37: The stabilisation using LQR and LQG controllers at the initial pitch angles $x_3 = 20.9°$ with saturation.

Figure 5.37 demonstrates that the maximum initial pitch angle of both controllers are the same, at 20.9°, for the system to be stabilised with the control constraint condition. When the initial pitch angle went beyond 20.9°, the LQG control produced an unstable system which is similar to the LQR method shown in Figure 5.35.

5.3.4 Experimental Results

This section illustrates the experimental results of the LQG method demonstration in Section 5.3.1 using the LEGO Mindstorms EV3 robot with the Simulink block diagrams shown in Appendix B (Figure B2.1). The outcomes of the LQG control experiments are given in Figures 5.38-5.41.

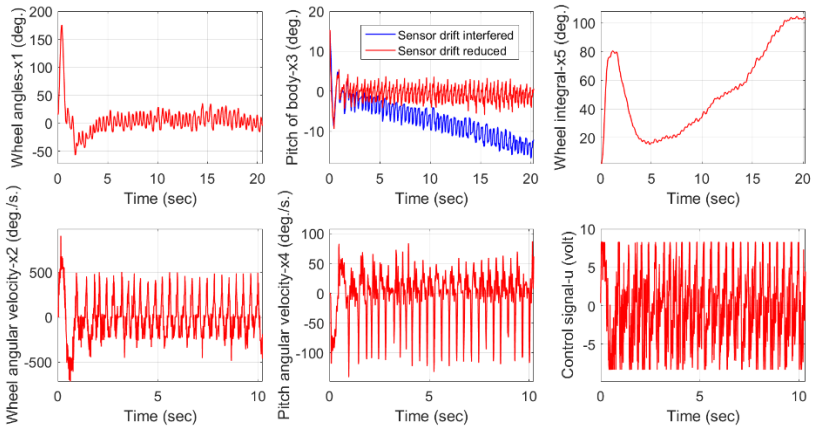


Figure 5.38: The outcomes of sensor drift reduced at initial pitch angles x_3=15° implemented by LEGO EV3 robot

Figure 5.38 demonstrates the main purpose of the LQG control in this research. It shows reduction of the sensor drift which originally appeared in the implemented system, at initial pitch angle 15°. The blue plot shows that the pitch angle generated by the gyro sensor is divergent to approximately -15° after the robot would be balanced in 20 seconds. The Kalman filter estimation successfully reduced the drifting error in the state variable x_3 (see the red curve) as it generated state estimation $\hat{x}_3$ for the LQR gain to use for stabilising the system instead of using the previous x_3 from the gyro sensor. However, there are slight

errors in the hardware implementation, shown in the state variable x_1; the average wheel angle is drifted from the centre by approximately 5°due to the effect of sensor drift. This error also interferes with the integral of wheel angle x_5 which caused a divergent signal to appear in the graph.

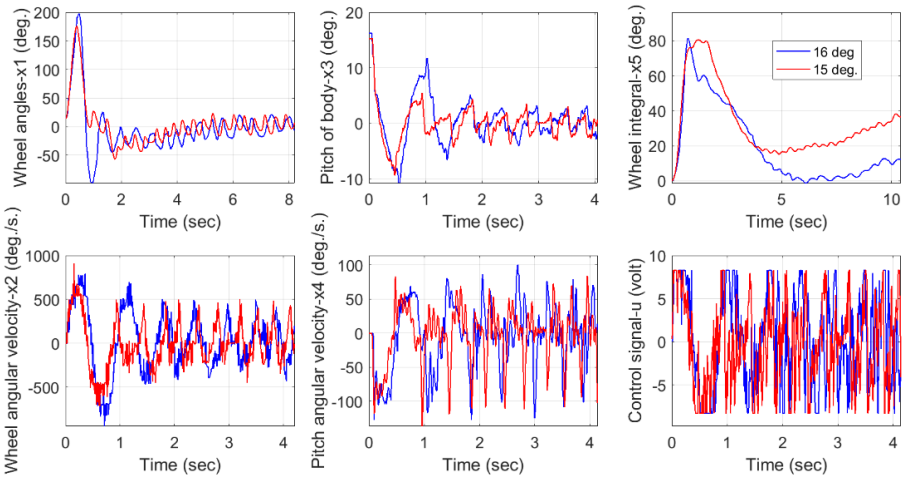


Figure 5.39: The stabilisation on different initial pitch angles (x_3) 15° and 16° implemented by LEGO EV3 robot

Moreover, the limitation of the LQG controller is also similar to the LQR method; for instance, both techniques provided a maximum initial pitch angle for stabilisation at 16°. Figure 5.39 presents the state evolutions when an increase of the initial angle from 15° to 16° is applied. As can be seen from the figure, the system responses starting from a pitch angle 16° display slightly higher magnitudes than 15° as the robot needs more time and a longer distance to reach the stabilised equilibrium position when the initial angle is increased. Over this limit angle, the two-wheeled robot will crash as the overall system becomes unstable, as shown in Figure 5.40. It can also be seen that the hardware reaches

the maximum capacity; for instance, the control signal shows signal at ± 8.3 V and the wheel angle velocity presents at approximately $-1{,}000^{\circ}/s$ when implemented at initial pitch angle 16.5°.

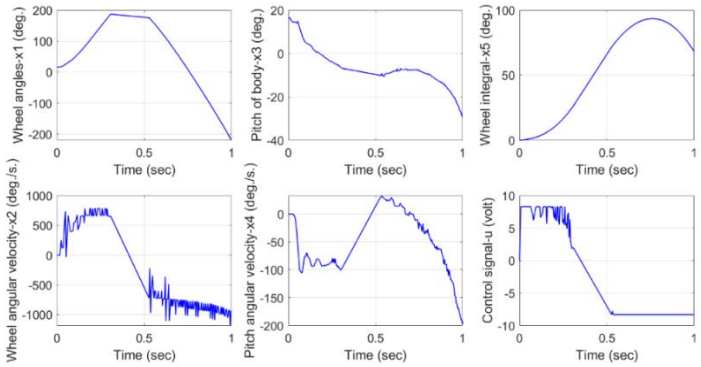


Figure 5.40: Unstable system starting from the initial pitch angle 16.5°

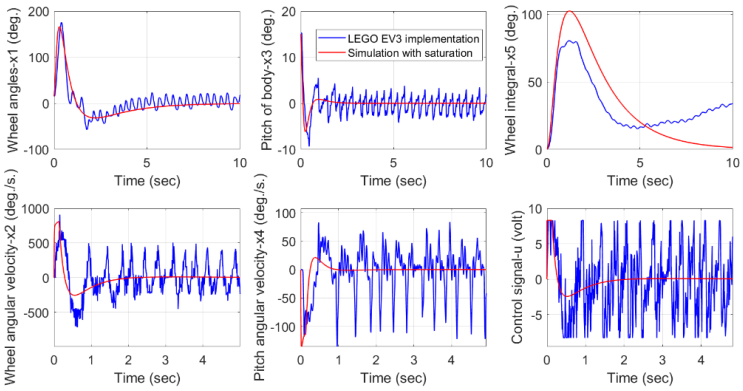


Figure 5.41: The stabilisation implemented on LEGO EV3 robot compared to simulation at the initial pitch angles $x_3 = 15°$

Furthermore, Figure 5.41 illustrates that the magnitudes of some state variables in Lego EV3 robot implementation are similar to the simulation results, for example, the overshoots of state variables x_1, x_2 and x_3, and the undershoots of state variables x_1, x_3 and x_4. Moreover, the average trends of both techniques are similar, although the state variable x_5 is different between the two methods,

as the robot's vibration in the wheel angle x_1 caused different integrals of wheel angle x_5. These results demonstrated the precisions of simulation and theoretical study, leading to the accurately matching results from hardware implementation.

5.4 Conclusion

In this chapter, a linear modern control technique, namely the optimal control or LQR was studied and applied to two systems: (1) the single inverted pendulum on a cart, and (2) the self-balancing two-wheeled robot system, which was also implemented on a Lego EV3 robot. In particular, the LQG technique was also utilised in the prototype robot to overcome sensor issues.

In the case of the single inverted pendulum and cart system, the LQR controller demonstrated a maximum stabilisation angle of the pendulum rod of 42.9°. This angle will be compared against the nonlinear controller in Chapter 6.

In terms of the two-wheeled robot simulation, the LQR and LQG techniques with unconstrained control input showed similar capability of balancing the system from the nearly identical, maximum initial angle, at approximately 65.7°. However, the LQG method displayed better stability than the LQR control, in terms of slightly smaller oscillation magnitudes and shorter settling time. Furthermore, the constrained input testing demonstrated that both controllers also provided the same initial angle limitation, at 20.9°. Therefore, the estimation provided by the Kalman filter is able to predict the state variables of the system as it operates at the same initial pitch angle, although there are some

small differences in the state evolutions of the two control schemes from initial pitch angle over 60° under the unconstraint condition. This is a property of state estimation as there is no perfect estimator (Anderson & Moore, 1989).

Regarding the Lego EV3 robot implementation, the experiments illustrated the capability of the two-wheeled robot maintaining the stabilisation with input saturation conditions. The designed controllers achieved the requirements of self-balancing and the maximum initial angle matched the results obtained from simulation. In particular, the linear quadratic Gaussian control provided elimination of the sensor drift problem which contributed to the system stability.

As both systems investigated in this thesis are nonlinear systems, it is envisaged that nonlinear control methods would show advantages in controlling the two systems to self-balance, over the linear techniques presented in this chapter. For instance, the nonlinear iteration technique and the nonlinear freezing control technique will be presented, applied and their associated results discussed in details in the next chapter.

Chapter 6

Nonlinear Control Designs and Implementations

6.1 Introduction

In Chapter 5, linear controls of the inverted pendulum (IP) model and the two-wheeled robot (TWR) model were demonstrated, including the linear quadratic regulator (LQR) and the linear quadratic Gaussian (LQG). This chapter, by contrast, presents two nonlinear control methods, namely, the nonlinear freezing technique and the nonlinear iteration technique, and their associated applications to the IP and TWR systems. The chapter is organised as follows. The nonlinear freezing control and iteration scheme theories are presented in Section 6.2 and 6.3, respectively. In Section 6.4, controllability and observability tests are conducted. Then, the simulated control results of the IP models and TWR models are demonstrated and analysed in Section 6.5. Furthermore, hardware experimentations of the nonlinear freezing control on a LEGO EV3 robot are examined in Sections 6.6. Finally, all outcomes are summarised in Section 6.7.

6.2 Nonlinear Freezing Control Strategy

In Chapter 5, the general linear systems are represented in the form of Eq. (5.1), as shown below

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u},$$

and the optimal control is defined by Eq. (5.2):

$$\dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{T}\boldsymbol{P})\boldsymbol{x},$$

which generates a fixed optimal control gain vector to control the system. In this chapter, an optimal control of the digitised system can be conducted in every time step by applying a nonlinear control method known as the freezing technique, first introduced in (Banks & Mhana, 1992). The benefits of this method is that local linearisation around the operating point is no longer necessary, as the system can be controlled globally using nonlinear system equations directly. Additional applications can be found in many researches, including the control of: a single inverted pendulum on a cart (Harrison, 2003), a F-8 crusader (Çimen & Banks, 2004a) and a double inverted pendulum on a cart (Xu, Zhang, & Carbone, 2017). The freezing control technique introduced by Banks & Mhana (1992) is in the form of

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}, \tag{6.1}$$

where $\boldsymbol{A}(\boldsymbol{x})$ and $\boldsymbol{B}(\boldsymbol{x})$represent the nonlinear system matrices which form controllability matrices and $\boldsymbol{u}$ is the nonlinear optimal control.

The cost function of the quadratic infinite-time is defined by:

$$J = \int_0^\infty (\boldsymbol{x}^T \boldsymbol{Q}(\boldsymbol{x})\boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R}(\boldsymbol{x})\boldsymbol{u})dt, \tag{6.2}$$

where $\boldsymbol{Q}(\boldsymbol{x})$ and $\boldsymbol{R}(\boldsymbol{x})$ denote the positive semi-definite and positive definite, respectively.

Banks and Mhana (1992) presented that the nonlinear optimal feedback control from Eqs. (6.1)-(6.2) is given by

$$\boldsymbol{u} = -\boldsymbol{R}^{-1}(\boldsymbol{x})\boldsymbol{B}^T(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x})\boldsymbol{x}. \tag{6.3}$$

Thus, feedback gain matrix $\boldsymbol{K}(\boldsymbol{x})$ of the system is given by

$$\boldsymbol{K}(\boldsymbol{x}) = \boldsymbol{R}^{-1}(\boldsymbol{x})\boldsymbol{B}^{T}(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x}), \tag{6.4}$$

where matrix $\boldsymbol{P}(\boldsymbol{x})$ is the solution of the algebraic matrix Riccati equation:

$$0 = \boldsymbol{P}(\boldsymbol{x})\boldsymbol{A}(\boldsymbol{x}) + \boldsymbol{A}^{T}(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x}) - \boldsymbol{P}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})\boldsymbol{R}^{-1}(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x}) + \boldsymbol{Q}(\boldsymbol{x}). \tag{6.5}$$

Equation (6.5) can be solved numerically by applying the linear quadratic regulator function in MATLAB as shown in Appendix A.5.1.

Therefore, the optimal control is implemented by substituting Eq. (6.3) into Eq. (6.1) and obtaining:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} - \boldsymbol{B}(\boldsymbol{x})(\boldsymbol{R}^{-1}(\boldsymbol{x})\boldsymbol{B}^{T}(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x})\boldsymbol{x}) \\ &= (\,\boldsymbol{A}(\boldsymbol{x}) - \boldsymbol{B}(\boldsymbol{x})\boldsymbol{R}^{-1}(\boldsymbol{x})\boldsymbol{B}^{T}(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{x})\,)\boldsymbol{x}. \end{aligned} \tag{6.6}$$

The equations above demonstrate that computationally the solution $\boldsymbol{P}(\boldsymbol{x})$ of the algebraic matrix Riccati equation can be obtained at every time step at each point on the state trajectory, which is a fixed $\boldsymbol{x}$, generating frozen matrices $\boldsymbol{A}(\boldsymbol{x})$ and $\boldsymbol{B}(\boldsymbol{x})$. Then the nonlinear dynamical system becomes a pseudo-linear system to provide the requirement of feedback gains in every step of dynamic equation for stabilising the system globally. The block diagram of the freezing control technique for a system combined with a tracker is presented in Figure 6.1.

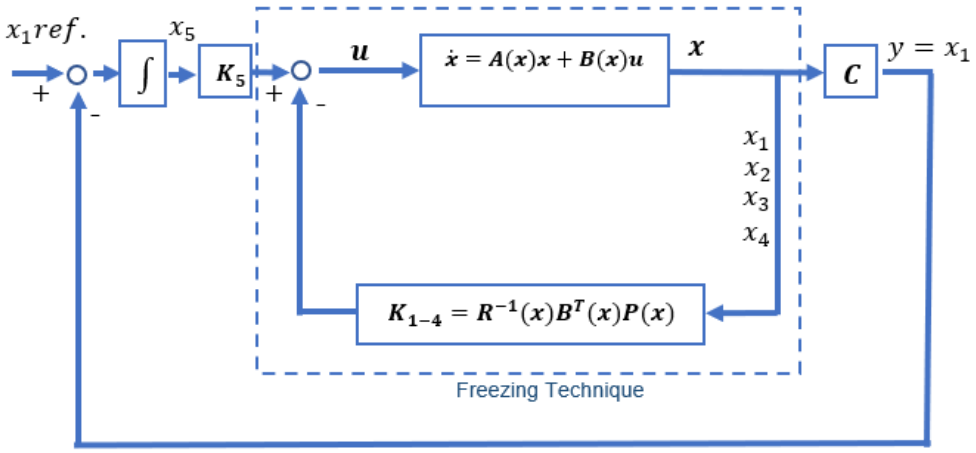


Figure 6.1: Structure of freezing control technique and tracking system.

6.2.1 Freezing Technique - Input Saturation

In the linear control strategy in Section 5.2.3.3, the hard input constraint was presented to restrict the input signal of the system model. In contrast, the nonlinear model offers flexibility over the linear system, because linearisation is no longer necessary and the dynamic equations can be modified to combine a soft constraint at the input signal. Harrison (2003) utilised this idea and presented the integration of a state constraint, representing the input saturation for the case of a scalar control v by redefining as follows:

$$v = \phi(x_{n+1}), \tag{6.7}$$

and introducing an extra state variable x_{n+1}, where

$$\dot{x}_{n+1} = w, \tag{6.8}$$

and w is the new control signal. Then substitute Eqs. (6.7) and (6.8) into Eq.(6.1),

we obtain

$$\dot{\boldsymbol{X}} = \boldsymbol{A}(\boldsymbol{X})\boldsymbol{X} + \boldsymbol{B}(\boldsymbol{X})\phi(x_{n+1}), \tag{6.9}$$

which can be rewritten by adding the term of x_{n+1} and w as

$$\dot{\boldsymbol{X}} = \boldsymbol{A}(\boldsymbol{X})\boldsymbol{X} + \boldsymbol{B}(\boldsymbol{X})\phi(x_{n+1}) \times \left(\frac{x_{n+1}}{x_{n+1}}\right) + 0 \times w. \tag{6.10}$$

Combining Eqs. (6.8) and (6.10), we have

$$\begin{bmatrix} \dot{\boldsymbol{X}} \\ \dot{x}_{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}(\boldsymbol{X}) & \frac{\boldsymbol{B}(\boldsymbol{X})\emptyset(x_{n+1})}{x_{n+1}} \\ \boldsymbol{0_{1\times n}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{X} \\ x_{n+1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0_{n\times 1}} \\ 1 \end{bmatrix} w. \tag{6.11}$$

Note that Harrison (2003) presented only a scalar control signal. However, this research will use multiple controls forming a control vector $\boldsymbol{u} = \begin{bmatrix} v_L \\ v_R \end{bmatrix}$; thus, there are some factors which are different from the ones presented in (Harrison, 2003). For instance, the scalar control signal $\phi(x_{n+1})$ is transformed to a column vector, including control signals of the left and the right motors as follows:

$$\boldsymbol{\phi}(\boldsymbol{x_{n+1}}) = \begin{bmatrix} \phi_L(x_{n+1}) \\ \phi_R(x_{n+1}) \end{bmatrix} \tag{6.12}$$

In the state-space matrix form, the extra state variable x_{n+1} can be rewritten as x_6. Therefore, the state-space representation of the two-wheeled balancing robot with soft constraint, after substituting the nonlinear system Eq. (4.61) into Eq. (6.11) can be shown as follows:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \\ \dot{x_6} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{e_{m22}}{a+b} & \frac{e_{m23}}{(a+b)x_3} & \frac{e_{m24}}{a+b} & 0 & \frac{f_{m21}\phi_L(x_6)+f_{m22}\phi_R(x_6)}{(a+b)(x_6)} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{e_{m42}}{a+b} & \frac{e_{m43}}{(a+b)x_3} & \frac{e_{m44}}{a+b} & 0 & \frac{f_{m41}\phi_L(x_6)+f_{m42}\phi_R(x_6)}{(a+b)(x_6)} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} (w). \tag{6.13}$$

Rewrite Eq. (6.11) in the form of a new nonlinear system as follows:

$$\dot{\boldsymbol{x}}_{\boldsymbol{a}} = \boldsymbol{A}_{\boldsymbol{a}}(\boldsymbol{x}_{\boldsymbol{a}})\boldsymbol{x}_{\boldsymbol{a}} + \boldsymbol{B}_{\boldsymbol{a}}(\boldsymbol{x}_{\boldsymbol{a}})w\ . \tag{6.14}$$

The cost function now becomes

$$J = \int_0^\infty (\boldsymbol{x}_{\boldsymbol{a}}{}^{\boldsymbol{T}}\boldsymbol{Q}_{\boldsymbol{a}}(\boldsymbol{x}_{\boldsymbol{a}})\boldsymbol{x}_{\boldsymbol{a}} + w^T R_a(\boldsymbol{x}_{\boldsymbol{a}})w)dt, \tag{6.15}$$

where the subscript $\boldsymbol{a}$ indicates the constrained system and the weighting matrix $\boldsymbol{Q_a}$, which is given by

$$\boldsymbol{Q}_{\boldsymbol{a}} = \begin{bmatrix} \boldsymbol{Q} & 0 \\ 0 & \phi^2(x_{n+1})\boldsymbol{R} \end{bmatrix}, \tag{6.16}$$

and R_a is set as a small value ($R_a = 0.001$) as it provides the results of input constraint close to the unconstrainted condition (Harrison, 2003).

In Chapter 5, the weight matrix $\boldsymbol{R}$ is set as $\boldsymbol{R} = \begin{bmatrix} 10 & 1 \\ 1 & 10 \end{bmatrix}$, in which R_{11}=10 and R_{22}=10. These values can be applied to control the left and right motors $\boldsymbol{u} = \begin{bmatrix} v_L \\ v_R \end{bmatrix}$. However, the value of $\phi^2(x_{n+1})\boldsymbol{R}$ in Eq.(6.16) is required as a scalar.

Harrison (2003) presented the value of $\phi^2(x_{n+1})\boldsymbol{R}$ for a single control signal. In terms of two control inputs, the $\phi^2(x_{n+1})\boldsymbol{R}$ is considered to compensate for two control signals given by

$$2 \times \phi_L^2(x_{n+1})(R_{11}) \text{ or } 2 \times \phi_R^2(x_{n+1})(R_{22}),$$

where $\phi_L(x)$ is equal to $\phi_R(x)$ as this research focuses on the stabilisation of the pitch angle and the yaw angle motion that needs different control signals between the two motors is not considered.

Previously, matrix $\boldsymbol{Q}$ was set as $\boldsymbol{Q}$ =diag{20,1,1,1,5}; thus, the weighting matrix $\boldsymbol{Q_a}$ of the constrained system is presented by substituting matrix $\boldsymbol{Q}$ and $2\phi_{L,R}^2(x_{n+1})R_{11,22}$ into Eq. (6.16) and we obtain

$$\boldsymbol{Q_a} = \begin{bmatrix} 20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 20\phi_{L,R}^2(x_6) \end{bmatrix}. \tag{6.17}$$

Moreover, the saturation conditions (Harrison, 2003) with a smooth function $sin(x)$ were presented as the following:

$$\phi_L(x_6) = \phi_R(x_6) = \begin{cases} \lambda\ , & x_6 > \lambda \\ \lambda sin\left(\frac{\pi x_6}{2\lambda}\right), & |x_6| \geq \lambda \\ -\lambda\ , & x_6 < -\lambda \end{cases} \tag{6.18}$$

where λ is the limitation of control signal as the maximum voltage of the LEGO EV3 motor is 8.3V, in the implementation.

6.2.2 Freezing Technique and Extended Kalman Filter

Recall in Chapter 5 that sensor drift occurred in the hardware application. The nonlinear freezing control technique on its own cannot solve this issue as it extends the LQR theory to stabilise the system and the results of the LQR method showed a diverged error signal in Chapter 5.

In this section, therefore, an extended version of the Kalman filter utilised in Chapter 5 will be combined with the freezing technique to overcome the sensor drift issue. The linear Kalman filter will be extended for estimating the state variable of the nonlinear system known as an extended Kalman filter (EKF) (Simon, 2006). There are many applications of this technique, such as missile guidance (Çimen & Merttopçuoğlu, 2008), Unmanned Aerial Vehicle (UAV) (Nemra & Aouf, 2010) and cancer treatment (Batmani & Khaloozadeh, 2013).

General continuous time-invariant systems (Frank , Xie, & Popa, 2007) are given by

$$\dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{t}) + \boldsymbol{G}(\boldsymbol{t})\boldsymbol{w_n}, \tag{6.19}$$

$$\boldsymbol{y} = \boldsymbol{c}(\boldsymbol{x}, \boldsymbol{t}) + \boldsymbol{v_n}, \tag{6.20}$$

where $\boldsymbol{w_n}$ and $\boldsymbol{v_n}$ are supposed to be process noise and measurement noise, respectively, with $\boldsymbol{w_n} \sim (0,\ \boldsymbol{Q_k})$, $\boldsymbol{v_n} \sim (0,\ \boldsymbol{R_K})$, and $\boldsymbol{G}$ is matrix of process noise, which is defined as $\boldsymbol{G} = \boldsymbol{I_{5\times5}}$.

A state estimator of a nonlinear dynamic system is defined as follows:

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{a}(\hat{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{t}) + \boldsymbol{K_f}(\boldsymbol{y} - \boldsymbol{c}(\hat{\boldsymbol{x}}, \boldsymbol{t})), \tag{6.21}$$

and the Jacobian matrices are

$$\boldsymbol{A}(\boldsymbol{x}, \boldsymbol{t}) = \frac{\boldsymbol{\partial a}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{t})}{\boldsymbol{\partial x}}, \boldsymbol{A}(\hat{\boldsymbol{x}}, \boldsymbol{t}) = \frac{\boldsymbol{\partial a}(\hat{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{t})}{\boldsymbol{\partial \hat{x}}},$$
$$\boldsymbol{C}(\boldsymbol{x}, \boldsymbol{t}) = \frac{\boldsymbol{\partial c}(\boldsymbol{x}, \boldsymbol{t})}{\boldsymbol{\partial x}}, \text{and } \boldsymbol{C}(\hat{\boldsymbol{x}}, \boldsymbol{t}) = \frac{\boldsymbol{\partial c}(\hat{\boldsymbol{x}}, \boldsymbol{t})}{\boldsymbol{\partial \hat{x}}}. \tag{6.22}$$

The Kalman filter gain $\boldsymbol{K_f}$ is given by

$$\boldsymbol{K_f} = \boldsymbol{P}\boldsymbol{C}^{\boldsymbol{T}}(\hat{\boldsymbol{x}}, \boldsymbol{t})\boldsymbol{R_k^{-1}} \tag{6.23}$$

where $\boldsymbol{P}$ is the solution of algebraic Riccati equation as shown below

$$\boldsymbol{A}(\hat{\boldsymbol{x}}, \boldsymbol{t})\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A}^{\boldsymbol{T}}(\hat{\boldsymbol{x}}, \boldsymbol{t}) - \boldsymbol{P}\boldsymbol{C}^{\boldsymbol{T}}(\hat{\boldsymbol{x}}, \boldsymbol{t})\boldsymbol{R_k^{-1}}\boldsymbol{C}(\hat{\boldsymbol{x}}, \boldsymbol{t})\boldsymbol{P} + \boldsymbol{Q_k} = 0 \tag{6.24}$$

Approximated solutions of Eq. (6.24) can be obtained numerically by applying the linear quadratic regulator function in MATLAB, shown in Appendix A.5.5. The block diagram of the freezing control technique with an extended Kalman filter combined with a tracker is presented in Figure 6.2.

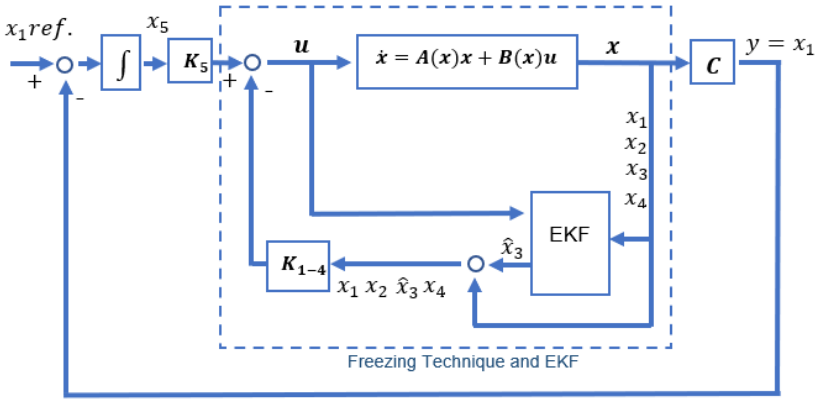


Figure 6.2: Structure of freezing technique with EKF and tracking system.

6.3. Iteration Scheme Strategy

Another nonlinear control method considered in this research is an iteration scheme introduced by Banks & McCaffrey (1998). The technique is used to approximate the original nonlinear system into a sequence of linear, time-varying (LTV) equations. This method can be applied to control various nonlinear systems, e.g., an inverted pendulum on a cart (Banks & Dinesh, 2000), super-tankers autopilot (ÇImen & Banks, 2004b), drug therapy control in cancer treatment (Itik, Salamci, & Banks, 2009), velocity tracking of hydraulic press (Du, Xu, Banks, & Wu, 2009), and tunnel diode oscillator control (Itik, 2016).

The approximating sequence of the iteration scheme is introduced next (Tomás-Rodríguez & Banks, 2010).

Consider the pseudo-linear system Eq.(6.25) and the finite-time quadratic cost function Eq.(6.26):

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}, \tag{6.25}$$

$$J = \frac{1}{2}\boldsymbol{x}^{\boldsymbol{T}}(t_f)\boldsymbol{F}\boldsymbol{x}(t_f) + \frac{1}{2}\int_0^{t_f}(\boldsymbol{x}^{\boldsymbol{T}}(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^{\boldsymbol{T}}(t)\boldsymbol{R}\boldsymbol{u}(t))dt, \tag{6.26}$$

where $\boldsymbol{A}(\boldsymbol{x})$ and $\boldsymbol{B}(\boldsymbol{x})$ form controllability matrices, t_f represents the final time, $\boldsymbol{F}$ and $\boldsymbol{Q}$ are positive semi-definite matrices and $\boldsymbol{R}$ is a positive definite matrix.

Then, a sequence of linear, time-varying equation can be written as follows:

$$\dot{\boldsymbol{x}}^{[i]}(t) = \boldsymbol{A}(\boldsymbol{x}^{[i-1]}(t))\boldsymbol{x}^{[i]}(t) + \boldsymbol{B}(\boldsymbol{x}^{[i-1]}(t))\boldsymbol{u}^{[i]}(t), \quad \boldsymbol{x}^{[i]}(t_0) = \boldsymbol{x}_0, \tag{6.27}$$

and

$$\boldsymbol{J}^{[i]} = \frac{1}{2}\boldsymbol{x}^{[i]^T}(t_f)\boldsymbol{F}\boldsymbol{x}^{[i]}(t_f) + \frac{1}{2}\int_0^{t_f}\left(\boldsymbol{x}^{[i]^T}(t)\boldsymbol{Q}\boldsymbol{x}^{[i]}(t) + \boldsymbol{u}^{[i]^T}\boldsymbol{R}(t)\boldsymbol{u}^{[i]}(t)\right)dt, \tag{6.28}$$

where $[i]$ denotes the iteration number.

Hence, the first approximation can be written as the following:

$$\dot{\boldsymbol{x}}^{[1]}(t) = \boldsymbol{A}(\boldsymbol{x}_0)\boldsymbol{x}^{[1]}(t) + \boldsymbol{B}(\boldsymbol{x}_0)\boldsymbol{u}^{[1]}(t), \qquad \boldsymbol{x}^{[1]}(t_0) = \boldsymbol{x}_0, \tag{6.29}$$

Note that $\boldsymbol{x}^{[i-1]}(t)$ is assumed as $\boldsymbol{x}_0$ at initial sequence $i = 1$.

Then, evaluating Eqs. (6.27) and (6.28) produce the following optimal control equation:

$$\boldsymbol{u}^{[i]} = -\boldsymbol{R}^{-1}\boldsymbol{B}^T(\boldsymbol{x}^{[i-1]}(t))\boldsymbol{P}^{[i]}(t)\boldsymbol{x}^{[i]}(t), \tag{6.30}$$

where $\boldsymbol{P}^{[i]}(t)$ is the solution of the Riccati equation below

$$\begin{aligned}\dot{\boldsymbol{P}}^{[i]}(t) = -\boldsymbol{Q} - \boldsymbol{P}^{[i]}(t)\boldsymbol{A}\left(\boldsymbol{x}^{[i-1]}(t)\right) - \boldsymbol{A}^T\left(\boldsymbol{x}^{[i-1]}(t)\right)\boldsymbol{P}^{[i]}(t) \\ + \boldsymbol{P}^{[i]}(t)\boldsymbol{B}\left(\boldsymbol{x}^{[i-1]}(t)\right)\boldsymbol{R}^{-1}\boldsymbol{B}^T\left(\boldsymbol{x}^{[i-1]}(t)\right)\boldsymbol{P}^{[i]}(t),\end{aligned} \tag{6.31}$$

and

$$\boldsymbol{P}^{[i]}(t_f) = \boldsymbol{F}. \tag{6.32}$$

Hence, the optimal control system is implemented by substituting Eq. (6.30) into Eq. (6.27) as the following:

$$\begin{aligned}\dot{\boldsymbol{x}}^{[i]}(t) = \boldsymbol{A}\left(\boldsymbol{x}^{[i-1]}(t)\right)\boldsymbol{x}^{[i]}(t) \\ + \boldsymbol{B}(\boldsymbol{x}^{[i-1]}(t))(-\boldsymbol{R}^{-1}\boldsymbol{B}^T\left(\boldsymbol{x}^{[i-1]}(t)\right)\boldsymbol{P}^{[i]}(t)\boldsymbol{x}^{[i]}(t)), \quad \boldsymbol{x}^{[i]}(t_0) = \boldsymbol{x}_0.\end{aligned} \tag{6.33}$$

6.4 Controllability and Observability

In this section, the controllability and observability of the two-wheel robot system will be analysed before the design of controllers and the corresponding simulations take place. In particular, recall that in Section 6.2.1, matrix $\boldsymbol{A}$ was modified to embed the input saturation condition into the nonlinear model. Therefore, the analysis will include both unconstrained and constrained TWR systems for consideration.

6.4.1 Controllability

In terms of controllability, the test and the system analysis are similar to the ones presented in Chapter 5, by substituting matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ into Eq. (5.7) to produce controllability matrix $\boldsymbol{\mathcal{C}}$, and then calculating the rank of the controllability test matrix. As mentioned previously, mathematical models of linear control system were linearised around equilibria. It is well-known that state-space representations of nonlinear control systems are not unique. In addition, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices of a nonlinear system vary for different values of time t and so does the controllability matrix; therefore, the ranks of controllability matrices (for two systems) can be presented as a 2D plot, shown in Figures 6.3-6.4.

- **Inverted Pendulum on a Cart System**

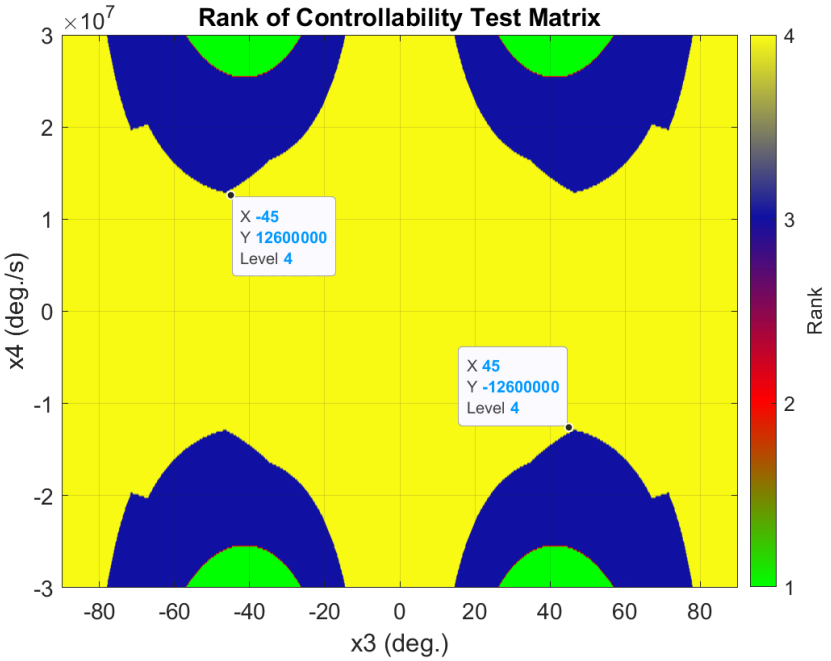


Figure 6.3: The rank of controllability matrix for the nonlinear inverted pendulum and cart system

Figure 6.3 presents the rank test result of the 4th order nonlinear inverted pendulum model from Eq. (4.10). The contour plot specifies the state variables x_3 and x_4 in the x and y axes, respectively. The yellow region represents Rank($\mathcal{C}$) = 4, which means the system is fully controllable, appearing in the central region of the plot, Other non-yellow regions represent Rank ($\mathcal{C}$) < 4, which indicate that the system is uncontrollable (or not fully controllable).

- **Two-Wheeled Robot System**

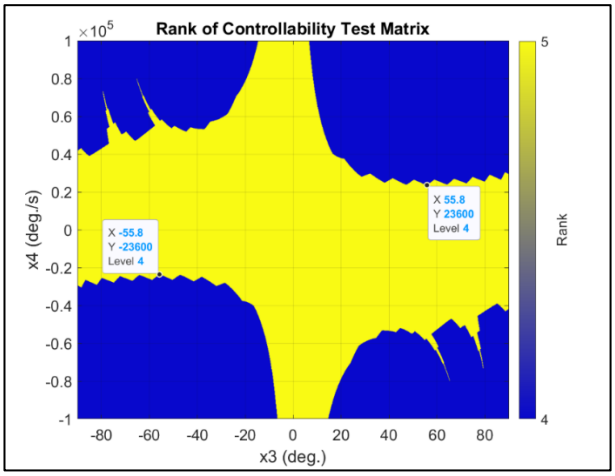


Figure 6.4: The rank of controllability matrix for the nonlinear two-wheeled robot system - without input saturation

Similarly, the rank test of the 5th order nonlinear two-wheeled robot model from Eq. (4.61) is demonstrated in Figure 6.4. The yellow and blue areas represent Rank($\mathcal{C}$) = 5 and Rank($\mathcal{C}$) = 4, respectively. The system is fully controllable when Rank($\mathcal{C}$) = 5, as shown in the central region of the figure; by contrast, the system is only partially controllable in all corner regions as Rank ($\mathcal{C}$) = 4. There are some fully controllable regions on left and right sides of the graph, near $x_3 \approx \pm 90^{\circ}$, when the pitch angular velocity x_4 operates between $-2.3 \times 10^4\ ^{\circ}/s$ and $2.3 \times 10^4\ ^{\circ}/s$. Note here that physically realistic ranges of x_4 for the LEGO EV3 gyro sensor are between $-440^{\circ}/s$ and $440^{\circ}/s$.

Note, the nonlinear two-wheeled robot control is applied to a practical robot rather than a simulated mathematical model in this research and therefore offers flexibility in which model to use for the generation of control gains. The model in Eq. (4.61) can be rewritten in several different forms and then the controllability test can be applied to investigate each model's controllability range. Next, two

other state-space model representations of the TWR system are defined, with names Model B and Model C, whilst the original model given in Eq. (4.61) is named as Model A.

Model B:

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\end{pmatrix}=\begin{pmatrix}0 & 1 & 0 & 0 & 0\\ 0 & \frac{e_{m22}(x_3)}{a+b(x_3)} & \frac{e_{23}(x_3)+e_{m24}(x_3,x_4)x_4}{[a+b(x_3)]x_3} & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & \frac{e_{m42}(x_3)}{a+b(x_3)} & \frac{e_{43}(x_3)+e_{m44}(x_3,x_4)x_4}{[a+b(x_3)]x_3} & 0 & 0\\ 1 & 0 & 0 & 0 & 0\end{pmatrix}\times\begin{pmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{pmatrix} \quad (6.34)$$

$$+\begin{pmatrix}0 & 0\\ \frac{f_{m21}(x_3)}{a+b(x_3)} & \frac{f_{m22}(x_3)}{a+b(x_3)}\\ 0 & 0\\ \frac{f_{m41}(x_3)}{a+b(x_3)} & \frac{f_{m42}(x_3)}{a+b(x_3)}\\ 0 & 0\end{pmatrix}\begin{pmatrix}v_1\\ v_2\end{pmatrix},$$

where the Model B presents the 3rd column by embedding equations in the 3rd and 4th columns of Model A.

Model C:

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\end{pmatrix}=\begin{pmatrix}0 & 1 & 0 & 0 & 0\\ 0 & \frac{e_{m22}(x_3)}{a+b(x_3)} & 0 & \frac{e_{23}(x_3)+e_{m24}(x_3,x_4)x_4}{a+b(x_3)x_4} & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & \frac{e_{m42}(x_3)}{a+b(x_3)} & 0 & \frac{e_{43}(x_3)+e_{m44}(x_3,x_4)x_4}{a+b(x_3)x_4} & 0\\ 1 & 0 & 0 & 0 & 0\end{pmatrix}\times\begin{pmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{pmatrix} \quad (6.35)$$

$$+\begin{pmatrix}0 & 0\\ \frac{f_{m21}(x_3)}{a+b(x_3)} & \frac{f_{m22}(x_3)}{a+b(x_3)}\\ 0 & 0\\ \frac{f_{m41}(x_3)}{a+b(x_3)} & \frac{f_{m42}(x_3)}{a+b(x_3)}\\ 0 & 0\end{pmatrix}\begin{pmatrix}v_1\\ v_2\end{pmatrix},$$

where the Model C presents the 4th column by embedding equations in the 3rd and 4th columns of Model A.

Likewise, the graphs of controllability test matrices from Model B and C are demonstrated in Figures 6.5 – 6.6.

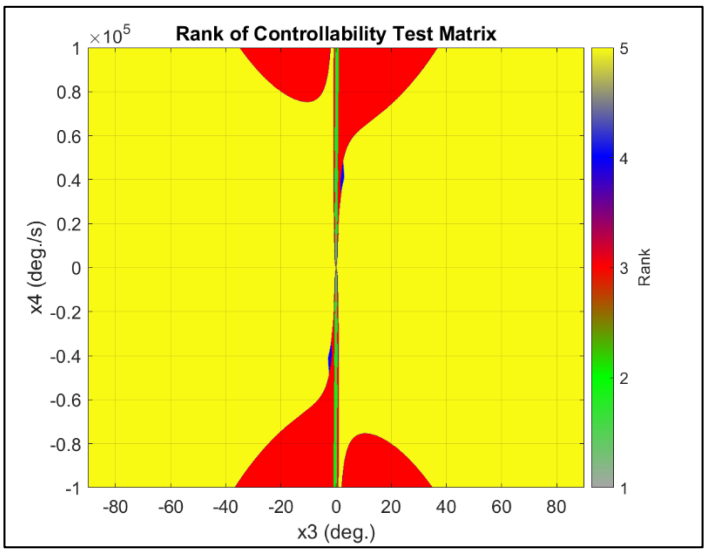


Figure 6.5 : The rank of controllability matrix for the nonlinear TWR system - without input saturation (Model B)

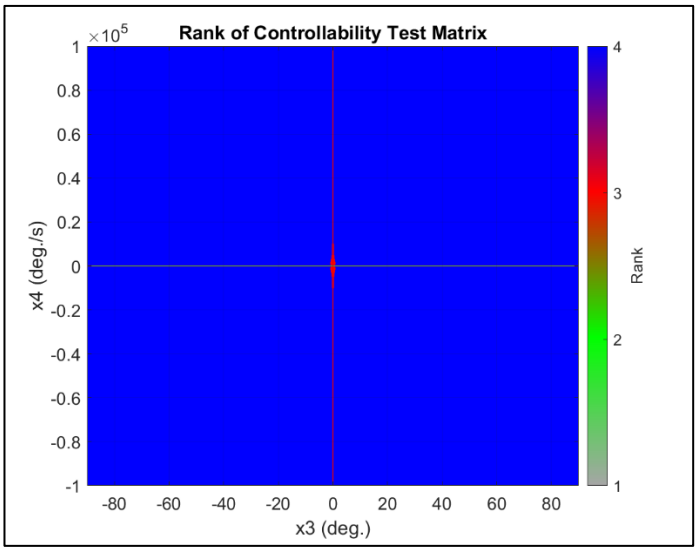


Figure 6.6 : The rank of controllability matrix for the nonlinear TWR system - without input saturation (Model C)

It can be seen that Models A, B and C produce very different controllability test results. In Figure 6.5, the area which is not fully controllable appears in the middle

of the x_3 axis for a large range of x_4. This means Model B is not fully controllable at the equilibrium point. Furthermore, Figure 6.6 demonstrates that Model C is only partially controllable for the whole region investigated (i.e., $x_3 \in [-90°, 90°]$ and $x_4 \in [-1 \times 10^5\,°/s\,, 1 \times 10^5\,°/s]$), with Rank ($\mathcal{C}$) < 5.

One advantage of Model B from its controllability test result presented in Figure 6.5 is that the fully controllable region is larger than Model A's when the state variable x_3 is far away from $x_3 = 0°$. Therefore, Models A and B can be combined to create a mixed model to take advantage of each individual model's controllability range. For example, Model A's central region, i.e., when $-10° \leq x_3 \leq 10°$ can be selected to be combined with Model B in other regions to create a new model, named Model AB, which generates better controllability outcome, as shown in Figure 6.7.

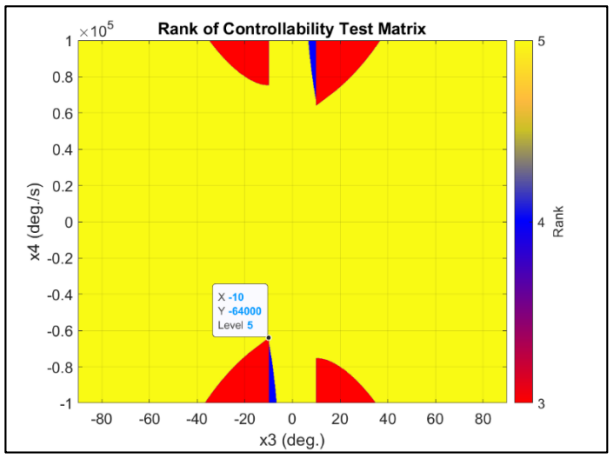


Figure 6.7 : The rank of controllability matrix for the nonlinear TWR system - without input saturation (Model AB)

It can be seen from Figure 6.7 that the not fully controllable area in the centre of the x_3 axis of Model B disappears, and the fully controllable region of Model A is displayed instead.

Additionally, the rank tests of nonlinear control system under input constrain condition of Model A defined by Eq. (4.61) are presented in Figures 6.8-6.9.

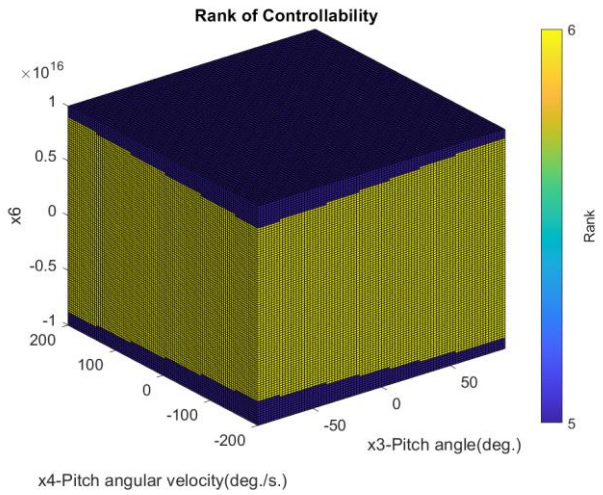
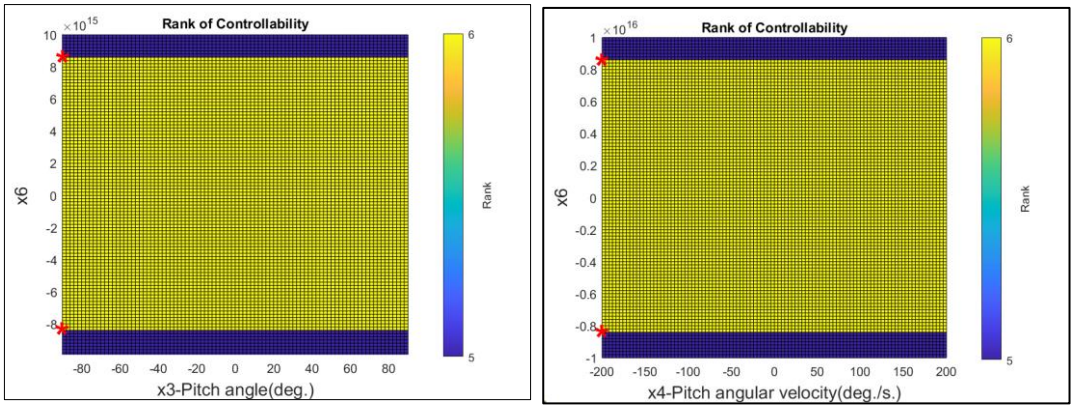


Figure 6.8: The rank of controllability for freezing technique system – with input saturation

Figure 6.8 demonstrates a controllability cube in the six-dimensional space, with three variables x_3, x_4 and x_6 on the axes, because a new variable x_6 has been introduced into the constrained system, shown in Eq. (6.13). The yellow and blue colours represent Rank($\mathcal{C}$) = 6 (full rank) and Rank($\mathcal{C}$) = 5, respectively.

Further details of Figure 6.8 can be seen by showing cross-sections of the cube next. Figure 6.9(a) shows the rank of the controllability matrix affected by state variable x_3 and the constraint parameter x_6 when x_4 is fixed at zero; by contrast, Figure 6.9(b) presents the rank of controllability, influenced by x_4 and the constraint parameter x_6 when x_3 is equal to zero.



(a) *(b)*

Figure 6.9: Controllability plot for the TWR system (with input saturation), (a) cross-section at $x_4 = 0°/s$. Note, the coordinates (x_3, x_6) of the 2 points marked by red asterisks are $(-90°,\ -8.4 \times 10^{15})$ and $(-90°,\ 8.6 \times 10^{15})$, (b) cross-section at $x_3 = 0°$. Note, the coordinates (x_4, x_6) of the 2 points marked by red asterisks are $(-200°/s,\ -8.4 \times 10^{15})$ and $(-200\ °/s,\ 8.6 \times 10^{15})$,

With the input constraint condition included, the system model is of 6th order, as shown in Eq.(6.13). The system model would be completely state controllable if Rank($\mathcal{C}$) = 6. Figure 6.9, therefore, demonstrates a fully controllable system for any values of state variables x_3 and x_4 if the constraint parameter x_6 is restricted approximately between -8.4×10^{15} and 8.6×10^{15} .

Furthermore, the rank test results of Models B and AB with the input voltage constrained are demonstrated in Figures 6.10-6.13. Note, controllability test of Model C will not be implemented as it showed no region to be fully controllable in the non-saturated input simulation previously.

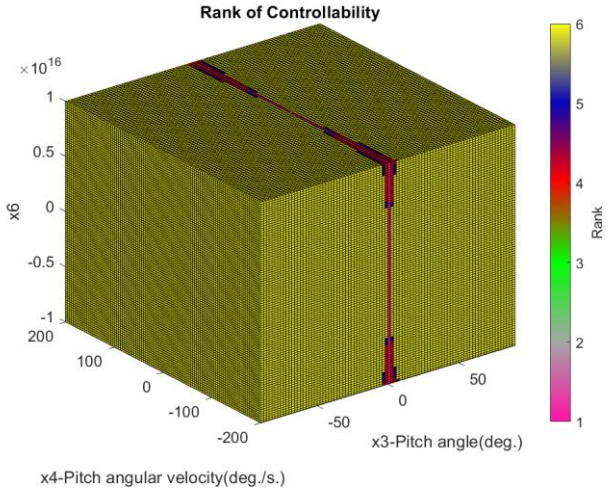


Figure 6.10: The rank of controllability matrix of TWR for the freezing technique – with input saturation (Model B)

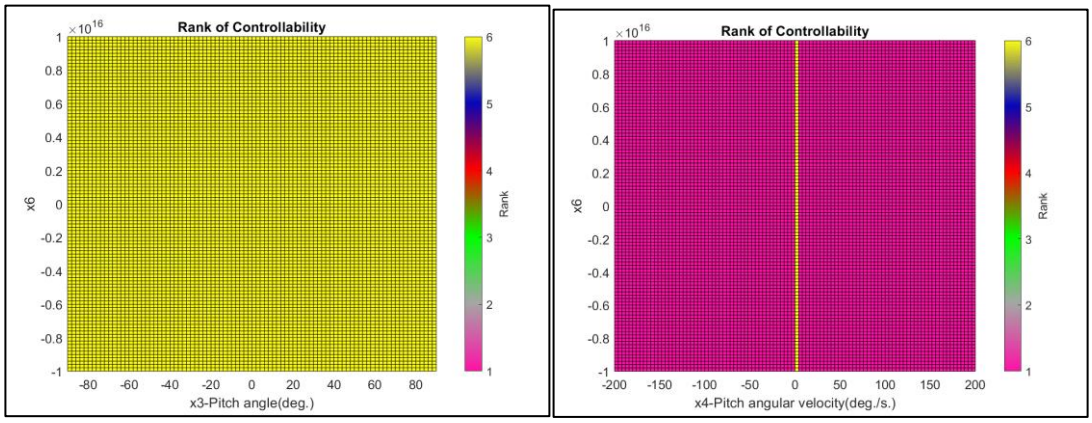


Figure 6.11: Controllability plot for the TWR system of Model B (with input saturation), (a) cross-section at x_4 =0°/s. (b) cross-section at $x_3 = 0°$.

Figures 6.10 and 6.11 display the rank test result of Model B in a 3D plot and in cross-sectional graphs, respectively. Figure 6.11(b) presents a very large region of Rank($\mathcal{C}$) = 1 (sliced at $x_3 = 0°$), indicating only one of the six poles of the system is controllable, similar to result shown in the unconstrained voltage simulation of Model B.

Moreover, the combined Model AB with a constrained control voltage, illustrates a broader fully controllable area as shown in Figure 6.12. Similar as before, cross sections are taken at $x_4 = 0°/s$ and $x_3 = 0°$, shown in Figure 6.13 (a) and (b), respectively. It can be seen from Figure 6.13(a) that the blue Rank($\mathcal{C}$) = 5 (not full rank) region, when from $-10° \leq x_3 \leq 10°$, from Figure 6.9 (a) is now fully controllable.

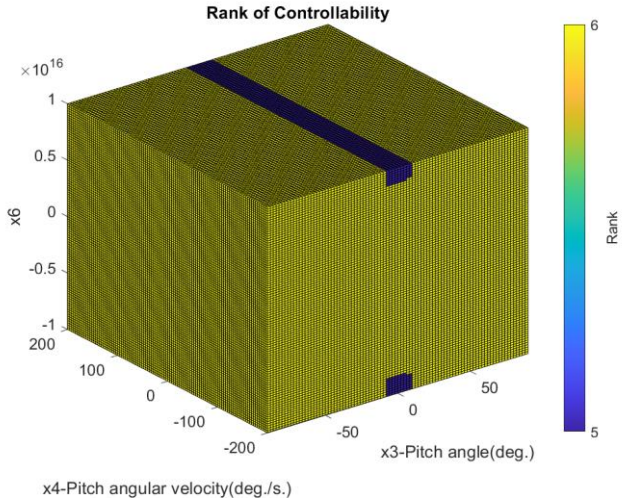


Figure 6.12: The rank of controllability matrix for the TWR system – with input saturation (Model AB)

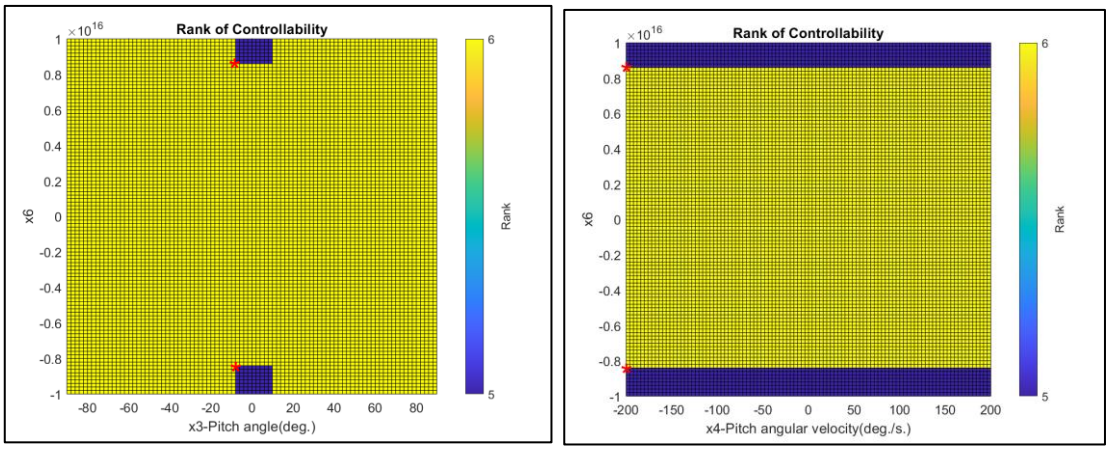


Figure 6.13: Controllability plot for the TWR with Model AB (input saturation), (a) cross-section at $x_4 = 0°/s$. Note, the coordinates (x_3, x_6) of the 2 points marked by red asterisks are $(-10°, -8.4 \times 10^{15})$ and $(-10°, 8.6 \times 10^{15})$, (b) cross-section at $x_3 = 0°$. Note, the coordinates (x_4, x_6) of the 2 points marked by red asterisks are $(-200°/s, -8.4 \times 10^{15})$ and $(-200\,°/s, 8.6 \times 10^{15})$,

6.4.2 Observability

With regard to observability test, it can be implemented by substituting matrices $\boldsymbol{A}$ and $\boldsymbol{C}$ into the Eq. (5.19) to provide the observability matrix $\boldsymbol{\mathcal{O}}$. Although matrix $\boldsymbol{C}$ is constant, matrix $\boldsymbol{A}$ is varied by state variables in the form of the nonlinear model. The rank of observability, hence, presented in a three-dimension subspace, is given in Figures 6.14-6.17.

- **Inverted Pendulum on a Cart System**

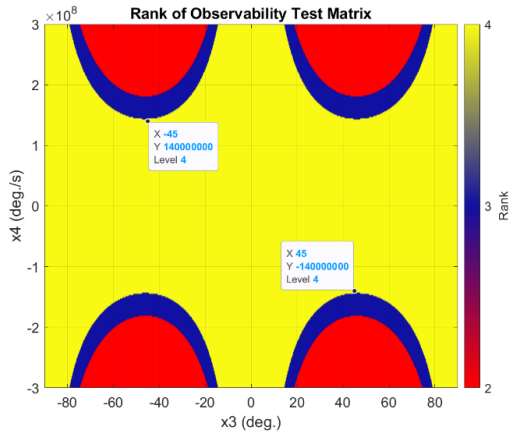


Figure 6.14: The rank of observability matrix for a 4th order nonlinear inverted pendulum system

It can be seen in Figure 6.14 that the fully observable region (Rank ($\mathcal{O}$) = 4, shown in yellow) appears in the centre of the x_3 and x_4 axes, and the regions which are not fully observable (Rank ($\mathcal{O}$) = 3 or 2) emerges when $|x_4| > 1.4 \times 10^8\ °/s$ and $15° < |x_3| < 78°$. In the latter regions, the 4th order inverted pendulum system is said to be partially state observable.

- **Two-Wheeled Robot System**

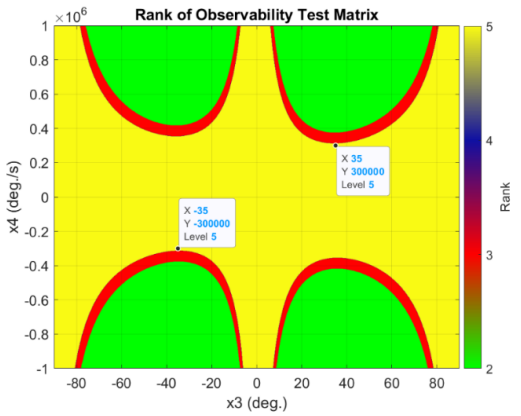


Figure 6.15: The rank of observability matrix for the 5th order nonlinear TWR system - without input saturation

Figure 6.15 demonstrates a full rank (Rank ($\mathcal{O}$) = 5) of the observability matrix in the central region, similar to the observability test result on the inverted pendulum system in Figure 6.14. However, the not fully observable regions appear over the absolute values of x_4 at approximately $3 \times 10^5\ °/s$, which is less than the x_4 value ($1.4 \times 10^8\ °/s$) shown in Figure 6.14. This is because the two-wheeled robot system is more complex than the inverted pendulum and they have different parameters and system dynamics.

Next, the input constraint is added to the TWR model and the rank of the new observability matrix is demonstrated in a three-dimensional subspace in the same way as the controllability test earlier, as given in Figures 6.16-6.17.

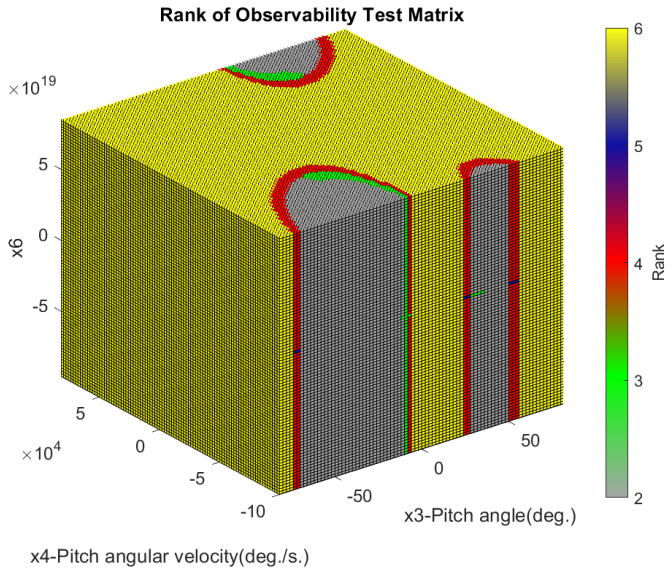
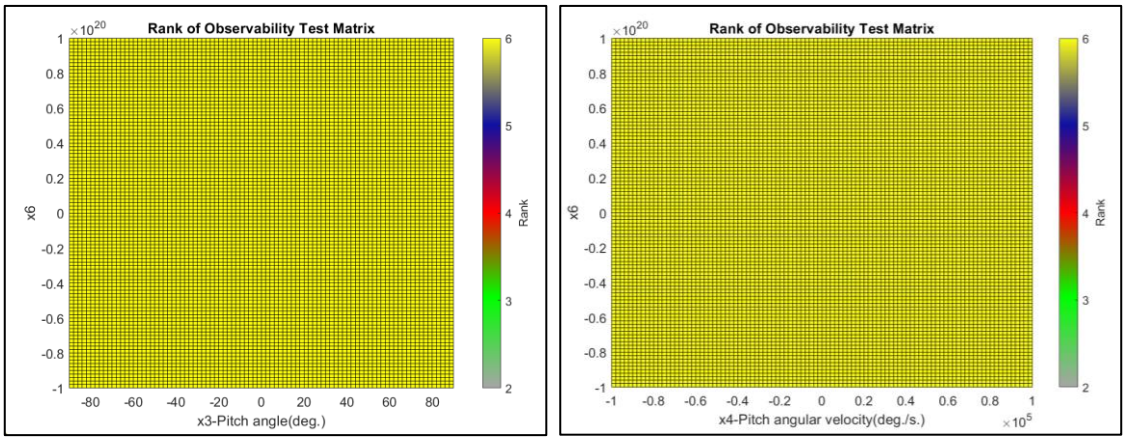


Figure 6.16: The rank of observability matrix for the 6th order nonlinear TWR system – with input saturation



(a) *(b)*

Figure 6.17: The rank of observability matrix for the 6th order nonlinear TWR system (with input saturation) by cross-section at (a) x_4=0°/s and (b) x_3=0°.

Note, Figure 6.16 demonstrates a cube, with x_3, x_4 and the constraint parameter x_6 as axes. The region is nearly entirely covered by Rank ($\mathcal{O}$) = 6 which implies complete observability; however, there are some non-full-rank areas when the absolute of x_4 goes beyond approximately 5×10^4 °/s, which is lower than the case in the unconstrained system (3×10^5 °/s) shown in Figure 6.15. Moreover, Figure 6.17 presents two cross-sections of Figure 6.16, when x_4=0°/s and x_3=0°, respectively. All areas in Figure 6.17 are yellow, representing Rank ($\mathcal{O}$) = 6 and the TWR system being full observable at these cross-sectional areas. This result illustrates that an extended Kalman filter is applicable to this system for the purpose of state estimations.

6.5 Simulation Results

In this section, the nonlinear freezing control method, the freezing technique with EKF, and the nonlinear iteration scheme will be applied to simulate the control problems of the inverted pendulum on a cart and the self-balancing two-wheeled robot model in the otherwise unstable vertical upright reference positions. Moreover, these methods will be compared with the traditional linear control result obtained in Chapter 5, such as LQR and the LQG under input unconstraint and constraint conditions.

6.5.1 Simulations of IP and TWR without Input Saturations

- Inverted Pendulum on a Cart System

In this subsection, the simulation results of stabilising an inverted pendulum on a cart system are demonstrated using two different nonlinear controllers, i.e., the freezing control and the iteration scheme, when the initial pitch angle x_3 is set from a range of values. The MATLAB programmes of the freezing control and iteration scheme are presented in Appendix A.6.7 and Appendix A.6.8, respectively. Furthermore, the weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are selected to be the same as in the LQR control in Chapter 5, for the purpose of easy comparison of outcomes between the linear and nonlinear control techniques.

To begin with, the simulation results of nonlinear freezing control are shown in Figures 6.18-6.19. In Figure 6.18, the graphs present the stabilisation of an inverted pendulum from three initial pitch angles: 60°, 75° and 80.5°.

Unsurprisingly, system responses and the control signal starting from the largest initial pitch angle 80.5°, generate strongest oscillations with large amplitudes. This angle is in fact the maximum initial pitch angle, of which the nonlinear freezing control method can stabilise, for the IP system. Beyond this angle, the system becomes unstable, producing unbounded output responses, as shown in Figure 6.19.

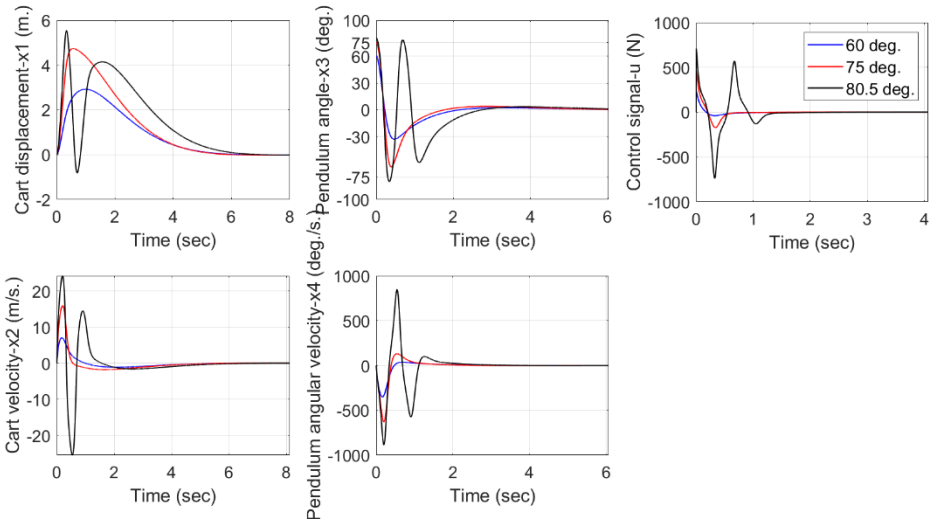


Figure 6.18: The stabilisation of an inverted pendulum system by the nonlinear freezing technique, from different initial pitch angles x_3

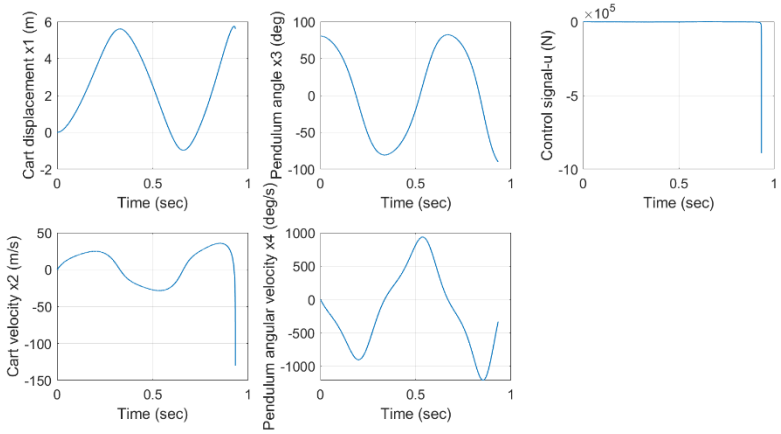


Figure 6.19: Unstable system response of an inverted pendulum system at the initial pitch angle x_3=80.6°, using the freezing technique

In the case of applying the iteration scheme technique to the inverted pendulum system, the simulation results are shown in Figures 6.20-6.22 as follows:

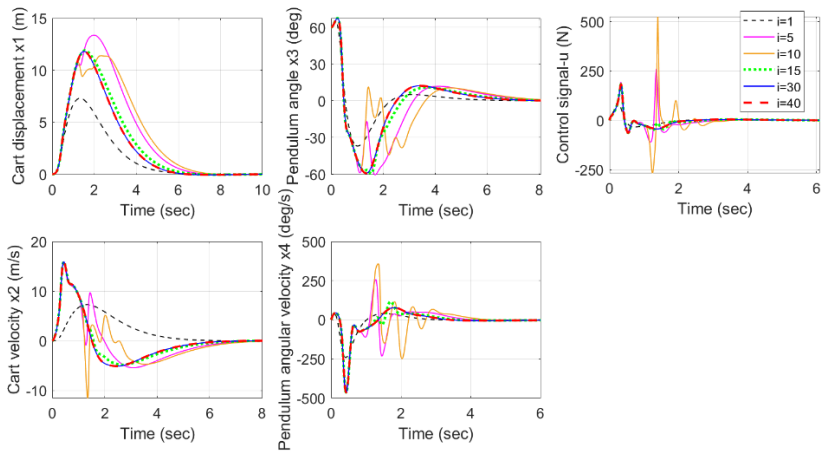


Figure 6.20: The stabilisation of an IP system from the initial pitch angle x_3= 60°, using the nonlinear iteration method, at different iteration steps

A number of iteration sequences, up to the 40th, of the nonlinear iteration control (also called LTV) to balance the inverted pendulum model, are plotted in Figure 6.20. Note, in this technique, the 1st iteration result, although appear to be smoother, is generally not considered. This is because the system matrices A and B are fixed using the initial conditions (see in Eq.(6.29)) rather than time dependent. The function of the 1st iteration is to generate state evolution results to be used in the next sequence. In Figure 6.20, the x_1 graph demonstrates that the 5th iteration displays the highest overshoot. The overshoot reduces with increasing state sequences, of up to the 15th iteration, and then converges at the 30th iteration (it can be seen that the 40th iteration presents the same result as the 30th). Furthermore, oscillations appear in the $x_2 - x_4$ graphs and the control signal

graph in low order iterations, i.e., 5th and 10th ; in contrast, the higher-order iterations such as the 30th and 40th present smooth responses.

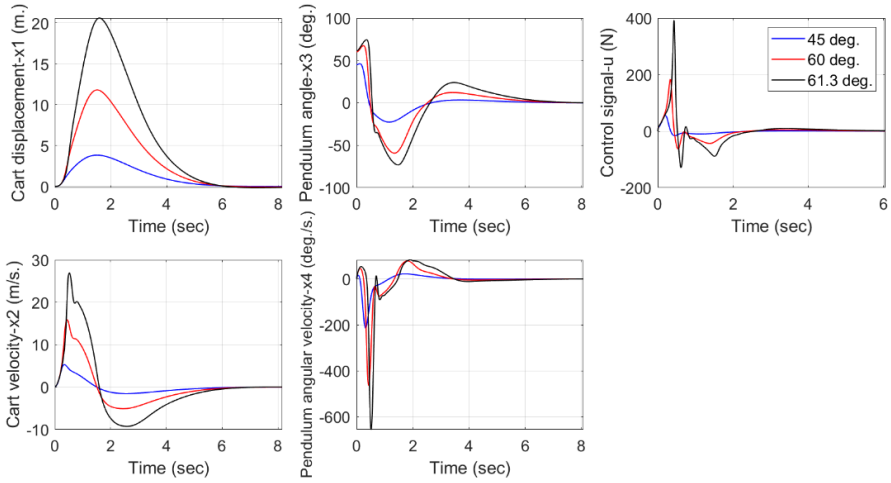


Figure 6.21: The stabilisation of an inverted pendulum system by the iteration scheme (40th) from different initial pitch angles x_3 – converged responses

Figure 6.21 demonstrates the converged response results when using the iteration scheme (the 40th iteration) to balance the inverted pendulum system, from three initial pitch angles: 45°, 60° and 61.3°. Comparisons of these graphs show that there are more oscillations associated with larger overshoots and undershoots when the initial pitch angle is increased, for all state variable responses and the control signal. Noticeably, spiky or erratic signals appear in the x_2, x_4 and control signal graphs, for the case starting from the initial pitch angle 61.3°. This is because the system reaches the iteration control limitation (the 40th iteration) at this angle and the system becomes unstable when operating beyond 61.3°, as presented in Figure 6.22.

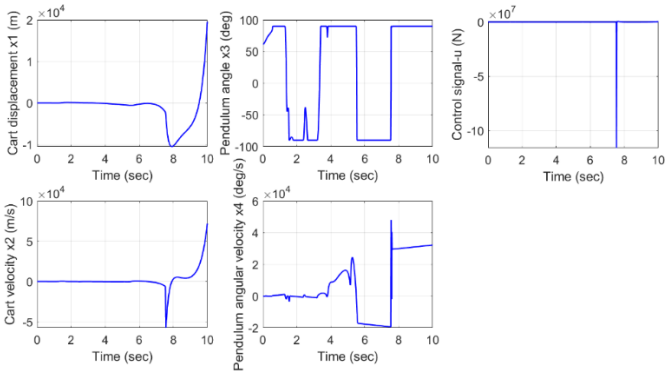


Figure 6.22: Unstable system responses from the initial pitch angle x_3=61.4°, using the iteration scheme (40th)

This research presents the iteration scheme without state observer; therefore, the freezing control without the Kalman filter and LQR controllers are selected to compare against the iteration scheme method. The stabilising system of an inverted pendulum model at the initial pitch angle of x_3= 42.9° from three controllers are illustrated in Figure 6.23.

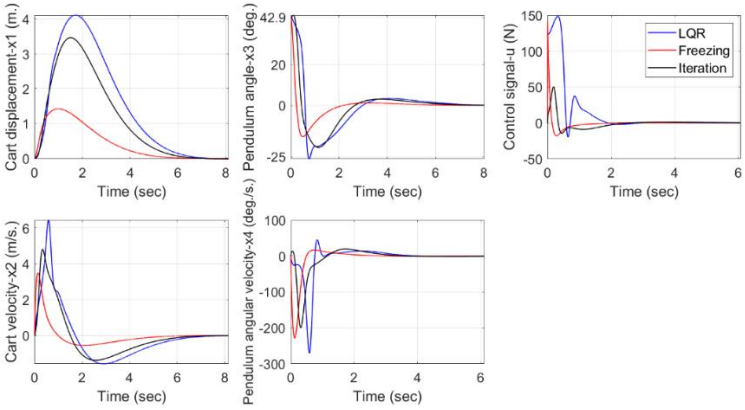


Figure 6.23: Stabilisation of the IP system using three different controllers, starting from an initial pitch angle x_3= 42.9°

It can be seen that the nonlinear freezing controlled system displays the lowest maximum overshoot in the x_1 graph, followed by the iteration scheme (order 40th) and then LQR. Significantly, the freezing technique also obtains the lowest

maximum deviations and has the shortest settling times in the x_2 and x_3 graphs. On the other hand, LQR, which is a linear controller, demonstrates the highest overshoot out of the three controllers. At this initial pitch angle, the LQR method generates some spiky responses as the system reaches its stabilisation limitation; in contrast, both nonlinear controllers produce much smoother response curves.

To summarise, both nonlinear controllers provide more comprehensive operational ranges than the linear method as the feedback gains are continuously calculated to reflect the changes of the time-dependent state variables. Furthermore, the freezing control demonstrates a higher capability than the iteration scheme in terms of shorter cart displacement and broader initial pitch angle ranges. Therefore, the freezing control technique has been selected to be the nonlinear controller used in the experimental subsection later; moreover, due to the memory limitation of the LEGO EV3 at 5 MB, it is extremely challenging to design the iteration scheme programming with various variables to be stored in the look-up table for being uploaded to the LEGO EV3 memory. In the case of the freezing technique, the programming codes have been generated at approximately 4.5 MB. Moreover, the approximation memory capacity of the iteration scheme at least doubled as there are additional state variables (x_1 and x_2), which are needed to be considered.

- Two-Wheeled Robot System

In this subsection, the simulation results of balancing a two-wheeled robot model (model A, defined by Eq. (4.61)) with different initial pitch angles x_3, under

no input constraint, are presented. Note that the weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ of the two-wheeled robot model are chosen to be the same as in the LQR and LQG controls analysed in Chapter 5 for easy comparison. Furthermore, the MATLAB programme of the nonlinear freezing control with and without an extended Kalman filter is demonstrated in Appendices A.6.9 and A.6.10, and uses the structures of feedback control, as shown in Figure 6.1 and Figure 6.2, respectively. Thus, the nonlinear freezing control results are presented in Figures 6.24-6.28.

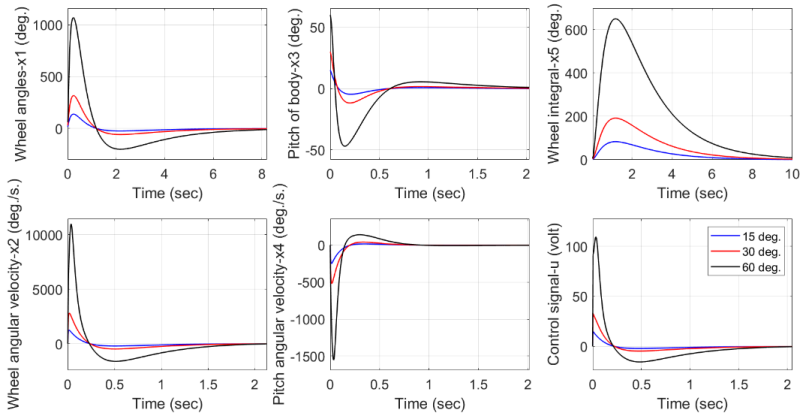


Figure 6.24: The stabilisation of a TWR system using freezing technique from different initial pitch angles x_3

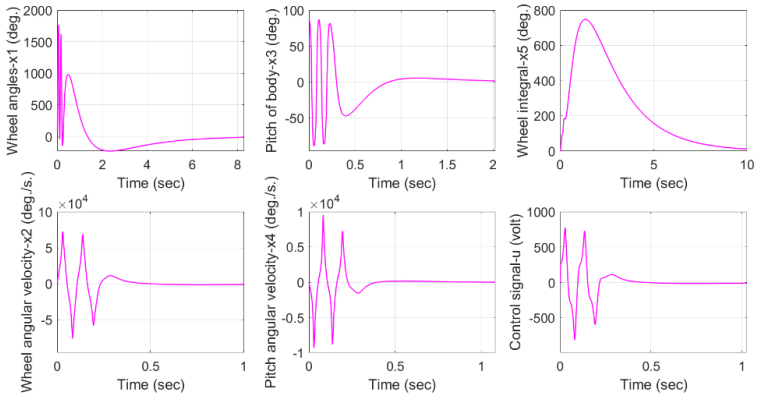


Figure 6.25: The stabilisation of a TWR system using freezing technique from initial pitch angle x_3= 87.2°

Figure 6.24 demonstrates the dynamical evolution of state variables and control signal, when applying the nonlinear freezing technique and changing initial pitch angle to x_3 = $15^{\circ}, 30^{\circ}$ and 60°, respectively. In the same way as the linear quadratic regulator control in Chapter 5 the deviation of all state variables and the control signal are increased when the initial pitch angle x_3 rises.

It can be seen that the pink curve in Figure 6.25, which represents results from an initial pitch angle x_3 = 87.2°, presents significant oscillations as the freezing technique reaches its maximum capability. Noticeably and importantly, the nonlinear freezing technique demonstrates the capability of stabilising the system starting from a much higher pitch angle than the LQR and the LQG, by approximately 21.5°, as shown in Chapter 5.

When going above the initial angle 87.2°, the system becomes unstable and crashes, as shown in Figure 6.26.

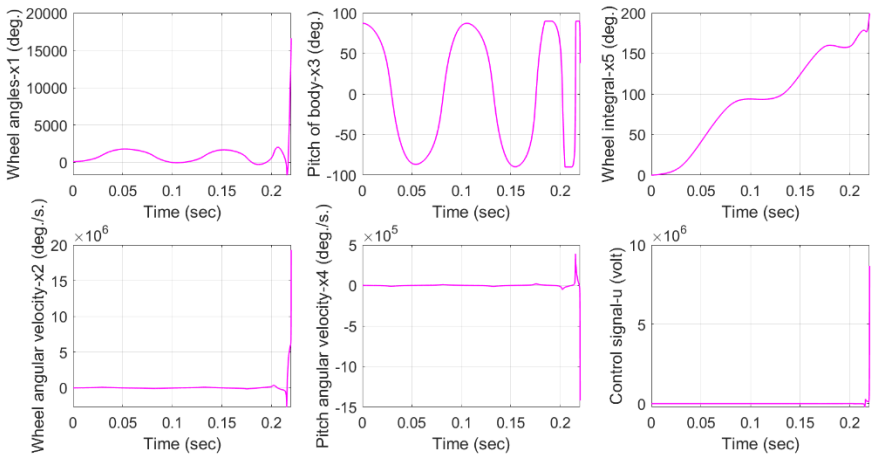
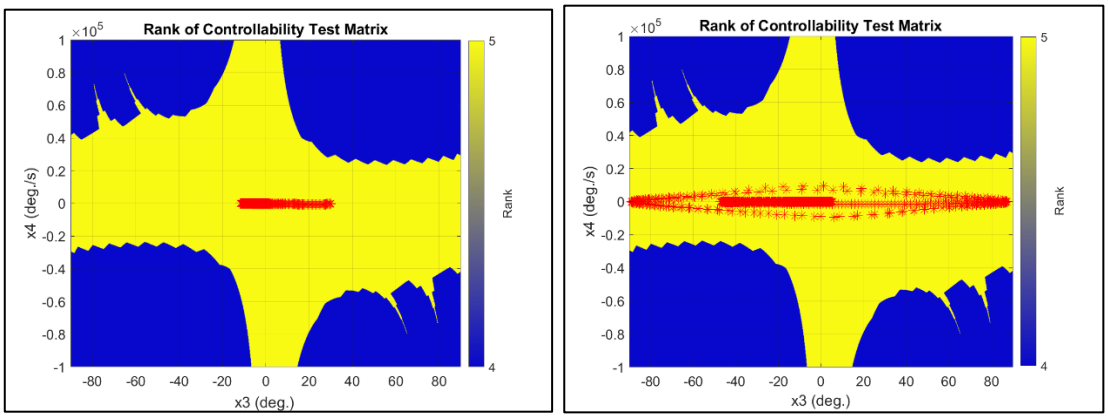


Figure 6.26: Uncontrollable system at the initial pitch angles x_3=87.3°, with the freezing technique applied

Recall that in Section 6.4.1, the controllability test of nonlinear freezing technique was described. The advantage of this method is its ability to display the rank of the controllability matrix at every evolution step of state variables x_3 and x_4 from any initial values. Figures 6.27-6.28 will present the rank of controllability matrix, combined with dynamic evolution of state variables x_3 and x_4 starting from different initial pitch angles, showing the controllable and uncontrollable areas.

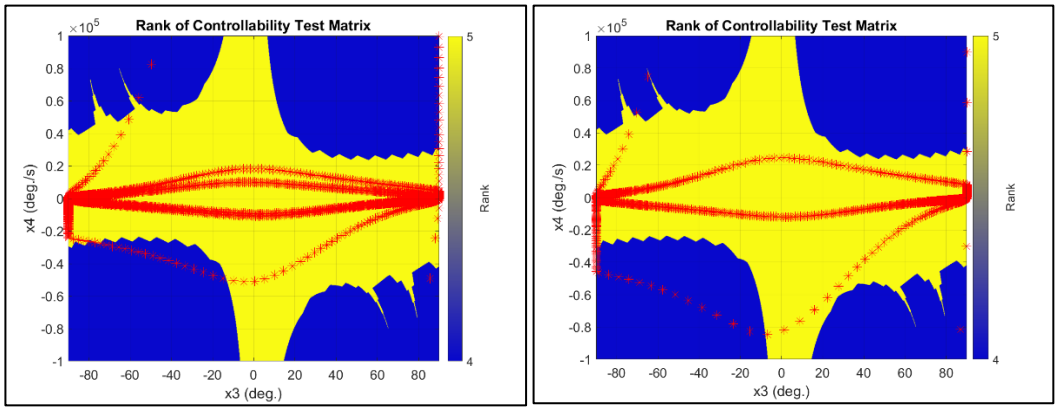


(a) *(b)*

Figure 6.27: The rank of controllability and dynamical evolution of state variables x_3 and x_4 at the initial pitch angles: (a) x_3 =30° and (b) x_3 =87.2°, controlled by freezing technique

Firstly, Figure 6.27 demonstrates the two stable systems with the nonlinear freezing control applied, when simulating from initial pitch angles $x_3 = 30^{\circ}$ and $x_3 = 87.2^{\circ}$. The dynamical state evolution trajectory (of x_3 and x_4) is shown in red stars, completely embedded in the yellow area representing full rank. Therefore, the system is fully controllable when starting from these x_3 values. Note, the red stars in Figure 6.27 (b) spread widely around the centre of

the figure; in contrast, they move in a small area in Figure 6.27 (a). This is because of the large oscillations of the state variable x_3 evolution, when starting at the initial pitch angle $x_3 = 87.2^{\circ}$, as shown in Figure 6.25.



(a) *(b)*

Figure 6.28: The rank of controllability and dynamical evolution of state variables x_3 and x_4 at the initial pitch angles: (a) x_3 =87.3° and (b) x_3 =90°, controlled by the freezing technique

In contrast to Figure 6.27, in Figure 6.28, some red stars appear in the blue regions, which represent rank deficiency, i.e., Rank($\mathcal{C}$)=4, when simulating from initial pitch angles $x_3 = 87.3^{\circ}$ and $x_3 = 90^{\circ}$. The results show that the system is now not fully controllable, which matches the simulation results shown in Figure 6.26 that the system cannot be stabilised using the nonlinear freezing control method.

Next, the combination of the freezing control technique with an EKF is investigated, where the magnitudes of state variables and control signal increase when the initial pitch angle is increased, similar to the freezing technique without EKF shown in Figure 6.29. Significantly, in Figure 6.30, the cut-off limitation of

the initial pitch angle (87.2°) is the same as the freezing technique without EKF. This is because that the addition of EKF supports the practical control implementation by providing filtered estimations of state variables, but do not contribute to the improvement of control capability in simulation.

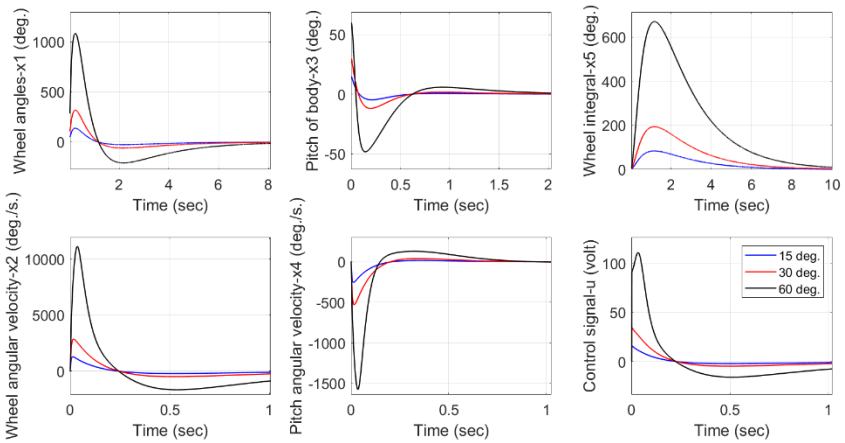


Figure 6.29: The stabilisation of a TWR system using the freezing technique with EKF from different initial pitch angles x_3

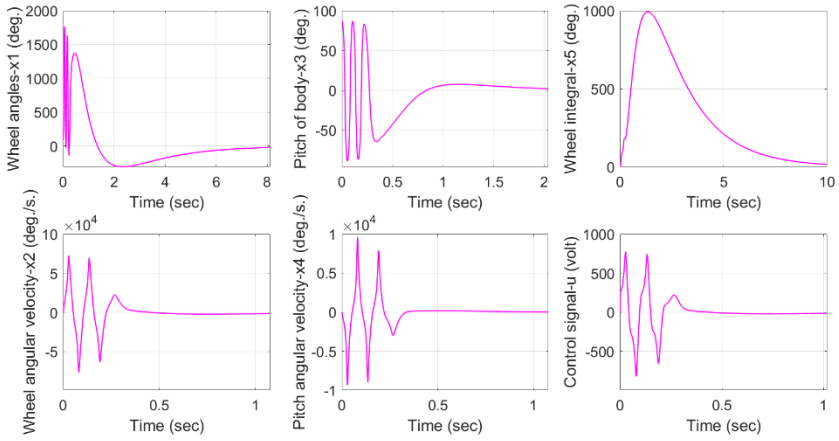


Figure 6.30: The stabilisation of a TWR system using the freezing technique with EKF from the initial pitch angle x_3=87.2°.

Similar as before, increasing the initial pitch angle to 87.3°or over leads to an unstable system response, when the freezing control and EKF fail to stabilise the TWR, as presented in Figure 6.31.

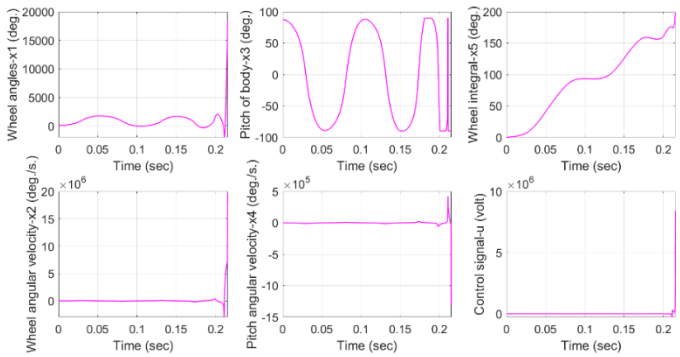


Figure 6.31: Unstable system from the initial pitch angles x_3=87.3°, using freezing technique with EKF

The above stability results are supported by the controllability test outcomes. As can be seen in Figure 6.32, the controllability matrices starting from the maximum controllable initial angle 87.2° stay full rank (the yellow region) during the evolutionary trajectory of x_3 and x_4 shown by red stars, indicating the system is fully controllable. But in Figure 6.33, when the initial pitch angle is set to 87.3°, the controllability test result shows some red stars appearing in the blue area where there is a rank deficiency. This means the system is not fully controllable and matches the unstable response observed in Figure 6.31.

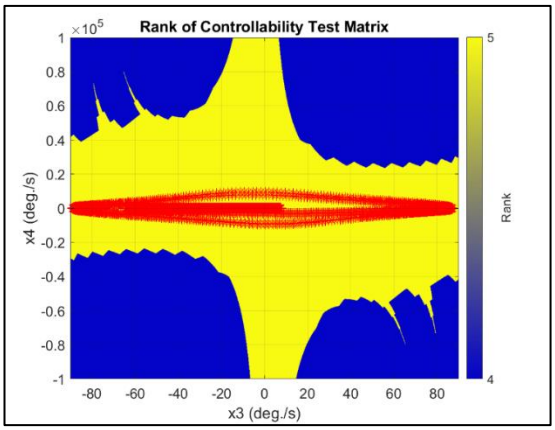


Figure 6.32: The rank of controllability matrix and dynamical evolution of state variable x_3 and x_4 from the initial pitch angles x_3 =87.2°, using freezing technique with EKF

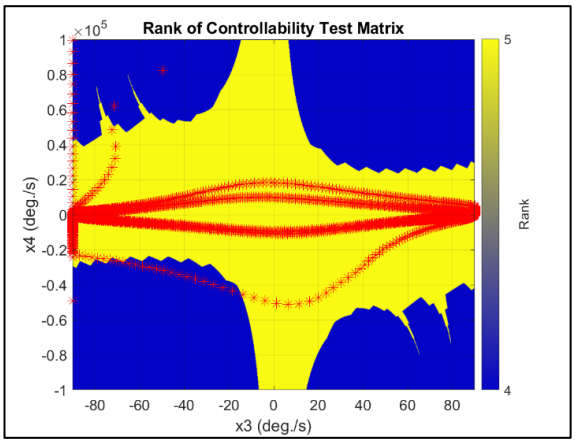


Figure 6.33: The rank of controllability matrix and dynamical evolution of state variable x_3 and x_4 from the initial pitch angles x_3 =87.3°, using freezing technique with EKF

Next, the LQR and LQG controllers analysed in Chapter 5 are compared against the nonlinear freezing control technique with and without EKF, as demonstrated below.

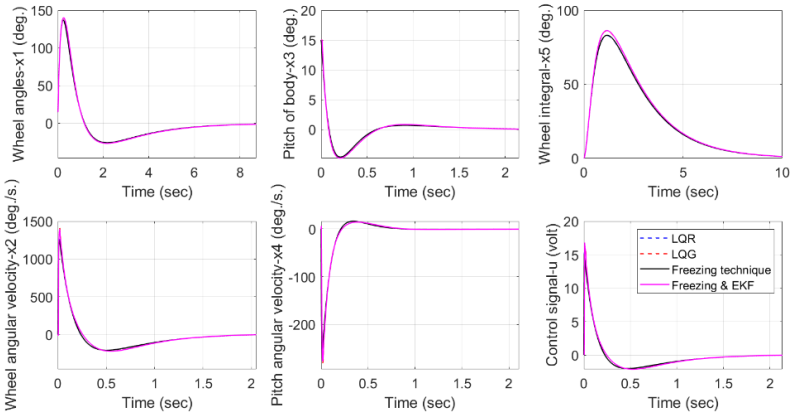


Figure 6.34: Stabilisation of a TWR system by four controllers from the initial pitch angles x_3= 15°

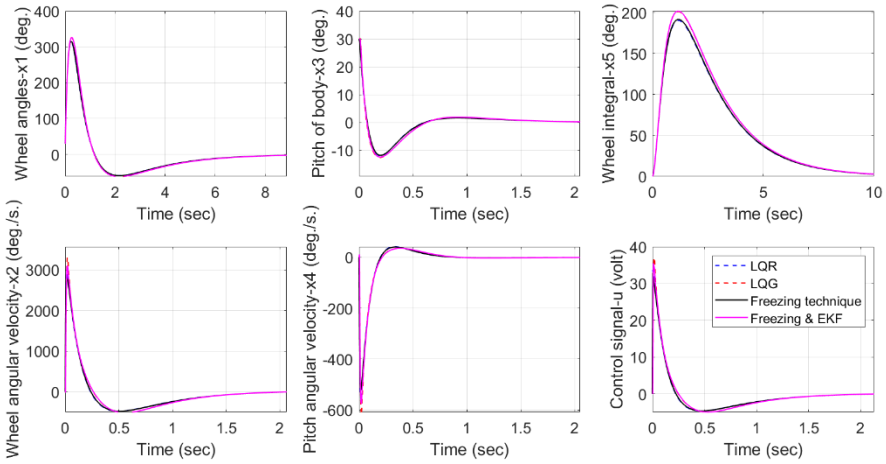


Figure 6.35: Stabilisation of a TWR system by four controllers from the initial pitch angles x_3= 30°

In Figures 6.34-6.37, the blue and red dashed curves represent control outcomes from the linear controllers, i.e., the LQR and LQG, respectively; the black and pink solid curves represent the responses from the nonlinear control methods, which are the freezing technique and freezing technique combined with EKF, respectively. It can be seen that the outcomes of four controlled systems are almost the same when starting from narrow initial pitch angles 15° and 30°.

When the initial pitch angle is increased to 60°, however, noticeable differences in the systems' responses between linear and nonlinear controls are present, as shown in Figure 6.36.

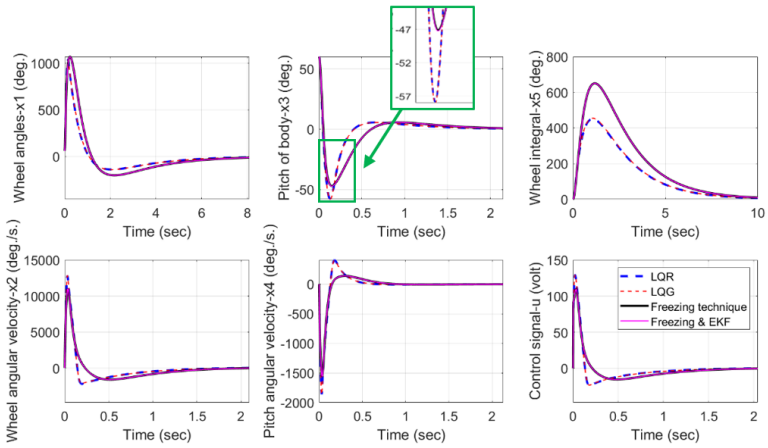


Figure 6.36: Stabilisation of a TWR system by four controllers from the initial pitch angle x_3= 60°

Firstly, the nonlinear controls display slightly higher deviations than linear methods in the wheel angle x_1 graph, causing the magnitudes of wheel angle integral x_5 also higher than the linear methods. However, the nonlinear methods demonstrate lower deviations in pitch angle x_3 than both linear methods by approximately 10° (see the magnified graph for x_3 response), which is an important improvement. In addition, there are some sharp changes in the linear controlled systems' responses, for state variable x_2, x_4 and control signal u, whereas the nonlinear control techniques present much smoother response, in comparison.

In Chapter 5, the maximum initial pitch angles which could be stabilised by applying the LQR and LQG methods were 65.7°. Hence, the performance of

linear controls at this initial pitch angle is selected to compare with the nonlinear methods (at the same angle), with results shown in Figures 6.37 and 6.38.

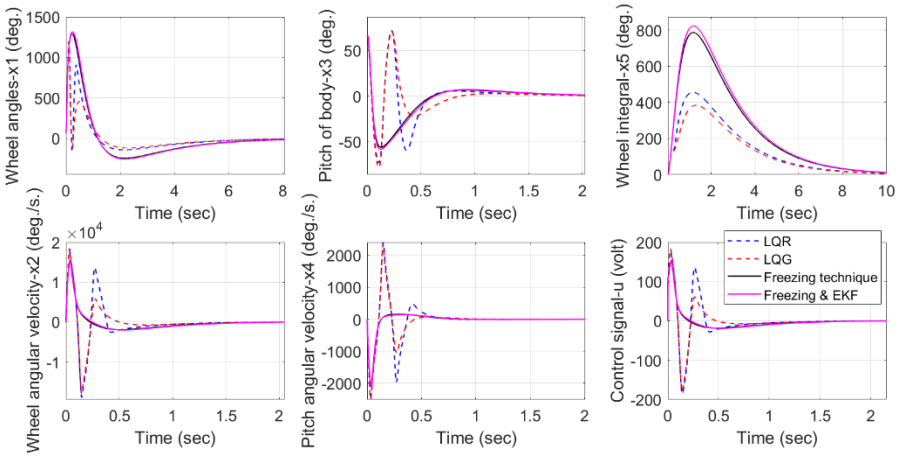


Figure 6.37: Stabilisation of a TWR system by four controllers from the initial pitch angles x_3= 65.7°

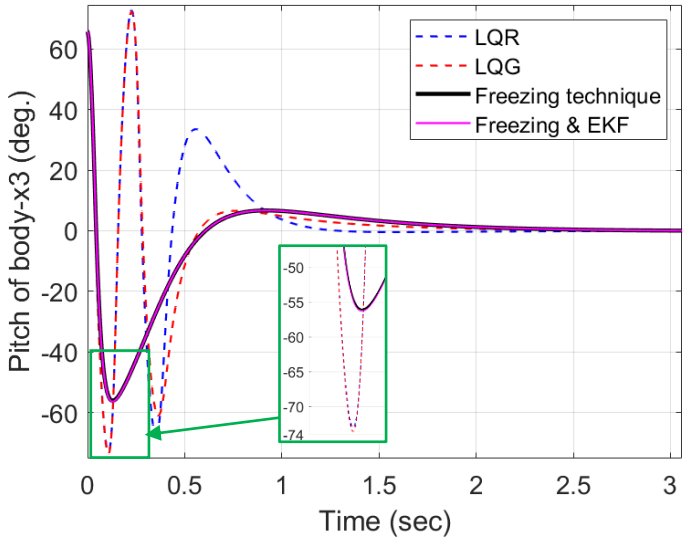


Figure 6.38: Magnified dynamical evolution of x_3 from the initial pitch angle x_3= 65.7° using four controllers.

Dramatic oscillations occur from the applications of both linear control techniques, on all state variables $x_1 - x_4$ and the control signal u in Figure 6.37;

on the other hand, the nonlinear control techniques demonstrate more stable results, displaying smooth curves on all signals. Note, in the magnified response graph of x_3, shown in Figure 6.38, large oscillations generated from the linear controls can be seen more clearly and are compared against the control outcomes from the nonlinear techniques. The lowest deviation value reached using the linear methods (approximately at -74°) is lower than using the nonlinear controls by approximately 18°. It can also be seen from this figure, the two nonlinear controls present similar results when stabilising the TWR system from the initial pitch angle x_3= 65.7°.

Figures 6.37 and 6.38 have also demonstrated that linear and nonlinear controls without input constraints generate quite different control responses, when the initial pitch angle is over 60°. This is because the feedback gains of the linear control are fixed for the linearised model (around the equilibrium point); in contrast, the nonlinear controller gains are always optimised globally and are therefore varying.

So far, the primary model (or Model A) has been used for the investigation of stabilisation control. Next, the other models, i.e., Model B and Model AB defined in Section 6.4.1 will be studied in the stabilisation simulations with different control techniques.

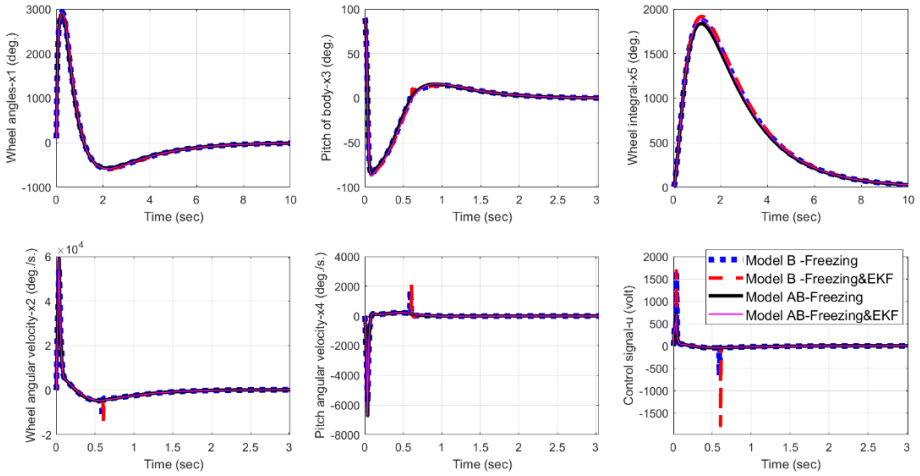
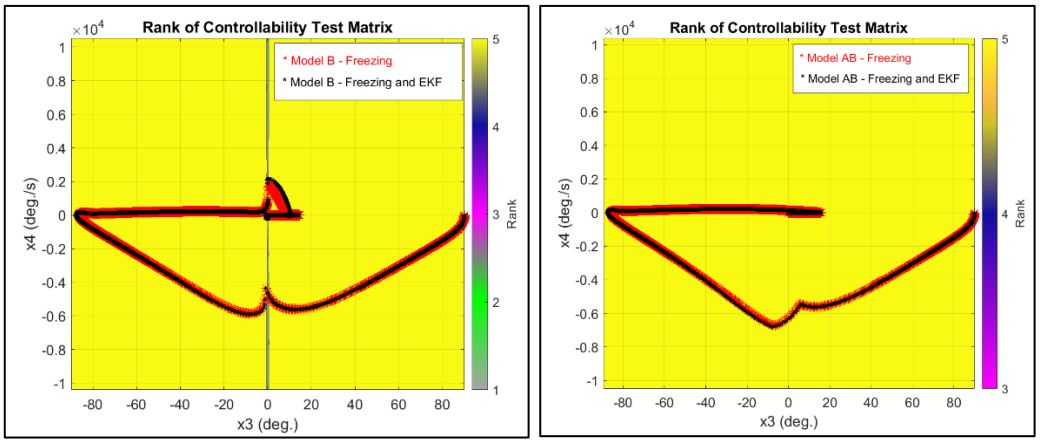


Figure 6.39: Stabilisation of a TWR system by freezing controllers, using Models B and AB at the initial pitch angle x_3= 90°

Significantly, Figure 6.39 shows that using Model B and Model AB, the freezing technique with and without EKF can stabilise the TWR system from an initial pitch angel of 90°, wider than both freezing techniques could achieve using the primary model (87.2°) and larger than any other techniques reported to be capable of achieving in the literature. Although the maximum deviations on all signals are much larger than the results shown when starting from 87.2° and the very large control voltage needed makes it impractical for physical realisations, this still demonstrates the outstanding control range that the nonlinear freezing control technique can achieve in theory and its superiority over other linear and nonlinear control methods. The control results on Model AB, in particular, have shown smooth response curves and illustrate the benefits of combining two models together.



(a) (b)

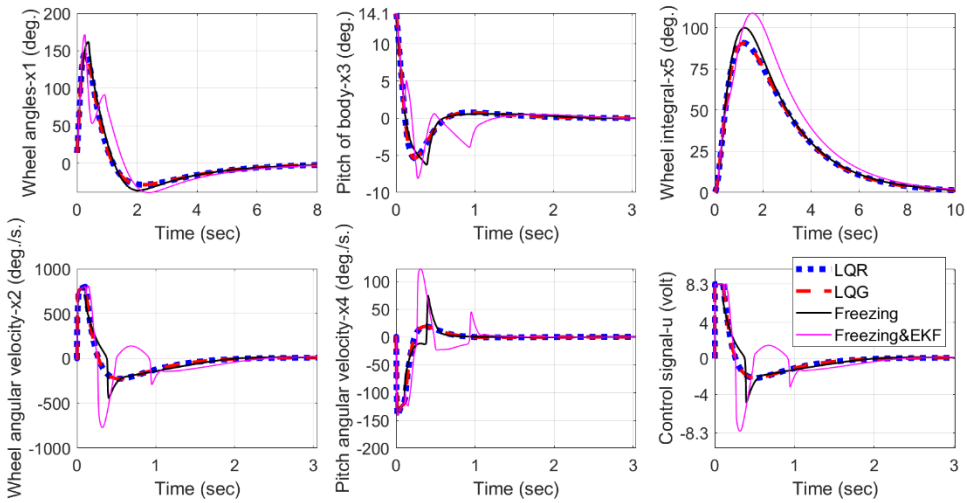
Figure 6.40: The rank of controllability matrix and dynamical evolution of x_3 and x_4 from the initial pitch angles x_3 =90°, using freezing technique alone and with EKF for: (a) Model B and (b) Model AB

Furthermore, Figure 6.40 (a) illustrates the cause of sharp signals observed in the Model B response graphs in Figure 6.39, mapping to the rank test (in Figure 6.40 (a)) result when the x_3 and x_4 evolution trajectory goes across to the uncontrollable region near $x_3 = 0^{\circ}$. In contrast, the dynamical evolution of x_3 and x_4 of Model AB shown in Figure 6.40 (b) lie in the completely controllable region, therefore its response curves in Figure 6.39 are smoother than the ones from Model B.

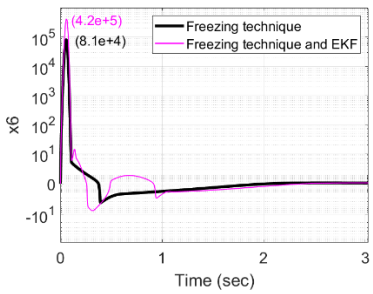
In these results, some generated magnitudes of the state variables as well the control signals are at unrealistically high levels as the simulation was conducted without taking into account of physical limits. The effect of limiting inputs based on hardware capacity in the simulation will be discussed in the next sections.

6.5.2 Simulations of TWR with Input Saturations

This subsection demonstrates the simulation of the two-wheeled balancing robot model with different initial pitch angles (x_3) between the linear and nonlinear controls with input saturation. In the cases of the LQR and LQG linear controls, the hard constraint was applied, as shown in Eq.(5.10); in contrast, the soft constraint defined in Eq.(6.13) will be used with the nonlinear freezing technique with and without EKF. The limitations of motor voltages are the same, at 8.3 V.



(a)

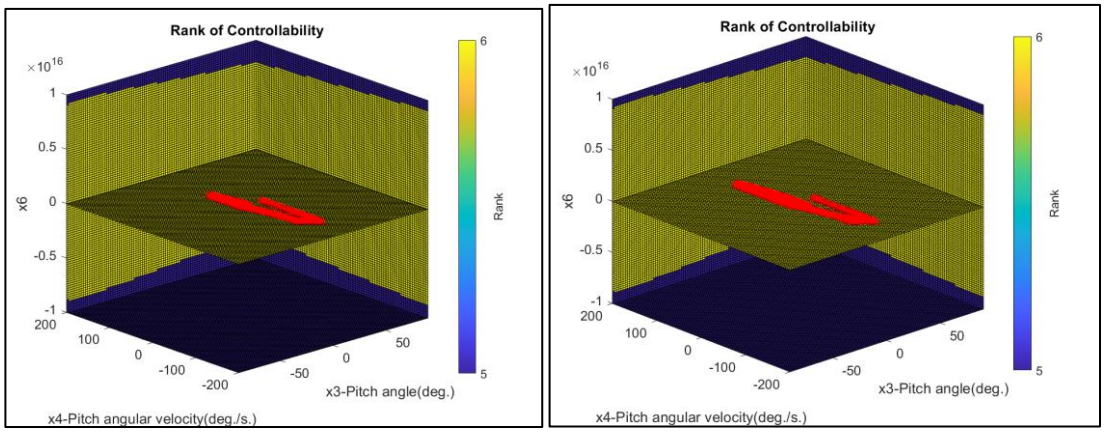


(b)

Figure 6.41: Stabilisation of a TWR system by four controllers from the initial pitch angle x_3= 14.1° with input saturation: (a) $x_1 - x_5$ and u against time and (b) x_6 with logarithmic scale against time.

Firstly, a narrow initial pitch angle x_3 at 14.1° is selected for the results shown in Figure 6.41, as the freezing technique with EKF provides the lowest initial angle for stabilising the TWR system amongst the four controllers under the input saturation condition. Note that both linear controls are applied with the hard constraint. It can be seen that slight oscillations appear in the graphs of state variables x_2, x_3, x_4 and control signal u in Figure 6.41(a). Moreover, the maximum deviation of all state variables in Figure 6.41(a) are almost the same because of the small initial pitch angle. Note, some state variables from the four controlled systems are restricted at the cut-off limits; for instance, the magnitudes of the control signal u reach the limitation at 8.3V and the wheel angle velocity graphs show the maximum magnitude at approximately $800^{\circ}/s$.

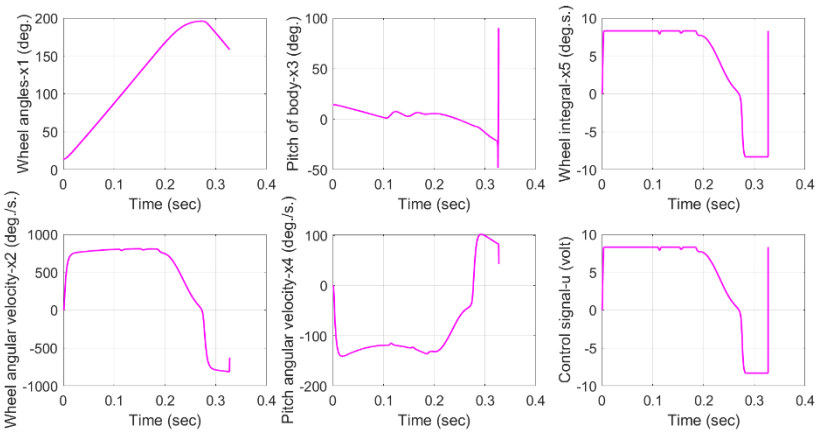
Figure 6.41(b), furthermore, presents the new state variable x_6 generated by nonlinear control with input constraint defined in Eq.(6.13). The maximum magnitude of the control signal with the freezing technique with EKF (4.2×10^5) is significantly higher than the without EKF (8.1×10^4). Significantly, in Figure 6.41(b), both plots show peak values lower than the limitations required of a controllable system (approximately 8.6×10^{15}), as presented in Figure 6.9 in Section 6.4.1. Moreover, it can be seen in Figure 6.42, that red stars representing the system's dynamical evolution path remain in the controllable area, when starting at this initial angle.



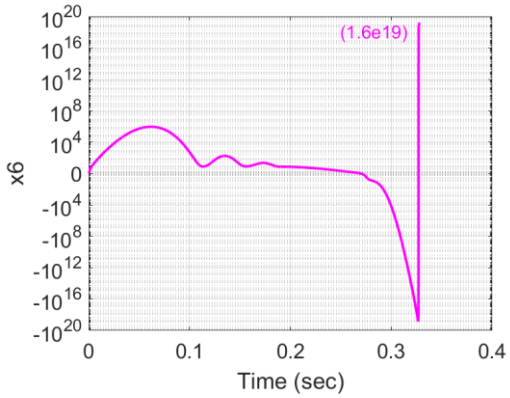
(a) *(b)*

Figure 6.42: The rank of controllability matrix and dynamical evolution of state variables x_3 and x_4 from the initial pitch angle x_3 =14.1°, using (a) freezing technique and (b) freezing technique with EKF.

When the initial pitch angle goes over 14.1°, the TWR system with control constraint cannot be stabilised by the freezing technique with EKF and the system becomes unstable and then crashes, as shown in Figure 6.43.



(a)



(b)

Figure 6.43: Unstable system using freezing technique and EKF from initial pitch angle x_3= 14.2° with input saturation: (a) $x_1 - x_5$ and u against time and (b) x_6 with logarithmic scale against time.

It can be seen from Figure 6.43(b) that the maximum value of x_6 is now approximately 1.6×10^{19}, higher than the limit of a controllable system (approximately 0.8×10^{16}).

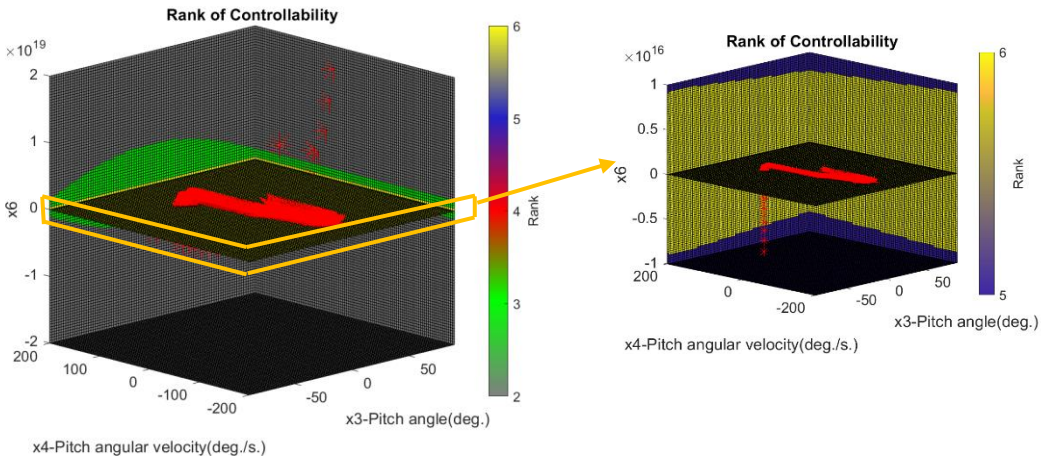
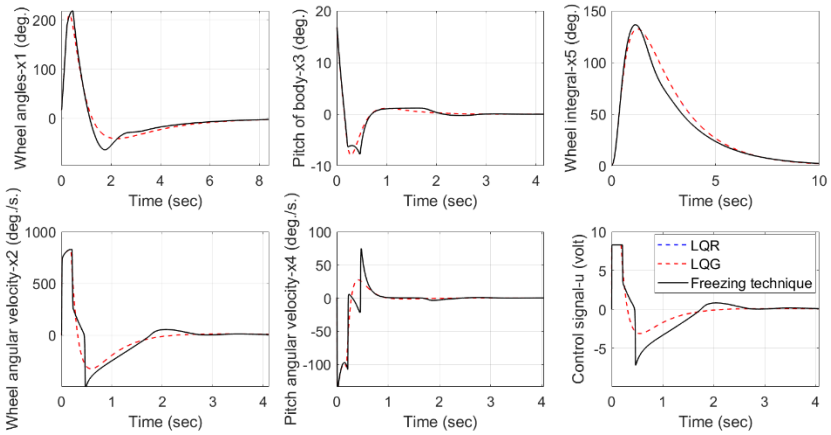


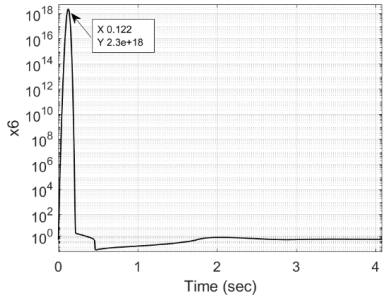
Figure 6.44: The rank of controllability matrix and dynamical evolution of state variables x_3 and x_4 from the initial pitch angle x_3 =14.2°, using freezing technique and EKF

This is illustrated in Figure 6.44, where the dynamical evolutions of state variables x_3 , x_4 and x_6 display that whilst they start in the fully controllable area (yellow plate) between $x_6 = -0.8 \times 10^{16}$ and 0.8×10^{16} in the magnified figure, the red stars then enters the not fully controllable area (up to approximately 1.6×10^{19}). This causes failure to the controlled system.

When the initial pitch angle x_3 is risen to 16.8°, the standard freezing technique with saturation reaches its control limit. At this angle, three of the four controllers are still functioning well in stabilising the system model but the freezing technique with EKF can no longer operate, as shown in Figure 6.45.



(a)



(b)

Figure 6.45: Stabilisation results from three controllers at the initial pitch angles x_3= 16.8° with input saturation: (a) $x_1 - x_5$ and u against time and (b) x_6 with logarithmic scale against time.

At this initial pitch angle, the maximum deviations of linear controls and nonlinear freezing control are almost of the same values, as shown in Figure 6.45(a). Moreover, the linear methods demonstrate smoother curve signals than the freezing technique; for example, sharp curves appear at the undershoot in the state variable x_2, x_3 and control signal u when using the nonlinear method. Furthermore, in Figure 6.45 (b), the new state variable x_6 when using the freezing technique presents a magnitude of approximately 2.3×10^{18} , which is beyond

the fully controllable requirement (at approximately 0.8×10^{16}), but the system is still stabilisable. This is because, in some cases, it is still possible to partially control the system when it presents rank deficiency as long as the unstable modes are controllable (Dutton, Thompson, & Barraclough, 1997), as shown by the red stars outside the fully controllable area in Figure 6.46. The red stars appear around Rank($\mathcal{C}$) = 3 (green), which is not of full rank; however, it is still stabilisable. Note the fully controllable area has Rank($\mathcal{C}$) = 6.

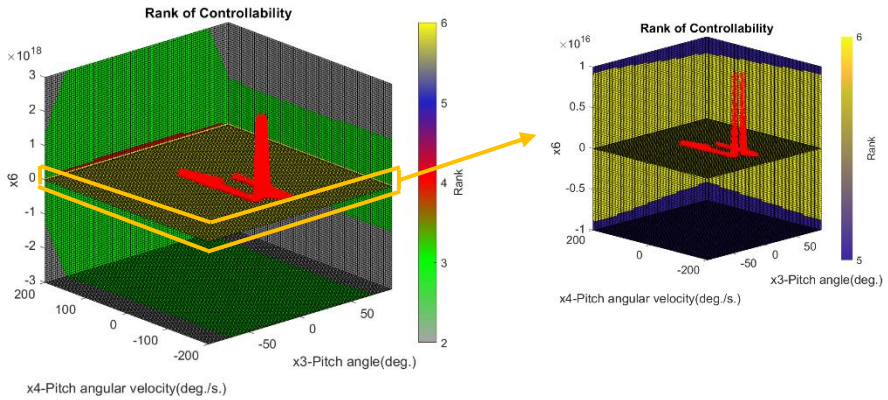
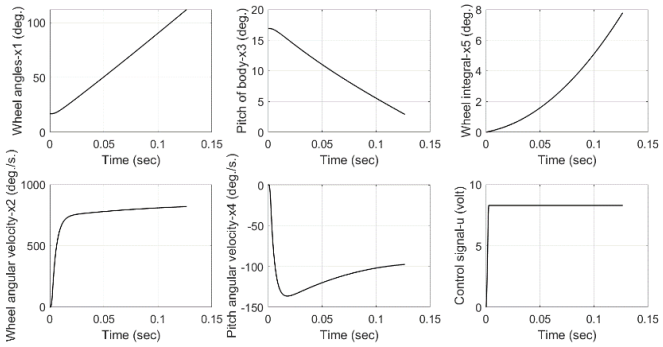
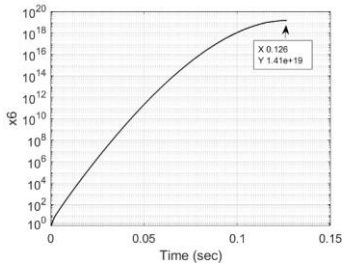


Figure 6.46: The rank of controllability matrix and dynamical evolution of state variables x_3 and x_4 at the initial pitch angle x_3 =16.8°, using freezing technique

When the initial pitch angle is over 16.8°, the system controlled by the nonlinear freezing technique with soft constraint is unstable, as shown in Figure 6.47. The highest value of the state variable x_6 is approximately 1.4×10^{19} in Figure 6.47(b), which is over the range of a controllable system. This is shown by the controllability rank graph in Figure 6.48, where the red stars occur outside the fully controllable area.



(a)



(b)

Figure 6.47: Unstable system response using freezing technique at the initial pitch angle x_3= 16.9° with input saturation: (a) $x_1 - x_5$ and u against time and (b) x_6 with logarithmic scale against time.

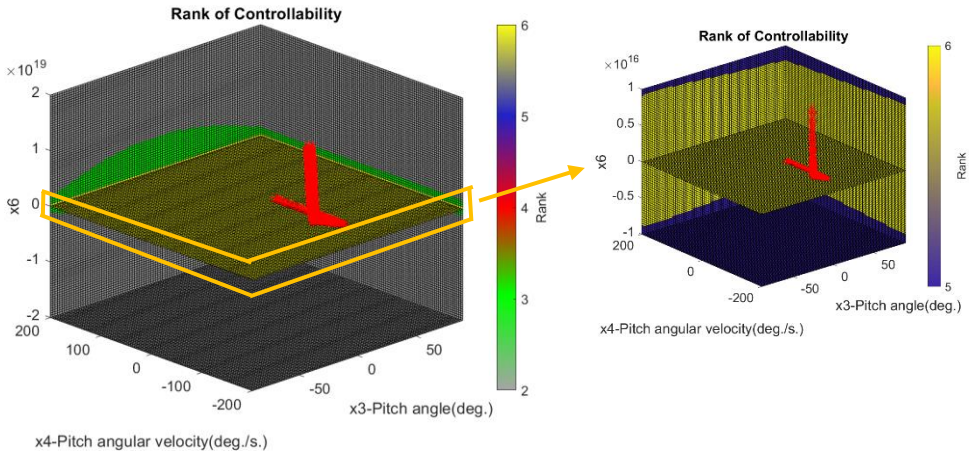
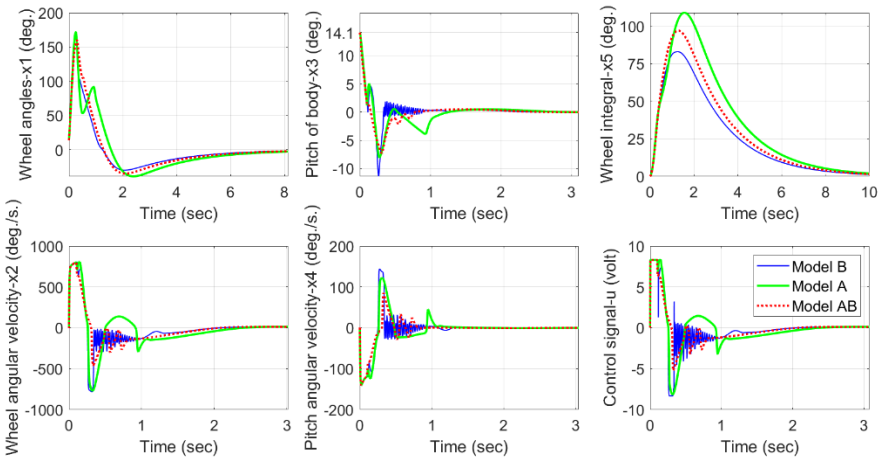


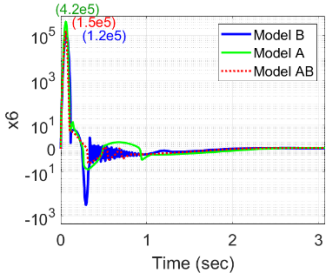
Figure 6.48: The rank of controllability matrix and dynamical evolution of state variables x_3 and x_4 at the initial pitch angle x_3 =16.9°, using freezing technique

Next, Models B and AB are simulated using the freezing control with EKF with constrained voltage input, compared against Model A. Note: the freezing control with EKF is selected for simulation in this subsection instead of the stand-alone freezing technique, because the Kalman filter is advantageous in reducing the gyro sensor drift issue in the practical experiments later.

Simulation results of the alternative models are given in Figures 6.49-6.54.



(a)



(b)

Figure 6.49: Stabilisation of the TWR system using freezing controllers with EKF from on Models A, B and AB at the initial pitch angles x_3 =14.1°: (a) $x_1 - x_5$ and u against time and (b) x_6 with logarithmic scale against time.

Model B reaches the same limitation of maximum initial angle for stabilisation as Model A, at 14.1°, but it can be seen from Figure 6.49 that there are severe oscillations in several response graphs. This is because of the not fully controllable area near $x_3 = 0^{\circ}$ of Model B, causing vibrations in the system response. In terms of Model AB, the oscillations are much less frequent than Model B and are generally smoother than the results obtained for Model A. Furthermore, the controllability rank test results of Models B and AB are demonstrated in Figure 6.50. It is evident that red stars lie mostly in the yellow regions of both graphs. The pink lines in the Model B controllability graph corresponds to controllability deficiency, which explains the high frequency oscillations in graphs shown in Figure 6.49.

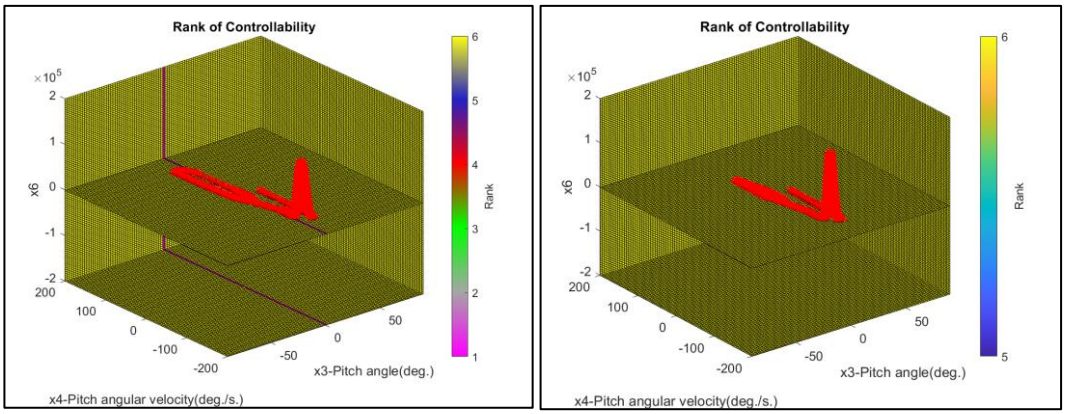
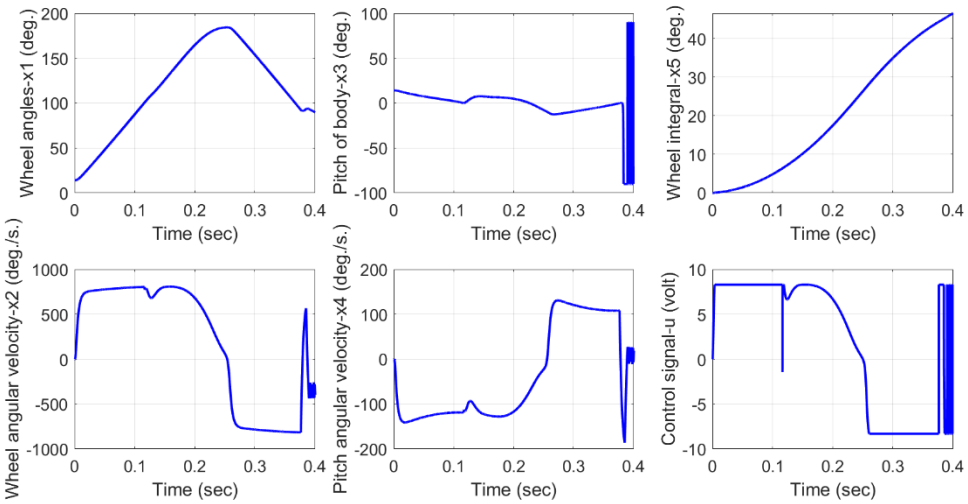
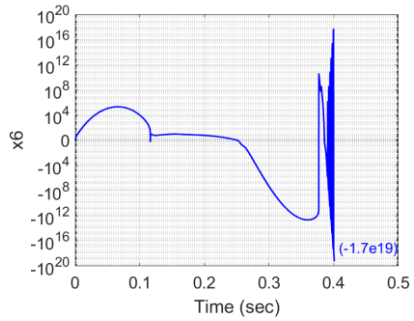


Figure 6.50: The rank of controllability matrix and dynamical evolution of state variables x_3 and x_4 from (a) Model B and (b) Model AB, at initial pitch angle x_3 =14.1°, using freezing technique and EKF

Beyond this initial pitch angle (14.1°), Model B cannot be stabilised using freezing and EKF method and crashes, as shown in Figure 6.51.



(a)



(b)

Figure 6.51: Uncontrollable system responses generated using freezing controllers with EKF for Model B at the initial pitch angle x_3 =14.2°: (a) $x_1 - x_5$ and u against time and (b) x_6 with logarithmic scale against time.

Furthermore, the rank of the controllability matrix of Model B from x_3 =14.2° is demonstrated in Figure 6.52. It can be seen that red stars appear outside the yellow area (fully controllable region) and reaches x_6 values of approximately -1.7×10^{19}, causing the system to be not fully controllable.

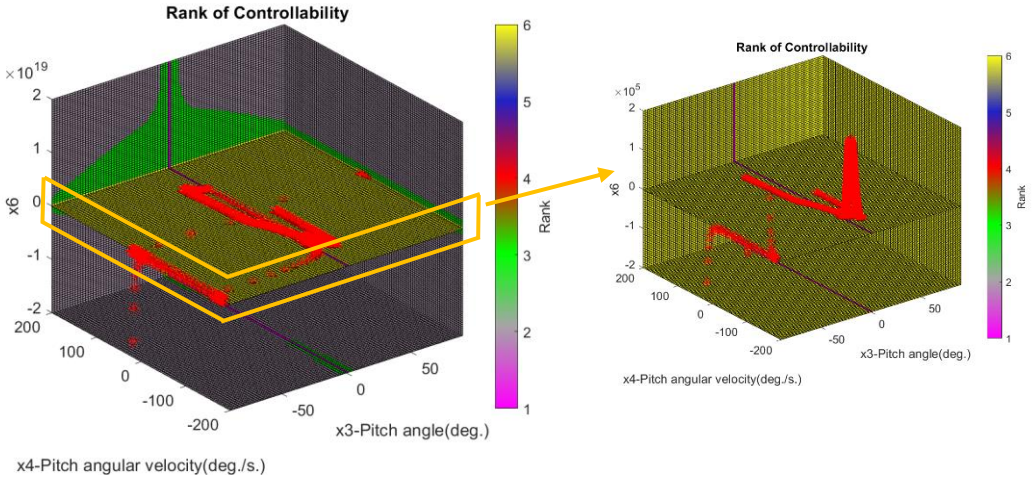
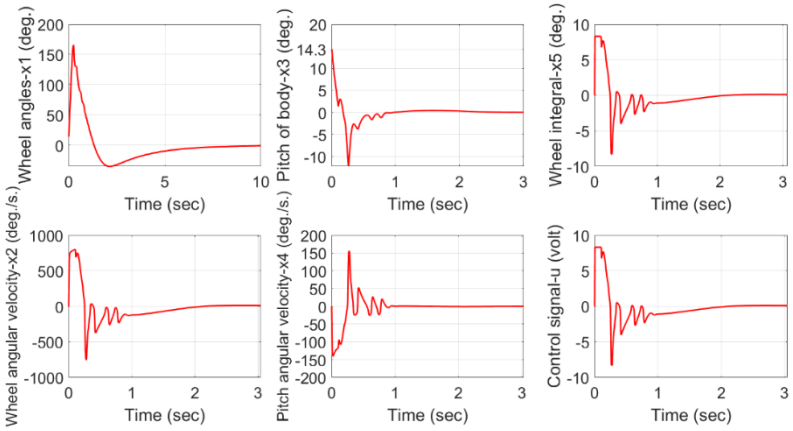
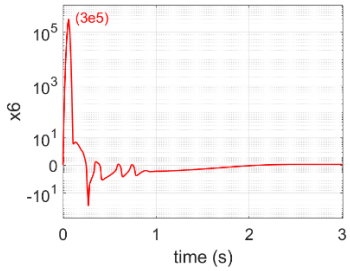


Figure 6.52: The rank of controllability matrix and dynamical evolution of state variables x_3 and x_4 of Model B from initial pitch angle x_3 =14.2°, using freezing technique and EKF (Right figure: Magnified)

Simulation has shown that the maximum initial pitch angle which can be stabilised from Model AB is slightly increased to 14.3° , which is the broadest angle comparing against Models A and B with control results presented in Figure 6.53. At this maximum initial pitch angle, high-frequency oscillations appear before the signals settle down. Moreover, controllability results show red stars travelling inside fully controllable region shown in Figure 6.54, which matches the stable responses observed in Figure 6.53.



(a)



(b)

Figure 6.53: Stabilisation of Model AB using freezing controller with EKF at initial pitch angle x_3 =14.3°: (a) $x_1 - x_5$ and u against time and (b) x_6 with logarithmic scale against time.

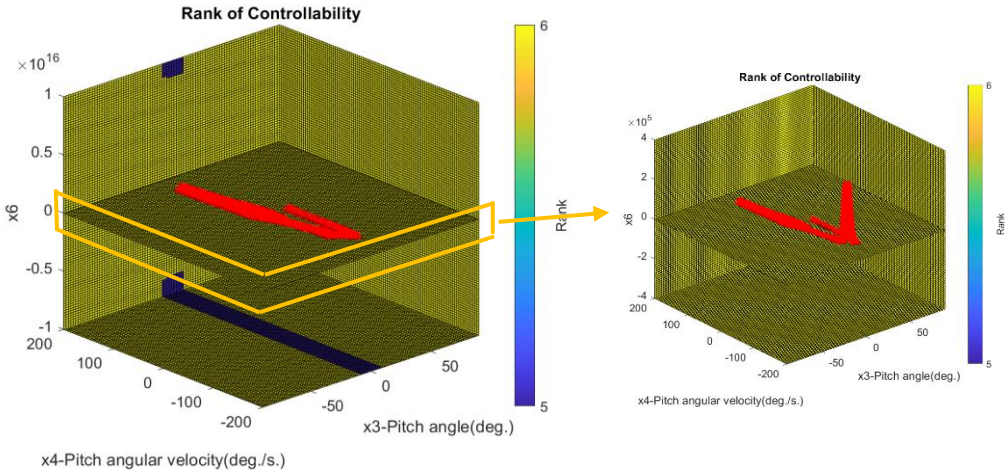


Figure 6.54: The rank of controllability matrix and dynamical evolution of state variables x_3 and x_4 of Model AB from initial pitch angles x_3 =14.3°, using freezing technique and EKF (Right figure: Magnified)

As soft constraints of the control input were presented above, an example of hard input constraint is investigated next and system responses are shown in Figure 6.55, where the control outcomes from x_3= 20.9° on Model A are identical using the four different controllers, namely, LQR, LQG, standalone freezing and freezing & EKF combined. Moreover, the maximum initial pitch angle of the nonlinear controls with hard constraint is also the same as the linear methods at 20.9°.

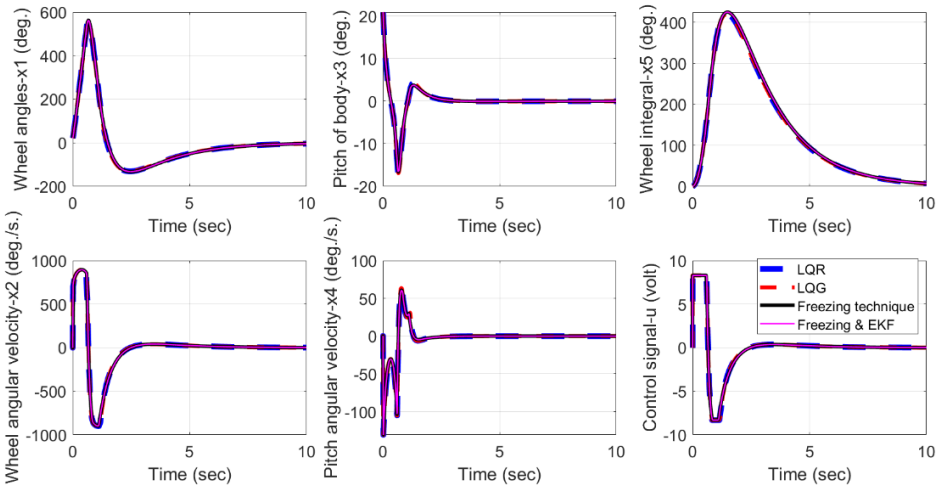


Figure 6.55: Stabilisation of the TWR system (Model A) by four controllers with hard constraint at initial pitch angle x_3=20.9°.

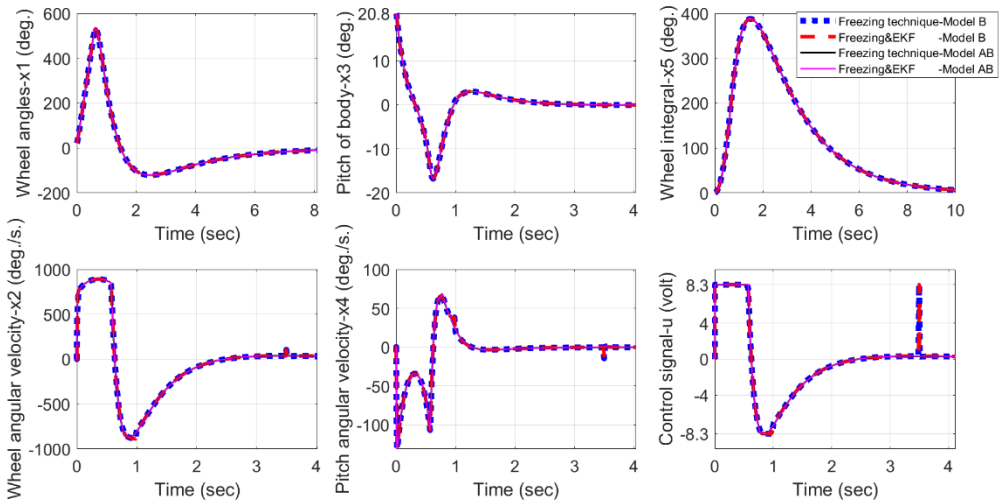


Figure 6.56: Stabilisation of the TWR system using four controllers on Models B and AB with hard constraint at the initial pitch angle x_3= 20.8°

Next, it can be seen from Figure 6.56 that control outcomes using the standalone freezing technique and freezing combined with EKF on Models B and AB, from x_3= 20.8° are also identical (when hard control constraints are applied), shown by the overlapping curves on the six graphs. However, they present a maximum initial pitch angle of x_3= 20.8°, which is slightly smaller than Model A.

So far, the simulations demonstrated balancing of the two-wheel robot at a maximum power supply of 8.3V, which is the limitation from a LEGO EV3 robot; however, in Chapter 3, motor specification of the maximum voltage of the LEGO EV3 motor from supplier is 12V. Therefore, Table 6.1 illustrates brief simulation results of stabilising a TWR model with the maximum initial pitch angles if the voltage supply reaches 12V maximum, compared to 8.3V.

Table 6.1 : The maximum initial pitch angle stabilisable from different controllers with input constraints 8.3V and 12V.

Controllers	Input constraint 8.3V	Input constraint 12V
Hard constraint, Model A		
LQR	20.9°	29.1°
LQG	20.9°	29.1°
Freezing technique	20.9°	29.2°
Freezing with EKF	20.9°	29.2°
Soft constraint, Model A		
Freezing technique	16.8°	22.8°
Freezing with EKF	14.1°	19.3°
Hard constraint, Alternative models		
Freezing with EKF (Model B)	20.8°	29.1°
Freezing with EKF (Model AB)	20.8°	29.1°
Soft constraint, Alternative models		
Freezing with EKF (Model B)	14.1°	19.5°
Freezing with EKF (Model AB)	14.3°	19.7°

In Table 6.1, when the input constraint is increased to 12V, as shown in the 3rd column, all controllers present significantly higher stabilisation ranges of the TWR, than previously at 8V. Note, the maximum pitch angles of all controls with hard constraint at 12V nearly reach 30°, rising by over 8°. When combined with soft constraint, the maximum pitch angles of nonlinear methods on different model forms have all increased by $5^{\circ} - 6^{\circ}$. It is evident that soft and hard constraints lead to different maximum initial pitch angles. In Section 6.6, the

experimental result using LEGO hardware will be compared against simulation outcomes of soft and hard constraints, for further demonstration and analysis.

Next, DC motors with voltages higher than a standard LEGO EV3 motor is investigated and their specifications are presented in Table 6.2.

Table 6.2: Motor specifications from Maxon company, series EC 32 flat 15W (Maxongroup, EC 32 flat 15W, 2020)

	Motor series EC 32 flat 15W		
1.Nominal voltage (V)	12	24	48
2.No load speed (rpm)	4,610	4,530	4,780
3.Nominal torque (mNm)	25	25.5	24.7
4.Nominal current (A)	1	0.5	0.257
5.Terminal resistance (Ω)	3.51	13.8	53.1
6.Torque constant (mNm/A)	24.1	49	92.8
7.Rotor inertia (gcm^2)	35		
8. Weight (g)	57		
9. Diameter (mm)	32		

Three motors from the Maxon series EC 32 flat, shown in Table 6.2, are selected to for simulation, combining with the LEGO EV3 robot model parameters. Here, it is assumed that the power supply can vary within the maximum motor voltage range, and the mass is similar to the current robot power supply. The simulation outcomes of stabilising the TWR Model A, using these motors, are summarised in Table 6.3.

Table 6.3: The maximum initial pitch angle stabilisable from different controllers, with varying motor voltages.

Controllers	Motor 12V 15W (Nominal torque 25 mNm)	Motor 24V 15W (Nominal torque 25.5 mNm)	Motor 48V 15W (Nominal torque 24.7 mNm)
Hard constraint			
LQR	28.4°	29.8°	29.6°
LQG	28.4°	29.8°	29.6°
Freezing technique	28.7°	29.9°	29.6°
Freezing with EKF	28.7°	29.9°	29.6°
Soft constraint			
Freezing technique	22.9°	22.1°	20.7
Freezing with EKF	20.0°	19.6°	18.9°

Note, from Table 6.3, there are only small differences amongst the maximum initial pitch angles which can be stabilised by each control method, when the motor voltage is increased from 12V to 48V. For instance, the maximum initial angle achieved from using the LQR controller with motor voltages 12V, 24V and 48V (all with similar nominal torques at ~25 mNm) are 28.4°, 29.8° and 29.6°, respectively. Moreover, the outcomes obtained from using the 12V Maxon motor shown in Table 6.3 are almost the same as the one from the 12V LEGO EV3 motor, shown in Table 6.1. This is also because the nominal torques are similar between the two motors, at approximately 25 mNm and 20 mNm, respectively.

The results above prompted the following investigation, using another motor with a significantly higher nominal torque than the previous two motor series EC 32 flat and the LEGO EV3 motor, i.e., Maxon series EC 60 flat, with its specification

given in Table 6.4. Note, the weight and diameter parameters of Maxon series EC 60 flat are also higher than the previous two motors.

Table 6.4: The motor specifications of the series EC 60 flat 100W motor from Maxon company (Maxongroup, 2020)

	Motor series EC 60 flat, 100W
1.Nominal voltage (V)	48
2.No load speed (rpm)	4,020
3.Nominal torque (mNm)	298
4.Nominal current (A)	2.61
5.Terminal resistance (Ω)	1.11
6.Torque constant (mNm/A)	113
7.Rotor inertia (gcm^2)	835
8. Weight (g)	355
9. Diameter (mm)	60

The results of stabilising system Model A with the motor series EC 60 flat are demonstrated in Table 6.5.

Table 6.5: The maximum initial pitch angle stabilisable from different controllers, with the 48V series EC 60 flat motor.

Controllers	Motor series EC 60 flat, 100W 48V (Nominal torque 298 mNm)
Hard constraint	
LQR	56.3°
LQG	56.3°
Freezing technique	90°
Freezing with EKF	90°
Soft constraint	
Freezing technique	88.0°
Freezing with EKF	86.2°

It can be seen that the maximum initial pitch angle achieved for stabilisation using all controllers increase significantly, as shown in Table 6.5, compared against the results from 48V series EC 32 motor shown in Table 6.3, because the motor nominal torque is now significantly higher. For instance, the maximum stabilisable pitch angles achieved using the standalone freezing technique (88°) and freezing technique with EKF (86.2°) with soft constraint shown in Table 6.5 rise by approximately 67°. Moreover, these angles using the two linear methods also go up by approximately 27° when using the new motor.

Noticeably, the maximum initial pitch angles which can be stabilised by both nonlinear control techniques with hard constrained inputs are much higher than what the linear methods could achieve, by approximately 33.7°. Furthermore, the freezing technique with and without EKF, using soft constrained inputs, provide larger initial pitch angles over the linear controllers, by approximately 31.7° and 29.9°, respectively.

This simulation outcome matches the theoretical analysis given in Section 6,3, that the nonlinear freezing control provides wider operation range than linear controls. Furthermore, the simulation without input saturation, described in Section 6.5.1, also presented that maximum initial pitch angles obtained from the nonlinear methods were higher than linear controls, similar to the test results with a higher power motor, shown in Table 6.5. These parameters of the new motor in Table 6.5 can be used to predict the simulation results; however, the practical experimentation needs to considerate the weight of new power supply or gear systems and the total mass of the robot system also requires to be updated.

6.5.3 Simulation Results on Model Uncertainty

In this subsection, robustness tests of the control designs for the self-balancing TWR models are performed in simulation. This is done by adding extra mass and increasing height of the TWR, to determine the impact of model uncertainties on system performances. The results are shown and analysed when comparing nonlinear freezing controls on Models A, B and AB against the LQG controller. Note that all controllers are subject to the same input saturation as described in section 6.5.2.

Normally, the mass of the robot's body and the total height of LEGO EV3 are 0.64kg and 0.21m, respectively, as shown in Chapter4: Table 4.1. However, in the next simulation, 10% mass and height increases are applied to represent modelling uncertainty, making the new (actual) mass and height to be 0.7 kg and 0.23m, respectively.

To begin with, Figure 6.57 displays the responses of three TWR models controlled by the nonlinear freezing technique and EKF from an initial pitch angle $x_3 = 12.5^{\circ}$, which is the maximum stabilisable angle for Models A and B, taking into account of model uncertainties. It can be seen that Model B generates most oscillations in its response graphs; in contrast, the smoothest curve signals are generated when controlling Model AB. Moreover, when mass and height of the TWR are increased to represent model uncertainty but are not reflected in the models, the maximum initial pitch angles which can be stabilised by freezing and EKF, on Models A and B, have decreased by 1.6°. Models A and B become

unstable, when the initial pitch angle is beyond$x_3 = 12.5^{\circ}$, as demonstrated in Figure 6.58..

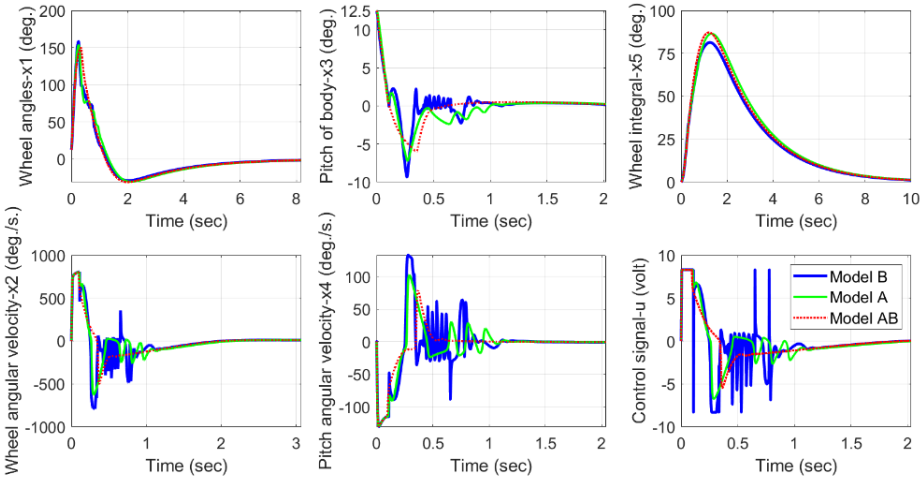


Figure 6.57: Stabilisation of TWR Models A, B and AB (with model uncertainties) by freezing controller with EKF from initial pitch angle x_3 =12.5°.

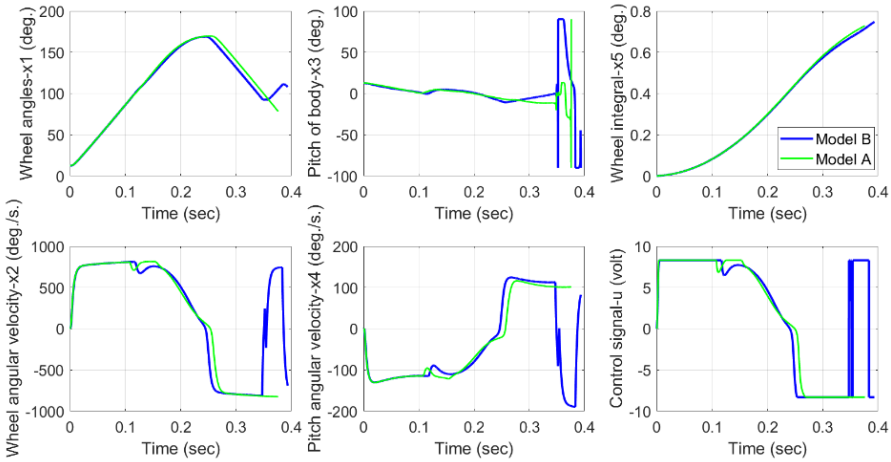


Figure 6.58: Unstable response of TWR Models A and B (with model uncertainties), by freezing control with EKF from initial pitch angle x_3 =12.6°.

Additionally, Model AB can achieve slightly larger initial pitch angle (13°) for stabilisation than Models A and B (12.6°). Figure 6.59 shows the stabilised system response from an initial pitch angle $x_3 = 13^{\circ}$, which represents a decrease from 14.3° achieved previously when weight and height parameters were correctly modelled. When increasing the initial pitch angle to 13.1°, Model AB displays unstable responses, as shown in Figure 6.60.

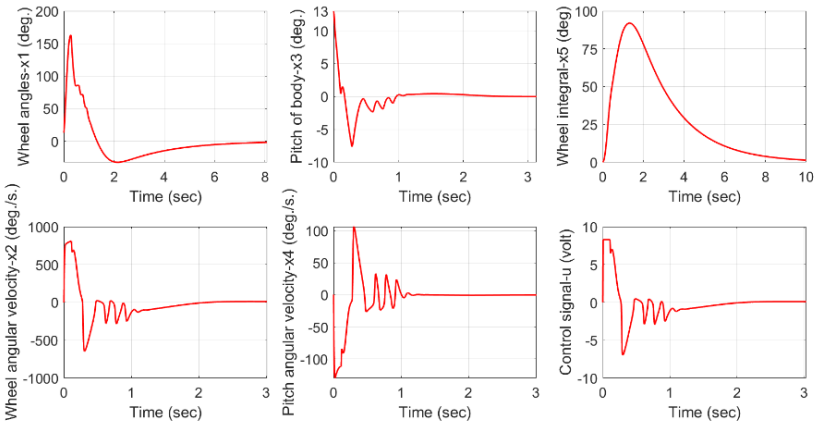


Figure 6.59: Stabilisation of TWR Model AB (with model uncertainties) by freezing controller with EKF from the initial pitch angle x_3 =13°.

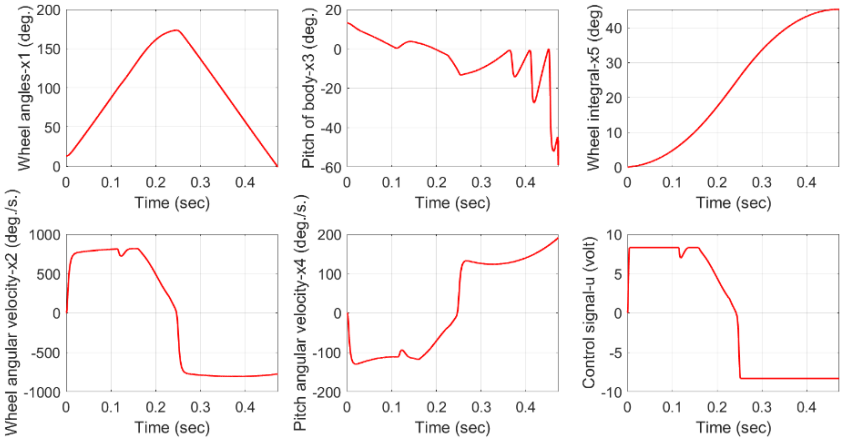


Figure 6.60: Unstable response of TWR Model AB (with model uncertainties), by freezing controller with EKF at initial pitch angle x_3 =13.1°.

In terms of linear control, the LQG with hard constrained input is applied to stabilise the system in Figure 6.61, presenting the maximum initial angle $x_3 = 19.7^{\circ}$ lower than before adding weight and height approximately 1.2°. Beyond this angle, the system is uncontrollable, as demonstrated in Figure 6.62. Significantly, the maximum initial pitch angle by the LQG is wider than the nonlinear controllers. This is because they apply the input constraint method to control the systems, which is different from the nonlinear controllers.

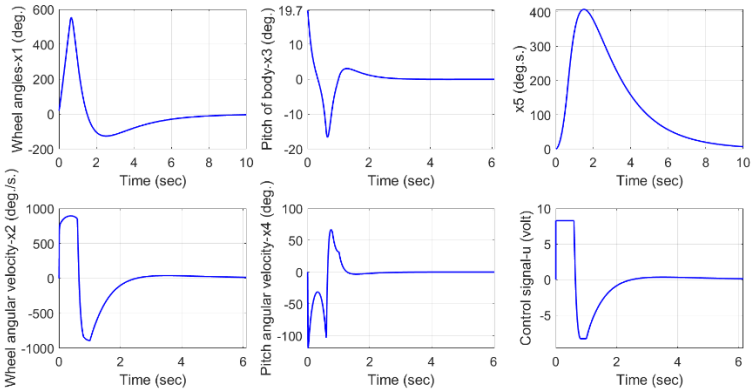


Figure 6.61: Stabilisation of LQG controller at the initial pitch angle $x_3 = 19.7°$

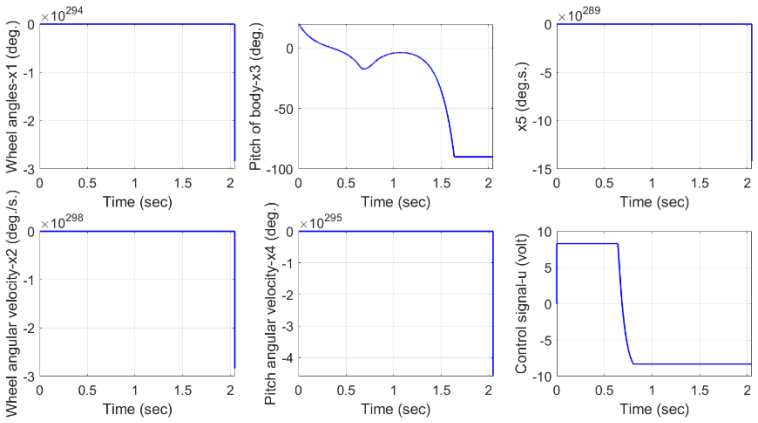


Figure 6.62: Uncontrollable system of LQG controller at the initial pitch angle $x_3 = 19.8°$

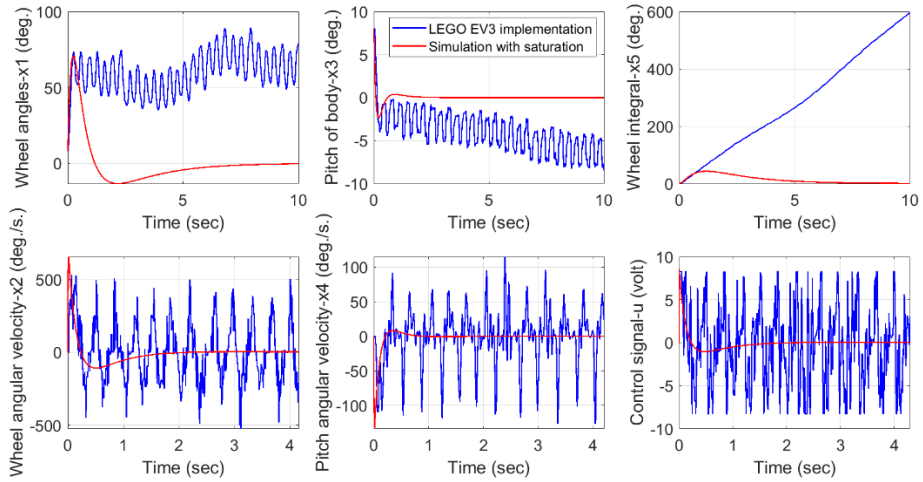
6.6 Experimental Results

In this section, the standard freezing control and freezing technique with extended Kalman filter (EKF) will be applied to the LEGO Mindstorms EV3 robot, using the Simulink block diagrams shown in Appendix B (Figures B3.1 and B3.3).

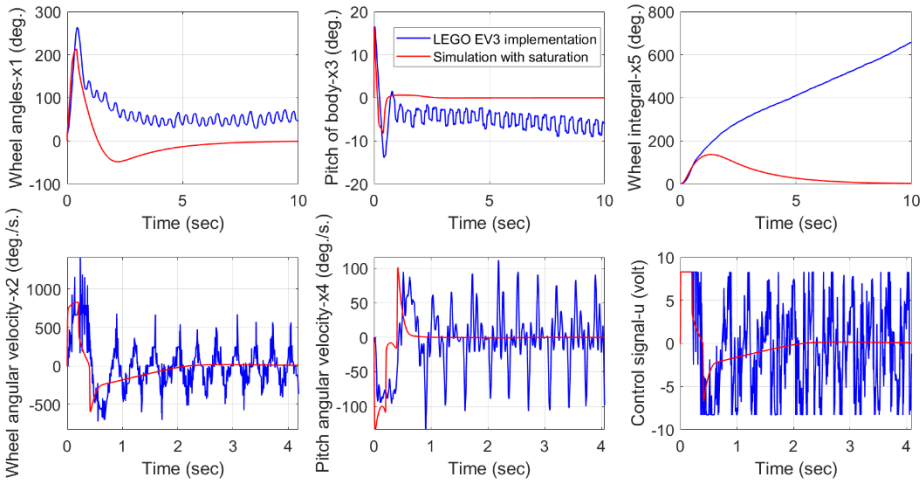
As described in Section 6.3, the feedback gains of nonlinear controllers were generated by solving algebraic Riccati equation at every time step. Unfortunately, the algebraic Riccati MATLAB function does not support code generation in Simulink programme; in this case, the algebraic Riccati function cannot run on the LEGO EV3 robot directly. Therefore, lookup tables of the algebraic Riccati equation have been used to store the feedback gains $\boldsymbol{K}$ and $\boldsymbol{K_f}$ for standard freezing technique and freezing technique with EKF, calculated in MATLAB. Because of the limitation of LEGO EV3's memory, the lookup tables were designed by restricting the state variable x_3 to be between $-20^{\circ}$and 20° , and state variable x_4 to be between $-130^{\circ}/s$ and $130^{\circ}/s$, covering the operation ranges achievable using the hardware specifications; moreover, the measurement precisions for the pitch angle x_3 and the pitch angular velocity x_4 were at nearest 1° and $5^{\circ}/s$, respectively (see details in Appendix B, Table B3.1).

6.6.1 Implementations from Varied Initial Pitch Angles

The results of hardware implementation by applying the freezing techniques with varied initial pitch angles are demonstrated in Figures 6.63-6.68.



(a)



(b)

Figure 6.63: The stabilising freezing control implemented on LEGO EV3 robot compared to simulation, at initial pitch angles (x_3): (a) 8° and (b) 16.8°

In Figure 6.63, first of all, the outcomes of balancing TWR system using the standard freezing technique with input constraint in simulation (red plot) and hardware implementation (blue plot) are presented, from the initial pitch angles $x_3 = 8^{\circ}$ and 16.8° (previous simulation results show the maximum stabilisation angle is at 16.8°). It can be seen that hardware signals of state variables x_1, x_3

and x_5 in Figure 6.63 (a) and (b) diverge from the centre, due to the issues of gyro sensor drifts. In particular, both pitch angles x_3 drift away significantly (by approximately -8°) after 10 seconds of simulations. This error also affects state variables x_1 and x_5 as they are continually compensating for the sensor drift problem. For example, state variables x_1 in both Figure 6.63 (a) and (b) deviate from the reference positions by approximately 60° and also in both figures, x_5 diverge from the reference angle by approximately 600° after the robot is stabilised in 10 seconds.

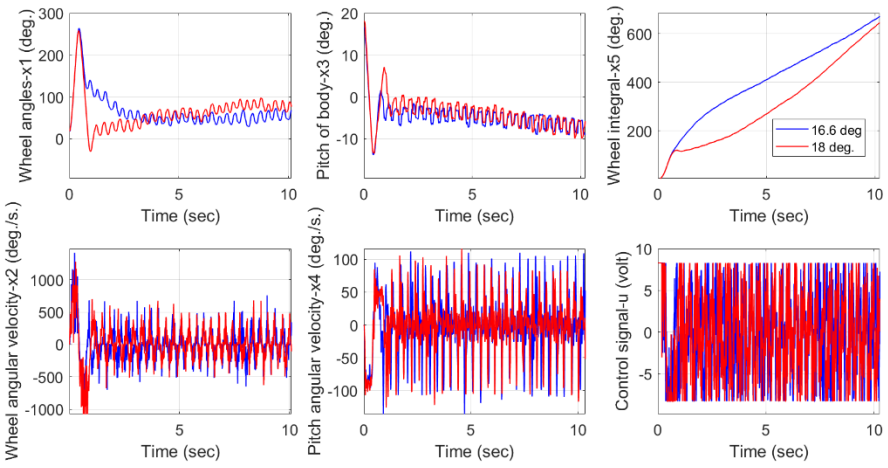


Figure 6.64: The stabilisation from different initial pitch angles (x_3) 16.6° and 18°, implemented on LEGO EV3 robot using freezing technique

When the initial pitch angle is increased to 18° in Figure 6.64, the sensor drift problem is still apparent. Furthermore, the maximum deviations from the two sets of responses are similar; for instance, both maximum overshoots of wheel angles x_1 are at approximately 250°, and both lowest drifts of the pitch angles x_3 are about -12°. Slightly more oscillations occur in the x_3 response when starting

from 18°, with an approximate 6° increase in maximum overshoot when compared against the x_3 response when starting from 16.6°. In addition, there are a few more undershoots in the x_1 plot for the initial pitch angle =18° case. These additional overshoots and undershoots occur because the LEGO EV3 robots requires longer distance and more time to balance itself when the initial angle increases.

Over this limitation angle, the hardware crashes because it resulted in an unstable system, as shown in Figure 6.65. Significantly, the freezing technique illustrates slightly more extensive operation range (18°) than the LQR method (by approximately 2°) when performed on the Lego two-wheel robot with input constraint. This is because the feedback gain of nonlinear freezing control is varied by state variables, in contrast to the linear method where the gain is constant.

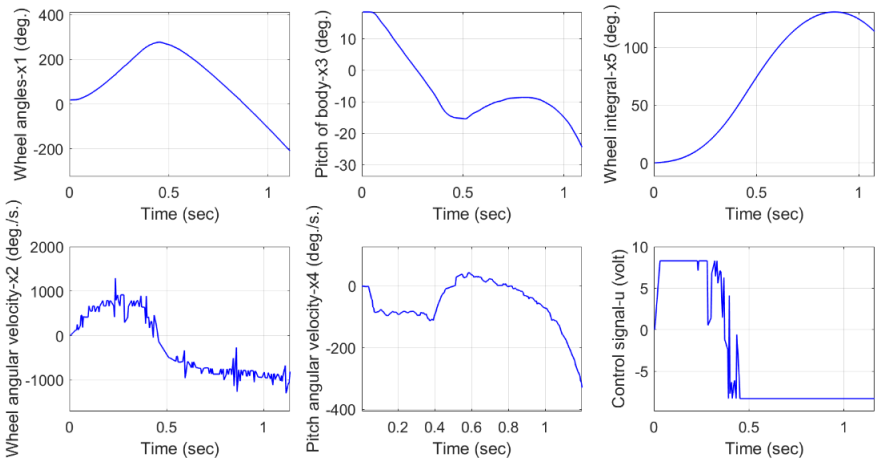
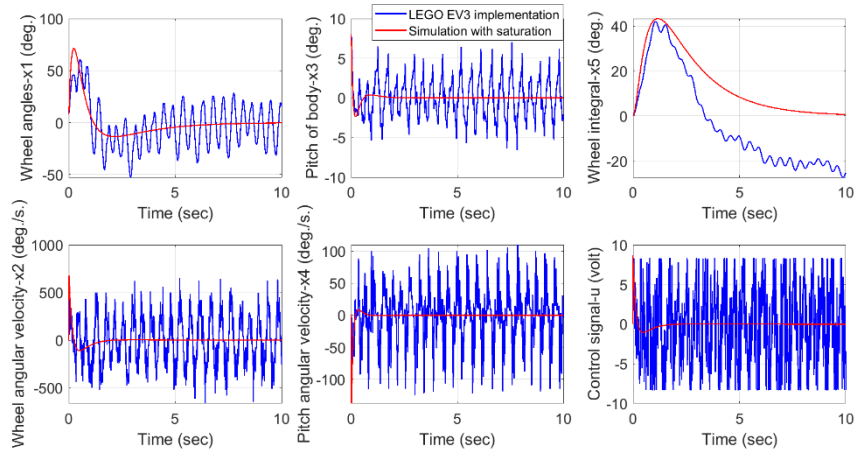
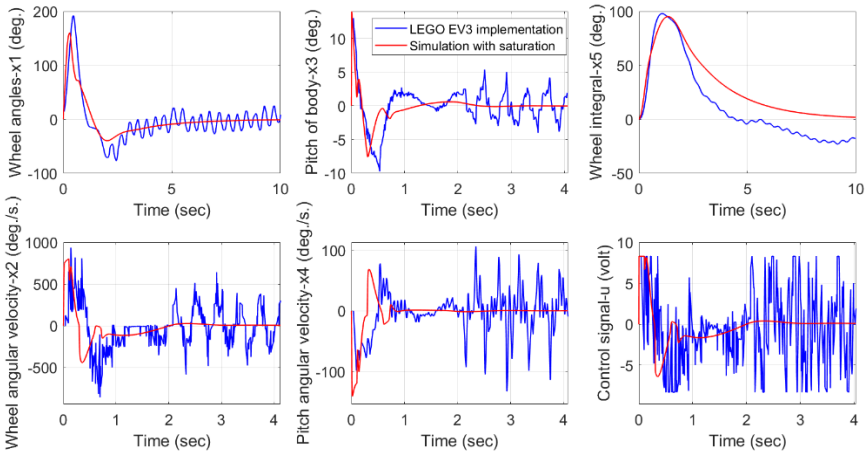


Figure 6.65: Unstable responses from the initial pitch angle18.5°, implemented on LEGO EV3 robot using freezing technique

In terms of combining freezing technique with EKF, Figure 6.66 compares the outcomes of simulation and LEGO EV3 robot when implementing at the initial pitch angles 8° and 14°, which is close to the maximum initial angle in simulation (14.1°).



(a)



(b)

Figure 6.66: Stabilising control of the LEGO EV3 robot compared to simulation at the initial pitch angles: (a) x_3 =8° and (b) x_3 =14°,using freezing technique with EKF

It can be seen from Figure 6.66 (a) and (b) that both hardware signals (blue curves) in the state variable x_3 graphs converge to the reference position in the same way as the simulation signals (red curves), because gyro sensor drifts are reduced by the application of the extended Kalman filter. In contrast, there are slight errors shown in both wheel integrals x_5 of the LEGO EV3 robot, which do not converge to the reference position. This is due to the averaged signals of wheel angles x_1 not being centred, caused by hardware vibrations. For instance, both initial pitch angles show that signals diverge from the centre by approximately -20°. Note, the maximum overshoots of state variables x_1 and x_5 in hardware implementation show almost the same results as in simulation.

In the case of increasing initial pitch angle, the maximum initial angle achievable by the freezing technique with EKF is 18°, as shown in Figure 6.67, which is similar to the standalone freezing technique (without EKF). Similarly, simulation results in Section 6.5.2 show that the freezing technique with EKF stabilised system equally well as the standalone freezing technique, under constraint conditions. Noticeably, the gyro sensor drift is now much reduced. Moreover, the maximum deviations of the two sets of responses shown in Figure 6.67 have slight differences when the initial pitch angle is increased from 14° to 18°. For instance, the maximum magnitude of the wheel angle x_1 grows by approximately 70° and the undershoot of x_3 expands by approximately 2°,when initial pitch angle increases.

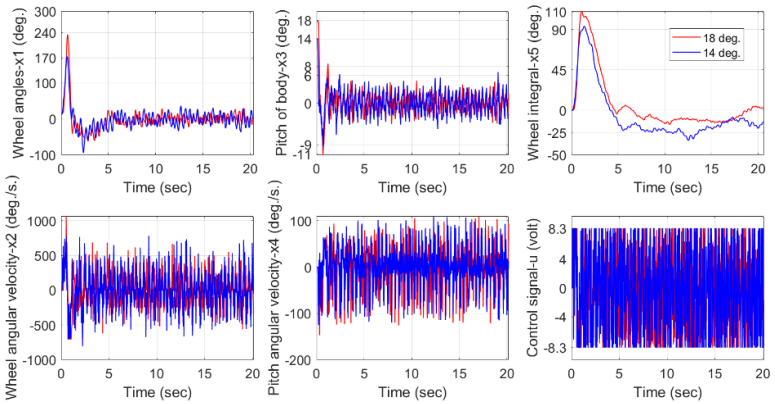


Figure 6.67: The stabilisation from different initial pitch angles (x_3) 14° and 18°, implemented on LEGO EV3 robot using freezing technique with EKF (sensor drift reduced)

At over the initial angle of 18°, the LEGO EV3 controlled by the freezing technique with EKF results in an unstable system and crashes and the responses are given in Figure 6.68.

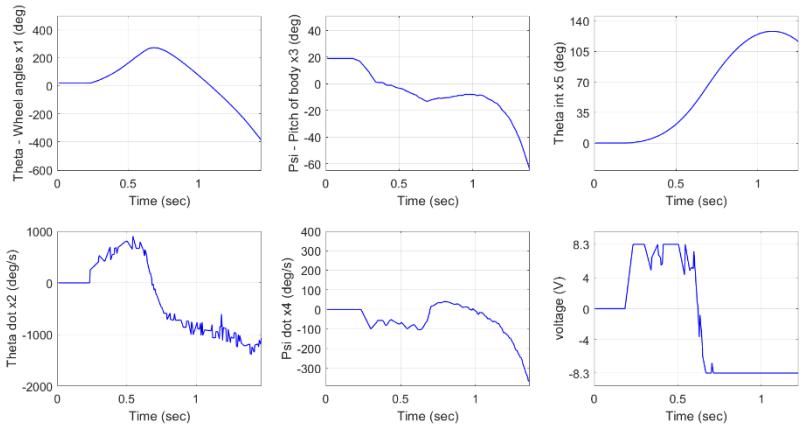


Figure 6.68: Unstable responses from the initial pitch angle 18.5°, implemented on LEGO EV3 robot using freezing technique with EKF

6.6.2 Alternative Models' Implementations

In this subsection, nonlinear feedback control of alternative models, including Models B and AB, are tested on a practical robot with varied initial pitch angles, compared against the primary model (Model A). Note that, in the case of nonlinear control, only freezing control combined with EKF is selected for implementation (to resolve the sensor drift issue), of which the simulation tests have been completed in section 6.5.2

Firstly, the freezing control and EKF gains obtained from using Model B, are implemented on the LEGO EV3 robot. The result in Figure 6.69 shows that the system is unstable, although the initial pitch angle is set as 0° or at the balancing point. This unstable system response matches the controllability test outcome, which demonstrated that the area near $x_3 = 0^{\circ}$ is not fully controllable when Model B is used to represent the TWR system.

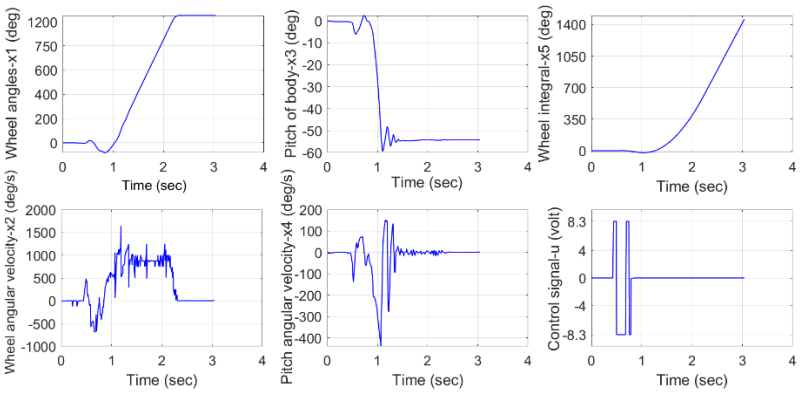


Figure 6.69: Unstable response from the initial pitch angle x_3= 0°, implemented on LEGO EV3 robot using freezing technique with EKF on Model B

Additionally, Figure 6.70 illustrates the maximum initial pitch angle achievable for Model AB, $x_3 = 20^{\circ}$, slightly larger than the one from Model A (18°)

and also wider than what the linear controllers could achieve, i.e., LQR (16°) and LQG (16°). Once goes over the initial angle of 20°, the TWR represented by Model AB becomes unstable and crashes, as shown in Figure 6.71.

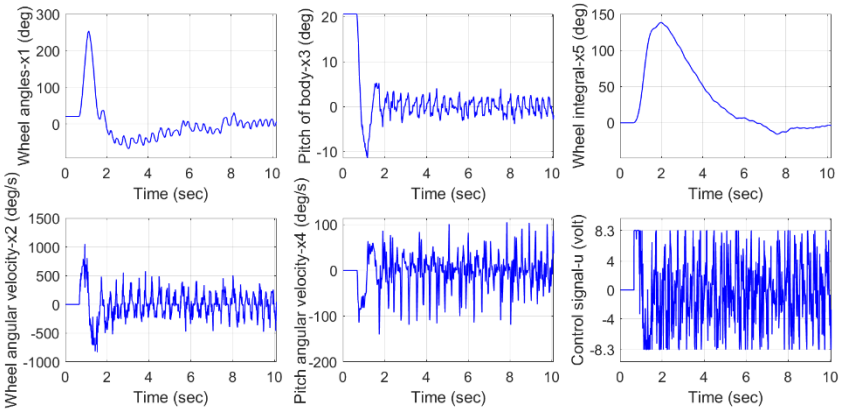


Figure 6.70: The stabilisation from initial pitch angle $x_3 = 20°$, implemented on LEGO EV3 robot using freezing technique with EKF on Model AB

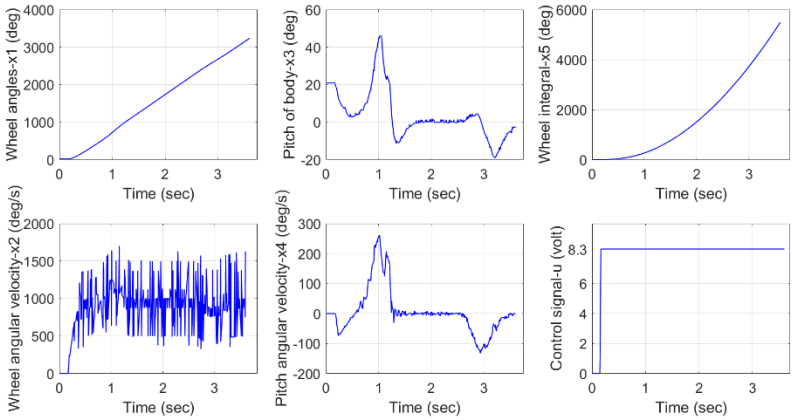


Figure 6.71: Unstable response from initial pitch angle $x_3 = 20.5°$, implemented on LEGO EV3 robot using freezing technique with EKF on Model AB

It can be seen that the balancing system using Model AB gains benefit from combining the models, utilising the strengths of Models A and B in different regions: i.e., the feedback gains of Model B are used to control wide pitch angles

and the feedback gains of Model A are used to balance the system at around the equilibrium (upright position). This is the reason that a mixed nonlinear model can provide more comprehensive operation range than otherwise.

6.6.3 Model Uncertainty Implementations

The robustness tests of implementing control designs on the LEGO TWR with model uncertainties (by increasing 10% of mass and height) are demonstrated in this subsection. Three balancing systems are investigated through a series of tests, including the applications of the LQG controller, and the freezing controller with EKF using feedback gains calculated using Models A and AB. Note that Model B is not considered in the robustness test as the feedback gains of the model could not stabilise the robot at equilibrium, as presented in Figure 6.69.

To begin with, the maximum initial pitch angle of balancing robot system using LQG controller under the specified model uncertainties, is 15°, and the responses are shown in Figure 6.72. This angle is lower than an accurately modelled system (without extra weight and height) using the LQG controller by approximately 2°. Moreover, the system becomes unstable when the initial pitch angle is increased over 15°, as shown in Figure 6.73.

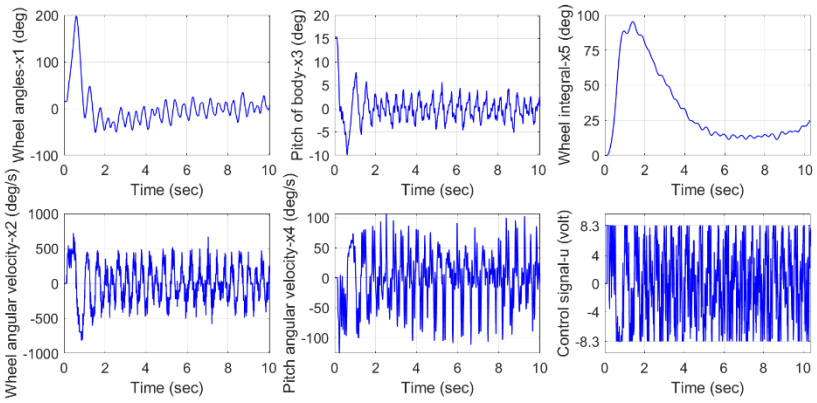


Figure 6.72: The stabilisation from initial pitch angle $x_3 = 15°$, implemented on LEGO EV3 robot using LQG controller with added mass and height

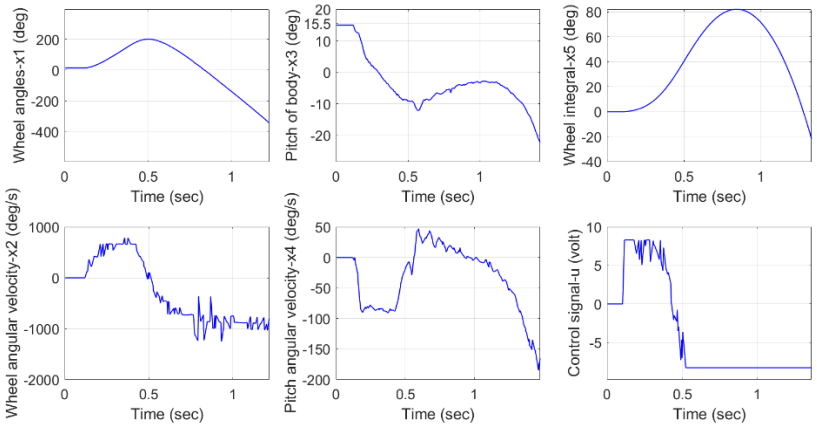


Figure 6.73: Unstable response from initial pitch angle $x_3 = 15.5°$, implemented on LEGO EV3 robot using LQG controller with added mass and height

Furthermore, the practical control of the LEGO robot with added mass and height, utilising feedback gains calculated from Model A, is presented in Figure 6.74. As the mass and height are both increased by 10%, the maximum initial pitch angle stabalisable is dropped from 18° to 16°. When goes over the initial pitch angle 16°, the LEGO EV3 robot crashes as the system becomes unstable, shown in Figure 6.75. However, this initial angle is slightly larger than when the

robot was control using LQG controller (shown in Figure 6.72), by approximately 1°.

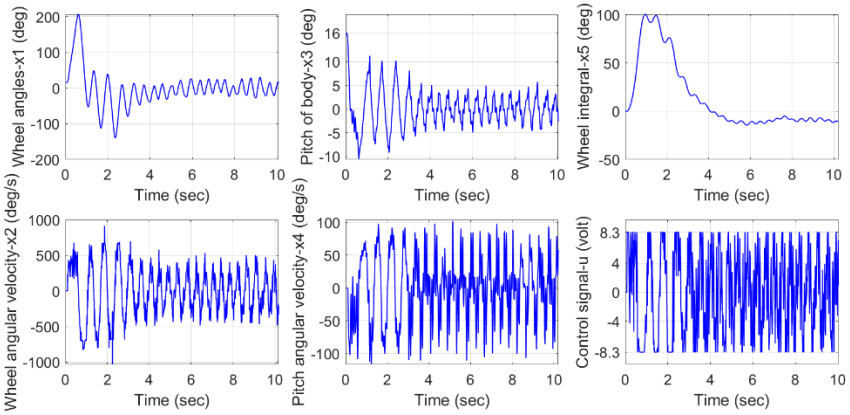


Figure 6.74: The stabilisation from initial pitch angle $x_3 = 16°$ implemented on LEGO EV3 robot using freezing control and EKF on Model A, with added mass and height

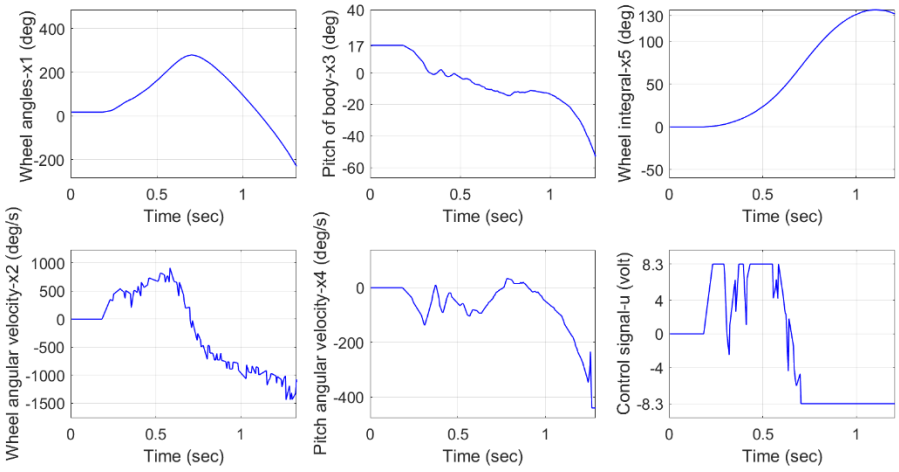


Figure 6.75: Unstable responses from initial pitch angle $x_3 = 17°$, implemented on LEGO EV3 robot using freezing control and EKF on Model A with added mass and height

Finally, the extra mass and height are added to a TWR controlled by freezing control and EKF with feedback gains obtained using Model AB. The stabilising system responses are demonstrated in Figure 6.76 with the maximum

initial pitch angle reaching 18°, representing a decrease of roughly 2°, from the unloaded condition. Significantly, this initial angle of 18° is the widest when comparing against outcomes obtained from other controllers, under the same model uncertainty condition. The maximum initial pitch angle achieved here is larger than using the freezing control and EKF with Model B by approximately 2°, and wider than the LQG controller by approximately 3°.

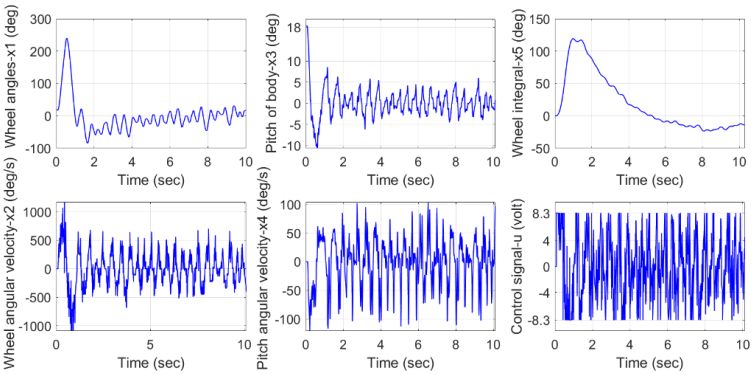


Figure 6.76: The stabilisation from initial pitch angle $x_3 = 18°$, implemented on LEGO EV3 robot using freezing control and EKF on Model AB, with added mass and height

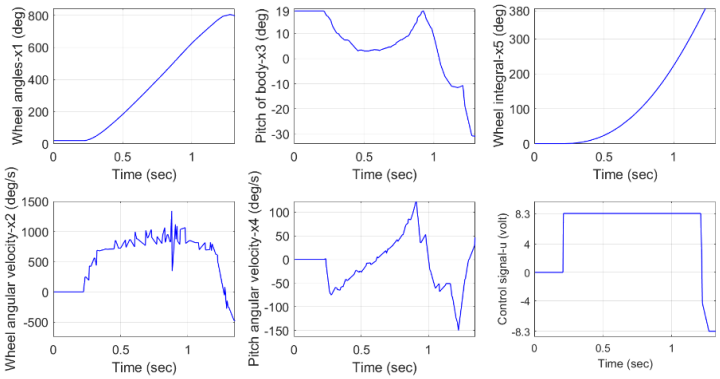


Figure 6.77: Unstable responses from initial pitch angle $x_3 = 19°$, implemented on LEGO EV3 robot using freezing control and EKF on Model AB, with added mass and height

The maximum initial pitch angle of a TWR which can be stabilised by different controllers are summarised in Table 6.6.

Table 6.6: The maximum initial pitch angles achieved using different controllers, in simulations and in practical implementations.

	Input unconstrained simulation	Hard input constrained at 8.3V, simulation	Soft input constrained at 8.3V, simulation	LEGO EV3 robot implementation
LQR	65.7°	20.9°	Incapable	16°
LQG	65.7°	20.9°	Incapable	16°
Freezing (Model A)	87.2°	20.9°	16.8°	18°
Freezing with EKF (Model A)	87.2°	20.9°	$14^{\circ}.1$	18°
Alternative models				
Freezing (Model B)	90°	20.8°		
Freezing with EKF (Model B)	90°	20.8°	14.1°	Not fully controllable
Freezing (Model AB)	90°	20.8°		
Freezing with EKF (Model AB)	90°	20.8°	14.3°	20°
Model uncertainty test: Adding 10% extra mass and height				
LQG		19.7°	Incapable	15°
Freezing with EKF (Model A)			12.5°	16°
Freezing with EKF (Model B)			12.5°	Not fully controllable
Freezing with EKF (Model AB)			13°	18°

It is evident from the summary table, Table 6.6, that nonlinear controllers provide larger stabilisation ranges and are more robust (when subject to model uncertainties) than linear controllers, in both simulation and practical

experimentations. Furthermore, nonlinear controls also present opportunities of combining multiple state-space models so their strengths in different controllable regions can be utilised. The combined model, Model AB, controlled by the nonlinear freezing method and EKF therefore demonstrated the widest operational range, as shown in this research.

6.7 Conclusion

In this chapter, the stabilisation problem of an inverted pendulum on a cart system was studied first, through controller designs (LQR, nonlinear freezing control and nonlinear iteration (also called LTV) scheme) and simulation verifications. The results demonstrated that both nonlinear control methods were capable of providing stabilising control on broader ranges of the pendulum rod angle than the optimal linear control method could. The largest angle stabilisable was obtained by the nonlinear freezing controller at 80.5°, followed by the LTV method at 61.3° and the smallest angle achievable amongst the three powerful techniques was given by the LQR controller, at 49.2°. This is because both the nonlinear freezing technique and the iteration scheme calculate and refresh feedback gains based on the time-varying state variables at the current time (at every new time step, in digital implementation). In contrast, the feedback gains of linear control are calculated based on the linearised model and fixed, assuming the system would operate around a small neighbourhood of the equilibrium. As illustrated by the simulation results in this chapter, the advanced nonlinear freezing control demonstrated higher capability of stabilise the inverted pendulum system than the LTV method. On top of that, practical considerations revealed

that the memory of LEGO EV3 was too limited to store large data sets needed for the LTV controller. Therefore, the nonlinear freezing method is selected as the control technique for the stabilisation of a two-wheeled robot, in simulation and practical implementation on a LEGO EV3 prototype in the later sections.

In the case of TWR simulation, both the standalone nonlinear freezing technique and the nonlinear freezing + EKF showed very similar results for all initial pitch angles tested under an unconstrained input condition. Moreover, the maximum initial pitch angles achievable from these two controllers were also nearly identical, at $\sim 87.2^{\circ}$. In particular, both nonlinear freezing controls demonstrated significantly wider operation ranges to balance the TWR system than the LQR (by approximately 21.5°) with the same parameter settings otherwise. Noticeably, a 2nd advantage of the nonlinear freezing method was shown in this chapter, that one could analyse controllabilities of different state-space models of the system and combine appropriate models to enlarge the overall controllability range. It was demonstrated that a mixed model, namely Model AB, controlled by the freezing + EKF method, could reach an initial pitch angle up to 90°, wider than the angle achievable from the primary model (Model A) and any other methods reported in the literature.

In terms of the TWR control simulation under input saturation, both freezing techniques presented similar results as the implementation; on the other hand, the limitation of initial pitch angles were slightly different, where the standalone freezing method demonstrated slightly higher initial pitch angle, by approximately 3°. Note, the maximum initial pitch angles of nonlinear standalone

freezing technique and freezing technique with EKF were identical to the LQR and LQG methods (20.9°), when a hard constraint of 8.3V control input by the LEGO EV3 power supply was applied. Moreover, the combination model (Model AB) was also investigated in simulations, and it presented slightly wider operation range over the primary model (Model A) and Model B using nonlinear freezing controllers with EKF, but less than the LQG method using hard input constraint by approximately 7°. Furthermore, the inclusion of other motors with high powers were studied in simulations. The results demonstrated both freezing techniques operating at extensive initial pitch angles (over 86°) under soft constrained input condition, which were much wider than the linear methods (by ~30°), when the maximum motor voltage was increased to 48V and the motor torque was increased to nearly 15 times higher (298 mNm) than LEGO EV3's motor (20 mNm). Significantly, the initial pitch angles using both nonlinear freezing techniques (with and without EKF) reached to 90°, when using the hard constrained input.

Additionally, robustness tests were conducted in simulation and on the LEGO robot. The TWR models with input saturation were simulated under model uncertainties by adding 10% mass and height to the robot in simulation. It was demonstrated that both the freezing control and EKF (applied to three models) and LQG, were capable of stabilising the system in the upright position, even though the initial pitch angles were reduced by approximately 1°.

With regard to hardware implementation, the LEGO EV3 robot was tested to show it could be stabilised from different initial pitch angles with suitable control

designs. Both freezing techniques (without and with EKF) demonstrated excellent control of the LEGO EV3 robot, satisfying the self-balancing requirement. In particular, the nonlinear controls with the feedback gains obtained from using the primary model presented slightly better capability than the linear methods, where the maximum initial pitch angles achieved are larger than the linear controllers (16°) by approximately 2°. When the mixed model (Model AB) is utilised, this difference of using the nonlinear freezing technique with EKF over LQG is increased to 4°. In the case of adding model uncertainties, the practical robot remains stabilised using the linear optimal and nonlinear optimal controllers; however, the maximum initial pitch angles from both methods dropped by approximately $1^{\circ} - 2^{\circ}$. Importantly, clear advantage of using the nonlinear freezing method for the mixed model was shown when the maximum stabilisable angle was 2° and 3° larger than using the LQG method and using freezing on the primary model, respectively.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The aim of the research presented in this thesis is to study, analyse and apply nonlinear optimal control techniques to stabilise two highly unstable systems, namely, the inverted pendulum on a cart and the two wheeled robot. This has been achieved through mathematical modelling, controller design, signal filtering estimation, simulation and practical implementation.

7.1.1 Mathematical Models

The inverted pendulum on a cart system is a classical benchmarking tool for testing capabilities and effectiveness of different control methods and shares similar dynamical structures with the two-wheeled robot. Therefore, the inverted pendulum model was investigated in this research first. The initial model in nonlinear differential equation form was obtained using the Lagrangian approach based on the system's potential energy and kinetic energy. Then, a nonlinear state-space matrix form of the inverted pendulum on a cart system was presented and the system matrix A heavily depended on the state vector, in particular, the pendulum angle and the angular velocity. This nonlinear state-space model was approximated as a linear one, assuming the pendulum angle was within a small neighbourhood of the equilibrium, i.e., near the upright position. Note that state-space representations are generally non-unique for any system which affect the controllability of the system. Therefore, different forms of the state-space matrices

of the inverted pendulum system were derived and the associated controllability analyses were performed.

Likewise, a nonlinear model consisting of a set of differential equations of the TWR were created using the Lagrangian method which led to nonlinear and (subsequently approximated) linear state-space models. A key difference between the controls for the IP system and the TWR was the physical control variables designed: for the IP system, this was a force on the cart in the horizontal axis, whilst for the TWR, they were motor voltages. Therefore, forces were converted to voltages in the system equations for the TWR. Additionally, a tracking design was combined with the TWR equations to form a higher order system, supporting the robot to track a pre-defined wheel displacement reference.

7.1.2 Linear Control Implementations

Controllability tests were utilised to analyse whether a linear state-space controller such as LQR would be appropriate for the control of the linearised IP and TWR systems first. The rank tests of the controllability matrices of both models demonstrated that the linearised systems were completely controllable. Furthermore, observability tests were also conducted on these linear models which examined whether the state variables could be estimated or observed using measurements made at the outputs. The results illustrated that both systems were completely state observable and therefore suitable for Kalman filter designs and implementations.

Next, simulations of an inverted pendulum on a cart controlled by LQR, starting from a range of initial pendulum angles, were performed. It was found that the maximum initial pitch angle of an IP system which the LQR method could stabilise without input saturation was 42.9°. In the case of a TWR model, the balancing system by LQR and LQG controllers demonstrated a similarly limited initial pitch angle of 65.7°, without input saturation. Moreover, when a hard input constraint of 8.3V was applied in simulation, the TWR model reached a maximum initial pitch angle at 20.3°, for both linear control methods (LQR and LQG).

Furthermore, a practical TWR prototype, built with LEGO Mindstorms EV3 kit, was used to verify simulation outcomes from both linear controllers. The implementations showed that the robot could be stabilised and maintained in the upright vertical position, from a maximum initial pitch angle of 16°, when using both linear control techniques. In particular, a gyro sensor drift issue experienced in the LQR control system was significantly reduced by the Kalman filter embedded in the LQG controller.

7.1.3 Nonlinear Control Implementations

Similar design procedures as the linear controls took place for the nonlinear controls. First, controllability and observability tests were conducted for nonlinear models of the IP system and TWR system. However, different to the linear cases, the controllability and observability matrices depended on the state variables of the nonlinear systems and therefore the rank test results vary. These were explored pictorially as 2D or 3D plots, showing fully controllable regions and others which are not fully controllable. This information was helpful in predicting

suitability of the nonlinear system to be stabilised by nonlinear state-space controllers.

Two advanced nonlinear control methods, namely the nonlinear optimal freezing technique and the nonlinear iteration scheme, were designed and applied to control the IP model without input saturation, in simulations. The results demonstrated that both nonlinear controllers achieved larger initial pendulum angle ranges than LQR and LQG. This is because the nonlinear methods calculated state-variable dependent feedback gains and applied them to the systems at appropriate points; on the other hand, the feedback gains of linear controls were fixed for the linearised model which assumed the operation of IP around a restricted area around the balancing position. Moreover, the stabilising system by nonlinear freezing control method presented a wider initial pitch angle than the nonlinear iteration scheme by approximately 20°, therefore, the freezing control was selected as the best performing controller for the study on the practical TWR next.

In the case of TWR models, excellent simulation results were obtained when balancing the system using both a standalone nonlinear freezing controller and a freezing control with EKF (with no input saturation), illustrating maximum initial pitch angles of the TWR's body, both at 87.2°. These were larger than both linear methods by approximately 21.5°. These results were supported by the controllability test, showing dynamic evolution paths of the state variables (starting from 87.2°) stayed within the fully controllable area. When the initial angles were over 87.2°, in contrast, the state variables travelled outside of the fully controllable regions. Next, input saturations were introduced to the TWR

control in simulation to take into account of physical limitations on motor voltages. Several voltage saturations between 8.3V to 48V were experimented, and simulation results demonstrated that the stand-alone freezing technique and freezing with EKF when operating with a motor at 48V voltage and 298 mNm nominal torque, achieved maximum stabilised initial pitch angles, at 88° and 86°, respectively. These pitch angles were larger than any other single closed-loop control technique could achieve, as reported in the literature, and were much wider than the ones obtained by linear optimal controllers shown in this thesis, by approximately 30°- 32°.

Additionally, the advantage of performing controllability tests was that different state-space models of the TWR system could be combined to create better controllability outcomes. For example, simulation results showed that the new mixed model, Model AB, of the TWR system, demonstrated the largest initial pitch angle, when compared against the primary model and Model B.

Experimentations using the TWR prototype constructed from LEGO EV3 demonstrated satisfying results for balancing the system in the upright position, using nonlinear freezing controls with and without EKF. Significantly, the largest initial pitch angle stabilisable was achieved as 20°, using the freezing control and EKF gains calculated from the mixed model AB, which was wider than the nonlinear freezing control from the primary model (Model A) by approximately 2°, and larger than the linear optimal methods by 4°. Furthermore, robustness tests (with introduced model uncertainties) were conducted on the LEGO robot with nonlinear freezing and EKF applied. The implementation illustrated that all controllers still stabilised the robot in the upright position when undergoing mass

and height increments and the nonlinear controllers still operated at larger initial pitch angles than the linear methods. All the above simulation and implementation outcomes demonstrated that the nonlinear optimal freezing control was a powerful technique, in achieving global control with excellent performance.

7.2 Recommendation for Future Work

In this thesis, research work focused on the applications of two advanced nonlinear control techniques on the stabilisations of the inverted pendulum and the two-wheeled robot systems in simulation and practical implementation. In future work, a practical robot prototype should be upgraded to one with a higher motor voltage and nominal toque, as well as being more flexible. For instance, the robot built from an Arduino microcontroller or NI myRIO Embedded Device may be a better choice, as it can apply 3rd party hardware to make the system more stable. In the experiments conducted during this research, a LEGO Mindstorms EV3 was selected to build the TWR prototype because of the availability of this resource; however, it was complicated to integrate other sensors or actuators into the LEGO EV3 controller. For example, there was merely a gyro sensor that could be used to calculate the pitch angle which caused the sensor drift problem. An accelerometer could not be added to the LEGO robot for calculating the pitch angle by the sensor fusion technique.

With regard to the actuator performance, the voltage range of motor should be increased in future work. It could be seen in the simulation of nonlinear systems in section 6.5.2 that a high voltage motor provided a wider operational range to stabilise the system.

Additionally, the flash memory storage of a LEGO EV3 controller had been limited to 5 MB for downloading a file, which was a small capacity for storing data; for instance, the look-up table of nonlinear freezing control technique needed to manage the limited data range to store inside the LEGO robot's memory. Therefore, the replacement robot should have more memory to increase the amount of control gain data stored and would therefore lead to better control outcomes.

For control strategy development, the simulation of nonlinear control, namely the iteration scheme based on the LQR controller, demonstrated a smaller operating range than the freezing technique; however, the iteration scheme can be combined with various other control techniques to provide feedback gains, which are likely to achieve a more comprehensive operation range or more stable system. For instance, the combination of a sliding mode controller and the iteration scheme was used to control velocity tracking of a hydraulic press model (Du et al.,2009); furthermore, the mixing of iteration scheme and pole placement technique was applied to control F-8 aircraft (Tomas-Rodriguez & Banks, 2013)

When the practical robot is upgraded in the future, the operational range of the stabilising system is expected to be more extensive, as the simulation resulted had so far indicated. Therefore, implementations on the upgraded robot will demonstrate the advantages of the nonlinear control systems against the linear methods more obviously, such as when the TWR robot is subject to external force disturbances or travelling on uneven surfaces, etc.

References

Ahmad, S., & Siddique, N. (2011). A Modular Fuzzy Control Approach for Two-Wheeled Wheelchair. *Journal of Intelligent & Robotic Systems, 64(3)*, 401–426.

Ahn, J., & Jung, S. (2014). Development of a two-wheel mobile manipulator: balancing and interaction control. *Robotica, 32(7)*, 1135–1152.

Alkamachi, A. (2020). Integrated SolidWorks and Simscape platform for the design and control of an inverted pendulum system. *Journal of Electrical Engineering, 71(2)*, 122–126.

Anderson, B., & Moore, J. (1989). *Optial control : linear quadratic methods.* New Jersey: Prentice Hall.

Aranda-Escolástico, G., Guinaldo, M., Santos, M., & Dormido, S. (2016). Control of a Chain Pendulum: A fuzzy logic approach. *International Journal of Computational Intelligence Systems, 9(2)*, 281–295.

Åström, F., & Furuta, K. (2000). Swinging up a pendulum by energy control. *Automatica (Oxford), 36(2)*, 287–295.

Banks, S. P., & Dinesh, K. (2000). Approximate Optimal Control and Stability of Nonlinear Finite- and Infinite-Dimensional Systems. *Annals of Operations Research, 98(1)*, 19–44.

Banks, S. P., & McCaffrey, D. (1998). Lie algebras, structure of nonlinear systems and chaotic motion. *International Journal of Bifurcation and Chaos, 8 (7),*, 1437–1462.

Banks, S., & Mhana, K. (1992). Optimal control and stabilization for nonlinear system. *IMA Journal of Mathematical Control and Information*, 179-196.

Batmani, Y., & Khaloozadeh, H. (2013). Optimal chemotherapy in cancer treatment: state dependent Riccati equation control and extended Kalman filter. *Optimal Control Applications and Methods*, 562–577.

Brunton, S., & Kutz, J. (2019). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control.* Cambridge: Cambridge University Press.

Burns, R. (2001). *Advanced Control Engineering.* Oxford: Butterworth-Heinemann.

Chang, C., & Liu, T. (2007). LQG Controller for Active Vibration Absorber in Optical Disk Drive. *IEEE Transactions on Magnetics, 43*(2), 799–801.

Chiasson, J. (2005). *Modeling and High Performance Control of Electric Machines.* New York: Wiley-IEEE Press.

Chu, T., & Chen, C. (2017). Design and Implementation of Model Predictive Control for a Gyroscopic Inverted Pendulum. *Applied Sciences, 7(12)*, 1272.

Çimen, T., & Banks, S. P. (2004a). Global optimal feedback control for general nonlinear systems with nonquadratic performance criteria. *Systems & Control Letters*, 327-346.

ÇImen, T., & Banks, S. P. (2004b). Nonlinear optimal tracking control with application to super-tankers for autopilot design. *Automatica, 40(11)*, 1845-1863.

Çimen, T., & Merttopçuoğlu, A. (2008). Asymptotically Optimal Nonlinear Filtering: Theory and Examples with Application to Target State Estimation. *IFAC Proceedings*, 8611–8617.

Cruz, D., García, S., & Bandala, M. (2016). ANN-Based Control of a Wheeled Inverted Pendulum System Using an Extended DBD Learning Algorithm. *International Journal of Advanced Robotic Systems, 13(3)*, 99.

da Silva, A., & Sup, F. (2017). A Robotic Walker Based on a Two-Wheeled Inverted Pendulum. *Journal of Intelligent and Robotic Systems, 86*(1), 17–34.

Du, C., Xu, X., Banks, S., & Wu, A. (2009). Control of nonlinear functional differential equations. *Nonlinear Analysis, 71(12)*, e1850–e1857.

Du, J., Gerdtman, C., Gharehbaghi, A., & Lindén, M. (2017). A signal processing algorithm for improving the performance of a gyroscopic head-borne computer mouse. *Biomedical Signal Processing and Control, 35*, 30-37.

Dutton, K., Thompson, S., & Barraclough, W. (1997). *The art of control engineering.* Addison-Wesley.

Ev3dev. (2020). Retrieved from ev3dev: https://www.ev3dev.org/

Fang, J. (2014). The LQR controller design of two-wheeled self-balancing robot based on the particle swarm optimization algorithm. *Mathematical Problems in Engineering, 2014*, 1-6.

Ford, J. (2011). *Lego Mindstormstm NXT 2.0 for Teens.* Boston: Course Technology.

Grasser, F., D'Arrigo, A., Colombi, S., & Rufer, A. (2002). JOE: a mobile, inverted pendulum. *IEEE Transactions on Industrial Electronics, 49(1)*, 107–114.

Hanselmann, H., & Engelke, A. (1988). LQG-control of a highly resonant disk drive head positioning actuator. *IEEE Transactions on Industrial Electronics, 35*(1), 100–104.

Harrison, R. F. (2003). Asymptotically optimal stabilising quadratic control of an inverted pendulum. *IEE Proceedings - Control Theory and Applications*.

Hassenplug, S. (2003). *Steve's LegWay*. Retrieved from Teamhassenplug: http://www.teamhassenplug.org/robots/legway/

Itik, M. (2016). Optimal control of nonlinear systems with input constraints using linear time varying approximations. *Nonlinear Analysis: Modelling and Control*, 21(3), 400–412.

Itik, M., Salamci, M. U., & Banks, S. P. (2009). Optimal control of drug therapy in cancer treatment. *Nonlinear Analysis: Theory, Methods & Applications, 71(12)*, e1473-e1486.

Jung, S., & Kim, S. S. (2008). Control Experiment of a Wheel-Driven Mobile Inverted Pendulum Using Neural Network. *IEEE Transactions on Control Systems Technology, 16(2)*, 297–303.

Kalman, R. (1960). A New Approach to Linear Filtering and Prediction Problems. *ASME Journal of Basic Engineering, 82*, 35-45.

Kalman, R., & Bucy, R. (1961). New Results in Linear Filtering and Prediction Theory. *ASME Journal of Basic Engineering*, 95-108.

Kharola, A., & Patil, P. (2017a). Dynamic stabilization of one wheel mobile robot (OWMR): a soft-computing approach. *Journal of Industrial and Production Engineering, 34(6)*, 477–485.

Kickstarter. (2021). *Jyrobike - Auto Balance Bicycle*. Retrieved from Kickstarter: https://www.kickstarter.com/projects/529668138/jyrobike-auto-balance-bicycle

Kim, H., & Jung, S. (2016). Control of a two-wheel robotic vehicle for personal transportation. *Robotica, 34(5)*, 1186-1208.

Kokkrathoke, S. (2018). *Design of Self-Balancing Two-Wheeled Robot Using Neural Networks Based Model Predictive Control [Unpublished master's degree dissertation].* United Kingdom: Sheffield Hallam University.

Kralev, S., Slavov , T., & Petkov, P. (2016). Design and experimental evaluation of robust controllers for a two-wheeled robot. *International Journal of Control, 89(11)*, 2201–2226.

Lee, C., Dao, N., Jang, S., Kim, D., Kim, Y., & Cho, S. (2016). Gyro Drift Correction for An Indirect Kalman Filter Based Sensor Fusion Driver. *Sensors, 16*(6), 864.

LEGO. (1999). *Robotic Invention System 1.5*. Retrieved from LEGO: www.lego.com/cdn/product-assets/product.bi.core.pdf/4129418.pdf

LEGO. (2020a). *LEGO® Mindstorms*. Retrieved from Lego: https://www.lego.com/en-gb/themes/mindstorms/about

LEGO. (2020b). *LEGO*. Retrieved from MINDSTORMS: https://www.lego.com/en-gb/themes/mindstorms

LEGO. (2020c). *LEGO® MINDSTORMS® Education EV3 Developer Kits*. Retrieved from Education.Lego: https://education.lego.com/en-gb/support/mindstorms-ev3/developer-kits

LEGO. (2021). *LEGO MINDSTORMS Education EV3*. Retrieved from education.lego.com: https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/building-instructions

LeJOS. (2009). Retrieved from LeJOS: http://www.lejos.org/index.php

Lewis, F., Xie, L., & Popa, D. (2007). *Optimal and Robust Estimation With an Introduction to Stochastic Control.* Florida: CRC Press.

Mahmoud, M., & Nasir, M. (2017). Robust Control Design of Wheeled Inverted Pendulum Assistant Robot. *IEEE/CAA Journal of Automatica Sinica, 4(4)*, 628–638.

Manousiouthakis, V., & Chmielewski, D. (2002). On constrained infinite-time nonlinear optimal control. *Chemical Engineering Science*, 105–114.

Masakatsu, M. (2015). Development of rotary inverted pendulum using LEGO MINDSTORMS. *Measurement and Control*, 192–195.

Mathworks. (2020). *Mathworks*. Retrieved from https://uk.mathworks.com/

Maxongroup. (2020, April). *EC 32 flat 15W*. Retrieved from Maxongroup: https://www.maxongroup.co.uk/medias/sys_master/root/8841185394718/EN-282.pdf

Maxongroup. (2020, April). *EC 60 flat 100W*. Retrieved from Maxongroup: https://www.maxongroup.co.uk/medias/sys_master/root/8841185656862/EN-294.pdf

MSCsoftware. (2020). *About: MSC Software Simulating Reality, Delivering Certainty*. Retrieved from MSCsoftware: https://www.mscsoftware.com/About-MSC-Software

Nemra, A., & Aouf, N. (2010). Robust INS/GPS Sensor Fusion for UAV Localization Using SDRE Nonlinear Filtering. *IEEE Sensors Journal*, 789–798.

Netgear. (2020). *WNA1100 - N150 Wireless USB Adapter*. Retrieved from Netgear: https://www.netgear.com/support/product/WNA1100.aspx

Ogata, K., & Yang, Y. (1970). *Modern control engineering.* Boston: Pearson.

Park, M., & Chwa, D. (2009). Swing-Up and Stabilization Control of Inverted-Pendulum Systems via Coupled Sliding-Mode Control Method. *IEEE Transactions on Industrial Electronics (1982), 56(9)*, 3541–3555.

Park, S., & Yi, S. (2020). Active Balancing Control for Unmanned Bicycle Using Scissored-pair Control Moment Gyroscope. International Journal of Control. *Automation, and Systems, 18(1)*, 217–224.

Prasad, L., Tyagi, B., & Gupta, H. (2014). Optimal Control of Nonlinear Inverted Pendulum System Using PID Controller and LQR: Performance Analysis Without and With Disturbance Input. *International Journal of Automation and Computing, 11*(6), 661-670.

Quanser. (2020). *ROTARY INVERTED PENDULUM*. Retrieved from Quanser: https://www.quanser.com/products/rotary-inverted-pendulum/

Roslovets, P. (2020). *Gyroboy - self-balancing two-wheel robot based on Lego EV3*. Retrieved from Mathworks: https://www.mathworks.com/matlabcentral/fileexchange/60322-gyroboy-self-balancing-two-wheel-robot-based-on-lego-ev3

Segway. (2021). Retrieved from Segway: https://uk-en.segway.com/products/segway-i2-se

Seman, P., Rohal'-Ilkiv, B., Juh´as, M., & Salaj. (2013). Swinging up the Furuta Pendulum and its Stabilization Via Model Predictive Control. *Journal of Electrical Engineering, 64(3)*, 152–158.

Simon, D. (2006). *Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches.* New Jersey: John Wiey & Sons.

Takagi, T., & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-15(1),* , 116–132.

Tanaka, N. (2020). Posture stability control of a small inverted pendulum robot in trajectory tracking using a control moment gyro. *Advanced Robotics, 34(9)*, 610–620.

Tao, C., Taur, J., Hsieh, T. W., & Tsai, C. (2008). Design of a Fuzzy Controller With Fuzzy Swing-Up and Parallel Distributed Pole Assignment Schemes for an Inverted Pendulum and Cart System. *IEEE Transactions on Control Systems Technology, 16(6)*, 1277.

Tomás-Rodríguez, M., & Banks, S. (2010). *Linear, time-varying approximations to nonlinear dynamical systems: with applications in control and optimization.* Berlin: Springer Science & Business Media.

Tomas-Rodriguez, M., & Banks, S. (2013). An iterative approach to eigenvalue assignment for nonlinear systems. *International Journal of Control, 86(5)*, 883–892.

Tsai, C., Huang, H., & Lin, S. (2010). Adaptive Neural Network Control of a Self-Balancing Two-Wheeled Scooter. *IEEE Transactions on Industrial Electronics, 57(4)*, 1420–1428.

Van De Vegte, J. (1990). In *Feedback control systems* (p. 358). Prentice-Hall.

Xu, X., Zhang, H., & Carbone, G. (2017). Chapter10: Case studies on nonlinear control theory of the inverted pendulum. In *Inverted pendulum: from theory to new innovations in control and robotics* (pp. 225-262). IET.

Yamamoto, Y. (2009, August). *NXTway-GS (Self-Balancing Two-Wheeled Robot) Controller Design)*. Retrieved from Mathworks: https://uk.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/19147/versions/6/download/zip

Zabihifar, S., Yushchenko, A., & Navvabi, H. (2020). Robust control based on adaptive neural network for Rotary inverted pendulum with oscillation compensation. *Neural Computing & Applications, 32*, 14667–14679.

Zheng, J., Banks, S. P., & Alleyne, H. (2005). Optimal attitude control for three-axis stabilized flexible spacecraft. *Acta astronautica, 56(5)*, 519-528.

Zhu, Q. (2021). Complete model-free sliding mode control (CMFSMC). *Scientific Reports*, *11*(1).

Appendix A

MATLAB Codes

Chapter 5: Linear Control Strategies

Appendix A.5.1: Linear quadratic regulator function in MATLAB

The algebraic matrix Riccati equation:

$$\mathbf{0} = \boldsymbol{PA} + \boldsymbol{A'P} - \boldsymbol{PBR^{-1}P} + \boldsymbol{Q}. \tag{5.5}$$

Equation (5.5) can be solved by applying the linear quadratic regulator function in MATLAB function commanded by:

$$[\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{E}] = \boldsymbol{lqr}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R})$$

where the $\boldsymbol{E}$ is eigenvalue vector, the matrix $\boldsymbol{K}$ is optimal feedback gain and the matrix $\boldsymbol{P}$ is the positive definite solution of the algebraic matrix Riccati equation.

Appendix A.5.2: Rank of the controllability matrix command

To implement the rank of the controllability matrix, substitute $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices by into Eq. (5.8):

$$\boldsymbol{\mathcal{C}} = [\boldsymbol{B}\ \boldsymbol{AB}\ \boldsymbol{A^2B}\ \boldsymbol{A^3B}], \tag{5.8}$$

and then apply MATLAB command,

$$Result = rank(\boldsymbol{\mathcal{C}}) \text{ or } Result = rank(ctrb(\boldsymbol{A}, \boldsymbol{B})).$$

The result given is Rank($\mathcal{C}$) = 4, equal to the number of rows in matrix $\boldsymbol{B}$; therefore, the system is said to be completely state controllable. Similarly, the rank test of Eq.(5.9) can use the same MATLAB command.

Appendix A.5.3: MATLAB codes of an inverted pendulum on a cart system using LQR controller

Script file: LQR_single_pendulum_4s.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Linear control (LQR) of an inverted pendulum on a cart system
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;

m1=2; %Mass of the cart (kg)
m2=0.1; %mass of the pendulum (kg)
r=0.5; %the rod length (m)
g=9.8; %acceleration due to gravity (m^2/s)


x1(1)=0; %set initial cart displacement to be 0 (m)
x2(1)=0; %set initial cart velocity to be 0 (m/s)
x3(1)=40*pi/180; %set initial pendulum angle to be pi (rad)
x4(1)=0; %set initial pendulum angular velocity to be 0 (rad/s)

u(1)=0; % Control input

Ts = 0.001;   % step size
Duration=10;  % 10 sec
t=Duration*(1/Ts);
Time = 0:Ts:t*Ts;  % the range of x-axis

%set Q and R matrices
Q=[1, 0, 0,  0;
   0, 1, 0,  0;
   0, 0, 100, 0;
   0, 0, 0, 10];
R=0.01;


for i=1:t
x=[x1(i); x2(i); x3(i); x4(i)];
%define and update the x vector


%%% Nonlinear model from Xu's book chapter  %%%
x_3=x(3); x_4=x(4);

AN=[0, 1, 0, 0;
```

```
0, 0,-m2*g*sin(x_3)*cos(x_3)/((m1+m2*(sin(x_3))^2)*x_3),...
m2*r*x_4*sin(x_3)/(m1+m2*(sin(x_3)^2));
0, 0, 0, 1;
0, 0,(m1+m2)*g*sin(x_3)/(r*(m1+m2*(sin(x_3))^2)*x_3),...
-m2*r*x_4*sin(x_3)*cos(x_3)/(r*(m1+m2*(sin(x_3))^2))];

BN=[0; 1/(m1+m2*(sin(x_3))^2); 0 ; -cos(x_3)/(r*(m1+m2*(sin(x_3))^2))];


%%% Linear model from Xu's book chapter %%%
AL=[0, 1, 0, 0;
0, 0,-m2*g/m1, 0;
0, 0, 0, 1;
0, 0,(m1+m2)*g/(r*m1), 0];

BL=[0; 1/m1; 0 ; -1/(r*m1)];


[~,P,~]=lqr(AL,BL,Q,R);
%use the MATLAB 'lqr' function to solve Riccati equation and
%work out P

u_out=(-inv(R)*BL'*P)*x;

fx=(AN-BN*(1/R)*BN'*P)*x;
%calculate the function output 'fx' based on values of A, B, P
%and x.
 x = x + Ts * fx;    % Euler method

     %%% Limit the pitch angle between -90 to 90 deg.
    if x(3) > 90*pi/180
        x(3) =90*pi/180;
    end
     if x(3) < -90*pi/180
        x(3) =-90*pi/180;
     end


x1(i+1)=x(1);
x2(i+1)=x(2);
x3(i+1)=x(3);
x4(i+1)=x(4);
%Reset the x1, x2, x3 & x4 variables to new values and get ready
%for the next iteration.

u(i+1) = u_out;
      %%% print out %%%%
    Cal_percent = i*100/(t);
    if mod(Cal_percent , 10) == 0
        fprintf('Calculating  %f percent ...\n',Cal_percent)
    end



end


figure('Name','LQR Control');
Fn = 12; % font size

subplot(2,3,1);
p1=plot(Time,x1); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Cart displacement x1
(m)','FontSize', Fn);
set(gca,'FontSize',Fn);
```

```
subplot(2,3,4);
p2=plot(Time,x2); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Cart velocity x2
(m/s)','FontSize', Fn);
set(gca,'FontSize',Fn);

subplot(2,3,2);
p3=plot(Time,x3*180/pi); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Pendulum angle x3
(deg)','FontSize', Fn);
set(gca,'FontSize',Fn);

subplot(2,3,5);
p4=plot(Time,x4*180/pi); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Pendulum angular velocity x4
(deg/s)','FontSize', Fn);
set(gca,'FontSize',Fn);

subplot(2,3,3);
p5=plot(Time,u); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Control signal-u (N)','FontSize',
Fn);
set(gca,'FontSize',Fn);

Size=1.1;
p1.LineWidth = Size;
p2.LineWidth = Size;
p3.LineWidth = Size;
p4.LineWidth = Size;
p5.LineWidth = Size;
```

Appendix A.5.4:MATLAB codes of a TWR system using LQR controller

Script file: Gyroboy_5s_LQR_10_2021.m

```
%%%% Linear Control (LQR) for LEGO EV3 Robot

%%%%%%  Functions programme needed %%%
%
% evalrhs_gyroboy5s_LQR();      % Generating K1-K4, fx and Kf
%
%      Inside two functions
%
%           Gyroboy_Nonlinear_Model_5s();
%           Gyroboy_Nonlinear_Model_4s();
%           Maxon Motor parameters etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;

%%% Set initial x1-x5
%%% Always set x1=x3

x1(1)=14.1*pi/180;      %set Theta - Average of wheel angles (deg)
x2(1)=0;                %set Theta_DOT (deg/s)
x3(1)=14.1*pi/180; %set Psi - Pitch angle of body (deg) - CONTROL to ZERO
x4(1)=0*pi/180;         %set Psi_DOT (deg/s)
x5(1)=0;                %set Theta integral

 %%% Set Model No.1 for linear control method %%%%
```

```
  Model=1;


u_in1(1)=0;  % Left  Motor Voltage
u_in2(1)=0;  % Right Motor Voltage
u_feedback =[0;0]; % u_feedback = Kx = (K_LQR) x (X1-X4)


x1_err = 0;      % x1 error for tracking sys
x1_ref = 0;      % x1 reference
x1_err_int(1)=0; % x1 error integral

alphaa=1;   % Motor Variable
betaa=0;    % Motor Variable

Ts=0.0001; %time step length

Duration=10;       % time sec
t=Duration*(1/Ts); % time step in programming
Time = 0:Ts:t*Ts;   % Create real time step for plotting

for i=1:t
  u      =[u_in1(i); u_in2(i)];
  x_5s    =[x1(i); x2(i); x3(i); x4(i); x5(i)];   % whole system
  x_4s    =[x1(i); x2(i); x3(i); x4(i)];          % x1-x4 state feedback


  %%%%%% Programming Diagram %%%%%

%
%                    u_x1  +     u                              y=x1-x4            y=x1
%  X1ref->-o-->-Int--K5-->-->o---------------+-------| Plant |-->----------------+---C------+
%          ^-                ^ -                                                 |          |
%          |                 | u_feedback                                        |          |
%          |                 |                                                   |
%          x1                |                                                   |
%                            +--------------------<--K14-- ----------------- ---+
%                                                      x1-x4
%
% %%%%%%%%%%%%%%%

    %%% x1 error for tracking sys
    x1_err(i+1) = x1_ref - x1(i);

      %%%%% X1 error Integral  %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

     %%% x1 error integral
    x1_err_int(i+1) = x1_err_int(i) + x1_err(i)*Ts + 0.5*(x1_err(i+1)-x1_err(i))*Ts ;  %
[h1 x L]+[ 0.5 x(h2-h1) x L ] + old

    %%% Cal. K5 for x5 only (integral x1)

    %K_5 aleady has been calculated as gain fixed.
    %They are same value at [-0.5,-0.5] So,use this value for reducing calculation
     K_5=[-0.5000; -0.5000];

    %%% u=kx,  Motor Voltage of integral x1
    u_x1 = K_5 * x1_err_int(i+1);

    %%% The final voltages used to control robot motors
    u = u_x1 - u_feedback;


  %%% Hard Saturation  %%%
%    %%% Uncomment this part for using motor voltage hard saturation
%     Vmax = 8.3;
%     %Vmax = 36;
%     %Vmax = 48;
%              if u(1) > Vmax
%                 u(1) = Vmax;
%                 u(2) = Vmax;
%              end
%               if u(1) < -Vmax
%                 u(1) = -Vmax;
%                 u(2) = -Vmax;
%               end
    %%%%%%%
```

```
    u_in1(i+1)=u(1);     % update u1
    u_in2(i+1)=u(2);     % update u2


    %%% calculate the new 'x' vector using a 4th order
    %%% Euler integration method
    %%% fx = Ax + Bu;
    [fx,K_14] = evalrhs_gyroboy5s_LQR(x_4s,u, Model); %[fx,K_14,alpha,beta]
    x_4s = x_4s + Ts * fx; % Euler


      %%% Limit the robot pitch angle between -90 to 90 deg.
    if x_4s(3) > 90*pi/180
        x_4s(3) =90*pi/180;
    end

     if x_4s(3) < -90*pi/180
        x_4s(3) =-90*pi/180;
     end

     %%% Update x1-x4
    x1(i+1)=x_4s(1);
    x2(i+1)=x_4s(2);
    x3(i+1)=x_4s(3);
    x4(i+1)=x_4s(4);


      %%% Generate real x1 integral (x5)
      %%%%% X5 = Integral of X1 %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

      x5(i+1) = x5(i) + x1(i)*Ts + 0.5* (x1(i+1)-x1(i))* Ts;


      %%% x1-x4 motor voltage feedback %%%%
      u_feedback = K_14*[x_4s(1); x_4s(2); x_4s(3); x_4s(4)];


    %%% Command print for waiting %%%
    Cal_percent = i*100/t;

    if mod(Cal_percent , 10) == 0
       %fprintf('Calculating  %f percent ...\n',Cal_percent)
       fprintf('Calculating  %3.2f percent,  ',Cal_percent)
       fprintf('x3= %3.2f deg,  ',x3(i+1)*180/pi)
       fprintf('x4= %3.2f deg/s,  ',x4(i+1)*180/pi)
       fprintf('t= %3.2f sec...\n',Ts*i)
    end

    %Reset the x1, x2, x3, x4, x5 & x6 variables to new values and get ready for the
next iteration.
end


%%%%%%%%%%
figure('Name','LQR Controller of a TWR system');
Fn = 14; % font size

subplot(2,3,1);
p1=plot(Time,x1*180/pi,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Wheel angles-x1 (deg.)','FontSize', Fn);
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2=plot(Time,x2*180/pi,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Wheel angular velocity-x2
(deg./s.)','FontSize', Fn);
set(gca,'FontSize', Fn);

subplot(2,3,2);
p3=plot(Time,x3*180/pi,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Pitch of body-x3 (deg.)','FontSize', Fn);
set(gca,'FontSize', Fn);

 subplot(2,3,5);
 p4=plot(Time,x4*180/pi,'b'); grid;
```

```
 xlabel('Time (sec)','FontSize', Fn); ylabel('Pitch angular velocity-x4
(deg./s.)','FontSize', Fn);
 set(gca,'FontSize', Fn);

 subplot(2,3,3);
 p5=plot(Time,x5*180/pi,'b'); grid;
 xlabel('Time (sec)','FontSize', Fn); ylabel('Wheel integral-x5 (deg.)','FontSize', Fn);
 set(gca,'FontSize', Fn);

subplot(2,3,6);
p6=plot(Time,u_in1,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Control signal-u (volt)','FontSize', Fn);
set(gca,'FontSize', Fn);

LW=1.2;
p1(1).LineWidth = LW;
p2(1).LineWidth = LW;
p3(1).LineWidth = LW;
p4(1).LineWidth =LW;
p5(1).LineWidth = LW;
p6(1).LineWidth =LW;
```

Function file: evalrhs_gyroboy5s_LQR.m

```
function [fx,K_14] = evalrhs_gyroboy5s_LQR( x ,u, Model)

    %%%%%%  Functions programme needed %%%
    % Gyroboy_Nonlinear_Model_4s();  % Generating fx, x1-x4 without x5
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT


  %%% K14 parameters already have been calculated as the linear control gain is fixed
  %%% K14 = k1,k2,k3 and k4

  %%% Select one K by uncomment the coding

   %%%%% Fix K LEGO EV3 %%%
   K_14=[-1.3908   -1.4487  -59.8476   -7.1681     ;
         -1.3908   -1.4487  -59.8476   -7.1681    ];


%    %Maxon DC motor EC flat 60   48V 100w  2020 ***
%      K_14= [ -1.2606   -0.7236  -34.5003   -4.2534
%              -1.2606   -0.7236  -34.5003   -4.2534];


    %%% 4-states Model parameters
    %%% Real model parameters
    [A4,B4,~,alpha,beta]=Gyroboy_Nonlinear_Model_4s(x,Model); %[A4,B4,Vmax,alpha,beta]
    % When disturbancing weigth and heigh, these parameters are changed


    x14=[x1;x2;x3;x4];
    %%% Calculate fx for 4-state system
    %%% fx = Ax + Bu
    %%% Use u from controller voltage input
    fx = A4*x14 + B4*u;


    %calculate the function output 'fx' based on values of A, B, P and x.
end
```

Function file: Gyroboy_Nonlinear_Model_4s.m

```
function [A4,B4,Vmax,alpha,beta]=Gyroboy_Nonlinear_Model_4s(x,Model)

%%% LEGO EV3 parameters 4-states
%%% Physical constants
g = 9.81;                       % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                      % wheel weight [kg]  old 0.024 new 0.050
R = 0.027;                      % wheel radius [m]
Jw = m * R^2 / 2;               % wheel inertia moment [kgm^2]
W = 0.105;                      % body width [m]
D = 0.1;                        % body depth [m]


M = 0.64;                      % body weight [kg]  ,default,
h = 0.210;                      % body height [m]   ,default,


% %%%%% Adding new weight here %%%%%
% M = 0.64+0.06;                  % body weight [kg]  add 10%

% %%%%% Adding new height here %%%%%
% h = 0.21+0.02;                          % body height add 10%

L = h / 2;                      % distance of the center of mass from the
wheel axle [m] 10.5


%%% EV3 Motors parameters
Jm = 1e-5;                      % DC motor inertia moment [kgm^2]
Rm = 6.69;                      % DC motor resistance [Om]
Kb = 0.468;                     % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                     % DC motor torque constant [Nm/A]


Vmax=8.3; % Default Vmax for LEGO EV3

M2=M; % Use this when want to change new motor or mass

%%% Uncomment below when want to use new motor  %%%
%[M,Jm,Rm,Kb,Kt]=MaxonDCmotor_Ec60flat_100W_48V(M2);  Vmax = 48;

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


Jpsi = M * L^2 / 3;             % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;    % body yaw inertia moment [kgm^2]

n = 1;                          % Gear ratio
fm = 0.0022;                    % friction coefficient between body & DC motor
fw = 0;                         % friction coefficient between wheel & floor



%%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;

    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT
    %x5=x(5); %

     if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
```

```
     end
     if x4 == 0
       x4=1.0e-20;  % avoid Inf's and NaN's
    end


 %%% Nonlinear model %%%%

a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(x3)^2+4*M*R*L*cos(x3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(x3)-M^2*R*L^2*cos(x3)*g*sin(x3);
    e24=x4*sin(x3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(x3);
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x2dot equation

    e43=M*g*L*sin(x3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=x4*sin(x3)*(-M^2*R^2*L^2*cos(x3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(x3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation


    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);

    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);

    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;

    %define these extra function to reduce the coding complexity of A and B
    %matrices with the motor added in


    %define A and B using a nonlinear state-space gyro robot model,
    %including the motor part

    if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
    end
    if x4 == 0
       x4=1.0e-20;  % avoid Inf's and NaN's
    end


 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed
  % 4.Model C
```

```
   %  Model = 3;  % Select mode

  %%%-------- Model A ----------
if Model == 1
          %%% Primary Model A %%%%
     A4 =[0      1             0              0        ;
          0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)    ;
          0      0             0              1        ;
          0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)    ];


end


%%%-------- Model B ----------
if Model == 2
          %%%% Model B %%%%
     A4 =[0      1             0                      0        ;
          0  em22/(a+b)  (em23+em24*x4)/((a+b)*x3)    0        ;
          0      0             0                      1        ;
          0  em42/(a+b)  (em43+em44*x4)/((a+b)*x3)    0        ];
end
  %%%-------- Model AB10 ----------
  if Model == 3

            %%% Mix A&B %%%%%

             if x3 <= (10*pi/180) && x3 >= (-10*pi/180)
             %%%% Primary Model A %%%%
                A4 =[0      1             0              0         ;
                     0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)    ;
                      0      0             0              1        ;
                      0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)    ];

             else
                %%%% Model B %%%%
                A4 =[0      1             0                      0         ;
                       0  em22/(a+b)  (em23+em24*x4)/((a+b)*x3)    0         ;
                     0      0             0                      1        ;
                      0  em42/(a+b)  (em43+em44*x4)/((a+b)*x3)    0       ] ;
             end

  end


  %%%-------- Model C ----------
if Model == 4
       %%%% Model C %%%%
     A4 =[0      1             0              0                          ;
          0  em22/(a+b)         0        (em23+em24*x4)/((a+b)*x4)     ;
          0      0             0              1                        ;
          0  em42/(a+b)        0          (em43+em44*x4)/((a+b)*x4)   ];


end

    % Models Matrix B

    B4 =[    0           0;
        fm21/(a+b)  fm22/(a+b);
            0           0;
        fm41/(a+b)  fm42/(a+b)];
```

```
  end
-----------------------------------------
```

Function file: MaxonDCmotor_Ec60flat_100W_48V.m

```
function [M,Jm,Rm,Kb,Kt] = MaxonDCmotor_Ec60flat_100W_48V(Mi)

%Maxon DC motor Ec 60 flat 100W  v.2020

M = Mi+0.55;                        % New Robot body weight [kg] with new motor
%M = 0.64+0.55;                     % New Robot body weight [kg] with new motor
                                    % maxon 0.355 kg/ea,   two LEGO motors 0.160
kg
                                    %(0.355x2)-0.160=0.55)

Jm = 8.35e-5;                       % DC motor inertia moment [kgm^2]
Rm = 1.1;                           % DC motor resistance [Om]
Kb = 0.113;                         % DC motor back EMF constant [Vsec/rad]
Kt = 0.113;                         % DC motor torque constant [Nm/A]

end
```

Appendix A.5.5: Apply linear quadratic regulator function in MATLAB for the LQG gain

The algebraic Riccati equation (Brunton & Kutz, 2019):

$$AP + PA^T - PC^T R^{-1} CP + Q = 0 \tag{5.16}$$

Equation (5.16) can be solved by applying the linear quadratic regulator in MATLAB function commanded by:

$$[K, P, E] = lqr(A, C^T, Q, R).$$

Appendix A.5.6: Rank of the observability matrix command

The rank of the observability matrix is applied by substituting $\boldsymbol{A}$ and $\boldsymbol{C}$ matrices into Eq.(5.18),

$$\mathcal{O} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \boldsymbol{CA^2} \\ \boldsymbol{CA^3} \end{bmatrix} \tag{5.18}$$

then apply MATLAB command:

$$Result = rank(\mathcal{O}) \text{ or } Result = rank(obsv(\boldsymbol{A}, \boldsymbol{C})).$$

The result given is Rank($\mathcal{O}$) = 4, equal to the number of columns in matrix $\boldsymbol{C}$; therefore, the system is said to be completely state observable. Likewise, the rank test of Eq.(19) can use the same MATLAB command.

Appendix A.5.7:MATLAB codes of an inverted pendulum on a cart system using LQG controller

Script file: Gyroboy_5s_LQG_10_2021.m

```
%%%% Linear Control with EKF (LQG) for LEGO EV3 Robot

%%%%%%  Functions programme needed %%%
%
% evalrhs_gyroboy5s_LQG();  % Generating K1-K4, fx and Kf
%
%       Inside two functions
%
%            Gyroboy_Nonlinear_Model_4s();
%            Maxon Motor parameters etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all;
close all;
```

```
%%% Set initial x1-x5
%%% Always set x1=x3

x1(1)=10*pi/180;      %set Theta - Average of wheel angles (deg)
x2(1)=0;                %set Theta_DOT (deg/s)
x3(1)=10*pi/180; %set Psi - Pitch angle of body (deg) - CONTROL to ZERO
x4(1)=0*pi/180;         %set Psi_DOT (deg/s)
x5(1)=0;                %set Theta integral

 %%% Set Model No.1 for linear control method %%%%
  Model=1;

u_in1(1)=0;  % Left  Motor Voltage
u_in2(1)=0;  % Right Motor Voltage
u_feedback =[0;0]; % u_feedback = Kx = (K_LQR) x (X1-X4)

%%% Set initial Xhat
Xhat1(1) = x1(1);
Xhat2(1) = 0;
Xhat3(1) = x3(1);
Xhat4(1) = 0;

%%% Set initial Xhat integral
dXhat1(1) = 0;
dXhat2(1) = 0;
dXhat3(1) = 0;
dXhat4(1) = 0;

%%% Disturbance variables
x1_disturbance(1) = 0;
x2_disturbance(1) = 0;
x3_disturbance(1) = 0;
x4_disturbance(1) = 0;


x1_err = 0;      % x1 error for tracking sys
x1_ref = 0;      % x1 reference
x1_err_int(1)=0; % x1 error integral

Ts=0.0001; %time step length

alphaa=1;  % Motor Variable
betaa=0;   % Motor Variable

Duration=10;       % time sec
t=Duration*(1/Ts); % time step in programming
Time = 0:Ts:t*Ts;          % Create real time step for plotting

for i=1:t

  u      =[u_in1(i); u_in2(i)];
  x_5s    =[x1(i); x2(i); x3(i); x4(i); x5(i)];   % whole system
  x_4s    =[x1(i); x2(i); x3(i); x4(i)];          % x1-x4 state feedback
  Xhat =[Xhat1(i); Xhat2(i); Xhat3(i); Xhat4(i)]; % Xhat1-4 (exclude x5)
  dXhat=[dXhat1(i); dXhat2(i); dXhat3(i); dXhat4(i)]; %Xhat integral
```

```
 %%%%%% Programming Diagram %%%%%

%
%                     u_x1  +     u                                y=x1-x4            y=x1
%  X1ref->-o-->-Int--K5-->-->o---------------+-------| Plant |-->----------------+---C----
--+
%          ^-                ^ -            |                                ^   |
|
%          |                 | u_feedback   +------B------>|         Add   |   |
|
%          |                 |                             |       Distubance  |
%          x1                |       Xhat            dXhat |                   |
%                            +-<--K14-- o<--Integal <----- o <-- Kf-------- ---+
%                                       |                  ^       x1-x4
%                                       |                  |
%                                       +---- A -(Kf)C ->--+

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

    %%% x1 error for tracking sys
    x1_err(i+1) = x1_ref - x1(i);

      %%%%% X1 error Integral  %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

      %%% x1 error integral
    x1_err_int(i+1) = x1_err_int(i) + x1_err(i)*Ts + 0.5*(x1_err(i+1)-
x1_err(i))*Ts ;  % [h1 x L]+[ 0.5 x(h2-h1) x L ] + old

     %%% cal. K5 for x5 only (integral x1)
     K_5=[-0.5000; -0.5000];

     %%% u=kx,  Motor Voltage of integral x1
     u_x1 = K_5 * x1_err_int(i+1);

    %%% The final voltages used to control robot motors
    u = u_x1 - u_feedback;

   %%% Hard Saturation  %%%
   %%% Uncomment this part for using motor voltage hard saturation
%     Vmax = 8.3;
%     %Vmax = 36;
%     %Vmax = 48;
%              if u(1) > Vmax
%                 u(1) = Vmax;
%                 u(2) = Vmax;
%              end
%               if u(1) < -Vmax
%                 u(1) = -Vmax;
%                 u(2) = -Vmax;
%               end
    %%%%%%%

    u_in1(i+1)=u(1);   % update u1
    u_in2(i+1)=u(2);   % update u2



    %%% calculate the new 'x' vector using a 4th order
    %%% Euler integration method
    %%% fx = Ax + Bu;
    [fx,u,A,B,C,Kf,K_14] = evalrhs_gyroboy5s_LQG(x_4s,u,Model);
    x_4s = x_4s + Ts * fx; % Euler
     %%% Kf is the gain of Kalman filter
```

```
    %%% Limit the robot pitch angle between -90 to 90 deg.
    if x_4s(3) > 90*pi/180
        x_4s(3) =90*pi/180;
    end

     if x_4s(3) < -90*pi/180
        x_4s(3) =-90*pi/180;
     end

     %%% Update x1-x4
    x1(i+1)=x_4s(1);
    x2(i+1)=x_4s(2);
    x3(i+1)=x_4s(3);
    x4(i+1)=x_4s(4);


       %%% Generate real x1 integral (x5)
       %%%%% X5 = Integral of X1 %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

      x5(i+1) = x5(i) + x1(i)*Ts + 0.5* (x1(i+1)-x1(i))* Ts;


  %%%% Select mode for testing %%%%
  % 0.No disturbance
  % 1.Sensor X3 drift
  % 2.Noise disturbance in X3

     Mode = 0;  % Select mode 0

if Mode == 0
    %%%%% No disturbance in X
     x1_disturbance(i+1)=x_4s(1);
     x2_disturbance(i+1)=x_4s(2);
     x3_disturbance(i+1)=x_4s(3)+ 0;
     x4_disturbance(i+1)=x_4s(4);
    Kf_X = Kf*[x_4s(1); x_4s(2); x_4s(3); x_4s(4)];   %%% Update:   Kf_X

end



if Mode == 1

     %%%%%% Test signal disturbance, drift only X3  %%%%%
    x1_disturbance(i+1)=x_4s(1);
    x2_disturbance(i+1)=x_4s(2);
    x3_disturbance(i+1)=x_4s(3)+ (i/(100*400))*pi/180;
    % x3_disturbance(i+1)=x_4s(3)+ (i/(100*40))*pi/180;
    x4_disturbance(i+1)=x_4s(4);
    Kf_X = Kf*[x_4s(1); x_4s(2); x3_disturbance(i+1) ; x_4s(4)];  %%% Update:
Kf_X

end

if Mode == 2
    %%%%%%% Test noise disturbance, only X3  %%%%%%

   %%%ramdom noise fixed
   %%%ramdom noise %Ts=0.004 10sec.
   r=[0,0,0,0,0,2.42860700227814,0,0,0,0,3.98660366872073,0,0,0,0,-
3.99063370811201,0,0,0,0,-1.97150702498759,0,0,0,0,-3.74576261992009,0,0,0,0,-
```

```
0.00306732125453291,0,0,0,0,2.61619387183277,0,0,0,0,1.66560733620697,0,0,0,0,
…….. (cannot paste too many data at here)
.13838561080327,0,0,0,0,3.98947101136213];

      x1_disturbance(i+1)=x_4s(1);
      x2_disturbance(i+1)=x_4s(2);
      x3_disturbance(i+1)=x_4s(3)+ r(i+1)*pi/180;
      x4_disturbance(i+1)=x_4s(4);
     Kf_X = Kf*[x_4s(1); x_4s(2); x3_disturbance(i+1) ; x_4s(4)];  %%% Update:
Kf_X

end


    %%% Kalman filter variable
    %%% See more in coding diagram
    A_KfC_Xhat =  (A-Kf*C)*Xhat;
    dXhat = B*u + Kf_X + A_KfC_Xhat;

    %%% Update
    dXhat1(i+1)=dXhat(1);
    dXhat2(i+1)=dXhat(2);
    dXhat3(i+1)=dXhat(3);
    dXhat4(i+1)=dXhat(4);

      %%%%% dXhat Integral (Xhat) %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

    Xhat1(i+1) = Xhat1(i) + dXhat1(i)*Ts + 0.5* (dXhat1(i+1)-dXhat1(i))* Ts;
    Xhat2(i+1) = Xhat2(i) + dXhat2(i)*Ts + 0.5* (dXhat2(i+1)-dXhat2(i))* Ts;
    Xhat3(i+1) = Xhat3(i) + dXhat3(i)*Ts + 0.5* (dXhat3(i+1)-dXhat3(i))* Ts;
    Xhat4(i+1) = Xhat4(i) + dXhat4(i)*Ts + 0.5* (dXhat4(i+1)-dXhat4(i))* Ts;

    %%% Limit the robot pitch angle Xhat between -90 to 90 deg.
    if  Xhat3(i+1) > 90*pi/180
         Xhat3(i+1)=90*pi/180;
    end

     if  Xhat3(i+1) < -90*pi/180
         Xhat3(i+1)=-90*pi/180;
    end


    %%% Select control feedback : Xhat3 for x3 %%%%

   %u_feedback = K_14*[x_4s(1); x_4s(2); x_4s(3); x_4s(4)]; %%% X LQR only
   %u_feedback = K_14*[Xhat1(i+1); Xhat2(i+1); Xhat3(i+1); Xhat4(i+1)]; %Xhat

   u_feedback = K_14*[x_4s(1); x_4s(2); Xhat3(i+1); x_4s(4)];   %X mix only X3



      %%% Command print for waiting %%%
    Cal_percent = i*100/t;

    if mod(Cal_percent , 10) == 0
      % fprintf('Calculating  %f percent ...\n',Cal_percent)
         fprintf('Calculating  %3.2f percent,  ',Cal_percent)
       fprintf('x3= %3.2f deg,  ',x3(i+1)*180/pi)
       fprintf('x4= %3.2f deg/s,  ',x4(i+1)*180/pi)
       fprintf('t= %3.2f sec...\n',Ts*i)
    end

end
```

```
%%% Move all Xhat to x3 for plotting
x3=Xhat3;


%%%%%%%%%%
figure('Name','LQG Controller of a TWR system');
Fn = 14; % font size

subplot(2,3,1);
p1=plot(Time,x1*180/pi,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Wheel angles-x1
(deg.)','FontSize', Fn);
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2=plot(Time,x2*180/pi,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Wheel angular velocity-x2
(deg./s.)','FontSize', Fn);
set(gca,'FontSize', Fn);

subplot(2,3,2);
p3=plot(Time,x3*180/pi,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Pitch of body-x3
(deg.)','FontSize', Fn);
set(gca,'FontSize', Fn);

 subplot(2,3,5);
 p4=plot(Time,x4*180/pi,'b'); grid;
 xlabel('Time (sec)','FontSize', Fn); ylabel('Pitch angular velocity-x4
(deg./s.)','FontSize', Fn);
 set(gca,'FontSize', Fn);

 subplot(2,3,3);
 p5=plot(Time,x5*180/pi,'b'); grid;
 xlabel('Time (sec)','FontSize', Fn); ylabel('Wheel integral-x5
(deg.)','FontSize', Fn);
 set(gca,'FontSize', Fn);

subplot(2,3,6);
p6=plot(Time,u_in1,'b'); grid;
xlabel('Time (sec)','FontSize', Fn); ylabel('Control signal-u
(volt)','FontSize', Fn);
set(gca,'FontSize', Fn);

LW=1.2;
p1(1).LineWidth = LW;
p2(1).LineWidth = LW;
p3(1).LineWidth = LW;
p4(1).LineWidth =LW;
p5(1).LineWidth = LW;
p6(1).LineWidth =LW;
```

Function file: evalrhs_gyroboy5s_LQG

```
function [fx,u,A4,B4,C4,Kf,K_14] = evalrhs_gyroboy5s_LQG( x ,u,Model)

    %%%%%%  Functions programme needed %%%
    % Gyroboy_Nonlinear_Model_4s();  % Generating fx, x1-x4 without x5
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT


  %%% K14 parameters already have been calculated as the linear control gain
is fixed
  %%% K14 = k1,k2,k3 and k4

  %%% Select one K by uncomment the coding

   %%%%% Fix K LEGO EV3 %%%
   K_14=[-1.3908   -1.4487  -59.8476   -7.1681     ;
         -1.3908   -1.4487  -59.8476   -7.1681    ];


%    %Maxon DC motor EC flat 60   48V 100w  2020 ***
%      K_14= [ -1.2606   -0.7236  -34.5003   -4.2534
%             -1.2606   -0.7236  -34.5003   -4.2534];


    %%% 4-states Model parameters
    %%% Real model parameters
    [A4,B4,~,~,~]=Gyroboy_Nonlinear_Model_4s(x, Model);
%[A4,B4,Vmax,alpha,beta]
    % When disturbancing weigth and heigh, these parameters are changed



    x14=[x1;x2;x3;x4];
    %%% Calculate fx for 4-state system
    %%% fx = Ax + Bu
    %%% Use u from controller voltage input
    fx = A4*x14 + B4*u;


  %%% Matrix C
  C4=eye(4);


   %%% Kalman filter noise parameters
   Noise_V= 0.2*eye(4); %0.2   % increse to smooth cure % Rk of Kalman filter
   Noise_W= 1*eye(4);                                  % Qk of Kalman filter


  %%% Calculation gain Kf of Kalman filter
  %[~,Pk,~]=lqr(AL4,C4',Noise_W,Noise_V);

  %%% Reducing time by calcuting fixed Pk
  %%% Select robot motors by uncomment

  %%%% Fix Pk for EV3 Motor%%%
 Pk=[0.4375    0.0114    0.0795    0.0464;
    0.0114    0.0116    0.1403    0.0451;
    0.0795    0.1403    4.1167    0.6856;
    0.0464    0.0451    0.6856    0.2096];


     %Maxon DC motor EC flat 60   48V 100w  2020
%  Pk=[0.4396    0.0196    0.0463    0.0650
%     0.0196    0.0209    0.1031    0.0607
%     0.0463    0.1031    4.7106    0.6864
%     0.0650    0.0607    0.6864    0.2747];
```

```
   Kf= Pk*C4'*inv(Noise_V);

    %calculate the function output 'fx' based on values of A, B, P and x.
end
```

Chapter 6: Nonlinear Control Strategies

Appendix A.6.1: MATLAB codes of rank of controllability test matrix for an inverted pendulum on a cart model

Script file: Controllability_IP_4s_x3x4.m

```
clear all
close all

xx1=0;  %set initial cart displacement to be 0 (m)
xx2=0;  %set initial cart velocity to be 0 (m/s)
xx3=0; %set initial pendulum angle to be 0 (rad)
xx4=0; %set initial pendulum angular velocity to be 0 (rad/s)


m1=2; %Mass of the cart (kg)
m2=0.1; %mass of the pendulum (kg)
r=0.5; %the rod length (m)
g=9.8; %acceleration due to gravity (m^2/s)


%%%%%%%%%%%%%%%%%%%%%%%% start controllability test %%%%%%%%%%%%%%%%%%%%

%%% Variable 'for loop' %%%%

%%%% X3
Minimun_Angle_X3= -90;
Step_Angle_X3= 0.2;
Maximum_Angle_X3= 90;

%%%% X4

Minimun_Angular_Velo_X4= -0.3e8;
Step_Angular_Velo_X4= 0.2e6;
Maximum_Angular_Velo_X4= 0.3e8;

%%% Calculation size of X4
 Size_x4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4:
Maximum_Angular_Velo_X4;
 [m_size,n_size]= size(Size_x4);

%%%%% Plot 2D graph %%%%%%%%

ii=2;  % row of array   x3
jj=2;  % column of array x4

%%% match x3 and x4 %%%

for xx4 = Minimun_Angular_Velo_X4*pi/180 : Step_Angular_Velo_X4*pi/180 :
Maximum_Angular_Velo_X4*pi/180
```

```
  for xx3 = Minimun_Angle_X3*pi/180 : Step_Angle_X3*pi/180 :
Maximum_Angle_X3*pi/180

     if xx3 == 0
        xx3=1.0e-100;  % avoid Inf's and NaN's
     end
       if xx4 == 0
        xx4=1.0e-100;  % avoid Inf's and NaN's
     end


%%% Nonlinear model from Xu's chapter book %%%%

A4=[0, 1, 0, 0;
0, 0,-m2*g*sin(xx3)*cos(xx3)/((m1+m2*(sin(xx3))^2)*xx3),...
m2*r*xx4*sin(xx3)/(m1+m2*(sin(xx3)^2));
0, 0, 0, 1;
0, 0,(m1+m2)*g*sin(xx3)/(r*(m1+m2*(sin(xx3))^2)*xx3),...
-m2*r*xx4*sin(xx3)*cos(xx3)/(r*(m1+m2*(sin(xx3))^2))];

B4=[0; 1/(m1+m2*(sin(xx3))^2); 0 ; -cos(xx3)/(r*(m1+m2*(sin(xx3))^2))];


Rank_x3x4(ii,jj)= rank(ctrb(A4,B4));


Rank_x3x4(ii,1)= xx3*180/pi;  % Insert 1st column by x3
Rank_x3x4(1,jj)= xx4*180/pi;  % Insert 1st row by x4


 ii=ii+1; %x3
  end
  ii=2;
  jj=jj+1; %x4


  %%% print for waiting %%%
    Cal_percent = int16(jj*100/(n_size-1));


    if mod(Cal_percent , 10) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end


  end


 %%%% Remove 1st row & 1st column from table for plotting %%%
  Rank_x3x4_data = Rank_x3x4;
  Rank_x3x4_data(1,:)=[];  % Remove title row 1 ( x4 name)
  Rank_x3x4_data(:,1)=[];  % Remove title column 1 ( x3  name)
  Rank_x3x4_data = Rank_x3x4_data'; % Transpose matrix x3=x-axixs, x4=y-axis



 %%%%% plot %%%%%
 figure(); % x3 x4
 Fn = 14; % font size
 xx4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4 :
Maximum_Angular_Velo_X4;
 xx3 = Minimun_Angle_X3 : Step_Angle_X3 : Maximum_Angle_X3;

[x_mesh,y_mesh] = meshgrid(xx3,xx4);  % This generates the actual grid of x
and y values.
```

```
[Mc,c]=contourf(x_mesh,y_mesh,Rank_x3x4_data);
set(c,'Linecolor','none')

xlabel('x3 (deg.)');ylabel('x4 (deg./s)');
ylim([Minimun_Angular_Velo_X4 Maximum_Angular_Velo_X4]);
set(gca,'FontSize', Fn);
title(sprintf('Rank of Controllability Test Matrix'));

colorbar
 c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
%c =colorbar;
c.Label.String = 'Rank';
grid on;
```

Appendix A.6.2: MATLAB codes of rank of controllability test matrix for TWR models

Script file: Controllability_TWR_5s_x3x4.m

```
clear all
close all

xx1=0;  %set initial wheel angle to be 0 (m)
xx2=0;  %set initial wheel angular velocity to be 0 (m/s)
xx3=0; %set initial pitch angle to be 0 (rad)
xx4=0; %set initial pitch angular velocity to be 0 (rad/s)

%%% Physical constants
g = 9.81;                          % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                         % wheel weight [kg]
R = 0.027;                         % wheel radius [m]
Jw = m * R^2 / 2;                  % wheel inertia moment [kgm^2]
W = 0.105;                         % body width [m]
D = 0.1;                           % body depth [m]

M = 0.64;                         % body weight [kg]


%%% EV3 Motors parameters
Jm = 1e-5;                         % DC motor inertia moment [kgm^2]
Rm = 6.69;                         % DC motor resistance [Om]
Kb = 0.468;                        % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                        % DC motor torque constant [Nm/A]

h = 0.210;                         % body height [m]
L = h / 2;                         % distance of the center of mass from the
wheel axle [m] 10.5
Jpsi = M * L^2 / 3;                % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;       % body yaw inertia moment [kgm^2]

n = 1;                             % Gear ratio
fm = 0.0022;                       % friction coefficient between body & DC motor
fw = 0;                            % friction coefficient between wheel & floor


    %%% Helping variables
```

```
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;


%%%%%%%%%%%%%%%%%%%%%%%% start controllability test %%%%%%%%%%%%%%%%%%%%

%%% Variable 'for loop' %%%%

%%%% X3
Minimun_Angle_X3= -90;
Step_Angle_X3= 0.2;
Maximum_Angle_X3= 90;
%%%% X4
Minimun_Angular_Velo_X4= -1e5;
Step_Angular_Velo_X4= 0.2e3;
Maximum_Angular_Velo_X4= 1e5;


%%% Calculation size of X4
 Size_x4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4:
Maximum_Angular_Velo_X4;
 [m_size,n_size]= size(Size_x4);

%%%%% Plot 2D graph %%%%%%%%

ii=2;  % row of array   x3
jj=2;  % column of array x4


%%% match x3 and x4 %%%
for xx4 = Minimun_Angular_Velo_X4*pi/180 : Step_Angular_Velo_X4*pi/180 :
Maximum_Angular_Velo_X4*pi/180
  for xx3 = Minimun_Angle_X3*pi/180 : Step_Angle_X3*pi/180 :
Maximum_Angle_X3*pi/180

     if xx3 == 0
        xx3=1.0e-100;  % avoid Inf's and NaN's
     end
       if xx4 == 0
        xx4=1.0e-100;  % avoid Inf's and NaN's
     end


 %%% Dr Xu Xu Model %%%


a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(xx3)^2+4*M*R*L*cos(xx3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(xx3)-M^2*R*L^2*cos(xx3)*g*sin(xx3);
    e24=xx4*sin(xx3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(xx3);
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x2dot equation

    e43=M*g*L*sin(xx3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=xx4*sin(xx3)*(-M^2*R^2*L^2*cos(xx3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(xx3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation
```

```
    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);


    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);


    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;



 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model A&B mixed
  % 4.Model C

     Model = 1;  % Select mode

  %%%-------- Model A ----------
if Model == 1
          %%%% Primary Model A %%%%
      A5 =[0      1             0              0       0  ;
           0  em22/(a+b)  em23/((a+b)*xx3)  em24/(a+b)  0 ;
           0      0             0              1       0 ;
           0  em42/(a+b)  em43/((a+b)*xx3)  em44/(a+b)  0 ;
           1      0             0              0       0 ];
end


%%%-------- Model B ----------
if Model == 2
          %%%% Model B %%%%
    A5 =[0      1             0                     0       0  ;
         0  em22/(a+b)  (em23+em24*xx4)/((a+b)*xx3)   0       0 ;
         0      0             0                     1       0 ;
         0  em42/(a+b)  (em43+em44*xx4)/((a+b)*xx3)   0       0 ;
         1      0             0                     0       0 ];

end



  %%%-------- Model A&B ----------
if Model == 3

  %%% Mix A&B %%%%%

   if xx3 <= (10*pi/180) && xx3 >= (-10*pi/180)
    %%%% Primary Model A %%%%
     A5 =[0      1             0              0       0  ;
          0  em22/(a+b)  em23/((a+b)*xx3)  em24/(a+b)  0 ;
          0      0             0              1       0 ;
```

```
         0  em42/(a+b)   em43/((a+b)*xx3)   em44/(a+b)   0 ;
         1       0             0                0       0 ];


  else


      %%%% Model B %%%%
     A5 =[0       1             0                      0       0  ;
         0  em22/(a+b)   (em23+em24*xx4)/((a+b)*xx3)   0       0 ;
         0       0             0                      1       0 ;
         0  em42/(a+b)   (em43+em44*xx4)/((a+b)*xx3)   0       0 ;
         1       0             0                      0       0 ];
  end

end


 %%%-------- Model C ----------
if Model == 4
          %%%% Model C %%%%
     A5 =[0       1             0              0                          0  ;
         0  em22/(a+b)         0         (em23+em24*xx4)/((a+b)*xx4)    0 ;
         0       0             0               1                         0 ;
         0  em42/(a+b)         0          (em43+em44*xx4)/((a+b)*xx4)  0 ;
         1       0             0               0                         0 ];

end

     B5 =[    0           0;
        fm21/(a+b)   fm22/(a+b);
            0           0;
        fm41/(a+b)   fm42/(a+b);
          0             0];


Rank_x3x4(ii,jj)= rank(ctrb(A5,B5));

Rank_x3x4(ii,1)= xx3*180/pi;  % Insert 1st column by x3
Rank_x3x4(1,jj)= xx4*180/pi;  % Insert 1st row by x4


 ii=ii+1; %x3

  end
  ii=2;

  jj=jj+1; %x4


  %%% print for waiting %%%
    Cal_percent = int16(jj*100/(n_size-1));


    if mod(Cal_percent , 10) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end

  end


 %%%% Remove 1st row & 1st column from table for plotting %%%
  Rank_x3x4_data = Rank_x3x4;
  Rank_x3x4_data(1,:)=[];  % Remove title row 1 ( x4 name)
  Rank_x3x4_data(:,1)=[];  % Remove title column 1 ( x3  name)
```

```
  Rank_x3x4_data = Rank_x3x4_data'; % Transpose matrix x3=x-axixs, x4=y-axis


 %%%%% plot max %%%%%
 figure(); % x3 x4
 Fn = 14; % font size
 xx4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4 :
Maximum_Angular_Velo_X4;
 xx3 = Minimun_Angle_X3 : Step_Angle_X3 : Maximum_Angle_X3;

[x_mesh,y_mesh] = meshgrid(xx3,xx4);  % This generates the actual grid of x
and y values.
[Mc,c]=contourf(x_mesh,y_mesh,Rank_x3x4_data);
%c.LineWidth = 1;
set(c,'Linecolor','none')

xlabel('x3 (deg.)');ylabel('x4 (deg./s)');
xlim([-90 90]);
ylim([Minimun_Angular_Velo_X4 Maximum_Angular_Velo_X4]);
set(gca,'FontSize', Fn);
title(sprintf('Rank of Controllability Test Matrix'));

colorbar
 c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';
grid on;
```

Appendix A.6.3: MATLAB codes of rank of controllability test matrix for TWR models with saturation input

Script file: Controllability_TWR_5s_x3x4_saturation.m

```
clear all
close all


xx1=0;  %set initial wheel angle to be 0 (m)
xx2=0;  %set initial wheel angular velocity to be 0 (m/s)
xx3=0; %set initial pitch angle to be 0 (rad)
xx4=0; %set initial pitch angular velocity to be 0 (rad/s)
xx6=0; %set initial artificial control to be 0


%%% Physical constants
g = 9.81;                          % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                         % wheel weight [kg]
R = 0.027;                         % wheel radius [m]
Jw = m * R^2 / 2;                  % wheel inertia moment [kgm^2]
W = 0.105;                         % body width [m]
D = 0.1;                           % body depth [m]

M = 0.64;                         % body weight [kg]
```

```
h = 0.210;                          % body height [m]
L = h / 2;                          % distance of the center of mass from the
wheel axle [m] 10.5

Jpsi = M * L^2 / 3;                 % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;        % body yaw inertia moment [kgm^2]

%%% Motors parameters
Jm = 1e-5;                          % DC motor inertia moment [kgm^2]
Rm = 6.69;                          % DC motor resistance [Om]
Kb = 0.468;                         % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                         % DC motor torque constant [Nm/A]
n = 1;                              % Gear ratio
fm = 0.0022;                        % friction coefficient between body & DC motor
fw = 0;                             % friction coefficient between wheel & floor


    %%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;


%%%%%%%%%%%%%%%%%%%%%%%% start controllability test %%%%%%%%%%%%%%%%%%%%

%%% Variable 'for loop' %%%%

%%% X3
Minimun_Angle_X3= -90;
Step_Angle_X3= 2;
Maximum_Angle_X3= 90;

%%% X4
Minimun_Angular_Velo_X4= -200;
Step_Angular_Velo_X4= 4;
Maximum_Angular_Velo_X4= 200;


%%%%% X6
Minimun_x6= -1e16;
Step_x6= 2e14;
Maximum_x6= 1e16;

% Minimun_x6= -2e19;
% Step_x6= 4e17;
% Maximum_x6= 2e19;


 %%% Calculation size of Xnp1
 Size_x6 = Minimun_x6 : Step_x6: Maximum_x6;
 [m_x6,n_x6]= size(Size_x6);


%%%%% Plot 3D graph %%%%%%%%

ii=2;   % row of array    x3
jj=2;   % column of array x4
kk=1;   % page Xn+1


%%% convert deg to rad
%%% match x3,x4 and x6 %%%
 for xx6 = Minimun_x6 : Step_x6: Maximum_x6
 for xx3 = Minimun_Angle_X3*pi/180 : Step_Angle_X3*pi/180 :
Maximum_Angle_X3*pi/180
```

```
 for xx4 = Minimun_Angular_Velo_X4*pi/180 : Step_Angular_Velo_X4*pi/180 :
Maximum_Angular_Velo_X4*pi/180

     if xx3 == 0
        xx3=1.0e-20;  % avoid Inf's and NaN's
     end
     if xx4 == 0
        xx4=1.0e-20;  % avoid Inf's and NaN's
     end
     if xx6 == 0
        xx6=1.0e-20;  % avoid Inf's and NaN's
     end


 %%% Dr Xu Xu model %%%


a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(xx3)^2+4*M*R*L*cos(xx3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(xx3)-M^2*R*L^2*cos(xx3)*g*sin(xx3);
    e24=xx4*sin(xx3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(xx3);
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x2dot equation

    e43=M*g*L*sin(xx3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=xx4*sin(xx3)*(-M^2*R^2*L^2*cos(xx3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(xx3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation


    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);


    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);


    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;



 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed
  % 4.Model C
```

```
    Model = 1;  % Select mode

  %%%-------- Model A ----------
if Model == 1
          %%%% Primary Model A %%%%
      A5 =[0      1             0              0       0  ;
           0  em22/(a+b)  em23/((a+b)*xx3)  em24/(a+b)  0 ;
           0      0             0              1       0 ;
           0  em42/(a+b)  em43/((a+b)*xx3)  em44/(a+b)  0 ;
           1      0             0              0       0 ];
end


%%%-------- Model B ----------
if Model == 2
          %%%% Model B %%%%
     A5 =[0      1             0                      0       0  ;
          0  em22/(a+b)  (em23+em24*xx4)/((a+b)*xx3)   0       0 ;
          0      0             0                      1       0 ;
          0  em42/(a+b)  (em43+em44*xx4)/((a+b)*xx3)   0       0 ;
          1      0             0                      0       0 ];

end


  %%%-------- Model AB7 ----------
if Model == 3

  %%% Mix A&B %%%%%

   if xx3 <= (10*pi/180) && xx3 >= (-10*pi/180)
    %%%% Primary Model A %%%%
     A5 =[0      1             0              0       0  ;
          0  em22/(a+b)  em23/((a+b)*xx3)  em24/(a+b)  0 ;
          0      0             0              1       0 ;
          0  em42/(a+b)  em43/((a+b)*xx3)  em44/(a+b)  0 ;
          1      0             0              0       0 ];


  else


      %%%% Model B %%%%
     A5 =[0      1             0                      0       0  ;
          0  em22/(a+b)  (em23+em24*xx4)/((a+b)*xx3)   0       0 ;
          0      0             0                      1       0 ;
          0  em42/(a+b)  (em43+em44*xx4)/((a+b)*xx3)   0       0 ;
          1      0             0                      0       0 ];
  end

end

  %%%-------- Model C ----------
if Model == 4
       %%%% Model C %%%%
     A5 =[0      1             0              0                          0  ;
          0  em22/(a+b)        0        (em23+em24*xx4)/((a+b)*xx4)     0 ;
          0      0             0              1                          0 ;
          0  em42/(a+b)        0         (em43+em44*xx4)/((a+b)*xx4)  0 ;
          1      0             0              0                          0 ];

end


     B5 =[    0           0;
```

```
        fm21/(a+b)   fm22/(a+b);
            0            0;
        fm41/(a+b)   fm42/(a+b);
          0              0];


     %%%%--------------

      Vmax = 8.3;  % Set maximum saturation voltage


     if      (xx6 )  > Vmax  % Lego Motor Maximum Voltage 8.3 V
         Phi_L = Vmax;
         Phi_R = Vmax;
     elseif  (xx6) < -Vmax % Lego Motor Minimum Voltage -8.3 V
         Phi_L = -Vmax;
         Phi_R = -Vmax;
     else


         Phi_L = Vmax*sin((pi*xx6)/(2*Vmax));
         Phi_R = Vmax*sin((pi*xx6)/(2*Vmax));
     end

     Phi = [Phi_L ;
            Phi_R ];


Aa= [       A5             (B5*Phi)/xx6  ;
       zeros(1,5)             0             ];

Ba= [  zeros(5,1)  ;
          1        ];


Rank_x3x4x6(ii,jj,kk)= rank(ctrb(Aa,Ba));


%%% create table
Rank_x3x4x6(ii,1,kk)= xx4*180/pi;  % Insert 1st column by x3
Rank_x3x4x6(1,jj,kk)= xx3*180/pi;  % Insert 1st row    by x4


    if xx4 >-0.01 && xx4 <0.01
        kk=kk+1;
    Rank_x3x6(jj,kk) = rank(ctrb(Aa,Ba));
    Rank_x3x6(jj,1) = xx3*180/pi;  % Insert 1st row by  x3
    Rank_x3x6(1,kk) = xx6; % Insert 1st column by xpn1
        kk=kk-1;
    end

    if single(xx3)==single(1.0e-20)
         kk=kk+1;
    Rank_x4x6(ii,kk) = rank(ctrb(Aa,Ba));
    Rank_x4x6(ii,1) = xx4*180/pi;  % Insert 1st row by x4
    Rank_x4x6(1,kk) = xx6; % Insert 1st column by xpn1
         kk=kk-1;
    end


     if single(xx6)==single(1.0e-20)
    Rank_x3x4(ii,jj) = rank(ctrb(Aa,Ba));
    Rank_x3x4(ii,1) = xx4*180/pi;  % Insert 1st row by x4
    Rank_x3x4(1,jj) = xx3*180/pi; % Insert 1st column by xpn1
    end
```

```
 ii=ii+1; %x3
  end
  ii=2;
  jj=jj+1; %x4
 end



     if xx6 == 1.0e-20
        xx6=0;  % move zero back to table
     end


    Range_x6(1,kk)=xx6; % store x_np1

  jj=2;
  kk=kk+1; %x_np1

    %%% print for writing  %%%
    Cal_percent = int16(kk*100/(n_x6+1));

    if mod(Cal_percent , 10) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end


end



  %%%% Remove 1st row & 1st column from table for plotting %%%
  Rank_x3x4x6_data = Rank_x3x4x6;
  Rank_x3x4x6_data(1,:,:)=[];  % Remove title row 1 ( x4 name)
  Rank_x3x4x6_data(:,1,:)=[];  % Remove title column 1 ( x3  name)

  Rank_x3x6_data = Rank_x3x6;
  Rank_x3x6_data(1,:)=[];  % Remove title row 1 ( xnp1 name)
  Rank_x3x6_data(:,1)=[];  % Remove title column 1

  Rank_x4x6_data = Rank_x4x6;
  Rank_x4x6_data(1,:)=[];  % Remove title row 1 ( xnp1 name)
  Rank_x4x6_data(:,1)=[];  % Remove title column 1

  Rank_x3x4_data = Rank_x3x4;
  Rank_x3x4_data(1,:)=[];  % Remove title row 1 ( xnp1 name)
  Rank_x3x4_data(:,1)=[];  % Remove title column 1


 xx3 = Minimun_Angle_X3 : Step_Angle_X3 : Maximum_Angle_X3;
 xx4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4 :
Maximum_Angular_Velo_X4;
 xx6 = Minimun_x6 : Step_x6: Maximum_x6;


%%%% Plot 2D at x3=0 %%%%
 figure();
 Fn = 12; % font size
[x_mesh,y_mesh,z_mesh] = meshgrid(xx3,xx4,xx6);  % This generates the actual
grid of x and y values.

xslice = [0];
yslice = [];
zslice = [];
```

```
slice(x_mesh,y_mesh,z_mesh,Rank_x3x4x6_data,xslice,yslice,zslice)
xlabel('x3-Pitch angle(deg.)');ylabel('x4-Pitch angular
velocity(deg./s.)');zlabel('x6');
xlim([-90 90]);
title(sprintf('Rank of Controllability'));


colorbar
c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';

%%%% view %%%%
view(90,0) % y-view
%%%%%%
set(gca,'FontSize', Fn);


%%Plot 2D at x4=0%%%%
 figure();
 Fn = 12; % font size
[x_mesh,y_mesh,z_mesh] = meshgrid(xx3,xx4,xx6);  % This generates the actual
grid of x and y values.

xslice = [];
yslice = [0];
zslice = [];

slice(x_mesh,y_mesh,z_mesh,Rank_x3x4x6_data,xslice,yslice,zslice)
xlabel('x3-Pitch angle(deg.)');ylabel('x4-Pitch angular
velocity(deg./s.)');zlabel('x6');
xlim([-90 90]);
title(sprintf('Rank of Controllability'));

colorbar
c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';

%%%% view %%%%
    view(0,0)  % x-view
%%%%%%
set(gca,'FontSize', Fn);




%%Plot Cubic %%%%
 figure();
 Fn = 12; % font size
[x_mesh,y_mesh,z_mesh] = meshgrid(xx3,xx4,xx6);  % This generates the actual
grid of x and y values.

xslice = [xx3];
yslice = [xx4];
zslice = [xx6];

%%%% or select plane for plotting %%%%
%xslice = [Maximum_Angle_X3];
%yslice = [Maximum_Angular_Velo_X4];
%zslice = [Minimun_x6,0];

slice(x_mesh,y_mesh,z_mesh,Rank_x3x4x6_data,xslice,yslice,zslice)
xlabel('x3-Pitch angle(deg.)');ylabel('x4-Pitch angular
velocity(deg./s.)');zlabel('x6');
```

```
%ylim([-90 90]);
%zlim([-1e16 1e16]);
xlim([-90 90]);
title(sprintf('Rank of Controllability'));

colorbar
c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';
set(gca,'FontSize', Fn);
```

Appendix A.6.4: MATLAB codes of rank of observability test matrix for an inverted pendulum model

Script file: Observability_IP_4s_x3x4.m

```
clear all
close all

xx1=0;  %set initial cart displacement to be 0 (m)
xx2=0;  %set initial cart velocity to be 0 (m/s)
xx3=0; %set initial pendulum angle to be 0 (rad)
xx4=0; %set initial pendulum angular velocity to be 0 (rad/s)

m1=2; %Mass of the cart (kg)
m2=0.1; %mass of the pendulum (kg)
r=0.5; %the rod length (m)
g=9.8; %acceleration due to gravity (m^2/s)


%%%%%%%%%%%%%%%%%%%%%%%% start controllability test %%%%%%%%%%%%%%%%%%%%%

%%% Variable 'for loop' %%%%

%%%% X3
Minimun_Angle_X3= -90;  %ok
Step_Angle_X3= 0.2;
Maximum_Angle_X3= 90;

%%% X4
Minimun_Angular_Velo_X4= -3e8;
Step_Angular_Velo_X4= 1e6;
Maximum_Angular_Velo_X4= 3e8;


%%% Calculation size of X4
 Size_x4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4:
Maximum_Angular_Velo_X4;
 [m_size,n_size]= size(Size_x4);

%%%%% Plot 2D graph %%%%%%%%

ii=2;  % row of array   x3
jj=2;  % column of array x4
```

```
%%% match x3 and x4 %%%
for xx4 = Minimun_Angular_Velo_X4*pi/180 : Step_Angular_Velo_X4*pi/180 :
Maximum_Angular_Velo_X4*pi/180
  for xx3 = Minimun_Angle_X3*pi/180 : Step_Angle_X3*pi/180 :
Maximum_Angle_X3*pi/180


     if xx3 == 0
        xx3=1.0e-100;  % avoid Inf's and NaN's
     end
       if xx4 == 0
        xx4=1.0e-100;  % avoid Inf's and NaN's
     end


%%% Nonlinear model
A4=[0, 1, 0, 0;
0, 0,-m2*g*sin(xx3)*cos(xx3)/((m1+m2*(sin(xx3))^2)*xx3),...
m2*r*xx4*sin(xx3)/(m1+m2*(sin(xx3)^2));
0, 0, 0, 1;
0, 0,(m1+m2)*g*sin(xx3)/(r*(m1+m2*(sin(xx3))^2)*xx3),...
-m2*r*xx4*sin(xx3)*cos(xx3)/(r*(m1+m2*(sin(xx3))^2))];

B4=[0; 1/(m1+m2*(sin(xx3))^2); 0 ; -cos(xx3)/(r*(m1+m2*(sin(xx3))^2))];


     C4 = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];

Rank_x3x4(ii,jj)= rank(obsv(A4,C4));

Rank_x3x4(ii,1)= xx3*180/pi;  % Insert 1st column by x3
Rank_x3x4(1,jj)= xx4*180/pi;  % Insert 1st row by x4


 ii=ii+1; %x3
  end
  ii=2;
  jj=jj+1; %x4


  %%% print for waiting %%%
    Cal_percent = int16(jj*100/(n_size-1));


    if mod(Cal_percent , 10) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end


  end

 %%%% Remove 1st row & 1st column from table for plotting %%%
  Rank_x3x4_data = Rank_x3x4;
  Rank_x3x4_data(1,:)=[];  % Remove title row 1 ( x4 name)
  Rank_x3x4_data(:,1)=[];  % Remove title column 1 ( x3  name)
  Rank_x3x4_data = Rank_x3x4_data'; % Transpose matrix x3=x-axixs, x4=y-axis


 %%%%% plot max %%%%%
 figure(); % x3 x4
 Fn = 14; % font size
 xx4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4 :
Maximum_Angular_Velo_X4;
 xx3 = Minimun_Angle_X3 : Step_Angle_X3 : Maximum_Angle_X3;
```

```
[x_mesh,y_mesh] = meshgrid(xx3,xx4);  % This generates the actual grid of x
and y values.
[Mc,c]=contourf(x_mesh,y_mesh,Rank_x3x4_data);
set(c,'Linecolor','none')

xlabel('x3 (deg.)');ylabel('x4 (deg./s)');
ylim([Minimun_Angular_Velo_X4 Maximum_Angular_Velo_X4]);
set(gca,'FontSize', Fn);
title(sprintf('Rank of Observability Test Matrix'));

colorbar
 c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
%c =colorbar;
c.Label.String = 'Rank';
grid on;
```

Appendix A.6.5: MATLAB codes of rank of controllability test matrix for a TWR model

Script file: Observability_TWR_5s_x3x4.m

```
clear all
close all

xx1=0;  %set initial wheel angle to be 0 (m)
xx2=0;  %set initial wheel angular velocity to be 0 (m/s)
xx3=0; %set initial pitch angle to be 0 (rad)
xx4=0; %set initial pitch angular velocity to be 0 (rad/s)

%%% Physical constants
g = 9.81;                        % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                       % wheel weight [kg]
R = 0.027;                       % wheel radius [m]
Jw = m * R^2 / 2;                % wheel inertia moment [kgm^2]
W = 0.105;                       % body width [m]
D = 0.1;                         % body depth [m]

M = 0.64;                       % body weight [kg]


%%% EV3 Motors parameters
Jm = 1e-5;                       % DC motor inertia moment [kgm^2]
Rm = 6.69;                       % DC motor resistance [Om]
Kb = 0.468;                      % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                      % DC motor torque constant [Nm/A]

h = 0.210;                       % body height [m]
L = h / 2;                       % distance of the center of mass from the
wheel axle [m] 10.5
Jpsi = M * L^2 / 3;              % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;     % body yaw inertia moment [kgm^2]

n = 1;                           % Gear ratio
fm = 0.0022;                     % friction coefficient between body & DC motor
fw = 0;                          % friction coefficient between wheel & floor
```

```
    %%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;


%%%%%%%%%%%%%%%%%%%%%%%% start controllability test %%%%%%%%%%%%%%%%%%%%

%%% Variable 'for loop' %%%%

%%%% X3
Minimun_Angle_X3= -90;
Step_Angle_X3= 0.2;
Maximum_Angle_X3= 90;
%%%% X4
Minimun_Angular_Velo_X4= -1e6;
Step_Angular_Velo_X4= 0.2e4;
Maximum_Angular_Velo_X4= 1e6;


%%% Calculation size of X4
 Size_x4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4:
Maximum_Angular_Velo_X4;
 [m_size,n_size]= size(Size_x4);

%%%%% Plot 2D graph %%%%%%%%

ii=2;  % row of array   x3
jj=2;  % column of array x4

%%% match x3 and x4 %%%
for xx4 = Minimun_Angular_Velo_X4*pi/180 : Step_Angular_Velo_X4*pi/180 :
Maximum_Angular_Velo_X4*pi/180
  for xx3 = Minimun_Angle_X3*pi/180 : Step_Angle_X3*pi/180 :
Maximum_Angle_X3*pi/180

     if xx3 == 0
        xx3=1.0e-100;  % avoid Inf's and NaN's
     end
       if xx4 == 0
        xx4=1.0e-100;  % avoid Inf's and NaN's
     end


 %%% Dr Xu Xu Model %%%


a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(xx3)^2+4*M*R*L*cos(xx3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(xx3)-M^2*R*L^2*cos(xx3)*g*sin(xx3);
    e24=xx4*sin(xx3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(xx3);
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x2dot equation

    e43=M*g*L*sin(xx3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=xx4*sin(xx3)*(-M^2*R^2*L^2*cos(xx3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(xx3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
```

```
    %matrices for the x4dot equation

    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);


    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);


    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;



        %%%% Nonlinear model %%%%
      A5 =[0      1             0               0       0  ;
           0  em22/(a+b)  em23/((a+b)*xx3)  em24/(a+b)  0 ;
           0      0             0               1       0 ;
           0  em42/(a+b)  em43/((a+b)*xx3)  em44/(a+b)  0 ;
           1      0             0               0       0 ];


     B5 =[    0           0;
        fm21/(a+b)  fm22/(a+b);
            0           0;
        fm41/(a+b)  fm42/(a+b);
          0             0];

     C5 = [1 0 0 0 0; 0 1 0 0 0;0 0 1 0 0;0 0 0 1 0; 0 0 0 0 1];


Rank_x3x4(ii,jj)= rank(obsv(A5,C5));

Rank_x3x4(ii,1)= xx3*180/pi;  % Insert 1st column by x3
Rank_x3x4(1,jj)= xx4*180/pi;  % Insert 1st row by x4


 ii=ii+1; %x3

  end
  ii=2;

  jj=jj+1; %x4


  %%% print for waiting %%%
    Cal_percent = int16(jj*100/(n_size-1));


    if mod(Cal_percent , 10) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end

end

 %%%% Remove 1st row & 1st column from table for plotting %%%
```

```
  Rank_x3x4_data = Rank_x3x4;
  Rank_x3x4_data(1,:)=[];  % Remove title row 1 ( x4 name)
  Rank_x3x4_data(:,1)=[];  % Remove title column 1 ( x3  name)
  Rank_x3x4_data = Rank_x3x4_data'; % Transpose matrix x3=x-axixs, x4=y-axis


 %%%%% plot max %%%%%
 figure(); % x3 x4
 Fn = 14; % font size
 xx4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4 :
Maximum_Angular_Velo_X4;
 xx3 = Minimun_Angle_X3 : Step_Angle_X3 : Maximum_Angle_X3;

[x_mesh,y_mesh] = meshgrid(xx3,xx4);  % This generates the actual grid of x
and y values.
[Mc,c]=contourf(x_mesh,y_mesh,Rank_x3x4_data);
set(c,'Linecolor','none')

xlabel('x3 (deg.)');ylabel('x4 (deg./s)');
xlim([-90 90]);
ylim([Minimun_Angular_Velo_X4 Maximum_Angular_Velo_X4]);
set(gca,'FontSize', Fn);
title(sprintf('Rank of Observability Test Matrix'));

colorbar
 c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';
grid on;
```

Appendix A.6.6: MATLAB codes of rank of observability test matrix for a TWR model

Script file: Observability_TWR_5s_x3x4_saturation.m

```
clear all
close all

xx1=0;  %set initial wheel angle to be 0 (m)
xx2=0;  %set initial wheel angular velocity to be 0 (m/s)
xx3=0; %set initial pitch angle to be 0 (rad)
xx4=0; %set initial pitch angular velocity to be 0 (rad/s)

xx6=0; %set initial artificial control to be 0


%%% Physical constants
g = 9.81;                        % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                       % wheel weight [kg]
R = 0.027;                       % wheel radius [m]
Jw = m * R^2 / 2;                % wheel inertia moment [kgm^2]
W = 0.105;                       % body width [m]
D = 0.1;                         % body depth [m]

M = 0.64;                       % body weight [kg]
```

```
h = 0.210;                          % body height [m]
L = h / 2;                          % distance of the center of mass from the
wheel axle [m] 10.5

Jpsi = M * L^2 / 3;                 % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;        % body yaw inertia moment [kgm^2]


%%% Motors parameters
Jm = 1e-5;                          % DC motor inertia moment [kgm^2]
Rm = 6.69;                          % DC motor resistance [Om]
Kb = 0.468;                         % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                         % DC motor torque constant [Nm/A]
n = 1;                              % Gear ratio
fm = 0.0022;                        % friction coefficient between body & DC motor
fw = 0;                             % friction coefficient between wheel & floor


    %%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;


%%%%%%%%%%%%%%%%%%%%%%%% start controllability test %%%%%%%%%%%%%%%%%%%%

%%% Variable 'for loop' %%%%

%%% X3
Minimun_Angle_X3= -90;
Step_Angle_X3= 2;
Maximum_Angle_X3= 90;

%%% X4
Minimun_Angular_Velo_X4= -1e5;
Step_Angular_Velo_X4= 2e3;
Maximum_Angular_Velo_X4= 1e5;

%%%%% X6
Minimun_x6= -1e20;
Step_x6= 2e18;
Maximum_x6= 1e20;

 %%% Calculation size of Xnp1
 Size_x6 = Minimun_x6 : Step_x6: Maximum_x6;
 [m_x6,n_x6]= size(Size_x6);

%%%%% Plot 3D graph %%%%%%%%

ii=2;   % row of array    x3
jj=2;   % column of array x4
kk=1;   % page Xn+1


%%% convert deg to rad
%%% match x3 and x4 %%%
 for xx6 = Minimun_x6 : Step_x6: Maximum_x6
 for xx3 = Minimun_Angle_X3*pi/180 : Step_Angle_X3*pi/180 :
Maximum_Angle_X3*pi/180
 for xx4 = Minimun_Angular_Velo_X4*pi/180 : Step_Angular_Velo_X4*pi/180 :
Maximum_Angular_Velo_X4*pi/180

     if xx3 == 0
        xx3=1.0e-20;  % avoid Inf's and NaN's
     end
     if xx4 == 0
```

```
        xx4=1.0e-20;  % avoid Inf's and NaN's
     end
     if xx6 == 0
        xx6=1.0e-20;  % avoid Inf's and NaN's
     end


 %%% Dr Xu Xu model %%%


a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(xx3)^2+4*M*R*L*cos(xx3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(xx3)-M^2*R*L^2*cos(xx3)*g*sin(xx3);
    e24=xx4*sin(xx3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(xx3);
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x2dot equation

    e43=M*g*L*sin(xx3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=xx4*sin(xx3)*(-M^2*R^2*L^2*cos(xx3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(xx3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation

    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);


    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);


    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;


        %%%% Nonlinear Model %%%%
      A5 =[0      1             0              0       0  ;
           0  em22/(a+b)  em23/((a+b)*xx3)  em24/(a+b)  0 ;
           0      0             0              1       0 ;
           0  em42/(a+b)  em43/((a+b)*xx3)  em44/(a+b)  0 ;
           1      0             0              0       0 ];


     B5 =[    0           0;
        fm21/(a+b)  fm22/(a+b);
            0           0;
        fm41/(a+b)  fm42/(a+b);
          0             0];

     %%%%--------------
```

```
      Vmax = 8.3;  % Set maximum saturation voltage

     if      (xx6 )  > Vmax  % Lego Motor Maximum Voltage 8.3 V
         Phi_L = Vmax;
         Phi_R = Vmax;
     elseif  (xx6) < -Vmax % Lego Motor Minimum Voltage -8.3 V
         Phi_L = -Vmax;
         Phi_R = -Vmax;
     else


         Phi_L = Vmax*sin((pi*xx6)/(2*Vmax));
         Phi_R = Vmax*sin((pi*xx6)/(2*Vmax));
     end

     Phi = [Phi_L ;
            Phi_R ];


Aa= [       A5             (B5*Phi)/xx6   ;
       zeros(1,5)             0              ];

Ba= [  zeros(5,1)   ;
          1         ];

Ca= eye(6);

Rank_x3x4x6(ii,jj,kk)= rank(obsv(Aa,Ca));

%%% create table
Rank_x3x4x6(ii,1,kk)= xx4*180/pi;  % Insert 1st column by x3
Rank_x3x4x6(1,jj,kk)= xx3*180/pi;  % Insert 1st row    by x4


    if xx4 >-0.01 && xx4 <0.01
        kk=kk+1;
    Rank_x3x6(jj,kk) = rank(obsv(Aa,Ca));
    Rank_x3x6(jj,1) = xx3*180/pi;  % Insert 1st row by  x3
    Rank_x3x6(1,kk) = xx6; % Insert 1st column by xpn1
        kk=kk-1;
    end

    if single(xx3)==single(1.0e-20)
         kk=kk+1;
    Rank_x4x6(ii,kk) = rank(obsv(Aa,Ca));
    Rank_x4x6(ii,1) = xx4*180/pi;  % Insert 1st row by x4
    Rank_x4x6(1,kk) = xx6; % Insert 1st column by xpn1
         kk=kk-1;
    end


     if single(xx6)==single(1.0e-20)
    Rank_x3x4(ii,jj) = rank(obsv(Aa,Ca));
    Rank_x3x4(ii,1) = xx4*180/pi;  % Insert 1st row by x4
    Rank_x3x4(1,jj) = xx3*180/pi; % Insert 1st column by xpn1
    end


 ii=ii+1; %x3
  end
  ii=2;
  jj=jj+1; %x4
 end
```

```
     if xx6 == 1.0e-20
        xx6=0;  % move zero back
     end

    Range_x6(1,kk)=xx6; % store x_np1

  jj=2;
  kk=kk+1; %x_np1

    %%% print for writing  %%%
    Cal_percent = int16(kk*100/(n_x6+1));

    if mod(Cal_percent , 10) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end

end


  %%%% Remove 1st row & 1st column from table for plotting %%%
  Rank_x3x4x6_data = Rank_x3x4x6;
  Rank_x3x4x6_data(1,:,:)=[];  % Remove title row 1 ( x4 name)
  Rank_x3x4x6_data(:,1,:)=[];  % Remove title column 1 ( x3  name)

  Rank_x3x6_data = Rank_x3x6;
  Rank_x3x6_data(1,:)=[];  % Remove title row 1 ( xnp1 name)
  Rank_x3x6_data(:,1)=[];  % Remove title column 1

  Rank_x4x6_data = Rank_x4x6;
  Rank_x4x6_data(1,:)=[];  % Remove title row 1 ( xnp1 name)
  Rank_x4x6_data(:,1)=[];  % Remove title column 1

  Rank_x3x4_data = Rank_x3x4;
  Rank_x3x4_data(1,:)=[];  % Remove title row 1 ( xnp1 name)
  Rank_x3x4_data(:,1)=[];  % Remove title column 1


 xx3 = Minimun_Angle_X3 : Step_Angle_X3 : Maximum_Angle_X3;
 xx4 = Minimun_Angular_Velo_X4 : Step_Angular_Velo_X4 :
Maximum_Angular_Velo_X4;
 xx6 = Minimun_x6 : Step_x6: Maximum_x6;


%%Plot 2D at x3=0 %%%%
 figure();
 Fn = 12; % font size
[x_mesh,y_mesh,z_mesh] = meshgrid(xx3,xx4,xx6);  % This generates the actual
grid of x and y values.

xslice = [0];
yslice = [];
zslice = [];

slice(x_mesh,y_mesh,z_mesh,Rank_x3x4x6_data,xslice,yslice,zslice)
xlabel('x3-Pitch angle(deg.)');ylabel('x4-Pitch angular
velocity(deg./s.)');zlabel('x6');
xlim([-90 90]);
title(sprintf('Rank of Observability Test Matrix'));

colorbar
c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';

%%%% view %%%%
```

```
view(90,0) % y-view
%%%%%%
set(gca,'FontSize', Fn);



%%Plot 2D at x4=0 %%%%
 figure(); %x4=0
 Fn = 12; % font size
[x_mesh,y_mesh,z_mesh] = meshgrid(xx3,xx4,xx6);  % This generates the actual
grid of x and y values.


xslice = [];
yslice = [0];
zslice = [];

slice(x_mesh,y_mesh,z_mesh,Rank_x3x4x6_data,xslice,yslice,zslice)
xlabel('x3-Pitch angle(deg.)');ylabel('x4-Pitch angular
velocity(deg./s.)');zlabel('x6');
xlim([-90 90]);
title(sprintf('Rank of Observability Test Matrix'));

colorbar
c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';

%%%% view %%%%
    view(0,0)  % x-view
%%%%%%
set(gca,'FontSize', Fn);



%%Plot Cubic %%%%
 figure(); %
 Fn = 12; % font size
[x_mesh,y_mesh,z_mesh] = meshgrid(xx3,xx4,xx6);  % This generates the actual
grid of x and y values.

xslice = [xx3];
yslice = [xx4];
zslice = [xx6];

slice(x_mesh,y_mesh,z_mesh,Rank_x3x4x6_data,xslice,yslice,zslice)
xlabel('x3-Pitch angle(deg.)');ylabel('x4-Pitch angular
velocity(deg./s.)');zlabel('x6');
%ylim([-90 90]);
%zlim([-1e16 1e16]);
xlim([-90 90]);
title(sprintf('Rank of Observability Test Matrix'));

colorbar
c = colorbar('Ticks',[1,2,3,4,5,6],...
         'TickLabels',{'1','2','3','4','5','6'});
c.Label.String = 'Rank';
set(gca,'FontSize', Fn);
```

Appendix A.6.7: MATLAB codes of an inverted pendulum on a cart system using nonlinear freezing control

Script file: *Freezing_single_pendulum_4s.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Nonlinear Freezing control of an inverted pendulum on a cart system

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

clear all;
close all;

m1=2; %Mass of the cart (kg)
m2=0.1; %mass of the pendulum (kg)
r=0.5; %the rod length (m)
g=9.8; %acceleration due to gravity (m^2/s)

x1(1)=0; %set initial cart displacement to be 0 (m)
x2(1)=0; %set initial cart velocity to be 0 (m/s)
x3(1)=30*pi/180; %set initial pendulum angle to be pi (rad)
x4(1)=0; %set initial pendulum angular velocity to be 0 (rad/s)

u_out(1)=0; % Control signal

Ts = 0.0001;  % step siz
Duration=10;  % 10 sec
t=Duration*(1/Ts);
Time = 0:Ts:t*Ts;  % the range of x-axis

%set Q and R matrices
Q=[1, 0, 0,  0;
   0, 1, 0,  0;
   0, 0, 100, 0;
   0, 0, 0, 1];
R=0.01;


for i=1:t
x=[x1(i); x2(i); x3(i); x4(i)];
%define and update the x vector

%%% Nonlinear model from Xu' book chapter %%%%
x_3=x(3); x_4=x(4);

A=[0, 1, 0, 0;
0, 0,-m2*g*sin(x_3)*cos(x_3)/((m1+m2*(sin(x_3))^2)*x_3),...
m2*r*x_4*sin(x_3)/(m1+m2*(sin(x_3)^2));
0, 0, 0, 1;
0, 0,(m1+m2)*g*sin(x_3)/(r*(m1+m2*(sin(x_3))^2)*x_3),...
-m2*r*x_4*sin(x_3)*cos(x_3)/(r*(m1+m2*(sin(x_3))^2))];

B=[0; 1/(m1+m2*(sin(x_3))^2); 0 ; -cos(x_3)/(r*(m1+m2*(sin(x_3))^2))];


%set Q and R matrices

[~,P,~]=lqr(A,B,Q,R);
%use the MATLAB 'lqr' function to solve Riccati equation and
```

```
%work out P

u_out=(-inv(R)*B'*P)*x;

fx=(A-B*(1/R)*B'*P)*x;
%calculate the function output 'fx' based on values of A, B, P
%and x.
 x = x + Ts * fx;    % Euler

    %%% Limit the pitch angle between -90 to 90 deg.
    if x(3) > 90*pi/180
        x(3) =90*pi/180;
    end
     if x(3) < -90*pi/180
        x(3) =-90*pi/180;
     end



x1(i+1)=x(1);
x2(i+1)=x(2);
x3(i+1)=x(3);
x4(i+1)=x(4);
%Reset the x1, x2, x3 & x4 variables to new values and get ready
%for the next iteration.

u(i+1) = u_out;
      %%% print %%%
    Cal_percent = i*100/(t);
    if mod(Cal_percent , 10) == 0
        fprintf('Calculating  %f percent ...\n',Cal_percent)
    end



end


figure('Name','Freezing Control');
Fn = 14; % font size

subplot(2,3,1);
p1=plot(Time,x1); grid;
xlabel('Time (sec)'); ylabel('Cart displacement x1 (m)');
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2=plot(Time,x2); grid;
xlabel('Time (sec)'); ylabel('Cart velocity x2 (m/s)');
set(gca,'FontSize', Fn);

subplot(2,3,2);
p3=plot(Time,x3*180/pi); grid;
xlabel('Time (sec)'); ylabel('Pendulum angle x3 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,5);
p4=plot(Time,x4*180/pi); grid;
xlabel('Time (sec)'); ylabel('Pendulum angular velocity x4 (deg/s)');
set(gca,'FontSize', Fn);


subplot(2,3,3);
p5=plot(Time,u); grid;
xlabel('Time (sec)'); ylabel('Control signal-u (N)');
```

```
set(gca,'FontSize', Fn);



Size=1.2;
p1.LineWidth = Size;
p2.LineWidth = Size;
p3.LineWidth = Size;
p4.LineWidth = Size;
p5.LineWidth = Size;
```

Appendix A.6.8: MATLAB codes of an inverted pendulum on a cart system using iteration scheme

Script file: *Iteration_single_pendulum_4s.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Nonlinear iteration scheme of an inverted pendulum on a cart system
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all


m1=2; %Mass of the cart (kg)
m2=0.1; %mass of the pendulum (kg)
r=0.5; %the rod lenghth (m)
g=9.8; %acceleration due to gravity (m^2/s)

x1(1)=0; %set initial cart displacement to be 0
x2(1)=0; %set initial cart velocity to be 0
x3(1)=60*pi/180; %set initial pendulum angle to be pi/3
x4(1)=0; %set initial pendulum angular velocity to be 0
x_initial=[x1(1); x2(1); x3(1); x4(1)]; %define  the x vector


Ts = 0.00005;  % step size
Duration=10;   % 10sec
t=Duration*(1/Ts);
Time = 0:Ts:t*Ts;  % the range of x-axis

data = zeros(size(Time));  % allocate the result x-axis
n = numel(data);  % the number of x-axis values


u(1)=0;


%set Q and R matrices
Q=[1, 0, 0, 0;
    0, 1, 0, 0;
    0, 0, 100, 0;
    0, 0, 0, 1];
R=0.01;
```

```
F=eye(4);    % F = diag{1,1,1,1}
P_final=F;  % Final time penelty matrix

Iteration = 40;
for k=1:Iteration  %%%%%%%%%%%% iteration for loop %%%%%%%%%%%%

    x = x_initial;  % set x=x0 at the initial step
    P = P_final;

    for i=1:t %%%% Time step %%%

            %%%%%%%% evalrhs function start %%%%%%%%

            %%%% At initial step K=1
            if k==1
              x3_in = x_initial(3);
              x4_in = x_initial(4);

              %%% Nonlinear Model from Xu's book chapter %%%
               A=[0, 1, 0, 0;
                    0, 0,-
m2*g*sin(x3_in)*cos(x3_in)/((m1+m2*(sin(x3_in))^2)*x3_in),...
                   m2*r*x4_in*sin(x3_in)/(m1+m2*(sin(x3_in)^2));
                   0, 0, 0, 1;
                   0,
0,(m1+m2)*g*sin(x3_in)/(r*(m1+m2*(sin(x3_in))^2)*x3_in),...
                   -
m2*r*x4_in*sin(x3_in)*cos(x3_in)/(r*(m1+m2*(sin(x3_in))^2))];

              B=[0; 1/(m1+m2*(sin(x3_in))^2); 0 ; -
cos(x3_in)/(r*(m1+m2*(sin(x3_in))^2))];

            end

             %%%% When K > 1 %%%
             if k > 1
              x3_pre = x3_table(i,k-1);
              x4_pre = x4_table(i,k-1);

              %%% Nonlinear Model from Xu's book chapter %%%
              A=[0, 1, 0, 0;
                    0, 0,-
m2*g*sin(x3_pre)*cos(x3_pre)/((m1+m2*(sin(x3_pre))^2)*x3_pre),...
                   m2*r*x4_pre*sin(x3_pre)/(m1+m2*(sin(x3_pre)^2));
                   0, 0, 0, 1;
                   0,
0,(m1+m2)*g*sin(x3_pre)/(r*(m1+m2*(sin(x3_pre))^2)*x3_pre),...
                   -
m2*r*x4_pre*sin(x3_pre)*cos(x3_pre)/(r*(m1+m2*(sin(x3_pre))^2))];

              B=[0; 1/(m1+m2*(sin(x3_pre))^2); 0 ; -
cos(x3_pre)/(r*(m1+m2*(sin(x3_pre))^2))];

             end


          %%%% Backward Euler's method %%%
           P = P-(- Q - P*A - A.'*P + P*B*inv(R)*B.'*P) *Ts;

            u_new = -inv(R)*B'*P*x;

            fx = (A-B*inv(R)*B'*P)*x;
```

```
            x = x + Ts * fx; % Euler's method


     %%% Limit the pitch angle between -90 to 90 deg.
    if x(3) > 90*pi/180
        x(3) =90*pi/180;
    end
     if x(3) < -90*pi/180
        x(3) =-90*pi/180;
     end


        x1(i+1) = x(1); %%% Feedback to system
        x2(i+1) = x(2);
        x3(i+1) = x(3);
        x4(i+1) = x(4);


    u(i+1) = u_new;      %%% plot data only

    end

    x1_table(:,k)=x1;  %%% collect data, creating table
    x2_table(:,k)=x2;
    x3_table(:,k)=x3;
    x4_table(:,k)=x4;

    u_table(:,k)=u;

    %%% print %%%
    Cal_percent = k*100/(Iteration);
    if mod(Cal_percent , 10) == 0
        fprintf('Calculating  %f percent ...\n',Cal_percent)
    end

end


x3_table=x3_table*180/pi;
x4_table=x4_table*180/pi;


figure('Name','Iteration scheme');
Fn = 14; % font size
subplot(2,3,1);

p1(1)=plot(Time,x1_table(:,1),'k--'); hold on
p1(2)=plot(Time,x1_table(:,5),'m'); hold on
p1(3)=plot(Time,x1_table(:,10),'color','#F39C12'); hold on
p1(4)=plot(Time,x1_table(:,15),'g:'); hold on
p1(5)=plot(Time,x1_table(:,30),'b'); hold on
p1(6)=plot(Time,x1_table(:,40),'r--'); hold on
xlabel('Time (sec)'); ylabel('Cart displacement x1 (m)');grid;
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2(1)=plot(Time,x1_table(:,1),'k--'); hold on
p2(2)=plot(Time,x2_table(:,5),'m'); hold on
p2(3)=plot(Time,x2_table(:,10),'color','#F39C12'); hold on
p2(4)=plot(Time,x2_table(:,15),'g:'); hold on
p2(5)=plot(Time,x2_table(:,30),'b'); hold on
p2(6)=plot(Time,x2_table(:,40),'r--'); hold on
xlabel('Time (sec)'); ylabel('Cart velocity x2 (m/s)');grid;
set(gca,'FontSize', Fn);
```

```
subplot(2,3,2);
%plot(Time,x1); grid;
p3(1)=plot(Time,x3_table(:,1),'k--'); hold on
p3(2)=plot(Time,x3_table(:,5),'m'); hold on
p3(3)=plot(Time,x3_table(:,10),'color','#F39C12'); hold on
p3(4)=plot(Time,x3_table(:,15),'g:'); hold on
p3(5)=plot(Time,x3_table(:,30),'b'); hold on
p3(6)=plot(Time,x3_table(:,40),'r--'); hold on
xlabel('Time (sec)'); ylabel('Pendulum angle x3 (deg)');grid;
set(gca,'FontSize', Fn);

subplot(2,3,5);
p4(1)=plot(Time,x4_table(:,1),'k--'); hold on
p4(2)=plot(Time,x4_table(:,5),'m'); hold on
p4(3)=plot(Time,x4_table(:,10),'color','#F39C12'); hold on
p4(4)=plot(Time,x4_table(:,15),'g:'); hold on
p4(5)=plot(Time,x4_table(:,30),'b'); hold on
p4(6)=plot(Time,x4_table(:,40),'r--'); hold on
xlabel('Time (sec)'); ylabel('Pendulum angular velocity x4 (deg/s)');grid;
set(gca,'FontSize', Fn);

subplot(2,3,3);
p5(1)=plot(Time,u_table(:,1),'k--'); hold on
p5(2)=plot(Time,u_table(:,5),'m'); hold on
p5(3)=plot(Time,u_table(:,10),'color','#F39C12'); hold on
p5(4)=plot(Time,u_table(:,15),'g:'); hold on
p5(5)=plot(Time,u_table(:,30),'b'); hold on
p5(6)=plot(Time,u_table(:,40),'r--'); hold on
xlabel('Time (sec)'); ylabel('Control signal-u (N)');grid;
set(gca,'FontSize', Fn);

legend('i=1','i=5','i=10','i=15','i=30','i=40','FontSize', 12)


LW1=1; LW2=1.2; LW3=1.2; LW4=2.5; LW5=1.2;  LW6=2;

p1(1).LineWidth = LW1; p1(2).LineWidth = LW2; p1(3).LineWidth = LW3;
p1(4).LineWidth = LW4; p1(5).LineWidth = LW5; p1(6).LineWidth = LW6;
p2(1).LineWidth = LW1; p2(2).LineWidth = LW2; p2(3).LineWidth = LW3;
p2(4).LineWidth = LW4; p2(5).LineWidth = LW5; p2(6).LineWidth = LW6;
p3(1).LineWidth = LW1; p3(2).LineWidth = LW2; p3(3).LineWidth = LW3;
p3(4).LineWidth = LW4; p3(5).LineWidth = LW5; p3(6).LineWidth = LW6;
p4(1).LineWidth = LW1; p4(2).LineWidth = LW2; p4(3).LineWidth = LW3;
p4(4).LineWidth = LW4; p4(5).LineWidth = LW5; p4(6).LineWidth = LW6;
p5(1).LineWidth = LW1; p5(2).LineWidth = LW2; p5(3).LineWidth = LW3;
p5(4).LineWidth = LW4; p5(5).LineWidth = LW5; p5(6).LineWidth = LW6;
```

Appendix A.6.9: MATLAB codes of TWR systems using freezing control technique

Script file: *Gyroboy_5s_Freezing_10_2021.m*

```
%%%% Nonlinear Freezing Control for LEGO EV3 Robot


%%%%%%  Functions programme needed %%%
% evalrhs_Freezing_K5();         % Generating K5 only
```

```
% evalrhs_gyroboy5s_Freezing();  % Generating K1-K4 and fx
%
%      Inside two functions
%
%           Gyroboy_Nonlinear_Model_5s();
%           Gyroboy_Nonlinear_Model_4s();
%           Maxon Motor parameters etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all;
close all;

%%% Set initial x1-x5
%%% Always set x1=x3

x1(1)=20*pi/180;      %set Theta - Average of wheel angles (deg)
x2(1)=0;                %set Theta_DOT (deg/s)
x3(1)=20*pi/180; %set Psi - Pitch angle of body (deg) - CONTROL to ZERO
x4(1)=0*pi/180;         %set Psi_DOT (deg/s)
x5(1)=0;                %set Theta integral

  %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed

  Model=1;

u = [0;0];   % Control of Motors
u_in1(1)=0;  % Left  Motor Voltage
u_in2(1)=0;  % Right Motor Voltage
u_feedback =[0;0]; % u_feedback = Kx = (K_LQR) x (X1-X4)

x1_err = 0;      % x1 error for tracking sys
x1_ref = 0;      % x1 reference
x1_err_int(1)=0; % x1 error integral

Ts=0.0001; %time step length

Duration=10;       % time sec
t=Duration*(1/Ts); % time step
Time = 0:Ts:t*Ts;   % Create real time step for plotting

for i=1:t
  u      =[u_in1(i); u_in2(i)];
  x_5s    =[x1(i); x2(i); x3(i); x4(i); x5(i)];  % whole system
  x_4s    =[x1(i); x2(i); x3(i); x4(i)];         % x1-x4 state feedback


  %%%%%% Programming Diagram %%%%%

%
%                    u_x1  +     u                              y=x1-x4            y=x1
%  X1ref->-o-->-Int--K5-->-->o---------------+-------| Plant |-->----------------+---C------+
%         ^-                 ^ -                                                  |          |
%         |                  | u_feedback                                        |          |
%         |                  |                                                   |
%         x1                 |                                                   |
%                            +--------------------<--K14-- ---------------- ---+
%                                                          x1-x4
%
% %%%%%%%%%%%%%%%


    %%% x1 error for tracking sys
    x1_err(i+1) = x1_ref - x1(i);
```

```
      %%%%% X1 error Integral  %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

    %%% x1 error integral
    x1_err_int(i+1) = x1_err_int(i) + x1_err(i)*Ts + 0.5*(x1_err(i+1)-
x1_err(i))*Ts ;  % [h1 x L]+[ 0.5 x(h2-h1) x L ] + old

    %%% cal. K5 for x5 only (integral x1)
    K_5 = evalrhs_Freezing_K5(x_5s,Model);   % K_5=[-0.5000; -0.5000];


    %%% u=kx,  Motor Voltage of integral x1
    u_x1 = K_5 * x1_err_int(i+1);

    %%% The final voltages used to control robot motors
    u = u_x1 - u_feedback;


   %%% Hard Saturation  %%%
   %%% Uncomment this part for using motor voltage hard saturation
%     Vmax = 8.3;
%     %Vmax = 36;
%     %Vmax = 48;
%             if u(1) > Vmax
%                u(1) = Vmax;
%                u(2) = Vmax;
%             end
%              if u(1) < -Vmax
%                u(1) = -Vmax;
%                u(2) = -Vmax;
%              end
%     %%%%%%


    u_in1(i+1)=u(1);   % update u1
    u_in2(i+1)=u(2);   % update u2


    %%% calculate the new 'x' vector using a 4th order
    %%% Euler integration method
    %%% fx = Ax + Bu;
    [fx,K_14,~,~] = evalrhs_gyroboy5s_Freezing(x_4s,u,Model);
    x_4s = x_4s + Ts * fx;   % Euler

    %%% K14 = K1,K2,K3 and K4


    %%% Limit the robot pitch angle between -90 to 90 deg.
    if x_4s(3) > 90*pi/180
        x_4s(3) =90*pi/180;
    end
     if x_4s(3) < -90*pi/180
        x_4s(3) =-90*pi/180;
     end

    %%% Update x1-x4
    x1(i+1)=x_4s(1);
    x2(i+1)=x_4s(2);
    x3(i+1)=x_4s(3);
    x4(i+1)=x_4s(4);


    x3_deg(i+1)=x_4s(3)*180/pi;
    x4_deg(i+1)=x_4s(4)*180/pi;
```

```
      %%% Generate real x1 integral (x5)
      %%%%% X5 = Integral of X1 %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

      x5(i+1) = x5(i) + x1(i)*Ts + 0.5* (x1(i+1)-x1(i))* Ts;


   %%% x1-x4 motor voltage feedback %%%%
   u_feedback = K_14*[x_4s(1); x_4s(2); x_4s(3); x_4s(4)];


    %%% Command print for waiting %%%
    Cal_percent = i*100/t;

     if i== 1
       fprintf('Start at x3= %3.2f deg, x4= %1.1e
deg/s...\n',x3(1)*180/pi,x4(1)*180/pi)
     end

     if mod(Cal_percent , 2) == 0
       %fprintf('Calculating  %f percent ...\n',Cal_percent)
       fprintf('Cal.  %3.2f percent,  ',Cal_percent)
       fprintf('x3= %3.2f deg,  ',x3(i+1)*180/pi)
       fprintf('x4= %3.2f deg/s,  ',x4(i+1)*180/pi)
       fprintf('t= %3.2f sec...\n',Ts*i)
     end

end

% Use below line instead above when ploting the failing system
(uncontrollable)
% Time = 0:Ts:(i-1)*Ts;
 u=u_in1;  % Store as u


%%%%%% Save variables for plotting future %%%
%%% 'x1','x2','x3','x4','x5','u','Time'  %%%

%save('Sim_Output_workspace_nonlinear1','x1','x2','x3','x4','x5','u','Time');

%%%% Fig. [2x3] %%%
figure('Name','Freezing control');
Fn = 14; % font size

subplot(2,3,1);
p1=plot(Time,x1*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta - Wheel angles x1 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2=plot(Time,x2*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta dot x2 (deg/s)');
set(gca,'FontSize', Fn);

subplot(2,3,2);
p3=plot(Time,x3*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi - Pitch of body x3 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,5);
p4=plot(Time,x4*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi dot x4 (deg/s)');
set(gca,'FontSize', Fn);
```

```
 subplot(2,3,3);
 p5=plot(Time,x5*180/pi,'r'); grid;
 xlabel('Time (sec)'); ylabel('Control signal-u (volt)');
 set(gca,'FontSize', Fn);

subplot(2,3,6);  %5states
p6=plot(Time,u,'r'); grid;
xlabel('time (s)','FontSize', Fn); ylabel('u (volts)','FontSize', Fn);
set(gca,'FontSize', Fn);

%LineWidth
LW=1.2;
p1(1).LineWidth = LW;
p2(1).LineWidth = LW;
p3(1).LineWidth = LW;
p4(1).LineWidth =LW;
p5(1).LineWidth = LW;
p6(1).LineWidth =LW;


 ------------------------
```

Function file: evalrhs_Freezing_K5.m

```
function K_5 = evalrhs_Freezing_K5(x,model)

    %%% 5-states Model parameters
    [A5,B5,Q,R2,~,~,~]=Gyroboy_Nonlinear_Model_5s(x,model);


    [K_LQR,~,~]=lqr(A5,B5,Q,R2);   % K nonlinear
    %use the MATLAB 'lqr' function to solve Riccati equation and work out K,P

    K_5    = K_LQR;
    K_5(: , 1:4)=[];    % integral gain x5
    % delete column 1-4th

    %calculate the function output 'fx' based on values of A, B, P and x.
end
------------------------
```

Function file: Gyroboy_Nonlinear_Model_5s.m

```
  function [A5,B5,Q,R2,Vmax,alpha,beta]=Gyroboy_Nonlinear_Model_5s(x,Model)

%%% LEGO EV3 parameters 5-states
%%% Physical constants
g = 9.81;                          % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                         % wheel weight [kg]  old 0.024 new 0.050
R = 0.027;                         % wheel radius [m]
Jw = m * R^2 / 2;                  % wheel inertia moment [kgm^2]
W = 0.105;                         % body width [m]
D = 0.1;                           % body depth [m]

M = 0.64;                         % body weight [kg]
h = 0.210;                         % body height [m]
```

```
L = h / 2;                          % distance of the center of mass from the
wheel axle [m] 10.5

%%% EV3 Motors parameters
Jm = 1e-5;                          % DC motor inertia moment [kgm^2]
Rm = 6.69;                          % DC motor resistance [Om]
Kb = 0.468;                         % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                         % DC motor torque constant [Nm/A]

Vmax=8.3;  % Default Vmax for LEGO EV3

M2=M;      % Use this when want to change new motor or mass

%%% Uncomment below when want to use new motor  %%%

%[M,Jm,Rm,Kb,Kt]=MaxonDCmotor_Ec60flat_100W_48V(M2);  Vmax = 48;

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

Jpsi = M * L^2 / 3;                 % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;        % body yaw inertia moment [kgm^2]

n = 1;                              % Gear ratio
fm = 0.0022;                        % friction coefficient between body & DC motor
fw = 0;                             % friction coefficient between wheel & floor

%%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;

    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT
    %x5=x(5); %

     if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
     end
     if x4 == 0
       x4=1.0e-20;  % avoid Inf's and NaN's
    end


 %%% Nonlinear model %%%%

a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(x3)^2+4*M*R*L*cos(x3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(x3)-M^2*R*L^2*cos(x3)*g*sin(x3);
    e24=x4*sin(x3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(x3);
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x2dot equation

    e43=M*g*L*sin(x3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=x4*sin(x3)*(-M^2*R^2*L^2*cos(x3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(x3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation
```

```
    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);

    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);

    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;

    %define these extra function to reduce the coding complexity of A and B
    %matrices with the motor added in

    %define A and B using a nonlinear state-space gyro robot model,
    %including the motor part

    if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
    end
    if x4 == 0
       x4=1.0e-20;  % avoid Inf's and NaN's
    end


 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed
  % 4.Model C

    % Model = 3;  % Select mode

  %%%-------- Model A ----------
if Model == 1
          %%%% Primary Model A %%%%
      A5 =[0      1             0               0       0  ;
           0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)   0 ;
           0      0             0               1       0 ;
           0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)   0 ;
           1      0             0               0       0 ];
end


%%%-------- Model B ----------
if Model == 2
          %%%% Model B %%%%
     A5 =[0      1             0                       0       0  ;
          0  em22/(a+b)  (em23+em24*x4)/((a+b)*x3)    0       0 ;
          0      0             0                       1       0 ;
          0  em42/(a+b)  (em43+em44*x4)/((a+b)*x3)    0       0 ;
          1      0             0                       0       0 ];

end


  %%%-------- Model AB5 ----------
if Model == 3
```

```
  %%% Mix A&B %%%%%

   if x3 <= (10*pi/180) && x3 >= (-10*pi/180)
    %%%% Primary Model A %%%%
     A5 =[0      1             0              0       0  ;
         0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)   0 ;
         0      0             0              1       0 ;
         0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)   0 ;
         1      0             0              0       0 ];

  else

      %%%% Model B %%%%
     A5 =[0      1             0                      0       0  ;
         0  em22/(a+b)  (em23+em24*x4)/((a+b)*x3)    0       0 ;
         0      0             0                      1       0 ;
         0  em42/(a+b)  (em43+em44*x4)/((a+b)*x3)    0       0 ;
         1      0             0                      0       0 ];
  end

end


%%%-------- Model C ----------
if Model == 4
       %%%% Model C %%%%
     A5 =[0      1             0              0                     0     ;
         0  em22/(a+b)         0         (em23+em24*x4)/((a+b)*x4)   0 ;
         0      0             0              1                      0   ;
         0  em42/(a+b)        0          (em43+em44*x4)/((a+b)*x4)  0  ;
         1      0             0                      0              0 ];

end


    % Models Matrix B

     B5 =[    0           0;
        fm21/(a+b)   fm22/(a+b);
            0           0;
        fm41/(a+b)   fm42/(a+b);
          0             0];


     Q=[20, 0, 0, 0, 0;
          0, 1, 0, 0, 0;
          0, 0, 1, 0, 0;
          0, 0, 0, 1, 0;
          0, 0, 0, 0, 5];

    R2=  10*[1, 0;
             0, 1];
  %  set Q and R matrices

  end
```

Note that function file "*Gyroboy_Nonlinear_Model_4s.m*" is the same programme used in linear control simulation.

Appendix A.6.10: MATLAB codes of TWR systems using freezing control technique with EKF

Script file: *Gyroboy_5s_Freezing_EKF_10_2021.m*

```
%%%% Nonlinear Freezing Control with EKF for LEGO EV3 Robot

%%%%%%  Functions programme needed %%%
% evalrhs_Freezing_K5();              % Generating K5 only
% evalrhs_gyroboy5s_Freezing_EKF();  % Generating K1-K4, fx and Kf
%
%      Inside two functions
%
%           Gyroboy_Nonlinear_Model_5s();
%           Gyroboy_Nonlinear_Model_4s();
%           Maxon Motor parameters etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all;
close all;

%%% Set initial x1-x5
%%% Always set x1=x3

x1(1)=10*pi/180;      %set Theta - Average of wheel angles (deg)
x2(1)=0;                %set Theta_DOT (deg/s)
x3(1)=10*pi/180; %set Psi - Pitch angle of body (deg) - CONTROL to ZERO
x4(1)=0*pi/180;         %set Psi_DOT (deg/s)
x5(1)=0;                %set Theta integral

 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed

  Model=1;

u_in1(1)=0;  % Left  Motor Voltage
u_in2(1)=0;  % Right Motor Voltage
u_feedback =[0;0]; % u_feedback = Kx = (K_LQR) x (X1-X4)

%%% Set initial Xhat
Xhat1(1) = x1(1);
Xhat2(1) = 0;
Xhat3(1) = x3(1);
Xhat4(1) = 0;

%%% Set initial Xhat integral
dXhat1(1) = 0;
dXhat2(1) = 0;
dXhat3(1) = 0;
dXhat4(1) = 0;

%%% Disturbance variables
x1_disturbance(1) = 0;
x2_disturbance(1) = 0;
x3_disturbance(1) = 0;
x4_disturbance(1) = 0;

x1_err = 0;      % x1 error for tracking sys
```

```
x1_ref = 0;      % x1 reference
x1_err_int(1)=0; % x1 error integral

Ts=0.0001; %time step length

Duration=10;       % time sec
t=Duration*(1/Ts); % time step in programming
Time = 0:Ts:t*Ts;  % Create real time step for plotting

for i=1:t

  u      =[u_in1(i); u_in2(i)];
  x_5s    =[x1(i); x2(i); x3(i); x4(i); x5(i)];   % whole system
  x_4s    =[x1(i); x2(i); x3(i); x4(i)];          % x1-x4 state feedback
  Xhat =[Xhat1(i); Xhat2(i); Xhat3(i); Xhat4(i)]; % Xhat1-4 (exclude x5)
  dXhat=[dXhat1(i); dXhat2(i); dXhat3(i); dXhat4(i)]; %Xhat integral




 %%%%%% Programming Diagram %%%%%

%
%                     u_x1  +     u                              y=x1-x4            y=x1
%  X1ref->-o-->-Int--K5-->-->o---------------+-------| Plant |-->----------------+---C-------+
%         ^-                 ^ -            |                                ^   |           |
%         |                  | u_feedback   +-------B------>|         Add   |   |           |
%         |                  |                              |       Distubance  |
%         x1                 |       Xhat            dXhat  |                   |
%                            +-<--K14-- o<--Integal <------ o <-- Kf-------- ---+
%                                       |                   ^       x1-x4
%                                       |                   |
%                                       +---- A -(Kf)C ->--+

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


     %%% x1 error for tracking sys
    x1_err(i+1) = x1_ref - x1(i);

      %%%%% X1 error Integral  %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

   %%% x1 error integral
   x1_err_int(i+1) = x1_err_int(i) + x1_err(i)*Ts + 0.5*(x1_err(i+1)-
x1_err(i))*Ts ;  % [h1 x L]+[ 0.5 x(h2-h1) x L ] + old

    %%% cal. K5 for x5 only (integral x1)
    K_5 = evalrhs_Freezing_K5(x_5s, Model);  % K_5=[-0.5000; -0.5000];

    %%% u=kx,  Motor Voltage of integral x1
    u_x1 = K_5 * x1_err_int(i+1);

    %%% The final voltages used to control robot motors
    u = u_x1 - u_feedback;

    %%% Hard Saturation  %%%
   %%% Uncomment this part for using motor voltage hard saturation
%     Vmax = 8.3;
%     % Vmax = 36;
%     %Vmax = 48;
%             if u(1) > Vmax
%                u(1) = Vmax;
%                u(2) = Vmax;
%             end
%              if u(1) < -Vmax
```

```
%                u(1) = -Vmax;
%                u(2) = -Vmax;
%              end
%     %%%%%%
%
    u_in1(i+1)=u(1);    % update u1
    u_in2(i+1)=u(2);    % update u2


    %%% calculate the new 'x' vector using a 4th order
    %%% Euler integration method
    %%% fx = Ax + Bu;
    [fx,u,A,B,C,Kf,K_14] = evalrhs_gyroboy5s_Freezing_EKF(x_4s,u, Model);
    x_4s = x_4s + Ts * fx; % Euler
    %%% Kf is the gain of Kalman filter


    %%% Limit the robot pitch angle between -90 to 90 deg.
    if x_4s(3) > 90*pi/180
        x_4s(3) =90*pi/180;
    end

     if x_4s(3) < -90*pi/180
        x_4s(3) =-90*pi/180;
     end

   %%% Update x1-x4
    x1(i+1)=x_4s(1);
    x2(i+1)=x_4s(2);
    x3(i+1)=x_4s(3);
    x4(i+1)=x_4s(4);


       %%% Generate real x1 integral (x5)
       %%%%% X5 = Integral of X1 %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

      x5(i+1) = x5(i) + x1(i)*Ts + 0.5* (x1(i+1)-x1(i))* Ts;



  %%%% Select mode for testing disturbance on state-estimation %%%%
  % 0.No disturbance
  % 1.Noise disturbance in X3
  % 2.Sensor X3 drift

     Mode = 0;  % Select mode 0

if Mode == 0
    %%%%% No disturbance in X
     x1_disturbance(i+1)=x_4s(1);
     x2_disturbance(i+1)=x_4s(2);
     x3_disturbance(i+1)=x_4s(3)+ 0;
     x4_disturbance(i+1)=x_4s(4);
    Kf_X = Kf*[x_4s(1); x_4s(2); x_4s(3); x_4s(4)];  %%% Update:   Kf_X
end



if Mode == 1

     %%%%%% Test signal disturbance, drift only X3  %%%%%
    x1_disturbance(i+1)=x_4s(1);
```

```
    x2_disturbance(i+1)=x_4s(2);
    x3_disturbance(i+1)=x_4s(3)+ (i/(100*400))*pi/180;
    x4_disturbance(i+1)=x_4s(4);
    Kf_X = Kf*[x_4s(1); x_4s(2); x3_disturbance(i+1) ; x_4s(4)]; %%% Update:
Kf_X
end



if Mode == 2

   %%%%%%% Test noise disturbance, only X3  %%%%%%
   if  mod(i , 2) == 0   % disturbance frequency

    min = -5; %min Random
    max = 5; %max Random
    r = (max-min).*rand(1) + min; % Random
   else
     r=0;
   end

      x1_disturbance(i+1)=x_4s(1);
      x2_disturbance(i+1)=x_4s(2);
      %x3_disturbance(i+1)=x_4s(3)+ r(i+1)*pi/180;
      x3_disturbance(i+1)=x_4s(3)+ r*pi/180;
      x4_disturbance(i+1)=x_4s(4);
     Kf_X = Kf*[x_4s(1); x_4s(2); x3_disturbance(i+1) ; x_4s(4)]; %%% Update:
L*X
end


    %%% Kalman filter variable
    %%% See more in coding diagram
    A_KfC_Xhat =  (A-Kf*C)*Xhat;
    dXhat = B*u + Kf_X + A_KfC_Xhat;

    %%% Update
    dXhat1(i+1)=dXhat(1);
    dXhat2(i+1)=dXhat(2);
    dXhat3(i+1)=dXhat(3);
    dXhat4(i+1)=dXhat(4);

      %%%%% dXhat Integral (Xhat) %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

    Xhat1(i+1) = Xhat1(i) + dXhat1(i)*Ts + 0.5* (dXhat1(i+1)-dXhat1(i))* Ts;
    Xhat2(i+1) = Xhat2(i) + dXhat2(i)*Ts + 0.5* (dXhat2(i+1)-dXhat2(i))* Ts;
    Xhat3(i+1) = Xhat3(i) + dXhat3(i)*Ts + 0.5* (dXhat3(i+1)-dXhat3(i))* Ts;
    Xhat4(i+1) = Xhat4(i) + dXhat4(i)*Ts + 0.5* (dXhat4(i+1)-dXhat4(i))* Ts;


    %%% Limit the robot pitch angle Xhat between -90 to 90 deg.
    if  Xhat3(i+1) > 90*pi/180
        Xhat3(i+1)=90*pi/180;
    end

     if  Xhat3(i+1) < -90*pi/180
         Xhat3(i+1)=-90*pi/180;
    end


   %%% Select control feedback : Xhat3 for x3%%%%

   %u_feedback = K_14*[x_4s(1); x_4s(2); x_4s(3); x_4s(4)];              %Xsys
```

```
   %u_feedback = K_14*[Xhat1(i+1); Xhat2(i+1); Xhat3(i+1); Xhat4(i+1)]; %Xhat

   u_feedback = K_14*[x_4s(1); x_4s(2); Xhat3(i+1); x_4s(4)];
%Select only X3



     %%% Command print for waiting %%%
    Cal_percent = i*100/t;

     if i== 1
       fprintf('Start at x3= %3.2f deg, x4= %3.1f
deg...\n',x3(1)*180/pi,x4(1)*180/pi)
     end


    if mod(Cal_percent , 2) == 0
       %fprintf('Calculating  %f percent ...\n',Cal_percent)
       fprintf('Cal.  %3.2f percent,  ',Cal_percent)
       %fprintf('x3= %3.2f deg,  ',Xhat3(i+1)*180/pi)
       fprintf('x3= %3.2f deg,  ',x3(i+1)*180/pi)
       fprintf('x4= %3.2f deg/s,  ',x4(i+1)*180/pi)
       fprintf('t= %3.2f sec...\n',Ts*i)
    end

    %Reset the x1, x2, x3, x4, x5 & x6 variables to new values and get ready
for the next iteration.
end

% use below line instead above when ploting the failing system
(uncontrollable)
% Time = 0:Ts:(i-1)*Ts;
 u=u_in1;  % Store as u

%%% Move all Xhat to x3 for plotting
x3=Xhat3;


%%%%% Save variables for plotting future %%%
%%% 'x1','x2','x3','x4','x5','u','Time'  %%%


%%%% Fig. [2x3] %%%
figure('Name','Freezing control and EKF');
Fn = 14; % font size

subplot(2,3,1);
p1=plot(Time,x1*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta - Wheel angles x1 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2=plot(Time,x2*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta dot x2 (deg/s)');
set(gca,'FontSize', Fn);

subplot(2,3,2);
p3=plot(Time,x3*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi - Pitch of body x3 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,5);
p4=plot(Time,x4*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi dot x4 (deg/s)');
set(gca,'FontSize', Fn);
```

```
 subplot(2,3,3);
 p5=plot(Time,x5*180/pi,'r'); grid;
 xlabel('Time (sec)'); ylabel('Control signal-u (volt)');
 set(gca,'FontSize', Fn);

subplot(2,3,6);  %5states
p6=plot(Time,u,'r'); grid;
xlabel('time (s)','FontSize', Fn); ylabel('u (volts)','FontSize', Fn);
set(gca,'FontSize', Fn);

%LineWidth
LW=1.2;
p1(1).LineWidth = LW;
p2(1).LineWidth = LW;
p3(1).LineWidth = LW;
p4(1).LineWidth =LW;
p5(1).LineWidth = LW;
p6(1).LineWidth =LW;
```

Function file: *evalrhs_gyroboy5s_Freezing_EKF.m*

```
function [fx,u,A4,B4,C4,Kf,K_14] = evalrhs_gyroboy5s_Freezing_EKF( x ,u,
Model)

    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT

    %%% 5-states Model parameters
    [A5,B5,Q,R2,~,~,~]=Gyroboy_Nonlinear_Model_5s(x, Model);
     %[A5,B5,Q,R2,Vmax,alpha,beta]
    % When disturbancing weigth and heigh, these parameters are not changed


     [K_LQR,~,~]=lqr(A5,B5,Q,R2);  % K Nonlinear
    %use the MATLAB 'lqr' function to solve Riccati equation and work out K,P


    K_14 = K_LQR;
    K_14( : , 5)=[];   % feedback gain x1-x4
    % delete column 5th, not need k5 for x5


    %%% 4-states Model parameters
    %%% Real model parameters
    [A4,B4,~,~,~]=Gyroboy_Nonlinear_Model_4s(x, Model);
    %[A4,B4,Vmax,alpha,beta]
    % When disturbancing weigth and heigh, these parameters are changed



    x14=[x1;x2;x3;x4];
    %%% Calculate fx for 4-state system
    %%% fx = Ax + Bu
    %%% Use u from controller voltage input
    fx = A4*x14 + B4*u;
```

```
   %%% Matrix C
   C4=eye(4);

   %%% Kalman filter noise parameters
   Noise_V= 0.2*eye(4); %0.2   % increase to smooth cure % R of Kalman filter
   Noise_W= 1*eye(4);                                    % Q of Kalman filter

   %%% Calculation gain Kf of Kalman filter
   [~,Pk,~]=lqr(A4,C4',Noise_W,Noise_V);
   Kf= Pk*C4'*inv(Noise_V);


    %calculate the function output 'fx' based on values of A, B, P and x.
end
```

Appendix A.6.11: MATLAB codes of TWR systems using freezing control technique with soft constrained input

Script file: Gyroboy_5s_Freezing_SoftConsV_10_2021.m

```
%%%% Nonlinear Freezing Control with soft constrained voltage
%%% for LEGO EV3 Robot

%%%%%%  Functions programme needed %%%
% evalrhs_Freezing_K5_SoftCons();         % Generating K5 only
% evalrhs_gyroboy5s_Freezing_SoftCons();  % Generating K1-K4 and fx
%
%      Inside two functions
%
%           Gyroboy_Nonlinear_Model_5s();
%           Gyroboy_Nonlinear_Model_4s();
%           Maxon Motor parameters etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;

%%% Set initial x1-x5
%%% Always set x1=x3

x1(1)=10*pi/180;      %set Theta - Average of wheel angles (deg)
x2(1)=0;                %set Theta_DOT (deg/s)
x3(1)=10*pi/180; %set Psi - Pitch angle of body (deg) - CONTROL to ZERO
x4(1)=0*pi/180;         %set Psi_DOT (deg/s)
x5(1)=0;                %set Theta integral
x_np1(1)=0; %X-Saturation  %X6 %Aftiicial voltage

 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed

model=1;

u_real_all(1)=0;   % real voltage

%%% initial artificial voltage feedbck
```

```
u_new = 0;
u_feedback_new = 0;


x1_err = 0;      % x1 error for tracking sys
x1_ref = 0;      % x1 reference
x1_err_int(1)=0; % x1 error integral

Ts=0.0001; %time step length

Duration=10;       % time sec
t=Duration*(1/Ts); % time step in programming
Time = 0:Ts:t*Ts;   % Create real time step for plotting

for i=1:t

  x_6s    =[x1(i); x2(i); x3(i); x4(i); x5(i); x_np1(i)];   % whole system +
Xnp1
  x_5s    =[x1(i); x2(i); x3(i); x4(i); x_np1(i)];          % x1-x5 state
feedback


  %%%%%% Programming Diagram %%%%%

%
%                    u_x1  +     u_new                              y=x1-x4,xnp1         y=x1
%  X1ref->-o-->-Int--K5-->-->o--------------+-------| Plant |-->----------------+---C------+
%         ^-                 ^ -                                                |          |
%         |                  | u_feedback_new                                   |          |
%         |                  |                                                  |
%         x1                 |                                                  |
%                            +--------------------<--K14,Knp1-- ----------- ---+
%                                                                 x1-x4,xnp1
%
% %%%%%%%%%%%%%%%


    %%% x1 error for tracking sys
    x1_err(i+1) = x1_ref - x1(i);

      %%%%% X1 error Integral  %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

     %%% x1 error integral
    x1_err_int(i+1) = x1_err_int(i) + x1_err(i)*Ts + 0.5*(x1_err(i+1)-
x1_err(i))*Ts ;  % [h1 x L]+[ 0.5 x(h2-h1) x L ] + old

     %%% cal. K5 for x5 only (integral x1)
    K_5 = evalrhs_Freezing_K5_SoftCons(x_6s,i,model);

    %%% u=kx,  Motor Voltage of integral x1
    u_x1_new = K_5 * x1_err_int(i+1);

    %%% The final voltages used to control robot motors
    u_new(i) = u_x1_new - u_feedback_new;


    %%%%%%%%

    %%% calculate the new 'x' vector using a 4th order
    %%% Euler integration method
    %%% fx = Ax + Bu;
     [fx,u_real,K14_n_xnp1,alphaa,betaa] =
evalrhs_gyroboy5s_Freezing_SoftCons(x_5s,u_new(i),i,model);
    x_5s = x_5s + Ts * fx; % Euler

     %%% K14 = K1,K2,K3 and K4   plus K_xnp1
```

```
    %%% Limit the robot pitch angle between -90 to 90 deg.
    if x_5s(3) > 90*pi/180
        x_5s(3) =90*pi/180;
    end

     if x_5s(3) < -90*pi/180
        x_5s(3) =-90*pi/180;
     end

     %%% Update x1-x4
    x1(i+1)=x_5s(1);
    x2(i+1)=x_5s(2);
    x3(i+1)=x_5s(3);
    x4(i+1)=x_5s(4);
    x_np1(i+1)=x_5s(5);


    u_real_all(i+1)=u_real;   % Update u


       %%%%% X5 = Integral of X1 %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

      x5(i+1) = x5(i) + x1(i)*Ts + 0.5* (x1(i+1)-x1(i))* Ts;


   %%% x1-x5 motor voltage feedback %%%%
   u_feedback_new = K14_n_xnp1*[x_5s(1); x_5s(2); x_5s(3); x_5s(4); x_5s(5)];


      %%% Command print for waiting %%%

    if i== 1
       fprintf('Start at x3= %3.2f deg...\n',x3(1)*180/pi)
     end


    Cal_percent = i*100/t;

    if mod(Cal_percent , 2) == 0
       fprintf('Calculating  %3.2f percent,  ',Cal_percent)
       fprintf('x3= %3.2f deg,  ',x3(i+1)*180/pi)
       fprintf('x4= %3.2f deg/s,  ',x4(i+1)*180/pi)
       fprintf('x6= %3.2f ,  ',x_np1(i+1)*180/pi)
       fprintf('t= %3.2f sec...\n',Ts*i)
    end


    %Reset the x1, x2, x3, x4, x5 & x6 variables to new values and get ready
for the next iteration.
end

% Use below line instead above when ploting the failing system
(uncontrollable)
% Time = 0:Ts:(i-1)*Ts;
u = u_real_all;           % ploting real voltage

%%%%%% Save variables for plotting future %%%
%%% 'x1','x2','x3','x4','x5','u','Time'  %%%

%save('Sim_Output_workspace_nonlinear1_Vsat','x1','x2','x3','x4','x5','u','x_n
p1','Time');
```

```
figure('Name','Unew');
Fn = 14; % font size
p_Unew=plot(Time,x_np1,'r'); grid;
xlabel('time (s)','FontSize', Fn); ylabel('x6', 'FontSize', Fn);
set(gca,'FontSize', Fn);

% Robert (2020). symlog (https://www.github.com/raaperrotta/symlog),
% GitHub. Retrieved June 15, 2020.
symlog('y')

LW=1.2;
p_Unew(1).LineWidth = LW;



%%%% Fig. [2x3] %%%
figure('Name','Freezing control with soft constrained input');
Fn = 14; % font size

subplot(2,3,1);
p1=plot(Time,x1*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta - Wheel angles x1 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2=plot(Time,x2*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta dot x2 (deg/s)');
set(gca,'FontSize', Fn);

subplot(2,3,2);
p3=plot(Time,x3*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi - Pitch of body x3 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,5);
p4=plot(Time,x4*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi dot x4 (deg/s)');
set(gca,'FontSize', Fn);

 subplot(2,3,3);
 p5=plot(Time,x5*180/pi,'r'); grid;
 xlabel('Time (sec)'); ylabel('Control signal-u (volt)');
 set(gca,'FontSize', Fn);

subplot(2,3,6);  %5states
p6=plot(Time,u,'r'); grid;
xlabel('time (s)','FontSize', Fn); ylabel('u (volts)','FontSize', Fn);
set(gca,'FontSize', Fn);

%LineWidth
LW=1.2;
p1(1).LineWidth = LW;
p2(1).LineWidth = LW;
p3(1).LineWidth = LW;
p4(1).LineWidth =LW;
p5(1).LineWidth = LW;
p6(1).LineWidth =LW;
```

Function file: evalrhs_Freezing_K5_SoftCons.m

```
function K_5 = evalrhs_Freezing_K5_SoftCons(x,i,model)

    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT
    x5=x(5); %Wheel integral
    x_np1=x(6); % Artificial control signal


     if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
     end
    if x_np1 == 0
       x_np1=1.0e-20;  % avoid Inf's and NaN's
    end

   %%% 5-states Model parameters
    [A5,B5,Q5,R2,Vmax,alpha,beta]=Gyroboy_Nonlinear_Model_5s(x,model);


   RR=10;
  %  set R matrices


   %%% Set soft saturation voltage

     if     (x_np1 )  > Vmax  % Lego Motor Maximum Voltage 8.3 V
         Phi_L = Vmax;
         Phi_R = Vmax;
     elseif  (x_np1) < -Vmax % Lego Motor Minimum Voltage -8.3 V
         Phi_L = -Vmax;
         Phi_R = -Vmax;
     else

           Phi_L = x_np1;
           Phi_R = x_np1;
     end

     Phi = [Phi_L ;
            Phi_R ];

    %%%  Soft saturation voltage matrices
Aa= [       A5            (B5*Phi)/x_np1  ;
       zeros(1,5)            0       ];

Ba= [  zeros(5,1)  ;
          1        ];

Qa= [      Q5         zeros(5,1)     ;
      zeros(1,5)    2*((Phi_L)^2)*RR ];

Ra=  0.001;


     [K_LQR,~,~]=lqr(Aa,Ba,Qa,Ra);  % K Nonlinear
    %use the MATLAB 'lqr' function to solve Riccati equation and work out K,P


    K_56 = K_LQR;
    K_56( : , 1:4)=[];   % feedback gain  (delete column 1-4th)

    K_5 = K_56;       % Select K5 only
    K_5(: , 2)=[];    % integral gain (delete column 2th means column 6th )
```

```
    %calculate the function output 'fx' based on values of A, B, P and x.
end
```

Function file: evalrhs_gyroboy5s_Freezing_SoftCons.m

```
function [fx,u_real,K14_n_xnp1,alpha,beta] =
evalrhs_gyroboy5s_Freezing_SoftCons( x ,u_new,i,model)

     %%%%%%  Functions programme needed %%%
    % Gyroboy_Nonlinear_Model_5s();  % Generating K1-K4
    % Gyroboy_Nonlinear_Model_4s();  % Generating fx, x1-x4 without x5
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT
    x_np1=x(5); %x_np1 %x6


     if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
    end
     if x_np1 == 0
       x_np1=1.0e-20;  % avoid Inf's and NaN's
     end


   %%% 5-states Model parameters
    [A5,B5,Q5,~,Vmax,alpha,beta]=Gyroboy_Nonlinear_Model_5s(x,model);
    %[A5,B5,Q,R2,Vmax,alpha,beta]


 RR=10;
  %  set Q and R matrices



 %%% Set soft saturation voltage

     if      (x_np1 )  > Vmax  % Lego Motor Maximum Voltage 8.3 V
         Phi_L = Vmax;
         Phi_R = Vmax;
     elseif  (x_np1) < -Vmax % Lego Motor Minimum Voltage -8.3 V
         Phi_L = -Vmax;
         Phi_R = -Vmax;
     else
         Phi_L = Vmax*sin((pi*x_np1)/(2*Vmax));
         Phi_R = Vmax*sin((pi*x_np1)/(2*Vmax));
%
%            Phi_L = x_np1;
%            Phi_R = x_np1;
     end

     Phi = [Phi_L ;
            Phi_R ];


    %%% Send data real voltage for ploting
```

```
     u_real=   Phi(1);


    %%%  Soft saturation voltage matrices

Aa= [       A5             (B5*Phi)/x_np1  ;
       zeros(1,5)             0       ];

Ba= [  zeros(5,1)  ;
          1        ];

Ca= eye(6);

Qa= [      Q5         zeros(5,1)     ;
      zeros(1,5)    2*((Phi_L)^2)*RR ];

Ra=  0.001;

     [K_LQR,~,~]=lqr(Aa,Ba,Qa,Ra);  % K Nonlinear


    %%%% Need only K1-4 and X_np1, integral of x1 (x5) in not need
    %%%% K_LQR = 2x6 matrics

    K14_n_xnp1 = K_LQR;
    K14_n_xnp1( : , 5)=[];   % feedback gain  (delete column 5th)


   %%% Real parameters , used to disturbance weigth and height
    [A4,B4,~,~,~]=Gyroboy_Nonlinear_Model_4s(x,model);
   %[A4,B4,Vmax,alpha,beta]


     %%% Set soft saturation voltage for new matrix A

     if     (x_np1 )  > Vmax  % Lego Motor Maximum Voltage 8.3 V
         Phi_L = Vmax;
         Phi_R = Vmax;
     elseif  (x_np1) < -Vmax % Lego Motor Minimum Voltage -8.3 V
         Phi_L = -Vmax;
         Phi_R = -Vmax;
     else
         Phi_L = Vmax*sin((pi*x_np1)/(2*Vmax));
         Phi_R = Vmax*sin((pi*x_np1)/(2*Vmax));

     end

     Phi = [Phi_L ;
            Phi_R ];


Ab= [       A4             (B4*Phi)/x_np1  ;
       zeros(1,4)             0       ];

Bb= [  zeros(4,1)  ;
          1        ];


    x14_n_xnp1=[x1;x2;x3;x4;x_np1];
    %%% Calculate fx for 4-state+Xnp1 system
    %%% fx = Ax + Bu
    %%% Use u from controller voltage input
    fx = Ab*x14_n_xnp1 + Bb*u_new;
```

```
    %calculate the function output 'fx' based on values of A, B, P and x.
end

-----------------------
```

The function file "*symlog.m*" is provided by Mathworks from this website (Mathworks, 2020),

https://uk.mathworks.com/matlabcentral/fileexchange/57902-symlog?s_tid=mwa_osa_a

Appendix A.6.12: MATLAB codes of TWR systems using freezing control technique and EKF with soft constrained input

Script file: Gyroboy_5s_FreezingAndEKF_SoftConsV_10_2021.m

```
%%%% Nonlinear Freezing Control and EKF with soft constrained voltage
%%% for LEGO EV3 Robot

%%%%%%  Functions programme needed %%%
% evalrhs_Freezing_K5_SoftCons();               % Generating K5 only
% evalrhs_gyroboy5s_FreezingAndEKF_SoftCons();  % Generating K1-K4 and fx
%
%      Inside two functions
%
%           Gyroboy_Nonlinear_Model_5s();
%           Gyroboy_Nonlinear_Model_4s();
%           Maxon Motor parameters etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all;
close all;


x1(1)=10*pi/180; %set Theta - Average of wheel angles (rad)  %11 deg.
x2(1)=0; %set Theta_DOT (rad/s)
x3(1)=10*pi/180; %set Psi - Pitch angle of body (rad) - CONTROL to ZERO
x4(1)=0; %set Psi_DOT
x5(1)=0; %theta_int
x_np1(1)=0; %X-Saturation

 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed

model=1;

u_real_all(1)=0;   % real voltage
```

```
%%% initial artificial voltage feedbck
u_new = 0;
u_feedback_new = 0;


Xhat1(1) = x1(1);
Xhat2(1) = 0;
Xhat3(1) = x3(1);
Xhat4(1) = 0;
Xhat_Xnp1(1) = 0;

dXhat1(1) = 0;
dXhat2(1) = 0;
dXhat3(1) = 0;
dXhat4(1) = 0;
dXhat_Xnp1(1) = 0;

x1_disturbance(1) = x1(1);
x2_disturbance(1) = 0;
x3_disturbance(1) = x3(1);
x4_disturbance(1) = 0;
Xnp1_disturbance(1) = 0;

x1_err = 0;
x1_ref = 0;
x1_err_int(1)=0;

Ts=0.0001; %time step length

Duration=10;       % time sec
t=Duration*(1/Ts); % time step in programming
Time = 0:Ts:t*Ts;   % Create real time step for plotting

for i=1:t

  x_6s    =[x1(i); x2(i); x3(i); x4(i); x5(i); x_np1(i)];  % for cal K5
  x_5s    =[x1(i); x2(i); x3(i); x4(i); x_np1(i)];          % for cal K1-
K4,Xnp1
  Xhat =[Xhat1(i); Xhat2(i); Xhat3(i); Xhat4(i);Xhat_Xnp1(i)];
  dXhat=[dXhat1(i); dXhat2(i); dXhat3(i); dXhat4(i); dXhat_Xnp1(i)];



  %%%%%% Diagram %%%%%

%
%                           +     u                                 y=x1-x4,x_np1            y=x1
%  X1ref->-o-->-Int--K5-->-->o-------------------+-------| Plant |-->-----------o---+---C------+
%         ^-                ^ -                 |                                ^   |          |
%         |                 | u_feedback        +------B------>|         Add    |   |          |
%         |                 |                                   |       Distubance  |
%         x1                |       Xhat                dXhat   |                   |
%                           +-<--K14,np1-- o<--Integal <----- o <-- L -------- ---+
%                                         |                   ^       x1-x4,x_np1
%                                         |                   |
%                                         +------- A-LC -->--+


    x1_err(i+1) = x1_ref - x1(i);

      %%%%% X1 error Integral  %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

    x1_err_int(i+1) = x1_err_int(i) + x1_err(i)*Ts + 0.5*(x1_err(i+1)-
x1_err(i))*Ts ;  % [h1 x L]+[ 0.5 x(h2-h1) x L ] + old

    K_5 = evalrhs_Freezing_K5_SoftCons(x_6s,i,model);
```

```
    u_x1_new = K_5 * x1_err_int(i+1);

    % r = 0;
    % u = r - u_feedback;
    u_new = u_x1_new - u_feedback_new;


  %%% calculate the new 'x' vector using a 4th order
    %%% Euler integration method
    %%% fx = Ax + Bu;
    [fx,u_real,A,B,C,L,K14_n_xnp1] =
evalrhs_gyroboy5s_FreezingAndEKF_SoftCons(x_5s,u_new,model);
    x_5s = x_5s + Ts * fx; % Euler

%%% Limit the robot pitch angle between -90 to 90 deg.
    if x_5s(3) > 90*pi/180
        x_5s(3) =90*pi/180;
    end

     if x_5s(3) < -90*pi/180
        x_5s(3) =-90*pi/180;
     end

     %%% Update x1-x4
    x1(i+1)=x_5s(1);
    x2(i+1)=x_5s(2);
    x3(i+1)=x_5s(3);
    x4(i+1)=x_5s(4);
    x_np1(i+1)=x_5s(5);

    u_real_all(i+1)=u_real;  % Update u


       %%%%% X5 = Integral of X1 %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

      x5(i+1) = x5(i) + x1(i)*Ts + 0.5* (x1(i+1)-x1(i))* Ts;


  %%%% Select mode for testing %%%%
  % 0.No disturbance
  % 1.Noise disturbance in X3
  % 2.Sensor X3 drift

     Mode = 0;

if Mode == 0
    %%%%% No disturbance in X
     x1_disturbance(i+1)=x_5s(1);
     x2_disturbance(i+1)=x_5s(2);
     x3_disturbance(i+1)=x_5s(3)+ 0;
     x4_disturbance(i+1)=x_5s(4);
     Xnp1_disturbance(i+1)=x_5s(4);
    LX = L*[x_5s(1); x_5s(2); x_5s(3); x_5s(4); x_5s(5)];  % No disturbance
end



if Mode == 1

%     %%%%%% Test signal disturbance, drift only X3  %%%%%
    x1_disturbance(i+1)=x_5s(1);
    x2_disturbance(i+1)=x_5s(2);
    x3_disturbance(i+1)=x_5s(3)+ (i/(100*400))*pi/180;
```

```
    x4_disturbance(i+1)=x_5s(4);
    Xnp1_disturbance(i+1)=x_5s(5);
    LX = L*[x_5s(1); x_5s(2); x3_disturbance(i+1) ; x_5s(4); x_5s(5)];
end


if Mode == 2
%    %%%%%%% Test noise disturbance, only X3  %%%%%%
   if  mod(i , 2) == 0   % frequency

    min = -5; %min Random
    max = 5; %max Random
    r = (max-min).*rand(1) + min; % Random
   else
     r=0;
   end

      x1_disturbance(i+1)=x_5s(1);
      x2_disturbance(i+1)=x_5s(2);
      %x3_disturbance(i+1)=x_4s(3)+ r(i+1)*pi/180;  %%% ramdom noise variable
      x3_disturbance(i+1)=x_5s(3)+ r*pi/180;
      x4_disturbance(i+1)=x_5s(4);
      Xnp1_disturbance(i+1)=x_5s(5);
     LX = L*[x_5s(1); x_5s(2); x3_disturbance(i+1) ; x_5s(4) ; x_5s(5)];
end


   %%% Kalman filter variable
    %%% See more in coding diagram
    A_LC =  (A-L*C)*Xhat;
    dXhat = B*u_new + LX + A_LC;

    %%% Update
    dXhat1(i+1)=dXhat(1);
    dXhat2(i+1)=dXhat(2);
    dXhat3(i+1)=dXhat(3);
    dXhat4(i+1)=dXhat(4);
    dXhat_Xnp1(i+1)=dXhat(5);

      %%%%% dXhat Integral (Xhat) %%%%%
      %  Sum_Area = Previous_area +  New_area
      %  Sum(i+1) = Sum(i)        +  (Square(i) + triangle(i))

    Xhat1(i+1) = Xhat1(i) + dXhat1(i)*Ts + 0.5* (dXhat1(i+1)-dXhat1(i))* Ts;

    Xhat2(i+1) = Xhat2(i) + dXhat2(i)*Ts + 0.5* (dXhat2(i+1)-dXhat2(i))* Ts;
    Xhat3(i+1) = Xhat3(i) + dXhat3(i)*Ts + 0.5* (dXhat3(i+1)-dXhat3(i))* Ts;
    Xhat4(i+1) = Xhat4(i) + dXhat4(i)*Ts + 0.5* (dXhat4(i+1)-dXhat4(i))* Ts;
    Xhat_Xnp1(i+1) = Xhat_Xnp1(i) + dXhat_Xnp1(i)*Ts + 0.5* (dXhat_Xnp1(i+1)-
dXhat_Xnp1(i))* Ts;

    %%% Limit the robot pitch angle Xhat between -90 to 90 deg.
    if  Xhat3(i+1) > 90*pi/180
         Xhat3(i+1)=90*pi/180;
    end

     if  Xhat3(i+1) < -90*pi/180
         Xhat3(i+1)=-90*pi/180;
    end


   %%% Select x feedback %%%%

   %u_feedback = K_14*[x_4s(1); x_4s(2); x_4s(3); x_4s(4)];             %Xsys
   %u_feedback = K_14*[Xhat1(i+1); Xhat2(i+1); Xhat3(i+1); Xhat4(i+1)]; %Xhat
```

```
   u_feedback_new = K14_n_xnp1*[x_5s(1); x_5s(2); Xhat3(i+1); x_5s(4);
x_5s(5)];   %X mix only X3


        %%% print %%%

    if i== 1
       fprintf('Start at x3= %3.2f deg...\n',Xhat3(1)*180/pi)
     end


    Cal_percent = i*100/t;

    if mod(Cal_percent , 2) == 0
       fprintf('Cal.  %3.2f per,  ',Cal_percent)
       fprintf('x3= %3.2f deg,  ',Xhat3(i+1)*180/pi)
       fprintf('x4= %3.2f deg/s,  ',x4(i+1)*180/pi)
       fprintf('t= %3.2f sec...\n',Ts*i)
    end


    %Reset the x1, x2, x3, x4, x5 & x6 variables to new values and get ready
for the next iteration.
end

% Use below line instead above when ploting the failing system
(uncontrollable)
%Time = 0:Ts:(i-1)*Ts;
u = u_real_all;

x3=Xhat3;
%save('Sim_Output_workspace_nonlinear1_Vsat','x1','x2','x3','x4','x5','u','x_n
p1','Time');


figure('Name','Unew');
Fn = 14; % font size
p_Unew=plot(Time,x_np1,'m'); grid;
xlabel('time (s)','FontSize', Fn); ylabel('x6', 'FontSize', Fn);
set(gca,'FontSize', Fn);

% Robert (2020). symlog (https://www.github.com/raaperrotta/symlog),
% GitHub. Retrieved June 15, 2020.
symlog('y')

LW=1.5;
p_Unew(1).LineWidth = LW;


%%%% Fig. [2x3] %%%
figure('Name','Freezing control and EKF with soft constrained input');
Fn = 14; % font size

subplot(2,3,1);
p1=plot(Time,x1*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta - Wheel angles x1 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,4);
p2=plot(Time,x2*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Theta dot x2 (deg/s)');
set(gca,'FontSize', Fn);

subplot(2,3,2);
```

```
p3=plot(Time,x3*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi - Pitch of body x3 (deg)');
set(gca,'FontSize', Fn);

subplot(2,3,5);
p4=plot(Time,x4*180/pi,'r'); grid;
xlabel('Time (sec)'); ylabel('Psi dot x4 (deg/s)');
set(gca,'FontSize', Fn);

 subplot(2,3,3);
 p5=plot(Time,x5*180/pi,'r'); grid;
 xlabel('Time (sec)'); ylabel('Control signal-u (volt)');
 set(gca,'FontSize', Fn);

subplot(2,3,6);  %5states
p6=plot(Time,u,'r'); grid;
xlabel('time (s)','FontSize', Fn); ylabel('u (volts)','FontSize', Fn);
set(gca,'FontSize', Fn);

%LineWidth
LW=1.2;
p1(1).LineWidth = LW;
p2(1).LineWidth = LW;
p3(1).LineWidth = LW;
p4(1).LineWidth =LW;
p5(1).LineWidth = LW;
p6(1).LineWidth =LW;



------------------------
```

Function file: evalrhs_gyroboy5s_FreezingAndEKF_SoftCons.m

```
function [fx,u_real,Ab,Bb,Cb,Lk,K14_n_xnp1] =
evalrhs_gyroboy5s_FreezingAndEKF_SoftCons( x ,u_new,model)

    x1=x(1); %Theta - Average of wheel angles
    x2=x(2); %ThetaDOT
    x3=x(3); %Psi - Pitch of body CONTROL to ZERO
    x4=x(4); %PsiDOT
    x_np1=x(5); %x_np1 %x6


     if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
    end
     if x_np1 == 0
       x_np1=1.0e-20;  % avoid Inf's and NaN's
     end

    %%% 5-states Model parameters
    [A5,B5,Q5,~,Vmax,alpha,beta]=Gyroboy_Nonlinear_Model_5s(x,model);
    %[A5,B5,Q,R2,Vmax,alpha,beta]


 RR=10;
  %  set Q and R matrices


     if      (x_np1 )  > Vmax  % Lego Motor Maximum Voltage 8.3 V
         Phi_L = Vmax;
         Phi_R = Vmax;
     elseif  (x_np1) < -Vmax % Lego Motor Minimum Voltage -8.3 V
```

```
        Phi_L = -Vmax;
        Phi_R = -Vmax;
    else
        Phi_L = Vmax*sin((pi*x_np1)/(2*Vmax));
        Phi_R = Vmax*sin((pi*x_np1)/(2*Vmax));

    end

    Phi = [Phi_L ;
           Phi_R ];

    u_real=   Phi(1);


Aa= [       A5            (B5*Phi)/x_np1  ;
      zeros(1,5)             0       ];

Ba= [  zeros(5,1)  ;
         1        ];

Ca= eye(6);

Qa= [      Q5         zeros(5,1)     ;
     zeros(1,5)    2*((Phi_L)^2)*RR ];

Ra=  0.001;


    [K_LQR,~,~]=lqr(Aa,Ba,Qa,Ra);  % K Nonlinear
   %use the MATLAB 'lqr' function to solve Riccati equation and work out K,P

   %%%% Need only K1-4 and X_np1
   %%%% K_LQR = 2x6 matrics

   K14_n_xnp1 = K_LQR;
   K14_n_xnp1( : , 5)=[];   % feedback gain  (delete column 5th)


%%% Real parameters , used to disturbance weigth and height
   [A4,B4,~,~,~]=Gyroboy_Nonlinear_Model_4s(x,model);
  %[A4,B4,Vmax,alpha,beta]


    if     (x_np1 )  > Vmax  % Lego Motor Maximum Voltage 8.3 V
        Phi_L = Vmax;
        Phi_R = Vmax;
    elseif  (x_np1) < -Vmax % Lego Motor Minimum Voltage -8.3 V
        Phi_L = -Vmax;
        Phi_R = -Vmax;
    else
        Phi_L = Vmax*sin((pi*x_np1)/(2*Vmax));
        Phi_R = Vmax*sin((pi*x_np1)/(2*Vmax));

    end

    Phi = [Phi_L ;
           Phi_R ];


Ab= [       A4            (B4*Phi)/x_np1  ;
      zeros(1,4)             0       ];

Bb= [  zeros(4,1)  ;
         1        ];
```

```
    x14_n_xnp1=[x1;x2;x3;x4;x_np1];

    fx = Ab*x14_n_xnp1 + Bb*u_new;      %% ???????


  Cb=eye(5);   % add new control signal % x5 = Xnp1

   Noise_V= 0.2*eye(5); %0.2   % increse to smooth cure % R of Kalman filter
   Noise_W= 1*eye(5);                                  % Q of Kalman filter

   [~,Pk,~]=lqr(Ab,Cb',Noise_W,Noise_V);
   Lk= Pk*Cb'*inv(Noise_V);

   BU = Bb*u_new;


    %calculate the function output 'fx' based on values of A, B, P and x.
end
```

Appendix A.6.13: The initial programme of Simulink for implementing LEGO EV3 robot

This programme provides the main parameters of LEGO EV3 robot before running Simulink, such as Sample Time (Ts) and fixed gain K5.

Script file: parameters.m

```
clear all

%%% Parameters  Gyroboy EV3 %%%

%%% Physical constants
g = 9.81;                          % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                         % wheel weight [kg]  old 0.024 new 0.050
R = 0.027;                         % wheel radius [m]
Jw = m * R^2 / 2;                  % wheel inertia moment [kgm^2]
W = 0.105;                         % body width [m]
D = 0.1;                           % body depth [m]

M = 0.64;                         % body weight [kg]  old 0.80  new 0.64   (EV3
motor 0.160 kg/2ea)
h = 0.210;                         % body height [m]

L = h / 2;                         % distance of the center of mass from the
wheel axle [m] 10.5


%%%%% EV3 Motors parameters
Jm = 1e-5;                         % DC motor inertia moment [kgm^2]
Rm = 6.69;                         % DC motor resistance [Om]
Kb = 0.468;                        % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                        % DC motor torque constant [Nm/A]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Jpsi = M * L^2 / 3;             % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;    % body yaw inertia moment [kgm^2]

n = 1;                          % Gear ratio
fm = 0.0022;                    % friction coefficient between body & DC motor
fw = 0;                         % friction coefficient between wheel & floor

%%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;


Ts = 0.004;     % Simulink Sample Time
Direction = 1;   % Motor direction,  1 =+x  ,-1 = -x

a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=4*M*R*L*(1)*n^2*Jm;

    e23=2*n^2*Jm*M*g*L-M^2*R*L^2*(1)*g;
    e24=0;

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*(1);
%define these extra function to reduce the coding complexity of A and B
    %matrices for the x2dot equation

    e43=M*g*L*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=0;

    f41=2*n^2*Jm-M*R*L*(1);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;

%define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation

    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);

    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);

    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;
    %define these extra function to reduce the coding complexity of A and B
    %matrices with the motor added in


    %define A and B using a nonlinear state-space gyro robot model,
    %including the motor part



        %%%% 5-state Model %%%%
  A5 =[0      1             0              0       0  ;
       0  em22/(a+b)  em23/((a+b))  em24/(a+b)      0 ;
```

```
      0      0             0              1       0 ;
      0  em42/(a+b)  em43/((a+b))  em44/(a+b)      0 ;
      1      0             0              0       0 ];


B5 =[    0           0;
    fm21/(a+b)  fm22/(a+b);
        0           0;
    fm41/(a+b)  fm42/(a+b);
         0           0];


B2 =[        0 ;
      fm22/(a+b);
                0;
      fm42/(a+b);
                0];


C = eye(5);
D = zeros(5, 2);

%------ 4-state models for the Kalman filter

A4 =[0      1             0              0        ;
     0  em22/(a+b)  em23/((a+b))  em24/(a+b)   ;
     0      0             0              1        ;
     0  em42/(a+b)  em43/((a+b))  em44/(a+b)   ];

 B4 =[    0           0;
    fm21/(a+b)  fm22/(a+b);
        0           0;
    fm41/(a+b)  fm42/(a+b)];

 C4 = eye(4);
 D4 = zeros(4, 2);


 %%% Q and R matrices
 Q=[20, 0, 0, 0, 0;
    0, 1, 0, 0, 0;
    0, 0, 1, 0, 0;
    0, 0, 0, 1, 0;
    0, 0, 0, 0, 5];

R2 = 10*[1,0;
         0,1];


%%%% Riccati Eq solving gains K

 [K_LQR,~,~]=lqr(A5,B5,Q,R2);
 %%% K=[2x5]

 %%% Create K1-K4
 K_LQR_1to4 = K_LQR;
 K_LQR_1to4( : , 5)=[]   % feedback gain  (delete column 5th)

 %%% Create K5
 K_LQR_5    = K_LQR;
 K_LQR_5(: , 1:4)=[]    % integral gain (delete column 1-4th)
```

```
Noise_W = 1*eye(4);     % Qk of Kalman filter
Noise_V = 0.2*eye(4);   % Rk of Kalman filter


C44= eye(4);

%%%% Riccati Eq solving gains P
[~,Pk,~]=lqr(A4,C44',Noise_W,Noise_V)

Lk= Pk*C44'*inv(Noise_V)   % L Kalman feedback

% Lk= [   0         0          0          0;
%         0         0          0          0;
%      0.3831    0.4703    16.6273    2.2738;
%        0         0          0          0];

A_LC = A4-(Lk*C44);    % A-LC Kalman feedback
```

Appendix A.6.14: A lookup table of freezing control gains K1-K4

As described in section 6.6, the LEGO EV3 controller cannot calculate the Riccati equation; thus, the lookup table is created by the following:

(Note that the gain K5 is not varied; therefore, the gain K5 is set as fixed gain in Simulink because of the limation of LEGO EV3 robot's memory storage)

Script file: Gyroboy_Create_Freezing_LookupTable2D_5s.m

```
%%%%% The matrix gains K1-K4 of freezing technique
%%%%% Note that the K5 is not changed
%%%%% Thus, we use store data only K1-K4 into Lookup Table
%%%%% K5 is used as fixed gain in Simulink

clear all

xx3(1)=0;   % psi
xx4(1)=0;   % psi dot

%%% x3 %%% Use is range
Minimun_Angle_x3= -20;   % Deg. min.   -20
Step_Angle_x3= 1;
Maximum_Angle_x3= 20;    % Deg. max.    20

%%% x4 %%%
% Deg/s,  440 Deg/s max for gyroscope sensor, but it showed 200 deg/s (Max)
when testing.
Minimun_Angle_x4= -130;
Step_Angle_x4= 5;
Maximum_Angle_x4= 130;

i=1;
```

```
j=1;
shift_i=0;

m_array =((Maximum_Angle_x3-Minimun_Angle_x3)/Step_Angle_x3)+1 ; % calculate m
size of cell
n_array =((Maximum_Angle_x4-Minimun_Angle_x4)/Step_Angle_x4)+1 ; % calculate n
size of cell

m_array=int16(m_array);
n_array=int16(n_array);

DataRange= (Minimun_Angle_x4)*pi/180 : (Step_Angle_x4)*pi/180 :
(Maximum_Angle_x4)*pi/180;
[ms,ns]=size(DataRange);

%%% match x3 and x4
for x4 = (Minimun_Angle_x4)*pi/180 : (Step_Angle_x4)*pi/180 :
(Maximum_Angle_x4)*pi/180
   for x3 = (Minimun_Angle_x3)*pi/180 : (Step_Angle_x3)*pi/180 :
(Maximum_Angle_x3)*pi/180


if x3 == 0     % avoid error : Inf's or NaN's.
x3 = 1.0e-20;
end
if x4 == 0     % avoid error : Inf's or NaN's.
x4 = 1.0e-20;
end

%%% Physical constants
g = 9.81;                       % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                      % wheel weight [kg]  old 0.024 new 0.050
R = 0.027;                      % wheel radius [m]
Jw = m * R^2 / 2;               % wheel inertia moment [kgm^2]
W = 0.105;                      % body width [m]
D = 0.1;                        % body depth [m]

M = 0.64;                    % body weight [kg]  old 0.80  new 0.64
h = 0.210;                      % body height [m]

L = h / 2;                      % distance of the center of mass from the
wheel axle [m] 10.5

Jpsi = M * L^2 / 3;             % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;    % body yaw inertia moment [kgm^2]


%%% Motors parameters
Jm = 1e-5;                      % DC motor inertia moment [kgm^2]
Rm = 6.69;                      % DC motor resistance [Om]
Kb = 0.468;                     % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                     % DC motor torque constant [Nm/A]
n = 1;                          % Gear ratio
fm = 0.0022;                    % friction coefficient between body & DC motor
fw = 0;


%%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;


    %initialise x1-x6 using the input 'x' vector
```

```
a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(x3)^2+4*M*R*L*cos(x3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(x3)-M^2*R*L^2*cos(x3)*g*sin(x3);
    e24=x4*sin(x3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(x3);


    e43=M*g*L*sin(x3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=x4*sin(x3)*(-M^2*R^2*L^2*cos(x3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(x3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation


    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);

    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);

    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;

    %define these extra function to reduce the coding complexity of A and B
    %matrices with the motor added in

    %define A and B using a nonlinear state-space gyro robot model,
    %including the motor part

    if x3 == 0
       x3=1.0e-1000000;  % avoid Inf's and NaN's
    end
    if x4 == 0
       x4=1.0e-1000000;  % avoid Inf's and NaN's
    end


 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed


  Model = 1;  % Select mode

  %%%-------- Model A ----------
if Model == 1
          %%%% Primary Model A %%%%
      A5 =[0      1             0              0       0  ;
           0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)  0 ;
           0      0             0              1       0 ;
```

```
          0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)  0 ;
          1       0            0               0       0 ];
end


%%%-------- Model B ----------
if Model == 2
          %%%% Model B %%%%
     A5 =[0       1            0                      0       0  ;
          0  em22/(a+b)  (em23+em24*x4)/((a+b)*x3)    0       0 ;
          0       0            0                      1       0 ;
          0  em42/(a+b)  (em43+em44*x4)/((a+b)*x3)    0       0 ;
          1       0            0                      0       0 ];

end


%%%-------- Model AB10 ----------
 if Model == 3

  %%% Mix A&B %%%%%

   if x3 <= (10.5*pi/180) && x3 >= (-10.5*pi/180)
    %%%% Primary Model A %%%%
     A5 =[0       1            0              0       0  ;
          0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)  0 ;
          0       0            0              1       0 ;
          0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)  0 ;
          1       0            0              0       0 ];


   else

      %%%% Model B %%%%
     A5 =[0       1            0                      0       0  ;
          0  em22/(a+b)  (em23+em24*x4)/((a+b)*x3)    0       0 ;
          0       0            0                      1       0 ;
          0  em42/(a+b)  (em43+em44*x4)/((a+b)*x3)    0       0 ;
          1       0            0                      0       0 ];
   end



end


    % Models Matrix B

     B5 =[    0           0;
        fm21/(a+b)  fm22/(a+b);
            0           0;
        fm41/(a+b)  fm42/(a+b);
          0             0];


    Q=[20, 0, 0, 0  0;
        0, 1, 0, 0  0;
        0, 0, 1, 0  0;
        0, 0, 0, 1  0;
        0, 0, 0, 0  5];

     R2=10*[1, 0;
            0, 1];

    %set Q and R matrices
```

```
   %%% Riccati EQ %%%
   K = lqr(A5, B5, Q, R2);
   %%% K= [2x5];


if x3 == 1.0e-20 % get zero back
x3 = 0;
end

xx3(i)= x3*180/pi; % convert angle rad. to deg.
xx3 = double(xx3);

%%% Create 1st column by x3 deg [4x1]
%%% K1-K4 table
K_Riccati_2D_4s(i+1+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column
K_Riccati_2D_4s(i+2+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column
K_Riccati_2D_4s(i+3+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column
K_Riccati_2D_4s(i+4+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column

%%% Create 1st column by x3 deg
%%% K5 table
K_Riccati_2D_theta_int(i+1,1)= xx3(i);         % Insert x3 deg at 1st column

%%% K= [2x5];
%%% Select k1,k2,k3,k4,k5
%%% one motor [1x5] two motors is [2x5]
Kp1 = K(1);
Kp2 = K(3);
Kp3 = K(5);
Kp4 = K(7);
Kp5 = K(9);


%%%% Put gains K1-K4 in lookup table %%% K=[4x1]
%%% start at 2nd column
K_Riccati_2D_4s( i+1+(3*shift_i) ,j+1)= Kp1;
K_Riccati_2D_4s( i+2+(3*shift_i) ,j+1)= Kp2;
K_Riccati_2D_4s( i+3+(3*shift_i) ,j+1)= Kp3;
K_Riccati_2D_4s( i+4+(3*shift_i) ,j+1)= Kp4;

%%%% Put gains K1-K4 in lookup table
K_Riccati_2D_theta_int( i+1 ,j+1)= Kp5;
%%% Proof that K5 is fixed at -0.5, thus use the fix gain in Simulink instead

i=i+1;  %x3
shift_i=shift_i+1;
   end

  xx4(j)= x4*180/pi;
  xx4 = double(xx4);

  %%% Insert x4 deg/s at 1st row  in K1-K4 Table
  K_Riccati_2D_4s(1,j+1)= xx4(j);

  %%% Insert x4 deg/s at 1st row in K5 Table
  K_Riccati_2D_theta_int(1,j+1)= xx4(j);  % Insert x3 deg at 1st column

    i=1;
    shift_i=0;
  j=j+1; %x4

      %%% print %%%
    Cal_percent = (j-1)*100/(ns-1);
```

```
    if mod(Cal_percent , 5) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end

end

save('local_Freezing_2D_K_workspace','K_Riccati_2D_4s','K_Riccati_2D_theta_int
');
```

Appendix A.6.15: A lookup table of extended Kalman filter gains

Note that the gain P of the Ricatti solution is used to estimate the x_3 in this research. Moreover, the 3rd row of gain P is effected to approximate the x_3 directly. Thus, we decide to use merely the 3rd row to estimate the pitch angle as the restriction of the robot's memory.

Script file: Gyroboy_Create_EKF_LookupTable2D_4s.m

```
%%%%% The matrix P solution of extended Kalman filter %%%%
%%%%% Need P not K %%%
%%%% Use only the 3rd row of P[4x4] for estimating x3

clear all

%x1 % theta
%x2 % theta_dot
%x3 % psi
%x4 % psi_dot

xx3(1)=0;  % psi
xx4(1)=0;  % psi dot

 %%% Weight matrices Qk, Rk of Kalman filter
 Noise_W = 1*eye(4);
 Noise_V = 0.2*eye(4);

%%% x3 %%%
Minimun_Angle_x3= -20;  % Deg. min.  -20
Step_Angle_x3= 1;
Maximum_Angle_x3= 20;   % Deg. max.   20
%%% x4 %%%
% Deg/s,  440 Deg/s max for gyroscope sensor, but it showed 200 deg/s (Max)
when testing.
Minimun_Angle_x4= -130;
Step_Angle_x4= 5;
Maximum_Angle_x4= 130;


i=1;
```

```
j=1;
shift_i=0;

m_array =((Maximum_Angle_x3-Minimun_Angle_x3)/Step_Angle_x3)+1 ; % calculate m
size of cell
n_array =((Maximum_Angle_x4-Minimun_Angle_x4)/Step_Angle_x4)+1 ; % calculate n
size of cell

m_array=int16(m_array);
n_array=int16(n_array);

DataRange= (Minimun_Angle_x4)*pi/180 : (Step_Angle_x4)*pi/180 :
(Maximum_Angle_x4)*pi/180;
[ms,ns]=size(DataRange);

%%% Match x3 and x4

for x4 = (Minimun_Angle_x4)*pi/180 : (Step_Angle_x4)*pi/180 :
(Maximum_Angle_x4)*pi/180
   for x3 = (Minimun_Angle_x3)*pi/180 : (Step_Angle_x3)*pi/180 :
(Maximum_Angle_x3)*pi/180


if x3 == 0     % avoid error : Inf's or NaN's.
x3 = 1.0e-20;
end

%%% Physical constants
g = 9.81;                          % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                         % wheel weight [kg]  old 0.024 new 0.050
R = 0.027;                         % wheel radius [m]
Jw = m * R^2 / 2;                  % wheel inertia moment [kgm^2]
W = 0.105;                         % body width [m]
D = 0.1;                           % body depth [m]

M = 0.64;                     % body weight [kg]  old 0.80  new 0.64
h = 0.210;                         % body height [m]

L = h / 2;                         % distance of the center of mass from the
wheel axle [m] 10.5

Jpsi = M * L^2 / 3;                % body pitch inertia moment [kgm^2]
Jphi = M * (W^2 + D^2) / 12;       % body yaw inertia moment [kgm^2]


%%% Motors parameters
Jm = 1e-5;                         % DC motor inertia moment [kgm^2]
Rm = 6.69;                         % DC motor resistance [Om]
Kb = 0.468;                        % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                        % DC motor torque constant [Nm/A]
n = 1;                             % Gear ratio
fm = 0.0022;                       % friction coefficient between body & DC motor
fw = 0;


%%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;


    %initialise x1-x6 using the input 'x' vector
```

```
a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(x3)^2+4*M*R*L*cos(x3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(x3)-M^2*R*L^2*cos(x3)*g*sin(x3);
    e24=x4*sin(x3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(x3);

    e43=M*g*L*sin(x3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=x4*sin(x3)*(-M^2*R^2*L^2*cos(x3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(x3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;
    %define these extra function to reduce the coding complexity of A and B
    %matrices for the x4dot equation


    %defined constants from motor literature
    alpha= n*Kt/Rm;
    beta = (n*Kt*Kb/Rm) + fm;

    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);

    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);

    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;
    %define these extra function to reduce the coding complexity of A and B
    %matrices with the motor added in

    %define A and B using a nonlinear state-space gyro robot model,
    %including the motor part

    if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
    end


 %%%% Select Matrix A model for testing %%%%
  % 1.Model A (primary)
  % 2.Model B
  % 3.Model AB10 mixed

     Model = 1;  % Select mode

  %%%-------- Model A ----------
if Model == 1
          %%% Primary Model A %%%%
     A4 =[0      1             0              0        ;
          0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)    ;
          0      0             0              1        ;
          0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)    ];


end
```

```
%%%-------- Model B ----------
if Model == 2
          %%%% Model B %%%%
     A4 =[0      1             0                    0         ;
          0  em22/(a+b)   (em23+em24*x4)/((a+b)*x3)   0         ;
          0      0             0                    1         ;
          0  em42/(a+b)   (em43+em44*x4)/((a+b)*x3)   0         ];
end


  %%%-------- Model AB10 ----------
if Model == 3

  %%% Mix A&B %%%%%

   if x3 < (10.5*pi/180) && x3 > (-10.5*pi/180)
    %%%% Primary Model A %%%%
     A4 =[0      1             0              0         ;
          0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)    ;
          0      0             0              1         ;
          0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)    ];

  else

      %%%% Model B %%%%
     A4 =[0      1             0                    0          ;
          0  em22/(a+b)   (em23+em24*x4)/((a+b)*x3)   0         ;
          0      0             0                    1         ;
          0  em42/(a+b)   (em43+em44*x4)/((a+b)*x3)   0        ] ;
  end
end

    % Models Matrix B

    B4 =[    0           0;
        fm21/(a+b)  fm22/(a+b);
            0           0;
        fm41/(a+b)  fm42/(a+b)];

    C= eye(4);


   %%% Riccati Eq for Kalman filter %%
   %%% Need P not K
   [~,P,~] = lqr(A4, C', Noise_W, Noise_V);
   %%% P = [4x4]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%       P_out= [  p11 p12 p13 p14  ;
%%%                 0   0   0   0    ;
%%%                 p31 p32 p33 p34  ;
%%%                 0   0   0   0   ];
%%%
%%% We need only 3rd row to estimate x3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if x3 == 1.0e-20 % get zero back
x3 = 0;
end

xx3(i)= x3*180/pi; % convert angle rad. to deg.
xx3 = double(xx3);

%%% Create P31-P34 table setting 1st coulum as x3
P_Riccati_2D_4s_P3(i+1+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column
```

```
P_Riccati_2D_4s_P3(i+2+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column
P_Riccati_2D_4s_P3(i+3+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column
P_Riccati_2D_4s_P3(i+4+(3*shift_i),1)= xx3(i);  % Insert x3 deg at 1st column

%%% Rearrange %%%
P31 = P(3,1);
P32 = P(3,2);
P33 = P(3,3);
P34 = P(3,4);

P_Riccati_2D_4s_P3( i+1+(3*shift_i) ,j+1)= P31;  % start at 2nd column
P_Riccati_2D_4s_P3( i+2+(3*shift_i) ,j+1)= P32;
P_Riccati_2D_4s_P3( i+3+(3*shift_i) ,j+1)= P33;
P_Riccati_2D_4s_P3( i+4+(3*shift_i) ,j+1)= P34;


i=i+1;  %x3
shift_i=shift_i+1;
   end

  xx4(j)= x4*180/pi;
  xx4 = double(xx4);

  P_Riccati_2D_4s_P3(1,j+1)= xx4(j);  % Insert x4 deg at 1st row


    i=1;
    shift_i=0;
  j=j+1; %x4

       %%% print
    Cal_percent = (j-1)*100/(ns-1);

    if mod(Cal_percent , 5) == 0
       fprintf('Calculating  %f percent ...\n',Cal_percent)
    end

end

save('local_LQG_2D_P_workspace','P_Riccati_2D_4s_P3');
```

Appendix B

Simulink Block Diagrams

5.2.4 Experimental Results

-LQR Control (*Filename: Gyroboy_LQR_10_2021.slx*)

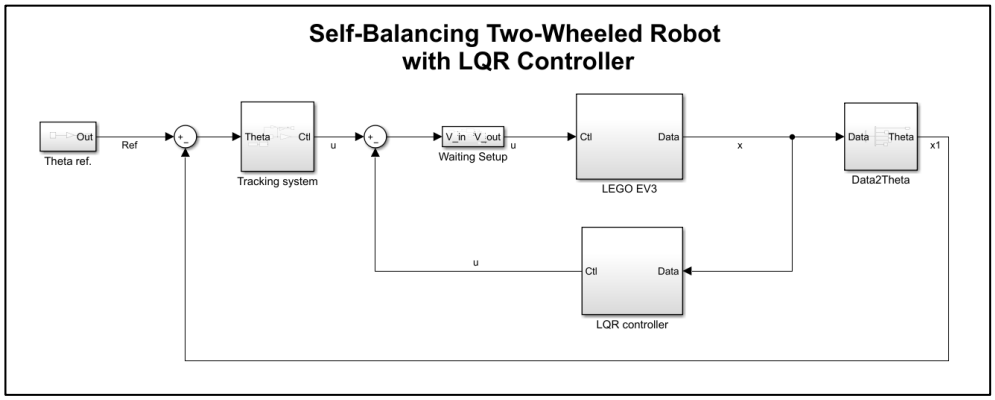


Figure B1.1: The 5-states control of linear quadratic regulator (LQR) in Simulink, adapted from (Roslovets, 2020)

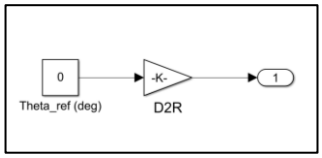


Figure B1.2: Theta (x_1) reference block diagrams

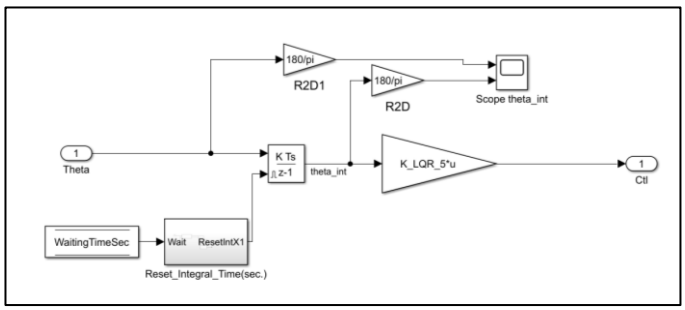


Figure B1.3: Tracking system block diagrams

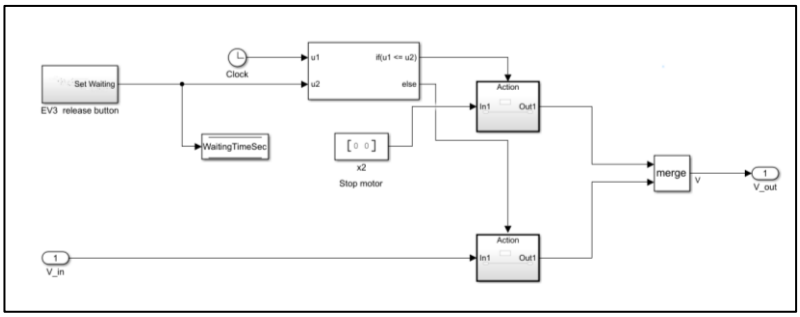


Figure B1.4: Waiting for setup block diagrams

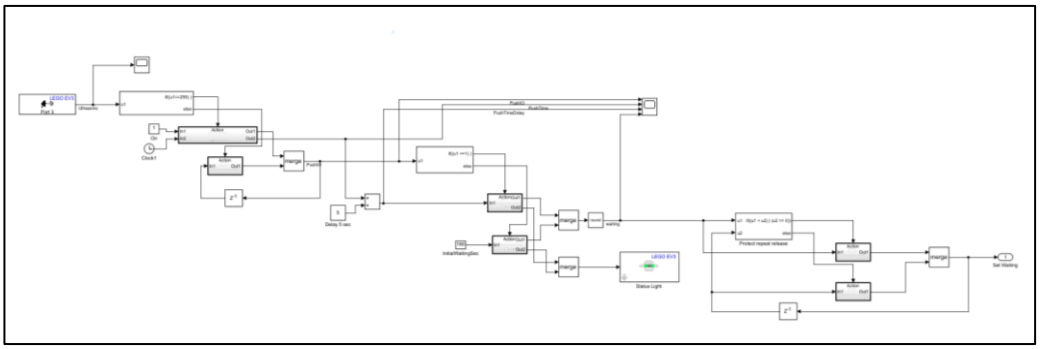


Figure B1.5: EV3 release button

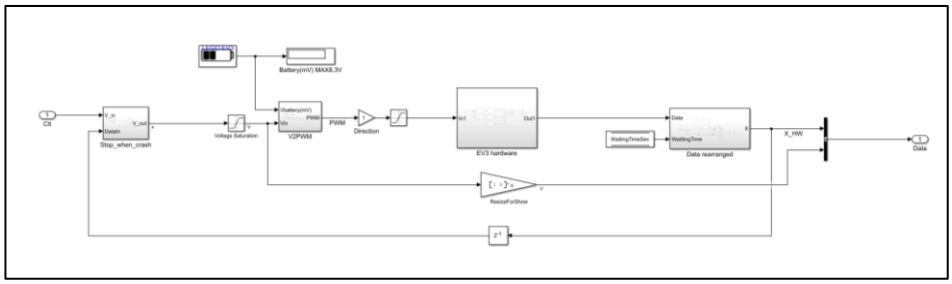


Figure B1.6: LEGO EV3 block diagrams

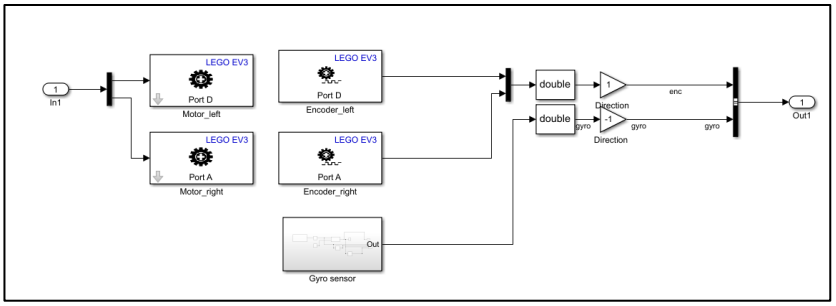


Figure B1.7: EV3 hardware block diagrams

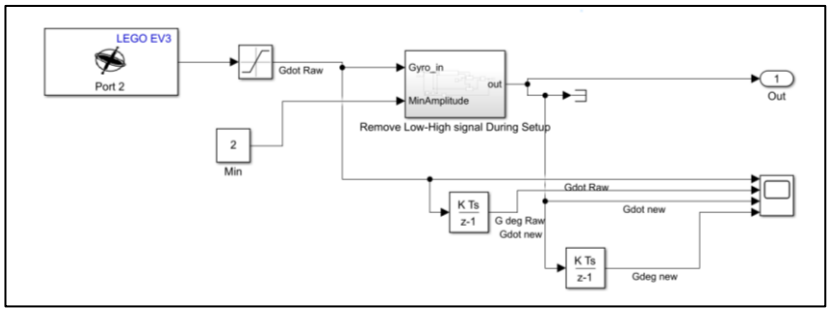


Figure B1.8: Gyro sensor block diagrams

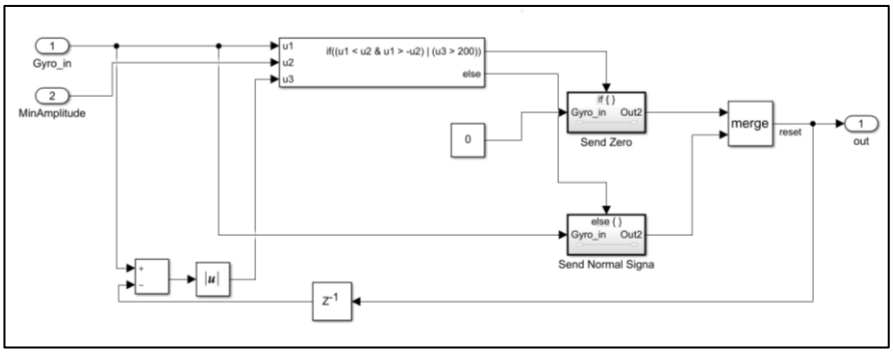


Figure B1.9: Remove Low High Signal During Setup block diagrams

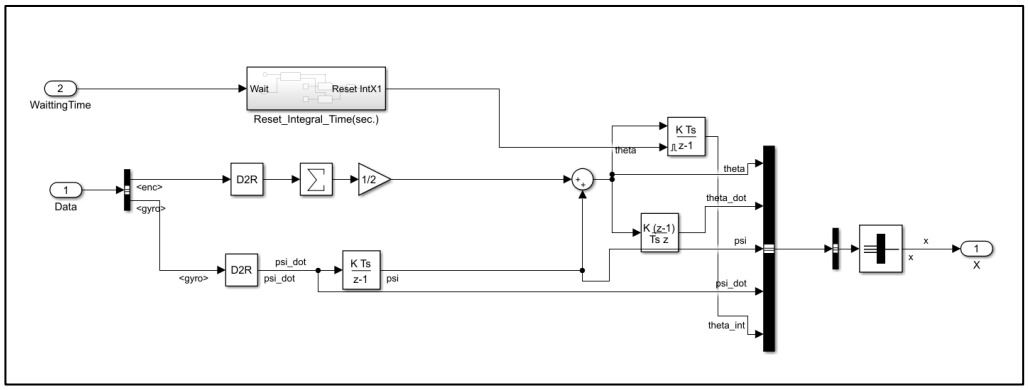


Figure B1.10: Data rearranged block diagrams

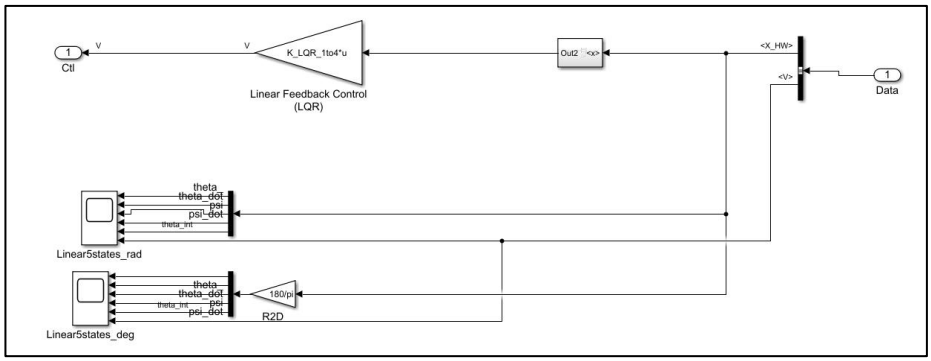


Figure B1.11: LQR Controller block diagrams

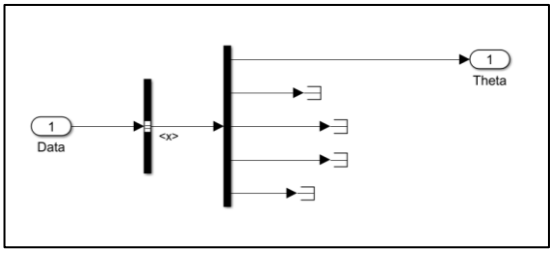


Figure B1.12: Data2theta

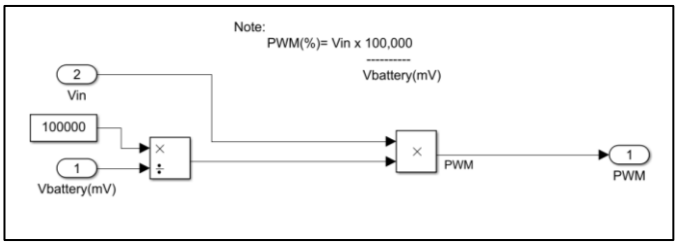


Figure B1.13: V2PWM block diagrams

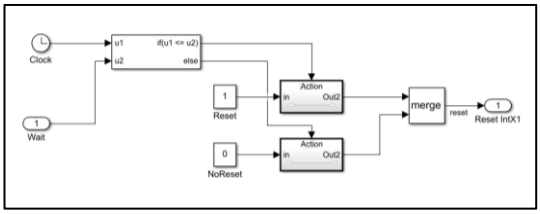


Figure B1.14: Reset Integral Time block diagrams

5.3.4 Experimental Results

- LQG Control *(Filename: Gyroboy_LQG_10_2021.slx)*

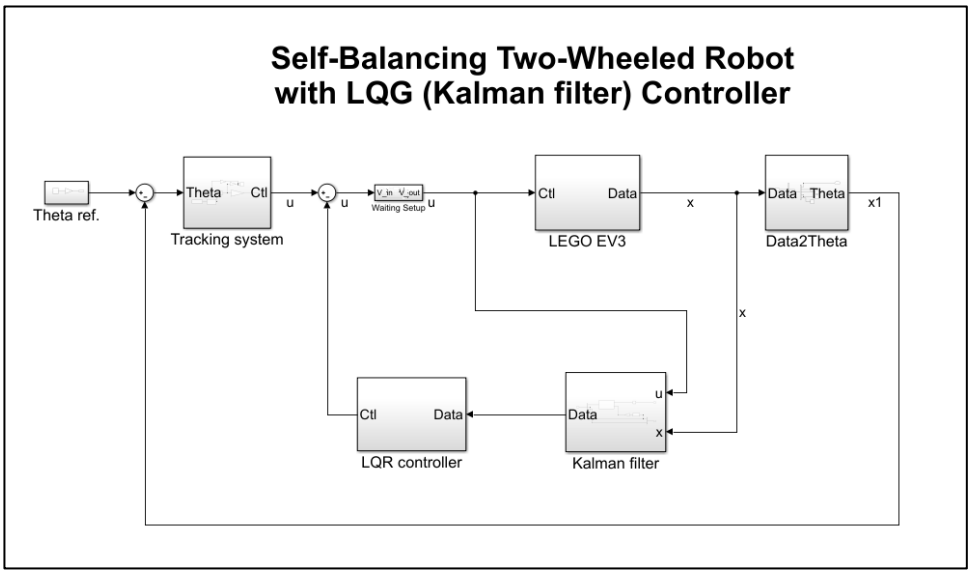


Figure B2.1: The linear quadratic Gaussian (LQG) control block diagrams in Simulink. Noticeably, merely pitch angle will be filtered.

As the LQG controller is extended from the LQR, some block diagrams are similar. Therefore, the different block diagrams are presented as follows:

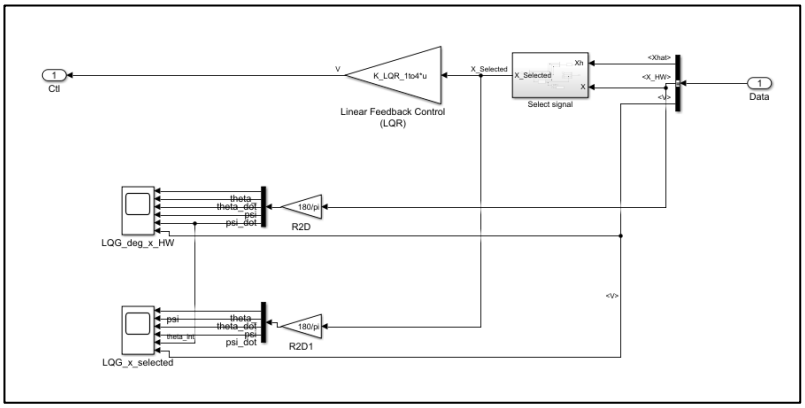


Figure B2.2: LQR controller block diagrams

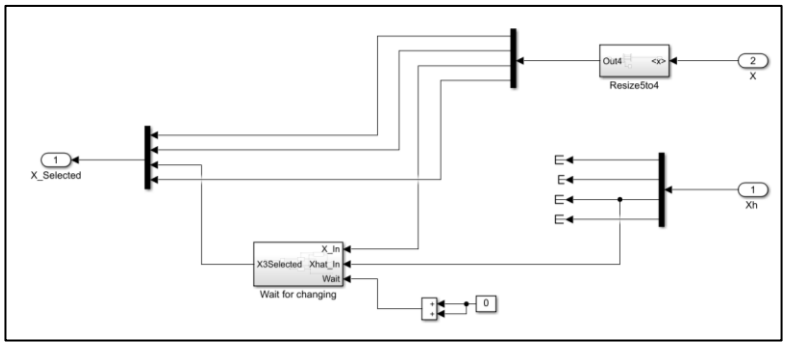


Figure B2.3: Select signal block diagrams

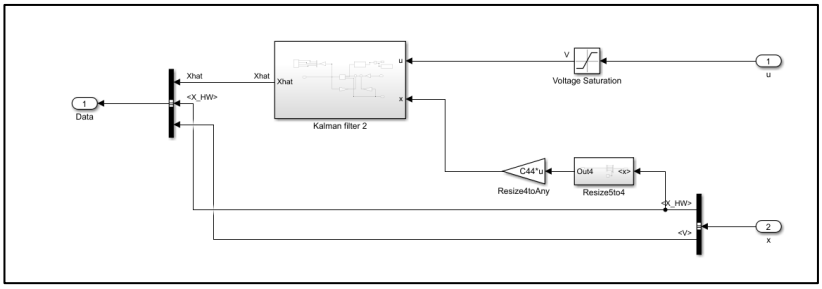


Figure B2.4: Kalman filter block diagrams

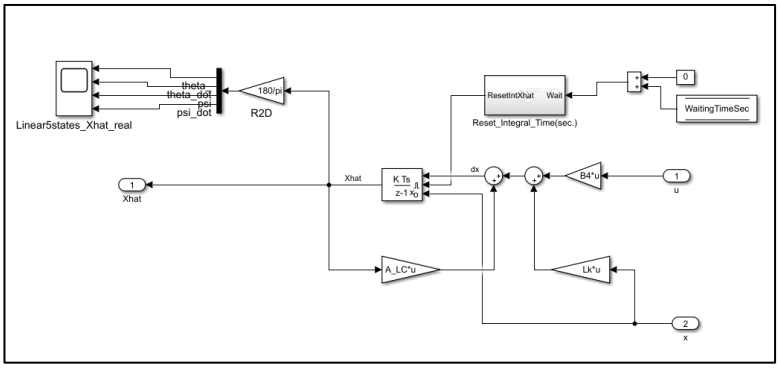


Figure B2.5: Kalman filter2 block diagrams

6.6 Experimental Results

-Freezing Technique *(Filename: Gyroboy_Freezing_10_2021.slx)*

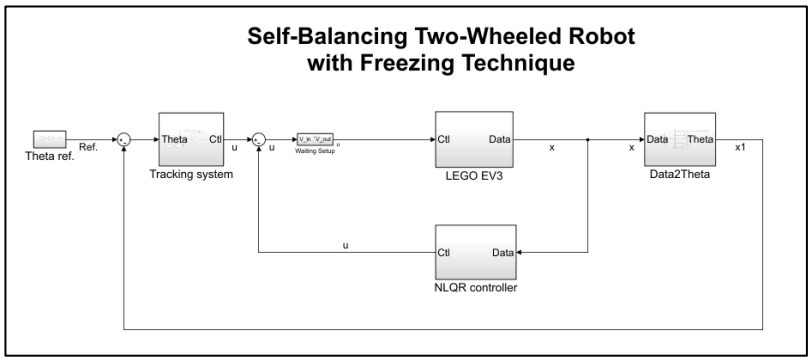


Figure B3.1: The nonlinear freezing control in Simulink.

Note that other block diagrams of freezing technique are similar to the LQR controller. Therefore, the NLQR controller block diagram in freezing control system, which includes lookup table function inside, is slightly different to the LQR programme, as shown in Figure B3.2.

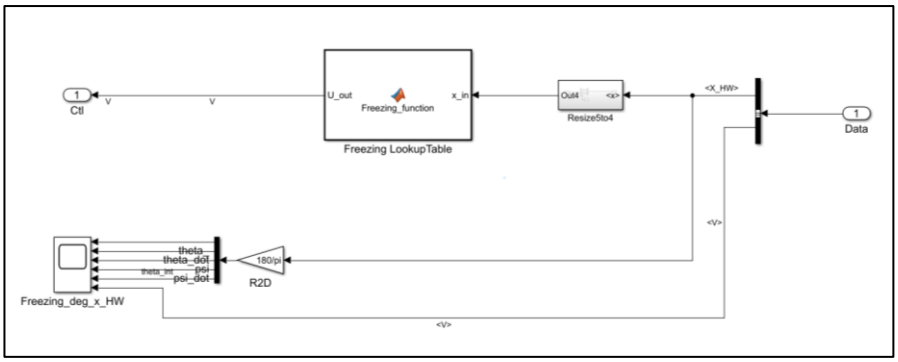


Figure B3.2: NLQR controller block diagrams.

The programming code of lookup table inside Freezing Lookup Table block diagram in Figure B3.2, as given by

```
function U_out    = Freezing_fucntion(x_in)

    U_out=zeros(2,1);

   %%% Lookup table matrix 165x54 %%%
          % K1-K4 Lookup table

       K_Riccati_2D_4s=zeros(165,54); %x3=1deg x4=5deg/s
       K_Riccati_2D_4s=[0   -130    -125    -120    -115    -110    -105 …
-20 -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39 …
-20 -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45 …
-20 -61.36  -61.38  -61.40  -61.42  -61.43  -61.45  -61.47  -61.49  -61.51 …
-20 -7.47   -7.47   -7.47   -7.47   -7.47   -7.47   -7.47   -7.47   -7.47 …
-19 -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39 …
-19 -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45 …
-19 -61.19  -61.20  -61.22  -61.24  -61.26  -61.27  -61.29  -61.31  -61.32 …
-19 -7.44   -7.44   -7.44   -7.44   -7.44   -7.44   -7.44   -7.44   -7.44 …
…
    % For example, lookup table between x3 = -20 to 20 deg and x4 = -130 to 130 deg/s.
    % Cannot present all table here

    x3=x_in(3);     % psi
    x4=x_in(4);     % psi dot


[size_m,size_n]=size(K_Riccati_2D_4s);

%%% find array information %%%
DegMaxN1 = K_Riccati_2D_4s(1,size_n);
DegMaxN2 = K_Riccati_2D_4s(1,size_n-1);
DegStepN = DegMaxN1-DegMaxN2; %x4
DegStepN = double(DegStepN);


DegMaxM1 = K_Riccati_2D_4s(size_m,1);
DegMaxM2 = K_Riccati_2D_4s(size_m-4,1);
DegStepM = DegMaxM1-DegMaxM2; %x3
DegStepM = double(DegStepM);



x_3 = x3*180/pi; % change to degree
x_3 = double(x_3);
x_4 = x4*180/pi; % change to degree
x_4 = double(x_4);
x3_i=0;
x4_j=0;

     %%%------- x_3 ---------

      if single(K_Riccati_2D_4s(size_m,1)) < x_3     %% Over

           x3_i = size_m-3;


      elseif single(K_Riccati_2D_4s(2,1)) > x_3 %% Under

           x3_i = 2;
```

```
      else

         i=2;
         while(~((single(x_3) >= (single(K_Riccati_2D_4s(i,1))-DegStepM/2)) &&
(single(x_3) < (single(K_Riccati_2D_4s(i,1))+DegStepM/2)) ))

        i=i+1;

        end
        x3_i = i;


      end

    %%%------- x_4 ---------

     if single(K_Riccati_2D_4s(1,size_n)) < x_4     %% Over

           x4_j = size_n;
      elseif single(K_Riccati_2D_4s(1,2)) > x_4 %% Under

           x4_j = 2;
      else                                       %% range

         j=2;
          while(~((single(x_4) >= (single(K_Riccati_2D_4s(1,j))-DegStepN/2)) &&
(single(x_4) < (single(K_Riccati_2D_4s(1,j))+DegStepN/2)) ))

         j=j+1;

         end

        x4_j = j;

     end



         K_out_p1 =  K_Riccati_2D_4s(x3_i,x4_j);
         K_out_p2 =  K_Riccati_2D_4s(x3_i+1,x4_j);
         K_out_p3 =  K_Riccati_2D_4s(x3_i+2,x4_j);
         K_out_p4 =  K_Riccati_2D_4s(x3_i+3,x4_j);
        % K_out_p5 =  K_Riccati_2D(x3_i+4,x4_j);   % Not use

         K_out2x4 = [K_out_p1,K_out_p2,K_out_p3,K_out_p4;
                     K_out_p1,K_out_p2,K_out_p3,K_out_p4];

                     %%% created [2x4] for two motors


  U_out = K_out2x4*x_in;

end
```

A partial lookup table of LQR feedback gains (Model A) demonstrated in Table B3.1.

Table B3.1: A partial lookup table of LQR feedback gains for the TWR Model A

x3\x4	-130	-125	-120	…	-10	-5	0	5	10	…	120	125	130
-20	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39
-20	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45
-20	-61.36	-61.38	-61.4	…	-61.8	-61.82	-61.83	-61.85	-61.87	…	-62.28	-62.29	-62.31
-20	-7.47	-7.47	-7.47	…	-7.48	-7.48	-7.48	-7.48	-7.48	…	-7.49	-7.49	-7.49
-19	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39
-19	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45
-19	-61.19	-61.2	-61.22	…	-61.6	-61.62	-61.63	-61.65	-61.67	…	-62.05	-62.07	-62.09
-19	-7.44	-7.44	-7.44	…	-7.45	-7.45	-7.45	-7.45	-7.45	…	-7.46	-7.46	-7.46
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
0	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39
0	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45
0	-59.85	-59.85	-59.85	…	-59.85	-59.85	-59.85	-59.85	-59.85	…	-59.85	-59.85	-59.85
0	-7.17	-7.17	-7.17	…	-7.17	-7.17	-7.17	-7.17	-7.17	…	-7.17	-7.17	-7.17
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
19	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39
19	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45
19	-62.09	-62.07	-62.05	…	-61.67	-61.65	-61.63	-61.62	-61.6	…	-61.22	-61.2	-61.19
19	-7.46	-7.46	-7.46	…	-7.45	-7.45	-7.45	-7.45	-7.45	…	-7.44	-7.44	-7.44
20	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39
20	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45
20	-62.31	-62.29	-62.28	…	-61.87	-61.85	-61.83	-61.82	-61.8	…	-61.4	-61.38	-61.36
20	-7.49	-7.49	-7.49	…	-7.48	-7.48	-7.48	-7.48	-7.48	…	-7.47	-7.47	-7.47

Moreover, the feedback gain variables in the Freezing Lookup Table in the block diagram in Simulink can be changed for other systems, i.e., a lookup table of Model B in Table B3.2.

Table B3.2: A partial lookup table of LQR feedback gains for the TWR Model B

x3\x4	-130	-125	-120	…	-10	-5	0	5	10	…	120	125	130
-20	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
-20	-1.42	-1.42	-1.42	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43
-20	-54.37	-54.48	-54.59	…	-57.31	-57.44	-57.58	-57.72	-57.86	…	-61.17	-61.34	-61.50
-20	-7.39	-7.39	-7.39	…	-7.42	-7.42	-7.42	-7.42	-7.42	…	-7.43	-7.43	-7.43
-19	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
-19	-1.42	-1.42	-1.42	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43
-19	-54.00	-54.11	-54.23	…	-57.09	-57.23	-57.38	-57.52	-57.67	…	-61.15	-61.32	-61.49
-19	-7.36	-7.36	-7.36	…	-7.39	-7.39	-7.39	-7.39	-7.39	…	-7.40	-7.40	-7.40
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
0	0	0	0	…	0	0	0	0	0	…	0	0	0
0	0	0	0	…	0	0	0	0	0	…	0	0	0
0	0	0	0	…	0	0	0	0	0	…	0	0	0
0	0	0	0	…	0	0	0	0	0	…	0	0	0
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
19	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
19	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	-1.42	-1.42	-1.42
19	-61.49	-61.32	-61.15	…	-57.67	-57.52	-57.38	-57.23	-57.09	…	-54.23	-54.11	-54.00
19	-7.40	-7.40	-7.40	…	-7.39	-7.39	-7.39	-7.39	-7.39	…	-7.36	-7.36	-7.36
20	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
20	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	0.14	0.14	0.14
20	-61.50	-61.34	-61.17	…	-57.86	-57.72	-57.58	-57.44	-57.31	…	4.12	4.12	4.12
20	-7.43	-7.43	-7.43	…	-7.42	-7.42	-7.42	-7.42	-7.42	…	0.70	0.70	0.70

Note that the feedback gains at x_3 = 0° are set manually as zero because this angle is uncontrollable for Model B, which MATLAB cannot obtain the feedback gains. Additionally, a partial lookup table of LQR feedback gains (Model AB) is presented in Table B3.3.

Table B3.3: A partial lookup table of LQR feedback gains for the TWR Model AB

x3\x4	-130	-125	-120	…	-10	-5	0	5	10	…	120	125	130
-20	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
-20	-1.42	-1.42	-1.42	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43
-20	-54.37	-54.48	-54.59	…	-57.31	-57.44	-57.58	-57.72	-57.86	…	-61.17	-61.34	-61.50
-20	-7.39	-7.39	-7.39	…	-7.42	-7.42	-7.42	-7.42	-7.42	…	-7.43	-7.43	-7.43
-19	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
-19	-1.42	-1.42	-1.42	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43
-19	-54.00	-54.11	-54.23	…	-57.09	-57.23	-57.38	-57.52	-57.67	…	-61.15	-61.32	-61.49
-19	-7.36	-7.36	-7.36	…	-7.39	-7.39	-7.39	-7.39	-7.39	…	-7.40	-7.40	-7.40
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
0	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39	-1.39	-1.39	…	-1.39	-1.39	-1.39
0	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45	-1.45	-1.45	…	-1.45	-1.45	-1.45
0	-59.85	-59.85	-59.85	…	-59.85	-59.85	-59.85	-59.85	-59.85	…	-59.85	-59.85	-59.85
0	-7.17	-7.17	-7.17	…	-7.17	-7.17	-7.17	-7.17	-7.17	…	-7.17	-7.17	-7.17
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
19	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
19	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	-1.42	-1.42	-1.42
19	-61.49	-61.32	-61.15	…	-57.67	-57.52	-57.38	-57.23	-57.09	…	-54.23	-54.11	-54.00
19	-7.40	-7.40	-7.40	…	-7.39	-7.39	-7.39	-7.39	-7.39	…	-7.36	-7.36	-7.36
20	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38	-1.38	-1.38	…	-1.38	-1.38	-1.38
20	-1.43	-1.43	-1.43	…	-1.43	-1.43	-1.43	-1.43	-1.43	…	-1.42	-1.42	-1.42
20	-61.50	-61.34	-61.17	…	-57.86	-57.72	-57.58	-57.44	-57.31	…	-54.59	-54.48	-54.37
20	-7.43	-7.43	-7.43	…	-7.42	-7.42	-7.42	-7.42	-7.42	…	-7.39	-7.39	-7.39

The gains K have been stored in matrix 4x1, including feedback gains K1-K4, which are varied by state variables x_3 and x_4, as shown in Table B3.4. Note, a gain K5 of state variable x_5 has been excluded as it has not changed by any x_3 and x_4. Moreover, the memory of LEGO EV3 is limited for data storage. Therefore, the gain K5 has not been used in the lookup table, but it has been used as a fixed gain in Simulink.

Table B3.4: Gains K1-K4 in the lookup table.

x3\x4	-130
-20	K1
-20	K2
-20	K3
-20	K4

-Freezing Technique with EKF

(Filename: Gyroboy_FreezEKF_10_2021.slx)

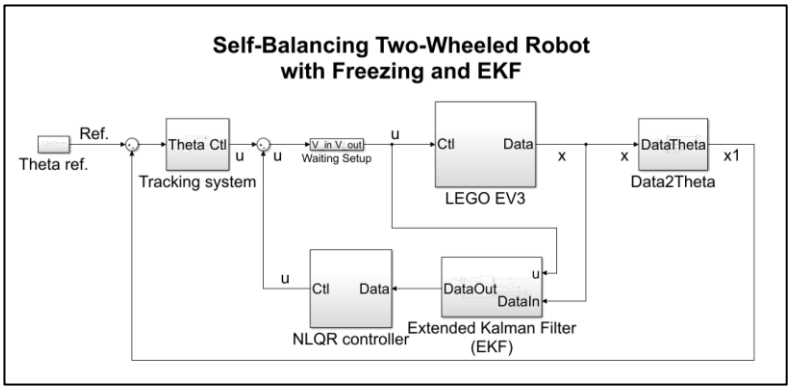


Figure B3.3: The nonlinear freezing control with EKF in Simulink.

Note that other block diagrams of freezing technique with EKF are similar to the LQG controller. Therefore, the different block diagrams are presented as follows:

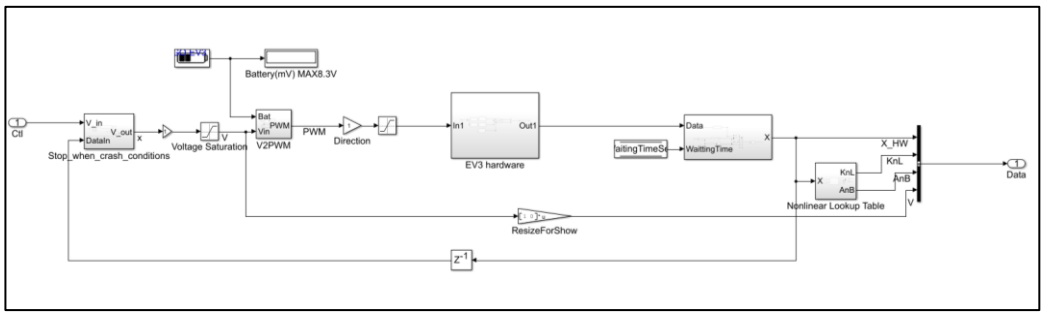


Figure B3.4: LEGO EV3 block diagrams.

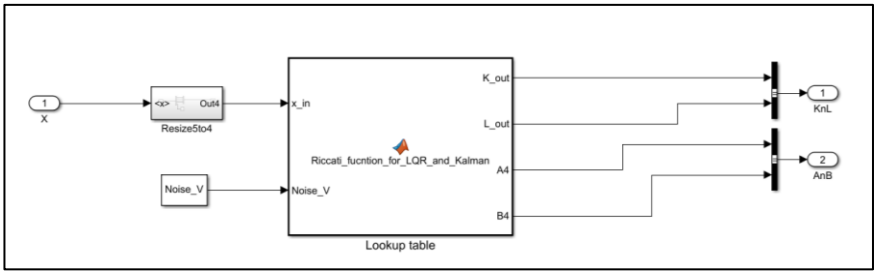


Figure B3.5: Nonlinear Lookup table block diagrams.

The programming code of lookup table inside Lookup Table block diagram in Figure B3.5, as given by

```
function [K_out,L_out,A4,B4]    =
Riccati_fucntion_for_LQR_and_Kalman(x_in,Noise_V)

    K_out=zeros(2,4);

   L_out = zeros(4,4);
   A4 = zeros(4,4);
   B4 = zeros(4,2);

    %Noise_W = 1;   %%% 5***
    %Noise_V = 0.3; %%% 1***
    C4 = eye(4);

       % K1-K4 Lookup table
      %Q11=20 scale  x3=1deg x4=5deg/s    20 deg
       K_Riccati_2D_4s=zeros(165,54);
K_Riccati_2D_4s=[0   -130    -125    -120    -115    -110    -105    -100    -95 -90…
-20 -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39 …
-20 -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45 …
-20 -61.36  -61.38  -61.40  -61.42  -61.43  -61.45  -61.47  -61.49  -61.51  -61.52 …
-20 -7.47   -7.47   -7.47   -7.47   -7.47   -7.47   -7.47   -7.47   -7.47   -7.47 …
-19 -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39   -1.39 …
-19 -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45   -1.45 …
```

```
-19 -61.19  -61.20  -61.22  -61.24  -61.26  -61.27  -61.29  -61.31  -61.32  -61.34 …
-19 -7.44   -7.44   -7.44   -7.44   -7.44   -7.44   -7.44   -7.44   -7.44   -7.44 …
…

    % For example, lookup table between x3 = -20 to 20 deg and x4 = -130 to 130 deg/s.
    % Cannot present all table here
    % P solution of Kalman filter
    % P31-P34 Lookup table
   % Q11=20
      P_Riccati_2D_4s_P3=zeros(165,54);    % W=1 % V=0.2
P_Riccati_2D_4s_P3=[0 -130    -125    -120    -115    -110    -105    -100    -95 -90 …
-20 0.08    0.08    0.08    0.08    0.08    0.08    0.08    0.08    0.08    0.08 …
-20 0.14    0.14    0.14    0.14    0.14    0.14    0.14    0.14    0.14    0.14 …
-20 4.12    4.12    4.12    4.12    4.11    4.11    4.11    4.11    4.11    4.11 …
-20 0.70    0.70    0.70    0.70    0.70    0.70    0.70    0.70    0.70    0.70 …
-19 0.08    0.08    0.08    0.08    0.08    0.08    0.08    0.08    0.08    0.08 …
-19 0.14    0.14    0.14    0.14    0.14    0.14    0.14    0.14    0.14    0.14 …
-19 4.12    4.12    4.12    4.12    4.12    4.11    4.11    4.11    4.11    4.11 …
-19 0.70    0.70    0.70    0.70    0.70    0.70    0.70    0.70    0.70    0.70 …
…

   % For example, lookup table between x3 = -20 to 20 deg and x4 = -130 to 130 deg/s.
    % Cannot present all table here

    %x1=x_in(1);      % theta
    %x2=x_in(2);      % theta dot
    x3=x_in(3);      % psi
    x4=x_in(4);      % psi dot
  %  x5=x_in(5);      % theta int


[size_m,size_n]=size(K_Riccati_2D_4s);


%%% find array information %%%
DegMaxN1 = K_Riccati_2D_4s(1,size_n);
DegMaxN2 = K_Riccati_2D_4s(1,size_n-1);
DegStepN = DegMaxN1-DegMaxN2; %x4
DegStepN = double(DegStepN);


DegMaxM1 = K_Riccati_2D_4s(size_m,1);
DegMaxM2 = K_Riccati_2D_4s(size_m-4,1);
DegStepM = DegMaxM1-DegMaxM2; %x3
DegStepM = double(DegStepM);



%x3=-80*pi/180;x4=-60*pi/180;

x_3 = x3*180/pi; % change to degree
x_3 = double(x_3);
x_4 = x4*180/pi; % change to degree
x_4 = double(x_4);
x3_i=0;
x4_j=0;

     %%%------- x_3 ---------

      if single(K_Riccati_2D_4s(size_m,1)) < x_3     %% Over

           x3_i = size_m-3;
           %%% Go to wide data


      elseif single(K_Riccati_2D_4s(2,1)) > x_3 %% Under
```

```
            x3_i = 2;
            %%% Go to wide data


       else                                    %% range

          i=2;
          while(~((single(x_3) >= (single(K_Riccati_2D_4s(i,1))-DegStepM/2)) &&
(single(x_3) < (single(K_Riccati_2D_4s(i,1))+DegStepM/2)) ))

         i=i+1;

         end
         x3_i = i;


       end

     %%%------- x_4 ---------

      if single(K_Riccati_2D_4s(1,size_n)) < x_4     %% Over

            x4_j = size_n;
       elseif single(K_Riccati_2D_4s(1,2)) > x_4 %% Under

            x4_j = 2;
       else                                    %% range

          j=2;
           while(~((single(x_4) >= (single(K_Riccati_2D_4s(1,j))-DegStepN/2))
&& (single(x_4) < (single(K_Riccati_2D_4s(1,j))+DegStepN/2)) ))

          j=j+1;

          end

         x4_j = j;

      end


          K_out_p1 =  K_Riccati_2D_4s(x3_i,x4_j);
          K_out_p2 =  K_Riccati_2D_4s(x3_i+1,x4_j);
          K_out_p3 =  K_Riccati_2D_4s(x3_i+2,x4_j);
          K_out_p4 =  K_Riccati_2D_4s(x3_i+3,x4_j);
         % K_out_p5 =  K_Riccati_2D(x3_i+4,x4_j);

          K_out2x4 = [K_out_p1,K_out_p2,K_out_p3,K_out_p4;  %%% created [2x4]
for two motors
                      K_out_p1,K_out_p2,K_out_p3,K_out_p4];


%%%%% Use same table index with K %%%%

          p31 =  P_Riccati_2D_4s_P3(x3_i,x4_j);
          p32 =  P_Riccati_2D_4s_P3(x3_i+1,x4_j);
          p33 =  P_Riccati_2D_4s_P3(x3_i+2,x4_j);
          p34 =  P_Riccati_2D_4s_P3(x3_i+3,x4_j);



   P_out= [  0   0   0   0  ;
             0   0   0   0  ;
           p31 p32 p33 p34  ;
```

```
             0    0    0    0   ];

%%% Physical constants
g = 9.81;                          % gravity acceleration [m/sec^2]
%%% Physical parameters
m = 0.050;                         % wheel weight [kg]  old 0.024 new 0.050
R = 0.027;                         % wheel radius [m]
Jw = m * R^2 / 2;                  % wheel inertia moment [kgm^2]
W = 0.105;                         % body width [m]
D = 0.1;                           % body depth [m]

M = 0.64;                       % body weight [kg]  old 0.80  new 0.64
h = 0.210;                         % body height [m]

L = h / 2;                         % distance of the center of mass from the
wheel axle [m] 10.5

Jpsi = M * L^2 / 3;                % body pitch inertia moment [kgm^2]

%%% Motors parameters
Jm = 1e-5;                         % DC motor inertia moment [kgm^2]
Rm = 6.69;                         % DC motor resistance [Om]
Kb = 0.468;                        % DC motor back EMF constant [Vsec/rad]
Kt = 0.317;                        % DC motor torque constant [Nm/A]
n = 1;                             % Gear ratio
fm = 0.0022;                       % friction coefficient between body & DC motor
fw = 0;                            % friction coefficient between wheel & floor

%%% Helping variables
alpha = n * Kt / Rm;
beta = n * Kt * Kb / Rm + fm;



%%%% AB

    %initialise x1-x6 using the input 'x' vector
    if x3 == 0
       x3=1.0e-20;  % avoid Inf's and NaN's
    end



a=2*Jw*Jpsi+2*m*R^2*M*L^2+4*m*R^2*n^2*Jm+2*M*R^2*n^2*Jm+2*n^2*Jm*M*L^2+2*m*R^2
*Jpsi+M*R^2*Jpsi+2*Jw*M*L^2+4*Jw*n^2*Jm+2*n^2*Jm*Jpsi;
    b=M^2*R^2*L^2*sin(x3)^2+4*M*R*L*cos(x3)*n^2*Jm;
    e23=2*n^2*Jm*M*g*L*sin(x3)-M^2*R*L^2*cos(x3)*g*sin(x3);
    e24=x4*sin(x3)*(M^2*R*L^3 +2*M*R*L*n^2*Jm +M*R*L*Jpsi);

    f21=M*L^2+2*n^2*Jm+Jpsi;
    f22=2*n^2*Jm-M*R*L*cos(x3);


    e43=M*g*L*sin(x3)*(2*Jw+2*m*R^2+M*R^2+2*n^2*Jm);
    e44=x4*sin(x3)*(-M^2*R^2*L^2*cos(x3)+2*M*R*L*n^2*Jm);

    f41=2*n^2*Jm-M*R*L*cos(x3);
    f42=2*n^2*Jm+2*Jw+2*m*R^2+M*R^2;


    em22=2*beta*(f22-f21)-2*fw*f21;
    em23=e23;
    em24=e24+2*beta*(f21-f22);
```

```
    em42=2*beta*(f42-f41)-2*fw*f41;
    em43=e43;
    em44=e44+2*beta*(f41-f42);


    fm21=alpha*(f21-f22);
    fm22=fm21;
    fm41=alpha*(f41-f42);
    fm42=fm41;


    A4 =[0      1             0              0        ;
        0  em22/(a+b)  em23/((a+b)*x3)  em24/(a+b)    ;
        0      0             0              1        ;
        0  em42/(a+b)  em43/((a+b)*x3)  em44/(a+b)    ];

         B4 =[    0           0;
            fm21/(a+b)  fm22/(a+b);
            0           0;
            fm41/(a+b)  fm42/(a+b)];

%%%% AB %%%%

 L_out = P_out*C4'*inv(Noise_V);

  K_out= K_out2x4;
end
```

It can be seen that the lookup table of the EKF gains has been added to the function.

A partial lookup table of EKF feedback gains (Model A) shows in Table B3.5.

Table B3.5: A partial lookup table of Kalman feedback gains for the TWR model A

x3\x4	-130	-125	-120	…	-10	-5	0	5	10	…	120	125	130
-20	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
-20	0.14	0.14	0.14	…	0.14	0.14	0.14	0.14	0.14	…	0.13	0.13	0.13
-20	4.12	4.12	4.12	…	4.09	4.09	4.09	4.09	4.09	…	4.06	4.06	4.06
-20	0.70	0.70	0.70	…	0.69	0.69	0.69	0.69	0.69	…	0.68	0.68	0.68
-19	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
-19	0.14	0.14	0.14	…	0.14	0.14	0.14	0.14	0.14	…	0.13	0.13	0.13
-19	4.12	4.12	4.12	…	4.09	4.09	4.09	4.09	4.09	…	4.07	4.07	4.07
-19	0.70	0.70	0.70	…	0.69	0.69	0.69	0.69	0.69	…	0.68	0.68	0.68
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
0	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
0	0.14	0.14	0.14	…	0.14	0.14	0.14	0.14	0.14	…	0.14	0.14	0.14
0	4.12	4.12	4.12	…	4.12	4.12	4.12	4.12	4.12	…	4.12	4.12	4.12
0	0.69	0.69	0.69	…	0.69	0.69	0.69	0.69	0.69	…	0.69	0.69	0.69
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
19	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
19	0.13	0.13	0.13	…	0.14	0.14	0.14	0.14	0.14	…	0.14	0.14	0.14
19	4.07	4.07	4.07	…	4.09	4.09	4.09	4.09	4.09	…	4.12	4.12	4.12
19	0.68	0.68	0.68	…	0.69	0.69	0.69	0.69	0.69	…	0.70	0.70	0.70
20	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
20	0.13	0.13	0.13	…	0.14	0.14	0.14	0.14	0.14	…	0.14	0.14	0.14
20	4.06	4.06	4.06	…	4.09	4.09	4.09	4.09	4.09	…	4.12	4.12	4.12
20	0.68	0.68	0.68	…	0.69	0.69	0.69	0.69	0.69	…	0.70	0.70	0.70

Moreover, a partial lookup table of EKF feedback gains (Model B) shows in Table B3.6.

Table B3.6: A partial lookup table of Kalman feedback gains for the TWR model B

x3\x4	-130	-125	-120	…	-10	-5	0	5	10	…	120	125	130
-20	0.03	0.03	0.03	…	-0.01	-0.01	-0.01	-0.02	-0.02	…	-0.05	-0.05	-0.05
-20	0.14	0.14	0.14	…	0.13	0.13	0.13	0.13	0.13	…	0.11	0.11	0.11
-20	4.18	4.14	4.11	…	4.01	4.04	4.07	4.11	4.15	…	5.47	5.55	5.62
-20	0.65	0.65	0.65	…	0.66	0.66	0.66	0.66	0.66	…	0.65	0.65	0.65
-19	0.04	0.03	0.03	…	-0.01	-0.01	-0.01	-0.02	-0.02	…	-0.05	-0.05	-0.06
-19	0.14	0.14	0.14	…	0.13	0.13	0.13	0.13	0.13	…	0.11	0.11	0.11
-19	4.23	4.19	4.15	…	4.01	4.04	4.08	4.11	4.15	…	5.58	5.66	5.74
-19	0.03	0.03	0.03	…	-0.01	-0.01	-0.01	-0.02	-0.02	…	-0.05	-0.05	-0.05
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
0	-0.18	-0.18	-0.18	…	-0.18	-0.18	-0.02	0.18	0.18	…	0.18	0.18	0.18
0	-0.44	-0.44	-0.44	…	-0.44	-0.44	0.13	0.44	0.44	…	0.44	0.44	0.44
0	1×10^{11}	9.9×10^{10}	9.7×10^{10}	…	2.8×10^{10}	2×10^{10}	4.12	1.8×10^{10}	2.5×10^{10}	…	8.9×10^{10}	9.1×10^{10}	9.2×10^{10}
0	0.30	0.30	0.30	…	0.30	0.30	0.66	0.10	0.10	…	0.10	0.10	0.10
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
19	-0.06	-0.05	-0.05	…	-0.02	-0.02	-0.01	-0.01	-0.01	…	0.03	0.03	0.04
19	0.11	0.11	0.11	…	0.13	0.13	0.13	0.13	0.13	…	0.14	0.14	0.14
19	5.74	5.66	5.58	…	4.15	4.11	4.08	4.04	4.01	…	4.15	4.19	4.23
19	0.65	0.65	0.65	…	0.66	0.66	0.66	0.66	0.66	…	0.65	0.65	0.64
20	-0.05	-0.05	-0.05	…	-0.02	-0.02	-0.01	-0.01	-0.01	…	0.03	0.03	0.03
20	0.11	0.11	0.11	…	0.13	0.13	0.13	0.13	0.13	…	0.14	0.14	0.14
20	5.62	5.55	5.47	…	4.15	4.11	4.07	4.04	4.01	…	4.11	4.14	4.18
20	0.65	0.65	0.65	…	0.66	0.66	0.66	0.66	0.66	…	0.70	0.70	0.70

Furthermore, a partial lookup table of EKF feedback gains (Model AB) is given in Table B3.7.

Table B3.7: A partial lookup table of Kalman feedback gains for the TWR Model AB

x3\x4	-130	-125	-120	…	-10	-5	0	5	10	…	120	125	130
-20	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
-20	0.14	0.14	0.14	…	0.14	0.14	0.14	0.14	0.14	…	0.13	0.13	0.13
-20	4.12	4.12	4.12	…	4.09	4.09	4.09	4.09	4.09	…	4.06	4.06	4.06
-20	0.70	0.70	0.70	…	0.69	0.69	0.69	0.69	0.69	…	0.68	0.68	0.68
-19	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
-19	0.14	0.14	0.14	…	0.14	0.14	0.14	0.14	0.14	…	0.13	0.13	0.13
-19	4.12	4.12	4.12	…	4.09	4.09	4.09	4.09	4.09	…	4.07	4.07	4.07
-19	0.70	0.70	0.70	…	0.69	0.69	0.69	0.69	0.69	…	0.68	0.68	0.68
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
0	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
0	0.14	0.14	0.14	…	0.14	0.14	0.14	0.14	0.14	…	0.14	0.14	0.14
0	4.12	4.12	4.12	…	4.12	4.12	4.12	4.12	4.12	…	4.12	4.12	4.12
0	0.69	0.69	0.69	…	0.69	0.69	0.69	0.69	0.69	…	0.69	0.69	0.69
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
19	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
19	0.13	0.13	0.13	…	0.14	0.14	0.14	0.14	0.14	…	0.14	0.14	0.14
19	4.07	4.07	4.07	…	4.09	4.09	4.09	4.09	4.09	…	4.12	4.12	4.12
19	0.68	0.68	0.68	…	0.69	0.69	0.69	0.69	0.69	…	0.70	0.70	0.70
20	0.08	0.08	0.08	…	0.08	0.08	0.08	0.08	0.08	…	0.08	0.08	0.08
20	0.13	0.13	0.13	…	0.14	0.14	0.14	0.14	0.14	…	0.14	0.14	0.14
20	4.06	4.06	4.06	…	4.09	4.09	4.09	4.09	4.09	…	4.12	4.12	4.12
20	0.68	0.68	0.68	…	0.69	0.69	0.69	0.69	0.69	…	0.70	0.70	0.70

The Simulink function in freezing control with EKF includes two lookup tables. The first table is for the gains of the normal freezing controller, which was presented previously, and the second is for the extended Kalman filter. The gains have been stored as matrix 4x1. Moreover, only the 3rd row of matrix P solution is achieved by solving the Riccati equation, which gives the feedback gain of state variable x_3, as shown in Table B3.8.

Table B3.8: Gains P31-34 in the lookup table

x3\x4	-130
-20	P31
-20	P32
-20	P33
-20	P34

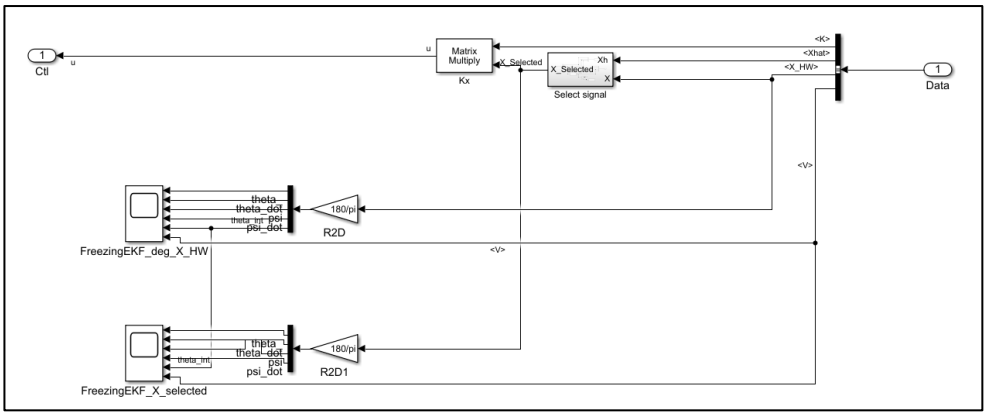


Figure B3.6: LQR Controller block diagrams

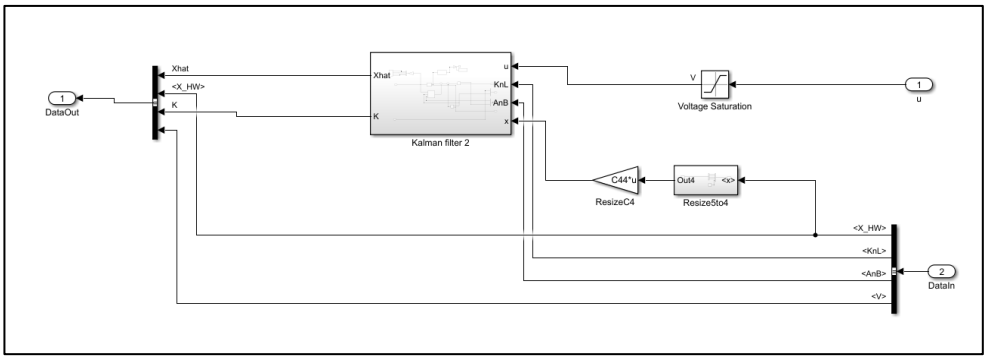


Figure B3.7: Extended Kalman filter block diagrams

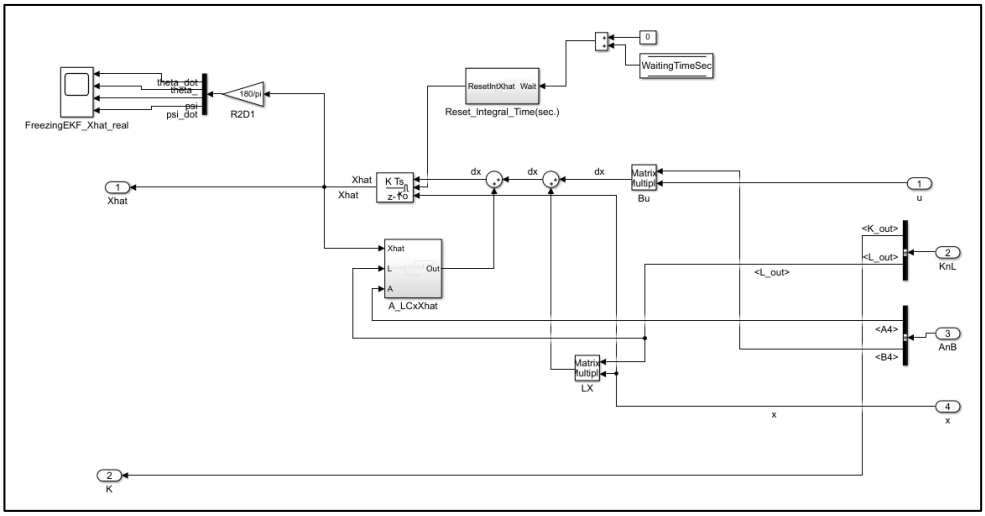


Figure B3.8: Kalman filter2 block diagrams

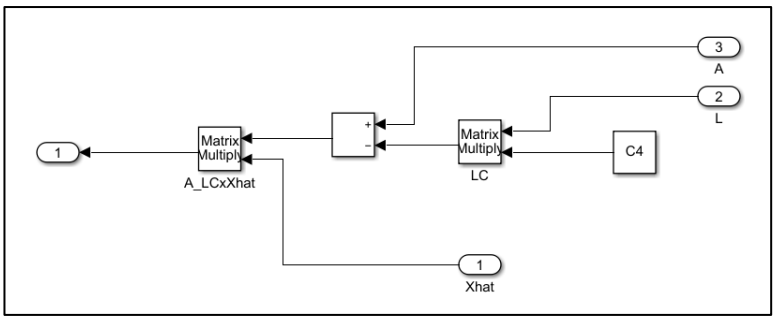


Figure B3.9: A_LCxXhat block diagrams