Development of BBot, a step traversing two-wheeled robot with active airborne control

空中姿勢制御による段差を走破できる BBot ロボットの研究開発

February, 2017

Huei Ee YAP
ヤップ　フェイ　イー

Development of BBot, a step traversing two-wheeled robot with active airborne control

空中姿勢制御による段差を走破できる BBot ロボットの研究開発

February, 2017

Research on Measurement and Information Technology
Department of Pure and Applied Physics
Graduate School of Advanced Science and Engineering
Waseda University

Huei Ee YAP
ヤップ　フェイ　イー

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Shuji Hashimoto, whose vast knowledge and expertise, continuous support, patience and trust, enables me to complete my dissertation. I appreciate his constructive feedback and excellent guidance throughout my graduate experience.

I would also like to acknowledge Prof. Hirochika Nakajima, Prof. Atsushi Takeuchi and Prof. Shigeo Morishima who gave me a lot of constructive feedback on my thesis and corrections of my dissertation.

A very special thanks goes to Dr. Klaus Petersen and Dr. Zhuohua Lin, without whose invaluable support and encouragement I would have given up on pursuing my PhD degree. I would like to thank Prof. Robert Shiurba for his feedback and suggestions on my writing. I would like to thank all members in the laboratory for the constant support and assistance they provided at all levels of the research.

Last but not least I would like to thank my family for the support they provided me throughout my entire life. Without their trust and support I would not have been able to pursue my own path in life.

ABSTRACT

Human friendly environment often poses challenging problems for mobile robot navigation. Terrains such as entrance steps, stairs are non-continuous and difficult to navigate. Many mechanism has been proposed to tackle this problem. Most common solution involve using a crawler belt wheels to increase the mobility of the robot, in the expense of increased weight and reduced speed. Wheels with adaptive legs in lunar rovers such as the NASA Soujourner [3][4] and the Ecole Polytechnique Federale de Lausanne (EPFL) Shrimp robot [5] uses a combination of movable wheels to adapt and traverse non-continuous terrains. This mechanism increases the complexity of the structural and driving mechanism of the robot. More advance approach such a biped humanoid robot mimics human gait to traverse complex terrain. Such robotic platform requirement highly complex mechanism and control system.

The limitation of the above mentioned approaches is the relatively slow moving speed when traversing non-continuous stepped terrain. These approaches use a relatively static manner to keep the robot balance when traversing an obstacle. By contrast, animals tend to approach a stepped terrain in a more dynamic manner. When jumping up or down a step, human tend to utilize forward momentum to push ourselves forward so that we are able to overcome the obstacle in a dynamic and efficient manner. The main challenge of this research is to design an approach to enable mobile robots to negotiate a stepped terrain efficiently.

In this thesis, we introduce BBot [28][29], a two wheeled mobile robot that is able to traverse common terrains such as steps and stairs in an efficient manner. BBot is a two wheeled robot consists of a lower body with wheels and a movable upper body connected by springs. The robot hops using the impact force released by pre-tensioned springs. Dynamic nature and simple design combine sturdy construction and complexity. Static instability requires that the attitude of the airborne robot be under active control to ensure a balanced landing. Torque generated by rotating the drive wheels determines the angle of body tilt. BBot

can descend step terrains and leap over gap obstacles swiftly and securely. The experiment results shows that our prototype BBot is able to negotiate step terrains in an efficient and reliable manner. Our proposed balance method expands the traversable environment of a conventional two wheeled robot.

CONTENTS

LIST OF ABBREVIATIONS

EKF	Extended Kalman filter
IMU	Inertial measurement unit
LQR	Linear quadratic regulator
MEMS	Micro-electro-mechanical system
PCB	Printed circuit board
TWP	Two wheeled pendulum
RWP	Reaction wheel pendulum
AGV	Autonomous ground vehicle

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

1.1 Background

Mobile intelligent robots are expected to play an increasing role in aiding humans in various tasks. Advancements in robotic technologies have enabled increasing number of robots to be deployed in the fields of exploration, surveillance, health care, and entertainment. Robots that need to operate in an uncontrolled human environment will have to be able to navigate stepped and uneven terrain. Properly implemented control methods will ensure the usability and safe deployment of these robots. Different designs and methods were developed to tackle the stair climbing problem. Different designs are summarized below.

1.1.1 Rocker Boogie Design

Many existing stair climber robots, lunar explorers use rocker-bogie mechanisms or crawler mechanisms. Rocker-bogie design used in lunar rovers, for example the Sojourner developed by NASA [3][4] and the Shrimp robot developed by EPFL [5], uses a combination of adaptive legs with the efficiency of wheels to traverse uneven terrain and steps. Robots using these mechanisms rely on the static stability of the platform to perform step traversal. Relying on static stability has drawbacks of low moving speed as well as increased complexity in the structure design. Any external perturbation which forces these robots outside their basin of stability will lead to loss of control and inability to recover.

1.1.2 Legged robot

Biped robots that mimic human gait have been developed to tackle the robot mobility problem in complex human environment. Well known examples of biped robot includes ASIMO [36] from Honda Inc., HRP-3 from Kawada Technologies as well as Atlas from Boston Dynamics. These robots are highly complex and requires sophisticated control algorithm to keep the robot balance. Simpler hexapod robots such as RHex [1][2] is developed to enable the robot to traverse various uneven terrains.

1.1.2 Jumping robots

In nature animals tend to jump over uneven terrains. Compare to static approach, jumping utilizes the dynamic natural of the body to increase efficiency and reduce energy consumption. Many bio-inspired jumping robots have been developed.7g [10], Grillo [11], MSU robot [12] and mowgli [13] are some examples of jumping robot. The linkage leg system and springs mechanism enables the robot to jump over large obstacles. The main drawbacks on these robots are the lack of efficient horizontal movement capabilities and are incapable of traversing large area due to limited mobility. Accuracy of movement is also compromised due to the discrete jumping nature of these robots.

Kikuchi and colleagues [17] introduced a wheeled robot that is able to hop up and down stairs. The robot consists of a wheel base and a spring-loaded movable upper body mass. The robot uses the upper body mass to generate lifting force for jumping and soft landing. The limitation of the robot is that the attitude of the robot during jumping is not actively control. In other words, safe landing of the robot is not guaranteed under the influence of external disturbances. iHop [18][19] is another example of jumping robots. It is uses the momentum of two weight wheels and a lockable hopping mechanism to achieve hopping and balancing. iHop exhibits hoping capability but it is not shown that the robot is capable of climbing up or down step terrains.

1.1.3 Crawler robot

Crawler type robot [7] is a common design for search and rescue robot that needs to overcome uneven terrains. One advantage of the crawler robot is the stability and robustness in tackling uneven terrain. Many crawler robots have been deploy in search and rescue operation due to its reliability in negotiating unknown environment. However the main drawback of the crawler is the relatively slow movement of the robot.

1.2 Problem Statement

As impressive these robots are, they also have some drawbacks. Some of these robots are both too complex and thus expensive, or they use tracks, which are not appropriate for indoor environments. These robots usually exploit the static stability of their support polygon and are passively balanced. This leads to coarse and slow dynamic responses of the robots when traversing stepped terrains and limits their performance. Also any external perturbation which forces the robot out of its basin of stability might lead to catastrophic failure. To ensure a rapid and stable transition through stepped terrain, a more robust control approach which takes into account the dynamics of the system is desired.

Dynamically stable robots offer better agility and are more robust to external disturbances. Such advantages can be used to achieve rapid, stable transition through stepped terrain. In our research, we focus specifically on using a two-wheeled robot to travel continuously through stepped terrain without losing balance. The problem of maintaining balance with a falling two-wheeled robot is highly non-linear. The nature of the balance problem changes as the robot is in different phases of motion. A free falling robot is a different control problem than a robot climbing a step or traversing flat ground. Two-wheeled robots have been a popular research platform due to their simple design yet complex dynamic behavior. Most of the research literature available focuses on continuous ground balance. The problem of balancing a two-wheeled robot through discontinuous terrain has received relatively little attention. Related research includes a reconfigurable hopping rover as proposed in Schmidt-Wetekam et al. [18][19]. The hopping rover resembles a 3 dimensional reaction wheel pendulum with a set of orthogonally arranged drive wheels. The drive wheels are used to provide torque for attitude correction to re-orient the vehicle during flight and ground balance. The hopping action is provided by an extendable leg. The hopping robot exhibited good dynamic stability on a flat surface, but performance on a stepped surface was not evaluated.

1.3 Goal of this Thesis

1.3.1 Aims

The goal of this thesis is to develop a robot that is able to traverse stepped terrain is an efficient and reliable manner. We have chosen the two wheeled robot as our research platform due to its simplicity in design while still exhibiting complex dynamic behavior. Our robot fulfills the following aims:

1) to develop a step traversing robot that is able to negotiate non-continuous ground terrains, i.e. stairs and step in a dynamic manner.
2) to develop a novel method to actively control the attitude of a robot in midair.
3) to verify the proposed method of control enables a conventional two wheeled robot to traverse stepped terrain and extend the traversable terrain.

1.3.2 Innovation and contribution of this research

This research demonstrates a two wheeled robot that is able to traverse stepped terrains while maintaining balance. The proposed method introduces an effective way to control the attitude of a two wheeled robot in air, which is critical to a safe landing. Our proposed method uses the wheels of the robot to generate balancing torque and requires no additional actuators or mechanical changes to a conventional two wheeled robot.

1.4 Methodology

In this thesis, we proposed a novel approach enabling the two wheeled inverted pendulum to traverse stepped terrain. The robot consists of a two wheeled lower body platform and a movable upper body mass connected by springs. The robot achieves hopping action by utilizing the impact force produced by releasing pre-tensioned springs. Due to statically unstable property of the robot, the attitude of the robot has to be actively controlled during airborne to ensure stability upon landing. The tilt angle of the robot body is controlled by torque generated by rotating the drive wheels. The dynamic nature of the robot enables it to climb up and down step terrains, leap over gap obstacles in a swift and robust manner.

1.5 Thesis Outline

This thesis contains three primary parts: Part I introduces the theory and basics of a reaction wheel pendulum; Part II introduces the two wheeled inverted pendulum and explores the effect of the wheel as a reaction wheel for attitude control; and Part III introduces BBot, a prototype two wheeled inverted pendulum that is capable of traversing stepped terrain and hopping. The layout of the thesis is as follows:

I. Theory and basics of a reaction flywheel and a two wheeled inverted pendulum.
   - Chapter 2 introduces the theoretical analysis, modeling, simulation, control and balancing a reaction wheel pendulum and a two wheeled inverted pendulum.

II. To develop a step traversing two wheeled inverted pendulum.
   - Chapter 3 introduces the hardware and embedded electronics needed to build an inverted pendulum robot.
   - Chapter 4 explores the possibility of using the wheels as a reaction wheel to actively control the attitude of the robot in air. The development of a prototype, BBot-1 is discussed in detail. Experiment results of the robot in negotiating steps with various heights and length is discussed.

III. To developed a hopping two wheeled robot capable of traversing stepped terrains.
   - Chapter 5 introduces BBot-2, a hopping two wheeled robot with active airborne control. This chapter discussed the mathematical analysis of the robot, design and implementation, as well as the balance control of the platform.

Finally, Chapter 6 concludes the effectiveness of BBot in negotiating stepped terrain compares to existing robots, and discuss about the future works to enable BBot to be deployed into real world situations.

Chapter 2
Reaction wheel pendulum and the inverted wheeled pendulum

2.1 Introduction

The inverted pendulum system is a popular problem often used as a simplified model to study complex problems. The study of flight dynamics of a rocket, balancing of biped humanoid robot is often simplified using the inverted pendulum model to understand the complex behavior of the systems. The inverted pendulum consists of a pendulum attached to a pivot point below its center of mass. The pivot point is mounted on a movable platform. The inverted pendulum is statically unstable and requires active control to keep it upright. The control of the inverted pendulum system can be categorized into two main groups:

1. where the actuator is placed on the pendulum (e.g. reaction wheeled pendulum, model of biped humanoid)
2. where the actuator is placed at the pivot point (e.g. wheeled pendulum, pendulum on a cart, etc...)

In this chapter, we will describe the modeling of wheeled pendulum and reaction wheeled pendulum and detailed analysis of the system dynamics. This chapter will serve as a foundation for the analysis of BBot in the subsequent chapters.

2.2 Reaction wheel pendulum

The reaction wheel pendulum is first introduced by Spong et al. [20][21]. It is one of the simplest inverted pendulum in terms of construction and analysis. The reaction wheel pendulum consists of a rotating flywheel mounted on the pendulum driven by an actuator. The lower end of the pendulum is free to rotate at its pivot. The pendulum achieves balance using the torque generated by accelerating the flywheel. However simple it is, the system exhibits interesting non-linear dynamics which provides a lot of insights into how to design a controller to balance such a system. In this section, we will provide an overview of the mathematical analysis and simulation of a reaction wheel pendulum.

2.2.1 Mathematical model

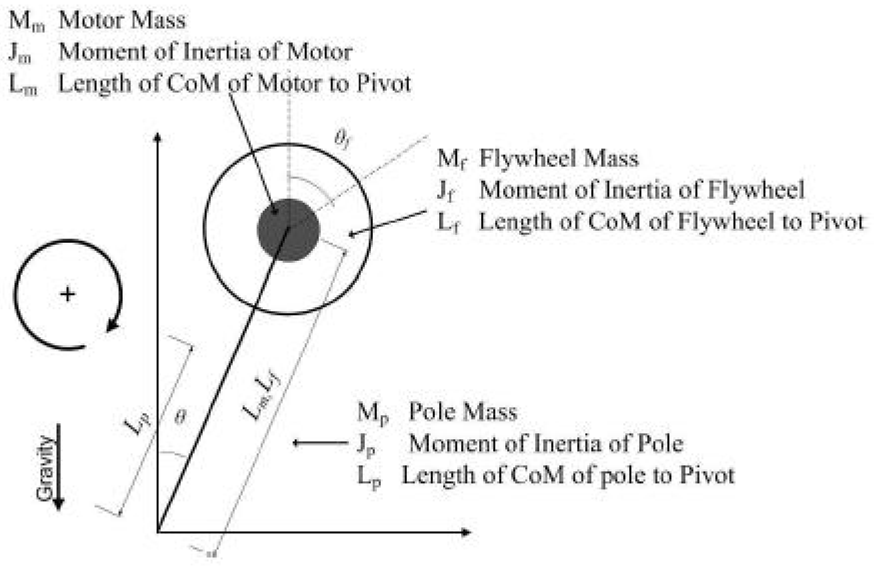

M_m Motor Mass
J_m Moment of Inertia of Motor
L_m Length of CoM of Motor to Pivot
θ_f
M_f Flywheel Mass
J_f Moment of Inertia of Flywheel
L_f Length of CoM of Flywheel to Pivot
+
L_p
θ
$L_m L_f$
M_p Pole Mass
J_p Moment of Inertia of Pole
L_p Length of CoM of pole to Pivot
Gravity

Figure 2-1: 2D schematic model of a reaction wheel pendulum

The reaction wheel pendulum can be modeled as a disc mass attached to the end of a pole. The disc is free to rotate about the end of the pole and the system is free to rotate about the fulcrum attached to the ground. Figure 2-1 shows the 2D schematic diagram of a reaction wheel pendulum. We obtain the equations of motion of the system by deriving the Euler-Lagrange equation:

$$J\ddot{\theta} - MLg\sin\theta = -\tau$$
$$J_f\ddot{\theta}_f = \tau$$

where

$$J = J_p + M_pL_p^2 + J_m + M_mL_m^2 + J_f + M_fL_f^2$$
$$ML = M_pL_p + M_mL_m + M_fL_f$$

Linearizing the equations of motion around small angle value of θ, the equations of motions can be rewritten as:

$$\ddot{\theta} - \frac{MLg}{J}\theta = -\frac{\tau}{J}$$
$$\ddot{\theta}_f = \frac{\tau}{J_f}$$

Table 2-1: List of symbols

Symbols	Units	Description
M_p	kg	Mass of robot body
M_m	kg	Mass of motor
M_f	kg	Mass of flywheel
L_p	m	Length of center of mass of robot body to ground
L_m	m	Length of center of mass of motor to ground
L_f	m	Length of center of mass of flywheel to ground
J_p	kgm^2	Moment of inertia of robot body
J_m	kgm^2	Moment of inertia of motor
J_f	kgm^2	Moment of inertia of flywheel
θ	rad	Angle of tilt of robot body
$\dot{\theta}$	rad/s	Angular velocity of robot body
θ_f	rad	Angular position of flywheel
$\dot{\theta}_f$	rad/s	Angular velocity of flywheel
τ	Nm	Input torque
g	$kgms^{-2}$	Gravity

From equations of motion, we know that the angular acceleration of the flywheel is directly proportional to the applied torque from the actuator. Torque is generated from the acceleration of the flywheel. It is important to note that due to the fact that acceleration of the flywheel is finite, once the actuator is saturated (i.e. spinning at maximum velocity) no more torque can be generated. The generated torque will applied an opposing torque in the opposite direction to the pendulum.

2.2.2 Simulation

We have constructed a reaction wheel model in MATLAB to validate the equations of motion and investigate the behavior of the system. We use MATLAB ode45 to solve the differential equations derived in the previous section. The non-linear and linear system of the reaction wheel pendulum system is simulated and compared.

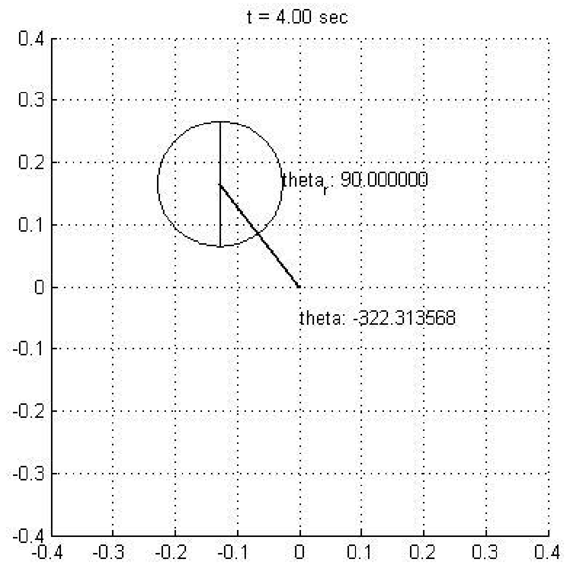


Figure 2-2: Animated drawing of the reaction wheeled pendulum in MATLAB

Figure 2-2 shows the snapshot of the animated drawing of the reaction wheel pendulum. The parameters of the simulation is summarized in the table below:

Table 2-2: reaction wheel simulation parameters

Parameter	Value

Mass of pole, M_p	0.2kg
Mass of wheel, M_f	0.125kg
Length of pole, L_p	0.2m
Radius of wheel, r	0.1m
Maximum applicable torque, τ_{max}	0.4Nm

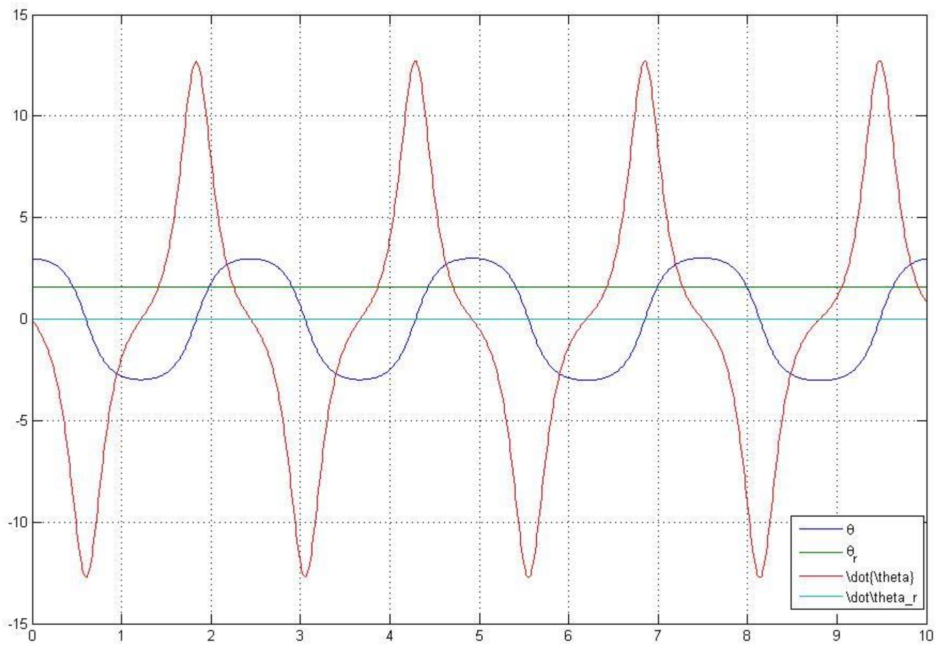


Figure 2-3: Large angle simulation of a non-linear reaction wheel pendulum

Figure 2-3 shows that the simulation results of the non-linear system over the course of 10s. Initial position of the pendulum is set to 10 degrees away from vertical top and allowed to swing freely after release. The system is assumed to be friction free. Due to lack of friction angular position and velocity of the reaction wheel remains constant. The angular position and velocity of the pole exhibits non linearity behavior. Maximum pole angular velocity occurs when the pole is at vertical downward position, when pole angular position is zero. On the contrary, minimum pole angular velocity (zero) occurs when the pendulum swings to the highest point.

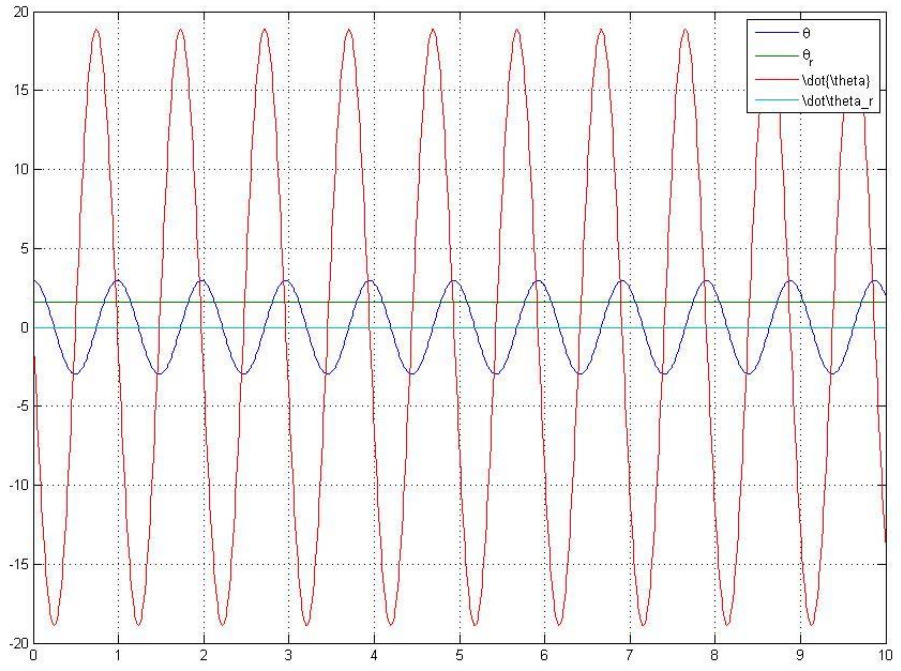
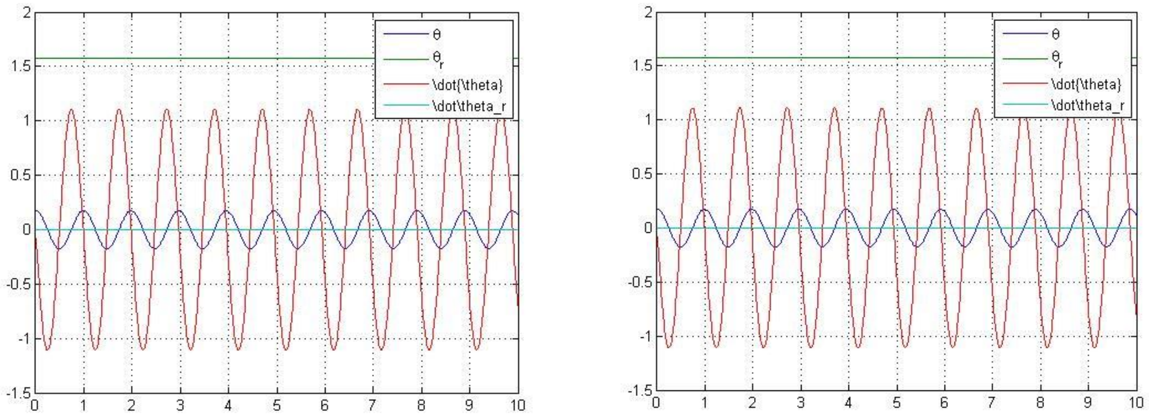


Figure 2-4: Large angle simulation of a linear reaction wheel pendulum

On the other hand, the simulation of the linear system exhibits very different response. Figure 2-4 shows the response the angular position and velocity of a linear system. Angular position and velocity display a sine wave response. This response does not comply with the actual response of a reaction wheel pendulum. The main reason of this inaccurate response is due to the large angle of swing of the simulation.



(a) (b)

Figure 2-5: (a) Small angle simulation of a non-linear reaction wheel pendulum. (b) Small angle simulation of a linear reaction wheel pendulum

Figure 2-5 shows the similar response for both non-linear and linear system for small angle simulation of the reaction wheel pendulum. It can be confirmed that a linear system is only valid under small angle assumption. The validity of this assumption will serve as a basis for the design of a linear controller in the following section.

2.3 Inverted wheeled pendulum

The inverted wheeled pendulum is an inverted pendulum attached to a wheeled platform. The motorized wheels generates balancing torque to keep the pendulum upright. A differential drive wheeled platform provides the ability for the system to spin on the spot, offering additional maneuverability. Personal transportation device such as the Segway^{TM} and Hoverboard are examples of application of the inverted wheeled pendulum model.

2.3.1 Mathematical model

In this section we will derive the equations of motion for the wheeled inverted pendulum.

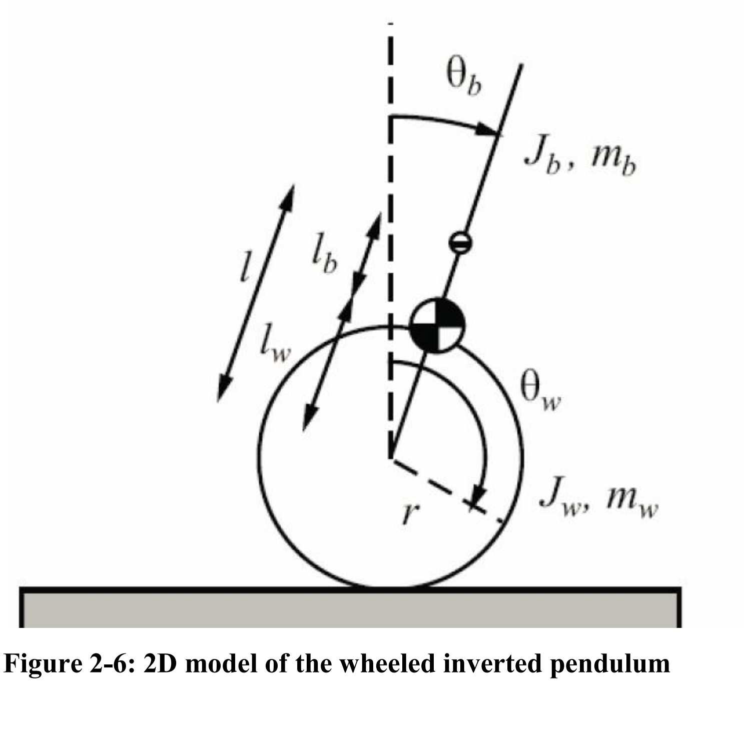


Figure 2-6: 2D model of the wheeled inverted pendulum

Table 2-3: List of symbols for inverted pendulum model

Symbols	Units	Description
m_b	kg	Mass of robot body
m_w	kg	Mass of wheel
J_b	kgm^2	Moment of inertia of robot body
J_w	kgm^2	Moment of inertia of wheels
r	m	Wheel radius
l	m	Length between center of mass of body and wheels
l_b	m	Length of body's center of mass to robot's center of mass
l_w	m	Length of wheel axis to robot's center of mass
G_r		Gear ratio
g	ms^{-2}	Gravity
θ_b	rad	Tilt angle of robot body
θ_w	rad	Rotational angle of wheels
θ_m	rad	Rotational angle of motor
$\dot{\theta}_f$	rad/s	Angular velocity of flywheel
τ_G	Nm	Motor torque in ground phase
τ_A	Nm	Motor torque in airborne phase

Figure 2-6 shows the 2D model of a wheeled inverted pendulum. The mathematical model of the system is derived using Lagrangian mechanics. The Lagrangian

$$L = T - U$$

is defined as the difference between the kinetic energy and potential energy of the system. The Euler Lagrangian equations of motion is given by:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \quad i = 1,2,\ldots,n$$

where $q = [\theta_w \quad \theta_b]^T$ are the generalized coordinates representing the angular position of the wheels and the inclination angle of the robot's body. The kinetic energy of the body, T_b, and the kinetic energy for the wheels, T_w, are expressed as:

$$T_b = \frac{1}{2}m_b r^2 \dot{\theta}_w^{\ 2} + m_b l r \dot{\theta}_w \dot{\theta}_b \cos\theta_b + \frac{1}{2}(J_b + m_b l^2)\dot{\theta}_b^{\ 2}$$

$$T_w = \frac{1}{2}m_w r^2 \dot{\theta}_w^{\,2} + \frac{1}{2}J_w \dot{\theta}_w^{\,2}$$

The potential energy for the robot is given by

$$U = m_b g l \cos \dot{\theta}_b$$

The Lagrangian of the system is hence:

$$L = \frac{1}{2}(m_b + m_w) r^2 \dot{\theta}_w^{\,2} + m_b l r \dot{\theta}_w \dot{\theta}_b \cos\theta_b + \frac{1}{2}(J_b + m_b l^2)\dot{\theta}_b^{\,2} + \frac{1}{2}J_w \dot{\theta}_w^{\,2} - m_b g l \cos \dot{\theta}_b$$

Evaluating the Euler Lagrangian equation for each of the coordinates gives the equation of motion as

$$(J_w + (m_b + m_w) r^2)\ddot{\theta}_w + m_b r l \cos\theta_b \ddot{\theta}_b - m_b r l \dot{\theta}_b^{\,2} \sin\theta_b = -\tau_G$$
$$(J_b + m_b l^2)\ddot{\theta}_b + m_b r l \cos\theta_b \, \ddot{\theta}_w - m_b g l \sin\theta_b = \tau_G$$

Rearranging the equation in to matrix form, we get:

$$M(q)\ddot{q} + C(q,\dot{q}) + G(q) = \begin{bmatrix} -\tau_G \\ \tau_G \end{bmatrix}$$

Where inertia matrix **M(q)** is

$$M(q) = \begin{bmatrix} J_w + (m_b + m_w) r^2 & m_b r l \cos\theta_b \\ m_b r l \cos\theta_b & J_b + m_b l^2 \end{bmatrix}$$

The vector coriolis/centrifugal forces is

$$C(q,\dot{q}) = \begin{bmatrix} -m_b r l \dot{\theta}_b^{\,2} \sin\theta_b \\ 0 \end{bmatrix}$$

and the vector of gravitational forces is

$$G(q) = \begin{bmatrix} 0 \\ -m_b g l \sin\theta_b \end{bmatrix}$$

The derived equations of motion represents the behavior of the wheeled inverted pendulum system. Detailed analysis of the system will be deferred to the next chapter where the actual parameters of the system is defined.

2.4 Conclusion

This chapter provides the detailed derivation of the equations of motion of both the reaction wheel pendulum and the wheeled inverted pendulum. The derived equations will served as a basis for further analysis of the actual robot in the subsequent chapters. In the next chapter, we will look at some hardware components required to construct the robot.

Chapter 3
Hardware and embedded electronics

3.1 Introduction

Simulation of a system provides valuable insights into the behavior and how a system reacts to a certain excitation or external disturbances. Simulation provides a way to understand and test a system before constructing an actual system. Hardware prototyping involves construction of an actual system. Various kinds of considerations such as choice of sensor, processing unit, power, mechanical designs need to be considered to make the whole system works. In this section, we provide an overview of the hardware construction of BBot.

3.2 Sensors

In simulations, system states are readily available. However in the actual system, not all system states and data can be obtained. Some system states might need to be estimated or derived implicitly from physical sensor data. The choice of sensors and sensing algorithm greatly affects the quality and usefulness of data acquired. The following table summarizes the list of sensors used to measure corresponding states of the robot.

Table 3-1: Sensor used for state measurement

Sensor	Measure state
Accelerometer, gyro	Body angular velocity, body angle
Ultrasonic sensor	Robot height
Encoder	Wheel angular velocity, wheel angle

3.2.1 Accelerometer

An accelerometer is a device which measures acceleration forces. Accelerations applied to the sensor includes static acceleration such as gravity, or dynamic forces due to object acceleration or vibration. A tri-axis accelerometer consists of three orthogonal sensing axes to measure acceleration in all x, y, z direction. Besides measurement acceleration, another primary usage of accelerometer is to detect inclination. This is due to the fact that there is constant global gravitation acceleration on the surface of the Earth. When the accelerometer is static, the gravitation force detected by the sensor provides a directional vector indicating the direction of gravity.

From this data, we can calculate the angle of tilt by taking the cosine angle between the accelerometer horizontal axis (e.g. x axis) and the horizon. Note that this method of angle calculation is only reliable when the accelerometer is not under any influence from acceleration. Another drawback in using accelerometer to calculate inclination angle is due to the noisy nature of accelerometer data. In other words, angle obtain using only accelerometer will not be reliable.

3.2.2 Gyroscope

A gyroscope is a device that measures rotational velocity. A gyroscope output is proportion to the angular velocity. A triple axis MEMS gyroscope can measure angular velocity around x, y and z axes. By integrating the output of a gyroscope, we can calculate the angular position with respect to an initial value. Due to the fact that a gyroscope can only detect a local rotation, without any external reference, it is impossible to use only gyro to detect the angle of tilt.

3.2.3 Inertial measurement unit (IMU)

Given the pros and cons of both accelerometer and gyro sensors, it is possible to combine the two sensors to detect inclination. This kind of sensor used to detect orientation is refered as an inertial measurement unit (IMU). The IMU includes an accelerometer and a gyroscope. Theoretically, rotational angle can be calculated from direct integrations of gyroscope data. However due to bias noise of gyroscope, angle calculated from integration will drift overtime. This leads to inaccurate angle estimation. We can use the accelerometer as a reference to improve the angle estimation and compensate the gyro angle drift. Various fusion methods such as Kalman filter [33], complementary filter [32] and direct cosine matrix (DCM) filter [34] have been proposed to estimated the orientation of the sensor. Each approach poses different accuracy and calculation complexity. From our experiments, we found that Kalman filter sensor fusion provides the best accuracy and robustness compare to other methods. This is especially obvious when the sensor undergoes large influence of linear acceleration, when the robot is moving.

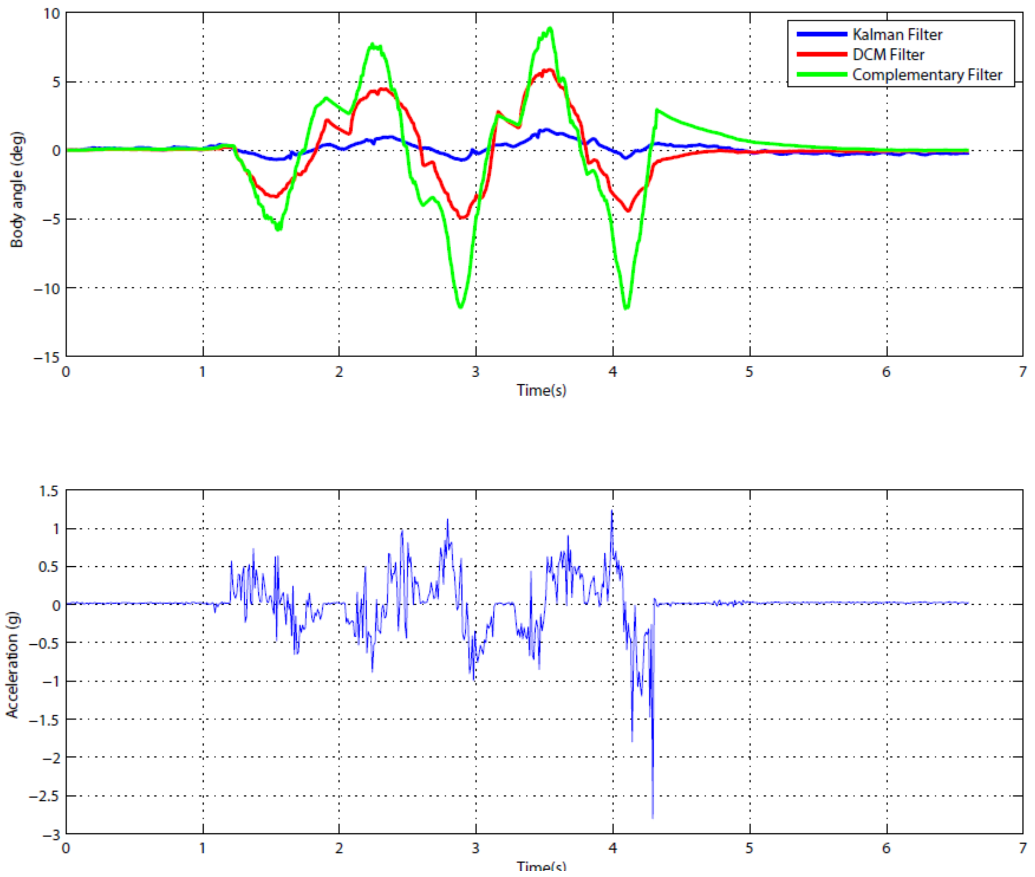


Figure 3-1: Comparison of angle estimation between Kalman filter, DCM filter and complementary filter under the influence of linear acceleration. Angle calculated from Kalman filter shows little influence from linear acceleration.

Figure 3-1 shows the comparison of performance between Kalman, DCM, and complementary filters in angle estimation. The IMU sensor is accelerated back and forth horizontally. The lower graph shows the corresponding raw accelerometer data. Ideally the estimated angle of the IMU should remain constant zero. From the figure, we conclude that the complementary filter performs poorly especially during acceleration. This is due to the fact that the complementary filter estimates the angle based on weighted angles from accelerometer and gyro sensor. This fixed ratio does not take into account the influence of varying accelerations. The DCM filter fluctuated less that the complementary filter, but the influence from linear acceleration was still obvious. By contrast, the Kalman filter showed little fluctuation in angle estimation [30].

3.2.4 Ultrasonic distant sensor

The ultrasonic distance sensor is a sensor that uses ultra sound to measure distance. The sensor consist of a signal emitter and a receiver. The emitter sends out a pulse signal and the receiver detects any reflected pulse. By comparing the delay between output and input signal, the sensor is able to deduct the distant of an object in front of the sensor. The ultrasonic sensor can be used to detect the height of the robot above ground when the robot is in air. The ultrasonic sensor is chosen over other distant sensor (e.g. infrared sensor) due the larger measurement distant and accuracy.

3.2.5 Encoder

Encoder is a rotary disc with tiny strips engraved. By shining a light beam through the strips, we can obtain pulses of light when the disc rotates. Counting the pulses enables us to track the relative rotation of an attached object with high precision. A microcontroller can be used for pulse counting. The encoder is attached to the motor to enable use to calculate the position and rotational speed of a motor.

3.3 Microcontroller

3.3.1 Mbed prototype platform

One of the requirement of the robot is to be fully self contained. In other words, the robot needs to be able to process all sensor data without an external connected PC. The reason for this requirement is due to the fact that any tangling wire or connection to an external pc for data acquisition and control will affect the results of our jumping robot. There are various choices of data processing unit available in the market. Powerful single board computer such as Raspberry Pi, Beagle Bone board provides attractive implementation choices. To keep the form factor, weight and power requirement to minimum, we decided to use an embedded microcontroller to perform all calculations on board of the robot. Taking in to account factors such as size and power consumption, we have chosen the Mbed microcontroller as our development platform. Mbed platform is a microcontroller based on the design of a 32-bit ARM Cortex-M3 architecture. The microcontroller, running at a frequency of 96MHz, has sufficient processing power to perform all the calculations. The microcontroller also has sufficient peripherals to interface with all the sensors.

3.4 Mechanical design

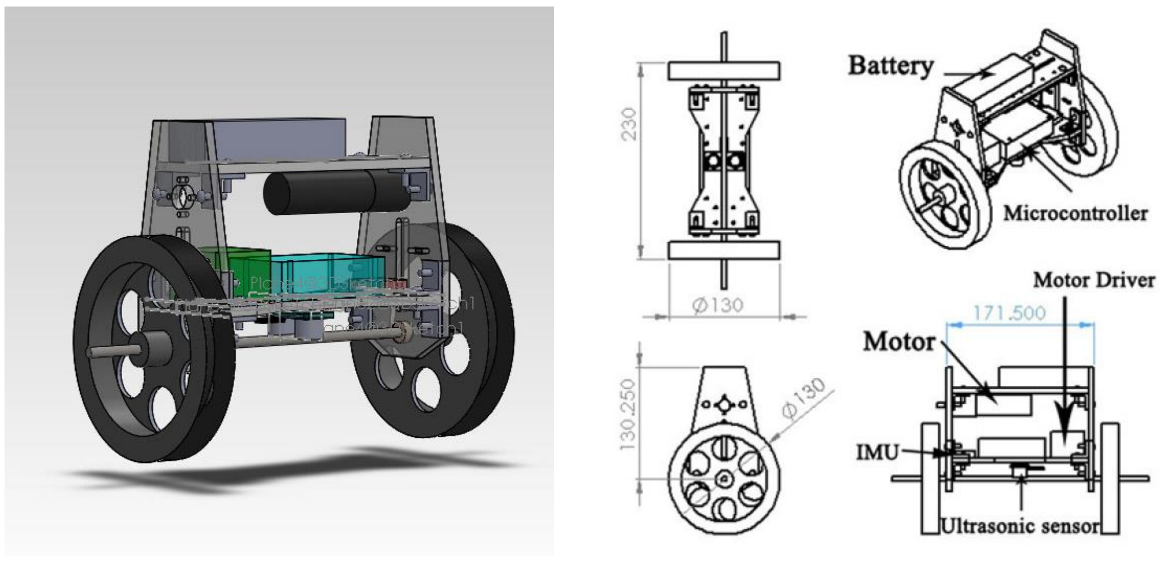


Figure 3-2: Mechanical design of the two wheeled pendulum

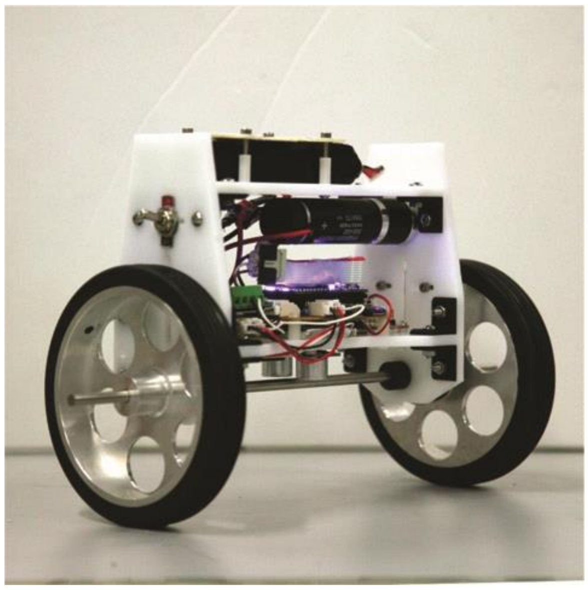


Figure 3-3: First prototype of BBot-1

We have listed the following requirements which the prototype robot needs to meet:

1. fully self contain (battery and control circuit included)
2. Sturdy enough to withstand repeated impact forces from falling from high steps

Figure 3-2 shows the CAD design of the prototype robot. The first version of BBot [27][28] prototype is shown in Figure 3-3. The main chassis is made of sturdy Duracon material. Onboard electronics board consist of power circuit, main microcontroller, an inertia measurement unit (IMU) and motor driver. An ultrasonic distance sensor attached to the bottom of the robot is used to detect the relative height of the robot from ground during experiment. The robot is equipped with a bluetooth interface to communicate with an external pc for data logging and remote control. Figure 3-4 shows the overview of the system architecture of the robot. Table 3-2 summarizes the physical parameters of the BBot-1.

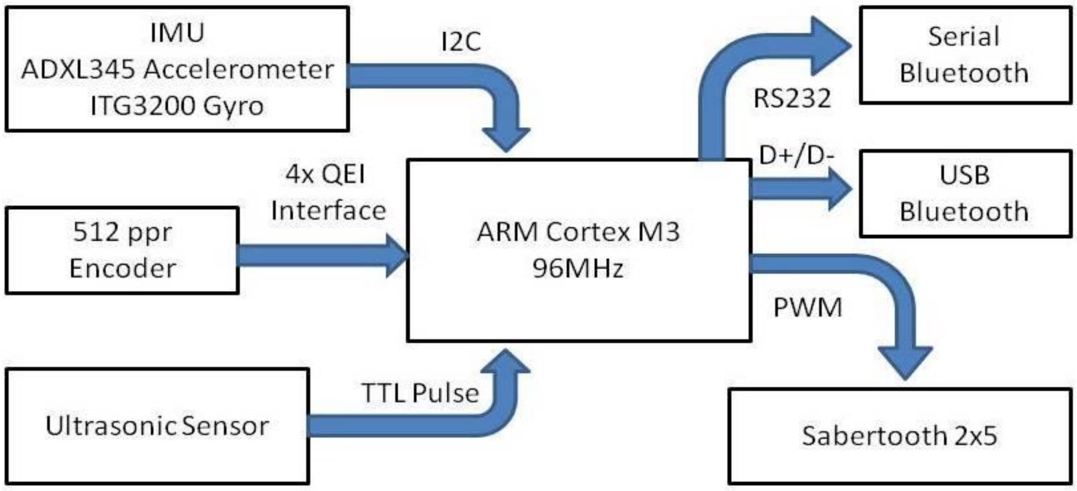


Figure 3-4: System architecture of BBot-1

Table 3-2: Physical parameters of BBot-1

Overall size	(W)130x(L)230x(H)190mm
Total mass	1.53kg
Mass of body	0.995kg
Mass of wheels	0.535kg
Moment of body	4.913×10^{-3} kgm^2
Moment of wheels	1.354×10^{-3} kgm^2
Motor	Maxon RE-25 12V 10W
Gear head	3.8:1 Maxon 26mm planetary gear head
Timing belt	4mm 5:3 speed ratio
Battery	11.1V 2200mAh lithium polymer
Battery life	Approx. 2 hours
Sensors	ADXL345 Accelerometer ITG3200 gyro Parallax ping ultrasonic sensor Maxon MR 512 ppr encoder

3.4 Conclusion

In this chapter we provide an overview of the hardware and electronics needed to construct a two wheeled pendulum platform. Choices of sensors, controller unit and hardware design are important to ensure the robot behaves close to expected behavior. Unlike simulation, a physical construction of the robot introduces unknown parameters which are hard to model. Physical parameters of the robot such as moment of inertia, center of mass is predicted using computer aided design (CAD) software. The parameters are verified experimentally. Besides hardware consideration, special care is need when designing the embedded processing software.

Chapter 4
Stair traversing two wheeled robot

4.1 Introduction

4.1.1 Background

Existing two wheeled robot platforms operates under the assumption that the robot is constantly in contact with the ground. This assumption limits the traversable terrain to continuous ground. The problem of enabling the two wheeled robot to traverse non continuous ground terrain poses an interesting challenge. This chapter explores the possibility of a two wheeled robot to traverse down a non-continuous ground step. When travelling down a stepped terrain, the robot undergoes state transition from on-ground to airborne and back to on-ground phase. This transition may repeat for continuous stepped terrain. Conventional control scheme will fail to keep the robot in balance during state phase transition. We proposed a novel attitude control scheme to control the attitude of the robot during airborne. During free fall, a two wheel robot behaves similar a reaction wheel pendulum, with pivot point at the center of mass [27]. Hence it is possible to use the momentum of the wheels to generate correction torque to alter the orientation of the robot and ensure a safe landing.

4.2 Dynamic Model

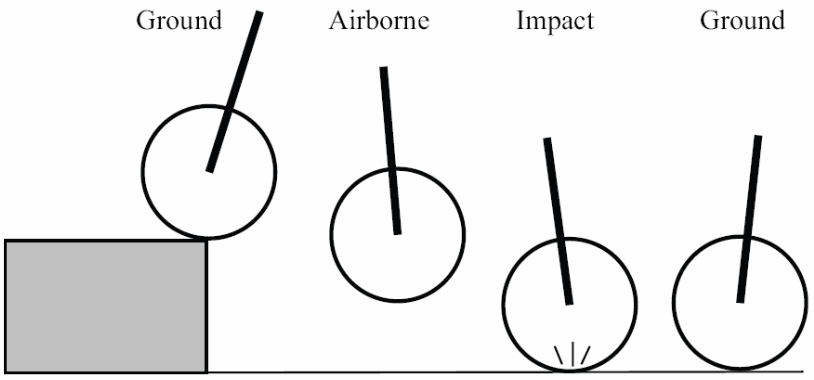


Figure 4-1: Unique phases when traversing through a step

During the process of traversing a stepped terrain, the robot undergoes phase transition as shown in Figure 4-1. In order to simplify the analysis of the problem, we separate the motion into three unique phases:

1. Ground
2. Airborne
3. Impact phase

We introduce additional assumptions to further simplify the assumption process:

1. Both of the drive wheels leaves the edge of step at the same time (i.e. moment around roll direction is zero during flight)
2. Fall height is within average stair height
3. Air resistance during freefall is neglected
4. The friction between wheels and ground contact is a large

Assumption 1 allows us to simplify the problem into a two dimensional pendulum problem for easier analysis. Under assumption 4, the robot lands without slipping. We use the mathematical model derived in chapter 2 to analyze the behavior of the robot when travelling on ground. However during airborne, a new set of equations has to be derived to accurately represents the behavior of the robot. Figure 4-2 shows the additional 2D model of the robot during airborne. During airborne the robot behaves like a freefall reaction wheel. We can use the reaction wheel equations of motion derived in chapter 2 to understand how the robot behaves when falling.

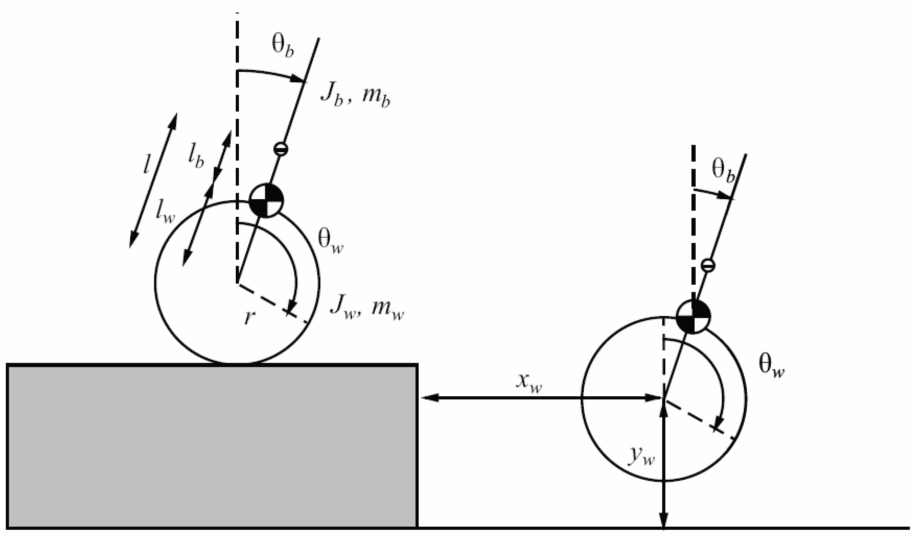


Figure 4-2: The 2D model of two wheeled robot (a) on ground and (b) in air

4.2.1 Airborne model

The airborne model is derived based on the reaction wheel model presented in chapter 2. During freefall, any external forces applied to the system is treated as disturbance forces. Under such circumstances, the model can be simplified into a 2D reaction wheel pendulum with pivot at its centre of mass. Eliminating the potential energy terms and the displacement terms, the simplified model becomes

$$\begin{bmatrix} J_w & 0 \\ 0 & J_b + m_b {l_b}^2 + m_w {l_w}^2 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_w \\ \ddot{\theta}_b \end{bmatrix} = \begin{bmatrix} \tau_A \\ -\tau_A \end{bmatrix}$$

Solving the equations fro the relationship between θ_w and θ_b yields

$$\theta_b = -\frac{J_w}{J_b + m_b l^2}\theta_w$$

From the above equation we can see that during free fall the change of angle of inclination is directly proportional to the angular displacement of the wheels due to the conservation of angular momentum.

4.2.2 Impact

Upon landing, the robot experience phase transition from airborne to ground. Depending on the height of the fall, the robot experience different impact forces at the moment the wheels make contact with the ground. The collision is not perfectly inelastic. In actual experiment, right after landing, the robot experience rebound before settling down. The design of BBot-1 does not include any active suspension system and hence rebounding problem becomes more obvious as height of fall increases. The rebound cause by the impact depends greatly on the coefficient of restitution for different ground material. Due to difficulty of determining such parameter, we do not derive the impact model. Instead, under assumption 2, robot falls from average stair height, the rebound is small and can be neglected. In practice, we mitigate the problem of rebound and impact forces by treating them as disturbance to airborne controller. The next section will discuss the design of the controller in detail.

4.3 Control scheme

The motion of the robot traversing through a step involves different phase transitions. Instead of designing a universal controller to balance the robot throughout the phase transitions, it is simpler to design separate controllers for each phase and use an additional controller to switch between phases. The architecture of the feedback system is shown in Figure 4-3.

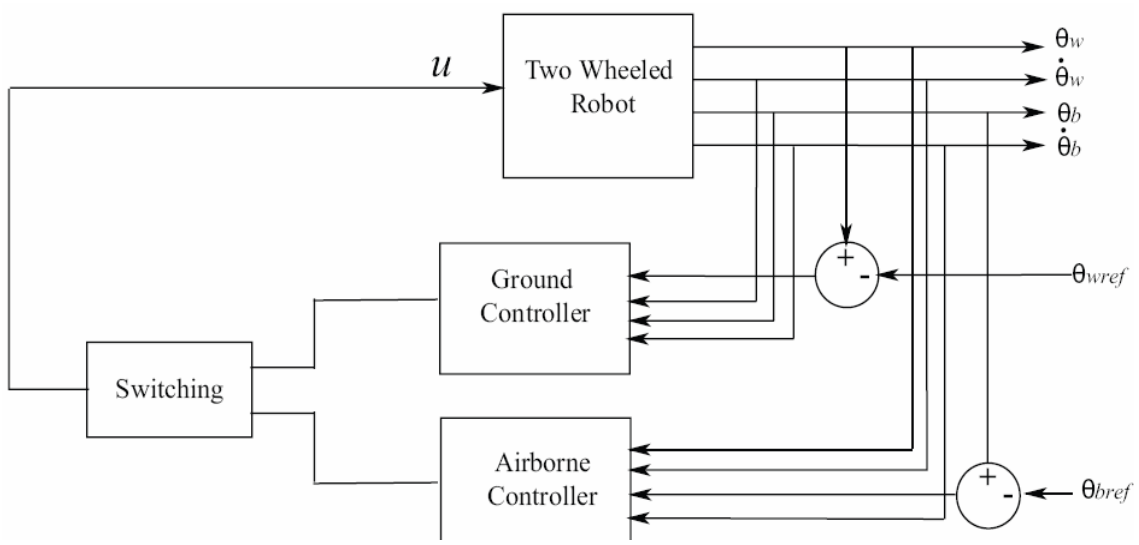


Figure 4-3: Structure of feedback controller

4.3.1 Ground balance control

The robot will remain balance if we keep the body inclination angle θ_b close to zero. To design a linear controller, the ground model equations of motion derived in chapter 2 is linearized about the unstable equilibrium. Arranging the equations into state-space representation yields

$$\dot{x}_G = A_G x_G + B_G u$$

where

$$x_G = \begin{bmatrix} \theta_w \\ \dot{\theta}_w \\ \theta_b \\ \dot{\theta}_b \end{bmatrix}, \quad A_G = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & a_{23} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & a_{43} & 0 \end{bmatrix}, \quad B_G = \begin{bmatrix} 0 \\ b_2 \\ 0 \\ b_4 \end{bmatrix}$$

$$a_{23} = \frac{-p_{12}}{p_{11}p_{22} - p_{12}p_{21}} m_b l g$$

$$a_{43} = \frac{p_{11}}{p_{11}p_{22} - p_{12}p_{21}} m_b l g$$

$$b_2 = \frac{-p_{11} - p_{22}}{p_{11}p_{22} - p_{12}p_{21}}$$

$$b_4 = \frac{p_{11} + p_{21}}{p_{11}p_{22} - p_{12}p_{21}}$$

$$p_{11} = J_w + (m_w + m_b) r^2$$

$$p_{12} = m_b r l$$

$$p_{21} = m_b r l$$

$$p_{22} = J_b + m_b l^2$$

The linear quadratic regulator (LQR) is used to design a linear full state feedback controller which minimizes the quadratic cost function

$$J(u) = \int_a^b x_G^T Q x_G + u^T R u \, dx$$

The values of matrix *Q* and *R* are chosen experimentally. The final feedback control input *u* is given by

$$u = \tau_G = -K_G x_G + k_{g1} \theta_{wref}$$

where $K_G = [k_{g1} \quad k_{g2} \quad k_{g3} \quad k_{g4}]$ is the feedback gain matrix. θ_{wref} is the reference value for controlling the rotational angle of the wheel. We can control the motion of the robot by change the reference value through a remotely connected pc. The system is simulated in MATLAB to examine the effectiveness of the controller.

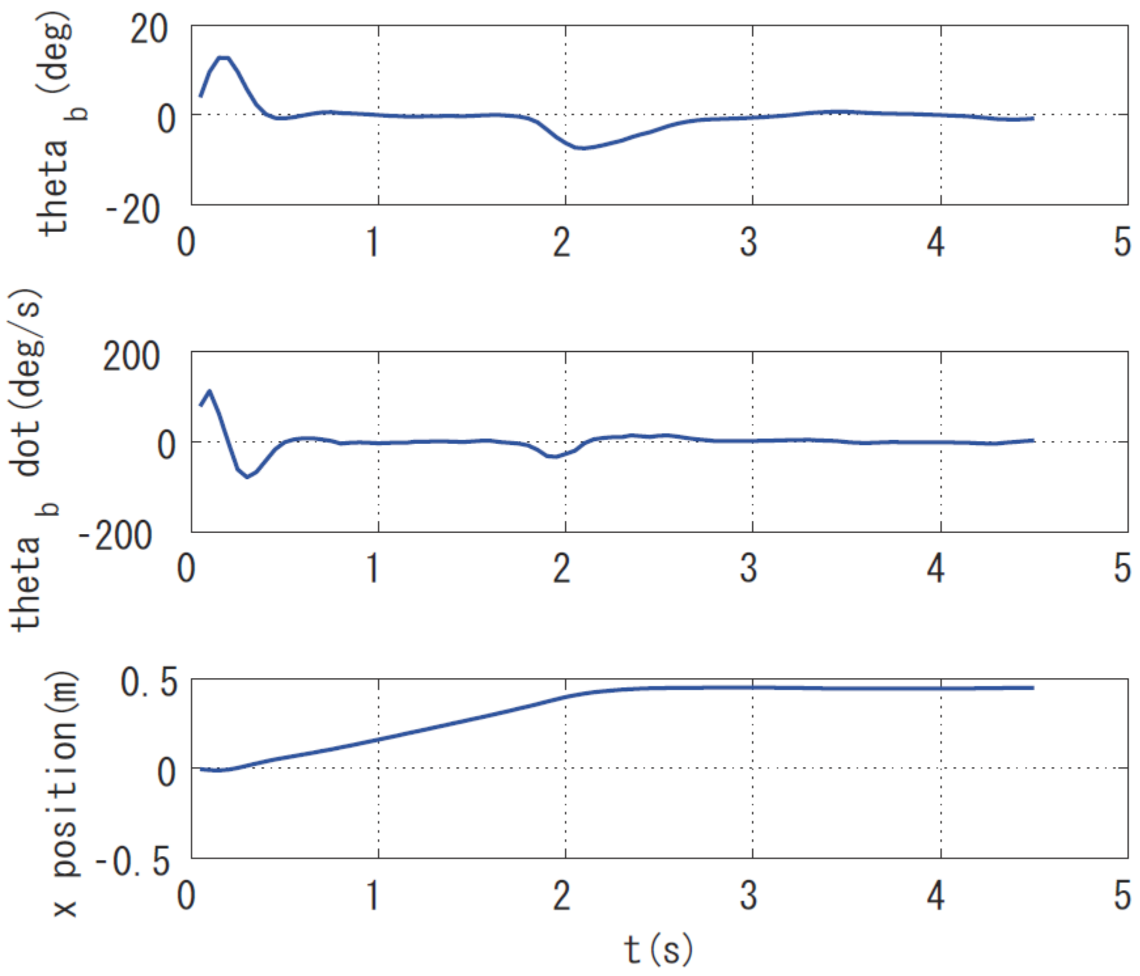


Figure 4-4: Simulation results of on ground feedback controller

Figure 4-4 shows the simulation results of position control of the controller. The robot starts from vertical position and is instructed to move to a reference point 0.4m forward. From the graph, the robot reaches to the reference point in roughly 2 seconds. From the body angle plot, the robot is able to keep the angle of tilt around equilibrium point.

4.3.2 Airborne control

The linear model derived in the previous section is used to design a linear controller. Controlling the attitude of the robot during freefall is relatively simple due to the direct proportional relationship of the body angle and wheel angle. The control law used to control the inclination angle is

$$u = \tau_A = k_{a1}(\theta_{bref} - \theta_b) - k_{a2}\dot{\theta}_b - k_{a3}\dot{\theta}_w$$

where k_{a1}, k_{a2} and k_{a3} are feedback gains and θ_{bref} is the reference body tilt angle during airborne. The angular position of the wheels θ_w is not included as we are not interested in controlling the position of wheel.

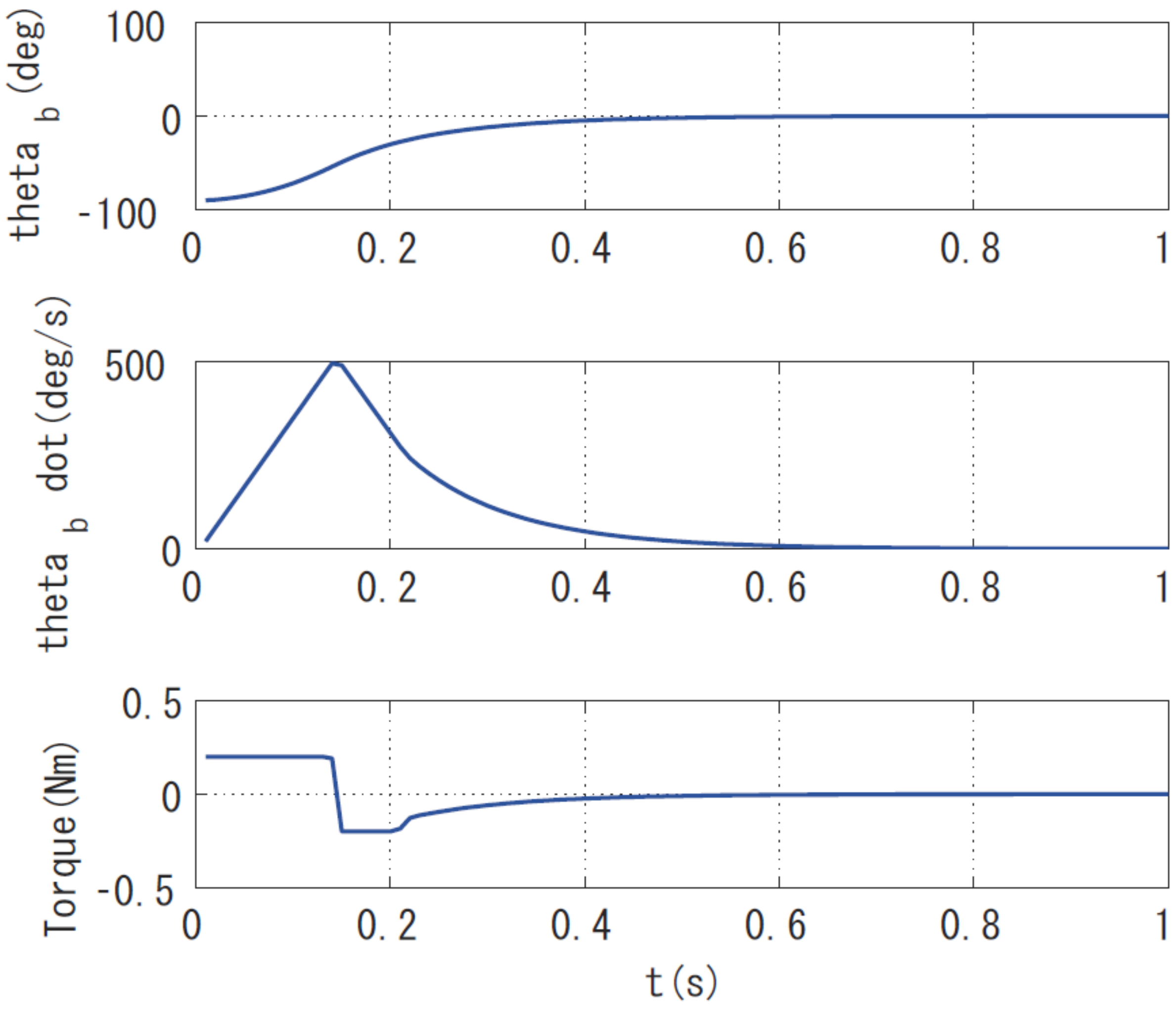


Figure 4-5: Simulation results of airborne feedback controller

Figure 4-5 shows the simulated feedback responses of the airborne feedback controller. The initial body tilt angle θ_b is set to –90 degrees. Reference angle θ_{bref} set to zero. The robot is able to recover to upright position in roughly 0.4s. In actual robot, changes in angle of tilt will be less that 90 degrees and hence response time will be much faster.

4.3.3 Switching controller

During phase transition the switching controller chooses the appropriate control scheme to keep the robot upright. The switching condition depends on the height of the robot above ground based on readings from the ultrasonic distance sensor. The switching controller is implemented as follow:

$$u = \begin{cases} \tau_G, & |h| \leq h_{threshold} \\ \tau_A, & |h| > h_{threshold} \end{cases}$$

where τ_G and τ_A are the control efforts calculated from the ground and airborne controller respectively, and $h_{threshold}$ is the height threshold value determined experimentally. Impact detection is conducted by analyzing the readings from accelerometer.

4.3.4 Periodic control of continuous step traversing

The previous sections discuss the control scheme traversing a single stepped terrain. In the case of traversing a flight of stairs, the motion can be considered as the continuous transition between ground and airborne phases. The length of steps in a flight of stairs is short. Consequently the duration of the robot in one phase is short. We found that deactivating position control when travelling down a flight of stairs improve the overall stability of the robot. This can be achieved by using another set of control gains, obtained by setting corresponding weight of the wheel angle, θ_w in the weight matrix Q to zero [30]. Once the robot reaches the bottom of the stairs, position control is reactivated.

4.4 Experiment and Discussions

The effectiveness of the proposed method is evaluated by driving the two wheeled robot down stepped terrains with different height and environment. The robot is remotely controlled by pc via bluetooth connection. Internal states of the robot such as body inclination angle θ_b, body angular velocity $\dot{\theta}_b$, wheel position θ_w and velocity $\dot{\theta}_w$ and sensor reading of ultrasonic distant sensor is logged. Motion of the robot is captured using a high speed camera at 240 fps for visual analysis. Experiment on specific terrain is repeated multiple times to confirm the effectiveness of our approach.

4.4.1 Single step (19cm)

In the first experiment, the step of 19cm high is used. This is the average height of stair step. The robot is placed on top of the step and controlled to move forward to fall off the step. Reference distance is set to 1m.

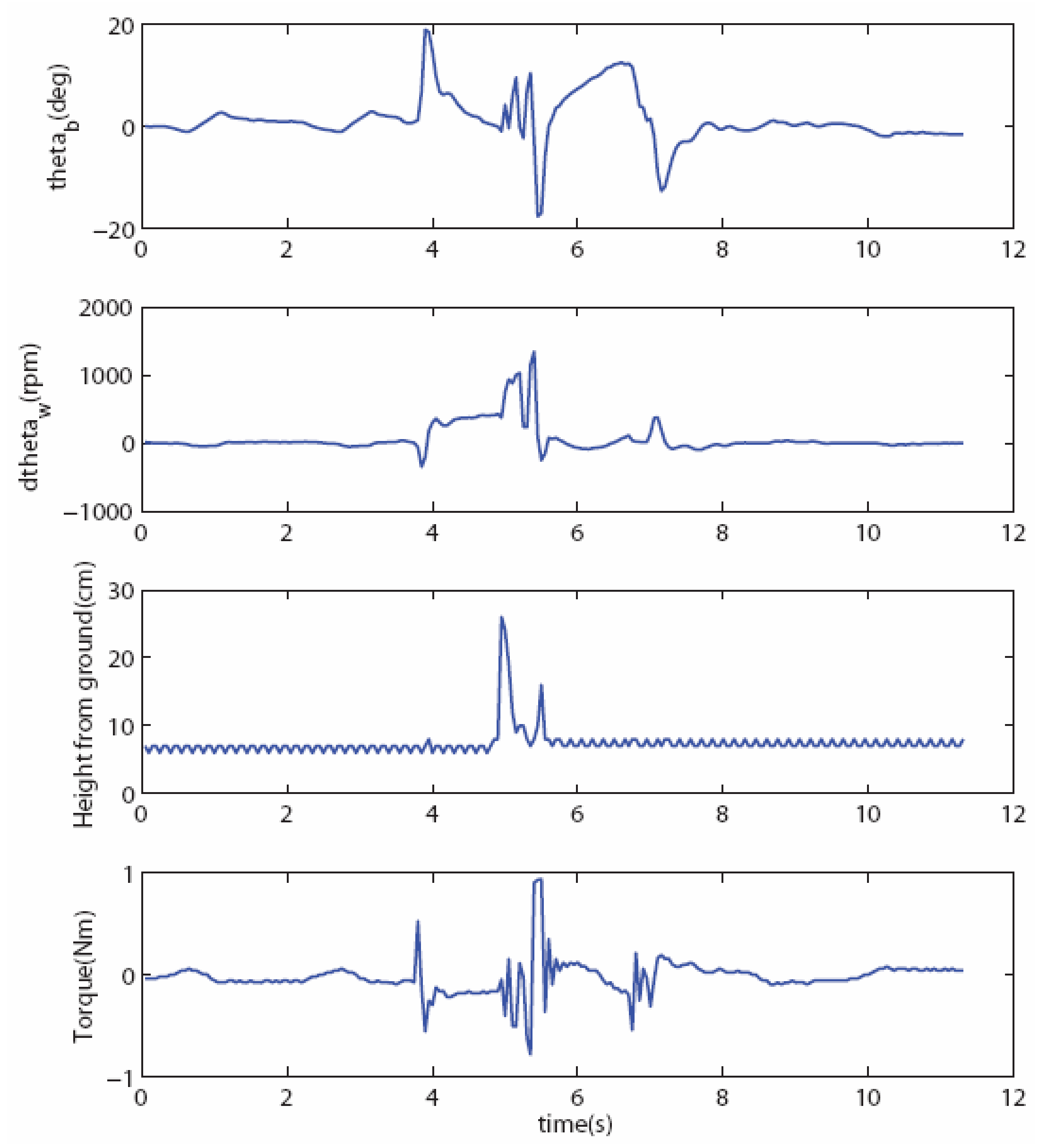


Figure 4-6: Body inclination angle θ_b, wheel velocity $\dot{\theta}_w$, height from ground and motor torque against time plot when traversing through a stepped terrain of height 19 cm

Figure 4-6 shows the plot of the body inclination angle θ_b, wheel angular velocity θ_w, height from ground and motor torque against time. For $t < 3.9$s the robot is balancing while maintaining body angle within ±2°. At time $t = 3.9$s the robot moves forward. Body angle increases. From the wheel angular velocity plot (at t = 3.9 s) we can see that the motor spins backward to tilt the robot forward, before spinning forward to keep the robot balance. The reason of this behavior is due to the nonholonomic nature of the two wheeled robot.

Height-from-ground graph plots the sensor reading from the ultrasonic sensor, mounted on the bottom of the body. The fluctuation of the sensor reading when the robot is balancing still is due to the fact that the sensor has a resolution of 1 cm. We can dictate the phase of the robot by analysis the height-from-ground plot. The spike at t = 5s from height-from-ground vs. time plot indicates that the robot approaches the edge of the step. The reference body angle during airborne θ_{bref} is determined experimentally. The value of θ_{bref} was fixed at 5° throughout the experiments. The airborne phase lasted roughly 0.15 seconds. A second spike can be seen from the height-from-ground plot at t = 5.5 s. This spike is caused by rebound of the robot after landing. During this rebound phase, the switching controller switches back and forth between ground and airborne controller, which causes the large fluctuation of the body tilt angle. Once the robot settles down at t = 6 s, the robot kept moving forward until it reached to the reference position (at t = 7 s).

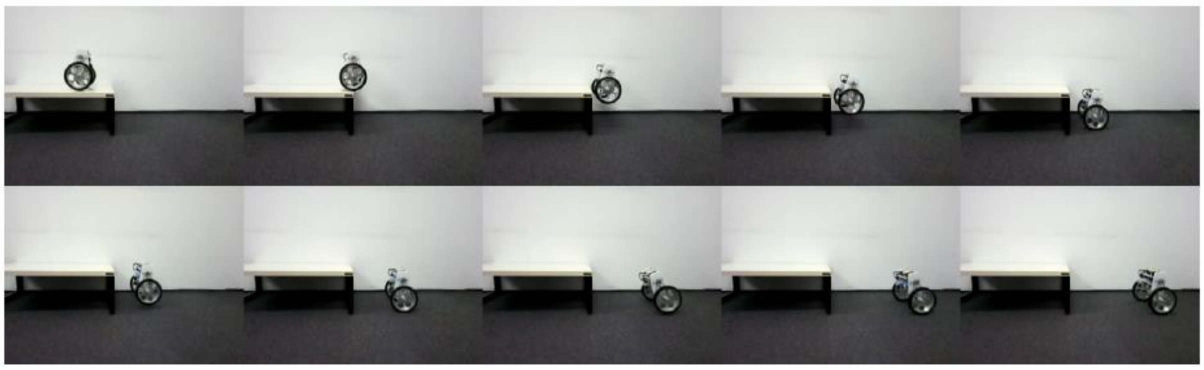


Figure 4-7: Snapshot of the motion of traversing through a single stepped terrain

Figure 4-7 shows the corresponding snapshots of the motion. Despite the presence of un-modeled non-linear disturbances (e.g. friction and impact forces) the proposed controller performs well and is able to keep the robot balance when traversing stepped terrain.

4.4.2 Single high step (70cm)

In this experiment, a step (table) of height 70cm is used. The purpose of this experiment is to investigate the limitation of our robot. Figure 4-8 shows the experiment data plots.

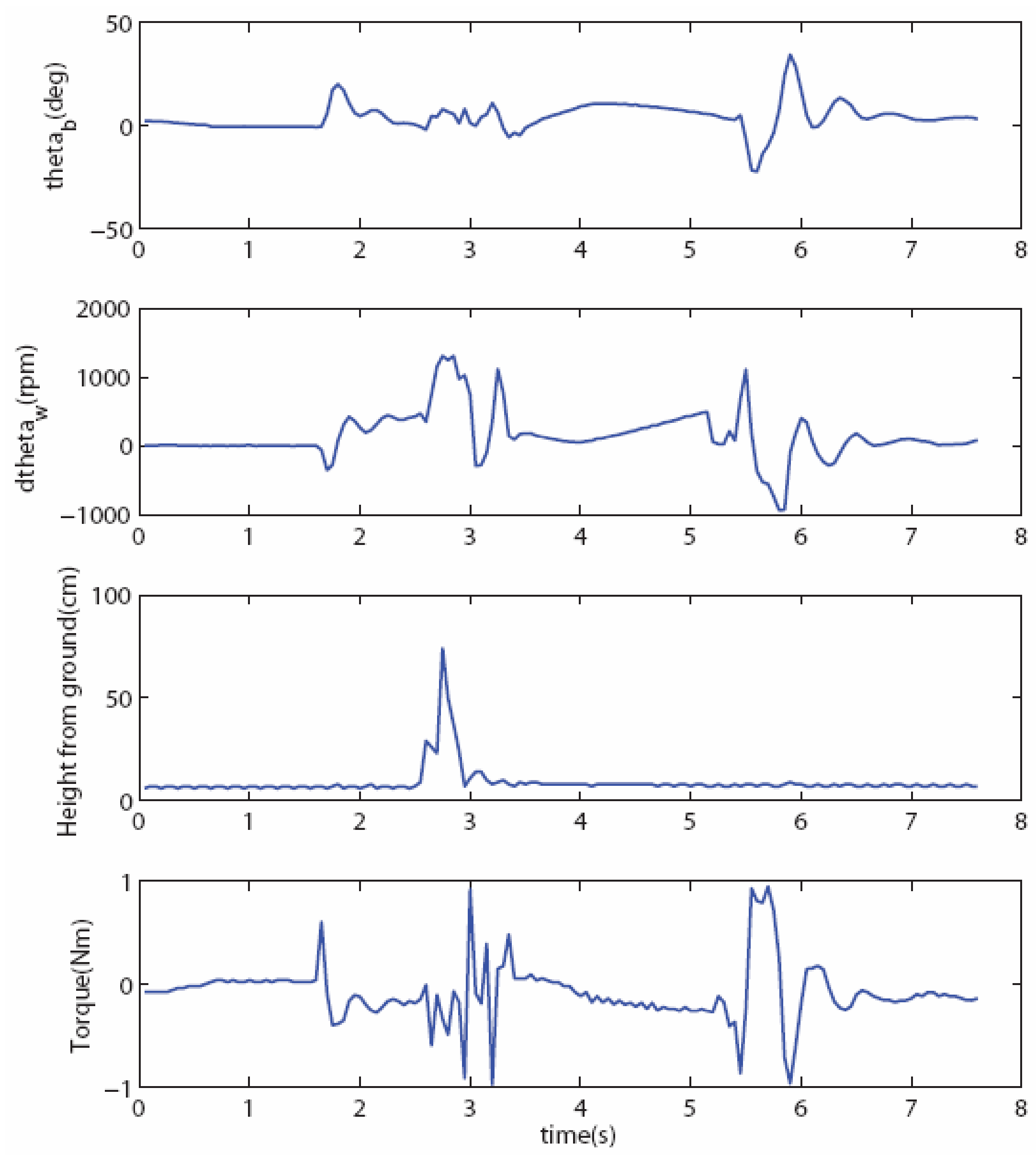


Figure 4-8: Tilt angle θ_b, wheel velocity $\dot{\theta}_w$ relative height from ground and motor torque against time plot of experiment of a fall from table of 70 cm high

Similarly the robot is controlled to move forward at time t = 1.6s. At time t = 2.5s the robot falls off the edge and impacts on ground at time t = 3s. During freefall the airborne controller tries to keep the body angle closed to the reference angle. Upon landing, the motor torque fluctuated as the controller tried to balance the robot. Large disturbance force caused by the impact increases the torque needed to keep the robot balanced. At time t = 5.5s the robot reaches the reference position.

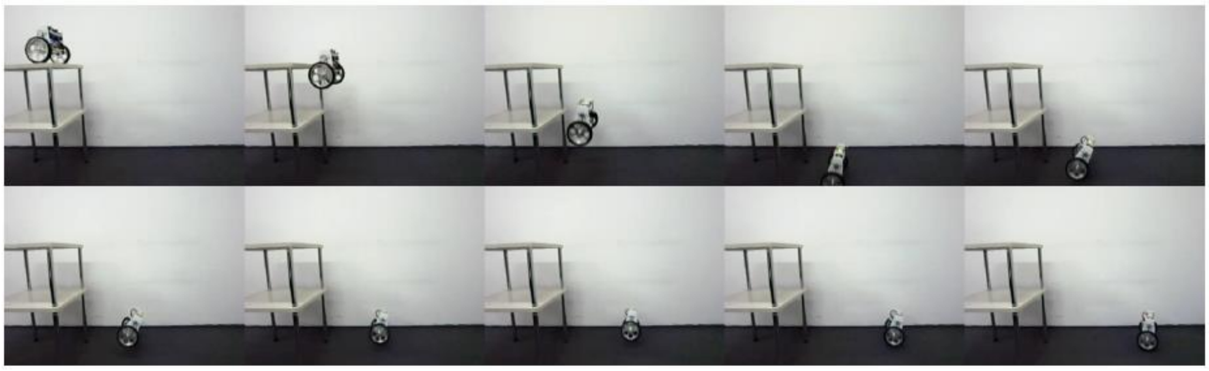


Figure 4-9: The consecutive snap shots of the experiment of falling off a table of 70 cm high

Figure 4-9 shows the corresponding snap shots of the experiment. Due to the height of the fall, the robot experienced significant rebound. During some of the experiments, we also observed uneven landing which causes sideways disturbance to the robot. Nevertheless, the controller is able to keep the robot balance under large disturbance.

4.4.3 Continuous steps and stairs

For the robot to be practical, it has to be able to traverse continuous steps. In this experiment, a double stepped terrain is used. Similarly the robot starts from the top of the steps and moved forward. Height of the steps is 19cm. The experiment results are shown in Figure 4-10.

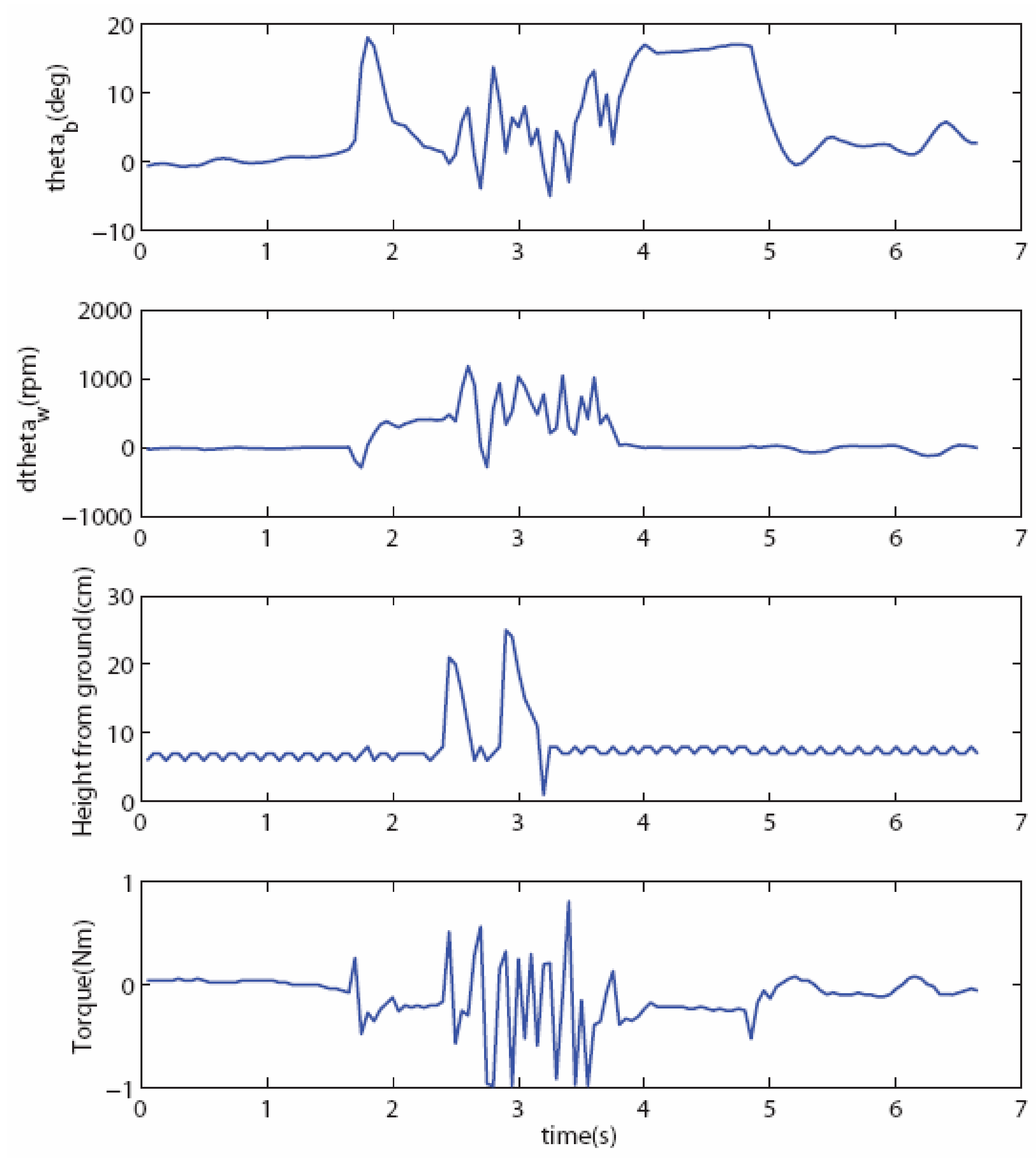


Figure 4-10: Body inclination angle θ_b, wheel velocity $\dot{\theta}_w$, height from ground and motor torque against time plot of experiment through a double stepped terrain

Two consecutive spikes can be seen from Height-from-ground versus time plot at time t = 2.5s and t = 3s. These peaks correspond to the traversing motion of two consecutive steps. During the entire motion, the robot switches from ground phase to airborne phase (at t = 2.5s) and back to ground phase (at t = 2.8s) followed by another set of ground-to-airborne and airborne-to-ground phases at t = 3s and t = 3.2s respectively. Body angle fluctuates between –4° and 13° during the transition phases. The main reason of these fluctuations is due to un-modeled impact forces on landing. One problem we observed in this experiments is the change in forward direction during consecutive step. After the first step, impact force changes yaw heading direction of the robot. This causes the robot to fall and land unevenly. This rotational disturbance in roll direction cannot be actively compensated. However, the robustness of the controller has successfully kept the robot in balance. Figure 4-11 shows the corresponding snapshots of the experiment.

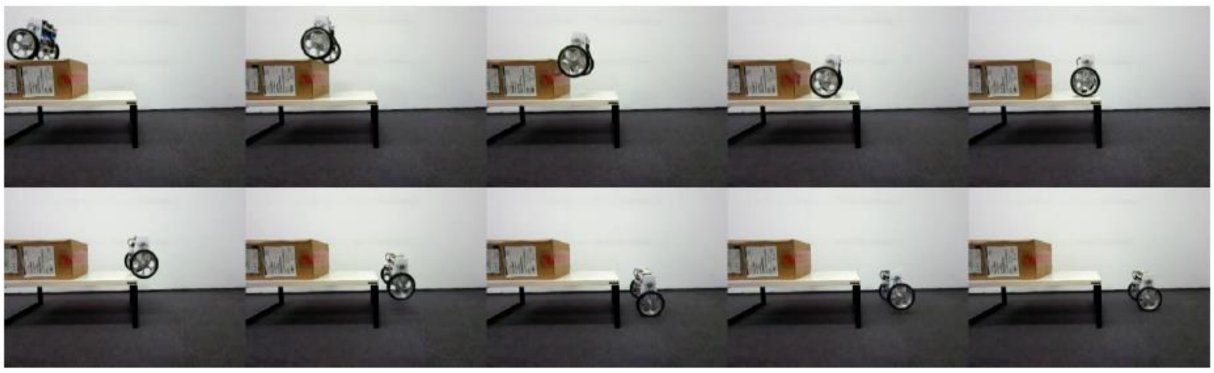


Figure 4-11: Experiment snap shots of the robot falling through two steps continuously

We have also conducted experiments on different environment and ground surfaces. Figure 4-12 shows the snapshots of the robot traversing concrete steps with slanting angle. The steps are relatively far apart. The controller effectively treat this terrain as a single step fall on different time interval. The steps are not flat and are slightly slanted. However, we found that slight slanting does not affect the behavior of the robot.

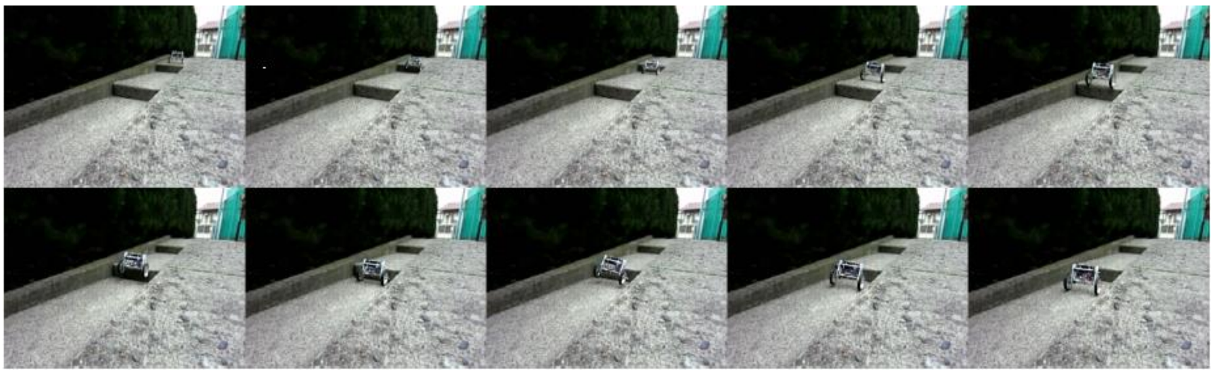


Figure 4-12: Snap shots of experiment of traversing on outdoor concrete stepped terrain

Figure 4-13 shows the experiment of the robot traversing through a flight of steps. As mentioned in previous section, due to short distance between consecutive steps, ground position control is temporarily turned off until the robot reaches the last step. Similarly the main problem encountered when traversing through continuous flight of steps is the change in forward direction. As can be seen from Figure 4-13, the robot falls with the right wheel first in the second step. This effect can permeate to the next step in increasing manner and may lead to failure. The current robot prototype does not have yaw control and can only move forward and backward. One solution to this problem is to use differential drive mechanism to align the robot to the edge before each fall.

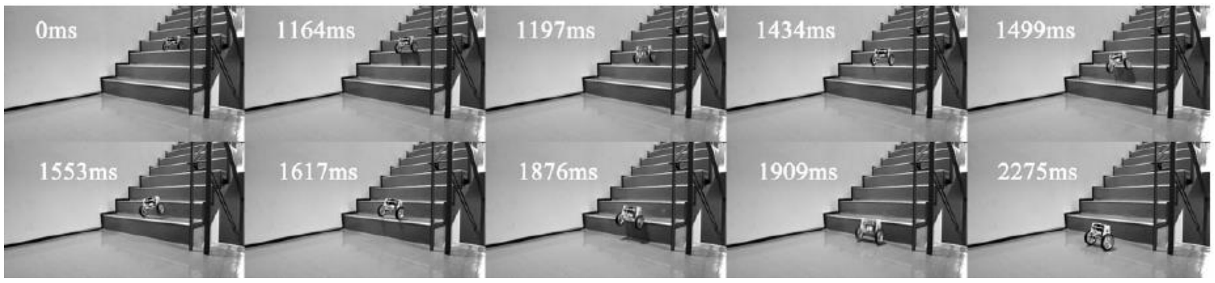


Figure 4-13: Experiment of the robot traversing three consecutive stair steps

4.5 Conclusion

In our experiment we have proven that it is possible for a two-wheeled robot to traverse through non-continuous stepped terrain dynamically. During freefall the robot uses the drive wheels to generate balancing torque to keep itself upright for a successful landing. The dynamic approach enables the robot to traverse through steps in a fast manner. Our prototype two-wheeled robot is able to traverse step up to a height of 70 cm. The two-wheeled robot is also able to traverse continuous steps. However, we found that without active yaw control, the robot is unable to recover from yaw misalignment when leaving the step. This causes rotational disturbance in roll direction. As the number of steps increases, in the worse case, the robot will fail to recover.

In the next chapter, we will introduce the second version of our prototype, BBot-2. The next prototype will try to tackle the problems that we found. We will introduce differential drive system to the platform to enable yaw control and step alignment ability. Suspension system will be introduced to mitigate the effect of rebound. Finally, an active hopping mechanism will be designed to attempt step traversing in upward direction as well.

Chapter 5
BBot - A hopping two wheeled robot

5.1 Introduction

The previous chapter introduces the implementation of a step traversing two wheeled robot. The implementation provides a foundation for the hopping two wheeled robot we will be introducing in this chapter. The improved version of the two wheeled robot, BBot-2 will attempt to address the shortcomings that was identified in the previous prototype.

In this chapter, we introduce the hopping two wheeled robot BBot-2. The robot utilizes the movement of a spring loaded mass to create momentum to achieve hopping motion. Our goal is to develop a mobile robot that is able to traverse common terrains such as steps and stairs in an efficient manner. The previous prototype BBot-1 can only move forward and backward. During landing, BBot-1 also experience recoil issue. In order to improve these problems, BBot-2 introduces a differentiate drive system to increase the mobility of the robot. The robot includes a hopping mechanism consists of a spring loaded movable upper body. This moveable upper body also reduces the impact recoil [30]. In the following sections we introduce the hopping mechanism and control algorithms. We discuss the experimental results and compare to the theoretical findings, as well as the limitations of the current model.

5.2 Mathematical model

In this section, we present the analysis of the robot during a hopping motion. We break down the hopping motion into three different phases [30]:

1) ground balance

2) pre-airborne impact

3) airborne balance.

Figure 5-1 shows the 3 phases during a hop. Separate models are derived and analyzed for each phase.

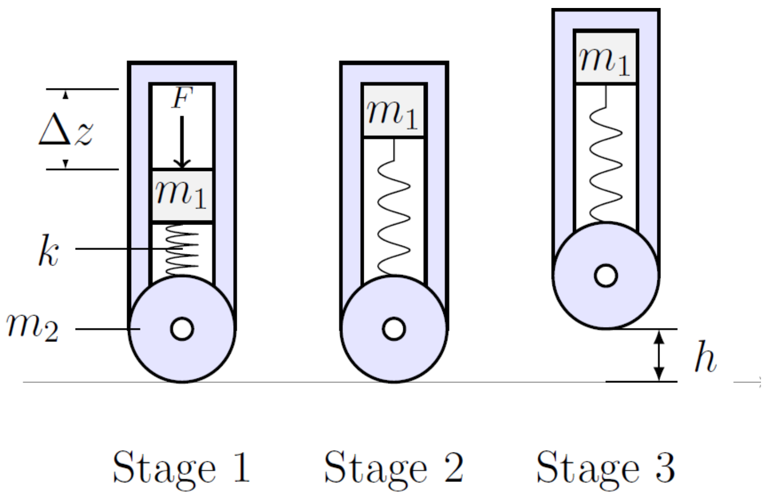


Figure 5-1: Three steps during hopping motion. Step 1) Spring is compressed to store potential energy. 2) Spring is released. Potential energy converts into kinetic energy 3) Conservation of momentum causes robot to lift off from ground.

5.2.1 Ground balance and airborne phase

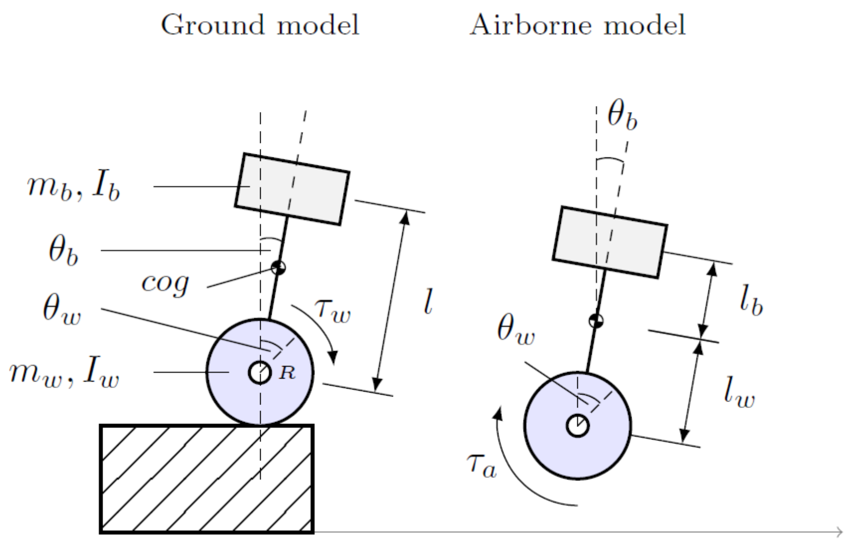


Figure 5-2: 2D model of ground and airborne model

Figure 5-2 depicts the dynamic behavior of the robot on the ground and in the air in simplified two-dimensional models. The ground and airborne model is identical to the model introduced in the previous chapter, detailed discussion will be omitted:

Ground model:

$$(J_w + (m_b + m_w)r^2)\ddot{\theta}_w + m_b r l \cos\theta_b \ddot{\theta}_b - m_b r l \dot{\theta}_b^{\,2} \sin\theta_b = -\tau_G$$

$$(J_b + m_b l^2)\ddot{\theta}_b + m_b r l \cos\theta_b \ddot{\theta}_w - m_b g l \sin\theta_b = \tau_G$$

Airborne mode:

$$(J_b + m_b {l_b}^2 + m_w {l_w}^2)\ddot{\theta}_b = -\tau_A$$

$$J_w \ddot{\theta}_w = \tau_A$$

5.2.2 Pre-airborne impact phase

The hopping capability of BBot-2 introduces an additional phase to the life cycle of the robot motion. Before a hop, BBot-2 undergoes a pre-airborne impact phase, which includes the following stages:

1. Compression of the spring increases potential energy.
2. Release of the spring converts potential energy into kinetic energy.
3. Upper body mass impacts on body frame and causes the robot to lift off from ground.

Hopping is achieve by converting the string potential energy to kinetic energy. The energy transition can be broken down in three stages.

1. The upper body mass is manually corked to store potential energy in the spring. Store potential energy is proportional to the spring constant k and compressed distance Delta z.
2. The upper body mass is released to unleashed stored potential energy. Potential energy is converted into kinetic energy. The upper body mass accelerates upwards.
3. The upper body mass impacts on the robot frame. The Law of Conservation of Momentum mandates changes in the velocities of the upper body mass and the lower body according to the following equation:

$$m_1 v_1 = (m_1 + m_2) v_2$$

With sufficient kinetic energy, the robot lifts off from the ground. Airborne phase begins the moment the wheels rise above ground. Jump height is directly proportional to the potential energy stored in the spring. Assuming potential energy is zero for the uncompressed spring (Delta z = 0), the potential energy of the system is

$$E_0 = \frac{1}{2} k \Delta z^2 - m_1 g \Delta z$$

Neglecting energy lost from friction, the total energy just before impact is

$$E_1 = \frac{1}{2} m_1 {v_1}^2$$

The Law of Conservation of Energy dictates that the velocity of m_1 prior to impact is

$$v_1 = \sqrt{\frac{2}{m_1} (\frac{1}{2} k \Delta z^2 - m_1 g \Delta z)}$$

Assuming a perfectly inelastic collision occurs after impact, the Law of Conservation of Momentum requires that the velocity of the robot obeys the following equation:

$$v_2 = \frac{m_1}{m_1 + m_2} v_1$$

The energy right after impact is

$$E_2 = \frac{m_1}{m_1 + m_2} E_0 = \frac{m_1}{m_1 + m_2} (\frac{1}{2} k \Delta z^2 - m_1 g \Delta z)$$

The maximum height of the jump is

$$h = \frac{E_2}{(m_1 + m_2)g} = \frac{m_1}{(m_1 + m_2)^2 g}(\frac{1}{2}k\Delta z^2 - m_1 g\Delta z)$$

Jump height increases in direct proportional to k and Delta z.

5.3 Design and Implementation

The stair climbing robot developed by kikuchi et al. [17] uses a movable upper body mass to create momentum for lift. The movement of the mass is constrained to the maximum extension length of the spring. This design requires a tall body frame to accommodate the full movement of the mass. The drawback of this design is that the large movement of the upper body mass would result in a large shifting in the center of mass. Such movement increate the difficulty to keep the robot balance, especially is statically unstable robot like BBot. To solve this problem our design limits the vertical movement of the upper body mass using a stopper. Figure 5-3 shows the 2D CAD model of BBot. Figure 5-4 shows the actual prototype of BBot. The upper body contains two 11.1V lipo batteries to power the motors and electronic components respectively. The upper body slides vertically along a slider. It is connected to the lower body frame with tension springs that pull upwards.

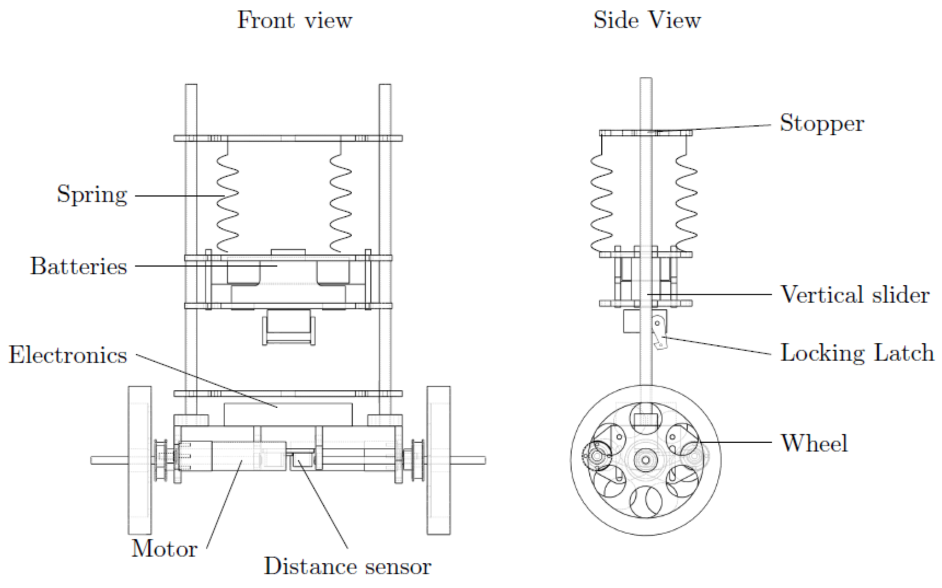


Figure 5-3: CAD Model of BBot-2

Spring tensioning mechanism is not implemented in this version of the prototype. The movable upper body mass is manually "corked" before performaning a jump motion. A servo activated latch controls the locking and releasing of the upper body. Varying the mass of the upper body affect the jump height.

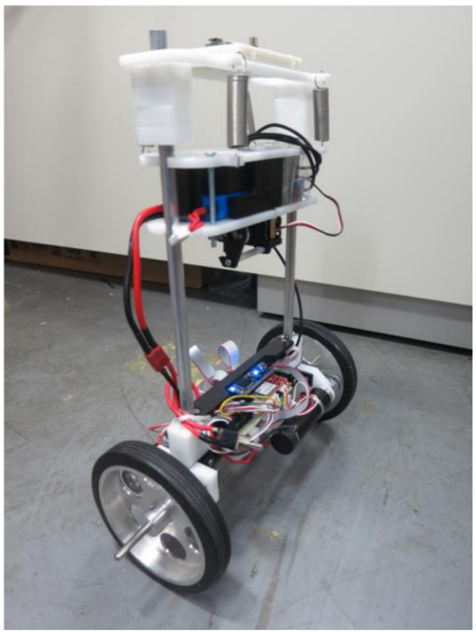


Figure 5-4: BBot-2 prototype

The lower body consists of two differential wheels powered separately by dc motors, a main electronic control board, and an ultrasonic distance sensor. The lower body mass is made as light as possible to achieve higher jump height. The energy conversion efficiency *n* is defined as the ratio of the kinetic energy at takeoff to the energy stored in the compressed spring before takeoff. The equation for conversion effiency is

$$\mathrm{n} = \frac{\mathrm{E}_2}{\mathrm{E}_0} = \frac{1}{1 + \mathrm{r}}$$

where $r=m_2=m_1$.

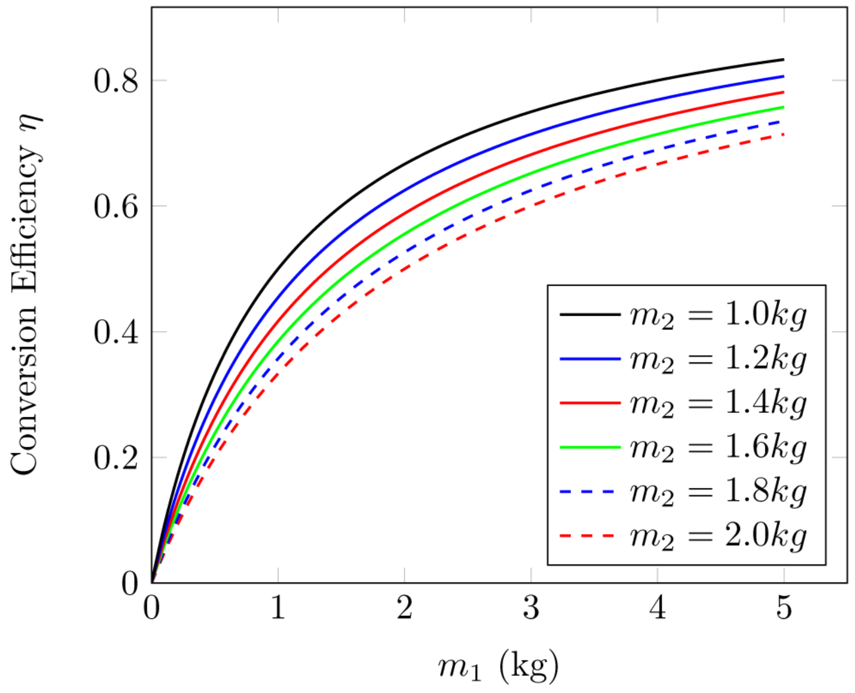


Figure 5-5: Plot of conversion efficiency for different upper body masses m_1 and lower body masses m_2 combination. Efficiency is proportional to m_1. Reducing m_2 increases efficiency.

Figure 5-5 plots the conversion efficiency for various combination of upper and lower body masses. Increasing upper body mass m1 increases the efficiency of energy conversion. For a fixed m1, lower body mass m_2 is inversely proportional to the efficiency of energy conversion. In other words, conversion efficency is maximized by increasing m_1 while minimizing m_2.

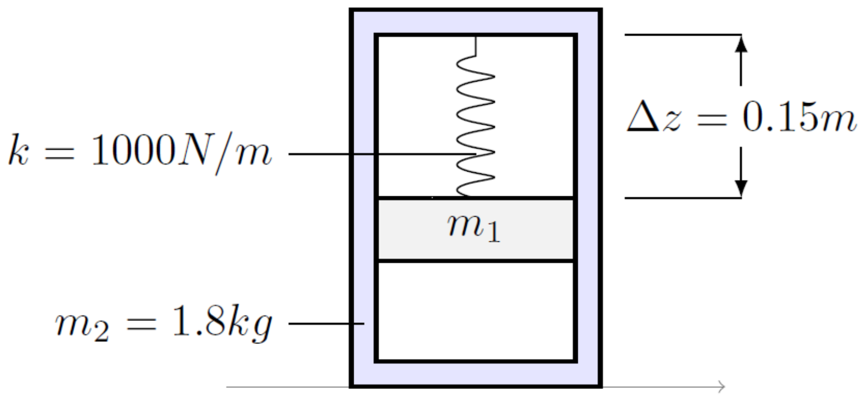


Figure 5-6: Simplified 2D model of the robot used for body weight simulation.

Figure 5-6 shows a simplified 2D model used to simulation the effect of body weight on jump height. We use Working Model 2D simulation software for simulation. The upper body mass m_1 is constrained to move vertically inside the robot frame m_2. Movement of m_2 is not contrained. m_2 and spring constant k are fixed. Gravitational force is set to 9.8m/s^2. Air resistance is ignore. Delta z is initialized to 0.15m.

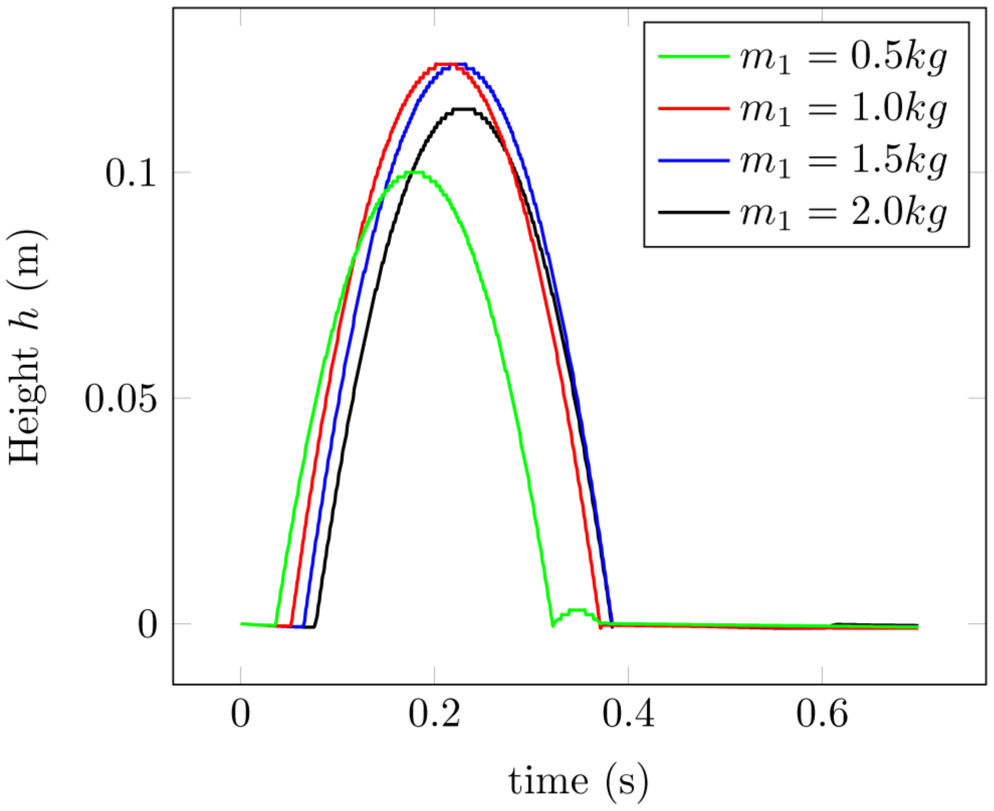


Figure 5-7: Height against time plot of various upper body mass

Figure 5-7 shows the plot of the jump height versus upper body masses. From the plot we can tell that the jump height is not linearly proportional to upper body mass. A small upper body mass does not generate sufficient jumping energy. A large upper body mass reduces the acceleration of the upper body mass and reduced generated jump force.

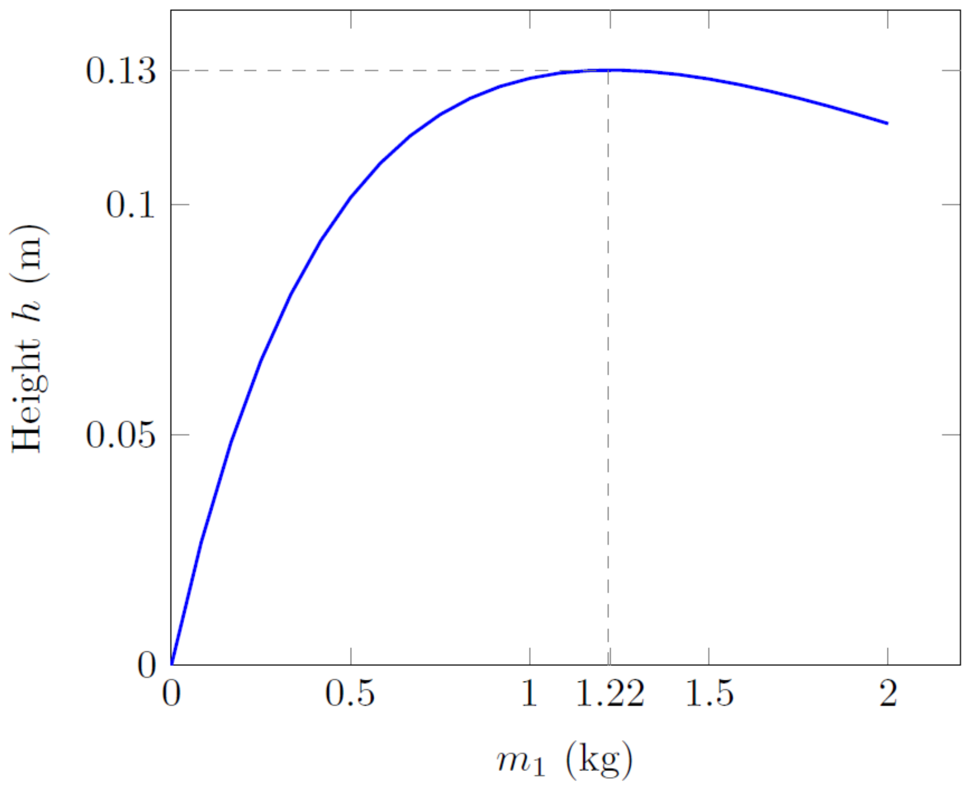


Figure 5-8: Height versus upper body mass plot

Figure 5-8 shows the plot of jump height h versus upper body mass m1. Optimum m1 occurs at the maximum point of the curve:

$$m_{1optimal} = \frac{m_2 k \Delta z}{4 m_2 g + k \Delta z}$$

Our design uses four separate springs arranged in parallel with a spring constant of 250N/m and m_2 of 1.8kg. m_1 is set at 1.2kg, close to the optimum of 1.22kg. The HxDxW dimensions are 300x160x420mm.

5.4 Balance and attitude control

The control scheme used in BBot-2 is similar to BBot-1, which parameters tuned to suit the physical parameters of BBot-2. The overview of the control scheme is shown in figure 5-9.

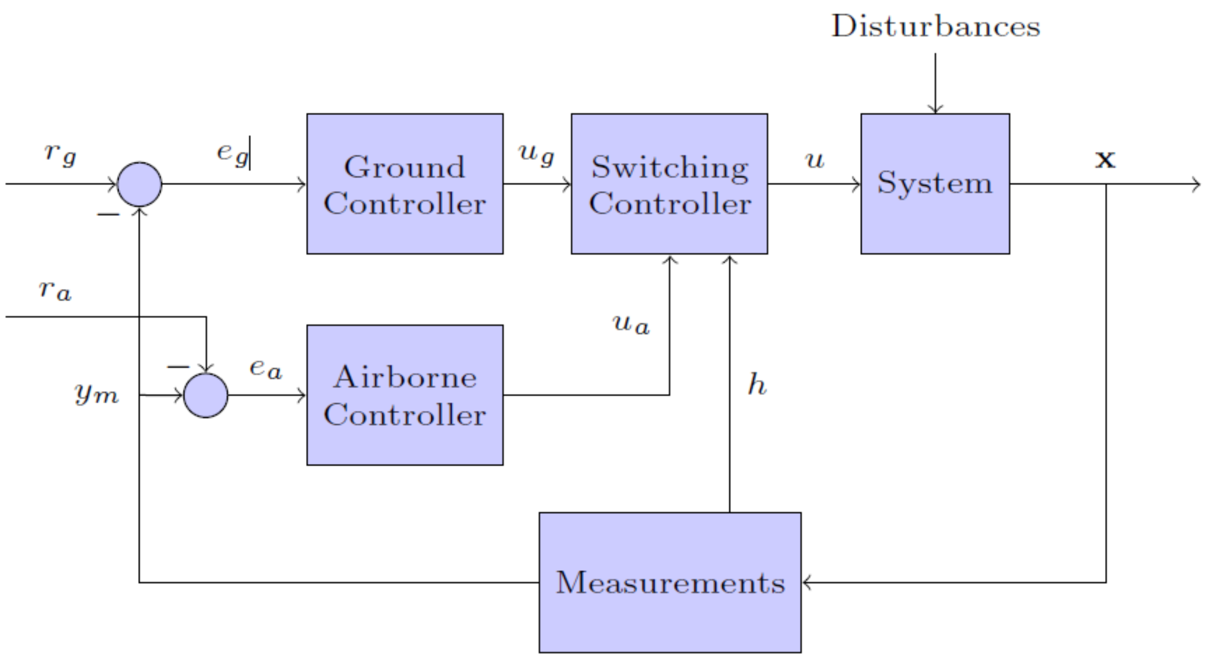


Figure 5-9: Overview of BBot-2 control scheme

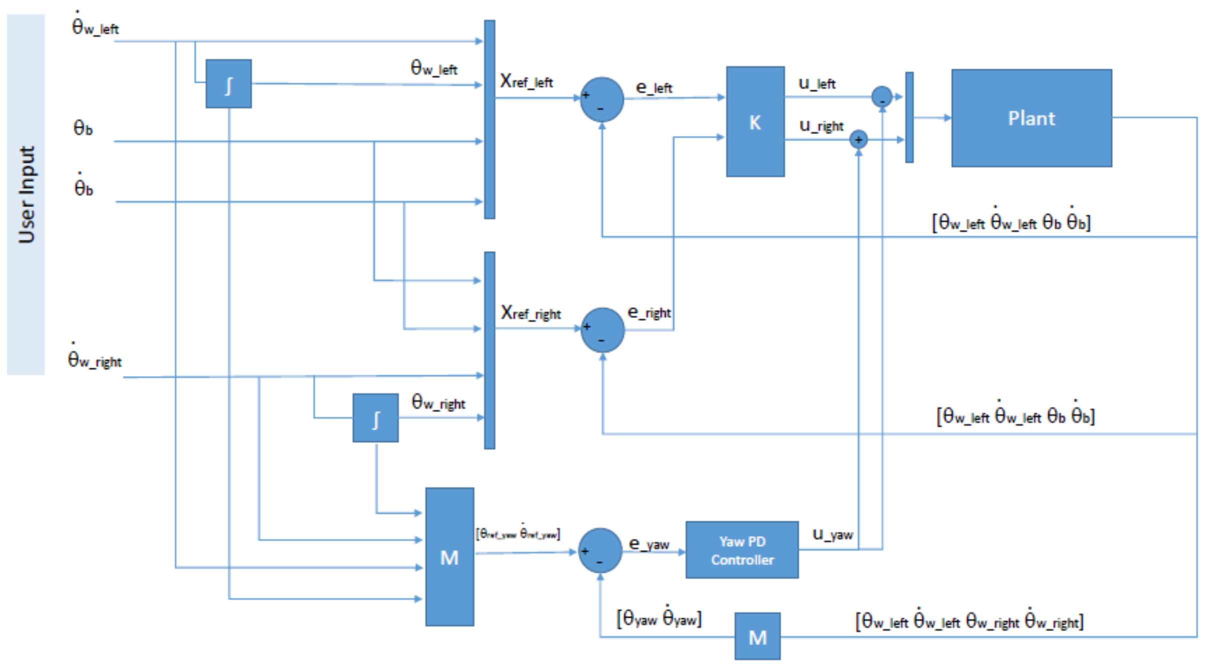


Figure 5-10: Feedback controller scheme for BBot-2. 4 user reference input left wheel velocity $\dot{\theta}_{wleft}$,right wheel velocity $\dot{\theta}_{wright}$ body tilt angle θ_b and body tilt angular velocity $\dot{\theta}_b$ are used to control the motion of BBot.

Figure 5-10 shows the detailed feedback control block diagram of the system. In addition to forward and backward motion, the newly introduced differential drive enables BBot to rotate in yaw direction. An additional yaw controller is used to control yaw movement. The control input states are left wheel velocity $\dot{\theta}_{wleft}$, right wheel velocity $\dot{\theta}_{wright}$, body tilt angle θ_b and body tilt angular velocity $\dot{\theta}_b$. Setting θ_b and $\dot{\theta}_b$ to zero will keep the robot upright. $\dot{\theta}_{wleft}$ and $\dot{\theta}_{wright}$ reference controls the motion of the robot.

We have introduced an additional yaw controller to control the yaw of the robot:

$$u_{yaw} = K_{yaw}(r_{yaw} - x_{yaw})$$

$$r_{yaw} = \begin{bmatrix}\theta_{yawref} & \dot{\theta}_{yawref}\end{bmatrix}^T$$

$$x_{yaw} = \begin{bmatrix}\theta_{yaw} & \dot{\theta}_{yaw}\end{bmatrix}^T$$

where the yaw angle θ_{yaw} and yaw velocity $\dot{\theta}_{yaw}$ are the differences between left and right wheel angles and wheel velocities.

5.4.1 Height and phase transition detection

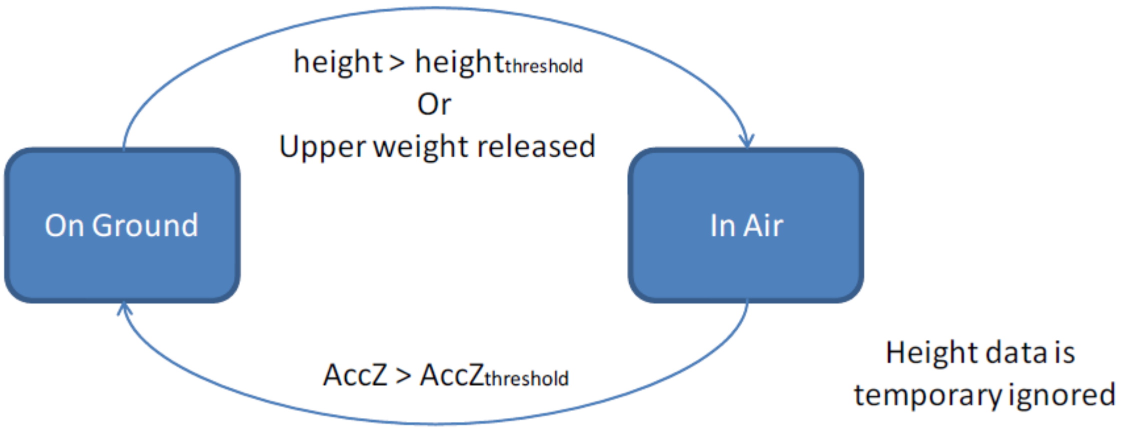


Figure 5-11: Flow diagram showing the phase transition detection scheme using both ultrasonic sensor and accelerometer data.

Similar to BBot-1, we used an ultrasonic distance sensor to measure the height of the robot from ground. The sensor is placed at the bottom of the robot facing down. In BBot-1, a simple thresholding on the distance sensor is used to determine the phase transition of the robot from air to ground. This approach tends to have false detection. When the robot is moving, a large tilt angle will cause ultrasonic waves from the sensor the reflects away from the sensor. To improve phase detection robustness, we includes accelerometer data for impact detection. When the robot lands, a distinct acceleration data from the accelerometer can be used to determined if impact occurs reliably.

5.5 Experiment Results

BBot-2 is a self-contained robot with an on-board microcontroller processing all sensor data and perform balance control the robot. BBot-2 connects to a host pc via Bluetooth connection for remote control. Real time sensor data is streamed to the host pc and logged at a rate of 100Hz. We conducted the step traversing experiment, manual tossing experiment and a hopping experiment to validate the performance of BBot-2

5.5.1 Step traversing experiment

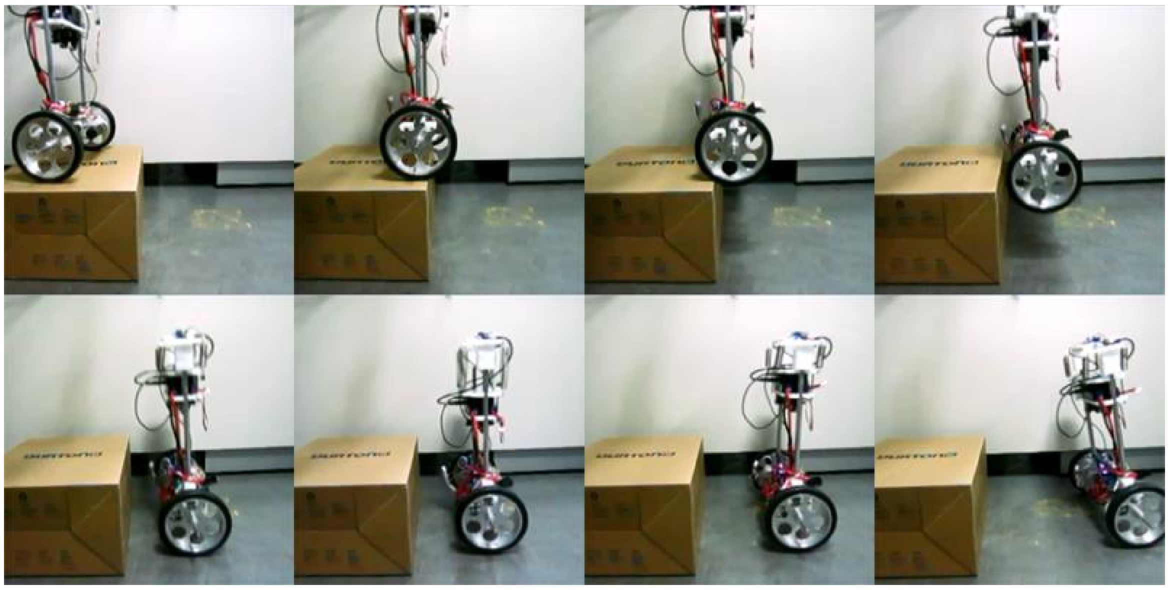


Figure 5-12: The robot traverses a stepped terrain of height 17cm. The robot uses the drive wheels to generate balancing torque to control its attitude in air. Upon landing, the robot switches to ground balance mode to keep balance.

In this experiment, the robot traverse a single step terrain. The step height is 17cm. Figure 5-12 shows the snap shots of the motion. Figure 5-13 shows the corresponding raw sensor data plot against time. Height data plots the height calculated from the ground to the sonar sensor. The height data has an offset of 5cm above ground when the robot is balancing still, indicating the ultrasonic distant sensor is attached 5cm above ground. AccZ graph plots the linear acceleration along z axis (pointing up). At time t = 1.86s, the peak in the measurement indicates that the robot is not in contact with the ground and is freefalling. The controller switches into airborne control mode. We have set the reference body tilt angle for airborne controller to a small positive value (+6 degrees), to tilt the robot backwards during airborne. The reason for this is to compensate the forward momentum during landing and reduce the torque needed to balance upon landing. BBot-2 has a higher center of mass compare to BBot-1. This is analogous to landing with feet in front and uses momentum to bring the body to a neutral position. At time t = 2.02s, large fluctuation in the accelerometer readings indicates a landing event. Phase changed is detected and the robot switches into ground mode to keep its balance. At the moment of impact upon landing, we observed the upper body mass moves downward to absorb the impact force from the ground. Compare to BBot-1, the rebound effect is mitigated by this mechanism. From the height versus time plot, there is a false positive indication of increase height right after landing. This is due to the fact that ultrasonic waves bounds away from the ground when the body tilt angle is large. The overall performance of the BBot-2 is more stable compare to the previous prototype.

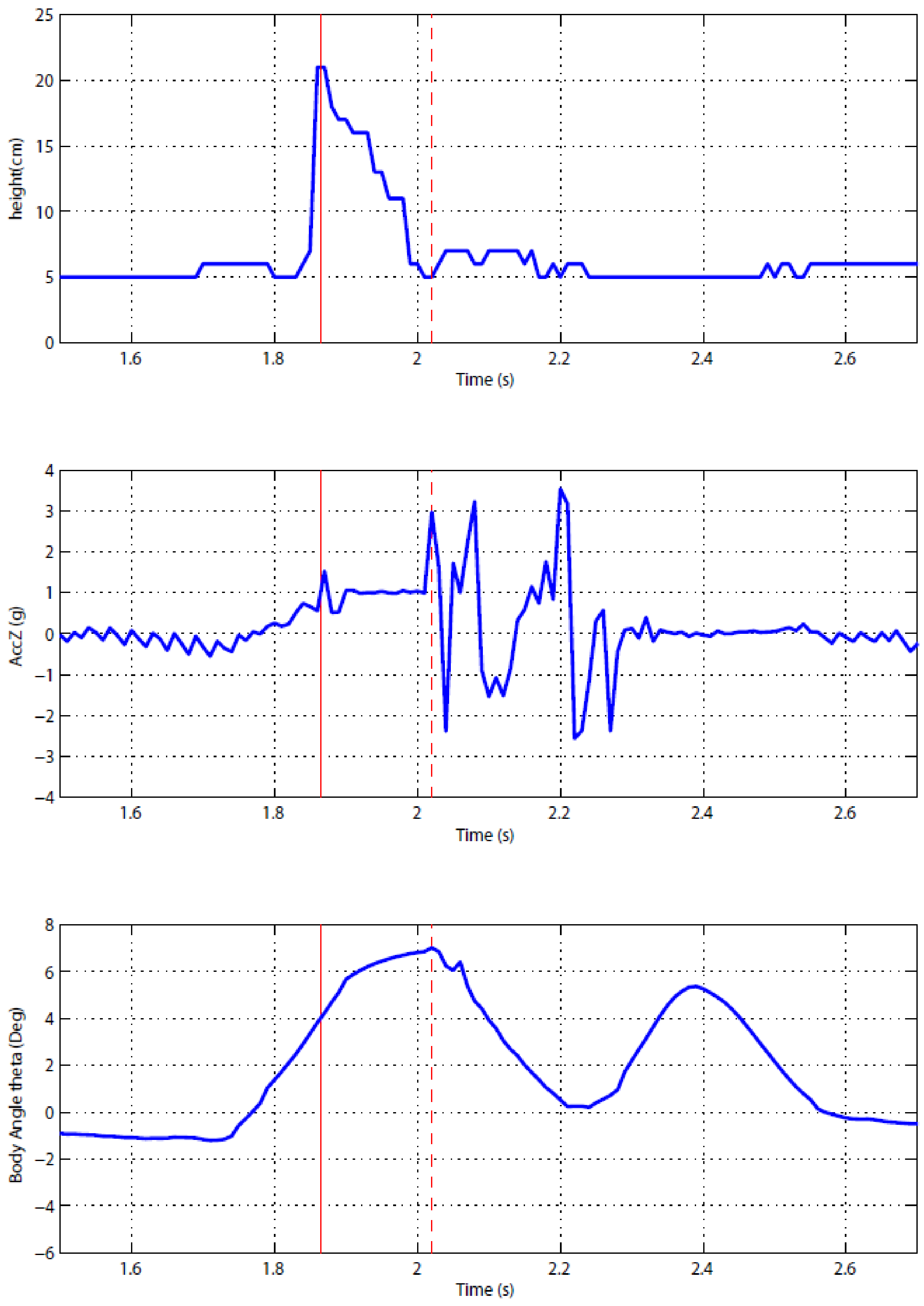


Figure 5-13: Height, vertical acceleration and body angle plot against time during step traverse. Solid red line and dotted red line indicates the beginning and the end of an airborne phase.

5.6.2 Toss landing experiment

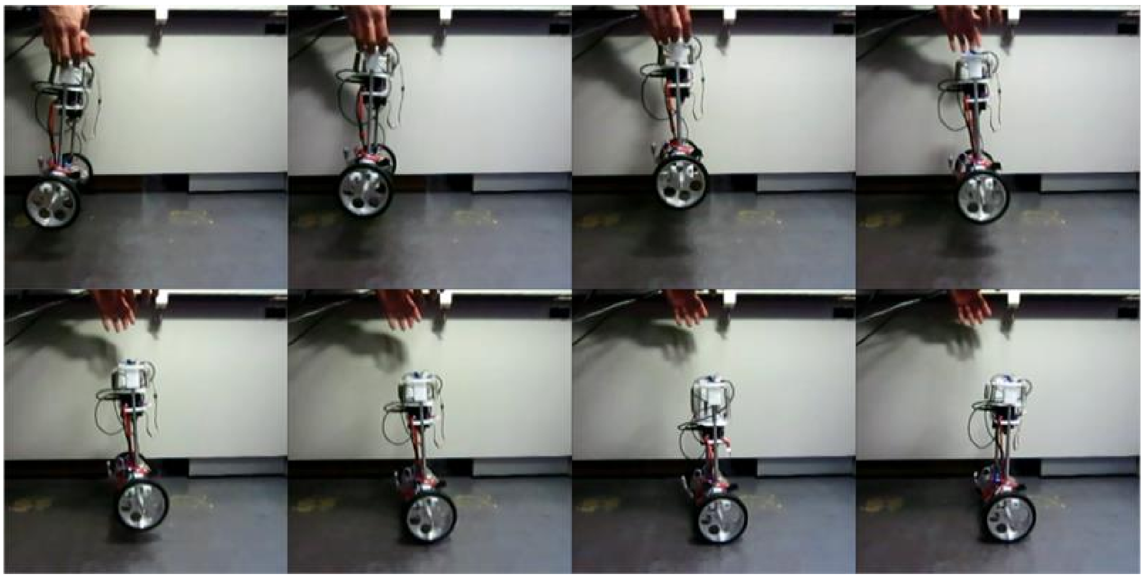


Figure 5-14: Snapshot of the toss landing experiment.

The next experiment is designed to test the stability of the robot during airborne. When tossing the robot, rotational torque is introduced. Without attitude control the rotational torque will cause the robot to tilt away from vertical and will not be able to maintain balance upon landing. When attitude control is on, the controller constantly keep robot close to vertical position. Upon landing, the small tilt angle require less balancing torque to keep the robot upright without saturating the motors. Figure 5-14 shows the snapshots of the toss landing experiment. Figure 5-15 shows sensor data plotted against time. Red solid lines in the graphs indicate the instant when the robot is released. At time t = 4.5s the robot reaches peak height and starts to free fall. During this period, the airborne controllers is activated to maintain a positive body tilt angle in air. Landing occurs at time t = 4.7s. The toss experiment introduces large rotational torque to the robot. From time t = 4.7s ~ 5.5s, the robot rocks back and forth to keep balance before settling down on a stable upright position at t > 6s. The motor duty graph shows that the pwm duty for the motor, which is proportional to the torque apply to the motor. The maximum pwm duty of the motor is below 40%, indicating that the motor is not saturated at any point of time during the whole motion.

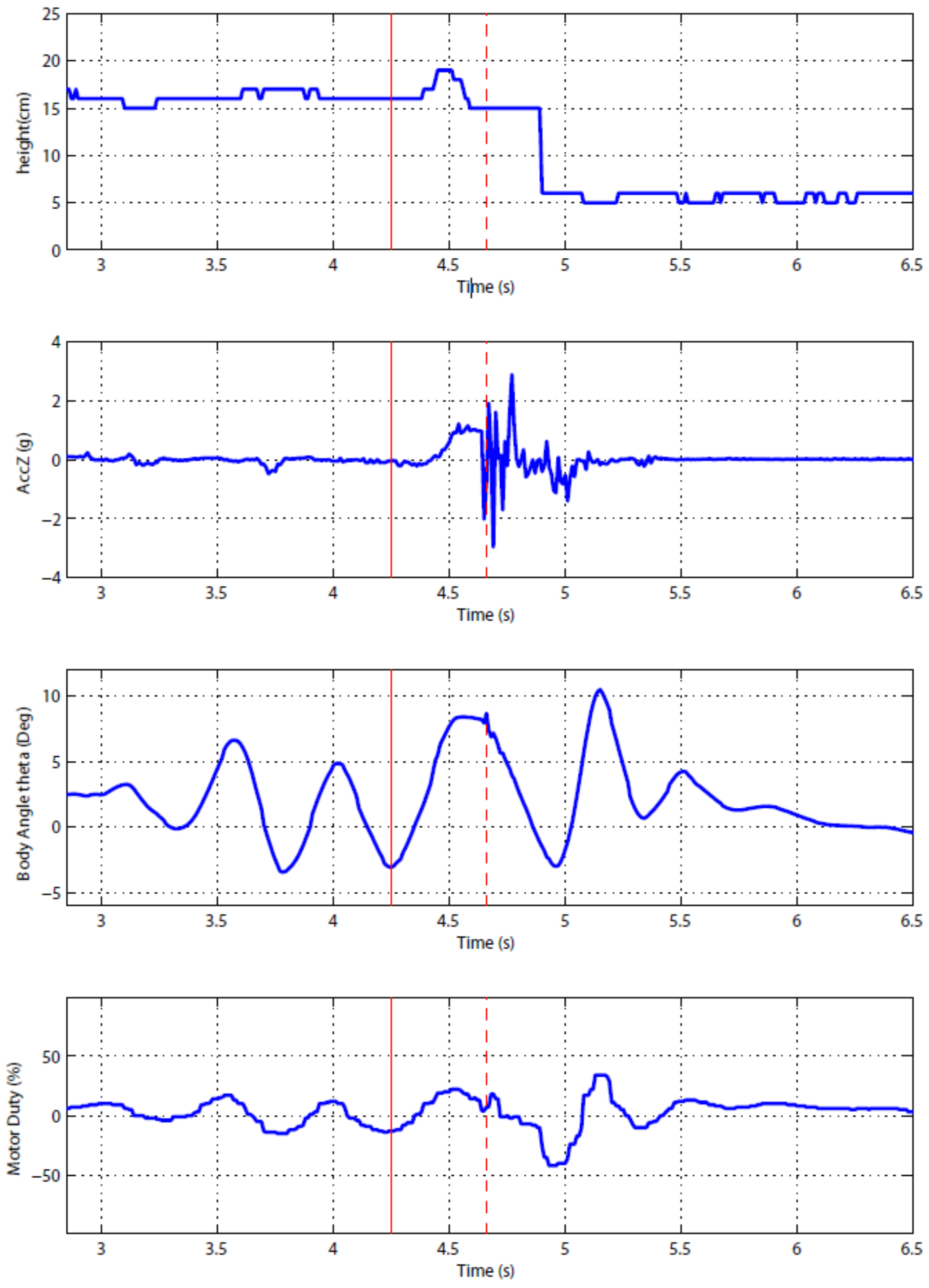


Figure 5-15: Vertical acceleration, body angle and motor during plot against time. Red solid line indicates the time when the robot is released. Dashed line shows the instant when the robot hits the ground. Throughout the motion, the motor duty is below 50%, indicating the motor is not saturated.

5.6.3 Hopping experiment

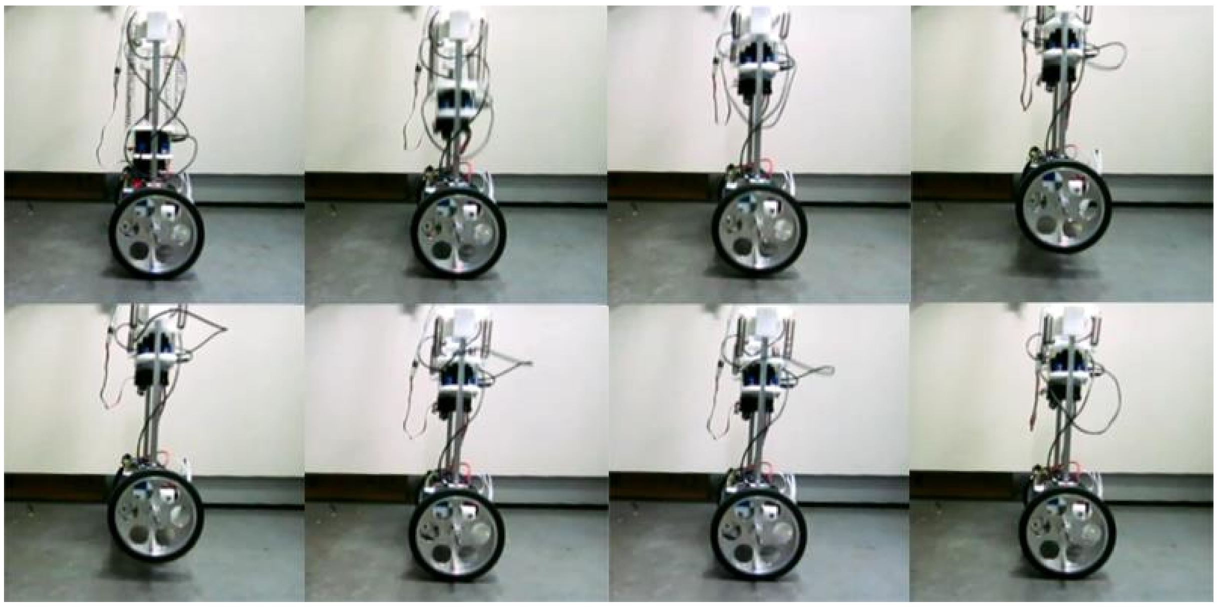


Figure 5-16: Hopping motion demonstrated by the prototype robot.

In this section, we test the hopping ability of BBot-2. The springs are pre-tensioned by manually corking the upper body weight. A servo activated locking mechanism is used to lock the upper body weight in place. The servo is remotely controlled to release the movable upper body. Figure 5-16 shows the snapshots of a hopping action. Figure 5-17 shows the sensor data plots against time. The upper body is accelerated upwards when the lock is released. The upper body mass impacts on the stopper at time t = 2.38s, converting stored potential energy into lifting force. This lifting force causes the robot to jump. The robot switches into airborne mode. At time t = 2.55s, large accelerometer reading indicates robot landing. The controller switches back to ground balance mode. The hopping motion is completed. From height plot, the jump height can be deduced from the maximum changed in height, i.e. roughly 4cm. The actual jump height is less than simulated results due to the following reasons:

1. The simulation does not take into account friction force generated in the sliding joint. The friction contributes to energy lost.
2. Discrepancy between simulated spring constant and actual spring constant used in the robot.

3. Simplified assumption that the impact collision is fully inelastic in simulation. Actual impact collision event is more complex. Energy lost during energy transition is not taken into account in the simulation

Nevertheless, the prototype BBot-2 proved that the proposed approach enables the robot to achieve hopping motion.

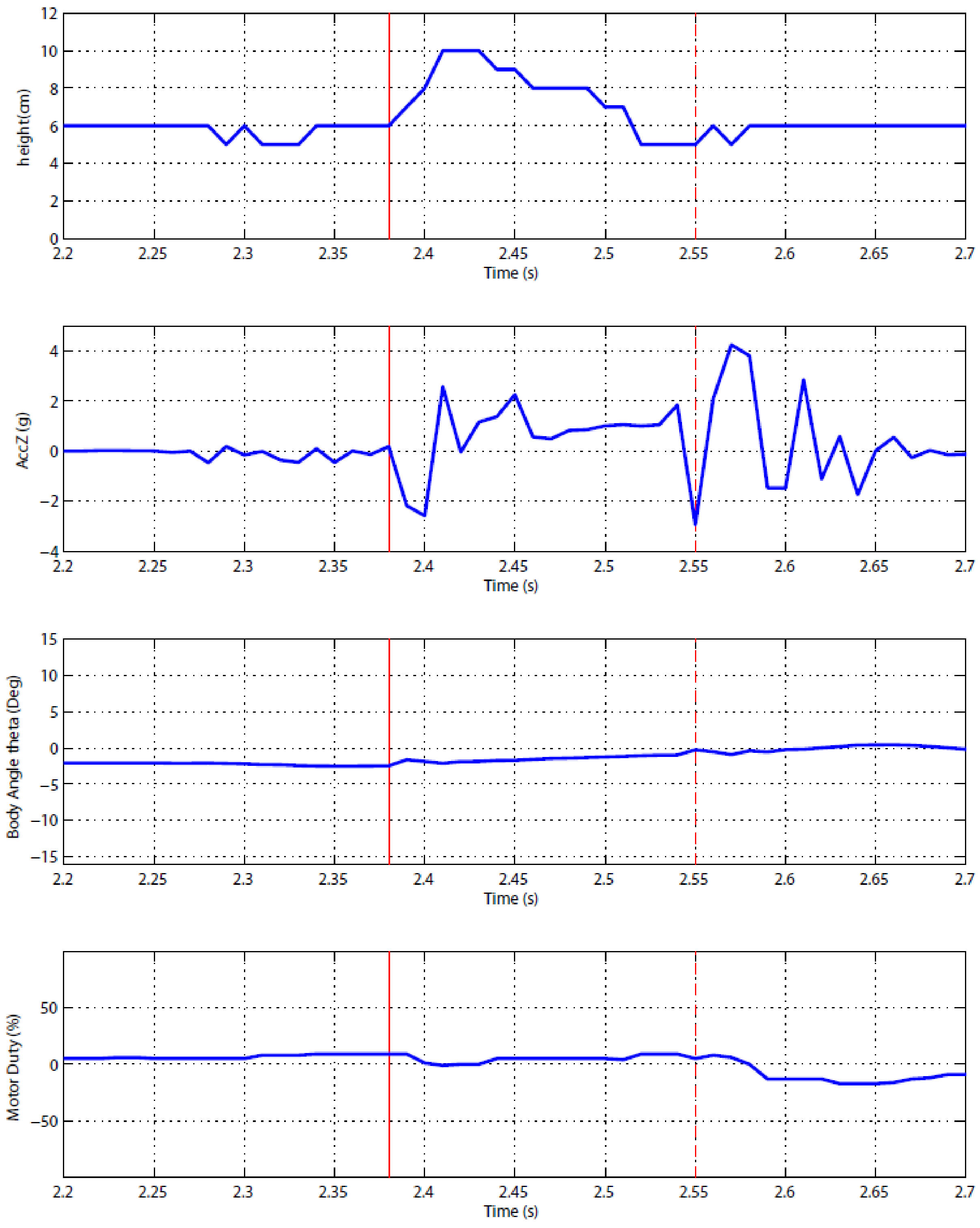


Figure 5-17: Acceleration, body angle and motor duty plots of the hopping motion. Red solid line indicates the instant when hopping starts (t = 2.38s). Dashed line indicates the moment the robot lands (t = 2.55s).

5.7 Conclusion

This chapter introduced the implementation of a hopping two wheeled robot, BBot-2. The modeling and analysis, hardware construction and control scheme are presented. We have built a prototype and conducted experiments to verify the performance of the robot. The experiments confirms that BBot-2 is able to traverse stepped terrain and performs hopping motion. However we observed a few limitations of BBot-2. Current jump height of the robot is relatively small. This is due to the mechanical constructions of the robot. In order to improve the jump height of the robot, the current sliding joint has to be replaced by a lubricated ball bearing sliding joint to further reduce the friction. The robot has to be able to clear a jump height larger than 17cm to be able to climb up stairs. One solution to this problem is to scale up the prototype to accommodate larger springs to create larger lifting force.

One problem we observed in the hopping mechanism design is the recoil effect when the upper body impacts on the body frame. The collision is partially inelastic, contrast to assumption of a perfectly inelastic collision in the simulation. This effect reduces the energy conversion efficiency and hence reduce the total jump height. In the current prototype, parameters such as the airborne reference tilt angle is experimentally decided. For future work, we plan to investigate the relationship between the reference tilt angle, forward velocity and jump height so that the robot is able to determine the optimal attitude during airborne automatically.

Chapter 6
Conclusions and Future work

Non-continuous terrains, such as steps and stairs, pose a challenging obstacles for mobile robots. Existing robots usually employs a static and slow approach to clear a stepped obstacles. In contrast, animals tends to jump over a step in a swift and dynamic manner. In this research, we aim to develop a robot that is able to negotiate a non-continuous terrain in a dynamic manner. We have created BBot, a two wheeled robot capable of traversing stepped terrain and hopping. We proposed to use the driving wheels of the robot to generate balancing torque when the robot is in air. Using the drive wheels eliminate the need for an additional actuator. The propose method enables conventional two wheeled robot to traverse stepped terrain without modification. The dynamic nature of this approach enables the robot to negotiate stepped terrain in a dynamic and fast manner. We have presented the theoretical analysis as well as the design of a working prototype. We have also presented a control scheme for the robot based on our theoretical analysis. Experiment shows that the proposed method enables a two wheeled robot to traverse a stepped terrain, which is not possible before.

Current prototype, BBot-2, is able to travel down stepped terrain effectively. The robot is capable of hopping 4cm above ground. The hopping height is significantly less than simulated results due to mechanical constraints and perfect world assumption in the simulation. Current prototype suffers from a few limitations. The current mechanical design does not convert stored potential energy to lifting force effectively. Assumptions of perfectly inelastic collision of the upper body mass with the chassis does not reflects the real situation. An improved model of the system needs to be investigated to better model the real world situation. Current jump height is small for practical application in real world environment.

Bibliography

[1] E. Z. Moore, D. Campbell, F. Grimminger and M. Buehler, "Reliable stair climbing in the simple hexapod 'RHex'," Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on, Washington, DC, 2002, pp. 2222-2227.

[2] Saranli, U., Buehler, M., & Koditschek, D. E. (2001). Rhex: A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, *20*(1), 616–631. doi:10.1177/02783640122067570.

[3] Volpe, R., Balaram J., Ohm T., Ivlev T. (1996) The Rocky 7 Mars Rover Prototype. IEEE/RSJ International Conference on Intelligent Robots and Systems, November 4-8 1996, Osaka Japan

[4] Hayati, S., Volpe, R., Backes, P., Balaram, J., Welch, R., Ivlev, R., & Laubach, S. (1997). The rocky 7 rover: A mars science craft prototype. In Proceedings of Robotics and Automation, 1997 (vol. 3, pp. 2458 2464). IEEE. doi: doi:10.1109/ ROBOT.1997.619330.

[5] Estier T., Crausaz Y., Merminod B., Lauria M., Piguet R., Siegwart R. (1994) An innovative Space Rover with Extended Climbing Abilities, Proceedings of Space and Robotics 2000, Albuquerque, USA, February 27-March 2, 2000.

[6] Lamon, P., Krebs, A., Lauria, M., Siegwart, R. & Shooter, S. (2004), Wheel torque control for a rough terrain rover, in 'Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on', Vol. 5, pp. 4682 – 4687 Vol.5.

[7] Hirose, S., Fukushima, E., Damoto, R. & Nakamoto, H. (2001), Design of terrain adaptive versatile crawler vehicle helios-vi, in 'Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on', Vol. 3, pp. 1540 –1545 vol.3.

[8] Guarnieri, M., Takao, I., Fukushima, E., & Hirose, S. (2007). Helios VIII search and rescue robot: Design of an adaptive gripper and system improvements. In *Proceedings of Intelligent Robots and Systems, 2007* (pp. 1775–1780). IEEE. doi:10.1109/IROS.2007.4399372.

[9] Siles, I. & Walker, I. (2009), Design, construction, and testing of a new class of mobile robots for cave exploration, in 'Mechatronics, 2009. ICM 2009. IEEE International Conference on', pp. 1 –6.

[10] Kovac, M., Fuchs, M., Guignard, A., Zu_erey, J.-C., Floreano, D.: A miniature 7g jumping robot. In: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference On, pp. 373{378 (2008). doi:10.1109/ROBOT.2008.4543236

[11] Scarfogliero, U., Stefanini, C., Dario, P.: A bioinspired concept for high efficiency locomotion in micro robots, the jumping robot grillo. In: Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference On, pp. 4037{4042 (2006). doi:10.1109/ROBOT.2006.1642322

[12] Zhao, J., Xi, N., Gao, B., Mutka, M.W., Xiao, L.: Development of a controllable and continuous jumping robot. In: Robotics and Automation (ICRA), 2011 IEEE International Conference On, pp. 4614{4619 (2011).doi:10.1109/ICRA.2011.5980166

[13] Niiyama, R., Nagakubo, A., Kuniyoshi, Y.: Mowgli: A bipedal jumping and landing robot with an artificial musculoskeletal system. In: Robotics and Automation, 2007 IEEE International Conference On, pp. 2546-2551(2007). doi:10.1109/ROBOT.2007.363848

[14] Lambrecht, B.G.A., Horchler, A.D., Quinn, R.D.: A small, insect-inspired robot that runs and jumps. In:Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference On, pp. 1240-1245 (2005). doi:10.1109/ROBOT.2005.1570285

[15] Armour, R., Paskins, K., Bowyer, A., Vincent, J., Megill, W.: Jumping robots: a biomimetic solution to locomotion across rough terrain. Bioinspiration and Biomimetics 3(3), 039801 (2008)

[16] Stoeter, S.A., Papanikolopoulos, N.: Kinematic motion model for jumping scout robots. Robotics, IEEETransactions on 22(2), 397{402 (2006). doi:10.1109/TRO.2006.862483

[17] Kikuchi, K., Sakaguchi, K., Sudo, T., Bushida, N., Chiba, Y., Asai, Y.: A study on a wheel-based stair-climbingrobot with a hopping mechanism. Mechanical Systems and Signal Processing 22(6), 1316-1326 (2008).doi:10.1016/j.ymssp.2008.03.002. Special Issue: Mechatronics

[18] Schmidt-Wetekam, C., Zhang, D., Hughes, R., & Bewley, T. (2007). Design, optimization, and control of a new class of reconfigurable hopping rovers. In *Proceedings of Decision and Control* (pp. 5150–5155). IEEE. doi:10.1109/ CDC.2007.4434975.

[19] Schmidt-Wetekam, C., Bewley, T.: An arm suspension mechanism for an underactuated single legged hoppingrobot. In: Robotics and Automation (ICRA), 2011 IEEE International Conference On, pp. 5529-5534 (2011).doi:10.1109/ICRA.2011.598033

[20] Astrom, K., Block, D.J., &Spong, Mark. W. (2007) The reaction wheel pendulum. New York: Morgan and Claypool

[21] Spong, Mark. W, Corke, Peter , & Lozano, Rogelio (2001) Nonlinear control of the Reaction Wheel Pendulum. Automatica

[22] Raibert, M., Blankespoor, K., Nelson, G., Playter, R.: Bigdog, the rough-terrain quaduped robot. In: Automatic Control, 2008 The International Federation Of (2008)

[23] Grasser, F., Arrigo, A., and Colombi, S. (2002) JOE: A mobile Inverted Pendulum, IEEE Transactions on Industrial Electronics, 49, 107-114]

[24] Stilman, M., Olson, J., & Gloss, W. (2010). Golem Krang: Dynamically stable humanoid robot for mobile manipulation. In Proceedings of Robotics and Automation (pp. 3304-3309). IEEE. doi:10.1109/ROBOT.2010.5509593.

[25] Yap, H.E., Hashimoto, S.: Spherical 2D Inverted Pole Balancing using Inertia Flywheels, RSJ2009

[26] Yap, H.E., Hashimoto, S.: Switching control of mobile reaction wheel pendulum, Proceedings of the 2010 JSME Conference on Robotics and Mechatronics, June 2010

[27] Yap, H.E., Hashimoto, S.: Attitude control of an airborne two wheeled robot, Proceedings of the Seventeenth International Symposium on Artificial Life and Robotics (AROB 17th '12)

[28] Yap, H.E., Hashimoto, S.: Development of a stair traversing two wheeled robot. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference On, pp. 3125{3131 (2012).doi:10.1109/IROS.2012.6385767

[29] Yap, H.E., Hashimoto, S.: Dynamic step traverse of a two wheeled robot, International Journal of Mechatronics and Manufacturing Systems, Special Issue on Advances in Robotics and Mechatronics, Vol.6, No.1, pp.3-22, 2013

[30] Yap, H.E., Hashimoto, S.: BBot, a hopping two wheeled robot with active airborne control, 2016 Robomech Journal, 3(1), 1-15, DOI: 10.11186/s40648-016-0045-3

[31] Burdick, J., Fiorini, P.: Minimalist jumping robots for celestial exploration. The International Journal of Robotics Research 22(7-8), 653{674 (2003). doi:10.1177/02783649030227013.http://ijr.sagepub.com/content/22/7-8/653.full.pdf+html

[32] Batista, P., Silvestre, C., Oliveira, P., Cardeira, B.: Low-cost attitude and heading reference system: Filter design and experimental evaluation. In: Robotics and Automation (ICRA), 2010 IEEE International Conference On, pp. 2624-2629 (2010). doi:10.1109/ROBOT.2010.5509537

[33] Lin, Z., Zecca, M., Sessa, S., Bartolomeo, L., Ishii, H., Takanishi, A.: Development of the wireless ultra-miniaturized inertial measurement unit wb-4: Preliminary performance evaluation. In: Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE, pp. 6927–6930 (2011). doi:10.1109/IEMBS.2011.6091751

[34] Madgwick, S.O.H., Harrison, A.J.L., Vaidyanathan, R.: Estimation of imu and marg orientation using a gradient descent algorithm. In: Rehabilitation Robotics (ICORR), 2011 IEEE International Conference On, pp. 1–7 (2011). doi:10.1109/ICORR.2011.5975346

[35] T. Lauwers, G. Kantor, and R. Hollis, A dynamically stable single-wheeled mobile robot with inverse mouseball drive, in Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, 2006, pp. 2884-2889

[36] http://world.honda.com/ASIMO/

Research Achievements

Journals

1. Huei Ee YAP, Shuji Hashimoto, "BBot, a hopping two-wheeled robot with active airborne control", Robomech Journal, 3(1), 1-15, DOI: 10.11186/s40648-016-0045-3 March 2016.

2. Huei Ee YAP, Shuji Hashimoto, "Dynamic step traverse of a two-wheeled robot", International Journal of Mechatronics and Manufacturing Systems, Special Issue on Advances in Robotics and Mechatronics, Vol.6, No.1, pp.3-22, 2013 January 2013.

International Conferences

1. Huei Ee YAP, Shuji Hashimoto, "Development of a stair traversing two wheeled robot", IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, October 2012.

2. Huei Ee YAP, Shuji Hashimoto, "Attitude control of an airborne two wheeled robot", Proceedings of the Seventeenth International Symposium on Artificial Life and Robotics (AROB 17th '12) January 2012.

Domestic Conferences

1. Huei Ee YAP, Shuji Hashimoto, "Switching Control of Mobile Reaction Wheel Pendulum", Proceedings of the 2010 JSME Conference on Robotics and Mechatronics June 2010.

2. Huei Ee YAP, Shuji Hashimoto , "Spherical 2D Inverted Pole Balancing using Inertia Flywheels", 27th Annual Conference of the RSJ, July 2009.

Other Publications

1. D.Matsuoka, G.Enriquez, H.E.Yap, S.Hashimoto, "Development of a Lightweight Manipulator with Constraint Mechanism", Proc. of 2014 International Symposium on Micro-Nano Mechatronics and Human Science(MHS2014),pp83-86, November 2014.

2. 小池宇織, Huei Ee Yap, Enriquez Guillermo, 山口友之, 三輪貴信, 橋本周司, "9 軸姿勢センサを用いた口腔内インタフェース", 2014 年電子情報通信学会総合大会基礎･境界論文集

3. 西尾智彦, Enriquez Guillermo, Huei Ee Yap, 山口友之, 橋本周司, "振動型ハプティックディスプレイの知覚精度", 2014 年電子情報通信学会総合大会基礎･境界論文集

4. 山田宏佑, Enriquez Guillermo, Huei Ee Yap, 橋本周司, "水上マルチユニットロボットによる物体の搬送" , 平成 26 年度電気学会全国大会

5. 山本一哉, Huei Ee Yap, 橋本周司, "エネルギー自給型水中ロボットの開発と効率解析", ロボティクス・メカトロニクス講演会 2013, pp. 2A2-M01

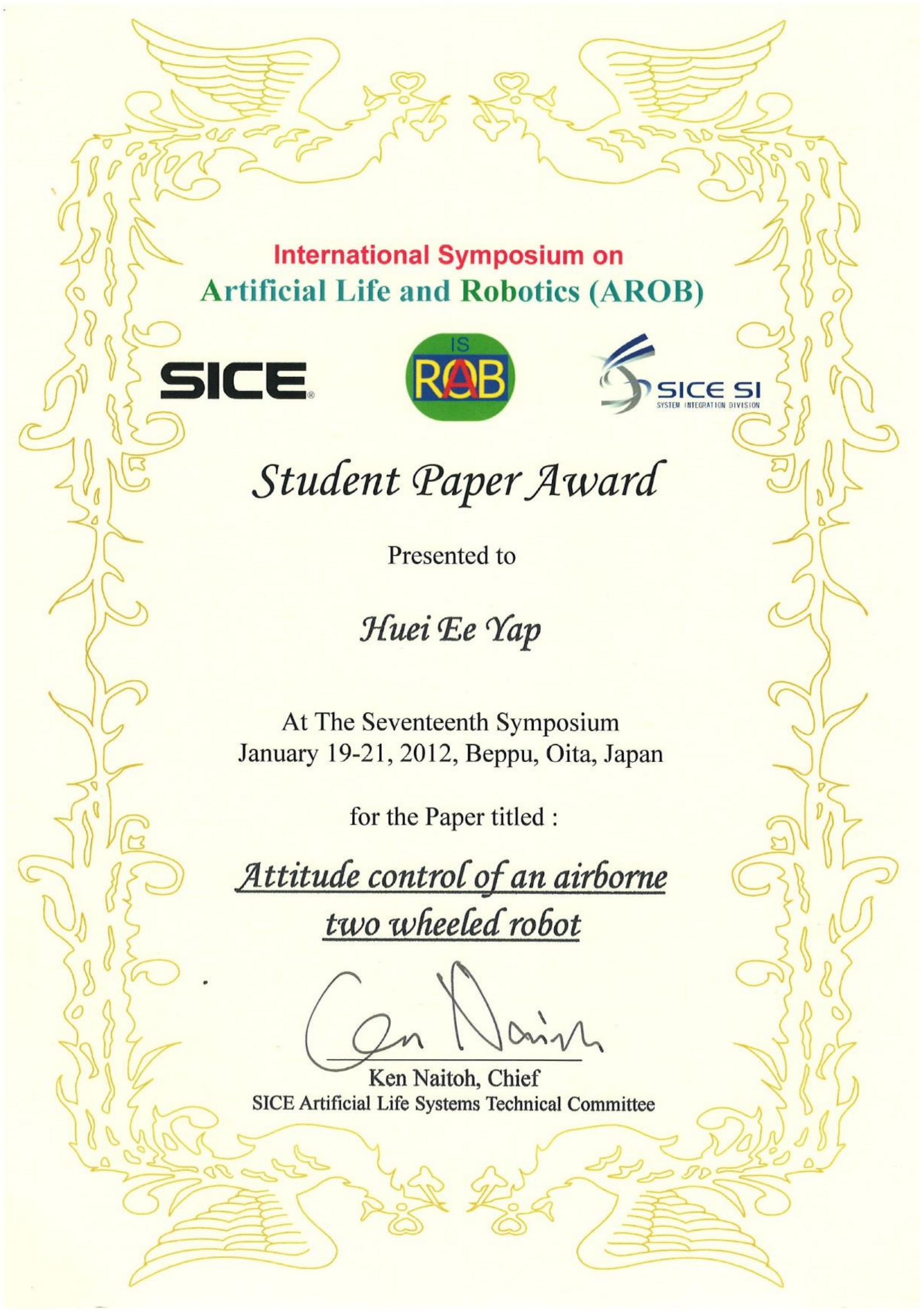
6. 佐野吉一, Enriquez Guillermo, Huei Ee Yap, 橋本周司, "口腔内インタフェースにより非調波成分を実時間制御する楽音シンセサイザー", 電子情報通信学会 2013 年総合大会講演論文集, pp.168, 2013.

Book Chapter

1. Huei Ee YAP, Shuji Hashimoto, "Design and implementation of a step traversing two-wheeled robot", Engineering Creative Design in Robotics and Mechatronics, Chapter 7, pg. 102-113, DOI: 10.4018/978-1-4666-4225-6, June 2013

Awards

Student Paper Award
Huei Ee YAP, Shuji Hashimoto, "Attitude control of an airborne two wheeled robot", Proceedings of the Seventeenth International Symposium on Artificial Life and Robotics (AROB 17th '12) January 2012.



International Symposium on
Artificial Life and Robotics (AROB)

SICE IS RAB SICE SI SYSTEM INTEGRATION DIVISION

Student Paper Award

Presented to

Huei Ee Yap

At The Seventeenth Symposium
January 19-21, 2012, Beppu, Oita, Japan

for the Paper titled :

Attitude control of an airborne two wheeled robot

[illegible]

Ken Naitoh, Chief
SICE Artificial Life Systems Technical Committee

Invited Talks and News Coverage

1. Huei Ee YAP, Shuji Hashimoto, "段差を飛び降りる 2 輪バランスロボット", 2014 年度青少年のためのロボフェスタ, November 2014

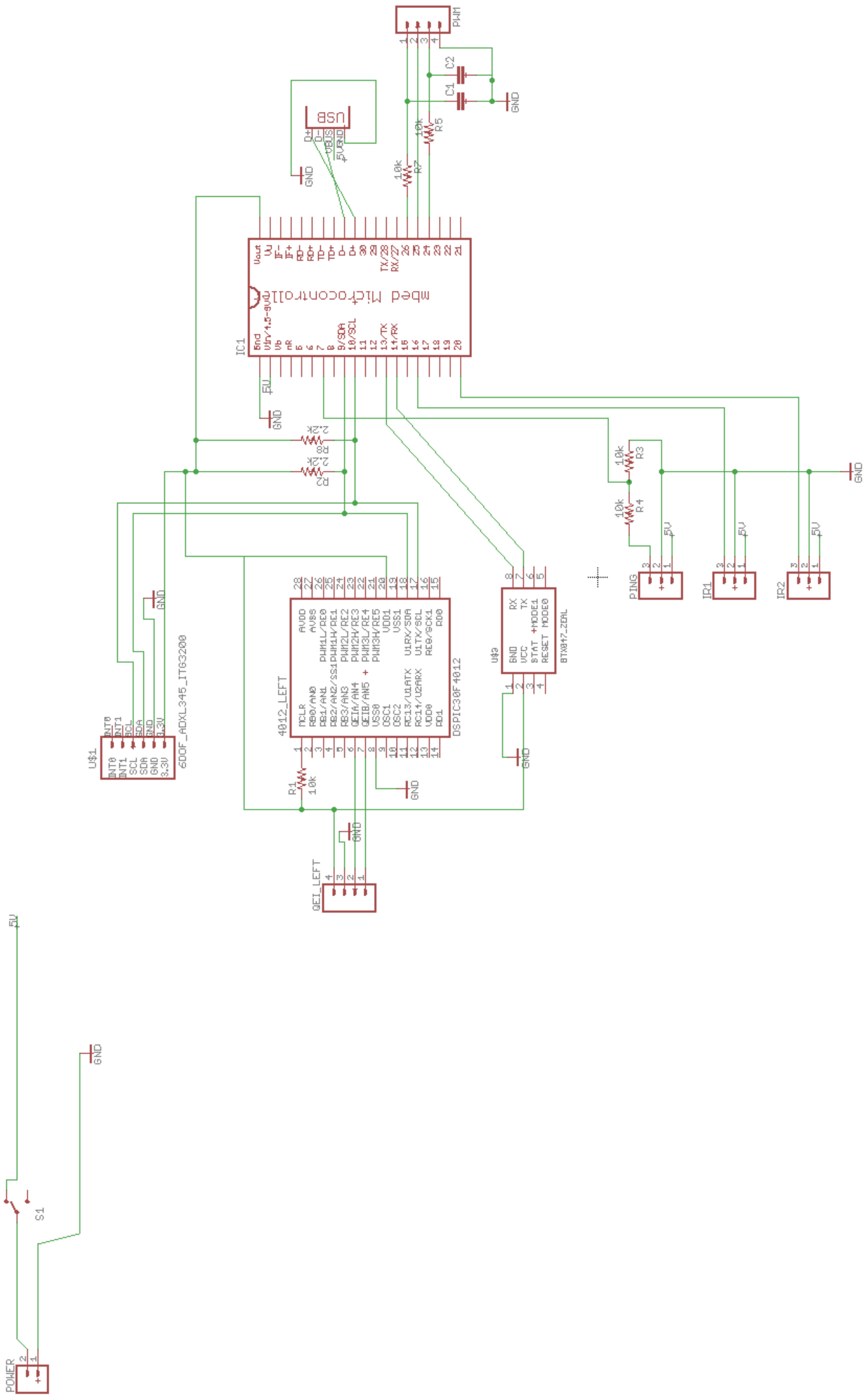
2. Huei Ee YAP, Shuji Hashimoto, "段差を飛び降りる 2 輪バランスロボット", 2013 年度青少年のためのロボフェスタ, November 2013

3. デコボコ道「まかせて」 2 輪倒立ロボを開発, 日刊工業新聞, 28th August 2012

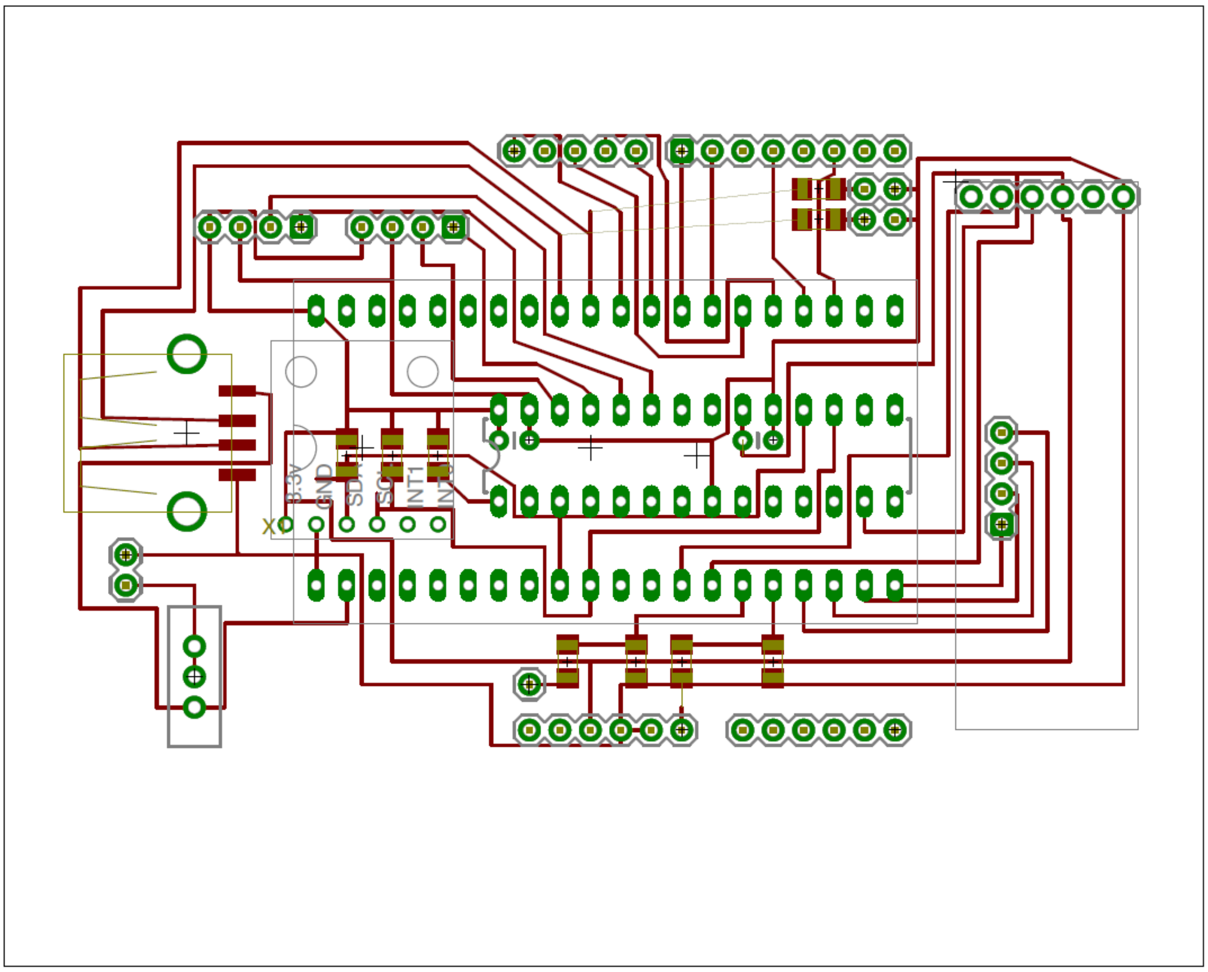
Appendix

Appendix A: Electronics

Mbed controller board circuit schematics



Mbed controller board circuit layout



Appendix B: Embedded program source code

common.h

```
const int ON = 1;
const int OFF = 0;
//Gravity at Earth's surface in m/s/s
const double g0 = 9.812865328;
//Number of samples to average.
const int SAMPLES = 4;
//Number of samples to be averaged for a null bias calculation
//during calibration.
const int CALIBRATION_SAMPLES = 128;
//ITG-3200 sensitivity is 14.375 LSB/(degrees/sec).
const double GYROSCOPE_GAIN = 1 / 13.5;
//Full scale resolution on the ADXL345 is 4mg/LSB.
const double ACCELEROMETER_GAIN  =0.004 * g0;
//Sampling accelerometer & gyroscope at 200Hz.
const double SENSOR_RATE  = 0.005;
//Updating filter at 40Hz.
const double FILTER_RATE = 0.005;
// balance rate 1ms
const double BALANCE_RATE = 0.005;

const char ENCODER_ADD_08 = 0x08; // define the I2C Address
const char ENCODER_ADD_09 = 0x09; // define the I2C Address
const int ENC_RESOLUTION = 512*2;
const double GEAR_RATIO = 6.24853;
const int ZERO_DUTY = 51;


 //toggle bit a ON and OFF;
inline void toggle(DigitalOut &a) {a=a^1;}

//Convert from radians to degrees.
inline double toDegrees(double x) {return x * 57.2957795;}

//Convert from degrees to radians.
inline double toRadians(double x) {return x * 0.01745329252;}

inline double motor(double x) {return (1.875+1.25*x/100)/3.3;} // x: PWM duty, 50 will stop motor
```

main.cpp

```
#include "mbed.h"
#include "common.h"
#include "ADXL345_I2C_6DOF.h"
#include "ITG3200_6DOF.h"
#include "IMUKalman.h"
#include "Ping.h"

#include "Wiimote.h" // Wii Remote message decoding
#include "Encoder_dspic.h"

//#define DEBUG
#define BLUETOOTH
//-- Peripheral Declaration --//
DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

Serial pc(USBTX, USBRX);
//USB Bluetooth
Serial bluetooth(p13, p14);
// IMU
ADXL345_I2C accelerometer(p9, p10);
ITG3200 gyroscope(p9, p10);
IMUKalman myfilter(0.0001, 0.0003, 0.69, FILTER_RATE);
// dspic Encoder
Encoder_dspic encoder(p9, p10, 1024, ENCODER_ADD_08);
// Ping
Ping Pinger(p7);
// Sabertooth
PwmOut Motor(p24);
DigitalOut MotorOn(p25);

typedef struct {
    float alpha; //alpha value (effects x, eg pos)
    float beta; //beta value (effects v, eg vel)
    float xk_1; //current x-estimate
    float vk_1; //current v-estimate
} AlphaBeta;

AlphaBeta ab_x;

//-- Parameters --//
const double a_Bias[3] = {22.875000, 0.484375, -20.234375 }; // x y z
const double w_Bias[3] = {6.250000, -14.562500, -24.796875 }; // x y z

//Accelerometer and gyroscope readings for x, y, z axes.
volatile double a_x;
volatile double a_y;
volatile double a_z;
volatile double w_x;
volatile double w_y;
volatile double w_z;
unsigned int current_buf = 1;

// States
enum {ONGROUND, AIRBORNE, RECOVERY, STOP};
int state;
int previousState;
bool invertBalance = false;

// Motor Control
//float gear_ratio = GEAR_RATIO;//3.8*50/30;
bool isReset = true;
bool isControlOn = true;
bool executeOnce=false;
bool executeOnceGround = false;
bool PositionControlOn = true;
double last_theta, last_theta_m;
double theta_ref = toRadians(0);
double theta_m_ref = 0;
double dtheta_m_ref = 0;
double ref_air_theta = toRadians(5);
double speed = 1800;
int duty = ZERO_DUTY;
int height = 0;

// Gains
// Without extension
double K[4] = {5.4772,    4.7691 ,115.4771  ,11.2818};
double Knp[4] = {005,3 ,120  ,12.2818};
//double Knp[4] = {0.1,3 ,120  ,10};
double Krw[4] ={ 250,   17, 0,  1};
double Kib[4] = {-0.2, -1, 400, 30};  // invert balance gains
double Kscale = 1 ;
double Kscale_rw=1;
// With extension
//double K[4] = {3,4, 131.0243,  13.1250};
//double Krw[4] ={ 150.0036,   14.4472, 0,    0.3};


Ticker readsensorsTicker;
Ticker filterTicker;

/**
 * Prototypes
 */
void sampleSensors(void);
float acc_angleX(void);
double gyro_rateX(void);
float acc_angleY(void);
double gyro_rateY(void);
void printAngle(void);
void filter(void);                //Update the filter and calculate the Euler angles.

void InitializeAlphaBeta(float x_measured, float alpha, float beta, AlphaBeta* pab);
void AlphaBetaFilter(float x_measured, float dt, AlphaBeta* pab);
void wii_data(char * data);
inline void cls();
inline void limit(double &input, const double &min, const double &max);
void Timer0_init(double sec);
void Timer1_init(double sec);
void Timer2_init(double sec);

double  dtheta_m;// debug

void reset (void) {
   theta_m_ref = 0;
   theta_ref = toRadians(0);
   // reset parameters
   last_theta = 0;
   last_theta_m = 0;
   encoder.reset();
   ab_x.xk_1 = 0;
   ab_x.vk_1 = 0;
}

void balanceControl(void) {
   double error_theta_m;
   double error_dtheta_m;
   double theta_m;//, dtheta_m;
   double theta, dtheta;
   double tmp_duty = ZERO_DUTY;

   // Update states
   previousState = state;
   theta = myfilter.getAngle();
   theta_m = (encoder.read())/GEAR_RATIO;
   AlphaBetaFilter(theta_m, BALANCE_RATE, &ab_x);
   height = Pinger.Read_cm();

   if ( ( fabs(theta) > toRadians(70) ) || !isControlOn){
      led2 = led3 = led4 = OFF;
```

```
    state = STOP;
    isReset = true;
  } else {
 //   if (height > 17){
 //       executeOnceGround = true;
 //   }
    // On ground
    if ( height < 11 ){//|| PositionControlOn  ) {
   //     if (executeOnceGround)
   //         executeOnce = true;

        if ( (fabs(theta) < toRadians(10)) && isReset   ) {
            // Recovery range, unset reset flag
            isReset = false;
        } else if ( (fabs(theta) < toRadians(70)) && !isReset   ) {
            // Balance mode
            led2 = led3 = led4 = OFF;
            state = ONGROUND;
            led2 = ON;
        } else {
            // Recovery mode
            led3 = ON;
            state = STOP;
        }
    } else {
        // Airborne
        if (!isReset){
            led2 = led3 = led4 = OFF;
            led4 = ON;
            state = AIRBORNE;
            PositionControlOn = false;
        } else {

            // Recovery mode
            led3 = ON;
            state = STOP;
        }

    }
  }
  switch (state) {
    case ONGROUND:
        MotorOn = ON;
        if (false ){//executeOnceGround){
            tmp_duty = ZERO_DUTY;
        } else {
            tmp_duty = 0;
            theta_m_ref += dtheta_m_ref * BALANCE_RATE;
            error_theta_m = theta_m_ref - ab_x.xk_1;
            limit (error_theta_m, -toRadians(270), toRadians(270));
            error_dtheta_m =  - ab_x.vk_1;
            theta = myfilter.getAngle();
            if (PositionControlOn){// && !executeOnceGround){
            // Theta Motor
                tmp_duty = K[0] * (error_theta_m);                 // P  theta_m

                //limit (error_dtheta_m, -toRadians(720), toRadians(720));
                tmp_duty = tmp_duty + K[1] * (error_dtheta_m);            // D  theta_m

                // Theta
                tmp_duty = tmp_duty + K[2] * ( 0 - theta);  // P  theta
                tmp_duty = tmp_duty - K[3] * myfilter.getdAngle();  // D  theta
             } else {
            // Theta Motor
                tmp_duty = Knp[0] * (error_theta_m);                 // P  theta_m

                tmp_duty = tmp_duty + Knp[1] * (error_dtheta_m);            // D  theta_m
                // Theta
                tmp_duty = tmp_duty + Knp[2] * ( theta_ref - theta);  // P  theta
                tmp_duty = tmp_duty - Knp[3] * myfilter.getdAngle();  // D  theta
             }
            tmp_duty = ZERO_DUTY + Kscale * tmp_duty;
            limit(tmp_duty, 1, 99);
            }
            break;
        case AIRBORNE:

            MotorOn =  ON;
        //   if (executeOnce){
        //      tmp_duty = ZERO_DUTY;
        //    } else {
                // Theta
                theta = myfilter.getAngle();
                tmp_duty =  Krw[0] * (ref_air_theta - theta);            // P  theta
                tmp_duty = tmp_duty - Krw[1] * myfilter.getdAngle();    // D  theta

                // Theta Motor
                theta_m = (encoder.read())/GEAR_RATIO;
                //tmp_duty = tmp_duty - Krw[2] *  ab_x.xk_1;
                tmp_duty = tmp_duty - Krw[3] * ab_x.vk_1;       // D  theta_m

                tmp_duty = ZERO_DUTY + Kscale_rw * tmp_duty;

                limit(tmp_duty, 1, 99);
         //   }
            break;
        case STOP:
            MotorOn =  OFF;
            reset();
            break;
        default:
            MotorOn =  OFF;
            reset();
    }
    duty = (int)tmp_duty;
    Motor = motor(duty);
}

void invertBalanceControl(void) {
    double error_theta_m;
    double error_dtheta_m;
    double theta_m;//, dtheta_m;
    double theta;
    double tmp_duty = ZERO_DUTY;

    // Update states
    previousState = state;
    theta = myfilter.getAngle();
    theta_m = (encoder.read())/GEAR_RATIO;
    AlphaBetaFilter(theta_m, BALANCE_RATE, &ab_x);

    if ( ( fabs(theta) > toRadians(45) ) || !isControlOn){
        led2 = led3 = led4 = OFF;
        state = STOP;
        isReset = true;
    } else {
        if ( (fabs(theta) < toRadians(10)) && isReset   ) {
            // Recovery range, unset reset flag
            isReset = false;
        } else if ( (fabs(theta) < toRadians(70)) && !isReset   ) {
            // Balance mode
            led2 = led3 = led4 = OFF;
            state = ONGROUND;
            led2 = ON;
        } else {
            // Recovery mode
            led3 = ON;
            state = STOP;
        }
    }
    switch (state) {
        case ONGROUND:
            MotorOn = ON;
            tmp_duty = 0;
            error_theta_m = - ab_x.xk_1;
            error_dtheta_m =  - ab_x.vk_1;
            theta = myfilter.getAngle();

            tmp_duty = Kib[0] * (error_theta_m);                    // P  theta_m

            //limit (error_dtheta_m, -toRadians(720), toRadians(720));
            tmp_duty = tmp_duty + Kib[1] * (error_dtheta_m);             // D  theta_m

            // Theta
            tmp_duty = tmp_duty + Kib[2] * ( theta_ref - theta);  // P  theta
            tmp_duty = tmp_duty - Kib[3] * myfilter.getdAngle();  // D  theta

            tmp_duty = ZERO_DUTY - Kscale * tmp_duty;
            limit(tmp_duty, 1, 99);
            break;

        case STOP:
            MotorOn =  OFF;
            reset();
            break;
        default:
            MotorOn =  OFF;
            reset();
    }
    duty = (int)tmp_duty;
    Motor = motor(duty);
}

void startWiiCom(const char * wiiMAC);
int main() {
    //-- Initialization --//
    state = ONGROUND;
    previousState = ONGROUND;
    //-- Detect is invert mode --//
    if (accelerometer.getAz() >60000){
        theta_ref = toRadians(-12.5);
        invertBalance = true;
    }

    // Serial COM
#ifdef BLUETOOTH
    bluetooth.baud(115200);
    bluetooth.printf("Starting IMU filter test...\r\n");
#endif
#ifdef DEBUG
    pc.baud(115200);
    pc.printf("Freefall pendulum...\r\n");
#endif
    // Encoder
    encoder.reset();

    InitializeAlphaBeta(0,0.65,0.12,&ab_x); //x position
    //Calibrate IMU sensors.
    //accelerometer.calibrate(a_Bias, CALIBRATION_SAMPLES);
    //gyroscope.calibrate(w_Bias, CALIBRATION_SAMPLES);

    // Sabertooth
    Motor.period_us(30);
    Motor = motor(ZERO_DUTY);

    //Set up timers.
    readsensorsTicker.attach(&sampleSensors, SENSOR_RATE);
    filterTicker.attach(&filter, FILTER_RATE);
    Timer0_init(BALANCE_RATE);
    Timer1_init(0.05); //data printing
    Timer2_init(0.01); // Pinger

    //-- End Initialization --//
    wait(0.5);

    if (invertBalance){
        while(1){}
    }else{
        // Wiimote
        char controller02[] = "00:1E:A9:71:B3:60";    // S.NAKAMURA 02
       // char controller01[] = "00:1F:C5:4C:29:99";   // S.NAKAMURA 01
        startWiiCom(controller02);
    }
}

extern "C" void TIMER0_IRQHandler (void) {  // Balance Control
    if ((LPC_TIM0->IR & 0x01) == 0x01) { // if MR0 interrupt, proceed
        LPC_TIM0->IR |= 1 << 0;          // Clear MR0 interrupt flag
        if (invertBalance){
            invertBalanceControl();
        } else {
            balanceControl();
        }
    }
}

extern "C" void TIMER1_IRQHandler (void) {  // Print data
    if ((LPC_TIM1->IR & 0x01) == 0x01) { // if MR0 interrupt, proceed
        LPC_TIM1->IR |= 1 << 0;          // Clear MR0 interrupt flag

        toggle(led1);
        //cls();

#ifdef BLUETOOTH
        bluetooth.printf("%+f, %+f, %+f, %+f, %+4d, %+4d\r\n", toDegrees(ab_x.xk_1), toDegrees(ab_x.vk_1),
toDegrees(myfilter.getAngle()), toDegrees(myfilter.getdAngle()), height, duty);
        //bluetooth.printf("%+f,%+f,%+4d\r\n",  toDegrees(encoder.read())/gear_ratio,toDegrees(myfilter.getAngle()), Pinger.Read_cm());
#endif
#ifdef DEBUG
        //pc.printf("%+f\r\n", toDegrees( encoder.read() ) );
        pc.printf("%+f, %+f, %+f, %+f, %+4d, %+4d\r\n", toDegrees(ab_x.xk_1), toDegrees(ab_x.vk_1),  toDegrees(myfilter.getAngle()),
toDegrees(myfilter.getdAngle()), height, duty);
#endif
        //pc.printf("%+f\n", toDegrees(myfilter.getAngle()) );
        //pc.printf("Enc: %f\r\n", toDegrees(encoder_left.read())) ;
        //bluetooth.printf("%+f,%+f,%+f\r\n", toDegrees(0),toDegrees(myfilter.getAngle()),toDegrees(gyro_rateY()));
    }
}

extern "C" void TIMER2_IRQHandler (void) {  // Pinger
    if ((LPC_TIM2->IR & 0x01) == 0x01) { // if MR0 interrupt, proceed
        LPC_TIM2->IR |= 1 << 0;          // Clear MR0 interrupt flag
        Pinger.Send();
    }
}

void Timer0_init(double sec) {
    LPC_SC->PCONP |=1<<1;            //timer0 power on
    LPC_TIM0->MR0 = 24000000*sec-1;  // 24e6 ticks/sec
    LPC_TIM0->MCR = 3;              //interrupt and reset control
    //3 = Interrupt & reset timer0 on match
    //1 = Interrupt only, no reset of timer0
    NVIC_EnableIRQ(TIMER0_IRQn);    //enable timer0 interrupt
    LPC_TIM0->TCR = 1;              //enable Timer0
}

void Timer1_init(double sec) {
    LPC_SC->PCONP |=1<<2;            //timer1 power on
    LPC_TIM1->MR0 = 24000000*sec-1;  // 24e6 ticks/sec
    LPC_TIM1->MCR = 3;              //interrupt and reset control
    //3 = Interrupt & reset timer1 on match
    //1 = Interrupt only, no reset of timer1
    NVIC_EnableIRQ(TIMER1_IRQn);    //enable timer1 interrupt
    LPC_TIM1->TCR = 1;              //enable Timer1
}

void Timer2_init(double sec) {
    LPC_SC->PCONP |=1<<22;           //timer2 power on(enable), timer 2 off by default, see table 46 of UM10360 LPC17xx user manual
    LPC_TIM2->MR0 = 24000000*sec-1;  // 24e6 ticks/sec
    LPC_TIM2->MCR = 3;              //interrupt and reset control
    //3 = Interrupt & reset timer0 on match
    //1 = Interrupt only, no reset of timer0
    NVIC_EnableIRQ(TIMER2_IRQn);    //enable timer0 interrupt
    LPC_TIM2->TCR = 1;              //enable Timer0
}

void sampleSensors(void) {
    // 4th order runge-kutta filter
    // dt ~ 0.00065
    unsigned int i;
    int readings[3]; //Buffer for accelerometer readings.
    static int sensor_buf[5];
    static int sensor_filter_buffer[4][5] = { {0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0},{0,0,0,0,0} };
    accelerometer.getOutput(readings);

    sensor_filter_buffer[current_buf][0] = (int16_t) readings[0];  //accX
    sensor_filter_buffer[current_buf][1] = (int16_t) readings[1];  //accY
    sensor_filter_buffer[current_buf][2] = (int16_t) readings[2];  //accZ

    sensor_filter_buffer[current_buf][3] = gyroscope.getGyroX();   //gyroX
    sensor_filter_buffer[current_buf][4] = gyroscope.getGyroY();   //gyroY

    for (i=0; i < 5; i++) {
        sensor_buf[i] = (sensor_filter_buffer[current_buf][i]/6 +
                    sensor_filter_buffer[(current_buf+1)%4][i]/6 +
                    sensor_filter_buffer[(current_buf+2)%4][i]/3 +
                    sensor_filter_buffer[(current_buf+3)%4][i]/3);
    }
    current_buf = (current_buf+1) % 4;

    a_x = (sensor_buf[0] - a_Bias[0]) * ACCELEROMETER_GAIN;
    a_y = (sensor_buf[1] - a_Bias[1]) * ACCELEROMETER_GAIN;
    a_z = (sensor_buf[2] - a_Bias[2]) * ACCELEROMETER_GAIN;

    w_x = toRadians((sensor_buf[3] - w_Bias[0]) * GYROSCOPE_GAIN);
    w_y = toRadians((sensor_buf[4] - w_Bias[1]) * GYROSCOPE_GAIN);
    w_z = 0;
}

float acc_angleX() {
    return  atan2(a_y,a_z);//-(atan2(-a_z, a_y)-(3.14159/2.0));
}

double gyro_rateX() {
    //return -w_y;
    return w_x; //6dof around x axis
}
float acc_angleY() {
    return  atan2(a_x,a_z);//-(atan2(-a_z, a_y)-(3.14159/2.0));
}

double gyro_rateY() {
//    return -w_x;
    return -w_y; // 6dof around x axis
}

float acc_angleY_invert()
{
    //return  atan2(a_x,-a_z); // facing upward
    return  atan2(a_x,-a_z);   // facing downward -(atan2(-a_z, a_y)-(3.14159/2.0));
}

double gyro_rateY_invert()
{
//    return -w_x;
//  return -w_y; // facing upward 6dof around x axis
    return w_y; //  facing downward
}

void printAngle(void) {

    pc.printf("%f,%f,%f\n", toDegrees(myfilter.getAngle()), acc_angleY(),gyro_rateY());
}

void filter(void) {
    // execute kalman filter pitch
    // dt = 0.00002
    if (invertBalance){
        myfilter.updateFilter(acc_angleY_invert(), gyro_rateY_invert());
    } else {
        myfilter.updateFilter(acc_angleY(), gyro_rateY());
    }
}

void InitializeAlphaBeta(float x_measured, float alpha, float beta, AlphaBeta* pab) {
    pab->xk_1 = x_measured;
    pab->vk_1 = 0;
    pab->alpha = alpha;
    pab->beta = beta;
}

void AlphaBetaFilter(float x_measured, float dt, AlphaBeta* pab) {
    float xk_1 = pab->xk_1;
    float vk_1 = pab->vk_1;
    float alpha = pab->alpha;
    float beta = pab->beta;

    float xk; //current system state (ie: position)
    float vk; //derivative of system state (ie: velocity)
    float rk; //residual error

    //update our (estimated) state 'x' from the system (ie pos = pos + vel (last).dt)
    xk = xk_1 + dt * vk_1;
    //update (estimated) velocity
    vk = vk_1;
    //what is our residual error (mesured - estimated)
    rk = x_measured - xk;
    //update our estimates given the residual error.
    xk = xk + alpha * rk;
    vk = vk + beta/dt * rk;
    //finished!

    //now all our "currents" become our "olds" for next time
    pab->vk_1 = vk;
    pab->xk_1 = xk;
}

Wiimote wii;
// this is called by the USB infrastructure when a wii message comes in
void wii_interrupt() {

    // temporary action triggers
    if (wii.up)   {
        theta_m_ref = theta_m_ref + toRadians(20);
    }
    if (wii.down)   {
        theta_m_ref = theta_m_ref - toRadians(20);
    }
    if (wii.left) {
        dtheta_m_ref = toRadians(speed);
    } else if (wii.right) {
        dtheta_m_ref = toRadians(-speed);
    } else {
        dtheta_m_ref = 0;
    }

    if (wii.home) {
      //  led3 = ON;
        theta_m_ref = 0;
        theta_ref = toRadians(0);

        encoder.reset();
        ab_x.xk_1 = 0;
        ab_x.vk_1 = 0;
     //   executeOnce=false;
    //    executeOnceGround=false;
        PositionControlOn = true;
    }
    if (wii.a) {
     //   executeOnceGround=false;
        PositionControlOn = false;
    }
    if (wii.b) {
      //  led3 = ON;
        PositionControlOn = true;
        theta_m_ref = toRadians(1.5*360);

    }
    if (wii.plus){
        isControlOn = true;
    }

    if (wii.minus){
        isControlOn = false;
    }
    if (wii.one){
        speed += 360;
        limit(speed, 360, 2520);
    }
    if (wii.two) {
        speed -= 360;
        limit(speed, 360, 2520);
    }
}

inline void limit(double &input, const double &min, const double &max)
{
    if (input > max) input = max; // duty limit
    if(input < min) input = min;
}

inline void cls() {
    pc.putc(27);   //Print "esc"
    pc.printf("[2J");
}
```

IMUKalman.h

```
#ifndef IMU_KALMAN_H
#define IMU_KALMAN_H

#include "mbed.h"

class IMUKalman {

public:
    /**
     * Constructor.
     *
     * @param
     */
     // Constructor
   IMUKalman(float Q_angle, float Q_gyro, float R_angle, float rate);

   // input angle and dot_angle have to be same signed,i.e. both clocked wise positive/negative
   void updateFilter(float angle, float dot_angle);

   float getAngle(void);
   float getdAngle(void);
private:
   struct Gyro1DKalman
   {
      /* These variables represent our state matrix x */
      float x_angle,
            x_bias;
      /* Our error covariance matrix */
      float P_00,
            P_01,
            P_10,
            P_11;

      /*
       * Q is a 2x2 matrix of the covariance. Because we
       * assuma the gyro and accelero noise to be independend
       * of eachother, the covariances on the / diagonal are 0.
       *
       * Covariance Q, the process noise, from the assumption
       *    x = F x + B u + w
       * with w having a normal distribution with covariance Q.
       * (covariance = E[ (X - E[X])*(X - E[X])' ]
       * We assume is linair with dt
       */
      float Q_angle, Q_gyro;
      /*
       * Covariance R, our observation noise (from the accelerometer)
       * Also assumed to be linair with dt
       */
      float R_angle;
   };

   struct Gyro1DKalman filter;
   float _filterRate;
   float _angle;
   float _dangle;

   // Kalman predict
   void ars_predict(float gyro);

   // Kalman update
   float ars_update(float angle_m);

};
#endif /* IMU_KALMAN_H */
```

IMUKalman.cpp

```
#include "IMUKalman.h"
#include <math.h>

IMUKalman::IMUKalman(float Q_angle, float Q_gyro, float R_angle, float rate)
{
    filter.Q_angle = Q_angle;
    filter.Q_gyro  = Q_gyro;
    filter.R_angle = R_angle;

    filter.x_angle =0.0;
    filter.P_00 =0.0;
    filter.P_01 =0.0;
    filter.P_10 =0.0;
    filter.P_11 =0.0;

    _filterRate = rate;
}

void IMUKalman::updateFilter(float angle_m, float dotAngle)
{
   // ars_predict(dot_angle);    // Kalman predict         (float(yrate)-w_yBias ) *GYROSCOPE_GAIN;
   // _angle = ars_update(angle);
    _dangle = dotAngle - filter.x_bias;
    filter.x_angle += _filterRate * (dotAngle - filter.x_bias);
    filter.P_00 +=  - _filterRate * (filter.P_10 + filter.P_01) + filter.Q_angle * _filterRate;
    filter.P_01 +=  - _filterRate * filter.P_11;
    filter.P_10 +=  - _filterRate * filter.P_11;
    filter.P_11 +=  + filter.Q_gyro * _filterRate;

    float y = angle_m - filter.x_angle;
    float S = filter.P_00 + filter.R_angle;
    float K_0 = filter.P_00 / S;
    float K_1 = filter.P_10 / S;

    filter.x_angle +=  K_0 * y;
    filter.x_bias  +=  K_1 * y;

    filter.P_00 -= K_0 * filter.P_00;
    filter.P_01 -= K_0 * filter.P_01;
    filter.P_10 -= K_1 * filter.P_00;
    filter.P_11 -= K_1 * filter.P_01;

    _angle = filter.x_angle;
}

float IMUKalman::getAngle(void)
{
   return _angle;
}
float IMUKalman::getdAngle(void)
{
   return _dangle;
}


/*
 * The predict function. Updates 2 variables:
 * our model-state x and the 2x2 matrix P
 *
 * x = [ angle, bias ]'
 *
 *   = F x + B u
 *
 *   = [ 1 -dt, 0 1 ] [ angle, bias ] + [ dt, 0 ] [ dotAngle 0 ]
 *
 *   => angle = angle + dt (dotAngle - bias)
 *      bias  = bias
 *
 *
 * P = F P transpose(F) + Q
 *
 *   = [ 1 -dt, 0 1 ] * P * [ 1 0, -dt 1 ] + Q
 *
 *  P(0,0) = P(0,0) - dt * ( P(1,0) + P(0,1) ) + dt&#65394; * P(1,1) + Q(0,0)
 *  P(0,1) = P(0,1) - dt * P(1,1) + Q(0,1)
 *  P(1,0) = P(1,0) - dt * P(1,1) + Q(1,0)
 *  P(1,1) = P(1,1) + Q(1,1)
 *
 *
 */
void IMUKalman::ars_predict(float dotAngle)
{
    filter.x_angle += _filterRate * (dotAngle - filter.x_bias);
    filter.P_00 +=  - _filterRate * (filter.P_10 + filter.P_01) + filter.Q_angle * _filterRate;
    filter.P_01 +=  - _filterRate * filter.P_11;
    filter.P_10 +=  - _filterRate * filter.P_11;
    filter.P_11 +=  + filter.Q_gyro * _filterRate;
}

/*
 *  The update function updates our model using
 *  the information from a 2nd measurement.
 *  Input angle_m is the angle measured by the accelerometer.
 *
 *  y = z - H x
 *
 *  S = H P transpose(H) + R
 *    = [ 1 0 ] P [ 1, 0 ] + R
 *    = P(0,0) + R
 *
 *  K = P transpose(H) S^-1
 *    = [ P(0,0), P(1,0) ] / S
 *
 *  x = x + K y
 *
 *  P = (I - K H) P
 *
 *    = ( [ 1 0,    [ K(0),
 *          0 1 ] -   K(1) ] * [ 1 0 ] ) P
 *
 *    = [ P(0,0)-P(0,0)*K(0)  P(0,1)-P(0,1)*K(0),
 *        P(1,0)-P(0,0)*K(1)  P(1,1)-P(0,1)*K(1) ]
 */
float IMUKalman::ars_update(float angle_m)
{
    float y = angle_m - filter.x_angle;
    float S = filter.P_00 + filter.R_angle;
    float K_0 = filter.P_00 / S;
    float K_1 = filter.P_10 / S;

    filter.x_angle +=  K_0 * y;
    filter.x_bias  +=  K_1 * y;

    filter.P_00 -= K_0 * filter.P_00;
    filter.P_01 -= K_0 * filter.P_01;
    filter.P_10 -= K_1 * filter.P_00;
    filter.P_11 -= K_1 * filter.P_01;

    return filter.x_angle;
}
```