THE UNIVERSITY OF ADELAIDE
FACULTY OF ENGINEERING, COMPUTER & MATHEMATICAL SCIENCES
SCHOOL OF MECHANICAL ENGINEERING
MECH ENG 4135: MECHATRONICS HONOURS PROJECT


# 899: DESIGN AND BUILD A BALLBOT

## Final Report

**AUTHORS:**

Justin FONG
Simon UPPILL

**SUPERVISOR:**

Ben CAZZOLATO

October 29, 2009

## Executive Summary

A ballbot is a robot which achieves dynamic stability to remain upright and balanced on a ball. This involves using control theory to actuate the ball and balance, rather than relying on gravity and a large wheel base like traditional robots, which rely on static stability. Only two ballbots had been constructed at the commencement of this project; the first by the Robotics Institute at Carnegie Mellon University (CMU) in 2005, and the second by Tohoku Gakuin University (TGU) in 2008. During the course of the project, the NXT Ballbot has also been created, by Yorihisa Yamamoto.

This project produced two Ballbots; a small scale version constructed from a Lego Mindstorms Kit, and a larger ballbot with dimensions similar to that of a human. The Ballbots were to be used as promotional and teaching tools at the University of Adelaide. Development of the two ballbots required derivation of the dynamics, design and construction of the ballbot hardware, and design and implementation of controllers to stabilise the ballbots.

Derivation of the equations of motion of a generic ballbot system was performed using the Lagrangian approach on a simplified ballbot model. The resulting dynamics of the simplified model were expressed in linearised state space form, and used to develop a generalised state space controller. For each ballbot, optimal control gains were generated using the Linear Quadratic Regulator method.

The Lego Ballbot was constructed using a Lego NXT Mindstorms kit and other available Lego parts in an iterative prototyping process, and the generalised controller implemented in Simulink utilising the ECRobot toolkit. This controller successfully balances the Lego ballbot, with the ability to reject small disturbances. In addition, command tracking has been implemented, allowing for remote control of the Lego Ballbot system. Furthermore, yaw control of the Lego Ballbot has been achieved with a secondary PID controller. To increase the potential of this ballbot as a teaching aide, further work included the production of graphical assembly instructions allowing for replication of the Lego Ballbot system and the creation of a virtual reality model of the Lego Ballbot system.

The Full Scale Ballbot was designed and constructed on a strict budget and time frame. The initial design used the Lego Mindstorms microprocessor and sensors, however, it was not successful in balancing primarily due to issues with the estimation of the body angle. The design was thus revised, encorporting an Inertial Measurement Unit and a new microprocessor. This revised Full Scale Ballbot was capable of balancing, however, moves around its mean point in a large limit cycle. It is thought that this is due to large degrees of non-linearities in the motors, which cannot be accounted for in a state space controller. The Full Scale Ballbot can also reject small disturbances, such as a small push, and is capable of command tracking at slow speeds, using commands which have been pre-programmed into the microprocessor.

The 2009 Ballbot Project successfully produced two ballots; a small scale Lego Ballbot, and a Full Scale Ballbot. Both ballbots are capable of balancing, a small amount of disturbance rejection and command tracking.

## Acknowledgements

The authors would like to acknowledge that this project would not have been possible without the help and support of many people. As such, we would like to thank the following for their contribution to this project.

Associate Professor Dr Ben Cazzolato, our project supervisor It is without a doubt that this project would not have been as successful or enjoyable without your guidance, time and support.

The School of Mechanical Engineering workshop, in particular Phil Schmidt, Silvio De Ieso and Steve Kloeden for their work in constructing (and fixing) the Full Scale Ballbot throughout the year.

The 2009 University of Adelaide Open Day Creativity and Innovation Fund for funding, which allowed us to create the Full Scale Ballbot.

Ralph Hollis and Yorihisa Yamamoto for their time and information which were so freely given through our correspondence.

And finally our family and friends for their support not just for this project, but throughout our education to this point.

## Disclaimer

The content of this report is entirely the work of the following students from the University of Adelaide. Any content obtained from other sources has been referenced accordingly.

Justin FONG
Date:

Simon UPPILL
Date:

# Contents

**1 Introduction** **1**
1.1 Motivation . . . 1
1.2 Background . . . 2
1.3 Scope and Objectives . . . 2
1.4 Approach . . . 4
1.5 Project Constraints . . . 4

**2 Literature Review** **5**
2.1 Equations of Motion . . . 5
2.1.1 Assumptions . . . 5
2.1.2 Derivation Method . . . 5
2.1.3 Existing Results . . . 6
2.2 Hardware Design . . . 7
2.2.1 Drive Mechanism . . . 7
2.2.2 Ball . . . 9
2.2.3 Tilt Sensors . . . 10
2.2.4 Controller Hardware . . . 10
2.3 Theoretical Controller . . . 11
2.3.1 Carnigie Mellon University Controller . . . 11
2.3.2 Tohoku Gakuin University Controller . . . 12
2.3.3 Sliding-Mode Control . . . 12
2.3.4 NXTway-GS and NXT Ballbot Controllers . . . 12

**3 Theoretical Controller** **14**
3.1 State Space Control . . . 14
3.2 Linear Quadratic Regulator . . . 14
3.3 State Space Model of a Ballbot . . . 15
3.3.1 Assumptions and Definitions . . . 15
3.3.2 Derivation of Equations of Motion . . . 16
3.3.3 Linear State Space Model . . . 17

**4 The Lego Ballbot** **19**
4.1 Design . . . 19
4.1.1 Lego NXT Mindstorms Range . . . 19
4.1.2 Conceptual design . . . 20
4.1.3 Construction of the Lego Ballbot . . . 23
4.2 Controller . . . 25
4.2.1 Controller Implementation . . . 26
4.2.2 Simulation . . . 30
4.3 Testing Procedure . . . 32
4.4 Results and Analysis . . . 33
4.4.1 Balancing from Release . . . 33
4.4.2 Disturbance Rejection . . . 34
4.4.3 Command Tracking and Remote Control . . . 35
4.5 Yaw Control . . . 35
4.5.1 Actuator . . . 36

4.5.2 Controller . . . 37
4.5.3 Results . . . 37

**5 The Full Scale Ballbot 39**
5.1 Hardware Design . . . 39
5.1.1 Conceptual Design . . . 39
5.1.2 Component Specification and Selection . . . 40
5.1.3 Structural Design . . . 43
5.1.4 Electrical Design . . . 46
5.1.5 Completed System . . . 47
5.2 Initial Controller Implementation . . . 48
5.2.1 Modifications to Lego Controller . . . 48
5.2.2 Calculation of Controller Gains . . . 49
5.3 Testing Procedure . . . 49
5.3.1 Safety and Physical Procedure . . . 49
5.3.2 Tuning Parameters . . . 50
5.4 Initial Design Performance and Modifications . . . 51
5.4.1 State Estimation Error . . . 51
5.4.2 Mechanical issues . . . 52
5.4.3 Modifications: Kalman Filter Implementation . . . 52
5.5 Revised Design . . . 53
5.5.1 Inertial Measurement Unit . . . 53
5.5.2 Dragonboard . . . 53
5.5.3 Controller Programming . . . 54
5.6 Revised Design Results and Analysis . . . 55
5.6.1 Balancing . . . 56
5.6.2 Command Tracking . . . 56

**6 Resource Analysis 58**
6.1 Component Costs . . . 58
6.1.1 Lego Ballbot . . . 58
6.1.2 Full Scale Ballbot . . . 58
6.2 Labour Costs . . . 59
6.2.1 Student Hours . . . 59
6.2.2 Workshop Hours . . . 60

**7 Project Outcomes and Future Work 61**
7.1 Project Outcomes . . . 61
7.2 Future Work . . . 62
7.2.1 General Ballbot Concept . . . 62
7.2.2 Lego Ballbot . . . 62
7.2.3 Full Scale Ballbot . . . 62

**8 Conclusion 65**

**9 References 66**

**A Derivation of Dynamics 67**

**B Matlab Code** **71**

**C NXT Mindstorms Component Specifications** **80**

**D Lego Ballbot Building Instructions** **82**

**E Lego Ballbot Simulink Controller** **100**

**F Virtual Reality Model Simulink Program** **141**

**G Component Datasheets** **163**

**H Full Scale Ballbot Drawings** **172**

**I Electrical Schematics** **185**

**J Full Scale Ballbot Code** **188**

**K Risk Assessment** **206**

**L Safe Operating Procedure** **216**

# List of Figures


1 A Segway (Segway Inc, 2009) . . . 2
2 Existing Ballbots . . . 3
3 Generalised coordinates for ball/wheel balancing systems . . . 6
4 Driving Mechanism of CMU Ballbot (Lauwers et al., 2006) . . . 8
5 Driving Mechanism of TGU Ballbot (Kumagai and Ochiai, 2008) . . . 8
6 A Proposed Drive Mechanism using Four Omniwheels (Wu and Hwang, 2008) . . . 9
7 Structure of the Controller used on the CMU Ballbot (Lauwers et al., 2006) . . . 11
8 Structure of the Controller used in the NXTway-GS (Yamamoto, 2008) . . . 13
9 State Space Control Block Diagram . . . 15
10 The Simplified Ballbot Model . . . 16
11 Drive mechanism concepts . . . 20
12 Layout of the Lego Ballbot . . . 22
13 Lego Ballbot, Original . . . 23
14 Lego Ballbot, Second Iteration . . . 24
15 Lego Ballbot, Third Iteration . . . 25
16 Final Lego Ballbot, showing coordinate system and states . . . 25
17 Finite State Machine Implementation . . . 26
18 Ballbot State Space Controller . . . 27
19 Measurement and Calculation of States, One Plane . . . 28
20 Motor Signal Generation - One Plane . . . 30
21 Controller Simulation Results - Balancing . . . 31
22 Controller Simulation Results - Command Tracking . . . 31
23 Virtual Reality Model . . . 32
24 Lego Ballbot - Balancing . . . 34
25 Lego Ballbot - Disturbance Rejection . . . 35
26 Lego Ballbot - Command Tracking . . . 36
27 Modified Lego Ballbot, with rotor . . . 37
28 PID controller structure . . . 37
29 Command Tracking, with Yaw Controller . . . 38
30 Concepts for the Full Scale Drive Mechanism Frame (shown in one plane) . . . 40
31 Omniwheel Candidates . . . 43
32 Full Scale Ballbot Frame . . . 44
33 Components designed for Motor Bracket . . . 45
34 Motor Bracket Complete Assembly . . . 45
35 The Motor Controller Boards used on the Full Scale Ballbot . . . 46
36 Power Distribution and Signal Routing, as designed . . . 47
37 The Constructed Full Scale Ballbot . . . 48
38 Generation of the $\theta$ Estimation using a Complementary Filter . . . 49
39 The Testing Area Used for the Full Scale Ballbot . . . 50
40 Performance of the Complementary Filter . . . 51
41 Structure of a Kalman Filter, adapted from Zarchan (2005) . . . 53
42 Revised Hardware Components . . . 54
43 Required PWM Signal . . . 55
44 Balancing Performance of the Full Scale Ballbot . . . 57
45 Command Tracking Performance of the Full Scale Ballbot . . . 57

# List of Tables

1 Decision Matrix for Lego Ballbot Drive Mechanism . . . . . 21
2 Lego Ballbot Physical Parameters . . . . . 26
3 Full Scale Ballbot Physical Parameters . . . . . 48
4 Component Costs for the Lego Ballbot . . . . . 58
5 Component Costs for the Full Scale System . . . . . 59
6 Hours Logged per Team Member . . . . . 59
7 Labour Costs . . . . . 60
8 Generalised Coordinate Relationships: Ball . . . . . 67
9 Generalised Coordinate Relationships: Body . . . . . 67
10 Generalised Coordinate Relationships: Motors . . . . . 68

# 1 Introduction

The concept of a ballbot is simple - it is a robot, which balances on a ball. The ball therefore acts as the single spherical wheel, allowing the robot to travel in any direction. In contrast to traditional robots, which rely on a low centre of mass and large wheel-base to remain upright, ballbots must actively balance, resulting in *dynamic stability*. This potentially allows for the advancement of human-sized robots, particularly with respect to stability.

This project aims to design and build two ballbots, primarily for educational purposes. The first is a ballbot constructed out of a Lego NXT Mindstorms Kit which can be used as a small teaching aide within a classroom environment. The second proposed ballbot is a Full-Scale, or human sized, Ballbot, which can be used at the University of Adelaide for promotional reasons.

The development of these ballbots is a full systems project, including derivation of equations of motion of a generalised ballbot, design and construction of the two ballbots, design and implementation of a controller and relevant software to balance each ballbot, and testing of the systems. The educational aspect of this project requires that these components of the project be documented clearly for student use.

## 1.1 Motivation

Man has long dreamed of robotic personal aides, created to perform his every task. The idea of automation appeals not only to those who perhaps are too lazy to complete the task itself, but to those dreaming of increasing productivity or efficiency within their own lives. A requirement for such a robot is that it is of similar size and shape to that of a human being. To maintain a human-like shape, the wheel-base of such a robot must be significantly less than the height. As a result, on traditional multi-wheeled robots, only a small shift in the position of the centre of gravity can cause the robot to become unstable. This may be corrected by lowering the centre of gravity of the robot to reduce the lever-arm, however, this generally comes at the cost of significant dead-weight being added to the robot. Additionally, the speed at which the robot can travel is generally limited lest the robot's momentum causes it to tip over.

One plausible solution to this problem, and the one developed in ballbot projects, is *dynamic stability*. That is, control theory is used to ensure that the robot stays upright, without the need to rely on static stability. Dynamic stability has been utilised most famously on the single-person transport unit, the Segway. The Segway (Figure 1) has two axially-aligned wheels, and utilises the solution to the 'Inverted Pendulum' control problem. The Segway does, however, rely on static stability in its transverse direction - a restriction overcome by the ballbot concept which aims to produce a robot that is dynamically stable in all directions. Furthermore, the area of dynamic stability is both relevant to a student's education, and an interesting application of control theory. This makes the ballbot an ideal project to develop at the University of Adelaide as an education display tool.

Figure 1: A Segway (Segway Inc, 2009)

## 1.2 Background

As previously mentioned, the concept of a ballbot is quite simple. It consists of a ball, and a robot which balances on top of the ball. The robot can balance by driving the ball, causing its base to move. Furthermore, the robot may also move along the ground by leaning, and driving the ball such that it does not fall over, but moves.

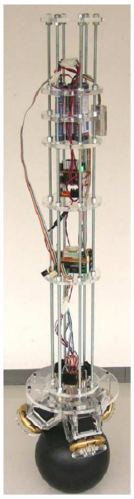
Published works on three preceding robots of this nature were found. The first of these robots, ERROSphere, was presented by Havasi (2005). ERROSphere is not of human-like proportions, however, and therefore was not considered significant with respect to this project. The remaining two robots meet this criteria form the basis of the previous work considered in this project. The first of these robots was constructed and tested at Carnegie Mellon University (CMU) in 2005, and is documented by Lauwers et al. (2005). This robot, was given the name 'Ballbot' which has been, within the context of this report, taken to the name of all such robots. The second robot was developed at Tohoku Gakuin University (TGU) in 2008, as documented by Kumagai and Ochiai (2008). This ballbot is similar, but uses a more complex driving mechanism, and presents a more elegant solution. Photos of these two robots can be seen in Figure 2.

Both of these ballbots are capable of balancing and point-to-point movement, with evidence of further development on the CMU ballbot including a retracting stand (Nagarajan et al., 2009) and the addition of an arm on top of the ballbot (Schearer, 2006). Additionally, since the commencement of a project, the NXT Ballbot (Figure 2(c)) was produced by Yamamoto (2009). This has been constructed very recently, and thus has not been included in great detail within this report.

## 1.3 Scope and Objectives

The scope of this project includes the development of a Lego Ballbot, and a Full Scale Ballbot. It is not expected that a large improvement be made on the existing ballbot prototypes, but simply that they be replicated in a simple, functional way, in order to the demonstrate the principle of control theory. The following were determined to be core objectives for the success of the Ballbot project.

(a) CMU's Ballbot (Lauwers et al., 2005)



(b) TGU's Ballbot (Kumagai and Ochiai, 2008)

(c) NXT Ballbot (Yamamoto, 2009)

Figure 2: Existing Ballbots

**Derivation of the Equations of Motion:** The equations of motion of the Ballbot were to be derived in state space form. Ideally, these equations were to perfectly model the actual dynamics of the system. However, given the complexity of the system and the number of unmeasurable effects, such as friction, it was anticipated that a number of assumptions would be required.

**Design and Construction of a Lego Ballbot:** The first ballbot to be created in the project was the Lego Ballbot. This ballbot was to be designed and constructed only from Lego parts, and, where possible, only those parts contained within the Lego 8527 NXT Mindstorms kit.

**Development of a Controller to Stabilise a Ballbot:** A software controller was to be designed, which could be used to dynamically balance a ballbot. The primary function of the controller was to keep the ballbot upright when faced with small disturbances. However, once this functionality of the controller was implemented, the controller was to be extended to allow for command tracking. Initially the controller would be applied to the Lego Ballbot, however would also form the basis of the Full Scale Ballbot controller.

**Remote Control of the Lego Ballbot:** Following the successful implementation of the controller on the Lego Ballbot, remote control of this ballbot was desired. As the Lego NXT kit allows for Bluetooth communication, it was anticipated that this would be used to provide the remote control capabilities.

**Design and Construction of a Full Scale Ballbot:** Based upon the Lego Ballbot, a Full Scale Ballbot was to be designed and constructed. This ballbot was to be of dimensions similar to a human, approximately 1.5m tall and no more than 0.7m wide. Design of the Full Scale Ballbot included development of the design concept, specification and selection of components, and detail design including integration of all components. The controller used for the Full Scale Ballbot was to be a modified version of the controller developed for the Lego Ballbot.

In addition to these core objectives, a number of extension goals were determined, as follows.

**Create Building Instructions for the Lego Ballbot:** It was desired that the Lego Ballbot be easily

reproduced for teaching purposes. For this reason, step by step instructions detailing the construction of the Lego Ballbot were desired.

**Produce a Virtual Reality Model of the Lego Ballbot:** To extend the use of the Lego Ballbot as a teaching aide, a virtual reality model of the Lego Ballbot was desired, which represents the derived equations of motion of the Ballbot and uses the implemented controller.

**Extend Functionality of the Lego Ballbot:** In order to further develop the concept of the Ballbot, the functionality of the Lego Ballbots could possibly be extended beyond simply balancing and command tracking. Possibilities included the addition of yaw control and/or some form of environmental interaction, such as obstacle avoidance.

**Extend Functionality of the Full Scale Ballbot:** As for the Lego Ballbot, it was desired to extend the functionality of the Full Scale Ballbot. This possibly included the implementation of command tracking and/or yaw control.

## 1.4 Approach

In order to ensure successful completion of the project objectives, work on the project was undertaken in a number of stages. Firstly, a review of the existing literature on ballbots and associated theory was undertaken to provide a basis for subsequent project work. Following this, the underlying theory of the ballbot system could be developed, including the derivation of the ballbot dynamics and development of a generalised controller.

Using the experience and knowledge gained from the literature review and theoretical development, the Lego Ballbot was designed and constructed and the controller realised on this platform. Once successful balancing was achieved, this Ballbot was further developed to include remote control capabilities and yaw control.

During Lego Ballbot development, the Full Scale Ballbot was also designed. This allowed for the experience gained from the Lego system to be applied to the Full Scale Ballbot at the design stage, such that potential problems could be identified and corrected early in development. This process lead to construction of the Full Scale system, which was then tested to achieve a successfully balancing system.

## 1.5 Project Constraints

The primary constraints on the ballbot project were those of timeframe and budget. The project commenced at the beginning of March 2009 with expected completion by the end of October in the same year. Funding for the project was received from the Open Day Innovation Fund, which provided $3000 in addition to the base project budget of $400 provided by the School of Mechanical Engineering. However, the additional funding was only confirmed on 14 April 2009, which delayed the start of design process of the Full Scale Ballbot, as this would not have proceeded had funding not been received. This further restricted the timeframe for design, construction and testing of the Full Scale system.

# 2 Literature Review

The literature review presented in this section details the previous work in three areas of ballbot development. The first is the derivation of the equations of motion, which aids development of a ballbot controller. The second is the physical construction of the hardware itself. The final area is the theoretical controller used to maintain the ballbot balancing or following a command.

As ballbots are a relatively undeveloped concept, only limited literature exists on the topic. However, as will be discussed in the following sections, the ballbot problem can be modelled as two independent inverted pendulum systems, a system which is well documented. As such, the review of previously-derived dynamics and controllers has been extended marginally into solutions for the inverted pendulum, or similar, problems.

## 2.1 Equations of Motion

The aim of this review of previously derived equations of motion was to provide a basis to derive the dynamics of the Ballbot for this project. This includes firstly analysing any assumptions that could be made when deriving the Ballbot dynamics. Secondly, current methods of deriving the equations of motion of the Ballbot are discussed. Finally, existing results are examined, including limitations of these results.

### 2.1.1 Assumptions

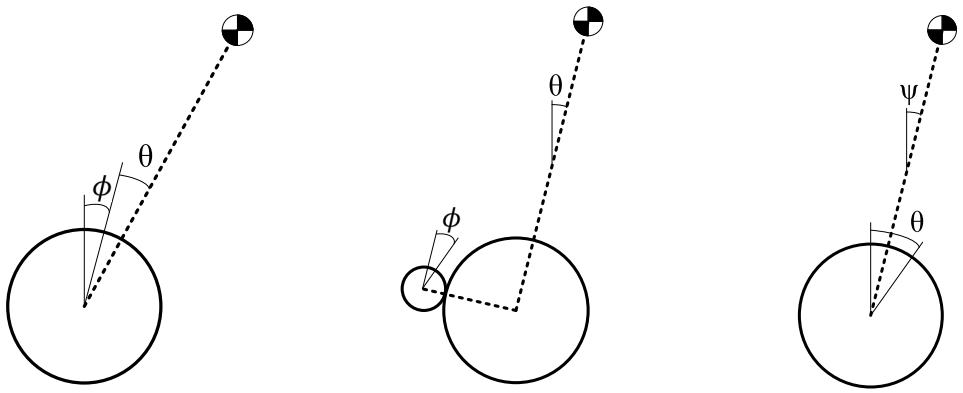
The dynamics of the Ballbot are complex and, for this reason, difficult to derive unless assumptions to simplify the model of the Ballbot are made. A simplified Ballbot model consists of a rigid body balancing atop a rigid sphere. For this model it can be assumed that the motion in the two planes of tip - pitch and roll - are decoupled, and that the equations of motion are identical in these two planes (Lauwers et al., 2005). This assumption results in the ability to design a controller for the full three dimensional system by designing controllers for the two independent planes (Lauwers et al., 2005). Further assumptions include modelling the friction in the system as viscous friction only, while ignoring static and non-linear friction effects (Lauwers et al., 2005, 2006, Yamamoto, 2008). This is because these effects introduce discontinuities and severe non-linearities into the model, which is undesirable for control (Lauwers et al., 2006). The assumptions of a simplified two plane ballbot model and viscous friction allow the equations of motion to be determined in a form which allows for controller design.

### 2.1.2 Derivation Method

The dynamics of the simplified Ballbot model can be derived using the Lagrangian approach, and has previously been performed by Lauwers et al. (2005) and Liao et al. (2008). This method defines a quantity the Lagrangian $L(\mathbf{q}, \dot{\mathbf{q}}, t)$, where $\mathbf{q}$ are the generalized co-ordinates of the system, and $t$ is time. This quantity $L$ is simply the sum of the kinetic energy of the system, $T$, minus the potential energy, $U$, shown in Equation (1) (Brizard, 2008).

$$L = T - U \tag{1}$$

The choice of co-ordinates $\mathbf{q}$ is arbitrary, with the constraint that they must fully define the system. However, it is advantageous to use co-ordinates that directly relate to measurable quantities, such as the body angle and body-motor angle (Schearer, 2006). Figure 3 shows examples of co-ordinates chosen when deriving the dynamics of one plane of the simplified Ballbot or similar systems.




(a) Body-ball angle $\theta$ and ball angle $\phi$, adapted from Lauwers et al. (2006)

(b) Body angle $\theta$ and body-motor angle $\phi$, adapted from Schearer (2006)

(c) Body angle $\psi$ and ball angle $\theta$, adapted from Yamamoto (2008)

Figure 3: Generalised coordinates for ball/wheel balancing systems

To derive the equations of motion, the Euler Lagrange Equations (2) are then applied to the Lagrangian.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = F_i \tag{2}$$

where the $F_i$ are generalized forces (Brizard, 2008). This results in one Lagrange equation for each co-ordinate $q_i$, of the form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{F} \tag{3}$$

where $\mathbf{M}(\mathbf{q})$ is the mass matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the coriolis matrix, $\mathbf{G}(\mathbf{q})$ is the gravity matrix and $\mathbf{F}$ is the vector of generalised forces. The Equations (3) are the equations of motion of the system, which may be non-linear.

### 2.1.3 Existing Results

The most relevant existing work on the derivation of the equations of motion of the Ballbot is presented by Lauwers et al. (2005) at CMU. This derivation uses the simplified ballbot model and Lagrangian method, as discussed in Sections 2.1.1 and 2.1.2. The generalised co-ordinates and model used is shown in Figure 3(a).

This derivation results in equations of motion in a similar form to that presented in Section 2.1.2, but also includes the effects of viscous friction as an additional term, $\mathbf{D}(\mathbf{q})$ (Lauwers et al., 2005), resulting in non-linear equations of the form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \mathbf{D}(\mathbf{q}) = \mathbf{F} \tag{4}$$

The derived equations provide the equations of motion of a ballbot. However, a major limitation of this result is that the generalised forces **F** are simply modelled as torques, and do not consider the motor dynamics. A model which includes the effect of motor dynamics was used by Yamamoto (2008). While this system is that of an inverted pendulum, not a Ballbot, it is analogous to one plane of the Ballbot dynamics. The model includes the effect of the motor by adding the motor rotational kinetic energy to the Lagrangian quantity, and replacing the control torques with dynamics based on the DC motor equation, as follows

$$\tau = \frac{K_t(v - K_b\dot{\theta})}{R} \tag{5}$$

where $\tau$ is motor torque, $K_t$ is the motor torque constant, $v$ is motor voltage, $\dot{\theta}$ is motor angular speed, $K_b$ is the motor back emf constant and $R$ is the motor resistance. It should be noted that this equation ignores inductance in the motor, but as this is generally negligible. Equation (5) can be used in conjunction with the result of Lauwers et al. (2005) to provide a basis to derive the dynamics of a ballbot.

## 2.2 Hardware Design

A physical ballbot is a relatively simple device, requiring few components. The basic functionality of the ballbot, that is to balance upright on a ball, requires a ball, a means of actuating the ball, sensors to measure the angle of ballbot tilt, and a microcontroller. The approaches to these areas vary slightly between the two existing implementations of a full scale ballbot, constructed by Carnegie Mellon University and Tohoku Gakuin University respectively.

### 2.2.1 Drive Mechanism

The drive mechanism - including the interface between the driving motors and the ball - is critical to the success of a ballbot. CMU and TGU take different approaches to the design of this component, which are discussed in this section. Furthermore a drive mechanism proposed by Wu and Hwang (2008) is examined.

The CMU ballbot took perhaps the simplest approach to the drive mechanism. It uses an 'inverse mouse-ball drive', illustrated in Figure 4, in its initial design. Two perpendicular driving rollers were used, each driven by a DC servomotor through a belt. Opposite each driving roller, two spring-loaded idling rollers are used to locate the ball (Lauwers et al., 2006). The advantage of employing this system is that, as the two driving rollers are orthogonal to each other, the control can be simplified to two inverted pendulum systems in separate planes. However, a degree of slip must occur between the ball and the roller when being driven by the orthogonal roller, whilst, at the same time, the a high degree of friction is required between the driving roller and the ball, and the ball and the ground. Thus, both high friction and low friction ball-roller interaction is required concurrently, which can be difficult to accomplish.

The approach to the drive mechanism of the TGU ballbot team was slightly more complicated, and can be seen in Figure 5. Three stepper motors, fixed rotationally symmetrically at 120° are used to drive the ball, with omniwheels used as the interface to the ball. These custom-built wheels were manufactured to allow the ball to roll freely along the axis of the wheel but provide actuation in the direction of wheel rotation.

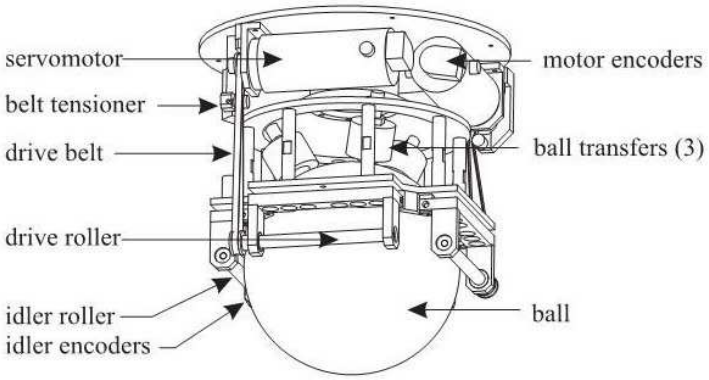



Figure 4: Driving Mechanism of CMU Ballbot (Lauwers et al., 2006)

The use of stepper motors in the TGU system allows for precise control, and removes the need of an encoder, which is required for the servomotors used in the CMU system. Additionally, the backlash induced by the belt-drive system in the CMU ballbot is removed due to the direct drive arrangement. This drive mechanism, however, comes at a computational cost to the controller, due to the added complexity of the three-wheel drive. It may also be important to note that the TGU ballbot will not transfer all of the energy from its motors to the ball, due to the non-orthogonal nature of the system.

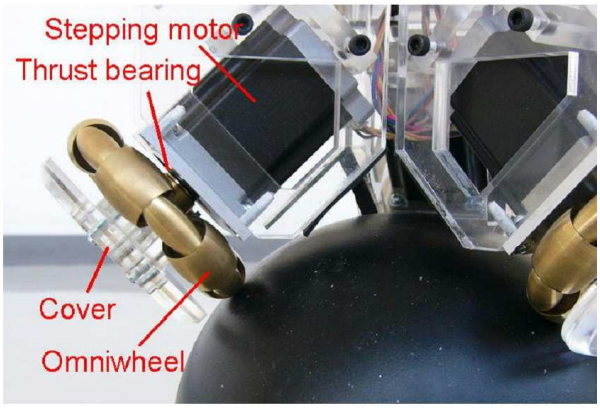



Figure 5: Driving Mechanism of TGU Ballbot (Kumagai and Ochiai, 2008)

Lauwers et al. (2006) noted that the 'frictional effects [were] asymmetric' on the CMU ballbot, as the single driving roller in each plane 'pushed' or 'pulled' the ball. This is not a problem with the TGU ballbot due to its completely symmetric nature. It should be noted that CMU revised their design, to drive all rollers (Nagarajan et al., 2009) and reduce these asymmetric effects. Furthermore, the CMU ballbot requires ball transfers (see Figure 4) to support the weight of the ballbot on top of the ball. This increases the friction on the ball, and may increase wear of the ball. This, again, is not a problem with the TGU driving mechanism.

In addition to this, the TGU ballbot is capable of yaw control - which cannot be achieved on the CMU ballbot without the addition of further actuators. Modifications to the CMU Ballbot included the actuation of a rotating platform above the drive mechanism, to achieve a degree of yaw control through the rotation of the body (Nagarajan et al., 2009). This type of yaw control,

however, has not been considered as part of the drive mechanism.

In addition to the two constructed ballbots, other studies have been done on the drive mechanism. Wu and Hwang (2008) propose a ballbot driven by four omniwheels (see Figure 6), named the Combination of Omniwheel and Spherical Wheel Unit (CWWU). The four driving wheels reduce the asymmetric frictional effects experienced by the CMU ballbot. It would not, however, be capable of yaw control without additional actuators. The CWWU also does not require ball transfers as the CMU mechanism does. Like the CMU drive mechanism, the entirety of the driving force is transferred to motion along the desired plane. This reduces the power required from the motors, and hence cheaper alternatives can be sought. However, this is probably more than offset by the cost of purchasing four motors.

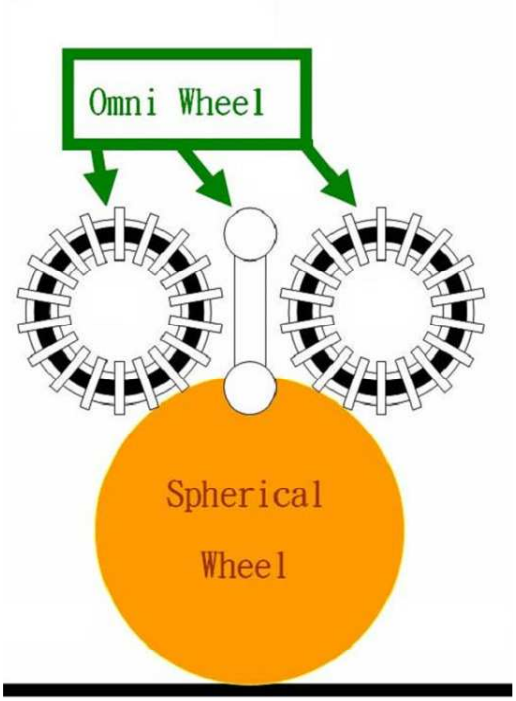



Figure 6: A Proposed Drive Mechanism using Four Omniwheels (Wu and Hwang, 2008)

### 2.2.2 Ball

The properties of the ball depend on the drive mechanism. As mentioned in section 2.2.1, the CMU ballbot suffers from requiring a high friction and low friction ball at the same time. This problem was reduced with the TGU ballbot, but still poses a slight problem (as contact points are required in order for the ball to spin). The balls have generally been fabricated with a trial and error mentality, due to the experimental nature of the ballbot system.

The CMU ballbot originally used a 200 mm diameter steel shell covered with a 3.2 mm urethane outer layer (Lauwers et al., 2006). A number of urethane formulations with different durometers have been constructed and trialled (Lauwers et al., 2006). It was noted that this 'worked well' (Lauwers et al., 2006) but the ball wore out. The ball was then replaced with a lighter aluminium shell of 190.5 mm diameter and a 12.7 mm urethane layer. Lauwers et al. (2006) notes that no visible wear has occurred on this ball, citing lower shear stresses in the urethane layer as a possible reason.

It is likely that this would have increased the damping in the system, although no comment on its effect on the performance of the ballbot has been found.

The TGU ballbot uses a bowling ball which is covered in rubber to increase grip (Kumagai and Ochiai, 2008). A basketball has also been trialled in the TGU ballbot, and, although it could maintain balance, it was unsteady (Kumagai and Ochiai, 2008). As such, the rubber-coated bowling ball was used as it was more rigid and provided better performance.

### 2.2.3 Tilt Sensors

In order to be dynamically stable, the ballbot must actuate the ball such that it is driven in the correct direction to remain upright. Sensors are therefore required to measure the tilt of the ballbot's body. However, errors in the measured state exist when only one type of sensor is used due to inherent sensor limitations. CMU and TGU ballbots employ similar solutions to this problem in their measurement of body tilt and angular velocity.

Gyroscopes and accelerometers can be used to measure body tilt. Gyroscopes are used to measure the angular velocity. However, due to the integration of an angular velocity measurement, a 'drifting' of the angle measurement can occur if an offset exists in the angular velocity measurement. Accelerometers can be used to measure the angle directly by using the direction of acceleration due to gravity. However, this assumes that no other acceleration is present, and thus any vibrations within the frame or movements of the body can cause the sensors to give incorrect readings.

The solutions to this problem on the CMU and TGU ballbots are similar. The TGU ballbot uses ADXRS401 gyroscopes and ADXL203 accelerometers, and uses a first-order digital filter to combine the signals from these two sensors, using the gyroscope signal for high frequency signals and the accelerometer signal for low frequency signals (Kumagai and Ochiai, 2008). The CMU ballbot uses an 'all-in-one' solution, with a Crossbow Technology VG700CA-200 Inertial Measuring Unit (IMU) (Lauwers et al., 2006). This unit contains both accelerometers and gyroscopes and provides a Kalman-filtered pitch and roll angles.

### 2.2.4 Controller Hardware

Due to the limited functionality of the existing ballbots, minimal processing power is required for the ballbot. As such, simple microcontrollers are suitable for use as the main controller on the ballbot. This can be seen in the TGU ballbot, which utilises a Renesas H8/3052 microcontroller (Kumagai and Ochiai, 2008). This microcontroller has 16-bit registers and a maximum clock rate of 25 MHz (Hitachi, 2001). It is a simple microcontroller which can be easily interfaced with the sensors and actuators used in the ballbot. In comparison, the processing power of the CMU ballbot is provided with a 200 MHz Pentium processor (Lauwers et al., 2006). This provides more processing power, and thus further functionality can be added to the ballbot.

## 2.3 Theoretical Controller

A variety of different approaches have been taken in the control of ballbot systems. As discussed in Section 2.1, the Ballbot system is a non-linear system. The control systems on the two existing ballbots (CMU and TGU) have both approximated the system as linear, and thus been able to use linear controllers. Only one other non-linear method of control for a ballbot is discussed, that of Sliding-Mode Control.

The ballbot system can also be approximated by two independent inverted pendulum systems, one for each plane. As such, these approaches can be translated to the Ballbot system. For this project, due to the use of a Lego Mindstorms kit, a controller of interest was that of the NXTway-GS, a two-wheeled self-balancing robot system. A review of this controller was undertaken due to its relevance to the project.

### 2.3.1 Carnigie Mellon University Controller

The controller implemented in the CMU ballbot has two loops - an inner Proportional-Integrator (PI) loop controlling the ball's angular velocity, and an outer Linear Quadratic Regulator (LQR) controller which uses full state feedback (Lauwers et al., 2006). Figure 7 gives a visual explanation of this structure.

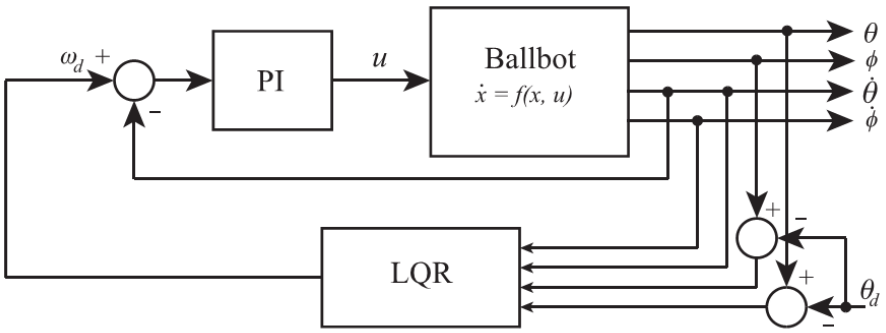



Figure 7: Structure of the Controller used on the CMU Ballbot (Lauwers et al., 2006)

The inner loop is used to ensure that the ball velocity tracks the desired ball velocity, and was experimentally tuned to overcome the effects of static and dynamic friction, which were not modelled (Lauwers et al., 2006). The outer loop controls all the states of the system (including an extra one introduced due to the integrator in the inner loop). The LQR controller was created using the Matlab *lqr* command, and the **Q** and **R** matrices were based on simulation results (Lauwers et al., 2006). It was also noted that the LQR controller generated by Matlab had to be slightly modified due to the model not taking into account some physical factors.

The controller used in the CMU ballbot is a well-known and common approach. Although it is simply a linear controller, it is apparent that this is sufficient to stabilise the ballbot, based on performance of the physical ballbot.

### 2.3.2 Tohoku Gakuin University Controller

The Tohoku Gakuin University ballbot uses a simple Proportional-Derivative (PD) controller to balance. Due to the unusual arrangement of the drive mechanism (discussed in Section 2.2.1, the controller does not control the wheels directly, but instead controls two 'virtual' motors, based in the planes in which the sensors lie, in a similar arrangement to that of the CMU ballbot (Kumagai and Ochiai, 2008). A simple mathematical relationship is then used to convert the signals from the virtual wheels to the actual wheels. Kumagai and Ochiai (2008) indicate that the equations used in the controller are as follows:

$$a_x = K_A\theta_x + K_{AV}\dot{\theta}_x + K_T(x - x_0) + K_V v_x \tag{6}$$

$$a_y = K_A\theta_y + K_{AV}\dot{\theta}_y + K_T(y - y_0) + K_V v_y \tag{7}$$

The proportional gains, $K_A$ and $K_T$, and the derivative gains, $K_{AV}$ and $K_V$, were all experimentally derived (Kumagai and Ochiai, 2008). Additionally, Kumagai and Ochiai (2008) note that the commands are acceleration, as opposed to conventional inverted pendulum systems which use torque as the command. This is due to the fact that stepper motors are used.

This controller is also very simple and linear in nature. Once again, based on the performance of the physical ballbot, it is evident that the controller is suitable. It is possible that this simpler control method is effective due to the properties of the ballbot, including less damping on the ball. Thus, although the construction and drive mechanism is more complicated, the resulting controller is simple at its core, due to a mechanical design which can be more accurately modelled.

### 2.3.3 Sliding-Mode Control

A Sliding-Mode Controller for the ballbot system was proposed by Liao et al. (2008) . This controller is designed for the inverse mouse-ball drive used in the CMU ballbot (discussed in section 2.2.1), however has not been implemented or tested in a physical ballbot. Liao et al. (2008) presents simulations of the sliding-mode controller, showing that this method is capable of controlling and stabilising a ballbot. However, it is difficult to determine how this compares to existing linear-decoupled controllers due to the lack of vigorous testing, and simplistic and single-test approach taken with the presented simulations.

### 2.3.4 NXTway-GS and NXT Ballbot Controllers

The development of a model-based controller for a the NXTway-GS, a two-wheeled self-balancing robot built with Lego Mindstorms components, is documented by Yamamoto (2008). This robot was constructed in 2008, and was considered a starting point for this project. Additionally, since the commencement of the project, the NXT Ballbot has also been constructed and documented by Yamamoto (2009). The controllers developed for these robots are very similar.

At the core of the controllers used in these robots are LQR controllers. However, a second loop is also used for the controller, in order to introduce integral control. The integrator is required in order to use servo control (Yamamoto, 2008), as shown in Figure 8.

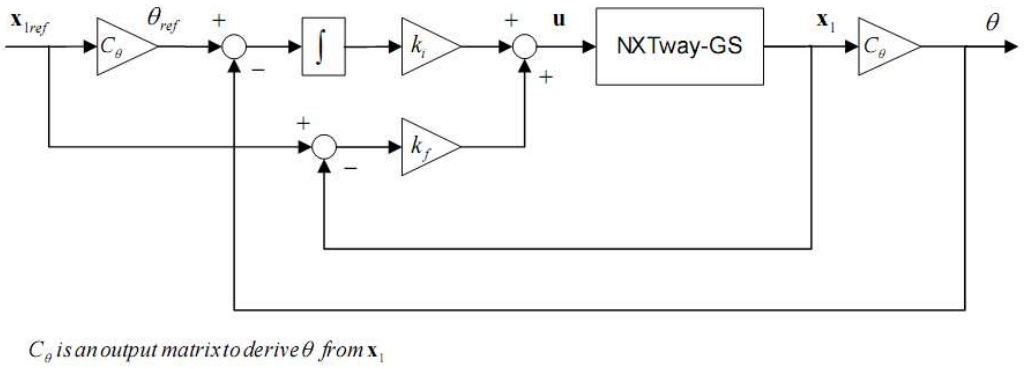



Figure 8: Structure of the Controller used in the NXTway-GS (Yamamoto, 2008)

Similar to the CMU ballbot, the Matlab *lqr* command is used to generate the gains for the controller. Once again, this is a well-known technique, which appears to work adequately well on the NXTway-GS two-wheeled self-balancing robot and NXT Ballbot.

# 3 Theoretical Controller

The controller used in the Ballbot project is a state space controller with gains derived using LQR. The state space controller requires knowledge of the equations of motion of the system, the creation of the state space representation of the model. This section provides an overview of state space control and LQR, as well as the development of the state space model of a ballbot.

It is important to note that within this report, the ballbot system is treated as two identical independent systems. Only a single plane is discussed within this section, with the assumption that the second plane is identical.

## 3.1 State Space Control

A state space controller was chosen for use within this project. It is taught in depth in the Automatic Control II course at the University of Adelaide, and thus its use in this project lends the ballbot system being used as a teaching aide in this course. Furthermore, state space control is well suited to a ballbot system, due to its applicability to Multiple Input Multiple Output (MIMO) systems. This is particularly relevant as multiple sensors are required for each plane in order to gain control over the body angle and position of a ballbot.

The major disadvantage of using a state space controller is that it is a linear controller. That is, it uses a linear model of the system, even if the system itself is non-linear. However, due to the success of controlling ballbot systems using linear controllers by Lauwers et al. (2005) and Kumagai and Ochiai (2008) as discussed in Section 2.3, it was felt that the state space controller would perform sufficiently well to be capable of controlling the system.

The form of a generalised state space controller is shown in Figure 9. State space control models the system as a system of linear differential equations, of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{8}$$

where $\mathbf{x}$ is the system state, $\mathbf{u}$ is the control signal, and $\mathbf{A}$ and $\mathbf{B}$ are matrices containing the linear dynamics of the system. In a state space controller, the control signal, $\mathbf{u}$ is based on the difference between the desired state $\mathbf{x_{ref}}$ and the state of the system $\mathbf{x}$ as follows

$$\begin{aligned} \mathbf{u} &= \mathbf{k}(\mathbf{x_{ref}} - \mathbf{x}) \\ &= \mathbf{k}\mathbf{e_x} \end{aligned} \tag{9}$$

where $\mathbf{k}$ is the matrix of gains. Thus, in order to apply this controller to the Ballbot system, suitable gains must be determined. Additionally, depending on the measurement data available, state estimation may be required to estimate any unmeasured states.

## 3.2 Linear Quadratic Regulator

The Linear quadratic regulator (LQR) attempts to choose gains which optimise the control signals, based on a heuristic. The significant disadvantage to this approach is that it is often hard to select an appropriate quantifiable heuristic, and this choice is often superficial. However, it was felt that an LQR controller was an appropriate choice for this system due to its simplicity and its similarities to the CMU ballbot and NXTway-GS controller, which also use LQR controllers

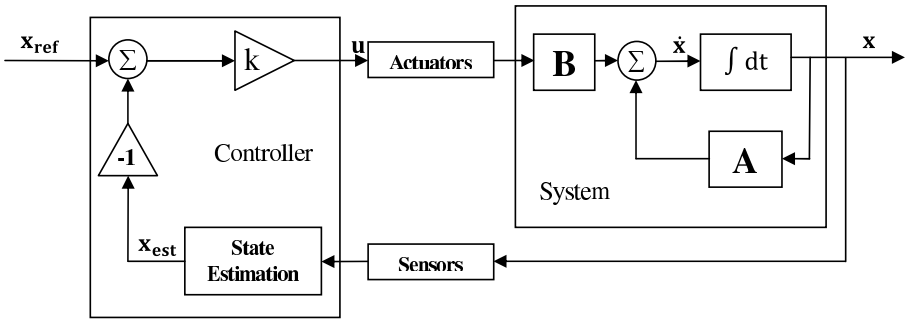



Figure 9: State Space Control Block Diagram

(Lauwers et al., 2006, Yamamoto, 2008). Additionally, LQR controllers are well-understood, and are taught in control subjects at the University of Adelaide.

The heuristic developed for LQR controller requires a **Q** matrix, and a **R** matrix. The **Q** matrix is a weighting matrix related to performance, which indicates the desirability of each state being equal to its set point. The **R** matrix is a matrix which penalises the use of larger control inputs. These matrices are then used to find the gain matrix **K** in the following equation:

$$\int_t^T [\mathbf{x}^\mathbf{T}(\tau)(\mathbf{Q} + \mathbf{K}^\mathbf{T}\mathbf{R}\mathbf{K})\mathbf{x}(\tau)]d\tau \tag{10}$$

The Matlab command *lqr* can be used to solve this equation, given the dynamics in linear state space form and selections for **Q** and **R**.

## 3.3 State Space Model of a Ballbot

In order to use state space control for a Ballbot, the equations of motion of the system are required in state space form. The dynamics are necessary for use with LQR and also allow for simulations. The equations of motion of a ballbot have been previously derived and documented by Lauwers et al. (2006, 2005) and Liao et al. (2008). However, was still deemed desirable to derive the dynamics within this project in order to overcome some of the limitations of the previous work, and also to provide a qualitative understanding of ballbot dynamics.

The aim of this section is to derive the dynamics in a form which may be used as a basis for controller design. This initially involves making assumptions in order to create a simplified ballbot model in one plane upon which the dynamics can be derived. Following this, the Lagrangian approach is used to derive the equations of motion of the simplified model. These equations are linearised and written in a form which can be used for controller design. Finally, the complete linear model of the Ballbot dynamics is determined.

### 3.3.1 Assumptions and Definitions

To facilitate the derivation of ballbot equations of motion, several assumptions were made. The following assumptions are used in the derivation, based on those discussed in Section 2.1.1.

- The Ballbot is comprised of two parts; a rigid body atop a rigid ball
- The control torques are applied between the body and the ball

- Motion is decoupled in the pitch and roll planes, and the equations of motion are identical in these planes
- Only viscous friction is present
- There is no slip

The generalised co-ordinates are chosen to match the anticipated quantities that can be directly measured; body angle and motor shaft angle. This results in the simplified Ballbot model in one plane, the x-z plane, is shown in Figure 10. An identical model exists for the y-z plane.

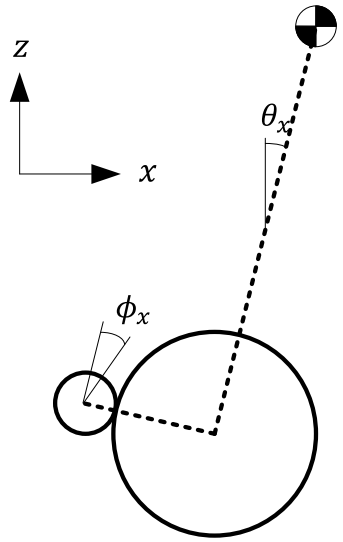



Figure 10: The Simplified Ballbot Model

In addition, the following parameters were defined to be used in the derivation.

- $R_b$ - Radius of the ball
- $L$ - Distance from the centre of mass of the body to the centre of the ball
- $M_B$ - Mass of the body
- $M_b$ - Mass of the ball
- $I_b$ - Moment of inertia of the ball
- $I_{Bx}$ - Moment of inertia of the body, about the x axis
- $I_M$ - Moment of inertia of the motor rotors (including wheels), about their rotational axis
- $n$ - Gear ratio, motor to ball, including direction
- $\mu_{Bb}$ - Friction coefficient between body and ball
- $\mu_{Bg}$ - Friction coefficient between body and ground
- $K_b$ - Back EMF constant of motors
- $K_t$ - Torque constant of motors
- $R_m$ - Resistance of the motors

### 3.3.2 Derivation of Equations of Motion

The equations of motion of the simplified Ballbot model in one plane (x-z) are derived in Matlab using the Lagrangian approach as discussed in Section 2.1.2. Following is a summary of the approach used to derive the dynamics in each plane. A more complete discussion, including all results, is included in Appendix A. The Matlab code used can be found in Appendix B.

The derivation using the Lagrangian approach begins by finding the Lagrangian, $L$, which is the difference between kinetic and potential energy of ballbot system, including the ball, body and motors, as follows

$$L = T_{linb} + T_{linB} + T_{rotb} + T_{rotB} + T_{rotm} - V_b - V_B \tag{11}$$

where $T_{linb}$, $T_{linB}$, $T_{rotb}$, $T_{rotB}$, $T_{rotm}$ are the linear ($lin$) and rotational ($rot$) kinetic energies of the ball ($b$), body ($B$) and motors ($m$), and $V_b$, and $V_B$ are the potential energies of the ball and body, all expressed in terms of the generalised co-ordinates and physical parameters given in Section 3.3.1.

The Euler-Lagrange Equation (2) is then applied, with the force matrix, $F_i$, including the effects of viscous friction and the motor dynamics by applying Equation (5). The resulting equations of motion are of the form:

$$\mathbf{M_x}(\mathbf{q}, \dot{\mathbf{q}})\ddot{\mathbf{q}} + \mathbf{C_x}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G_x}(\mathbf{q}) + \mathbf{D_x}(\dot{\mathbf{q}}) = \mathbf{F_x}(\mathbf{q}, \dot{\mathbf{q}}, v_x) \tag{12}$$

where the *Mass* matrix,$\mathbf{M_x}$ =

$$\begin{pmatrix} I_{Bx} + I_M + I_b + L^2\, M_B + M_B\, {R_b}^2 + M_b\, {R_b}^2 + 2\, L\, M_B\, R_b\, \cos(\theta_x) & I_M + I_b\, n + M_B\, n\, {R_b}^2 + M_b\, n\, {R_b}^2 + L\, M_B\, n\, R_b\, \cos(\theta_x) \\ I_M + I_b\, n + M_B\, n\, {R_b}^2 + M_b\, n\, {R_b}^2 + L\, M_B\, n\, R_b\, \cos(\theta_x) & I_M + I_b\, n^2 + M_B\, n^2\, {R_b}^2 + M_b\, n^2\, {R_b}^2 \end{pmatrix}$$

the *Coriolis* matrix, $\mathbf{C_x}$ =

$$\begin{pmatrix} -L\, M_B\, R_b\, \sin(\theta_x)\, \dot{\theta}_x^2 \\ -L\, M_B\, n\, R_b\, \sin(\theta_x)\, \dot{\theta}_x^2 \end{pmatrix}$$

the *Gravity* matrix, $\mathbf{G_x}$ =

$$\begin{pmatrix} -g\, L\, M_B\, \sin(\theta_x) \\ 0 \end{pmatrix}$$

the *Friction* matrix, $\mathbf{D_x}$ =

$$\begin{pmatrix} \mu_{Bg}\dot{\theta}_x \\ \mu_{Bb}\dot{\phi}_x \end{pmatrix}$$

and the *Force* matrix, $\mathbf{F_x}$ =

$$\begin{pmatrix} 0 \\ \frac{K_t(v_x - K_b\dot{\phi}_x)}{R_m} \end{pmatrix}$$

These equations provide a model of the non-linear dynamics of the ballbot system in one plane, the x-z. The dynamics of the y-z plane are identical, with appropriate subscript changes.

### 3.3.3 Linear State Space Model

To aid the development of a state space controller, knowledge of the system dynamics are required, in linear state space form. Generalised ballbot dynamics for one plane, the x-z, were derived in the previous section using suitable assumptions for a simplified ballbot. However these dynamics, expressed in Equation (12), are non-linear. To convert to linear state space form, these equations of motion are first rearranged to non-linear state space form as follows. Firstly, the equations are rearranged such that the highest order derivative is the subject:

$$\ddot{\mathbf{q}} = {\mathbf{M_x}}^{-1}(\mathbf{F_x} - (\mathbf{C_x} + \mathbf{G_x} + \mathbf{D_x})) \tag{13}$$

Following this the non-linear state space equations $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ can be derived by taking $\mathbf{x} = [\mathbf{q}^T \ \dot{\mathbf{q}}^T]^T$ and $u = v_x$, as follows

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}) \\ &= f(\mathbf{q}, \dot{\mathbf{q}}, v_x) \\ &= \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{v_x}) \end{bmatrix} \end{aligned} \tag{14}$$

As such, the state vector $\mathbf{x}$ for this model consists of

- $\theta_x$ - The angle of the body of the ballbot relative to the vertical in the x-plane
- $\phi_x$ - The angle of the motor shaft relative to the body in the x-plane
- $\dot{\theta_x}$ - The angular velocity of the body of the ballbot in the x-plane
- $\dot{\phi_x}$ - The angular velocity of the motor shaft relative to the body in the x-plane

and the control signal $\mathbf{u}$ is the motor voltage for the x motor, $v_x$. The non-linear state space equations are then linearised using a first order approximation as follows

$$\hat{\dot{\mathbf{x}}} = f(\bar{\mathbf{x}}, v_x) + J(\bar{\mathbf{x}})\hat{\mathbf{x}} \tag{15}$$

where $\hat{\mathbf{x}}$ is the linearised state about the point $\bar{\mathbf{x}} = [0 \ \phi_x \ 0 \ 0]^T$, and $J(\bar{\mathbf{x}})$ is the Jacobian at that point. Taking the linearised state $\hat{\mathbf{x}}$ to be the state vector results in the linear State Space Model of the ballbot in one plane

$$\dot{\mathbf{x}} = \mathbf{A_x}\mathbf{x} + \mathbf{B_x}v_x \tag{16}$$

where $\mathbf{A_x}$ and $\mathbf{B_x}$ are defined in Appendix A.

The states in this model give, in traditional control theory terms, Proportional and Derivative (PD) control. In addition to this the model is augmented to include an extra state

- $\int \phi_x$ the integral of the angle of the motor shaft relative to the body in the x-plane

The addition of this extra state eliminates steady state error, which may be introduced due to disturbances, noise or other properties of the system.

# 4 The Lego Ballbot

The Lego Ballbot was the first ballbot created during the project. The aim of this part of the project was to create an easily reproducible ballbot system suitable for use as a teaching aide. This ballbot, while a complete systems project on its own, also acted as a prototype for the Full Scale system. It allowed for the identification and solving of any general design or control issues present in a ballbot system at an earlier stage in the project.

This section describes the complete design process of the Lego Ballbot. This firstly includes a discussion of the creation of the Lego Ballbot, consisting of design of the ballbot itself and development of the software controller to stabilise the system. Following this a description of the test procedures is given, along with results displaying the successful operation of the Lego Ballbot. Finally, the implementation of a yaw controller, an extension goal, is presented.

## 4.1 Design

The Lego Ballbot was constructed using the Lego NXT Mindstorms products, due to its low cost and worldwide availability. This section begins by describing the parts available in the Lego NXT Mindstorms range, was used to construct the Lego Ballbot. Following this, the conceptual design of the Lego Ballbot is discussed, considering the drive mechanism and general layout of the Lego Ballbot. Finally, the detailed design and construction of the Lego Ballbot is shown.

### 4.1.1 Lego NXT Mindstorms Range

The NXT Mindstorms products extend the Lego product range into robotics applications. The 8527 Lego NXT Mindstorms kit is the starter kit for this line of products, and was considered a suitable starting kit for the project. The 8527 kit includes the NXT Brick, various sensors, three motors, and Lego Technic pieces. The NXT Brick is the processor in the NXT range, and has ports to interface with the sensors and motors. The sensors included in the kit include a touch sensor, an optical sensor, a sound sensor and as ultrasonic range sensor. The motors are 9V DC servo motors with built in encoders which provide a means of driving the ball of the Lego Ballbot. Specifications of these components can be found in Appendix C.

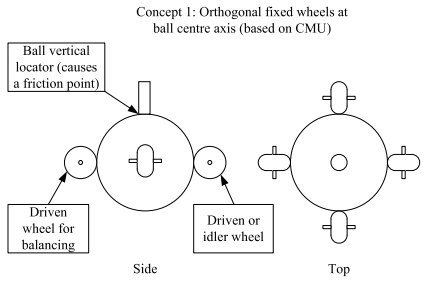
Additional parts not included in the kit, but available and of possible use for the Lego Ballbot include the HiTechnic Gyroscopic sensor and HiTechnic 3-Axis Accelerometer. The HiTechnic Gyroscopic sensor contains a single axis gyroscope and can be used to detect rotation in one axis. It measures the rotation rate in degrees per second up to a maximum rate of 360 degrees/sec (HiTechnic Products, 2008). The HiTechnic 3-Axis Accelerometer can be used to measure acceleration in all three axes, allowing measurement of tilt by detecting the direction of acceleration due to gravity. The resolution is approximately 1/200 g with a range of -2g to +2g (HiTechnic Products, 2008). In addition, various Lego wheels are available and may also be required, as the 8527 kit only includes one set of wheels which may not be of appropriate size for the Ballbot.

### 4.1.2 Conceptual design

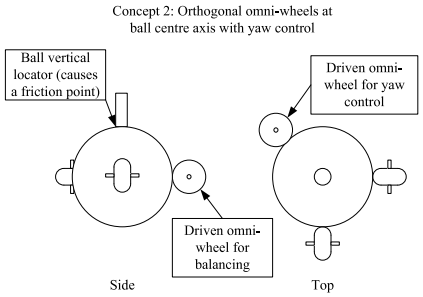
Design of the Lego Ballbot began with conceptual design, which considered the general layout of the Lego Ballbot and how the required functionality could be implemented. The aim of the concept design was to provide a design that will be used as a guide for Lego construction process. Firstly, several possible ball drive mechanism concepts were developed and evaluated. Based upon the drive mechanism and the anticipated requirements for the controller, the required components of the Lego Ballbot were determined, and the layout of these parts was considered to form the conceptual design. The conceptual design did not consider the use of individual Lego parts, as this was determined during the construction process, described in Section 4.1.3.

#### 4.1.2.1 Drive Mechanism

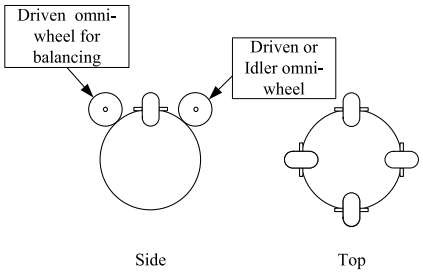
The drive mechanism of the Ballbot requires that the ball can be driven in any direction relative to the body. This is be achieved using actuated wheels or rollers which drive the ball. Several possible arrangements of ball and wheels were considered, shown in Figure 11. These designs were generated based on previous works, as discussed in Section 2.2.1, and brainstorming. The concepts were then evaluated in Table 1 based on weighted criteria outlined in the table.



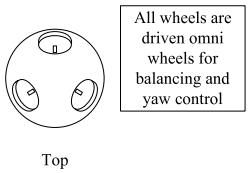

(1) Orthogonal fixed wheels at ball centre axis, adapted from Lauwers et al. (2006)



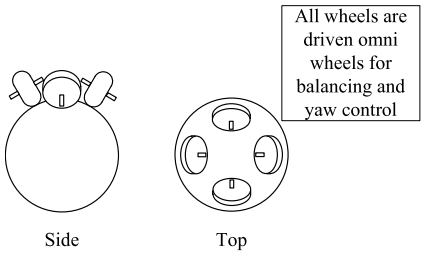

(2) Orthogonal omniwheels at ball centre axis




(3) Orthogonal omniwheels above ball centre axis, adapted from Wu and Hwang (2008)




(4) Tri omniwheels above ball centre axis, adapted from Kumagai and Ochiai (2008)




(5) Orthogonal omniwheel pairs above ball centre axis

Figure 11: Drive mechanism concepts

Table 1: Decision Matrix for Lego Ballbot Drive Mechanism

| Concept | Construction (/10): How easily or simply the concept can be constructed using Lego | Control (/10): How easily the concept can be integrated into the currently derived equations of motion and control strategy | Friction (/5): Anticipated friction of the concept (friction is undesirable) | Potential (/5): How easily the concept could be adapted to provide additional functionality, such as yaw control | Cost (/10): Does the concept require additional parts beyond those discussed in Section 4.1.1, either standard Lego parts or other parts | Other $\pm 5$ | Total Score |
|---|---|---|---|---|---|---|---|
| 1a (one driven wheel per pair) | Simple<br>10 | Simple<br>10 | High due to friction point and orthogonal wheels<br>0 | Low<br>0 | Low<br>10 | Applied actuation is not a pure torque<br>-2 | 28 |
| 1b (all wheels driven) | Relatively Simple<br>8 | Some difficulty as pairs of wheels must be synchronised<br>8 | High due to friction point and orthogonal wheels<br>0 | Low<br>0 | Requires an additional servo drive (beyond those included in the kit)<br>6 | Applied actuation is a pure torque<br>+2 | 24 |
| 2 | The use of a non-standard Lego part adds difficulty. Asymmetries also add difficulty<br>3 | Simple<br>10 | Low due to use of omniwheels<br>5 | High - location of omniwheels will provide yaw control from the ball<br>5 | Requires the use of omniwheels (non standard Lego part)<br>4 | Applied actuation is not a pure torque<br>-2 | 25 |
| 3a (one driven wheel per pair) | The use of a non-standard Lego part adds difficulty<br>5 | Simple<br>10 | Low due to use of omniwheels<br>5 | Moderate - use of omniwheels allow for easier addition of yaw control from the ball<br>3 | Requires the use of omniwheels (non standard Lego part)<br>4 | Applied actuation is not a pure torque<br>-2 | 25 |
| 3b (all wheels driven) | The use of a non-standard Lego part adds difficulty<br>5 | Simple<br>10 | Low due to use of omniwheels<br>5 | Moderate - use of omniwheels allow for easier addition of yaw control from the ball<br>3 | Requires a forth servo drive and the use of omniwheels (non standard Lego part)<br>0 | Applied actuation is not a pure torque (but lessened)<br>-1 | 22 |
| 4 | The use of a non-standard Lego part adds difficulty. Non orthogonal and angled wheels adds difficulty<br>0 | High difficulty as there must be a conversion between orthogonal torques and the three wheels<br>4 | Low due to use of omniwheels<br>5 | High - location of omniwheels will provide yaw control from the ball<br>5 | Requires the use of omniwheels (non standard Lego part)<br>4 | Applied actuation is not a pure torque (but lessened)<br>-1 | 17 |
| 5 | The use of a non-standard Lego part adds difficulty. Angled wheels add difficulty.<br>3 | Moderate difficulty as pairs of wheels must be synchronised<br>8 | Low due to use of omniwheels<br>5 | High - location of omniwheels will provide yaw control from the ball<br>5 | Requires a forth servo drive and the use of omniwheels (non standard Lego part)<br>0 | Applied actuation is not a pure torque (but lessened)<br>-1 | 20 |

Based upon this evaluation, the concept of using pairs of orthogonal wheels (one driven, one idler) mounted at the ball centre axis was selected.

#### 4.1.2.2 Lego Ballbot Components and Layout

Following the selection of the drive mechanism, the required components for the Lego Ballbot were determined, based upon the selected drive mechanism and anticipated controller requirements. The selected drive mechanism required two servo motors to drive the ball. The controller also required sensors to provide feedback on the Ballbots body and motor shaft angles. The Lego servo motors used include an encoder which provides a measure of the angle of the ball relative to the body, so additional sensors were only required for the body angle. Possible sensors to provide this measurement include gyroscopic sensors and accelerometers, as discussed in Section 2.2.3, both of which are available Lego parts. An additional sensor was also desired to detect the ball's presence, so the controller would only run when the Lego Ballbot is atop the ball. This could be achieved through the use of the Lego touch sensor which would be mounted against the ball to detect its presence. Finally, the Ballbot required the NXT Brick itself to run the controller program.

The best location for each component was also considered during the design. Firstly, the motors were to be located orthogonally, as required for the drive mechanism, and as close to the ball as possible to reduce compliance in axles or possible backlash if gearing is used. The second consideration was the Lego Brick. The best location for the Brick is the top of the Ballbot, as this raises the centre of mass as the Brick would likely account for 25%-50% of the Lego Ballbot's total weight. A higher centre of mass effectively slows the fall of the Ballbot which may allow for easier control. Additionally, locating the NXT brick at the top of the Lego Ballbot provides easy access to the buttons. Finally, sensor location was considered. The location of gyroscopic sensors is not important, however, accelerometers should be located as close to the centre of percussion as possible, to reduce errors caused by body motion. The layout of the Lego Ballbot was designed to satisfy the constraints as closely as possible. This resulted in the concept design for the Lego Ballbot, shown in Figure 12.

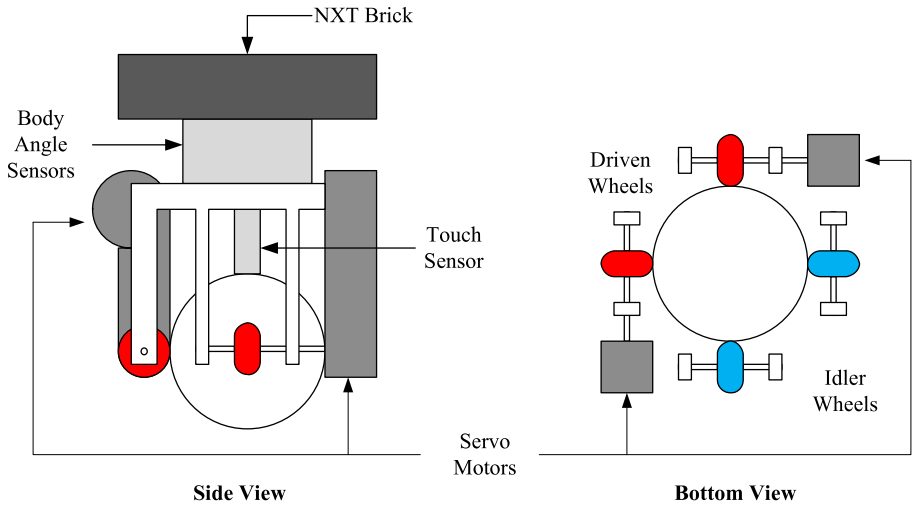



Figure 12: Layout of the Lego Ballbot

### 4.1.3 Construction of the Lego Ballbot

Following the concept design, the construction of the Lego Ballbot was undertaken. Construction was undertaken with the following aims (in order of priority):

- The layout of the Lego Ballbot matches as closely as possible to the concept design proposed in Section 4.1.2
- The Lego Ballbot, where possible, uses only Lego pieces available in the NXT Mindstorms kit, or those additional parts identified as necessary in Section 4.1.1.
- The Lego Ballbot is well balanced, ie: the centre of mass of the Ballbot is directly over the centre of ball when the body is vertical.
- The Lego Ballbot is as rigid as possible, as a flexible structure will result in resonances at low frequencies.

The construction process used a trial and error iterative approach. Each iteration was tested, and issues with the design were identified. The Ballbot was then modified in an attempt to address these issues. This process was made conducive due to the modular nature of Lego. The original Lego Ballbot and following iterations are discussed in the following sections.

#### 4.1.3.1 Original Lego Ballbot

The Original Lego Ballbot, shown in Figure 13, was constructed following the above guidelines. It included direct drive from the servo motors to the wheels and the use of gyroscopic sensors to measure body angular rate. This was believed to provide adequate actuation and measurement to allow control of the ballbot. Additionally, a touch sensor was used to both locate the ball vertically and detect its presence. The key design features of the ballbot are two orthogonal mounted inverted 'U' shapes to form the symmetric drive mechanism, the centrally mounted sensors, and the NXT processor mounted at the top.

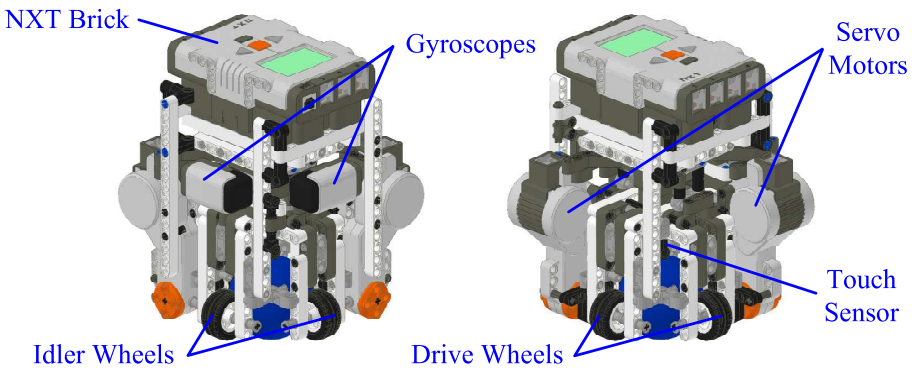



Figure 13: Lego Ballbot, Original

During controller development, several problems were identified with the original design. The controller was not able to stabilise the ballbot, and it was believed this was due to the ballbot falling too fast and a high amount of friction in the drive mechanism. Additionally, a steady state error was observed in the body angle estimate due to gyroscopic drift.

#### 4.1.3.2 Second Iteration

To address the issues identified with the Original Lego Ballbot, several modifications were made, resulting in the second iteration of the Lego Ballbot, shown in Figure 14. In order to slow down the fall of the ballbot, the centre of mass of the ballbot was raised by increasing the overall height. Additionally, gearing was implemented on the drive to allow for faster drive speeds, and an accelerometer was included to help overcome the estimation error from gyroscopic drift. Finally, friction in the drive was reduced through the use of narrower idler wheels.

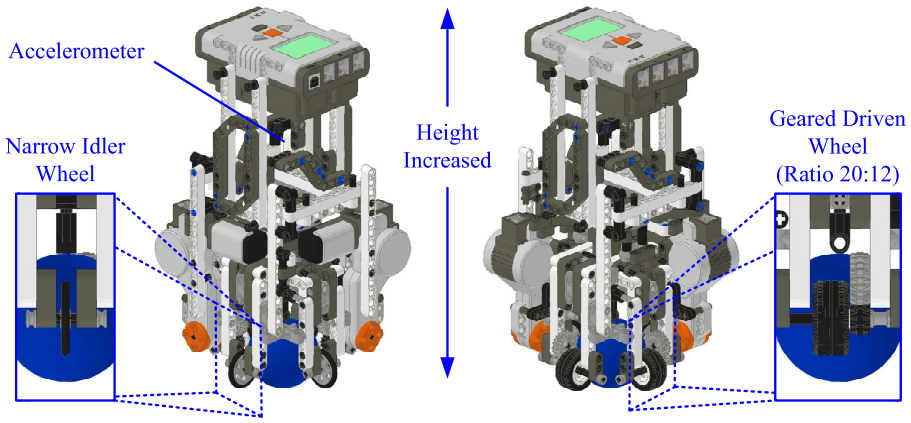



Figure 14: Lego Ballbot, Second Iteration

The second iteration improved performance, but still did not achieve stability. This was identified to be due to the still high levels of friction in the drive mechanism. It was also determined that the gearing to increase drive speed was unnecessary. Although the accelerometer was successful at removing the steady state error, additional errors were caused due to the rapid accelerations of the ballbot.

#### 4.1.3.3 Third Iteration

The third iteration involved a dramatically redesigned drive mechanism to further reduce friction. The idler wheels were completely removed and replaced with a single friction point. This friction point locates the ball and holds it against the drive wheels, which were also changed to slightly larger versions. An optical sensor replaced the touch sensor, as this could be used to detect the ball without requiring contact with the ball, further reducing friction. The gearing of the drive was also removed. It was also determined that the accelerometer was not improving performance, so this was removed. Finally, the frame of the ballbot was simplified and the overall weight lowered, as this reduced the required motor torque. The resulting design is shown in Figure 15.

These changes proved to be adequate in producing a system which could be balanced, and as such this iteration is the final version of the Lego Ballbot. In order to facilitate easy reproduction of the Lego Ballbot, building instructions were desired. Graphical step-by-step instructions have been produced, and are included in Appendix D.

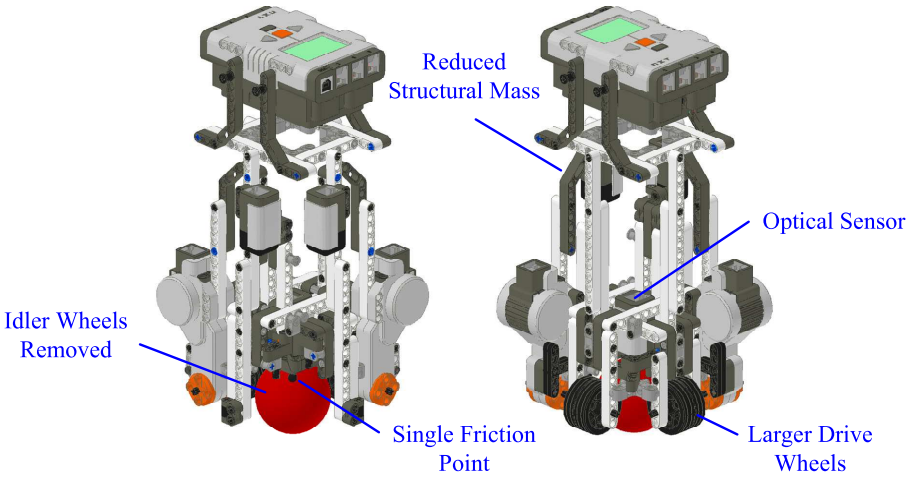



Figure 15: Lego Ballbot, Third Iteration

## 4.2 Controller

State Space control was used within this project, as discussed in Section 3. The theoretical controller is applied to the Lego Ballbot with states shown in Figure 16. Additionally, the derived generalised dynamics are applied to the Lego Ballbot using the physical parameters given in Table 2. This forms the basis for development of the Lego Ballbot State Space Controller.

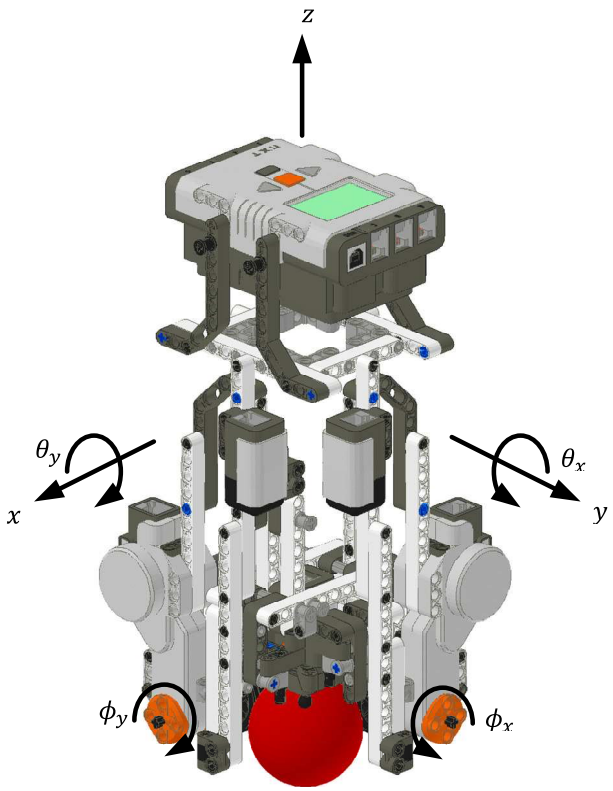



Figure 16: Final Lego Ballbot, showing coordinate system and states

Implementation of the Lego Ballbot Controller was based on the NXTway-GS controller developed by Yamamoto (2008). It was decided that basing the Ballbot Controller on this controller would allow for the rapid development of a controller for the prototype due to similarities between the NXTway-GS and Lego Ballbot systems. This controller is implemented in Simulink and

uses the ECRobot toolbox which was considered an ideal programming platform for rapid development. It was decided that the additional flexibility gained by using other more complicated programming languages available, such as RobotC, would not be required. This section details the implementation of the theoretical controller into software for use on the NXT microprocessor, and simulation results of the developed controller program.

Table 2: Lego Ballbot Physical Parameters

| | | |
|---|---|---|
| Mass of the Body | $M_B$ | 0.767 kg |
| Height of centre of mass | $L$ | 0.132 m |
| Mass of the Ball | $M_b$ | 0.015 kg |
| Radius of the Ball | $R_b$ | 0.026 m |
| Moments of Inertia of the Body | $I_{Bx}$<br>$I_{By}$ | 0.0054 kg.m$^2$<br>0.0054 kg.m$^2$ |
| Moment of Inertia of the Ball | $I_b$ | $6.76 \times 10^{-6}$ kg.m$^2$ |
| Gear Ratio, motor to ball | $n$ | $\frac{-18.6}{26}$ |
| Moment of Inertia of the Motors and Wheels | $I_M$ | $2 \times 10^{-5}$ kg.m$^2$ |
| Friction coefficient between body and ball | $\mu_{Bb}$ | 0.0022 |
| Friction coefficient between body and ground | $\mu_{Bg}$ | 0 |

### 4.2.1 Controller Implementation

The implementation of the state space controller, described in Section 3.1 included the basic setup of the program, the state space controller algorithm, calculation of the reference signal, state estimation, and signal generation for the motors. These processes are described in this section, and the full program can be found in Appendix E.

#### 4.2.1.1 Basic Program Structure

The controller program runs as a simple Finite State Machine (FSM). In an FSM, the program runs in one of a number of states. Within these states, a number of pre-defined tasks are run. Transitions between the states are well-defined, such that the program can only run in one state at a time. The use of an FSM allowed for simple restart of the ballbot without having to restart the entire program. The structure of the FSM designed for the Lego Ballbot controller can be seen in Figure 17.

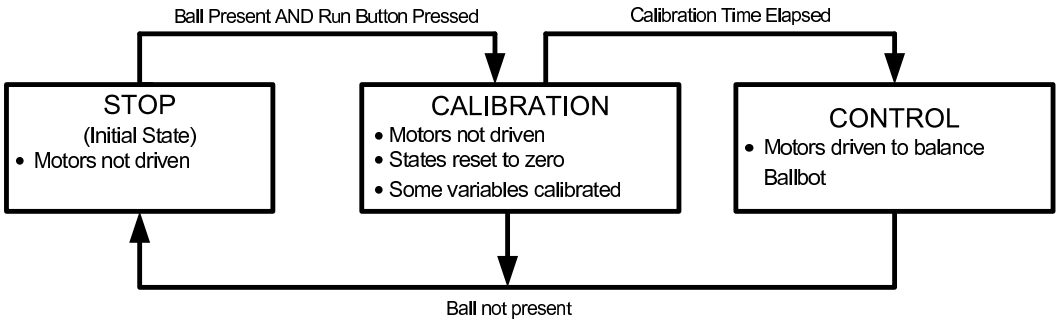



Figure 17: Finite State Machine Implementation

The FSM is implemented using three tasks running at different rates. Firstly, an Initialisation task runs once at program startup and is used to initialise the FSM and required variables. Secondly, a State Check task, running every 100ms, handles the state transitions, based upon the conditions given in Figure 17. Finally, majority of the program is contained with the Balance and Drive Control task, which runs every 4ms. This task includes the calibration procedures used in the Calibrate state, and, most importantly, the state space controller and associated subtasks, which keep the ballbot balancing when in the Control state. As the initialisation and state check tasks are fairly trivial, these will not be discussed further in the report.

#### 4.2.1.2 State Space Controller and Control Gains

The implementation of the theoretical state space controller algorithm is shown in Figure 18. Sensor data is used to estimate the state, which is then subtracted from the command signal, before multiplying by the control gains, $\mathbf{k}$. This produces values for the control voltages for each of the motors which actuate the ballbot. Interfacing with the sensors and motors is simple through the use of the ECRobot NXT Blockset. State estimation, command signal generation and motor signal generation are described in subsequent sections.

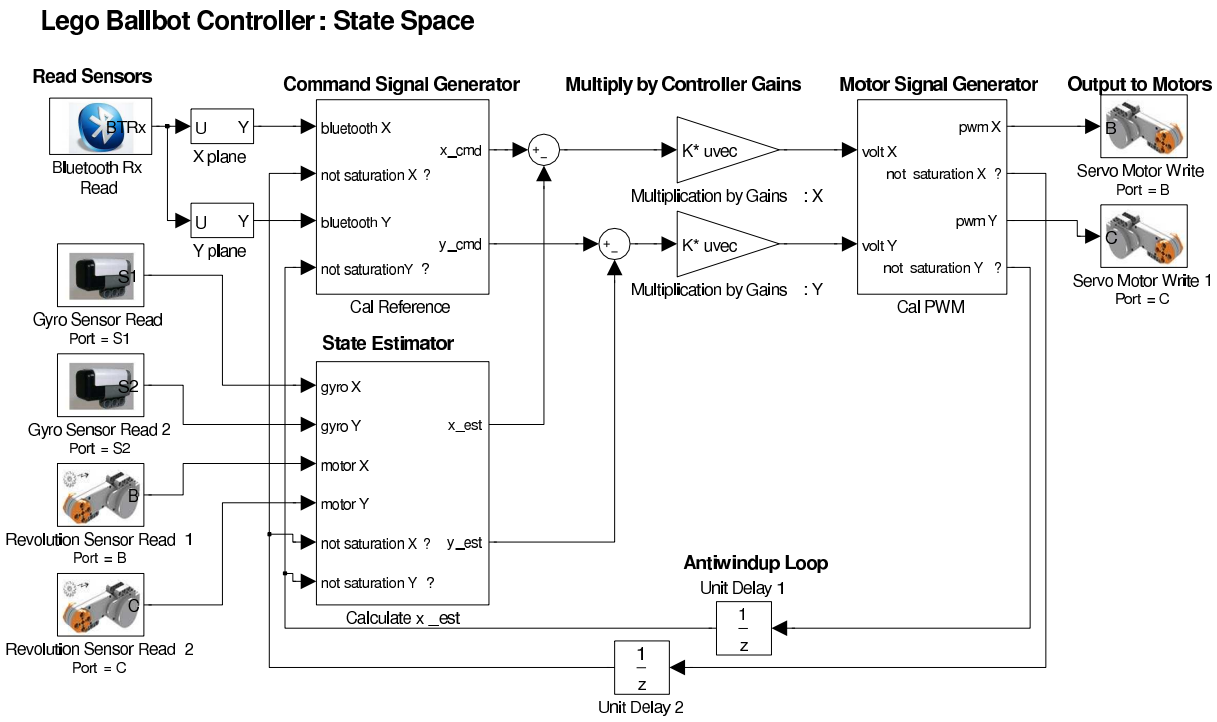



Figure 18: Ballbot State Space Controller

The core of the State Space controller are the control gains $\mathbf{k}$, which were calculated using a Linear Quadratic Regulator. This required the dynamics of the system and the selection of $\mathbf{Q}$ and $\mathbf{R}$ matrices. Dynamics of the Lego Ballbot were found by applying the parameters defined in Table 2 to the general ballbot dynamics derived in Section 3.3. The $\mathbf{Q}$ and $\mathbf{R}$ matrices chosen were based on those used by Yamamoto (2008) in the NXTway-GS controller, due to the similarities between the two systems.

$$\mathbf{Q} = \begin{bmatrix} 60000 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 400 \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} 1000 \end{bmatrix}$$

The Matlab *lqr* command was used to generate the optimal control gains, **k**, based upon the dynamics and **Q** and **R** matrices.

$$\mathbf{k} = \begin{bmatrix} 99.8510 & -1.0181 & 14.9155 & -1.3330 & -0.6325 \end{bmatrix} \tag{17}$$

#### 4.2.1.3 State Estimation

The state space controller requires measured or estimated values for all states used in the control law. The gyroscopes and motor encoders used on the ballbot provide direct measurements of body angle rate and motor angle respectively, and the remaining states must be estimated. The state estimation process for one plane of the ballbot is shown in Figure 19, and is identical for the other plane.

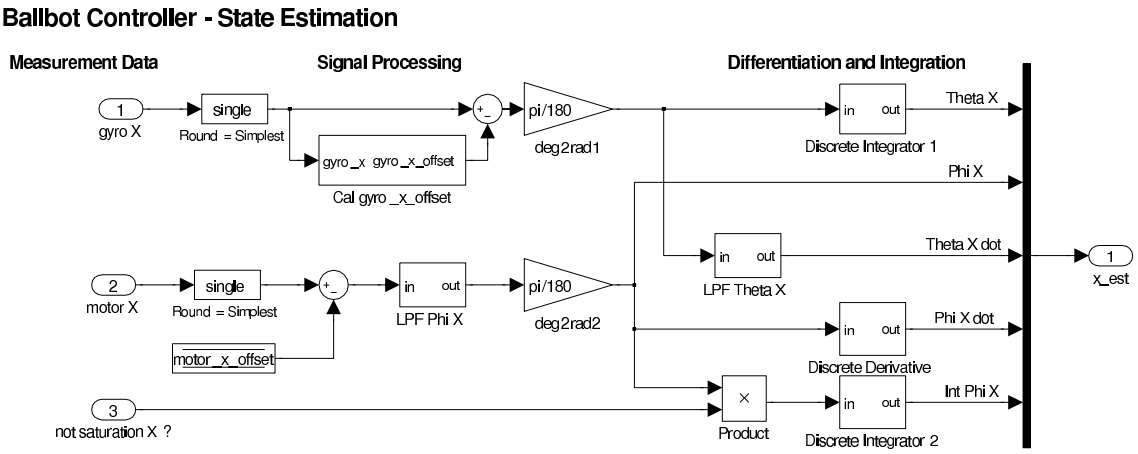



Figure 19: Measurement and Calculation of States, One Plane

The measurement of the body angle, $\theta$, uses two HiTechnic gyroscopes - one for each plane. The gyroscopes measure angular velocity in a single plane, and thus can be used to measure the $\dot{\theta}$ states. The gyroscopes reading is in deg/s, which is converted to rad/s for use in the controller. The $\theta$ states are then calculated simply by integrating the $\dot{\theta}$ states. The motor angle states, $\phi$ are determined by reading the value from the motor encoders. The initial value of the motor from the calibration state, in order to set the zero position to the position at the start of the controller program. The motor encoder programming block returns the angle of the motor shafts in degrees, and must be converted into radians for use within the controller. The $\dot{\phi}$ states are calculated by differentiating these values. The integral states, $\int \phi$, are simply calculated by the integration of the $\phi$ values. This integration is stopped if the actuation signal saturates, in order to prevent windup. Low pass filters are used to smooth the quantised measurement signals, with a time constant of 1.3ms for gyroscopes and 16ms for motor encoders.

A major complexity in the gyroscopes' readings is that each gyroscope has an offset, which must be calculated and subtracted from values measured by the gyroscopes. Each offset is initialised during the Calibration state, using a low pass filter with time constant of 16ms to average of the gyroscope readings during this time. This method is not ideal as it assumes that the ballbot is held still during this calibration stage. However, it was considered better than initialising the value to a constant as each physical gyroscope has a slightly different offset value and can be affected by other inputs and outputs. Additionally, this initialisation would be detrimental to the portability of the controller software. The offset is also continually updated while the program is running, again using a low pass filter with a slower time constant of 4s. This helps reduce the effects of an error during initialisation, but assumes that the long time average of ballbot body angular velocity is equal to zero. This condition should be met if the ballbot is balancing.

During testing it was found that errors were encountered in the estimation for $\theta$ due to changing gyro offsets. To help correct this, an accelerometer was added, seen in the second design iteration of the Lego Ballbot (Figure 14). A complementary filter was used to combine the gyroscope and accelerometer signals to produce the estimate of $\theta$. However, it was determined that errors from the rapid accelerations of the ballbot resulted meant that performance was not improved. As such the accelerometer and filter were removed from the program. Complementary filters are discussed in more detail in Section 5.2.1 as one was also implemented on the Full Scale systems initial controller.

#### 4.2.1.4 Command Signal Generation

The purpose of the Command Signal Generator is to produce the desired state of the Ballbot. Thus, the controller allows for command tracking by variation of the command signal. For the purposes of the Lego Ballbot, it was decided that position command tracking would be implemented, using ramp inputs to the $\phi$ command, with slope specified by the user. A limit on the maximum slope of the ramp was imposed, following tests to determine the maximum speed of the Ballbot.

Bluetooth signals are used to specify the slope of the position command ramps for each plane, and as such allows a user to remotely drive the Lego Ballbot forward, backward, left and right. The Bluetooth signals are produced by a computer using the NXTGamePad program developed by Yamamoto (2008) and are read in the controller program using the ECRobot Bluetooth programming block.

In addition to generating the $\phi$ command state, command values for the other states were required. Command values for the $\int \phi$ states were generated by integration of the $\phi$ command. This integration is stopped if the actuation signal saturates, in order to prevent windup. The command signal for $\dot{\phi}$ was set to zero. While this could in fact be set to the slope of the position ramp, it was determined that the zero setting resulted in better performance. Likewise commands for the $\theta$ and $\dot{\theta}$ command states are simply given the value zero, in order to keep the ballbot upright while moving.

#### 4.2.1.5 Motor Signal Generation

The motors are actuated using a Pulse Width Modulated (PWM) signal generated by the ECRobot NXT Blockset servo motor programming block, which takes a duty cycle parameter between -100 and 100. The control signal generated by the state space controller is converted to a duty cycle by linearly scales the control signal by the current battery voltage to produce the duty cycle parameter, limited to between -100 and 100. The maximum battery voltage is calculated using an experimental function developed by Yamamoto (2008). This process is identical for the two planes of the ballbot, and is shown in Figure 20. Additionally, the saturation limits of the motors are monitored as required for the anti-windup loops described in Sections 4.2.1.3 and 4.2.1.4.

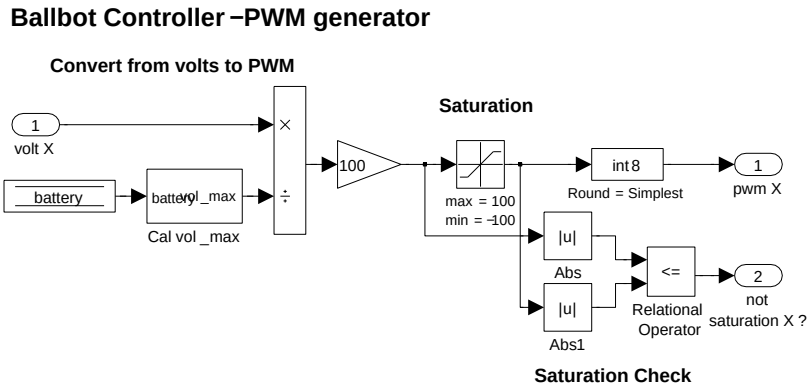



Figure 20: Motor Signal Generation - One Plane

### 4.2.2 Simulation

In order to validate the controller program and control gains, a simulator was created in Simulink. This simulator consists of two parts, the controller itself, and a model of the ballbot system based upon the non-linear dynamics derived in Section 3.3. The controller operates identical to the control state of the implemented controller program, including state estimation, state space control law, and motor signal generation. This part of the simulator runs at a sample time of 4ms, like the real life system. The model of the ballbot uses a discrete time version of the non-linear dynamics running at 0.1ms, and also includes models of the actuators and sensors, including any quantisation and offsets present in the real life system.

The simulation results for two cases are presented in Figure 21 and Figure 22. The first is balancing from an initial angle of 3 degrees in body angle, simulating a disturbance. The second is a command tracking example where the ballbot is commanded to move forward at 7.4cm/s for 8 seconds. The simulation results presented in this report are for one plane only, as the simulated controller operates identically in the two planes.

The results show that the ballbot system is stabilised from the initial disturbance and is capable of following the command given. This suggests that the controller is capable of controlling the ballbot, assuming the derived equations of motion adequately model the ballbot's dynamics. Additionally, it can be seen that the state estimation of body and motor angle provides relatively accurate estimates of the states, but some errors are present due to quantisation and delay from low pass filters. This is the likely cause of the oscillations present at the steady state. It is also

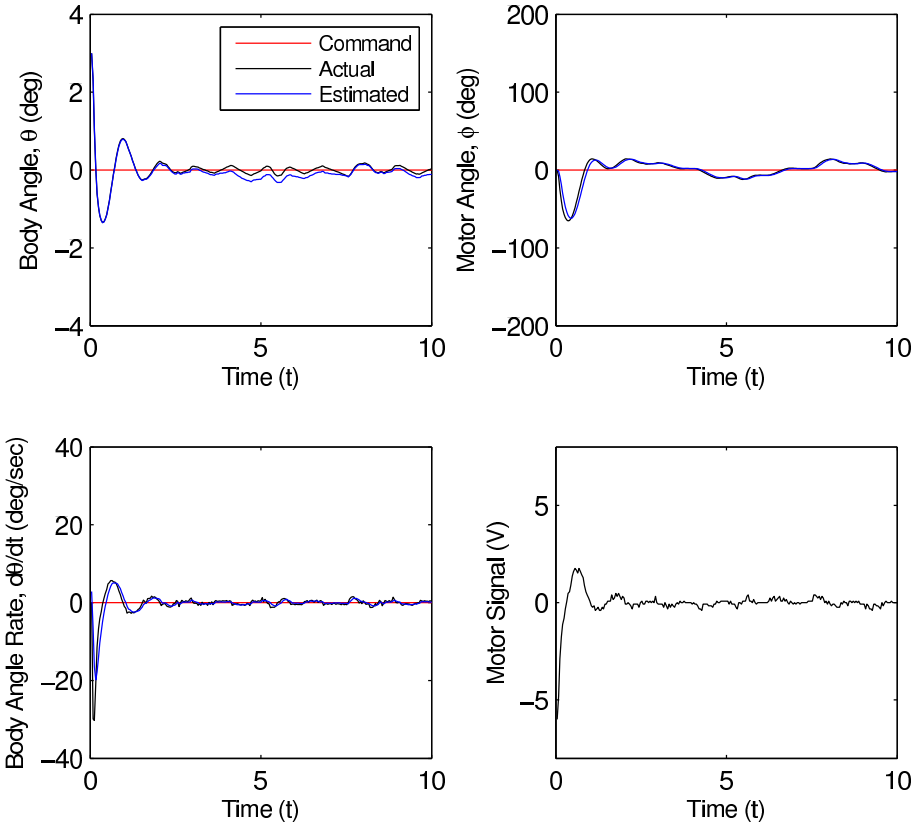



Figure 21: Controller Simulation Results - Balancing

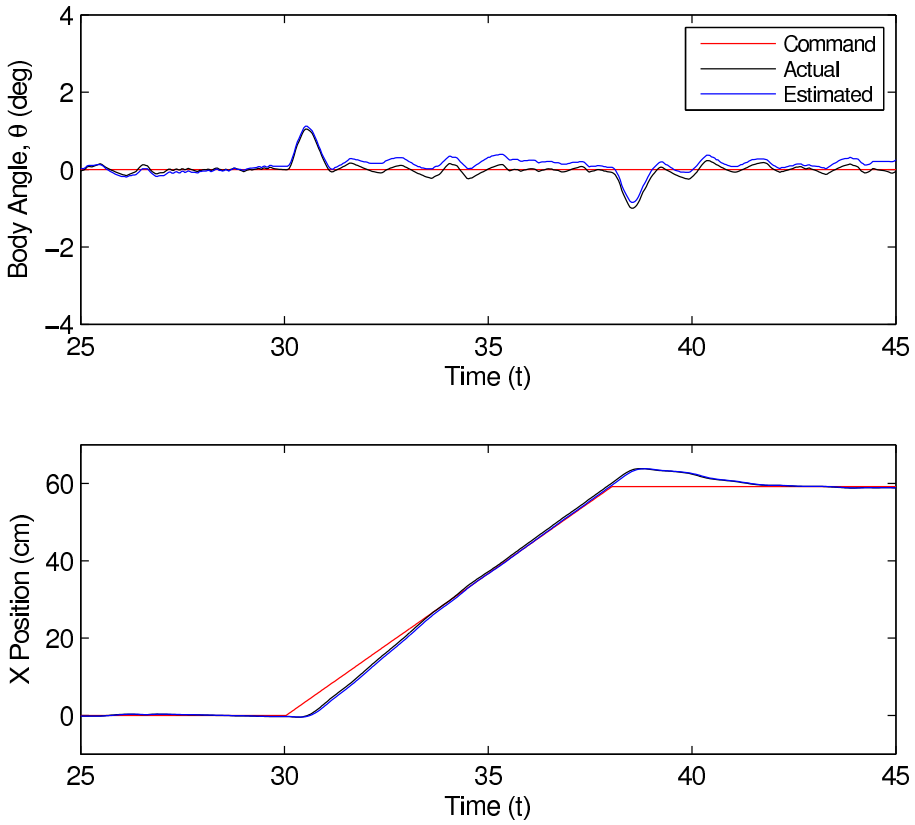



Figure 22: Controller Simulation Results - Command Tracking

important to note that the motor signal does not saturate (at $\pm$8V) at any time, indicating that the motors are capable of producing the required torque and velocity. Additionally, the gyro reading

is well within its saturation limits of $\pm 360$ deg/s, indicating the gyro will adequately measure the body angle rate.

#### 4.2.2.1 Virtual Reality Model

In addition to simply viewing logs of results, the simulator was used to produce a virtual reality (VR) model of the Ballbot. This uses a CAD model of the Lego Ballbot, and animates the model in real time based upon the calculated state information produced by the simulator. The VR model is useful as it provides better visual feedback of the system, and also allows the user to interact with the simulator in real time, making it an additional teaching aide. A screenshot of the VR model is shown in Figure 23, and the complete program code can be viewed in Appendix F.

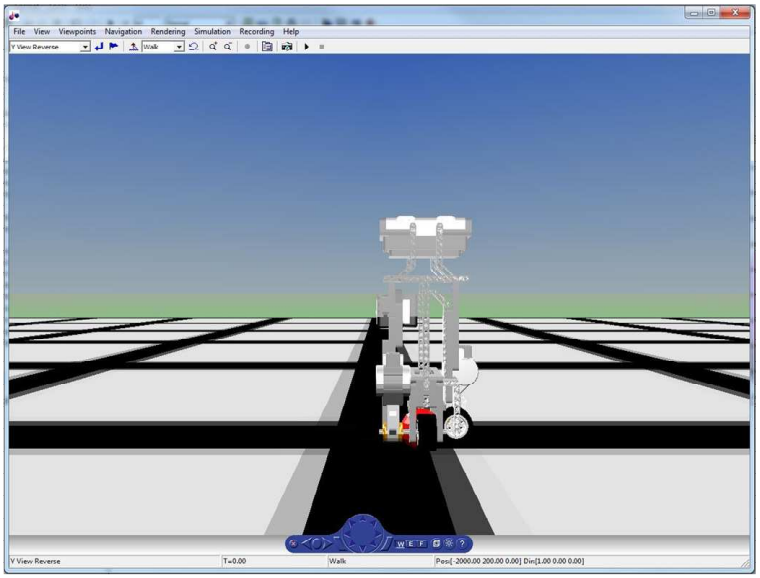



Figure 23: Virtual Reality Model

### 4.3 Testing Procedure

The aim of testing was to determine the feedback gains that resulted in the desired performance of the Ballbot. The gains determined using the LQR in Section 4.2.1.2 provide the starting point for the gains, and these were manually tuned until the Ballbot's performance was optimised.

Each set of gains was tested by simply running the controller program with those gains and assessing performance. The controller program was run simply by holding the ballbot upright on the ball, starting the program, and then releasing the ballbot at the end of calibration, indicated by an audible tone. A potential issue with this method was that a disturbance occurs when the ballbot is released, however this was generally found to be negligible. Testing of the ballbot was performed on a rubber mat to ensure adequate grip at the ball ground interface as well as consistency between tests. Finally, no safety measures were deemed necessary during the testing process due to the small size and light weight of the Lego Ballbot, as well as the inherent safety of Lego, which is designed for use by children.

Performance of the ballbot was assessed in two ways. Firstly, basic performance such as balancing could be assessed visually, and this formed the predominant method of assessment while

attempting to achieve this goal. Secondly, the Bluetooth connection of the Lego NXT processor was used to produce a log of the Ballbot's states and command signals. This gave additional insight into the performance of the Ballbot, such as settling times and the magnitude of oscillations. This was useful for fine tuning gains and optimising performance not only for balancing, but also for disturbance rejection and command tracking.

The test procedure resulted in the determination of the control gains. It was noted that different sets of gains gave better results in different areas. For example, the set of gains that gave best disturbance rejection differed from the set that gave best command tracking. The following set of gains being determined to give best overall performance.

$$\mathbf{k} = \begin{bmatrix} 100 & -1.2 & 15 & -1 & -0.3 \end{bmatrix} \tag{18}$$

These gains are similar to those calculated by the LQR method. This may be attributed to having a good model of the dynamics or a good choice of the **Q** and **R** matrices, however, may be just a coincidence.

## 4.4 Results and Analysis

When implemented on the Final Lego Ballbot with the set of feedback gains found in Section 4.3, the controller is successful at balancing the Ballbot and rejecting small disturbances. Furthermore command tracking and the use of remote control was achieved. These features are displayed in the following results. Additionally, the results are compared to the simulation results presented in Section 4.2.2.

### 4.4.1 Balancing from Release

The primary task of the controller is to balance the Ballbot. The results of a test in which the Lego Ballbot attempted to balance over the initial position can be seen in Figure 24. The plot shows the ballbot initially moving away from its initial position, before settling back to zero in about 20 seconds. This movement is primarily a result of offsets present in the gyroscope readings. Filters are used to attempt to remove this offset, as described in Section 4.2.1.3. However, a change in battery voltage at the beginning of operation causes the offset to change from its calibrated value. As a result, errors accumulate in the body angle estimate until the filter re-converges to the correct offset value in about 5 seconds. In the example shown this error accumulates to approximately 4 degrees before the filter re-converges. As such, when the body is upright the body angle estimate will be 4 degrees. The steady state error that would normally result from this error is eventually removed by the integral of motor angle state.

Once steady state is achieved, the results are quite similar to those found in the simulation discussed in Section 4.2.2. This suggests that the dynamics derived in Section 3.3, used in the simulator, provide an accurate model of the ballbot system. A limit cycle is entered, like in the simulations, where small oscillations are present at steady state. This is likely due to quantisation in the measured data causing small errors in the state estimate, as in the simulator. The slight differences from the simulation results are likely due to either the variation in gains from the simu-

lator or influences in the real life system which are not modelled in the simulator, such as backlash or deadzone in the motors.

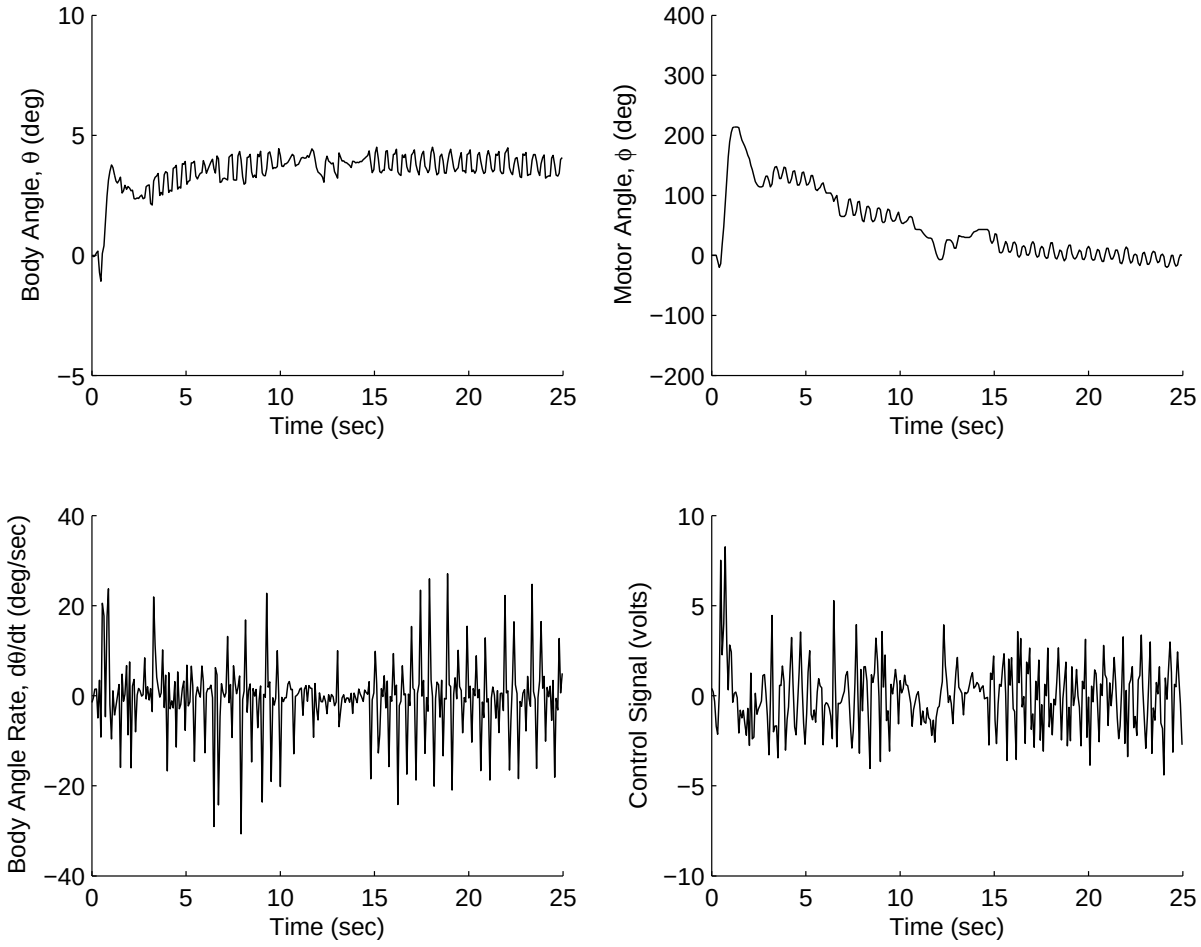



Figure 24: Lego Ballbot - Balancing

### 4.4.2 Disturbance Rejection

The controller is also capable of rejecting small disturbances. Such disturbances include a small push or an uneven surface. The primary limitation on disturbance rejection are the maximum torques and speeds that can be produced by the motors, however, the non-linear nature of the system also reduces controller performance as body angle moves away from zero degrees. Figure 25 shows an example where the Lego Ballbot has reached steady state, and is then given a small push at the 4 second mark. Similar to the previous results, only one plane is shown, and this is the plane in which the disturbance was applied. The plot shows that the ballbot has successfully remained upright when subjected to the push. The disturbance resulted in a maximum deviation from upright of approximately 3 degrees and a corresponding motor angle deviation of approximately 150 degrees. Following the push, the ballbot takes approximately 3 seconds to settle back to its initial position. Once again, this is similar to the simulation, where the ballbot stabilises from a 3 degree initial position in about 3 seconds. It should be noted that differences exist between the simulation and physical tests. In particular, the $\phi$ states at maximum body angle are zero in the simulation, but non-zero during the physical tests. As such, direct comparisons are not entirely accurate.

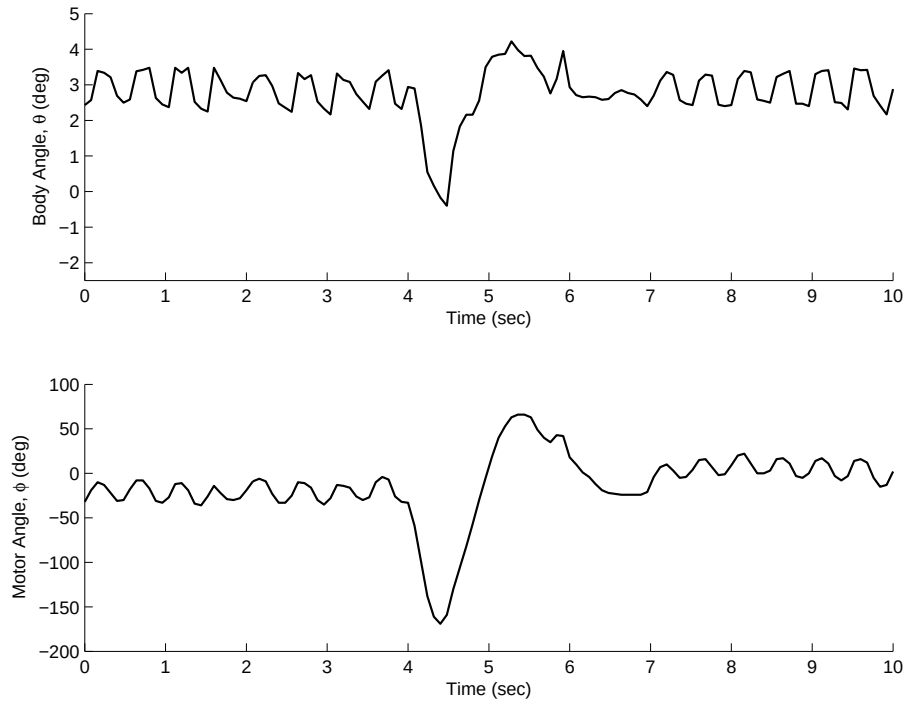


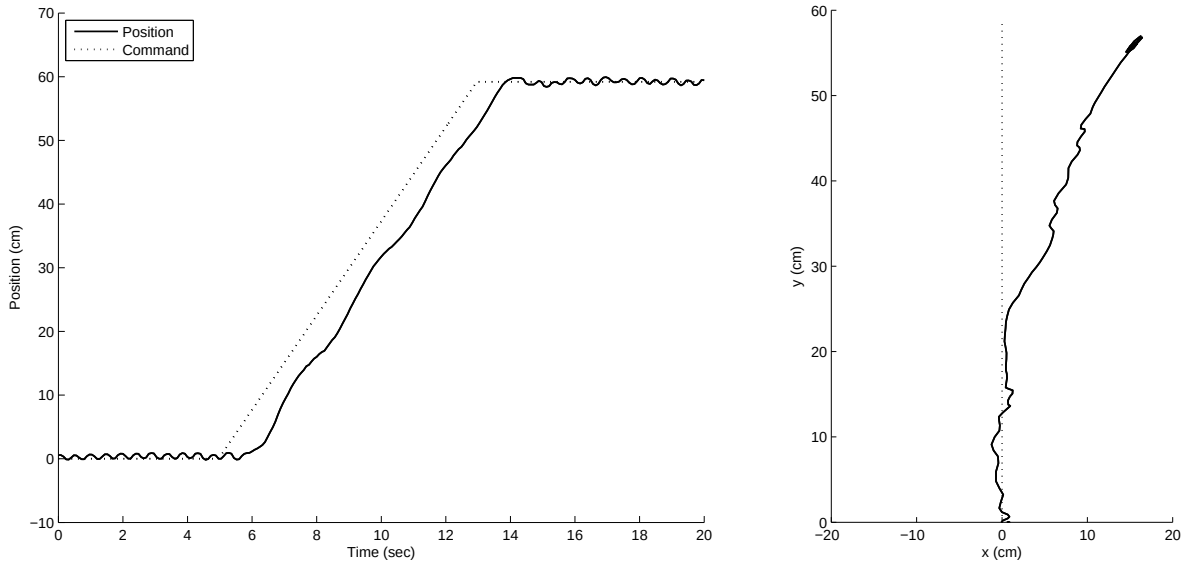

Figure 25: Lego Ballbot - Disturbance Rejection

### 4.4.3 Command Tracking and Remote Control

The final feature of the controller is the ability to command track, or follow user directions. Command tracking is implemented such that the command is generated from a Bluetooth signal. As such, remote control of the Ballbot is possible using a handheld controller connected to a computer which produces the Bluetooth signal. The Ballbot is capable of command tracking in each plane, and the orthogonal nature of the two planes of the Ballbot means these movements can be combined such that the ballbot can travel in any direction. An example of command tracking is shown in Figure 26(a), where the Ballbot is commanded to move forward in the Y direction at 7.4cm/s for 8 sec. This speed was determined to be close to the maximum speed at which the Ballbot could reliably command track, being limited by motor performance. The plot shows that the ballbot smoothly follows the command, with a lag of approximately 1 second which results in the Ballbot being 7.4 cm behind the command location while travelling. No significant overshoot is present beyond the normal limit cycle. However, rotation about the Ballbot's vertical axis was observed while command tracking, which is a result of the asymmetric drive system. A compass sensor was used to monitor this rotation and determine the ballbots actual trajectory while command tracking, and this is shown in Figure 26(b). It can be seen that the Ballbot does not travel in a straight line, rather, an arc motion has resulted. This suggests the need for some form of yaw control to maintain a constant heading.

## 4.5 Yaw Control

As discussed in Section 4.4.3, it was desired to modify the Lego Ballbot so that some form of control over the yaw rotation is achieved. Two potential methods of addressing the yaw rotation issue were considered. The first simply involved using a compass sensor to monitor the yaw rotation and alter the command signal such that the ballbot travels in a straight line, without actually




(a) Y plane (b) Top Down View

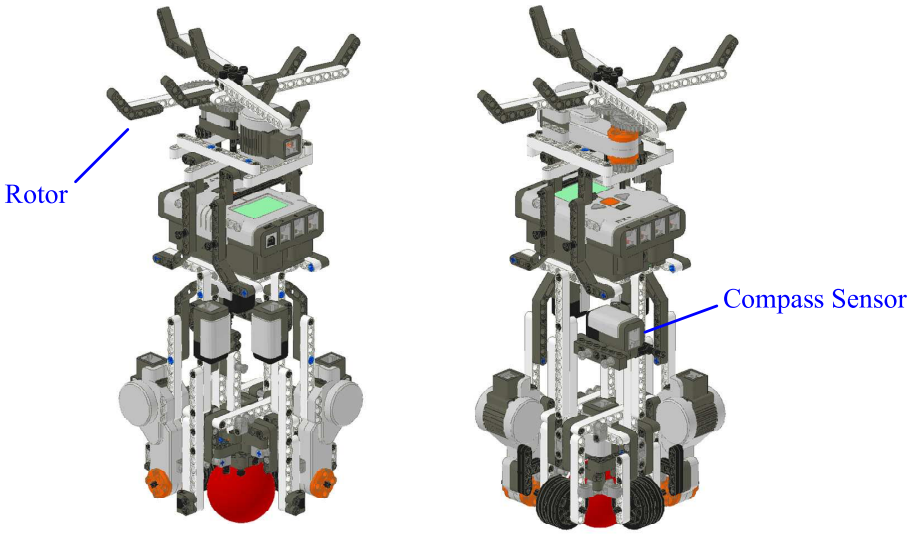
Figure 26: Lego Ballbot - Command Tracking

preventing the rotation. The second method involved the addition of an actuator which can rotate the ballbot around its vertical axis, which can be used in conjunction with the compass sensor to apply closed loop control of the yaw rotation. The second method is selected for use, due to its higher reliability, however comes at the cost of an additional actuator. The implementation of the actuator and corresponding control loop is discussed in the following sections.

### 4.5.1 Actuator

The purpose of the actuator is to cause yaw rotation of the ballbot. Two methods of achieving this were determined. The first involves yaw actuation of the ball, effectively changing the drive mechanism of the ball from two orthogonal directions to three. However, this would involve a much more complicated drive arrangement and likely have high friction. This may have negative effects on the controller performance, and as such is an undesirable method of yaw actuation. The second possibility involves the use of a large rotor which causes a torque on the Ballbot due conservation of rotational momentum and air resistance. This method is simpler to implement and does not significantly influence the performance of the balancing controller. For these reasons this method was selected as the method of yaw actuation.

The modified Lego Ballbot, showing the addition of the rotor is shown in Figure 27. An additional servo motor as been mounted above the NXT processor to drive the rotor. Testing showed that the rotor was capable of causing yaw rotation of the Lego Ballbot as desired, however the results are highly dependent upon the surface the ballbot is running on. It was also determined that rapid changes of rotor speed caused instability in the ballbot, due to the high non-linear torques produced by the change in rotational momentum. To address this problem a low pass filter with a time constant of 4 sec was placed on the voltage signal to the motor, modifying the motor response such that the rapid change in rotor speed did not occur.




NOTE: Rotor has paper blades (not shown) to increase air resistance

Figure 27: Modified Lego Ballbot, with rotor

### 4.5.2 Controller

Closed loop control of yaw rotation is implemented by considering the rotation as a Single Input Single Output (SISO) system. The control input is the voltage of the servo that drives the rotor, and the output is the rotation angle, measured by the compass sensor. The compass sensor reading was low pass filtered, with a time constant of 1s, to reduce measurement noise. Closed loop control was achieved using a Proportional Integral Derivative (PID) controller structure, shown in Figure 28.

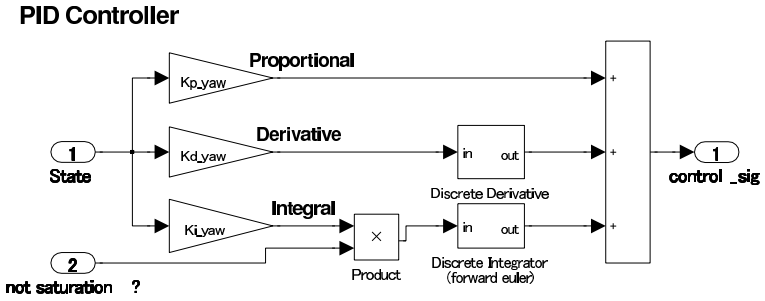



Figure 28: PID controller structure

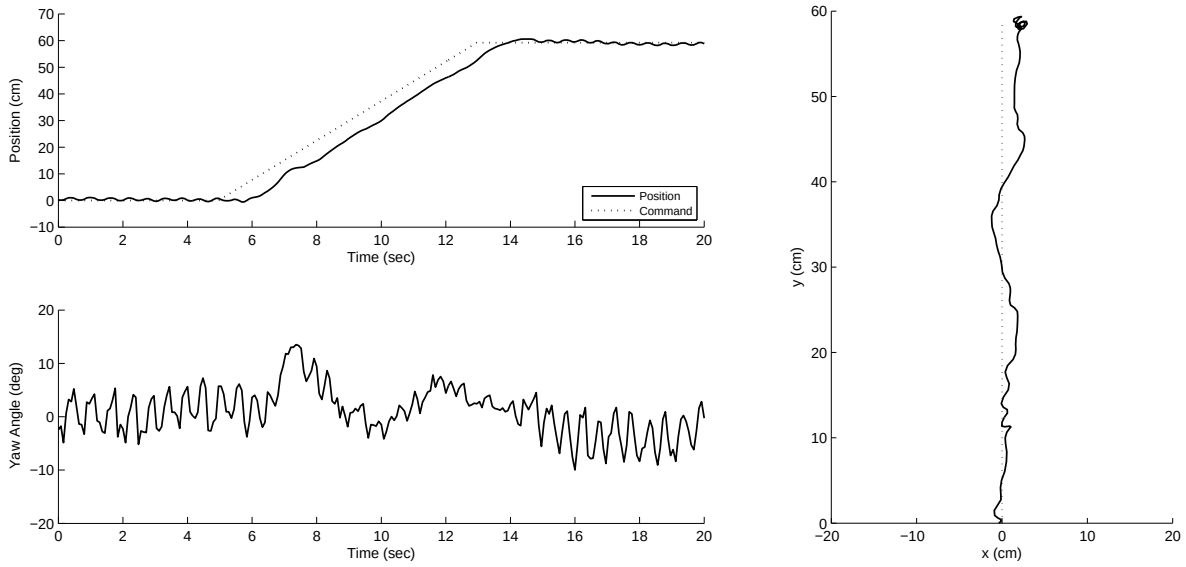
The control gains were initially determined using the Ziegler Nichols step response tuning method, and then manually tuned to achieve the desired performance. The final gains are

$$\begin{bmatrix} K_{p,yaw} & K_{i,yaw} & K_{d,yaw} \end{bmatrix} = \begin{bmatrix} 2.0579 & 0.0706 & 1.8748 \end{bmatrix} \tag{19}$$

### 4.5.3 Results

The primary aim of the yaw controller was to regulate heading when the Ballbot is travelling. The yaw controller's performance is shown in Figure 29(a), where the command tracking example of Section 4.4.3 is repeated with the yaw controller implemented. It can be seen that the yaw controller has successfully regulated the Ballbot's yaw to within 10 degrees of the initial heading, without causing adverse effects on the previous command tracking capabilities. The resulting

movement in space is also shown in Figure 29(b). This plot shows that the addition of the yaw controller has resulted in straighter movement. Additionally, the yaw controller is capable of rejecting disturbances in yaw, as well as remote control command tracking of the yaw angle. However these results are not presented within this report.




(a) (b)

Figure 29: Command Tracking, with Yaw Controller

# 5 The Full Scale Ballbot

The Full Scale Ballbot was the second Ballbot successfully constructed within this project. The Full Scale Ballbot aimed to apply the experience gained from producing the Lego Ballbot to the creation of a Ballbot of appropriate size to interact with humans.

The first phase of the Full Scale Ballbot was the design of the hardware components. This was followed by the adaptation of the existing Lego Ballbot controller for the Full Scale System. Due to problems encountered during testing of the Full Scale Ballbot at this stage, the sensors and controller hardware were replaced. Thus, the controller program was rewritten for this new controller and sensors. Testing was then carried out to confirm the success of the revised Full Scale Ballbot.

## 5.1 Hardware Design

The design of the Full Scale Ballbot primarily involved developing a design which replicates the functionality of the Lego Ballbot. In addition to this the design was governed by aesthetics and simplicity. The aesthetics of the Full Scale Ballbot were important due to its proposed use as a promotional tool for the University of Adelaide. Additionally, simplicity was essential to ensure successful construction of the Ballbot in the limited time frame available. Further considerations were also given to the cost of Ballbot, given the limited budget. Additionally, a short four-month time frame was imposed by the Open Day deadline, which reinforced the requirement for design choices which could be rapidly implemented.

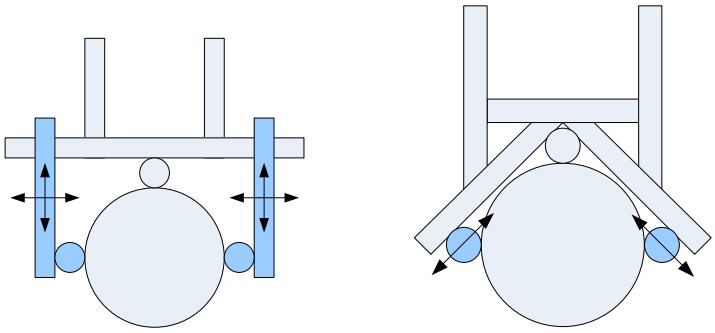
The design was developed by first considering conceptual options and general design of the Full Scale Ballbot. Based on this, the components required to achieve the design were selected. Finally, the detailed design was produced with the knowledge of these components. Detailed design included structural design of the frame and mounts for each of the components, as well as the electrical design of interfaces between electrical components.

### 5.1.1 Conceptual Design

The concept design for the Full Scale Ballbot was based upon the Lego Ballbot, which consisted of a tall thin body above the drive mechanism. This concept was repeated for the Full Scale Ballbot. Additionally, it was proposed that the column include adjustable shelves onto which components can be mounted.

The drive mechanism proposed for the Full Scale Ballbot was an arrangement similar to the 'Inverse-Mouse Ball Drive.' This allowed for a simpler design, as well as providing similarities to the Lego Ballbot which aided in controller implementation. However, to enhance flexibility, it was also considered desirable to allow for different ball sizes. Two main options were considered to implement this, as shown in Figure 30.

Concept 1 involves the construction of a square drive frame, with a larger base and a horizontal cross. Motors and wheels are mounted on adjustable vertical beams from the horizontal



(a) Concept 1 (b) Concept 2

Figure 30: Concepts for the Full Scale Drive Mechanism Frame (shown in one plane)

cross. This design is highly flexible due to all dimensions being adjustable. The design, however, requires much framing material and visually appears very bulky and bottom-heavy.

Concept 2 involves the drive mechanism supports being oriented at 45°. The wheels and motors are mounted on these supports and as such can be adjusted along the axis of the frame, allowing for any ball size. This concept requires less framework than concept 1 and has better structural characteristics, however the drive wheels can only be adjusted along one axis.

An additional advantage with both drive frame concepts is that they provide the option of actuating the ball at locations other than the side of the ball. This gives extra flexibility, and the facility to test and utilise a drive mechanism similar to that proposed by Wu and Hwang (2008), as discussed in Section 2.2.1, which removes the requirement for the top roller. Such a design would require the use of omniwheels, but does not require the simultaneous high and low friction coefficients as required in the 'Inverse-Mouse Ball Drive.'

The 45° drive mechanism frame was selected for the Full Scale Ballbot, as it is simpler to design and implement, and provides a more elegant design aesthetically. It was also decided that the primary location of the drive wheels would be atop the ball, not at the sides. As previously stated, this removes the need for the top roller, but requires the use of omniwheels.

### 5.1.2 Component Specification and Selection

The functional components required for the Full Scale Ballbot are the same as those required for the Lego Ballbot - that is, a controller, tilt sensors and motors. Unlike the Lego Ballbot however, these components must be sourced from a number of vendors, as many options exist for each of these components. In addition to this, other components such as the power supply, ball and wheels must also be selected and sourced.

#### 5.1.2.1 Controller Hardware

The requirements for the controller for the Full Scale ballbot are identical to those for the Lego Ballbot. As such, the Lego NXT Mindstorms brick was selected to be used as the controller for the Full Scale Ballbot, due to its suitability, the project team's experience with the controller and low

cost. Critically, the use of this controller also allowed for fast development of a controller program for the Full Scale Ballbot, simply by modification of the Lego Ballbot's controller program.

#### 5.1.2.2 Tilt Sensors

Due to the selection of the NXT Mindstorms Brick as the controller hardware, the simplest option for the tilt sensors was to use the HiTechnic Gyroscopes discussed in Section 4.1.1. The use of these components ensured that minimal modifications would be required to the software controller to estimate the tilt of the Ballbot. However, testing of the Lego Ballbot revealed that errors due to gyroscopic drift exist when using these sensors. While this did not prevent the Lego Ballbot's success, the error is still undesirable. As such it was proposed that the Full Scale Ballbot also include a HiTechnic 3-Axis Accelerometer to provide a more absolute measure of angle in order to attempt to address the drift error. This sensor did experience errors when used on the Lego Ballbot, due to accelerations other than gravity. However, it was anticipated these errors would be lower on the Full Scale Ballbot as it is a slower system, and as such experiences lower accelerations.

One further consideration was the possibility of errors in the readings due to saturation or the resolution of the sensors. However, it was felt that due to the tests conducted on the Lego Ballbot that the sensors would be adequate for the Full Scale Ballbot, again because it is a slower system.

#### 5.1.2.3 Motors

Selection of motors was based on three main criteria. The first was the desired performance of the Full Scale Ballbot, and whether the motors could produce the required control authority to achieve a stable ballbot. This was analysed via simulation, using motor characteristics from available data sheets and approximated Ballbot physical characteristics. The second criteria was the physical size and shape of the motors, and the ability to incorporate the motor in an aesthetically-pleasing matter in the design. The third criteria was cost and availability. Motor selection was limited to Brushed DC Servo motors, as these were considered preferable due to their similarity to the Lego NXT Servo Motors, their ease of control compared to brushless and other motors, and their relatively low cost.

After considering many possible options, the motors selected for the Full Scale Ballbot were Leadshine DCM50205 DC brushed servo motors. These are 24V, 125W motors and provide a rated torque of 219 mNm at 3000rpm, and 1.59 Nm at stall (see Appendix G). Simulation showed that these motors were capable of providing enough torque for a stable ballbot, however it was apparent that more torque would result in better performance. For this reason it was decided that all wheels would be driven. This effectively doubles the maximum possible torque, but requires two additional motors. This option was considered preferable to using a gearbox to increase torque, as a gearbox would impose a lower maximum motor speed and also result in significantly higher cost. The use of two motors in each plane also results in a more balanced ballbot, both physically and aesthetically. Finally, the motors were available in Australia from Ocean Controls, resulting in a very short delivery time.

The selected motors were supplied with built-in encoders. The encoders require a 5V supply and produce a 500 count per turn quadrature encoder signal. The Lego NXT Servo motor encoders also produce a quadrature encoder signal, and as such this encoder signal could be directly read by the NXT brick with only minor alteration to the controller program. A potential disadvantage considered was the possibility of saturation of this encoder signal due to excessive speed of the motor shaft. However, it was considered unlikely that this speed be reached, and the effects of this happening were not considered critical to the success of the project.

#### 5.1.2.4 Power Supply

A power supply is required on the Ballbot to power the motors and other electronics. The Full Scale Ballbot is a mobile robot, and thus it is desirable that it use an on-board power source. Batteries are the simplest and cheapest sources of portable energy, and thus were chosen as the power supply. Sealed Lead Acid (SLA) batteries were selected due to their relatively low cost and large capacities compared to other battery technologies such as Lithium-Polymer (LiPo). They also may be used at any orientation, which is important due to possible oscillation in the ballbot body.

The batteries selected for use in the Full Scale Ballbot were 12V SLA batteries available at Jaycar Electronics. Two batteries were selected to be used in series to provide the 24V required for the motors. A range of battery capacities is available, with larger capacity batteries being larger and heavier. The 18Ah version was selected, as this was the largest capacity battery that met the dimensional requirements. Two adjacent batteries have total dimensions 181x154x167mm with a total weight of 11.8kg, and thus fit within the frame of the Ballbot, as required.

#### 5.1.2.5 Frame Material

The frame of the Full Scale Ballbot includes the drive mechanism framing and the column. Key criteria for the frame material were that it be lightweight and strong enough to support all components. It was also desired that the frame material allow for ease of adjustability to provide the features discussed in Section 5.1.1. Furthermore, the frame is perhaps the most important component of the Ballbot in terms of aesthetics, and for this reason the appearance of the frame material was also considered a significant criteria.

Maytec Aluminium Extrusion was chosen to be used for the frame, provided by Metri-Tek Pty Ltd. This material is a product which is designed for frames, and thus can easily structurally accommodate the features used on the Ballbot while providing high strength and low weight. Additionally, the neat appearance of this material was deemed suitable for the Full Scale Ballbot Design. Furthermore, the modular nature and large variety of interacting components allowed for a robust and tailored design, which could be assembled with minimal use of the University of Adelaide Mechanical Engineering Workshop. This was considered advantageous due to the possibility of lengthy construction delays should the workshop be used.

#### 5.1.2.6 Ball

The ball used in the Ballbot must fulfil a number of design criteria. First, the ball surface must have a high friction coefficient to eliminate slip between the ball and the actuating wheels, as well

as between the ball and the ground. Slip at these interfaces have a detrimental effect on performance the Full Scale Ballbot. Additionally, the ball must be rigid, such that unnecessary damping is not introduced into the system, which would also reduce the performance of the Ballbot.

The Full Scale Ballbot has been designed to accommodate balls of varying sizes, allowing for a selection of balls to be used. However, for design purposes it was proposed that the ball be nominally 220mm. The ball selected was a bowling ball covered in 'Plasti Dip' - a rubber coating. The bowling ball provides a hard surface and thus the rigid property required. The thin layer of rubber provides a high friction coefficient without significantly reducing this rigidity. This approach was successfully used by Kumagai and Ochiai (2008) in the construction of the TGU Ballbot (discussed in Section 2.2.2, which required similar properties to that required by the Full Scale Ballbot designed in this project.

#### 5.1.2.7 Wheels

Due to the nature of the proposed driving mechanism of the Full Scale Ballbot, standard uniplanar wheels would not be suitable for any arrangement other than the traditional side-actuated inverse-mouse ball drive (as shown in Figure 4). Thus, omniwheels, which allow for slip in the axial direction, were sought for the design. Fabrication of omniwheels was considered, however was deemed too expensive and labour-intensive to be feasible for this project.

Two suitable omniwheels were considered for this project, both produced by Kornylak Corporation (see Figure 31). One candidate was the 2570 Omni Wheel, which feature three rollers and must be used as a pair to provide full motion. These were considered due to their aesthetically pleasing design. The second candidate was the 2052-3/8 Multiple Row Transwheel with Cat-Trak Rollers. These were considered as the rollers provide higher friction, which may result in better performance. Datasheets for each wheel can be viewed in Appendix G. Multiple Row Transwheel were selected for use on the Full Scale Ballbot, as this wheel provided higher friction and allowed for a simpler motor bracket design.

(a) 2570 Omni Wheels (Corporation, 2007)

(b) 2052-3/8 Transwheel (Corporation, 2007)

Figure 31: Omniwheel Candidates

### 5.1.3 Structural Design

Structural design of the Full Scale Ballbot included detail design of all mechanical components of the system. The design aimed to integrate the components discussed in the previous section in an aesthetically pleasing and practical manner. Structural design comprised of two main

areas; the main frame to provide the Ballbot structure, and motor brackets to mount the motors and drive wheels. Complete technical drawings, as submitted to the University of Adelaide Workshop, are included in Appendix H.

#### 5.1.3.1 Main Frame

The core of the structural design was the design of the frame. They key purpose of the frame is to provide the structure of the ballbot and drive mechanism, as well as providing mounting locations for components. The material selected for the frame was Maytec Aluminium Extrusions, as discussed in Section 5.1.2.5. The designed frame can be seen in Figure 32.

Figure 32: Full Scale Ballbot Frame

The overall dimensions of the Ballbot were to approximate that of a human being. Thus, the Full Scale Ballbot was based on approximate dimensions of 1.6 metres high and 0.3 metres wide. The frame design, as discussed in Section 5.1.1, was designed with four legs at 45° onto which the motor brackets are mounted. The body of the ballbot itself is a square frame, which was designed to be approximately 1.5m high. The width was designed to 'frame' the ball, as such, with an outer horizontal dimension of 260mm. This produced a ballbot that does not appear too 'bottom-heavy' but is also not too large along the column. Furthermore, it is also of suitable size to have shelves which are able to carry the batteries, motor controllers and other required components.

The 30×30mm cross-section extrusion was used for the majority of the frame, as this was determined to be visually well-proportioned with respect to the frame dimensions. The structural strength of this cross-section was qualitatively assessed to be far greater than that required to support the components, using available data, and thus was not a major consideration.

Shelves to provide mounting locations were designed to be interior to the frame, maintaining the streamlined appearance of the ballbot. These shelves are the same size as the interior of the main column and are positioned using screw-type mounting blocks available from Maytec.

This allows for flexible positioning of the shelves allowing some control over the centre of gravity. Components to be mounted to the shelves were the microcontroller, batteries, sensors, and assorted electronics.

#### 5.1.3.2 Motor Bracket and Shaft Extension

The motor brackets for the Full Scale Ballbot supports the motors and drive wheels. The motor brackets transfer the entirety of the weight of the Ballbot to the wheels, and thus must be capable of supporting this weight.

A U-shaped design, constructed from three blocks of aluminium, was designed for use as the motor brackets. The wheel is housed within the U-shape, and the motor is front mounted onto one side of the U, as seen in Figure 33(a).

A shaft extension is required, as the Leadshine DCM50205 Motor shaft is neither long nor thick enough to support either of the two sets of omniwheels discussed in Section 5.1.2.7. The shaft extension is simply a shaft with a changing diameter, as shown in Figure 33(b), was designed to fit the 2052-3/8 Multiple Row Transwheel, and is secured to the motor shaft using grub screws. A keyway is used to ensure no wheel rotation relative to the shaft. The shaft and extension are kept aligned using ball bearing at both inboard and outboard locations.

(a) Motor Bracket (b) Shaft Extension

Figure 33: Components designed for Motor Bracket

The bracket is mounted to the frame legs using two screws into threaded plates designed to fit into the slotted extrusion. This allows the bracket to be adjusted up and down the leg, giving the flexibility in the design as desired. The complete assembly of the motor mount can be seen in Figure 34.

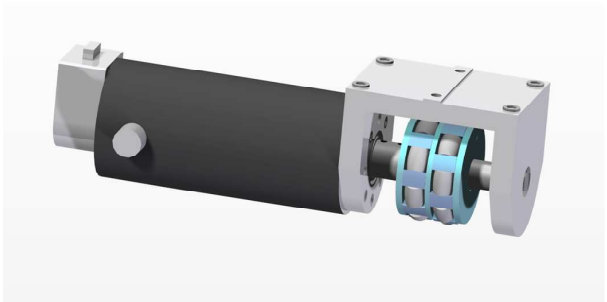


Figure 34: Motor Bracket Complete Assembly

### 5.1.4 Electrical Design

Electrical design of the Full Scale Ballbot included the circuitry and wiring required for the electrical components of the ballbot. This primarily involved integration of the DC motors with the Lego NXT brick, but also included determining the wiring connections for power distribution and signal routing. As the tilt sensors used are all Lego products, these were simply connected directly to the NXT processor and did not require any specific design.

#### 5.1.4.1 Motor Integration

Integration of the DC servo motors with the NXT brick involved matching the NXT brick motor port signals to the signals required by the DC servo motors. When connected to a normal NXT Servo motor, each of the NXT motor ports uses an 8V 0 to 100% duty cycle PWM signal at 8.17kHz to drive the motor, which switches polarity to change motor direction. The ports also produce 4.8V to power the motor encoder, and accepts the quadrature encoder signal from the motor.

Two motor controller boards are used to produce the required motor voltage based on the output signals from the Lego NXT Brick. One motor controller is used for each plane, driving both motors in that plane as they require the same signal. Each board produces an output PWM signal at 24V to the DC motors, with duty cycle and polarity determined by the NXT input signal. Each motor controller board is capable of providing 43.2A - twice the peak motor current - and also produces 5V to power the motor encoders. The motor controller boards were designed and constructed by the University of Adelaide Mechanical Engineering Electronics Workshop, and are shown in Figure 35. A schematic for this board can be seen in Appendix I.

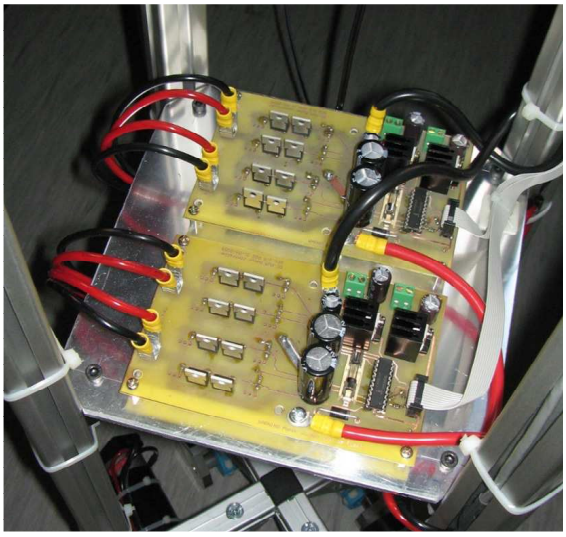


Figure 35: The Motor Controller Boards used on the Full Scale Ballbot

#### 5.1.4.2 Power Distribution and Signal Routing

The complete electrical layout showing power flow and signals can be seen in Figure 36. Electrical wiring of the Full Scale system was performed by the University of Adelaide Mechanical Engineering Electronics Workshop.

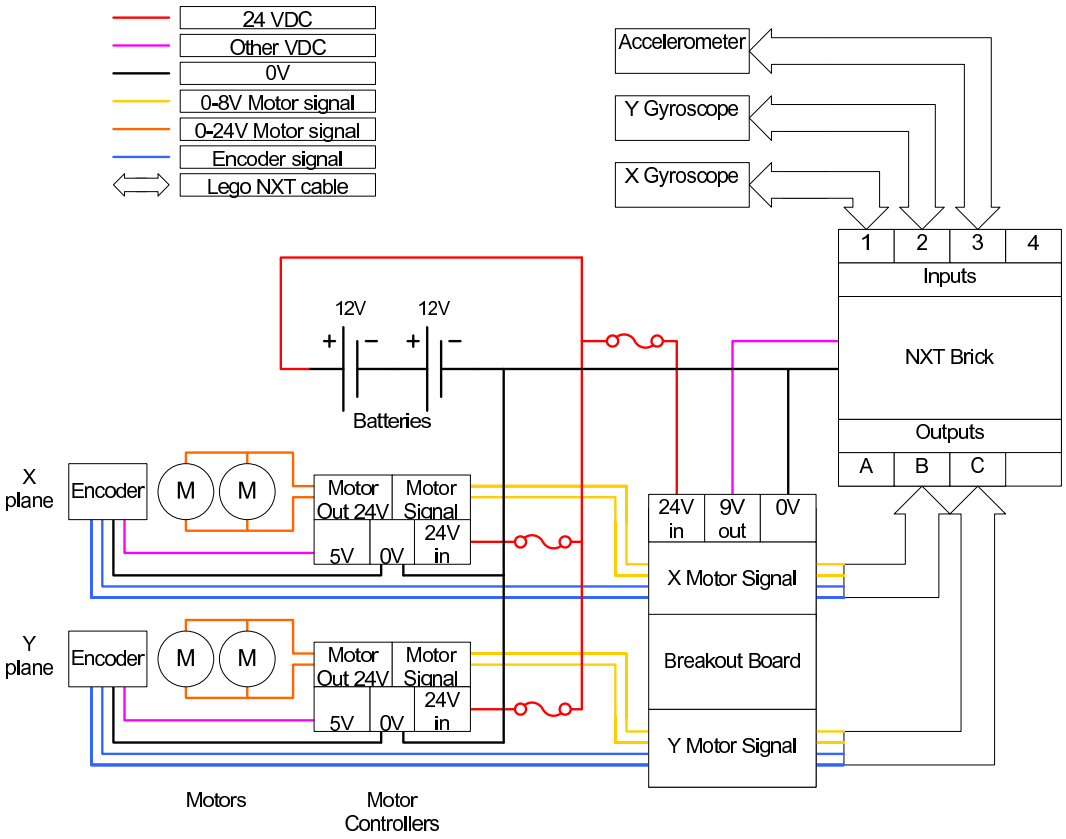



Figure 36: Power Distribution and Signal Routing, as designed

A breakout board was used to centrally locate all wiring in the Full Scale Ballbot, excluding connections to the Lego tilt sensors and the 24V supply of the motor controllers. This centralised board improves aesthetics by reducing loose cabling between the various components of the Ballbot. The output ports of the NXT brick were connected to the breakout board such that the motor and encoder signals from these ports could be connected to the motor controller and encoder respectively. The breakout board also produces 9V, used to power the Lego NXT brick. This board was designed and constructed by the University of Adelaide Mechanical Engineering Electronics Workshop. A schematic for this board can be seen in Appendix I.

### 5.1.5 Completed System

Following the construction of the structural and electrical components of the Full Scale Ballbot, the system was completely assembled. Electrical components were mounted on the shelves, and wiring was conducted by the Electronics workshop. The fully constructed and wired Full Scale Ballbot is shown in Figure 37. Physical parameters for this system are defined in Table 3.

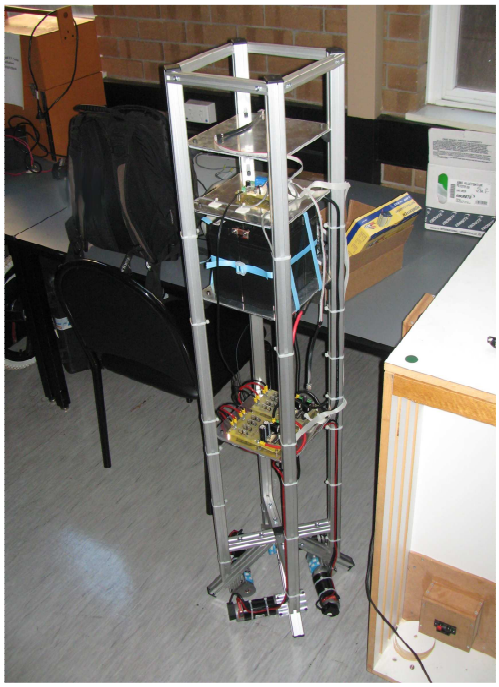


Figure 37: The Constructed Full Scale Ballbot

Table 3: Full Scale Ballbot Physical Parameters

| | | |
|---|---|---|
| Mass of the Body | $M_B$ | 29 kg |
| Height of centre of mass | $L$ | 0.8 m |
| Mass of the Ball | $M_b$ | 5.5 kg |
| Radius of the Ball | $R_b$ | 0.11 m |
| Moments of Inertia of the Body | $I_B$ | 5.6 kg.m$^2$ |
| Moment of Inertia of the Ball | $I_b$ | 0.0266 kg.m$^2$ |
| Gear Ratio, motor to ball | $n$ | $\frac{-0.0246}{0.11} = -0.223$ |
| Moment of Inertia of the Motors and Wheels | $I_M$ | $4.62 \times 10^{-5}$ kg |
| Friction coefficient between body and ball | $\mu_{Bb}$ | 0.2(*estimated*) |
| Friction coefficient between body and ground | $\mu_{Bg}$ | 0(*estimated*) |

## 5.2 Initial Controller Implementation

The initial controller implementation on the Full Scale Ballbot was based on the Lego Ballbot Controller. However, several small modifications to the controller program were required to make it suitable, as well as a recalculation of control gains.

### 5.2.1 Modifications to Lego Controller

The modifications to the controller program were fairly minor. The most major of these was the incorporation of the accelerometer into the state estimation, to provide an more accurate estimate of the body angle state $\theta$. In addition to this, minor modifications were also made to take into account differences in the motor encoder resolution and maximum battery voltage.

In order to combine the accelerometer and the gyroscope readings to estimate the body angle state $\theta$, a second order complementary filter was implemented. This filter, shown in the continuous

domain in Figure 38, attempts to generate an estimate for the body angle by using the gyroscope reading for the high frequency changes in $\theta$ and the accelerometer for low frequency changes in $\theta$. The complementary filter was converted to the discrete domain for use in the controller via Tustin's approximation.

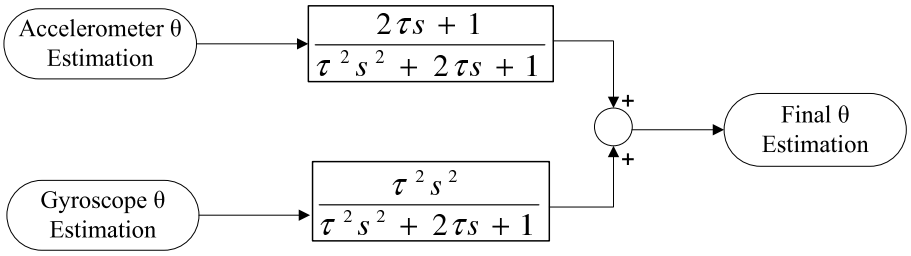



Figure 38: Generation of the $\theta$ Estimation using a Complementary Filter

Other required modifications to the controller program were minor. The motor angle $\phi$ state estimation was scaled to take into account increased resolution of the motor encoders used on the Full Scale Ballbot, compared to those on the NXT Servo Motors. Additionally, modifications were made to the signs of the signals produced by the sensors, as required for the hardware setup. Finally, the maximum battery voltage was changed to a constant 24 volts. Unlike the Lego Ballbot, this was not monitored for the Full Scale system due to the lack of hardware .

### 5.2.2 Calculation of Controller Gains

The calculations of gains for the Full Scale Ballbot was done using the LQR method discussed in Section 3.2. This process was identical to that for the Lego Ballbot, but using the Full Scale Ballbot physical parameters defined in Table 3. The *lqr* command in Matlab was used to calculate the gains as:

$$\mathbf{k} = \begin{bmatrix} 620.2566 & -1.8819 & 209.6209 & -4.3428 & -0.6325 \end{bmatrix} \tag{20}$$

The **Q** and **R** matrices used in the Lego Ballbot controller were retained. It was expected that the gains calculated would simply give an approximation of the order of magnitude, and that the gains would need to be tuned significantly.

## 5.3 Testing Procedure

Testing of the Full Scale Ballbot aimed to find controller parameters, primarily control gains, to achieve a balancing system. To do this, a Safe Operating Procedure (SOP) was written and a Risk Assessment conducted to clearly identify any potential safety hazards. Following this, the Ballbot could be tested in order to determine suitable controller parameters.

### 5.3.1 Safety and Physical Procedure

Due to the size and weight of the Full Scale Ballbot, and its natural tendency to fall, potential exists for the Ballbot to harm personnel or property. A Risk Assessment was conducted to formally identify any such hazards. The standard University of Adelaide Project Risk Assessment template was used for the Risk Assessment, which identifies hazards in categories such as entanglement, crushing, cuts and burns, as well as potential long term ergonomic and environmental issues. This

Risk Assessment was conducted by the project team, and approved by the Mechanical Engineering Safety Officer. A copy of the Risk Assessment can be found in Appendix K.

The Safe Operating Procedure was written to ensure correct operation of the Full Scale Ballbot, to prevent injury or damage. The SOP clearly defines checks which should be conducted each time the Ballbot is run, and provides instructions on the operation of the Ballbot. The authors highly recommend that the SOP be followed whenever the Full Scale Ballbot is run. The SOP was approved by the Mechanical Engineering Safety Officer, and a copy can be found in Appendix L.

The SOP was followed for the testing of the Full Scale Ballbot. The Ballbot was tethered to the gantry crane outside the Mechanical Engineering Acoustics Laboratory as shown in Figure 39. Two team members were required for each test - one to start and monitor the Ballbot, and one who exclusively operated the emergency stop button. This ensured that the Ballbot could not accelerate out of control during the tests.

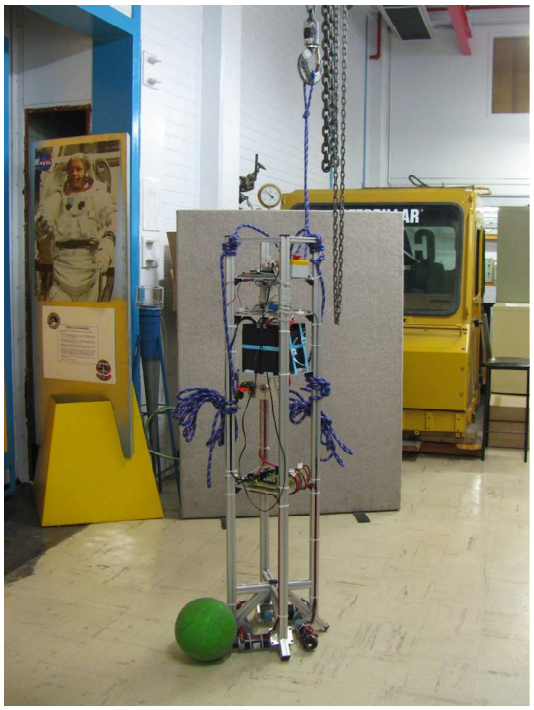


Figure 39: The Testing Area Used for the Full Scale Ballbot

### 5.3.2 Tuning Parameters

The controller parameters available for modification within the tests were the controller gains, filter poles, and friction compensator. The controller gains were initially based on those calculated by the LQR method, and could be adjusted individually with respect to each controlled state. The filter poles were initially the same as those on the Lego Ballbot controller, however, as the dynamics of the Full Scale Ballbot are slightly slower, the filter poles were adjusted where necessary. Finally, the friction compensator in the PWM signal generation was used to attempt to eliminate the dead zone in the motors, which will be discussed further in Section 5.4. Additionally, battery location could be varied as this had a significant effect on the Ballbot's dynamics.

Performance of the system and controller was assessed both visually and through data collected through Bluetooth transfer from the NXT Microprocessor. Visual feedback was useful primarily for coarse tuning of the controller gains - using qualitative analysis to determine the role of each gain, and the motion of the system. Data was consulted for a more quantitative assessment of the controller and to ensure that saturation was not achieved.

## 5.4 Initial Design Performance and Modifications

The controller created as described in Section 5.2 was not successful in balancing the Full Scale Ballbot. This was found to be primarily due to an error in the state estimation, however, other issues, included slip between the wheel and ball, and a large dead zone in the motors, were also encountered.

### 5.4.1 State Estimation Error

It was determined that the estimation of the body angle $\theta$ was incorrect, through interrogation of the logged data. The complementary filter pole was initially given a high value, in order to eliminate the reading of the accelerometer. However, this caused the system to fail due to gyroscopic drift. The use of a slower pole in the complementary filter, however, produced another problem with the state estimation. Figure 40 shows the recorded performance of the state estimation while testing the controller. In this case the filter pole has a time constant of 4s.

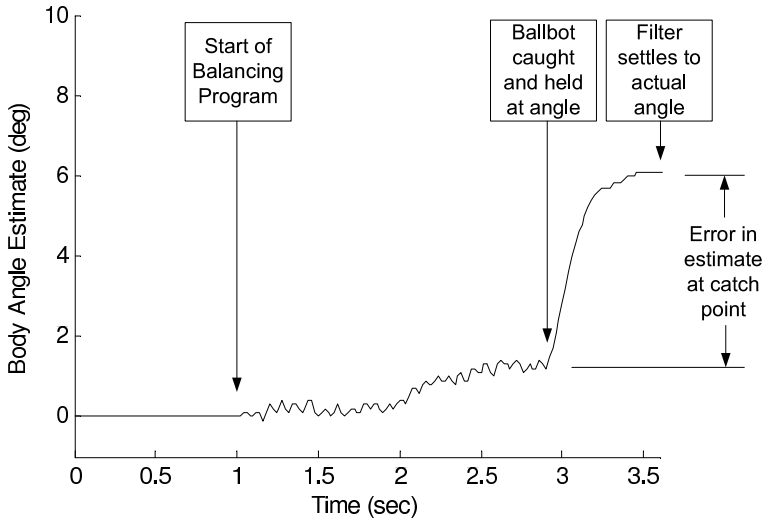



Figure 40: Performance of the Complementary Filter

In this test, the Ballbot is emergency stopped and caught at 2.9 seconds. The ballbot is held stationary from this point in time. It can be seen in the plot that the body angle estimate continues to change even after this has occurred. This suggests that the estimate is underestimating the true value of body angle, and only slowly converging to the true value when held steady. This effect is believed to be due to accelerations of the Ballbot causing errors in the accelerometer reading of tilt. Testing a variety of complementary filter poles was not successful in creating an filter which gave good performance. As such, it was concluded that the response of the complementary filter was not satisfactory, regardless of the value of the complementary filter pole.

### 5.4.2 Mechanical issues

During testing, two mechanical issues could also be observed. Slip was encountered particularly when the motors attempted to drive quickly, resulting in the motors wheels spinning but not driving the ball. The wheels were not able to re-engage due to the effects of the $\phi$ and $\dot{\phi}$ gains, and the lower dynamic friction coefficient. Thus this would cause the system to fail. It was found that the rubber coating on the ball was effective at reducing the slip.

A deadzone in the motors was also encountered. It was found that the motors would not be capable of driving the ball until given a signal of greater than approximately 40%, however, accelerated quickly once the shaft begun to spin. This is likely due to viscous friction in the drive mechanism, where the rapid acceleration is caused by the drop in resistance due to the change from static friction to dynamic friction. This produces a large non-linearity in the dynamics. To reduce the effects of this problem, a friction compensator (a Simulink programming block) was used to try to overcome this. However, this solution did not address the non-linearity in the system dynamics. This non-linearity was not modelled in the system dynamics, and severely affects the performance of the linear state space controller.

### 5.4.3 Modifications: Kalman Filter Implementation

Of the issues observed during initial testing, the error in the state estimation was the most major, and was therefore the first to be addressed in a more serious manner. A Kalman filter was suggested as an alternate method of combining the two sensor signals to produce a more accurate estimate. A predictor-updator approach is used, first predicting the state using previous state information and a model of the system, and then updating this prediction using the measured sensor data. The combination of the prediction and sensor information is governed by a set of optimal kalman gains $K$, which are based on the expected error in the system model and sensors respectively. An Extended Kalman Filter was implemented, using the non-linear dynamics as derived in Section 3.3, which was converted to the discrete domain. The structure of a Kalman Filter is shown in Figure 41. The filter implemented also included the expected dynamics of the sensors. This was particularly of importance for the accelerometer, where the Kalman filter attempts to remove all non-gravitational accelerations.

The implemented filter was initially tested for the open loop ballbot system with all control signals set to zero, and was successful at estimating body angle accurately for this case. However, the filter was not successful in generating reasonable estimates of the state once the control law was included and the ballbot operated normally. It is believed that this is due to the high degree of non-linearity in the motors, in particular the high deadzone. These effects were not modelled in the dynamics of the system, as they are extremely difficult to model.

Due to the failure of both filters in producing an accurate estimation for the state $\theta$, it was decided that the HiTechnic Accelerometers and Gyroscopes were inadequate for estimating the state. As such, it was concluded that a redesign of the system was required.

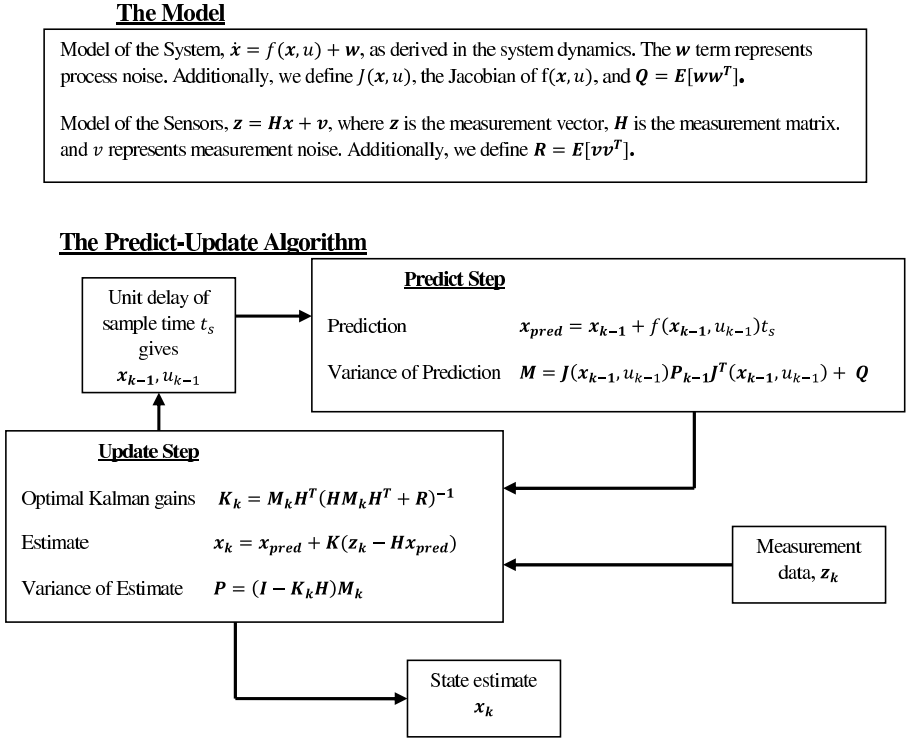



Figure 41: Structure of a Kalman Filter, adapted from Zarchan (2005)

## 5.5 Revised Design

Due to the issues identified during testing of the initial design of the Full Scale Ballbot, the design was revised in order to improve the state estimation. The major change in the revised design was the use of an Inertial Measurement Unit (IMU) to accurately measure the $\theta$ state. The IMU requires a serial interface, and thus the microprocessor was changed to a HCS12 Dragonboard. The change to this microprocessor required the re-programming of the controller program in C code. It should also be noted that at the time of modification, the project had failed to produce a balancing Ballbot for Open Day 2009. As such, the time constraints on the project were slightly relaxed.

### 5.5.1 Inertial Measurement Unit

The IMU used was a Microstrain 3DM-GX1 Attitude Heading Reference System (AHRS), shown in Figure 42(b). The sensor provides a very accurate measure of the angle of the sensor's inclination, as well as its bearing. The IMU uses a combination of sensors including gyroscopes and accelerometers, combined using various filters, to produce data which is then transferred through a serial interface. This IMU was used due to its availability at the University of Adelaide.

### 5.5.2 Dragonboard

The Lego Mindstorms Microprocessor is not capable of interfacing with serial devices, and thus a change in microprocessor was required. A Dragonboard, shown in Figure 42(a), was chosen again due to its availability at the University of Adelaide. The Dragonboard is a development board which includes the HCS12 Microprocessor as well as timer units, PWM signal generators and serial interfaces which were used within the project, as well as buttons and an LCD display

which was used for debugging. As such the Dragonboard includes all functionality required for the Ballbot.

(a) Dragonboard rev. E


(b) Microstrain 3DM-GX1 IMU

Figure 42: Revised Hardware Components

### 5.5.3 Controller Programming

The use of the Dragonboard required that the controller be modified to suit the new processor. The controller for the revised Full Scale Design was programmed in C code using Metroworks Codewarrior, even though a Simulink toolbox for the Dragonboard, similar to the ECRobot toolbox, was available for the project team. The use of this toolbox would have allowed for modification of the existing Simulink controller for use on the Dragonboard. However, C code was used due to the project team's experience with development in C code for the Dragonboard platform, and the additional flexibility and control granted through the use of C code. It was decided that the security of rewriting the code was preferable to the faster development, due to the more relaxed time constraints.

The controller program was required to estimate the states, calculate a control signal, and convert this control signal to a PWM for the motors. Due to the time-critical nature of many of these tasks, and interrupt-driven program was designed. The complete listing of the code for the Full Scale Ballbot controller can be seen in Appendix J.

#### 5.5.3.1 State Estimation

Estimates of the five defined states were required, using the data from the IMU and the motor encoders. In order to do this, communication with IMU and interpretation of the motor encoder signal had to be developed, and then processed.

The IMU was used to measure the $\theta$ states. Both the angle and the angular rate in the x and y planes are available from the IMU, and thus estimation of $\theta$ and $\dot{\theta}$ states is obtained directly from the IMU data. The controller program requests and receives data from the IMU through the serial interface. Continuous operation of the IMU was selected, where packets of data are continually sent from the IMU after an initial request. Code for this was developed by Zebb Prime at the University of Adelaide, and this was adapted for use in this project. Interrupts are used to receive each packet of data and a non-time-critical checksum routine is used to ensure the integrity of the data. If this checksum is passed, the estimates of states $\theta$ and $\dot{\theta}$ are updated.

The motor encoders are quadrature encoders, with a resolution of 500 turns per revolution. As such, the two signals of the motor encoder must be processed, in order to estimate $\phi$. Within the program, a simple count representing the rotation of the motor shaft in $500^{ths}$ of a revolution is incremented or decremented using an interrupt service routine (ISR) triggered by one of the encoder signals. This ISR checks the status of the second encoder signal, and increments or decrements as required. Using this count, the $\phi$ state can be estimated simply through scaling the count. In order to estimate $\dot{\phi}$ and $\int \phi$ states, a time-driven interrupt at 100 Hz was used. Using this known timestep, these states are calculated using first order differentiation and integration respectively. Due to the discrete nature of the encoder signal, however, the derivative was passed through a low pass filter to reduce any derivative spikes.

#### 5.5.3.2 Calculation of Control Signal

The calculation of the control signal was a simple task. This task was not as time critical as state estimation, and thus was not carried out within an interrupt. However, the control signal is updated at approximately 100 Hz. The program simply calculates the control signal for each plane as

$$u_i = -\left[k_\theta \theta_i + k_\phi(\phi - \phi_{cmd}) + k_{\dot{\theta}}\dot{\theta} + k_{\dot{\phi}}\dot{\phi} + k_{\int\phi}\left(\int \phi - \int \phi_{cmd}\right)\right] \tag{21}$$

The states are updated separately as discussed in Section 5.5.3.1, and are read from variables. Command variables are only used for the $\phi$ and $\int \phi$ states as the others remain zero at all times.

#### 5.5.3.3 PWM Signal Generation

The motor controller boards expect a PWM signal of the same form as generated by the Lego NXT Microprocessor. As such, the PWM units on the Dragonboard were used to mimic this signal, such that the motor controller boards would not require modification. The characteristics of the expected signal can be seen in Figure 43, where the duty cycle is proportional to the magnitude of the control signal.


(a) Forward Drive (b) Backward Drive

Figure 43: Required PWM Signal

## 5.6 Revised Design Results and Analysis

The revised design implementing the new controller and Inertial Measurement Unit was successful in balancing the Full Scale Ballbot. The testing approach outlined in Section 5.3 was continued on the new controller, with the gains, low pass filter on the state estimation of the $\dot{\phi}$ state, and the deadzone being the adjustable parameters. As before, the battery location could also be varied.

Following the testing, the gains which were judged to give the best performance were:

$$\mathbf{k} = \begin{bmatrix} 1200 & -1.0 & 130 & -4.0 & 0 \end{bmatrix} \tag{22}$$

The other parameters were selected included a time constant of 0.19s for the low pass filter pole, and a 10% offset in the friction compensator to overcome the deadzone. Finally, it was found that the best location for the batteries was the top of the Ballbot.

These parameters were successful in stabilising the ballbot, and provided adequate performance. However, performance was limited by the motors non-linear nature and limited strength. Slip in the drive mechanism also presented issues, as the ballbot would become unstable as soon as significant slip occurred. Additionally, a rotation of the Ballbot was observed while it balanced. The cause of such a rotation is likely coupling between planes in the drive mechanism, and the Ballbot has no way of preventing this rotation due to the omniwheel configuration. Results of specific tests performed with these gains are presented in the remainder of this section, including balancing and command tracking.

### 5.6.1 Balancing

Balancing was the fundamental objective of the Full Scale Ballbot. In order to do so, the Ballbot controller has to primarily reject noise from the sensors, as well as deal with non-linearities present in the system. Figure 44 shows the measured body and motor angle whilst the Full Scale Ballbot is balancing. The plot shows that the Full Scale Ballbot has successfully balanced, maintaining body angle to within $\pm 1^\circ$. However, in order to do so the Ballbot has entered a limit cycle, slowly moving back and forth in space with a period of approximately 5 seconds. This can be seen by the oscillations of the motor state $\phi$. This oscillation corresponds to a linear movement of the Ballbot within a 0.4m range. Similar oscillations were observed in simulations and the Lego Ballbot, however these were of significantly smaller magnitude. The increased magnitude is most likely due to non-linear properties of the motors and drive system, such as high deadzone.

Additionally, small pushes were applied in order to test the system's disturbance rejection capabilities. The ballbot successfully rejected these small disturbances, driving the ball such that it remained upright whilst pushed. Results are not presented in this report as the changes in body angle are not noticeable from the normal limit cycle.

### 5.6.2 Command Tracking

Command Tracking was the final goal of the Full Scale Ballbot. Due to the limit cycle and rotation that were observed when the Full Scale Ballbot was balancing, however, it was expected that command tracking capabilities of this ballbot would be limited. Additionally, unlike the Lego Ballbot, no wireless link was implemented for the Full Scale system, meaning that remote control command tracking would not be available. Despite these limitations, command tracking tests were performed to demonstrate the controller's potential in this area. An example is shown in Figure 45, where the Ballbot is commanded to move at approximately 7.7cm/sec (0.5 rps for the motors) for 15 seconds. The plot shows that the Ballbot ceases its normal limit cycle to smoothly follow the command, and then resumes the limit cycle around the final location at the completion of the

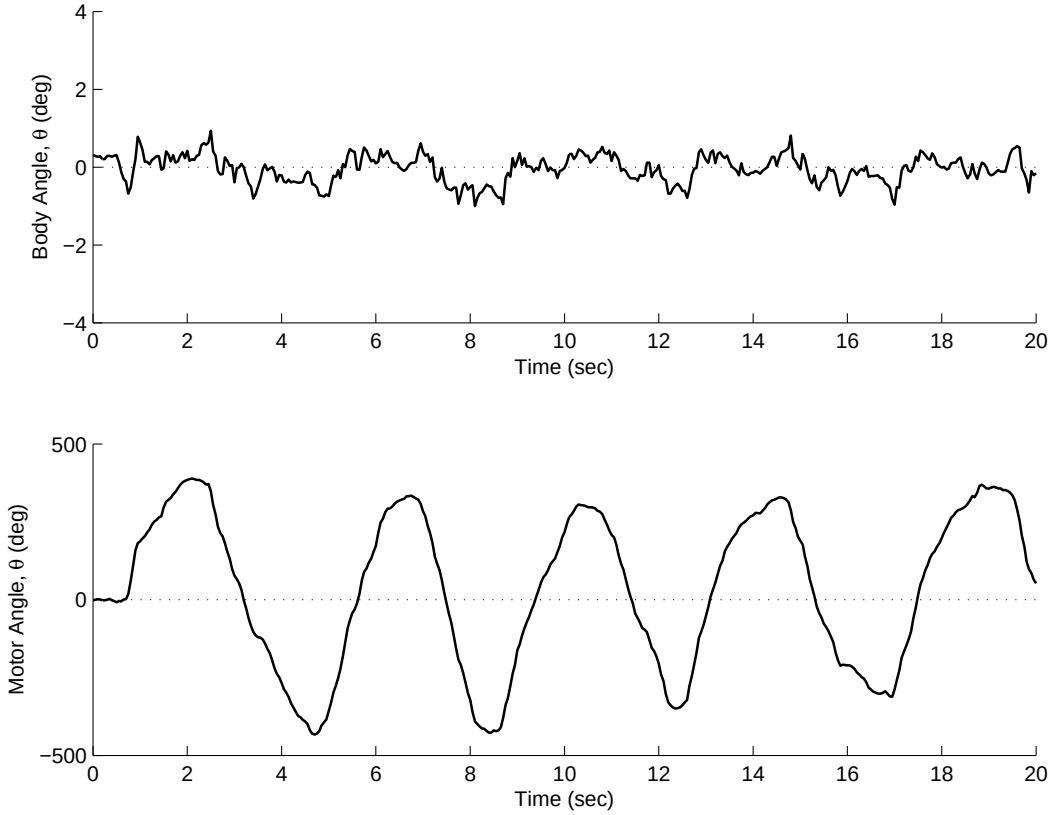



Figure 44: Balancing Performance of the Full Scale Ballbot

command. It is apparent that the limit cycle is ceased due to the ballbot no longer being affected by some of the non-linearities in the drive mechanism, such as backlash and viscous friction. Further assessment of performance of the tracking, including overshoot and settling time, is difficult due to the resumption of the limit cycle at the completion of the command.

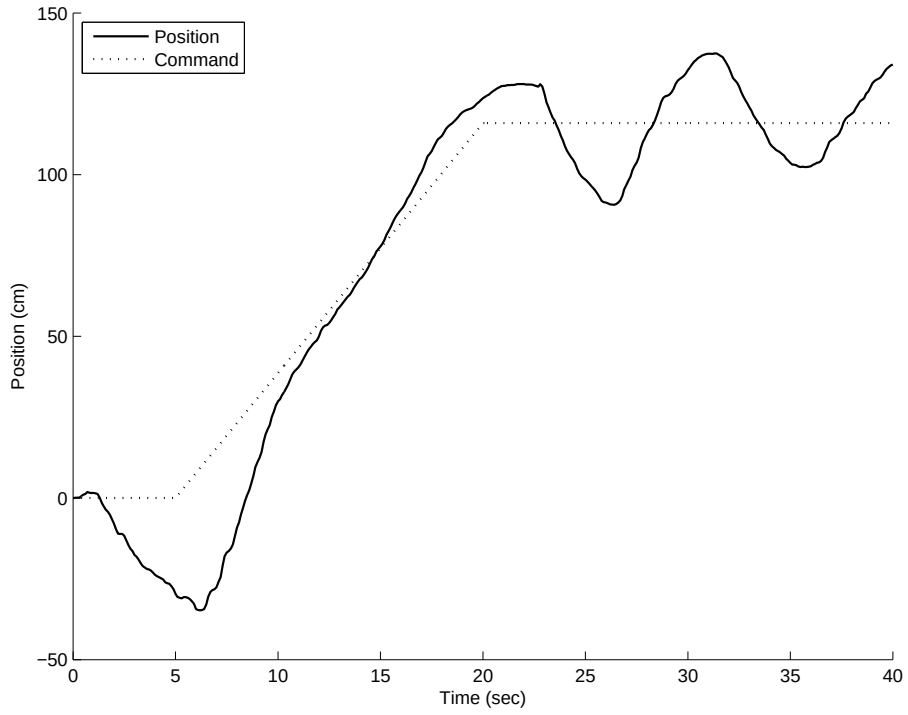



Figure 45: Command Tracking Performance of the Full Scale Ballbot

# 6 Resource Analysis

Two primary resources were used within the project; monetary costs and labour hours. The costs have been monitored with the intention of producing an estimate for the overall cost of the project. The estimated overall cost is $94258.06, which includes costs associated with all components used within the construction of the ballbots, as well as an estimated cost of the labour.

## 6.1 Component Costs

The monetary funds for the Ballbot project have been derived from two sources - the Mechanical Engineering Honours Project allocation and the Open Day Innovation Fund 2009. These were for the amounts of $400 and $3000 respectively, producing a total budget of $3400.

The price of the components for the Ballbot project have been divided into those required for the Lego Ballbot and those required for the Full Scale Ballbot. It should be noted that some of the parts have been used on loan from the University of Adelaide, and thus the costs have not been incurred by the project team. These costs are indicated with *, and have been estimated for inclusion in this analysis for completeness.

### 6.1.1 Lego Ballbot

The costs of components required for the Lego Ballbot were as shown in Table 4. The majority of the components required for the Lego Ballbot were purchased as part of the Lego Mindstorms Kit, with the exception of the gyroscopic sensors. As such, the Lego Ballbot used very little of the total budget.

Table 4: Component Costs for the Lego Ballbot

| Component | Source | Quantity | Total Cost |
|---|---|---|---|
| Lego Mindstorms NXT Kit | Dick Smith Electronics | 1 | $358.00 |
| HiTechnic Gyroscopic Sensor | Lego Shop | 2 | $199.80 |
| Rechargable Battery | Lego Shop | 1 | $69.99 |
| Total | | | $627.72 |

### 6.1.2 Full Scale Ballbot

Table 5 shows the costs of the Full Scale Ballbot. It can be seen that a large proportion of the cost of the Full Scale Ballbot was due to the motors. The cost of motors can vary greatly, and the motor configuration selected was the cheapest available option available by a significant margin. However, this may have led to some of the issues encountered while testing the Full Scale Ballbot, such as the large deadzone. Should a larger budget have been made available, better quality motors would have been selected. The frame was also an expensive item. However, it was felt that the purchase of this frame had many significant advantages, such as a short lead time and aesthetic qualities, that justified the cost for this project.

The additional costs of the Revised Design are also listed in Table 5. Both items are on loan from the University of Adelaide, and thus the costs have not been taken from the project budget. However, should the design be replicated, these items add significant cost to the Full Scale Ballbot.

Table 5: Component Costs for the Full Scale System

| Component | Source | Quantity | Total Cost |
|---|---|---|---|
| Lego Mindstorms NXT Kit | Dick Smith Electronics | 1 | $358.00 |
| HiTechnic Gyroscopic Sensor | Lego Shop | 2 | $199.80 |
| DC Motor DCM50205 with encoder | Ocean Controls | 4 | $1021.50 |
| 2052KX Cat-Trak Wheels | Kornylak Corparation | n/a | $171.93 |
| Ballbot Frame | Metri-tek Pty Ltd | 1 | $769.84 |
| Bowling Ball | AMF Port Road | 1 | $100.00 |
| Plasti-Dip | Ashdown-Ingram | 1 | $34.38 |
| 12V 18 AH Sealed Lead Acid Battery | Jaycar Electronics | 2 | $98.12 |
| 24V 6A Battery Charger | Jaycar Electronics | 1 | $80.37 |
| Motor Controller Boards | Electronics Workshop | 2 | $200** |
| Breakout Board | Electronics Workshop | 1 | $40** |
| Misc. Electrical Items | Electronics Workshop | n/a | $236.30 |
| Bearings | CBC Bearings | 8 | $138.03 |
| Motor Brackets and Shelves | Mechanical Workshop | 4 | $100** |
| Total | | | $3548.27 |
| Additional Costs for Revised Design | | | |
| Component | Source | Quantity | Total Cost |
| Wytec HCS12 DragonBoard | - | 1 | $165.00* |
| Microstrain 3DM-GX1 IMU | - | 1 | $2300.00* |
| Total | | | $2465.00 |

* Items were not purchased on project budget and cost is an estimate.
** Items are an estimate based on information from the University of Adelaide Workshop

## 6.2 Labour Costs

### 6.2.1 Student Hours

The project team have logged the time personally spent on the project. These hours were logged as shown in Table 6.

Table 6: Hours Logged per Team Member

| Month | Justin Fong (Hours) | Simon Uppill (Hours) |
|---|---|---|
| March | 61.5 | 57.5 |
| April | 63.5 | 66.5 |
| May | 97.5 | 95.5 |
| June | 6.5 | 6.5 |
| July | 76 | 53 |
| August | 96.5 | 95.5 |
| September | 90.5 | 82.5 |
| October | 157.5 | 158.5 |
| Total | 650.5 | 614.5 |

This time has been converted to a monetary cost approximation on the following assumptions:

- Annual Salary of $50 000 per year ($26 per hour)
- Direct Costs of 30% (including superannuation, payroll tax, etc)
- Indirect Costs of 130% (including admin and tech support, rent, phone, etc)

Leading to the cost breakdown shown in Table 7. It can be seen that the costs associated with labour are significantly greater than the component costs, and would need to be properly budgeted for should a similar project be conducted on a professional level.

Table 7: Labour Costs

| Cost Category | Cost |
|---|---|
| Costs at Hourly Rate | $32 449 |
| Direct Costs (30%) | $9 735 |
| Indirect Costs (130%) | $42 183 |
| Total Costs | $84 367 |

### 6.2.2 Workshop Hours

Details on hours worked on the project by the University of Adelaide Mechanical Engineering Workshop, both Mechanical Workshop and Electronics Workshop, were not available at the time of printing. However, it is expected that the total workshop time was approximately 65 hours, primarily from the Electronics Workshop. This equates to an approximate cost of $3250 at $50 an hour.

# 7 Project Outcomes and Future Work

## 7.1 Project Outcomes

At the commencement of the this project, several goals were identified to provide an measurement of the project's success, outlined in Section 1. All core goals defined have been achieved, and several extension goals have also been achieved or partially achieved. Details of the achievements with respect to each goal follows.

**Derivation of the Equations of Motion:** Equations of Motion of a generalised ballbot system have been derived in both non-linear and linearised form. Several assumptions were required during the derivation, and as such the equations do not give a perfect model of the system. However, the derived equations are believed to provide a fairly accurate model, as simulation results were similar to actual results.

**Design and Construction of a Lego Ballbot:** A Lego Ballbot has been successfully designed and constructed. This was done using Lego Parts available in the Lego 8527 NXT Mindstorms kit, but also required HiTechnic Gyroscopes and alternate Lego wheels.

**Development of a Controller to Stabilise a Ballbot:** A controller program has been developed in Simulink which successfully balances the Lego Ballbot. Additionally, the controller is able to maintain balance when subjected to small disturbances, and also provides command tracking capabilities.

**Remote Control of the Lego Ballbot:** Bluetooth was successfully used to allow for remote control of the Lego Ballbot. This was used in conjunction with the command tracking feature of the controller to allow the user to drive the ballbot forward, backward, left and right.

**Design and Construction of a Full Scale Ballbot:** A Full Scale Ballbot has been designed and constructed. However, implementation of the Simulink controller as used for the Lego Ballbot was unsuccessful at balancing this system. Instead, the controller was recreated in C code on a Dragonboard with a HCS12 processor, and this was successful at balancing the Full Scale system.

**Create Building Instructions for the Lego Ballbot:** Graphical step-by-step building and programming instructions for the Lego Ballbot have been produced, allowing the Lego Ballbot to be easily reproduced from a Lego 8527 NXT Mindstorms kit and the few additional required parts.

**Produce a Virtual Reality Model of the Lego Ballbot:** A virtual reality model of the Lego Ballbot has been created, using the simulated response created using the derived dynamics and the simulated control to provide a real-time visual representation of a ballbot system.

**Extend Functionality of the Lego Ballbot:** Yaw control was developed for the Lego Ballbot. This involved both the modification of the design of the Lego Ballbot, as well as the development of a controller for this purpose. This was successful at regulating the ballbots yaw angle. Additionally, remote control of the yaw of the Ballbot was also implemented. No further environmental interaction, such as obstacle avoidance, was developed.

**Extend Functionality of the Full Scale Ballbot:** This goal has been partially addressed, as command tracking was implemented on the Full Scale Ballbot. However, this was only tested by hard coding movement commands into the program, as opposed to remote control as implemented on the Lego Ballbot. No further environmental interactions, such as yaw control or obstacle avoidance, were attempted.

## 7.2 Future Work

Ballbots are a relatively unexplored concept, and thus much work must be done in their development before they can be used to fulfil the dream of a mobile robotic personal assistant. This includes further development of the general ballbot concept, as well as further work on the two ballbots produced in this project, the Lego Ballbot and Full Scale Ballbot.

### 7.2.1 General Ballbot Concept

The project was successful in developing dynamics for the Ballbot system, however, many simplifications and assumption were required, which lead to problems with unmodelled influences during the testing phase of the project. An attempt may be made to model additional significant influences on the Ballbot system, which may improve performance of subsequent Ballbot Systems.

The 2009 Ballbot Project used exclusively a state space control system. This imposed limitations on the performance of the system, particularly due to its linearity. Other methods of control, such as fuzzy logic, may be investigated to be used instead of, or in conjunction with, state space control, which may lead to better performance.

### 7.2.2 Lego Ballbot

The authors feel that further work on the Lego Ballbot is limited, as all desired capabilities of this ballbot have been achieved, including balancing, disturbance rejection and command tracking. Further work may be attempted in its optimisation, in order to achieve better performance in these areas. The authors feel, however, that a significant redesign of the physical Ballbot may be required to be successful in this endeavour. Additionally, it may be possible to extend the functionality of this Ballbot to include capabilities such as obstacle avoidance. Finally, the Simulator and Virtual Reality model of the Lego Ballbot may be improved, to model additional sources of noise, or to provide a more accurate model of the system.

### 7.2.3 Full Scale Ballbot

The Full Scale Ballbot produced within this project is capable of balancing and command tracking under controlled conditions. As such, its performance is not adequate for public use and much potential for further work exists for this system. This includes addressing several small issues currently present in the system, further testing to optimise the current system, and modification of the design to improve performance.

#### 7.2.3.1 Current Issues to be Addressed

Several hardware and software issues currently present on the Full Scale System can be addressed. These include:

**Slip in the drive mechanism:** During operation the drive wheels occasionally slip on the ball, causing the ballbot to become unstable. This is more prevalent in the X drive motors, suggesting that the motor mounts are not located in exactly the right position.

**Slip of the shaft extension:** During operation the grub screws loosen, leading to slip between the motor shaft and shaft extension. This can cause the Ballbot to become unstable.

**Restart procedure:** When the Dragonboard is restarted by pressing the on-board 'reset' button, the serial connection to the IMU is not properly re-established when the program is restarts, however other functionality performs normally. A restart and run from Metroworks on a connected computer is required to re-establish successful IMU communication. This issue should be investigated and addressed.

**Wear on the ball:** Slip of the drive wheels leads to significant wear of the balls rubber coating. This results in the coating having to be periodically reapplied. The authors suggest the investigation of more durable coatings or the use of a different ball.

**Unequal loading of the 12V Regulation circuits on Motor Controller Boards:** The loading across these is not equal, leading to one providing much more power and getting significantly hotter. Possible remedies include increasing the heat sink size, or adding a cooling fan.

**Intermittent motor enable signal to Y motor controller board:** Occasionally and intermittently this signal is not being provided when it should be. Possible causes include a defective relay on the breakout board, which is responsible for activating this signal. If this is the case, a replacement relay should solve the problem.

**Battery mounting:** The batteries are not rigidly held in place. Any movement of the battery may change the centre of mass of the ballbot, meaning that angle offsets may have to be recalibrated. Addition of rigid battery mounts will address this issue.

#### 7.2.3.2 Optimisation of the Current System

Some potential exists to further optimise the current Ballbot's performance. Testing on the balancing Full Scale Ballbot was limited, due to the large changes made in response to the initial issues encountered. As such, the performance of the Full Scale Ballbot can be further optimised. In particular, the performance may be optimised for disturbance rejection and command tracking. Furthermore, different configurations of the drive mechanisms may be trialled, due to the adjustable nature of the drive mechanism.

#### 7.2.3.3 Modification of the Design

The Full Scale Ballbot may be modified to improve performance. The authors feel this is required in order to achieve significant improvement in the Ballbot's performance. Possible modifications include:

**Replacement of the Motors** The motors used in the project may be replaced with more powerful and more reliable motors. This is likely to improve performance due to the deadzone issue encountered and discussed, and may improve the performance limits of the system. Alternatively, motors with current or torque control may be trialled, as this should produce more reliable control. However, it is likely that this will lead to a significant additional cost.

**Replacement of the Power Source** From the testing conducted in the project, it was apparently that the mass distribution of the Ballbot had a significant effect on the Ballbot's dynamics. As the batteries on the ballbot account for approximately 40 percent of the total weight, it is recommended that alternate power sources are trialled with the aim of reducing the Ballbot's weight.

**Redesign of the Drive Mechanism** The current drive mechanism has no capability of controlling the Ballbot's yaw. It may be possible to redesign the drive mechanism such that this can be achieved.

**Addition of other Control Mechanisms** Other control mechanisms, such as reaction wheels, may be added to the Ballbot to aid stability and/or provide yaw control.

# 8 Conclusion

The 2009 Ballbot Project successfully achieved its primary goals of design and building two Ballbots, stabilised using state space control. The first ballbot, the Lego Ballbot, was constructed from the Lego NXT Mindstorms range and was used to verify the state space controller and provide a teaching aide in control subjects. The second ballbot produced was the Full Scale Ballbot, designed to have similar dimensions to a human, to display the concept at a larger scale.

This project has demonstrated the application of state space control to a ballbot system. A generalised ballbot model was developed, resulting in non-linear equations of motion. The accuracy of these equations is supported due to the similarities between simulations using the derived dynamics and experimental results. The two ballbots produced have shown that state space control is capable of stabilising a ballbot system, as well as providing disturbance rejection and command tracking capabilities. However, the controller's performance was limited by non-linearities present each the system, particularly those introduced by the motors, as these cannot be accounted for due to the linear nature of the controller. It is suggested higher quality and more powerful motors may be used to reduce the non-linear effects. Alternatively, a non-linear controller may be implemented which may able to account for these non-linearity and thus improve performance.

The Lego Ballbot displays the ballbot concept at a small scale. This ballbot was to be the first of this nature, however another ballbot constructed from Lego has been developed by Yamamoto (2009) since the commencement of the project. Regardless, the Lego Ballbot is presented as a teaching aide, with the additional virtual reality simulator and assembly instructions furthering its potential in this area.

The Full Scale Ballbot successfully utilised a previously un-implemented symmetric four omniwheel drive mechanism, with the aim of overcoming limitations of the traditional 'inverse-mouseball-drive' used on a previous ballbot. However, physical factors, in particular slip between the drive wheels and ball, limited performance of this ballbot. This highlights the critical nature of the wheel-ball interface in a ballbot.

The production and testing of the Full Scale Ballbot also demonstrated the critical nature of sensors on a ballbot. Limitations in the sensors initially used, HiTechnic Gyroscopic Sensors and a 3-Axis Accelerometer, meant that a sufficiently accurate body angle estimate could not be produced. A 3DM-GX1 Inertial Measurement Unit, which replaced these sensors, improved the accuracy and reliability of the body angle measurement to a level which resulted in successful stabilisation of the Full Scale Ballbot.

Ballbots provide an exciting application of control theory, particularly the concept of *dynamic stability*. The ballbots produced in this project demonstrate the successful use of state space control for a ballbot system, and highlight areas critical to a successful ballbot, namely the drive mechanism and sensors used for body angle estimate. Additionally, the Lego Ballbot provides an easily replicable ballbot that may be used as a teaching aide, complete with a virtual reality simulator.

# 9 References

AJ Brizard. *An Introduction To Lagrangian Mechanics*. World Scientific Publishing Company, 2008.

Kornylak Corporation. *OMNI Wheels*, 2007. Viewed 19th May 2009, *http://www.omniwheel.com*.

L Havasi. 'ERROSphere: an Equilibrator Robot'. In *International Conference on Control and Automation*, Budapest, Hungary, June 2005. IEEE.

Hitachi. *Hitachi Single-Chip Microcomputer H8/3052 Z-TAT Hardware Manual*, 2001.

HiTechnic Products. *HiTechnic Products*, 2008. Viewed 30th April 2009, *http://www.hitechnic.com*.

M Kumagai and T Ochiai. 'Development of a Robot Balancing on a Ball'. In *International Conference on Control, Automation and Systems*, Seoul, Korea, October 2008.

TB Lauwers, GA Kantor, and RL Hollis. 'One is Enough!'. In *12th International Symp. on Robotics Research*, pages 1–10, San Francisco, October 2005.

TB Lauwers, GA Kantor, and RL Hollis. 'A Dynamically Stable Single-Wheeled Mobile Robot with Inverse Mouse-Ball Drive'. In *Proceedings IEEE International Conference on Robotics and Automation*, Orlando, Florida, May 2006. IEEE.

CW Liao, CC Tsai, YY Li, and CK Chan. 'Dynamic Modeling and Sliding-Mode Control of a Ball Robot with Inverse Mouse-Ball Drive'. In *SICE Annual Conference*, The University Electro-Communicaions, Japan, August 2008.

U Nagarajan, AK Mampetta, Kantor GA, and RL Hollis. 'State Transition, Balancing, Station Keeping, and Yaw Control for a Dynamically Stable Single Spherical Wheel Mobile Robot'. In *Proceedings IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009. IEEE.

EM Schearer. 'Modeling Dynamics and Exploring Control of a Single-Wheeled Dynamically Stable Mobile Robot with Arms'. Master's thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, August 2006.

Segway Inc. *Segway*, 2009. Viewed 28th April 2009, *http://www.segway.com*.

CW Wu and CK Hwang. 'A Novel Spherical Wheel Driven By Omniwheels'. In *Proceedings of the Seventh International Conference on Machine Learning and Cybernetics*, pages 3800–3803, Kunming, July 2008.

Y Yamamoto. 'NXTway-GS Model-Based Design - Control of Self-Balancing Two-Wheeled Robot, built with LEGO Mindstorms NXT', February 2008.

Y Yamamoto. 'NXT Ballbot Model-Based Design - Control of a Self-Balancing Robot on a Ball, built with LEGO Mindstorms NXT', April 2009.

P Zarchan. *Fundamentals of Kalman Filtering: A Practical Approach*. American Institute of Aeronautics & Astronautics, 2005.

# APPENDIX

## A Derivation of Dynamics

The equations of motion of the simplified Ballbot model in one plane are derived in Matlab using the Lagrangian approach as discussed in Section 2.1.2. The code used can be seen in Appendix B. As per the assumptions, only the equations of motion of one plane (x-z) are derived and then the other plane (y-z) is assumed to have identical dynamics.

The derivation using the Lagrangian approach begins by determining the kinetic and potential energy of each of the ball, body and motors, as follows:

The Ball:

Table 8: Generalised Coordinate Relationships: Ball

| Position | | Velocity | |
|---|---|---|---|
| Angle | $\theta_x + n\phi_x$ | Angular velocity | $\dot{\theta_x} + n\dot{\phi_x}$ |
| x position, $x_b$ | $R_b(\theta_x + n\phi_x)$ | x velocity, $\dot{x_b}$ | $R_b(\dot{\theta_x} + n\dot{\phi_x})$ |
| z position, $z_b$ | 0 | z velocity, $\dot{z_b}$ | 0 |

Linear Kinetic Energy of the Ball, $T_{linb}$ =

$$\frac{M_b\, {R_b}^2 \left(\dot{\theta_x} + n\,\dot{\phi_x}\right)^2}{2} \tag{23}$$

Rotational Kinetic Energy of the Ball, $T_{rotb}$ =

$$\frac{I_b \left(\dot{\theta_x} + n\,\dot{\phi_x}\right)^2}{2} \tag{24}$$

Potential Energy of the Ball, $V_b = 0$

The Body:

Table 9: Generalised Coordinate Relationships: Body

| Position | | Velocity | |
|---|---|---|---|
| Angle | $\theta_x$ | Angular Velocity | $\dot{\theta_x}$ |
| x position, $x_B$ | $x_b + L\sin(\theta_x)$ | x velocity, $\dot{x_B}$ | $\dot{x_b} + L\cos(\theta_x)\dot{\theta_x}$ |
| z position, $z_B$ | $L\cos(\theta_x)$ | z velocity, $\dot{z_B}$ | $-L\sin(\theta_x)\dot{\theta_x}$ |

Linear Kinetic Energy of the Body, $T_{linB}$ =

$$\frac{M_B \left(L^2\, {\dot{\theta_x}}^2 + 2\,\cos(\theta_x)\; L\, n\, R_b\, \dot{\phi_x}\, \dot{\theta_x} + 2\,\cos(\theta_x)\; L\, R_b\, {\dot{\theta_x}}^2 + n^2\, {R_b}^2\, {\dot{\phi_x}}^2 + 2\, n\, {R_b}^2\, \dot{\phi_x}\, \dot{\theta_x} + {R_b}^2\, {\dot{\theta_x}}^2\right)}{2} \tag{25}$$

Rotational Kinetic Energy of the Body, $T_{rotB}$ =

$$\frac{I_{Bx}\,\dot{\theta_x}^2}{2} \tag{26}$$

Potential Energy of the Body, $V_B$

$$g\,L\,M_B\,\cos(\theta_x) \tag{27}$$

The Motors:

Table 10: Generalised Coordinate Relationships: Motors

| Position | | Velocity | |
|---|---|---|---|
| Angle | $\theta_x + \phi_x$ | Angular velocity | $\dot{\theta_x} + \dot{\phi_x}$ |

Rotational Kinetic Energy of the Motors, $T_{rotm}$ =

$$\frac{I_M\left(\dot{\phi_x} + \dot{\theta_x}\right)^2}{2} \tag{28}$$

The Lagrangian is then calculated as

$$L = T_{linb} + T_{linB} + T_{rotb} + T_{rotB} + T_{rotm} - V_b - V_B \tag{29}$$

The Euler-Lagrange equations are then applied as follows

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q_i}}\right) - \frac{\partial L}{\partial q_i} = F_i \tag{30}$$

where

$$\mathbf{q} = [\theta_x\ \phi_x]^T$$

The force matrix was determined by applying Equation (5) and the effects of viscous friction.

As such, $F$ =

$$\begin{pmatrix} -\mu_{Bg}\,\dot{\theta_x} \\ \frac{K_t\,v_x}{R_m} - \mu_{Bb}\,\dot{\phi_x} - \frac{K_b\,K_t\,\dot{\phi_x}}{R_m} \end{pmatrix} \tag{31}$$

The resulting equations of motion are of the form

$$\mathbf{M_x}(\mathbf{q}, \dot{\mathbf{q}})\ddot{\mathbf{q}} + \mathbf{C_x}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G_x}(\mathbf{q}) + \mathbf{D_x}(\dot{\mathbf{q}}) = \mathbf{F_x}(\mathbf{q}, \dot{\mathbf{q}}, v_x) \tag{32}$$

where the *Mass* matrix,$\mathbf{M_x}$ =

$$\begin{pmatrix} I_{Bx} + I_M + I_b + L^2\,M_B + M_B\,{R_b}^2 + M_b\,{R_b}^2 + 2\,L\,M_B\,R_b\,\cos(\theta_x) & I_M + I_b\,n + M_B\,n\,{R_b}^2 + M_b\,n\,{R_b}^2 + L\,M_B\,n\,R_b\,\cos(\theta_x) \\ I_M + I_b\,n + M_B\,n\,{R_b}^2 + M_b\,n\,{R_b}^2 + L\,M_B\,n\,R_b\,\cos(\theta_x) & I_M + I_b\,n^2 + M_B\,n^2\,{R_b}^2 + M_b\,n^2\,{R_b}^2 \end{pmatrix}$$

the *Coriolis* matrix, $\mathbf{C_x}$ =

$$\begin{pmatrix} -L\,M_B\,R_b\,\sin(\theta_x)\,\dot{\theta_x}^2 \\ -L\,M_B\,n\,R_b\,\dot{\theta_x}^2\,\sin(\theta_x) \end{pmatrix}$$

the *Gravity* matrix, $\mathbf{G_x}$ =

$$\begin{pmatrix} -g\,L\,M_B\,\sin(\theta_x) \\ 0 \end{pmatrix}$$

the *Friction* matrix, $\mathbf{D_x}$ =

$$\begin{pmatrix} \mu_{Bg}\dot{\theta_x} \\ \mu_{Bb}\dot{\phi_x} \end{pmatrix}$$

and the *Force* matrix, $\mathbf{F_x}$ =

$$\begin{pmatrix} 0 \\ \frac{K_t(v_x - K_b\dot{\phi_x})}{R_m} \end{pmatrix}$$

This can be rearranged to

$$\ddot{\mathbf{q}} = \mathbf{M_x}^{-1}(\mathbf{F_x} - (\mathbf{C_x} + \mathbf{G_x} + \mathbf{D_x})) \tag{33}$$

Which can then written in standard non-linear state space form:

Following this the non-linear state space equations $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ can be derived by taking $\mathbf{x} = [\mathbf{q}\dot{\mathbf{q}}]^T$ and $u = v_x$, as follows

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}) \\ &= f(\mathbf{q}, \dot{\mathbf{q}}, v_x) \\ &= \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{v_x}) \end{bmatrix} \end{aligned} \tag{34}$$

As such, the state vector $\mathbf{x}$ for this model consists of

- $\theta_x$ - The angle of the body of the ballbot relative to the vertical in the x-plane
- $\phi_x$ - The angle of the motor shaft relative to the body in the x-plane
- $\dot{\theta_x}$ - The angular velocity of the body of the ballbot in the x-plane
- $\dot{\phi_x}$ - The angular velocity of the motor shaft relative to the body in the x-plane

and the control signal $\mathbf{u}$ is the motor voltage for the x motor, $v_x$.

The non-linear state space equations are then linearised using a first order approximation as follows

$$\hat{\dot{\mathbf{x}}} = f(\bar{\mathbf{x}}, v_x) + J(\bar{\mathbf{x}})\hat{\mathbf{x}} \tag{35}$$

where $\hat{\mathbf{x}}$ is the linearised state about the point $\bar{\mathbf{x}} = [0\ \phi_x\ 0\ 0]^T$, and $J(\bar{\mathbf{x}})$ is the Jacobian at that point. Taking the linearised state $\hat{\mathbf{x}}$ to be the state vector results in the linear State Space Model

$$\dot{\mathbf{x}} = \mathbf{A_x}\mathbf{x} + \mathbf{B_x}v_x \tag{36}$$

where we have the following matrices:

$$A_x = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ A_x(3,1) & 0 & A_x(3,3) & A_x(3,4) \\ A_x(4,1) & 0 & A_x(4,3) & A_x(4,4) \end{pmatrix} \tag{37}$$

$$B_x = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ B_x(3,1) & 0 \\ B_x(4,1) & 0 \end{pmatrix} \tag{38}$$

where

$$A_x(3,1) = \frac{g\,L\,M_B\,\left(I_M + I_b\,n^2 + M_B\,n^2\,{R_b}^2 + M_b\,n^2\,{R_b}^2\right)}{D_x}$$

$$A_x(4,1) = -\frac{g\,L\,M_B\,\left(I_M + I_b\,n + M_B\,n\,{R_b}^2 + M_b\,n\,{R_b}^2 + L\,M_B\,n\,R_b\right)}{D_x}$$

$$A_x(3,3) = -\frac{\mu_{Bg}\,\left(I_M + I_b\,n^2 + M_B\,n^2\,{R_b}^2 + M_b\,n^2\,{R_b}^2\right)}{D_x}$$

$$A_x(4,3) = \frac{\mu_{Bg}\,\left(I_M + I_b\,n + M_B\,n\,{R_b}^2 + M_b\,n\,{R_b}^2 + L\,M_B\,n\,R_b\right)}{D_x}$$

$$A_x(3,4) = \frac{\left(\mu_{Bb} + \frac{K_b\,K_t}{R_m}\right)\,\left(I_M + I_b\,n + M_B\,n\,{R_b}^2 + M_b\,n\,{R_b}^2 + L\,M_B\,n\,R_b\right)}{D_x}$$

$$A_x(4,4) = -\frac{\left(\mu_{Bb} + \frac{K_b\,K_t}{R_m}\right)\,\left(I_{Bx} + I_M + I_b + L^2\,M_B + M_B\,{R_b}^2 + M_b\,{R_b}^2 + 2\,L\,M_B\,R_b\right)}{D_x}$$

$$B_x(3,1) = -\frac{K_t\,\left(I_M + I_b\,n + M_B\,n\,{R_b}^2 + M_b\,n\,{R_b}^2 + L\,M_B\,n\,R_b\right)}{D_x\,R_m}$$

$$B_x(4,1) = \frac{K_t\,\left(I_{Bx} + I_M + I_b + L^2\,M_B + M_B\,{R_b}^2 + M_b\,{R_b}^2 + 2\,L\,M_B\,R_b\right)}{D_x\,R_m}$$

$$\begin{aligned} D_x = {} & I_{Bx}\,I_M + I_M\,I_b - 2\,I_M\,I_b\,n + I_{Bx}\,I_b\,n^2 + I_M\,I_b\,n^2 + I_M\,L^2\,M_B + I_M\,M_B\,{R_b}^2 + I_M\,M_b\,{R_b}^2 \\ & + I_b\,L^2\,M_B\,n^2 + I_{Bx}\,M_B\,n^2\,{R_b}^2 + I_M\,M_B\,n^2\,{R_b}^2 + I_{Bx}\,M_b\,n^2\,{R_b}^2 + I_M\,M_b\,n^2\,{R_b}^2 \\ & + 2\,I_M\,L\,M_B\,R_b - 2\,I_M\,M_B\,n\,{R_b}^2 - 2\,I_M\,M_b\,n\,{R_b}^2 + L^2\,M_B\,M_b\,n^2\,{R_b}^2 - 2\,I_M\,L\,M_B\,n\,R_b \end{aligned} \tag{39}$$

# B Matlab Code

This appendix includes the code used to generate the equations of motion for a simplified one plane ballbot system. Additionally, the code also prints out and displays the equations in a form suitable for inclusion into this report.

```
clear all

%% Variable Declaration
latexsyms Rb R_b L L                    %dimensions: ball radius, height of body COM
latexsyms Mb M_b MB M_B                   %mass: ball, Body
latexsyms Ib I_b IBx I_{Bx} IBy  I_{By} IBz I_{Bz} IM I_M      %Moments of inertia: ball, Body (x,y,z), motors
latexsyms G g                        %gravity
latexsyms N n                       %gear ratio motors to ball (N = Rb/(motor wheel radius))
latexsyms FbBx \mu_{Bb} FBgx \mu_{Bg}                 %friction b:ball, B:body g:ground !!!!!!!!!CHECK!!!!!!!!
latexsyms FbBy \mu_{Bb} FBgy \mu_{Bg}                 %friction b:ball, B:body g:ground !!!!!!!!!CHECK!!!!!!!!
latexsyms Kb K_b Kt K_t Rm R_m                   %electrical paramters: back emf, torque constant and resistance
syms O

%% X Z plane %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Function Variables
latexsyms thetaXddot \ddot{\theta_x}   thetaXdot \dot{\theta_x} thetaX \theta_x
latexsyms phiXddot \ddot{\phi_x} phiXdot \dot{\phi_x} phiX \phi_x

%vectors
syms q_x q_x_dot q_x_double_dot
q_x = [thetaX phiX];
q_x_dot = [thetaXdot phiXdot];
q_x_double_dot = [thetaXddot phiXddot];

%%%%%%%%% Ball Energy %%%%%%%%%%%%%
syms ball_x
syms ball_dot_x
syms T_lin_ball_x T_rot_ball_x V_ball_x

ball_x(1) = Rb*(thetaX + N*phiX);
ball_x(2) = 0;
ball_x(3) = 0;

ball_dot_x(1) = diff(ball_x(1),thetaX)*thetaXdot + diff(ball_x(1),phiX)*phiXdot
ball_dot_x(2) = diff(ball_x(2),thetaX)*thetaXdot + diff(ball_x(2),phiX)*phiXdot
ball_dot_x(3) = diff(ball_x(3),thetaX)*thetaXdot + diff(ball_x(3),phiX)*phiXdot

T_lin_ball_x = simple(Mb/2*(ball_dot_x(1)^2+ball_dot_x(2)^2+ball_dot_x(3)^2))
T_rot_ball_x = simple(Ib/2*(thetaXdot+N*phiXdot)^2)
V_ball_x = 0

%%%%%%%%%%%%% Body Energy %%%%%%%%%%%%
syms body_x
syms body_offset_x
syms body_dot_x
syms T_lin_body_x T_rot_body_x V_body_x

body_offset_x(1) = L*sin(thetaX);
body_offset_x(2) = 0;
body_offset_x(3) = L*(cos(thetaX));
```

```
body_x(1) = ball_x(1) + body_offset_x(1);
body_x(2) = ball_x(2) + body_offset_x(2);
body_x(3) = ball_x(3) + body_offset_x(3);

body_dot_x(1) = simple(diff(body_x(1),thetaX)*thetaXdot ...
                    + diff(body_x(1),phiX)*phiXdot)
body_dot_x(2) = simple(diff(body_x(2),thetaX)*thetaXdot ...
                    + diff(body_x(2),phiX)*phiXdot)
body_dot_x(3) = simple(diff(body_x(3),thetaX)*thetaXdot ...
                    + diff(body_x(3),phiX)*phiXdot)

T_lin_body_x = simple(expand(simple((MB/2*(body_dot_x(1)^2+body_dot_x(2)^2+body_dot_x↙
(3)^2)))))
T_rot_body_x = simple(expand(simple((IBx/2*(thetaXdot)^2))))
V_body_x = simple(MB*G*body_x(3))

%%%%%%%%%%%%%%%% Motors energy %%%%%%%%%%%%%%%
syms T_rot_motors_x

T_rot_motors_x = simple(IM/2*(thetaXdot+phiXdot)^2)

%%%%%%%%%%%%%%%   Lagrangian and EL Equations %%%%%%%%%%%
latexsyms    LagrangianX Lagrangian_x
syms    dL_dtheta_x_dot         dL_dphi_x_dot
syms    d_dL_dtheta_x_dot_dt    d_dL_dphi_x_dot_dt
syms    dL_dtheta_x             dL_dphi_x
syms    EL_theta_x              EL_phi_x

LagrangianX = T_lin_ball_x + T_rot_ball_x + T_lin_body_x + T_rot_body_x +↙
T_rot_motors_x - V_ball_x - V_body_x

%%EL Equations

%theta_x
dL_dtheta_x_dot = simple(expand(simple(diff(LagrangianX, thetaXdot))));
d_dL_dtheta_x_dot_dt =  simple(expand(simple( ...
                        diff(dL_dtheta_x_dot, thetaXdot)*thetaXddot ...
                        + diff(dL_dtheta_x_dot, phiXdot)*phiXddot ...
                        + diff(dL_dtheta_x_dot, thetaX)*thetaXdot ...
                        + diff(dL_dtheta_x_dot, phiX)*phiXdot ...
                            )));
dL_dtheta_x = simple(expand(simple(diff(LagrangianX, thetaX))));
EL_theta_x = simple(expand(simple(d_dL_dtheta_x_dot_dt - dL_dtheta_x)))

%phiX
dL_dphi_x_dot = simple(expand(simple(diff(LagrangianX, phiXdot))));
d_dL_dphi_x_dot_dt =  simple(expand(simple( ...
                        diff(dL_dphi_x_dot, thetaXdot)*thetaXddot ...
                        + diff(dL_dphi_x_dot, phiXdot)*phiXddot ...
                        + diff(dL_dphi_x_dot, thetaX)*thetaXdot ...
                        + diff(dL_dphi_x_dot, phiX)*phiXdot ...
                            )));
dL_dphi_x = simple(expand(simple(diff(LagrangianX, phiX))));
EL_phi_x = simple(expand(simple(d_dL_dphi_x_dot_dt - dL_dphi_x)))
```

```
%%%%%%%% System dynamics as Matrices %%%%%%%%%%%%%%%%%

% Dynamics represented as: M*q(double dot) + R = F where F is matrix of forces
% ==> q(double dot) = (M^-1)*(F-R)     (this is what must be linearised)

%%%Make M and R matrix

syms temp M_x R_x F_x

for i = 1:2

    switch i
        case 1,
            temp = EL_theta_x;
        case 2,
            temp = EL_phi_x;
    end

    [c,t] = coeffs(temp, thetaXddot);
    M_x(i,1) = 0;
    remainder = 0;
    for j=1:length(c)
        if t(j)== thetaXddot
            M_x(i,1) = c(j);
        elseif t(j)== 1
            remainder = c(j);
        end
    end
    [c,t] = coeffs(remainder, phiXddot);
    M_x(i,2) = 0;
    R_x(i,1) = 0;
    for j=1:length(c)
        if t(j)== phiXddot
            M_x(i,2) = c(j);
        elseif t(j)== 1
            R_x(i,1) = c(j);
        end
    end
end

%%% Force matrix definition
latexsyms voltX v_x                    %volts
latexsyms iX i_x                       %currents

%motor current dynamics    !!!!!!!!!!!!!!!CHECK!!!!!!!!!!!!!!!
iX = 1/Rm*(voltX - Kb*phiXdot);

%F          !!!!!check!!!!!
F_x = [   -FBgx*thetaXdot ;...
        expand(Kt*iX - FbBx*phiXdot)];

syms non_linear_q_x_double_dot_RHS
non_linear_q_x_double_dot_RHS = inv(M_x)*(F_x-R_x);

%%Linearise
syms x x_bar
```

```
syms non_linear_state_matrix_x non_linear_state_matrix_x_bar
syms Jacobian_Matrix_x Jacobian_Matrix_x_bar
syms linear_q_x_dot_RHS

x = [q_x q_x_dot];
x_bar = [0, phiX, 0, 0];

non_linear_state_matrix_x = [   thetaXdot;...
                                phiXdot;...
                                non_linear_q_x_double_dot_RHS    ];

non_linear_state_matrix_x_bar = subs(non_linear_state_matrix_x, x, x_bar);

Jacobian_Matrix_x = jacobian(non_linear_state_matrix_x, x);
Jacobian_Matrix_x_bar = subs(Jacobian_Matrix_x, x, x_bar);

linear_q_x_dot_RHS = non_linear_state_matrix_x_bar + Jacobian_Matrix_x_bar*(transpose↙
(x));

%%% create State matrices
%state is trans([q_x q_x_dot])

syms A_x B_x

for i = 1:4
    remainder = linear_q_x_dot_RHS(i);
    for j = 1:4
        [c,t] = coeffs(remainder, x(j));
        A_x(i,j) = 0;
        for k = 1:length(c)
            if t(k) == 1
                remainder = c(k);
            else
                A_x(i,j) = c(k);
            end
        end
    end
    [c,t] = coeffs(remainder, voltX);
    B_x(i,1) = 0;
    remainder = 0;
    for k = 1:length(c)
        if t(k)==1
            remainder = c(k);
        else
            B_x(i,1) = c(k);
        end
    end

end

A_x
B_x
remainder
```

```
%% Print stuff
fid = fopen('ballbotdynamicsequationoutput.tex','w');
%fwrite(fid,['\paragraph{} File generated at ',datestr(now)]);

fprintf(fid,['Lagrangian'])
fprintf(fid,['\\begin{dmath} L = T_{lin b} + T_{lin B} + T_{rot b} + T_{rot B} + T_{rot m} - V_{b} - V_{B} \\end{dmath} \n'])

fprintf(fid,['Euler Lagrange Equation'])
fprintf(fid,['\\begin{dmath} \\frac{d}{dt}\\Bigl\\lgroup\\frac{\\partial L}{\\partial \\dot{q_i}}\\Bigr\\rgroup - \\frac{\\partial L}{\\partial q_i} = F_i \\end{dmath}\n'])
fprintf(fid,['where \n $${\\bf q} = [\\theta_x \\; \\phi_x]^T$$ \n\n\paragraph{}and $F_i$ are the generalised forces\n\n'])

fprintf(fid,['\\paragraph{} The resulting equations of motion can then be expressed in the form: \n\\begin{dmath}\n$$M_x({\\bf q},{\\bf \\dot{q}}) {\\bf \\ddot{q}} + R_x ({\\bf q},{\\bf \\dot{q}}) = F_x({\\bf q},{\\bf \\dot{q}},{\\bf v_x}) $$ \n\\end{dmath} \n\n'])
fprintf(fid,['\\paragraph{} The where $M_x$ is the \\emph{Mass} matrix, $F_x$ is the \\emph{Force} matrix and $R_x$ the \\emph{Remainder} matrix. \n\n'])

fprintf(fid,['\\paragraph{} The $M_x$, $F_x$ and $R_x$ matrices are as follows\n\n'])
fprintf(fid,['\\paragraph{} Mass Matrix, $M_x$=\n'])
fprintf(fid,['\\begin{dmath} \\left( \\begin{array}{cc} \n M_x(1,1) & M_x(1,2) \\\\ \n M_x(2,1) & M_x(2,2) \\\\ \n \\end{array} \\right) \n \\end{dmath}'])
fprintf(fid,['where'])
write_variables2(fid,M_x(1,1),'$M_x(1,1)$ =',LatexSymbolTable)
write_variables2(fid,M_x(1,2),'$M_x(1,2)$ =',LatexSymbolTable)
write_variables2(fid,M_x(2,1),'$M_x(2,1)$ =',LatexSymbolTable)
write_variables2(fid,M_x(2,2),'$M_x(2,2)$ =',LatexSymbolTable)
write_variables(fid,R_x,'Remainder Matrix, $R_x$ =',LatexSymbolTable)
fprintf(fid,['\\paragraph{} Force Matrix, $F_x$ = \n$$F$$\n'])

fprintf(fid,['\\paragraph{} This can be rearranged to \n\n \\begin{dmath}{\\bf \\ddot{q}} = M_x^{-1}(F_x-R_x)\\end{dmath}\n'])
fprintf(fid,['\\paragraph{} Which can then written in standard non-linear state space form:\n\n \\begin{dmath}\n{\\bf \\dot{x}} = f({\\bf x})\n\\end{dmath}\n where \n $${\\bf x} = [\\theta_x \\; \\phi_x \\; \\dot{\\theta_x} \\; \\dot{\\phi_x}]^T$$\n\n'])

fprintf(fid,['\\paragraph{} The state space controller used in this project requires that the state space equations be linear. Thus, the non-linear state space equations are linearised using a first order approximation as follows \n\\begin{dmath}\n $${\\bf \\hat{\\dot{x}}} = f({\\bf \\bar{x}}) + J({\\bf \\bar{x}}){\\bf \\hat{x}}$$ \n\\end{dmath}\n'])
fprintf(fid,['where ${\\bf \\hat{x}}$ is the linearised state, ${\\bf \\bar{x}} = [0 \\; \\phi_x \\; 0 \\; 0]^T$ is the point we linearise about, and $J({\\bf \\bar{x}})$ is the Jacobian at that point \n\n'])

[N,D] = numden(A_x(3,1))

fprintf(fid,['\\paragraph{} Writing in Linear State Space Form, and taking ${\\bf \\hat{x}}$ to be the state vector ${\\bf x}$\n\n'])
fprintf(fid,['\\begin{dmath} \n {\\bf \\dot{x}} = A{\\bf x} + B{\\bf u} \n \\end{dmath} \n where we have the following matrices:'])
fprintf(fid,['\\begin{dmath} \n A_x = \\left( \\begin{array}{cccc} \n 0 & 0 & 1 & 0\\\\ \n 0 & 0 & 0 & 1\\\\ \n A_x(3,1) & 0 & A_x(3,3) & A_x(3,4)\\\\ \n A_x(4,1) & 0 & A_x
```

```
(4,3) & A_x(4,4)\\\\ \n \\end{array} \\right) \n \\end{dmath}'])
fprintf(fid,['\\begin{dmath} \n B_x = \\left( \\begin{array}{cc} \n 0 & 0 \\\\ \n 0 & 0↙
\\\\ \n B_x(3,1) & 0 \\\\ \n B_x(4,1) & 0 \\\\ \n \\end{array} \\right) \n \\end↙
{dmath}'])

latexsyms Dx D_x

fprintf(fid,['where\n\n'])
write_variables2(fid,A_x(3,1)*D/Dx,'${A_x}(3,1)$ =',LatexSymbolTable)
write_variables2(fid,A_x(4,1)*D/Dx,'${A_x}(4,1)$ =',LatexSymbolTable)
write_variables2(fid,A_x(3,3)*D/Dx,'${A_x}(3,3)$ =',LatexSymbolTable)
write_variables2(fid,A_x(4,3)*D/Dx,'${A_x}(4,3)$ =',LatexSymbolTable)
write_variables2(fid,A_x(3,4)*D/Dx,'${A_x}(3,4)$ =',LatexSymbolTable)
write_variables2(fid,A_x(4,4)*D/Dx,'${A_x}(4,4)$ =',LatexSymbolTable)
write_variables2(fid,B_x(3,1)*D/Dx,'${B_x}(3,1)$ =',LatexSymbolTable)
write_variables2(fid,B_x(4,1)*D/Dx,'${B_x}(4,1)$ =',LatexSymbolTable)
write_variables(fid,D,'$D_x$ =',LatexSymbolTable)

fprintf(fid,['\\paragraph{} It should be noted that these equations are those for the↙
x-z plane only. The equations for the y-z plane are identical, with appropriate↙
subscript changes.\n\n'])

fclose(fid)

close all

%% Lego Ballbot
Rb=0.026;
L=0.132;
Mb =0.015;
MB=0.767;
Ib=6.76*10^-6;
IBx=0.0054;
IM =2*10^-5; %%%%%%%%%%%%%
G =9.81;
N = (-0.0186/Rb); %%%%%%%%%%%%%%%%%%%%%%%
FbBx= 0.0022;  %%
FBgx =0;
Kb =0.468;
Kt =0.317;
Rm=6.69;

A_sub=subs(A_x);
B_sub=subs(B_x);
C_sub = eye(4);
D_sub = zeros(4,1);

%controllability
Co = ctrb(A_sub,B_sub);
UncontrollableStates = length(A_sub)-rank(Co)
Condition = cond(Co)

%add two reference states for thetaX and thetaY (integral control)
C = [ 0 1 0 0 ]
A_bar = [   A_sub    zeros(4,1)  ;...
            C        zeros(1,1)  ];
```

```
B_bar = [   B_sub       ;...
            zeros(1,1)  ];
Co2 = ctrb(A_bar,B_bar);
UncontrollableStates2 = length(A_bar)-rank(Co2)
Condition2 = cond(Co2)

% LQR control
Q = eye(5);
Q(1,1) = 6e5;
Q(5,5) = 4e2;
R = (1e3)*eye(1);

K = lqr(A_bar, B_bar, Q, R)

Kf = K(1:4)
Ki = K(5)

OpenLoopPoles = eig(A_bar)
ClosedLoopPoles = eig(A_bar-B_bar*K)

%% FS ballbot
Rb=0.11;
L=.8;
Mb =5.5;
MB=29;
Ib=0.0266;
IBx=5.6;
IM =4.62*10^-5;
G =9.81;
N = (-0.223);
FbBx= 0.2;  %%
FBgx =0.0;
Kb =0.04768;
Kt =0.0742*2;
Rm=1.11;

A_sub=subs(A_x);
B_sub=subs(B_x);
C_sub = eye(4);
D_sub = zeros(4,1);

%controllability
Co = ctrb(A_sub,B_sub);
UncontrollableStates = length(A_sub)-rank(Co)
Condition = cond(Co)

%add two reference states for thetaX and thetaY (integral control)
C = [ 0 1 0 0 ]
A_bar = [   A_sub   zeros(4,1)  ;...
            C       zeros(1,1)  ];
B_bar = [   B_sub       ;...
            zeros(1,1)  ];
Co2 = ctrb(A_bar,B_bar);
UncontrollableStates2 = length(A_bar)-rank(Co2)
Condition2 = cond(Co2)
```

```
% LQR control
Q = eye(5);
Q(1,1) = 6e5;
Q(5,5) = 4e2;
R = (1e3)*eye(1);

K = lqr(A_bar, B_bar, Q, R)

Kf = K(1:4)
Ki = K(5)

OpenLoopPoles = eig(A_bar)
ClosedLoopPoles = eig(A_bar-B_bar*K)
```

# C NXT Mindstorms Component Specifications

This Appendix contains the specifications of the parts available in the Lego Mindstorms NXT range which are relevant to this project.

**NXT Brick** The NXT Brick is the processor used in the Lego NXT system. The brick is normally programmed using Lego NXT software provided in the 8527 kit. However, alternative methods are available including the use of MATLAB's Simulink Real Time workshop and RobotC. The NXT Brick has the following specifications:

- Main processor: Atmel 32-bit 48 MHz ARM processor, with 256 KB FLASH and 64 KB RAM
- Co-processor: Atmel 8-bit 8 MHz AVR processor, with 4 KB FLASH and 512 Byte RAM.
- I/O: Four input interfaces, and three output interfaces for servo motors, which also support input from encoders.
- Power source: 9V (using Alkaline AA batteries or a 1400 mAH rechargeable Lithium-Ion Battery)

**Servos Motors** Three 9V DC servo motors are included in the 8527 kit, with built in encoders. These provide the available actuation system to drive the ball of the Lego Ballbot. The motors have the following parameters:

- No load characteristics at 9V: Speed = 170 rpm, Current = 60 mA
- Stall characteristics at 9V: Torque = 0.050 Nm, Current = 2A
- Torque Constant $K_t$ = 0.317 N/A
- Back Emf Constant $K_b$ = 0.468 Vs
- Terminal Resistance $R_m$ = 6.69 ohm
- Encoder Resolution: 360 counts per turn (1 degree)

**Touch Sensor** The touch sensor included in the 8527 kit provides a binary signal to indicate whether it has been actuated.

**Light Sensor** The light sensors in the NXT range measures light intensity. It can operate in one of two modes. The first simply measures the ambient light intensity. The second uses an included LED to measure the light reflected back from a surface. However, ambient light is not filtered out for the second mode.

**Hitechnic Gyroscopic Sensor** The HiTechnic Gyroscopic sensor contains a single axis gyroscope and can be used to detect rotation in one axis. It measures the rotation rate in degrees per second up to a maximum rate of 360 degrees/sec. It should be noted that the gyroscope contains an offset with needs to be remvoed.

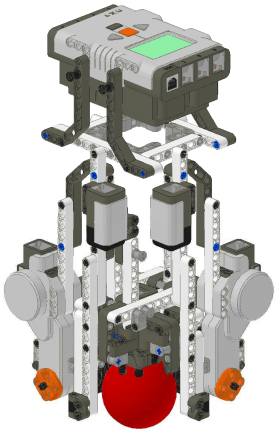
**HiTechnic 3-axis Accelerometer** The HiTechnic 3-Axis Accelerometer can be used to measure acceleration in all three axes, allowing measurement of tilt by detecting the direction of acceleration due to gravity. The resolution is approximately 1/200 g with a range of -2g to +2g .

**Lego Technic Pieces** The 8527 kit contains hundreds of Lego Technic pieces which allowed construction of the Lego Ballbot, including the 52mm diameter ball, frame pieces and various connection elements.

# D Lego Ballbot Building Instructions

The graphical step-by-step building instructions are presented in this appendix.

# Lego Ballbot



**Assembly Instructions**
**Software Installation Instructions**
**Programming Instructions**

Justin Fong, Simon Uppill

## Assembly Instructions

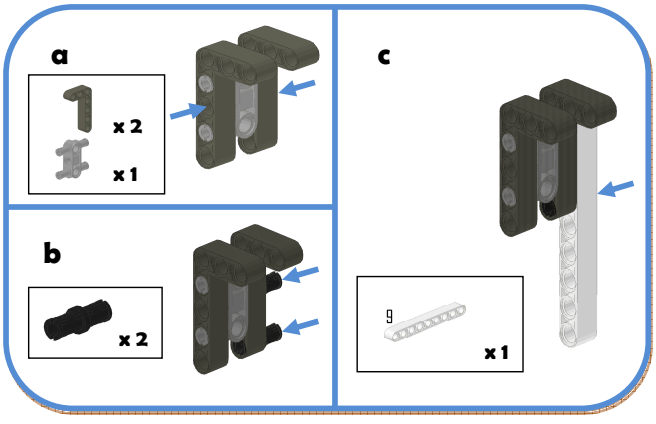
**Construction of the Lego Ballbot requires**

1. **Lego Mindstorms NXT kit (8527)**

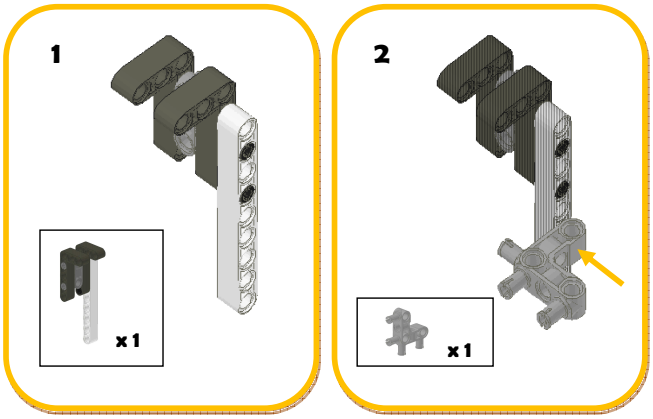

2. **Two HiTechnic Gyroscopic Sensors**

3. **Two 36.8mm diameter Lego Wheels**

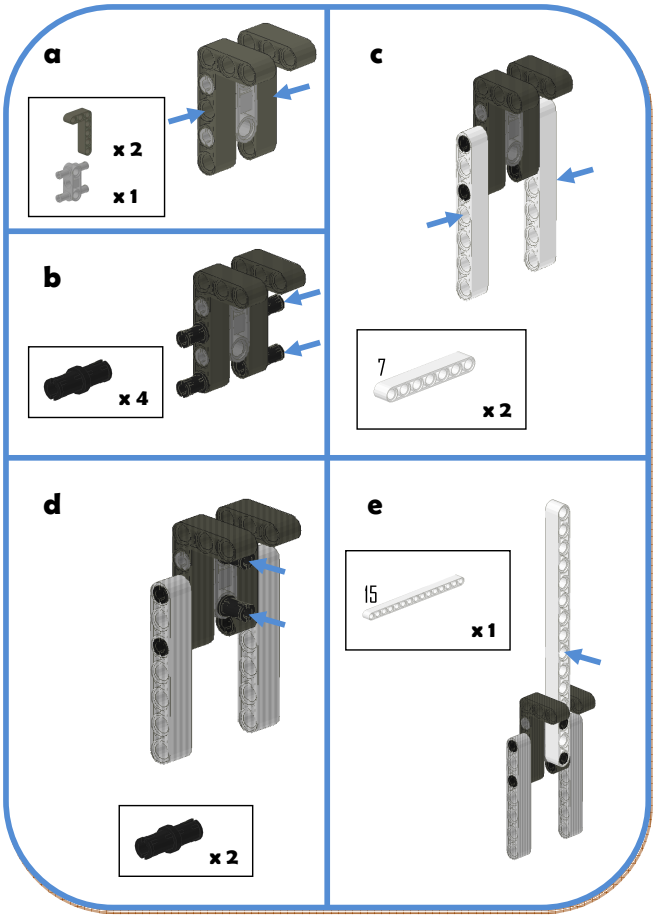
## Subassembly – Claw A




## Ballbot Assembly

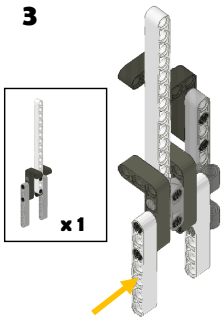



## Subassembly – Claw B

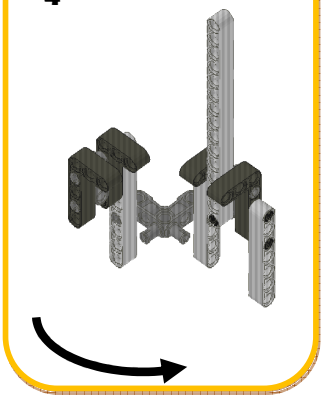


## Ballbot Assembly

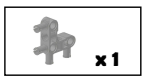
**3**

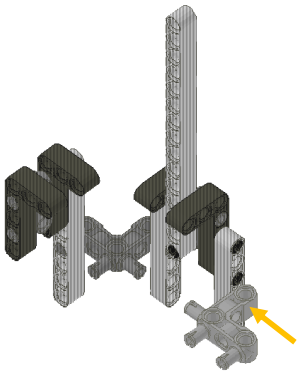



**4**



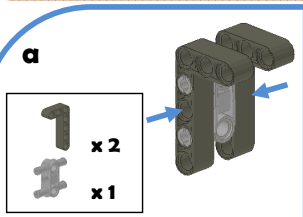
**5**



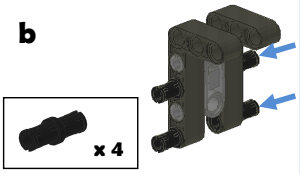



## Subassembly – Claw C

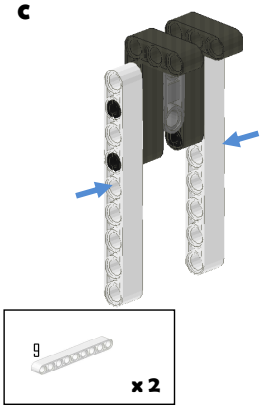
**a**



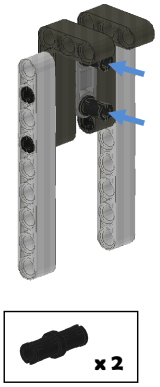

**b**



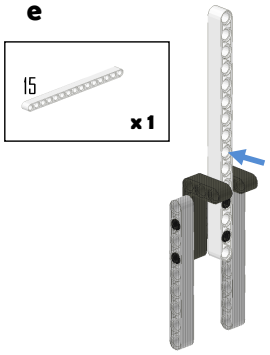

**c**




**d**

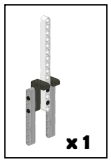


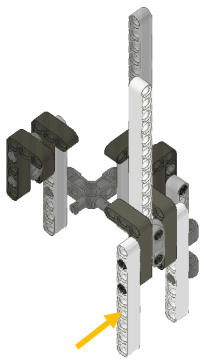

**e**



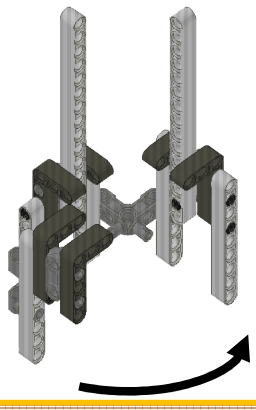

## Ballbot Assembly

**6**



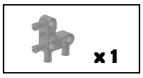



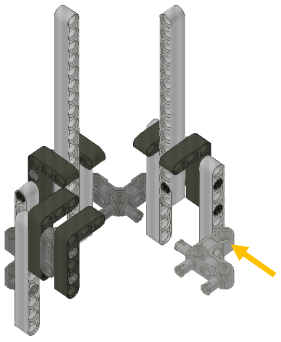
**7**



## Ballbot Assembly

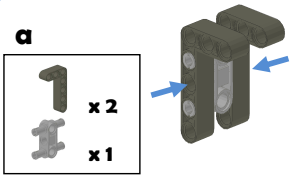
**8**



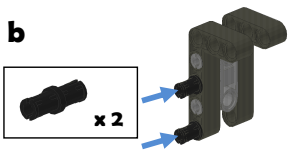



## Subassembly – Claw D

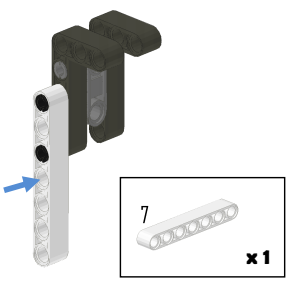
**a**




**b**

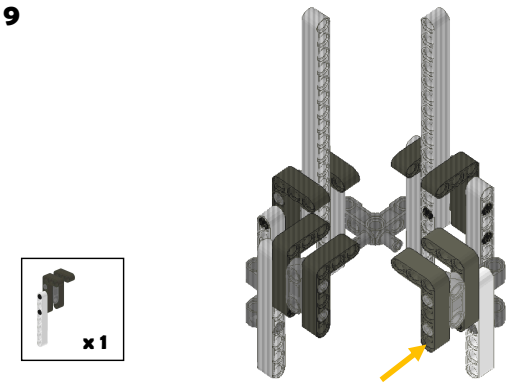



**c**

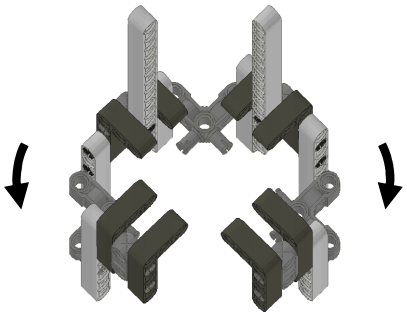


## Ballbot Assembly

**9**

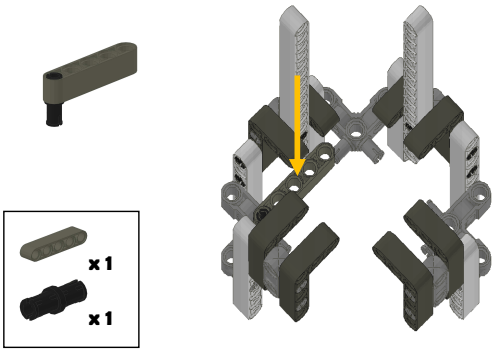



**10**

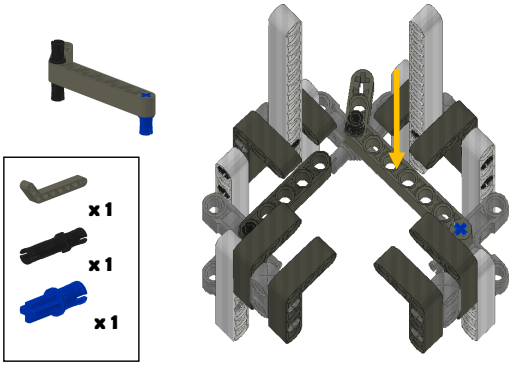


## Ballbot Assembly continued

**11**

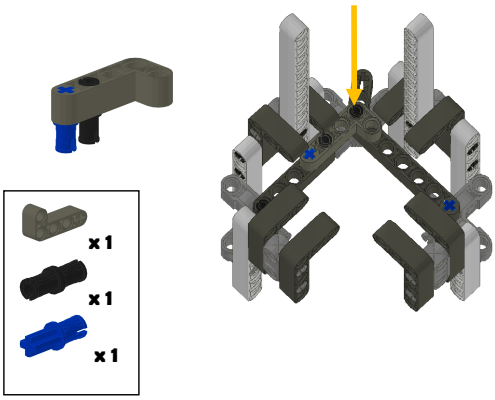



**12**

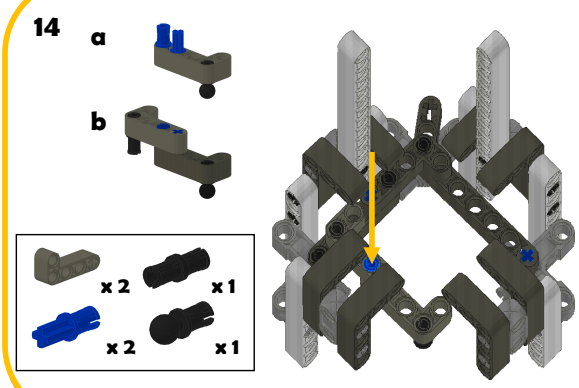



## Ballbot Assembly continued

**13**

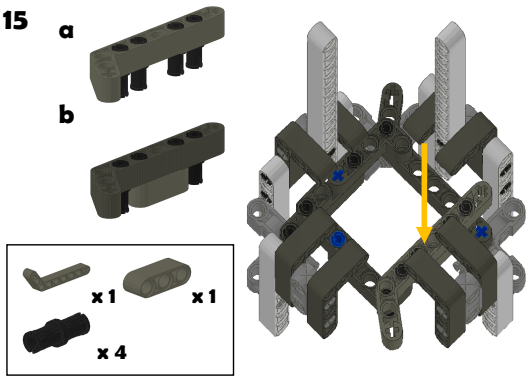



**14**

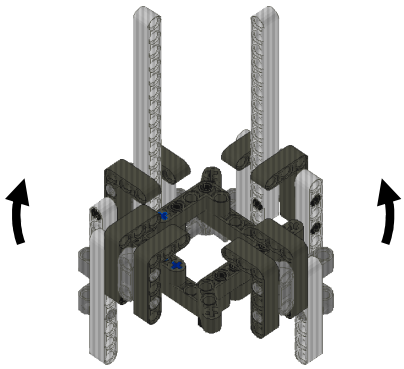



## Ballbot Assembly continued

**15**

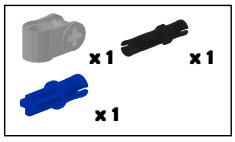


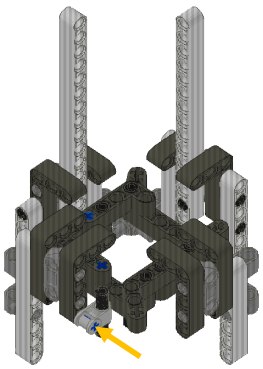

**16**



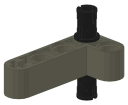
## Ballbot Assembly continued

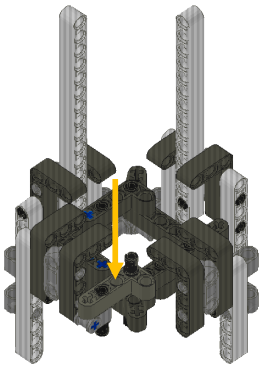
**17**



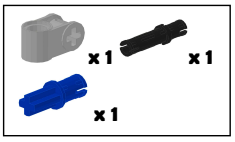
**18**

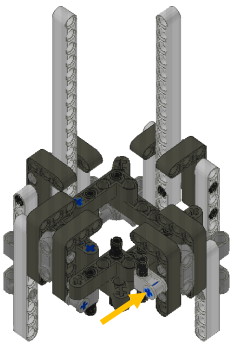


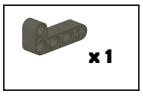
## Ballbot Assembly continued

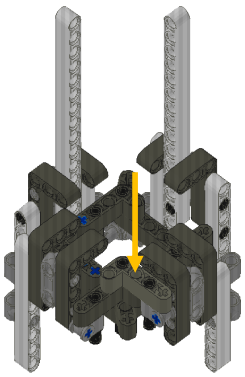
**19**



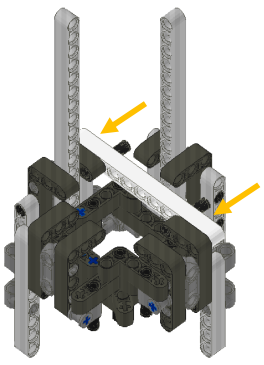
**20**



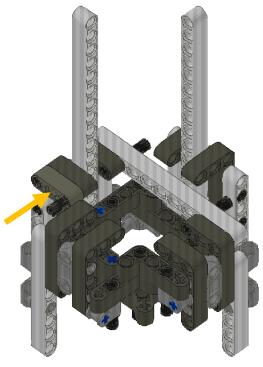



## Ballbot Assembly continued

**21**

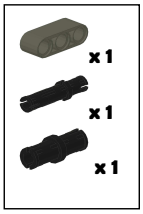



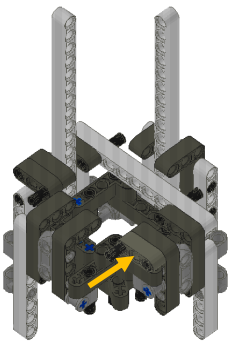
**22**

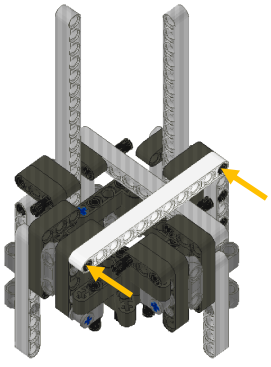
## Ballbot Assembly continued

**23**



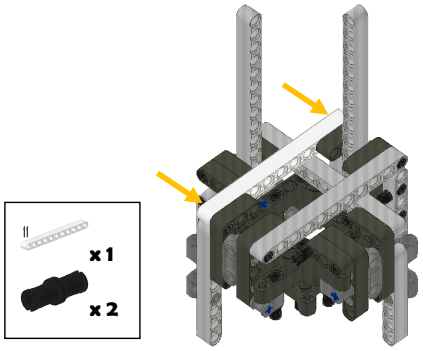



**24**

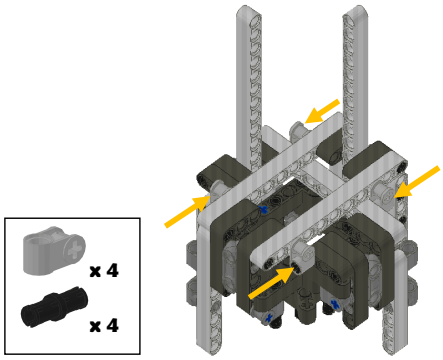



## Ballbot Assembly continued

**25**

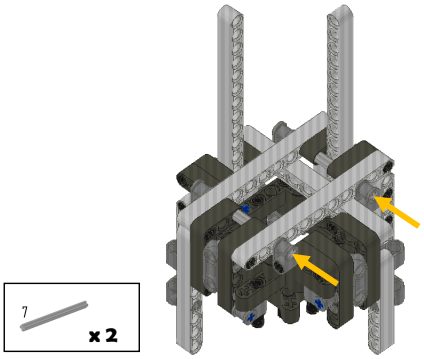



**26**

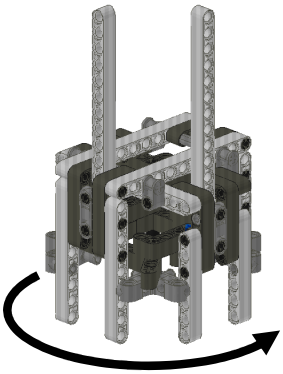



## Ballbot Assembly continued

**27**

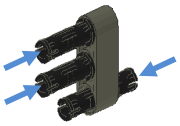



**28**

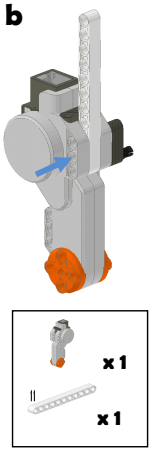


## Subassembly – Servo A

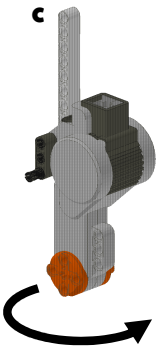
**a**



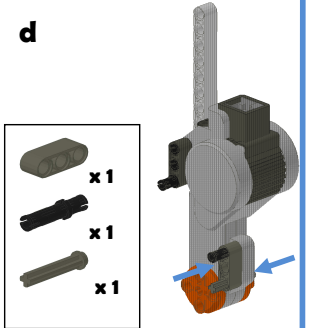

**b**



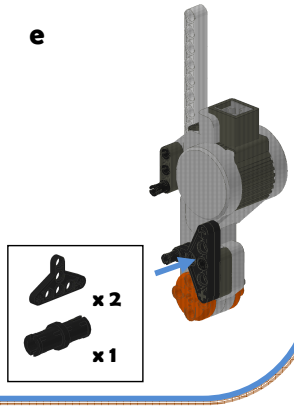

**c**



**d**

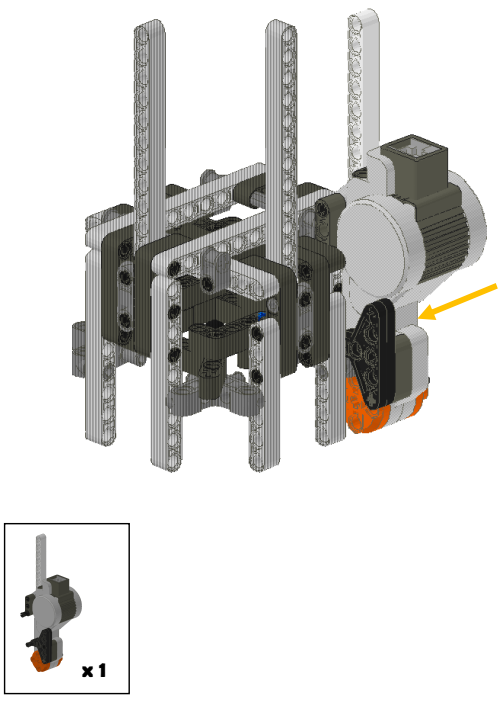



**e**



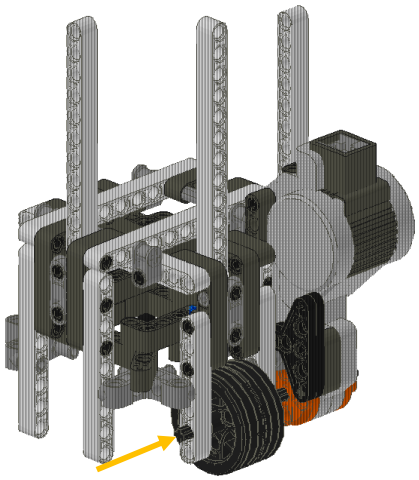

## Ballbot Assembly continued

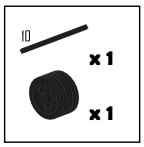
**29**



## Ballbot Assembly continued

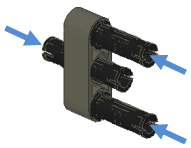
**30**



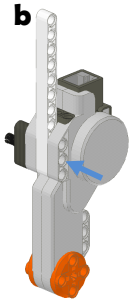


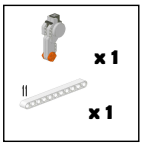

## Subassembly – Servo B

**a**

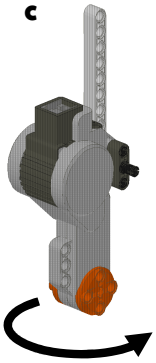



**b**

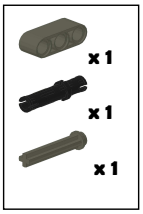


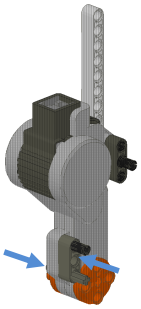



**c**

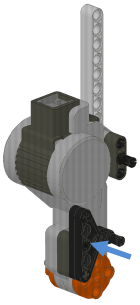


**d**



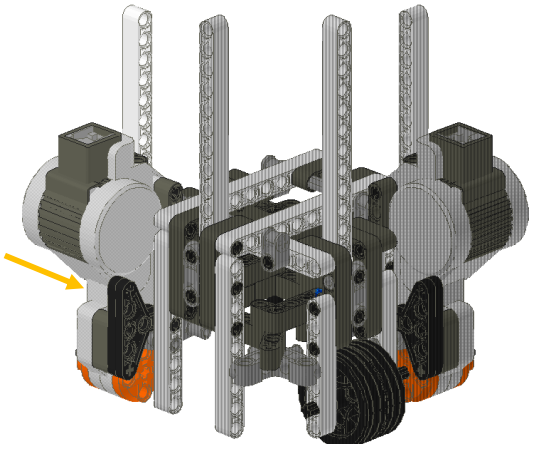



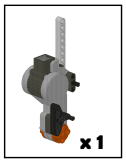
**e**

## Ballbot Assembly continued

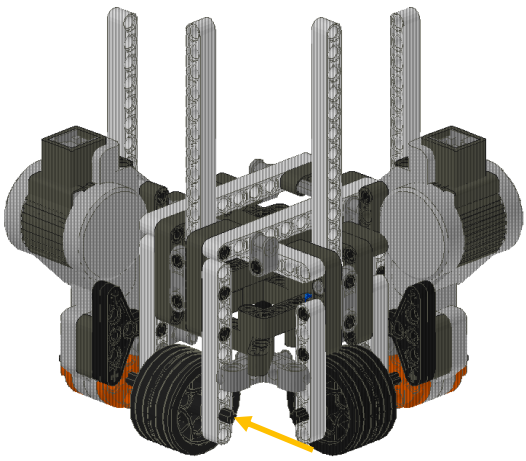
**31**

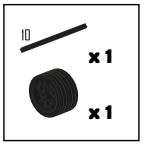





## Ballbot Assembly continued

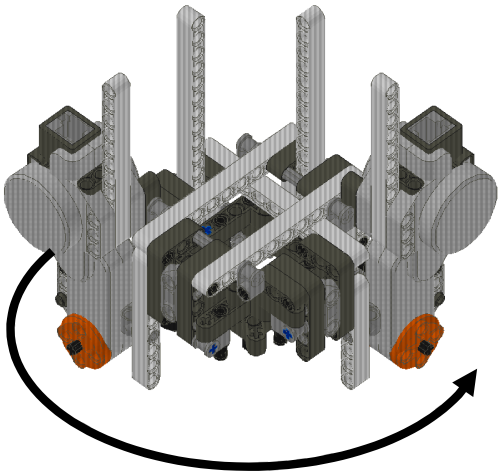
**32**



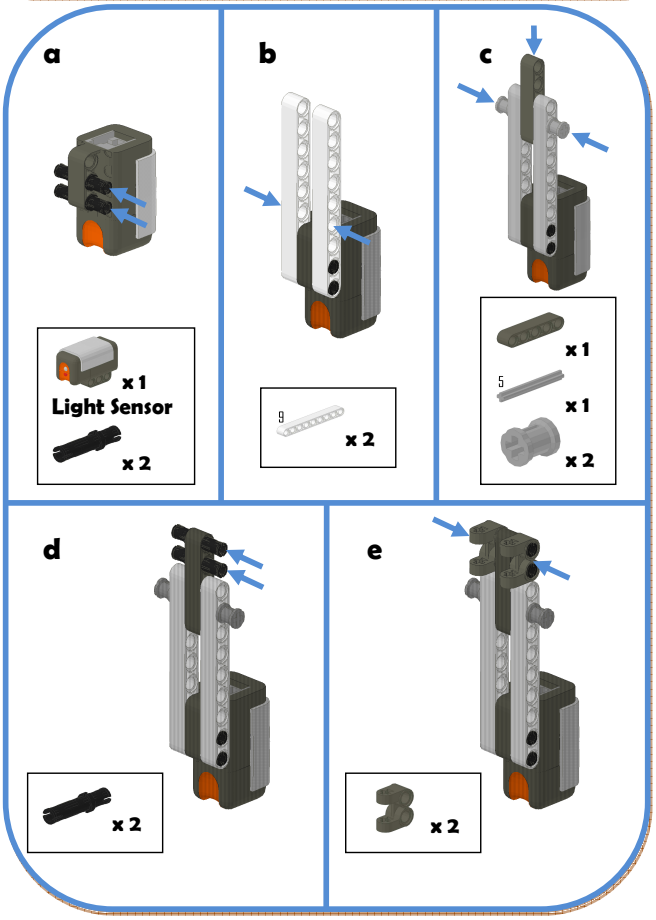


## Ballbot Assembly continued

**33**

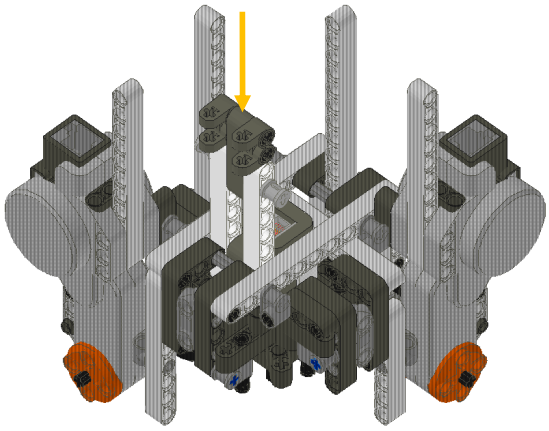


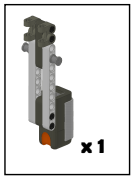
## Subassembly – Light Sensor




## Ballbot Assembly continued

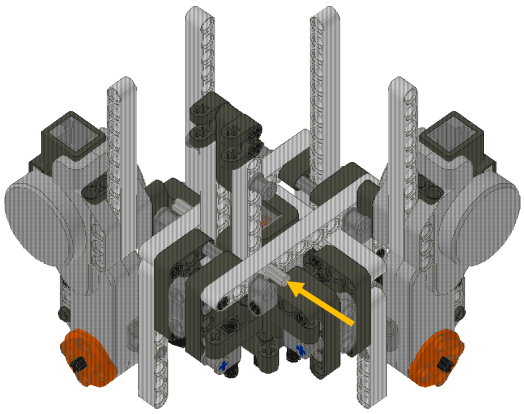
**34**

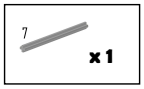





## Ballbot Assembly continued

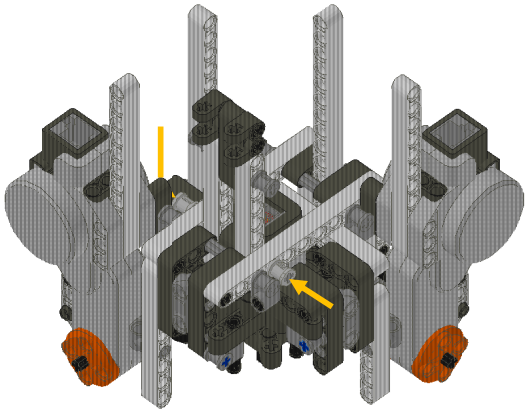
**35**





## Ballbot Assembly continued

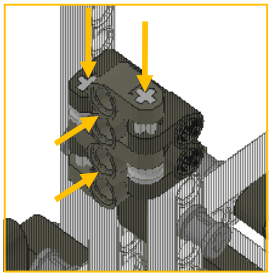
**36**

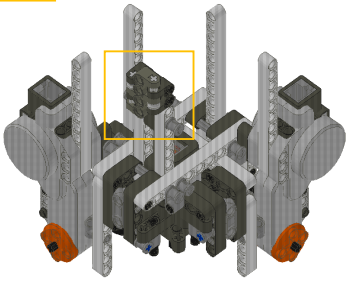


x 2

## Ballbot Assembly continued

**37**





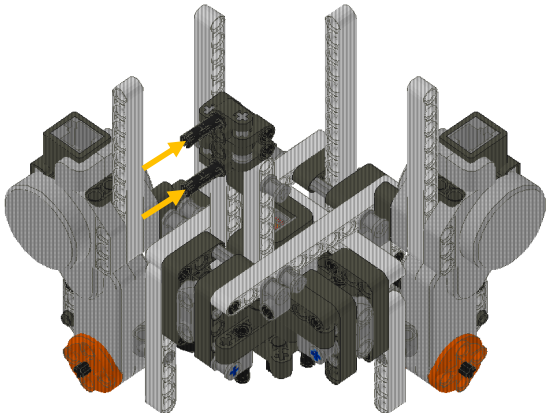
x 2

3

x 2

## Ballbot Assembly continued

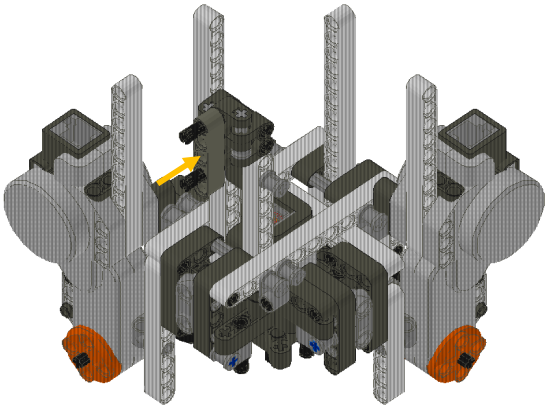
**38**



x 2

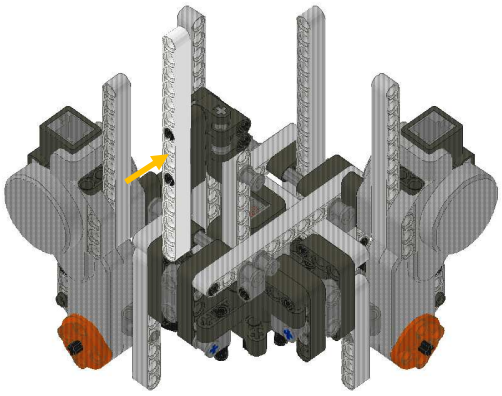
## Ballbot Assembly continued

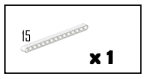
**39**



x 1

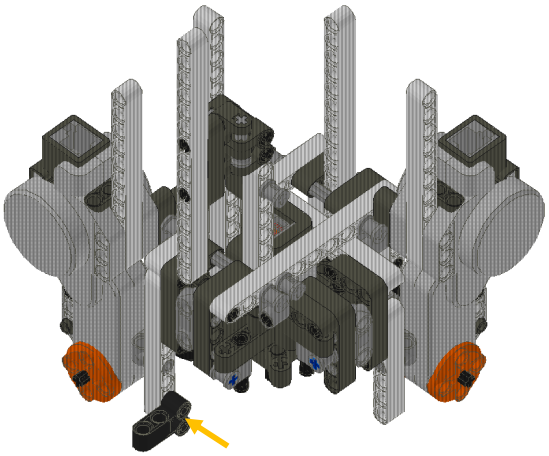
40

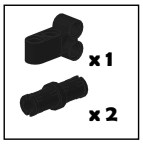






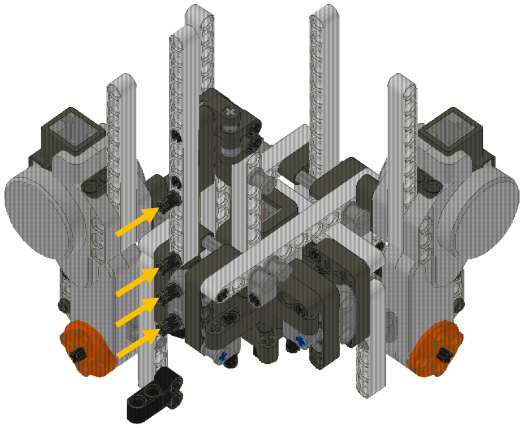

41





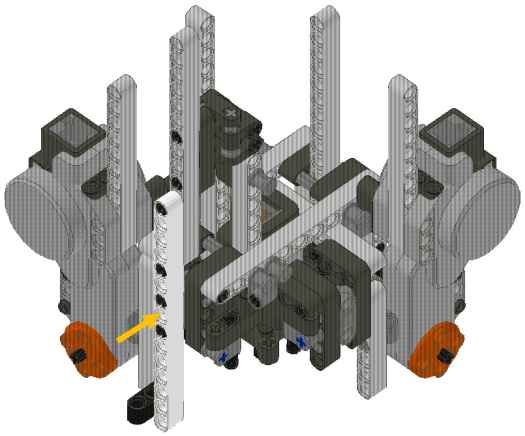


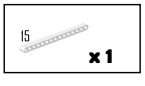

42





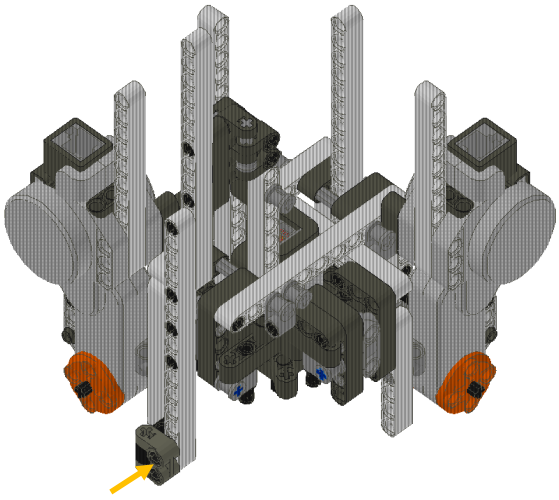

43





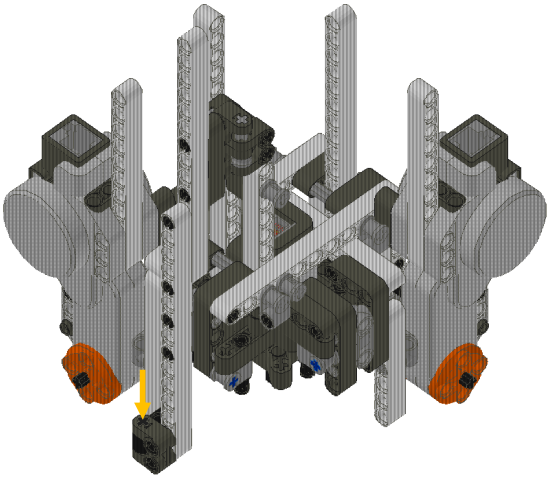
## Ballbot Assembly continued

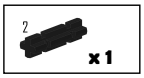
**44**




## Ballbot Assembly continued

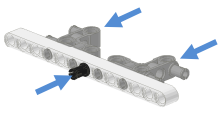
**45**

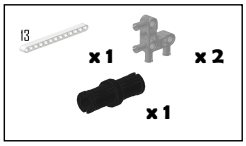




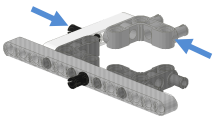

## Subassembly – Top Frame

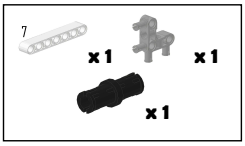
**a**



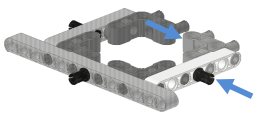


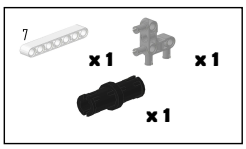

**b**






**c**

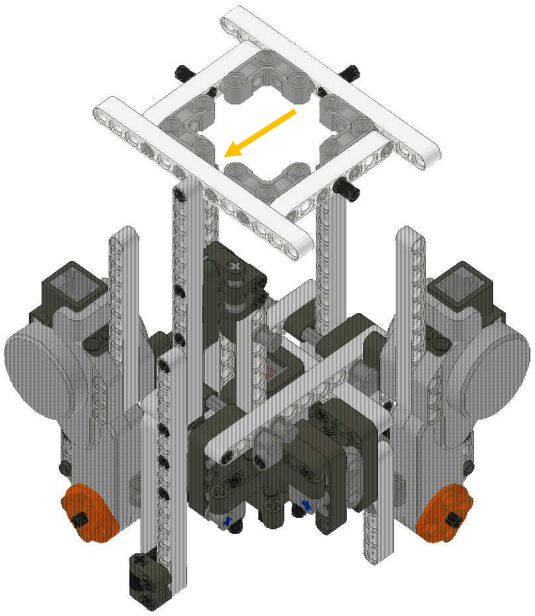


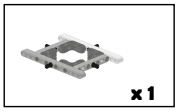



**d**


## Ballbot Assembly continued

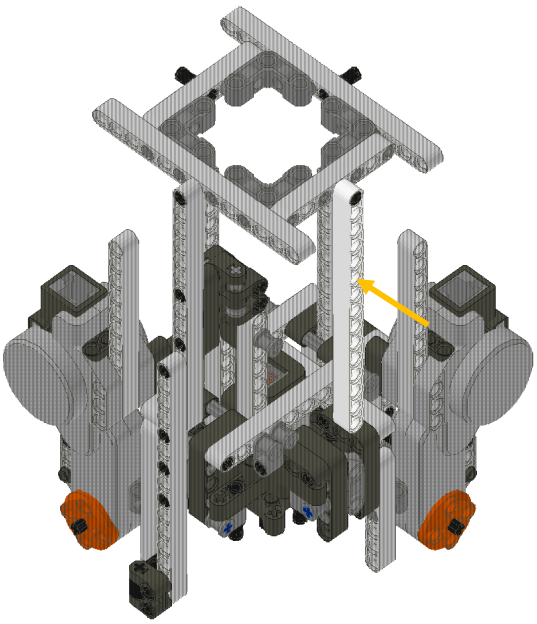
**46**

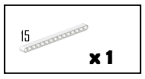




## Ballbot Assembly continued

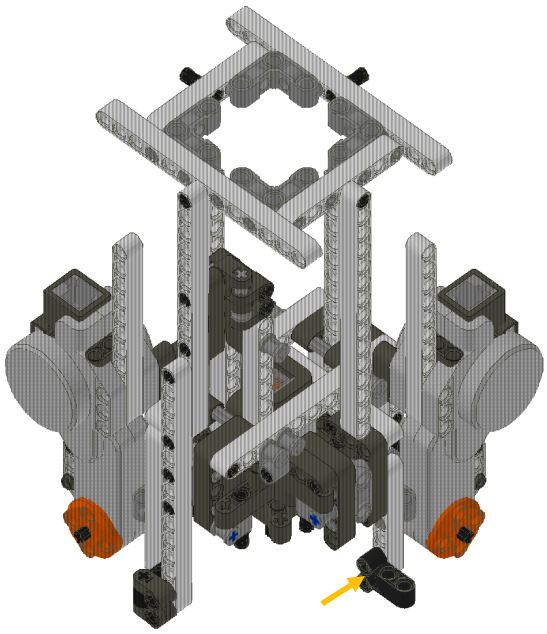
**47**

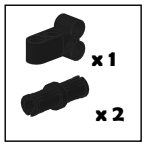





## Ballbot Assembly continued

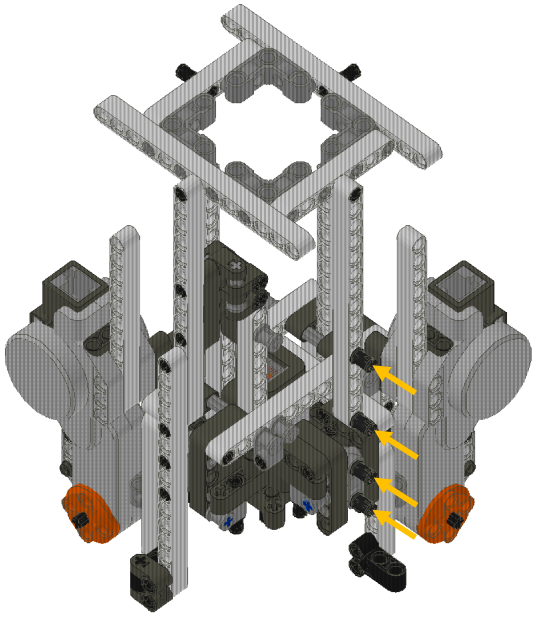
**48**





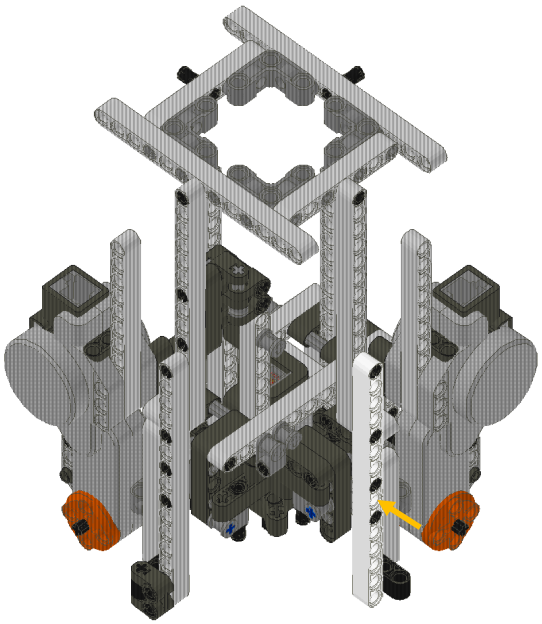

## Ballbot Assembly continued

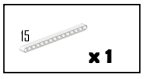
**49**



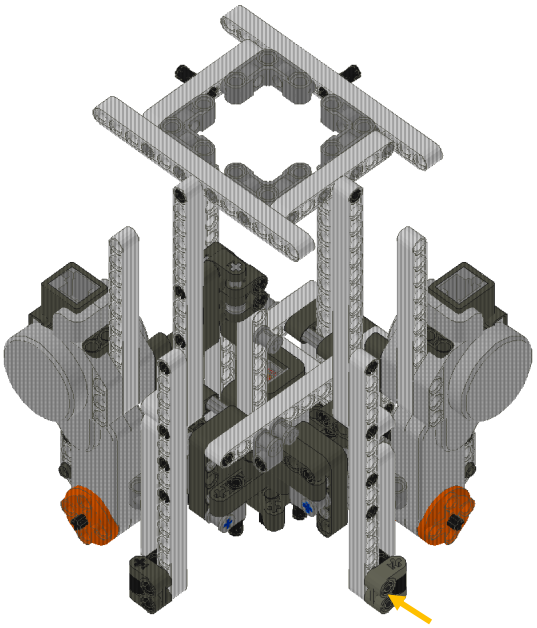

## Ballbot Assembly continued

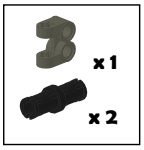
**50**





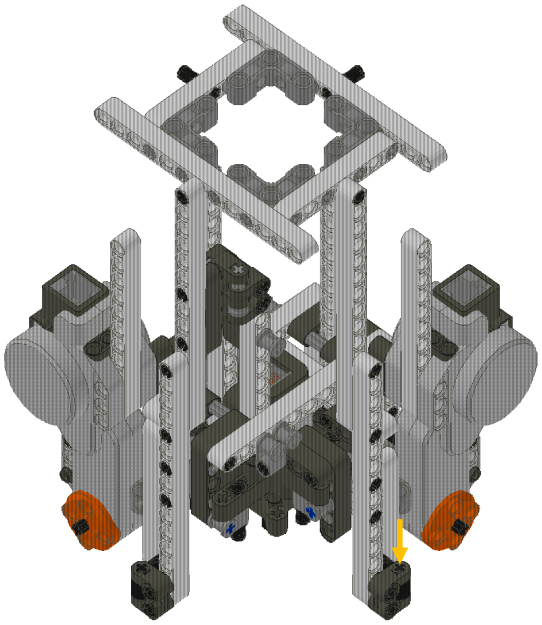
51

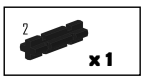






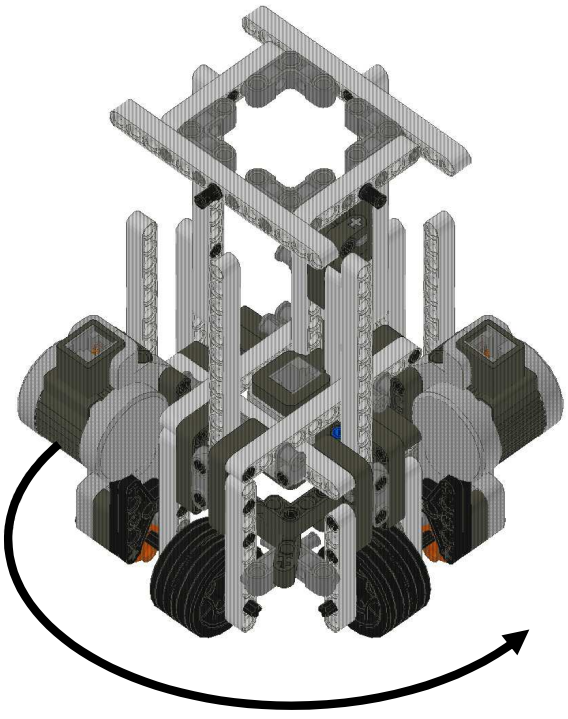

52





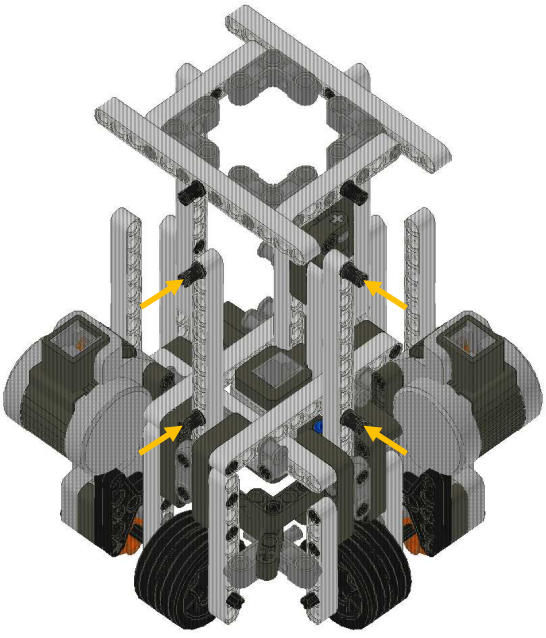



53

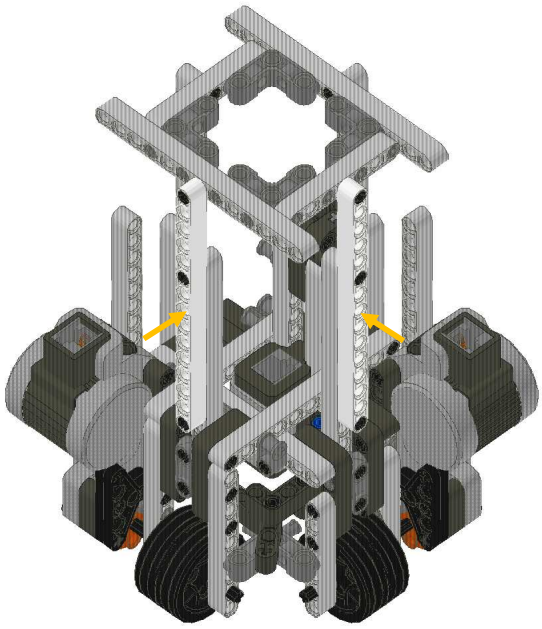


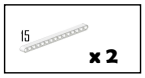


54



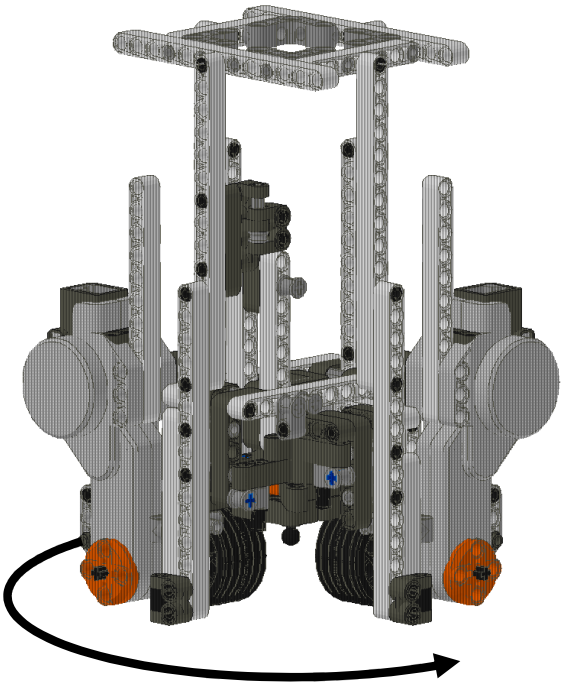
55





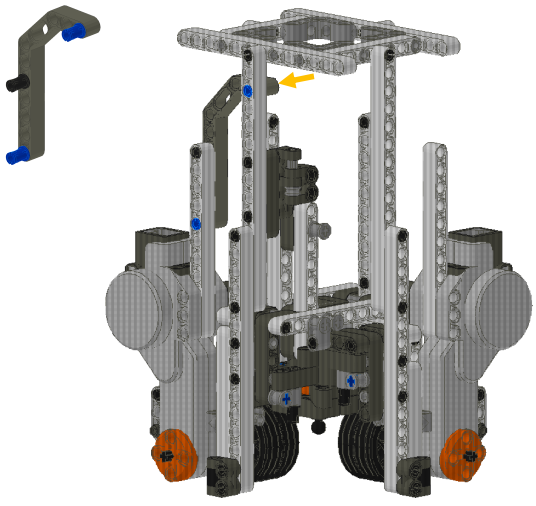


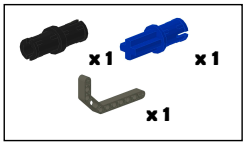

56





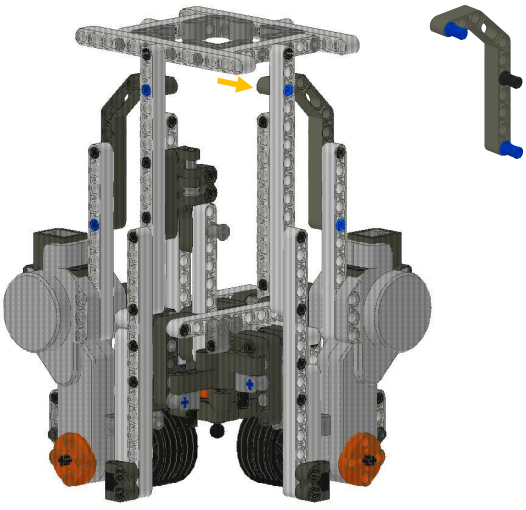
57

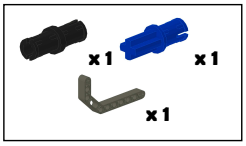






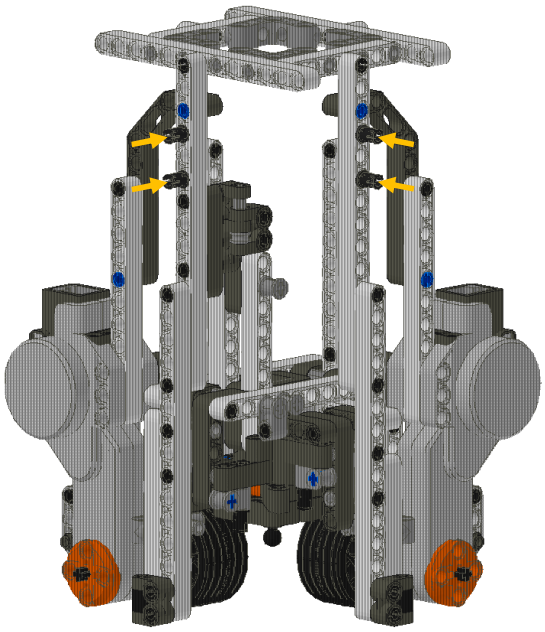

58





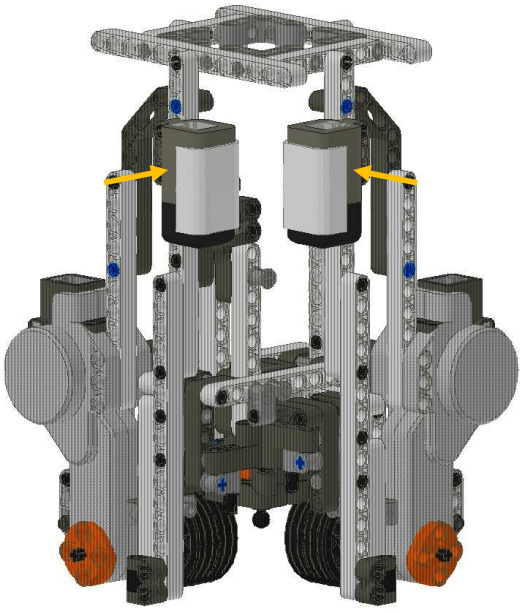
## Ballbot Assembly continued

**59**



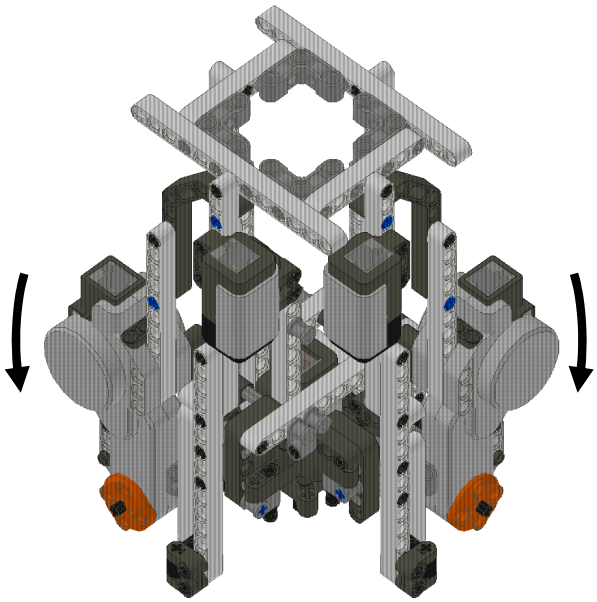

## Ballbot Assembly continued

**60**



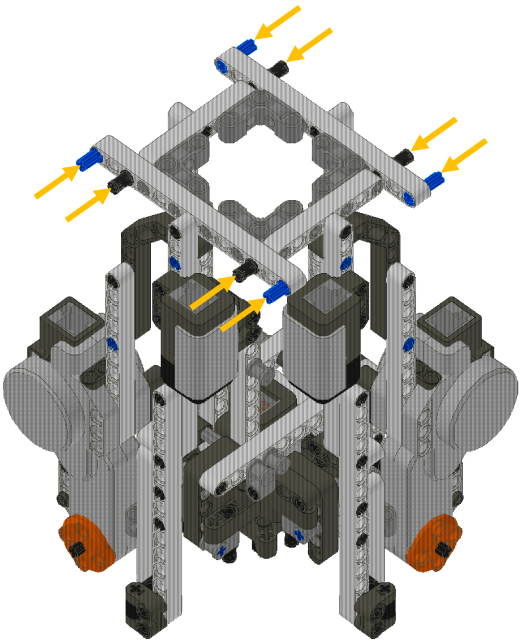

## Ballbot Assembly continued

**61**



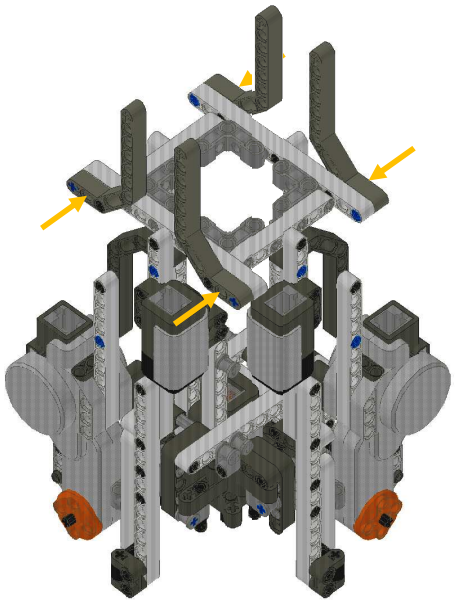
## Ballbot Assembly continued

**62**



## Ballbot Assembly continued

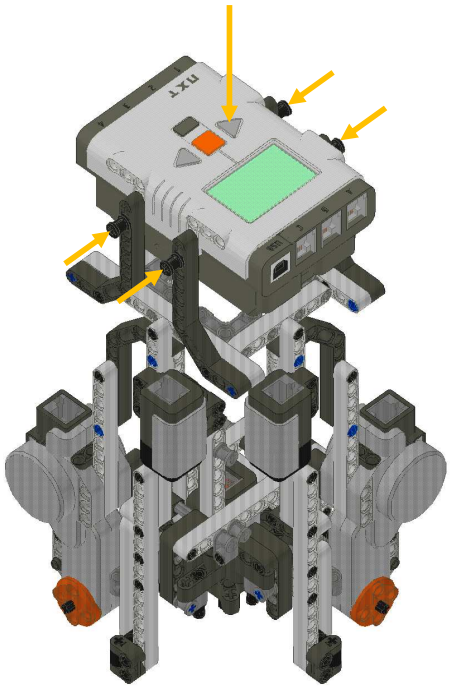
63



x 1

## Ballbot Assembly continued

64

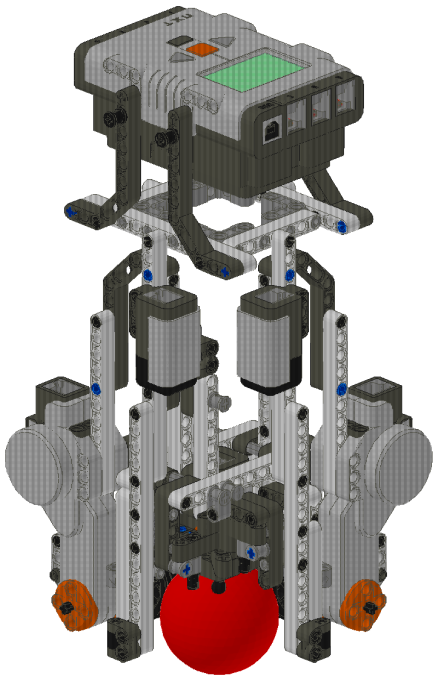


x 1

x 4

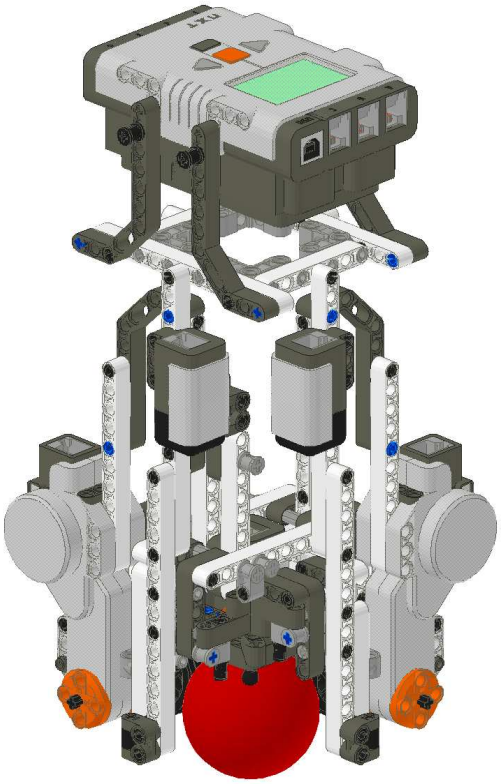
## Ballbot Assembly continued

65

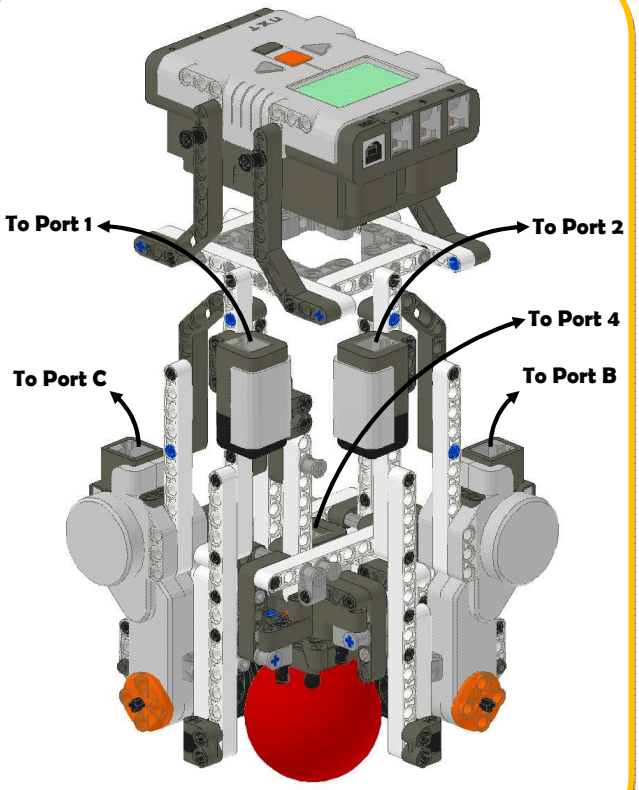


x 1
5.2 cm Red Ball

## Ballbot Assembly complete



## Ballbot Wiring Connections




# Software Installation Instructions

**The Lego Ballbot requires:**

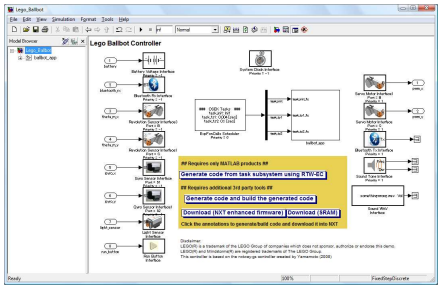
- **MATLAB and Simulink (with Real Time Workshop and Embedded Coder)**

**Additionally, the Ballbot requires installation of the ECRobot toolbox for Simulink. This is available at http://www.mathworks.com/matlabcentral/fileexchange/13399**

**To use remote control via Bluetooth, the NXTGamePad is also required. This program can be found with instructions at http://lejos-osek.sourceforge.net/nxtgamepad.htm**

# Programming Instructions

1. **Open the Ballbot parameter file "ballbot_params.m" and Simulink model "Lego_ballbot.mdl"**
2. **Run the Parameter File**
3. **In the Simulink model, click "Generate code and build the generated code"**
4. **Turn the NXT Brick on (by pushing the orange button) and connect it to the computer with the USB cable**
5. **In the Simulink Model, click "Download (NXT enhanced firmware)"**



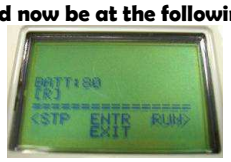

**Note: This process requires all software to be installed**

# Running the Ballbot

**Once downloaded the Ballbot can be run by**

1. **Placing the Ballbot on the ball and holding it upright**
2. **Open the program by selecting**
   **My Files > Software Files > ballbot_app**
   **using the ENTR button**


3. **You should now be at the following screen.**




4. **Optional: To connect via Bluetooth for remote control follow the instructions at http://lejos-osek.sourceforge.net/nxtgamepad.htm**
5. **Start the Ballbot by pressing the RUN button**
6. **Hold the Ballbot upright until it beeps, so the gyroscopes can calibrate**
7. **At the beep, release the Ballbot and it will balance**
8. **If the Ballbot falls over, it can be restarted by placing it atop the ball again, and pressing the RUN button. Program restart is not required.**

# Acknowledgements

**The Author wishes to acknowledge the following in the creation of this document**

- **The Embedded Coder Robot NXT Demo for development of the ECRobot Toolbox by Takashi Chikamasa**
- **The NXTway-GS (Self Balancing Two Wheeled Robot) Controller Design by Yorihisa Yamamoto**
- **NXTOSEK and the NXTGamepad**
- **The University of Adelaide**

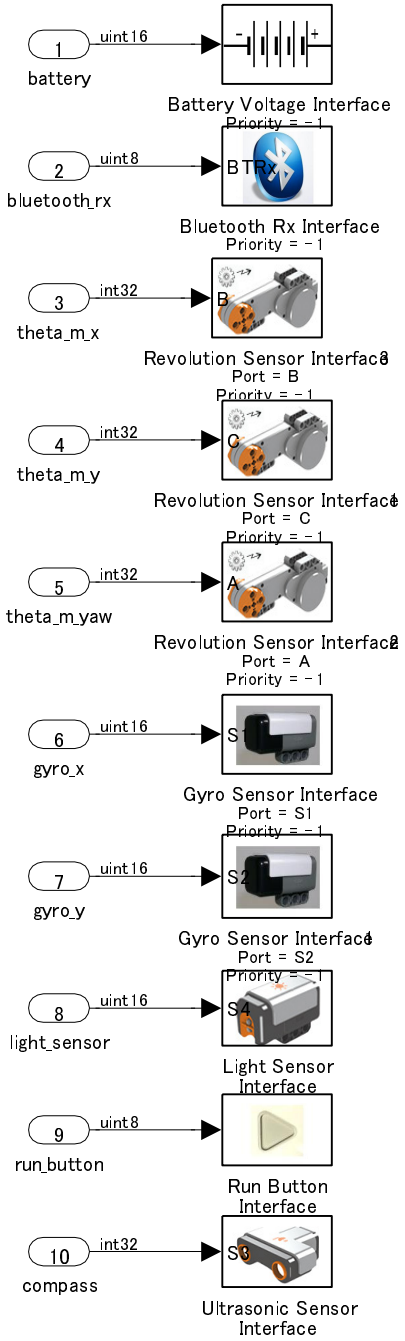
# E Lego Ballbot Simulink Controller

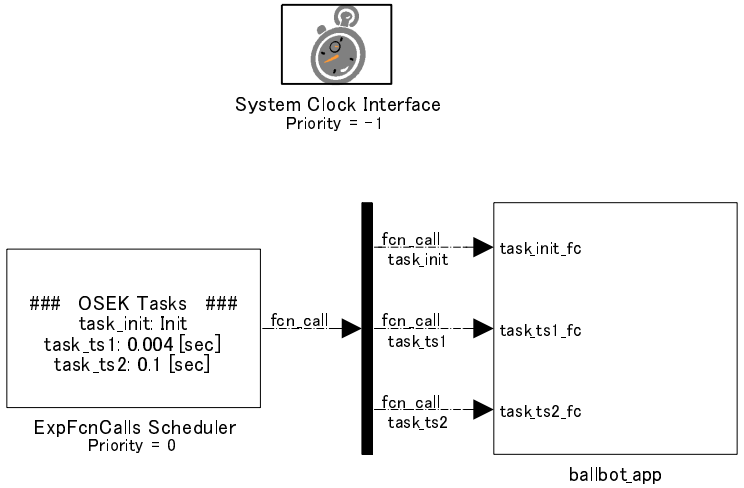
Presented in this appendix is the Lego Ballbot Controller, including the additional Yaw controller.

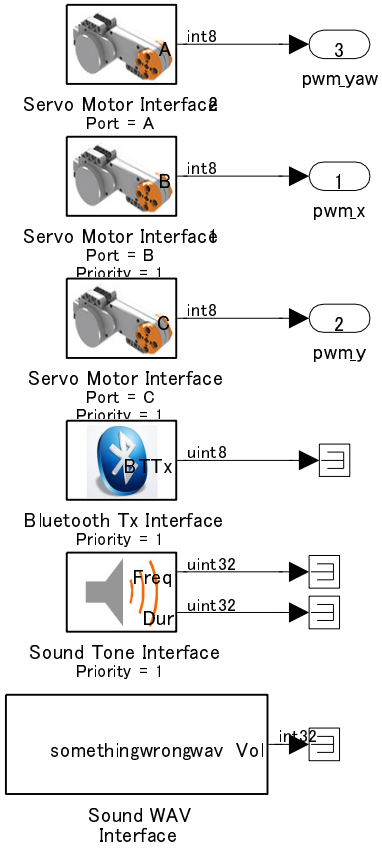
```
% Constants required for Lego Ballbot Controller
% Final Year Project 899 (2009)
% Justin Fong, Simon Uppill

% Sample times
ts1 = 0.004;                        % ts1 sample time [sec]
ts2 = 0.1;                          % ts3 sample time [sec]

% Balancing Controller Gains
k = [     100          -1.2      15          -1      -0.3];

% Yaw Controller Gains (calculated using ZN Tuning Method)
Kp_yaw = 1.6*1.286173633;
Ki_yaw = 0.2*0.352941176;
Kd_yaw = 1.6*1.171755186;

% State Definitions for Finite State Machine
stop = 0;
calib = 1;
control = 2;

% Timing
time_start = 1000;                  % Length of calibration [ms]
cmd_start = 30000;                  % Start of command signal
cmd_end = 38000;                    % End of command signal

% Light Sensor Threshold
ls_thres = 570;

% Low Pass Filter Coefficients
a_b = 0.8;                          % Smooth battery value calculation
a_d = 0.8;                          % Smooth Phi estimates
a_gyro = 0.25;                      % Smooth Theta Dot estimates
a_r = 0.996;                        % Smooth reference signal (on Phi)

a_gc = 0.8;                         % Gyro offset in Calibration (initialise)
a_gd = 0.999;                       % Gyro offset in operation (compensate for drift)

a_c = 0.9;                          % Smooth compass signal
a_yaw_ms = 0.996;                   % Smooth orientation estimate
a_yaw = 0.999;                      % Smooth Yaw Motor

% Parameters of Coulombic & Viscous Friction Compensator
pwm_gain = 1;                       % pwm gain
pwm_offset = 0;                     % pwm offset
yaw_deadzone = 20;                  % offset for the yaw motor

% Parameters for Command tracking
k_phidot = 4;                       % Movement speed gain (maximum velocity expressed↙
in rad/s)
k_z_cmd = 10;                       % Yaw speed gain
gp_max = 100;                       % Maximum game pad input

% Parameters for the initial beep
sound_freq = 440;                   % sound frequency [Hz]
sound_dur = 500;                    % sound duration [msec]
```

```
% Parameters for data logger
log_count = 20;                     % data logging count (logging sample time = ts1 *↙
log_count)

% Sample Rates
TS = 0.001;                         % base sample rate [sec]
```

# Final Year Project 899: Lego Ballbot Controller




**## Requires only MATLAB products##**

**Generate code from task subsystem using RTW -EC**

**## Requires additional 3rd party tools ##**

**Generate code and build the generated code**

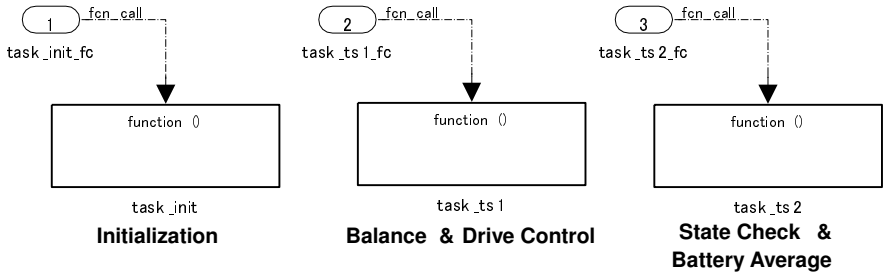
**Download (NXT enhanced firmware)** **Download (SRAM)**

**Click the annotations to generate/build code and download it into NXT**

Disclaimer:
LEGO(R) is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this demo.
LEGO(R) and Mindstorms(R) are registered trademarks of The LEGO Group.
This controller is based on the nxtway-gs controller created by Yamamoto (2008)

## Application Task Subsystems

**The three tasks run at different timesteps .**




## Shared Data

**Data Store Memory is used as a shared data between tasks .**

calib_time
DataType = uint 32
**Time at start of last calibration**

state
DataType = int 8
**Current state of the program (Stop, Calibration or Control )**

flag_firstcycle
DataType = boolean
**Flag indicating whether it is the first cycle of control program after calibration**

motor_x_signal
DataType = int 8
motor_y_signal
DataType = int 8
motor_z_signal
DataType = int 8
**Current Motor Signals**

battery
DataType = single
**Average battery voltage**

gyro_x_offset
DataType = single
**x-gyro offset (zero point value )**

gyro_y_offset
DataType = single
**y-gyro offset (zero point value )**

motor_x_offset
DataType = single
motor_y_offset
DataType = single
**Offsets for motor readings (Used to reset reading to zero at re-calibration )**

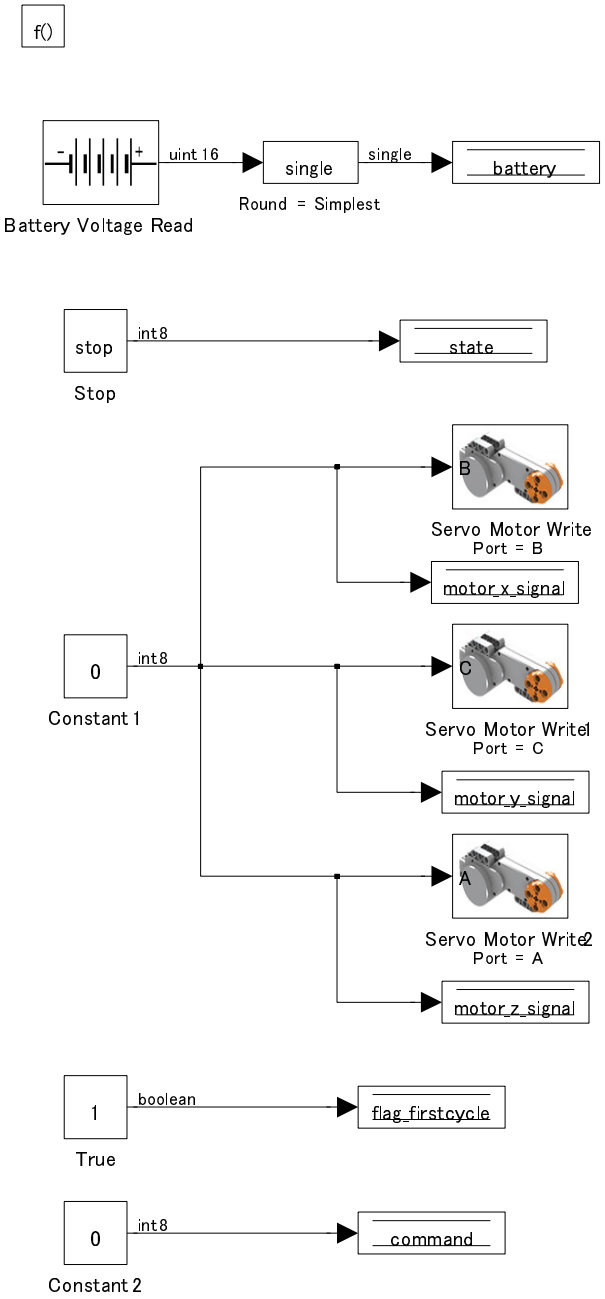
command
DataType = int 8
**Control Signal for automatically generated commands**

**Variables for Data Logger**

data 1
DataType = int 8
data 3
DataType = int 16
data 5
DataType = int 16
data 2
DataType = int 8
data 4
DataType = int 16
data 6
DataType = int 16

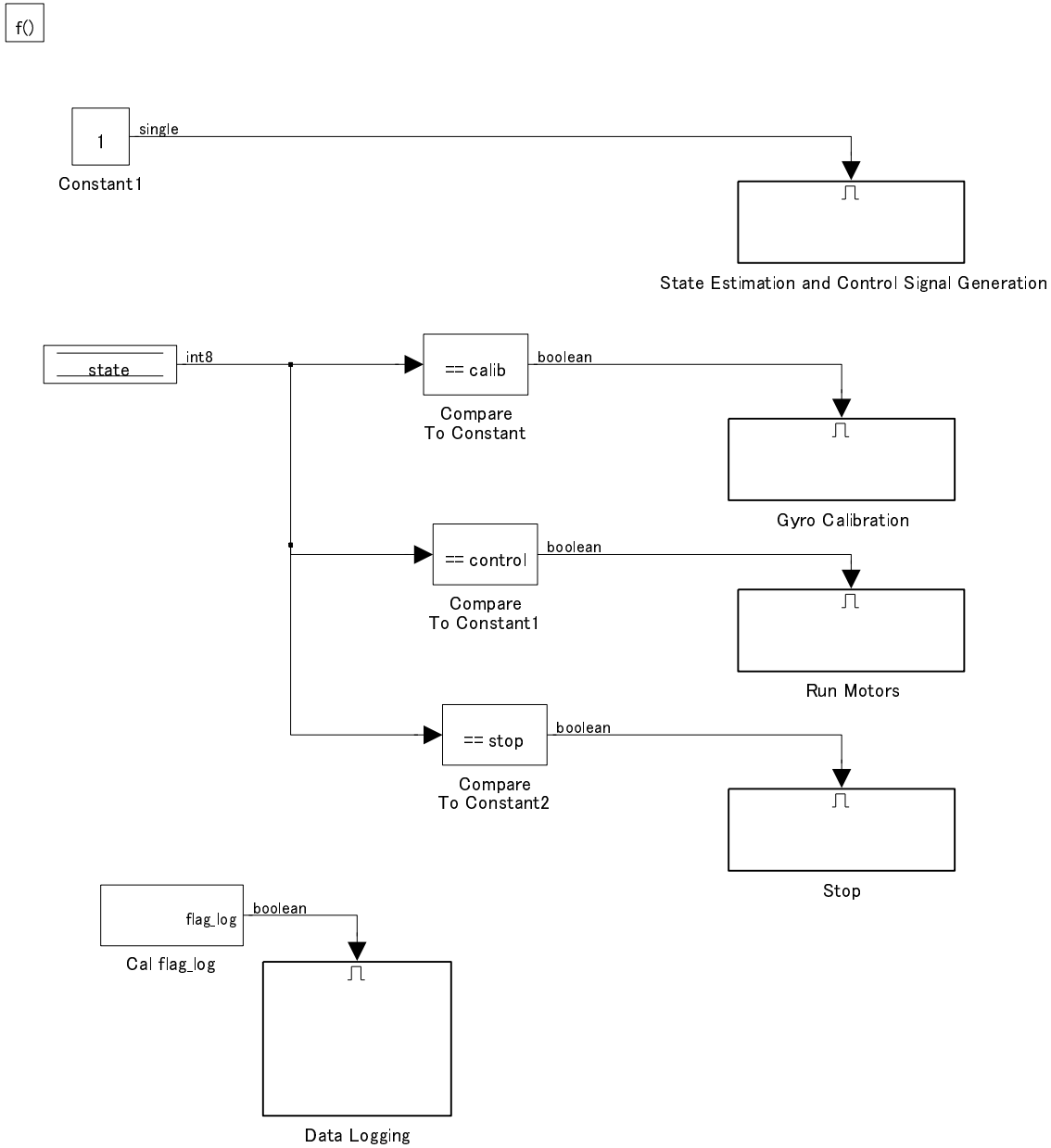
# Initialization Task

**Set initial values , set the motor signals to zero ， and initialise to stop state**



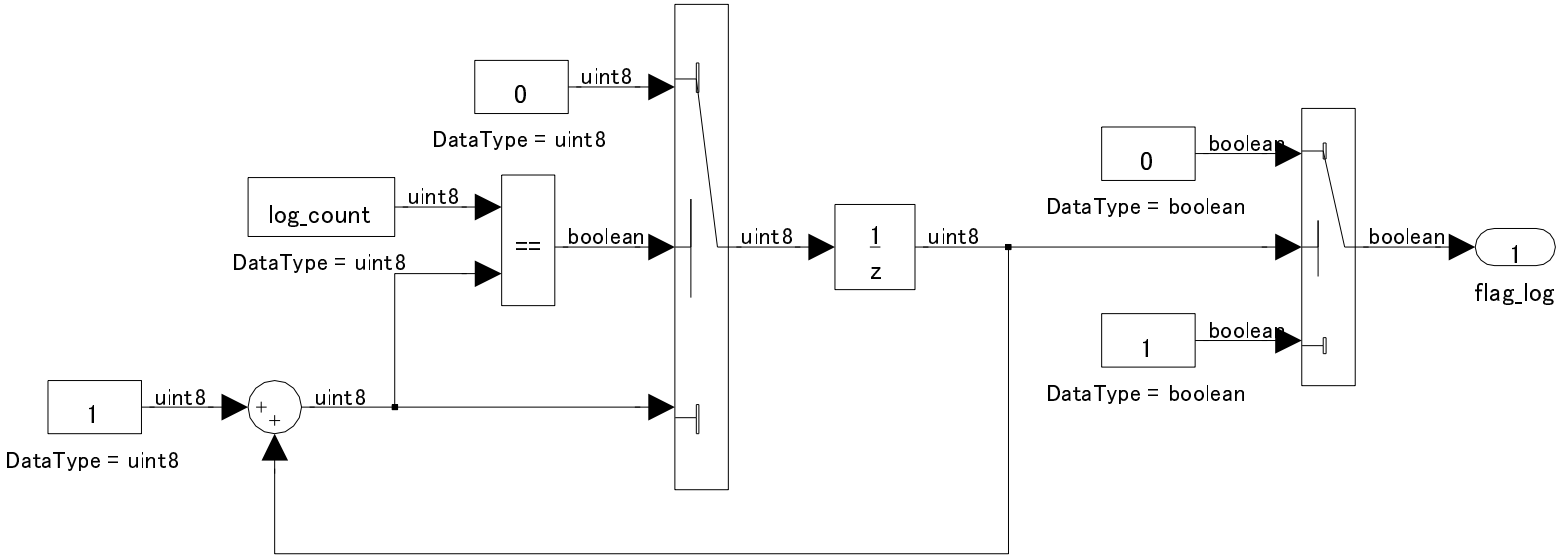
# Balance & Drive Control Task

**Runs the State Estimation and Control Signal Generation continously, and then drives the motors appropriately**



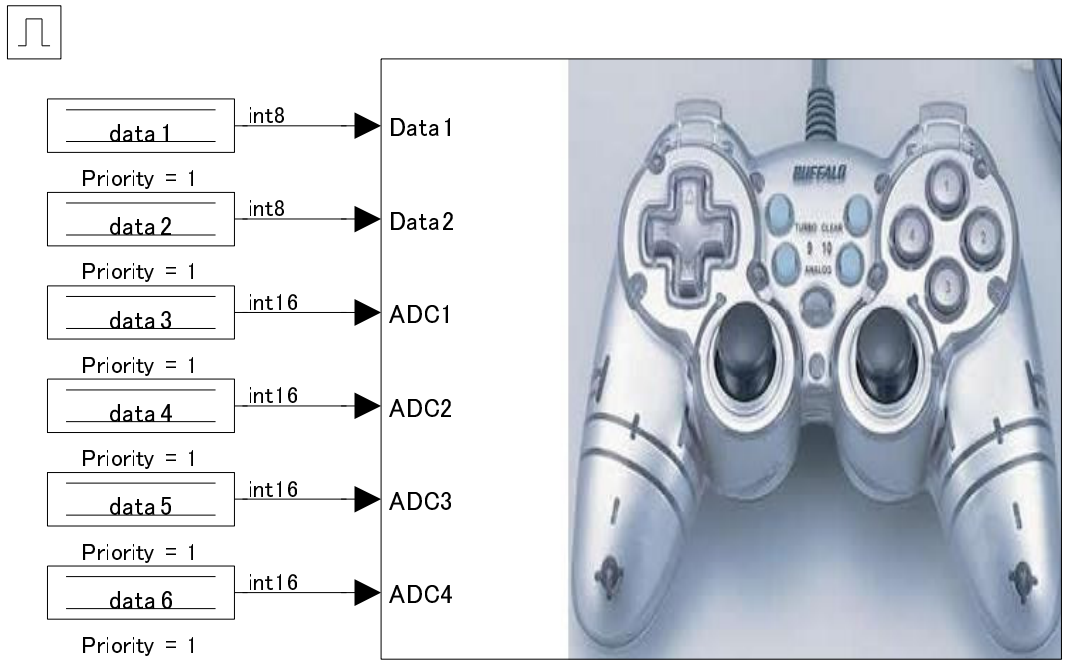
# Cal Flag_log

**Ensures logging occurs at the right rate, using counts and only activing the flag when the count reaches the log coun**



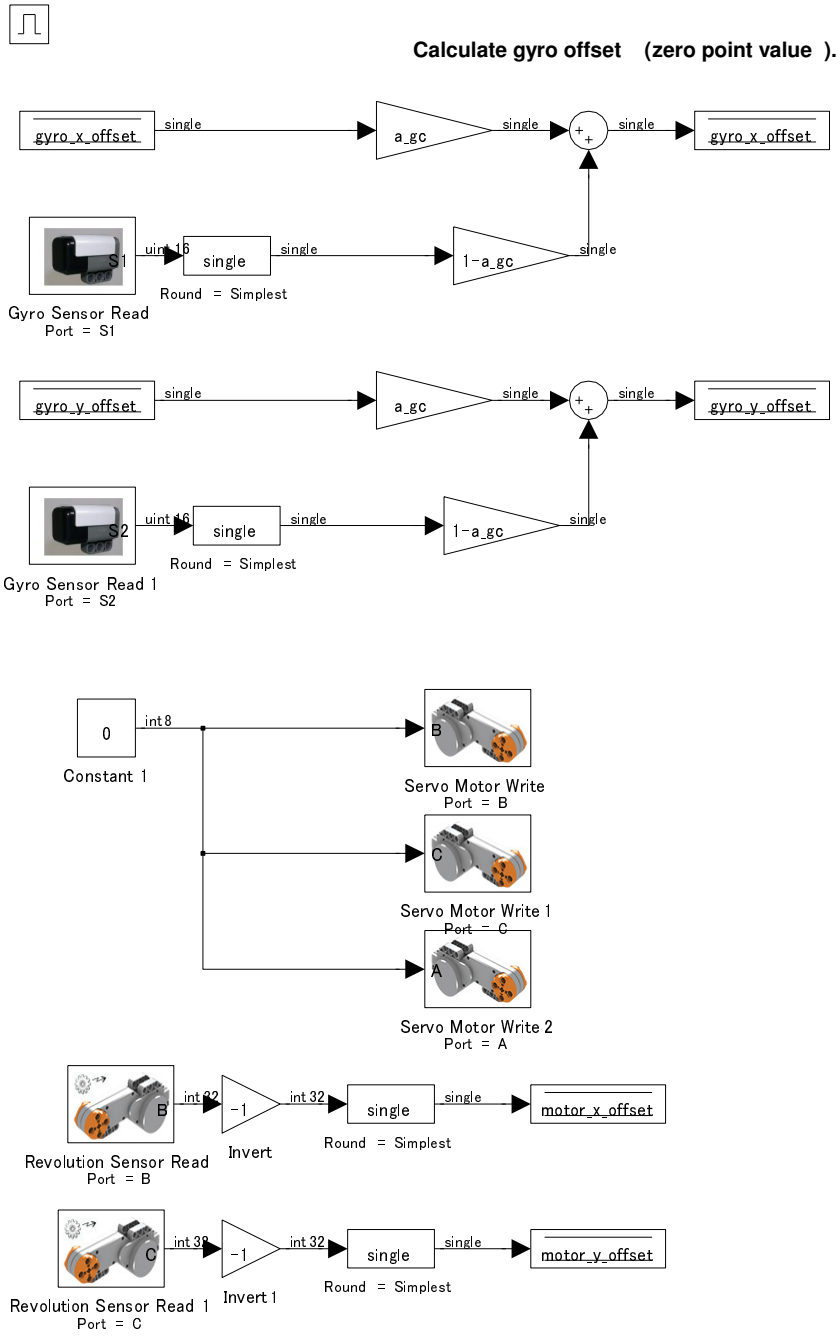
# Data Logging

## Log the body angle using NXT GamePad ADC Data Logger




NXT GamePad
ADC Data Logger

# Gyro Calibration



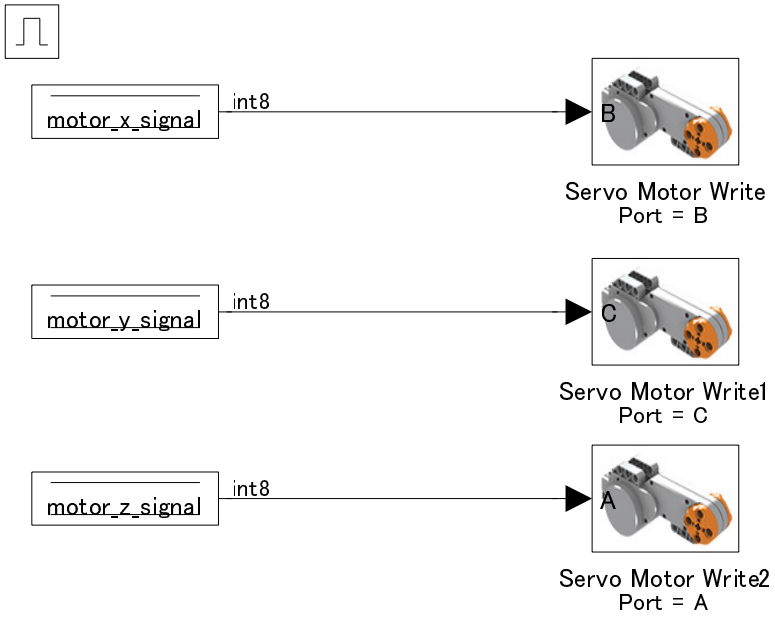

**Low Pass Filter Function**

**1-a_gc**

**----------**

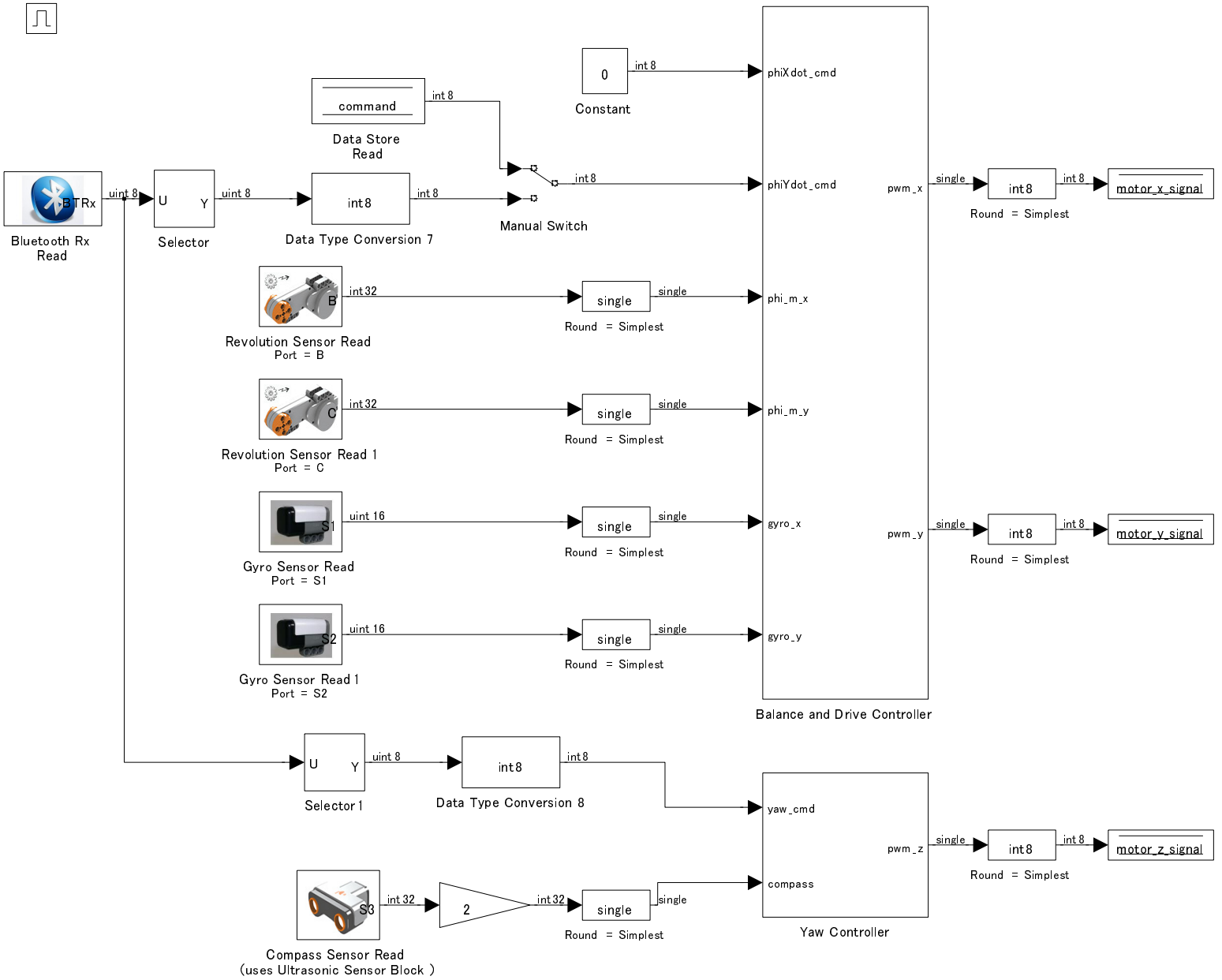
**1-a_gc*z^(-1)**

# Run Motors


motor_x_signal
int8
B
Servo Motor Write
Port = B
motor_y_signal
int8
C
Servo Motor Write1
Port = C
motor_z_signal
int8
A
Servo Motor Write2
Port = A

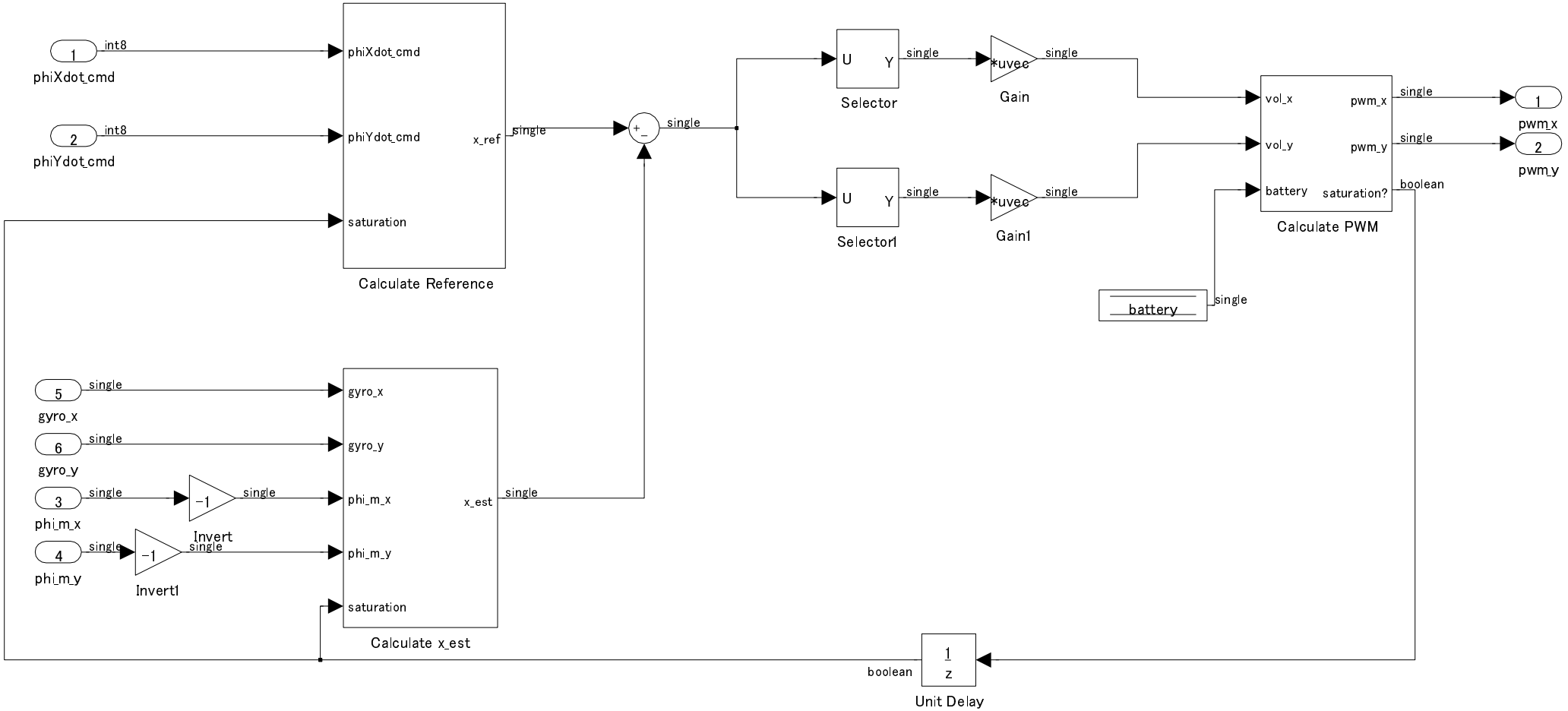
# State Estimation and Control Signal Generation

**Task runs two controllers - one for the balancing and horizontal movement , and one for controlling the yaw**



## Balance and Drive Controller

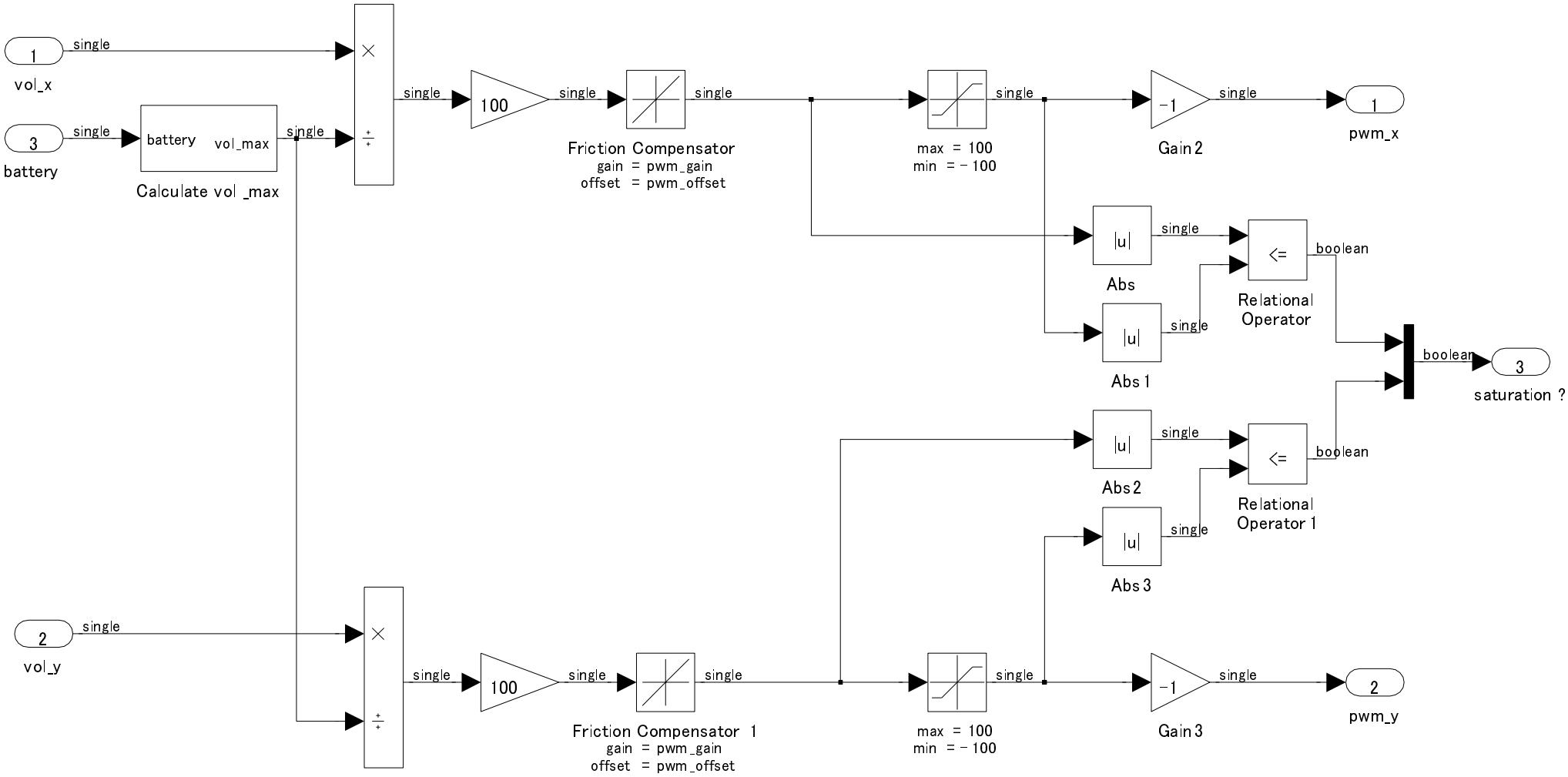
**Calculate PWM duty to minimize the difference between reference and measured value .**



# Calculate PWM

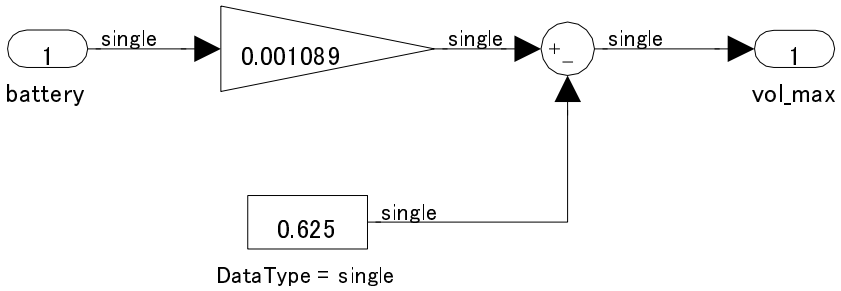
**Calculate maximum of DC motor voltage from battery using an experimental expression developed by Yamamoto (2009 )**

**Compensate coulombic and viscous friction of drive system**



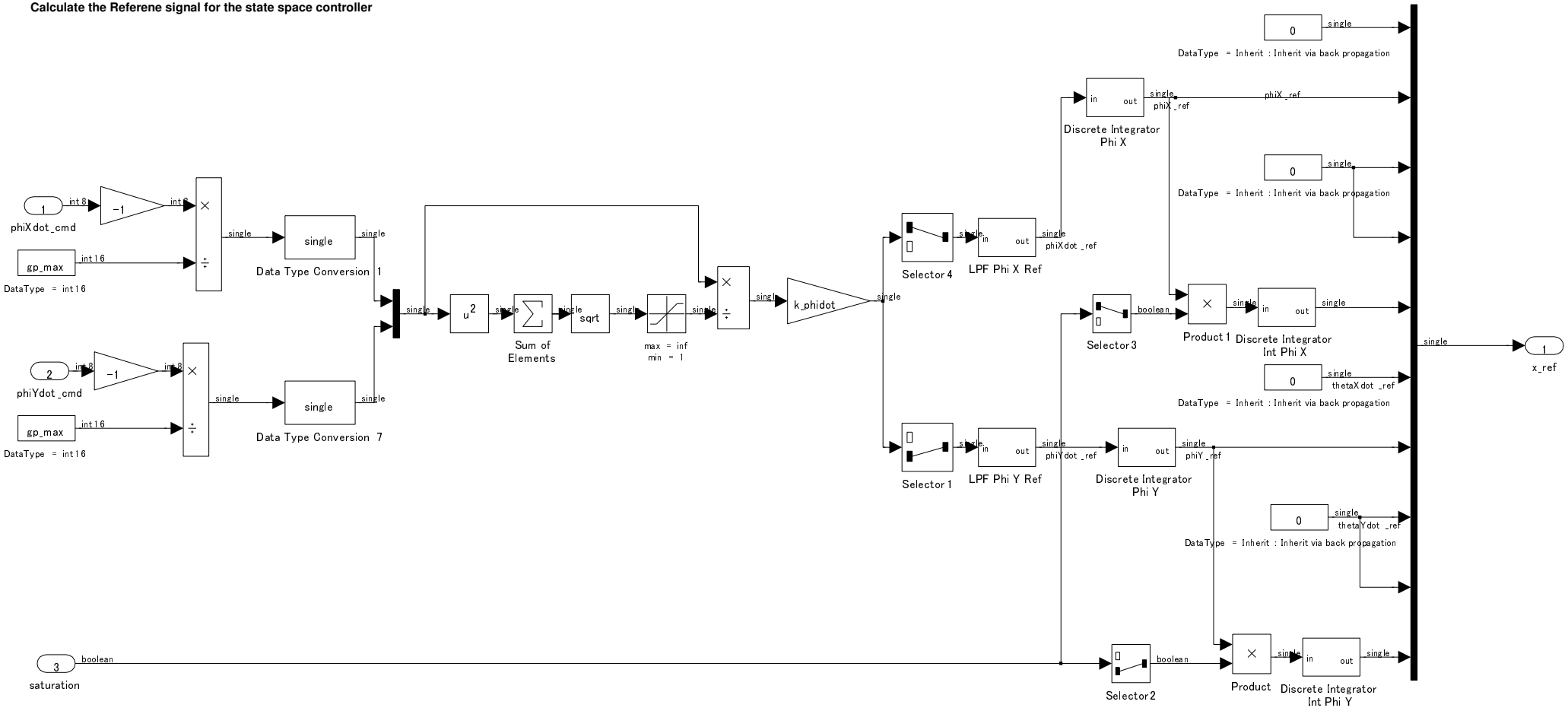
# Calculate vol_max

**Calculating Maximum Voltage using experimental expression developed by Yamamoto (2008)**

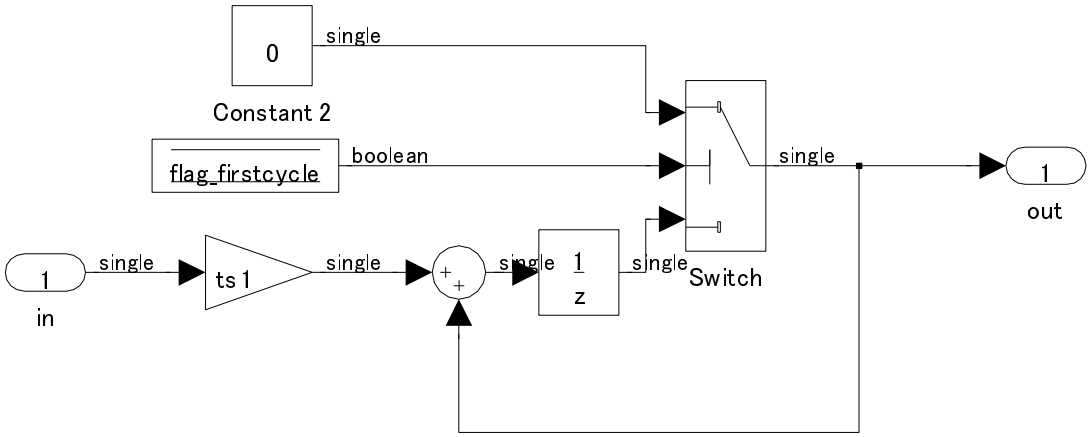


## Calulate Reference

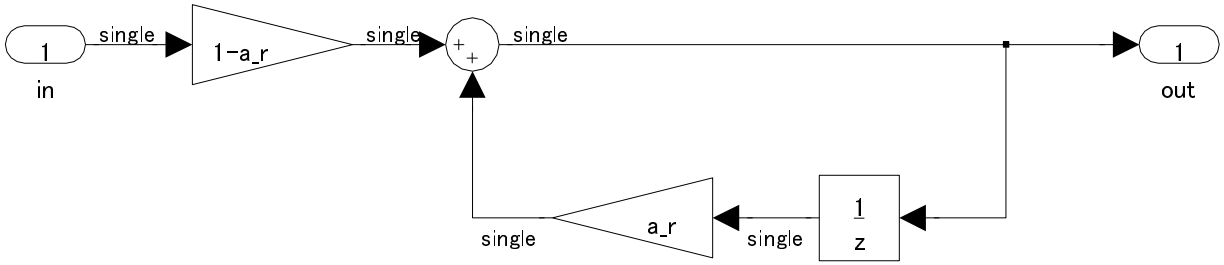
**Calculate the Referene signal for the state space controller**



# Discrete Integrator



# Low Pass Filter Phi X/Y Ref

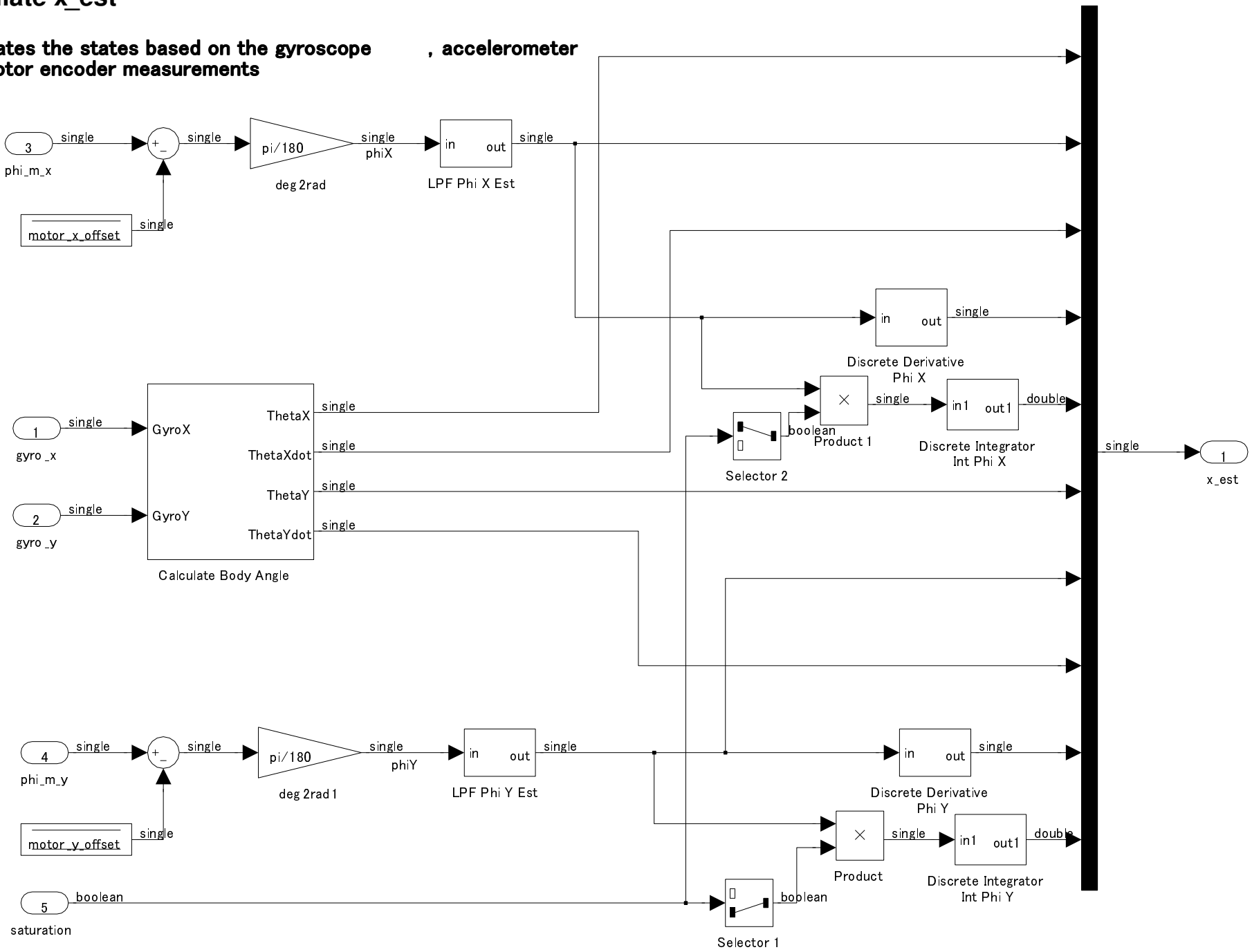



**Low Pass Filter Function**

$$\frac{1-a\_r}{1-a\_r*z^{(-1)}}$$

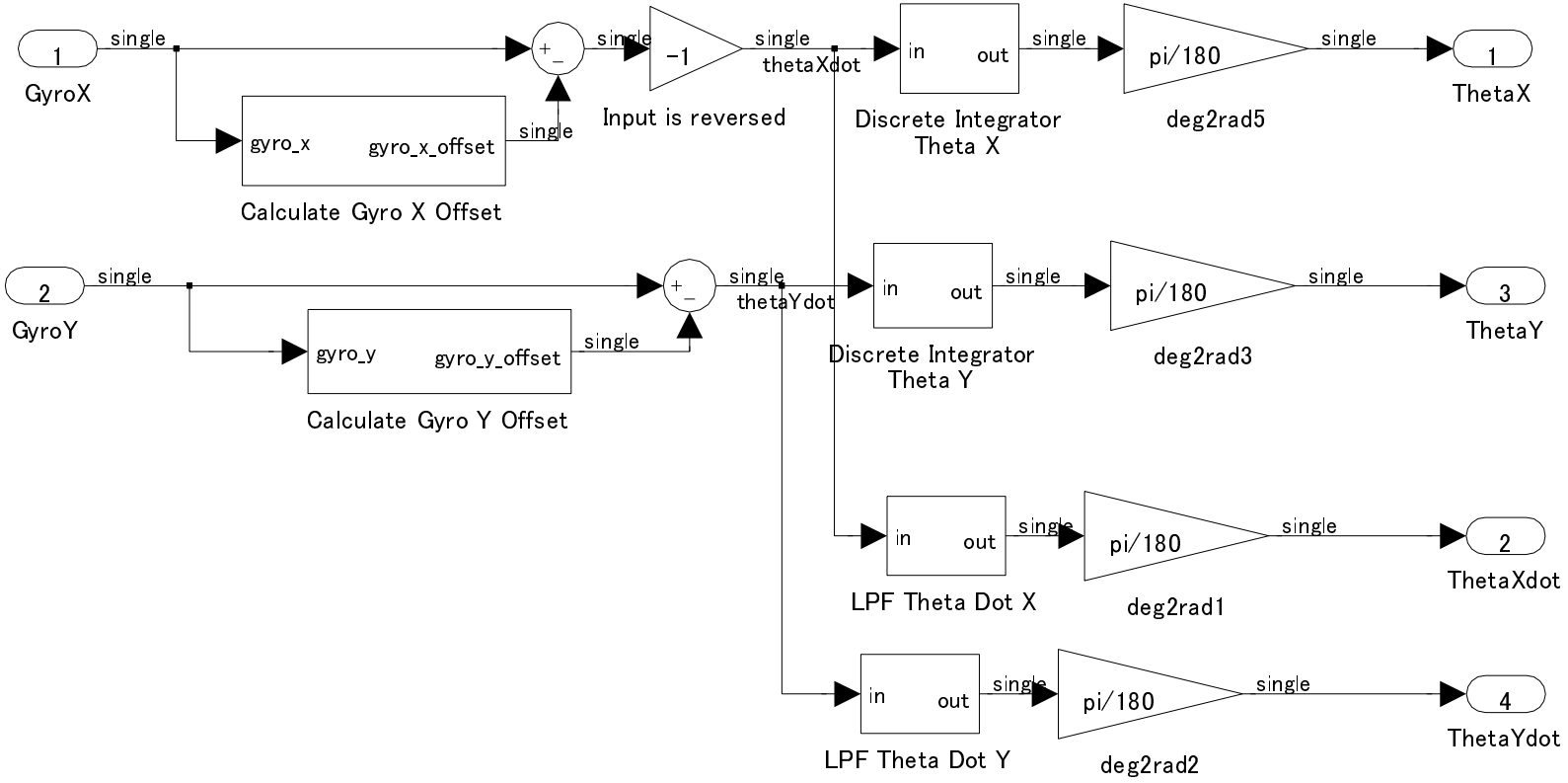
# Calculate x_est

**Calculates the states based on the gyroscope , accelerometer and motor encoder measurements**

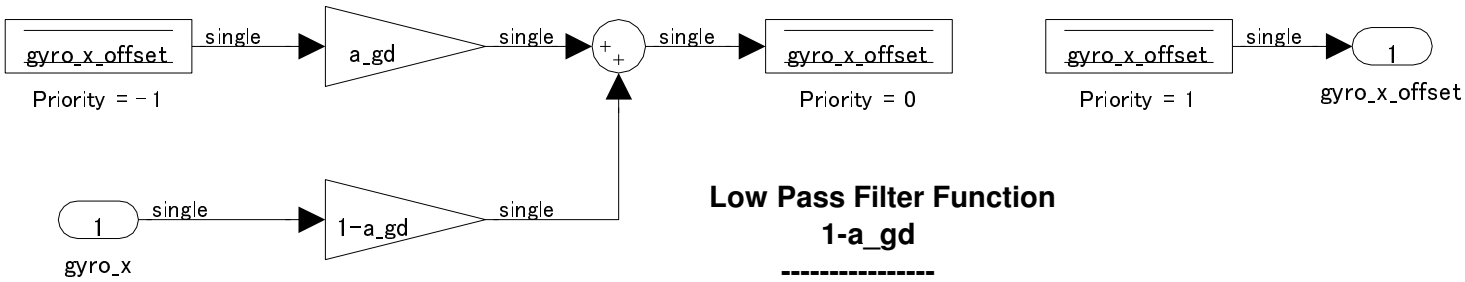


# Calculate Body Angle

## Calculates body angle in x and y planes



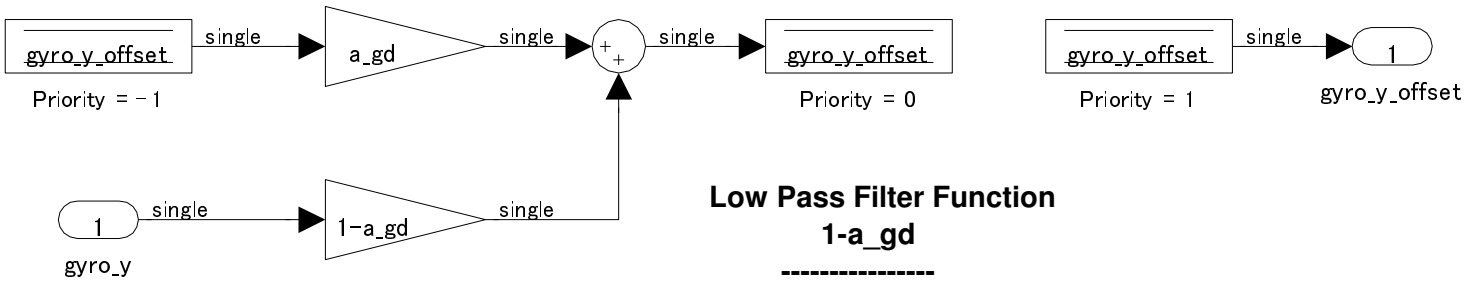
# Calculate Gyro X Offset




**Low Pass Filter Function**

$$\frac{1-a\_gd}{1-a\_gd*z^{(-1)}}$$

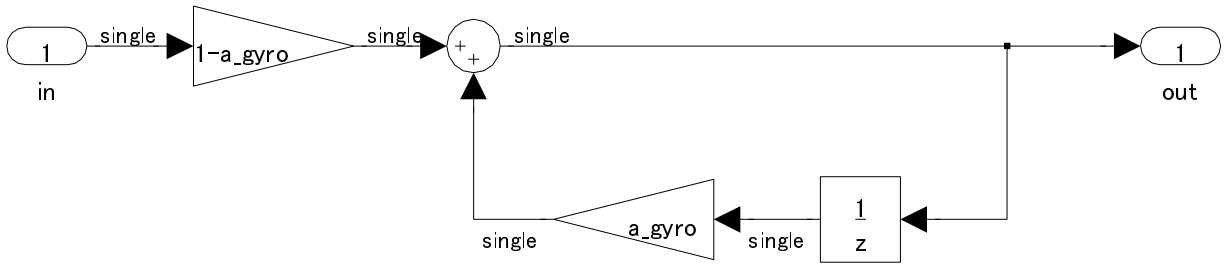
# Calculate Gyro Y Offset




**Low Pass Filter Function**

$$\frac{1-a\_gd}{1-a\_gd*z^{(-1)}}$$

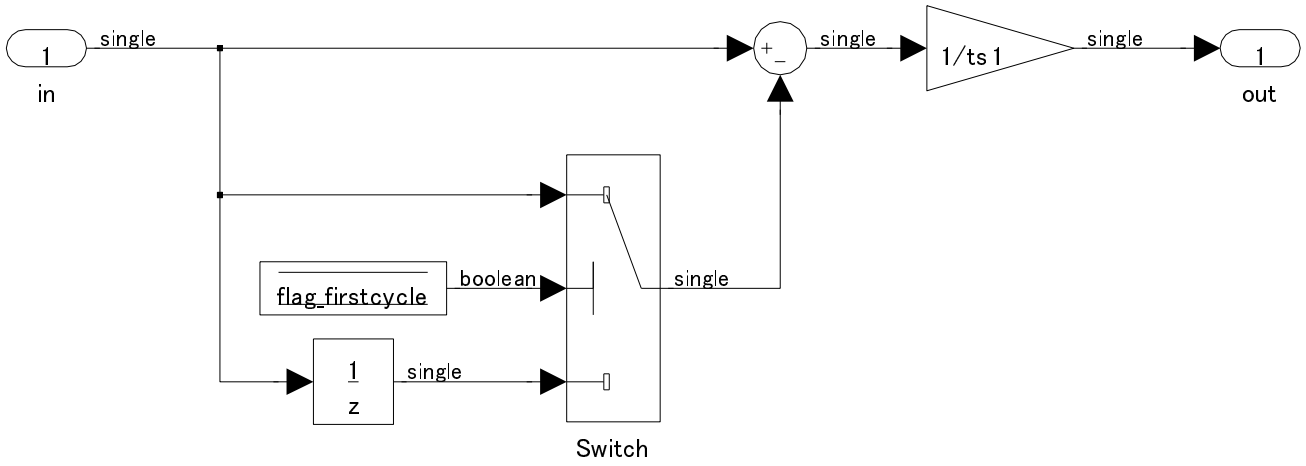
# Low Pass Filter Theta Dot X/Y



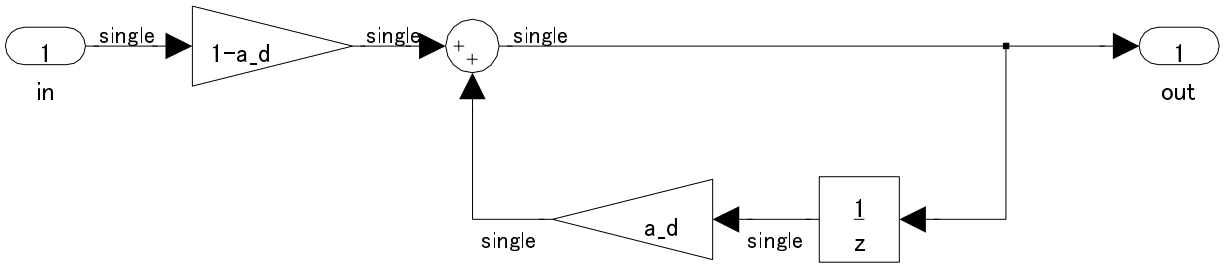

**Low Pass Filter Function**

$$\frac{1\text{-a\_gyro}}{1\text{-a\_gyro}*z^{(-1)}}$$

# Discrete Derivative



# Low Pass Filter Phi X/Y Est

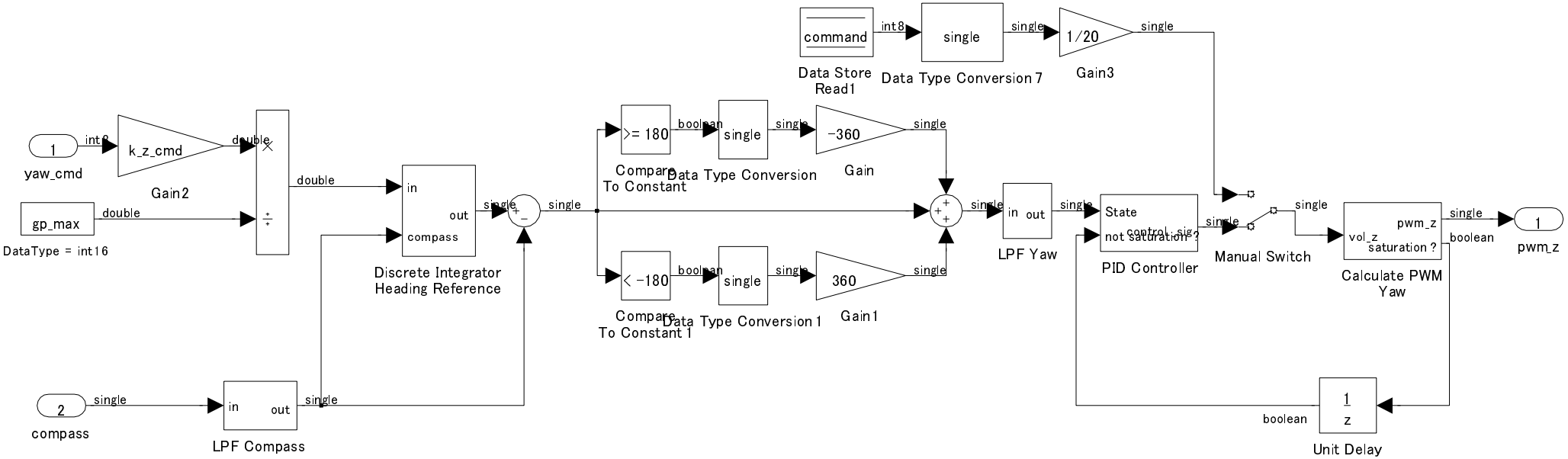



**Low Pass Filter Function**

$$\frac{1-a\_d}{1-a\_d*z^{(-1)}}$$

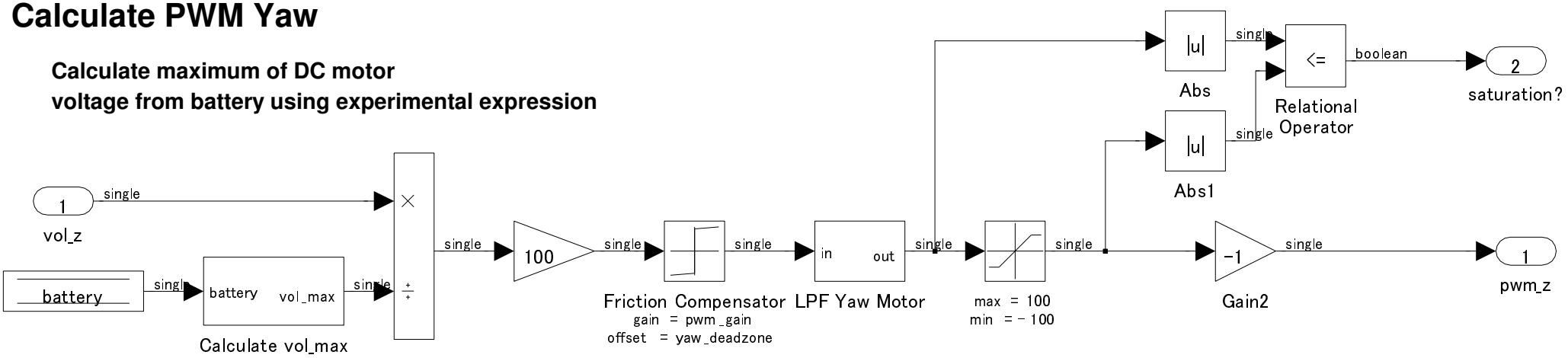
# Yaw Controller

**Calculate PWM Duty using a PID controller to control the orientation of the Ballbot**

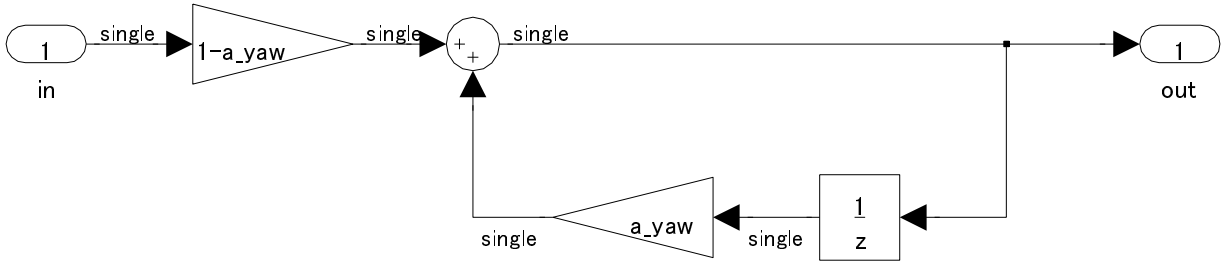


# Calculate PWM Yaw

**Calculate maximum of DC motor voltage from battery using experimental expression**



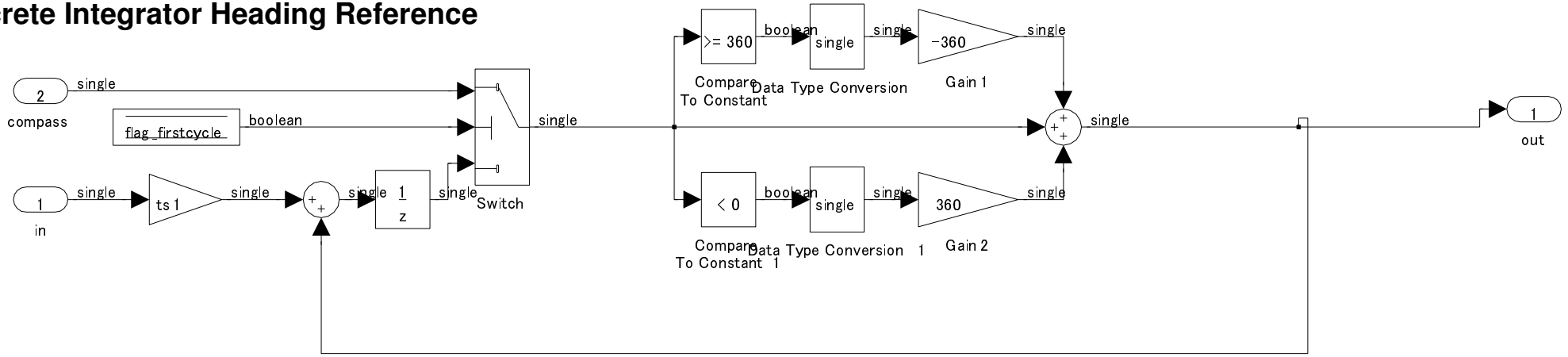
# Low Pass Filter Yaw Motor



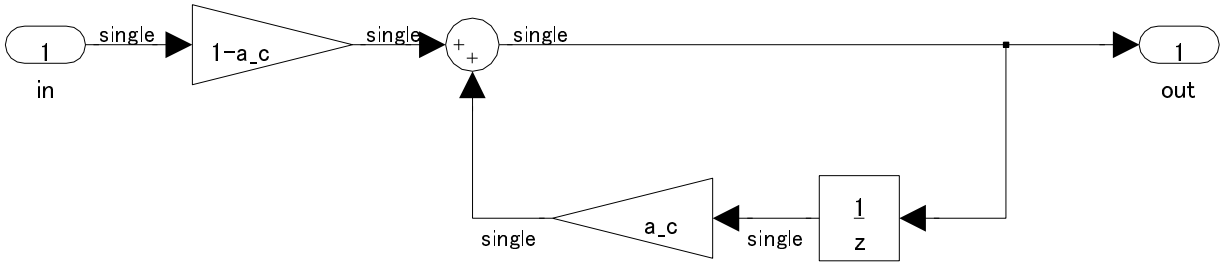

**Low Pass Filter Function**

$$\frac{1-a\_yaw}{1-a\_yaw*z^{(-1)}}$$

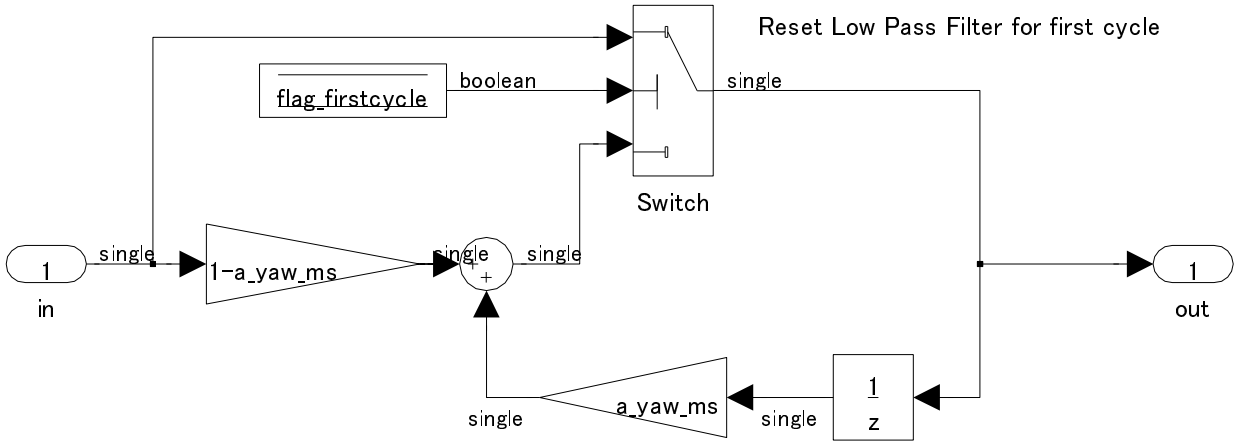
## Discrete Integrator Heading Reference



# Low Pass Filter Compass



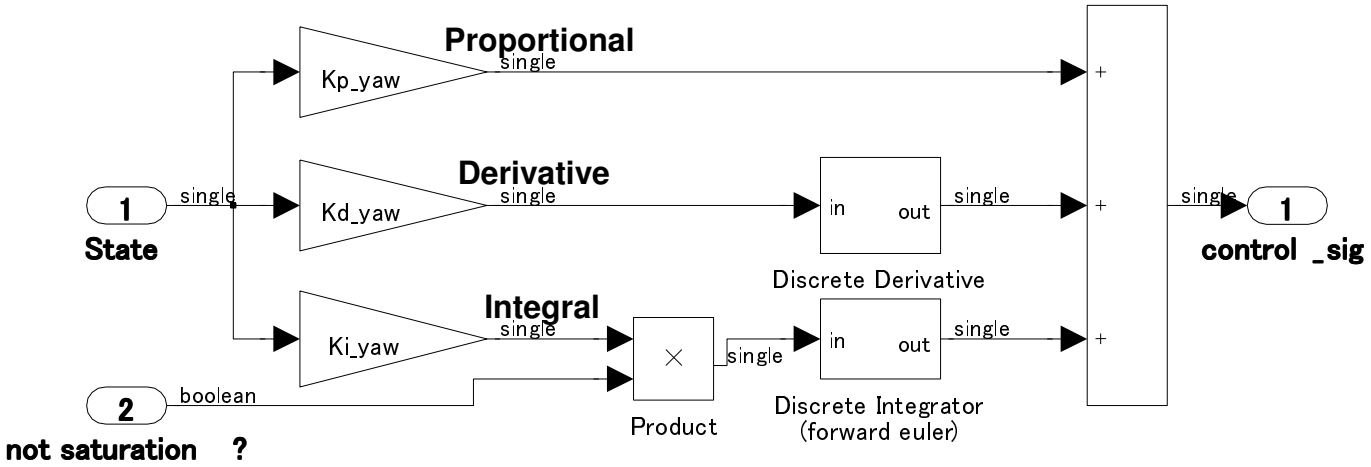
# Low Pass Filter Yaw



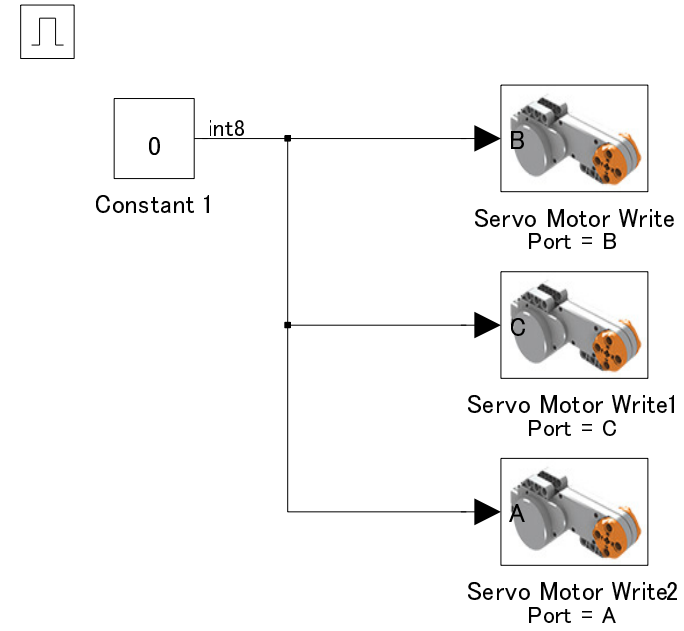

**Low Pass Filter Function**

$$\frac{1\text{-a\_yaw\_ms}}{1\text{-a\_yaw\_ms} * z^{(-1)}}$$

# PID Controller



# Stop

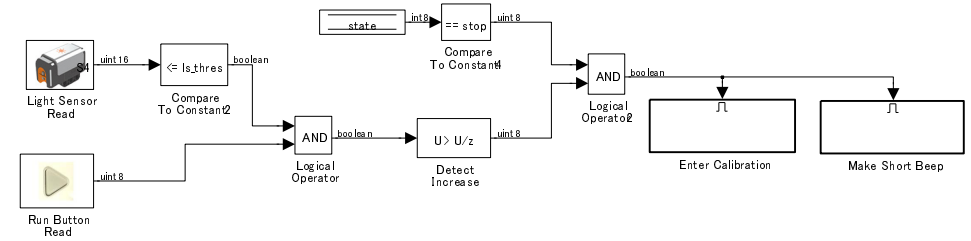

0
int8
Constant 1
B
Servo Motor Write
Port = B
C
Servo Motor Write1
Port = C
A
Servo Motor Write2
Port = A

## State Check & Battery Average Task

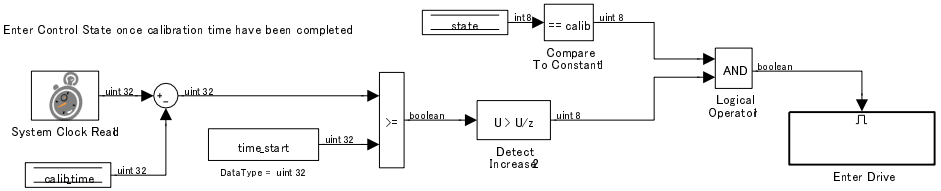
**Smooth measured battery value using Low Path Filter to reduce the noise .**

f()

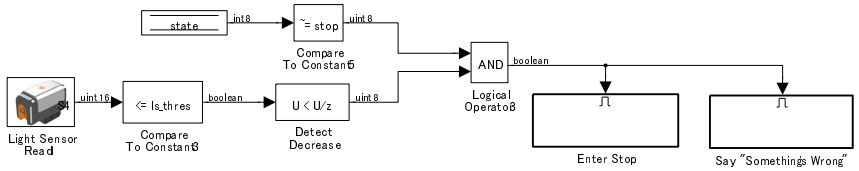
Enter Calibration if in Stop state, ball is present and run button is pressed



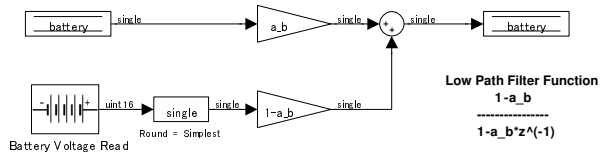

Enter Control State once calibration time have been completed



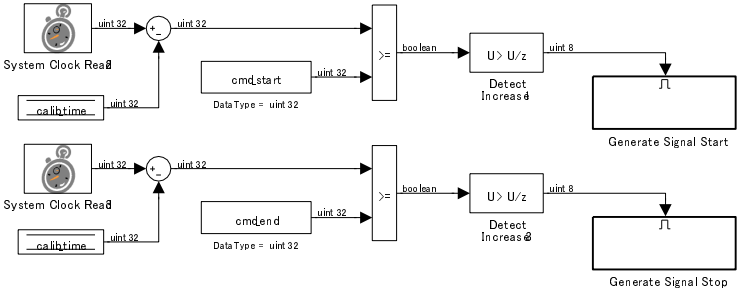

Enter Stop state if ball is removed



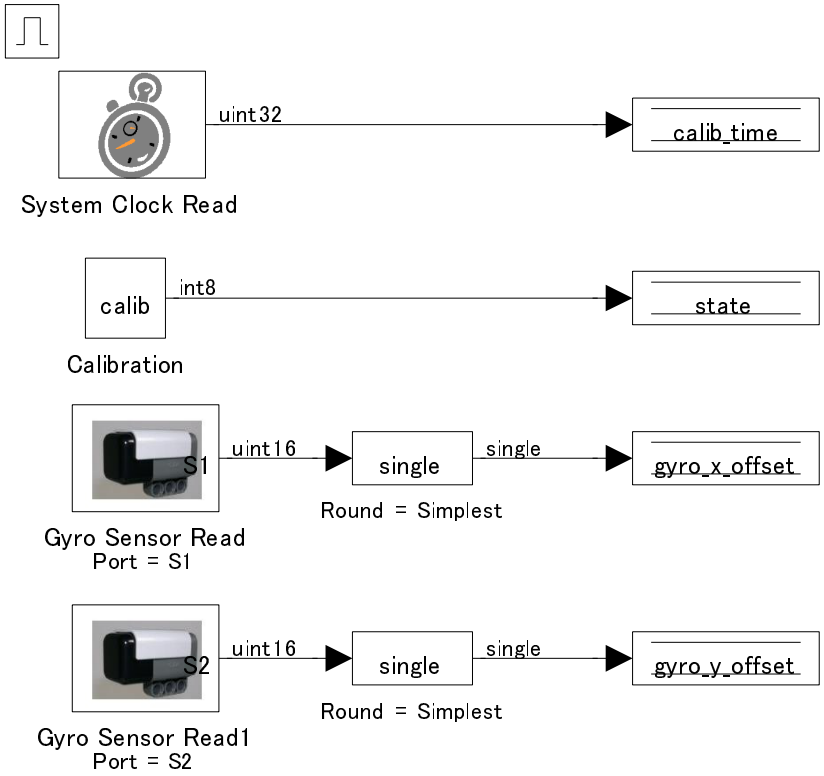

Find Battery Voltage (low pass filter reading)



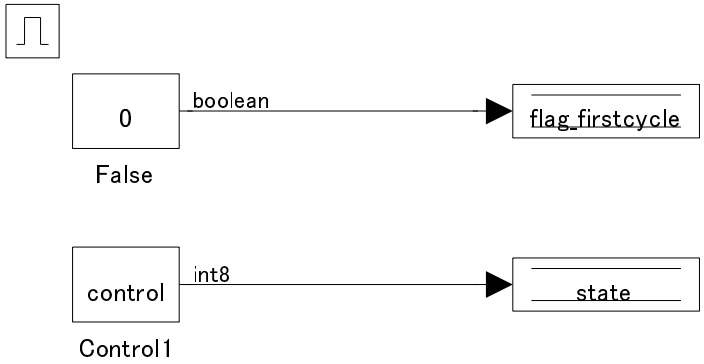

Generate a Command Signal (Start moving at 30000 ms, stop at 38000 ms)



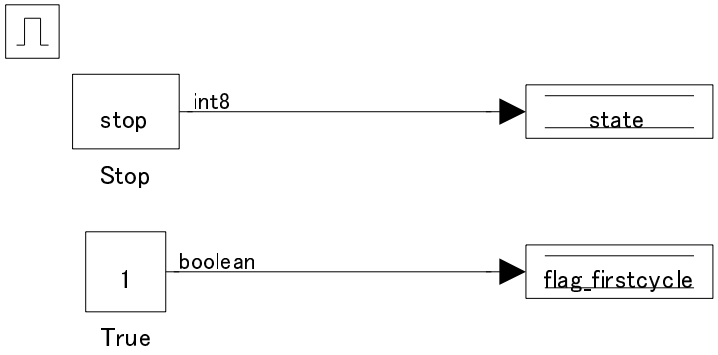
# Enter Calibration



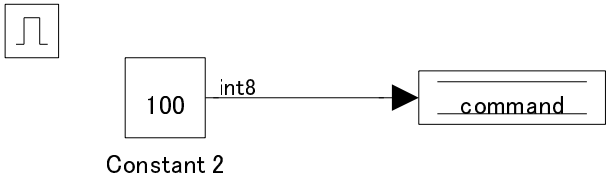
# Enter Drive


0
boolean
flag_firstcycle
False
control
int8
state
Control1

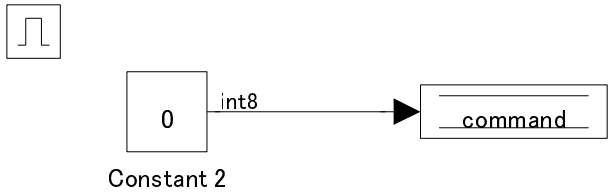
# Enter Stop



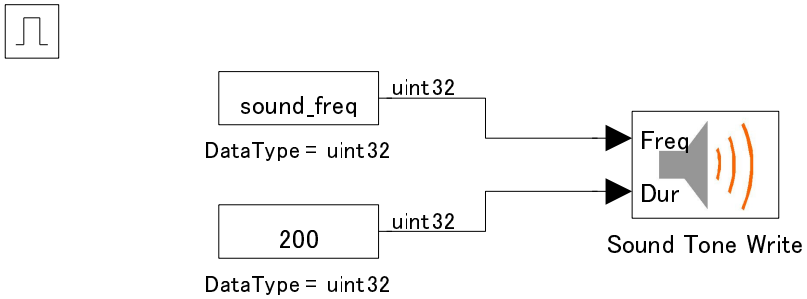
# Generate Signal Start



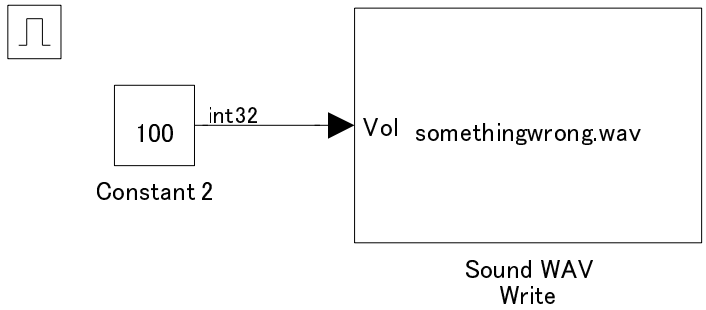
# Generate Signal Stop



# Make Short Beep



# Say "Something's Wrong"



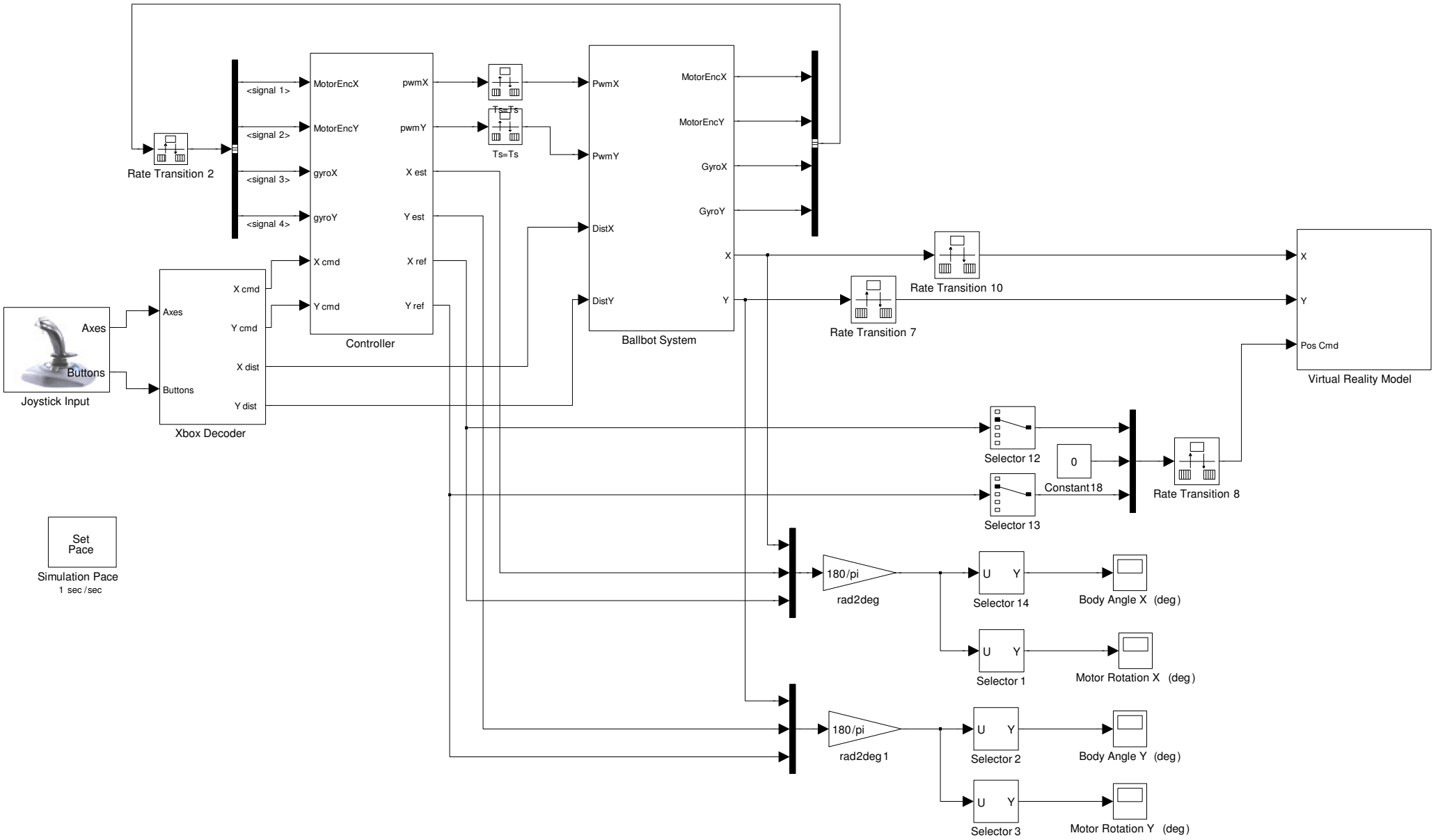
# F Virtual Reality Model Simulink Program

Presented in this appendix is the program for the virtual reality model of the Lego Ballbot.

```
% Parameters for the Virtual Reality Simulation of the Lego Ballbot
% Final Year Project 899 (2009)
% Justin Fong, Simon Uppill

% Sample time of the plant
Ts = 0.0005;
Tsc = 0.004;
Tvr = 0.1;

% Initial Values for the States
ballbot_x_init = [4*pi/180 0 0 0];
ballbot_y_init = [4*pi/180 0 0 0];

% Values for the Actuators
max_pwm = 100;
max_volt = 8;

% Values for the Sensors
gyro_x_offset = 1000;
gyro_y_offset = 1000;

% Parameters for VR
r_w = 18.4;          % radius of the wheel
r_b = 26;            % radius of the ball
Kimpulse = 600;      % Magnitude of the impulse for disturbance
Kcmd = 10;           % Scale the speed of the command signal

% Gains of the Controller
K = [100 -1 15 -1.3 -0.635]';

% Low Pass Filter Coefficients
a_d = 0.95;          % Smooth Phi Estimate
a_g = 0.95;          % Smooth Theta estimate
a_cam = 0.9;         % Lag camera slightly
a_m = 0.9;           % Simulates lag in motor reponse

% Parameters of Coulombic & Viscous Friction
pwm_gain = 1;                       % pwm gain
pwm_offset = 0;                     % pwm offset
```

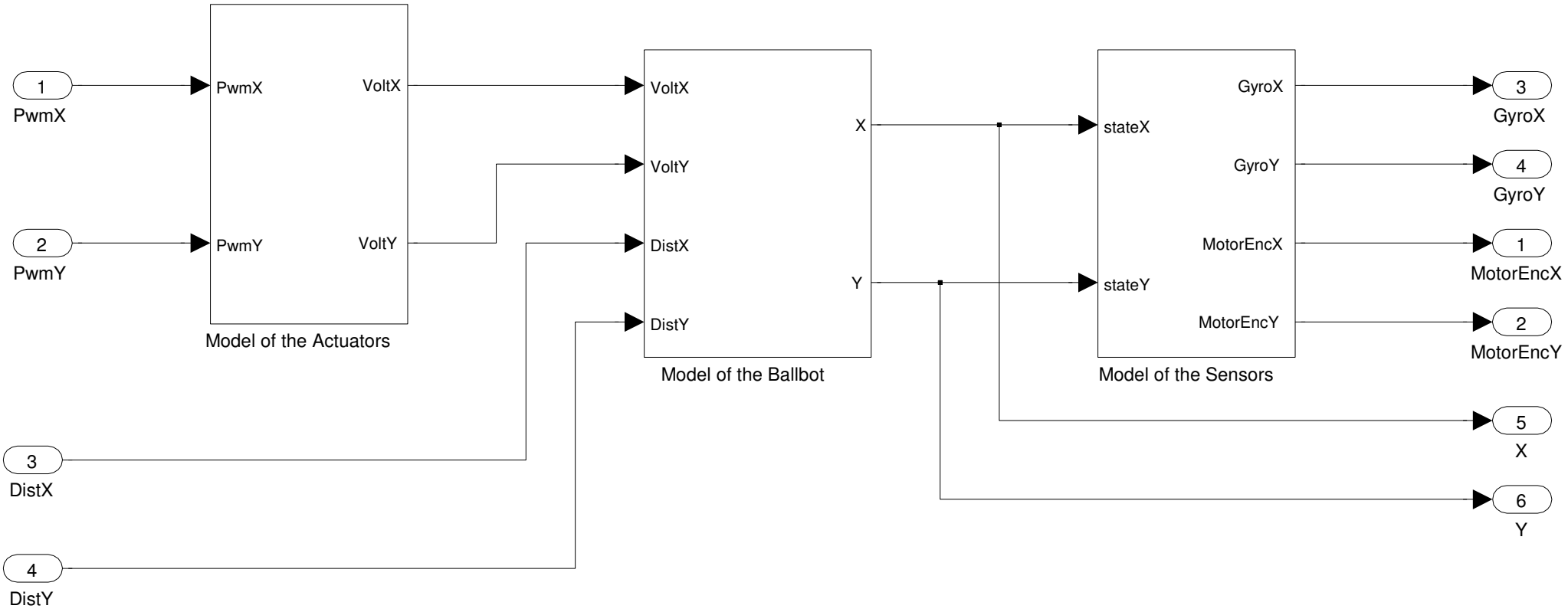
# Virtual Reality Model of the Lego Ballbot

Developed as part of Final Year Project 899 (2009)



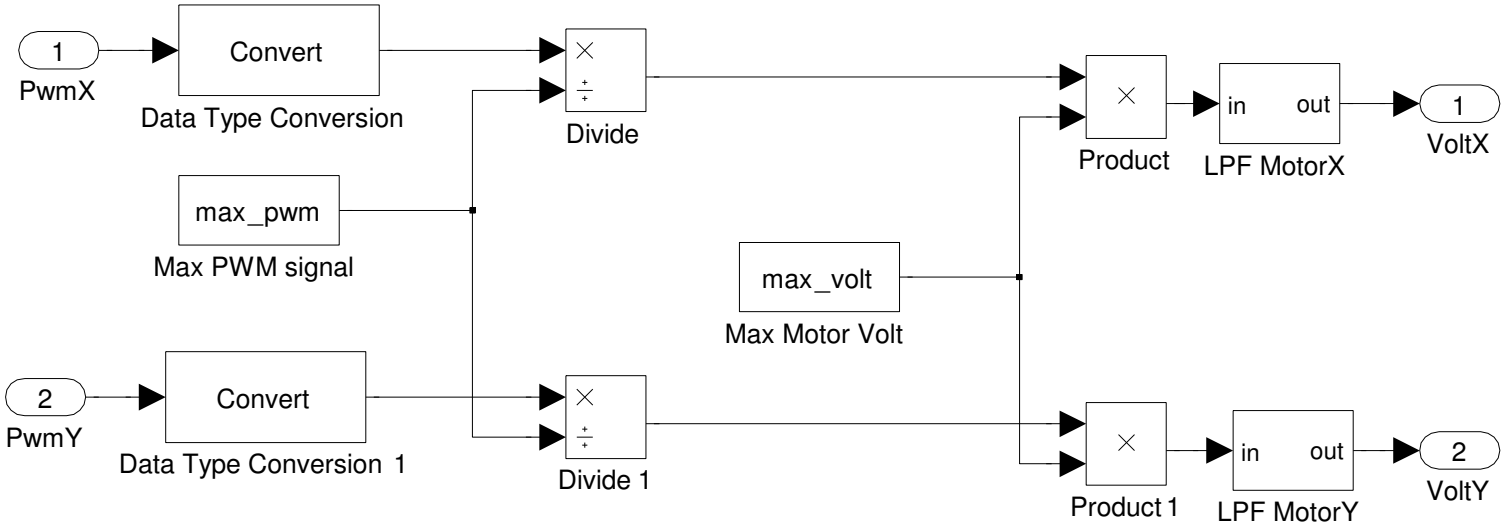
# Ballbot System

**Models the Actuators , Sensors and the Ballbot**

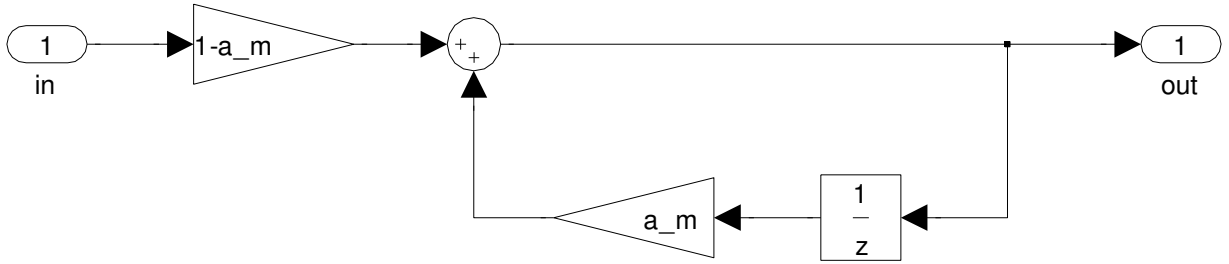


# Model of the Actuators

**Changes the PWM signal to a voltage for the Motors**
**-100 to 100 => -8 to 8**



# Low Pass Filter Motor X/Y

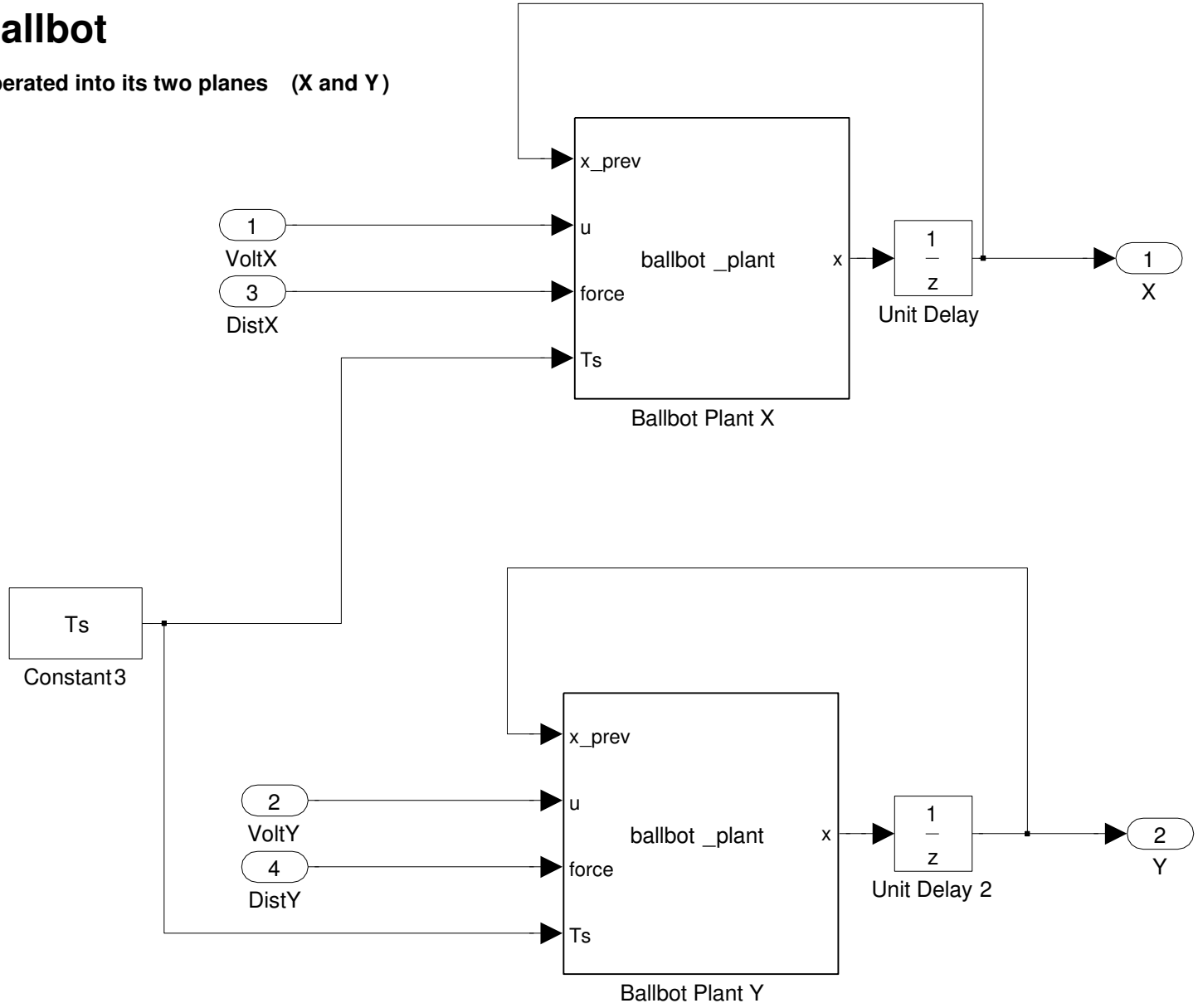



**Low Pass Filter Function**

$$\frac{1-a_m}{1-a_m z^{-1}}$$

# Model of the Ballbot

**Ballbot Model is seperated into its two planes (X and Y)**

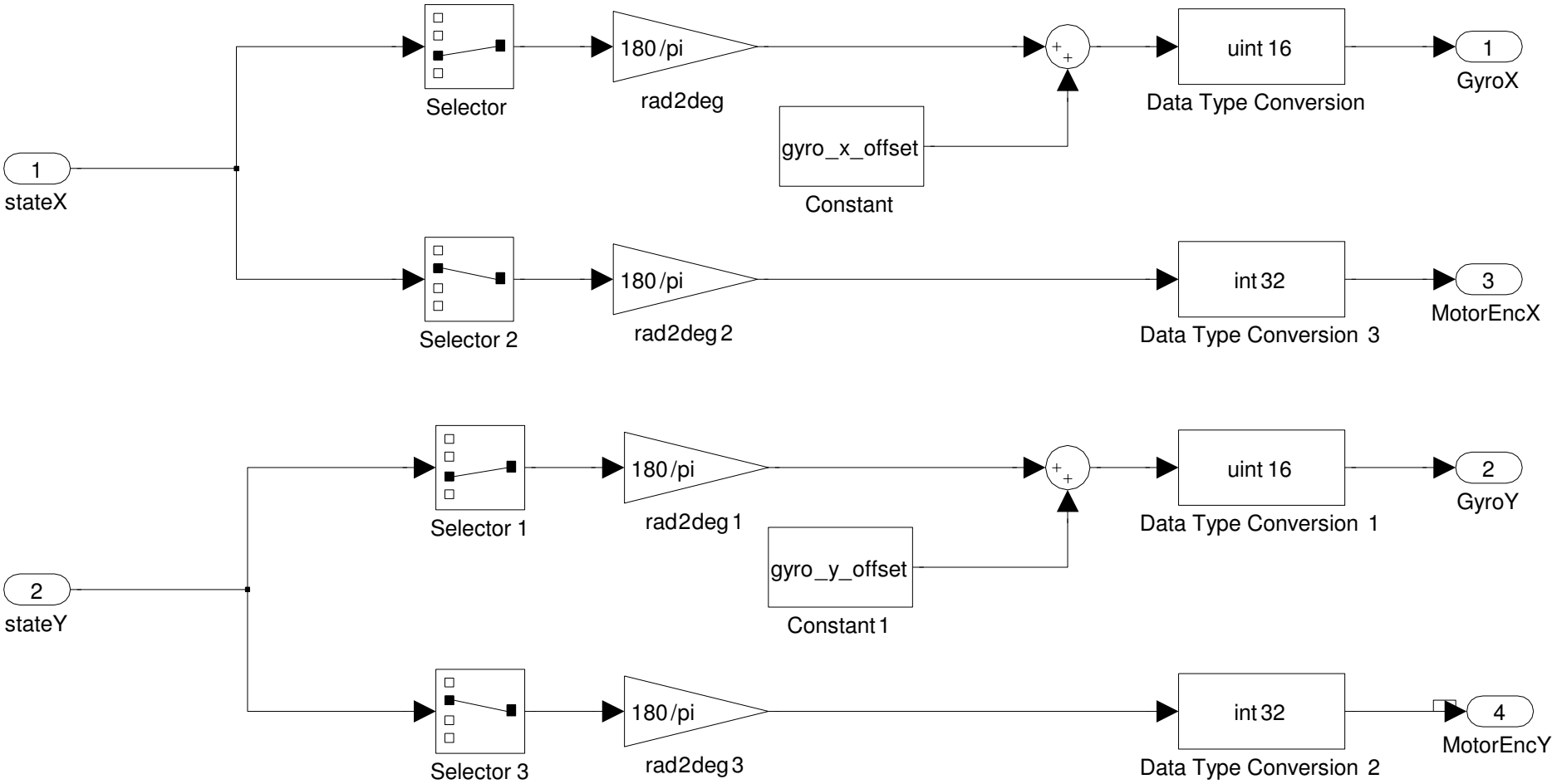


```
function x = ballbot_plant(x_prev, u, force, Ts)
% Ballbot Plant - Script Simulating the Dynamics of the Ballbot Plant

theta = x_prev(1);
phi = x_prev(2);
thetaDot = x_prev(3);
phiDot = x_prev(4);
volt = u;
dist = force;

% Calculate the second derivatives from the dynamics script
thetaDDot = (28650178602873656997263181 09*((329043*sin(theta)*thetaDot^2)/125000000 +
(24830091*sin(theta))/25000000))/(974218671392785694720000000000*
((167286029305964047723 53*cos(theta))/92233720368547758080000000000 -
(5540953489929*cos(theta)^2)/1562500000000000000 +
155840485497206506433628206437824920625804282655305 93/280799414109563478150929241484222
791680000000000000000000)) - (((2353923*cos(theta))/1250000000 +
271977772365706212256358 13/749398977994450534400000000000)*((2353923*sin(theta)
*thetaDot^2)/1250000000 + (27179*phiDot)/1115000 - (317*volt)/6690))/
((167286029305964047723 53*cos(theta))/92233720368547758080000000000 -
(5540953489929*cos(theta)^2)/1562500000000000000 +
155840485497206506433628206437824920625804282655305 93/280799414109563478150929241484222
791680000000000000000000) + dist;
phiDDot = (((2353923*cos(theta))/1250000000 +
271977772365706212256358 13/749398977994450534400000000000)*((329043*sin(theta)
*thetaDot^2)/125000000 + (24830091*sin(theta))/25000000))/((167286029305964047723 53*cos
(theta))/92233720368547758080000000000 - (5540953489929*cos(theta)^2)
/1562500000000000000 +
155840485497206506433628206437824920625804282655305 93/280799414109563478150929241484222
791680000000000000000000) - (((329043*cos(theta))/62500000 +
222740783771944914054205 9/115292150460684697600000000)*((2353923*sin(theta)*thetaDot^2)
/1250000000 + (27179*phiDot)/1115000 - (317*volt)/6690))/((167286029305964047723 53*cos
(theta))/92233720368547758080000000000 - (5540953489929*cos(theta)^2)
/1562500000000000000 +
155840485497206506433628206437824920625804282655305 93/280799414109563478150929241484222
791680000000000000000000);

x= [0 0 0 0];
% Calculate the first derivatives from the second derivatives and previous value
x(3) = thetaDot + thetaDDot*Ts;
x(4) = phiDot + phiDDot*Ts;

% Calculate theta and phi from the derivatives and previous value
x(1) = theta+x(3)*Ts;
x(2) = phi+x(4)*Ts;
```

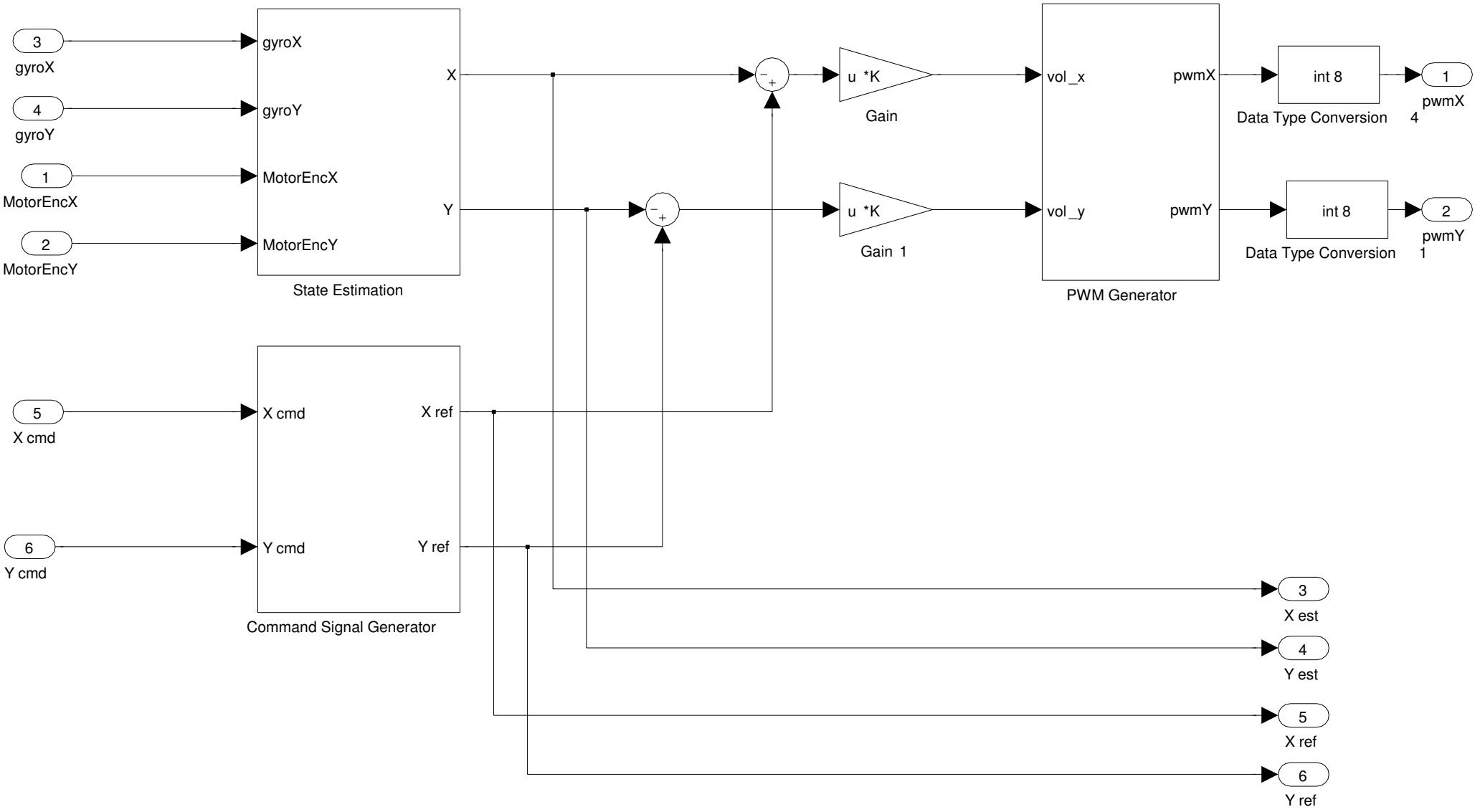
# Model of the Sensors

**Calculate the Gyro Signal and Motor States**

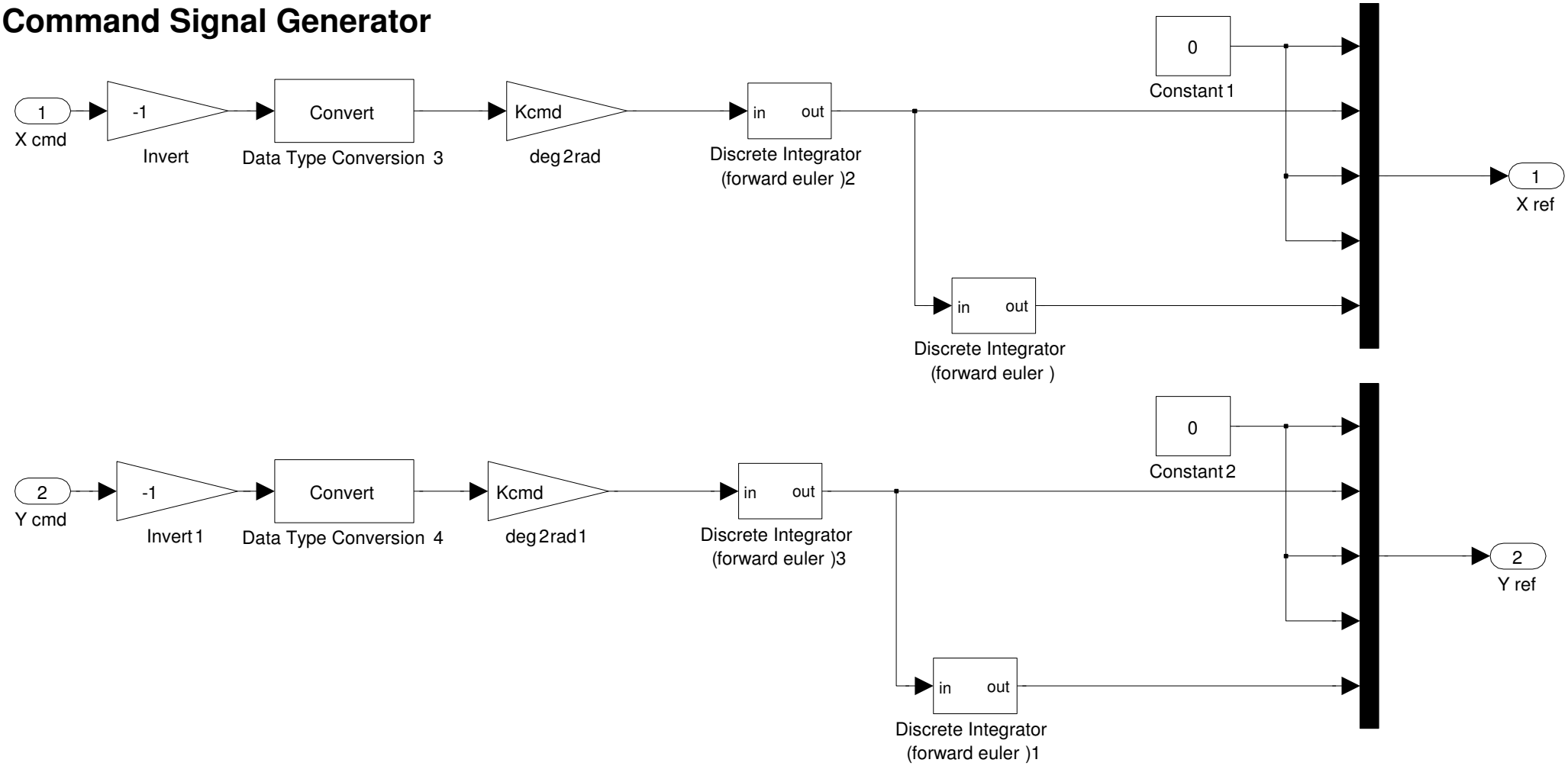
**NOTE: Does not model a changing gyro offset**



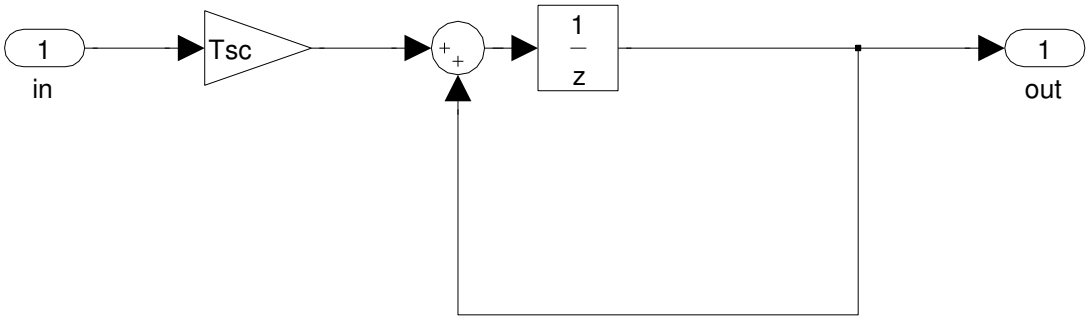
# Controller


3
gyroX
4
gyroY
1
MotorEncX
2
MotorEncY
gyroX
gyroY
MotorEncX
MotorEncY
X
Y
State Estimation
u *K
Gain
u *K
Gain 1
vol _x
vol _y
pwmX
pwmY
PWM Generator
int 8
Data Type Conversion
int 8
Data Type Conversion
1
pwmX
4
2
pwmY
1
5
X cmd
6
Y cmd
X cmd
Y cmd
X ref
Y ref
Command Signal Generator
3
X est
4
Y est
5
X ref
6
Y ref

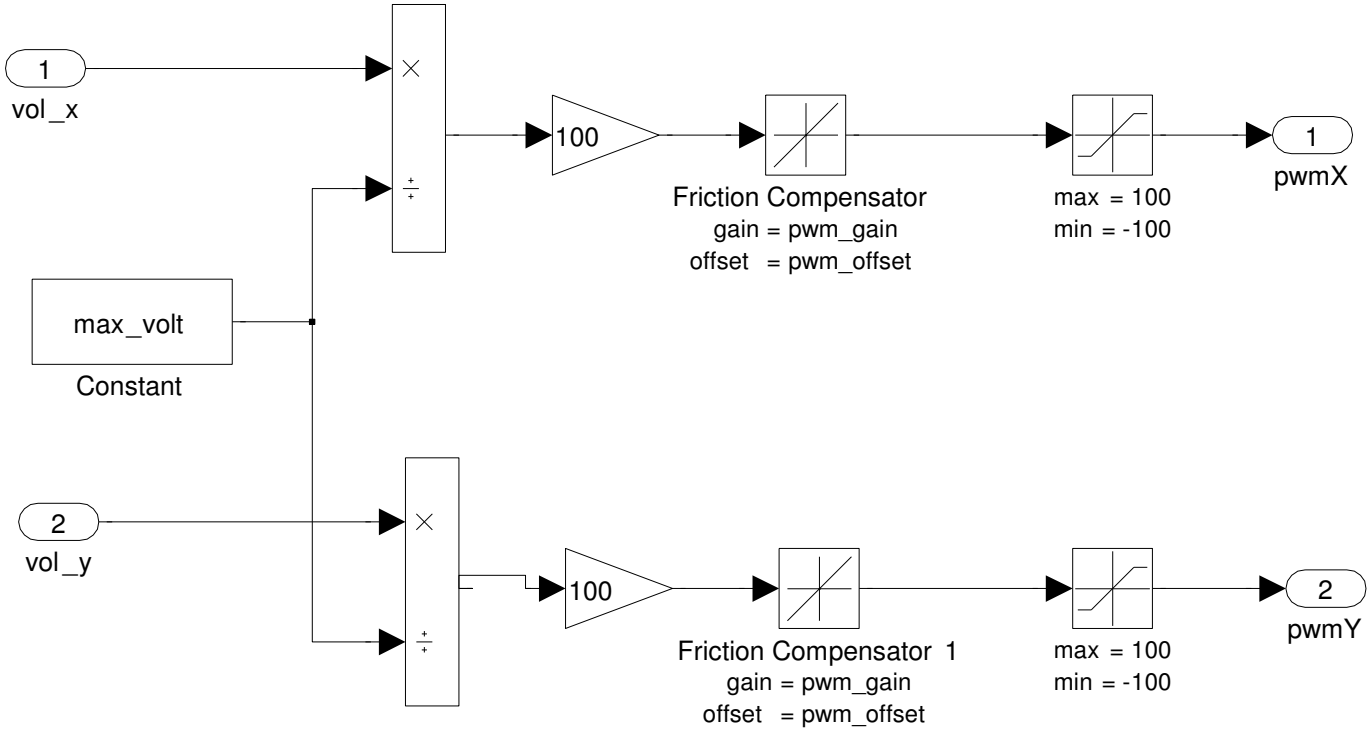
# Command Signal Generator



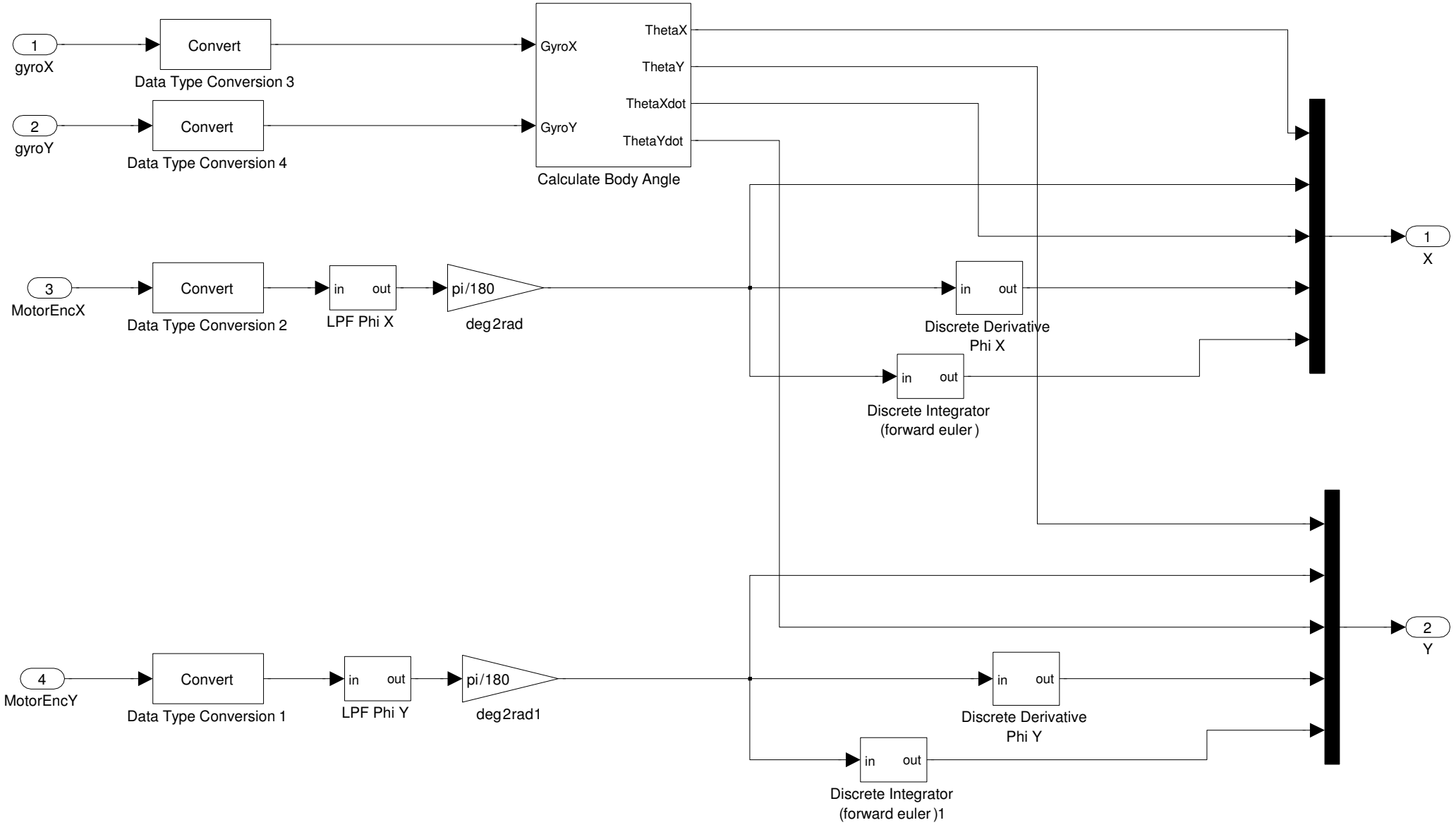
# Discrete Integrator



# PWM Generator



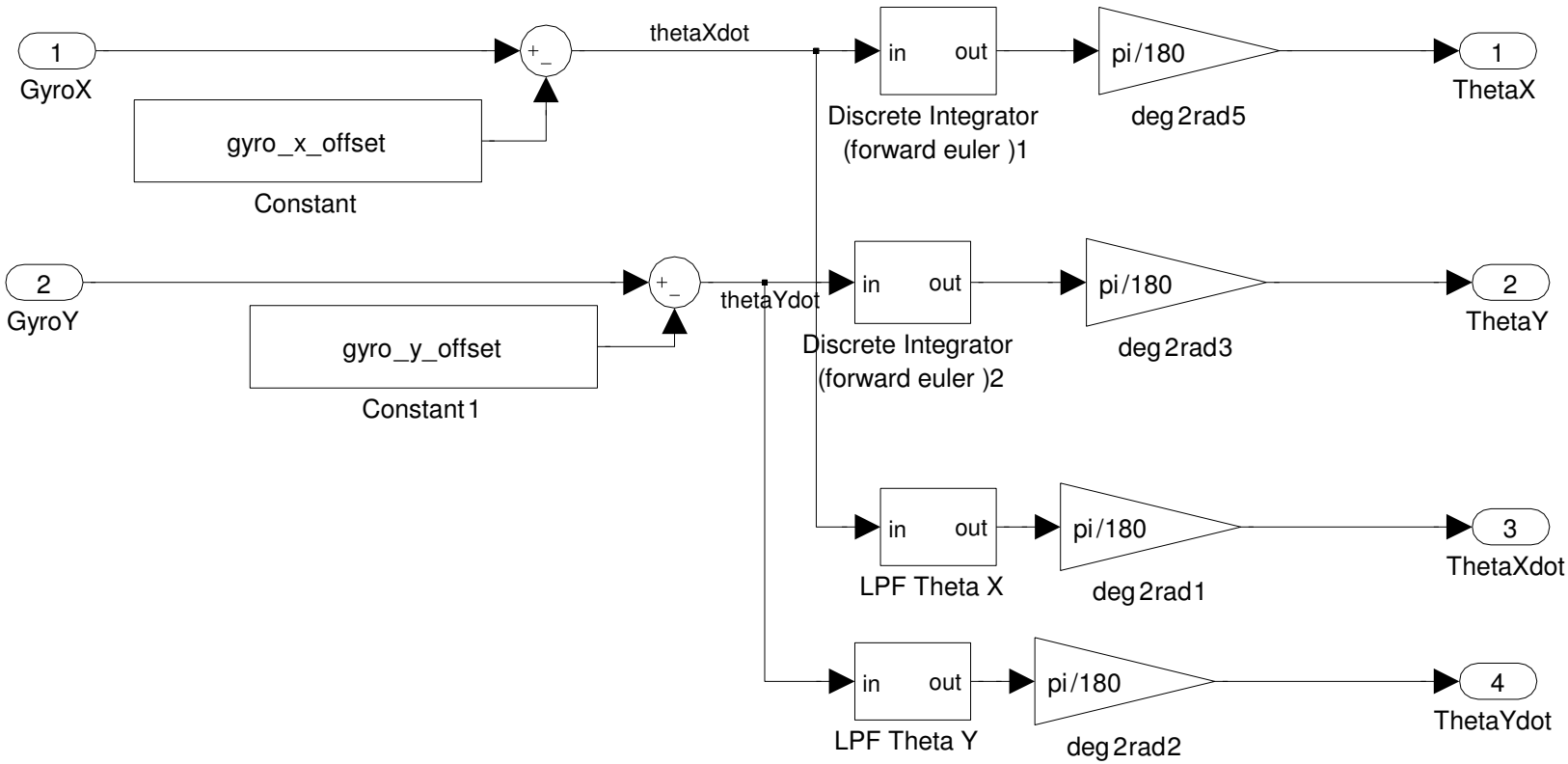
# State Estimation



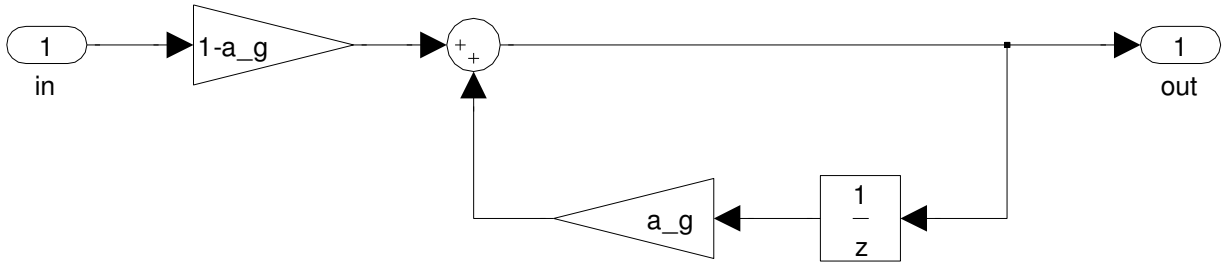
# Calculate Body Angle

**Calculate body angle in x and y planes using the Gyroscope Reading**

**NOTE: Gyro offsets are not calculated in this simulator**



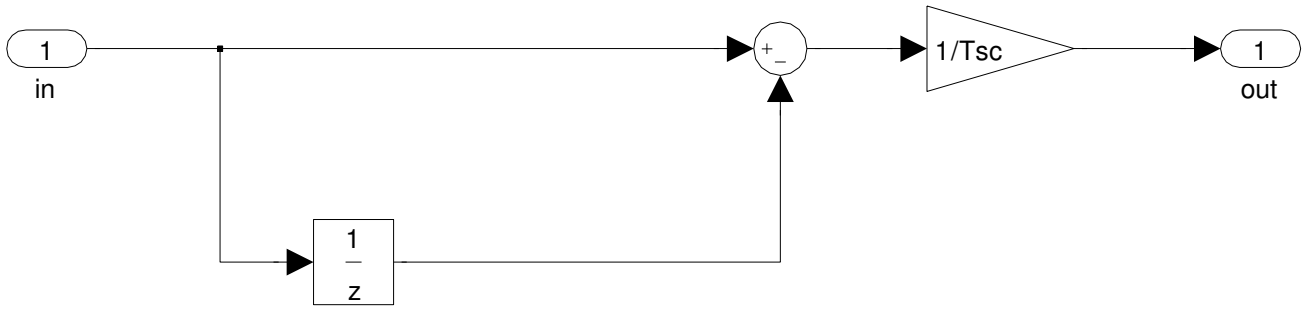
# Low Pass Filter Theta X/Y



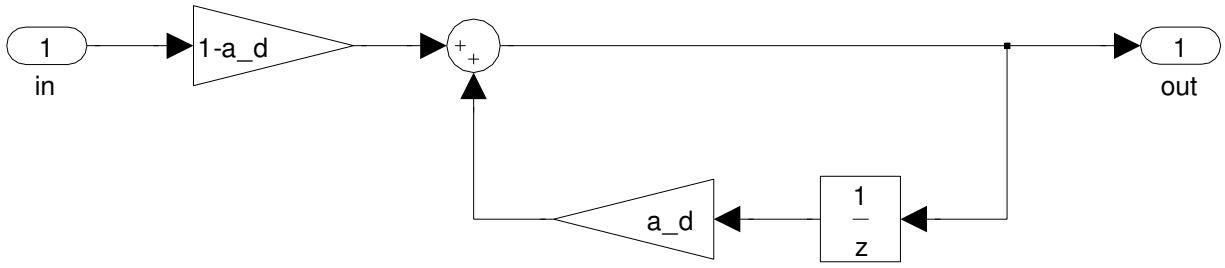

**Low Pass Filter Function**

$$\frac{1-a\_g}{1-a\_g*z^{(-1)}}$$

# Discrete Derivative



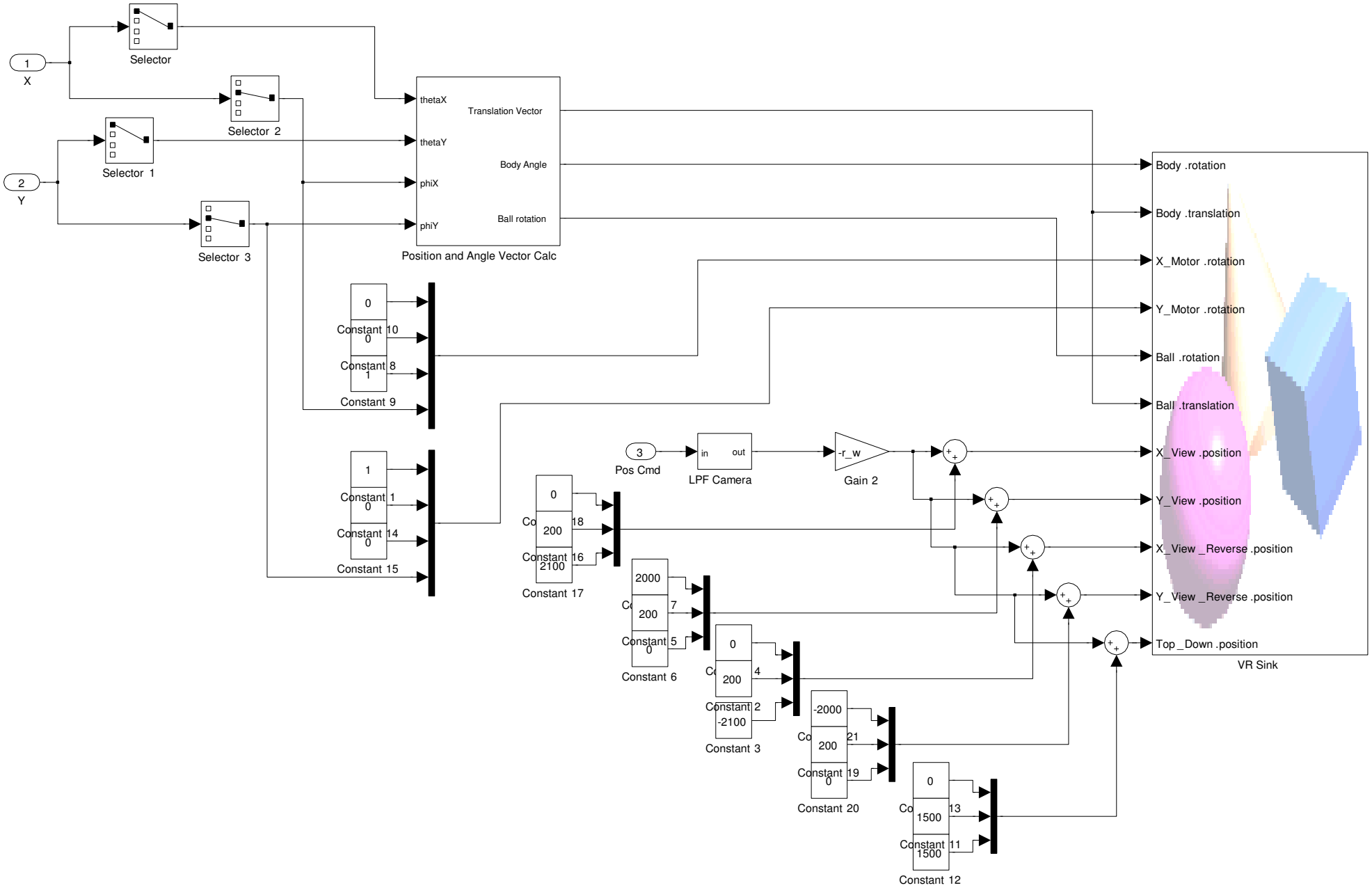
# Low Pass Filter Phi X/Y




**Low Pass Filter Function**

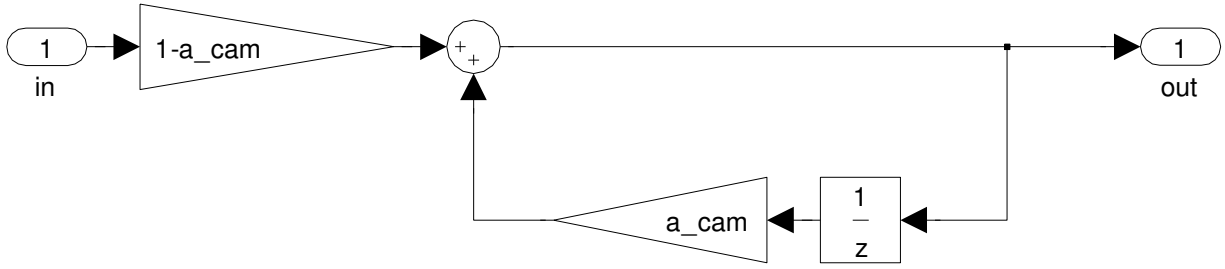
$$\frac{1-a\_d}{1-a\_d*z^{(-1)}}$$

## Virtual Reality Model




Position of Camera Tracks movement of Ballbot

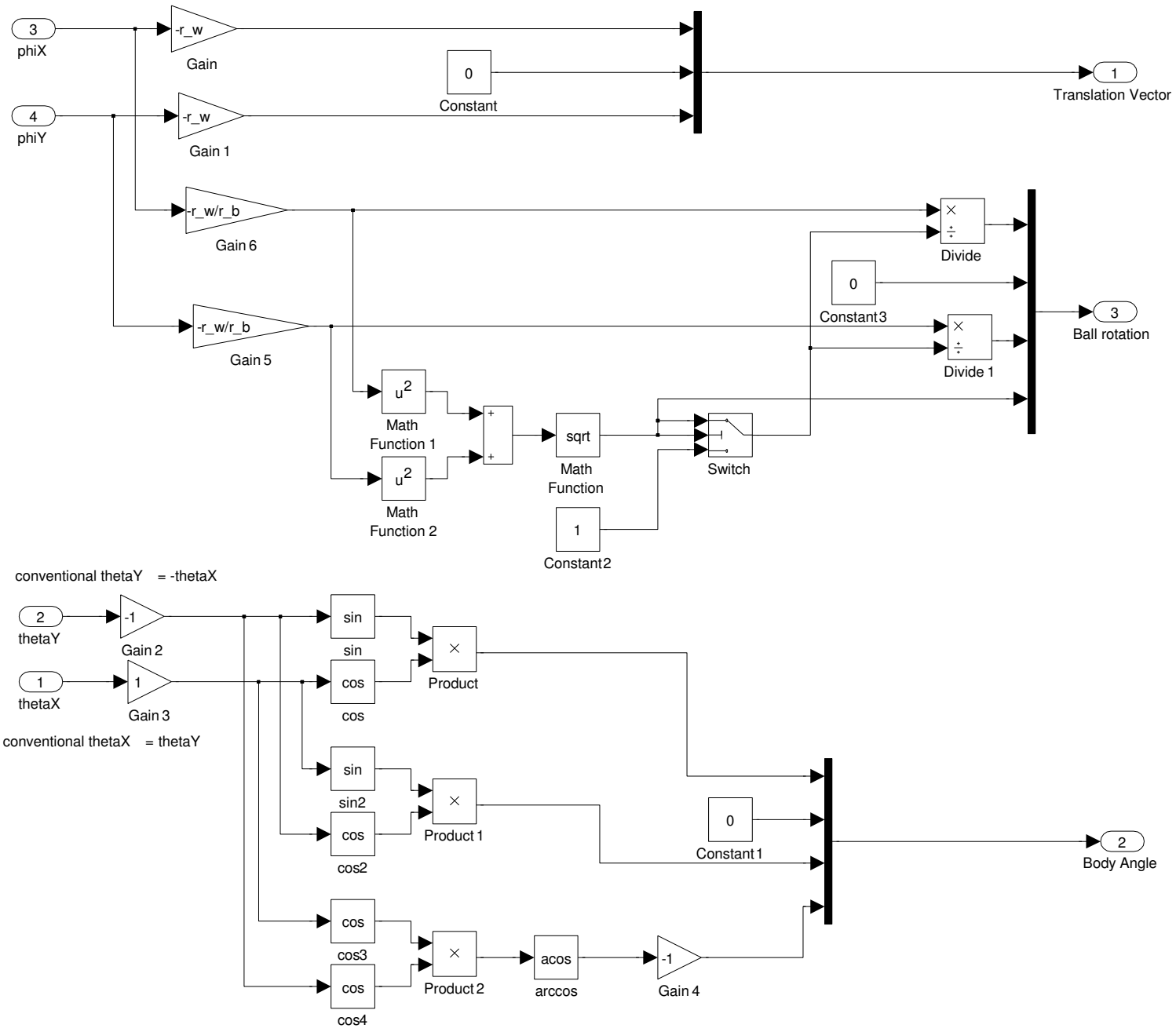
# Low Pass Filter Camera




**Low Pass Filter Function**

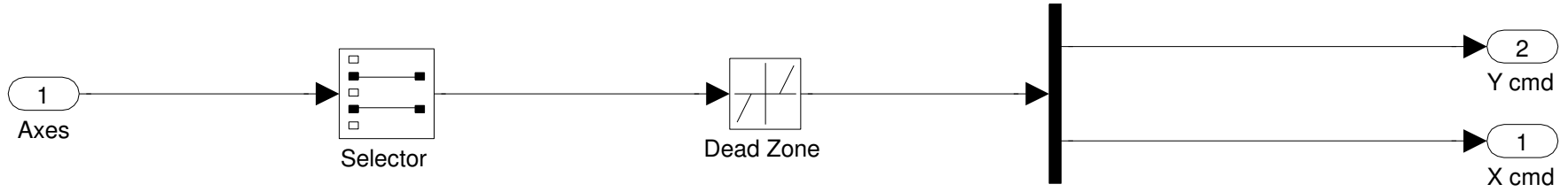
$$\frac{1-a\_cam}{1-a\_cam*z^{(-1)}}$$

## Position and Angle Vector Calculation

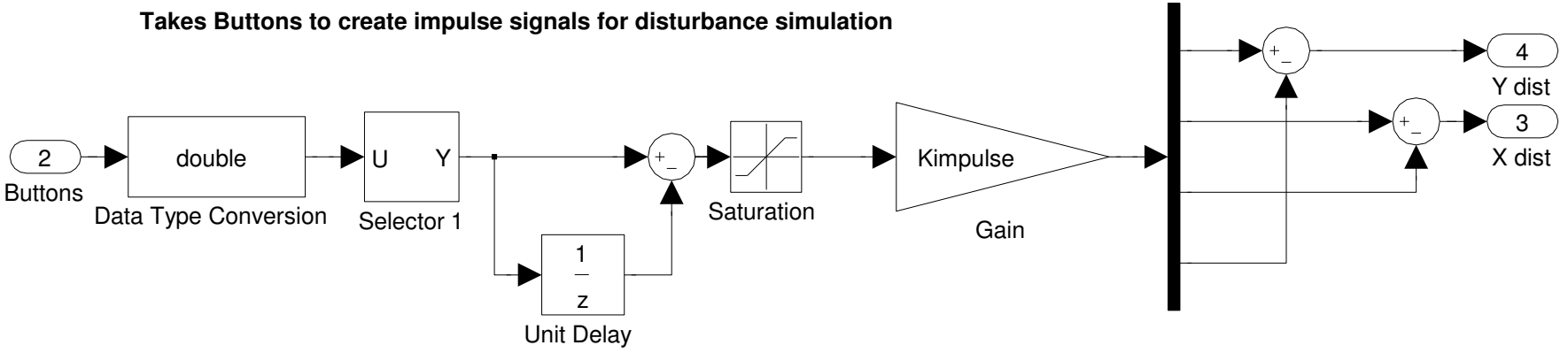


# XBox Decoder

**Takes the Up/Down Signal from the left joystick , and the Left/Right movment from the right joystick to create movement commands**




**Takes Buttons to create impulse signals for disturbance simulation**



# G Component Datasheets

This appendix includes the datasheets of components to be used in the Full Scale Ballbot.

# DCM50xxx/57xxx Series

## DC Brush Servo Motors

### *Product Description*

The DCM50xxx/57xxx series motors are permanent magnet DC brush servo motors. The motors are ideal for cost sensitive applications. They include an attached encoder which provides position feedback to controllers.

### *Features*

- High performance (Smooth operation, High precision and Low noise)
- Low cost
- Small disturbance
- Encoder optional (1000 line or 500 line), Positional error can be eliminated to one pulse
- Mounting dimensions of DCM57xxx brush servo motors are the same as those of NEMA frame size 23 motors

### *Typical Applications*

- Engraving machines, Cutting machines, Jet-ink machines
- Experimental installations and Instructional machines
- Measuring devices, such as Imager, Analytic Instruments, etc.
- Electronic packaging equipments and Loading and unloading devices
- Medical instruments

### *Brush Servo Motor Specifications*

| No. | Parameter | Symbol | Units | DCM50202A | DCM57202 | DCM5x205 | DCM 5x207 |
|---|---|---|---|---|---|---|---|
| 1 | Continuous Torque (Max) | $T_C$ | N·m | $98.9\times10^{-3}$ | $98.9\times10^{-3}$ | $218.9\times10^{-3}$ | $353.1\times10^{-3}$ |
| 2 | Peak Torque (Stall) | $T_{PK}$ | N·m | 0.76 | 0.76 | 1.59 | 2.90 |
| 3 | No-load Speed | $S_{NL}$ | rpm | 4600±10% | 4600±10% | 4000±10% | 3600±10% |
| 4 | Rated Speed | $S_R$ | rpm | 3500 | 3500 | 3000 | 2900 |
| 5 | Rotor Inertia | $J_M$ | kg ·$m^2$ | $1.62\times10^{-5}$ | $1.62\times10^{-5}$ | $3.11\times10^{-5}$ | $4.73\times10^{-5}$ |
| 6 | Maximum Winding Temperature | $\theta_{MAX}$ | °C | 155 | 155 | 155 | 155 |
| 7 | Thermal Impedance | $R_{TH}$ | °C/watt | 9.00 | 9.00 | 7.30 | 4.98 |
| 8 | Motor Weight （Plus encoder） | $W_M$ | g | 694 | 754 | 1182 | 1338 |
| 9 | Motor Length （Plus encoder） | $L_1$ | mm | 121±2 | 129±2 | 161±2 | 196±2 |
| 10 | Rated Voltage | E | V | 24 | 24 | 24 | 30.3 |
| 11 | Rated Current | I | A | 1.79 | 1.79 | 2.95 | 3.94 |
| 12 | Torque Constant | $K_T$ | N·m/A | $55.1\times10^{-3}$ | $55.1\times10^{-3}$ | $74.2\times10^{-3}$ | $89.7\times10^{-3}$ |
| 13 | Resistance | $R_T$ | Ω | 1.73 | 1.73 | 1.11 | 0.93 |
| 14 | No-Load Current | $I_{NL}$ | A | 0.5 | 0.5 | 0.8 | 0.6 |
| 15 | Peak Current (Stall) | $I_P$ | A | 13.9 | 13.9 | 21.6 | 32.6 |
| 16 | Encoder Specification | —— | Lines/rev | 500/1000 | 500/1000 | 500/1000 | 500/1000 |

## *Mechanical Specifications*

Mechanical specification of DCM57202 motor (plus encoder) is shown as Figure 1.

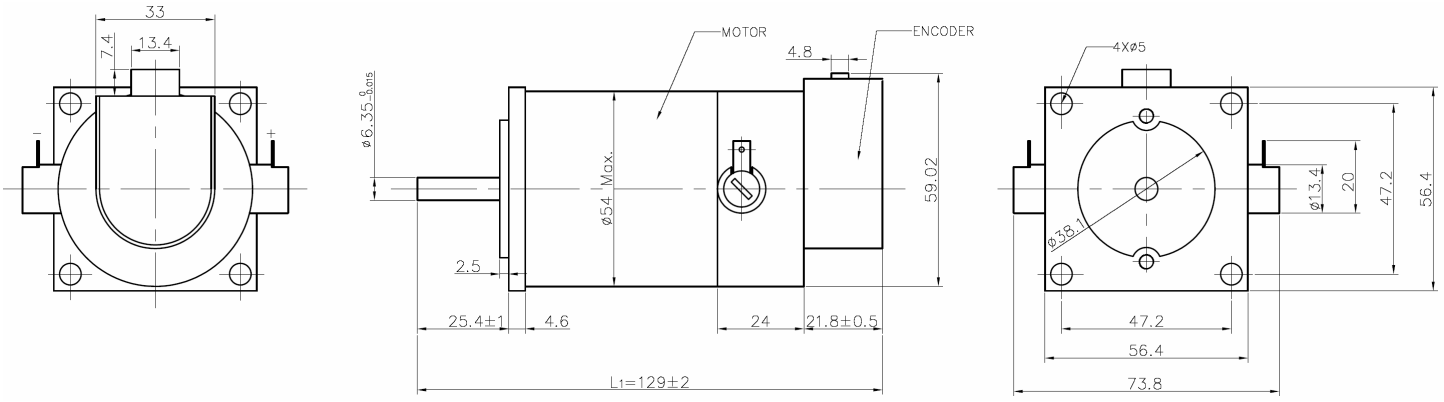



Figure 1: Mechanical specification of DCM57202 motor (plus encoder)

Mechanical specification of DCM57205 motor (plus encoder) is shown as Figure 2.

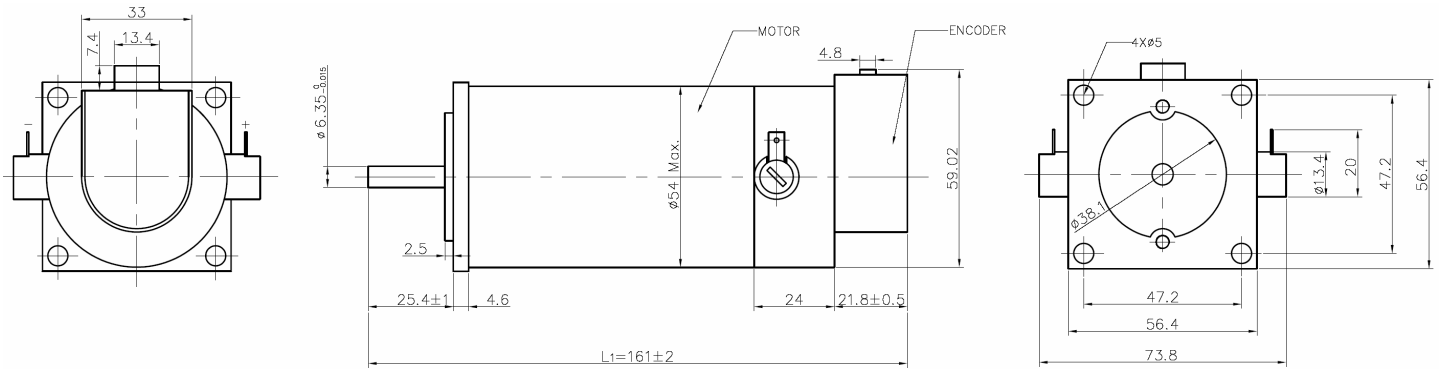



Figure 2: Mechanical specification of DCM57205 motor (plus encoder)

Mechanical specification of DCM57207 motor (plus encoder) is shown as Figure 3.

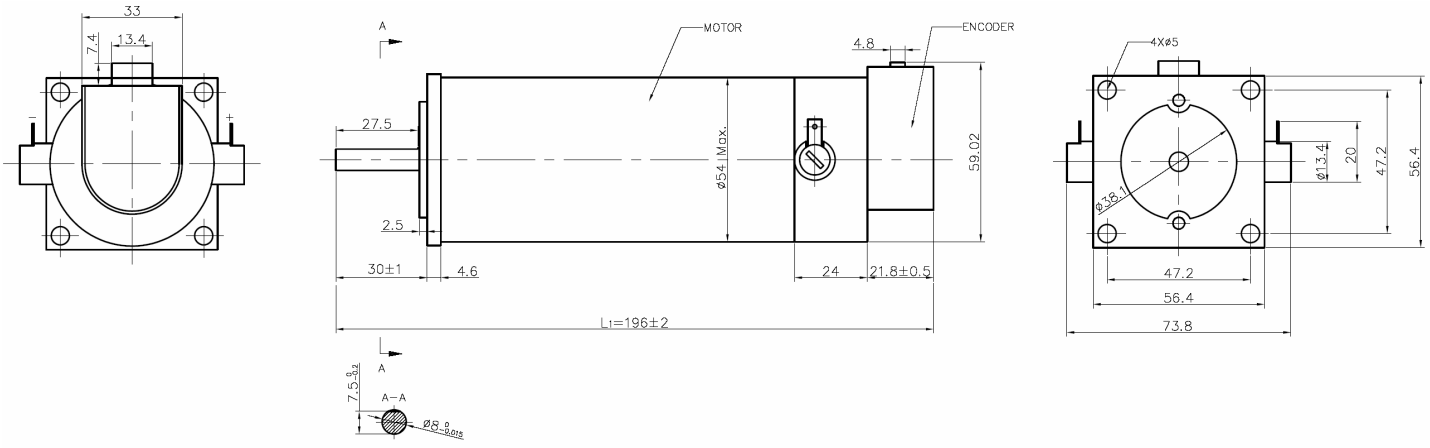



Figure 3: Mechanical specification of DCM57207 motor (plus encoder)

Mechanical specification of DCM50202A motor (plus encoder) is shown as Figure 4.

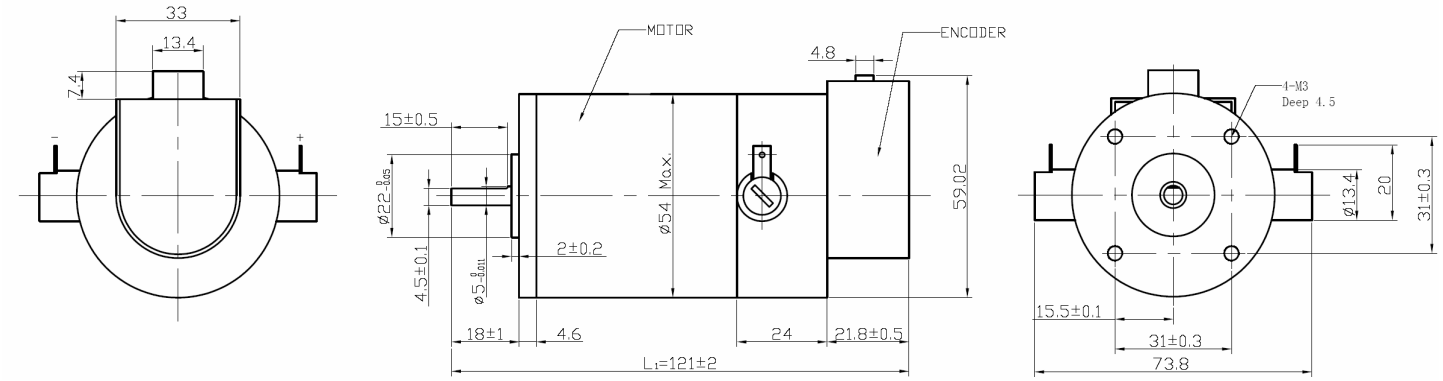



Figure 4: Mechanical specification of DCM50202A motor (plus encoder)

Mechanical specification of DCM50205 motor (plus encoder) is shown as Figure 5.

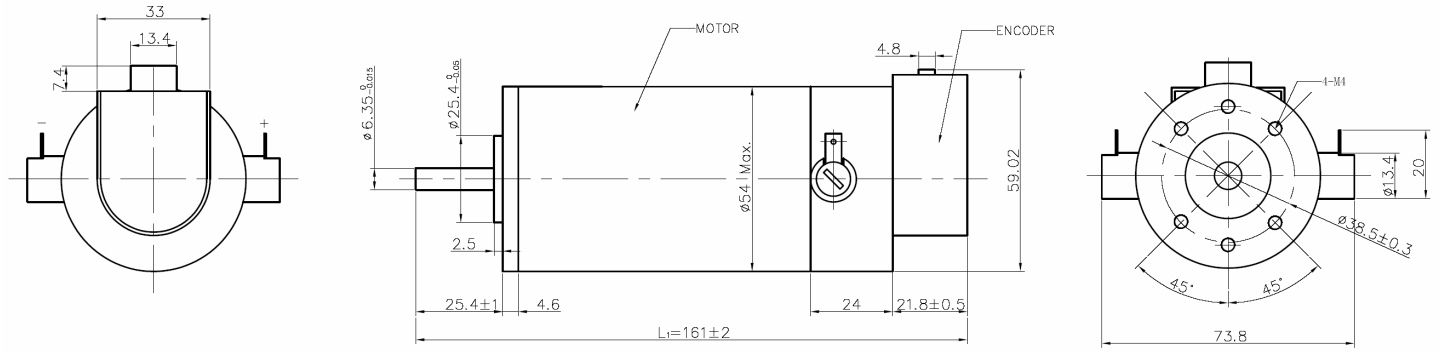



Figure 5: Mechanical specification of DCM50205 motor (plus encoder)

Mechanical specification of DCM50207 motor (plus encoder) is shown as Figure 6.

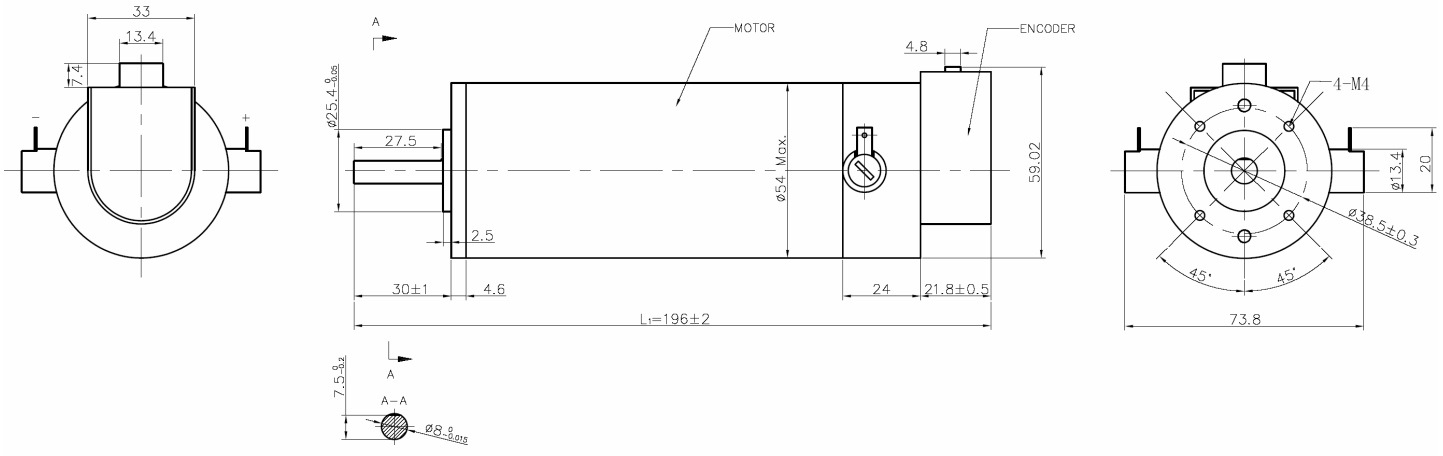



Figure 6: Mechanical specification of DCM50207 motor (plus encoder)

## *Encoder* (Line Count and Signal Type Optional)

DCM5xxxx series motors include an attached encoder, and the user can select line count and signal type of the encoder. The DCM5xxxx-1000 models include a 1000 line single-ended encoder and the DCM5xxxx-500 models include a 500 line single-ended encoder; while the DCM5xxxxD-1000 models include a 1000 line differential encoder and the DCM5xxxxD-500 models include a 500 line differential encoder. All encoders on the standard models have A signal and B signal. No Z signal is offered by the standard models, please specify the requirement when placing an order if the user needs an encoder which has Z signal.

- **Connection Chart for Single-ended Encoder:**

| Pin | Color | Connection (DB810-50V/DB810A) |
|---|---|---|
| 1 | Blue | Channel B (Phase B/EB+) |
| 2 | Yellow | Channel A (Phase A/EA+) |
| 3 | Red | VCC (E+5V/E+5V) |
| 4 | Black | Ground (EGND/EGND) |
| 5 | Green | Index/NC (NC/NC) |

- **Connection Chart for Differential Encoder:**

| Pin | Color | Connection (DB810A) |
|---|---|---|
| 1 | Black | Channel A+ (EA+) |
| 2 | Blue | Channel A- (EA-) |
| 3 | Yellow | Channel B+ (EB+) |
| 4 | Green | Channel B- (EB-) |
| 5 | Red | VCC (E+5V/E+5V) |
| 6 | White | Ground (EGND/EGND) |

## *Typical Connections*

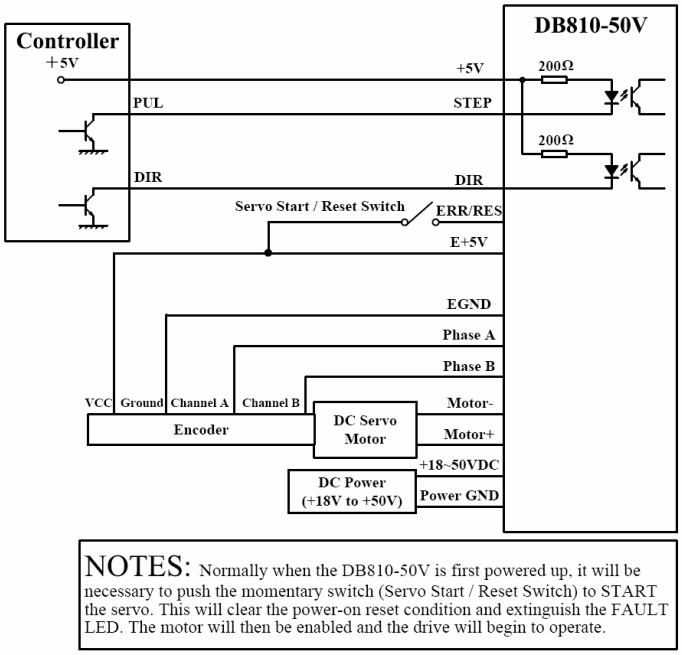



Figure 7: Typical connections of DCM5xxx series motors and DB810-50V servo driver

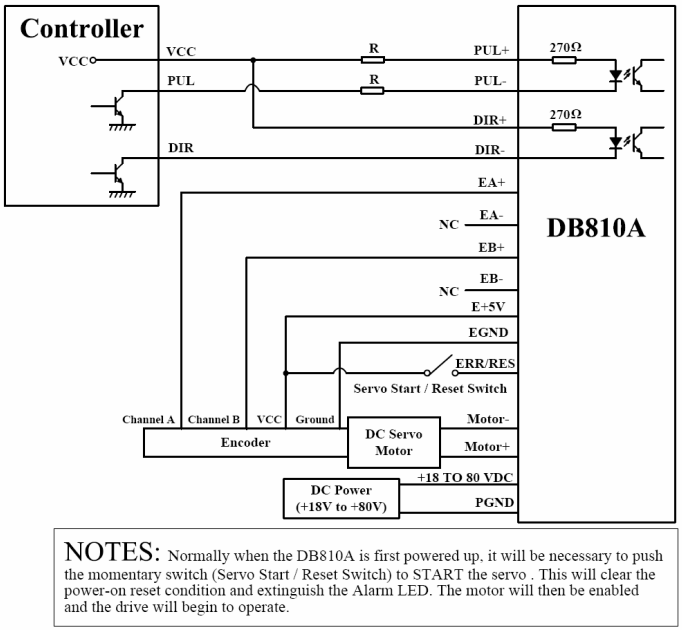



Figure 8: Typical connections of DCM5xxx series motors and DB810A servo driver

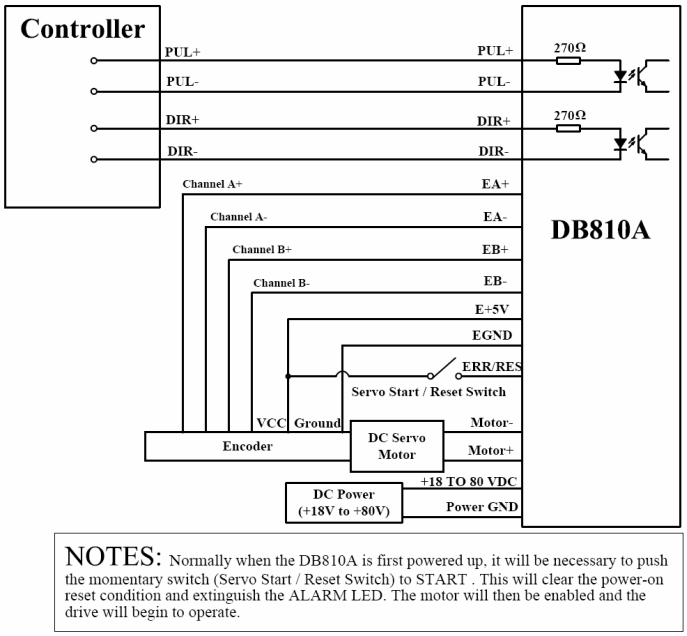



Figure 9: Typical connections of DCM5xxxD series motors and DB810A servo driver

## *Speed-Torque Characteristics*

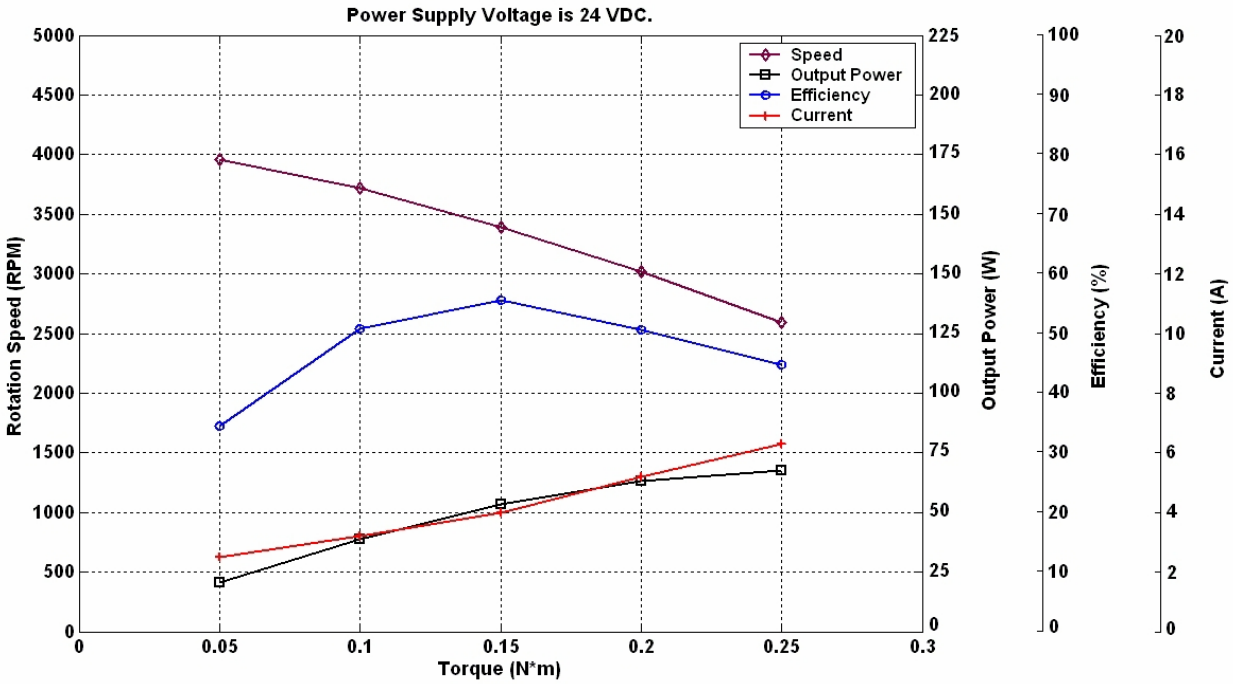



Figure 10: Speed-torque characteristics of DCM5x202 and DCM50202A

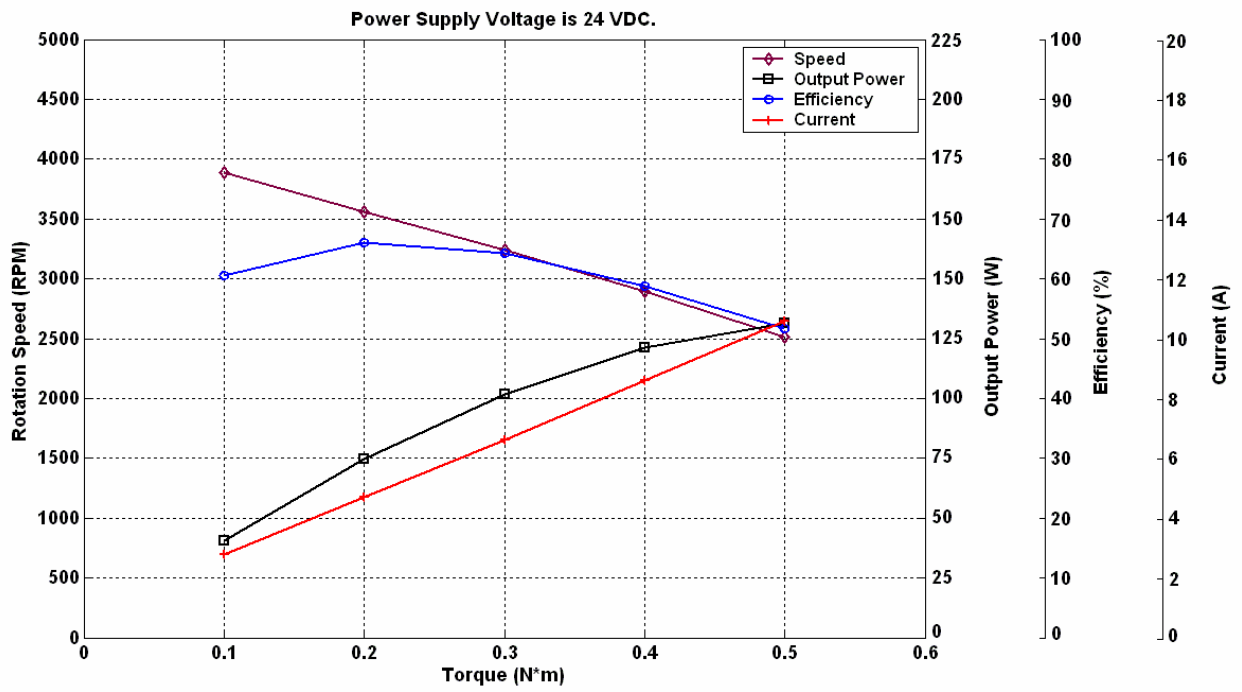



Figure 11: Speed-torque characteristics of DCM5x205

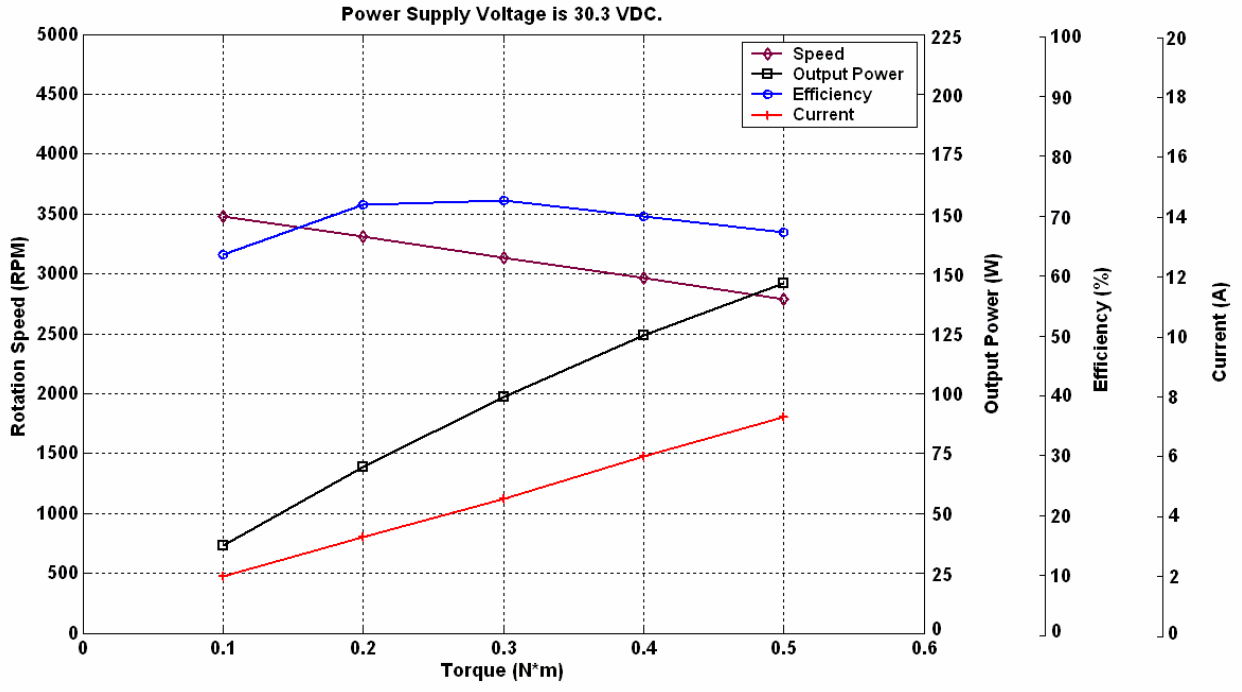



Figure 12: Speed-torque characteristics of DCM5x207

## *Order Information*

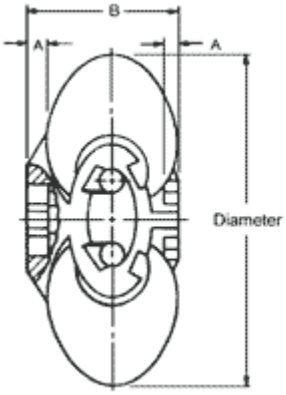
- DCM50xxx-1000 is a bolting motor including a single-ended 1000 line encoder, such as the DCM50202A-1000 motor, the DCM50205-1000 motor and the DCM50207-1000 motor.
- DCM50xxx-500 is a bolting motor including a single-ended 500 line encoder, such as the DCM50202A-500 motor, the DCM50205-500 motor and the DCM50207-500 motor.
- DCM57xxx-1000 is a flange connected motor including a single-ended 1000 line encoder, such as the DCM57202-1000 motor, the DCM57205-1000 motor and the DCM57207-1000 motor.
- DCM57xxx-500 is a flange connected motor including a single-ended 500 line encoder, such as the DCM57202-500 motor, the DCM57205-500 motor and the DCM57207-500 motor.
- DCM50xxxD-1000 is a bolting motor including a differential 1000 line encoder, such as the DCM50202AD-1000 motor, the DCM50205D-1000 motor and the DCM50207D-1000 motor.
- DCM50xxxD-500 is a bolting motor including a differential 500 line encoder, such as the DCM50202AD-500 motor, the DCM50205D-500 motor and the DCM50207D-500 motor.
- DCM57xxxD-1000 is a flange connected motor including a differential 1000 line encoder, such as the DCM57202D-1000 motor, the DCM57205D-1000 motor and the DCM57207D-1000 motor.
- DCM57xxxD-500 is a flange connected motor including a differential 500 line encoder, such as the DCM57202D-500 motor, the DCM57205D-500 motor and the DCM57207D-500 motor.

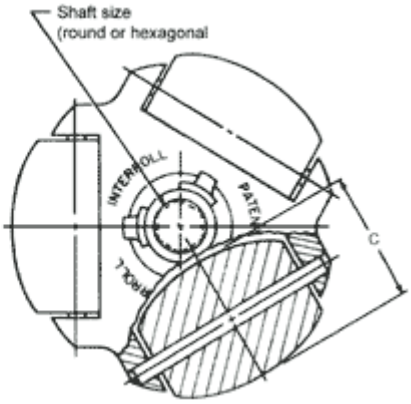
**Note:** No Z signal is offered by the standard models, please specify the requirement when placing an order if the user needs an encoder which has Z signal.

# Transwheels

| | |
|---|---|
|  **2052K** | **Standard** 2" O.D. - 1/2" Bore Double Row w/ Keyway<br>**Metric** 49.2mm O.D. - 12.7mm Bore w/ Keyway<br>Recommended max load \| Steel Bottom = 25 lbs. 11.3kg<br>Plywood Surface = 15 lbs. 6.8kg<br>200# Test Corrugated Bottom = 10 lbs. 4.5kg<br>Weight = 1.75 oz<br><br>**List price - $5.26 Each** \| FXA115 |
| **2052KX** | **Cat-Trak Model \| $7.48 Each** \| FXA315 |

# Lightweight Omni Wheels




**2570**

| | | |
|---|---|---|
| Diameter | 1.9" | 48mm |
| Bore | .315" | 8mm |
| Width | .85" | 21.6mm |
| Barrel length | .94" | 23.8mm |
| Barrel diameter | .69" | 17.5mm |

**2580**

| | | |
|---|---|---|
| Diameter | 3.15" | 80mm |
| Bore | 0.5" | 12.7mm |
| Width | 1.34" | 34mm |
| Barrel length | 1.55" | 39.4mm |
| Barrel diameter | 1.18" | 30mm |

The 2530 series wheels have aluminum bodies and a choice of aluminum barrels, rubber or polyurethane barrels. The 2530 Omniwheels are heavy duty with a load rating of 220 lbs, or 99.79kg per wheel set.

The 2570 and 2580 wheels have plastic bodies and barrels. They are lightweight with a load rating of 25 lbs, or 11.34kg per set for the 2570 and 75 lbs, or 34.02kg per set for the 2580.

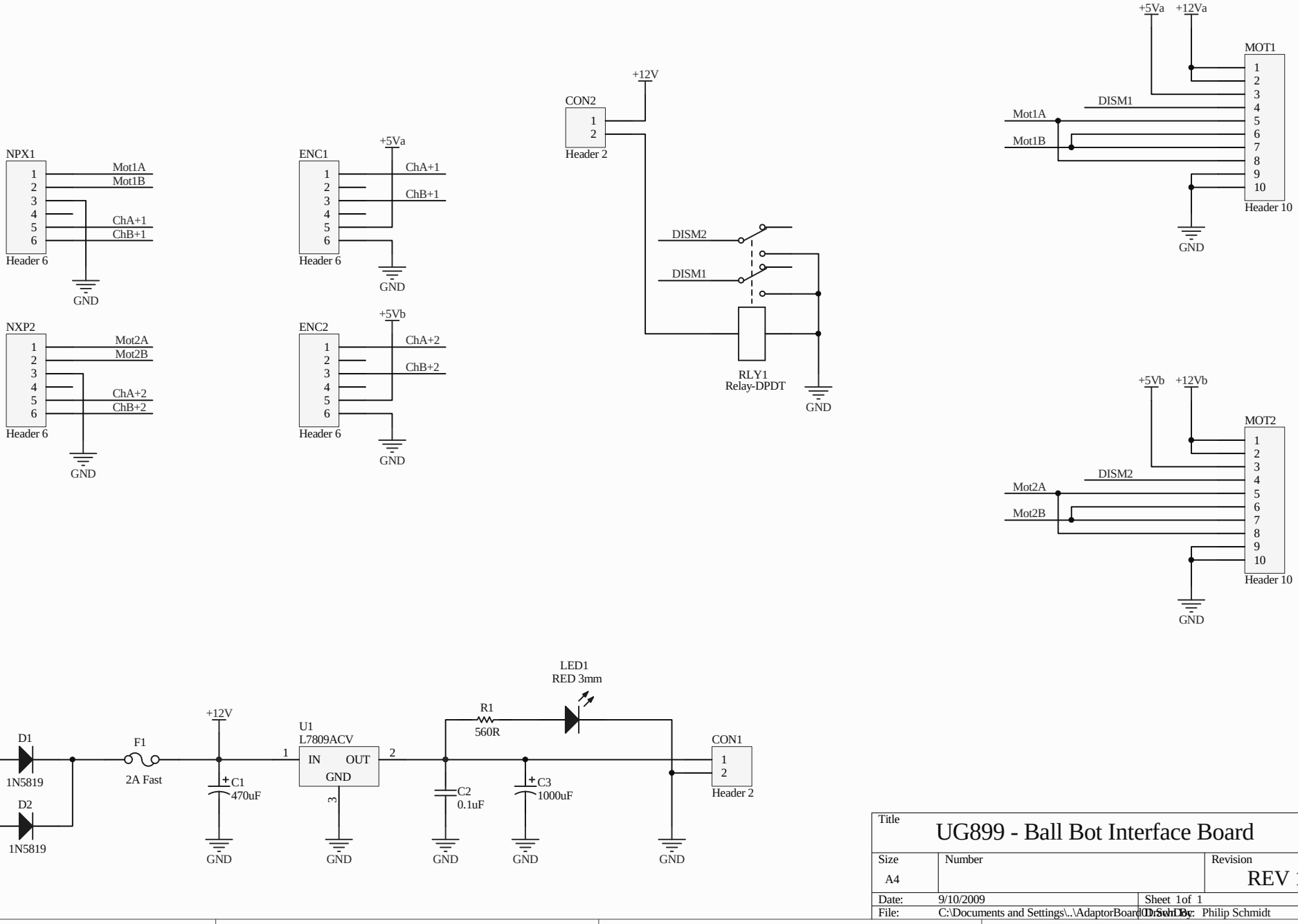
# H Full Scale Ballbot Drawings

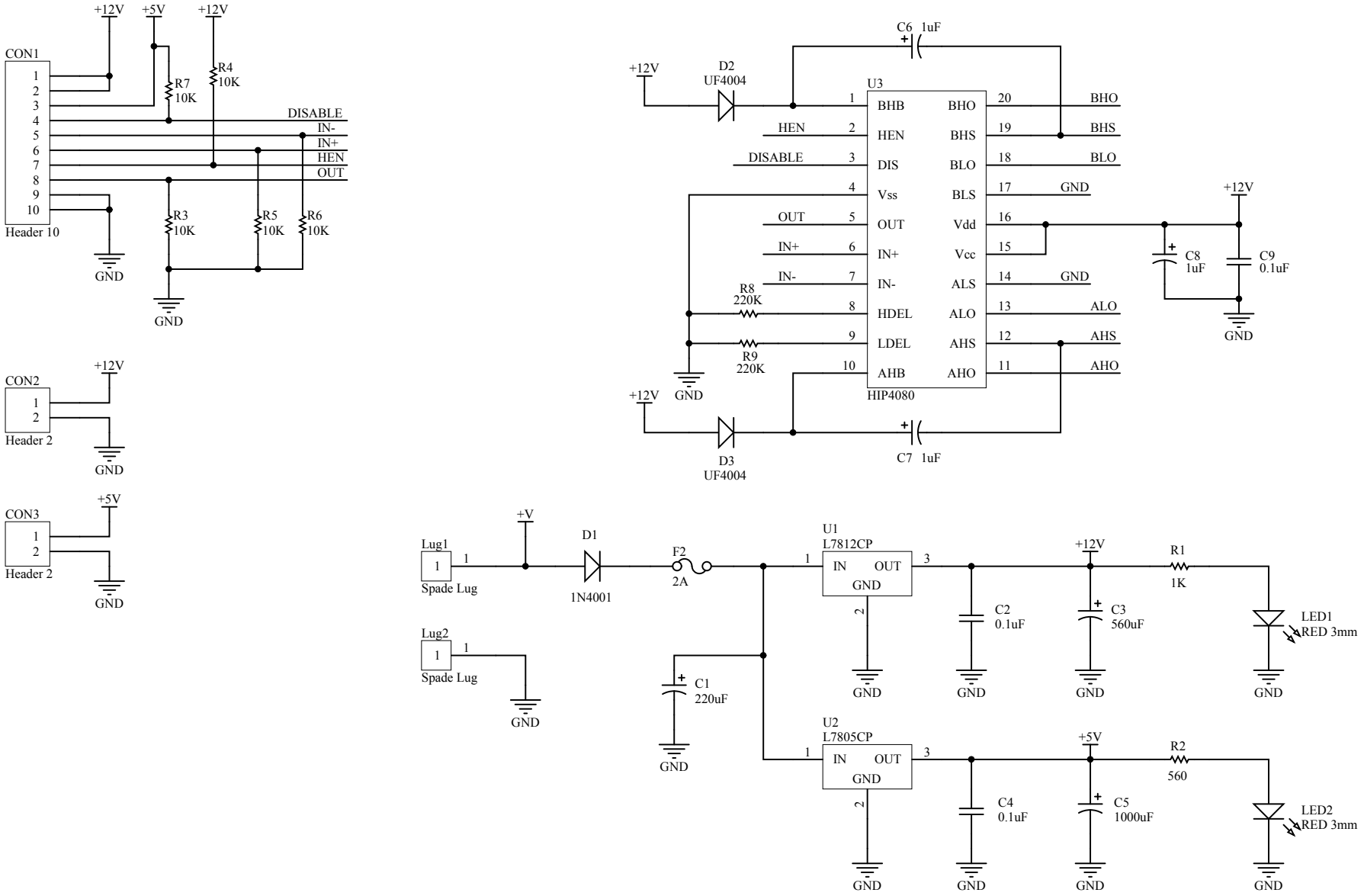
In this appendix are the construction drawings for the Full Scale Ballbot as designed.

# I Electrical Schematics

Electrical schematics, designed by Philip Schmidt at the University of Adelaide Mechanical Engineering Electronics Workshop are included in this appendix. This includes schematics for:

- The Breakout board
- The Motor Controller boards


NPX1
Mot1A
Mot1B
ChA+1
ChB+1
Header 6
GND
NXP2
Mot2A
Mot2B
ChA+2
ChB+2
Header 6
GND
ENC1
+5Va
ChA+1
ChB+1
Header 6
GND
ENC2
+5Vb
ChA+2
ChB+2
Header 6
GND
CON2
+12V
Header 2
DISM2
DISM1
RLY1
Relay-DPDT
GND
MOT1
+5Va
+12Va
DISM1
Mot1A
Mot1B
Header 10
GND
MOT2
+5Vb
+12Vb
DISM2
Mot2A
Mot2B
Header 10
GND
+12Va
+12Vb
D1
1N5819
D2
1N5819
F1
2A Fast
+12V
C1
470uF
U1
L7809ACV
IN
OUT
GND
LED1
RED 3mm
R1
560R
C2
0.1uF
C3
1000uF
CON1
Header 2
Title
UG899 - Ball Bot Interface Board
Size
A4
Number
Revision
REV 1.0
Date:
9/10/2009
Sheet 1of 1
File:
C:\Documents and Settings\..\AdaptorBoard.SchDoc
Drawn By:
Philip Schmidt




| | | |
|---|---|---|
| Title | H Bridge Controller Board | |
| Size: A4 | Number: | Revision: REV 1.0 |
| Date: 28/05/2009 | Sheet 1 of 2 | |
| File: C:\Documents and Settings\...\DiffMotorCon2.SchDoc | Drawn By: Philip Schmidt | |

# J Full Scale Ballbot Code

This appendix includes the code used on the Full Scale Ballbot. The code is written in C for the HCS12 Dragonboard.

```
/********** MAIN.C ***************/
#include "main.h"
#include "main_asm.h" /* interface to the assembly module */
#include "vars.h"

// Timer header files
#include "timer.h"

// PWM header
#include "pwm.h"

// IMU Unit Header
#include "imu_0x30_sci1.h"


/* Define the state variables for the controller */
struct state currState;

long phiXcount;
long phiYcount;

/* Define thing for logging */
#ifdef OP_NORMAL
struct state logStates[MAX_LOG];
#endif

#ifdef OP_CALIBRATION
float thetaXlog[MAX_LOG];
float thetaYlog[MAX_LOG];
#endif

#ifdef OP_ONEPLANE
float thetaXlog[MAX_LOG];
float thetaXdotlog[MAX_LOG];
float phiXlog[MAX_LOG];
float phiXdotlog[MAX_LOG];
#endif

/*Previous values of theta and phi, for derivative calculations */
float phiXprev;
float phiYprev;

/* command values */
float phiXcmd = 0;
float phiYcmd = 0;

/* system time */
long currTime = 0;


/* Flag to update the controller signals */
int runFlag = FALSE;

/* Temp Variables */
int counting;
int count = 0;
int temp = 10;
int logCount = 0;
```

```
int logCount2 = LOG_FREQ;

// ***********************************************************************************************
void main(void) {
  PLL_init();        // set system clock frequency to 24 MHz

  DDRB  = 0xff;       // Port B is output - Lights
//  DDRP  = 0xff;       // Port P is output - PWM Signals
  DDRH  = 0xC0;       // Port H is an input - Encoder Check Sign
  DDRT = 0x00;        // Port T is an input - Encoder Timer Interrupts

  PTJ = 0x00;         // enable LED    ????
  PORTB= 0xFF;
  // -------------------------------------------------------------------------------------
  // Initialisation
  // -------------------------------------------------------------------------------------

  // Initialize LCD (using main.asm)
  lcd_init();
  set_lcd_addr(0x00);

  // Enable all interrupts
  {__asm sei;}

  // Enable keypad
  keypad_enable();

  // Wait for user input
  type_lcd("Press any key to continue");

  while(PTH_PTH0){
  }

  clear_lcd();
  type_lcd("Program started");
  PORTB^=0x80;  // Flash Light
  PORTB= 0x00;


  // Initialise all timers
  timer0_init();  // Interrupt for Logging
  timer2_init();  // Interrupt for Encoder X
  timer3_init();  // Interrupt for Encoder YH
  timer7_init();

  // Initialise PWM signal
  PWM_init();

  // Initialise IMU
  IMU_RX_Init();
  PORTB^=0x20;
  clear_lcd();
  //PORTB^= 0x10;

  ms_delay(1000);
  PORTB= 0xFF;


  // Run forever - program is running as a scheduled task off interrupts
  for(;;){
    if(imuDataReadyFlag){
      if(IMU_TestChecksum_Copy()){
```

```
    }

  }


  if(runFlag){
     float motorXSignal;
     float motorYSignal;


     PTH_PTH6 ^= 1;
     // Generate the motor signals
     motorXSignal = generateSignal(X_MOTOR);
     motorYSignal = generateSignal(Y_MOTOR);

     // Saturate motor signals to +- 24V
     if(motorXSignal>24){
      motorXSignal = 24;
     } else if(motorXSignal < -24){
      motorXSignal = -24;
     }


     if(motorYSignal>24){
      motorYSignal = 24;
     } else if(motorYSignal < -24){
      motorYSignal = -24;
     }

  // Hard coded commands
  //    if(currTime>500 && currTime < 2000){
  //       phiXcmd = (currTime-500)/100*PI;
  //    } else if(currTime > 2000){
  //       phiXcmd =  15*PI; //20*PI - (currTime-2000)/100*2*PI;
  //    } else {
  //       phiXcmd = 0;
  //    }

     // Convert motor signals to -200 to 200 for PWM, and run motors
     runXMotors((motorXSignal*200)/24);
     runYMotors((motorYSignal*200)/24);

     // Log stuff
     if(!logCount2){
       logCount2=LOG_FREQ;
       PTH_PTH6 ^= 1;
       if(logCount < (MAX_LOG)-1){
         #ifdef OP_CALIBRATION
         thetaXlog[logCount]=currState.thetaX;
         thetaYlog[logCount]=currState.thetaY;
         #endif

         #ifdef OP_NORMAL
          logStates[logCount] = currState;
          logStates[logCount].phiXint = motorXSignal;
          logStates[logCount].phiYint = motorYSignal;
         #endif

         #ifdef OP_ONEPLANE
          thetaXlog[logCount]=currState.thetaX;
          thetaXdotlog[logCount]=currState.thetaXdot;
          phiXlog[logCount] = currState.phiX;
          phiXdotlog[logCount] = currState.phiXdot;
```

```
          #endif

          logCount++;

         }else{
           PORTB = 0x00;
           logCount2 = -1;
         }
        }
        logCount2--;


       // Reset the run flag
       runFlag = FALSE;

      }//if
    }   // for
  }  // main



// ------------------------------------------------------------------------------------
// Controller Functions
// ------------------------------------------------------------------------------------
void updateStates(void){
 // Update the states
  currState.phiX = ((float)phiXcount)*PI/250;
  currState.phiY = ((float)phiYcount)*PI/250;

 currState.phiXdot = currState.phiXdot*phiDotFilter + (1-phiDotFilter)*(currState.phiX-phiXprev)/timeStep;
 currState.phiYdot = currState.phiYdot*phiDotFilter + (1-phiDotFilter)*(currState.phiY-phiYprev)/timeStep;

 currState.phiXint = currState.phiXint + currState.phiX*timeStep;
 currState.phiYint = currState.phiYint + currState.phiY*timeStep;

 phiXprev = currState.phiX;
 phiYprev = currState.phiY;

 //May need to low pass filter the states
 // APPLY A LOW PASS FILTER

}


/* Generate the voltage signal required for the motors  */
float generateSignal(int motorSelect){
  if(motorSelect==X_MOTOR){

    return -(currState.thetaX*((float)K_theta) + currState.thetaXdot*((float)K_thetaDot)
     + (currState.phiX-phiXcmd)*((float)K_phi) + currState.phiXdot*((float)K_phiDot)
     + currState.phiXint*((float)K_phiInt));

  } else if(motorSelect == Y_MOTOR){

    return -(currState.thetaY*((float)K_theta) + currState.thetaYdot*((float)K_thetaDot)
     + (currState.phiY-phiYcmd)*((float)K_phi) + currState.phiYdot*((float)K_phiDot)
     + currState.phiYint*((float)K_phiInt));

  } else{

    return 0;

  }
```

```
}


// ------------------------------------------------------------------------------------
// Additional Functions
// ------------------------------------------------------------------------------------
void write_sign_int_lcd(int val16){
  if(val16 < 0){
    data8(0x2d); // - sign
    write_int_lcd(-val16);
  }
  else{
    write_int_lcd(val16);
    data8(0x20);
  }

}

void write_sign_long_lcd(long val32){
  if(val32 < 0){
    data8(0x2d); // - sign
    write_long_lcd(-val32);
  }
  else{
    write_long_lcd(val32);
    data8(0x20);
  }

}

/*********** MAIN.H ***************/
#ifndef _MAIN_H_

#include <hidef.h>      /* common defines and macros */
#include <mc9s12dg256.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dg256b"

#include "frtype.h"


/* Define the Operation Mode */
#define OP_ONEPLANE



#define FIVE_PERCENT 1638
#define X_MOTOR 0
#define Y_MOTOR 1
#define MAX_VOLTAGE 24

// Typedefs

// Misc. functions
void write_sign_int_lcd(int val16);
void write_sign_long_lcd(long val32);
void updateStates(void);
void motorTest(void);
float generateSignal(int);


#ifdef OP_NORMAL
#define MAX_LOG 200
#define LOG_FREQ 20
```

```
#endif
#ifdef OP_CALIBRATION
#define MAX_LOG 1000
#define LOG_FREQ 5
#endif
#ifdef OP_ONEPLANE
#define MAX_LOG 500
#define LOG_FREQ 10
#endif

#define SAJ 500000000000000


/* Offsets for the 'upright' position */
#define THETA_X_OFFSET 0.007871591
#define THETA_Y_OFFSET -0.007917513

/* The controller gains */
#define K_theta 1200
#define K_thetaDot 130
#define K_phi -1
#define K_phiDot -4
#define K_phiInt -0

#define phiDotFilter 0.95
#define DEADZONE_X 10
#define DEADZONE_Y 10

#endif _MAIN_H_


/********** PWM.C ***************/
#include "pwm.h"
#include "main.h"

/* Code to generate PWMs for the motors
 Require 4 PWM signals - two for each motor
 Changing the active PWM will result in a change in direction

 Motor pairs:

 X plane - PWM0 and PWM1 (Pins PP0 and PP1)
 Y plane - PWM4 and PWM5 (Pins PP4 and PP5)

 Using these pin combinations as they can run off the same clock.

 Generate ~200 Hz PWM signal

 Allow for 0-200 signal, requires a clock of 40 000 Hz.

 Use scalars of 2^3 and 38, gives clock frequency of ~39473 Hz
  -> PWM at 197.4 Hz
*/
/**
PWM DETAILS

Registers:

PWME : Enable PWM on the pin
      1 - pwm enabled, begins at next clock pulse
      0 - pwm disabled on pin.

PWMPOL : Determines polarity of PWM pulse
```

```
        1 - pwm is high at beginning of period and goes low when duty count reached
        0 - pwm is low                 ""               high         ""

PWMCLK : Determines whether a channel uses clock A/B or SA/SB
        1 - use scaled clock, SA/SB
        0 - use normal clock, A/B

PWMPRCLK : Prescaler for clocks A and B

        Scales: 1, 2, 4, 8, 16, 32, 64, 128

        Bits 2:0 - determines prescaler B (scale 2^-[2:0])
        Bits 6:4 - determines prescaler A (scale 2^-[6:4])

PWMCAE : PWM output is left aligned or center aligned
      1: Center aligned
      0: Edge aligned


PWMCTL : Concatenates pwm channels for 16-bit PWM, freeze mode, wait mode
      Set to 0x00


PWMSCLA : prescale for clock A

        frequency is reduced by 2*PWMSCLA.
        If PWMSCLA is 0, it is reduced by 512

PWMSCLB : prescale for clock B, same as above

PWMCNTx : counter for each channel
          resets on a write
          if PWME is 0, counter does not count
          counts to PERIOD - 1

PWMPERx : period register

PWMDTYx : duty cycle register

**/
void PWM_init(void){
  /* Select Clock SA for 0,1,4 and 5
     This will allow for slower clock rates (we want a period of approx 500 Hz)
   */
  PWMCLK=0x33;

  /* Select Clock Prescale value
     Clock A runs at Bus Clock/8 = 3MHz
   */
  PWMPRCLK=0x03;

  /* Select Scalar for Clock SA
    Use PWMSCLA = 38
    Gives clock frequency = 3MHz/(2*38) = 39473.7 Hz
   */
  PWMSCLA = 15;

  /* Set the polarity such that the PWM signals go high -> low */
  PWMPOL = 0x33;

  /* Set the PWM to be left aligned (not centre aligned) */
  PWMCAE = 0x00;
```

```
  /* Set alll PWMs to be 8-bit. Is not set to 16-bit (but may be if desired) */
  PWMCTL = 0x00;

  /* Initialise period to be 200 */
  PWMPER0=200;
  PWMPER1=200;
  PWMPER4=200;
  PWMPER5=200;

  // Set 0% duty
  PWMDTY0=0;
  PWMDTY1=0;
  PWMDTY4=0;
  PWMDTY5=0;

  // Disable PWMs to start with
  PWME = 0x33;
}


// Run the X motors, value between -200 and 200
void runXMotors(short value){
  // If running forward, make PP0 high, and leave PP1 = 0
  if(value > 0){
    value = (value+DEADZONE_X*2);

      if(value > 200){
      value = 200;
    }
    PWMDTY0 = value;
    PWMDTY1 = 0;
  } else if(value ==0){
  // If equal to zero, disable both Channels 0 and 1
    PWMDTY0 = 0;
    PWMDTY1 = 0;
  } else{
   value = (value-DEADZONE_X*2);
  // Otherwise set duty cycle for PP1, and leave PP0 = 0
    if(value < -200){
      value = -200;
    }
    PWMDTY0 = 0;
    PWMDTY1 = -value;
  }

  PWME &= 0xFC;
  PWME |= 0x03;
}

// Run the Y motors, value between -200 and 200
void runYMotors(short value){
  // If running forward, make PP0 high, and leave PP1 = 0
  if(value > 0){
  value = (value+DEADZONE_Y*2);
    if(value > 200){
      value = 200;
    }
    PWMDTY4 = value;
    PWMDTY5 = 0;
  } else if(value ==0){
  // If equal to zero, disable both Channels 0 and 1
    PWMDTY4 = 0;
    PWMDTY5 = 0;
```

```
  } else{
  // Otherwise set duty cycle for PP1, and leave PP0 = 0
  value = (value-DEADZONE_Y*2);
    if(value < -200){
      value = -200;
    }
    PWMDTY4 = 0;
    PWMDTY5 = -value;
  }
  PWME &= 0xCF;
  PWME |= 0x30;
}

/*********** PWM.H ***************/
#ifndef _PWM_H_
#define _PWM_H_

extern void PWM_init(void);

extern void runXMotors(short);

extern void runYMotors(short);

#endif _PWM_H_

/*********** TIMER.C ***************/

//#include "main_asm.h"
#include "timer.h"
#include "vars.h"
#include "pwm.h"
#include "imu_0x30_sci1.h"
#include "main.h"
#include "sci0_logger.h"

#define RUN_COUNT_RESET 24000000*timeStep/(timerValue);
int runCount;

#define RDRF 0x20         // Receive Data Register Full Bit
#define TDRE 0x80         // Transmit Data Register Empty Bit

extern long currTime;

// Timer Interrupt for logging - initialisation
void timer0_init(void){
  asm sei;        // Disable Interrupts

  TIOS &= 0xFE;   // Set Timer 0 to input compare mode
  TCTL2 &= 0xFC;  // Disconnect output pin logic (not really relevant due to input compare mode)

  TCTL4 &= 0xFC;  // Set input compare trigger to be a rising edge
  TCTL4 |= 0x01;

  // TTOV - don't care as we are in input compare mode
  TIE |= 0x01;    // Enable interrupt on channel 0

  TSCR1 = 0x80;   // Enable timer
  TFLG1 = 0x01;   // Clear Flag
  asm cli         // Enable all interrupts

}
```

```
// The interrupt vector
__interrupt 8 void timer0_isr(void) {
  int i = 0;
  runXMotors(0);
  runYMotors(0);
   SCI1CR1 = 0;

  // Send the continuous mode command to the IMU
  SCI1CR2 = 0x08;       // Transmit enable
  while( !(SCI1SR1 & TDRE) ){}
  SCI1DRL = 0x10;
  while( !(SCI1SR1 & TDRE) ){}
  SCI1DRL = 0x00;
  while( !(SCI1SR1 & TDRE) ){}
  SCI1DRL = 0x00;
  while( !(SCI1SR1 & TDRE) ){}

  ms_delay(500);
  SCI0_sendData();

  while(TRUE){
  }
  TFLG1 |= 0x04;
}

/* Timers 2 and 3 are used in Input Capture mode for the encoders
   Timer 7 is used as the scheduled task
 */

/* TIMER 2 - MOTOR ENCODER X
   Connect one pin to PT2, and check PH2 for direction.
 */
void timer2_init(void){
  asm sei;        // Disable Interrupts

  TIOS &= 0xFB;   // Set Timer 0 to input compare mode
  TCTL2 &= 0xCF;  // Disconnect output pin logic (not really relevant due to input compare mode)

  TCTL4 &= 0xCF;  // Set input compare trigger to be a rising edge
  TCTL4 |= 0x10;

  // TTOV - don't care as we are in input compare mode
  TIE |= 0x04;    // Enable interrupt on channel 0

  TSCR1 = 0x80;   // Enable timer
  TFLG1 = 0x04;   // Clear Flag
  asm cli         // Enable all interrupts

}

/* ISR - checks PH2 for direction and increments/decrements phiX as neccessary
 */
__interrupt 10 void timer2_isr(void) {

  //type_lcd("Interrupt 2 has occurred");
  // Needs to check the status of the other encoder pin, and increment or decrement as neccessary
  // Acknowledge the interrupt
  if(PTH_PTH2){
    phiXcount++;
  } else{
    phiXcount--;
  }
```

```
  TFLG1 |= 0x04;
}


/* TIMER 3 - MOTOR ENCODER Y
   Connect one pin to PT3, and check PH3 for direction.
 */
void timer3_init(void){
  asm sei;        // Disable Interrupts

  TIOS &= 0xF7;   // Set Timer 3 to input compare mode
  TCTL2 &= 0x3F;  // Disconnect output pin logic (not really relevant due to input compare mode)

  TCTL4 &= 0x3F;  // Set input compare trigger to be a rising edge
  TCTL4 |= 0x40;

  // TTOV - don't care as we are in input compare mode
  TIE |= 0x08;    // Enable interrupt on channel 0

  TSCR1 = 0x80;   // Enable timer
  TFLG1 = 0x08;   // Clear interrupt flag

  asm cli         // Enable all interrupts
}

/* ISR - checks PH2 for direction and increments/decrements phiY as neccessary
 */
__interrupt 11 void timer3_isr(void) {

  //type_lcd("Interrupt 3 has occurred");
  // Needs to check the status of the other encoder pin, and increment or decrement as neccessary
  // Acknowledge the interrupt
  if(PTH_PTH3){
    phiYcount++;
  } else{
    phiYcount--;
  }

  TFLG1 |= 0x08;
}


/* TIMER 7 - Scheduled Task
   Does the bulk of the program's work...

  */

  void timer7_init(void){
  asm sei;        // Disable Interrupts

  TIOS |= 0x80;   // Set Timer 7 to output compare mode
  TCTL2 &= 0x3F;  // Disconnect output pin logic (not really relevant due to input compare mode)
  TCTL1 &= 0x3F;      // 00xx.xxxx -> OM7 = OM7 = 0 (disconnect pin 'IOC7' from o/p logic)
  OC7M = 0x00;   // Do not change link any output pins

  // TTOV - overflow bit - don't care about it

  TIE |= 0x80;    // Enable interrupt on channel 7

  TSCR2 = 0x08;   // Set the divisor to be 4 for period

  TC7 = timerValue;
```

```
  TSCR1 |= 0x80;   // Enable timer
  TFLG1 |= 0x80;       // initially clear C7F (event flag for channel 7)

  runCount = RUN_COUNT_RESET;

  asm cli         // Enable all interrupts
}


/* Timer 7 ISR:
   - Updates Derivative and Integral States
   - Calculates gains
   - Changes Motor PWM signal
 */
__interrupt 15 void timer7_isr(void) {
  IMU_RX_Tick();
  if(!runCount--){
     currTime++;
     PTH_PTH7^=1;
     // Update the states
     updateStates();
     runFlag = TRUE;
     runCount = RUN_COUNT_RESET;
     PTH_PTH7^=1;
  }
  TFLG1 |= 0x80;
}


/*********** TIMER.H ***************/
#ifndef _TIMER_H_
#define _TIMER_H_


extern void timer0_init(void);

extern __interrupt 8 void timer0_isr(void);

extern void timer2_init(void);

extern __interrupt 10 void timer2_isr(void);

extern void timer3_init(void);

extern __interrupt 11 void timer3_isr(void);

extern void timer7_init(void);

extern __interrupt 15 void timer7_isr(void);

// Define the maximum timer value as 60000. With clock divisor of 4 gives period of 10ms
#define timerValue 11500
#define timeStep 0.01

#endif

/*********** VARS.H ***************/
#ifndef _CONTROLLER_VARS_
#define _CONTROLLER_VARS_

struct state {
  float thetaX;
```

```
  float thetaXdot;
  float phiX;
  float phiXdot;
  float phiXint;

  float thetaY;
  float thetaYdot;
  float phiY;
  float phiYdot;
  float phiYint;
};

#define PI 3.141592654

extern struct state currState;
extern struct state logStates[];
extern float thetaXlog[];
extern float thetaYlog[];
extern float thetaXdotlog[];
extern float phiXlog[];
extern float phiXdotlog[];

extern long phiXcount;
extern long phiYcount;

extern int runFlag;

#endif _CONTROLLER_VARS_

/*********** IMU_0X30_SCI1.C ***************/
#include "imu_0x30_sci1.h"
#include "main_asm.h"
#include "vars.h"
#include "main.h"

#define RDRF 0x20         // Receive Data Register Full Bit
#define TDRE 0x80         // Transmit Data Register Empty Bit
#define TIMEOUT 2 // Number of ticks before RX timeout
#define RXLENGTH 23 // Number of bytes to receive
#define IMUBAUD_115200
#define SCALE_FACTOR_THETA PI/32768
#define SCALE_FACTOR_THETADOT 8500/32768000

unsigned char IMUBuffer[ RXLENGTH ];
unsigned char RXBuffer[ RXLENGTH ];
//unsigned char PVTOLBuffer[ PvtolBufferLength ];
unsigned char imuIndex;
unsigned char imuDataReadyFlag;
unsigned char rxTimeout;
extern int counting;

// Requires busclock == 24Mhz (i.e. PLL enabled)
void IMU_RX_Init( void ) {
  char temp;
  asm sei;

#ifdef IMUBAUD_19200
    SCI1BDH=0;
    SCI1BDL=78;
#endif
#ifdef IMUBAUD_38400
    SCI1BDH=0;
    SCI1BDL=39;
```

```
#endif
#ifdef IMUBAUD_115200
    SCI1BDH=0;
    SCI1BDL=13;
#endif

  SCI1CR1 = 0x0;
  temp = SCI1SR1;
  temp = SCI1DRL;
  temp = SCI1SR2;
  temp = SCI1DRH;

  // Send the continuous mode command to the IMU
  SCI1CR2 = 0x08;       // Transmit enable

  while( !(SCI1SR1 & TDRE) ){}
  SCI1DRL = 0x10;
  while( !(SCI1SR1 & TDRE) ){}
  SCI1DRL = 0x00;
  while( !(SCI1SR1 & TDRE) ){}
  SCI1DRL = 0x31;
  while( !(SCI1SR1 & TDRE) ){}

  SCI1CR1 = 0;
  SCI1CR2 = 0x24;       // Receive Interrupts, Receive enable
  //imuIndex = 0;
  //rxTimeout = TIMEOUT;
  //imuDataReadyFlag = 0;
  PORTB^=0x10;
  asm cli
}

unsigned char IMU_TestChecksum_Copy( void ){
  unsigned char i;
  unsigned int checksum;
  unsigned int computedChecksum;

  imuDataReadyFlag = 0x00;

  checksum = 256*IMUBuffer[ RXLENGTH-2 ] + IMUBuffer[ RXLENGTH-1 ];
  computedChecksum = IMUBuffer[0];
  for(i=0;i<(RXLENGTH-3)/2;i++) computedChecksum += 256*IMUBuffer[ 2*i+1 ] + IMUBuffer[ 2*i+2 ];
  if( computedChecksum == checksum ){
    // Save stuff
    currState.thetaX = ((float) (256*IMUBuffer[1]+IMUBuffer[2]))*((float)SCALE_FACTOR_THETA) - THETA_X_OFFSET;
    currState.thetaY = -((float) (256*IMUBuffer[3]+IMUBuffer[4]))*((float) SCALE_FACTOR_THETA) - THETA_Y_OFFSET;
    currState.thetaXdot = ((float) (256*IMUBuffer[13]+IMUBuffer[14]))*((float)SCALE_FACTOR_THETADOT);
    currState.thetaYdot = -((float) (256*IMUBuffer[15]+IMUBuffer[16]))*((float)SCALE_FACTOR_THETADOT);
    return 1;
  }
  return 0;
}


void IMU_RX_Tick( void ){
  if(!--rxTimeout) imuIndex = 0;
}

__interrupt 0x15 void IMU_RX_ISR( void ){
  char temp;
  char i = 0;
  PORTB^=0x01;
  //IMUBuffer[counting] = temp;
```

```
  counting++;


 // PORTBˆ=0x01;
//  type_lcd("sdf");
  temp = SCI1SR1;       // Read SR1 (required to clear the flag)
  temp = SCI1DRL;       // read the data (flag should be cleared now)
  rxTimeout = TIMEOUT;
  if( !imuIndex ){     // If in waiting mode
    if( temp == 0x31 ){ //  and we receive the header byte
      RXBuffer[0] = temp;   //  start filling the buffer
      imuIndex = 1;

    }                       // otherwise discard the byte
  }else{
                            // If receiving, place the byte in the buffer
    RXBuffer[ imuIndex++ ] = temp;
                            // If full, copy to another buffer and raise flag.
    if( imuIndex == RXLENGTH ){
      for(temp=0;temp<RXLENGTH;temp++){
        IMUBuffer[ temp ] = RXBuffer[ temp ];
      }
      imuDataReadyFlag = 0xFF;
      imuIndex = 0;
    }
  }
//  PTH &= ˜0x02;
}


/*********** IMU_0X30_SCI1.H ***************/
#ifndef _IMU_RX_
#define _IMU_RX_

extern unsigned char imuDataReadyFlag;

void IMU_RX_Init( void );
unsigned char IMU_TestChecksum_Copy( void );
void IMU_RX_Tick( void );
__interrupt 0x15 void IMU_RX_ISR( void );

#endif _IMU_RX_


/*********** SCI0_LOGGER.C ***************/
#include "main.h"
#include "sci0_logger.h"
#include "vars.h"

#define RDRF 0x20         // Receive Data Register Full Bit
#define TDRE 0x80         // Transmit Data Register Empty Bit
 //-------------------------SCI0_OutChar------------------------
// Wait for buffer to be empty, output 8-bit to serial port
// busy-waiting synchronization
// Input: 8-bit data to be transferred
// Output: none
void SCI0_OutChar(char data) {
  while((SCI0SR1 & TDRE) == 0){}
  SCI0DRL = data;

}

//----------------------SCI0_OutUDec-----------------------
```

```
// Output a 16-bit number in unsigned decimal format
// Input: 16-bit number to be transferred
// Output: none
// Variable format 1-5 digits with no space before or after
void SCI0_OutUDec(unsigned short n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n >= 10){
    SCI0_OutUDec(n/10);
    n = n%10;
  }
  SCI0_OutChar(n+'0'); /* n is between 0 and 9 */
}

//----------------------SCI0_OutUDec----------------------
// Output a 16-bit number in unsigned decimal format
// Input: 16-bit number to be transferred
// Output: none
// Variable format 1-5 digits with no space before or after
void SCI0_OutFloat(float n){
unsigned short o;
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n < 0){
    SCI0_OutChar('-');
    n = -n;
  }
  o = ((unsigned short)100*n);

  SCI0_OutUDec(o);
}

 //-------------------------SCI0_OutString-------------------------
// Output String (NULL termination), busy-waiting synchronization
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void SCI0_OutString(char *pt) {

  while(*pt) {

    SCI0_OutChar(*pt);
    pt++;

  }
}



void SCI0_sendData(void){
  // Set baud rate to 115200
  int i = 0;
  SCI0BDH=0;
  SCI0BDL=13;

  SCI0CR1 = 0;
  SCI0CR2 = 0x08;       // Transmit enable

  // Header
  #ifdef OP_CALIBRATION
  SCI0_OutString("thetaX, thetaY,");
  #endif

  #ifdef OP_NORMAL
```

```
SCI0_OutString("thetaX, thetaXdot, phiX, phiXdot, MotorSignalX, thetaY, thetaYdot, phiY, phiYdot, MotorSignalY
SCI0_OutFloat(K_theta);
SCI0_OutChar(',');
    SCI0_OutFloat(K_thetaDot);
SCI0_OutChar(',');
    SCI0_OutFloat(K_phi);
SCI0_OutChar(',');
    SCI0_OutFloat(K_phiDot);
SCI0_OutChar(',');
SCI0_OutFloat(K_phiInt);
SCI0_OutChar(',');

    SCI0_OutFloat(DEADZONE_X*24/100);
SCI0_OutChar(',');

SCI0_OutFloat(K_theta);
SCI0_OutChar(',');
    SCI0_OutFloat(K_thetaDot);
SCI0_OutChar(',');
    SCI0_OutFloat(K_phi);
SCI0_OutChar(',');
    SCI0_OutFloat(K_phiDot);
SCI0_OutChar(',');
SCI0_OutFloat(K_phiInt);
SCI0_OutChar(',');

SCI0_OutFloat(DEADZONE_Y*24/100);
SCI0_OutChar(',');
#endif

#ifdef OP_ONEPLANE
SCI0_OutString("thetaX, thetaXdot, phiX, phiXdot");
#endif

SCI0_OutString("\n\r");

// Now Print the data
for(i = 0; i < MAX_LOG; i++){
  #ifdef OP_CALIBRATION
    SCI0_OutFloat(thetaXlog[i]*1000);
    SCI0_OutChar(',');
    SCI0_OutFloat(thetaYlog[i]*1000);
    SCI0_OutChar(',');
  #endif
  #ifdef OP_NORMAL
    SCI0_OutFloat(logStates[i].thetaX*100);
    SCI0_OutChar(',');

    SCI0_OutFloat(logStates[i].thetaXdot*100);
    SCI0_OutChar(',');

    SCI0_OutFloat(logStates[i].phiX);
    SCI0_OutChar(',');

    SCI0_OutFloat(logStates[i].phiXdot);
    SCI0_OutChar(',');

    SCI0_OutFloat(logStates[i].phiXint);
    SCI0_OutChar(',');

    SCI0_OutFloat(logStates[i].thetaY*100);
    SCI0_OutChar(',');
```

```
      SCI0_OutFloat(logStates[i].thetaYdot*100);
      SCI0_OutChar(',');

      SCI0_OutFloat(logStates[i].phiY);
      SCI0_OutChar(',');

      SCI0_OutFloat(logStates[i].phiYdot);
      SCI0_OutChar(',');

      SCI0_OutFloat(logStates[i].phiYint);
      SCI0_OutChar(',');
     #endif

     #ifdef OP_ONEPLANE
      SCI0_OutFloat(thetaXlog[i]*100);
      SCI0_OutChar(',');
      SCI0_OutFloat(thetaXdotlog[i]*100);
      SCI0_OutChar(',');
      SCI0_OutFloat(phiXlog[i]);
      SCI0_OutChar(',');
      SCI0_OutFloat(phiXdotlog[i]);
      SCI0_OutChar(',');
    #endif

    SCI0_OutString("\n\r");
  }
}

/********** SCI0_LOGGER.H ***************/
#ifndef _LOGGER_
#define _LOGGER_

#define CALIBRATION 0
#define NORMAL 1

void SCI0_sendData(void);

#endif _LOGGER_
```

# K Risk Assessment

This appendix includes the Risk Assessment completed for the Full Scale Ballbot.

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

**This form is to be completed in conjunction with the Plant Hazard Identification Form and in accordance with the Hazard Management Policy/Procedure and Plant and Equipment Safety Management Policy/Procedure.**

| **Overall risk rating (existing controls) highest score** eg L, M, VH, H | M |
|---|---|

**STEP 1: ENTER INFORMATION ABOUT THE ITEM OF PLANT/EQUIPMENT, IT'S LOCATION AND THE PEOPLE COMPLETING THE RISK ASSESSMENT**

| Campus | School/Branch | Building | Room | Date assessed: |
|---|---|---|---|---|
| NORTH TCG | MECHANICAL ENG. | ENGINEERING SOUTH | VIBES/ACOUSTICS LAB | 30/7/09 |
| **Plant (Include name and model)** [i] | BALLBOT | | | **Review date:** |
| **Purpose of Plant** | BALANCE | | | |

**AFFIX PHOTO HERE**

**REGISTRATION/LICENCES/ COMPETENCIES**
(Refer Appendix B)

**Registration required?**
- ☐ Yes
- ☒ No

**Licence/ Trade certificate required?**
- ☐ Yes
- ☒ No

**Supervisor assessment required? (Induction required for this plant)**
- ☒ Yes
- ☐ No

**Other competency required?**
- ☒ Yes
- ☐ No

Include in Safe Operating Procedure

**LEGISLATION**

**OHS Legislation and/or Australian Standard to be used for further reference.**

OHS Regulations (1995) Divisions:
- 1.2 OHS Responsibilities
- 1.3 Information, Instruction, Training, Induction, Supervision
- 2.10 Noise
- Part 3 Plant (including)
  - 3.2.25 Plant with moving parts
  - 3.2.15 Hazard Management
  - 3.2.17 Control of risk

AS 4024 - 2006 Safety of machinery

AS 1788 Abrasive wheels

AS 60204.1 electrical equipment for industrial machines

**RISK ASSESSMENT TEAM**

Operator: JUSTIN FONG, SIMON UPPILL

HSO/~~Manager~~: IAN MCNAIR

HSR: VICKY SAMRA

Other: (SUPERVISOR): BEN CAZZOLATO

Other:

**Risk assessment rating and action required endorsed by Head of School/Branch (see also Step 3)**

/ /

(Signature) (Name)



**Project No:** **Issue No:**

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

| LEGEND | | |
|---|---|---|
| ☐ El = Elimination | ☐ Is = Isolation | ☐ CAR = Corrective Action Register |
| ☐ Su = Substitution | ☐ Ad = Administration | |
| ☐ En = Engineering | ☐ PPE = Personal Protective Equipment | |

| Hazard Identified | Likelihood | Consequence | Score | Comments (i.e. when and where hazard is present, task/activity) | Hierarchy of Control | Current Controls | Action Required? Y / N | Tfr to CAR Y / N |
|---|---|---|---|---|---|---|---|---|
| **Can the following items become entangled (eg in moving parts)?** | | | | | | | | |
| ☒ Yes (e.g. Hair, Jewellery, Clothing, Cleaning aids (cloth), Gloves or Other_____<br>☐ No<br>☐ NA | Unlikely | First Aid | Low | Motors and rotating parts are at base of robot and well protected. Extremely unlikely to be an issue during normal use | ☐ El<br>☐ Su<br>☐ En<br>☒ Is<br>☒ Ad<br>☐ PPE | Motors and rotating parts well sheilded with little to no access during operation | N | |
| **Are Emergency Stop buttons adequate?** | | | | | | | | |
| ☒ Yes (within easy reach and clearly marked)<br>☐ No<br>☐ Other issues_____<br>☐ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | Emergency stop button on handheld remote. | | |



**Project No:** **Issue No:**

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

| Hazard Identified | Likelihood | Consequence | Score | Comments (i.e. when and where hazard is present, task/activity) | Hierarchy of Control | Current Controls | Action Required? Y / N | Tfr to CAR Y / N |
|---|---|---|---|---|---|---|---|---|
| **Can anyone be crushed by:** | | | | | | | | |
| ☒ Plant falling or unexpected movement of the plant eg<br>☒ Tipping<br>☐ Falling<br>☐ Rolling over<br>☐ Rolling forward<br>☐ The plant's load<br>☐ Under/between plant and a structure eg wall<br>☒ Inability to apply brake<br>☐ Falling off the plant<br>☐ Part of the plant collapsing/changing shape<br>☐ Other issues _____<br>☐ No<br>☐ NA | Unlikely<br><br>Unlikely | Major<br><br>– | Medium<br><br>– | If controller goes unstable, Ballbot will fall over<br><br>Will result in above consequence | ☐ El<br>☐ Su<br>☒ En<br>☒ Is<br>☒ Ad<br>☐ PPE | Will be tethered from gantry to arrest any risk of tipping.<br>All personnel and others to maintain a safe distance from rig during operation | N | N |
| **Can anyone be cut, stabbed or punctured, amputated by coming into contact with:** | | | | | | | | |
| ☐ Moving plant or parts<br>☐ Sharp or flying objects<br>☐ Work pieces ejected<br>☐ Work pieces disintegrating<br>☐ Other issues_____<br>☒ No<br>☐ NA | | | | No exposed points on Ballbot | ☐ El<br>☐ Su<br>☒ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | N/A | N/A |



**Project No:** **Issue No:**

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

| Hazard Identified | Likelihood | Consequence | Score | Comments (i.e. when and where hazard is present, task/activity) | Hierarchy of Control | Current Controls | Action Required? Y / N | Tfr to CAR Y / N |
|---|---|---|---|---|---|---|---|---|
| **Can anyone be injured from an electrical shock?** | | | | | | | | |
| ☐ Water near equipment<br>☐ Plant located near or in contact with exposed live electrical conductors<br>☐ Leads/switch in poor condition<br>☐ Overhead and underground wires<br>☐ Other issues_____<br>☒ No<br>☐ NA | | | | No external electrical source used. | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |
| **Can anyone be injured by an explosion?** | | | | | | | | |
| ☐ Gas<br>☐ Vapour<br>☐ Dust<br>☐ Liquid<br>☐ Other issues _____<br>☐ No<br>☒ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |
| **Can anyone be burnt due to Friction?** | | | | | | | | |
| ☒ Contact with moving parts or surface of the plant<br>☐ Material handled by the plant<br>☐ Other issues _____<br>☐ No<br>☐ NA | Rare | First Aid | Low | Moving parts only at base of robot. | ☐ El<br>☐ Su<br>☒ En<br>☐ Is<br>☒ Ad<br>☐ PPE | Rotating elements well shielded | N | N |



**Project No:** **Issue No:**

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

| Hazard Identified | Likelihood | Consequence | Score | Comments (i.e. when and where hazard is present, task/activity) | Hierarchy of Control | Current Controls | Action Required? Y / N | Tfr to CAR Y / N |
|---|---|---|---|---|---|---|---|---|
| **Can anyone be struck by moving objects due to:** | | | | | | | | |
| ☐ Plant/materials being ejected<br>☒ Plant/material movement<br>☐ Other issues _____<br>☐ No<br>☐ NA | Possible | Minor | Low | Robot may lose direction | ☐ El<br>☐ Su<br>☐ En<br>☒ Is<br>☐ Ad<br>☐ PPE | Robot will be tethered<br>All personnel and others to maintain a safe distance from rig during operation | N | N |
| **Can anyone suffocate due to:** | | | | | | | | |
| ☐ Lack of oxygen<br>☐ Atmospheric contamination<br>☐ Engulfment<br>☐ Other issues _____<br>☐ No<br>☒ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |
| **Can anyone be burnt due to:** | | | | | | | | |
| ☐ High/low temperature<br>☐ Naked flame<br>☐ Steam<br>☐ Laser beams<br>☐ Other issues _____<br>☐ No<br>☒ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

| Hazard Identified | Likelihood | Consequence | Score | Comments (i.e. when and where hazard is present, task/activity) | Hierarchy of Control | Current Controls | Action Required? Y / N | Tfr to CAR Y / N |
|---|---|---|---|---|---|---|---|---|
| **Can anyone be affected by temperature extremes?** | | | | | | | | |
| ☐ Exposure to high temperature<br>☐ Exposure to low temperature<br>☐ Other issues _____<br>☐ No<br>☒ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |
| **Can anyone slip, trip or fall due to:** | | | | | | | | |
| ☐ The location of the plant<br>☐ Uneven work surfaces<br>☐ Lack of safe guards (eg rails)<br>☐ Slippery work surfaces<br>☐ Other issues _____<br>☐ No<br>☒ NA | | | | No oils used<br>Tests conducted on flat, even surface | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |
| **Can anyone come into contact with Fluids or Gases under high pressure due to:** | | | | | | | | |
| ☐ Failure of the plant<br>☐ Nature of the plant<br>☐ Other issues _____<br>☐ No<br>☒ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |



**Project No:** **Issue No:**

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

| Hazard Identified | Likelihood | Consequence | Score | Comments (i.e. when and where hazard is present, task/activity) | Hierarchy of Control | Current Controls | Action Required? Y / N | Tfr to CAR Y / N |
|---|---|---|---|---|---|---|---|---|
| **Can anyone injured due to Ergonomic issues due to:** | | | | | | | | |
| ☐ Repetitive body movement or posture<br>☐ Insufficient space<br>☒ Excessive effort (push/pull)<br>☐ Working at a height<br>☐ Seating design<br>☐ Poor lighting<br>☐ Other issues _____<br>☐ No<br>☐ NA | Rare | First Aid | Low | May occur when lifting or carrying Ballbot (weight is approx. 30 kg) | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☒ Ad<br>☐ PPE | Ensure two people lift/carry simultaneously.<br>Use of trolley or similar device for longer transportation distances | N | N |
| **Can anyone be injured or suffer ill health from exposure to other hazards?** | | | | | | | | |
| ☐ Chemicals<br>☐ Radiation<br>☐ Fumes<br>☐ Dusts<br>☐ Vibration<br>☐ Noise<br>☐ Toxic gases or vapours<br>☐ Other issues _____<br>☒ No<br>☐ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

| Hazard Identified | Likelihood | Consequence | Score | Comments (i.e. when and where hazard is present, task/activity) | Hierarchy of Control | Current Controls | Action Required? Y / N | Tfr to CAR Y / N |
|---|---|---|---|---|---|---|---|---|
| **Does the plant generate significant environmental hazards** | | | | | | | | |
| ☐ Energy consumption<br>☐ Water consumption<br>☐ Hazardous waste<br>☐ Hazardous emissions<br>☐ Nuisance noise<br>☐ Produce ignition to the surrounding area<br>☐ Other issues _____<br>☒ No<br>☐ NA | | | | | ☐ El<br>☐ Su<br>☐ En<br>☐ Is<br>☐ Ad<br>☐ PPE | | | |

**STEP 3: ACTION REQUIRED BY MANAGER/SUPERVISOR/AUTHORISED PERSON (Note: Sign off required by Head of School/Branch on Page 1)**

☐ All action items have been transferred to the Corrective Actions Register (CAR)

☒ If no actions required and residual risk is medium to very high, the activity and the hazard(s) have been transferred to the Hazard Register and communicated to the relevant personnel.

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# PROJECT RISK ASSESSMENT:

## RISK ASSESSMENT TABLES

**Likelihood Table**

| CATEGORY | DESCRIPTION |
|---|---|
| Almost Certain | Incident will occur at some time (0 – 1 month) |
| Likely | Incident could occur at some time (1 month – 1 year) |
| Possible | Incident is possible to occur (1 year – 2 years) |
| Unlikely | Incident is possible, but unlikely to occur (2 years – 5 years) |
| Rare | Cannot imagine that this could occur (over 5 years) |

**Consequences Table**

| CATEGORY | DESCRIPTION |
|---|---|
| Minor | Effects unlikely to last until the next day. |
| First Aid | Likely to affect employee the next day. |
| Major | Medical Treatment injury needs formal medical treatment |
| Critical | Injury requiring extensive medical treatment and/or hospitalization |
| Catastrophic | Injury resulting in death or permanent incapacity |

**Risk Score Calculator**

| Likelihood | Consequences | | | | |
|---|---|---|---|---|---|
| | Minor | First Aid | Major | Critical | Catastrophic |
| Almost certain | Medium | High | High | Very High | Very High |
| Likely | Medium | Medium | High | High | Very High |
| Possible | Low | Medium | High | High | High |
| Unlikely | Low | Low | Medium | Medium | High |
| Rare | Low | Low | Medium | Medium | High |

**Risk Priority Table**

| Descriptor | Priority | Action |
|---|---|---|
| Very High | 1 | Immediate action required.<br>The activity should cease immediately and short-term safety controls implemented.<br>Notify Manager and assess activity. |
| High | 2 | Implement short-term safety controls immediately.<br>Notify Manager and assess activity |
| Medium | 3 | Short-term safety controls implemented to minimise risk of injury.<br>Notify Manager and assess activity.<br>Corrective Actions within one month. |
| Low | 4 | Notify Manager and assess activity.<br>Corrective Actions within three months (if possible). |

# L Safe Operating Procedure

This appendix includes the Safe Operating Procedure for the Full Scale Ballbot.

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# SAFE OPERATING PROCEDURE: BALLBOT

| LOCATION DETAILS | |
|---|---|
| **School/Branch: Mechanical Engineering** | |
| **TASK/ACTIVITY** | |
| **Ballbot Operation** | **Date: 31/7/09** |

**PREPARED BY** Name, Position and Signature (insert names of the supervisor, HSR, HSO and operator involved)

| Name | Position | Signature |
|---|---|---|
| Justin Fong | Undergraduate Student | |

| HAZARD IDENTIFICATION: | RISK ASSESSMENT |
|---|---|
| **See Risk Assessment dated / /** | **Medium** |

## SAFE OPERATING PROCEDURE DETAILS

# STOP

**DO NOT OPERATE PLANT IF YOU HAVE NOT COMPLETED (1) THE COMPULSORY UNIVERSITY OF ADELAIDE OCCUPATIONAL HEALTH AND SAFETY INDUCTION COURSE, AND; (2) DO NOT POSSESS THE REQUISITE QUALIFICATIONS OR TRAINING FOR THIS PIECE OF PLANT.**

### Preparation – work area check:

- ❑ Ready access to and egress from the machinery (min of 600mm clearance required)
- ❑ Area is free from grease, oil, debris and objects, which can be tripped over.
  (Use diatomaceous earth ("kitty litter") or absorption pillow to soak up grease, coolant, oil and other fluids)
- ❑ Area is clear of unauthorised people before commencing work.

### Personal Attire & Safety Equipment:

- ❑ Approved closed toe type shoes must be worn at all times.
- ❑ Approved safety spectacles/goggles must be worn at all times, where required.
- ❑ Clothing must be tight fitting.
- ❑ Long hair must be confined close to the head by an appropriate restraint.
- ❑ Finger rings and exposed loose jewellery (eg bracelets and necklaces) must not be worn. Medic Alert bracelet must be taped if exposed.
- ❑ Gloves must not be worn when operating machine, excepting where specifying otherwise.

### Operation Of Emergency Stop Button:

- ❑ Switch lights and power on at their respective main switches, where required.
- ❑ Start machine using the Start Button.
- ❑ Check that the Emergency Stop Button is working.
- ❑ Release the Emergency Stop button in preparation for immediate use.
- ❑ Leave Emergency Stop Button in stopped position if not to be run immediately.

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# SAFE OPERATING PROCEDURE: BALLBOT

**Machine Pre-operational Safety Checks – Safety Precautions that MUST be Observed:**

- ❑ Visual inspection of machine to verify it is in good operational order, ensuring no damage to any stationary or moving parts, electrical cords etc. Any unsafe equipment is to be reported to an authorised staff member and tagged out.
- ❑ Ensure lighting and power are switched on their respective main switches, where required.
- ❑ Be aware of other activities happening in the immediate area.
- ❑ Ensure that no slip and/or trip hazards are present.
- ❑ Ensure that machine lighting is adequate.
- ❑ Start machine using the Start Button.
- ❑ Check that the Emergency Stop is working.
- ❑ Check that all machine guards, electrical interlocks and Emergency Stop micro-switch are correctly positioned, locked in place and in proper working condition.

## IF IN DOUBT, ASK

- ❑ Locate and be familiar with the operation of the ON/OFF starter switch.
- ❑ Locate and be familiar with the operation of the Emergency Stop button. Check Emergency Stop button is working.
- ❑ Where required, tether top of ballbot frame to overhead bar with adequate load capacity, to ensure ballbot does not move beyond operating range. Note that Ballbot should be stopped before rope is in tension.

**Operation:**

- ❑ Ensure power is switched on through its switch. This will be indicated by lit LEDs on the motor controller, breakout and Dragon12 boards.
- ❑ Ensure emergency stop switch is at released position, and in easily-accessed location (ie held in hand)
- ❑ Start controller program, and release after first audible tone. (Ballbot will start automatically)
- ❑ Monitor Ballbot closely while running, maintaining a safe distance.
- ❑ NEVER LEAVE BALLBOT RUNNING WHILST UNATTENDED.
- ❑ To stop ballbot, press the emergency stop button, and allow to come to rest.
- ❑ Ensure that the program has been stopped (by pressing the dark grey rectangular button on the NXT brick) before releasing the emergency stop switch.
- ❑ Clean up machine and surrounding area.

THE UNIVERSITY OF ADELAIDE AUSTRALIA

# SAFE OPERATING PROCEDURE: BALLBOT

## General Safety

- ❑ Visual inspection of plant prior to use. Unsafe plant to be tagged out and reported to Workshop Manager.
- ❑ Keep all parts of your body and attire safely clear of the rotating and moving parts, at all times.
- ❑ Ensure scheduled maintenance for this machine has been carried out; including scheduled testing of Emergency Stop.
- ❑ Only authorised qualified staff to operate this machine, or students who have received full Competency Training from an authorised qualified staff member (recorded in training register).
- ❑ Assistance from other staff or students to be sought and support frames used when handling and drilling large, long and/or heavy sections of material.
- ❑ NEVER LEAVE BALLBOT RUNNING WHILST UNATTENDED.
- ❑ Safety glasses must be worn at all times during the operation of this machine.
- ❑ Closed Toe Type Shoes must be worn during the operation of this machine.
- ❑ Loose hair to be securely tied back, loose clothing to be rolled up and/or secured, loose jewellery to be removed.
- ❑ Hearing protection to be worn, where appropriate to the task being performed.
- ❑ Leather Safety Gloves to be worn, where appropriate to the task being performed, and;
- ❑ Rag or cotton waste (eg for cleaning) must not be used near the machine while the motors are rotating.
- ❑ Switch Off machine before leaving it unattended.

**Note:** This Safe Operating Procedure must be reviewed:
a) after any accident, incident or near miss;
b) when training new staff;
c) if adopted by new work group;
d) if equipment, substances or processes change; or
e) within 1 year of date of issue.