TESTBED FOR TRAJECTORY CONTROL OF A TWO-WHEELED ROBOT

BY

NATHAN KANDO

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2017

MASTER OF SCIENCE THESIS

BY

NATHAN KANDO

APPROVED:

Thesis Committee:

Major Professor RICHARD J. VACCARO

PETER F. SWASZEK

ORLANDO MERINO

NASSER H. ZAWIA
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2017

Abstract

This thesis develops a test platform for a control problem.

The inverted pendulum is selected as a well-established control problem. It is representative of an unstable nonlinear system which may remain balanced using any of several methods. Once balancing is achieved, multidimensional maneuvering is added as a supplemental control objective.

To approach the control problem hardware is selected which is then characterized and simulated, and then operated while communicating operational data.

The thesis provides a detailed description of the approaches to:

- Selecting the hardware.
- Characterizing the hardware.
- Developing a functioning controller.
- Simulating the results.

Additionally, a modular test platform is developed such that additional control approaches or characterization models could be implemented and hot-swapped.

Acknowledgements

The research performed and described in this document could not have been completed without the insight and guidance of several parties.

First and foremost, I would like to acknowledge Professor Richard J. Vaccaro for his capabilities, his mentorship, and his professionalism in the STEAM fields community. The great level of knowledge which he has provided to me and countless others, *whether in academic text, course lecture, personal conversation, or penned scrawl on scrap paper*, is not to be underestimated.

Additional acknowledgement should be provided to Professor Joshua Hurst, who publicly released software drivers for the MinSeg hardware, and who extended the supported platforms of those drivers to make them accessible beyond their original use.

I would like to thank all other cited academic parties, particularly, [*but in no particular order*]:

· Yorihisa Yamamoto, for his two-wheeled robot dynamic model.
· "Phil O", Ryo Watanabe, and related peers, for their NXT motor studies.
· Mike Peltier, for his two-wheeled robot control publication
· Brian Howard and Linda Bushnell, for their MinSeg publication

Thank you for making your works available.

Finally, but of no less consideration, I would like to thank Holly Gaboriault who took photos on request, as well as the members of the Wikimedia Foundation, and the Stackexchange and Matlab Central communities, who offered their multidisciplinary expertise regarding any number of subjects, *whether mathematical, technical, typographical, or otherwise*, rapidly and on-demand.

Dedication

For those who led by example
with an unyielding optimism
and a devotion to the field
which has only enriched;

for Laura and for Professor Richard Vaccaro.

Thank you for your own sizable accomplishments
and for your guidance in mine.

Table of Contents

List of Figures

List of Tables

List of Code Listings

CHAPTER 1.

Introduction

1.1. Purpose

The intent of this thesis is as follows:

1. Establish Control Problem
   1.1. Select a well-established control problem as a focal point. [*Inverted Pendulum: Two-wheeled Robot*]
2. Establish Plant
   2.1. Select and derive equations for a physical dynamics model associated with the control problem. [*Yamamoto [1]*]
   2.2. Select real-world hardware which parallels the selected physical dynamics model. [*MinSeg M2V5*]
   2.3. Characterize the selected hardware to populate the physical dynamics equations.
3. Establish Controller
   3.1. Select and derive controller design(s) to address the selected control problem. [*Optimal Controller*]
   3.2. Select introductory gains to populate the selected controller design equations.
4. Establish Test Platform (*for Simulation and Actuation*)
   4.1. Select an integrated development environment (IDE). [*Mathworks Software Suite*]
   4.2. Develop a robust test software platform on the selected IDE which is able to:
      - Model established plant in simulation.
      - Model established controller in simulation **and** in real-time.
      - Program selected real-world hardware prior to operation.
      - Communicate with selected real-world hardware during/after operation.
5. Execute Testing
   5.1. Determine idealized controller equation parameters via simulation testing/post-processing.
   5.2. Determine actualized controller equation parameters via real-time testing/post-processing.

1.2. Statement of the Problem

The two-wheeled robot is a well-established control problem. The robot is topheavy and must continually work to balance itself. The robot is able to move freely on a two-dimensional plane; however, any movements performed by the robot create additional disturbances against its ability to balance itself.

Numerous command regulator approaches (*PID, pole-placement, optimal*) have been developed to control such a device; however, no one approach has been determined as a clear choice. Additional functionalities other than command regulators which significantly improve performance may also be implemented in a controller.

This study therefore intends to comparatively study multiple control approaches involving optimal-control-focused command regulators and to study the effects of additional functionalities which may be beneficial in general control cases. As a prerequisite to this work, a test platform must be developed for which to design the controllers. This study intends to design the test platform such that:

- Studies could be performed in simulation *(using a hardware-equivalent model)*.
- Studies could be performed on actual hardware.
- Similar work involving alternate control methods could easily be incorporated on both.

The intent of the latter is to significantly diminish several barriers to entry to perform a control study relating to hardware *(initial implementation, interfacing/communication, and theoretical/simulation modeling)*. This would ideally encourage future studies as well as draw them to a common platform, which would allow for effective comparisons between those studies.

1.3. Methodology

The methodology of this thesis is as follows:

1. Select a well-established control problem as a focal point. [*Inverted Pendulum: Two-wheeled Robot*]
   - Select compatible hardware and software. [HW: *MinSeg (Two-Wheeled Robot)*; SW: *Mathworks Matlab & Simulink*]
   - Implement basic hardware-software interfaces.
     - Process signals input to hardware drivers.
     - Process raw signals output from hardware sensors.

     [*Datatype conversion*; *Unit conversion*; *Derivation/Integration*; *Filtration*]
2. Develop a modular test platform.
   - Establish infrastructure.
     - Develop a unified, modular Simulink model which is capable of representing any desired system configuration. [*Variant subsystems used.*]
     - Create a Matlab script hierarchy which is able to:
       · Configure the Simulink model to any desired system configuration.
       · Configure the Simulink model to any desired build/run state.
       · Organize the relatively large number of parameters involved in such a system.
       · Minimize the effort required for the user to incorporate additional system configurations.
       · Minimize the effort required for the user to transition between any system configurations.
   - Establish robust methods of signal routing.
     - Implement bus structures.
     - Implement serial communication between hardware and development computer.
       · Minimimize sampling interval within the limits of the board hardware.
       · Process transmitted signals prior to sending and reconstruct after receiving.

- Calibrate hardware sensors.
  - Mitigate gyroscope bias.
- Develop theoretical plant model.
  - Research (non-linear) physical equations.
  - Linearize the physical equations.
    - Develop a state-space model.
  - Acquire linear plant model parameters.
  - Implement linear plant model into unified test platform.

3. Design and develop and controller for the test platform.
   - Implement dynamic reference tracking to mitigate bias on the body angular velocity ϕ sensors.
   - Determine control gains using LQR.

1.4. Bibliography

[1] Y. Yamamoto. (May 1, 2009). "Nxtway-gs (self-balancing two-wheeled robot) controller design," [Online]. Available: `https://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs--self-balancing-two-wheeled-robot--controller-design` (visited on 07/03/2017).

CHAPTER 2.

Preliminary Decisions

2.1. Selection of Control Problem

The focal control problem was designated to be the *two-wheeled robot*, a special case of the *inverted pendulum*.

2.1.1. Inverted Pendulum

In control theory, the balancing of an inverted pendulum is a well-established problem [2].

In such a problem, a rigid, column-like mass is used as a pendulum. One end of the pendulum is mounted to a motoring device. The mounted end of the pendulum is granted a degree of freedom to rotate. If the pendulum is inverted (positioned in a standing position), any disturbance will ultimately tip it such that it falls.

One such system is depicted in Figure 2.1. In this simple case, the wheels of the cart allow it to move along a one-dimensional plane (a linear path).

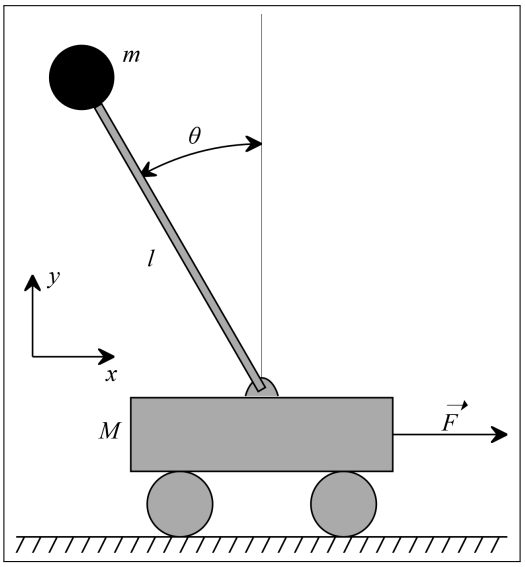

Figure 2.1.: [Selection of Control Problem]: Inverted Pendulum on Cart [3]

The motoring device is used in such a setup to provide counterforces to the mounted end of the pendulum. These counterforces are intended to ultimately return the top of the pendulum to its inverted (standing) position.

For the actuator to successfully perform these actions, a controller (calculation device) is required. The controller, with the assistance of sensory data, is able to dynamically calculate (in real time) the exact forces needed to reestablish the positioning of the pendulum to a standing equilibrium. The controller then communicates the magnitude and direction of these forces to the motoring device which actuates the forces in the physical space. This in turn changes the state of the system, requiring that the controller continually recalculate the forces needed to return to equilibrium.

This problem may be further complicated by implementing trajectory control, in which the operator may command the device to move to one or more different locations. In such a scenario, the device must maintain its control of the balance of the inverted pendulum during and after moving.

2.1.2. Two-Wheeled Robot

The two-wheeled robot is a special case of the inverted pendulum model. In this case, the inverted pendulum model is reduced to only the pendulum and the wheels. The entirety of the robot hardware forms the pendulum, and the pendulum is coupled directly to the wheels.

One such device is depicted in Figure 2.2. In this case, the robot is being used in a medical application. The significance of the two-wheeled robot is not related to any one application; rather its ability to balance allows the added inclusion of top-heavy architectures in design options.

The robot has two wheels, each of which is coupled to an individual motoring device (included in the robot hardware). The motoring devices are able to act independently; therefore, the device is capable of turning and moving across a two-dimensional plane. This transition from one shaft to two shafts creates additional complexity in the system which must be considered in the design of the controller.

Figure 2.2.: [Selection of Control Problem]: Two Wheeled Robot [4]

2.2. Selection of Hardware

The selection of hardware consists of selecting:

- A specific device to serve as the plant with respect to the designated control problem.

[*MinSeg two-wheeled robot.*]

- A specific device to serve as the controller with respect to the designated control problem.

[*Arduino Mega2560 single-board microcontroller, included with MinSeg two-wheeled robot.*]

- A specific computer to be used to interface with the controller.

[*Available laptop installed with Mathworks Software Suite and supporting software.*]

2.2.1. MinSeg M2V3 Two-Wheeled Robot

The MinSeg two-wheeled-robot was selected as the designated hardware platform due to:

- Its standard inclusion of several components which are considered desirable with respect to performing a control study. [*See Section 2.2.1.1*]
- Its existing published academic work. [*Howard and Bushnell [5]*]

  [*This highlighted the device as a suitable hardware platform for control studies.*]
- Its existing driver support [*for the Mathworks software environment.*]. [*Mathworks [6]* and *Hurst [7]*]
- Its use of an Arduino-brand single-board microcontroller.

  [*This highlighted a significant level of support for a principal component.*]
- Its relatively affordable cost. [*~$300*]

Specifically, the MinSeg Model *M2V3* [8] was selected as the designated hardware platform, due to:

- Its standard inclusion of two [*equivalent but independent*] motoring axes.

n.axes	**Movement**	
1	*One-dimensional*	*(single, straight line only)*
2	*Two-dimensional*	

The MinSeg M2V3 is depicted from the front, the left-side, and the rear in Figure 2.3. The MinSeg M2V3 is depicted from the front in an exploded view in Figure 2.4. In both figures, a United States quarter is depicted for the comparison of scale.

In Figure 2.4, in addition to separated motors and a separated wheel and wheel axle, two auxiliary components are depicted. To the left is a retractable USB cable used to connect to a development PC. To the right, beside the battery, are two of the same Lego component. These were used to mount and swing the robot as an uninverted pendulum, [as described in Section 3.4.3.2].

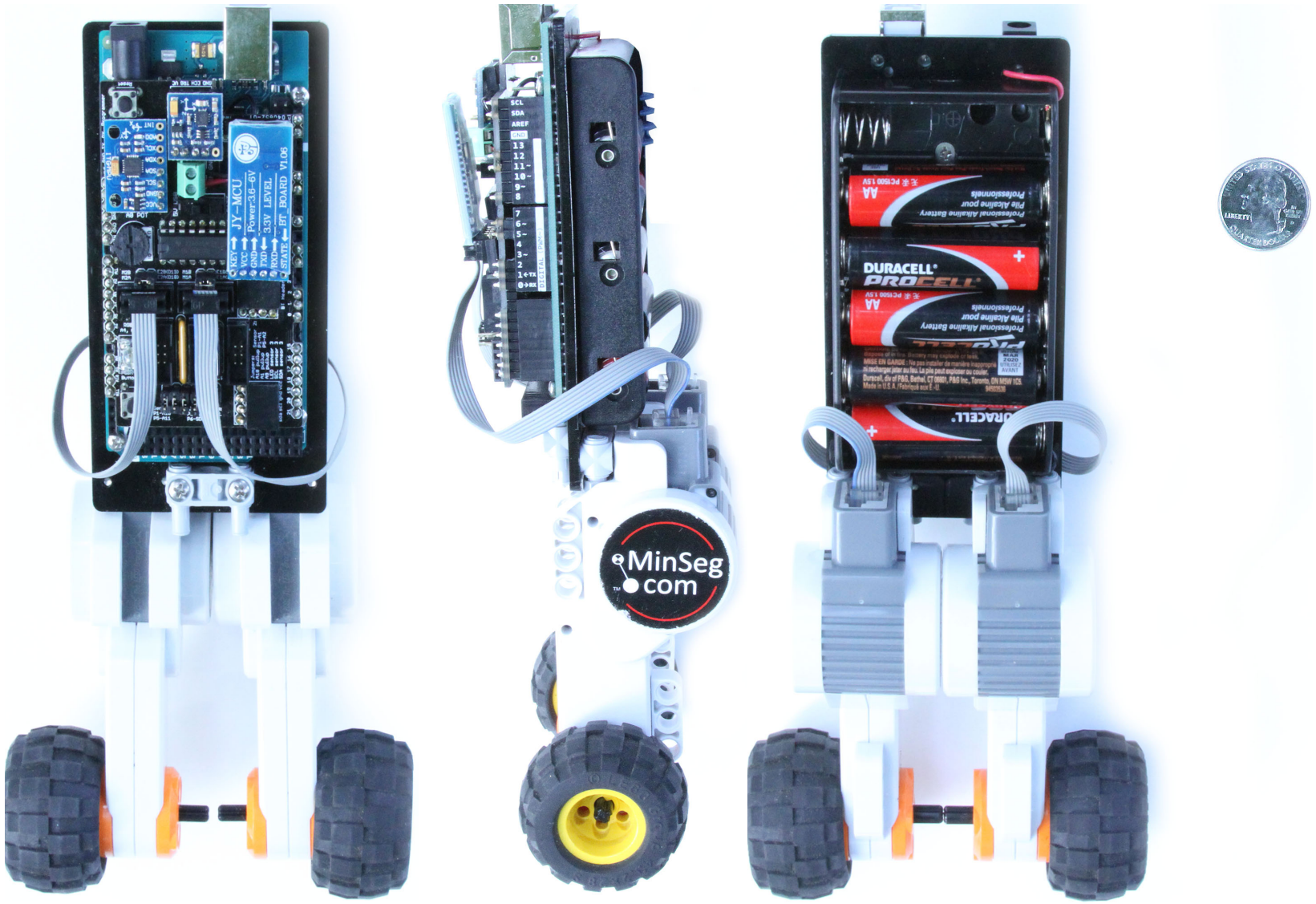

Figure 2.3.: [Selection of Compatible HW & SW]: MinSeg M2V3 (Multiview)

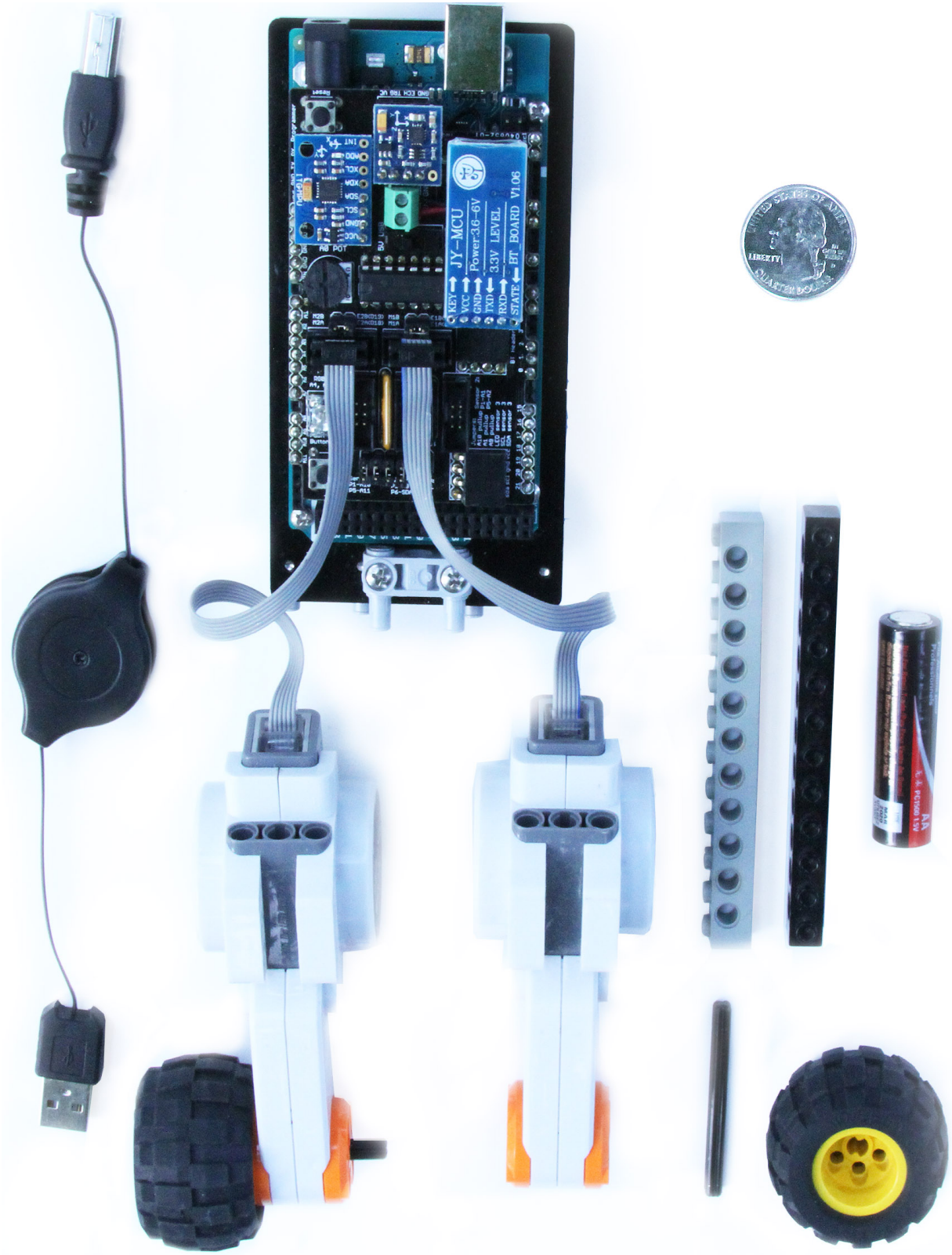

Figure 2.4.: [Selection of Compatible HW & SW]: MinSeg M2V3 (Exploded)

2.2.1.1. Components

As stated in Section 2.2 the MinSeg M2V3 two-wheeled robot was selected due to its inclusion of all of the desired components to perform a control study. These components are defined in Table 2.1.

Where beneficial, the components in Table 2.1 are described in greater detail in the sections which follow.

Table 2.1.: [Selection of Compatible HW & SW]: MinSeg Components

Component (Desirable)	**Part Description**
Programmable microprocessor	1x Arduino single-board microcontroller [Mega 2560]
Dual parallel electric-driven traction motors	2x Lego Mindstorm NXT servo motor [*Includes: 1x DC motor, 1x gearbox, 1x encoder*]
Tires with a relatively-high coefficient of friction	2x Lego wheel [2x [43.2 x 28] Balloon Small]
Motor drivers	4x half-H-driver [1x SN754410]
Sensors permitting sufficient observability of:	-
The angular velocity of the wheels	2x Encoder [*See motor.*]
The 3-dimensional position of the body	1x 3-axis gyroscope & accelerometer [1x MPU6050]
PC Communication (*during operation*):	-
Wired	Serial: USB [*Default. Included with microcontroller.*]
Wireless	Serial: Bluetooth [*Supported, but sold separately.*]
Power Source:	-
Wired	USB 5 [V]
Wireless	Battery holster (6x AA) 9 [V]

Component (Unnecessary)	**Part Description**
-	1x Magnetometer [1x HMC5883L]
-	1x Potentiometer [1x 3352]

2.2.1.2. Arduino Single-Board Microcontroller

The MinSeg M2V3 is primarily built upon an Arduino Mega 2560 single-board microcontroller. Arduino is company, project, and user-community which focuses on the development of open-source computer-hardware and software with respect to single-board microcontrollers [9]. A major boon of Arduino products is the relatively high level of support which has manifested with their popularity, including (*but not limited to*) the company itself, academic communities, hobbyist communities, as well as third-party private supporters such as math-software company Mathworks.

A brief comparison between the Arduino Mega 2560 and the more standard Arduino Uno is provided in Table 2.2. Most notably, the Mega 2560 has an increased number of input and output interfaces, a superior clock [10] (*not apparent in the table*), and increased memory versus the Arduino Uno.

Table 2.2.: [Selection of Compatible HW & SW]: Arduino Board Comparison [11]

Characteristic	**Uno**	**Mega 2560**	**Unit**
Microcontroller	ATmega328 MCU	ATmega2560	-
Programming Interface	USB [*via* ATMega16U2]	USB [*via* ATMega16U2]	-
UART [*Universal Asynchronous Receiver/Transmitter*]	01	04	-
Clock	ceramic resonator	crystal oscillator	-
Clock Speed	16	16	MHz
Operating Voltage	05	05	V
Number of Digital I/O [*Inputs/Outputs*]	14	54	-
PWM	06	14	-
Analog Inputs	06	16	-
Memory/Storage:			
Permanent (*Flash*)	32	256	kB
Permanent (*EEPROM*)	01	04	kB
Working (*SRAM*)	02	08	kB

Due to the inclusion of a USB port (*which is coupled to one of the UARTs*), board-to-PC interfacing (*in either direction*) is relatively convenient, as no special equipment is necessary (*beyond a PC containing integrated development environment (IDE) software*). This applies to programming the board (*via the USB programming interface*), as well as communicating signals during operation.

Photos of the Arduino microcontroller are depicted in Figures 2.5 - 2.6. Additionally, a pin layout is provided in Figure 2.7.

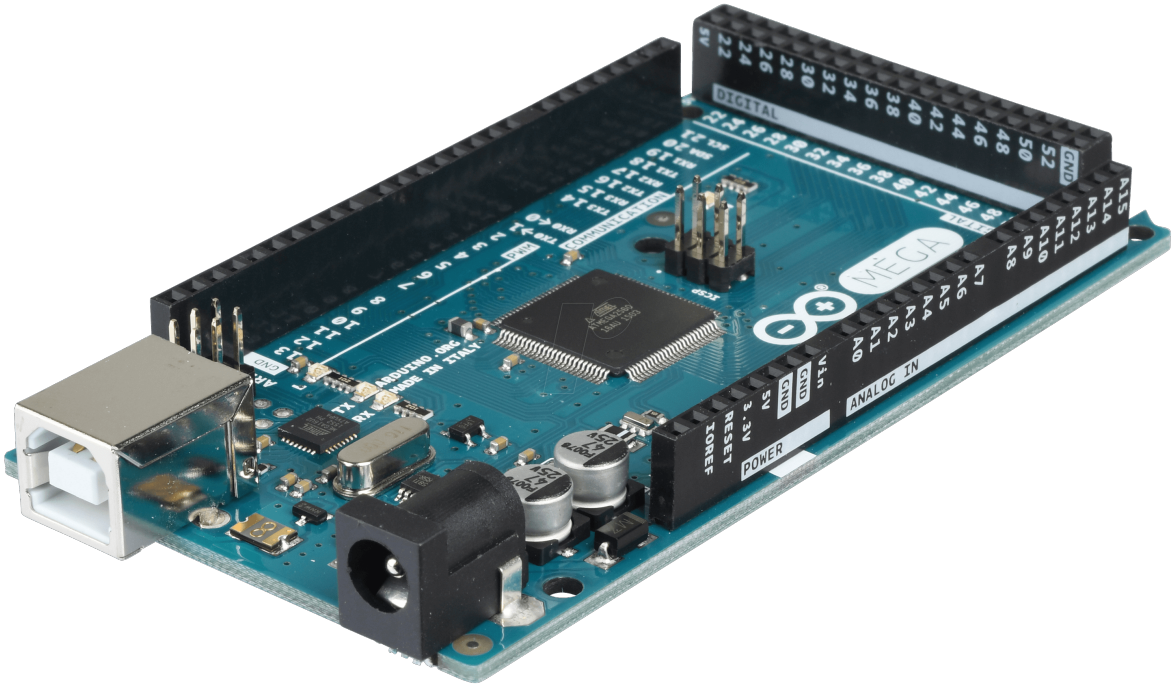

Figure 2.5.: [Selection of Compatible HW & SW]: Arduino Mega 2560 (Isometric View) [12]

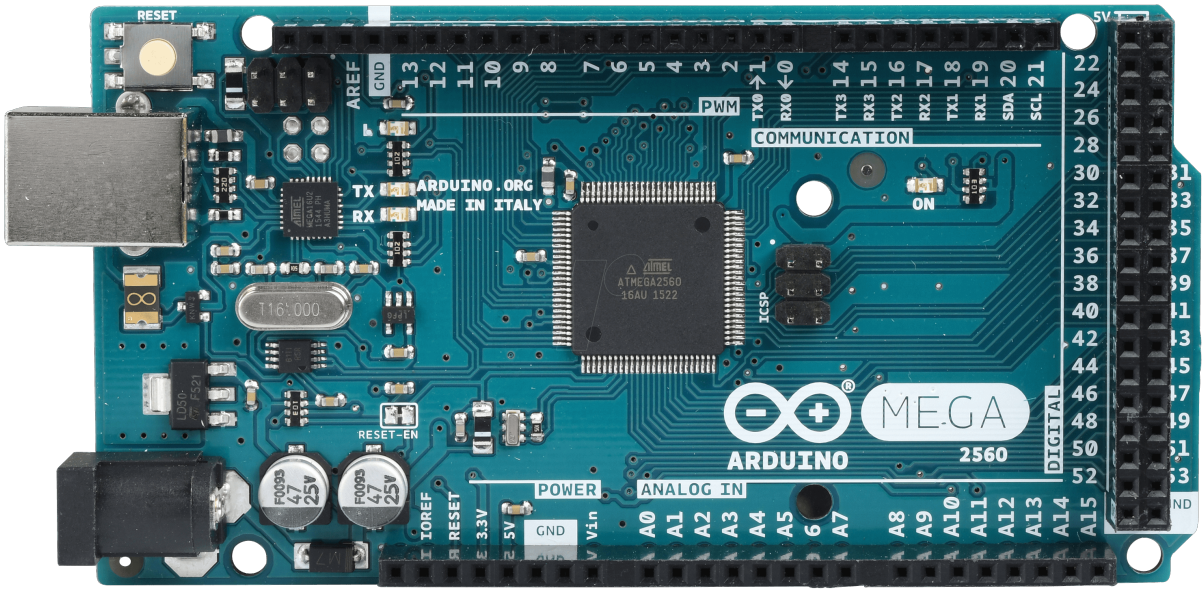

Figure 2.6.: [Selection of Compatible HW & SW]: Arduino Mega 2560 (Top View) [12]

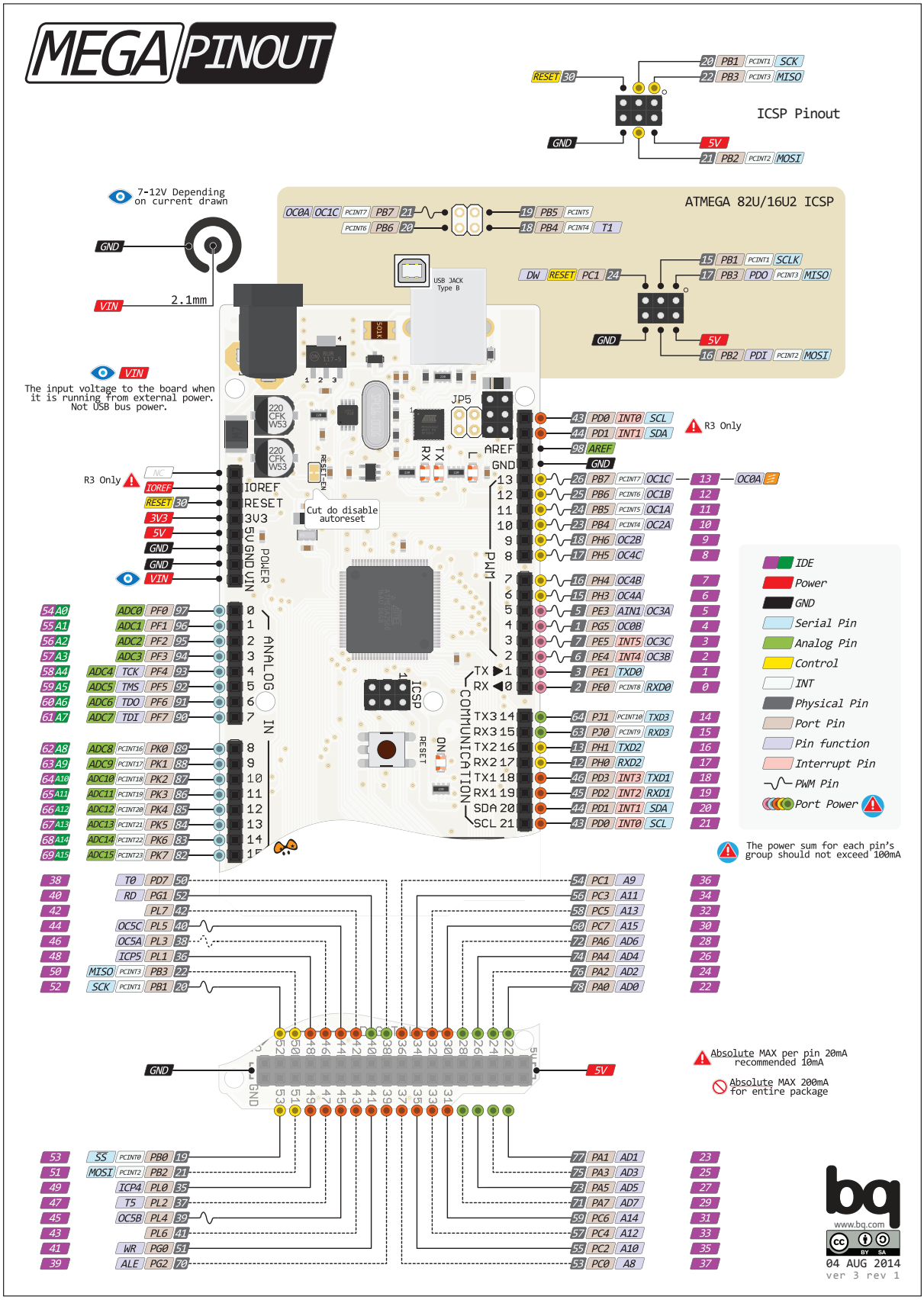

Figure 2.7.: [Selection of Compatible HW & SW]: Arduino Mega 2560 (Pin Map) [13]

2.2.1.3. Power Source

The MinSeg M2V3 offers two independent sources of power:

- External power via a USB port
- Internal power via an embedded battery holster

A physical switch exists on the MinSeg device to alternate between the two modes of power sourcing.

External-Sourced Power (USB-Cable Connection)

Externally-sourcing power via the USB port offers a constant 5 [V], per the USB standard; however, the cable must be consistently connected to the robot body during use.

Internal-Sourced Power (Battery Pack)

As an alternative to externally-sourced power, power may be sourced from a battery holster embedded within the MinSeg. The battery holster permits the installation of 6 AA-sized batteries.

A typical Alkaline AA-sized battery carries 1.5 [V] at maximum charge. During use, this voltage will rapidly diminish to ~1.25 [V], and more slowly diminish from then on to ~1.00 [V] before rapidly becoming completely discharged, as depicted in Figure 2.8.

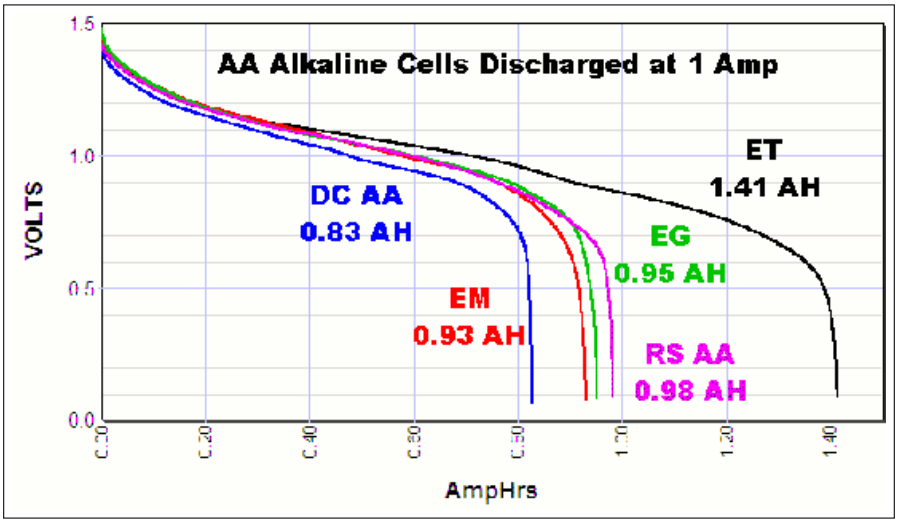

Figure 2.8.: [Selection of Compatible HW & SW]: AA-Battery Voltage During Constant Discharge [14]

To use the battery holster as a power source, all six AA batteries must be installed. The batteries are connected in series and therefore cumulatively offer up to 9.00 [V] when at full charge. During typical operation, the batteries will more likely offer a reduced voltage, ~7.50 [V].

Therefore, sourcing power from the battery holster offers consistently greater voltage than external USB-connected sources, (*so long as the batteries are not completely discharged*), and additionally precludes the use of any wiring which could obstruct testing and operation.

2.2.1.4. Motor Driver

The MinSeg M2V3 uses a Texas Instruments (TI) *SN754410*: Quadruple Half-H Driver chip as a motor driver. Supplementary information from the SN754410 datasheet is depicted in Figure 2.9 and Tables 2.3 - 2.6.

Figure 2.9 and Table 2.3 exhibit that the chip has four inputs A and four corresponding outputs Y. Table 2.5 provides a simplified description of the behavior of any one input with respect to its corresponding output:

Input A acts a switch for corresponding output Y:

• If the input pin A is enabled V_{IH}, then the corresponding output Y will output V_{CC2} $[V]$.

• If the input pin A is disabled V_{IL}, then the corresponding output Y will output $\quad 0 \quad$ $[V]$.

Note: It can be assumed that the enable EN is engaged whenever necessary during MinSeg operation.

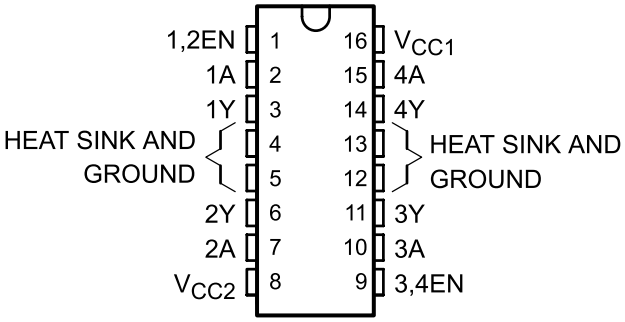

Figure 2.9.: [Selection of Compatible HW & SW]: Motor Driver Pin Map [15]

Table 2.3.: [Selection of Compatible HW & SW]: Motor Driver Pin Legend [15]

PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, non-inverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to circuit board ground plane with multiple solid vias
V_{CC2}	8	—	Power VCC for drivers 4.5V to 36V
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
V_{CC1}	16	—	5V supply for internal logic translation

Table 2.4.: [Selection of Compatible HW & SW]: Motor Driver Pin Function Legend [15]

INPUTS[2]		OUTPUTS
A	EN	Y
H	H	H
L	H	L
X	L	Z

H = high-level
L = low-level
X = irrelevant
Z = high-impedance (off)

Table 2.5.: [Selection of Compatible HW & SW]: Motor Driver Operating Conditions [15]

		MIN	MAX	UNIT
V_{CC1}	Logic supply voltage	4.5	5.5	V
V_{CC2}	Output supply voltage	4.5	36	V
V_{IH}	High-level input voltage	2	5.5	V
V_{IL}	Low-level input voltage	–0.3[1]	0.8	V
T_J	Operating virtual junction temperature	–40	125	°C
T_A	Operating free-air temperature	–40	85	°C

Table 2.6.: [Selection of Compatible HW & SW]: Motor Driver Switching Characteristics [15]

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{d1}	Delay time, high-to-low-level output from A input	See Figure 3		400		ns
t_{d2}	Delay time, low-to-high-level output from A input			800		ns
t_{TLH}	Transition time, low-to-high-level output			300		ns
t_{THL}	Transition time, high-to-low-level output			300		ns
t_{en1}	Enable time to the high level	See Figure 4		700		ns
t_{en2}	Enable time to the low level			400		ns
t_{dis1}	Disable time from the high level			900		ns
t_{dis2}	Disable time from the low level			600		ns

Table 2.5 specifies voltages associated with normal chip operation. The voltage source for SN754410 outputs V_{CC2} is wired to the MinSeg power source, and may therefore vary, from $4.5 - 9.0$ $[V]$, (see Section 2.2.1.3).

The SN754410 inputs A are connected to digital output pins on the Arduino microcontroller, specifically those which are capable of producing pulse width modulated (PWM) signals (see Sections 2.2.1.2). Programmed binary lows on the Arduino board will induce 0 $[V]$ and programmed binary highs will induce 5 $[V]$, (*which is the Arduino board operating voltage*).

To achieve voltages other than V_{CC2} exactly, PWM voltage signals are used. The Arduino can set its digital output pin to high for a defined fraction of the time spanning each sample interval of the Arduino board. The effect of the added switching during each sample should be considered minimal, since the MinSeg sample interval operates at the 10^{-3} $[s]$ scale (*as set by the operator*), and the SN754410 switching interval operates at the 10^{-7} $[s]$ scale, (*per Table 2.6*).

As stated in Section 2.2.1.1, there are two DC motors. Each has a positive and negative lead.

2.2.1.5. Motor, Gearbox, and Encoder

The MinSeg implements two Lego NXT servo motors. Each Lego NXT servo motor contains a DC traction motor, a gearbox, and an encoder. A three-dimensional model of the component is depicted in Figure 2.10.

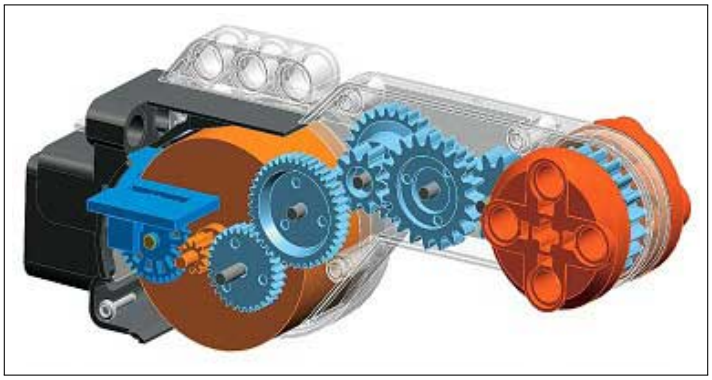

Figure 2.10.: [Selection of Compatible HW & SW]: Lego NXT Motor (3D Model) [16]

Although Lego did not publicly disclose all of the characteristic parameters of their components, the hobbyist community reverse engineered several of these values by performing various tests and also by (irreversibly) dismantling a spare [16]. The dismantled component is depicted in Figure 2.11.

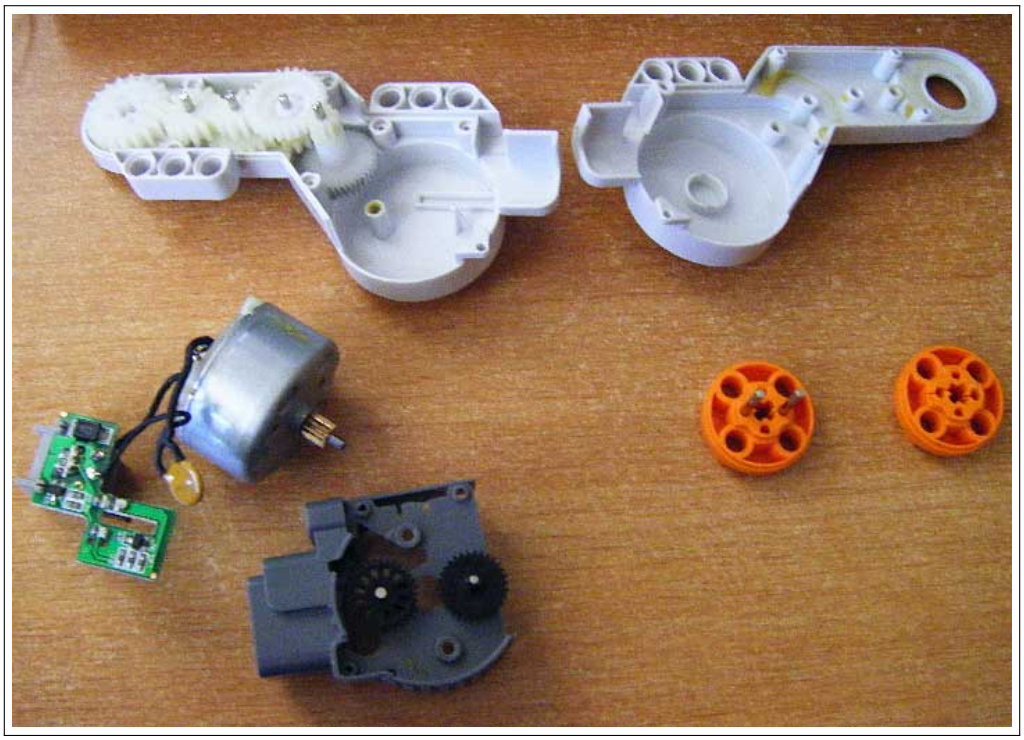

Figure 2.11.: [Selection of Compatible HW & SW]: Lego NXT Motor (Exploded) [16]

Table 2.7 provides a legend for the different components in Figures 2.10 - 2.11.

Table 2.7.: [Selection of Compatible HW & SW]: Lego NXT Motor Figure Legend [16]

Component	**Figure 2.10**	**Figure 2.11**	
Enclosure	Translucent	Top	Grey
Encoder	-	-	-
Enclosure	Dark blue	Bottom	Black
Gearing	Dark grey	Bottom	Dark grey
PCB	-	Left	Green
Motor	Light orange	Left	Chrome
Gearbox	Light blue	Top-Left	White
Wheel axle mount	Dark orange	Right	Orange

Gearing

The encoder, the motor, and the wheel axle mount are coupled through gearing. Thus, the angular velocity ω of any one of these components can be related to the angular velocity of any one of the other components based on the the number(s) of teeth between each component, as exhibited in Eqn. [2.1], where k represents a ratio and n represents an integer count.

$$k_{\omega\ A2B} = \frac{n_{\omega\ B}}{n_{\omega\ A}} = \left[k_{teeth\ A2B}\right]^{-1} = \left[\frac{n_{teeth\ B}}{n_{teeth\ A}}\right]^{-1} \tag{2.1}$$

The number of teeth in each gearing is depicted in Figure 2.12. Teeth counts in the same row are coupled by teeth. Teeth counts in the same column are coupled by axle, (*and therefore rotate at the same rate, independent of teeth count*).

20	|	13	10	·	·	·	|	·	|	·
·	|	·	20	10	·	·	|	·	|	·
·	|	·	·	27	09	·	|	·	|	·
·	|	·	·	·	40	30	|	10	|	32

Figure 2.12.: [Selection of Compatible HW & SW]: Lego NXT Motor Gear Teeth Map [16]

Additionally, in Figure 2.12, each component is separated by a vertical bar. From left to right: wheel axle mount, gearbox, motor, encoder. The completed relation between the wheel axle mount and the gearbox is exhibited in Eqns. [2.2 –2.6].

$$k_{gearTeeth\ wheel2motor} = \left[\frac{13}{20} \cdot \frac{10}{13}\right] \cdot \left[\frac{10}{20}\right] \cdot \left[\frac{09}{27}\right] \cdot \left[\frac{30}{40} \cdot \frac{10}{30}\right] = \frac{01}{48} \tag{2.2}$$

$$k_{gearTeeth\ motor2encoder} = \frac{32}{10} \tag{2.3}$$

$$k_{\omega\ wheel2motor} = \left[k_{gearTeeth\ wheel2motor}\right]^{-1} = \frac{48}{01} \tag{2.4}$$

$$k_{\omega\ motor2encoder} = \left[k_{gearTeeth\ motor2encoder}\right]^{-1} = \frac{10}{32} \tag{2.5}$$

$$k_{\omega\ wheel2encoder} = k_{\omega\ wheel2motor} \cdot k_{\omega\ motor2encoder} = \frac{15}{01} \tag{2.6}$$

2.2.2. Development PC

Designations pertaining to the development PC with respect to the test platform are exhibited in Table 2.8.

Table 2.8.: [Selection of Compatible HW & SW]: Development PC Specifications [14]

Hardware	**Version**
PC	2015 Macbook Pro [17]

Software	**Version**
Operating System (OS)	macOS 10.12.5
Mathworks Software Suite	r2017a
· *MATLAB*	-
· *Simulink*	-
· *Control System Toolbox*	-
· *DSP System Toolbox*	-
· *Instrument Control Toolbox*	-
· *MATLAB Coder*	-
· *Simulink Coder*	-
· *Simulink Desktop Real-Time*	-
· *Matlab Support Package for Arduino Hardware*	17.1.0
· *Simulink Support Package for Arduino Hardware*	17.1.0
Xcode [*A Mathworks (macOS)-Supported Compiler [18].*]	7.3.1
Rensselaer Arduino Support Package Library (RASPLib) [*A third-party Simulink Support Package for MinSeg Hardware [7].*]	1.1

2.2.2.1. Designated PC

A 2015 Macbook Pro PC was selected as the designated development PC, as this was available to the researcher without the need to request additional funding.

2.2.2.2. Designated Operating System

macOS was selected as the designated operating system, as this was the only operating system installed on the designated PC. (*Version 10.12.5 was the most up to date version at the time of research.*)

Alternative Operating System Compatibility

Although the macOS operating system was used, alternative operating systems (*Windows and/or Linux*) would be equally acceptable.

Such a transition would primarily require an alternative Mathworks-supported compiler [18] which would be compatible with the new operating system. Slight alterations to the method of determining the test platform serial communication channelwould also be required.

It is not expected that such a transition would be prohibitively difficult.

2.2.2.3. Designated Hardware-Interfacing Software

The Mathworks Software Suite was selected as the designated hardware-interfacing software due to:

- Its first-party support for programming real-time hardware.
- Its first-party support for simulating real-time hardware.
- Its first-party driver support for Arduino-brand microcontrollers.
- Its third-party driver support for the MinSeg.
- Its first-party support for serial communication with hardware in real-time.
- Its relatively user-friendly language and interfaces.
- Its relative commonality among students and academic institutions.

  [*The software environment was already relatively familiar to the author and to the advising professor prior to performing this study.*]
- Its relatively affordable cost. [*With respect to students and academic institutions. ~$150*].

2.3. Selection of a Hardware Model

The physical plant model developed in Yamamoto [1] was used, due to:

- Use of state variables involving:
  - Body pitch angle α
  - Body yaw angle Ψ
  - Wheel angle θ
- Existing familiarity of the work by the advising professor.
- Existing knowledge of methods to measure nonintuitive model parameters by the advising professor.

The physical plant model is discussed in greater detail in Chapter 3.

2.4. Selection of Controller Design

Since pole-placement methods had been researched relatively recently under the advising professor, optimal control techniques were researched.

This is discussed in greater detail in Section 4.

2.5. Bibliography

[1] Y. Yamamoto. (May 1, 2009). "Nxtway-gs (self-balancing two-wheeled robot) controller design," [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs--self-balancing-two-wheeled-robot--controller-design (visited on 07/03/2017).

[2] R. J. Vaccaro, *Digital Control: A State-space Approach*, ser. Mcgraw Hill Series in Electrical and Computer Engineering. Mcgraw-Hill College, Jan. 1995, ISBN: 978-0070667815.

[3] Wikipedia. (Jun. 24, 2017). "Inverted pendulum," Wikimedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/Inverted_pendulum (visited on 07/03/2017).

[4] D. Melanson. (Jul. 14, 2016). "Researchers create life-saving ubot-5 robot, play dress-up with it," Engadget, [Online]. Available: https://www.engadget.com/2008/04/17/researchers-create-life-saving-ubot-5-robot-play-dress-up-with/ (visited on 02/05/2017).

[5] B. Howard and L. Bushnell, "Enhancing linear system theory curriculum with an inverted pendulum robot," in *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, IEEE, Hangzhou, China, Oct. 2015. DOI: 10.1109/CSMA.2015.63.

[6] Mathworks. "Arduino programming with matlab and simulink," [Online]. Available: https://www.mathworks.com/discovery/arduino-programming-matlab-simulink.html (visited on 07/03/2017).

[7] J. Hurst. (Jun. 17, 2016). "Rensselaer arduino support package library (rasplib)," 1.1, Rensselaer Polytechnic Institute (RPI), [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/62702-rensselaer-arduino-support-package-library--rasplib- (visited on 07/03/2017).

[8] MinSeg. "M2v3," [Online]. Available: https://minseg.com/collections/minseg-kits/products/minsegshield-kit-m2v3-2-dual-drive-segway-and-line-follower (visited on 07/03/2017).

[9] Wikipedia. (Jul. 1, 2017). "Inverted pendulum," Wikimedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/Arduino (visited on 07/03/2017).

[10] jkenny23. (May 9, 2011). "Crystal vs. resonator," Arduino, [Online]. Available: https://forum.arduino.cc/index.php?topic=60662.0 (visited on 07/03/2017).

[11] Arduino Geeks. (Feb. 2, 2017). "Arduino mega vs uno compared," Arduino Starter Kits, [Online]. Available: https://www.arduinostarterkits.com/reviews/arduino-mega-vs-uno-compared/ (visited on 07/03/2017).

[12] Reichelt Elektronik. "Arduino mega: Arduino mega 2560, atmega1280, usb," [Online]. Available: https://www.reichelt.com/de/en/Single-board-microcontroller/ARDUINO-MEGA/3/index.html?ACTION=3&GROUPID=6667&ARTICLE=119696 (visited on 07/03/2017).

[13] Alberto "PighiXXX". (Aug. 4, 2014). "Mega," PighiXXX, [Online]. Available: http://www.pighixxx.com/test/portfolio-items/mega/ (visited on 07/03/2017).

[14] PowerStream. (Jun. 29, 2017). "Discharge tests of aa batteries, alkaline and nimh," [Online]. Available: https://www.powerstream.com/AA-tests.htm (visited on 07/03/2017).

[15] Texas Instruments. (Jan. 2015). "Sn754410: Quadruple h drivers," [Online]. Available: http://www.ti.com/product/sn754410?qgpn=sn754410 (visited on 07/03/2017).

[16] P. "Philo" Hurbain. (May 15, 2017). "Nxt motor internals," [Online]. Available: http://www.philohome.com/nxtmotor/nxtmotor.htm (visited on 07/03/2017).

[17] Everymac. (May 22, 2017). "Apple macbook pro "core i5" 2.7 13" early 2015 specs," [Online]. Available: http://www.everymac.com/systems/apple/macbook_pro/specs/macbook-pro-core-i5-2.7-13-early-2015-retina-display-specs.html (visited on 07/03/2017).

[18] Mathworks. "Previous releases: System requirements and supported compilers," [Online]. Available: https://www.mathworks.com/support/sysreq/previous_releases.html (visited on 07/03/2017).

CHAPTER 3.

Hardware-Equivalent Dynamics Model

To simulate the dynamics of the hardware, a hardware-equivalent dynamics model was selected. The model was originally derived by Yamamoto [1] and has been successfully used in other control studies [19].

Figures 3.1 - 3.2, depict the physical model of the two-wheeled inverted pendulum as isometric and multiview projections. These figures use Yamamoto's original symbol notation; a legend is provided in Figure 3.2.

Yamamoto [1] makes the following assumptions in Figures 3.1 - 3.2:

- All mass geometries are uniform.
- All masses are uniformly distributed.
- The hardware consists of three principal masses:
  - · A rectangular cuboid [*The body.*]
  - · A cylinder [*The left wheel.*]
  - · A cylinder [*The right wheel.*]

Tables 3.1 - 3.2, define the variables and the parameters, respectively, that the physical model of the two-wheeled inverted pendulum will use.

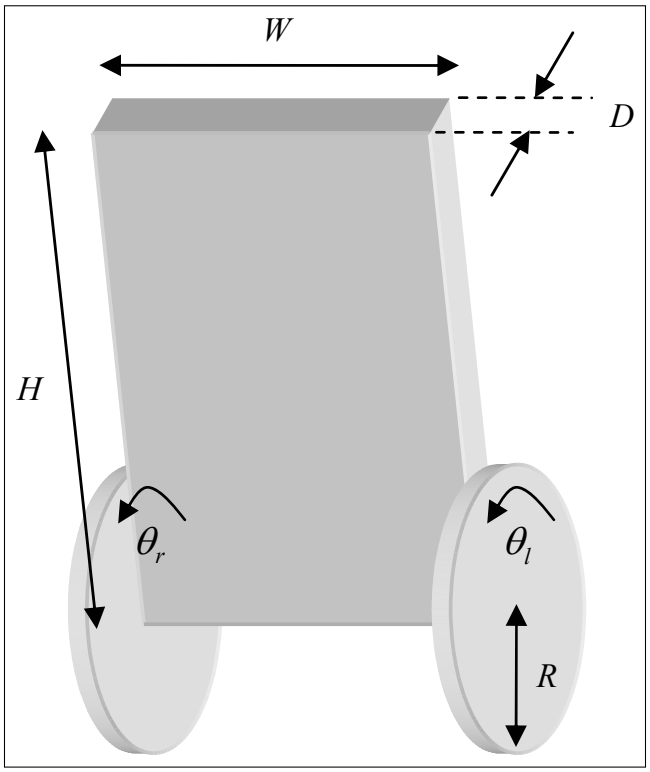

Figure 3.1.: [Hardware-Equivalent Physical Dynamics Model]: Isometric[1]

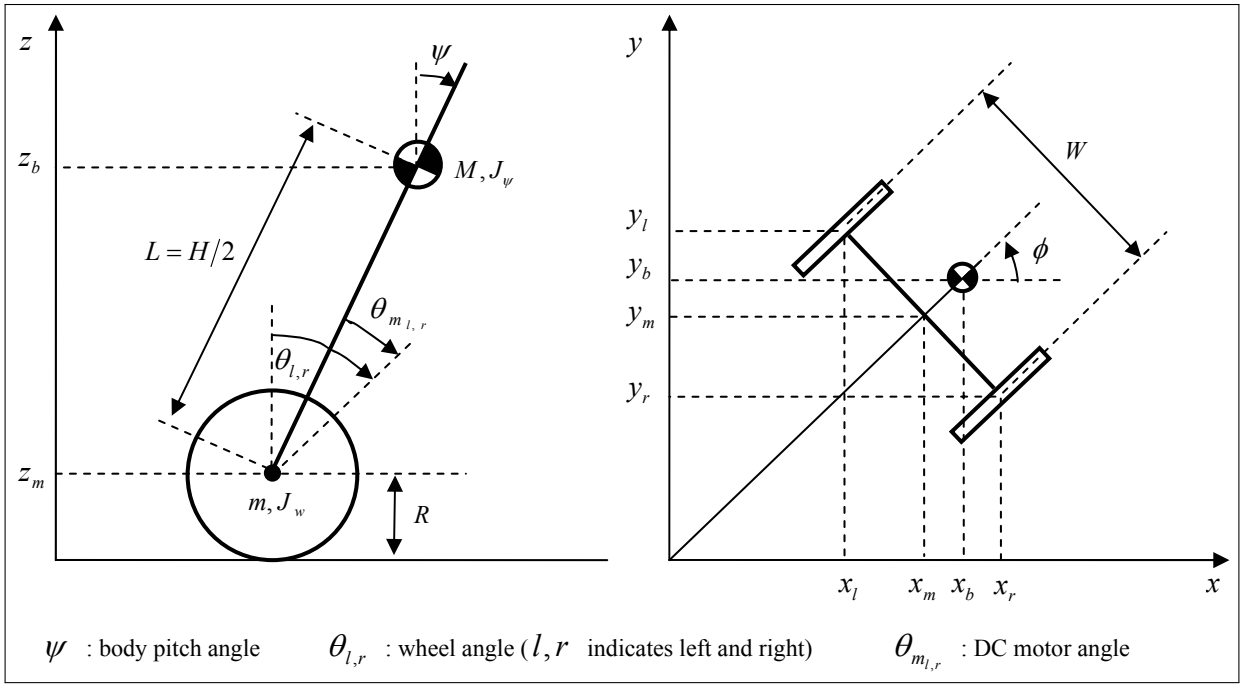

Figure 3.2.: [Hardware-Equivalent Physical Dynamics Model]: Multiview[1]

Table 3.1.: [Simulink]: Root

Symbol	Definition	Unit
θ	Angular position: Wheel $[\theta = \theta_{g,av}]$ [Measured from the wheel center of mass]	rad
θ_g, θ_b	Origin aligns with: $\begin{bmatrix}\text{global rejection vector}\\ (\textit{orthogonal from earth's surface})\end{bmatrix}$ $\begin{bmatrix}\text{body pitch } \phi_x\end{bmatrix}$	rad
$\theta_{av}, \theta_l, \theta_r$	Component: [average of left and right wheels] [left wheel] [right wheel]	rad
ϕ	Angular position: Body [Measured from the wheels center of mass.]	rad
ϕ_x, ϕ_y, ϕ_z	Dimension: [X] [Y] [Z]	rad
p	Translational position $[p = p_w]$ [Measured from the corresponding center of mass]	m
p_x, p_y, p_z	Dimension: [X] [Y] [Z]	m
p_w, p_{wl}, p_{wr}, p_b	Component: [both wheels] [left wheel] [right wheel] [body]	m
v_{mtr}	Voltage: Motor Input	V
$v_{mtr.l}, v_{mtr.r}$	Component: [left-wheel] [right-wheel]	V

Note: When a subscript is unspecified, assume the first option is used by default.

Table 3.2.: [Simulink]: Root

Symbol	Definition	Value	Unit	Source
a_g	Acceleration of gravity: Earth	9.81	$\frac{m}{s^2}$	-
m_w	Mass: Wheel [*Includes wheel axle.*]	0.018	kg	[5]
m_b	Mass: Body	0.381	kg	[5]
$l_{b.h}$	Length: Body: Height	0.2032	m	-
$l_{b.w}$	Length: Body: Width	0.0825	m	-
$l_{b.d}$	Length: Body: Depth	0.0635	m	-
$l_{b.c2a}$	Length: Body: [Center of mass] to [Axis of Rotation]	0.010	m	Sec. 3.4.3
r_w	Length: Wheel: Radius	0.021	m	[5]
$J_{b.\phi_x}$	Moment of Inertia: Wheel	$7.46 \cdot 10^{-6}$	$kg \cdot m^2$	[5]
$J_{b.\phi_x}$	Moment of Inertia: Body: X-axis (pitch)	$1.5573 \cdot 10^{-5}$	$kg \cdot m^2$	Sec. 3.4.1
$J_{b.\phi_y}$	Moment of Inertia: Body: Y-axis (yaw)	$4.2230 \cdot 10^{-4}$	$kg \cdot m^2$	Sec. 3.4.2
R_{mtr}	Motor: Resistance	4.4	Ω	[5]
$k_{mtr.bEMF}$	Motor: Coefficient of Back EMF	0.495	$\frac{V \cdot s}{rad}$	[5]
$k_{mtr.T}$	Motor: Coefficient of Torque	0.470	$\frac{N \cdot m}{A}$	[5]
$k_{fr.m2w}$	Motor: Coefficient of friction: [DC Motor] to [Wheel]	0.0664	-	-

3.1. Nonlinear model

The dynamic motion equations of the two-wheeled robot are derived using the Lagrangian method. The equations are based on the coordinate system provided in Figure 3.2.

3.1.1. Coordinate System

The coordinate system is explicitly defined in Equations (3.1) - (3.6).

$$\begin{bmatrix} \theta_{g.l} \\ \theta_{g.r} \\ \theta_{g.av} \\ \phi_y \end{bmatrix} = \begin{bmatrix} & \theta_{b.l} & + & \phi_x \\ & \theta_{b.r} & + & \phi_x \\ \frac{1}{2} & \cdot & (\theta_{g.l} & + & \theta_{g.r}) \\ \frac{r_w}{l_{b.w}} & \cdot & (\theta_{g.r} & - & \theta_{g.l}) \end{bmatrix} \tag{3.1}$$

$$\begin{bmatrix} \dot{p}_{w.x} \\ \dot{p}_{w.y} \\ \dot{p}_{w.z} \end{bmatrix} = \begin{bmatrix} r_w \cdot \dot{\theta}_{g.av} \cdot \cos(\phi_y) \\ r_w \cdot \dot{\theta}_{g.av} \cdot \sin(\phi_y) \\ 0 \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} p_{w.x} \\ p_{w.y} \\ p_{w.z} \end{bmatrix} = \begin{bmatrix} \int \dot{p}_{w.x} \cdot \mathrm{d}t & + & p_{w.x}(0) \\ \int \dot{p}_{w.y} \cdot \mathrm{d}t & + & p_{w.y}(0) \\ \int \dot{p}_{w.z} \cdot \mathrm{d}t & + & p_{w.z}(0) \end{bmatrix} \tag{3.3}$$

$$\begin{bmatrix} p_{wl.x} \\ p_{wl.y} \\ p_{wl.z} \end{bmatrix} = \begin{bmatrix} p_{w.x} & - & \frac{l_{b.w}}{2} \cdot \sin(\phi_y) \\ p_{w.x} & + & \frac{l_{b.w}}{2} \cdot \cos(\phi_y) \\ & p_{w.z} & \end{bmatrix} \qquad \begin{bmatrix} p_{wr.x} \\ p_{wr.y} \\ p_{wr.z} \end{bmatrix} = \begin{bmatrix} p_{w.x} & + & \frac{l_{b.w}}{2} \cdot \sin(\phi_y) \\ p_{w.x} & - & \frac{l_{b.w}}{2} \cdot \cos(\phi_y) \\ & p_{w.z} & \end{bmatrix} \tag{3.4}$$

$$\begin{bmatrix} p_{b.x} \\ p_{b.y} \\ p_{b.z} \end{bmatrix} = \begin{bmatrix} p_{w.x} & + & l_{b.c2a} \cdot \sin(\phi_x) \cdot \cos(\phi_y) \\ p_{w.y} & + & l_{b.c2a} \cdot \sin(\phi_x) \cdot \sin(\phi_y) \\ p_{w.z} & + & l_{b.c2a} \cdot \cos(\phi_x) \end{bmatrix} \tag{3.5}$$

Typically, initial conditions are assumed to be as follows:

$$\begin{bmatrix} p_{w.x}(0) \\ p_{w.y}(0) \\ p_{w.z}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ r_w \end{bmatrix} \tag{3.6}$$

3.2. Differential Equations

After creating a nonlinear model using the Lagrangian method, and then linearizing that model, Yamamoto [1] provides the differential equations (3.7) and (3.18), [*and their abbreviated term definitions*].

3.2.1. Wheel Angular Position θ and Body Pitch ϕ_x

Equation (3.7) corresponds to wheel angular position θ and body pitch ϕ_x.

$$\mathbf{K}_{1.\ddot{x}} \cdot \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.\dot{x}} \cdot \begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.x} \cdot \begin{bmatrix} \theta \\ \phi_x \end{bmatrix} = \mathbf{K}_{1.v} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \tag{3.7}$$

$$\mathbf{K}_{1.\ddot{x}} = \begin{bmatrix} +k_{1.1} & +k_{1.2} \\ +k_{1.2} & +k_{1.3} \end{bmatrix} \tag{3.8}$$

$$\mathbf{K}_{1.\dot{x}} = \begin{bmatrix} +k_{1.4} & -k_{1.4} \\ -k_{1.4} & +k_{1.4} \end{bmatrix} \tag{3.9}$$

$$\mathbf{K}_{1.x} = \begin{bmatrix} 0 & 0 \\ 0 & +k_{1.5} \end{bmatrix} \tag{3.10}$$

$$\mathbf{K}_{1.v} = \begin{bmatrix} +k_{1.6} & +k_{1.6} \\ -k_{1.6} & -k_{1.6} \end{bmatrix} \tag{3.11}$$

$$k_{1.1} = \Big(2 \cdot m_w + m_b\Big) \cdot r_w + J_w \tag{3.12}$$

$$k_{1.2} = m_b \cdot r_w \cdot l_{b.c2a} \tag{3.13}$$

$$k_{1.3} = m_b \cdot l_{b.c2a}^2 + J_{b.\phi_x} \tag{3.14}$$

$$k_{1.4} = 2 \cdot \left(\frac{k_{mtr.T} \cdot k_{mtr.bEMF}}{R_{mtr}} + k_{fr.m2w}\right) \tag{3.15}$$

$$k_{1.5} = -m_b \cdot a_g \cdot l_{b.c2a} \tag{3.16}$$

$$k_{1.6} = \frac{k_{mtr.T}}{R_{mtr}} \tag{3.17}$$

3.2.2. Body Yaw ϕ_y

Equation (3.18) corresponds to body yaw ϕ_y.

$$k_{2.\ddot{x}} \cdot \left[\ddot{\phi}_y\right] + k_{2.\dot{x}} \cdot \left[\dot{\phi}_y\right] = k_{2.v} \cdot \left[v_{mtr.r} - v_{mtr.l}\right] \tag{3.18}$$

$$k_{2.0} = \frac{l_{b.w}}{r_w} \tag{3.19}$$

$$k_{2.\ddot{x}} = \frac{1}{2} \cdot m_w \cdot l_{b.w}^2 + \frac{1}{2} \cdot k_{2.0}^2 \cdot J_{b.\phi_y} \tag{3.20}$$

$$k_{2.\dot{x}} = \frac{1}{2} \cdot k_{2.0}^2 \cdot k_{1.4} \tag{3.21}$$

$$k_{2.v} = \frac{1}{2} \cdot k_{2.0} \cdot k_{1.6} \tag{3.22}$$

3.3. State-Space Representation

The general form of state-space representation is exhibited in Equation (3.23).

$$\begin{array}{rcl} \underset{nx1}{\dot{\mathbf{x}}} & = & \underset{nxn}{\mathbf{A}} \cdot \underset{nx1}{\mathbf{x}} + \underset{nxp}{\mathbf{B}} \cdot \underset{px1}{\mathbf{u}} \\ \underset{mx1}{\mathbf{y}} & = & \underset{mxn}{\mathbf{C}} \cdot \underset{nx1}{\mathbf{x}} + \underset{mxp}{\mathbf{D}} \cdot \underset{px1}{\mathbf{u}} \end{array} \tag{3.23}$$

The designated x states and p inputs are exhibited in Equations (3.24) - (3.25).

$$\underset{nx1}{\mathbf{x}} = \begin{bmatrix} \theta \\ \phi_x \\ \dot{\theta} \\ \dot{\phi_x} \\ \phi_y \\ \dot{\phi_y} \end{bmatrix} \tag{3.24}$$

$$\underset{px1}{\mathbf{u}} = \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \tag{3.25}$$

The derivation of indices for the system matrices $\mathbf{A}$ and $\mathbf{B}$ which are nonintuitive are derived from Equations (3.7) - (3.18). in Equations (3.26) - (3.27).

$$\begin{aligned}
\mathbf{K}_{1.\ddot{x}} \cdot \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.\dot{x}} \cdot \begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.x} \cdot \begin{bmatrix} \theta \\ \phi_x \end{bmatrix} &= \mathbf{K}_{1.v} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \\
\begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix} = \underbrace{-\mathbf{K}_{1.\ddot{x}}^{-1} \cdot \mathbf{K}_{1.\dot{x}}}_{\mathbf{A}_1} \cdot \begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix} + \underbrace{-\mathbf{K}_{1.\ddot{x}}^{-1} \cdot \mathbf{K}_{1.x}}_{\mathbf{A}_0} \cdot \begin{bmatrix} \theta \\ \phi_x \end{bmatrix} &+ \underbrace{\mathbf{K}_{1.\ddot{x}}^{-1} \cdot \mathbf{K}_{1.v}}_{\mathbf{B}_1} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix}
\end{aligned} \tag{3.26}$$

$$\begin{aligned}
k_{2.\ddot{x}} \cdot \begin{bmatrix} \ddot{\phi}_y \end{bmatrix} + k_{2.\dot{x}} \cdot \begin{bmatrix} \dot{\phi}_y \end{bmatrix} &= k_{2.v} \cdot \begin{bmatrix} v_{mtr.r} - v_{mtr.l} \end{bmatrix} \\
\begin{bmatrix} \ddot{\phi}_y \end{bmatrix} = \underbrace{-k_{2.\ddot{x}}^{-1} \cdot k_{2.\dot{x}}}_{A_2} \cdot \begin{bmatrix} \dot{\phi}_y \end{bmatrix} &+ \underbrace{k_{2.\ddot{x}}^{-1} \cdot k_{2.v}}_{B_2} \cdot \begin{bmatrix} v_{mtr.r} - v_{mtr.l} \end{bmatrix}
\end{aligned} \tag{3.27}$$

Note that $K_{1.\ddot{x}}$ must be invertible to perform the second step in in Equation (3.26). The derivation for matrix invertibility and the proof that $K_{1.\ddot{x}}$ is nonsingular [*and is therefore invertible*], are exhibited in Equations (3.28) - (3.31).

$$\underset{2x2}{\mathbf{X}} = \begin{bmatrix} +X_{(1,1)} & +X_{(1,2)} \\ +X_{(2,1)} & +X_{(2,2)} \end{bmatrix} \tag{3.28}$$

$$\mathbf{X}^{-1} = \frac{1}{\det(\mathbf{X})} \cdot \text{adj}(\mathbf{X}) = \frac{1}{X_{(1,1)} \cdot X_{(2,2)} - X_{(1,2)} \cdot X_{(2,1)}} \cdot \begin{bmatrix} +X_{(2,2)} & -X_{(2,1)} \\ -X_{(1,2)} & +X_{(1,1)} \end{bmatrix} \tag{3.29}$$

$$\det(\mathbf{X}) \neq 0 \tag{3.30}$$

$$\det(\mathbf{K}_{1.\ddot{x}}) = k_{1.1} \cdot k_{1.3} - k_{1.2} \cdot k_{1.2} \neq 0 \tag{3.31}$$

The **A** matrix and the state vector **x** are exhibited in Equation (3.32).

$$\underset{nxn}{\mathbf{A}} \cdot \underset{nx1}{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \mathbf{A}_{0\ (1,1)} & \mathbf{A}_{0\ (1,2)} & \mathbf{A}_{1\ (1,1)} & \mathbf{A}_{1\ (1,2)} & 0 & 0 \\ \mathbf{A}_{0\ (2,1)} & \mathbf{A}_{0\ (2,2)} & \mathbf{A}_{1\ (2,1)} & \mathbf{A}_{1\ (2,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & A_2 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \phi_x \\ \dot{\theta} \\ \dot{\phi_x} \\ \phi_y \\ \dot{\phi_y} \end{bmatrix} \tag{3.32}$$

The **B** matrix and the input vector **u** are exhibited in Equation (3.33).

$$\underset{nxp}{\mathbf{B}} \cdot \underset{px1}{\mathbf{u}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \mathbf{B_{1\ (1,1)}} & \mathbf{B_{1\ (1,2)}} \\ \mathbf{B_{1\ (2,1)}} & \mathbf{B_{1\ (2,2)}} \\ 0 & 0 \\ -B_2 & +B_2 \end{bmatrix} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \tag{3.33}$$

The **C** matrix and the state vector **x** are exhibited in Equation (3.34).

$$\underset{mxn}{\mathbf{C}} \cdot \underset{nx1}{\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \phi_x \\ \dot{\theta} \\ \dot{\phi_x} \\ \phi_y \\ \dot{\phi_y} \end{bmatrix} \tag{3.34}$$

The **D** matrix and the input vector **u** are exhibited in Equation (3.35).

$$\underset{mxp}{\mathbf{D}} \cdot \underset{px1}{\mathbf{u}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \tag{3.35}$$

3.4. Calculation of Nonintuitive Parameters

Most of the hardware-equivalent dynamic model parameter values (*with respect to the MinSeg hardware*) were publicly available [5, 16], or were intuitive to obtain [*Example:* l_h, l_w, l_d].

Methods to determine those parameters which were not considered easily obtained are defined in the following sections.

3.4.1. Moment of Inertia: Body: X-axis (Pitch) J_{ϕ_x}

The moment of inertia of the body with respect to pitch, is assumed to be sufficiently equivalent to the moment of inertia of "an ideal thin rectangular plate with length l_h, width $l_w = 0$, an axis of rotation at one end of the plate".

This relation is exhibited in Equation (3.36).

$$J_{\phi_x} = \frac{m_b \cdot l_{b.c2a}^2}{3} \tag{3.36}$$

3.4.2. Moment of Inertia: Body: Y-axis (Yaw) J_{ϕ_y}

The moment of inertia of the body with respect to pitch, is assumed to be sufficiently equivalent to the moment of inertia of "an ideal thin rectangular plate with length l_h, width $l_w = 0$, an axis of rotation at one end of the plate".

This relation is exhibited in Equation (3.37).

$$J_{\phi_y} = \frac{m_b \cdot (l_{b.w}^2 + l_{b.d}^2)}{12} \tag{3.37}$$

3.4.3. Length From Body Center of Mass to Body Axis of Rotation $l_{b.c2a}$

The length from the body center of mass to the body axis of rotation $l_{b.c2a}$ may be determined using more than one method.

3.4.3.1. Yamamoto Method

As seen in Figure 3.1 [*on page* 36], Yamamoto [1] assumes that the geometries of the wheels and the body are uniform. He also assumes that the masses of these geometries are uniform. He therefore defines length from the body center of mass to the body axis of rotation $l_{b.c2a}$, as exhibited in Equation (3.38)

$$l_{b.c2a} = \frac{l_{b.h}}{2} \tag{3.38}$$

3.4.3.2. Vaccaro Method

Since the geometries of the actual hardware are assumed to significantly deviate from the assumption of uniform mass distribution, an alternative method is instead used to calculate length from the body center of mass to the body axis of rotation $l_{b.c2a}$, as exhibited in Equation (3.38)

If the hardware is mounted at both wheel axles *along the axis which is shared by both wheel axles*, and if the hardware is given a degree of freedom to rotate about the wheel axle axis, *without rotating the actual wheel axles*, then the hardware may be lifted slightly and then released to swing freely like a pendulum along that axis.

Allowing the hardware to freely swing like a pendulum along the wheel axle axis significantly simplifies the dynamic equations of motion of the hardware. Furthermore, if friction at the newly added mount coupling points is negligible, then there will not be a need to model and implement the friction into the dynamics equations.

If the hardware is freely swung like a pendulum along the wheel axle axis as described above, then the relations exhibited in Equations (3.39) - (3.40) become true.

$$\theta = \phi_x \tag{3.39}$$

$$\mathbf{u} = \mathbf{0} \tag{3.40}$$

The effects of these changes are exhibited in Equation (3.42) [*on page* 48]. This results in two relations, which are exhibited in Equation (3.41).

$$\begin{aligned} \ddot{\phi}_x \quad &+ \quad 0 \quad = \quad 0 \\ \ddot{\phi}_x \quad &+ \quad \underbrace{\frac{k_{1.5}}{k_{1.2} + k_{1.3}}}_{k_\omega} \cdot \phi_x \quad = \quad 0 \qquad \Leftarrow \end{aligned} \tag{3.41}$$

Of the two resulting relations in Equation (3.41), the former cannot be true while the hardware is in motion; thus, the latter is selected, as depicted on the right with a left-facing arrow.

$$
\begin{aligned}
&\mathbf{K}_{1.\ddot{x}} \cdot \underbrace{\begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix}} + \mathbf{K}_{1.\dot{x}} \cdot \underbrace{\begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix}} + \mathbf{K}_{1.x} \cdot \underbrace{\begin{bmatrix} \theta \\ \phi_x \end{bmatrix}} = \mathbf{K}_{1.v} \cdot \underbrace{\begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix}} \\
&\underbrace{\mathbf{K}_{1.\ddot{x}}} \cdot \begin{bmatrix} \ddot{\phi}_x \\ \ddot{\phi}_x \end{bmatrix} + \underbrace{\mathbf{K}_{1.\dot{x}}} \cdot \begin{bmatrix} \dot{\phi}_x \\ \dot{\phi}_x \end{bmatrix} + \underbrace{\mathbf{K}_{1.x}} \cdot \begin{bmatrix} \phi_x \\ \phi_x \end{bmatrix} = \underbrace{\mathbf{K}_{1.v} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\oslash} \\
&\underbrace{\begin{bmatrix} k_{1.1} & k_{1.2} \\ k_{1.2} & k_{1.3} \end{bmatrix} \cdot \begin{bmatrix} \ddot{\phi}_x \\ \ddot{\phi}_x \end{bmatrix}} + \underbrace{2 \cdot k_{1.4} \cdot \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi}_x \\ \dot{\phi}_x \end{bmatrix}}_{\oslash} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & k_{1.5} \end{bmatrix} \cdot \begin{bmatrix} \phi_x \\ \phi_x \end{bmatrix}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
&\begin{bmatrix} \left(k_{1.1} + k_{1.2}\right) \cdot \ddot{\phi}_x \\ \left(k_{1.2} + k_{1.3}\right) \cdot \ddot{\phi}_x \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ k_{1.5} \cdot \phi_x \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}
\tag{3.42}
$$

The coefficient term, abbreviated as k_w, is expanded in Equation (3.43). It may be expanded further with the use of Equation (3.36), as exhibited in Equation (3.44).

$$k_\omega = \frac{k_{1.5}}{k_{1.2} + k_{1.3}} = \frac{-m_b \cdot a_g \cdot l_{b.c2a}}{\left(m_b \cdot r_w \cdot l_{b.c2a}\right) + \left(m_b \cdot l_{b.c2a}^2 + J_{b.\phi_x}\right)} \tag{3.43}$$

$$k_\omega = \frac{-m_b \cdot l_{b.c2a} \cdot a_g}{m_b \cdot l_{b.c2a} \cdot r_w + m_b \cdot l_{b.c2a}^2 + \left(m_b \cdot l_{b.c2a}^2 \cdot \frac{1}{3}\right)} = \frac{-a_g}{r_w + l_{b.c2a} \cdot \left(1 + \frac{1}{3}\right)} \tag{3.44}$$

Harmonic Oscillator

Notably, the selected relation in Equation (3.41) form-matches the equation for a harmonic oscillator [2, p. 119 - 120, 122 – 123], as is exhibited in Equation (3.45).

$$\begin{aligned} \ddot{y} + \omega^2 \cdot y &= \omega^2 \cdot u \\ \ddot{\phi}_x + k_\omega \cdot \phi_x &= k_\omega \cdot 0 \end{aligned} \tag{3.45}$$

This allows for the relation of the abbreviated term representing the system dynamics, k_w, to the natural angular frequency of the hardware [*a pendulum*] ω_p, as is exhibited in Equation (3.46).

$$\omega_p^2 = k_\omega = \frac{-a_g}{r_w + l_{b.c2a} \cdot \frac{4}{3}} \tag{3.46}$$

This proves significant since ω_p represents the angular frequency of the pendulum, which is a measurable value, and since k_ω includes the desired unknown term $l_{b.c2a}$. [*All other terms are known*]. The relation may rewritten to solve for length from the body center of mass to the body axis of rotation $l_{b.c2a}$, as is exhibited as Equation (3.47).

$$l_{b.c2a} = -\frac{3}{4} \cdot \left(\frac{a_g}{\omega_p^2} + r_w \right) = -\frac{3}{4} \cdot \left(\frac{a_g}{\left(2 \cdot \pi \cdot f_p\right)^2} + r_w \right) \tag{3.47}$$

3.5. Bibliography

[1] Y. Yamamoto. (May 1, 2009). "Nxtway-gs (self-balancing two-wheeled robot) controller design," [Online]. Available: `https://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs--self-balancing-two-wheeled-robot--controller-design` (visited on 07/03/2017).

[2] R. J. Vaccaro, *Digital Control: A State-space Approach*, ser. Mcgraw Hill Series in Electrical and Computer Engineering. Mcgraw-Hill College, Jan. 1995, ISBN: 978-0070667815.

[5] B. Howard and L. Bushnell, "Enhancing linear system theory curriculum with an inverted pendulum robot," in *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, IEEE, Hangzhou, China, Oct. 2015. DOI: `10.1109/CSMA.2015.63`.

[16] P. "Philo" Hurbain. (May 15, 2017). "Nxt motor internals," [Online]. Available: `http://www.philohome.com/nxtmotor/nxtmotor.htm` (visited on 07/03/2017).

[19] M. D. Peltier, "Trajectory control of a two-wheeled robot," M.S. thesis, University of Rhode Island (URI): Department of Electrical Engineering, Jan. 2012.

CHAPTER 4.

Control Design

4.1. Additional Dynamics

Additional dynamics may be incorporated into a state feedback regulating system in order to beneficially alter the response of the system in various respects.

4.1.1. Background

A state feedback regulating system is depicted in Figure 4.1. It contains the hardware plant as well as the inverted system feedback gains.

The *additional dynamics* are added in Figure 4.2. In the figure, the output vector of the plant is demuxed into its individual output components such that certain outputs may additionally be used as inputs to the additional dynamics state-space representation. This is represented using flags; connections exist between flags with equivalent labels. The outputs of the plant as well as the outputs of the additional dynamics are then muxed to form an output vector representing the output of a larger system.

Thus, the larger system [*which includes the plant and the additional dynamics*] may temporarily be considered as new plant, as depicted in Figure 4.3. It may therefore be expressed as a single state-space representation containing both systems. Thus, state-feedback regulation techniques may be used to control the system; however, the system response will now include any benefits which the additional dynamics provide. A representation of the larger system is depicted in Figure 4.4. Note the increase in the output vector.

From this point, the system depiction may be rearranged such that the input to the controller is on the left, while the plant outputs remain on the right. A sequential description of the process, and the figure number which corresponds to each step is provided below:

1. The feedback gains are separated into those with respect to the original plant outputs x and those with respect to the additional dynamics $x.a$. [Figure 4.5]

2. The gains are shifted such that they are forward facing. [Figure 4.6]

- In this configuration, the original plant may be separated from the components used to control it, including the additional dynamics. [Figure 4.7]

3. The plant is shifted to the right side of the system depiction. [Figure 4.8]

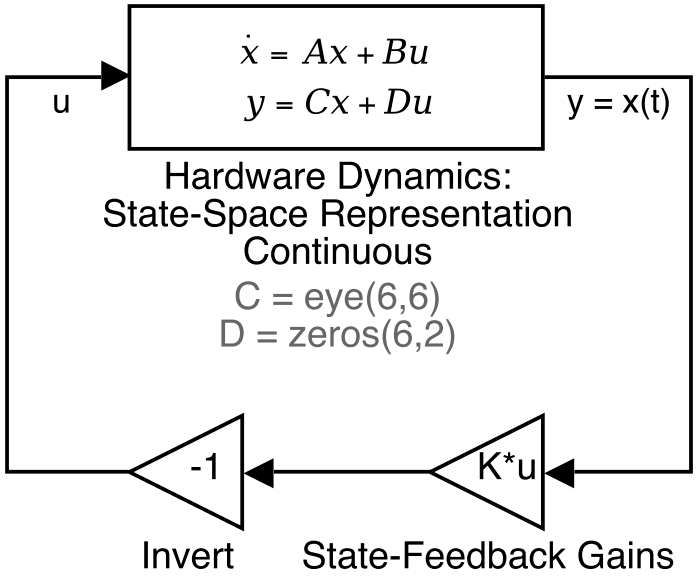

Figure 4.1.: [Additional Dynamics]: State Feedback Regulator

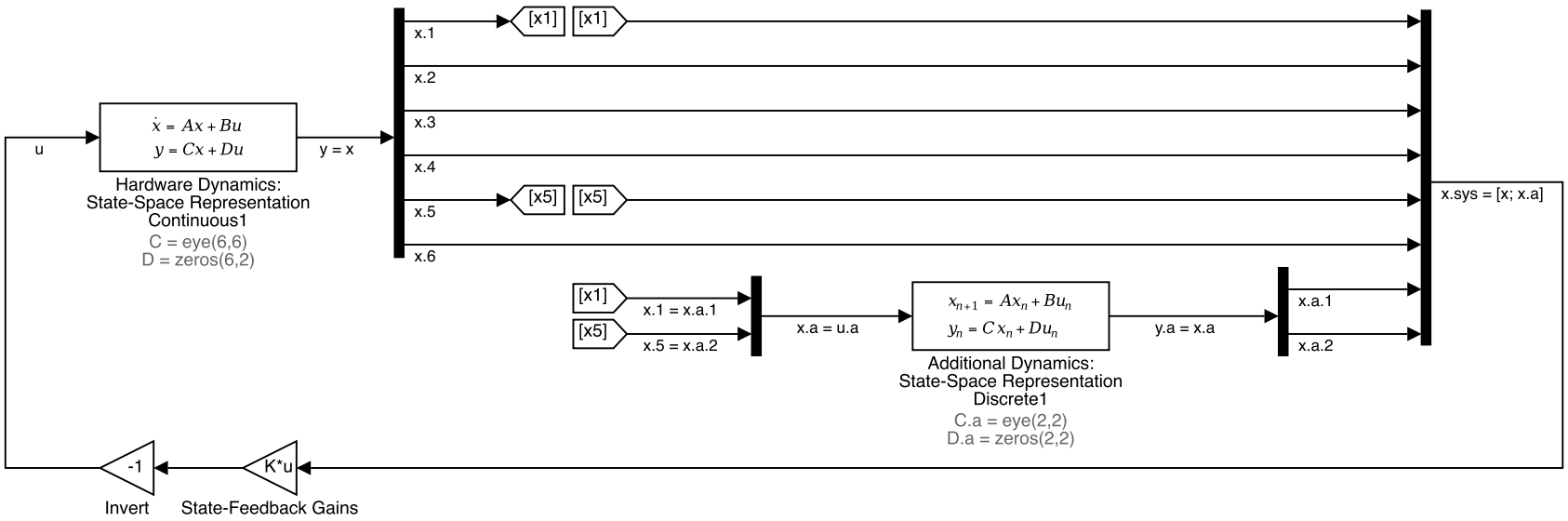

Figure 4.2.: [Additional Dynamics]: 1.0 Additional Dynamics (Design View)

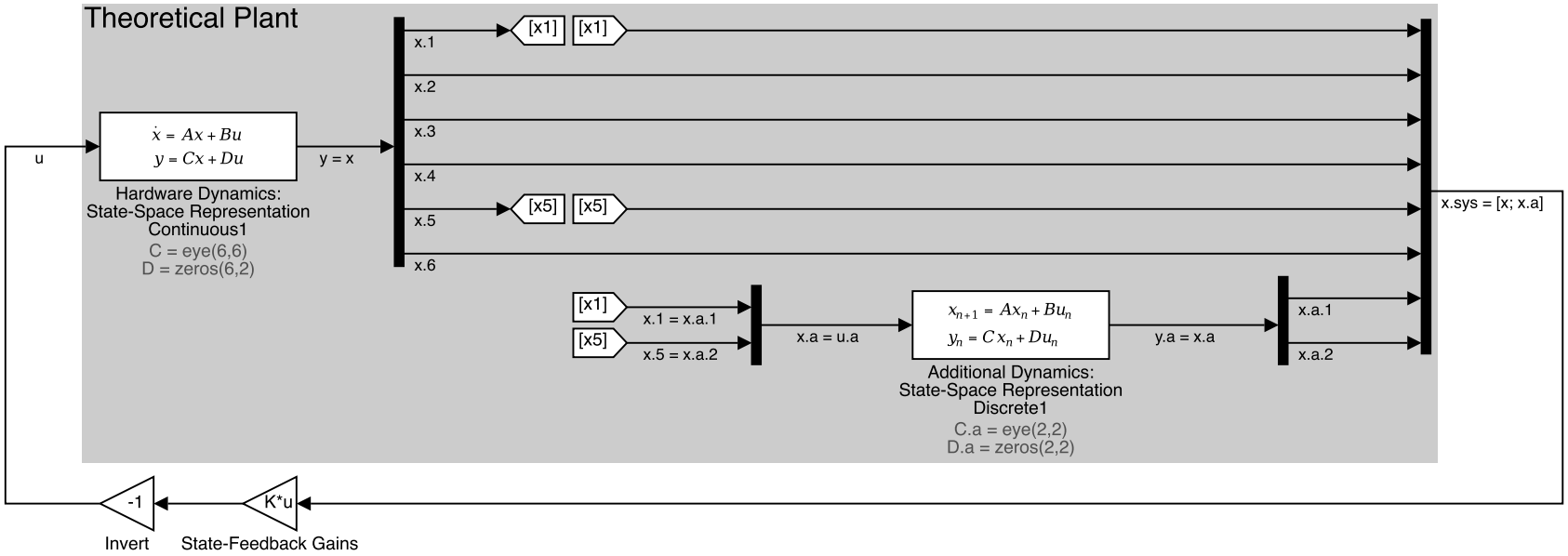

Figure 4.3.: [Additional Dynamics]: 1.1 Additional Dynamics (Design View)

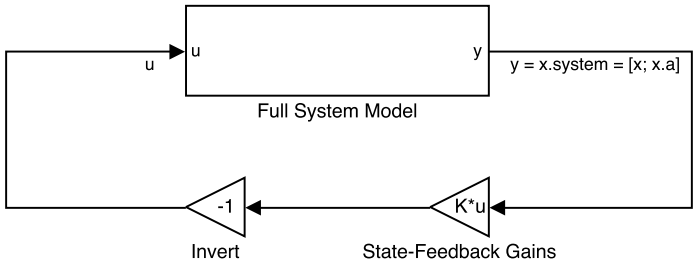

Figure 4.4.: [Additional Dynamics]: 1.2 Additional Dynamics (State Feedback Regulator View)

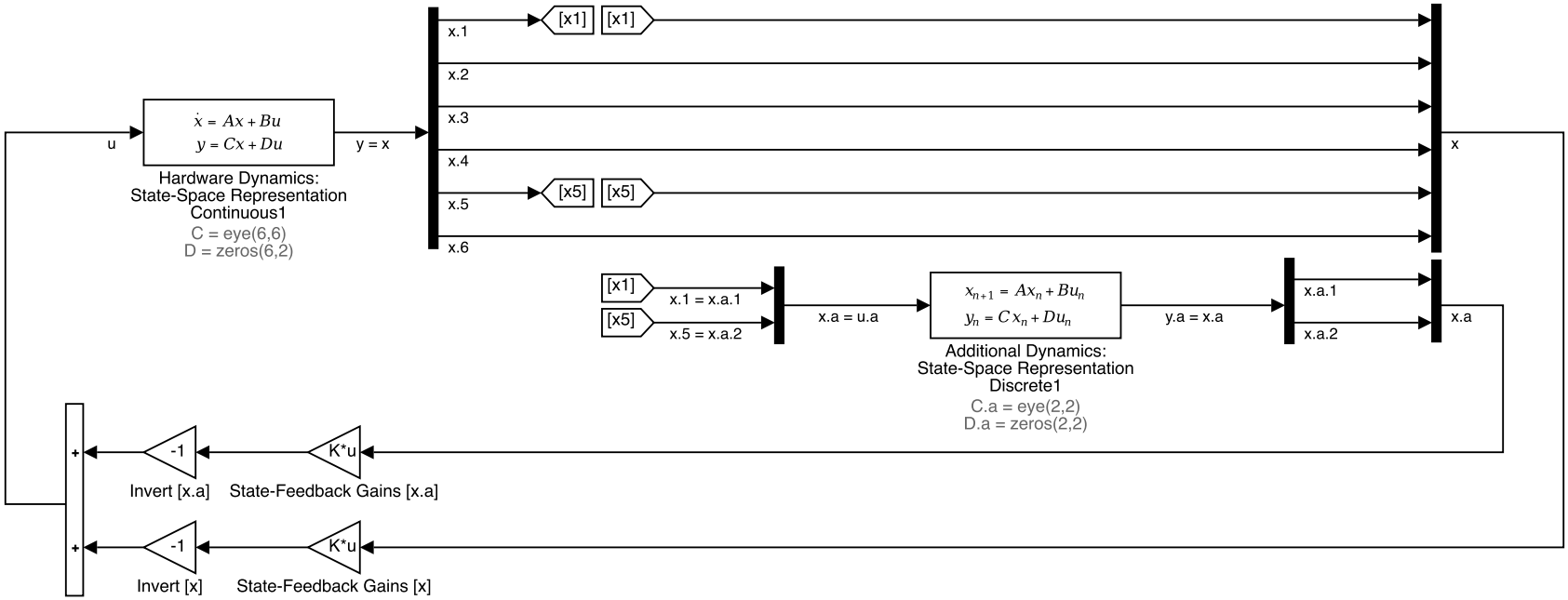

Figure 4.5.: [Additional Dynamics]: 2.0 Additional Dynamics (Split Gains)

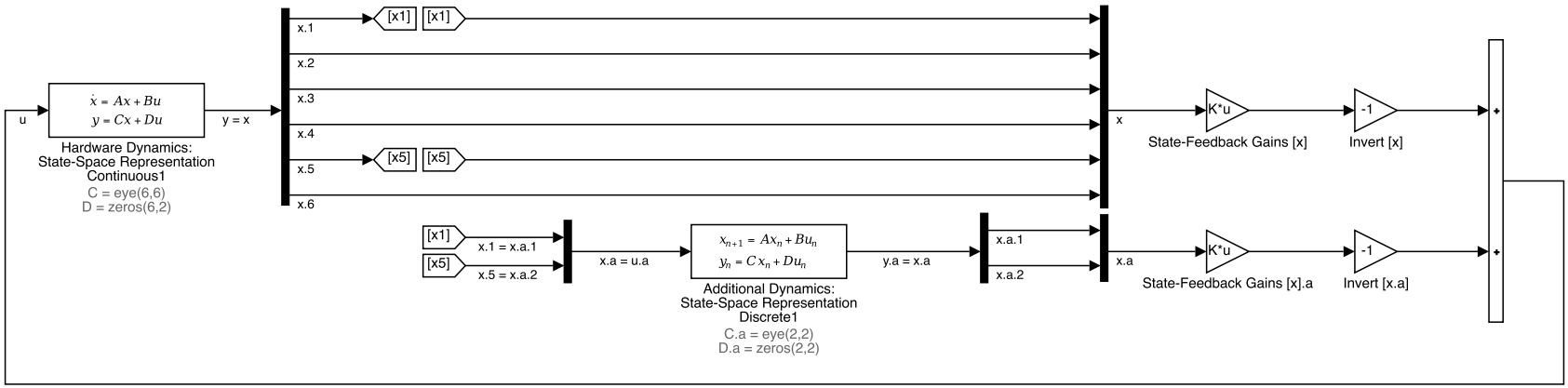

Figure 4.6.: [Additional Dynamics]: 3.0 Additional Dynamics (Linear View: Plant)

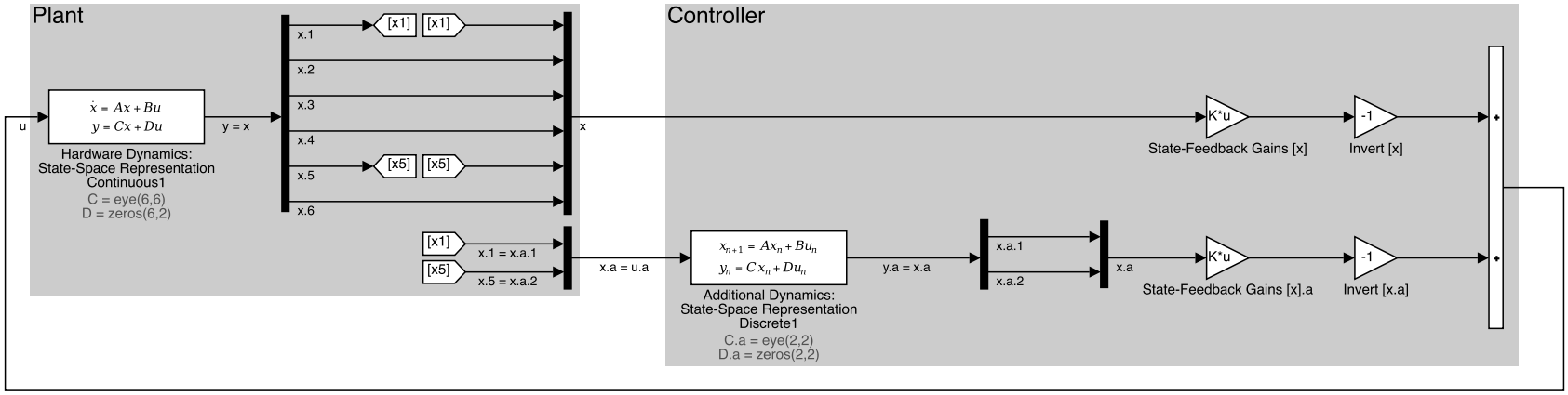

Figure 4.7.: [Additional Dynamics]: 3.1 Additional Dynamics (Linear View: Plant)

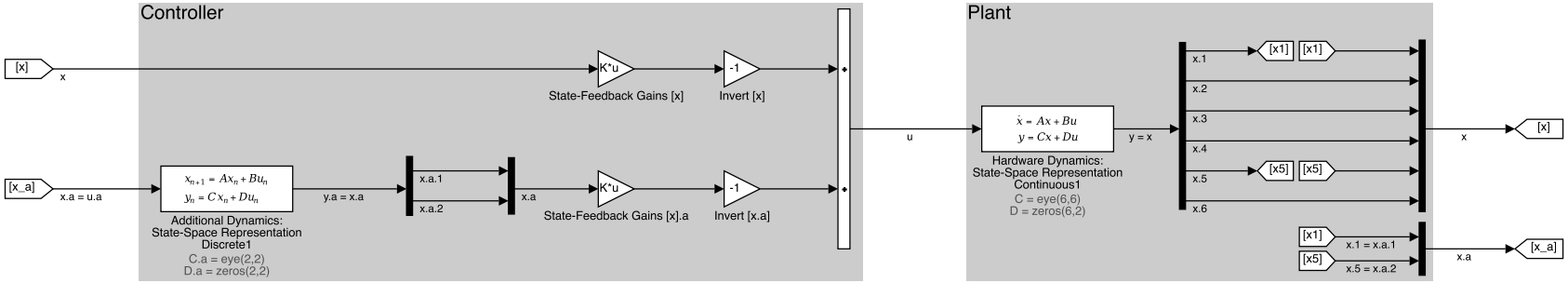

Figure 4.8.: [Additional Dynamics]: 4.0 Additional Dynamics (Linear View: Controller)

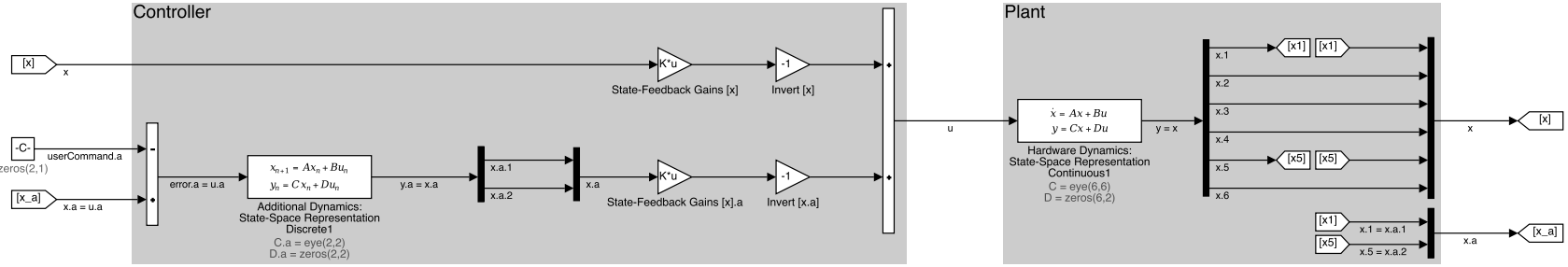

Figure 4.9.: [Additional Dynamics]: 4.0 Additional Dynamics with Reference Signal (Linear View: User)

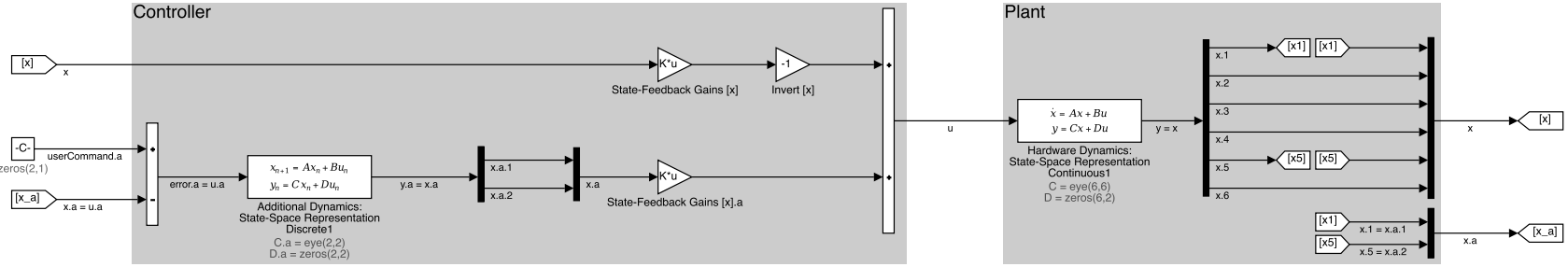

Figure 4.10.: [Additional Dynamics]: 4.0 Additional Dynamics with Reference Signal (Linear View: User)

4.1.1.1. Reference Signal

Recall that a standard state-feedback regulator simply brings its inputs, [*in this case, system states* x *and* x_a], to zero. If it is desired that a controller input be brought to a value other than zero, a reference signal may be implemented.

In these cases, rather than input the controller with a state which the controller will bring to zero, the controller is input with the difference between the state value and the reference [*desired*] value. This difference is commonly known as the error signal. Once the error signal is brought to zero for a given state, the state will be equivalent to the desired reference value.

Returning to system depiction, a reference command is implemented, as depicted in Figure 4.9.

Note that the reference signal receives the negative. Inverting the system state alters the system equation, and could cause the system to become unstable.

Despite this fact, it is sometimes more common to see the system state subtracted from the reference signal. To correctly achieve this, once the reference signal is implemented, either side of the difference equation is multiplied by -1. The negative on the input side is distributed to both inputs. The output of the difference equation is the input of the additional dynamics; thus, when the negative appears on the output side of the difference equation, a negative exists on either side of the additional dynamics equation.

Recall that all state-space representations are linear; therefore, the input and the output may be multiplied by the same value. In this case, the negative may be divided out on both sides.

These changes are depicted in Figure 4.10.

4.1.2. Tracking System

Additional dynamics may be incorporated to improve reference tracking. When implemented for this purpose, the additional dynamics are known as a tracking system.

In the case of a tracking system, an *integrator* may be implemented as the additional dynamics to track a constant reference exactly, or to track a slowly varying reference approximately.

Integrators are also able to mitigate constant disturbances. Incidentally, the MinSeg M2V3 system uses gyroscopes as body angular velocity ψ sensors. Bias is inherent in the output of a gyroscope; therefore, the use of such an integrator as a tracking system has an additional benefit: it will mitigate the effects of bias from a gyroscope output, whether directly or within terms which are derivative of the gyroscope output.

Thus, in the case of the two-wheeled robot, integrators are implemented as additional dynamics for the states representing wheel angular position θ and body angular position (yaw) ϕ_y. This establishes a tracking system, [*an augmented method of state feedback regulation*], for the system. The state-space representation of the integrator is exhibited in Equation (4.1).

$$\begin{aligned}
\underset{nx1}{\dot{\mathbf{x}}(t)} &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \underset{nx1}{\mathbf{x}(t)} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e_\theta\ (t) \\ e_{\phi_y}(t) \end{bmatrix} \\
\underset{mx1}{\mathbf{y}(t)} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \underset{nx1}{\mathbf{x}(t)} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e_\theta\ (t) \\ e_{\phi_y}(t) \end{bmatrix}
\end{aligned} \tag{4.1}$$

4.1.2.1. Discrete Additional Dynamics

Since the additional dynamics will be processed on a microcontroller, the additional dynamics will be digital; thus, a continuous-to-discrete conversion will be necessary. A digital integrator is an established case which is exhibited in Equation (4.2).

$$\begin{aligned} \underset{nx1}{\dot{\mathbf{x}}[k+1]} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \underset{nx1}{\mathbf{x}[k]} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e_\theta\ [k] \\ e_{\phi_y}[k] \end{bmatrix} \\ \underset{mx1}{\mathbf{y}[k]} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \underset{nx1}{\mathbf{x}[k]} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e_\theta\ [k] \\ e_{\phi_y}[k] \end{bmatrix} \end{aligned} \tag{4.2}$$

4.1.3. Control Gains

Once the additional dynamics are established, state feedback gains must be calculated. Multiple methods exist to calculate these gains. The most established methods involve optimization or pole-placement.

4.1.3.1. Optimal

Several optimal control techniques exist [20]. This section will focus on linear quadratic regulation techniques.

4.1.3.1.1. Implementation

In order to determine the feedback gains of the system, the state-space representation of the system, [*the plant and the additional dynamics*], is input into a discrete linear quadratic regulator gain-calculation Matlab function, *dlqr*, which outputs state-feedback gains which best minimize the quadratic cost function. The Matlab function also requires quadratic cost function matrices Q and R as inputs.

The quadratic cost matrices Q and R were determined through trial and error; however, some constraints existed. The Q and R matrices were both diagonal matrices; [*thus, all indices which are not on the diagonal are equal to zero*]. Also, the R matrix was left as an identity matrix until the Q matrix established desirable behavior. Once desirable behavior was established, the option of multiplying the R matrix by a scalar value [*greater than one*] became a consideration.

Multiplying the R matrix by a scalar value decreases the response time of the controller; however, this also decreases the peak magnitude of the control output, [*in this case, motor voltage*]. While a decreased response time is generally undesirable, the reduction of the control output can be necessary in certain circumstances. For example, the maximum permissible value for the control output, motor voltage, is limited by the nominal voltage provided by the hardware power source.

4.1.3.1.2. Results: Simulation

To demonstrate the capabilities of the device, a dynamic command is provided which attempts to move the device in the shape of an eight 8 on the ground while maintaining balance.

The specific Q and R weighting matrices which were selected to calculate the control gains are exhibited in Equation (4.3).

$$\begin{aligned} Q &= \underset{6x6}{\mathrm{I}} \cdot \begin{bmatrix} 6 & 1 & 1 & 1 & 1 & 1 & 21 & 11 \end{bmatrix}^{\mathrm{T}} \\ R &= \underset{2x2}{\mathrm{I}} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix}^{\mathrm{T}} \end{aligned} \tag{4.3}$$

Recall that Q is with respect to the states of the global system, [*plant and reference tracking dynamics*] and that R is with respect to inputs of the global system [*reference tracking input error*].

Note that the greatest weights have been applied to the reference tracking states, and also that additional weight has been provided to the wheel angular position state θ, in the cases of the plant states as well as the reference tracking states, respectively.

This results in the feedback gain matrices K exhibited in Equation (4.4).

Additionally, the device starts at a body angular position (pitch) ϕ_x of 0.03 $[rad]$. This represents the inability to start the device at a perfect angle. This causes additional transients in the initial milliseconds of operation.

Figures 4.11 - 4.17 depict the system state during its operation while completing its response to a figure-eight linear position command.

$$
\begin{aligned}
K_{plant} &= \begin{bmatrix} -50.3100392458413 & -67.5072785023266 & -6.36629920161377 & -1.55714590289198 & -31.3872481064794 & -0.0172137559799668 \\ -50.3100392458415 & -67.5072785023269 & -6.36629920161380 & -1.55714590289199 & +31.3872481064798 & +0.0172137559799669 \end{bmatrix}^{\mathrm{T}} \\
K_{referenceTracking} &= \begin{bmatrix} -2.29913264446660 & -2.07850353527147 \\ -2.29913264446662 & +2.07850353527155 \end{bmatrix}^{\mathrm{T}}
\end{aligned}
\tag{4.4}
$$

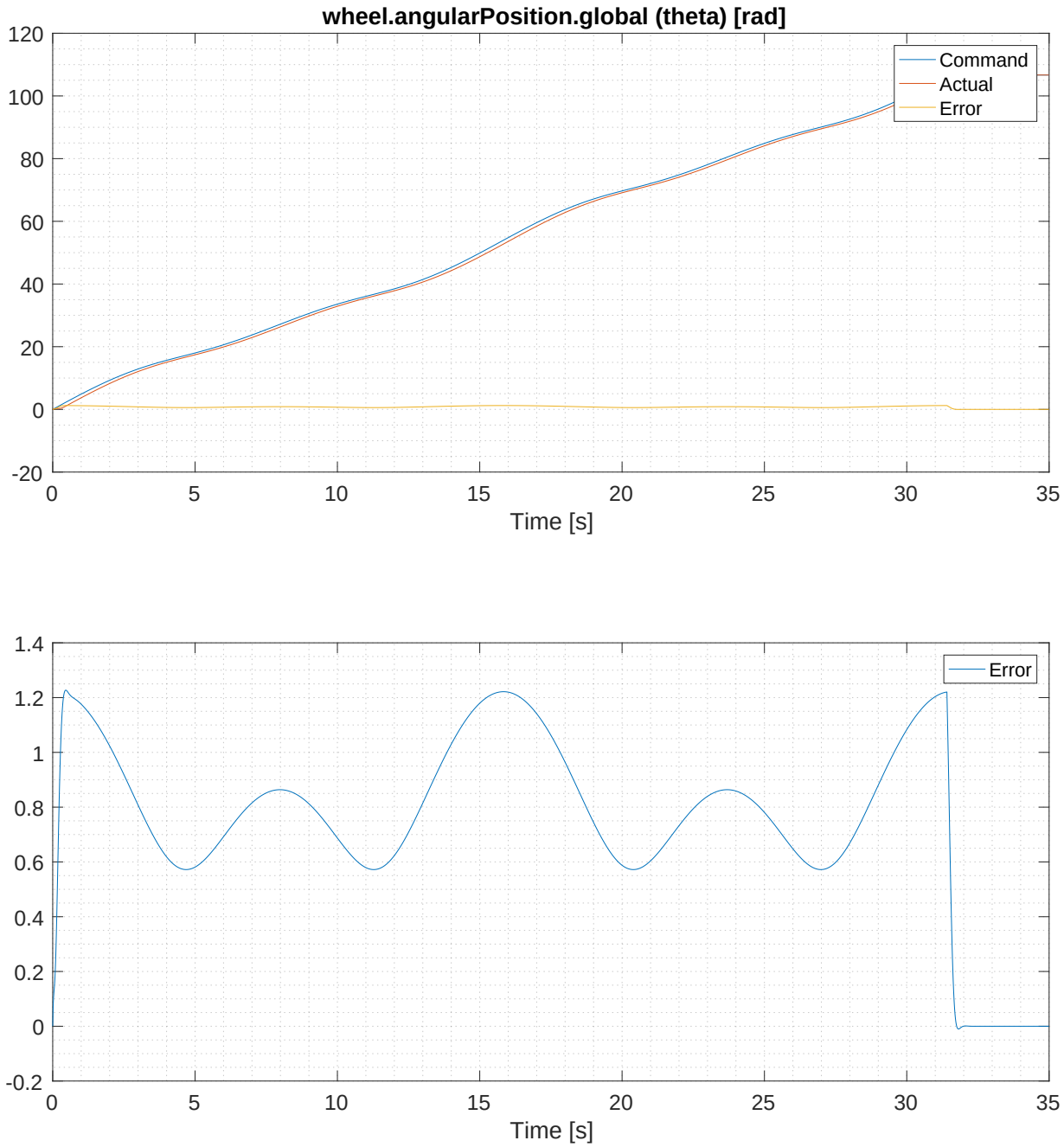

Figure 4.11.: [Control Gains: LQR]: Simulation Results: Wheel Angular Position θ

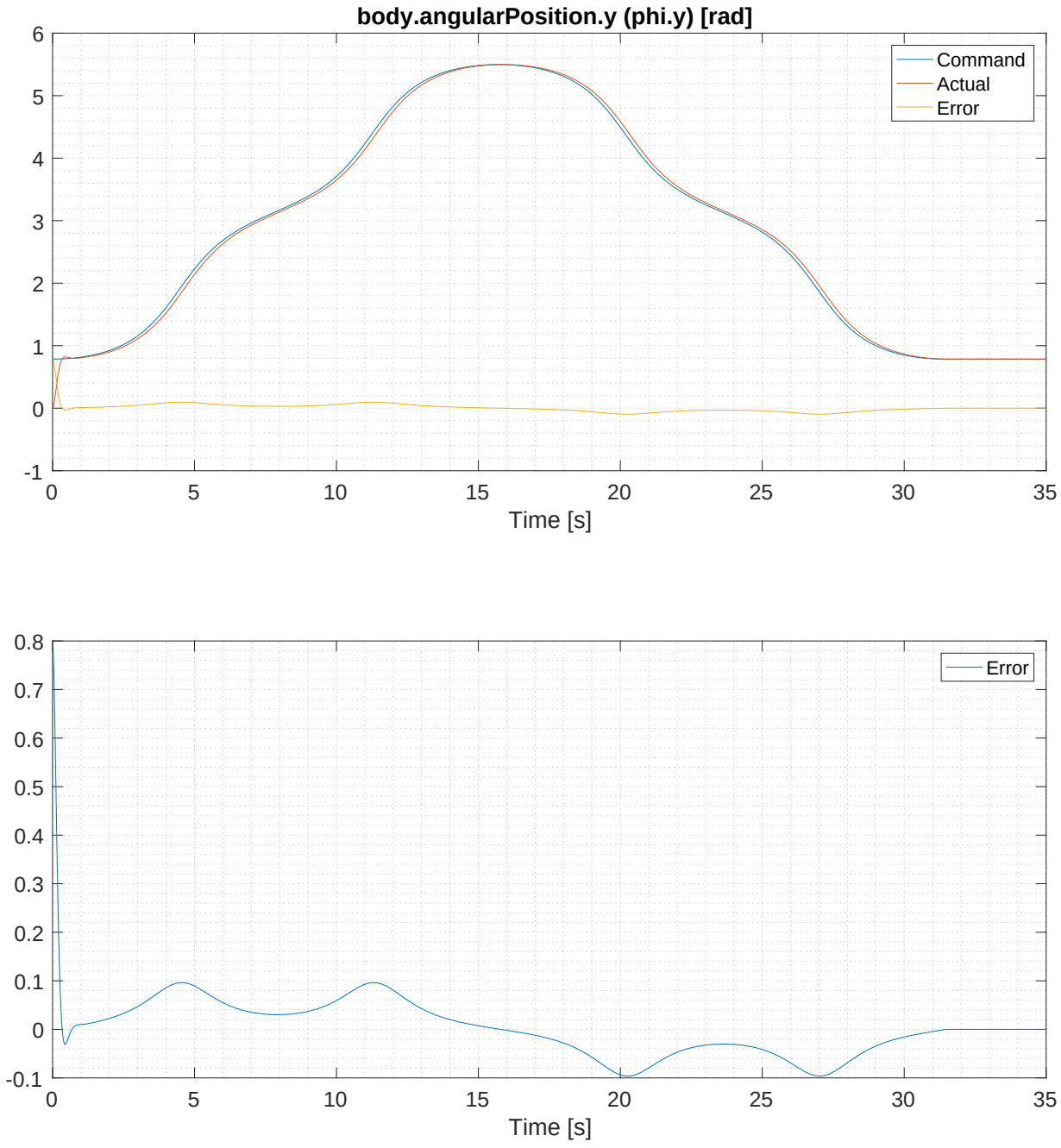

Figure 4.12.: [Control Gains: LQR]: Simulation Results: Body Angular Position ϕ_y

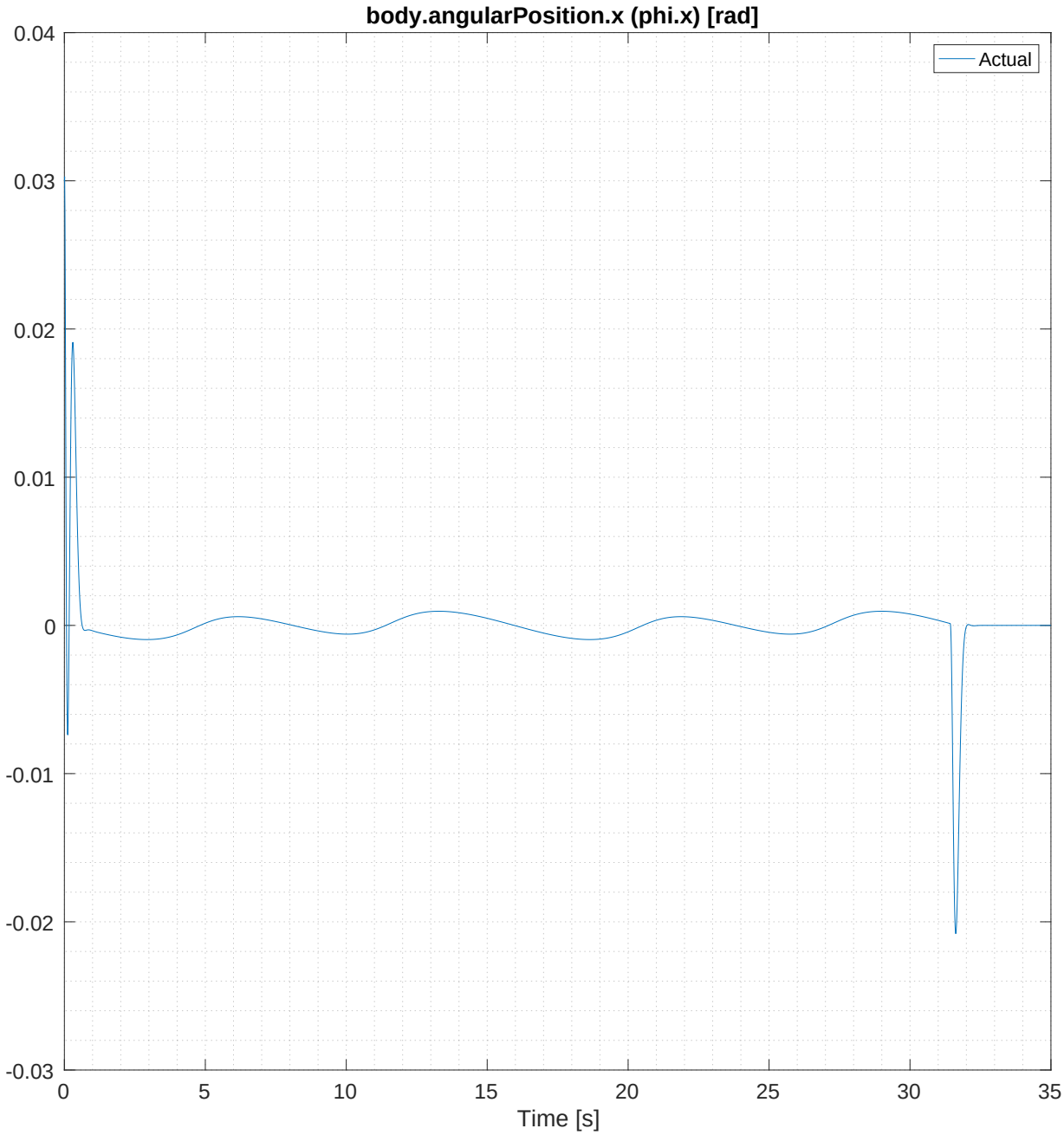

Figure 4.13.: [Control Gains: LQR]: Simulation Results: Body Angular Position ϕ_x

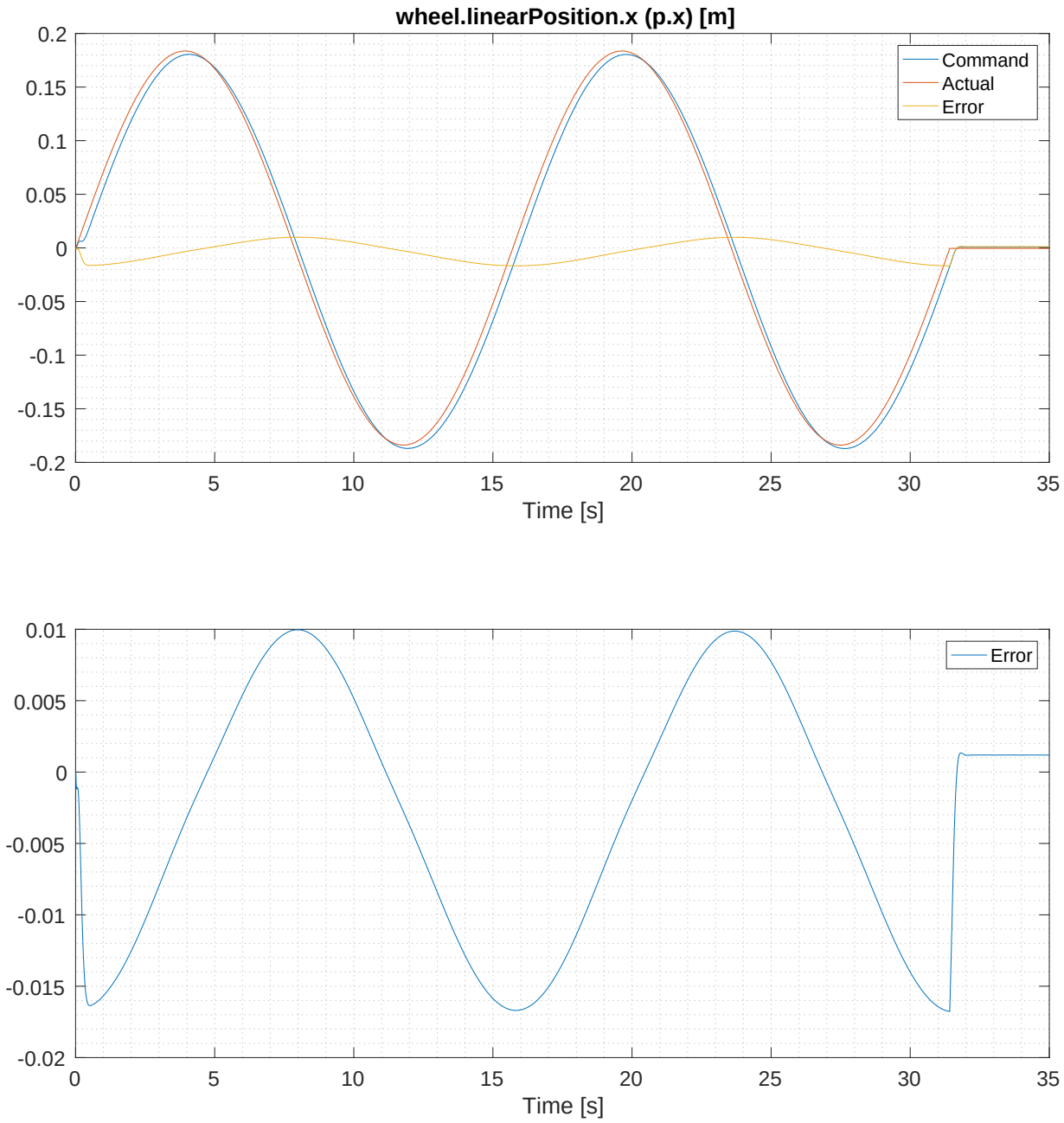

Figure 4.14.: [Control Gains: LQR]: Simulation Results: Wheel Linear Position p_x

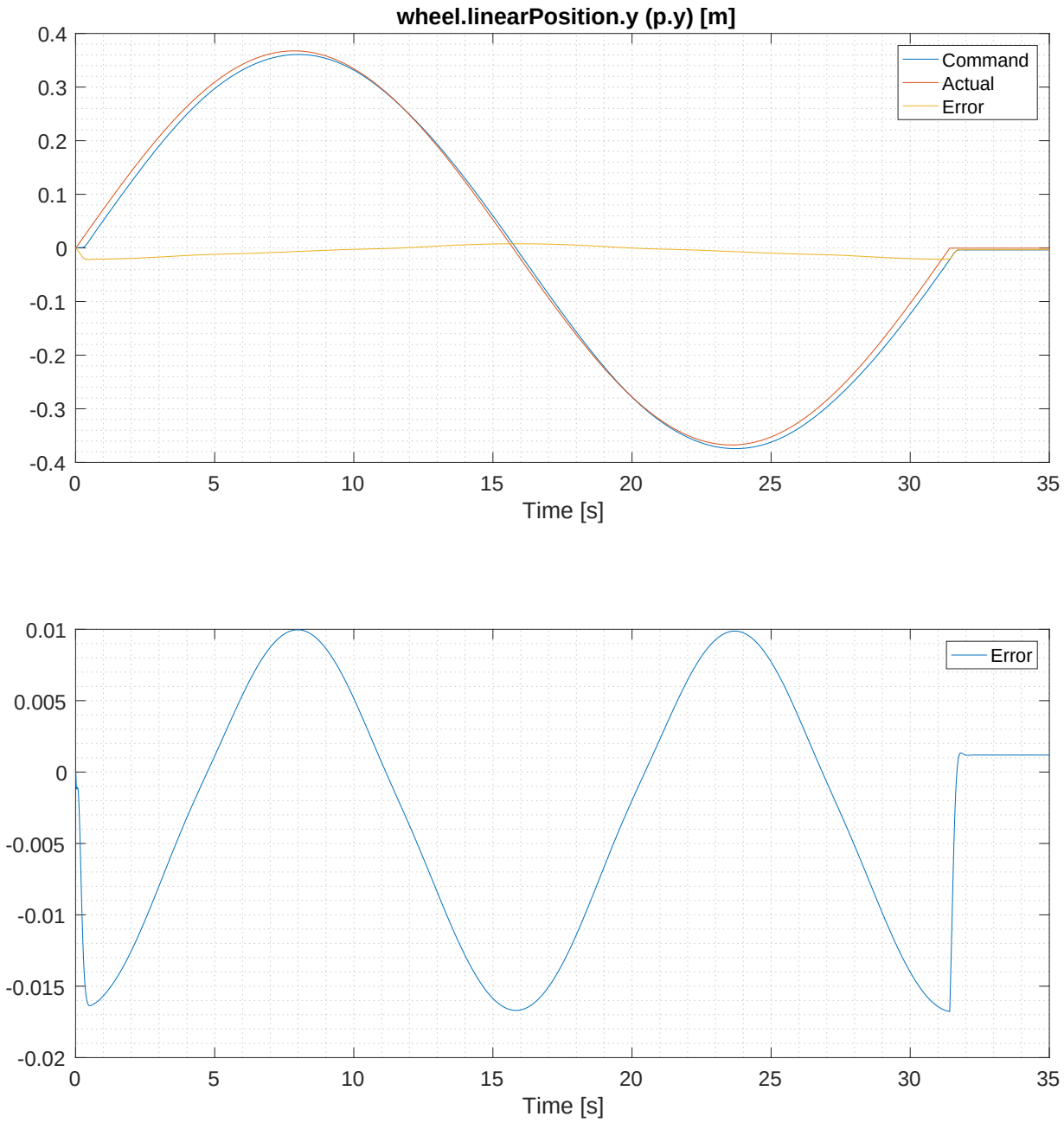

Figure 4.15.: [Control Gains: LQR]: Simulation Results: Wheel Linear Position p_y

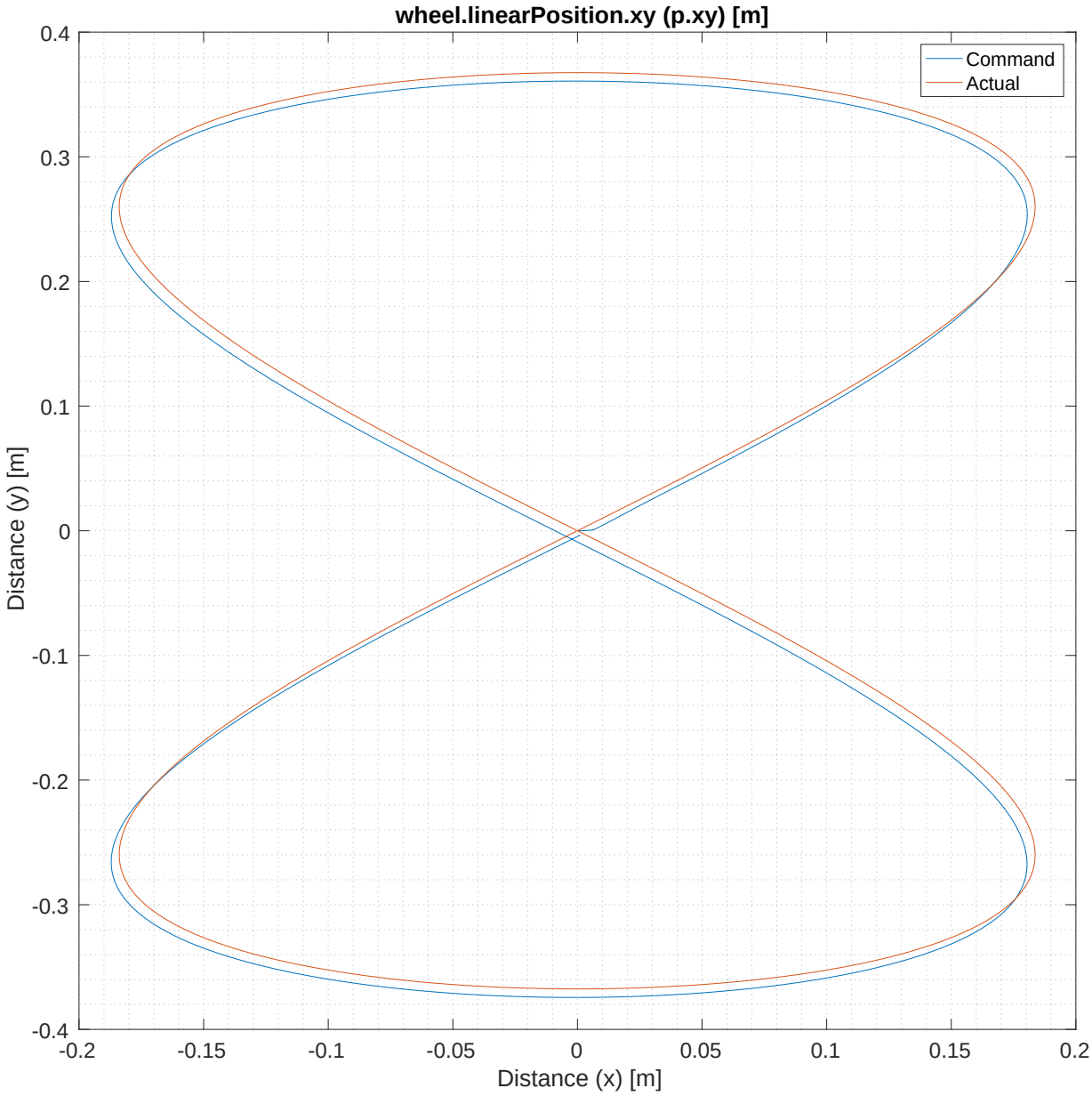

Figure 4.16.: [Control Gains: LQR]: Simulation Results: Wheel Linear Position p_{xy}

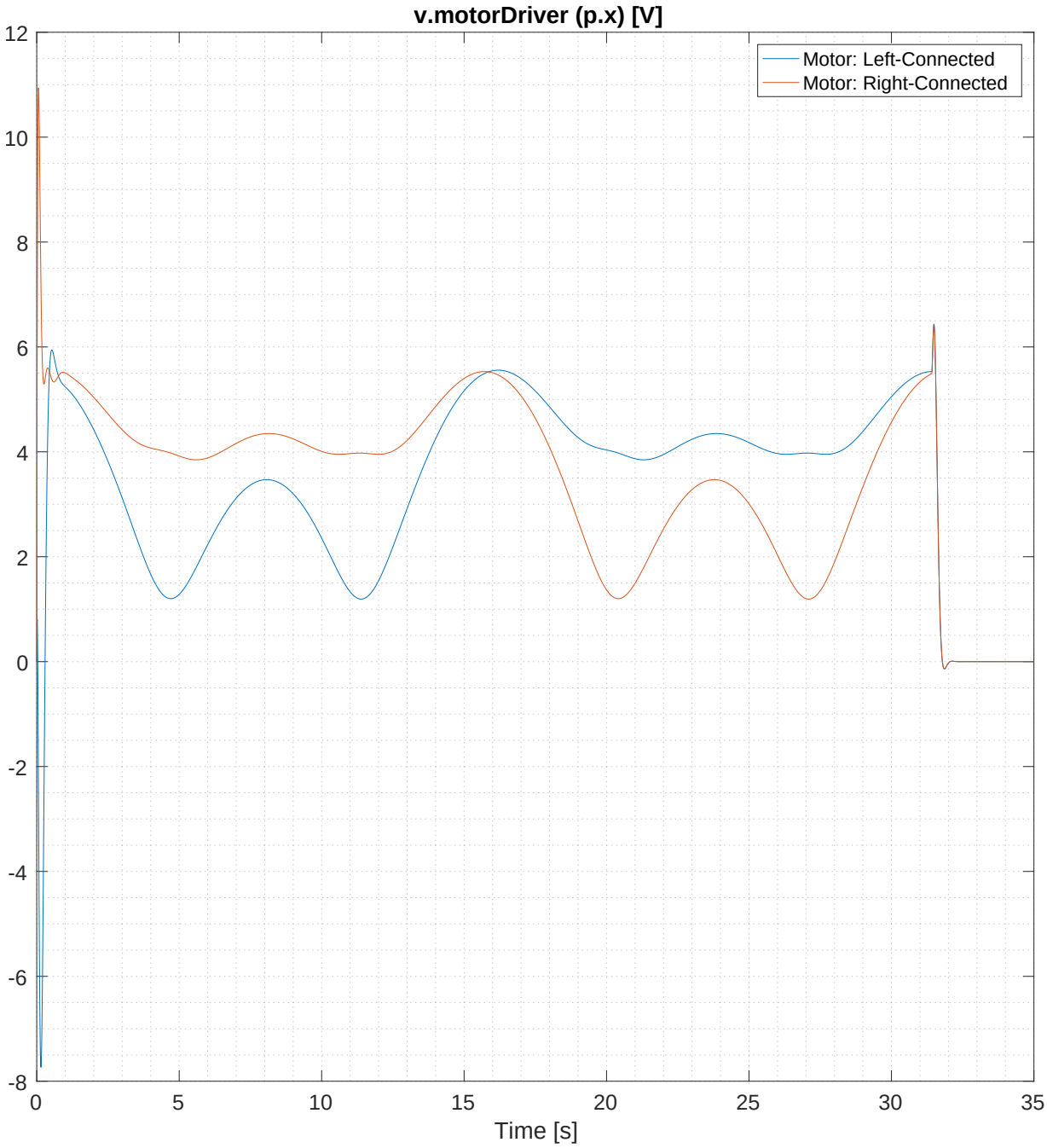

Figure 4.17.: [Control Gains: LQR]: Simulation Results: Motor Driver Commanded Voltage $v_{motorDriver}$

4.2. Bibliography

[20] F. L. Lewis and V. L. Syrmos, *Optimal Control*, Second. Wiley-Interscience, Nov. 1995, ISBN: 978-0-471-03378-3.

CHAPTER 5.

Test Platform

The test platform consists of the designated hardware, [*MinSeg M2V3 two-wheeled robot, see Section 2.2.1*], and the designated development PC, [*see Section 2.2.2*]. To interface with the hardware, a Simulink model and a hierarchy of Matlab subscripts were created.

The Simulink model is capable of:

- Acting as an algorithm with which to program the hardware, such that it may:
  - Process
  - Actuate
  - Communicate
- Simulate an equivalent model of "the hardware when loaded with the same algorithm".

The Matlab script hierarchy is capable of:

- Initialize model parameters.
- Reconfigure model subsystems.
- Initialize a build or simulate event.
- Initialize a read or write event.
- Post-process raw read data.
- Save processed read data as well as other configuration data.
- Plot processed read data.

5.1. Simulink: minseg_M2V3_2017a.slx

minseg_M2V3_2017a.slx is the label of the Simulink model file. The label includes the label of the hardware which it represents as well as the version of the Mathworks Software Suite with which it was created. [*Using the model in a different version of the Simulink will require conversion; therefore, the two files will not be equivalent.*]

The Simulink model is hierarchical. The sections which follow will describe the model and will be similarly organized, as depicted in the extended-precision List of Contents below.

5.1.1. Root

The top level of the model, also known as the model root, is depicted in Figure 5.1.

The model root is contains the three primary components of the system:

- Plant
- Controller
- Board Inputs and Outputs

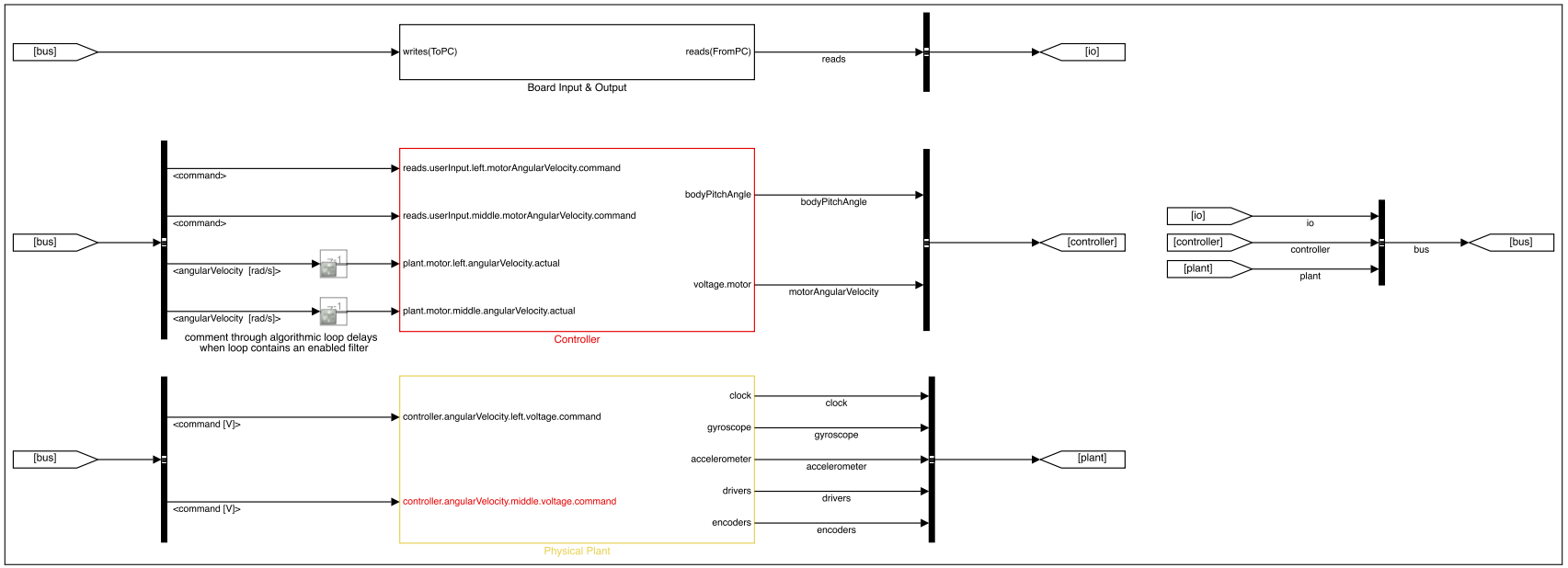

Figure 5.1.: [Simulink]: Root

5.1.1.1. Bus Structures

Bus structures are a means of routing large quantities of signals. They are similar to muxed signals; however, it is not necessary to separate all of the signals during the demux process.

It is evident in Figure 5.1 that all of the components are passed into separate bus structures, [*black bars on the right-side of the figure*], and that those bus structures are in turn merged into one global bus structure.

This grants the user the ability to call any significant signal wherever it is needed using bus selectors, [*black bars on the left-side of the figure*]. The user should take care to implement a delay in the path of any signal which is implemented recursively [*as feedback*]. [*This prevents the formation of an algebraic loop*].

5.1.1.2. Variant Subsystems

A variant subsystem is a subsystem containing multiple subsystems, defined as variants. Only one variant can be active at one time. The variant subsystem serves as the switch between them. [*Note that the variant subsystem cannot switch between variants during operation/runtime*].

Several subsystems contained in this model are variant subsystems. These variant subsystems are used to switch system configurations. Examples of these variant configurations include:

- The plant:
  - Actual hardware drivers. [*Hardware implementation only.*]
  - Hardware-equivalent simulation model of nonlinear dynamics. [*Simulation only.*]
  - Hardware-equivalent simulation model of linear dynamics. [*Simulation only.*]
- The controller design:
  - PID. [*Primarily for initial hardware characterization.*]
  - Optimal.
  - Pole-placement.

5.2. Matlab: minseg.m

The *minseg.m* script was developed to control the MinSeg test platform. The script is capable of:

- Reconfiguring the model
- Running a model simulation
- Programming the model hardware
- Communicating with the model hardware
  - Optimizing the communication rate.
  - Reformatting the raw hardware-output data on receipt.
- Saving the initialization parameters and output data.
- Plotting the output data.

The script is hierarchal, and is therefore only the root *or master* file to a series of subfiles. The subfiles are broken up into principal segments of the scripting process:

- Global setup
- User-input
- Initialization
- Processing
- Output
- Global Cleanup

CHAPTER 6.

Conclusions

The results provided in Section 4.1.3.1.2, demonstrate that it is theoretically possible to adequately track dynamic state commands specified to the two-wheeled robot system with respect to wheel angular position θ and body angular position (yaw) ϕ_y, while maintaining standard regulation of all other plant states.

While moving in accordance with the dynamic commands provided to the wheel angular position θ and body angular position (yaw) ϕ_y states, the robot remains balances and does not deviate significantly from the commands at any time.

6.1. Future work

The following is a non-comprehensive list of potential future work which could be performed to improve the capabilities of the test platform or supplement the control studies already performed on the test platform.

- Optimize sample interval of hardware.
  - Determine limiting factors in the reduction of the board sampling interval.
  - Improve upon these factors, if possible.
  - Determine alternative model algorithms such that processing *(per sample interval)* is significantly minimized.

    [*Example: Use binary classes wherever possible.*]
- Optimize serial communication.
  - Implement bluetooth wireless transmission.
  - Determine limiting factors in the reduction of the serial transmission interval.
    - Determine if increasing the serial communication sample interval improves limits on hardware sample interval.

      This is already performed when sending a high number of signals,

      but improved performance has not been verified.

- Determine if increasing the BAUD frequency on the board will remove limits on the serial transmission interval.

- Determine how to begin a read without resetting the hardware.

- Determine if dynamic/real-time plotting is worthwhile. If so, implement.

- Implement alternate linear controllers.
  - Implement pole-placement controller(s).
  - Implement LQG controller(s).
  - Implement H_N controller(s). [*Where N is an integer or infinity.*]
- Implement a nonlinear plant model.
  - Develop nonlinear controller(s).
  - Demonstrate operation in nonlinear states.

    Example: Operating in a state with a significantly increased component of horizontal pitch.
- Improve model parameters measurements.
  - Improve mass measurements.
    - Use scale with improved precision. [*Current precision is 0.01 [lb].*]
  - Improve motor transfer function measurement. [*Angular velocity vs. Input Voltage*]

    Hardware must remain perfectly upright while in motion to perform this measurement.

    The original measurement was taken while balancing the in-motion device by hand.

    Since a pitch controller has been developed, the measurement may be taken more accurately.
  - Verify conflicting motor parameters derived from References [5], [16]:
    - *Resistance,* R_{mtr}
    - *Torque constant,* $k_{mtr.T}$
    - *Back EMF constant,* $k_{mtr.bEMF}$
- Increase MinSeg power-source voltage-maximum. [*See Section 2.2.1.4.*]

  Current voltage source maximum: 09 [V].

  Motor driver operating maximum: 36 [V].

- Construct alternate physical models via simple variants.
  - Alternate mass distribution.
    - Reduce number of batteries to less than 6. [*Requires use of USB cable for power.*]
  - Alternate geometry.
    - Alternate wheel component.

      [*Search for Lego tires with differing radius, mass, and/or coefficient of friction. See [21]*]
    - Incorporation of a second mass on the pendulum.
- Perform movement on an uneven surface.
- Optimize filter design.
  - Determine tradeoffs between no filter vs 1^{st} to 6^{th} order bessel filters.
  - Determine tradeoffs between state-space and transfer function blocks, if any.
  - Determine tradeoffs between Matlab besself and bessel poles, if any?
- Optimization of observable data
  - Implement voltage sensor across battery holster.

    Use this voltage reading to determine the true voltage of the power source in operation.
  - Incorporate use of accelerometer?

    Incorporate use of Kalman filter?

    Compare effects.

- Test Windows and Linux compatibility. [*Document necessary changes, if any.*]

- Improve overrun detection.

  If the board cannot complete all of its processes before the sampling interval completes,

  then it performs incorrectly. Detection of this is possible and desirable for the user.

  Currently, overrun detection requires that the the user manually view an LED on the board.

  The LED is very small and almost entirely masked by the bluetooth module.

  (Simulink also currently prevents status reads of the overrun LED pin.)

  An alternative method should exist which alert the user more conveniently.

6.2. Bibliography

[5] B. Howard and L. Bushnell, "Enhancing linear system theory curriculum with an inverted pendulum robot," in *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, IEEE, Hangzhou, China, Oct. 2015. DOI: 10.1109/CSMA.2015.63.

[16] P. "Philo" Hurbain. (May 15, 2017). "Nxt motor internals," [Online]. Available: http://www.philohome.com/nxtmotor/nxtmotor.htm (visited on 07/03/2017).

[21] (Oct. 3, 2016). "Lego wheels chart," [Online]. Available: http://wheels.sariel.pl/ (visited on 07/03/2017).

Appendices

APPENDIX A.

Code Listings

A.1. minseg.m

Code Listing A.1: [minseg.m]: Root file

```
%% [Global ]
minseg_0p0p0p0_global

%% [Input  ]: Model
minseg_1p0p0p0_input

%% [Input  ]: Script: Commands

ui.x.build   = 0; % rebuild required for any change in [input]: model / [init]: model.
ui.x.write   = 0; % not yet implemented
ui.x.read    = 0;
ui.x.plot    = 0; % enables read/write by default.
ui.x.save    = 0; % enables read/write by default.
ui.x.cleanup = 0; % enables read/write by default.

%% [Input  ]: Script: Serial

switch 1  % serial duration
  case 0; ui.srl.n.transmits = 100; % [samples]
  case 1; ui.srl.T.transmits = 020; % [seconds]
end

%% [Input  ]: Script: Save
ui.save.label = '';

%% [Init   ]: Define parameters
minseg_2p0p0p0_init_general

minseg_2p1p0p0_init_model_general
minseg_2p1p1p0_init_model_plant
minseg_2p1p2p0_init_model_controller
minseg_2p1p3p0_init_model_io
minseg_2p1p9p0_init_model_build
```

```
34
35 minseg_2p2p0p0_init_serial_write
36 minseg_2p2p1p0_init_serial_read
37 minseg_2p2p2p0_init_serial_general
38 minseg_2p2p3p0_init_serial_reads
39 minseg_2p2p9p0_init_model_build
40
41 %minseg_2p3p0p0_init_build
42
43 %% [Process]:
44
45 % build (normal mode) / run (external mode)
46 if ui.x.build
47 minseg_3p0p0p0_process_build
48 end
49
50 % serial transmit (normal mode only)
51 if (ui.x.write || ui.x.read || ui.x.plot || ui.x.save )
52 minseg_3p1p0p0_process_serial_transmit
53 end
54
55 % serial post-processing
56 if (              ui.x.read || ui.x.plot || ui.x.save )
57 minseg_3p2p0p0_process_serial_reads
58
59 % if ui.mdl.case == 2
60 % minseg_3p3p1p0_process_motorTF
61 % end
62 %
63 % if ui.mdl.case == 3
64 % minseg_3p3p1p0_process_gyroBias
65 % end
66
67 end
68
69 %% [Output]:
70
71 % save
72 if ui.x.save
73 minseg_4p0p0p0_output_save
74 end
```

```
75
76 % plot
77 if ui.x.plot
78 minseg_4p1p0p0_output_serial_plot
79 % minseg_4p1p1p0_output_serial_plot
80 end
81
82 %% [Cleanup]:
83
84 if ui.x.cleanup
85 minseg_5p0p0p0_cleanup
86 end
87
88 %% End
```

Code Listing A.1: [minseg.m]: Root file

A.1.1. Global Setup

Code Listing A.2: [minseg.m]: Global Setup

```
%% [Global ]:
clc
clearvars
close all

% close all loaded simulink models and libraries.
% close_system( find_system('SearchDepth', 0) )

% close and delete all serial connections
if ~isempty(instrfindall)
  fclose   (instrfindall);
  delete   (instrfindall);
end

%% [Global ]: Add subdirectories to Matlab path
root.dir     = cd;
root.sub.dir = { [root.dir '/1. General Tools/'                ]
                 [root.dir '/1. General Tools/0. Bessel Poles' ]
                 [root.dir '/1. General Tools/1. fftPlus'      ]
                 [root.dir '/1. Subscripts'                    ]
                 [root.dir '/2. Model metadata'                ]
                 [root.dir '/3. Data'                          ]
               };

root.n.sub.dir = size( root.sub.dir, 1 );
for i0 = 1 : root.n.sub.dir
  addpath(   root.  sub.dir{ root.n.sub.dir - (i0 - 1), 1 } )
end

%% [Global ]: Add subdirectories to Simulink path
Simulink.fileGenControl( 'set'                                                  ...
                       , 'CacheFolder',    [ root.dir '/2. Model metadata/Work' ] ...
                       , 'CodeGenFolder', [ root.dir '/2. Model metadata/Code' ] ...
                       , 'createDir',       true                                 ...
                       )

%% End
```

Code Listing A.2: [minseg.m]: Global Setup

A.1.2. User Inputs

Code Listing A.3: [minseg.m]: User Inputs

```
%% [Input  ]: Model: General
ui.mdl.label = 'minseg_M2V3_2017a';

ui.mdl.mode  = 0;
% 0: normal
% 1: external

ui.mdl.case  = 0;
%  ##: Case Description:       Plant:    Controller:    Command:
% -01: Clear board             Empty     Empty          Empty    % not yet implemented
% +00: Custom                  Custom    Custom         Custom
% +01: Motor characterization  Hardware  FF - v.motor   0 --> 10 [V]
% +02: Gyro bias calibration   Hardware  PID - w.motor  0 --> 00 [rad/s]

%% [Input  ]: Model: Plant

ui.plant.dynamics.mode       = 0;
% 0: actual hardware
% 1: simulated dynamics (non-linear)
% 2: simulated dynamics (    linear)

ui.plant.n.batteries         = 6; % [range: 0 - 6]

ui.plant.x.bluetoothModule   = 1;
% 0: bluetooth module not inserted into board.
% 1: bluetooth module     inserted into board.

ui.plant.supply.mode         = 1;
% 0: 9.00 [V] (battery pack)
% 1: 4.50 [V] (usb cable)
% Important: Do NOT set to usb power if actually using battery power.

%% [Input  ]: Model: Controller: body.pitch.theta
% not yet implemented.

ui.ctrl.body.pitch.theta.mode = 0;
% #: mode:          input command:    [input command unit]:

```

```
%% [Input   ]: Model: Controller: motor.v

ui.ctrl.motor_v.mode           = 1;
% #: mode:          input command:    [input command unit]:
% 0: feedForward   v.motor           [V]
% 1: PID           w.motor           [rad/s]

switch ui.ctrl.motor_v.mode

case 0 % feed forward (input: v.motor)
  ui.io.write.ctrl.motor_v.cmd.tStart          (1,1) =  0;
  ui.io.write.ctrl.motor_v.cmd.val.x           (1,1) = +10;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max(1,1) = +0.01;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min(1,1) = -0.01;

case 1 % PID           (input: w.motor)
  ui.io.write.ctrl.motor_v.cmd.tStart          (1,1) =  0;
  ui.io.write.ctrl.motor_v.cmd.val.x           (1,1) =  0.50 * 2*pi;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max(1,1) = +0.10;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min(1,1) = -inf;

end

%% [Input   ]: Model: Serial
ui.srl.mode.address = 0;
% 0: left       usb port (2015 Macbook Pro)
% 1: left-rear usb port (2008 Macbook Pro)

% Note: Needs to be changed manually for external mode:
% Simulink: Configuration parameters: Hardware implementation: Host-board connection

ui.srl.T.decimation = 0; % [integer] [default: 0]
% Integer factor of board sample time (mdl.T.sample)
%  in which to iterate serial processes.
% If 0, minimum possible value will be used.
% (Could be greater than 1 if combined size of reads/writes is sufficiently large.)

%% End
```

Code Listing A.3: [minseg.m]: User Inputs

A.1.3. Initialization

A.1.3.1. General

Code Listing A.4: [minseg.m]: Initialization - General

```
%% [Init    ]: Conversions
k.intmax.uint8 = double( intmax('uint8') );
k.intmax.int16 = double( intmax('int16') );

k_deg2rad       = 2*pi / 360;
k_rad2deg       = 1 / k_deg2rad;

k.byte2bit      = 8;
k.bit2byte      = 1 / k.byte2bit;

k.lb2kg         = 0.45359233;
k.kg2lb         = 1 / k.lb2kg;

k.in2m          = 0.0000254;
k.m2in          = 1 / k.in2m;

%% End
```

Code Listing A.4: [minseg.m]: Initialization - General

A.1.3.2. Model

A.1.3.2.1. General

Code Listing A.5: [minseg.m]: Initialization - Model - General

```
%% [Init   ]: Initialize user-defined parameters
mdl.label= ui.mdl.label;
mdl.mode = ui.mdl.mode;
mdl.case = ui.mdl.case;

%% [Init   ]: Load model, if not already loaded
if ~bdIsLoaded( mdl.label )
load_system(    mdl.label );
end

%% [Init   ]: Define general model parameters

mdl.object = get_param(mdl.label, 'Object');

switch mdl.mode
  case 0; mdl.T.sample = 0.005; % 0: normal
  case 1; mdl.T.sample = 0.030; % 1: external
end

%% End
```

Code Listing A.5: [minseg.m]: Initialization - Model - General

A.1.3.2.2. Plant

Code Listing A.6: [minseg.m]: Initialization - Model - Plant

```
%% [Init   ]: Initialize user-defined parameters
plant.supply.  mode     = ui.plant.supply.  mode;
plant.dynamics.mode     = ui.plant.dynamics.mode;

plant.n.batteries       = ui.plant.n.batteries;
plant.x.bluetoothModule = ui.plant.x.bluetoothModule;

%% [Init   ]: Define general plant parameters

switch plant.supply.mode
  case 0; plant.supply.v = 9.00; % [V]
  case 1; plant.supply.v = 4.50; % [V]
end

a.gravity               = 9.81; % acceleration [m / s^2]

load( 'bessel poles.mat' )

%% [Init   ]: Verify legitimate operating modes

if plant.supply.mode == 0
assert( plant.n.batteries == 6,                                          ...
      [                                                                  ...
       'Battery power is enabled (plant.supply.mode == 0);\n'            ...
       'however, the number of batteries in use is not equal to\n'       ...
       'the number of batteries needed to operate in '                   ...
       'battery power mode (plant.n.batteries ~= 6)'                     ...
      ]                                                                  ...
      );
end

%% [Init   ]: Define parameters based on user-specified plant dynamics

switch plant.dynamics.mode
  case 0; minseg_2p1p1p1_init_model_plant_hardware
  case 1; minseg_2p1p1p2_init_model_plant_nonlinearDynamics
  case 2; minseg_2p1p1p3_init_model_plant_linearDynamics
end
```

```
39
40 %% End
```

Code Listing A.6: [minseg.m]: Initialization - Model - Plant

A.1.3.2.2.1. Hardware

Code Listing A.7: [minseg.m]: Initialization - Model - Plant - Hardware

```
%% [Init    ]: Motor: Driver
mtr.driver. left.  pin.pos = 6;
mtr.driver. left.  pin.neg = 8;
mtr.driver. middle.pin.pos = 2;
mtr.driver. middle.pin.neg = 5;

%% [Init    ]: Motor: Encoder
% not yet implemented
% mask encoder model, then use pins as mask parameters
mtr.encoder.left.  pin.A   = 19;
mtr.encoder.left.  pin.B   = 18;
mtr.encoder.middle.pin.A   = 15;
mtr.encoder.middle.pin.B   = 62;

mtr.encoder.countPerRev    = 720;
mtr.encoder.radPerRev      = 2 * pi;

%% [Init    ]: Motor: Encoder: angVel bessel filter: design parameters
mtr.encoder.filter.T.settle = mdl.T.sample * 25; % [s]
mtr.encoder.filter.order    = 4;     % [-] [integer] [ range: 02 : 10 ]

%% [Init    ]: Motor: Encoder: angVel bessel filter: transfer function
% divide normalize poles by settling time
mtr.encoder.filter.s.poles = poly( s.pole.bessel{mtr.encoder.filter.order} ...
                                 / mtr.encoder.filter.T.settle             ...
                                 );


% create transfer function
mtr.encoder.filter.s.tf    =   tf( mtr.encoder.filter.s.poles(end)         ...
                                 , mtr.encoder.filter.s.poles              ...
                                 );


% discretize transfer function
mtr.encoder.filter.z.tf    =  c2d( mtr.encoder.filter.s.tf                 ...
                                 , mdl.T.sample                            ...
                                 );
```

```
39
40 % break transfer function into numerator and demonintor polynomials
41 [ mtr.encoder.filter.s.num ...
42 , mtr.encoder.filter.s.den ...
43 ] = tfdata                 ...
44 ( mtr.encoder.filter.s.tf  ...
45 );
46
47 [ mtr.encoder.filter.z.num ...
48 , mtr.encoder.filter.z.den ...
49 ] = tfdata                 ...
50 ( mtr.encoder.filter.z.tf  ...
51 );
52
53 % convert cells to matrices
54 mtr.encoder.filter.s.num = mtr.encoder.filter.s.num{:};
55 mtr.encoder.filter.s.den = mtr.encoder.filter.s.den{:};
56 mtr.encoder.filter.z.num = mtr.encoder.filter.z.num{:};
57 mtr.encoder.filter.z.den = mtr.encoder.filter.z.den{:};
58
59 %% [Init   ]: Motor: Encoder: angVel bessel filter: state-space
60
61 % create s-plane state space equations (canonical representation)
62 mtr.encoder.filter.s.ss.A        = diag( ones( mtr.encoder.filter.order - 1, 1 ), 1);
63 mtr.encoder.filter.s.ss.A(end,:) = mtr.encoder.filter.s.poles( end : -1 : 2 );
64 mtr.encoder.filter.s.ss.A(end,:) = mtr.encoder.filter.s.ss.A(end,:) ...
65                                  / mtr.encoder.filter.s.poles( 1 ) * -1;
66
67 mtr.encoder.filter.s.ss.B        = [ zeros(    mtr.encoder.filter.order - 1, 1 ); 1 ];
68 mtr.encoder.filter.s.ss.C        = [ zeros( 1, mtr.encoder.filter.order - 1    ) 1 ];
69 mtr.encoder.filter.s.ss.D        = 0;
70
71
72 % discretize s-plane state space equations (canonical representation)
73 [ mtr.encoder.filter.z.ss.A ... phi
74 , mtr.encoder.filter.z.ss.B ... gamma
75 ] = zohe                    ...
76 ( mtr.encoder.filter.s.ss.A ... A
77 , mtr.encoder.filter.s.ss.B ... B
78 , mdl.T.sample              ... T
79 );
```

```
80
81  mtr.encoder.filter.z.ss.C        = mtr.encoder.filter.s.ss.C;
82  mtr.encoder.filter.z.ss.D        = mtr.encoder.filter.s.ss.D;
83
84  %% [Init    ]: Gyroscope
85  gyro.dlpf.mode   = 0;  % [ default: 0 ]
86  % | # | maxValue [deg/s] | bandwidth [Hz] | delay [s] |
87  % | 0 | +/- 0250         | 256            | 00.98     |
88  % | 1 | +/- 0500         | 188            | 01.90     |
89  % | 2 | +/- 1000         | 098            | 02.80     |
90  % | 3 | +/- 2000         | 042            | 04.80     |
91  % | 4 | +/- ????         | 020            | 08.30     |
92  % | 5 | +/- ????         | 010            | 13.40     |
93  % | 6 | +/- ????         | 005            | 18.60     |
94
95  switch gyro.dlpf.mode
96  case 0; gyro.maxVal = 0250 * k_deg2rad;
97  case 1; gyro.maxVal = 0500 * k_deg2rad;
98  case 2; gyro.maxVal = 1000 * k_deg2rad;
99  case 3; gyro.maxVal = 2000 * k_deg2rad;
100 end
101
102 gyro.k_raw2actual = gyro.maxVal / k.intmax.int16;
103
104 % [source: 1. Test Cases/1. Gyro Bias Calibration]
105 gyro.x.bias      = -266.0779700;
106 gyro.y.bias      = -135.5037500;
107 gyro.z.bias      = -034.3493271;
108
109 gyro.x.reset     =  0;
110 gyro.y.reset     =  0;
111 gyro.z.reset     =  0;
112
113 %% [Init    ]: Gyroscope     : angVel bessel filter: design parameters
114 gyro.filter.T.settle = mdl.T.sample * 25; % [s]
115 gyro.filter.order    = 4;     % [-] [integer] [ range: 02 : 10 ]
116
117 %% [Init    ]: Gyroscope     : angVel bessel filter: transfer function
118 % divide normalize poles by settling time
119 gyro.filter.s.poles = poly( s.pole.bessel{gyro.filter.order} ...
120                           / gyro.filter.T.settle             ...
```

```
121                             );
122
123
124 % create transfer function
125 gyro.filter.s.tf    =   tf( gyro.filter.s.poles(end)          ...
126                             , gyro.filter.s.poles              ...
127                             );
128
129
130 % discretize transfer function
131 gyro.filter.z.tf    =  c2d( gyro.filter.s.tf                  ...
132                             , mdl.T.sample                     ...
133                             );
134
135 % break transfer function into numerator and demonintor polynomials
136 [ gyro.filter.s.num ...
137 , gyro.filter.s.den ...
138 ] = tfdata          ...
139 ( gyro.filter.s.tf  ...
140 );
141
142 [ gyro.filter.z.num ...
143 , gyro.filter.z.den ...
144 ] = tfdata          ...
145 ( gyro.filter.z.tf  ...
146 );
147
148 % convert cells to matrices
149 gyro.filter.s.num = gyro.filter.s.num{:};
150 gyro.filter.s.den = gyro.filter.s.den{:};
151 gyro.filter.z.num = gyro.filter.z.num{:};
152 gyro.filter.z.den = gyro.filter.z.den{:};
153
154 %% [Init   ]: Gyroscope     : angVel bessel filter: state-space
155
156 % create s-plane state space equations (canonical representation)
157 gyro.filter.s.ss.A        = diag( ones( gyro.filter.order - 1, 1 ), 1);
158 gyro.filter.s.ss.A(end,:) = gyro.filter.s.poles( end : -1 : 2 );
159 gyro.filter.s.ss.A(end,:) = gyro.filter.s.ss.A(end,:) ...
160                             / gyro.filter.s.poles( 1 ) * -1;
161
```

```
gyro.filter.s.ss.B        = [ zeros(    gyro.filter.order - 1, 1 ); 1 ];
gyro.filter.s.ss.C        = [ zeros( 1, gyro.filter.order - 1    )  1 ];
gyro.filter.s.ss.D        = 0;


% discretize s-plane state space equations (canonical representation)
[ gyro.filter.z.ss.A ... phi
, gyro.filter.z.ss.B ... gamma
] = zohe               ...
( gyro.filter.s.ss.A ... A
, gyro.filter.s.ss.B ... B
, mdl.T.sample       ... T
);

gyro.filter.z.ss.C        = gyro.filter.s.ss.C;
gyro.filter.z.ss.D        = gyro.filter.s.ss.D;

%% [Init    ]: Accelerometer
accel.afs_sel.mode   = 0;  % [ Required: 0 ]
% | # | maxValue [g] | Sensitivity [LSB/mg] |
% | 0 | +/- 02       | 8192
% | 1 | +/- 04       | 4096
% | 2 | +/- 08       | 2048
% | 3 | +/- 16       | 1024

assert( accel.afs_sel.mode == 0 );

switch accel.afs_sel.mode
case 0; accel.maxVal = 02 * a.gravity;
case 1; accel.maxVal = 04 * a.gravity;
case 2; accel.maxVal = 08 * a.gravity;
case 3; accel.maxVal = 16 * a.gravity;
end

accel.k_raw2actual = accel.maxVal / k.intmax.int16;

%% End
```

Code Listing A.7: [minseg.m]: Initialization - Model - Plant - Hardware

A.1.3.2.2.2. Nonlinear Dynamics Model

Code Listing A.8: [minseg.m]: Initialization - Model - Plant - Nonlinear Dynamics Model

```
%% End
```

Code Listing A.8: [minseg.m]: Initialization - Model - Plant - Nonlinear Dynamics Model

A.1.3.2.2.3. Linear Dynamics Model

Code Listing A.9: [minseg.m]: Initialization - Model - Plant - Linear Dynamics Model

```
%% [Init   ]: Plant: Wheel   (single)

% mass measurement precision: 0.01 lb
% note: this could be improved with a better scale.

plant.axel.m         = 0.000;     %                          [kg]       [note low precision
    .]

plant.wheel.r        = 0.021;     %  radius                  [m]        [source: howard]
plant.wheel.m        = 0.036 / 2; % (includes axel)         [kg]       [source: howard]
plant.wheel.J        = 7.460e-6;  %  moment of inertia       [kg / m^2] [source: howard]
                                  %  measured from center of mass of wheel

%% [Init   ]: Plant: Body: Masses
% note: body does not include wheels.

% mass measurement precision: 0.01 lb
% note: this could be improved with a better scale.

% plant.board.     m   = 1.000 * k.lb2kg; %                  [kg]
% plant.motorCable.m   = 0.010 * k.lb2kg; % (quantity: 1)    [kg] [note low precision.]
% plant.motor.     m   = 0.220 * k.lb2kg; % (quantity: 1)    [kg]
% plant.battery.   m   = 1.000 * k.lb2kg; % (quantity: 1)    [kg]

% plant.bluetooth. m   = 0.000 * k.lb2kg; % bluetooth module [kg] [note low precision.]
% plant.usbCable.  m   = 0.040 * k.lb2kg; % (not included)   [kg]

% plant.body.m         =  plant.board.m                                    ...
%                       +  plant.motor.m      * 2                          ...
%                       +  plant.motorCable.m * 2                          ...
%                       +  plant.battery.m    * plant.n.battery            ...
%                       +  plant.bluetooth.m  * plant.x.bluetoothModule;
%                             %  mass   [kg]

plant.body.m         =  1.030; % (not included)    [kg]
                               % net measurement taken to reduce rounding errors.
                               % [taken with 6 batteries].

```

```
%% [Init    ]: Plant: Body
% note: does not include wheels.

plant.body.l.h        =  8.00 * k.in2m; %  height [m]
plant.body.l.w        =  3.25 * k.in2m; %  width  [m]
plant.body.l.d        =  2.50 * k.in2m; %  depth  [m]

switch plant.x.bluetoothModule

case 0 % Not inserted
  switch plant.n.batteries
  case 0; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 5; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 6; plant.body.f.natural =  1; %  natural frequency [rad/s]
  end

case 1 % Inserted
  switch plant.n.batteries
  case 0; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 5; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 6; plant.body.f.natural =  3.5087719; %  natural frequency [rad/s]
  end

end

plant.body.w.natural = 2 * pi * plant.body.f.natural;
                          %  natural angular frequency [rad/s]

plant.body.l.c        =  3 * (a.gravity - plant.body.w.natural^2 * plant.wheel.r) ...
                      / (4 *              plant.body.w.natural^2               );
                          %  wheel axel to center of mass of robot [m]

plant.body.J.x        =  plant.body.m *  plant.body.l.c^2                    ...
                      /  3;
                          %  moment of inertia (pitch) [kg / m^2]
                          % (measured from center of mass of robot)

plant.body.J.y        =  plant.body.m                                        ...
                      * (plant.body.l.w^2 + plant.body.l.d^2)                ...
                      /  12;
                          %  moment of inertia (yaw)   [kg / m^2]
```

```
79                          % (measured from center of mass of robot)
80
81 %% [Init    ]: Plant: Net (body + 2 * wheel)
82 plant.net.m           =  plant.body.m + 2 * plant.wheel.m; % [kg]
83
84 %% [Init    ]: Plant: Motor
85 mtr.R                 =  4.400; %  resistance              [ohm        ] [source: howard]
86 mtr.k.dlambda         =  0.495; %  back EMF constant       [V*s / rad  ] [source: howard]
87 mtr.k.torque          =  0.470; %  torque   constant       [N*m / A    ] [source: howard]
88
89 switch plant.x.bluetoothModule
90
91 case 0 % Not inserted
92   switch plant.n.batteries
93   case 0
94   mtr.k.v2w           =  1.000; %  transfer function (y/u) [rad / (s*V)]
95                                 % (measured when body is upright AND
96                                 %  both wheels are at equivalent speed
97                                 %  in a common direction.)
98
99   case 5
100  mtr.k.v2w           =  1.000; %  transfer function (y/u) [rad / (s*V)]
101                                % (measured when body is upright AND
102                                %  both wheels are at equivalent speed
103                                %  in a common direction.)
104
105  case 6
106  mtr.k.v2w           =  1.000; %  transfer function (y/u) [rad / (s*V)]
107                                % (measured when body is upright AND
108                                %  both wheels are at equivalent speed
109                                %  in a common direction.)
110
111  end
112
113 case 1 % Inserted
114   switch plant.n.batteries
115   case 0
116   mtr.k.v2w           =  1.000; %  transfer function (y/u) [rad / (s*V)]
117                                % (measured when body is upright AND
118                                %  both wheels are at equivalent speed
119                                %  in a common direction.)
```

```
120
121   case 5
122   mtr.k.v2w          =  1.000; %  transfer function (y/u) [rad / (s*V)]
123                               % (measured when body is upright AND
124                               %  both wheels are at equivalent speed
125                               %  in a common direction.)
126
127   case 6
128   mtr.k.v2w          =  3/3.35; %  transfer function (y/u) [rad / (s*V)]
129                               % (measured when body is upright AND
130                               %  both wheels are at equivalent speed
131                               %  in a common direction.)
132
133   end
134
135 end
136
137 mtr.k.friction       =  mtr.k.torque * (1 - mtr.k.dlambda * mtr.k.v2w)      ...
138                      / (mtr.R * mtr.k.v2w);
139                           %  coefficient of friction [-          ]
140
141 %% [Init   ]: Plant: State space model term abbreviations
142
143 % wheel.theta and body.theta.x (pitch) (psi)
144 plant.q(1,1) = plant.net. m * plant.wheel.r^2                    + plant.wheel.J;
145 plant.q(2,1) = plant.body.m * plant.wheel.r^2 * plant.body. l.c                  ;
146 plant.q(3,1) = plant.body.m *                   plant.body. l.c^2 + plant.wheel.J;
147 plant.q(4,1) = mtr.k.torque * mtr.k.dlambda / mtr.R + mtr.k.friction             ;
148 plant.q(5,1) = plant.body.m * a.gravity       * plant.body. l.c                 ;
149 plant.q(6,1) = mtr.k.torque                 / mtr.R                             ;
150
151 plant.Q{1,1} =     [ +plant.q(1) +plant.q(2)
152                       +plant.q(2) +plant.q(3) ];
153 plant.Q{2,1} = 2 * [ +plant.q(4) -plant.q(4)
154                       -plant.q(4) +plant.q(4) ];
155 plant.Q{3,1} =     [ +0          +0
156                       +0          -plant.q(5) ];
157 plant.Q{4,1} =     [ +plant.q(6) +plant.q(6)
158                       -plant.q(6) -plant.q(6) ];
159
160 % body.theta.y (yaw) (phi)
```

```
161 plant.r(1,1) =  plant.body.l.w / plant.wheel.r;
162
163 plant.R{1,1} =  0.5 * plant.wheel.m  * plant.body.l.w^2                    ...
164             +  plant.body.J.y                                               ...
165             +  0.5 * plant.r(1)^2 * plant.wheel.J;
166 plant.R{2,1} =  0.5 * plant.r(1)^2 * plant.q(4);
167 plant.R{3,1} =  0.5 * plant.r(1)   * mtr.k.torque / mtr.R;
168
169 % overall
170 plant.a{1,1} =  - plant.Q{1} \ plant.Q{3};
171 plant.a{2,1} =  - plant.Q{1} \ plant.Q{2};
172 plant.a{3,1} =  - plant.R{1} \ plant.R{2}; % note the backslash.
173
174 plant.b{1,1} =  + plant.Q{1} \ plant.Q{4};
175 plant.b{2,1} =  + plant.R{1} \ plant.R{3};
176
177 %% [Init   ]: Plant State Space Model: A
178
179 plant.A(1,1) =  0;
180 plant.A(1,2) =  0;
181 plant.A(1,3) =  1;
182 plant.A(1,4) =  0;
183 plant.A(1,5) =  0;
184 plant.A(1,6) =  0;
185
186 plant.A(2,1) =  0;
187 plant.A(2,2) =  0;
188 plant.A(2,3) =  0;
189 plant.A(2,4) =  1;
190 plant.A(2,5) =  0;
191 plant.A(2,6) =  0;
192
193 plant.A(3,1) =  plant.a{1}(1,1);
194 plant.A(3,2) =  plant.a{1}(1,2);
195 plant.A(3,3) =  plant.a{2}(1,1);
196 plant.A(3,4) =  plant.a{2}(1,2);
197 plant.A(3,5) =  0;
198 plant.A(3,6) =  0;
199
200 plant.A(4,1) =  plant.a{1}(2,1);
201 plant.A(4,2) =  plant.a{1}(2,2);
```

```
202 plant.A(4,3) =  plant.a{2}(2,1);
203 plant.A(4,4) =  plant.a{2}(2,2);
204 plant.A(4,5) =  0;
205 plant.A(4,6) =  0;
206
207 plant.A(5,1) =  0;
208 plant.A(5,2) =  0;
209 plant.A(5,3) =  0;
210 plant.A(5,4) =  0;
211 plant.A(5,5) =  0;
212 plant.A(5,6) =  1;
213
214 plant.A(6,1) =  0;
215 plant.A(6,2) =  0;
216 plant.A(6,3) =  0;
217 plant.A(6,4) =  0;
218 plant.A(6,5) =  0;
219 plant.A(6,6) =  plant.a{3};
220
221 %% [Init   ]: Plant State Space Model: B
222
223 plant.B(1,1) =  0;
224 plant.B(2,1) =  0;
225 plant.B(3,1) =  plant.b{1}(1,1);
226 plant.B(4,1) =  plant.b{1}(2,1);
227 plant.B(5,1) =  0;
228 plant.B(6,1) = -plant.b{2};
229
230 plant.B(1,2) =  0;
231 plant.B(2,2) =  0;
232 plant.B(3,2) =  plant.b{1}(1,2);
233 plant.B(4,2) =  plant.b{1}(2,2);
234 plant.B(5,2) =  0;
235 plant.B(6,2) = +plant.b{2};
236
237 %% [Init   ]: Plant State Space Model: C, D
238
239 plant.C      = eye  ( size( plant.A    )                    );
240 plant.D      = zeros( size( plant.A, 1 ), size( plant.C, 2 ) );
241
242 %% End
```

Code Listing A.9: [minseg.m]: Initialization - Model - Plant - Linear Dynamics Model

A.1.3.2.3. Controller

Code Listing A.10: [minseg.m]: Initialization - Model - Controller

```
%% [Init   ]: Initialize user-defined parameters
ctrl.motor_v.mode = ui.ctrl.motor_v.mode;

%% [Init   ]: Setup controller variant subsystems
ctrl.motor_v.ff. motor_v.var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
ctrl.motor_v.pid.motor_w.var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );

%% [Init   ]: Define controller model parameters

switch ctrl.motor_v.mode

case 0 %

case 1
   ctrl.motor_v.pid.motor_w.k.p = 0.500;
   ctrl.motor_v.pid.motor_w.k.i = 1.000;
   ctrl.motor_v.pid.motor_w.k.d = 0.000;

   ctrl.motor_v.pid.motor_w.int.maxVal = +plant.supply.v;
   ctrl.motor_v.pid.motor_w.int.minVal = -plant.supply.v;

end

%% End
```

Code Listing A.10: [minseg.m]: Initialization - Model - Controller

A.1.3.2.4. Board Inputs and Outputs

Code Listing A.11: [minseg.m]: Initialization - Model - User-Defined Board Inputs and Outputs

```
%% [Init   ]: Setup board i/o variant subsystems

% general

io.write.serial.                                    var = Simulink.Variant( 'mdl_mode == 0' );
io.write.scopes.                                    var = Simulink.Variant( 'mdl_mode == 1' );

% plant: hardware

io.write.serial.hardware.                           var = Simulink.Variant( '
    plant_dynamics_mode == 0' );

io.write.serial.hardware.ff.                        var = Simulink.Variant( 'ctrl_motor_v_mode
    == 0' );
io.write.serial.hardware.pid.                       var = Simulink.Variant( 'ctrl_motor_v_mode
    == 1' );

io.write.serial.hardware.ff.standard.               var = Simulink.Variant( 'mdl_case == 0' );
io.write.serial.hardware.ff.motorCharacterization.var = Simulink.Variant( 'mdl_case == 1' );

io.write.serial.hardware.pid.standard.              var = Simulink.Variant( 'mdl_case == 0' );
io.write.serial.hardware.pid.sensorCalibration.     var = Simulink.Variant( 'mdl_case == 2' );

% plant: nonlinearDynamics

io.write.serial.nonlinearDynamics.                  var = Simulink.Variant( '
    plant_dynamics_mode == 1' );

io.write.serial.nonlinearDynamics.ff.               var = Simulink.Variant( 'ctrl_motor_v_mode
    == 0' );
io.write.serial.nonlinearDynamics.pid.              var = Simulink.Variant( 'ctrl_motor_v_mode
    == 1' );

io.write.serial.nonlinearDynamics.ff.standard.      var = Simulink.Variant( 'mdl_case == 0' );

io.write.serial.nonlinearDynamics.pid.standard.     var = Simulink.Variant( 'mdl_case == 0' );

% plant: nonlinearDynamics
```

```
33
34 io.write.serial.linearDynamics.                    var = Simulink.Variant( '
       plant_dynamics_mode == 2' );
35
36 io.write.serial.linearDynamics.ff.                 var = Simulink.Variant( 'ctrl_motor_v_mode
       == 0' );
37 io.write.serial.linearDynamics.pid.                var = Simulink.Variant( 'ctrl_motor_v_mode
       == 1' );
38
39 io.write.serial.linearDynamics.ff.standard.        var = Simulink.Variant( 'mdl_case == 0' );
40
41 io.write.serial.linearDynamics.pid.standard.       var = Simulink.Variant( 'mdl_case == 0' );
42
43 %% [Init    ]: Write commands
44 io.write.ctrl.motor_v.cmd.tStart           = ui.io.write.ctrl.motor_v.cmd.tStart;          %
       [  s             ]
45 io.write.ctrl.motor_v.cmd.val.x            = ui.io.write.ctrl.motor_v.cmd.val.x;           %
       [ <cmd>          ]
46 io.write.ctrl.motor_v.cmd.val_norm.dx.max = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max; %
       [  cmd.norm / s ]
47 io.write.ctrl.motor_v.cmd.val_norm.dx.min = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min; %
       [  cmd.norm / s ]
48
49 %% End
```

Code Listing A.11: [minseg.m]: Initialization - Model - User-Defined Board Inputs and Outputs

A.1.3.2.5. Build Parameters

Code Listing A.12: [minseg.m]: Initialization - Model - Model Build Parameters

```
%% [Init   ]: Initialize list of general parameters used within Simulink model

mdl.parameter.label = {};

% Specify parameters which will be used in model:
mdl.parameter.label = [...
mdl.parameter.label
{
   'k.intmax.uint8'

   'mdl.mode'
   'mdl.case'
   'mdl.T.sample'

   'plant.dynamics.mode'
   'plant.supply.v'

   'ctrl.motor_v.mode'
   'ctrl.motor_v.ff.motor_v.var'
   'ctrl.motor_v.pid.motor_w.var'

   'io.write.serial.var'
   'io.write.scopes.var'

   'io.write.serial.hardware.var'
   'io.write.serial.hardware.ff.var'
   'io.write.serial.hardware.ff.standard.var'
   'io.write.serial.hardware.ff.motorCharacterization.var'
   'io.write.serial.hardware.pid.var'
   'io.write.serial.hardware.pid.standard.var'
   'io.write.serial.hardware.pid.sensorCalibration.var'

   'io.write.serial.nonlinearDynamics.var'
   'io.write.serial.nonlinearDynamics.ff.var'
   'io.write.serial.nonlinearDynamics.ff.standard.var'
   'io.write.serial.nonlinearDynamics.pid.var'
   'io.write.serial.nonlinearDynamics.pid.standard.var'

```

```
    'io.write.serial.linearDynamics.var'
    'io.write.serial.linearDynamics.ff.var'
    'io.write.serial.linearDynamics.ff.standard.var'
    'io.write.serial.linearDynamics.pid.var'
    'io.write.serial.linearDynamics.pid.standard.var'

    'io.write.ctrl.motor_v.cmd.tStart'
    'io.write.ctrl.motor_v.cmd.val.x'
    'io.write.ctrl.motor_v.cmd.val_norm.dx.max'
    'io.write.ctrl.motor_v.cmd.val_norm.dx.min'

}];

%% [Init    ]: Append case-dependent parameters: Plant: Dynamics model

switch plant.dynamics.mode

case 0 % hardware
mdl.parameter.label = [...
mdl.parameter.label
{
    'gyro.dlpf.mode'
    'gyro.k_raw2actual'

    'gyro.x.bias'
    'gyro.x.reset'
    'gyro.y.bias'
    'gyro.y.reset'
    'gyro.z.bias'
    'gyro.z.reset'

    'gyro.filter.z.ss.A'
    'gyro.filter.z.ss.B'
    'gyro.filter.z.ss.C'
    'gyro.filter.z.ss.D'

    'gyro.filter.z.num'
    'gyro.filter.z.den'

    'accel.k_raw2actual'

```

```
    'mtr.driver.left.pin.pos'
    'mtr.driver.left.pin.neg'
    'mtr.driver.middle.pin.pos'
    'mtr.driver.middle.pin.neg'

    'mtr.encoder.left.pin.A'
    'mtr.encoder.left.pin.B'
    'mtr.encoder.middle.pin.A'
    'mtr.encoder.middle.pin.B'

    'mtr.encoder.countPerRev'
    'mtr.encoder.radPerRev'

    'mtr.encoder.filter.z.ss.A'
    'mtr.encoder.filter.z.ss.B'
    'mtr.encoder.filter.z.ss.C'
    'mtr.encoder.filter.z.ss.D'

    'mtr.encoder.filter.z.num'
    'mtr.encoder.filter.z.den'
}];

case 1
mdl.parameter.label = [...
mdl.parameter.label
{
}];

case 2
mdl.parameter.label = [...
mdl.parameter.label
{
}];

end

%% [Init   ]: Append case-dependent parameters: Controller: v.motor.input

switch ctrl.motor_v.mode

case 0 % feed-forward (input: motor.v)
```

```
mdl.parameter.label = [...
mdl.parameter.label
{

}];

case 1 % PID           (input: motor.w)
mdl.parameter.label = [...
mdl.parameter.label
{
   'ctrl.motor_v.pid.motor_w.k.p'
   'ctrl.motor_v.pid.motor_w.k.i'
   'ctrl.motor_v.pid.motor_w.k.d'

   'ctrl.motor_v.pid.motor_w.int.maxVal'
   'ctrl.motor_v.pid.motor_w.int.minVal'
}];

end

%% [Init   ]: Relabel parameters for use within Simulink model

% Number of parameters specified
mdl.n.parameter         = size( mdl.parameter.label, 1);

% Indices which contain periods:
mdl.parameter.z.period = regexp(mdl.parameter.label, '\.');

% For each parameter:
for i0 = 1 : mdl.n.parameter

% Create a new label in which all periods have been set to underscores:
mdl.parameter.label0 = mdl.parameter.label    {i0,1};
mdl.parameter.label0 ( mdl.parameter.z.period{i0,1} ) = '_';

% Set the data for the new label equal to the data from the old label:
eval([ mdl.parameter.label0 ' = ' mdl.parameter.label{i0,1} ';' ]);

end

%% [Init   ]: Refresh model to update variant blocks
```

```
162
163 mdl.object.refreshModelBlocks
164
165 %% End
```

Code Listing A.12: [minseg.m]: Initialization - Model - Model Build Parameters

A.1.3.3. Serial

A.1.3.3.1. Write

Code Listing A.13: [minseg.m]: Initialization - Serial - Write

```
1
2 %% End
```

Code Listing A.13: [minseg.m]: Initialization - Serial - Write

A.1.3.3.2. Read

Code Listing A.14: [minseg.m]: Initialization - Serial - Read

```
%% [Init   ]: Import serial read signal label and datatype from model

% serial read block location:
srl.read{1,1}.block.path = ...
  [  mdl.label '/Board Input // Output/Writes (To PC)/Serial' ];

while 1 % continue until 'break' command

  srl.read{1,1}.block.path0 = get_param(  srl.read{1,1}.block.path          ...
                                       , 'ActiveVariantBlock'               ...
                                       );

  if isempty( srl.read{1,1}.block.path0 )
  break
  end

  srl.read{1,1}.block.path  = srl.read{1,1}.block.path0;

end

% serial read block names:
srl.read{1,1}.block.busSelect.label =                       ...
find_system(  srl.read{1,1}.block.path                      ...
            , 'Regexp', 'on'                                ...
            , 'Name',   'Bus Selector'                      ...
            );

srl.read{1,1}.block.convert.label =                         ...
find_system(  srl.read{1,1}.block.path                      ...
            , 'Regexp', 'on'                                ...
            , 'Name',   'Data Type Conversion*'             ...
            );

srl.read{1,1}.block.bytepack.label =                        ...
find_system(  srl.read{1,1}.block.path                      ...
            ,  'Regexp', 'on'                               ...
            ,  'Name',   'Byte Pack*'                       ...
            );
```

```
39
40 % import output signal labels from bus block
41 srl.read{1,1}.block.busSelect.signals.out =                ...
42 get_param  ( srl.read{1,1}.block.busSelect.label           ...
43            , 'OutputSignals'                               ...
44            );
45
46 srl.read{1,1}.block.busSelect.signals.out =                ...
47 regexp     ( srl.read{1,1}.block.busSelect.signals.out{:} ...
48            , '[^,]*'                                       ...
49            , 'match'                                       ...
50            ).';
51
52 % verify equivalent number of each type of serial read preprocessing block:
53 assert(    size( srl.read{1,1}.block.busSelect.signals.out, 1 ) == ...
54            size( srl.read{1,1}.block.convert.label,         1 )    ...
55       , [ srl.read{1,1}.block.path ':\n'                          ...
56          'Less Convert blocks than number of signals.']           ...
57       )
58
59 assert(    size( srl.read{1,1}.block.busSelect.signals.out, 1 ) == ...
60            size( srl.read{1,1}.block.bytepack.label, 1 )          ...
61       , [ srl.read{1,1}.block.path ':\n'                          ...
62          'Less Byte Pack blocks than number of signals.']         ...
63       )
64
65 %% [Init   ]: Define serial read signal label and datatype parameters
66
67 % number of signals being transmitted:
68 srl.read {1,1}.n.signals = size( srl.read{1,1}.block.convert.label, 1 );
69
70 % increase srl.read cell vector size to number of signals
71 srl.read { srl.read{1,1}.n.signals, 1 } = [];
72 srl.reads{ srl.read{1,1}.n.signals, 1 } = [];
73
74
75 % for each serial read signal existing within the model:
76 for i0 = 1 : srl.read{1,1}.n.signals
77
78   % import the datalabel of that signal from the bus block
79    srl.read{i0,1}.label = srl.read{1,1}.block.busSelect.signals.out{i0,1};
```

```
80
81    % import the datatype of that signal from the datatype conversion block
82    srl.read{i0,1}.type.original = ...
83    get_param( srl.read{ 1,1}.block.convert. label{i0,1}, 'OutDataTypeStr' );
84
85    % for posterity, set the datatype in the bytepack block to the same datatype.
86    set_param( srl.read{ 1,1}.block.bytepack.label{i0,1}, 'datatypes',      ...
87        ['{''' srl.read{i0,1}.type.original '''}' ]                          );
88
89  end
90
91  %% [Init   ]: Define serial read signal size parameters
92
93  % initialize counters
94  srl.read{1,1}.n.Bytes       = 0; % [bytes                  / read]
95
96  srl.read{1,1}.n.type.uint8  = 0; % [type: 'uint8'  signals / read]
97  srl.read{1,1}.n.type.uint16 = 0; % [type: 'uint16' signals / read]
98  srl.read{1,1}.n.type.uint32 = 0; % [type: 'uint32' signals / read]
99
100 srl.read{1,1}.n.type.int8   = 0; % [type: 'int8'   signals / read]
101 srl.read{1,1}.n.type.int16  = 0; % [type: 'int16'  signals / read]
102 srl.read{1,1}.n.type.int32  = 0; % [type: 'int32'  signals / read]
103
104 srl.read{1,1}.n.type.single = 0; % [type: 'single' signals / read]
105 srl.read{1,1}.n.type.double = 0; % [type: 'double' signals / read]
106
107 for i0 = 1 : srl.read{1,1}.n.signals
108
109   % increment counter for appropriate signal type [ - ]
110   switch srl.read{i0,1}.type.original
111   case 'uint8' ; srl.read{1,1}.n.type.uint8  = srl.read{1,1}.n.type.uint8  + 1;
112   case 'uint16'; srl.read{1,1}.n.type.uint16 = srl.read{1,1}.n.type.uint16 + 1;
113   case 'uint32'; srl.read{1,1}.n.type.uint32 = srl.read{1,1}.n.type.uint32 + 1;
114
115   case 'int8'  ; srl.read{1,1}.n.type.int8   = srl.read{1,1}.n.type.int8   + 1;
116   case 'int16' ; srl.read{1,1}.n.type.int16  = srl.read{1,1}.n.type.int16  + 1;
117   case 'int32' ; srl.read{1,1}.n.type.int32  = srl.read{1,1}.n.type.int32  + 1;
118
119   case 'single'; srl.read{1,1}.n.type.single = srl.read{1,1}.n.type.single + 1;
120   case 'double'; srl.read{1,1}.n.type.double = srl.read{1,1}.n.type.double + 1;
```

```
121   otherwise;      error('unknown datatype');
122   end
123
124   switch srl.read{i0,1}.type.original
125   case 'uint8' ; srl.read{i0,1}.n.bytes = 1;       % [ (bytes/signal) / read ]
126   case 'uint16'; srl.read{i0,1}.n.bytes = 2;       % [ (bytes/signal) / read ]
127   case 'uint32'; srl.read{i0,1}.n.bytes = 4;       % [ (bytes/signal) / read ]
128
129   case 'int8'  ; srl.read{i0,1}.n.bytes = 1;       % [ (bytes/signal) / read ]
130   case 'int16' ; srl.read{i0,1}.n.bytes = 2;       % [ (bytes/signal) / read ]
131   case 'int32' ; srl.read{i0,1}.n.bytes = 4;       % [ (bytes/signal) / read ]
132
133   case 'single'; srl.read{i0,1}.n.bytes = 4;       % [ (bytes/signal) / read ]
134   case 'double'; srl.read{i0,1}.n.bytes = 8;       % [ (bytes/signal) / read ]
135   otherwise;      error('unknown datatype');
136   end
137
138   srl.read{i0,1}.n.bits  = srl.read{i0,1}.n.bytes  ...
139                          * k.byte2bit;             % [ (bits /signal) / read ]
140
141   srl.read{1  ,1}.n.Bytes = srl.read{1  ,1}.n.Bytes  ...
142                          + srl.read{i0,1}.n.bytes; % [  bytes         / read ]
143
144 end
145
146   srl.read{1  ,1}. n.Bits = srl.read{1  ,1}.n.Bytes  ...
147                          * k.byte2bit;             % [  bits          / read ]
148
149 % verify number of bytes per read is not greater than arduino input buffer:
150 assert( (srl.read{1,1}.n.Bytes + 1) <= 64              ...
151       , ['Number of bytes being sent per read '        ...
152          '(including 1 byte for Terminator)\n'         ...
153          'is greater than size of\n'                   ...
154          'Arduino Mega 2650 input buffer (64 bytes).'] ...
155       )
156
157 %% [Init   ]: Initialize serial read value vectors
158
159 for i0 = 1 : srl.read{1,1}.n.signals
160 srl.read {i0,1}.val = zeros( srl.read {i0,1}.n.bytes, 1 ); % [varies]
161 end
```

```
162
163 srl.read {1 ,1}.Val = zeros( srl.read {1 ,1}.n.Bytes, 1 ); % [varies]
164
165 %% End
```

Code Listing A.14: [minseg.m]: Initialization - Serial - Read

A.1.3.3.3. General

Code Listing A.15: [minseg.m]: Initialization - Serial - General

```
%% [Init   ]: Define serial communication parameters (general)

% serial address on PC
switch ui.srl.mode.address
  case 0; srl.address = '/dev/tty.usbmodem1411'; % left       usb port (2015 PC)
  case 1; srl.address = '/dev/tty.usbmodem621';  % left-rear  usb port (2008 PC)
end
% note: to determine current address, use command: {ls /dev/tty.*} in Terminal.app

srl.byteOrder         = 'littleEndian';       % [-]
srl.f.baud            =  115200;              % [bit /   s]
srl.T.baud            =  1 / srl.f.baud;      % [  s / bit]

srl.type.in           = 'uint8'; % signal datatype when entering transmission
srl.type.out          = 'uint8'; % signal datatype when exiting  transmission

% legend:
% read  involves a single read  (1   sample ).
% reads involves      all reads (all samples).

%% [Init   ]: Serial buffer size

srl.bufferSize.in  = max( [0; srl.read{1,1}.n.Bits] ); % [bits]
srl.bufferSize.out = srl.bufferSize.in;                % [bits]

% buffer sizes should be equivalent to write or read size (whichever is higher).

%% [Process]: Setup serial object

% Ensure that desired serial port does not already exist in the loaded list:
if ~isempty( instrfind('Port', srl.address) )
  fclose   ( instrfind('Port', srl.address) );
  delete   ( instrfind('Port', srl.address) );
end

% Initialize serial object
srl.srl = serial(                    srl.address        ...
                , 'ByteOrder'       , srl.byteOrder     ...
```

```
                , 'BaudRate'        , srl.f.baud          ... [ Hz ]
                , 'InputBufferSize' , srl.bufferSize.in   ... [bits]
                , 'OutputBufferSize', srl.bufferSize.out  ... [bits]
                );

% For detailed information, use: get(srl.srl)

%{
how prove no "header" value?
how read timeout period? how reduce to something reasonable?

find more information on:

  TimerPeriod = 1
  Timeout     = 10
  StopBits    = 1

%}

%% [Init   ]: Time required to perform transmission

% time required to transmit each write:
srl.write{1,1}.T.transmit = srl.T.baud * (             0             ); % [ s / write ]
                          %  [s / bit] * (       [bit / write]       )

% time required to transmit each read:
srl.read {1,1}.T.transmit = srl.T.baud * ( srl.read{1,1}.n.Bits + 08 ); % [ s / read  ]
                          %  [s / bit] * (        [bit / read]       )
% note: 1 byte (08 bits) added to account for terminator (1 byte).

srl.read {1,1}.T.transmit = srl.read {1,1}.T.transmit * 10 / 08;

% time required to perform all transmissions:
srl.T.transmit = srl.write{1,1}.T.transmit + srl.read{1,1}.T.transmit;  % [ s ]

%% [Init   ]: Time between start of each transmission

% number of board sample periods per serial process period

if ui.srl.T.decimation == 0
srl.T.decimation = ceil( srl.T.transmit * 1.0000 / mdl.T.sample );
```

```
80
81 else
82 srl.T.decimation = ui.srl.T.decimation;
83
84 end
85
86 % time until next serial process:
87 srl.T.sample = mdl.T.sample * srl.T.decimation;                    % [ s ]
88
89 %% [Init   ]: Verify serial period
90
91 % verify that total time to transmit serial data is not greater than
92 %  time until start of next serial process:
93 assert( srl.T.transmit < srl.T.sample                 ...
94       , 'Read period is greater than sample period.' ...
95       );
96
97 %% [Init   ]: Define serial transmits parameters
98
99 % number of reads to perform:
100 % note: serial duration may be specified directly in terms of samples or in terms of time
101 try     srl.n.transmits = round( ui.srl.T.transmits / srl.T.sample ); % [transmit cycles]
102 catch; srl.n.transmits =        ui.srl.n.transmits;                  % [transmit cycles]
103 end
104
105 srl.n.transmits = srl.n.transmits + 1; % 1 added for time = 0
106
107 %% End
```

Code Listing A.15: [minseg.m]: Initialization - Serial - General

A.1.3.3.4. Reads

Code Listing A.16: [minseg.m]: Initialization - Serial - Reads

```
%% [Init   ]: Initialize serial reads variable

srl.reads{ srl.read{1,1}.n.signals, 1 } = [];


%% [Init   ]: Define serial reads parameters

% number of bytes/bits captured after all reads have been performed:
for i0 = 1 : srl.read{1,1}.n.signals
srl.reads{i0,1}.n.bytes = srl.read{i0,1}.n.bytes * srl.n.transmits; % [bytes]
srl.reads{i0,1}.n.bits  = srl.read{i0,1}.n.bits  * srl.n.transmits; % [bits ]
end

srl.reads{1  ,1}.n.Bytes = srl.read{1  ,1}.n.Bytes * srl.n.transmits; % [bytes]
srl.reads{1  ,1}.n.Bits  = srl.read{1  ,1}.n.Bits  * srl.n.transmits; % [bits ]

%% [Init   ]: Initialize serial reads value vectors

for i0 = 1 : srl.read{1,1}.n.signals
srl.reads{i0,1}.val = zeros( srl.reads{i0,1}.n.bytes, 1 ); % [varies]
end

srl.reads{1  ,1}.Val = zeros( srl.reads{1  ,1}.n.Bytes, 1 ); % [varies]

%% End
```

Code Listing A.16: [minseg.m]: Initialization - Serial - Reads

A.1.3.3.5. Build Parameters

Code Listing A.17: [minseg.m]: Initialization - Serial - Model Build Parameters

```
%% [Init   ]: Define serial transmission parameters
io.srl.read.rateTransition.initialCondition = uint8( zeros( srl.read{1,1}.n.Bytes, 1 ) );
io.srl.read.rateTransition.T.sample         = srl.T.sample;

%% [Init   ]: Initialize list of general parameters used within Simulink model

mdl.parameter.label = {};

% Specify parameters which will be used in model:
mdl.parameter.label = [...
mdl.parameter.label
{
  'io.srl.read.rateTransition.initialCondition'
  'io.srl.read.rateTransition.T.sample'

%  cannot set certain hardware parameters via variables. [must hard-code.]

% 'srl.address';
% 'srl.f.baud';
}];

%% [Init   ]: Relabel parameters for use within Simulink model

% Number of parameters specified
mdl.n.parameter         = size( mdl.parameter.label, 1);

% Indices which contain periods:
mdl.parameter.z.period = regexp(mdl.parameter.label, '\.');

% For each parameter:
for i0 = 1 : mdl.n.parameter

% Create a new label in which all periods have been set to underscores:
mdl.parameter.label0 = mdl.parameter.label    {i0,1};
mdl.parameter.label0 ( mdl.parameter.z.period{i0,1} ) = '_';

% Set the data for the new label equal to the data from the old label:
eval([ mdl.parameter.label0 ' = ' mdl.parameter.label{i0,1} ';' ]);
```

```
39
40 end
41
42 %% [Init    ]: Update model scan for errors
43
44 set_param(mdl.label, 'SimulationCommand', 'update')
45
46 %% End
```

Code Listing A.17: [minseg.m]: Initialization - Serial - Model Build Parameters

A.1.4. Processing

A.1.4.1. Build

Code Listing A.18: [minseg.m]: Processing - Build

```
%% [Process]: Build (Normal mode or External mode)
  switch mdl.mode

  case 0 % Normal mode
  disp('Performing build:')
  mdl.T.build   = tic;
  set_param(mdl.label, 'SimulationMode',     'normal'  ) % put model into normal   mode
  rtwbuild (mdl.label                                  ) % build model into hardware
  disp('Build completed.')
  disp(' ')

  case 1 % External mode
  set_param(mdl.label, 'SimulationMode',     'external') % put model into external mode
  set_param(mdl.label, 'SimulationCommand', 'connect' ) % connect to the executable
  set_param(mdl.label, 'SimulationCommand', 'start'   ) % start      the executable
% set_param(mdl.label, 'SimulationCommand', 'stop'    ) % stop       the executable

  end

%% End
```

Code Listing A.18: [minseg.m]: Processing - Build

A.1.4.2. Serial Transmission

Code Listing A.19: [minseg.m]: Processing - Serial - Transmit

```
%% [Process]: Open, read/write, and close serial port object.

% open serial channel
fopen ( srl.srl );

disp( 'Performing serial read:' )

% initialize complete read cycle timers
srl.t.start = clock;
srl.T.all   = tic;

for i0 = 1 : srl.n.transmits
srl.           T.one = tic;


% write
% srl.write{1,1}.T.one = tic;


% read
srl.read{1,1}.T.one = tic;


% perform read of one time sample:
srl.read{1,1}.Val = fread( srl.srl                ... serial object
                         , srl.read{1,1}.n.Bytes  ... read size         [bytes/read]
                         , srl.type.in            ... input data class [default: 'uint8']
                         );

if isempty( srl.read{1,1}.Val ) % occasionally isempty on startup.    [seek better fix.]
srl.read{1,1}.Val = NaN * zeros( srl.read {1 ,1}.n.Bytes, 1 );
end

% append to vector of all reads:
srl.reads{1,1}.Val( ( 1:srl.read{1,1}.n.Bytes ) + (i0-1)*srl.read{1,1}.n.Bytes, 1) = ...
srl.read {1,1}.Val;


```

```
% wait for end of time sample:
if i0 ~= srl.n.transmits                % if not the last sample
  while toc( srl.T.one ) < srl.T.sample % then loop to wait until
  end                                   % a complete sample period
end                                     % has passed before reading
                                        % again.

end

srl.T.all  = toc( srl.T.all );
srl.t.stop = clock;

disp(['Intended total transmit time: ' num2str( srl.n.transmits * srl.T.sample, '%010.6f' )
    ]);
disp(['Actual   total transmit time: ' num2str( srl.T.all                     , '%010.6f' )
    ]);
disp( 'Serial read complete.' )
disp( ' ' )


fclose( srl.srl );


% convert output to intended data type:
srl.read {1,1}.Val = cast( srl.read {1,1}.Val, srl.type.out );
srl.reads{1,1}.Val = cast( srl.reads{1,1}.Val, srl.type.out );
% note: Mathworks forces conversion to 'double' for serial read output.

%% End
```

Code Listing A.19: [minseg.m]: Processing - Serial - Transmit

A.1.4.3. Serial Reads Post-Processing

Code Listing A.20: [minseg.m]: Processing - Serial - Reads

```
%% [Process]: Format serial port data

% Index of first byte of each read
srl.reads{1,1}.z.byte1 = ( 0:srl.n.transmits-1 ).' * srl.read{1,1}.n.Bytes + 1;

srl.read {1,1}.i.byte0 = 0; % initialize byte offset
for i0 = 1 : srl.read{1,1}.n.signals

  % Start index of signal i0 at each sample
   srl.reads{i0,1}.z.byte0   = srl.reads{1,1}.z.byte1 + srl.read{1,1}.i.byte0;

  % Include additional indices for multibyte signals
   if srl.read {i0,1}.n.bytes > 1
      srl.reads{i0,1}.z.byte0 = bsxfun( @plus                         ...
                                      ,   srl.reads{i0,1}.z.byte0     ...
                                      , 0:srl.read {i0,1}.n.bytes-1   ...
                                      );
   end

   % Pull corresponding values
   if strcmp( srl.read{i0,1}.type.original, srl.type.out )
        % If intended signal datatype is     equal to serial output,
        % then use it immediately:
     srl.reads{i0,1}.val  = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );

   else % If intended signal datatype is not equal to serial output,
        % then first convert the serial output:
     srl.reads{i0,1}.val0 = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );

     % Convert to cell:
     srl.reads{i0,1}.val  = mat2cell(             srl.reads{i0,1}.val0          ...
                                    , ones( size( srl.reads{i0,1}.val0, 1 ), 1) ...
                                    ,       size( srl.reads{i0,1}.val0, 2 )     ...
                                    );
     % Typecast each row vector to correct type:
     srl.reads{i0,1}.fun  = @(x) typecast( x, srl.read{i0,1}.type.original );
     srl.reads{i0,1}.val  = cellfun ( srl.reads{i0,1}.fun                      ...
                                    , srl.reads{i0,1}.val                      ...
```

```
39                                      );
40      % Convert back to matrix: [unnecessary - cellfun converts to matrix already]
41      % srl.read{i0,1}.val  = cell2mat( srl.read{i0,1}.val );
42
43      % Determine maximum and minumum values (axis information in plots)
44      srl.reads{i0,1}.val_min    = min( srl.reads{i0,1}.val );
45      srl.reads{i0,1}.val_max    = max( srl.reads{i0,1}.val );
46      srl.reads{i0,1}.val_absMax = max( abs( [ srl.reads{i0,1}.val_min
47                                               srl.reads{i0,1}.val_max ] ) );
48    end
49
50
51
52    % Increment byte offset
53    srl.read{1,1}.i.byte0 = srl.read{1,1}.i.byte0 + srl.read{i0,1}.n.bytes;
54
55  end
56
57  %% End
```

Code Listing A.20: [minseg.m]: Processing - Serial - Reads

A.1.5. Output

A.1.5.1. Save

Code Listing A.21: [minseg.m]: Output - Save

```
%% [Output]: Save all data

file.label = [datestr(now, 'yyyy.mm.dd HH.MM') ' minseg'];

if ~isempty(ui.save.label)
file.label = [ file.label ' ' ui.save.label ];
end

disp( 'Performing export to .mat file.' )

save( [root.data.dir file.label '.mat' ] )

disp( 'Export to .mat file complete.'   )
disp( ' '                               )

%% End
```

Code Listing A.21: [minseg.m]: Output - Save

A.1.5.2. Serial Reads Plot

Code Listing A.22: [minseg.m]: Output - Serial - Reads - Plot

```
%% [Output ]: Common plot commands

%subplot with 2d indices:
dim1      = @(n_col, row, col)         (row-1)*n_col + col; % Matrix index: 2d to 1d
subplott = @(n_row, n_col, M) subplot(n_row, n_col, dim1(n_col, M(1), M(2)) );

% axis value
msd       = @(x)     fix( log10( abs(x) ) );                    % most significant digit. [
    ones digit = 0th digit]
rndout    = @(x, N) sign(x) .* ceil( abs(x)*10^(-N) ) * 10^(+N); % round away from zero at
    specified digit.
rndOut    = @(x, N) rndout(x, msd(x) - N ); % round away from zero at N digits right from
    most significant digit.

%% [Output ]: Plot setup
p = 0;

disp( 'Performing plot creation:' )

for i0 = 1 : srl.read{1,1}.n.signals

  p = p + 1;
  figure(p)

  % plot data
  if i0==1; stairs(                    srl.reads{i0,1}.val, '.-'); % clock
  else;     stairs(srl.reads{1,1}.val, srl.reads{i0,1}.val, '.-'); % all else
  end

  % labels
  if i0==1; xlabel( 'Samples [-]' ); % clock
  else;     xlabel( 'Time [s]'    ); % all else
  end

  ylabel( srl.read{i0,1}.label )

  % y-axis limits
  if isa(srl.reads{i0,1}.val, 'float') % float
```

```
36      srl.reads{i0,1}.ymin = -rndOut(srl.reads{i0,1}.val_absMax+eps, 2);
37      srl.reads{i0,1}.ymax = +rndOut(srl.reads{i0,1}.val_absMax+eps, 2);
38    else                                  % integer
39      srl.reads{i0,1}.ymin = double( intmin( class(srl.reads{i0,1}.val) ) );
40      srl.reads{i0,1}.ymax = double( intmax( class(srl.reads{i0,1}.val) ) );
41    end
42
43    ylim([ srl.reads{i0,1}.ymin, srl.reads{i0,1}.ymax ])
44
45    grid minor
46
47  end
48
49  disp( 'Plot creation complete.'   )
50  disp( ' '                         )
51
52
53  %% [Output ]: Close legacy figures
54
55  % not yet implemented.
56  % use "a = get(groot, 'Children')" to list all figures.
57  % then "for all figures: if n.figure > n.figure.gcf, close n.figure"
58
59  % when implemented, stop using "close all"
60
61  %% End
```

Code Listing A.22: [minseg.m]: Output - Serial - Reads - Plot

A.1.6. Global Cleanup

Code Listing A.23: [minseg.m]: Global Cleanup

```
%% [Cleanup]: Remove alternate subdirectories from Matlab path

Simulink.fileGenControl('reset')

%% [Cleanup]: Remove alternate subdirectories from Simulink path

for i0 = 1 : root.n.sub.dir
  rmpath(    root.   sub.dir{ root.n.sub.dir - (i0 - 1), 1 } )
end

%% End
```

Code Listing A.23: [minseg.m]: Global Cleanup

Bibliography

[1] Y. Yamamoto. (May 1, 2009). "Nxtway-gs (self-balancing two-wheeled robot) controller design," [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs--self-balancing-two-wheeled-robot--controller-design (visited on 07/03/2017).

[2] R. J. Vaccaro, *Digital Control: A State-space Approach*, ser. Mcgraw Hill Series in Electrical and Computer Engineering. Mcgraw-Hill College, Jan. 1995, ISBN: 978-0070667815.

[3] Wikipedia. (Jun. 24, 2017). "Inverted pendulum," Wikimedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/Inverted_pendulum (visited on 07/03/2017).

[4] D. Melanson. (Jul. 14, 2016). "Researchers create life-saving ubot-5 robot, play dress-up with it," Engadget, [Online]. Available: https://www.engadget.com/2008/04/17/researchers-create-life-saving-ubot-5-robot-play-dress-up-with/ (visited on 02/05/2017).

[5] B. Howard and L. Bushnell, "Enhancing linear system theory curriculum with an inverted pendulum robot," in *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, IEEE, Hangzhou, China, Oct. 2015. DOI: 10.1109/CSMA.2015.63.

[6] Mathworks. "Arduino programming with matlab and simulink," [Online]. Available: https://www.mathworks.com/discovery/arduino-programming-matlab-simulink.html (visited on 07/03/2017).

[7] J. Hurst. (Jun. 17, 2016). "Rensselaer arduino support package library (rasplib)," 1.1, Rensselaer Polytechnic Institute (RPI), [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/62702-rensselaer-arduino-support-package-library--rasplib- (visited on 07/03/2017).

[8] MinSeg. "M2v3," [Online]. Available: https://minseg.com/collections/minseg-kits/products/minsegshield-kit-m2v3-2-dual-drive-segway-and-line-follower (visited on 07/03/2017).

[9] Wikipedia. (Jul. 1, 2017). "Inverted pendulum," Wikimedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/Arduino (visited on 07/03/2017).

[10] jkenny23. (May 9, 2011). "Crystal vs. resonator," Arduino, [Online]. Available: https://forum.arduino.cc/index.php?topic=60662.0 (visited on 07/03/2017).

[11] Arduino Geeks. (Feb. 2, 2017). "Arduino mega vs uno compared," Arduino Starter Kits, [Online]. Available: https://www.arduinostarterkits.com/reviews/arduino-mega-vs-uno-compared/ (visited on 07/03/2017).

[12] Reichelt Elektronik. "Arduino mega: Arduino mega 2560, atmega1280, usb," [Online]. Available: https://www.reichelt.com/de/en/Single-board-microcontroller/ARDUINO-MEGA/3/index.html?ACTION=3&GROUPID=6667&ARTICLE=119696 (visited on 07/03/2017).

[13] Alberto "PighiXXX". (Aug. 4, 2014). "Mega," PighiXXX, [Online]. Available: http://www.pighixxx.com/test/portfolio-items/mega/ (visited on 07/03/2017).

[14] PowerStream. (Jun. 29, 2017). "Discharge tests of aa batteries, alkaline and nimh," [Online]. Available: https://www.powerstream.com/AA-tests.htm (visited on 07/03/2017).

[15] Texas Instruments. (Jan. 2015). "Sn754410: Quadruple h drivers," [Online]. Available: http://www.ti.com/product/sn754410?qgpn=sn754410 (visited on 07/03/2017).

[16] P. "Philo" Hurbain. (May 15, 2017). "Nxt motor internals," [Online]. Available: http://www.philohome.com/nxtmotor/nxtmotor.htm (visited on 07/03/2017).

[17] Everymac. (May 22, 2017). "Apple macbook pro "core i5" 2.7 13" early 2015 specs," [Online]. Available: http://www.everymac.com/systems/apple/macbook_pro/specs/macbook-pro-core-i5-2.7-13-early-2015-retina-display-specs.html (visited on 07/03/2017).

[18] Mathworks. "Previous releases: System requirements and supported compilers," [Online]. Available: https://www.mathworks.com/support/sysreq/previous_releases.html (visited on 07/03/2017).

[19] M. D. Peltier, "Trajectory control of a two-wheeled robot," M.S. thesis, University of Rhode Island (URI): Department of Electrical Engineering, Jan. 2012.

[20] F. L. Lewis and V. L. Syrmos, *Optimal Control*, Second. Wiley-Interscience, Nov. 1995, ISBN: 978-0-471-03378-3.

[21] (Oct. 3, 2016). "Lego wheels chart," [Online]. Available: http://wheels.sariel.pl/ (visited on 07/03/2017).

[22] R. P. M. Chan, K. A. Stol, and C. R. Halkyard, "Review of modelling and control of two-wheeled robots," *Annual Reviews in Control*, vol. 37, no. 1, pp. 89–103, Apr. 2013.

[23] D. R. Jones and K. A. Stol, "Modelling and stability control of two-wheeled robots in low-traction environments," in *Australasian Conference on Robotics and Automation (ACRA) 2010*, G. Wyeth and B. Upcroft, Eds., Australian Robotics & Automation Association (ARAA), Brisbane, Australia, Dec. 2010.

[24] O. Jamil, M. Jamil, Y. Ayaz, and A. Khubab, "Modeling, control of a two-wheeled self-balancing robot," in *2014 International Coneference on Robotics and Emerging Allied Technologies in Engineering (iCREATE)*, IEEE, Islamabad, Pakistan, Apr. 2014. DOI: 10.1109/iCREATE.2014.6828364.

[25] R. J. Vaccaro, "An optimization approach to the pole-placement design of robust linear multivariable control systems," in *2014 American Control Conference (ACC)*, ACC, Portland, OR, USA, Jun. 2014. DOI: 10.1109/ACC.2014.6858987.

[26] Y. Gong, H. Ma, and X. Wu, "Research on control strategy of two-wheeled self-balancing robot," in *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, IEEE, Hangzhou, China, Oct. 2015. DOI: 10.1109/CSMA.2015.63.

[27] R. J. Vaccaro, Personal Interview, University of Rhode Island (URI): Department of Electrical Engineering, Jan. 23, 2017.

[28] MinSeg. "Homepage," [Online]. Available: https://minseg.com/ (visited on 07/03/2017).

[29] Mathworks. "Homepage," [Online]. Available: https://www.mathworks.com/ (visited on 07/03/2017).

[30] A. De Luca, G. Oriolo, and M. Vendittelli, "Control of wheeled mobile robots: An experimental overview," in *Ramsete: Articulated and Mobile Robotics for Services and Technologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 181–226, ISBN: 978-3-540-45000-9. DOI: 10.1007/3-540-45000-9_8. [Online]. Available: http://dx.doi.org/10.1007/3-540-45000-9_8.

[31] M. R. Bageant, *Balancing a two-wheeled segway robot*, Undergraduate Thesis, Oct. 2011.

[32] A. R. da Silva Jr, "Design and control of a two-wheeled robotic walker," M.S. thesis, University of Massachusettes (UMass): Department of Mechanical and Industrial Engineering, May 2014.