

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.
- Mark 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

Abstract	ii
Acknowledgements	iii
Dedication	iv
Table of Contents	iv
List of Figures	viii
List of Tables	x
List of Code Listings	xi
1. Introduction	1
1.1. Purpose	1
1.2. Statement of the Problem	2
1.3. Methodology	3
1.4. Bibliography	5
2. Preliminary Decisions	6
2.1. Selection of Control Problem	6
2.1.1. Inverted Pendulum	6
2.1.2. Two-Wheeled Robot	7
2.2. Selection of Hardware	9
2.2.1. MinSeg M2V3 Two-Wheeled Robot	10
2.2.2. Development PC	25
2.3. Selection of a Hardware Model	28
2.4. Selection of Controller Design	29
2.5. Bibliography	30
3. Test Platform	32
3.1. Simulink: minseg_M2V3_2017a.slx	33
3.1.1. Root	33
3.2. Matlab: minseg.m	36

4. Hardware-Equivalent Dynamics Model	37
4.1. Nonlinear model	40
4.1.1. Coordinate System	40
4.2. Differential Equations	42
4.2.1. Wheel Angular Position θ and Body Pitch ϕ_x	42
4.2.2. Body Yaw ϕ_y	43
4.3. State-Space Representation	44
4.4. Calculation of Nonintuitive Parameters	47
4.4.1. Moment of Inertia: Body: X-axis (Pitch) J_{ϕ_x}	47
4.4.2. Moment of Inertia: Body: Y-axis (Yaw) J_{ϕ_y}	47
4.4.3. Length From Body Center of Mass to Body Axis of Rotation $l_{b,c2a}$	48
4.5. Bibliography	52
5. Control Design	53
5.1. Additional Dynamics	53
5.1.1. Background	53
5.1.2. Tracking System	66
5.1.3. Control Gains	68
5.2. Bibliography	77
6. Conclusions	78
6.1. Future work	78
6.2. Bibliography	83
Appendices	84
A. Code Listings	85
A.1. minseg.m	85
A.1.1. Global Setup	88
A.1.2. User Inputs	90
A.1.3. Initialization	92
A.1.4. Processing	132
A.1.5. Output	138
A.1.6. Global Cleanup	142

Bibliography	143
---------------------	-----

List of Figures

2.1	[Selection of Control Problem]: Inverted Pendulum on Cart	6
2.2	[Selection of Control Problem]: Two Wheeled Robot	8
2.3	[Selection of Compatible HW & SW]: MinSeg M2V3 (Multiview)	11
2.4	[Selection of Compatible HW & SW]: MinSeg M2V3 (Exploded)	12
2.5	[Selection of Compatible HW & SW]: Arduino Mega 2560 (Isometric View)	15
2.6	[Selection of Compatible HW & SW]: Arduino Mega 2560 (Top View)	15
2.7	[Selection of Compatible HW & SW]: Arduino Mega 2560 (Pin Map)	16
2.8	[Selection of Compatible HW & SW]: AA-Battery Voltage During Constant Discharge	17
2.9	[Selection of Compatible HW & SW]: Motor Driver Pin Map	19
2.10	[Selection of Compatible HW & SW]: Lego NXT Motor (3D Model)	22
2.11	[Selection of Compatible HW & SW]: Lego NXT Motor (Exploded)	22
2.12	[Selection of Compatible HW & SW]: Lego NXT Motor Gear Teeth Map	23
3.1	[Simulink]: Root	34
4.1	[Hardware-Equivalent Physical Dynamics Model]: Isometric	38
4.2	[Hardware-Equivalent Physical Dynamics Model]: Multiview	38
5.1	[Additional Dynamics]: State Feedback Regulator	55
5.2	[Additional Dynamics]: 1.0 Additional Dynamics (Design View)	56
5.3	[Additional Dynamics]: 1.1 Additional Dynamics (Design View)	57
5.4	[Additional Dynamics]: 1.2 Additional Dynamics (State Feedback Regulator View) .	58
5.5	[Additional Dynamics]: 2.0 Additional Dynamics (Split Gains)	59
5.6	[Additional Dynamics]: 3.0 Additional Dynamics (Linear View: Plant)	60
5.7	[Additional Dynamics]: 3.1 Additional Dynamics (Linear View: Plant)	61
5.8	[Additional Dynamics]: 4.0 Additional Dynamics (Linear View: Controller)	62
5.9	[Additional Dynamics]: 4.0 Additional Dynamics with Reference Signal (Linear View: User)	63
5.10	[Additional Dynamics]: 4.0 Additional Dynamics with Reference Signal (Linear View: User)	64
5.11	[Control Gains: LQR]: Simulation Results: Wheel Angular Position θ	70

5.12	[Control Gains: LQR]: Simulation Results: Body Angular Position ϕ_y	71
5.13	[Control Gains: LQR]: Simulation Results: Body Angular Position ϕ_x	72
5.14	[Control Gains: LQR]: Simulation Results: Wheel Linear Position p_x	73
5.15	[Control Gains: LQR]: Simulation Results: Wheel Linear Position p_y	74
5.16	[Control Gains: LQR]: Simulation Results: Wheel Linear Position p_{xy}	75
5.17	[Control Gains: LQR]: Simulation Results: Motor Driver Commanded Voltage $v_{motorDriver}$	76

List of Tables

2.1	[Selection of Compatible HW & SW]: MinSeg Components	13
2.2	[Selection of Compatible HW & SW]: Arduino Board Comparison	14
2.3	[Selection of Compatible HW & SW]: Motor Driver Pin Legend	19
2.4	[Selection of Compatible HW & SW]: Motor Driver Pin Function Legend	20
2.5	[Selection of Compatible HW & SW]: Motor Driver Operating Conditions	20
2.6	[Selection of Compatible HW & SW]: Motor Driver Switching Characteristics	20
2.7	[Selection of Compatible HW & SW]: Lego NXT Motor Figure Legend	23
2.8	[Selection of Compatible HW & SW]: Development PC Specifications	25
4.1	[Hardware-Equivalent Physical Dynamics Model]: Variables	39
4.2	[Hardware-Equivalent Physical Dynamics Model]: Parameters	39

List of Code Listings

A.1 [minseg.m]: Root file	85
A.2 [minseg.m]: Global Setup	88
A.3 [minseg.m]: User Inputs	90
A.4 [minseg.m]: Initialization - General	93
A.5 [minseg.m]: Initialization - Model - General	95
A.6 [minseg.m]: Initialization - Model - Plant	96
A.7 [minseg.m]: Initialization - Model - Plant - Hardware	98
A.8 [minseg.m]: Initialization - Model - Plant - Nonlinear Dynamics Model	103
A.9 [minseg.m]: Initialization - Model - Plant - Linear Dynamics Model	104
A.10 [minseg.m]: Initialization - Model - Controller	111
A.11 [minseg.m]: Initialization - Model - User-Defined Board Inputs and Outputs	112
A.12 [minseg.m]: Initialization - Model - Model Build Parameters	114
A.13 [minseg.m]: Initialization - Serial - Write	120
A.14 [minseg.m]: Initialization - Serial - Read	121
A.15 [minseg.m]: Initialization - Serial - General	126
A.16 [minseg.m]: Initialization - Serial - Reads	129
A.17 [minseg.m]: Initialization - Serial - Model Build Parameters	130
A.18 [minseg.m]: Processing - Build	133
A.19 [minseg.m]: Processing - Serial - Transmit	134
A.20 [minseg.m]: Processing - Serial - Reads	136
A.21 [minseg.m]: Output - Save	139
A.22 [minseg.m]: Output - Serial - Reads - Plot	140
A.23 [minseg.m]: Global Cleanup	142

CHAPTER 1.

Introduction

1.1. Purpose

The intent of this thesis is as follows:

1. Select a well-established control problem as a focal point. [Inverted Pendulum: Two-wheeled Robot]
2. Develop a modular test platform, such that differing control methods could be independently applied to the selected control problem, in real-time and in simulation.

[MinSeg Two-Wheeled Robot and Mathworks Software Suite]

3. Select or derive the dynamic equations for the physical model, and populate it. [Yamamoto [1]]
4. Select the first controller design to address the selected control problem on the test platform, and implement it in simulation. [Optimal Controller]
5. Time permitting, implement the same controller design on the hardware.

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

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.

- Minimize 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 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).

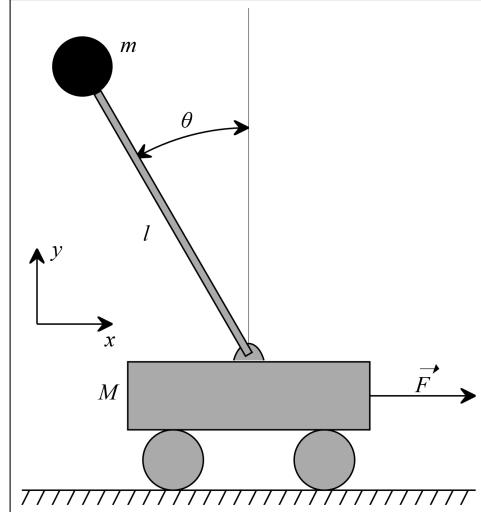


Figure 2.1.: [Selection of Control Problem]: Inverted Pendulum on Cart [3]

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	<i>One-dimensional (single, straight line only)</i>
2	<i>Two-dimensional</i>

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 4.4.3.2].

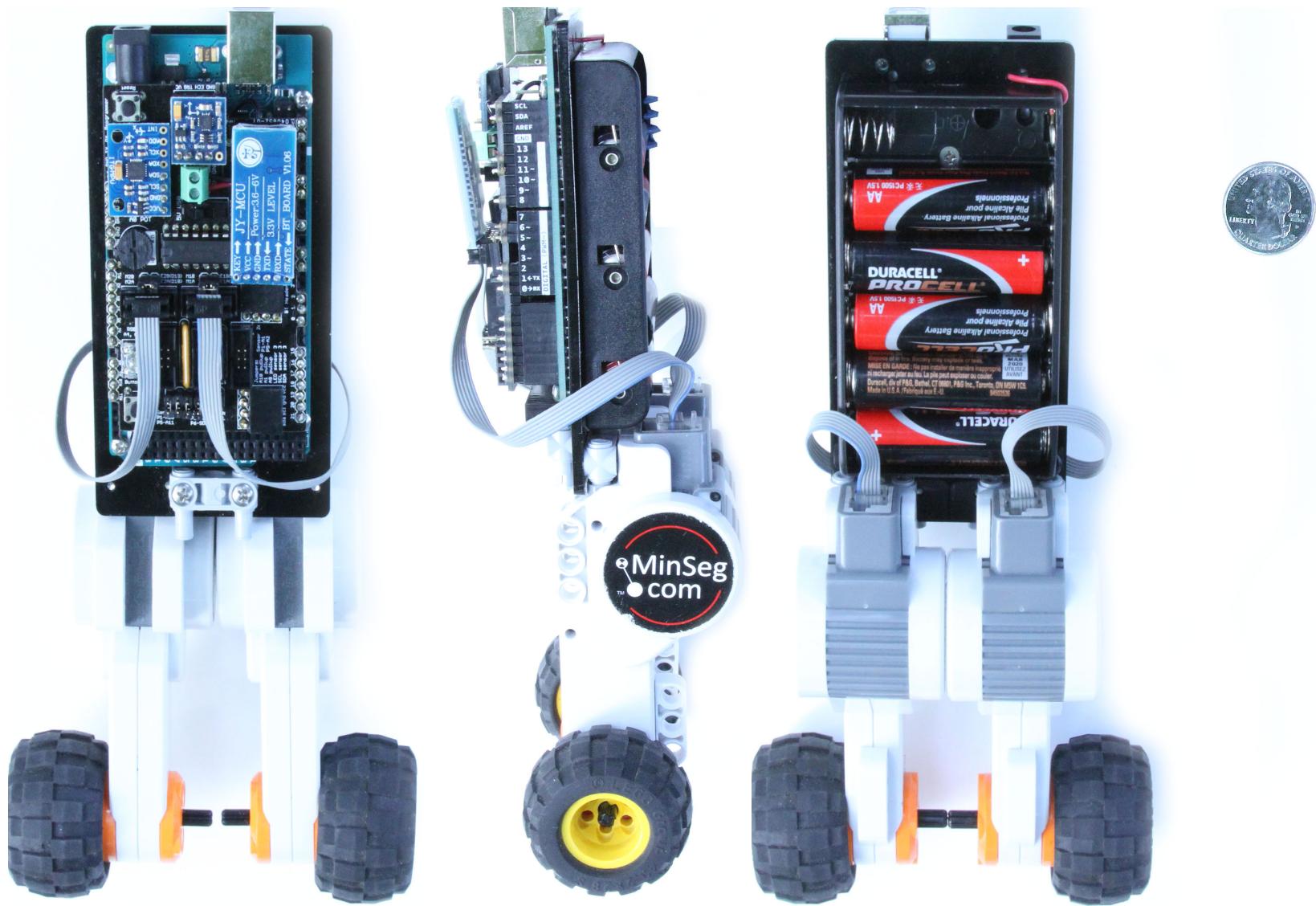


Figure 2.3.: [Selection of Compatible HW & SW]: MinSeg M2V3 (Multiview)

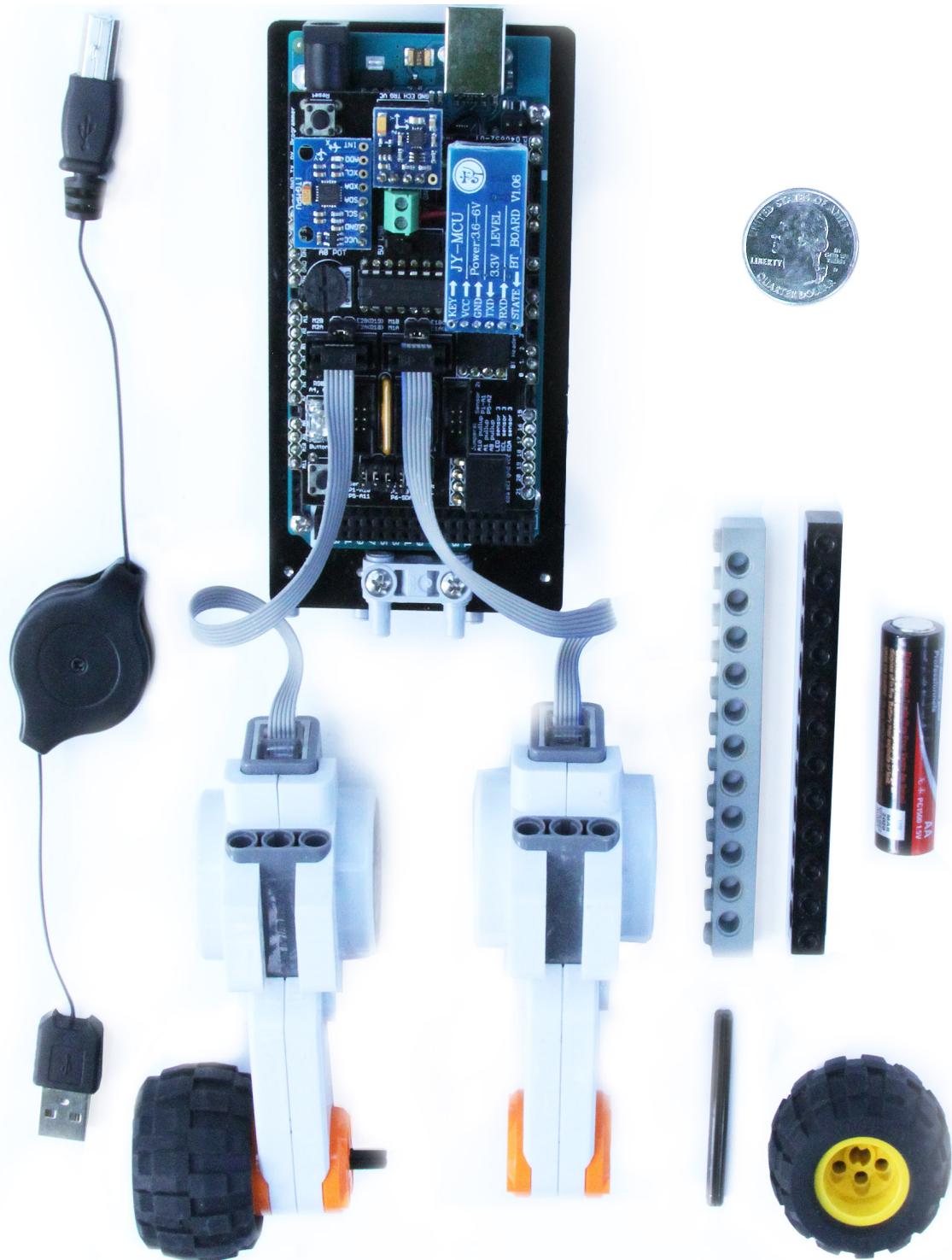


Figure 2.4.: [Selection of Compatible HW & SW]: MinSeg M2V3 (Exploded)

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 <i>[Includes: 1x DC motor, 1x gearbox, 1x encoder]</i>	
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:	<ul style="list-style-type: none"> - 2x Encoder [See motor.] 1x 3-axis gyroscope & accelerometer [1x MPU6050] 	
The angular velocity of the wheels		
The 3-dimensional position of the body		
PC Communication (<i>during operation</i>):	<ul style="list-style-type: none"> - Serial: USB [Default. Included with microcontroller.] Serial: Bluetooth [Supported, but sold separately.] 	
Power Source:	<ul style="list-style-type: none"> - USB 5 [V] Battery holster (6x AA) 9 [V] 	
<i>Wired</i>		
<i>Wireless</i>		

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	-
<i>Memory/Storage:</i>			
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.

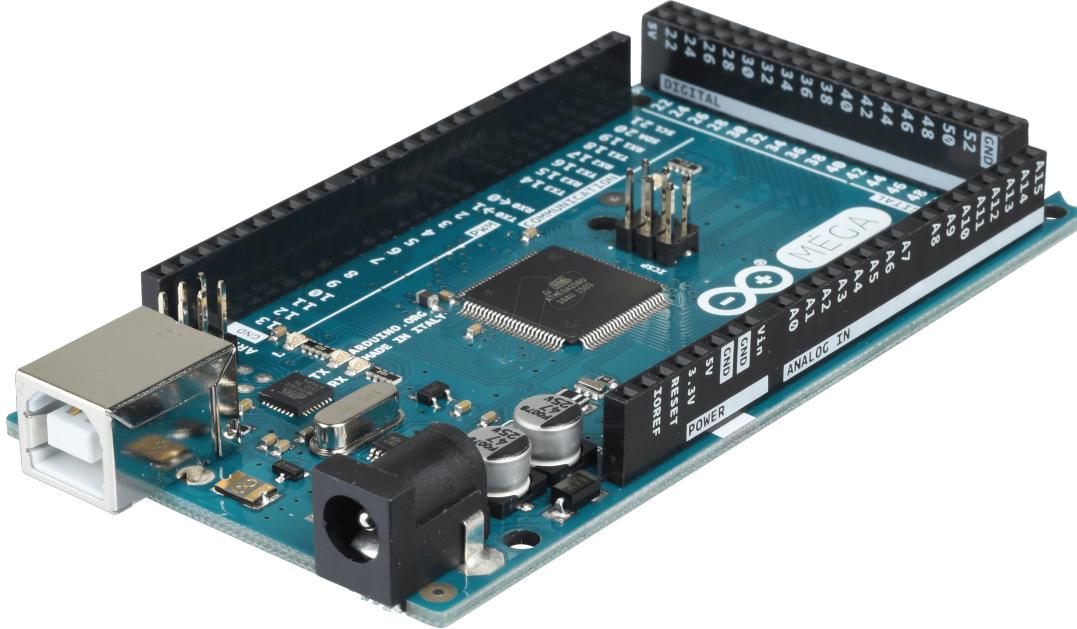


Figure 2.5.: [Selection of Compatible HW & SW]: Arduino Mega 2560 (Isometric View) [12]

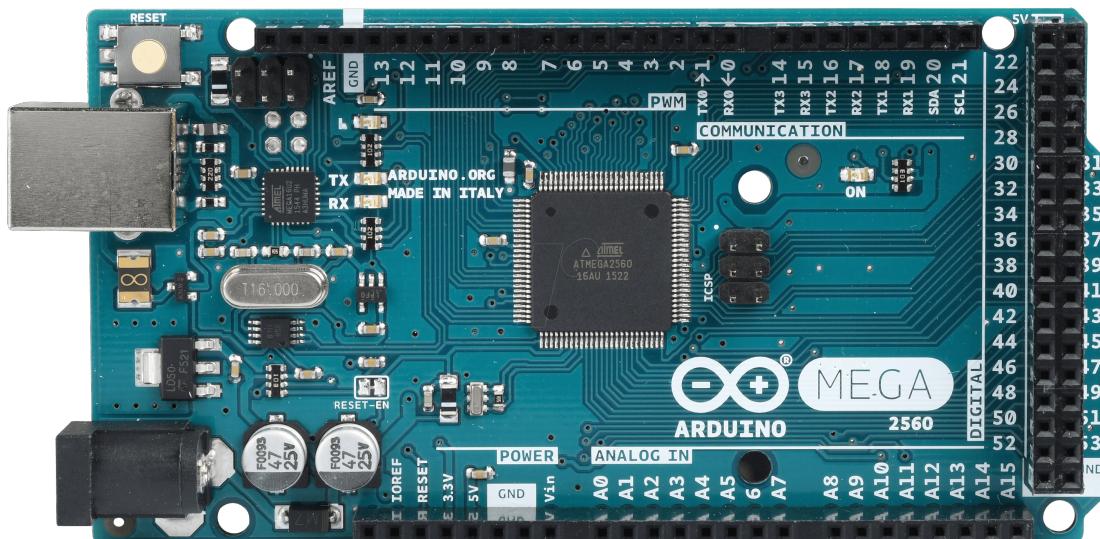
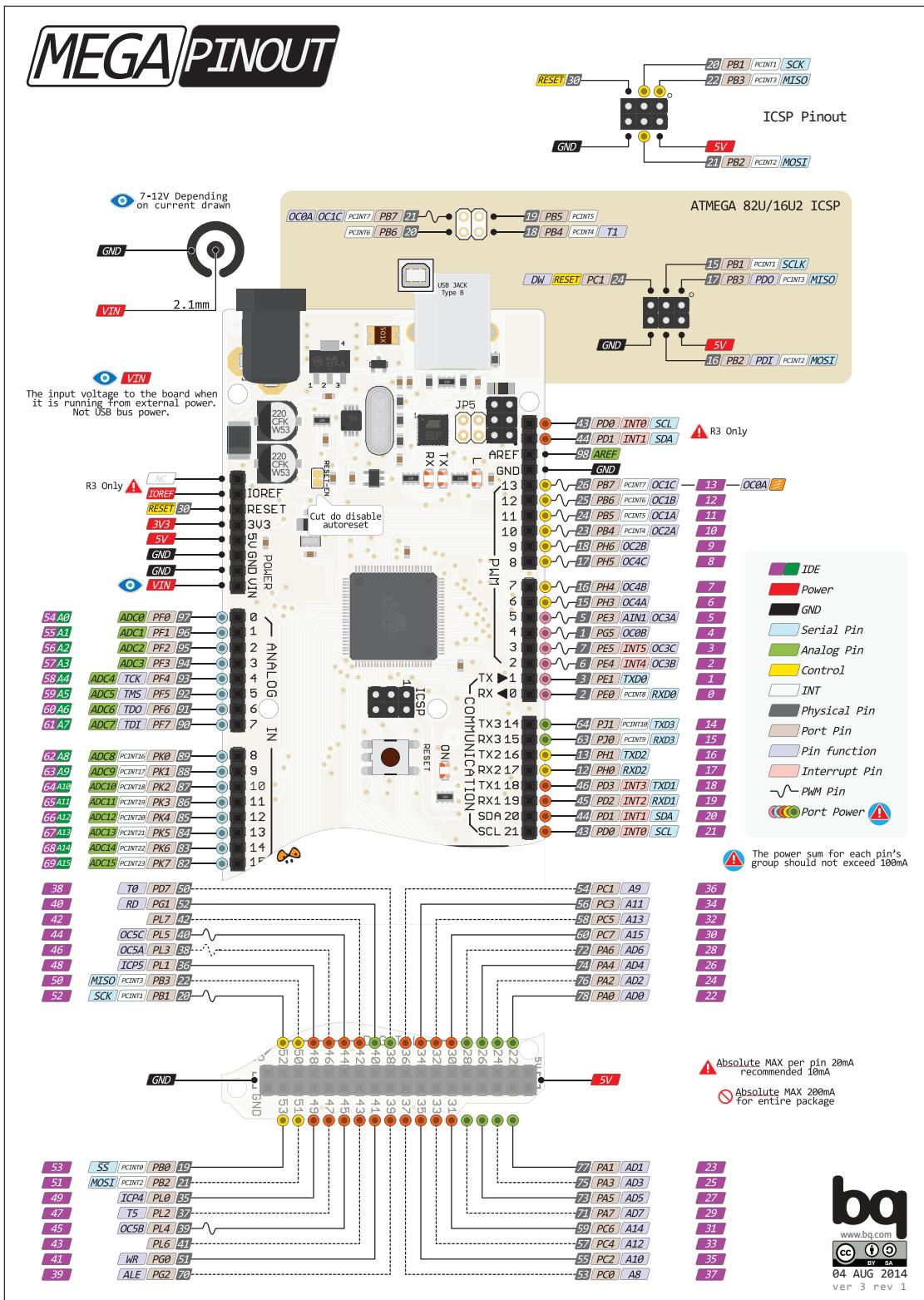


Figure 2.6.: [Selection of Compatible HW & SW]: Arduino Mega 2560 (Top View) [12]



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.

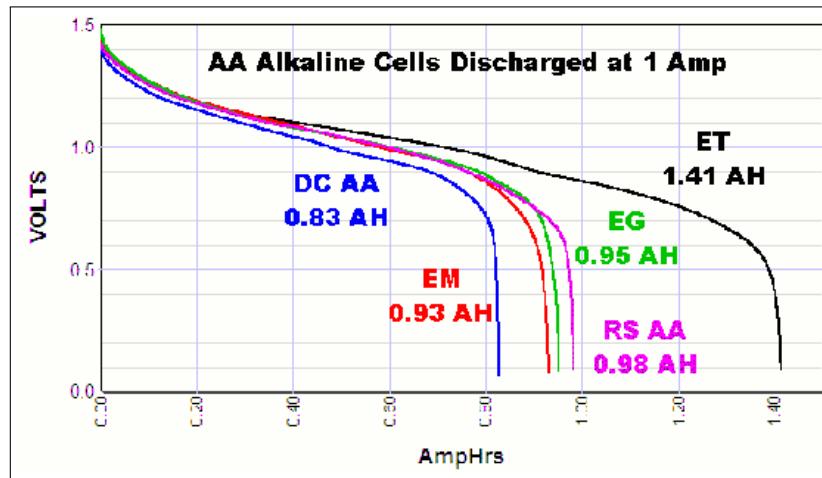


Figure 2.8.: [Selection of Compatible HW & SW]: AA-Battery Voltage During Constant Discharge [14]

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, \sim 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 0 [V].

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

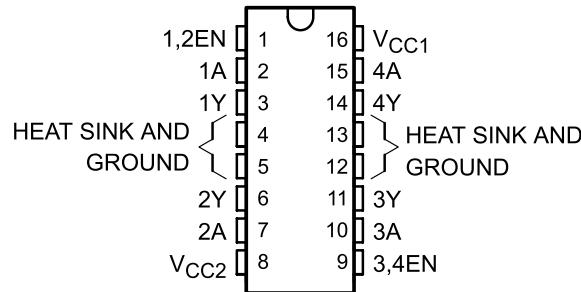


Figure 2.9.: [Selection of Compatible HW & SW]: Motor Driver Pin Map [15]

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 Y
A	EN	
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}	See Figure 3		400		ns
t _{d2}			800		ns
t _{TLH}			300		ns
t _{THL}			300		ns
t _{en1}	See Figure 4		700		ns
t _{en2}			400		ns
t _{dis1}			900		ns
t _{dis2}			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.

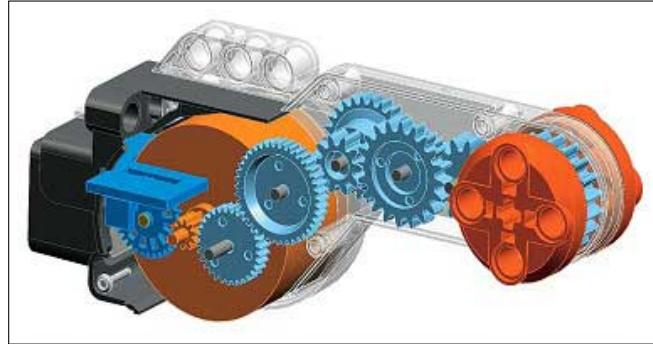


Figure 2.10.: [Selection of Compatible HW & SW]: Lego NXT Motor (3D Model) [16]

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.

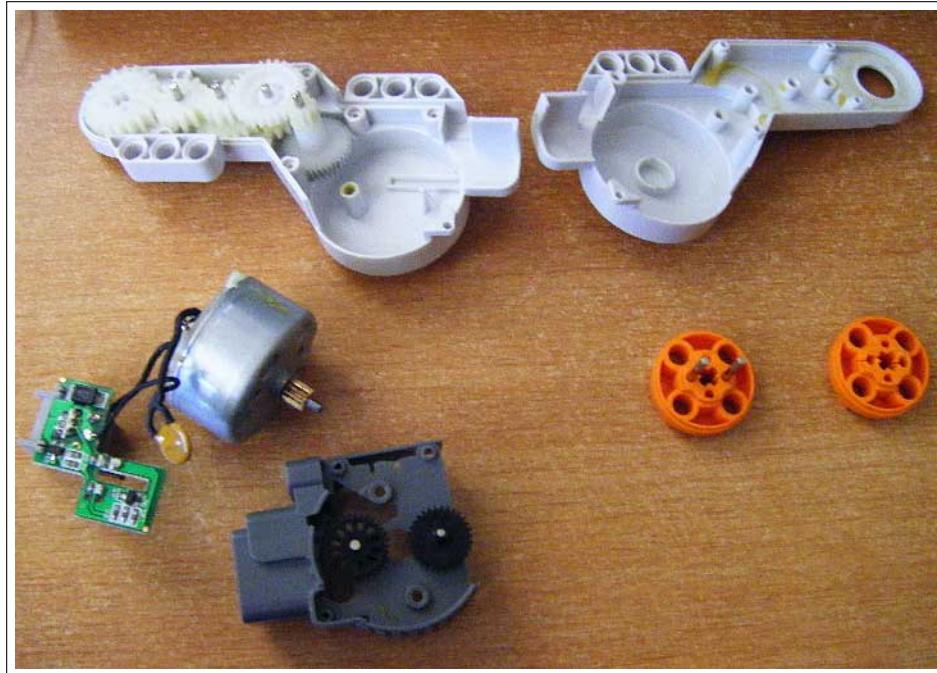


Figure 2.11.: [Selection of Compatible HW & SW]: Lego NXT Motor (Exploded) [16]

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	-	-	-
<i>Enclosure</i>	Dark blue	Bottom	Black
<i>Gearing</i>	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 number(s) of teeth between each component, as exhibited in Eqn. [2.1], where k represents a ratio and n represents an integer count.

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

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].

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

$$k_{gearTeeth \text{ } motor2encoder} = \frac{32}{10} \quad (2.3)$$

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

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

$$k_{\omega \text{ } wheel2encoder} = k_{\omega \text{ } wheel2motor} \cdot k_{\omega \text{ } motor2encoder} = \frac{15}{01} \quad (2.6)$$

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 · <i>MATLAB</i> · <i>Simulink</i> · <i>Control System Toolbox</i> · <i>DSP System Toolbox</i> · <i>Instrument Control Toolbox</i> · <i>MATLAB Coder</i> · <i>Simulink Coder</i> · <i>Simulink Desktop Real-Time</i> · <i>Matlab Support Package for Arduino Hardware</i> · <i>Simulink Support Package for Arduino Hardware</i>	r2017a - - - - - - - - - - 17.1.0 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 channel would also be required.

It is not expected that such a transition would be preventatively 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 Section ??.

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 5.

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-rensselear-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.arduino-starterkits.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?ggpn=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.

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.

3.1. 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.

3.1.1. Root

The top level of the model, also known as the model root, is depicted in Figure 3.1.

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

- Plant
- Controller
- Board Inputs and Outputs

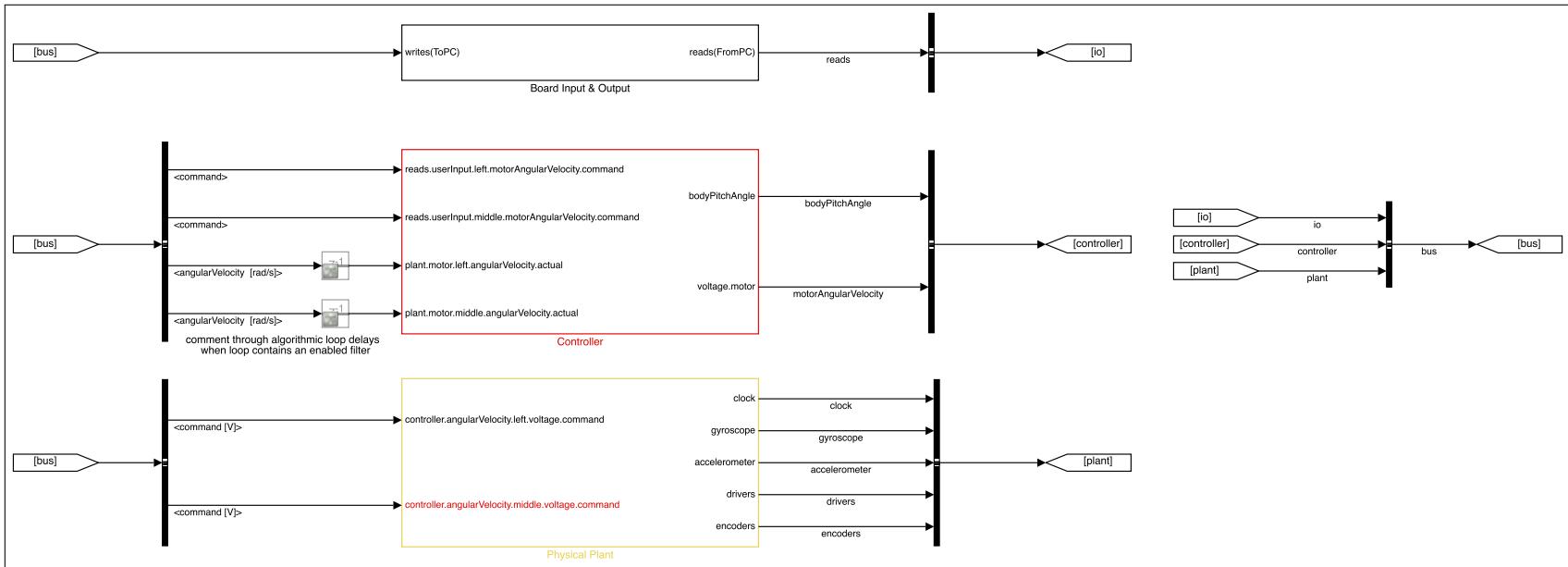


Figure 3.1.: [Simulink]: Root

3.1.1.1. 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 3.1 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*].

3.1.1.2. 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.

3.2. 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 hierachal, 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 4.

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 4.1 - 4.2, 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 4.2.

Yamamoto [1] makes the following assumptions in Figures 4.1 - 4.2:

- All mass geometries are uniform.
- All masses are uniformly distributed.
- The hardware consists of three principal masses:
 - A rectangular cuboid
 - A cylinder
 - A cylinder

[*The body.*]

[*The left wheel.*]

[*The right wheel.*]

Tables 4.1 - 4.2, define the variables and the parameters, respectively, that the physical model of the two-wheeled inverted pendulum will use.

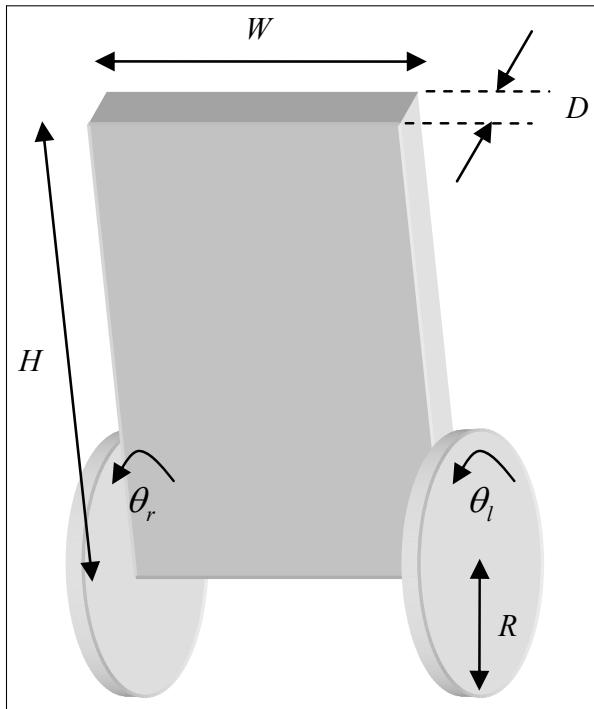


Figure 4.1.: [Hardware-Equivalent Physical Dynamics Model]: Isometric[1]

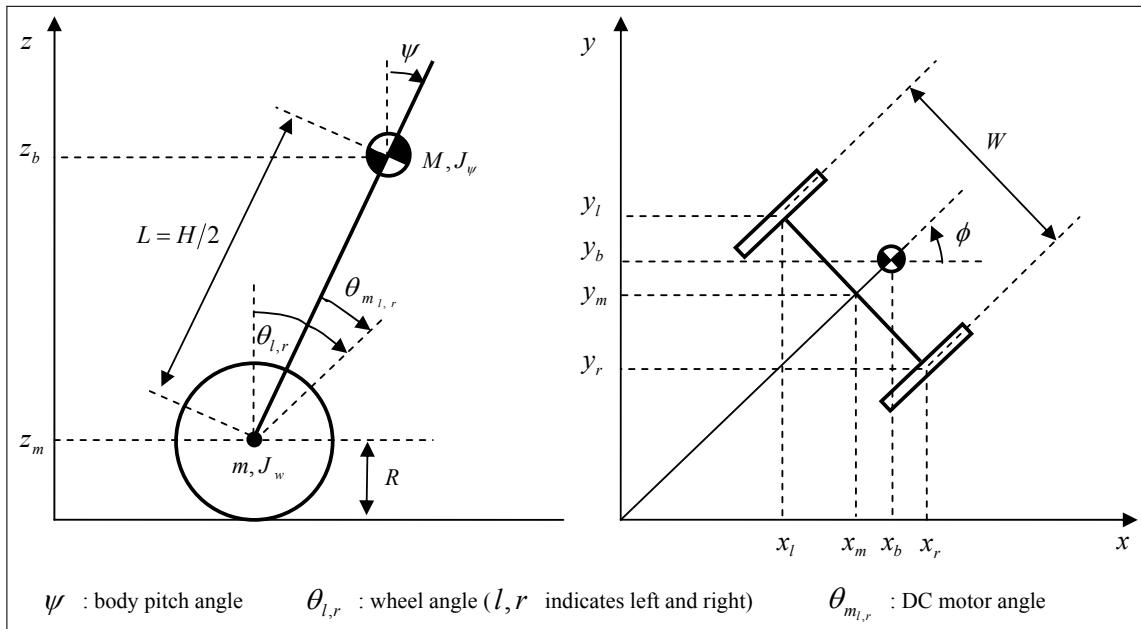


Figure 4.2.: [Hardware-Equivalent Physical Dynamics Model]: Multiview[1]

Table 4.1.: [Simulink]: Root

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

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

Table 4.2.: [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	??	m	-
$l_{b,w}$	Length: Body: Width	??	m	-
$l_{b,d}$	Length: Body: Depth	??	m	-
$l_{b,c2a}$	Length: Body: [Center of mass] to [Axis of Rotation]	??	m	Sec. 4.4.3
r_w	Length: Wheel: Radius	0.021	m	[5]
J_{b,ϕ_x}	Moment of Inertia: Wheel	$7.46 \cdot 10^{-6}$	$kg \cdot m^2$	[5]
J_{b,ϕ_x}	Moment of Inertia: Body: X-axis (pitch)	??	$kg \cdot m^2$	Sec. 4.4.1
J_{b,ϕ_y}	Moment of Inertia: Body: Y-axis (yaw)	??	$kg \cdot m^2$	Sec. 4.4.2
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]	??	-	Sec. ??

4.1. 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 4.2.

4.1.1. Coordinate System

The coordinate system is explicitly defined in Equations (4.1) - (4.6).

$$\begin{bmatrix} \theta_{g,l} \\ \theta_{g,r} \\ \theta_{g,av} \\ \phi_y \end{bmatrix} = \begin{bmatrix} \theta_{b,l} + \phi_x \\ \theta_{b,r} + \phi_x \\ \frac{1}{2} \cdot (\theta_{g,l} + \theta_{g,r}) \\ \frac{r_w}{l_{b,w}} \cdot (\theta_{g,r} - \theta_{g,l}) \end{bmatrix} \quad (4.1)$$

$$\begin{bmatrix} \dot{p}_{w,x} \\ \dot{p}_{w,y} \\ \dot{p}_{w,z} \end{bmatrix} = \begin{bmatrix} r_w \cdot \dot{\theta}_{g,av} \cdot \cos(\phi_y) \\ r_w \cdot \dot{\theta}_{g,av} \cdot \sin(\phi_y) \\ 0 \end{bmatrix} \quad (4.2)$$

$$\begin{bmatrix} p_{w,x} \\ p_{w,y} \\ p_{w,z} \end{bmatrix} = \begin{bmatrix} \int \dot{p}_{w,x} \cdot dt + p_{w,x}(0) \\ \int \dot{p}_{w,y} \cdot dt + p_{w,y}(0) \\ \int \dot{p}_{w,z} \cdot dt + p_{w,z}(0) \end{bmatrix} \quad (4.3)$$

$$\begin{bmatrix} p_{wl,x} \\ p_{wl,y} \\ p_{wl,z} \end{bmatrix} = \begin{bmatrix} p_{w,x} - \frac{l_{b,w}}{2} \cdot \sin(\phi_y) \\ p_{w,x} + \frac{l_{b,w}}{2} \cdot \cos(\phi_y) \\ p_{w,z} \end{bmatrix} \quad \begin{bmatrix} p_{wr,x} \\ p_{wr,y} \\ p_{wr,z} \end{bmatrix} = \begin{bmatrix} p_{w,x} + \frac{l_{b,w}}{2} \cdot \sin(\phi_y) \\ p_{w,x} - \frac{l_{b,w}}{2} \cdot \cos(\phi_y) \\ p_{w,z} \end{bmatrix} \quad (4.4)$$

$$\begin{bmatrix} p_{b,x} \\ p_{b,y} \\ p_{b,z} \end{bmatrix} = \begin{bmatrix} p_{w,x} + l_{b,c2a} \cdot \sin(\phi_x) \cdot \cos(\phi_y) \\ p_{w,y} + l_{b,c2a} \cdot \sin(\phi_x) \cdot \sin(\phi_y) \\ p_{w,z} + l_{b,c2a} \cdot \cos(\phi_x) \end{bmatrix} \quad (4.5)$$

Typically, initial conditions are assumed to be as follows:

$$\begin{bmatrix} p_{w,x}(0) \\ p_{w,y}(0) \\ p_{w,z}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ r_w \end{bmatrix} \quad (4.6)$$

4.2. Differential Equations

After creating a nonlinear model using the Lagrangian method, and then linearizing that model, Yamamoto [1] provides the differential equations (4.7) and (4.18), [*and their abbreviated term definitions*].

4.2.1. Wheel Angular Position θ and Body Pitch ϕ_x

Equation (4.7) corresponds to wheel angular position θ and body pitch ϕ_x .

$$\mathbf{K}_{1.\ddot{x}} \cdot \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.\dot{x}} \cdot \begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.x} \cdot \begin{bmatrix} \theta \\ \phi_x \end{bmatrix} = \mathbf{K}_{1.v} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \quad (4.7)$$

$$\mathbf{K}_{1.\ddot{x}} = \begin{bmatrix} +k_{1.1} & +k_{1.2} \\ +k_{1.2} & +k_{1.3} \end{bmatrix} \quad (4.8)$$

$$\mathbf{K}_{1.\dot{x}} = \begin{bmatrix} +k_{1.4} & -k_{1.4} \\ -k_{1.4} & +k_{1.4} \end{bmatrix} \quad (4.9)$$

$$\mathbf{K}_{1.x} = \begin{bmatrix} 0 & 0 \\ 0 & +k_{1.5} \end{bmatrix} \quad (4.10)$$

$$\mathbf{K}_{1.v} = \begin{bmatrix} +k_{1.6} & +k_{1.6} \\ -k_{1.6} & -k_{1.6} \end{bmatrix} \quad (4.11)$$

$$k_{1.1} = \left(2 \cdot m_w + m_b \right) \cdot r_w + J_w \quad (4.12)$$

$$k_{1.2} = m_b \cdot r_w \cdot l_{b.c2a} \quad (4.13)$$

$$k_{1.3} = m_b \cdot l_{b.c2a}^2 + J_{b.\phi_x} \quad (4.14)$$

$$k_{1.4} = 2 \cdot \left(\frac{k_{mtr.T} \cdot k_{mtr.bEMF}}{R_{mtr}} + k_{fr.m2w} \right) \quad (4.15)$$

$$k_{1.5} = -m_b \cdot a_g \cdot l_{b.c2a} \quad (4.16)$$

$$k_{1.6} = \frac{k_{mtr.T}}{R_{mtr}} \quad (4.17)$$

4.2.2. Body Yaw ϕ_y

Equation (4.18) corresponds to body yaw ϕ_y .

$$k_{2.\ddot{x}} \cdot \left[\ddot{\phi}_y \right] + k_{2.\dot{x}} \cdot \left[\dot{\phi}_y \right] = k_{2.v} \cdot \left[v_{mtr.r} - v_{mtr.l} \right] \quad (4.18)$$

$$k_{2.0} = \frac{l_{b.w}}{r_w} \quad (4.19)$$

$$k_{2.\ddot{x}} = \frac{1}{2} \cdot m_w \cdot l_{b.w}^2 + \frac{1}{2} \cdot k_{2.0}^2 \cdot J_{b.\phi_y} \quad (4.20)$$

$$k_{2.\dot{x}} = \frac{1}{2} \cdot k_{2.0}^2 \cdot k_{1.4} \quad (4.21)$$

$$k_{2.v} = \frac{1}{2} \cdot k_{2.0} \cdot k_{1.6} \quad (4.22)$$

4.3. State-Space Representation

The general form of state-space representation is exhibited in Equation (4.23).

$$\begin{aligned}\dot{\mathbf{x}}_{nx1} &= \mathbf{A}_{nxn} \cdot \mathbf{x}_{nx1} + \mathbf{B}_{nxp} \cdot \mathbf{u}_{px1} \\ \mathbf{y}_{mx1} &= \mathbf{C}_{m \times n} \cdot \mathbf{x}_{nx1} + \mathbf{D}_{m \times p} \cdot \mathbf{u}_{px1}\end{aligned}\quad (4.23)$$

The designated x states and p inputs are exhibited in Equations (4.24) - (4.25).

$$\mathbf{x}_{nx1} = \begin{bmatrix} \theta \\ \phi_x \\ \dot{\theta} \\ \dot{\phi}_x \\ \phi_y \\ \dot{\phi}_y \end{bmatrix} \quad (4.24)$$

$$\mathbf{u}_{px1} = \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \quad (4.25)$$

The derivation of indices for the system matrices \mathbf{A} and \mathbf{B} which are nonintuitive are derived from Equations (4.7) - (4.18). in Equations (4.26) - (4.27).

$$\begin{aligned} \mathbf{K}_{1.\ddot{x}} \cdot \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.\dot{x}} \cdot \begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix} + \mathbf{K}_{1.x} \cdot \begin{bmatrix} \theta \\ \phi_x \end{bmatrix} &= \mathbf{K}_{1.v} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \\ \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix} &= \underbrace{-\mathbf{K}_{1.\ddot{x}}^{-1} \cdot \mathbf{K}_{1.\dot{x}}}_{\mathbf{A}_1} \cdot \begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix} + \underbrace{-\mathbf{K}_{1.\ddot{x}}^{-1} \cdot \mathbf{K}_{1.x}}_{\mathbf{A}_0} \cdot \begin{bmatrix} \theta \\ \phi_x \end{bmatrix} + \underbrace{\mathbf{K}_{1.\ddot{x}}^{-1} \cdot \mathbf{K}_{1.v}}_{\mathbf{B}_1} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \end{aligned} \quad (4.26)$$

$$\begin{aligned} k_{2.\ddot{x}} \cdot \begin{bmatrix} \ddot{\phi}_y \end{bmatrix} + k_{2.\dot{x}} \cdot \begin{bmatrix} \dot{\phi}_y \end{bmatrix} &= k_{2.v} \cdot \begin{bmatrix} v_{mtr.r} - v_{mtr.l} \end{bmatrix} \\ \begin{bmatrix} \ddot{\phi}_y \end{bmatrix} &= \underbrace{-k_{2.\ddot{x}}^{-1} \cdot k_{2.\dot{x}}}_{\mathbf{A}_2} \cdot \begin{bmatrix} \dot{\phi}_y \end{bmatrix} + \underbrace{k_{2.\ddot{x}}^{-1} \cdot k_{2.v}}_{\mathbf{B}_2} \cdot \begin{bmatrix} v_{mtr.r} - v_{mtr.l} \end{bmatrix} \end{aligned} \quad (4.27)$$

Note that $K_{1.\ddot{x}}$ must be invertible to perform the second step in in Equation (4.26). The derivation for matrix invertibility and the proof that $K_{1.\ddot{x}}$ is nonsingular [*and is therefore invertible*], are exhibited in Equations (4.28) - (4.31).

$$\mathbf{X}_{2x2} = \begin{bmatrix} +X_{(1,1)} & +X_{(1,2)} \\ +X_{(2,1)} & +X_{(2,2)} \end{bmatrix} \quad (4.28)$$

$$\mathbf{X}^{-1} = \frac{1}{\det(\mathbf{X})} \cdot \text{adj}(\mathbf{X}) = \frac{1}{X_{(1,1)} \cdot X_{(2,2)} - X_{(1,2)} \cdot X_{(2,1)}} \cdot \begin{bmatrix} +X_{(2,2)} & -X_{(2,1)} \\ -X_{(1,2)} & +X_{(1,1)} \end{bmatrix} \quad (4.29)$$

$$\det(\mathbf{X}) \neq 0 \quad (4.30)$$

$$\det(\mathbf{K}_{1.\ddot{x}}) = k_{1.1} \cdot k_{1.3} - k_{1.2} \cdot k_{1.2} \neq 0 \quad (4.31)$$

The **A** matrix and the state vector **x** are exhibited in Equation (4.32).

$$\begin{array}{c} \mathbf{A}_{n \times n} \\ \cdot \\ \mathbf{x}_{n \times 1} \end{array} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \mathbf{A}_0_{(1,1)} & \mathbf{A}_0_{(1,2)} & \mathbf{A}_1_{(1,1)} & \mathbf{A}_1_{(1,2)} & 0 & 0 \\ \mathbf{A}_0_{(2,1)} & \mathbf{A}_0_{(2,2)} & \mathbf{A}_1_{(2,1)} & \mathbf{A}_1_{(2,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & A_2 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \phi_x \\ \dot{\theta} \\ \dot{\phi}_x \\ \phi_y \\ \dot{\phi}_y \end{bmatrix} \quad (4.32)$$

The **B** matrix and the input vector **u** are exhibited in Equation (4.33).

$$\begin{array}{c} \mathbf{B}_{n \times p} \\ \cdot \\ \mathbf{u}_{p \times 1} \end{array} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \mathbf{B}_1_{(1,1)} & \mathbf{B}_1_{(1,2)} \\ \mathbf{B}_1_{(2,1)} & \mathbf{B}_1_{(2,2)} \\ 0 & 0 \\ -B_2 & +B_2 \end{bmatrix} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \quad (4.33)$$

The **C** matrix and the state vector **x** are exhibited in Equation (4.34).

$$\begin{array}{c} \mathbf{C}_{m \times n} \\ \cdot \\ \mathbf{x}_{n \times 1} \end{array} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \phi_x \\ \dot{\theta} \\ \dot{\phi}_x \\ \phi_y \\ \dot{\phi}_y \end{bmatrix} \quad (4.34)$$

The **D** matrix and the input vector **u** are exhibited in Equation (4.35).

$$\begin{array}{c} \mathbf{D}_{m \times p} \\ \cdot \\ \mathbf{u}_{p \times 1} \end{array} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix} \quad (4.35)$$

4.4. 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.

4.4.1. 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 (4.36).

$$J_{\phi_x} = \frac{m_b \cdot l_{b,c2a}^2}{3} \quad (4.36)$$

4.4.2. 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 (4.37).

$$J_{\phi_y} = \frac{m_b \cdot (l_{b,w}^2 + l_{b,d}^2)}{12} \quad (4.37)$$

4.4.3. 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.

4.4.3.1. Yamamoto Method

As seen in Figure 4.1 [*on page* 38], 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 (4.38)

$$l_{b.c2a} = \frac{l_{b.h}}{2} \quad (4.38)$$

4.4.3.2. 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 (4.38)

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 (4.39) - (4.40) become true.

$$\theta = \phi_x \quad (4.39)$$

$$\mathbf{u} = \mathbf{0} \quad (4.40)$$

The effects of these changes are exhibited in Equation (4.42) [on page 50]. This results in two relations, which are exhibited in Equation (4.41).

$$\begin{aligned} \ddot{\phi}_x &+ 0 = 0 \\ \ddot{\phi}_x + \underbrace{\frac{k_{1.5}}{k_{1.2} + k_{1.3}}}_{k_\omega} \cdot \dot{\phi}_x &= 0 \Leftarrow \end{aligned} \quad (4.41)$$

Of the two resulting relations in Equation (4.41), 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.

$$\begin{aligned}
& \underbrace{\mathbf{K}_{1.\ddot{x}} \cdot \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi}_x \end{bmatrix}}_{\text{---}} + \underbrace{\mathbf{K}_{1.\dot{x}} \cdot \begin{bmatrix} \dot{\theta} \\ \dot{\phi}_x \end{bmatrix}}_{\text{---}} + \underbrace{\mathbf{K}_{1.x} \cdot \begin{bmatrix} \theta \\ \phi_x \end{bmatrix}}_{\text{---}} = \underbrace{\mathbf{K}_{1.v} \cdot \begin{bmatrix} v_{mtr.l} \\ v_{mtr.r} \end{bmatrix}}_{\text{---}} \\
& \underbrace{\mathbf{K}_{1.\ddot{x}} \cdot \begin{bmatrix} \ddot{\phi}_x \\ \ddot{\phi}_x \end{bmatrix}}_{\text{---}} + \underbrace{\mathbf{K}_{1.\dot{x}} \cdot \begin{bmatrix} \dot{\phi}_x \\ \dot{\phi}_x \end{bmatrix}}_{\text{---}} + \underbrace{\mathbf{K}_{1.x} \cdot \begin{bmatrix} \phi_x \\ \phi_x \end{bmatrix}}_{\text{---}} = \underbrace{\mathbf{K}_{1.v} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\text{---}} \quad (4.42) \\
& \underbrace{\begin{bmatrix} k_{1.1} & k_{1.2} \\ k_{1.2} & k_{1.3} \end{bmatrix} \cdot \begin{bmatrix} \ddot{\phi}_x \\ \ddot{\phi}_x \end{bmatrix}}_{\text{---}} + \underbrace{2 \cdot k_{1.4} \cdot \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi}_x \\ \dot{\phi}_x \end{bmatrix}}_{\text{---}} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & k_{1.5} \end{bmatrix} \cdot \begin{bmatrix} \phi_x \\ \phi_x \end{bmatrix}}_{\text{---}} = \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\text{---}} \\
& \underbrace{\begin{bmatrix} (k_{1.1} + k_{1.2}) \cdot \ddot{\phi}_x \\ (k_{1.2} + k_{1.3}) \cdot \ddot{\phi}_x \end{bmatrix}}_{\text{---}} + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\text{---}} + \underbrace{\begin{bmatrix} 0 \\ k_{1.5} \cdot \phi_x \end{bmatrix}}_{\text{---}} = \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\text{---}}
\end{aligned}$$

The coefficient term, abbreviated as k_w , is expanded in Equation (4.43). It may be expanded further with the use of Equation (4.36), as exhibited in Equation (4.44).

$$k_\omega = \frac{k_{1.5}}{k_{1.2} + k_{1.3}} = \frac{-m_b \cdot a_g \cdot l_{b.c2a}}{\left(m_b \cdot r_w \cdot l_{b.c2a} \right) + \left(m_b \cdot l_{b.c2a}^2 + J_{b.\phi_x} \right)} \quad (4.43)$$

$$k_\omega = \frac{-m_b \cdot l_{b.c2a} \cdot a_g}{m_b \cdot l_{b.c2a} \cdot r_w + m_b \cdot l_{b.c2a}^2 + \left(m_b \cdot l_{b.c2a}^2 \cdot \frac{1}{3} \right)} = \frac{-a_g}{r_w + l_{b.c2a} \cdot \left(1 + \frac{1}{3} \right)} \quad (4.44)$$

Harmonic Oscillator

Notably, the selected relation in Equation (4.41) form-matches the equation for a harmonic oscillator [2, p. 119 - 120, 122 - 123], as is exhibited in Equation (4.45).

$$\begin{aligned} \ddot{y} + \omega^2 \cdot y &= \omega^2 \cdot u \\ \ddot{\phi}_x + k_\omega \cdot \phi_x &= k_\omega \cdot 0 \end{aligned} \quad (4.45)$$

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 (4.46).

$$\omega_p^2 = k_\omega = \frac{-a_g}{r_w + l_{b.c2a} \cdot \frac{4}{3}} \quad (4.46)$$

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 (4.47).

$$l_{b.c2a} = -\frac{3}{4} \cdot \left(\frac{a_g}{\omega_p^2} + r_w \right) = -\frac{3}{4} \cdot \left(\frac{a_g}{\left(2 \cdot \pi \cdot f_p \right)^2} + r_w \right) \quad (4.47)$$

4.5. 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," Master's thesis, University of Rhode Island (URI): Department of Electrical Engineering, Jan. 2012.

CHAPTER 5.

Control Design

5.1. 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.

5.1.1. Background

A state feedback regulating system is depicted in Figure 5.1. It contains the hardware plant as well as the inverted system feedback gains.

The *additional dynamics* are added in Figure 5.2. 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 5.3. 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 5.4. 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 5.5]
2. The gains are shifted such that they are forward facing. [Figure 5.6]

- In this configuration, the original plant may be separated from the components used to control it, including the additional dynamics. [Figure 5.7]
3. The plant is shifted to the right side of the system depiction. [Figure 5.8]

GT

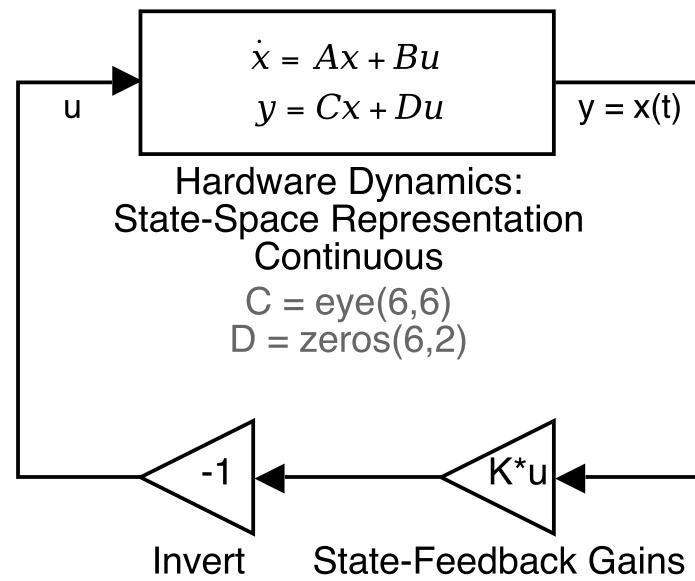


Figure 5.1.: [Additional Dynamics]: State Feedback Regulator

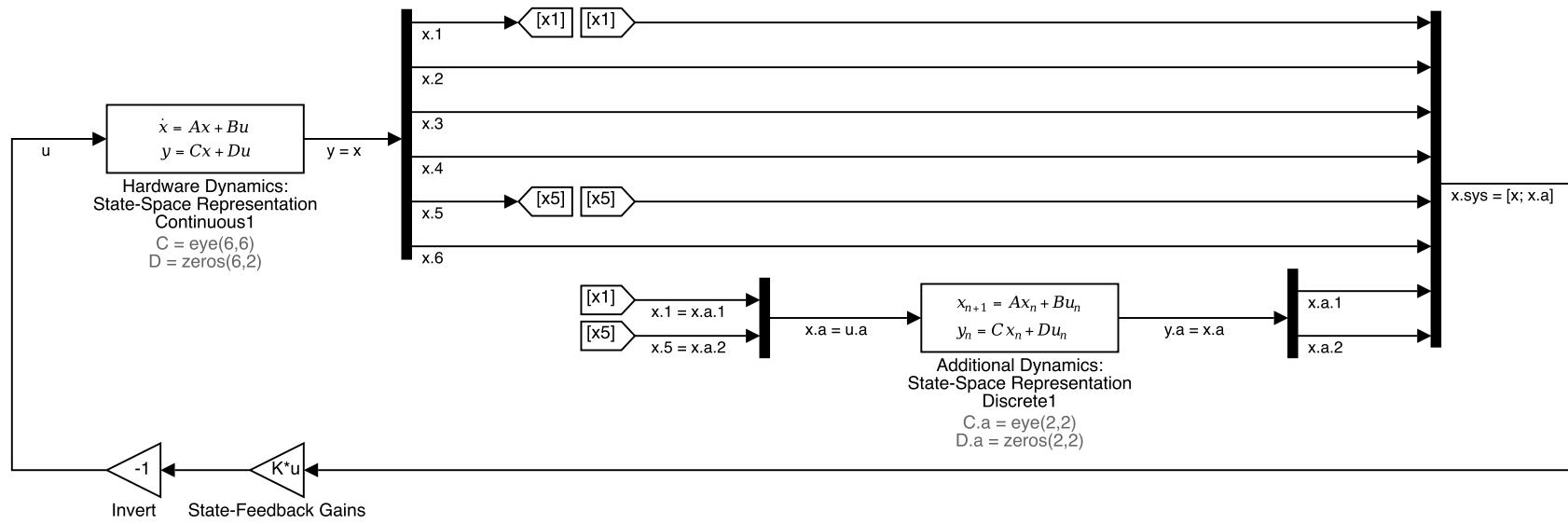


Figure 5.2.: [Additional Dynamics]: 1.0 Additional Dynamics (Design View)

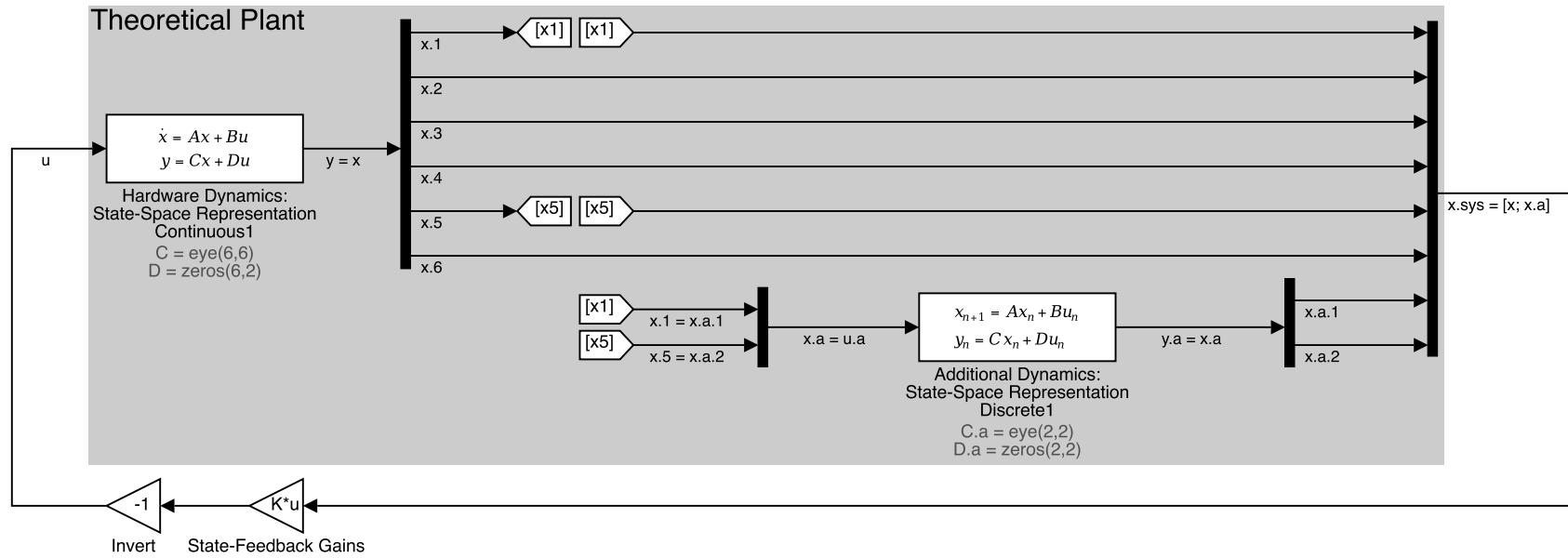


Figure 5.3.: [Additional Dynamics]: 1.1 Additional Dynamics (Design View)

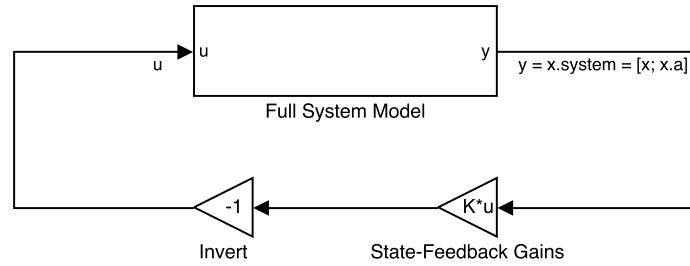


Figure 5.4.: [Additional Dynamics]: 1.2 Additional Dynamics (State Feedback Regulator View)

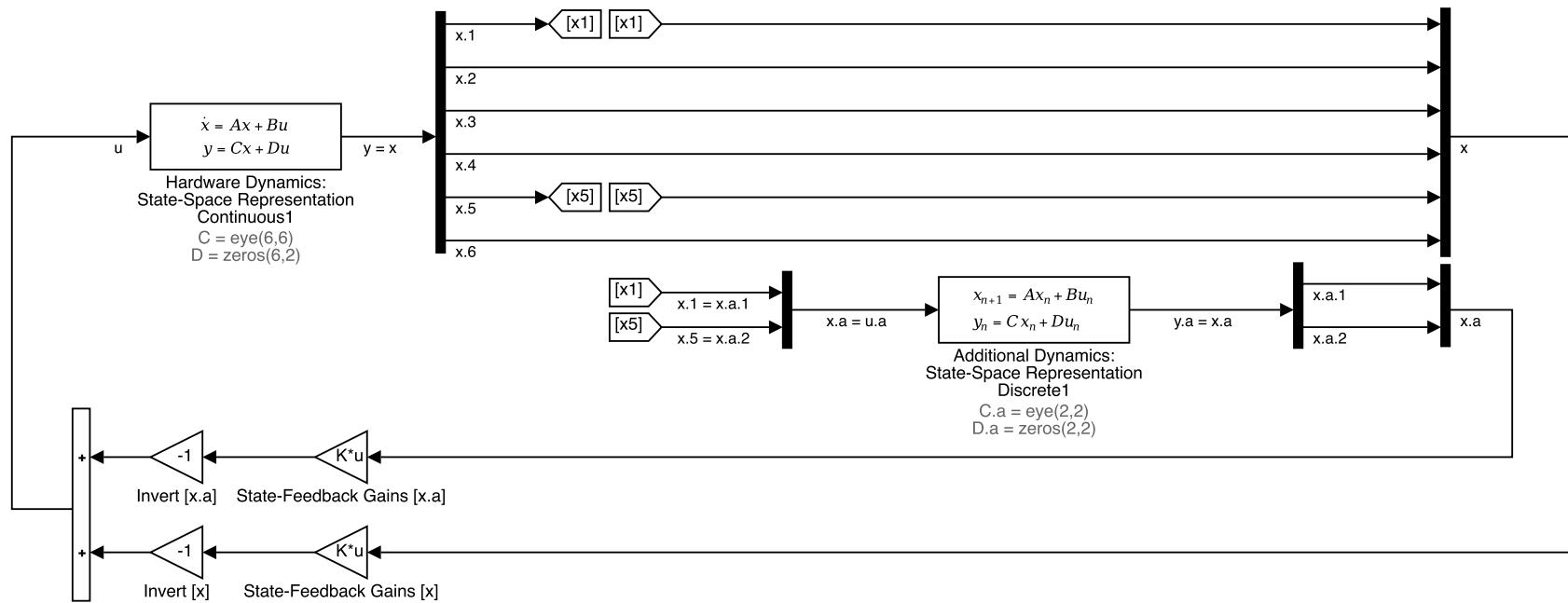


Figure 5.5.: [Additional Dynamics]: 2.0 Additional Dynamics (Split Gains)

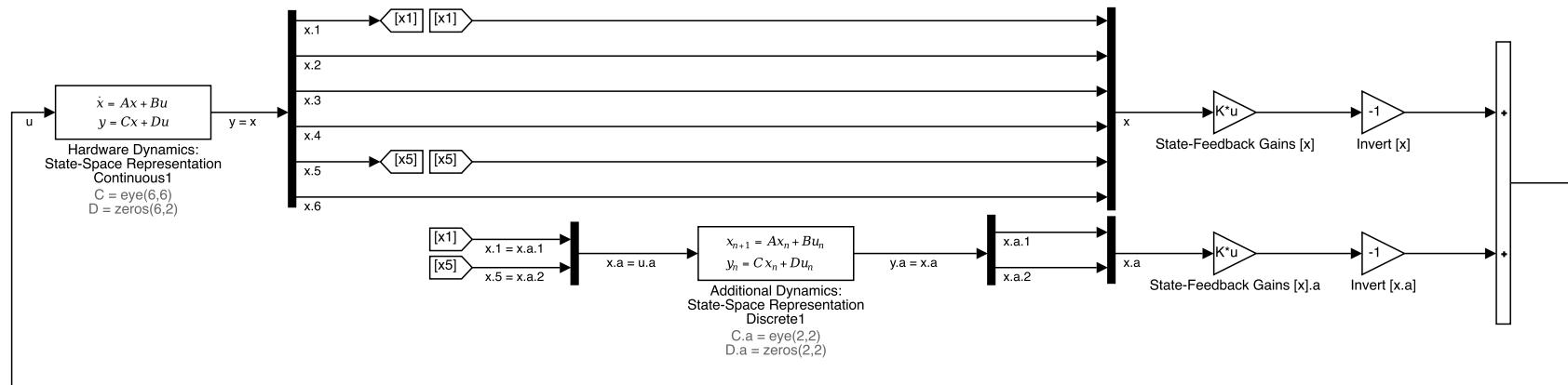


Figure 5.6.: [Additional Dynamics]: 3.0 Additional Dynamics (Linear View: Plant)

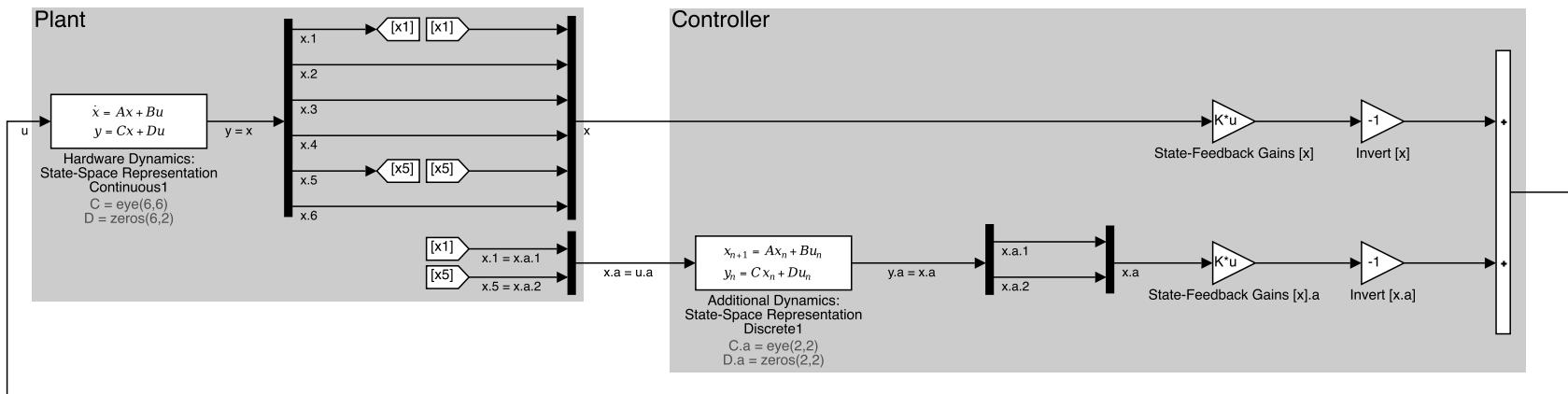


Figure 5.7.: [Additional Dynamics]: 3.1 Additional Dynamics (Linear View: Plant)

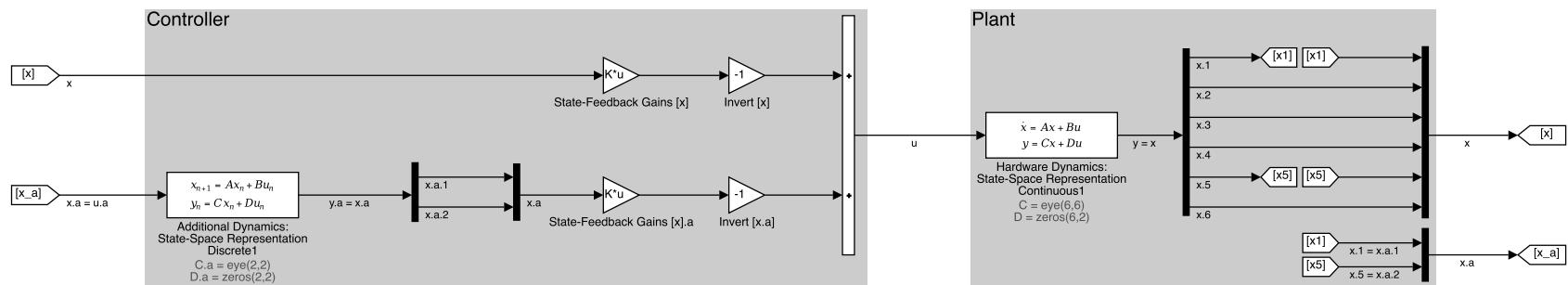


Figure 5.8.: [Additional Dynamics]: 4.0 Additional Dynamics (Linear View: Controller)

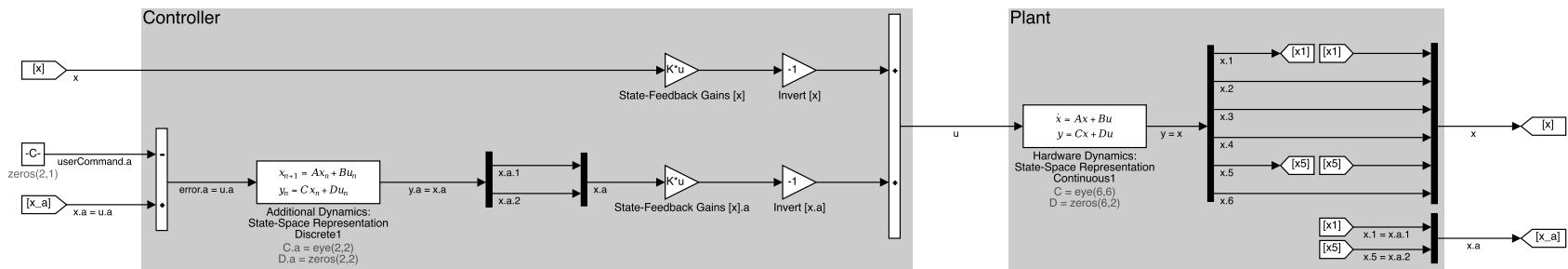


Figure 5.9.: [Additional Dynamics]: 4.0 Additional Dynamics with Reference Signal (Linear View: User)

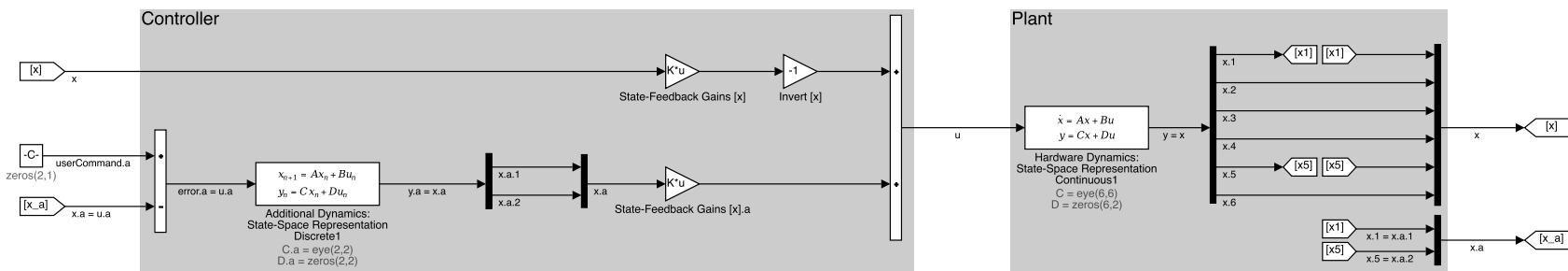


Figure 5.10.: [Additional Dynamics]: 4.0 Additional Dynamics with Reference Signal (Linear View: User)

5.1.1.1. 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 5.9.

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 5.10.

5.1.2. 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 (5.1).

$$\begin{aligned}\dot{\mathbf{x}}_{nx1} &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \mathbf{x}_{nx1} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_{\phi_y} \end{bmatrix} \\ \mathbf{y}_{mx1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \mathbf{x}_{nx1} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_{\phi_y} \end{bmatrix}\end{aligned}\tag{5.1}$$

5.1.2.1. 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. An integrator is an established case which is exhibited in Equation (5.2).

$$\begin{aligned}\dot{\mathbf{x}}_{nx1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \mathbf{x}_{nx1} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_{\phi_y} \end{bmatrix} \\ \mathbf{y}_{mx1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \mathbf{x}_{nx1} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_{\phi_y} \end{bmatrix}\end{aligned}\quad (5.2)$$

5.1.3. 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.

5.1.3.1. Optimal

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

5.1.3.1.1. 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.

5.1.3.1.2. 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.

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 5.11 - 5.17 depict the system state during its operation while completing its response to a figure-eight linear position command.

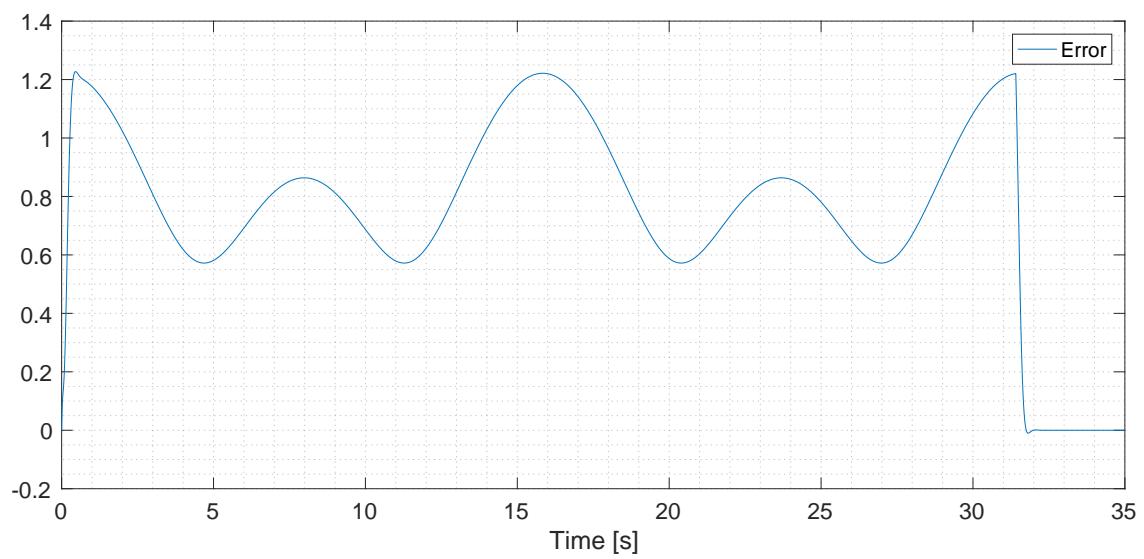
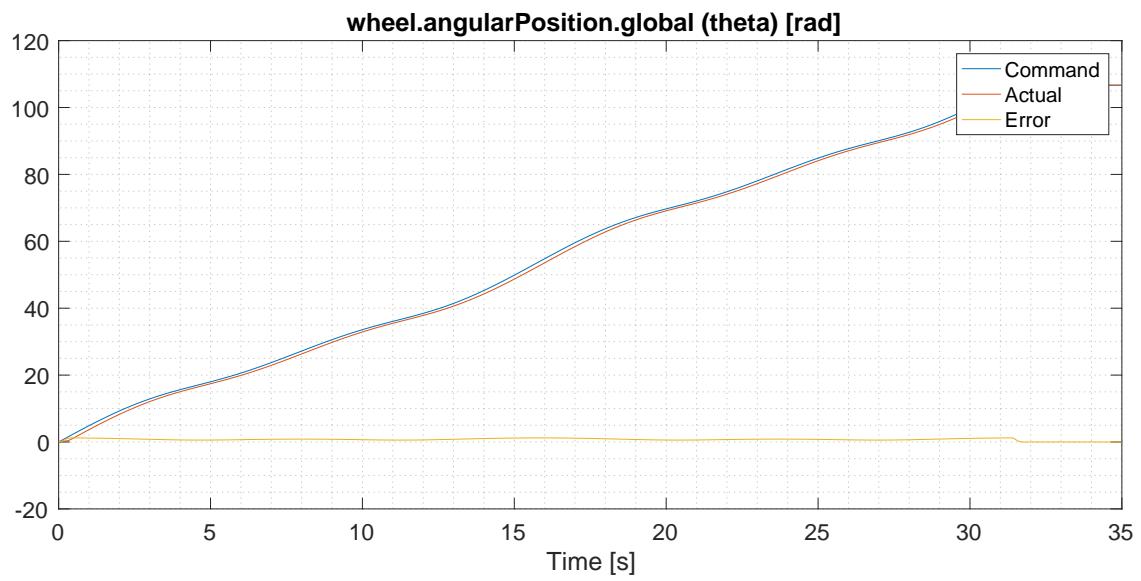


Figure 5.11.: [Control Gains: LQR]: Simulation Results: Wheel Angular Position θ

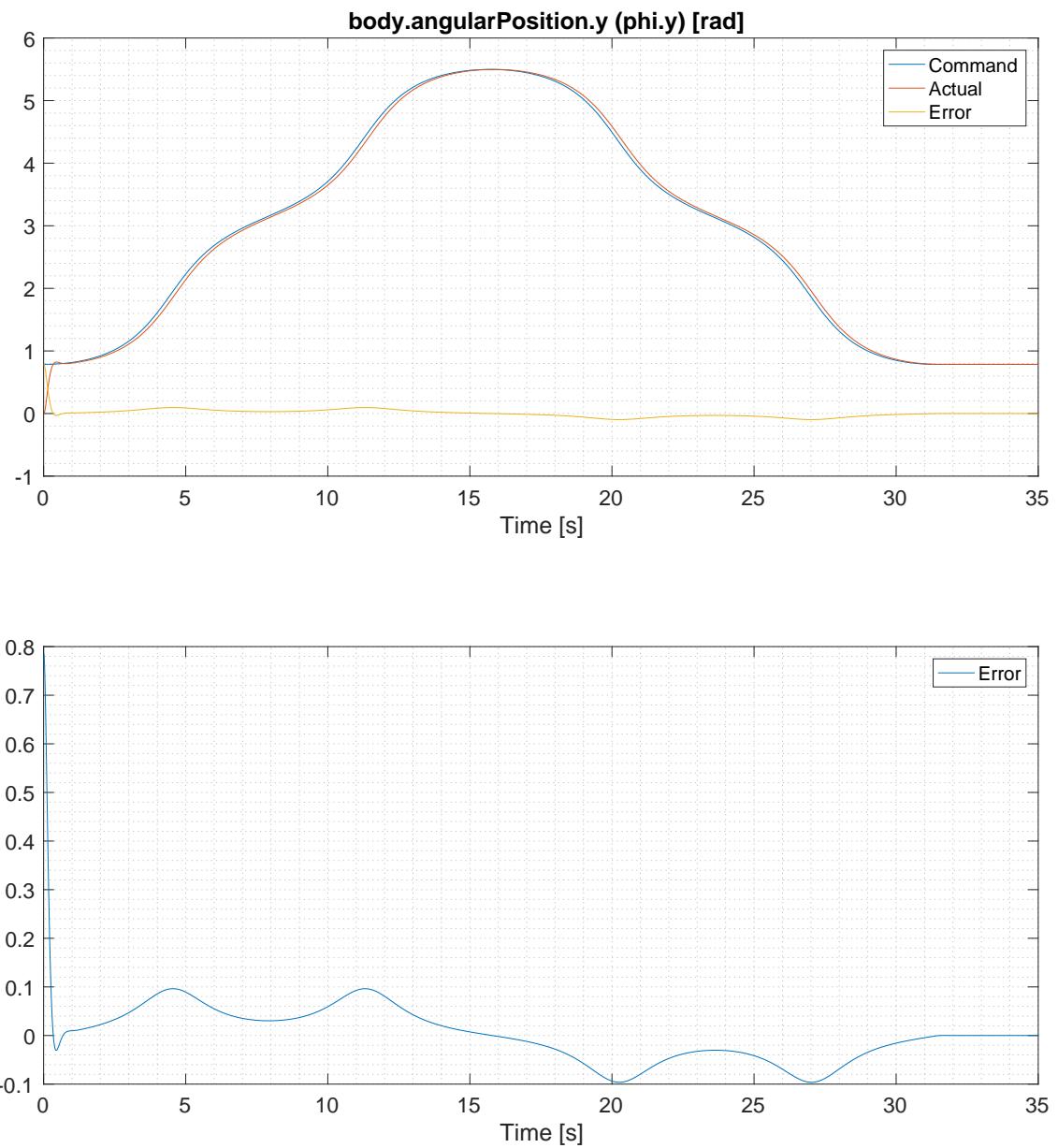


Figure 5.12.: [Control Gains: LQR]: Simulation Results: Body Angular Position ϕ_y

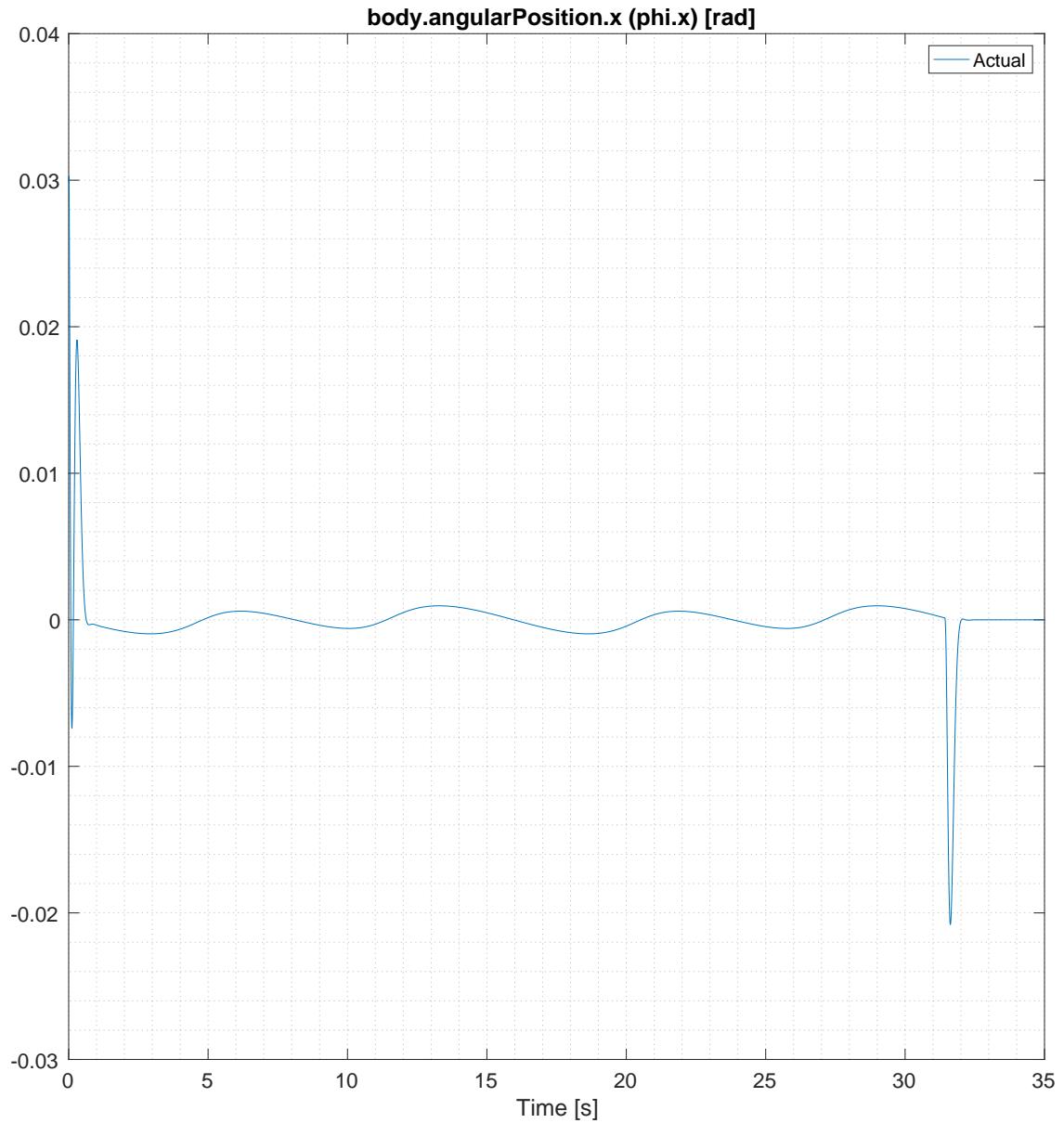


Figure 5.13.: [Control Gains: LQR]: Simulation Results: Body Angular Position ϕ_x

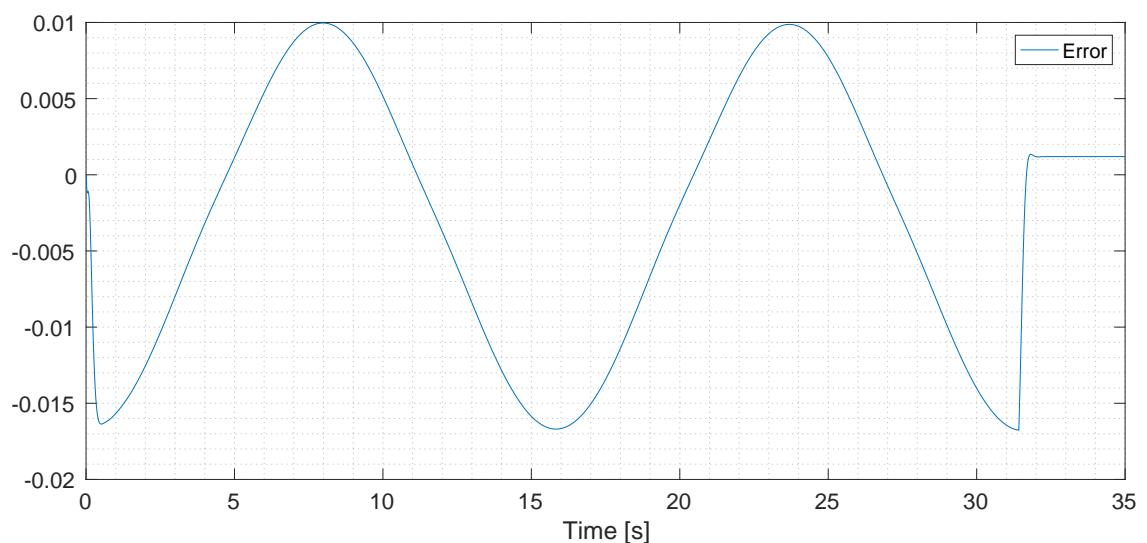
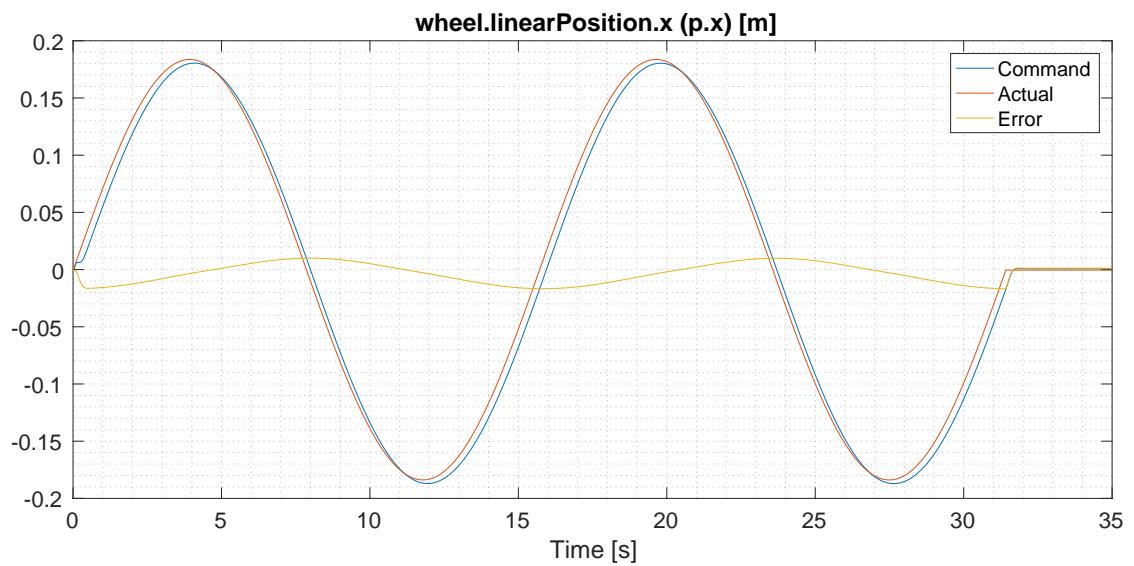


Figure 5.14.: [Control Gains: LQR]: Simulation Results: Wheel Linear Position p_x

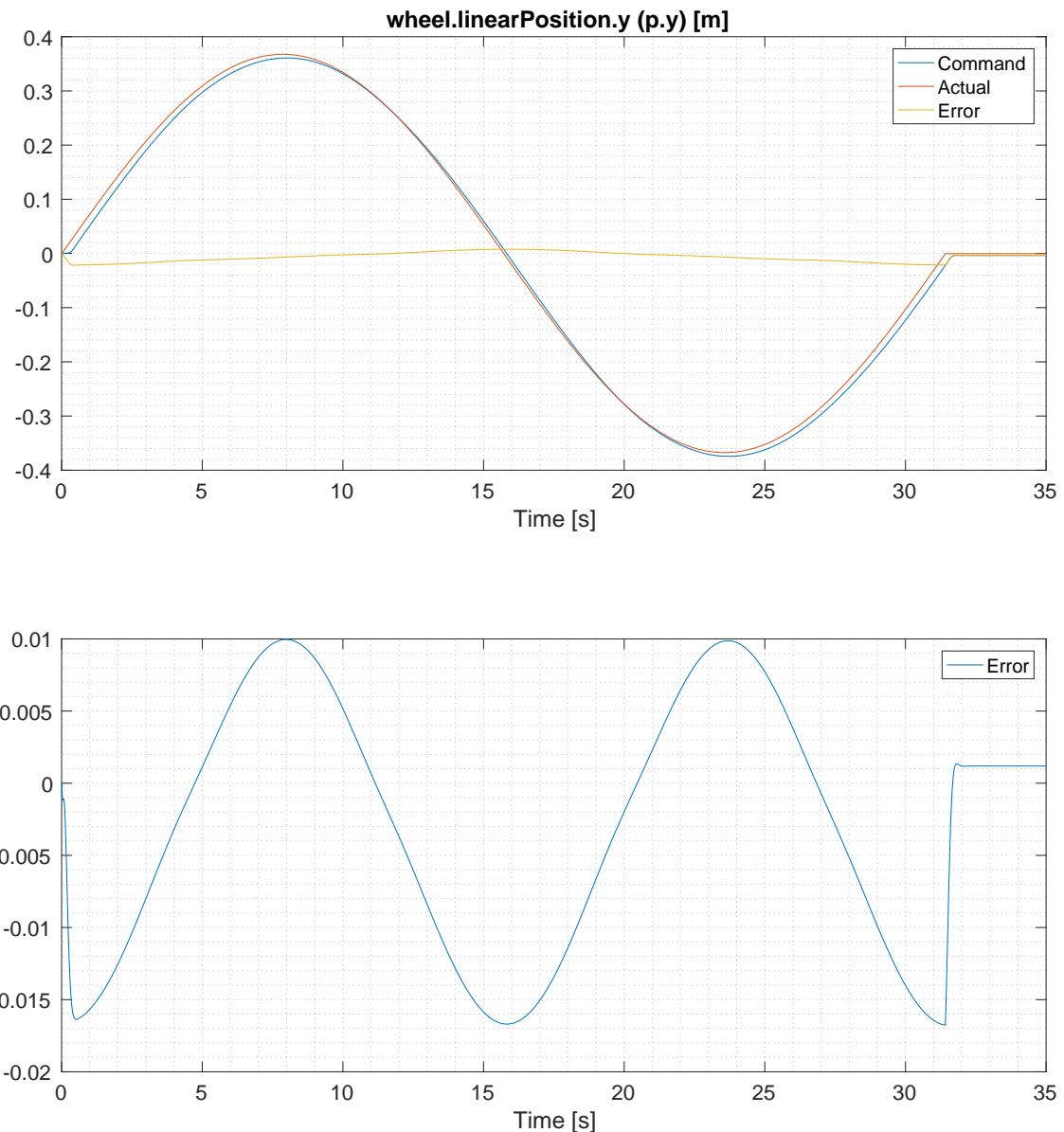


Figure 5.15.: [Control Gains: LQR]: Simulation Results: Wheel Linear Position p_y

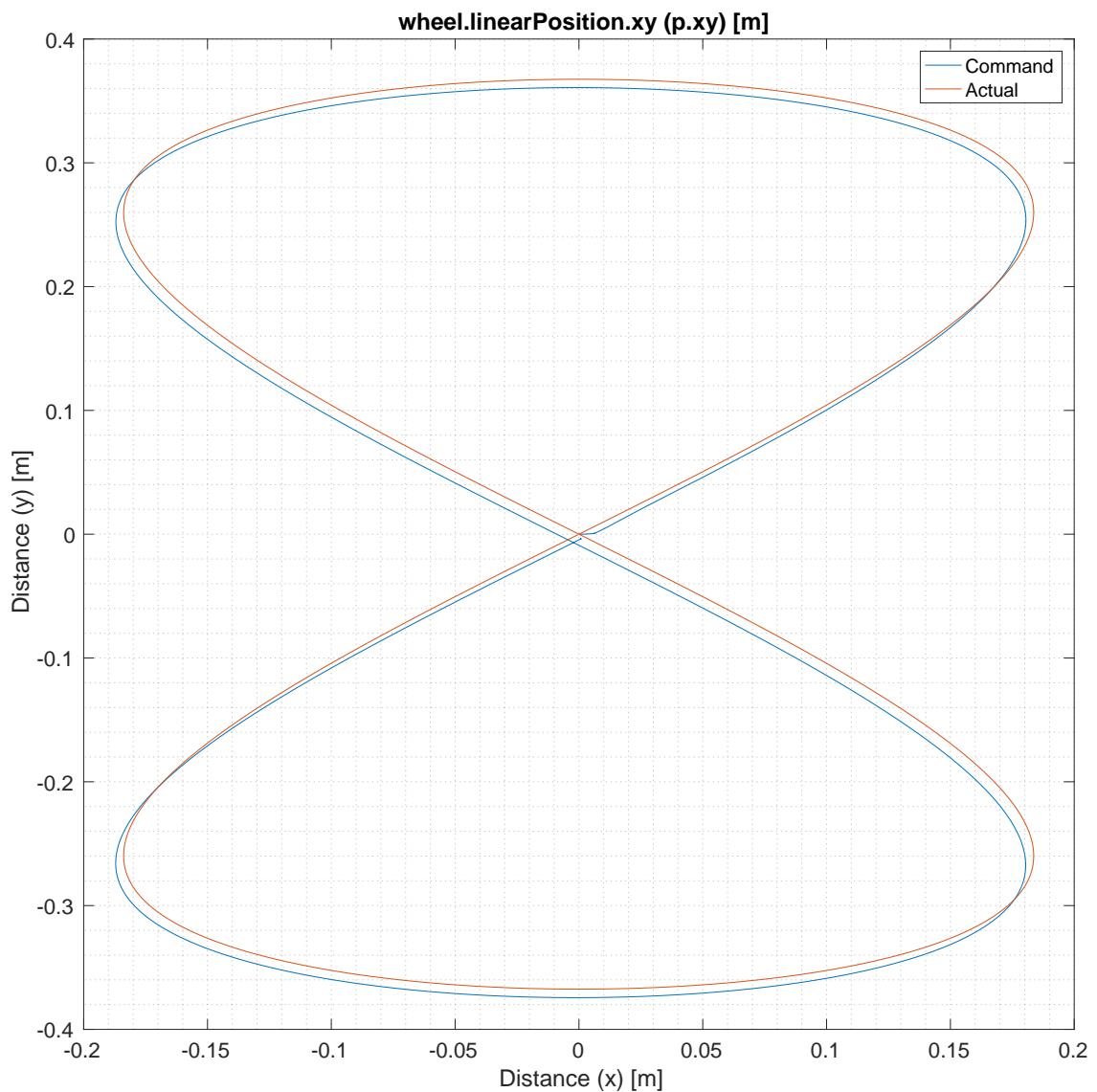


Figure 5.16.: [Control Gains: LQR]: Simulation Results: Wheel Linear Position p_{xy}

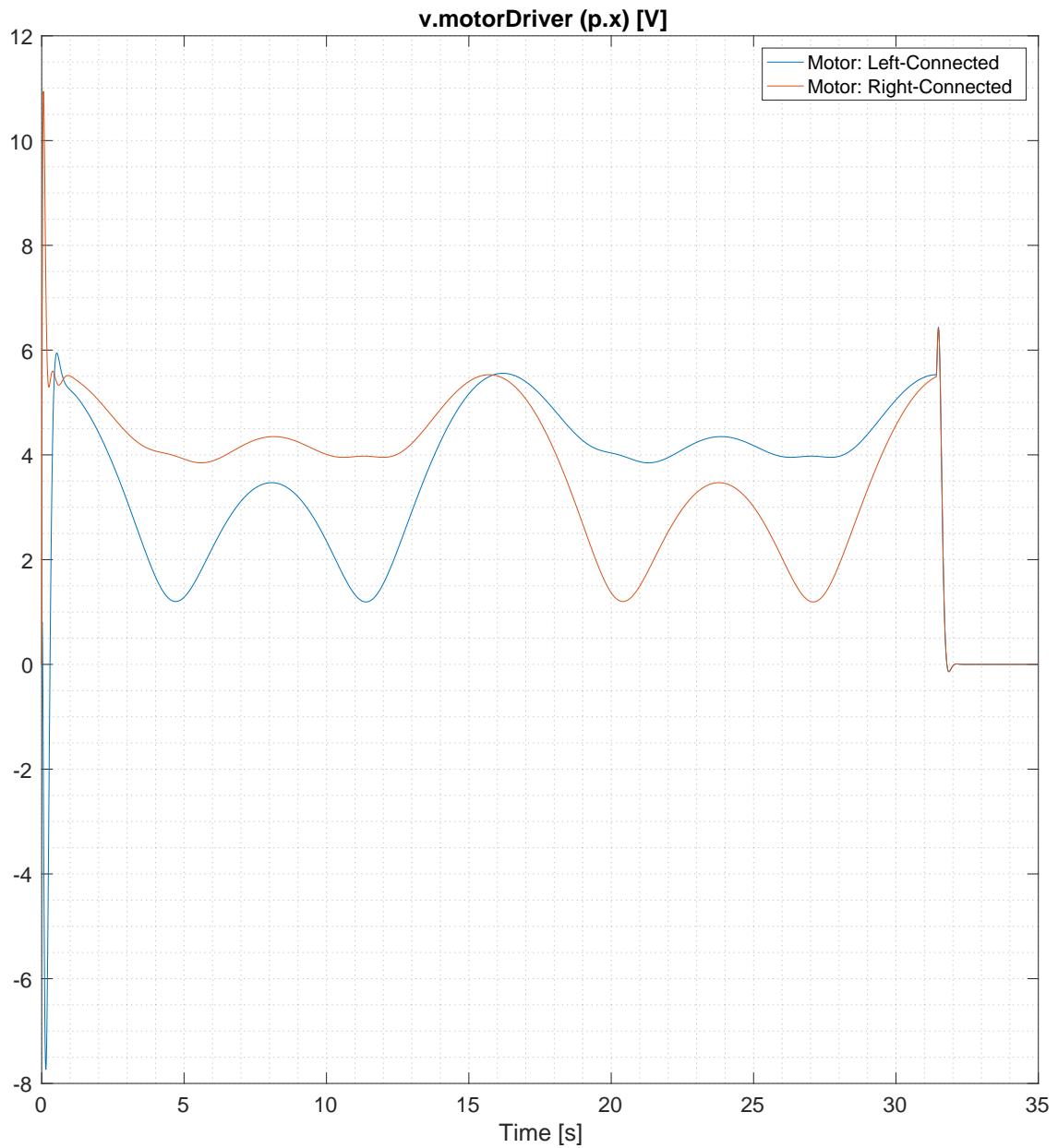


Figure 5.17.: [Control Gains: LQR]: Simulation Results: Motor Driver Commanded Voltage
 $v_{motorDriver}$

5.2. Bibliography

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

CHAPTER 6.

Conclusions

The results provided in Section 5.1.3.1.2, 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 balanced 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 1st to 6th 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 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

```
1 %% [ Global ]
2 minseg_0p0p0p0_global
3
4 %% [ Input ]: Model
5 minseg_1p0p0p0_input
6
7 %% [ Input ]: Script: Commands
8
9 ui.x.build = 0; % rebuild required for any change in [input]: model / [init]: model
10 .
11 ui.x.write = 0; % not yet implemented
12 ui.x.read = 0;
13 ui.x.plot = 0; % enables read/write by default.
14 ui.x.save = 0; % enables read/write by default.
15 ui.x.cleanup = 0; % enables read/write by default.
16
17 %% [ Input ]: Script: Serial
18
19 switch 1 % serial duration
20   case 0; ui.srl.n.transmits = 100; % [samples]
21   case 1; ui.srl.T.transmits = 020; % [seconds]
22 end
23
24 %% [ Input ]: Script: Save
25 ui.save.label = '';
26
27 %% [ Init ]: Define parameters
28 minseg_2p0p0p0_init_general
29
30 minseg_2p1p0p0_init_model_general
31 minseg_2p1p1p0_init_model_plant
32 minseg_2p1p2p0_init_model_controller
33 minseg_2p1p3p0_init_model_io
```

```

33 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 (
57     ui.x.read || ui.x.plot || ui.x.save )
58 minseg_3p2p0p0_process_serial_reads
59
60 % if ui.mdl.case == 2
61 % minseg_3p3p1p0_process_motorTF
62 % end
63 %
64 % if ui.mdl.case == 3
65 % minseg_3p3p1p0_process_gyroBias
66 % end
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

```
1 %% [ Global ]:  
2 clc  
3 clearvars  
4 close all  
5  
6 % close all loaded simulink models and libraries.  
7 % close_system( find_system('SearchDepth', 0) )  
8  
9 % close and delete all serial connections  
10 if ~isempty(instrfindall)  
11     fclose(instrfindall);  
12     delete(instrfindall);  
13 end  
14  
15 %% [ Global ]: Add subdirectories to Matlab path  
16 root.dir = cd;  
17 root.sub.dir = { [root.dir '/1. General Tools/']  
18                 [root.dir '/1. General Tools/0. Bessel Poles']  
19                 [root.dir '/1. General Tools/1. fftPlus']  
20                 [root.dir '/1. Subscripts']  
21                 [root.dir '/2. Model metadata']  
22                 [root.dir '/3. Data']  
23             };  
24  
25 root.n.sub.dir = size(root.sub.dir, 1);  
26 for i0 = 1 : root.n.sub.dir  
27     addpath( root. sub.dir{root.n.sub.dir - (i0 - 1), 1} )  
28 end  
29  
30 %% [ Global ]: Add subdirectories to Simulink path  
31 Simulink.fileGenControl( 'set' ...  
32             , 'CacheFolder' , [root.dir '/2. Model metadata/Work'] ...  
33             , 'CodeGenFolder' , [root.dir '/2. Model metadata/Code'] ...  
34             , 'createDir' , true ...  
35         )  
36  
37 %% End
```

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

A.1.2. User Inputs

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

```

1 %% [ Input ]: Model: General
2 ui.mdl.label = 'minseg_M2V3_2017a';
3
4 ui.mdl.mode = 0;
% 0: normal
% 1: external
5
6 ui.mdl.case = 0;
7
8 %% ##: Case Description: Plant: Controller: Command:
9 % -01: Clear board Empty Empty Empty % not yet implemented
10 % +00: Custom Custom Custom Custom
11 % +01: Motor characterization Hardware FF - v.motor 0 --> 10 [V]
12 % +02: Gyro bias calibration Hardware PID - w.motor 0 --> 00 [rad/s]
13
14
15 %% [ Input ]: Model: Plant
16
17 ui.plant.dynamics.mode = 0;
% 0: actual hardware
% 1: simulated dynamics (non-linear)
% 2: simulated dynamics ( linear )
18
19 ui.plant.n.batteries = 6; % [range: 0 - 6]
20
21 ui.plant.x.bluetoothModule = 1;
% 0: bluetooth module not inserted into board.
% 1: bluetooth module inserted into board.
22
23
24 ui.plant.supply.mode = 1;
% 0: 9.00 [V] (battery pack)
% 1: 4.50 [V] (usb cable)
25
26 % Important: Do NOT set to usb power if actually using battery power.
27
28
29 %% [ Input ]: Model: Controller: body.pitch.theta
30 % not yet implemented.
31
32
33 ui.ctrl.body.pitch.theta.mode = 0;
% #: mode: input command: [input command unit]:
34
35
36
37
38

```

```

39 %% [Input ]: Model: Controller: motor.v
40
41 ui.ctrl.motor_v.mode = 1;
42 % #: mode:           input command: [input command unit]:
43 % 0: feedForward    v.motor          [V]
44 % 1: PID            w.motor          [rad/s]
45
46 switch ui.ctrl.motor_v.mode
47
48 case 0 % feed forward (input: v.motor)
49   ui.io.write.ctrl.motor_v.cmd.tStart (1,1) = 0;
50   ui.io.write.ctrl.motor_v.cmd.val.x (1,1) = +10;
51   ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max(1,1) = +0.01;
52   ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min(1,1) = -0.01;
53
54 case 1 % PID          (input: w.motor)
55   ui.io.write.ctrl.motor_v.cmd.tStart (1,1) = 0;
56   ui.io.write.ctrl.motor_v.cmd.val.x (1,1) = 0.50 * 2*pi;
57   ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max(1,1) = +0.10;
58   ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min(1,1) = -inf;
59
60 end
61
62 %% [Input ]: Model: Serial
63 ui.srl.mode.address = 0;
64 % 0: left      usb port (2015 Macbook Pro)
65 % 1: left-rear usb port (2008 Macbook Pro)
66
67 % Note: Needs to be changed manually for external mode:
68 % Simulink: Configuration parameters: Hardware implementation: Host-board connection
69
70 ui.srl.T.decimation = 0; % [integer] [default: 0]
71 % Integer factor of board sample time (mdl.T.sample)
72 % in which to iterate serial processes.
73 % If 0, minimum possible value will be used.
74 % (Could be greater than 1 if combined size of reads/writes is sufficiently large.)
75
76 %% 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

```
1 %% [ Init    ]: Conversions
2 k.intmax.uint8 = double( intmax('uint8') );
3 k.intmax.int16 = double( intmax('int16') );
4
5 k.deg2rad      = 2*pi / 360;
6 k.rad2deg      = 1 / k.deg2rad;
7
8 k.byte2bit     = 8;
9 k.bit2byte     = 1 / k.byte2bit;
10
11 k.lb2kg        = 0.45359233;
12 k.kg2lb        = 1 / k.lb2kg;
13
14 k.in2m         = 0.0000254;
15 k.m2in         = 1 / k.in2m;
16
17 %% 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

```
1 %% [Init] Initialize user-defined parameters
2 mdl.label = ui.mdl.label;
3 mdl.mode = ui.mdl.mode;
4 mdl.case = ui.mdl.case;
5
6 %% [Init] Load model, if not already loaded
7 if ~bdIsLoaded( mdl.label )
8 load_system( mdl.label );
9 end
10
11 %% [Init] Define general model parameters
12
13 mdl.object = get_param(mdl.label, 'Object');
14
15 switch mdl.mode
16   case 0; mdl.T.sample = 0.005; % 0: normal
17   case 1; mdl.T.sample = 0.030; % 1: external
18 end
19
20 %% 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

```

1  %% [ Init ]: Initialize user-defined parameters
2  plant.supply.mode = ui.plant.supply.mode;
3  plant.dynamics.mode = ui.plant.dynamics.mode;
4
5  plant.n.batteries = ui.plant.n.batteries;
6  plant.x.bluetoothModule = ui.plant.x.bluetoothModule;
7
8  %% [ Init ]: Define general plant parameters
9
10 switch plant.supply.mode
11   case 0; plant.supply.v = 9.00; % [V]
12   case 1; plant.supply.v = 4.50; % [V]
13 end
14
15 a.gravity = 9.81; % acceleration [m / s^2]
16
17 load( 'bessel_poles.mat' )
18
19 %% [ Init ]: Verify legitimate operating modes
20
21 if plant.supply.mode == 0
22 assert( plant.n.batteries == 6,
23 [
24   'Battery power is enabled (plant.supply.mode == 0);\n'
25   'however, the number of batteries in use is not equal to\n'
26   'the number of batteries needed to operate in '
27   'battery power mode (plant.n.batteries ~= 6)'\n
28 ]
29 );
30 end
31
32 %% [ Init ]: Define parameters based on user-specified plant dynamics
33
34 switch plant.dynamics.mode
35   case 0; minseg_2p1p1p1_init_model_plant_hardware
36   case 1; minseg_2p1p1p2_init_model_plant_nonlinearDynamics
37   case 2; minseg_2p1p1p3_init_model_plant_linearDynamics
38 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

```

1 %% [Init] : Motor: Driver
2 mtr.driver.left.pin.pos = 6;
3 mtr.driver.left.pin.neg = 8;
4 mtr.driver.middle.pin.pos = 2;
5 mtr.driver.middle.pin.neg = 5;
6
7 %% [Init] : Motor: Encoder
8 % not yet implemented
9 % mask encoder model, then use pins as mask parameters
10 mtr.encoder.left.pin.A = 19;
11 mtr.encoder.left.pin.B = 18;
12 mtr.encoder.middle.pin.A = 15;
13 mtr.encoder.middle.pin.B = 62;
14
15 mtr.encoder.countPerRev = 720;
16 mtr.encoder.radPerRev = 2 * pi;
17
18 %% [Init] : Motor: Encoder: angVel bessel filter: design parameters
19 mtr.encoder.filter.T.settle = mdl.T.sample * 25; % [s]
20 mtr.encoder.filter.order = 4; % [-] [integer] [ range: 02 : 10 ]
21
22 %% [Init] : Motor: Encoder: angVel bessel filter: transfer function
23 % divide normalize poles by settling time
24 mtr.encoder.filter.s.poles = poly( s.pole.bessel{mtr.encoder.filter.order} ...
25 % ...
26 / mtr.encoder.filter.T.settle ...
27 );
28
29 % create transfer function
30 mtr.encoder.filter.s.tf = tf( mtr.encoder.filter.s.poles(end) ...
31 % ...
32 , mtr.encoder.filter.s.poles ...
33 );
34
35 % discretize transfer function
36 mtr.encoder.filter.z.tf = c2d( mtr.encoder.filter.s.tf ...
37 % ...
38 , 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 ];
69 mtr.encoder.filter.s.ss.C = [ zeros( 1, mtr.encoder.filter.order - 1 ) 1
70 ];
71 mtr.encoder.filter.s.ss.D = 0;
72
73 % discretize s-plane state space equations (canonical representation)
74 [ mtr.encoder.filter.z.ss.A ... phi
75 , mtr.encoder.filter.z.ss.B ... gamma
76 ] = zohe ...
77 ( mtr.encoder.filter.s.ss.A ... A
78 , 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
162 gyro.filter.s.ss.B      = [ zeros(     gyro.filter.order - 1, 1 ); 1 ];
163 gyro.filter.s.ss.C      = [ zeros( 1, gyro.filter.order - 1 ) 1 ];
164 gyro.filter.s.ss.D      = 0;
165
166
167 % discretize s-plane state space equations (canonical representation)
168 [ gyro.filter.z.ss.A ... phi
169 , gyro.filter.z.ss.B ... gamma
170 ] = zohe ...
171 ( gyro.filter.s.ss.A ... A
172 , gyro.filter.s.ss.B ... B
173 , mdl.T.sample ... T
174 );
175
176 gyro.filter.z.ss.C      = gyro.filter.s.ss.C;
177 gyro.filter.z.ss.D      = gyro.filter.s.ss.D;
178
179 %% [Init]: Accelerometer
180 accel.efs_sel.mode = 0; % [ Required: 0 ]
181 % | # | maxValue [g] | Sensitivity [LSB/mg] |
182 % | 0 | +/- 02 | 8192
183 % | 1 | +/- 04 | 4096
184 % | 2 | +/- 08 | 2048
185 % | 3 | +/- 16 | 1024
186
187 assert( accel.efs_sel.mode == 0 );
188
189 switch accel.efs_sel.mode
190 case 0; accel.maxVal = 02 * a.gravity;
191 case 1; accel.maxVal = 04 * a.gravity;
192 case 2; accel.maxVal = 08 * a.gravity;
193 case 3; accel.maxVal = 16 * a.gravity;
194 end
195
196 accel.k_raw2actual = accel.maxVal / k.intmax.int16;
197
198 %% End

```

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

A.1.3.2.2. Nonlinear Dynamics Model

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

```
1 %% 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

```

1 %% [Init] : Plant: Wheel (single)
2
3 % mass measurement precision: 0.01 lb
4 % note: this could be improved with a better scale.
5
6 plant.axel.m = 0.000; % [kg] [note low
7     precision.]
8
9 plant.wheel.r = 0.021; % radius [m] [source:
10    howard]
11 plant.wheel.m = 0.036 / 2; % (includes axel) [kg] [source:
12    howard]
13 plant.wheel.J = 7.460e-6; % moment of inertia [kg / m^2] [source:
14    howard]
15
16 % measured from center of mass of wheel
17
18 %% [Init] : Plant: Body: Masses
19 % note: body does not include wheels.
20
21 % mass measurement precision: 0.01 lb
22 % note: this could be improved with a better scale.
23
24 % plant.board.m = 1.000 * k.lb2kg; % [kg]
25 % plant.motorCable.m = 0.010 * k.lb2kg; % (quantity: 1) [kg] [note low
26     precision.]
27 % plant.motor.m = 0.220 * k.lb2kg; % (quantity: 1) [kg]
28 % plant.battery.m = 1.000 * k.lb2kg; % (quantity: 1) [kg]
29
30 % plant.bluetooth.m = 0.000 * k.lb2kg; % bluetooth module [kg] [note low
31     precision.]
32 % plant.usbCable.m = 0.040 * k.lb2kg; % (not included) [kg]
33
34 % plant.body.m = plant.board.m ...
35 % + plant.motor.m * 2 ...
36 % + plant.motorCable.m * 2 ...
37 % + plant.battery.m * plant.n.battery ...
38 % + plant.bluetooth.m * plant.x.bluetoothModule;
39
40 % mass [kg]

```

```

33
34 plant.body.m      = 1.030; % (not included) [kg]
35                                     % net measurement taken to reduce rounding errors.
36                                     % [taken with 6 batteries].
37
38 %% [Init]: Plant: Body
39 % note: does not include wheels.
40
41 plant.body.l.h     = 8.00 * k.in2m; % height [m]
42 plant.body.l.w     = 3.25 * k.in2m; % width [m]
43 plant.body.l.d     = 2.50 * k.in2m; % depth [m]
44
45 switch plant.x.bluetoothModule
46
47 case 0 % Not inserted
48     switch plant.n.batteries
49         case 0; plant.body.f.natural = 1; % natural frequency [rad/s]
50         case 5; plant.body.f.natural = 1; % natural frequency [rad/s]
51         case 6; plant.body.f.natural = 1; % natural frequency [rad/s]
52     end
53
54 case 1 % Inserted
55     switch plant.n.batteries
56         case 0; plant.body.f.natural = 1; % natural frequency [rad/s]
57         case 5; plant.body.f.natural = 1; % natural frequency [rad/s]
58         case 6; plant.body.f.natural = 3.5087719; % natural frequency [rad/s]
59     end
60
61 end
62
63 plant.body.w.natural = 2 * pi * plant.body.f.natural;
64                                     % natural angular frequency [rad/s]
65
66 plant.body.l.c      = 3 * (a.gravity - plant.body.w.natural^2 * plant.wheel.r) ...
67             / (4 * plant.body.w.natural^2);
68                                     % wheel axel to center of mass of robot [m]
69
70 plant.body.J.x      = plant.body.m * plant.body.l.c^2 ...
71             / 3;
72                                     % moment of inertia (pitch) [kg / m^2]
73                                     % (measured from center of mass of robot)

```

```

74
75 plant.body.J.y = plant.body.m ...  

76 * (plant.body.l.w^2 + plant.body.l.d^2) ...  

77 / 12;  

78 % 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:  

86 howard]  

86 mtr.k.dlambda = 0.495; % back EMF constant [V*s / rad] [source:  

87 howard]  

87 mtr.k.torque = 0.470; % torque constant [N*m / A] [source:  

88 howard]  

89  

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

```
1 %% [ Init ]: Initialize user-defined parameters
2 ctrl.motor_v.mode = ui.ctrl.motor_v.mode;
3
4 %% [ Init ]: Setup controller variant subsystems
5 ctrl.motor_v.ff.motor_v.var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
6 ctrl.motor_v.pid.motor_w.var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );
7
8 %% [ Init ]: Define controller model parameters
9
10 switch ctrl.motor_v.mode
11
12 case 0 %
13
14 case 1
15   ctrl.motor_v.pid.motor_w.k.p = 0.500;
16   ctrl.motor_v.pid.motor_w.k.i = 1.000;
17   ctrl.motor_v.pid.motor_w.k.d = 0.000;
18
19   ctrl.motor_v.pid.motor_w.int.maxVal = +plant.supply.v;
20   ctrl.motor_v.pid.motor_w.int.minVal = -plant.supply.v;
21
22 end
23
24 %% 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

```

1 %% [Init      ]: Setup board i/o variant subsystems
2
3 % general
4
5 io.write.serial.                                var = Simulink.Variant( 'mdl_mode
6     == 0' );
7
8 % plant: hardware
9
10 io.write.serial.hardware.                      var = Simulink.Variant( 'mdl_case
11     plant_dynamics_mode == 0' );
12
13 io.write.serial.hardware.ff.                  var = Simulink.Variant( 'mdl_case
14     ctrl_motor_v_mode == 0' );
15 io.write.serial.hardware.pid.                 var = Simulink.Variant( 'mdl_case
16     ctrl_motor_v_mode == 1' );
17
18 io.write.serial.hardware.ff.standard.        var = Simulink.Variant( 'mdl_case
19     == 0' );
20 io.write.serial.hardware.ff.motorCharacterization.var = Simulink.Variant( 'mdl_case
21     == 1' );
22
23 io.write.serial.hardware.pid.standard.       var = Simulink.Variant( 'mdl_case
24     == 0' );
25 io.write.serial.hardware.pid.sensorCalibration. var = Simulink.Variant( 'mdl_case
26     == 2' );
27
28 % plant: nonlinearDynamics
29
30 io.write.serial.nonlinearDynamics.            var = Simulink.Variant( 'mdl_case
31     plant_dynamics_mode == 1' );
32
33 io.write.serial.nonlinearDynamics.ff.        var = Simulink.Variant( 'mdl_case
34     ctrl_motor_v_mode == 0' );
35 io.write.serial.nonlinearDynamics.pid.       var = Simulink.Variant( 'mdl_case
36     ctrl_motor_v_mode == 1' );

```

```

27
28 io.write.serial.nonlinearDynamics.ff.standard.    var = Simulink.Variant( 'mdl_case
29     == 0' );
30
31
32 % plant: nonlinearDynamics
33
34 io.write.serial.linearDynamics.                      var = Simulink.Variant( '
35     plant_dynamics_mode == 2' );
36
37 io.write.serial.linearDynamics.ff.                  var = Simulink.Variant( '
38     ctrl_motor_v_mode == 0' );
39 io.write.serial.linearDynamics.pid.                var = Simulink.Variant( '
40     ctrl_motor_v_mode == 1' );
41
42
43 %% [ Init ]: Write commands
44 io.write.ctrl.motor_v.cmd.tStart.                 = ui.io.write.ctrl.motor_v.cmd.tStart;
45     % [ s ]
46 io.write.ctrl.motor_v.cmd.val.x.                 = ui.io.write.ctrl.motor_v.cmd.val.x;
47     % [ <cmd> ]
48 io.write.ctrl.motor_v.cmd.val_norm.dx.max.      = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.
49     max; % [ cmd.norm / s ]
50 io.write.ctrl.motor_v.cmd.val_norm.dx.min.      = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.
51     min; % [ cmd.norm / s ]
52
53 %% 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

```
1 %% [Init] : Initialize list of general parameters used within Simulink model
2
3 mdl.parameter.label = {};
4
5 % Specify parameters which will be used in model:
6 mdl.parameter.label = [...]
7 mdl.parameter.label
8 {
9     'k.intmax.uint8'
10
11    'mdl.mode'
12    'mdl.case'
13    'mdl.T.sample'
14
15    'plant.dynamics.mode'
16    'plant.supply.v'
17
18    'ctrl.motor_v.mode'
19    'ctrl.motor_v.ff.motor_v.var'
20    'ctrl.motor_v.pid.motor_w.var'
21
22    'io.write.serial.var'
23    'io.write.scopes.var'
24
25    'io.write.serial.hardware.var'
26    'io.write.serial.hardware.ff.var'
27    'io.write.serial.hardware.ff.standard.var'
28    'io.write.serial.hardware.ff.motorCharacterization.var'
29    'io.write.serial.hardware.pid.var'
30    'io.write.serial.hardware.pid.standard.var'
31    'io.write.serial.hardware.pid.sensorCalibration.var'
32
33    'io.write.serial.nonlinearDynamics.var'
34    'io.write.serial.nonlinearDynamics.ff.var'
35    'io.write.serial.nonlinearDynamics.ff.standard.var'
36    'io.write.serial.nonlinearDynamics.pid.var'
37    'io.write.serial.nonlinearDynamics.pid.standard.var'
38
```

```

39 'io.write.serial.linearDynamics.var'
40 'io.write.serial.linearDynamics.ff.var'
41 'io.write.serial.linearDynamics.ff.standard.var'
42 'io.write.serial.linearDynamics.pid.var'
43 'io.write.serial.linearDynamics.pid.standard.var'
44
45 'io.write.ctrl.motor_v.cmd.tStart'
46 'io.write.ctrl.motor_v.cmd.val.x'
47 'io.write.ctrl.motor_v.cmd.val_norm.dx.max'
48 'io.write.ctrl.motor_v.cmd.val_norm.dx.min'
49
50 }];
51
52 %% [Init]: Append case-dependent parameters: Plant: Dynamics model
53
54 switch plant.dynamics.mode
55
56 case 0 % hardware
57 mdl.parameter.label = [...];
58 mdl.parameter.label =
59 {
60   'gyro.dlpf.mode',
61   'gyro.k_raw2actual',
62
63   'gyro.x.bias',
64   'gyro.x.reset',
65   'gyro.y.bias',
66   'gyro.y.reset',
67   'gyro.z.bias',
68   'gyro.z.reset',
69
70   'gyro.filter.z.ss.A',
71   'gyro.filter.z.ss.B',
72   'gyro.filter.z.ss.C',
73   'gyro.filter.z.ss.D',
74
75   'gyro.filter.z.num',
76   'gyro.filter.z.den',
77
78   'accel.k_raw2actual',
79

```

```

80    'mtr.driver.left.pin.pos'
81    'mtr.driver.left.pin.neg'
82    'mtr.driver.middle.pin.pos'
83    'mtr.driver.middle.pin.neg'
84
85    'mtr.encoder.left.pin.A'
86    'mtr.encoder.left.pin.B'
87    'mtr.encoder.middle.pin.A'
88    'mtr.encoder.middle.pin.B'
89
90    'mtr.encoder.countPerRev'
91    'mtr.encoder.radPerRev'
92
93    'mtr.encoder.filter.z.ss.A'
94    'mtr.encoder.filter.z.ss.B'
95    'mtr.encoder.filter.z.ss.C'
96    'mtr.encoder.filter.z.ss.D'
97
98    'mtr.encoder.filter.z.num'
99    'mtr.encoder.filter.z.den'
100  }];
101
102 case 1
103 mdl.parameter.label = [...];
104 mdl.parameter.label
105 {
106  ];
107
108 case 2
109 mdl.parameter.label = [...];
110 mdl.parameter.label
111 {
112  ];
113
114 end
115
116 %% [Init]: Append case-dependent parameters: Controller: v.motor.input
117
118 switch ctrl.motor_v.mode
119
120 case 0 % feed-forward (input: motor.v)

```

```

121 mdl.parameter.label = [...  

122 mdl.parameter.label  

123 {  

124 }];  

125  

126  

127 case 1 % PID           (input: motor.w)  

128 mdl.parameter.label = [...  

129 mdl.parameter.label  

130 {  

131   'ctrl.motor_v.pid.motor_w.k.p'  

132   'ctrl.motor_v.pid.motor_w.k.i'  

133   'ctrl.motor_v.pid.motor_w.k.d'  

134  

135   'ctrl.motor_v.pid.motor_w.int.maxVal'  

136   'ctrl.motor_v.pid.motor_w.int.minVal'  

137 }];  

138  

139 end  

140  

141 %% [Init] : Relabel parameters for use within Simulink model  

142  

143 % Number of parameters specified  

144 mdl.n.parameter      = size( mdl.parameter.label , 1);  

145  

146 % Indices which contain periods:  

147 mdl.parameter.z.period = regexp(mdl.parameter.label , '\.');//  

148  

149 % For each parameter:  

150 for i0 = 1 : mdl.n.parameter  

151  

152 % Create a new label in which all periods have been set to underscores:  

153 mdl.parameter.label0 = mdl.parameter.label {i0,1};  

154 mdl.parameter.label0 ( mdl.parameter.z.period{i0,1} ) = '_';  

155  

156 % Set the data for the new label equal to the data from the old label:  

157 eval([ mdl.parameter.label0 ' = ' mdl.parameter.label{i0,1} ';' ]);  

158  

159 end  

160  

161 %% [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

```

1 %% [Init] : Import serial read signal label and datatype from model
2
3 % serial read block location:
4 srl.read{1,1}.block.path = ...
5     [ mdl.label '/Board Input // Output/Writes (To PC)/Serial' ];
6
7 while 1 % continue until 'break' command
8
9     srl.read{1,1}.block.path0 = get_param( srl.read{1,1}.block.path ...
10                                         , 'ActiveVariantBlock' ...
11                                         );
12
13 if isempty( srl.read{1,1}.block.path0 )
14     break
15 end
16
17 srl.read{1,1}.block.path = srl.read{1,1}.block.path0;
18
19 end
20
21 % serial read block names:
22 srl.read{1,1}.block.busSelect.label = ...
23 find_system( srl.read{1,1}.block.path ...
24             , 'Regexp' , 'on' ...
25             , 'Name' , 'Bus Selector' ...
26             );
27
28 srl.read{1,1}.block.convert.label = ...
29 find_system( srl.read{1,1}.block.path ...
30             , 'Regexp' , 'on' ...
31             , 'Name' , 'Data Type Conversion*' ...
32             );
33
34 srl.read{1,1}.block.bytepack.label = ...
35 find_system( srl.read{1,1}.block.path ...
36             , 'Regexp' , 'on' ...
37             , 'Name' , 'Byte Pack*' ...
38             );

```

```

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 bytепack block to the same datatype.
86 set_param( srl.read{ 1,1}.block.bytепack.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

```

1 %% [Init] : Define serial communication parameters (general)
2
3 % serial address on PC
4 switch ui.srl.mode.address
5   case 0; srl.address = '/dev/tty.usbmodem1411'; % left      usb port (2015 PC)
6   case 1; srl.address = '/dev/tty.usbmodem621'; % left-rear usb port (2008 PC)
7 end
8 % note: to determine current address, use command: {ls /dev/tty.*} in Terminal.app
9
10 srl.byteOrder      = 'littleEndian';          % [-]
11 srl.f.baud         = 115200;                  % [bit / s]
12 srl.T.baud         = 1 / srl.f.baud;          % [ s / bit]
13
14 srl.type.in        = 'uint8'; % signal datatype when entering transmission
15 srl.type.out       = 'uint8'; % signal datatype when exiting transmission
16
17 % legend:
18 % read involves a single read (1 sample).
19 % reads involves all reads (all samples).
20
21 %% [Init] : Serial buffer size
22
23 srl.bufferSize.in = max( [0; srl.read{1,1}.n.Bits] ); % [bits]
24 srl.bufferSize.out = srl.bufferSize.in;                   % [bits]
25
26 % buffer sizes should be equivalent to write or read size (whichever is higher).
27
28 %% [Process] : Setup serial object
29
30 % Ensure that desired serial port does not already exist in the loaded list:
31 if ~isempty( instrfind('Port', srl.address) )
32   fclose( instrfind('Port', srl.address) );
33   delete( instrfind('Port', srl.address) );
34 end
35
36 % Initialize serial object
37 srl.srl = serial(                               srl.address      ...
38                   , 'ByteOrder'           , srl.byteOrder    ...

```

```

39      , 'BaudRate' , srl.f.baud ... [ Hz ]
40      , 'InputBufferSize' , srl.bufferSize.in ... [bits]
41      , 'OutputBufferSize' , srl.bufferSize.out ... [bits]
42    );
43
44 % For detailed information , use: get(srl.srl)
45
46 %{
47 how prove no "header" value?
48 how read timeout period? how reduce to something reasonable?
49
50 find more information on:
51
52 TimerPeriod = 1
53 Timeout      = 10
54 StopBits     = 1
55
56 %}
57
58 %% [Init ]: Time required to perform transmission
59
60 % time required to transmit each write:
61 srl.write{1,1}.T.transmit = srl.T.baud * ( 0 ); % [ s / write
62
63 % [ s / bit ] * ( [ bit / write ] )
64
65 % time required to transmit each read:
66 srl.read {1,1}.T.transmit = srl.T.baud * ( srl.read{1,1}.n.Bits + 08 ); % [ s / read
67
68 % [ s / bit ] * ( [ bit / read ] )
69
70 % note: 1 byte (08 bits) added to account for terminator (1 byte).
71
72 srl.read {1,1}.T.transmit = srl.read {1,1}.T.transmit * 10 / 08;
73
74 %% [Init ]: Time between start of each transmission
75
76 % number of board sample periods per serial process period
77

```

```

78 if ui.srl.T.decimation == 0
79 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
101 % time
102 try srl.n.transmits = round( ui.srl.T.transmits / srl.T.sample ); % [transmit
103 % cycles]
104 catch; srl.n.transmits = ui.srl.n.transmits; % [transmit
105 % cycles]
106 end
107 srl.n.transmits = srl.n.transmits + 1; % 1 added for time = 0
108
109 %% 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

```
1 %% [Init] Initialize serial reads variable
2
3 srl.reads{ srl.read{1,1}.n.signals , 1 } = [];
4
5
6 %% [Init] Define serial reads parameters
7
8 % number of bytes/bits captured after all reads have been performed:
9 for i0 = 1 : srl.read{1,1}.n.signals
10 srl.reads{i0,1}.n.bytes = srl.read{i0,1}.n.bytes * srl.n.transmits; % [bytes]
11 srl.reads{i0,1}.n.bits = srl.read{i0,1}.n.bits * srl.n.transmits; % [bits]
12 end
13
14 srl.reads{1,1}.n.Bytes = srl.read{1,1}.n.Bytes * srl.n.transmits; % [bytes]
15 srl.reads{1,1}.n.Bits = srl.read{1,1}.n.Bits * srl.n.transmits; % [bits]
16
17 %% [Init] Initialize serial reads value vectors
18
19 for i0 = 1 : srl.read{1,1}.n.signals
20 srl.reads{i0,1}.val = zeros( srl.reads{i0,1}.n.bytes , 1 ); % [varies]
21 end
22
23 srl.reads{1,1}.Val = zeros( srl.reads{1,1}.n.Bytes , 1 ); % [varies]
24
25 %% 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

```

1  %% [Init] : Define serial transmission parameters
2  io.srl.read.rateTransition.initialCondition = uint8( zeros( srl.read{1,1}.n.Bytes , 1
3      ) );
4
5  %% [Init] : Initialize list of general parameters used within Simulink model
6
7  mdl.parameter.label = {};
8
9  % Specify parameters which will be used in model:
10 mdl.parameter.label = [...];
11 mdl.parameter.label
12 {
13     'io.srl.read.rateTransition.initialCondition'
14     'io.srl.read.rateTransition.T.sample'
15
16 % cannot set certain hardware parameters via variables. [must hard-code.]
17
18 % 'srl.address';
19 % 'srl.f.baud';
20 }];
21
22 %% [Init] : Relabel parameters for use within Simulink model
23
24 % Number of parameters specified
25 mdl.n.parameter = size( mdl.parameter.label , 1 );
26
27 % Indices which contain periods:
28 mdl.parameter.z.period = regexp(mdl.parameter.label , '\. ');
29
30 % For each parameter:
31 for i0 = 1 : mdl.n.parameter
32
33 % Create a new label in which all periods have been set to underscores:
34 mdl.parameter.label0 = mdl.parameter.label {i0,1};
35 mdl.parameter.label0 ( mdl.parameter.z.period{i0,1} ) = '_';
36
37 % Set the data for the new label equal to the data from the old label:

```

```
38 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

```
1 %% [ Process ]: Build (Normal mode or External mode)
2 switch mdl.mode
3
4 case 0 % Normal mode
5 disp('Performing build :')
6 mdl.T.build = tic;
7 set_param(mdl.label, 'SimulationMode', 'normal') % put model into normal
8 mode
9 rtwbuild (mdl.label) % build model into hardware
10 disp('Build completed.')
11 disp(' ')
12
13 case 1 % External mode
14 set_param(mdl.label, 'SimulationMode', 'external') % put model into external
15 mode
16 set_param(mdl.label, 'SimulationCommand', 'connect') % connect to the executable
17 set_param(mdl.label, 'SimulationCommand', 'start') % start the executable
18 % set_param(mdl.label, 'SimulationCommand', 'stop') % stop the executable
19
20 end
21
22 %% End
```

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

A.1.4.2. Serial Transmission

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

```

1 %% [ Process ]: Open, read/write, and close serial port object.
2
3 % open serial channel
4 fopen ( srl.srl );
5
6 disp( 'Performing serial read:' )
7
8 % initialize complete read cycle timers
9 srl.t.start = clock;
10 srl.T.all = tic;
11
12 for i0 = 1 : srl.n.transmits
13     srl.T.one = tic;
14
15
16 % write
17 % srl.write{1,1}.T.one = tic;
18
19
20 % read
21 srl.read{1,1}.T.one = tic;
22
23
24 % perform read of one time sample:
25 srl.read{1,1}.Val = fread( srl.srl           ... serial object
26                           , srl.read{1,1}.n.Bytes ... read size      [bytes/read]
27                           , srl.type.in        ... input data class [default:
28                           'uint8']);
29
30 if isempty( srl.read{1,1}.Val ) % occasionally isempty on startup.    [seek better
31     fix.]
32 srl.read{1,1}.Val = NaN * zeros( srl.read{1,1}.n.Bytes, 1 );
33 end
34
35 % append to vector of all reads:
36 srl.reads{1,1}.Val( ( 1:srl.read{1,1}.n.Bytes ) + ( i0-1)*srl.read{1,1}.n.Bytes, 1 ) =
37 ...

```

```

36 srl.read {1,1}.Val;
37
38
39 % wait for end of time sample:
40 if i0 ~= srl.n.transmits % if not the last sample
41     while toc( srl.T.one ) < srl.T.sample % then loop to wait until
42         end % a complete sample period
43     end % has passed before reading
44         % again.
45
46 end
47
48 srl.T.all = toc( srl.T.all );
49 srl.t.stop = clock;
50
51 disp(['Intended total transmit time: ' num2str( srl.n.transmits * srl.T.sample, '
52     %010.6f' ) ]);
53 disp(['Actual total transmit time: ' num2str( srl.T.all , '
54     %010.6f' ) ]);
55 disp('Serial read complete.')
56 disp(' ')
57
58
59
60 % convert output to intended data type:
61 srl.read {1,1}.Val = cast( srl.read {1,1}.Val, srl.type.out );
62 srl.reads{1,1}.Val = cast( srl.reads{1,1}.Val, srl.type.out );
63 % note: Mathworks forces conversion to 'double' for serial read output.
64
65 %% 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

```

1 %% [ Process ]: Format serial port data
2
3 % Index of first byte of each read
4 srl.reads{1,1}.z.byte1 = ( 0:srl.n.transmits-1 ).' * srl.read{1,1}.n.Bytes + 1;
5
6 srl.read{1,1}.i.byte0 = 0; % initialize byte offset
7 for i0 = 1 : srl.read{1,1}.n.signals
8
9 % Start index of signal i0 at each sample
10 srl.reads{i0,1}.z.byte0 = srl.reads{1,1}.z.byte1 + srl.read{1,1}.i.byte0;
11
12 % Include additional indices for multibyte signals
13 if srl.read{i0,1}.n.bytes > 1
14     srl.reads{i0,1}.z.byte0 = bsxfun( @plus
15                             , srl.reads{i0,1}.z.byte0 ...
16                             , 0:srl.read{i0,1}.n.bytes-1 ...
17                             );
18 end
19
20 % Pull corresponding values
21 if strcmp( srl.read{i0,1}.type.original , srl.type.out )
22     % If intended signal datatype is equal to serial output,
23     % then use it immediately:
24     srl.reads{i0,1}.val = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );
25
26 else % If intended signal datatype is not equal to serial output,
27     % then first convert the serial output:
28     srl.reads{i0,1}.val0 = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );
29
30 % Convert to cell:
31 srl.reads{i0,1}.val = mat2cell( srl.reads{i0,1}.val0 ...
32                               , ones( size( srl.reads{i0,1}.val0 , 1 ) , 1 ) ...
33                               , size( srl.reads{i0,1}.val0 , 2 ) ...
34                               );
35 % Typecast each row vector to correct type:
36 srl.reads{i0,1}.fun = @(x) typecast( x , srl.read{i0,1}.type.original );
37 srl.reads{i0,1}.val = cellfun( srl.reads{i0,1}.fun ...
38                               , srl.reads{i0,1}.val ...
39                               );

```

```

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

```
1 %% [Output]: Save all data
2
3 file.label = [datestr(now, 'yyyy.mm.dd HH.MM') ' minseg'];
4
5 if ~isempty(ui.save.label)
6 file.label = [file.label ' ' ui.save.label];
7 end
8
9 disp('Performing export to .mat file.')
10
11 save([root.data.dir file.label '.mat'])
12
13 disp('Export to .mat file complete.')
14 disp(' ')
15
16 %% 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

```

1 %% [Output ]: Common plot commands
2
3 %subplot with 2d indices:
4 dim1      = @(n_col, row, col)          (row-1)*n_col + col; % Matrix index: 2d to 1d
5 subplott = @(n_row, n_col, M) subplot(n_row, n_col, dim1(n_col, M(1), M(2)) );
6
7 % axis value
8 msd       = @(x) fix(log10(abs(x))) ; % most significant
9         digit. [ones digit = 0th digit]
10 rndout   = @(x, N) sign(x) .* ceil(abs(x)*10^(-N)) * 10^(+N); % round away from
11         zero at specified digit.
12 rndOut   = @(x, N) rndout(x, msd(x) - N); % round away from zero at N digits right
13         from most significant digit.
14
15 %% [Output ]: Plot setup
16 p = 0;
17
18 disp('Performing plot creation: ')
19
20 for i0 = 1 : srl.read{1,1}.n.signals
21
22 p = p + 1;
23 figure(p)
24
25
26 % plot data
27 if i0==1; stairs( srl.reads{i0,1}.val, '.-'); % clock
28 else;    stairs(srl.reads{1,1}.val, srl.reads{i0,1}.val, '.-'); % all else
29 end
30
31
32 % labels
33 if i0==1; xlabel('Samples [-]'); % clock
34 else;    xlabel('Time [s]'); % all else
35 end
36
37
38 ylabel(srl.read{i0,1}.label)
39
40
41 % y-axis limits
42 if isa(srl.reads{i0,1}.val, 'float') % float
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135

```

```

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

```
1 %% [Cleanup]: Remove alternate subdirectories from Matlab path
2
3 Simulink.fileGenControl('reset')
4
5 %% [Cleanup]: Remove alternate subdirectories from Simulink path
6
7 for i0 = 1 : root.n.sub.dir
8     rmpath(root. sub.dir{root.n.sub.dir - (i0 - 1), 1})
9 end
10
11 %% 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-rensselear-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.arduino-starterkits.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," Master'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 Conference 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 multi-variable 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,” Master’s thesis, University of Massachusetts (UMass): Department of Mechanical and Industrial Engineering, May 2014.