Testbed for Trajectory Control of a Two-Wheeled Robot

Nathan Kando
University of Rhode Island, nate.kando@gmail.com

Follow this and additional works at: https://digitalcommons.uri.edu/theses

Recommended Citation
https://digitalcommons.uri.edu/theses/1136

This Thesis is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Master's Theses by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons@etal.uri.edu.
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
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 $\theta$ and Body Pitch $\phi_x$ ......................... 42
      4.2.2. Body Yaw $\phi_y$ ............................................................................. 43
   4.3. State-Space Representation ......................................................................... 44
   4.4. Calculation of Nonintuitive Parameters ..................................................... 47
      4.4.1. Moment of Inertia: Body: X-axis (Pitch) $J_{\phi_x}$ .......................... 47
      4.4.2. Moment of Inertia: Body: Y-axis (Yaw) $J_{\phi_y}$ ............................ 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
## 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 $\theta$ .......... 70
5.12  [Control Gains: LQR]: Simulation Results: Body Angular Position $\phi_y$ . . . . . . . . . 71
5.13  [Control Gains: LQR]: Simulation Results: Body Angular Position $\phi_x$ . . . . . . . . . 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
<table>
<thead>
<tr>
<th>Code Listing</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>[minseg.m]: Root file</td>
<td>85</td>
</tr>
<tr>
<td>A.2</td>
<td>[minseg.m]: Global Setup</td>
<td>88</td>
</tr>
<tr>
<td>A.3</td>
<td>[minseg.m]: User Inputs</td>
<td>90</td>
</tr>
<tr>
<td>A.4</td>
<td>[minseg.m]: Initialization - General</td>
<td>93</td>
</tr>
<tr>
<td>A.5</td>
<td>[minseg.m]: Initialization - Model - General</td>
<td>95</td>
</tr>
<tr>
<td>A.6</td>
<td>[minseg.m]: Initialization - Model - Plant</td>
<td>96</td>
</tr>
<tr>
<td>A.7</td>
<td>[minseg.m]: Initialization - Model - Plant - Hardware</td>
<td>98</td>
</tr>
<tr>
<td>A.8</td>
<td>[minseg.m]: Initialization - Model - Plant - Nonlinear Dynamics Model</td>
<td>103</td>
</tr>
<tr>
<td>A.9</td>
<td>[minseg.m]: Initialization - Model - Plant - Linear Dynamics Model</td>
<td>104</td>
</tr>
<tr>
<td>A.10</td>
<td>[minseg.m]: Initialization - Model - Controller</td>
<td>111</td>
</tr>
<tr>
<td>A.11</td>
<td>[minseg.m]: Initialization - Model - User-Defined Board Inputs and Outputs</td>
<td>112</td>
</tr>
<tr>
<td>A.12</td>
<td>[minseg.m]: Initialization - Model - Model Build Parameters</td>
<td>114</td>
</tr>
<tr>
<td>A.13</td>
<td>[minseg.m]: Initialization - Serial - Write</td>
<td>120</td>
</tr>
<tr>
<td>A.14</td>
<td>[minseg.m]: Initialization - Serial - Read</td>
<td>121</td>
</tr>
<tr>
<td>A.15</td>
<td>[minseg.m]: Initialization - Serial - General</td>
<td>126</td>
</tr>
<tr>
<td>A.16</td>
<td>[minseg.m]: Initialization - Serial - Reads</td>
<td>129</td>
</tr>
<tr>
<td>A.17</td>
<td>[minseg.m]: Initialization - Serial - Model Build Parameters</td>
<td>130</td>
</tr>
<tr>
<td>A.18</td>
<td>[minseg.m]: Processing - Build</td>
<td>133</td>
</tr>
<tr>
<td>A.19</td>
<td>[minseg.m]: Processing - Serial - Transmit</td>
<td>134</td>
</tr>
<tr>
<td>A.20</td>
<td>[minseg.m]: Processing - Serial - Reads</td>
<td>136</td>
</tr>
<tr>
<td>A.21</td>
<td>[minseg.m]: Output - Save</td>
<td>139</td>
</tr>
<tr>
<td>A.22</td>
<td>[minseg.m]: Output - Serial - Reads - Plot</td>
<td>140</td>
</tr>
<tr>
<td>A.23</td>
<td>[minseg.m]: Global Cleanup</td>
<td>142</td>
</tr>
</tbody>
</table>
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.  \[ \text{Inverted Pendulum: Two-wheeled Robot} \]

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.  
   \[ \text{MinSeg Two-Wheeled Robot and Mathworks Software Suite} \]

3. Select or derive the dynamic equations for the physical model, and populate it.  \[ \text{Yamamoto [1]} \]

4. Select the first controller design to address the selected control problem on the test platform, and implement it in simulation.  \[ \text{Optimal Controller} \]

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 top-heavy 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 $\phi$ sensors.
  • Determine control gains using LQR.
1.4. Bibliography

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).

![Inverted Pendulum on Cart](image)
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.
  
  \[ \text{MinSeg two-wheeled robot.} \]

- A specific device to serve as the controller with respect to the designated control problem.
  
  \[ \text{Arduino Mega2560 single-board microcontroller, included with MinSeg two-wheeled robot.} \]

- A specific computer to be used to interface with the controller.
  
  \[ \text{Available laptop installed with Mathworks Software Suite and supporting software.} \]
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.

  \[
  \begin{array}{c|c}
  \text{n.axes} & \text{Movement} \\ 
  1 & \text{One-dimensional (single, straight line only)} \\ 
  2 & \text{Two-dimensional} \\
  \end{array}
  \]

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

<table>
<thead>
<tr>
<th>Component (Unnecessary)</th>
<th>Part Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>1x Magnetometer</td>
</tr>
<tr>
<td></td>
<td>[1x HMC5883L]</td>
</tr>
<tr>
<td>-</td>
<td>1x Potentiometer</td>
</tr>
<tr>
<td></td>
<td>[1x 3352]</td>
</tr>
</tbody>
</table>
2.2.1.2. **Arduino Single-Board Microcontroller**

The MinSeg M2V3 is primarily built upon an Arduino Mega 2560 single-board microcontroller. Arduino is a 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>
<thead>
<tr>
<th>Characteristic</th>
<th>Uno</th>
<th>Mega 2560</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>ATmega328 MCU</td>
<td>ATmega2560</td>
<td>-</td>
</tr>
<tr>
<td>Programming Interface</td>
<td>USB</td>
<td>USB</td>
<td>-</td>
</tr>
<tr>
<td>UART [Universal Asynchronous Receiver/Transmitter]</td>
<td>01</td>
<td>04</td>
<td>-</td>
</tr>
<tr>
<td>Clock</td>
<td>ceramic resonator</td>
<td>crystal oscillator</td>
<td>-</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16</td>
<td>16</td>
<td>MHz</td>
</tr>
<tr>
<td>Operating Voltage</td>
<td>05</td>
<td>05</td>
<td>V</td>
</tr>
<tr>
<td>Number of Digital I/O [Inputs/Outputs]</td>
<td>14</td>
<td>54</td>
<td>-</td>
</tr>
<tr>
<td>PWM</td>
<td>06</td>
<td>14</td>
<td>-</td>
</tr>
<tr>
<td>Analog Inputs</td>
<td>06</td>
<td>16</td>
<td>-</td>
</tr>
</tbody>
</table>

### Memory/Storage:

- Permanent (*Flash*): 32 kB, 256 kB
- Permanent (*EEPROM*): 01 kB, 04 kB
- Working (*SRAM*): 02 kB, 08 kB

Due to the inclusion of a USB port (*which is coupled to one of the UARTs*), board-to-PC interfacing (*in either direction*) is relatively convenient, as no special equipment is necessary (*beyond a PC containing integrated development environment (IDE) software*). This applies to programming the board (*via the USB programming interface*), as well as communicating signals during operation.
Photos of the Arduino microcontroller are depicted in Figures 2.5 - 2.6. Additionally, a pin layout is provided in Figure 2.7.

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

Figure 2.6.: [Selection of Compatible HW & SW]: Arduino Mega 2560 (Top View) [12]
Figure 2.7.: [Selection of Compatible HW & SW]: Arduino Mega 2560 (Pin Map) [13]
2.2.1.3. Power Source

The MinSeg M2V3 offers two independent sources of power:

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

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

**External-Sourced Power (USB-Cable Connection)**

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

**Internal-Sourced Power (Battery Pack)**

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

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

![Figure 2.8.: [Selection of Compatible HW & SW]: AA-Battery Voltage During Constant Discharge [14]](image-url)
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, \( \text{(so long as the batteries are not completely discharged)} \), 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.

![Motor Driver Pin Map](image)

**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]**

<table>
<thead>
<tr>
<th>PIN</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>NO.</td>
<td></td>
</tr>
<tr>
<td>1.2EN</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>&lt;1:4&gt;A</td>
<td>2, 7, 10, 15</td>
<td>I</td>
</tr>
<tr>
<td>&lt;1:4&gt;Y</td>
<td>3, 6, 11, 14</td>
<td>O</td>
</tr>
<tr>
<td>GROUND</td>
<td>4, 5, 12, 13</td>
<td>—</td>
</tr>
<tr>
<td>( V_{CC2} )</td>
<td>8</td>
<td>—</td>
</tr>
<tr>
<td>3.4EN</td>
<td>9</td>
<td>I</td>
</tr>
<tr>
<td>( V_{CC1} )</td>
<td>16</td>
<td>—</td>
</tr>
</tbody>
</table>
### Table 2.4: [Selection of Compatible HW & SW]: Motor Driver Pin Function Legend [15]

<table>
<thead>
<tr>
<th>INPUTS(2)</th>
<th>OUTPUTS</th>
<th>H</th>
<th>L</th>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>EN</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>L</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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]

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>MIN</th>
<th>MAX</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{CC1}$ Logic supply voltage</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>$V_{CC2}$ Output supply voltage</td>
<td>4.5</td>
<td>36</td>
<td>V</td>
</tr>
<tr>
<td>$V_{HL}$ High-level input voltage</td>
<td>2</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL}$ Low-level input voltage</td>
<td>$-0.3^{(1)}$</td>
<td>0.8</td>
<td>V</td>
</tr>
<tr>
<td>$T_J$ Operating virtual junction temperature</td>
<td>$-40$</td>
<td>125</td>
<td>°C</td>
</tr>
<tr>
<td>$T_A$ Operating free-air temperature</td>
<td>$-40$</td>
<td>85</td>
<td>°C</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{d1}$ Delay time, high-to-low-level output from A input</td>
<td></td>
<td>400</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{d2}$ Delay time, low-to-high-level output from A input</td>
<td></td>
<td>800</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{T_{HL}}$ Transition time, low-to-high-level output</td>
<td></td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{T_{HL}}$ Transition time, high-to-low-level output</td>
<td></td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{eH1}$ Enable time to the high level</td>
<td></td>
<td></td>
<td></td>
<td>700</td>
<td>ns</td>
</tr>
<tr>
<td>$t_{eL1}$ Enable time to the low level</td>
<td></td>
<td></td>
<td></td>
<td>400</td>
<td>ns</td>
</tr>
<tr>
<td>$t_{dH1}$ Disable time from the high level</td>
<td></td>
<td></td>
<td></td>
<td>900</td>
<td>ns</td>
</tr>
<tr>
<td>$t_{dL1}$ Disable time from the low level</td>
<td></td>
<td></td>
<td></td>
<td>600</td>
<td>ns</td>
</tr>
</tbody>
</table>
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.

![Lego NXT Motor (3D Model)](image)

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.

![Lego NXT Motor (Exploded)](image)
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]

<table>
<thead>
<tr>
<th>Component</th>
<th>Figure 2.10</th>
<th>Figure 2.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enclosure</td>
<td>Translucent</td>
<td>Top</td>
</tr>
<tr>
<td>Encoder</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gearing</td>
<td>Dark blue</td>
<td>Bottom</td>
</tr>
<tr>
<td>PCB</td>
<td>-</td>
<td>Left</td>
</tr>
<tr>
<td>Motor</td>
<td>Light orange</td>
<td>Left</td>
</tr>
<tr>
<td>Gearbox</td>
<td>Light blue</td>
<td>Top-Left</td>
</tr>
<tr>
<td>Wheel axle mount</td>
<td>Dark orange</td>
<td>Right</td>
</tr>
</tbody>
</table>

Gearing

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

$$k_\omega A_{2B} = \frac{n_\omega B}{n_\omega A} = \left[k_{\text{teeth} A_{2B}}\right]^{-1} = \left[\frac{n_{\text{teeth} B}}{n_{\text{teeth} A}}\right]^{-1} \quad (2.1)$$

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  |
```
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].

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

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

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

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

\[
k_{\omega \text{ wheel2encoder}} = k_{\omega \text{ wheel2motor}} \cdot k_{\omega \text{ motor2encoder}} = \frac{15}{01} \quad (2.6)
\]
2.2.2. Development PC

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

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>2015 Macbook Pro [17]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System (OS)</td>
<td>macOS 10.12.5</td>
</tr>
<tr>
<td>Mathworks Software Suite</td>
<td>r2017a</td>
</tr>
<tr>
<td>· MATLAB</td>
<td>-</td>
</tr>
<tr>
<td>· Simulink</td>
<td>-</td>
</tr>
<tr>
<td>· Control System Toolbox</td>
<td>-</td>
</tr>
<tr>
<td>· DSP System Toolbox</td>
<td>-</td>
</tr>
<tr>
<td>· Instrument Control Toolbox</td>
<td>-</td>
</tr>
<tr>
<td>· MATLAB Coder</td>
<td>-</td>
</tr>
<tr>
<td>· Simulink Coder</td>
<td>-</td>
</tr>
<tr>
<td>· Simulink Desktop Real-Time</td>
<td>-</td>
</tr>
<tr>
<td>· Matlab Support Package for Arduino Hardware</td>
<td>17.1.0</td>
</tr>
<tr>
<td>· Simulink Support Package for Arduino Hardware</td>
<td>17.1.0</td>
</tr>
<tr>
<td>Xcode [A Mathworks (macOS)-Supported Compiler [18].]</td>
<td>7.3.1</td>
</tr>
<tr>
<td>Rensselaer Arduino Support Package Library (RASPLib)</td>
<td>1.1</td>
</tr>
</tbody>
</table>
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 $\alpha$
  - Body yaw angle $\Psi$
  - Wheel angle $\theta$

- 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


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

\textit{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. \textit{[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 contains the three primary components of the system:

- Plant
- Controller
- Board Inputs and Outputs
3.1.1. 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 hierarchal, and is therefore only the root *or master* file to a series of subfiles. The subfiles are broken up into principal segments of the scripting process:

- Global setup
- User-input
- Initialization
- Processing
- Output
- Global Cleanup
CHAPTER 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 [The body.]
  - A cylinder [The left wheel.]
  - A cylinder [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.
This chapter describes mathematical model and motion equations of NXTway-GS.

### 3.1 Two-Wheeled Inverted Pendulum Model

NXTway-GS can be considered as a two-wheeled inverted pendulum shown in Figure 3.1.

Figure 3.2 shows side view and plane view of the two-wheeled inverted pendulum. The coordinate system used is described in Figure 3.2.

- \( \psi \): body pitch angle
- \( \theta_{l,r} \): wheel angle (\( l, r \) indicates left and right)
- \( \theta_{m,r} \): DC motor angle

\[
2HL = \psi JM, \quad wJm, \quad \phi
\]

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

Figure 4.2.: [Hardware-Equivalent Physical Dynamics Model]: Multiview
### Table 4.1.: [Simulink]: Root

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta$</td>
<td>Angular position: Wheel</td>
<td>[Measured from the wheel center of mass]</td>
</tr>
<tr>
<td>$\theta_g, \theta_b$</td>
<td>Origin aligns with:</td>
<td>$\text{rad}$</td>
</tr>
<tr>
<td>$\theta_{av}, \theta_l, \theta_r$</td>
<td>Component:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$</td>
<td>Angular position: Body</td>
<td>[Measured from the wheels center of mass.]</td>
</tr>
<tr>
<td>$\phi_x, \phi_y, \phi_z$</td>
<td>Dimension:</td>
<td>$\text{rad}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Value</th>
<th>Unit</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_g$</td>
<td>Acceleration of gravity: Earth</td>
<td>9.81</td>
<td>$\frac{m}{s^2}$</td>
<td>-</td>
</tr>
<tr>
<td>$m_w$</td>
<td>Mass: Wheel</td>
<td>0.018</td>
<td>kg</td>
<td>[5]</td>
</tr>
<tr>
<td>$m_b$</td>
<td>Mass: Body</td>
<td>0.381</td>
<td>kg</td>
<td>[5]</td>
</tr>
<tr>
<td>$l_b.h$</td>
<td>Length: Body: Height</td>
<td>??</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>$l_b.w$</td>
<td>Length: Body: Width</td>
<td>??</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>$l_b.d$</td>
<td>Length: Body: Depth</td>
<td>??</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>$l_b.c2a$</td>
<td>Length: Body: [Center of mass] to [Axis of Rotation]</td>
<td>??</td>
<td>m</td>
<td>Sec. 4.4.3</td>
</tr>
<tr>
<td>$r_w$</td>
<td>Length: Wheel: Radius</td>
<td>0.021</td>
<td>m</td>
<td>[5]</td>
</tr>
<tr>
<td>$J_{b.\phi_x}$</td>
<td>Moment of Inertia: Wheel</td>
<td>$7.46 \cdot 10^{-6}$</td>
<td>kg $\cdot m^2$</td>
<td>[5]</td>
</tr>
<tr>
<td>$J_{b.\phi_y}$</td>
<td>Moment of Inertia: Body: Y-axis (pitch)</td>
<td>??</td>
<td>kg $\cdot m^2$</td>
<td>Sec. 4.4.1</td>
</tr>
<tr>
<td>$J_{b.\phi_y}$</td>
<td>Moment of Inertia: Body: Y-axis (yaw)</td>
<td>??</td>
<td>kg $\cdot m^2$</td>
<td>Sec. 4.4.2</td>
</tr>
<tr>
<td>$R_{mtr}$</td>
<td>Motor: Resistance</td>
<td>4.4</td>
<td>$\Omega$</td>
<td>[5]</td>
</tr>
<tr>
<td>$k_{mtr.bEMF}$</td>
<td>Motor: Coefficient of Back EMF</td>
<td>0.495</td>
<td>$\frac{V}{V_{rad}}$</td>
<td>[5]</td>
</tr>
<tr>
<td>$k_{mtr.T}$</td>
<td>Motor: Coefficient of Torque</td>
<td>0.470</td>
<td>$\frac{N \cdot m}{V_{rad}}$</td>
<td>[5]</td>
</tr>
<tr>
<td>$k_{fr.m2w}$</td>
<td>Motor: Coefficient of friction: [DC Motor] to [Wheel]</td>
<td>??</td>
<td>-</td>
<td>Sec. ??</td>
</tr>
</tbody>
</table>

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

### Table 4.2.: [Simulink]: Root

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Value</th>
<th>Unit</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_w$</td>
<td>Mass: Wheel</td>
<td>Includes wheel axle.]</td>
<td>0.018</td>
<td>kg</td>
</tr>
<tr>
<td>$m_b$</td>
<td>Mass: Body</td>
<td>0.381</td>
<td>kg</td>
<td>[5]</td>
</tr>
<tr>
<td>$l_b.h$</td>
<td>Length: Body: Height</td>
<td>??</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>$l_b.w$</td>
<td>Length: Body: Width</td>
<td>??</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>$l_b.d$</td>
<td>Length: Body: Depth</td>
<td>??</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>$l_b.c2a$</td>
<td>Length: Body: [Center of mass] to [Axis of Rotation]</td>
<td>??</td>
<td>m</td>
<td>Sec. 4.4.3</td>
</tr>
<tr>
<td>$r_w$</td>
<td>Length: Wheel: Radius</td>
<td>0.021</td>
<td>m</td>
<td>[5]</td>
</tr>
<tr>
<td>$J_{b.\phi_x}$</td>
<td>Moment of Inertia: Wheel</td>
<td>$7.46 \cdot 10^{-6}$</td>
<td>kg $\cdot m^2$</td>
<td>[5]</td>
</tr>
<tr>
<td>$R_{mtr}$</td>
<td>Motor: Resistance</td>
<td>4.4</td>
<td>$\Omega$</td>
<td>[5]</td>
</tr>
<tr>
<td>$k_{mtr.bEMF}$</td>
<td>Motor: Coefficient of Back EMF</td>
<td>0.495</td>
<td>$\frac{V}{V_{rad}}$</td>
<td>[5]</td>
</tr>
<tr>
<td>$k_{mtr.T}$</td>
<td>Motor: Coefficient of Torque</td>
<td>0.470</td>
<td>$\frac{N \cdot m}{V_{rad}}$</td>
<td>[5]</td>
</tr>
<tr>
<td>$k_{fr.m2w}$</td>
<td>Motor: Coefficient of friction: [DC Motor] to [Wheel]</td>
<td>??</td>
<td>-</td>
<td>Sec. ??</td>
</tr>
</tbody>
</table>

39
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).

\[
\begin{bmatrix}
\theta_{g.l} \\
\theta_{g.r} \\
\theta_{g.av} \\
\phi_y
\end{bmatrix} =
\begin{bmatrix}
\theta_{b.l} + \phi_x \\
\theta_{b.r} + \phi_x \\
\frac{1}{2} \cdot (\theta_{g.l} + \theta_{g.r}) \\
r_w \cdot \left(\frac{\theta_{g.r} - \theta_{g.l}}{l_{b.w}}\right)
\end{bmatrix}
\] (4.1)

\[
\begin{bmatrix}
p_{w.x} \\
p_{w.y} \\
p_{w.z}
\end{bmatrix} =
\begin{bmatrix}
r_w \cdot \dot{\theta}_{g.av} \cdot \cos(\phi_y) \\
r_w \cdot \dot{\theta}_{g.av} \cdot \sin(\phi_y) \\
0
\end{bmatrix}
\] (4.2)

\[
\begin{bmatrix}
p_{w.x} \\
p_{w.y} \\
p_{w.z}
\end{bmatrix} =
\begin{bmatrix}
\int \dot{p}_{w.x} \cdot dt + p_{w.x}(0) \\
\int \dot{p}_{w.y} \cdot dt + p_{w.y}(0) \\
\int \dot{p}_{w.z} \cdot dt + p_{w.z}(0)
\end{bmatrix}
\] (4.3)

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

\[
\begin{bmatrix}
p_{b.x} \\
p_{b.y} \\
p_{b.z}
\end{bmatrix} =
\begin{bmatrix}
p_{w.x} + l_{b.c2a} \cdot \sin(\phi_x) \cdot \cos(\phi_y) \\
p_{w.y} + l_{b.c2a} \cdot \sin(\phi_x) \cdot \sin(\phi_y) \\
p_{w.z} + l_{b.c2a} \cdot \cos(\phi_x)
\end{bmatrix}
\] (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 $\theta$ and Body Pitch $\phi_x$

Equation (4.7) corresponds to wheel angular position $\theta$ and body pitch $\phi_x$.

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

$$
\mathbf{K}_{1,\ddot{x}} = \begin{bmatrix} +k_{1,1} & +k_{1,2} \\ +k_{1,2} & +k_{1,3} \end{bmatrix} \tag{4.8}
$$

$$
\mathbf{K}_{1,\ddot{x}} = \begin{bmatrix} +k_{1,4} & -k_{1,4} \\ -k_{1,4} & +k_{1,4} \end{bmatrix} \tag{4.9}
$$

$$
\mathbf{K}_{1,x} = \begin{bmatrix} 0 & 0 \\ 0 & +k_{1,5} \end{bmatrix} \tag{4.10}
$$

$$
\mathbf{K}_{1,v} = \begin{bmatrix} +k_{1,6} & +k_{1,6} \\ -k_{1,6} & -k_{1,6} \end{bmatrix} \tag{4.11}
$$

$$
k_{1,1} = \left(2 \cdot m_w + m_b\right) \cdot r_w + J_w \tag{4.12}
$$

$$
k_{1,2} = m_b \cdot r_w \cdot l_{b,c2a} \tag{4.13}
$$

$$
k_{1,3} = m_b \cdot l_{b,c2a}^2 + J_{b,\phi_x} \tag{4.14}
$$

$$
k_{1,4} = 2 \cdot \left( k_{mtr,T} \cdot k_{mtr,bEMF} \cdot \frac{R_{mtr}}{R_{mtr}} + k_{fr,m2w} \right) \tag{4.15}
$$

$$
k_{1,5} = -m_b \cdot a_g \cdot l_{b,c2a} \tag{4.16}
$$

$$
k_{1,6} = \frac{k_{mtr,T}}{R_{mtr}} \tag{4.17}
$$
4.2.2. Body Yaw $\phi_y$

Equation (4.18) corresponds to body yaw $\phi_y$.

$$k_{2,\ddot{x}} \cdot \left[ \ddot{\phi}_y \right] + k_{2,\dot{x}} \cdot \left[ \dot{\phi}_y \right] = k_{2,v} \cdot \left[ v_{mtr,r} - v_{mtr,l} \right]$$  \hspace{1cm} (4.18)

$$k_{2,0} = \frac{l_{b,w}}{r_w}$$  \hspace{1cm} (4.19)

$$k_{2,\dot{x}} = \frac{1}{2} \cdot m_w \cdot l_{b,w}^2 + \frac{1}{2} \cdot k_{2,0} \cdot J_{b,\phi_y} \cdot k_{2,\ddot{x}}$$  \hspace{1cm} (4.20)

$$k_{2,\ddot{x}} = \frac{1}{2} \cdot k_{2,0} \cdot k_{1,4}$$  \hspace{1cm} (4.21)

$$k_{2,v} = \frac{1}{2} \cdot k_{2,0} \cdot k_{1,6}$$  \hspace{1cm} (4.22)
4.3. State-Space Representation

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

\[
\begin{align*}
\dot{x}_{n \times 1} &= A_{n \times n} \cdot x_{n \times 1} + B_{n \times p} \cdot u_{p \times 1} \\
y_{m \times 1} &= C_{m \times n} \cdot x_{n \times 1} + D_{m \times p} \cdot u_{p \times 1}
\end{align*}
\] (4.23)

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

\[
x_{n \times 1} = \begin{bmatrix}
\theta \\
\dot{\phi}_x \\
\dot{\theta} \\
\dot{\phi}_x \\
\phi_y \\
\phi_y
\end{bmatrix}
\] (4.24)

\[
u_{p \times 1} = \begin{bmatrix}
v_{mtr.l} \\
v_{mtr.r}
\end{bmatrix}
\] (4.25)
The derivation of indices for the system matrices $A$ and $B$ which are nonintuitive are derived from Equations (4.7) - (4.18). in Equations (4.26) - (4.27).

\[
\begin{bmatrix}
\dot{\theta} \\
\dot{\phi}_x
\end{bmatrix}
+ \begin{bmatrix}
\ddot{\theta} \\
\ddot{\phi}_x
\end{bmatrix}
+ \begin{bmatrix}
\theta \\
\phi_x
\end{bmatrix}
= \begin{bmatrix}
v_{\text{mtr}.l} \\
v_{\text{mtr}.r}
\end{bmatrix}
\]

\[
A_1 \cdot \begin{bmatrix}
\dot{\theta} \\
\dot{\phi}_x
\end{bmatrix}
= -K_{1,\ddot{x}} \cdot \begin{bmatrix}
\dot{\theta} \\
\dot{\phi}_x
\end{bmatrix}
+ -K_{1,\dot{x}} \cdot \begin{bmatrix}
\theta \\
\phi_x
\end{bmatrix}
+ K_{1,\dot{\theta}} \cdot K_{1,\theta} \cdot \begin{bmatrix}
v_{\text{mtr}.l} \\
v_{\text{mtr}.r}
\end{bmatrix}
\]

(4.26)

\[
k_{2,\ddot{x}} \cdot \begin{bmatrix}
\ddot{\phi}_y
\end{bmatrix}
+ k_{2,\dot{x}} \cdot \begin{bmatrix}
\dot{\phi}_y
\end{bmatrix}
= k_{2,v} \cdot \begin{bmatrix}
v_{\text{mtr}.r} - v_{\text{mtr}.l}
\end{bmatrix}
\]

(4.27)

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

\[
X_{2x2} = \begin{bmatrix}
+X_{(1,1)} & +X_{(1,2)} \\
+X_{(2,1)} & +X_{(2,2)}
\end{bmatrix}
\]

(4.28)

\[
X^{-1} = \frac{1}{\det(X)} \cdot \text{adj}(X) = \frac{1}{X_{(1,1)} \cdot X_{(2,2)} - X_{(1,2)} \cdot X_{(2,1)}} \cdot \begin{bmatrix}
+X_{(2,2)} & -X_{(2,1)} \\
-X_{(1,2)} & +X_{(1,1)}
\end{bmatrix}
\]

(4.29)

\[
\det(X) \neq 0
\]

(4.30)

\[
\det(K_{1,\dot{x}}) = k_{1,1} \cdot k_{1,3} - k_{1,2} \cdot k_{1,2} \neq 0
\]

(4.31)
The $A$ matrix and the state vector $x$ are exhibited in Equation (4.32).

\[
A \cdot x = \begin{bmatrix}
A_0^{(1,1)} & A_0^{(1,2)} & A_1^{(1,1)} & A_1^{(1,2)} & 0 & 0 \\
A_0^{(2,1)} & A_0^{(2,2)} & A_1^{(2,1)} & A_1^{(2,2)} & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & A_2
\end{bmatrix}
\begin{bmatrix}
\theta \\
\dot{\theta} \\
\phi_x \\
\dot{\phi}_x \\
\phi_y \\
\dot{\phi}_y
\end{bmatrix}
\tag{4.32}
\]

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

\[
B \cdot u = \begin{bmatrix}
B_1^{(1,1)} & B_1^{(1,2)} \\
B_1^{(2,1)} & B_1^{(2,2)} \\
0 & 0 \\
-B_2 & +B_2
\end{bmatrix}
\begin{bmatrix}
v_{mtr.l} \\
v_{mtr.r}
\end{bmatrix}
\tag{4.33}
\]

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

\[
C \cdot x = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\theta \\
\dot{\theta} \\
\phi_x \\
\dot{\phi}_x \\
\phi_y \\
\dot{\phi}_y
\end{bmatrix}
\tag{4.34}
\]

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

\[
D \cdot u = \begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
v_{mtr.l} \\
v_{mtr.r}
\end{bmatrix}
\tag{4.35}
\]
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) \(J_{\phi_x}\)

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).

\[
J_{\phi_x} = \frac{m_b \cdot l_{b,c}^2}{3} \quad (4.36)
\]

4.4.2. Moment of Inertia: Body: Y-axis (Yaw) \(J_{\phi_y}\)

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)

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

\[
\begin{align*}
\theta &= \phi_x \\
\mathbf{u} &= \mathbf{0}
\end{align*}
\]

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).

\[
\begin{align*}
\ddot{\phi}_x + \frac{k_{1.5}}{k_{1.2} + k_{1.3}} \cdot \phi_x &= 0 \\
&\Leftrightarrow \phi_x &= 0
\end{align*}
\]

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.
\[
\begin{align*}
\mathbf{K}_{1,\dddot{x}} \cdot \begin{bmatrix} \dddot{\theta} \\ \dddot{\phi}_x \end{bmatrix} + \mathbf{K}_{1,\dddot{x}} \cdot \begin{bmatrix} \dddot{\theta} \\ \dddot{\phi}_x \end{bmatrix} + \mathbf{K}_{1,x} \cdot \begin{bmatrix} \dot{\theta} \\ \phi_x \end{bmatrix} & = \mathbf{K}_{1,v} \cdot \begin{bmatrix} v_{mtr,l} \\ v_{mtr,r} \end{bmatrix} \\
\mathbf{K}_{1,\dddot{x}} \cdot \begin{bmatrix} \dddot{\phi}_z \\ \dddot{\phi}_z \end{bmatrix} + \mathbf{K}_{1,\dddot{x}} \cdot \begin{bmatrix} \dddot{\phi}_z \\ \dddot{\phi}_z \end{bmatrix} + \mathbf{K}_{1,x} \cdot \begin{bmatrix} \phi_x \\ \phi_x \end{bmatrix} & = \mathbf{K}_{1,v} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{align*}
\]
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).

\[
k_w = \frac{k_{1.5}}{k_{1.2} + k_{1.3}} = \frac{-m_b \cdot a_g \cdot l_{b,c2a}}{\left(m_b \cdot r_w \cdot l_{b,c2a}\right) + \left(m_b \cdot l_{b,c2a}^2 + J_b \cdot \phi_x\right)} \tag{4.43}
\]

\[
k_w = \frac{-m_b \cdot l_{b,c2a} \cdot a_g}{m_b \cdot l_{b,c2a} \cdot r_w + m_b \cdot l_{b,c2a}^2 + \left(m_b \cdot l_{b,c2a}^2 \cdot \frac{1}{3}\right)} = \frac{-a_g}{r_w + l_{b,c2a} \cdot \left(1 + \frac{1}{3}\right)} \tag{4.44}
\]

**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).

\[
\ddot{y} + \omega^2 \cdot y = \omega^2 \cdot u \tag{4.45}
\]

\[
\ddot{\phi}_x + k_w \cdot \phi_x = k_w \cdot 0
\]

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] $\omega_p$, as is exhibited in Equation (4.46).

\[
\omega_p^2 = k_w = \frac{-a_g}{r_w + l_{b,c2a} \cdot \frac{4}{3}} \tag{4.46}
\]

This proves significant since $\omega_p$ represents the angular frequency of the pendulum, which is a measurable value, and since $k_w$ 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).

\[
l_{b,c2a} = -\frac{3}{4} \left(\frac{a_g}{\omega_p^2} + r_w\right) = \frac{3}{4} \cdot \left(\frac{a_g}{\left(2 \cdot \pi \cdot f_p\right)^2} + r_w\right) \tag{4.47}
\]
4.5. Bibliography


5.1. Additional Dynamics

Additional dynamics may be incorporated into a state feedback regulating system in order to benefi-
cially 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.  

3. The plant is shifted to the right side of the system depiction.
\[ \dot{x} = Ax + Bu \]
\[ y = Cx + Du \]

Hardware Dynamics:
State-Space Representation
Continuous
\[ C = \text{eye}(6,6) \]
\[ D = \text{zeros}(6,2) \]

-1
Invert
\[ u \]
\[ y = x(t) \]

Figure 5.1.: [Additional Dynamics]: State Feedback Regulator
Figure 5.2.: [Additional Dynamics]: 1.0 Additional Dynamics (Design View)
Theoretical Plant

\[ K^*u \]
State-Feedback Gains

Invert

\[ \begin{bmatrix} x_1 \\ x_1 \\ x_5 \\ x_5 \\ x_1 \\ x_5 \end{bmatrix} \]

Hardware Dynamics:
State-Space Representation

Continuous

\[ C = \text{eye}(6,6) \]
\[ D = \text{zeros}(6,2) \]

Discrete

\[ C.a = \text{eye}(2,2) \]
\[ D.a = \text{zeros}(2,2) \]

Additional Dynamics:
State-Space Representation

\[ x_{a.1} = A_{a}x_{a} + Bu_{a} \]
\[ y_{a} = C_{a}x_{a} + D_{a}u_{a} \]

Additional Dynamics: 1.1 Additional Dynamics (Design View)

\[ x_{sys} = [x; x.a] \]

\[ u \]
\[ y = x \]
\[ x.5 = x.a.2 \]
\[ x.sys = [x; x.a] \]

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)
Controller

Plant

K*u

State-Feedback Gains [x]

-1

Invert [x]

K*u

State-Feedback Gains [x].a

-1

Invert [x.a]

Additional Dynamics:
State-Space Representation
Continuous

\[
C = \text{eye}(6, 6)
\]

\[
D = \text{zeros}(6, 2)
\]

Discrete

\[
C.a = \text{eye}(2, 2)
\]

\[
D.a = \text{zeros}(2, 2)
\]

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, \([\text{in this case, system states } x \text{ and } x_a]\), 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 \([\text{desired}]\) 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 $\psi$ 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 $\theta$ and body angular position (yaw) $\phi_y$. 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).

$$
\begin{align*}
\dot{x}_{nx1} &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot x_{nx1} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_{\phi_y} \end{bmatrix} \\
\dot{y}_{nx1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot x_{nx1} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_{\phi_y} \end{bmatrix}
\end{align*}
$$

(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).

\[
\begin{align*}
\dot{x}_{nx1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot x_{nx1} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_\phi \end{bmatrix} \\
y_{mx1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot x_{nx1} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e_\theta \\ e_\phi \end{bmatrix}
\end{align*}
\]

(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, \textit{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) $\phi_x$ 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 $\theta$
Figure 5.12.: [Control Gains: LQR]: Simulation Results: Body Angular Position $\phi_y$
Figure 5.13.: [Control Gains: LQR]: Simulation Results: Body Angular Position $\phi_x$
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 $v_{\text{motorDriver}}$
5.2. Bibliography

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 $\theta$ and body angular position (yaw) $\phi_y$, while maintaining standard regulation of all other plant states.

While moving in accordance with the dynamic commands provided to the wheel angular position $\theta$ and body angular position (yaw) $\phi_y$ 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


Appendices
A.1. minseg.m

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

```matlab
%% [Global ]
minseg_0p0p0p0_global

%% [Input ]: Model
minseg_1p0p0p0_input

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

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

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

%% [Init ]: Define parameters
minseg_2p0p0p0_init_general
minseg_2p1p0p0_init_model_general
minseg_2p1p1p0_init_model_plant
minseg_2p1p2p0_init_model_controller
minseg_2p1p3p0_init_model_io
```
minseg_2p1p9p0_init_model_build
minseg_2p2p0p0_init_serial_write
minseg_2p2p1p0_init_serial_read
minseg_2p2p2p0_init_serial_general
minseg_2p2p3p0_init_serial_reads
minseg_2p2p9p0_init_model_build

%minseg_2p3p0p0_init_build

%% [Process]:
%
% build (normal mode) / run (external mode)
if ui.x.build
minseg_3p0p0p0_process_build
end
%
% serial transmit (normal mode only)
if (ui.x.write || ui.x.read || ui.x.plot || ui.x.save )
minseg_3p1p0p0_process_serial_transmit
end
%
% serial post-processing
if (ui.x.read || ui.x.plot || ui.x.save )
minseg_3p2p0p0_process_serial_reads
%
% if ui.mdl.case == 2
% minseg_3p3p1p0_process_motorTF
% end
%
% if ui.mdl.case == 3
% minseg_3p3p1p0_process_gyroBias
% end
end
%% [Output]:
%
% save
if ui.x.save
minseg_4p0p0p0_output_save
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

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

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

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

%% [ Input ] : Model: Plant

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

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

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

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

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

ui.ctrl.body.pitch.theta.mode = 0;
% #: mode: input command: [input command unit]:
```
%% [Input ]: Model: Controller: motor.v

ui.ctrl.motor_v.mode = 1;

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

switch ui.ctrl.motor_v.mode

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

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

end

%% [Input ]: Model: Serial

ui.srl.mode.address = 0;

% 0: left usb port (2015 Macbook Pro)
% 1: left−rear usb port (2008 Macbook Pro)

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

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

%% End

Code Listing A.3: [minseg.m]: User Inputs
A.1.3. Initialization
A.1.3.1. General

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

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

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

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

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

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

%% End
```

Code Listing A.4: [minseg.m]: Initialization - General
A.1.3.2. Model
A.1.3.2.1. General

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

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

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

%% [Init]: Define general model parameters
mdl.object = get_param(mdl.label, 'Object');

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

%% End
```

Code Listing A.5: [minseg.m]: Initialization - Model - General
A.1.3.2.2. Plant

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

```matlab
%% [Init]: Initialize user-defined parameters
plant.supply.mode = ui.plant.supply.mode;
plant.dynamics.mode = ui.plant.dynamics.mode;
plant.n.batteries = ui.plant.n.batteries;
plant.x.bluetoothModule = ui.plant.x.bluetoothModule;

%% [Init]: Define general plant parameters
switch plant.supply.mode
    case 0; plant.supply.v = 9.00; % [V]
    case 1; plant.supply.v = 4.50; % [V]
end

a.gravity = 9.81; % acceleration [m / s^2]
load('bessel poles.mat')

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

%% [Init]: Define parameters based on user-specified plant dynamics
switch plant.dynamics.mode
    case 0; minseg_2p1p1p1_init_model_plant_hardware
    case 1; minseg_2p1p1p2_init_model_plant_nonlinearDynamics
    case 2; minseg_2p1p1p3_init_model_plant_linearDynamics
end
```
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

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

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

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

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

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

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

% discretize transfer function
mtr.encoder.filter.z.tf = c2d( mtr.encoder.filter.s.tf ... 
                      , mdl.T.sample ... 
                      );
```
% break transfer function into numerator and denominator polynomials
[mtr.encoder.filter.s.num ..., mtr.encoder.filter.s.den ...]
= tfdata ...
(mtr.encoder.filter.s.tf ...);

[mtr.encoder.filter.z.num ..., mtr.encoder.filter.z.den ...]
= tfdata ...
(mtr.encoder.filter.z.tf ...);

% convert cells to matrices
mtr.encoder.filter.s.num = mtr.encoder.filter.s.num{:};
mtr.encoder.filter.s.den = mtr.encoder.filter.s.den{:};
mtr.encoder.filter.z.num = mtr.encoder.filter.z.num{:};
mtr.encoder.filter.z.den = mtr.encoder.filter.z.den{:};

%% [Init]: Motor: Encoder: angVel bessel filter: state-space

% create s-plane state space equations (canonical representation)
mtr.encoder.filter.s.ss.A = diag(ones(mtr.encoder.filter.order - 1, 1), 1);
mtr.encoder.filter.s.ss.A(end,:) = mtr.encoder.filter.s.poles(end-1:2);
mtr.encoder.filter.s.ss.A(end,:) = mtr.encoder.filter.s.ss.A(end,:) ./ mtr.encoder.filter.s.poles(1) * -1;
mtr.encoder.filter.s.ss.B = [zeros(mtr.encoder.filter.order - 1, 1); 1]

mtr.encoder.filter.s.ss.C = [zeros(1, mtr.encoder.filter.order - 1) 1]

mtr.encoder.filter.s.ss.D = 0;

% discretize s-plane state space equations (canonical representation)
[mtr.encoder.filter.z.ss.A ... phi
, mtr.encoder.filter.z.ss.B ... gamma
] = zohe ...
(mtr.encoder.filter.s.ss.A ... A
, mtr.encoder.filter.s.ss.B ... B

99
mtr. encoder. filter. z.ss.C = mtr. encoder. filter. s.ss.C;
mtr. encoder. filter. z.ss.D = mtr. encoder. filter. s.ss.D;

%% [ Init ]: Gyroscope
gyro. dlpf. mode = 0; % [ default: 0 ]
% | # | maxValue [deg/s] | bandwidth [Hz] | delay [s] |
% | 0 | +/- 0250 | 256 | 00.98 |
% | 1 | +/- 0500 | 188 | 01.90 |
% | 2 | +/- 1000 | 098 | 02.80 |
% | 3 | +/- 2000 | 042 | 04.80 |
% | 4 | +/- ??? | 020 | 08.30 |
% | 5 | +/- ??? | 010 | 13.40 |
% | 6 | +/- ??? | 005 | 18.60 |

switch gyro. dlpf. mode
    case 0; gyro. maxVal = 0250 * k_deg2rad;
    case 1; gyro. maxVal = 0500 * k_deg2rad;
    case 2; gyro. maxVal = 1000 * k_deg2rad;
    case 3; gyro. maxVal = 2000 * k_deg2rad;
end

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

% [ source: 1. Test Cases/1. Gyro Bias Calibration]
gyro. x. bias = -266.0779700;
gyro. y. bias = -135.5037500;
gyro. z. bias = -034.3493271;

 gyro. x. reset = 0;
gyro. y. reset = 0;
gyro. z. reset = 0;

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

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

% create transfer function

gyro.filter.s.tf = tf( gyro.filter.s.poles(end) ... 
, gyro.filter.s.poles ... 
);

% discretize transfer function

gyro.filter.z.tf = c2d( gyro.filter.s.tf ... 
, mdl.T.sample ... 
);

% break transfer function into numerator and denominator polynomials

[ gyro.filter.s.num ... 
, gyro.filter.s.den ... 
] = tfdata ... ( gyro.filter.s.tf ... 
);

[ gyro.filter.z.num ... 
, gyro.filter.z.den ... 
] = tfdata ... ( gyro.filter.z.tf ... 
);

% convert cells to matrices

gyro.filter.s.num = gyro.filter.s.num{:};
gyro.filter.s.den = gyro.filter.s.den{:};
gyro.filter.z.num = gyro.filter.z.num{:};
gyro.filter.z.den = gyro.filter.z.den{:};

%% [Init]: Gyroscope : angVel bessel filter: state-space

% create s-plane state space equations (canonical representation)

gyro.filter.s.ss.A = diag( ones( gyro.filter.order - 1, 1 ), 1); 
gyro.filter.s.ss.A(end,:) = gyro.filter.s.poles( end : -1 : 2 );
gyro.filter.s.ss.A( end,:) = gyro.filter.s.ss.A( end,:) ...
gyro_filter.s.poles( 1 ) * -1;

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

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

 gyro_filter.z.ss.C = gyro_filter.s.ss.C;
gyro_filter.z.ss.D = gyro_filter.s.ss.D;

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

assert( accel afs_sel.mode == 0 );

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

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

%% End

Code Listing A.7: [minseg.m]: Initialization - Model - Plant - Hardware
A.1.3.2.2. Nonlinear Dynamics Model

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

%% End

Code Listing A.8: [minseg.m]: Initialization - Model - Plant - Nonlinear Dynamics Model
A.1.3.2.3. Linear Dynamics Model

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

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

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

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

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

%% [ Init ]: Plant: Body: Masses

% note: body does not include wheels.

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

% plant.board.m = 1.000 * k.lb2kg; % [kg]
% plant.motorCable.m = 0.010 * k.lb2kg; % (quantity: 1) [kg] [note low precision.]
% plant.motor.m = 0.220 * k.lb2kg; % (quantity: 1) [kg]
% plant.battery.m = 1.000 * k.lb2kg; % (quantity: 1) [kg]
% plant.bluetooth.m = 0.000 * k.lb2kg; % bluetooth module [kg] [note low precision.]
% plant.usbCable.m = 0.040 * k.lb2kg; % (not included) [kg]

% plant.body.m = plant.board.m + plant.motor.m * 2 + plant.motorCable.m * 2 + plant.battery.m * plant.n.battery + plant.bluetooth.m * plant.x.bluetoothModule;
% % mass [kg]
```
plant.body.m = 1.030; % (not included) [kg]

% net measurement taken to reduce rounding errors.
% [taken with 6 batteries].

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

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

switch plant.x.bluetoothModule

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

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

end

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

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

plant.body.J.x = plant.body.m * plant.body.l.c^2 ...
  / 3;
  % moment of inertia (pitch) [kg / m^2]
  % (measured from center of mass of robot)
\[ \text{plant.body.J.y} = \text{plant.body.m} \ldots \]
\[ \times \left( \text{plant.body.l.w^2} + \text{plant.body.l.d^2} \right) \ldots \]
\[ / 12; \]
\[ \% \text{moment of inertia (yaw) [kg / m^2]} \]
\[ \% \text{(measured from center of mass of robot)} \]

\[ \% \text{[Init]: Plant: Net (body + 2 \times wheel)} \]
\[ \text{plant.net.m} = \text{plant.body.m} + 2 \times \text{plant.wheel.m}; \% \text{[kg]} \]

\[ \% \text{[Init]: Plant: Motor} \]
\[ \text{mtr.R} = 4.400; \% \text{resistance [ohm]} \] \[ \text{[source: howard]} \]
\[ \text{mtr.k.ddlambda} = 0.495; \% \text{back EMF constant [V*s / rad]} \] \[ \text{[source: howard]} \]
\[ \text{mtr.k.torque} = 0.470; \% \text{torque constant [N*m / A]} \] \[ \text{[source: howard]} \]

\text{switch plant.x.bluetoothModule}

\text{case 0} \% \text{Not inserted}
\text{switch plant.n.batteries}
\text{case 0}
\text{mtr.k.v2w} = 1.000; \% \text{transfer function (y/u) [rad / (s*V)]}
\% (measured when body is upright AND \% both wheels are at equivalent speed \% in a common direction.)

\text{case 5}
\text{mtr.k.v2w} = 1.000; \% \text{transfer function (y/u) [rad / (s*V)]}
\% (measured when body is upright AND \% both wheels are at equivalent speed \% in a common direction.)

\text{case 6}
\text{mtr.k.v2w} = 1.000; \% \text{transfer function (y/u) [rad / (s*V)]}
\% (measured when body is upright AND \% both wheels are at equivalent speed \% in a common direction.)

\text{end}
```matlab
% [Init]: Plant: State space model term abbreviations

% wheel.theta and body.theta.x (pitch) (psi)
plant.q(1,1) = plant.net.m * plant.wheel.r^2 + plant.wheel.J;
plant.q(2,1) = plant.body.m * plant.wheel.r^2 * plant.body.l.c;
plant.q(3,1) = plant.body.m * plant.body.l.c^2 + plant.wheel.J;
plant.q(4,1) = mtr.k.torque * mtr.k.dlambda / mtr.R + mtr.k.friction;
plant.q(5,1) = plant.body.m * a.gravity * plant.body.l.c;
plant.q(6,1) = mtr.k.torque / mtr.R;

plant.Q{1,1} = [ +plant.q(1) +plant.q(2) +plant.q(2) +plant.q(3) ];
```
plant.Q{2,1} = 2 * [ +plant.q(4) -plant.q(4)
-plant.q(4) +plant.q(4) ];

plant.Q{3,1} = [ 0 0
0 -plant.q(5) ];

plant.Q{4,1} = [ +plant.q(6) +plant.q(6)
-plant.q(6) -plant.q(6) ];

% body.theta.y (yaw) (phi)
plant.r(1,1) = plant.body.l.w / plant.wheel.r;

plant.R{1,1} = 0.5 * plant.wheel.m * plant.body.l.w^2
+ plant.body.J.y
+ 0.5 * plant.r(1)^2 * plant.wheel.J;

plant.R{2,1} = 0.5 * plant.r(1)^2 * plant.q(4);

plant.R{3,1} = 0.5 * plant.r(1) * mtr.k.torque / mtr.R;

% overall
plant.a{1,1} = -plant.Q{1} \ plant.Q{3};

plant.a{2,1} = -plant.Q{1} \ plant.Q{2};

plant.a{3,1} = -plant.R{1} \ plant.R{2}; % note the backslash.

plant.b{1,1} = +plant.Q{1} \ plant.Q{4};

plant.b{2,1} = +plant.R{1} \ plant.R{3};

%%% [Init]: Plant State Space Model: A

plant.A(1,1) = 0;

plant.A(1,2) = 0;

plant.A(1,3) = 1;

plant.A(1,4) = 0;

plant.A(1,5) = 0;

plant.A(1,6) = 0;

plant.A(2,1) = 0;

plant.A(2,2) = 0;

plant.A(2,3) = 0;

plant.A(2,4) = 1;

plant.A(2,5) = 0;

plant.A(2,6) = 0;

plant.A(3,1) = plant.a{1}(1,1);
plant.A(3,2) = plant.a{1}(1,2);
plant.A(3,3) = plant.a{2}(1,1);
plant.A(3,4) = plant.a{2}(1,2);
plant.A(3,5) = 0;
plant.A(3,6) = 0;

plant.A(4,1) = plant.a{1}(2,1);
plant.A(4,2) = plant.a{1}(2,2);
plant.A(4,3) = plant.a{2}(2,1);
plant.A(4,4) = plant.a{2}(2,2);
plant.A(4,5) = 0;
plant.A(4,6) = 0;

plant.A(5,1) = 0;
plant.A(5,2) = 0;
plant.A(5,3) = 0;
plant.A(5,4) = 0;
plant.A(5,5) = 0;
plant.A(5,6) = 1;

plant.A(6,1) = 0;
plant.A(6,2) = 0;
plant.A(6,3) = 0;
plant.A(6,4) = 0;
plant.A(6,5) = 0;
plant.A(6,6) = plant.a{3};

%%% [ Init ]: Plant State Space Model: B

plant.B(1,1) = 0;
plant.B(2,1) = 0;
plant.B(3,1) = plant.b{1}(1,1);
plant.B(4,1) = plant.b{1}(2,1);
plant.B(5,1) = 0;
plant.B(6,1) = -plant.b{2};

plant.B(1,2) = 0;
plant.B(2,2) = 0;
plant.B(3,2) = plant.b{1}(1,2);
plant.B(4,2) = plant.b{1}(2,2);
plant.B(5,2) = 0;
Code Listing A.9: [minseg.m]: Initialization - Model - Plant - Linear Dynamics Model

```matlab
plant.B(6,2) = +plant.b(2);

%% [ Init ]: Plant State Space Model: C, D

plant.C = eye(size(plant.A));
plant.D = zeros(size(plant.A,1),size(plant.C,2));
```

110
A.1.3.2.3. Controller

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

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

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

%% [ Init ]: Define controller model parameters
switch ctrl.motor_v.mode
    case 0
    case 1
        ctrl.motor_v.pid.motor_w.k.p = 0.500;
        ctrl.motor_v.pid.motor_w.k.i = 1.000;
        ctrl.motor_v.pid.motor_w.k.d = 0.000;
        ctrl.motor_v.pid.motor_w.int.maxVal = +plant.supply.v;
        ctrl.motor_v.pid.motor_w.int.minVal = -plant.supply.v;
end
```

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

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

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

% plant: hardware
io.write.serial.hardware.
    var = Simulink.Variant( 'plant_dynamics_mode == 0' );
io.write.serial.hardware.ff.
    var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
io.write.serial.hardware.pid.
    var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );
io.write.serial.hardware.ff.standard.
    var = Simulink.Variant( 'mdl_case == 0' );
io.write.serial.hardware.ff.motorCharacterization.
    var = Simulink.Variant( 'mdl_case == 1' );
io.write.serial.hardware.pid.standard.
    var = Simulink.Variant( 'mdl_case == 0' );
io.write.serial.hardware.pid.sensorCalibration.
    var = Simulink.Variant( 'mdl_case == 2' );

% plant: nonlinearDynamics
io.write.serial.nonlinearDynamics.
    var = Simulink.Variant( 'plant_dynamics_mode == 1' );
io.write.serial.nonlinearDynamics.ff.
    var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
io.write.serial.nonlinearDynamics.pid.
    var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );
```
io.write.serial.nonlinearDynamics.ff.standard.var = Simulink.Variant('mdl_case == 0');

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

% plant: nonlinearDynamics

io.write.serial.linearDynamics.var = Simulink.Variant('plant_dynamics_mode == 2');

io.write.serial.linearDynamics.ff.var = Simulink.Variant('ctrl_motor_v_mode == 0');

io.write.serial.linearDynamics.pid.var = Simulink.Variant('ctrl_motor_v_mode == 1');

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

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

%%% [ Init ]: Write commands

io.write.ctrl.motor_v.cmd.tStart = ui.io.write.ctrl.motor_v.cmd.tStart;

% [ s ]

io.write.ctrl.motor_v.cmd.val.x = ui.io.write.ctrl.motor_v.cmd.val.x;

% [ cmd ]

io.write.ctrl.motor_v.cmd.val_norm.dx.max = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max;

% [ cmd.norm / s ]

io.write.ctrl.motor_v.cmd.val_norm.dx.min = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min;

% [ cmd.norm / s ]

%%% End
A.1.3.2.5. Build Parameters

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

```matlab
%% [Init ]: Initialize list of general parameters used within Simulink model
mdl.parameter.label = { };

% Specify parameters which will be used in model:
mdl.parameter.label = [ ...
mdl.parameter.label
{
'k.intmax.uint8'
'mdl.mode'
'mdl.case'
'mdl.T.sample'
'plant.dynamics.mode'
'plant.supply.v'
'ctrl.motor_v.mode'
'ctrl.motor_v.ff.motor_v.var'
'ctrl.motor_v.pid.motor_w.var'
'io.write.serial.var'
'io.write.scopes.var'
'io.write.serial.hardware.var'
'io.write.serial.hardware.ff.var'
'io.write.serial.hardware.ff.standard.var'
'io.write.serial.hardware.ff.motorCharacterization.var'
'io.write.serial.hardware.pid.var'
'io.write.serial.hardware.pid.standard.var'
'io.write.serial.hardware.pid.sensorCalibration.var'
'io.write.serial.nonlinearDynamics.var'
'io.write.serial.nonlinearDynamics.ff.var'
'io.write.serial.nonlinearDynamics.ff.standard.var'
'io.write.serial.nonlinearDynamics.pid.var'
'io.write.serial.nonlinearDynamics.pid.standard.var'
```
'io.write.serial.linearDynamics.var'
'io.write.serial.linearDynamics.ff.var'
'io.write.serial.linearDynamics.ff.standard.var'
'io.write.serial.linearDynamics.pid.var'
'io.write.serial.linearDynamics.pid.standard.var'

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

}

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

switch plant.dynamics.mode

case 0 % hardware

mdl.parameter.label = [...
mdl.parameter.label
{
'gyro.dlpf.mode'
'gyro.k_raw2actual'

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

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

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

'accel.k_raw2actual'
'mtr.driver.left.pin.pos'
'mtr.driver.left.pin.neg'
'mtr.driver.middle.pin.pos'
'mtr.driver.middle.pin.neg'

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

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

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

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

});

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

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

%% [ Init ]: Append case-dependent parameters: Controller: v.motor.input
switch ctrl.motor_v.mode

case 0 % feed-forward (input: motor.v)
mdl.parameter.label = [...
mdl.parameter.label
{
}
];

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

end

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

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

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

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

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

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

%% [Init]: Refresh model to update variant blocks
mdl.object.refreshModelBlocks

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

```matlab
%% End
```

Code Listing A.13: [minseg.m]: Initialization - Serial - Write
A.1.3.3.2. Read

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

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

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

while 1 % continue until 'break' command

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

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

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

end

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

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

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

121
% import output signal labels from bus block
srl.read{1,1}.block.busSelect.signals.out = ...
get_param ( srl.read{1,1}.block.busSelect.label ...
, 'OutputSignals'
);

srl.read{1,1}.block.busSelect.signals.out = ...
regexp ( srl.read{1,1}.block.busSelect.signals.out{:} ...
, '[^-;]*' ...
, 'match'
).
Enter

% verify equivalent number of each type of serial read preprocessing block:
assert ( size ( srl.read{1,1}.block.busSelect.signals.out , 1 ) == ...
size ( srl.read{1,1}.block.convert.label , 1 ) ...
, [ srl.read{1,1}.block.path ':\n' ...
'Less Convert blocks than number of signals.' ] ...
)

assert ( size ( srl.read{1,1}.block.busSelect.signals.out , 1 ) == ...
size ( srl.read{1,1}.block.bytepack.label , 1 ) ...
, [ srl.read{1,1}.block.path ':\n' ...
'Less Byte Pack blocks than number of signals.' ] ...
)

%%%% [Init]: Define serial read signal label and datatype parameters

% number of signals being transmitted:
srl.read{1,1}.n.signals = size ( srl.read{1,1}.block.convert.label , 1 );

% increase srl.read cell vector size to number of signals
srl.read{1,1}.n.signals = srl.read{1,1}.n.signals , 1 } = [];
srl.reads{ srl.read{1,1}.n.signals , 1 } = [];

% for each serial read signal existing within the model:
for i0 = 1 : srl.read{1,1}.n.signals
    % import the datalabel of that signal from the bus block
    srl.read{i0,1}.label = srl.read{1,1}.block.busSelect.signals.out{i0,1};
% import the datatype of that signal from the datatype conversion block
srl.read{i0,1}.type.original = ...
get_param( srl.read{1,1}.block.convert.label{i0,1}, 'OutDataTypeStr' );

% for posterity, set the datatype in the bytepack block to the same datatype.
set_param( srl.read{1,1}.block.bytepack.label{i0,1}, 'datatypes', ...
[ '{' ' ' srl.read{i0,1}.type.original ' ' '} ' ] )
end

%% [Init]: Define serial read signal size parameters

% initialize counters
srl.read{1,1}.n.Bytes = 0; % [bytes / read]
srl.read{1,1}.n.type.uint8 = 0; % [type: 'uint8' signals / read]
srl.read{1,1}.n.type.uint16 = 0; % [type: 'uint16' signals / read]
srl.read{1,1}.n.type.uint32 = 0; % [type: 'uint32' signals / read]
srl.read{1,1}.n.type.int8 = 0; % [type: 'int8' signals / read]
srl.read{1,1}.n.type.int16 = 0; % [type: 'int16' signals / read]
srl.read{1,1}.n.type.int32 = 0; % [type: 'int32' signals / read]
srl.read{1,1}.n.type.single = 0; % [type: 'single' signals / read]
srl.read{1,1}.n.type.double = 0; % [type: 'double' signals / read]

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

% increment counter for appropriate signal type [-]
switch srl.read{i0,1}.type.original
    case 'uint8'; srl.read{1,1}.n.type.uint8 = srl.read{1,1}.n.type.uint8 + 1;
    case 'uint16'; srl.read{1,1}.n.type.uint16 = srl.read{1,1}.n.type.uint16 + 1;
    case 'uint32'; srl.read{1,1}.n.type.uint32 = srl.read{1,1}.n.type.uint32 + 1;
    case 'int8'; srl.read{1,1}.n.type.int8 = srl.read{1,1}.n.type.int8 + 1;
    case 'int16'; srl.read{1,1}.n.type.int16 = srl.read{1,1}.n.type.int16 + 1;
    case 'int32'; srl.read{1,1}.n.type.int32 = srl.read{1,1}.n.type.int32 + 1;
    case 'single'; srl.read{1,1}.n.type.single = srl.read{1,1}.n.type.single + 1;
    case 'double'; srl.read{1,1}.n.type.double = srl.read{1,1}.n.type.double + 1;
end
 otherwise; error ('unknown datatype'); end

switch srl.read{i0,1}.type.original
  case 'uint8'; srl.read{i0,1}.n.bytes = 1; % [ (bytes/signal) / read ]
  case 'uint16'; srl.read{i0,1}.n.bytes = 2; % [ (bytes/signal) / read ]
  case 'uint32'; srl.read{i0,1}.n.bytes = 4; % [ (bytes/signal) / read ]
  case 'int8'; srl.read{i0,1}.n.bytes = 1; % [ (bytes/signal) / read ]
  case 'int16'; srl.read{i0,1}.n.bytes = 2; % [ (bytes/signal) / read ]
  case 'int32'; srl.read{i0,1}.n.bytes = 4; % [ (bytes/signal) / read ]
  case 'single'; srl.read{i0,1}.n.bytes = 4; % [ (bytes/signal) / read ]
  case 'double'; srl.read{i0,1}.n.bytes = 8; % [ (bytes/signal) / read ]
  otherwise; error ('unknown datatype'); end

srl.read{i0,1}.n.bits = srl.read{i0,1}.n.bytes ... * k.byte2bit; % [ (bits /signal) / read ]
srl.read{1,1}.n.Bytes = srl.read{1,1}.n.Bytes ... + srl.read{i0,1}.n.bytes; % [ bytes / read ]
end

srl.read{1,1}.n.Bits = srl.read{1,1}.n.Bytes ... * k.byte2bit; % [ bits / read ]

% verify number of bytes per read is not greater than arduino input buffer:
assert((srl.read{1,1}.n.Bytes + 1) <= 64 ... , ['Number of bytes being sent per read ' ... '(including 1 byte for Terminator)\n' ... 'is greater than size of\n' ... 'Arduino Mega 2650 input buffer (64 bytes).'] ... )

% [ Init ]; Initialize serial read value vectors

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

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

% serial address on PC
switch ui.srl.mode.address
    case 0; srl.address = '/dev/tty.usbmodem1411'; % left usb port (2015 PC)
    case 1; srl.address = '/dev/tty.usbmodem621'; % left-rear usb port (2008 PC)
end

% note: to determine current address, use command: {ls /dev/tty.*} in Terminal.app

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

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

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

%% [ Init ]: Serial buffer size

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

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

%% [Process]: Setup serial object

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

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

126
, 'BaudRate'      , srl.f.baud     ... | Hz |
, 'InputBufferSize', srl.bufferSize.in ... | bits |
, 'OutputBufferSize', srl.bufferSize.out ... | bits |
);

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

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

find more information on:

  TimerPeriod = 1
  Timeout    = 10
  StopBits   = 1
%

%% [Init]: Time required to perform transmission

% time required to transmit each write:

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

% time required to transmit each read:

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

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

% time required to perform all transmissions:

srl.T.transmit = srl.write{1,1}.T.transmit + srl.read{1,1}.T.transmit; % [ s ]

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

% number of board sample periods per serial process period
if ui.srl.T.decimation == 0
    srl.T.decimation = ceil( srl.T.transmit * 1.0000 / mdl.T.sample );
else
    srl.T.decimation = ui.srl.T.decimation;
end

% time until next serial process:
    srl.T.sample = mdl.T.sample * srl.T.decimation; % [ s ]

% [ Init ]: Verify serial period
% verify that total time to transmit serial data is not greater than
% time until start of next serial process:
assert( srl.T.transmit < srl.T.sample , 'Read period is greater than sample period.' );

% [ Init ]: Define serial transmits parameters
% number of reads to perform:
% note: serial duration may be specified directly in terms of samples or in terms of
time
try  srl.n.transmits = round( ui.srl.T.transmits / srl.T.sample ); % [ transmit
cycles]
catch; srl.n.transmits = ui.srl.n.transmits; % [ transmit
cycles]
end

srl.n.transmits = srl.n.transmits + 1; % 1 added for time = 0

% 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

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

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

%% [Init]: Define serial reads parameters

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

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

%% [Init]: Initialize serial reads value vectors

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

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

Code Listing A.16: [minseg.m]: Initialization - Serial - Reads
A.1.3.5. Build Parameters

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

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

4 %% [Init]: Initialize list of general parameters used within Simulink model
5 mdl.parameter.label = {};
6
7 % Specify parameters which will be used in model:
8 mdl.parameter.label = [...] mdl.parameter.label
9 { 'io.srl.read.rateTransition.initialCondition'
10 'io.srl.read.rateTransition.T.sample'
11 % cannot set certain hardware parameters via variables. [must hard-code.]
12 % 'srl.address';
13 % 'srl.f.baud';
14 }

15 %% [Init]: Relabel parameters for use within Simulink model
16 % Number of parameters specified
17 mdl.n.parameter = size( mdl.parameter.label, 1);

18 % Indices which contain periods:
19 mdl.parameter.z.period = regexp(mdl.parameter.label, '\.');

20 % For each parameter:
21 for i0 = 1 : mdl.n.parameter
22 % Create a new label in which all periods have been set to underscores:
23 mdl.parameter.label0 = mdl.parameter.label{ i0, 1};
24 mdl.parameter.label0( mdl.parameter.z.period{ i0, 1 } ) = '_';
25 % Set the data for the new label equal to the data from the old label:
```
eval([ mdl.parameter.label0 ' = ' mdl.parameter.label{i0,1} ';']);
end

%% Init: Update model scan for errors

set_param(mdl.label, 'SimulationCommand', 'update')

%% End
A.1.4. Processing
A.1.4.1. Build

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

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

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

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

%% End
A.1.4.2. Serial Transmission

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

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

% open serial channel
fopen ( srl.srl );

disp('Performing serial read:')

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

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

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

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

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

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

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

134
% wait for end of time sample:
if i0 ~= srl.n.transmits  % if not the last sample
    while toc( srl.T.one) < srl.T.sample  % then loop to wait until
        end  % a complete sample period
    end   % has passed before reading
    end   % again.
end

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

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

fclose(srl.srl);

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

% End

Code Listing A.19: [minseg.m]: Processing - Serial - Transmit
A.1.4.3. Serial Reads Post-Processing

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

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

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

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

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

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

% Pull corresponding values
if strcmp( srl.read{i0,1}.type.original, srl.type.out )
    % If intended signal datatype is equal to serial output, then use it immediately:
srl.reads{i0,1}.val = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );
else % If intended signal datatype is not equal to serial output, then first convert the serial output:
    srl.reads{i0,1}.val0 = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );

    % Convert to cell:
srl.reads{i0,1}.val = mat2cell( srl.reads{i0,1}.val0, ones( size( srl.reads{i0,1}.val0, 1 ), 1 ) ..., size( srl.reads{i0,1}.val0, 2 ) ... );

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

136


\texttt{\% Convert back to matrix: \texttt{\textit{unnecessary - cellfun converts to matrix already}}} \\
\texttt{\% \texttt{srl.read(i0,1).val = cell2mat( srl.read(i0,1).val ) ;}}

\texttt{\% Determine maximum and minimum values (axis information in plots)}
\texttt{srl.reads{i0,1}.val_min = min( srl.reads{i0,1}.val ) ;}
\texttt{srl.reads{i0,1}.val_max = max( srl.reads{i0,1}.val ) ;}
\texttt{srl.reads{i0,1}.val_absMax = max( abs( srl.reads{i0,1}.val_min}
\texttt{  srl.reads{i0,1}.val_max ) ) ) ;}

\texttt{end}

\texttt{\% Increment byte offset}
\texttt{srl.read{1,1}.i.byte0 = srl.read{1,1}.i.byte0 + srl.read{i0,1}.n.bytes ;}

\texttt{end}

\texttt{\% End}

\textbf{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

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

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

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

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

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

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

%% End
```

Code Listing A.21: [minseg.m]: Output - Save
A.1.5.2. Serial Reads Plot

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

%% [Output ]: Common plot commands

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

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

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

disp( 'Performing plot creation:' )

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

    p = p + 1;
    figure(p)

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

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

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

    % y-axis limits
    if isa(srl.reads{i0,1}.val , 'float') % float
srl.reads{i0,1}.ymin = -rndOut(srl.reads{i0,1}.val_absMax+eps, 2);
srl.reads{i0,1}.ymax = +rndOut(srl.reads{i0,1}.val_absMax+eps, 2);
else
    % integer
    srl.reads{i0,1}.ymin = double(intmin(class(srl.reads{i0,1}.val)));
    srl.reads{i0,1}.ymax = double(intmax(class(srl.reads{i0,1}.val)));
end

ylim([srl.reads{i0,1}.ymin, srl.reads{i0,1}.ymax])
grid minor

disp( 'Plot creation complete.' )
disp( ' ' )

% [Output ]: Close legacy figures
% not yet implemented.
% use "a = get(groot, 'Children')" to list all figures.
% then "for all figures: if n.figure > n.figure.gcf, close n.figure"
% when implemented, stop using "close all"

% End
A.1.6. Global Cleanup

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

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

Simulink.fileGenControl('reset')

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

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

%% End
```

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


