Modeling and Control of a Longitudinal Platoon of Ground Robotic Vehicles

by

Zhichao Li

A Thesis Presented in Partial Fulfillment of the Requirements for the Degree Master of Science

Approved August 2016 by the Graduate Supervisory Committee:

Armando A. Rodriguez, Chair
Panagiotis K. Artemiadis
Spring Melody Berman

ARIZONA STATE UNIVERSITY
December 2016
ABSTRACT

Toward the ambitious long-term goal of a fleet of cooperating Flexible Autonomous Machines operating in an uncertain Environment (FAME), this thesis addresses several critical modeling, design and control objectives for ground vehicles. One central objective is formation of multi-robot systems, particularly, longitudinal control of platoon of ground vehicle. In this thesis, the author use low-cost ground robot platform shows that with leader information, the platoon controller can have better performance than one without it.

Based on measurement from multiple vehicles, motor-wheel system dynamic model considering gearbox transmission has been developed. Noticing the difference between on ground vehicle behavior and off-ground vehicle behavior, on ground vehicle-motor model considering friction and battery internal resistance has been put forward and experimentally validated by multiple same type of vehicles. Then simplified longitudinal platoon model based on on-ground test were used as basis for platoon controller design.

Hardware and software has been updated to facilitate the goal of control a platoon of ground vehicles. Based on previous work of Lin on low-cost differential-drive (DD) RC vehicles called Thunder Tumbler, new robot platform named Enhanced Thunder Tumbler (ETT 2) has been developed with following improvement: (1) optical wheel-encoder which has 2.5 times higher resolution than magnetic based one, (2) BNO055 IMU can read out orientation directly that LSM9DS0 IMU could not, (3) TL-WN722N Wi-Fi USB Adapter with external antenna which can support more stable communication compared to Edimax adapter, (4) duplex serial communication between Pi and Arduino than single direction communication from Pi to Arduino, (5) inter-vehicle communication based on UDP protocol.

All demonstrations presented using ETT vehicles. The following summarizes key
hardware demonstrations: (1) cruise-control along line, (2) longitudinal platoon control based on local information (ultrasonic sensor) without inter-vehicle communication, (3) longitudinal platoon control based on local information (ultrasonic sensor) and leader information (speed). Hardware data/video is compared with, and corroborated by, model-based simulations. Platoon simulation and hardware data reveals that with necessary information from platoon leader, the control effort will be reduced and space deviation be diminished among propagation along the fleet of vehicles. In short, many capabilities that are critical for reaching the longer-term FAME goal are demonstrated.
To my parents.
ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my MS thesis advisor Professor Armando Antonio Rodriguez for his continuous support of my MS studies and research, for his patience, motivation, enthusiasm, and immense knowledge. His encouragement, insightful comments and guidance helped me throughout the course of the research and writing of this thesis. I would also like to thank Zhenyu Lin for his contribution on building low-cost differential drive robot platform, which provided a good start for my research. I take this opportunity to express my gratitude to all of the ME and EE faculty members for their help and support. I also thank my parents for their encouragement, support and attention. I am also grateful for lab colleagues especially Duo Lv, Jesus Aldaco, Venkatraman Renganathan, Xianglong Lu, and Nikhilesh Ravishankar. They help me throughout this tough task, their help has been greatly appreciated.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>viii</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ix</td>
</tr>
</tbody>
</table>

## CHAPTER

1 INTRODUCTION AND OVERVIEW OF WORK ........................................... 1
   1.1 Introduction and Motivation ............................................. 1
   1.2 Literature Survey: Robotics - State of the Field .................... 3
   1.3 Contributions of Work: Questions to be Addressed .................... 11
   1.4 Organization of Thesis .................................................. 22
   1.5 Summary and Conclusions .................................................. 24

2 OVERVIEW OF GENERAL FAME ARCHITECTURE ................................. 25
   2.1 Introduction and Overview .............................................. 25
   2.2 FAME Architecture ...................................................... 25
   2.3 Summary and Conclusions ................................................ 29

3 MODELING FOR SINGLE VEHICLE ................................................. 30
   3.1 Introduction and Overview .............................................. 30
   3.2 Description of Hardware ................................................ 30
   3.3 Modeling of a Differential-Drive Ground Robotic Vehicle .......... 36
   3.4 Differential-Drive Robot Kinematics ................................... 38
   3.5 Differential-Drive Robot Dynamics ...................................... 40
     3.5.1 DC Motor (Actuator) Dynamics with Gearbox ....................... 42
     3.5.2 Empirically Obtained Experimental Data for Motor-Wheel System .................................................. 44
     3.5.3 Robot TITO LTI Model with Actuator Dynamics .................... 59
   3.6 Uncertainty of Parameter ............................................... 76
### 3.6.1 Frequency Response Trade Studies

### 3.6.2 Time Response Trade Studies

### 3.7 Differential-Drive Robot Model with Dynamics on Ground

#### 3.7.1 Empirically Obtained Experimental Data for Ground Model

#### 3.7.2 Fitting Model to Collected Data

#### 3.7.3 On Ground Nominal Model

### 3.8 Summary and Conclusion

### 4 SINGLE VEHICLE CASE STUDY FOR A LOW-COST MULTI-CAPABILITY DIFFERENTIAL-DRIVE ROBOT: ENHANCED THUNDER TUMBLE (ETT)

#### 4.1 Introduction and Overview

#### 4.2 Inner-Loop Speed Control Design and Implementation

##### 4.2.1 Frequency Domain \((g,z)\) Trade Studies

##### 4.2.2 Time Domain \((g,z)\) Trade Studies

##### 4.2.3 Inner-Loop Experimental Result

#### 4.3 Outer-Loop Control Design and Implementation

##### 4.3.1 Outer-Loop 1: \((v, \theta)\) Cruise Control Along Line - Design and Implementation

##### 4.3.2 Outer-Loop 2: Separation-Direction \((\Delta x, \theta)\) Control - Design and Implementation

#### 4.4 Summary and Conclusion

### 5 LONGITUDINAL CONTROL OF A PLATOON OF VEHICLES

#### 5.1 Introduction and Overview

#### 5.2 Platoon Configuration
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Modeling for Longitudinal Platoon of Vehicles</td>
<td>153</td>
</tr>
<tr>
<td>5.4 Control for Longitudinal Platoon of Vehicles</td>
<td>154</td>
</tr>
<tr>
<td>5.5 Longitudinal Platoon Separation Controller Tradeoff Study</td>
<td>156</td>
</tr>
<tr>
<td>5.6 Experimental Results for Controlled Platoon of Vehicles</td>
<td>172</td>
</tr>
<tr>
<td>5.7 Platoon Simulation with model Uncertainty and Stiction Deadzone</td>
<td>179</td>
</tr>
<tr>
<td>5.8 Summary and Conclusions</td>
<td>180</td>
</tr>
<tr>
<td>6 SUMMARY AND FUTURE DIRECTIONS</td>
<td>181</td>
</tr>
<tr>
<td>6.1 Summary of Work</td>
<td>181</td>
</tr>
<tr>
<td>6.2 Directions for Future Research</td>
<td>183</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>185</td>
</tr>
<tr>
<td>APPENDIX</td>
<td></td>
</tr>
<tr>
<td>A C CODE</td>
<td>190</td>
</tr>
<tr>
<td>B MATLAB CODE</td>
<td>264</td>
</tr>
<tr>
<td>C ARDUINO CODE</td>
<td>294</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.1</td>
<td>Hardware Components for Enhanced Differential-Drive Thunder Tumbler Robotic</td>
</tr>
<tr>
<td></td>
<td>Vehicle</td>
</tr>
<tr>
<td>3.2</td>
<td>Thunder Tumbler Nominal Parameter Values with Uncertainty</td>
</tr>
<tr>
<td>3.3</td>
<td>Thunder Tumbler Gearbox Symbols</td>
</tr>
<tr>
<td>5.1</td>
<td>Platoon Design Parameter</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Visualization of Fully-Loaded (Enhanced) Thunder Tumbler</td>
<td>12</td>
</tr>
<tr>
<td>1.2 Optical Wheel Encoders - RPR220 photo-interrupter Sensors on Left, code disk on Right</td>
<td>13</td>
</tr>
<tr>
<td>1.3 Adafruit BNO055 9DOF Inertial Measurement Unit (IMU)</td>
<td>13</td>
</tr>
<tr>
<td>1.4 Arduino Uno Open-Source Microcontroller Development Board</td>
<td>14</td>
</tr>
<tr>
<td>1.5 Adafruit Motor Shield for Arduino v2.3 - Provides PWM Signal to DC Motors</td>
<td>14</td>
</tr>
<tr>
<td>1.6 Raspberry Pi 3 Model B Open-Source Single Board Computer</td>
<td>15</td>
</tr>
<tr>
<td>1.7 Raspberry Pi 5MP Camera Module</td>
<td>15</td>
</tr>
<tr>
<td>1.8 TP-LINK Wireless High Gain USB Adapter</td>
<td>16</td>
</tr>
<tr>
<td>1.9 HC-SR04 Ultrasonic Sensor</td>
<td>16</td>
</tr>
<tr>
<td>1.10 Visualization of Inner- and Outer-Loop Control Laws</td>
<td>19</td>
</tr>
<tr>
<td>2.1 <em>FAME</em> Architecture to Accommodate Fleet of Cooperating Vehicles</td>
<td>26</td>
</tr>
<tr>
<td>3.1 Visualization of Differential-Drive Mobile Robot</td>
<td>38</td>
</tr>
<tr>
<td>3.2 Visualization of DC Motor Speed-Voltage Dynamics</td>
<td>42</td>
</tr>
<tr>
<td>3.3 dc Motor Armature Inductance of 10 ETT Motors</td>
<td>45</td>
</tr>
<tr>
<td>3.4 dc Motor Armature Inductance of 10 ETT Motors</td>
<td>46</td>
</tr>
<tr>
<td>3.5 Disassembled Gearbox in ETT</td>
<td>48</td>
</tr>
<tr>
<td>3.6 DC Motor back EMF Constant of Left and Right ETT Motors</td>
<td>49</td>
</tr>
<tr>
<td>3.7 DC Motor Output $\omega_s$ Response to 1.02V Step Input - Hardware</td>
<td>50</td>
</tr>
<tr>
<td>3.8 DC Gain Distribution of Step Response at Different Voltage Step Input</td>
<td>51</td>
</tr>
<tr>
<td>3.9 DC Gain Distribution of Step Response at Different Voltage Step Input</td>
<td>52</td>
</tr>
<tr>
<td>3.10 Equivalent Moment of Inertia Distribution of Left and Right Motors</td>
<td>53</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3.11 Equivalent Speed Damping Constant Distribution of Left and Right Motors</td>
<td>53</td>
</tr>
<tr>
<td>3.12 Motor Output $\omega_s$ Response to 1.02V Step Input - Hardware and Decoupled Model</td>
<td>55</td>
</tr>
<tr>
<td>3.13 Motor Output $\omega_s$ Response to 2.04V Step Input - Hardware and Decoupled Model</td>
<td>55</td>
</tr>
<tr>
<td>3.14 Motor Output $\omega_s$ Response to 3.06V Step Input - Hardware and Decoupled Model</td>
<td>56</td>
</tr>
<tr>
<td>3.15 Motor Output $\omega_s$ Response to 4.08V Step Input - Hardware and Decoupled Model</td>
<td>56</td>
</tr>
<tr>
<td>3.16 Motor Output $\omega_s$ Response to 5.10V Step Input - Hardware and Decoupled Model</td>
<td>57</td>
</tr>
<tr>
<td>3.17 DC Motor Output $\omega_s$ Response to High Voltage Step Input - Hardware and Decoupled Model</td>
<td>58</td>
</tr>
<tr>
<td>3.18 TITO LTI Robot-Motor Wheel Speed ($\omega_R,\omega_L$) Dynamics - $P(\omega_R,\omega_L)$</td>
<td>59</td>
</tr>
<tr>
<td>3.19 Differential-Drive Mobile Robot Dynamics</td>
<td>60</td>
</tr>
<tr>
<td>3.20 Robot Singular Values (Voltages to Wheel Speeds) - Including Low Frequency Approximation</td>
<td>64</td>
</tr>
<tr>
<td>3.21 Robot Frequency Response (Voltages to Wheel Speeds) - Including Low Frequency Approximation</td>
<td>65</td>
</tr>
<tr>
<td>3.22 Robot Plant Singular Values (Voltages to $v$ and $\omega$) - Including Low Frequency Approximation</td>
<td>68</td>
</tr>
<tr>
<td>3.23 Robot Plant Frequency Response (Voltages to $[v,\omega]$) - Including Low Frequency Approximation</td>
<td>69</td>
</tr>
</tbody>
</table>
3.24 Frequency Response for Vehicle-Motor - Coupled ($\omega_R, \omega_L$) Model .... 70
3.25 Vehicle-Motor Response to Unit Step Input - Coupled ($\omega_R, \omega_L$) Model . 70
3.26 Bode Magnitude for Robot (Voltages to Wheel Speeds) - Mass Variations .......................................................... 76
3.27 Bode Magnitude for Robot (Voltages to Wheel Speeds) - $I$ Variations . 77
3.28 Bode Magnitude for Robot (Voltages to Wheel Speeds) - $K_b$ Variations 77
3.29 Bode Magnitude for Robot (Voltages to Wheel Speeds) - $K_t$ Variations 78
3.30 Bode Magnitude for Robot (Voltages to Wheel Speeds) - $R_a$ Variations 79
3.31 System Wheel Angular Velocity Responses to Step Voltages - Mass Variations ......................................................... 79
3.32 System Wheel Angular Velocity Responses to Step Voltages - Moment of Inertia Variations ........................................... 80
3.33 System Wheel Angular Velocity Responses to Step Voltages - back EMF Constant Variations ............................................ 81
3.34 System Wheel Angular Velocity Responses to Step Voltages - Torque Constant Variations .................................................. 82
3.35 System Wheel Angular Velocity Responses to Step Voltages - Armature Resistance Variations .......................................... 83
3.36 Target Characteristics of TB6612FNG ........................................ 84
3.37 Off Ground and On Ground Step Response to 50 PWM Command (1V) 85
3.38 Output $\omega_s$ Response to PWM 50 Voltage Step Input - Hardware and Decoupled Model .................................................. 86
3.39 On Ground Motor Fitting Left Motor - Hardware and Decoupled Model 87
<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.40 On Ground Motor Fitting Right Motor - Hardware and Decoupled Model</td>
<td>87</td>
</tr>
<tr>
<td>3.41 On Ground Approximate Transfer Function Fitting Model Distribution of V1</td>
<td>88</td>
</tr>
<tr>
<td>3.42 On Ground Nominal Model</td>
<td>89</td>
</tr>
<tr>
<td>3.43 Step Response of Nominal Model with other ETT Model</td>
<td>90</td>
</tr>
<tr>
<td>3.44 On Ground Nominal Model with other ETT Model</td>
<td>90</td>
</tr>
<tr>
<td>4.1 Visualization of ( (v, \omega) ) and ( (\omega_R, \omega_L) ) Inner-Loop Control</td>
<td>95</td>
</tr>
<tr>
<td>4.2 ( PK ) and ( L_o = MPKM^{-1} ) Singular Values</td>
<td>96</td>
</tr>
<tr>
<td>4.3 ( S_o = (I + L_o)^{-1} = S_i ) Singular Values - Using Decoupled Model</td>
<td>98</td>
</tr>
<tr>
<td>4.4 ( T_o = L_o(I + L_o)^{-1} = T_i ) Singular Values - Using Decoupled Model</td>
<td>98</td>
</tr>
<tr>
<td>4.5 ( T_{ru} ) Singular Values (No Pre-filter) - Using Decoupled Model</td>
<td>99</td>
</tr>
<tr>
<td>4.6 ( T_{ru}W ) Singular Values (with Pre-filter) - Using Decoupled Model</td>
<td>99</td>
</tr>
<tr>
<td>4.7 ( KSM^{-1} ) Singular Values</td>
<td>100</td>
</tr>
<tr>
<td>4.8 ( T_{diy} ) Singular Values - Using Decoupled Model</td>
<td>100</td>
</tr>
<tr>
<td>4.9 ( MSP ) Singular Values</td>
<td>101</td>
</tr>
<tr>
<td>4.10 Inner-Loop ( [\omega_R, \omega_L] ) Filtered Step Response - Using Decoupled Model</td>
<td>101</td>
</tr>
<tr>
<td>4.11 Inner-Loop ( [\omega_R, \omega_L] ) Unfiltered Step Response - Using Decoupled Model</td>
<td>102</td>
</tr>
<tr>
<td>4.12 Control Response to Step Command (with Pre-filter) - with Decoupled Model</td>
<td>103</td>
</tr>
<tr>
<td>4.13 Control Response to Unfiltered Step Command - with Decoupled Model</td>
<td>103</td>
</tr>
<tr>
<td>4.14 Singular Values for ( L ) (( g=0.10, 0.30, 0.50, 0.70; z=2 ))</td>
<td>104</td>
</tr>
<tr>
<td>4.15 Singular Values for ( L ) (( g=0.50; z=1, 2, 3, 4 ))</td>
<td>105</td>
</tr>
<tr>
<td>4.16 Singular Values for ( S ) (( g=0.10, 0.30, 0.50, 0.70; z=2 ))</td>
<td>106</td>
</tr>
</tbody>
</table>
4.17 Singular Values for $S$ (g=0.50; z=1, 2, 3, 4) ........................................... 106
4.18 Singular Values for $T$ (g=0.10, 0.30, 0.50, 0.70; z=2) .............................. 107
4.19 Singular Values for $T$ (g=0.50; z=1, 2, 3, 4).................................................. 108
4.20 Singular Values for $T_{ru}$ (g=0.10, 0.30, 0.50, 0.70; z=2) - $(\omega_R, \omega_L)$ Commands .......................................................... 109
4.21 Singular Values for $T_{ru}$ (g=0.50; z=1, 2, 3, 4)- $(\omega_R, \omega_L)$ Commands .... 109
4.22 Singular Values for $KSM^{-1}$ (g=0.10, 0.30, 0.50, 0.70; z=2) - $(v, \omega)$
    Commands .......................................................... 110
4.23 Singular Values for $KSM^{-1}$ (g=0.50; z=1, 2, 3, 4)- $(v, \omega)$ Commands . 111
4.24 Singular Values for $W \cdot T_{ru}$ (g=0.10, 0.30, 0.50, 0.70; z=2) - $(\omega_R, \omega_L)$
    Commands .......................................................... 112
4.25 Singular Values for $W \cdot T_{ru}$ (g=0.50; z=1, 2, 3, 4) - $(\omega_R, \omega_L)$ Commands
4.26 Singular Values for $T_{div}$ (g=0.10, 0.30, 0.50, 0.70; z=2) - $(\omega_R, \omega_L)$
    Responses .......................................................... 114
4.27 Singular Values for $T_{div}$ (g=0.50; z=1, 2, 3, 4) - $(\omega_R, \omega_L)$ Responses ... 114
4.28 Singular Values for $MSP$ (g=0.50; z=1, 2, 3, 4) - $(v, \omega)$ Responses ..... 115
4.29 Singular Values for $MSP$ (g=0.50; z=1, 2, 3, 4) - $(v, \omega)$ Responses ..... 116
4.30 Output Response to Step Command (g = 0.10, 0.30, 0.50, 0.70; z = 2) ........ 117
4.31 Output Response to Step Command (g = 0.50; z = 1, 2, 3, 4) ............ 118
4.32 Control Response to Step Command (g = 0.10, 0.30, 0.50, 0.70; z = 2) .... 118
4.33 Control Response to Step Command (g = 0.50; z = 1, 2, 3, 4) ............. 119
4.34 Output Response to Step Command (g = 0.10, 0.30, 0.50, 0.70; z = 2) .......... 120
4.35 Output Response to Step Command (g = 0.50; z = 1, 2, 3, 4) ........... 120
<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.36</td>
<td>121</td>
</tr>
<tr>
<td>Control Response to Filtered Step Command $g = 0.10, 0.30, 0.50, 0.70$; $z = 2$</td>
<td>121</td>
</tr>
<tr>
<td>4.37</td>
<td>121</td>
</tr>
<tr>
<td>Control Response to Filtered Step Command ($g = 0.50$; $z = 1, 2, 3, 4$)</td>
<td>121</td>
</tr>
<tr>
<td>4.38</td>
<td>122</td>
</tr>
<tr>
<td>Output Response to filtered Step Command ($\omega_{Rref} = 10$, $\omega_{Lref} = 10$)</td>
<td>122</td>
</tr>
<tr>
<td>4.39</td>
<td>123</td>
</tr>
<tr>
<td>Output Response to filtered Step Command ($v_{ref} = 0.5$, $\omega_{ref} = 0$)</td>
<td>123</td>
</tr>
<tr>
<td>4.40</td>
<td>123</td>
</tr>
<tr>
<td>Control Response to filtered Step Command ($\omega_{Rref} = 10$, $\omega_{Lref} = 10$)</td>
<td>123</td>
</tr>
<tr>
<td>4.41</td>
<td>125</td>
</tr>
<tr>
<td>Visualization of Cruise Control Along a Line</td>
<td>125</td>
</tr>
<tr>
<td>4.42</td>
<td>126</td>
</tr>
<tr>
<td>$T_{\theta_{ref}}$ Frequency Response for $\theta$ Outer-Loop (P Control)</td>
<td>126</td>
</tr>
<tr>
<td>4.43</td>
<td>126</td>
</tr>
<tr>
<td>$T_{\theta_{ref}}$ Frequency Response for $\theta$ Outer-Loop (PD control, $K_d = 1$)</td>
<td>126</td>
</tr>
<tr>
<td>4.44</td>
<td>127</td>
</tr>
<tr>
<td>Cruise Control $\theta$ Response Using P Control ($\theta_o = 0.1$ rad)</td>
<td>127</td>
</tr>
<tr>
<td>4.45</td>
<td>127</td>
</tr>
<tr>
<td>Cruise Control $\theta$ Response Using PD Control ($\theta_o = 0.1$ rad)</td>
<td>127</td>
</tr>
<tr>
<td>4.46</td>
<td>128</td>
</tr>
<tr>
<td>Visualization of $(\Delta x, \theta)$ Separation-Direction Control System</td>
<td>128</td>
</tr>
<tr>
<td>4.47</td>
<td>129</td>
</tr>
<tr>
<td>Vehicle Separation Control (Proportional Control: $K_p = 0.5, 1.0, 1.5, 2.0$; $\Delta x_o = 1$)</td>
<td>129</td>
</tr>
<tr>
<td>4.48</td>
<td>130</td>
</tr>
<tr>
<td>Vehicle Separation Control (PD control: $K_p = 0.5, 1.0, 1.5, 2.0$; $K_d=1$; $\Delta x_o = 1$)</td>
<td>130</td>
</tr>
<tr>
<td>4.49</td>
<td>135</td>
</tr>
<tr>
<td>Minimizing Counter Value Versus Desired $f = \frac{x}{x_{resolution}}$</td>
<td>135</td>
</tr>
<tr>
<td>4.50</td>
<td>136</td>
</tr>
<tr>
<td>Minimum Percent Error Versus Desired $f = \frac{x}{x_{resolution}}$</td>
<td>136</td>
</tr>
<tr>
<td>4.51</td>
<td>144</td>
</tr>
<tr>
<td>Cruise Control Along a Line</td>
<td>144</td>
</tr>
<tr>
<td>4.52</td>
<td>147</td>
</tr>
<tr>
<td>Vehicle Separation Convergence Using Proportional Control ($K_p = 1$; $\Delta x(0) \approx 1$)</td>
<td>147</td>
</tr>
<tr>
<td>4.53</td>
<td>147</td>
</tr>
<tr>
<td>Vehicle Separation Convergence Using Proportional Control ($K_p = 1$; $\Delta x(0) \approx 1$)</td>
<td>147</td>
</tr>
</tbody>
</table>
4.54 Vehicle Separation Convergence Using PD Control \((K_p = 1.5; K_d = 1; \Delta x(0) \approx 1)\) ................. \(148\)

5.1 Platoon of 4 Vehicles .............................................. \(152\)
5.2 Leader Model of Platoon ............................................. \(153\)
5.3 i-th Vehicle Model of Platoon ................................. \(153\)
5.4 Visualization of i-th Follower in Platoon Separation Control System .... \(154\)
5.5 Visualization of Platoon Controller ........................ \(155\)
5.6 Simulation of Vehicle Separation Control of Platoon (Proportional Control for \(\Delta x (K_p = 0.5)\)) ........................................ \(156\)
5.7 Simulation of Control Response of Platoon (Proportional Control for \(\Delta x (K_p = 0.5)\)) ........................................ \(157\)
5.8 Simulation of Vehicle Separation Control of Platoon (Proportional Control for \(\Delta x (K_p = 1.0)\)) ........................................ \(157\)
5.9 Simulation of Control Response of Platoon (Proportional Control for \(\Delta x (K_p = 1.0)\)) ........................................ \(158\)
5.10 Simulation of Vehicle Separation Control of Platoon (Proportional Control for \(\Delta x (K_p = 2.0)\)) ........................................ \(158\)
5.11 Simulation of Control Response of Platoon (Proportional Control for \(\Delta x (K_p = 2.0)\)) ........................................ \(159\)
5.12 Simulation of Vehicle Separation Control of Platoon (PD Control for \(\Delta x (g = 0.2)\)) ........................................ \(160\)
5.13 Simulation of Control Response of Platoon (PD Control for \(\Delta x (g = 0.2)\)) \(160\)
5.14 Simulation of Vehicle Separation Control of Platoon (PD Control for \(\Delta x (g = 0.5)\)) ........................................ \(161\)
5.15 Simulation of Control Response of Platoon (PD Control for $\Delta_x (g = 0.5)$) 161
5.16 Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 1.0)$) ................................................................. 163
5.17 Simulation of Control Response of Platoon (PID Control for $\Delta_x (g = 1.0)$) ................................................................. 164
5.18 Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 2.0)$) ................................................................. 164
5.19 Simulation of Control Response of Platoon (PID Control for $\Delta_x (g = 2.0)$) ................................................................. 165
5.20 Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and $k_{pff} = 0.5$ for FF-path) ................. 166
5.21 Simulation of Control Response of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and $k_{pff} = 0.5$ for FF-path) .......................... 167
5.22 Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and $k_{pff} = 1.5$ for FF-path) ...................... 167
5.23 Simulation of Control Response of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and $k_{pff} = 1.5$ for FF-path) .......................... 168
5.24 Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and PI Control for FF-path ($g_{ff} = 0.5$, $z_{ff} = 1.0$)) 169
5.25 Simulation of Control Response of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and PI Control for FF-path ($g_{ff} = 0.5$, $z_{ff} = 1.0$)) ...... 170
5.26 Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and PI Control for FF-path ($g_{ff} = 0.5$, $z_{ff} = 2.0$)) 170
5.27 Simulation of Control Response of Platoon (PID Control for $\Delta x$ ($g = 1.0, z = 1.0$) and PI Control for FF-path ($g_{ff} = 0.5, z_{ff} = 2.0$)) ........... 171
5.28 Simulation of Vehicle Separation Control of Platoon (Proportional Control for $\Delta x$ ($K_p = 1$)) ........................................ 173
5.29 Simulation of Control Response of Platoon (Proportional Control for $\Delta x$ ($K_p = 1$)) ........................................ 173
5.30 Experimental Separation Response of Platoon Proportional Control for $\Delta x$ ($K_p = 1$) ........................................ 174
5.31 Simulation of Vehicle Separation Control of Platoon (PD Control for $\Delta x$ ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$)) ........................................ 175
5.32 Simulation of Control Response of Platoon (PD Control for $\Delta x$ ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$)) ........... 175
5.33 Experimental Separation Response of Platoon (PD Control for $\Delta x$ ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$)) 176
5.34 Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta x$ ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$)) ........................................ 177
5.35 Simulation of Control Response of Platoon (PID Control for $\Delta x$ ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$)) ........................................ 177
5.36 Experimental Separation Response of Platoon (PID Control for $\Delta x$ ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$)) ........................................ 178
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.37</td>
<td>Simulation of Separation Response of Platoon with model uncertainty and stiction</td>
<td>179</td>
</tr>
<tr>
<td>5.38</td>
<td>Simulation of Control Response of Platoon with model uncertainty and stiction</td>
<td>180</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION AND OVERVIEW OF WORK

1.1 Introduction and Motivation

With the rapid growth of population in the world, severe congestion and pollution happens every day in some of the worlds most populated cities (e.g. Beijing, Tokyo, and New Delhi). More efficient, cost-effective, and clean ground transportation system is desperately needed. So self-driving technology and electric vehicle draw more and more attention in recent years. In May 2012, Google’s autonomous car passed the world’s first self-driving test in Las Vegas which sparked research on intelligent transportation system (ITS) again. More and more automotive companies are shifting their focus towards electric vehicle market, like Telsa Motors, BMW, etc. The seminal intelligent vehicle and highway systems (IVHS) work of S.E. Sheikholeslam [3], [6] demonstrated that a longitudinal platoon of cars can be tightly controlled in both speed and spacing in order to promote more effective traffic flow. In this thesis, the author will reconsider the vehicle platoon modeling, design and control problem, providing simulation and hardware result using low-cost multi-capability electric ground vehicles.

In previous MS thesis work of Lin [14], off-the-shelf technologies (e.g. Arduino, Raspberry Pi, commercially available RC cars) have been exploited to develop low-cost ground vehicles that can be used for multi-vehicle robotics research. The first step toward the longer-term goal of achieving a fleet of Flexible Autonomous Machines operating in an uncertain Environment (FAME) has been done. Such a fleet can involve multiple ground and air vehicles that work collaboratively to accomplish
coordinated tasks. Such a fleet may be called a swarm [43]. Potential applications can include: remote sensing, mapping, intelligence gathering, intelligence-surveillance-reconnaissance (ISR), search, rescue and much more. It is this vast application arena as well as the ongoing accelerating technological revolution that continues to fuel robotic vehicle research.

This thesis continues to address the modeling, design and control issues associated with the coordination of multiple ground-based robotic vehicles. Same Framework of [14] is used for consistency toward the same longer-term FAME goal. Central objective of the thesis was to show how to control multiple robots in a certain formation, particularly how to control a platoon of vehicle cruise in a straight line and keep constant separation distance. This problem has been called longitudinal control of vehicle platoon. This is shown for differential-drive vehicle class. Multiple Enhanced Thunder Tumbler (ETT) vehicles were used in this research, both kinematic and dynamical models are examined. Here, differential-drive means that the speed of each of the rear wheels are controlled independently by separate DC motors. This vehicle class is non-holonomic; i.e. the two (2) \((x, y)\) or \((v, \theta)\) controllable degrees of freedom is less than the three (3) total \((x, y, \theta)\) degrees of freedom.

This fundamentally limits the ability of a single continuous (non-switching) control law to “precisely park the vehicle” (see discussions below based on work of [61], [63], [25]). Despite this, it is shown how continuous linear control theory can be used to develop suitable control laws that are essential for achieving various critical capabilities (e.g. speed/position control along a line/path, spacing control) [14]. Following is a comprehensive literature survey - one that summarizes relevant literature and how it has been used.
1.2 Literature Survey: Robotics - State of the Field

In an effort to shed light on the state of ground robotic vehicle modeling, hardware, design, and control, the following topically organized literature survey is offered. An effort is made below to highlight what technical papers/works are most relevant to this thesis. In short, the following works are most relevant for the developments within this thesis:

- low-cost ground robotic modeling, design and control work within [14];
- DC motor modeling work within [16] (addressing DC motor modeling with gearbox and limitation of linear model), [46] (addressing modeling and identification of DC motor with nonlinearity);
- non-holonomic differential-drive vehicle modeling and control work within [15] (addressing dynamic two-input two-output LTI model for differential-drive vehicles), [59] (addressing control of differential-drive vehicles);
- vision-based line/curve following work within [24];
- vehicle separation modeling and longitudinal platoon control work within [3], [6] (presenting vehicle separation control laws);

An attempt is made below to provide relevant insightful technical details.

- **Differential-Drive Robot Modeling.** Within this thesis, differential-drive (Thunder Tumbler) ground vehicles represent a central focus of the work. Here, differential-drive means that there are two rear wheels - each with an independent torque generating armature controlled DC motor on it [52]. As such, these DC motors can be used to independently control the speed of the rear wheels. Nominally, we assume that the motors are identical. The motor inputs (vehicle
controls) are voltages. The sum of these voltages is used to control the vehicle’s speed \( v \). The difference is used to control the direction \( \theta \) of the vehicle.

- **Kinematic Model.** A kinematic model for differential-drive robot (ignoring dynamic mass-inertia effects) is presented within [19], [18]. Within this kinematic model, it is assumed that the translational and angular velocities \((v, \omega)\) of the robot are realized instantaneously. This, of course, is not realistic because of real-world actuator (e.g. motor) limitations and mass-inertia constraints. From Newton’s second law of motion, we know that an instantaneously achieved velocity generally requires infinite acceleration and force. The kinematic model is therefore less accurate than a dynamical model (i.e. one which includes acceleration constraining mass-inertia effects).

- **Dynamical Model.** A dynamical model can take the torques applied to the robot wheels as inputs (controls) to the system. This is done within [21], [23]. The model presented within these works incorporates dynamic (acceleration constraining) mass-inertia effects as well as friction, wheel slippage etc. Given this, it is apparent that a dynamic model generally gives a much more accurate model of the vehicle. Within [15], a two-input two-output (TITO) linear time invariant (LTI) model - including DC motor dynamics as well as vehicle mass-inertia effects - is presented for a differential-drive ground vehicle. The model describes the TITO LTI map from the two DC motor input voltages (vehicle controls) to the two rear wheel angular velocities \((\omega_R, \omega_L)\). The map from the voltages to the vehicle longitudinal and angular speeds \((v, \omega)\) is also a TITO LTI transfer function matrix. This
model was exploited within [59] for control design. This TITO LTI model, and its diagonal approximation, shall be used as the main differential-drive vehicle model within this thesis (e.g. see work within Chapters 3 and 4). It will be used to understand the robot’s linear (voltage to wheel angular velocity or voltages to speed and angular velocity) dynamics as well as to develop linear inner-loop \((\omega_R, \omega_L)\) and \((v, \omega)\) speed control laws. It is very important to note that the vehicle model becomes nonlinear when one considers the planar \((x, y)\) coordinates of the vehicle.

Given the above, it should be noted that the map from the motor voltages to \((\omega_R, \omega_L)\) is a TITO LTI coupled model that is nearly decoupled (decentralized) at low frequencies; i.e. frequencies below \(\frac{\beta}{I}\), where \(\beta\) denotes motor shaft rotational speed damping and \(I\) denotes rotational moment of inertia. (This is not true for \((v, \omega)\).) It is this decoupling (and our non-aggressive moderate-bandwidth performance objectives) that permits us to use a decoupled (decentralized) model for control law development. This is discussed further within Chapters 3 and 4. For our differential-drive Thunder Tumble vehicle (Chapter 4), vehicle parameters for the fourth order TITO LTI model from motor voltages to \((v, \omega)\) or \((\omega_R, \omega_L)\) were estimated by iterating between experiments and model-based time simulations. Vehicle mass \(m\) was measured. It was assumed that the DC motors are identical. DC motor armature inductance \(L_a\) was neglected - thus making the model second order. Settling time, steady state speed and armature current were used to (approximately) solve for the three remaining model parameters: angular speed damping \(\beta\), back emf and torque constant \(K_b = K_t\), armature resistance \(R_a\). (Additional relevant details

5
are provided within Chapter 4). While the left-right motor model parameters were assumed to be identical, it should be noted that the feedback laws implemented implicitly compensate (to some extant) for real-world parametric uncertainty.

The above summarizes basic principles regarding differential-drive ground vehicles.

- **Classical Controls.** Classical control design fundamentals are addressed within the text [52]. Internal model principle ideas - critical for command following and disturbance attenuation - are presented within [50], [52]. General PID (proportional plus integral plus derivative) control theory, design and tuning are addressed within the text [32]. Fundamental performance limitations are discussed with [69],[52].

- **Robot Inner-Loop Control.** A proportional-plus-integral-plus-derivative (PID) inner-loop control design is addressed within [48], [49]. A PI controller is used for inner-loop control within [51], [59]. Within Chapter 3-4, we examine PI inner-loop speed ($\omega_R, \omega_L$) and $(v, \omega)$ control laws for our differential-drive vehicles. Inner-loop control law parameter trade studies are presented within Chapter 3. Classically-based decentralized [52] control [26] was examined in the frequency- and time-domains. It was used to select a decentralized inner-loop control law for implementation in the hardware. A centralized inner-loop control law may become essential when we have stringent high bandwidth constraints and large plant coupling. [59].

- **Robot Outer-Loop Control.** Within this thesis, various outer-loop control laws are examined. When relevant, existing work in the literature was exploited.
1. **Cruise Control Along a Line.** Within this thesis, it is important to note the difference between trajectory tracking and path following. Trajectory tracking addresses following $x(t), y(t)$ commands; i.e. $(x, y)$ commands with very specific temporal constraints [29]. Path following addresses following a path/curve in the plane (without temporal constraints)[29]. To address trajectory tracking and path following tasks, standard linear techniques are used within [27]. Nonlinear approaches are used within the following: feedback linearization within [28], Lyapunov-based techniques within [19], [20], [30], [31].

Cruise control is a fundamentally important feature for a ground robotic system. Within this thesis, we therefore develop an encoder-IMU-camera based (PD with roll off) outer-loop $(v, \theta)$ control law that permits cruise control along a camera visible line/path. The camera, here, resolves encoder-IMU dead reckoning issues. See work within Chapter 4. The cruise control law is based on the TITO LTI $(v, \omega)$ or $(\omega_R, \omega_L)$ inner-loop model presented within [15] and the associated inner-loop control law (e.g. see work within Chapters 3 and 4). The map from the reference commands $(v_{ref}, \omega_{ref})$ to the actual velocities $(v, \omega)$ looks like a simple diagonal system (e.g. $diag\left(\frac{a}{s+a}, \frac{b}{s+b}\right)$) at low frequencies - a consequence of a well-designed inner-loop control system. (See inner-loop work within Chapters 3 and 4; outer-loop work in Chapter 4). The outer-loop $\theta$ controller therefore “sees” $\frac{b}{s(s+b)}$. From classical root locus ideas [52], a proportional controller is therefore justified - provided that the gain is not too large. If the gain is too large, oscillations (or even limit cycle behavior) are expected in $\theta$. A PD controller with roll off would help with this issue. (See work within Chapter 4).
2. *Separation Control.* Within [3], [6], vehicle separation modeling and longitudinal platoon control is presented. The ideas presented within [3], [6] motivate the PID ultrasonics-encoder-IMU-based separation control laws used for the separation-direction ($\Delta x, \theta$) outer-loop control within this thesis. The ideas here are also used to have multiple differential-drive vehicles following an autonomous or remotely controlled leader vehicle. See work within Chapter 4. Future work will examine the related saturation prevention issues within [64]. Relevant outer-loop control law parameter trade studies are also presented within Chapter 4.

3. *Robot/Car Spacing Control.* Robot/car spacing control - *intelligent vehicles and highway systems (IVHS)* - is briefly addressed within [52]. A more comprehensive treatment of vehicle separation modeling and longitudinal platoon control is presented within [3], [6]. These works provide a theoretical foundation for the (inter- and multi-vehicle) separation control laws developed within this thesis. The ideas presented within [3], [6] specifically motivate the PID separation control laws used within this thesis. (See work within Chapter 4). Future work will examine the related saturation prevention issues within [64].

- **Actuators and Sensors.** Actuators and sensors are addressed within the [34].

- **DC Motors.** Simple armature controlled DC motor modeling concepts are addressed from a controls perspective within [52]. DC motor modeling for wheeled robot applications is addressed within [44]. In this paper, nonlinear effects are neglected. Nonlinear modeling and identification for DC motors is addressed within [45], [46]. Also, see detailed discussion presented above on the TITO LTI vehicle-motor model presented within[15]. This model will serve as the ba-
sis for inner-loop control law development for our differential-drive (DD) robots.

- **Encoders.** Rotary optical encoders are the most widely used encoder design. They consist of an LED light source, light detector, code disc, and signal processor [47]. Magnetic encoders consist of magnets and a hall effect sensor. They are inherently rugged and operate reliably under shock, vibration, and high temperature [47]. Within this thesis, we used home-made optical encoders rather than magnetic encoders in [14] on the wheels of our differential-drive Thunder Tumbler ground vehicles, more accurate measurement is obtained. These wheel encoders allow us to estimate right and left angular speed and displacement information. From this, we then can compute the vehicle’s translational speed $v$ and angular speed $\omega$. These are used to design our proportional plus integral (PI) $(\omega_R, \omega_L)$ or $(v, \omega)$ inner-loop control systems. (See work in Chapters 3 and 4). We will see in Chapter 4, how encoder improvement can benefit us and also the new optical wheel encoders used (20 Counts Per Turn (CPT)) will limit how well the inner-loop can perform. A static position error of $r_{\text{wheel}}(\frac{2\pi}{20}) \approx 0.0157$ m (where $r_{\text{wheel}} = 0.05$ m is the wheel radius), for example, can result. This error can build up as the robot stops and goes. It can also result in undesirable position control oscillations because the exact position cannot be achieved. While the oscillations can be corrected with some nonlinear control logic, the error cannot be corrected unless we have some dead-reckoning correction mechanism; e.g. camera, GPS, lidar. Within this thesis, a camera is used to address dead-reckoning errors.

- **Cameras.** Within this research, we make use of the Raspberry Pi camera (2592 \times 1944 pixel or 5 MP static images; 1080p30 (30 fps), 720p60 and 640\times480p60/90 MPEG-4 video). It connects directly to the Raspberry Pi 3’s
GPU (graphical processing unit). It is capable of 1080p full HD video. Because the camera is directly connected to the GPU, there is very little impact on the CPU (central processing unit). This makes the CPU available for other processing tasks [57]. Within this thesis, cameras are used for outer-loop control law implementation (e.g. \((v, \theta), (x, y), \Delta x, \theta\)) and as a tool for correcting the inevitable dead-reckoning errors associated with encoders and IMUs.

• **Vision Algorithms.** The line/curve image processing ideas within the text [24] are exploited within this thesis. Specifically, we use the Raspberry Pi 3 camera [57] information to obtain vehicle directional information. This information is used within the following outer-loop control laws: \((v, \theta)\) cruise control, planar \((x, y)\) Cartesian stabilization [25]. The vision algorithm used within this thesis is a color filtering algorithm [24]. This algorithm can filter out irrelevant colors (e.g. turn them into black) and select the desired color of interest (e.g. turn it into white). After applying this algorithm, the camera only sees the color of interest. This can be used to develop camera-based line/curve following [60] separation-direction and platoon control laws. Within this thesis, these ideas are used to follow a visible continuous black tape straight line on the ground. These purposely fixed references essentially provide a very inexpensive form of GPS. See work within Chapter 4.

• **Global Positioning System (GPS).** An overview of GPS is presented within [56]. Differential GPS (DGPS) techniques are also described within [56].

• **Arduino.** Within this thesis, we make great use of the Arduino Uno microcontroller board (16MHZ ATmega328 processor, 32KB Flash Memory, 14 digital I/O pins, 6 analog inputs, $25). More detailed specifications for the Arduino Uno board are presented within [36]. It is used to implement inner- and outer-
loop control laws for our differential-drive Thunder Tumbler vehicle.

- **Raspberry Pi 3.** Within this thesis, we make great use of the Raspberry Pi 3 computer board (1200 MHz quad-core ARM Cortex-A53 CPU (Pi 2: 900 MHz quad-core ARM Cortex-A7 CPU), 1GB SDRAM, 40 GPIO pins, camera interface, $40). Introductory and technical details for the Raspberry Pi 3 are discussed within [42]. Comparison between Pi 2 and Pi 3 can be found [66]. The Raspberry PI 3 is used to implement outer-loop \((v, \theta), (\Delta x, \theta)\), platoon without leader information, and platoon with leader information control laws within this thesis.

- **Commercially Available Ground Robotic Vehicles.** A comprehensive commercially available robot systems can be found in [14]. Robotic systems with capabilities and price comparison has been done for the following: Seekur ($70K), Pioneer 3DX ($7.5K), Boebot robot ($160), AAR robot ($79), Lego Mindstorm EV3 permits building of Track3r, R3ptar, Spik3r, Ev3rstorm, Gripp3r ($350); VEX Robotics Design System($500); APM 2.5 Arducopter.

1.3 Contributions of Work: Questions to be Addressed

Within this thesis, the following fundamental questions are addressed. When taken collectively, the answers offered below, and details within the thesis, represent a useful contribution to researchers in the field. Moreover, it must be emphasized that answers to these questions are critical in order to move substantively toward the longer-term FAME goal.

1. **How can off-the-shelf “toy” vehicles be suitably augmented to yield effective low-cost research platforms?** This question was one of the main objectives within [14], in which Thunder Tumbler ($10) toy vehicle was used and
converted to multi-capability ground robot platform. Based on hardware design of him, the author make improvements about hardware on following aspects: (1) optical wheel-encoders which 2.5 times accurate than magnetic based one and absolute orientation sensor BNO055 IMU can give out azimuth directly that LSM9DS0 can only get angular rate, (2) Raspberry Pi 3 with more computation power than pi 2, this would benefit when image processing is needed. (3) TL-WN722N Wi-Fi USB Adapter with external antenna which can support more stable communication compared to Edimax Wi-Fi adapter in longer distance. Other hardware setting are the same, just repeat here for completeness.

The updated fully-loaded (enhanced) Thunder Tumbler vehicle is shown in Figure 1.1.

![Figure 1.1: Visualization of Fully-Loaded (Enhanced) Thunder Tumbler](image-url)
Each differential-drive Thunder Tumbler vehicle was augmented to provide a suite of substantive capabilities. Each augmented (“enhanced” Thunder Tumbler) vehicle costs less than $220 but offers the capability of commercially available vehicles costing over $500.

(1) optical wheel encoders and IMU (RPR220 photo-interrupter see Figure 1.2) and (Adafruit BNO055 Absolute Orientation Sensor, see Figure 1.3) inertial measurement unit (IMU) to facilitate (dead-reckoning-based) inner-loop speed control as well as outer-loop position and directional control,

Figure 1.2: Optical Wheel Encoders - RPR220 photo-interrupter Sensors on Left, code disk on Right

Figure 1.3: Adafruit BNO055 9DOF Inertial Measurement Unit (IMU)
(2) an Arduino Uno open-source microcontroller development board (16MHz AT-mega328 processor, 32KB Flash Memory, 14 digital I/O pins, 6 analog inputs, $25, see Figure 1.4) for both encoder-IMU-based speed $(v, \omega)$ or $(\omega_R, \omega_L)$ inner-loop control and encoder-IMU-ultrasound-based directional-separation outer-loop control,

![Figure 1.4: Arduino Uno Open-Source Microcontroller Development Board](image)

(3) an Arduino motor shield (see Figure 1.5) for inner-loop motor speed control.

![Figure 1.5: Adafruit Motor Shield for Arduino v2.3 - Provides PWM Signal to DC Motors](image)

(4) a Raspberry Pi 3 Model B single board computer (1.2 GHz quad-core ARM Cortex-A53 CPU, 1GB SDRAM, 40 GPIO pins, camera interface, $40, see
Figure 1.6) for more demanding vision-based cruise-position-directional outer-loop control,

![Raspberry Pi 3 Model B Open-Source Single Board Computer](image)

Figure 1.6: Raspberry Pi 3 Model B Open-Source Single Board Computer

(5) a **Raspberry Pi 5MP camera** (2592 \times 1944 pixel or 5 MP static images; 1080p30 (30 fps), 720p60 and 640x480p60/90 MPEG-4 video, see Figure 1.7) for outer-loop cruise-position-directional control,

![Raspberry Pi 5MP Camera Module](image)

Figure 1.7: Raspberry Pi 5MP Camera Module
(6) a **TP-LINK TL-WN722N USB Wi-Fi Adapter** (150 Mbps wireless transmission rate with 4dBi omni-directional antenna, see Figure 1.8) for inter-vehicle communication.

![Figure 1.8: TP-LINK Wireless High Gain USB Adapter](image)

(7) a **forward-pointing (SR04) ultrasonic distance/range-finder sensor** (40kHz, 0.02-5 m, approximately ±8° directional, see Figure 1.9) for outer-loop separation control, and

![Figure 1.9: HC-SR04 Ultrasonic Sensor](image)
2. Why should a hierarchical inner-outer loop control architecture be used? Hierarchical inner-outer loop controllers are found across many industrial/commercial/military application areas (e.g. aircraft, spacecraft, robots, manufacturing processes, etc.) where it is natural for slower (outer-loop generated) high-level commands to be followed by a faster inner control loop that must deliver robust performance (e.g. low frequency reference command following, low-frequency disturbance attenuation and high-frequency sensor noise attenuation) in the presence of significant signal and system uncertainty. A well designed inner-loop can greatly simplify outer-loop design. An excellent example of how inner-outer loop architectures are used is in the missile-target application arena. Here, an autopilot (inner-loop) follows commands generated from the guidance system (outer-loop). More substantively, inner-outer loop control structures are used to tradeoff properties at distinct loop breaking points (e.g. outputs/errors versus inputs/controls) [53], [54].

**Inner-Loop Control**

3. **What are typical inner-loop objectives?** Typical inner-loop objectives can be speed control; i.e. requiring the design of a speed \((v, \omega)\) or \((\omega_R, \omega_L)\) control system. Within this thesis, inner-loop control for our differential-drive Thunder Tumbler vehicles specifically refers to classical proportional-plus-integral (PI) pulse-width-modulation (PWM) based speed control for each motor (with high-frequency roll-off and a reference command prefilter). In Chapter 4, this is addressed for our enhanced low-cost Thunder Tumbler. The inner-loop control laws developed in Chapters 4 is based on the TITO LTI fourth order vehicle-motor \((v, \omega)\) or \((\omega_R, \omega_L)\) dynamical model presented in [15] (see discussion above).
4. **What is a suitable inner-loop model?** For a differential-drive robotic vehicle, the robot-actuator model from DC motor input voltages to the angular wheel rates is a suitable inner-loop TITO LTI model [15] (see discussion above). As such, many tools are available for design [26], [52].

5. **What is a suitable inner-loop control structure? When is a classical (decentralized) PI structure sufficient? When is a multivariable (centralized) structure essential?** For many applications (as the vehicle applications considered within this thesis), a simple PI/PID (decentralized) control law with high frequency roll-off and a command pre-filter suffices (see Chapters 3 and 4). Such an approach should work when the plant is not too coupled and the design specifications are not too aggressive relative to frequency dependent modeling uncertainty. A multivariable (centralized) structure becomes essential when the plant is highly coupled and the design specifications are very aggressive (e.g. high bandwidth relative to coupling/uncertainty)[59].

6. **What is a suitable inner-loop processor/microcontroller?** For the vehicle applications considered within this thesis, the Arduino Uno open source microcontroller development board (16MHZ ATmega328 processor, 32KB Flash Memory, 14 digital I/O pins, 6 analog inputs, $25) can be a very useful inner-loop computing engine and sensor information collecting.

The Raspberry Pi 3 Model B (1.2 GHz quad-core ARM Cortex-A53 CPU, 1GB SDRAM, 40 GPIO pins, camera interface, $40) is ideal for more intense outer-loop computations (e.g. vision based) and wireless communication. Multiple Raspberries are used for inter-vehicle communication using UDP protocol through Wi-Fi adapter.
7. **What are suitable inner-loop sensors and actuators?** For the ground vehicle applications considered within this thesis, wheel encoders and an IMU are useful inner-loop sensors. We used optical encoders for implementing our differential-drive inner-loop \((\omega_R, \omega_L)-(v, \omega)\) control laws. The IMU was used to implement our \((v, \theta)\) differential-drive outer-loop control laws. Armature controlled DC motors are useful inner-loop actuators. This is what was utilized for our ETT.

Figure 1.10 summarizes inner- and outer-loop control laws considered, analyzed and implemented within this thesis for our enhanced differential-drive Thunder Tumbler vehicles: one inner-loop \((v, \omega)\) speed control law and three outer-loop control laws: (1) \((v, \theta)\), (2) \(\Delta x, \theta\), and (3) vehicular platoon separation control

<table>
<thead>
<tr>
<th>Mathematic Name</th>
<th>Control Law</th>
<th>Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>((v, \omega))</td>
<td>Inner-loop speed control (encoders)</td>
<td><img src="image1" alt="Visualization of Inner-Loop Control Law" /></td>
</tr>
<tr>
<td>((v, \theta))</td>
<td>Cruise control along a line ((v\text{-encoders, } \theta\text{-IMU or camera}))</td>
<td><img src="image2" alt="Visualization of Outer-Loop Control Law" /></td>
</tr>
<tr>
<td>(\Delta x, \theta)</td>
<td>Separation control along a line ((\Delta x\text{-ultrasonic, } \theta\text{-IMU}))</td>
<td><img src="image3" alt="Visualization of Outer-Loop Control Law" /></td>
</tr>
<tr>
<td>(\Delta x_i)</td>
<td>Control of longitudinal platoon of vehicles ((\Delta x\text{-ultrasonic, } \theta\text{-IMU, Wi-Fi Adapter}))</td>
<td><img src="image4" alt="Visualization of Outer-Loop Control Law" /></td>
</tr>
</tbody>
</table>

**Outer-Loop Control**

8. **What are typical outer-loop objectives?** For the vehicle applications considered within this thesis, three (3) outer-loop objectives are examined (see Figure 1.10):
(1) speed-direction \((v, \theta)\) cruise control along a line by exploiting encoders for speed information and IMU or camera for directional information.

(2) lineal (directed) separation \((\Delta x, \theta)\) control by exploiting encoders for positional information, IMU or camera for directional information, and ultrasound for nearly-lineal separation information.

(3) platoon separation control. Specifically, position control involves wrapping a \((\Delta x, \theta)\) control with feed-forward path from wireless communication around a inner-loop cruise control system.

Here, outer-loop control laws are based on proportional-plus-derivative (PD) laws (with high-frequency roll-off). If the reference command is a ramp signal, proportional-integral-derivative (PID) law is needed in platoon control separation design. It must be noted that local asymptotic stability is theoretically guaranteed (and practically observed) for all of the implemented control laws. Relevant references for each outer-loop objective are presented and described in Section 1.2 on page 3. Also see work within Chapter 4 and 5.

9. **What is a suitable outer-loop model?** If the inner-loop is designed well, after it is closed it can yield a system (seen by the outer-loop controller) that is very simple looping (e.g. \(\text{diag}(\frac{a}{s+a}, \frac{b}{s+b})\), looks like identity at low frequencies). This can greatly facilitate the design of the outer-loop control system. (See work within Chapters 3 and 4.)

10. **What is a suitable outer-loop control structure? When is a more complex structure needed?** Suppose that an inner-loop speed control system has been designed. Suppose that it looks like \(\frac{a}{s+a}\). It then follows that if position is concerned, then we have a system that looks like \(\frac{a}{s(s+a)}\); i.e. there is an additional integrator present. Given this, classical control (root locus) concepts \([52]\)
can be used to motivate an outer-loop control structure $K_o = g(s + z)$. In an effort to attenuate the effect of high frequency sensor noise, one might introduce additional roll-off; e.g. $K_o = g(s + z) \left[ \frac{b}{s + b} \right]^n$ where $n = 2$ or greater. (See work within Chapters 3 and 4.) When PID is needed, the controller structure can be e.g. $K_o = \frac{g(s + z)^2}{s} \left[ \frac{b}{s + b} \right]^n$ where $n = 2$ or greater. (See work within Chapters 3 and 4.)

11. **What is a suitable outer-loop processor/microcontroller?** For the vehicle applications considered within this thesis, both Arduino Uno and Raspberry Pi 3 are each used for distinct outer-loop controller implementations.

Arduino Uno is used for inner-loop control and sensor information gathering. The Raspberry Pi 3 is used for all outer-loop control in this thesis. Raspberry Pi and Uno communicate to each other through serial communication. It is used for more demanding vision-based cruise-position-directional outer-loop control and wireless communication. The speed of Arduino Uno is limited. As such, it cannot handle intense outer-loop vision-based processing. In contrast, the Raspberry Pi 3 is very fast and has a large memory. It is very well-suited for intense outer-loop vision-based processing and inter-vehicle wireless communication.

While partial answers have been provided above, the thesis (when applicable) provides more detailed answers. When taken collectively, the contributions of this thesis are significant - particularly to those interested in developing low-cost platforms for conducting robotics/FAME research.
Key Demonstrations. To further highlight contributions of the thesis, the following demonstrations are presented and analyzed within the thesis. All demonstrations are based on differential-drive enhanced Thunder Tumbler vehicles.

- cruise control along a straight line
- vehicle-target spacing control
- longitudinal platoon separation control

For most cases, hardware (empirically obtained) data is compared with, and corroborated by, model-based simulation data. In short, the thesis uses multiple enhanced/augmented low-cost ground vehicles to demonstrate many capabilities that are critical in order to reach the longer-term FAME goal.

1.4 Organization of Thesis

The remainder of the thesis is organized as follows.

- Chapter 2 (page 25) presents an overview for a general FAME architecture describing candidate technologies (e.g. sensing, communications, computing, actuation).

- Chapter 3 (page 30) describes modeling and control issues for a differential-drive ground vehicle. In this chapter, motor-wheel system TITO LTI model (off ground model) and vehicle-motor model(on ground model) are developed carefully. The ideas presented in [59] [14] provide basis for our work. Base on this, we presents system-theoretic as well as hardware results for our differential-drive Thunder Tumbler ground robotic vehicles.
• Chapter 4 (page 92) inner loop and outer loop design for a single robot was done based on vehicle-motor on ground model. System-theoretic as well as hardware results for our differential-drive Thunder Tumbler ground robotic vehicles will be presented. Related demonstrations are described.

• Chapter 5 (page 151) longitudinal platoon control problem will first be stated. Then vehicle model for leader and followers will be developed. Design with and without leader information have been discussed. Simulation results will show that with the leader information vehicle spacings get attenuated from the front to the back of the platoon and control effort is decreasing along the platoon. Without it, spacings cannot get attenuated along the platoon or even get magnified.

• Chapter 6 (page 181) summarizes the thesis and presents directions for future robotics/FAME research. While much has been accomplished in this thesis, lots remains to be done.

• Appendix A (page 190) contains Arduino program files used to generate inner loop results for this thesis.

• Appendix B (page 264) contains all MATLAB mfiles used to generate the results for this thesis.

• Appendix C (page 294) contains Arduino code used to generate the results for this thesis.
1.5 Summary and Conclusions

In this chapter, we provided an overview of the work presented in this thesis and the major contributions. A central contribution of the thesis is improved modeling of DC motor, motor-wheel system with gearbox transmission, on ground longitudinal model. Uncertainty of modeling parameters has been exploited by testing 9 ETT robots. A updated low-cost multi-capability differential-drive Thunder Tumbler robotic ground vehicle that can further facilitates on ground robot research. New platform software has been provided, lots of hardware results can be logged and analyzed. Hardware demonstrations were conducted using our differential drive vehicles. The thesis attempts to address most critical modeling, design, and control issues of longitudinal platoon problem and makes another step toward long-term FAME research.
Chapter 2

OVERVIEW OF GENERAL FAME ARCHITECTURE

2.1 Introduction and Overview

In this chapter, we will describe a general architecture for our general FAME research. This section is based on [14], small modification is made to suit this thesis. The architecture described attempts to shed light on command, control, communications, computing (C⁴), and sensing (S) requirements needed to support a fleet of collaborating vehicles. Collectively, the C⁴ and S requirements are referred to as C⁴S requirements.

2.2 FAME Architecture

In this section, we describe a candidate system-level architecture that can be used for a fleet of robotic vehicles. The architecture can be visualized as shown in Figure 2.1. The architecture addresses global/central as well as local command, control, computing, communications (C⁴), and sensing (C⁴S) needs. Elements within the figure are now described.

- **Central Command: Global/Central Command, Control, Computing.**
  A global/central computer (or suite of computers) can be used to perform all of the very heavy computing requirements. This computer gathers information from a global/central (possibly distributed) suite of sensors (e.g. GPS, radar, cameras). The information gathered is used for many purposes. This includes temporal/spatial mission planning, objective adaptation, optimization, decision making (control), information transmission/broadcasting and the generation of
commands that can be issued to members of the fleet. Within this thesis, we simply use a central command laptop.

- **Global/Central Sensing.** In order to make global/central decisions, a suite of sensors should be available (e.g. GPS, radar, cameras). This suite provides information about the state of the fleet (or individual members) that can be used by central command. Within this thesis, global sensing is achieved by feeding back real-time video from our enhanced differential-drive robotic Thunder Tumbler vehicles to our central command laptop. Such a lab-based system offers the benefit that it can be fairly easily transported for use elsewhere (with some peruse calibration). Such a system can be used to examine a wide range of scenarios. Also ongoing is an effort to more profoundly exploit vision on individual vehicles [60], [24].
• **Fleet of Vehicles.** The fleet of vehicles can consist of ground, air, space, sea or underwater vehicles. Ground vehicles can consist of semi-autonomous/autonomous robotic vehicles (e.g. differential-drive, rear-wheel drive, etc.). Here, autonomous implies that no human intervention is involved (a longer-term objective). Semi-autonomous implies that some human intervention is involved. Air vehicles can consist of quadrotors, micro/nano air vehicles, drones, other air vehicles and space vehicles. Sea vehicles can consist of a variety of surface and underwater vehicles. Within this thesis the focus is on ground vehicles (e.g. enhanced Thunder Tumbler differential-drive). In previous work by Lin [14], he had done one demonstration whereby a differential-drive Thunder Tumbler ground vehicle follows a remotely controlled (AR drone) quadrotor. Within this thesis, all research work concentrate on ETT.

• **Local Computing.** Every vehicle in the fleet will (generally speaking) have some computing capability. Some vehicles may have more than others. Local computing here is used to address command, control, computing, planning and optimization needs for a single vehicle. The objective for the single vehicle, however, may (in general) involve multiple vehicles in the fleet (e.g. maintaining a specified formation, controlling the inter-vehicle spacing for a platoon of vehicles). Local computing can consist of a computer, microcontroller or suite of computers/microcontrollers. Within this thesis, we primarily exploit Arduino Uno microcontroller (16MHZ ATmega328 processor, 32KB Flash Memory, 14 digital I/O pins, 6 analog inputs, $25) [36] and Raspberry Pi 3(1200 MHz quad-core ARM Cortex-A53 CPU (Pi 2: 900 MHz quad-core ARM Cortex-A7 CPU), 1GB SDRAM, 40 GPIO pins, camera interface, $40) computer boards for local computing on a vehicle. They are low-cost, well supported (e.g. some high-level
software development tools Arduino IDE and customized Linux operation system Raspbian on Raspberry Pi), and easy to use.

- **Local Sensing.** Local sensing, in general, refers to sensors on individual vehicles. As such, this can involve a variety of sensors. These can include encoders, IMUs (containing accelerometers, gyroscopes, magnetometers), ultrasonic range sensors, LIDAR, GPS, radar, and cameras. Within this thesis, we exploit optical encoders (RPR220 photo-interrupter REFL 6mm 800nm, home-made encoder disk 20 black-white pair, ) [47], Adafruit BNO055 IMUs to measure vehicle rotation (9DOF, Accelerometer ±2,4,6,8,16g. Gyro ±125−2000°/sec. Compass ±1300 µT (x-,y-axis) ±2500 µT (z-axis)) [58], ultrasonic range sensors (40kHz, 0.02-5 m, approximately ±8° directional) [68], and Raspberry Pi cameras (2592×1944, 30 fps, 150 MPs, MPEG-4) [57]. LIDAR, GPS and radar are not used.

- **Local Communications.** Here, local communications refers to how fleet vehicles communicate with one another as well as with central command. In this thesis, vehicles exploit WiFi (IEEE 802.11 (2.4, 5GHz) standard) [67] to send locally obtained Raspberry Pi camera video (2592×1944, 30 fps, 150 MPs, MPEG-4) [57] to a central command laptop.
2.3 Summary and Conclusions

In this chapter, we described a general (candidate) FAME architecture for a fleet of cooperating robotic vehicles. Of critical importance to properly assess the utility of a FAME architecture is understanding the fundamental limitations imposed by its subsystems (e.g. bandwidth/dynamic, accuracy/static). This “fundamental limitation” issue is addressed within Chapter 4 where enhanced differential-drive Thunder Tumbler vehicles are used as the central building block for the fleet.
Chapter 3

MODELING FOR SINGLE VEHICLE

3.1 Introduction and Overview

The purpose of this chapter is to illustrate fundamental modeling and control design methods for a differential-drive (DD) robotic ground vehicle. This is achieved by presenting relevant model trade studies and then illustrating the design of an inner-loop \((v, \omega)\) speed control law and associated tradeoffs. As discussed in Chapter 1, such a control law is generally the basis for any outer-loop control law (e.g. cruise control along a line, target-separation control, platoon separation control). A two-input two-output (TITO) linear time invariant (LTI) model, taken from [15], is used as the basis for all developments within the chapter. The model is analyzed and used to conduct relevant parametric trade studies (e.g. mass, moment of inertia, motor back EMF constant, motor armature resistance) which provided insight about the vehicle being addressed. The decentralized controller is based on classical single-input single-output (SISO) methods [32], [52]. Centralized controller designs for the inner loop had been discussed within [14].

3.2 Description of Hardware

In [14], the authors has shown how to take off-the-shelf (low-cost) remote control “toy” vehicles and convert them into intelligent multi-capability robotic platforms that can be used for conducting robotics/FAME research. In this section we briefly describe each component on our low-cost robots, only hardware updates will be discussed in detail. All discussion provided below focuses on our differential-drive
Thunder Tumbler vehicles (9 ETTs in total). To distinguish the work from [14], we named the old one ETT 1 and the new one ETT 2 when needed. More specifically, all differential-drive Thunder Tumbler vehicles (ETT 2) were augmented with the following: Arduino Motor Shield, Arduino Uno microcontroller board, Optical wheel encoders (Magnets wheel encoder on ETT 1), BNO055 IMU (LSM9DS0 IMU on ETT1), Raspberry Pi 3 (Raspberry Pi 2 on ETT 1), Raspberry 5MP camera module, HC-SR04 Ultrasonic Distance Sensor, TP-LINK WiFi adapter (Edimax WiFi adapter on ETT 1). Only updates will be described in detail below, other overlapped work detail can be found in his thesis [14] page 107-115, we just keep same structure and list each item for completeness. An enhanced Thunder Tumbler is shown in Figure 1.1, page 12.

A hardware components list for an enhanced Thunder Tumbler is given in Table 3.1. The table shows the cost is less than $220.
Table 3.1: Hardware Components for Enhanced Differential-Drive Thunder Tumbler Robotic Vehicle

1. **Differential-Drive Thunder Tumbler.** Each Thunder Tumbler is a $10 “toy” vehicle, it is a differential-drive vehicle with two DC motors - one on left wheel, one on right wheel. Each differential-drive Thunder Tumbler vehicle was augmented/enhanced to provide a suite of substantive capabilities.

2. **DC Motors.** Two 6V brushed armature controlled DC motors are on each differential-drive Thunder Tumbler vehicle. The DC motors receive voltage sig-
nals from an Arduino motor shield and apply the required torques to each of the Thunder Tumbler’s wheels.

3. **Arduino Uno Open-Source Microcontroller Board.** Each Thunder Tumbler is equipped with an onboard Arduino Uno microcontroller board (see Figure 1.4 on page 14). In our thesis, Arduino only served as inner loop controller and provide necessary sensor information to Raspberry Pi.

4. **Arduino Motor Shield.** An Adafruit Motor/Stepper/Servo Shield for Arduino v2 Kit (v2.3) was used in this thesis see Figure 1.5 on page 14,

5. **Arduino Uno Open-Source Microcontroller Board.** Each Thunder Tumbler is equipped with an onboard Arduino Uno microcontroller board (see Figure 1.4 on page 14).

6. **Optical Infra-red Reflective Photosensor-Based Encoders.** A reflective photosensor sensor (photo-interrupter) RPR220 and home-made code disk (20 black-white pairs per wheel, see Figure 1.2 on page 13) are used as wheel encoders. Wheel encoders are used for (dead-reckoning) speed/position control. The wheel encoders count the pulses that black-white stripe pair passes the photo-interrupter. This information is sent to the Arduino Uno which can then calculate/estimate vehicle velocity and translational displacement, vehicle angular velocity and angular displacement.

7. **Inertial Measurement Unit (IMU).** The IMU can collect acceleration, angular velocity and orientation of the vehicle. In this thesis, it mainly collects the absolute orientation information \( \theta \) of the robot and sends the information to the Arduino Uno. An (BNO055 9dof) inertial measurement unit (IMU) is used for directional control (see Figure 1.3 on page 13). The 9 dof include 3 acceleration channels, 3 angular rate channels and 3 magnetic field channels.
Range features are as follows: ± 2/4/6/8/16 g linear acceleration full scale, ± 13 gauss (x-,y-axis) ± 25 gauss (z-axis)), magnetic full scale and ± 125-2000 degree/sec angular rate full scale. It has 16-bit data output and SPI/I2C serial interfaces.

8. **Raspberry Pi 3 Single Board Computer.** Each Thunder Tumbler has an onboard Raspberry Pi 3 Model B single board computer (see Figure 1.6 on page 15). Raspberry Pi 3 Model B characteristics include:

Broadcom BCM2837 with a 1.2GHz quad-core ARM Cortex-A53 64-bit CPU and VideoCore IV GPU, 1GB SDRAM (bus synchronous dynamic RAM) at 900 MHz (shared with GPU), on-board Bluetooth Low Energy (BLE), on-board BCM43143 WiFi. Other detail hardware information can be found [14] and [39].

For raspberry pi 3 and pi 2 comparison, one can refer to online reports [41].

Summary of Raspberry Pi Use:

- **Communicate with Arduino During Robot Operation.** We also used C language USB (Serial) communication between the Pi and Arduino. There are many ways to establish communication between the Raspberry Pi and the Arduino such as using the GPIO and Serial pins or using I2C communication (using the SCL-clock and SDL-data pins). The simplest way to get the two devices talking is to use the micro USB to USB adapter that comes with the Arduino Uno. By using the open-source serial library package, we can use C to read from and write to Arduino’s serial port.

- **Implement all outer-loops include the following:** (1) Outer-loop 1 \((v, \theta)\) cruise control along a line (2) Outer-loop 2 \((\Delta x, \theta)\) separation-direction control along a line, (3) Outer-loop 3 is our platoon control law with leader information.
- **Communication with central command laptop.** This remote capability permit a robot to switch between autonomous and semi-autonomous modes of operation. Such switching has been done by keyboard command thread running on Pi.

- **Communication with other Pi**  When platoon control need leader informamtion, Pi is used to send/receive leader information

9. **Raspberry 5MP Camera Module.** Each Thunder Tumbler has an onboard Raspberry Pi 5MP camera (see Figure 1.7 on page 15). The camera module collects image information and sends it to the onboard Raspberry Pi.

10. **Ultrasonic Distance Sensor.** A forward-looking (40kHz SR04) ultrasonic distance/rangefinder sensor was placed on each robot (see Figure 1.9 on page 16). The ultrasonic sensor collects range/separation \( \Delta x \) information (at ranges of 2 cm to 5m) and sends the information to the Arduino Uno.

11. **TP-Link WiFi Adapter.** An TP-Link adapter (see Figure 1.8 on page 16) is used to connect Pi via wireless router.

12. **Wireless Router.** A TP-LINK wireless router is used to receive radio signals from a remotely situated WiFi adapter on the differential-drive mobile robot and transmits the radio signals to a wireless adapter on the remotely situated central command laptop.
3.3 Modeling of a Differential-Drive Ground Robotic Vehicle

Many mobile robots use a so-called differential-drive drive mechanism. Such a mechanism involves two rear wheels that are independently controlled via torque-generating DC motors. The inputs to the DC motors are voltages. Within this thesis, the motors are assumed to be identical in order to simplify the presentation. In practice, motor differences must be accounted for. This, in part, is addressed by the motor control laws being employed. Within this section, we first examine the TITO LTI model that was presented within [15]. This model was used for control law design within the MS thesis [59] and [14]. It serve as the basis for plant analysis and inner-loop control designs for differential-drive vehicle. Then we modified this model and put forward on ground vehicle-motor dynamic model from on ground experimental results and verify the model with hardware. The on ground model will be used for designing inner-loop and outer-loops in this thesis. Before presenting the model, we first discuss the robot kinematics. A great deal of insight can be gained by first understanding the kinematics. The robot dynamics are then examined - first without and then with the DC motor dynamics.

Before continuing with our presentation it is useful to define key robot variables and parameters to be used for introduction of differential-drive robot modeling. The nominal value is from ETT V1 and uncertainties from multiple ETTs. There are 9 ETTs in total, we named it from V1 to V9. This is done within Table 3.2.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Nominal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>Mass (Enhanced Vehicle)</td>
<td>0.87 ± 0.02 kg</td>
</tr>
<tr>
<td>$m_o$</td>
<td>Mass (Original Vehicle)</td>
<td>0.56 ± 0.02 kg</td>
</tr>
<tr>
<td>$I_z$</td>
<td>Moment of Inertia</td>
<td>0.0051 kg $m^2$</td>
</tr>
<tr>
<td>$r$</td>
<td>Wheel Radius</td>
<td>0.05 ± 0.005 m</td>
</tr>
<tr>
<td>$d_w$</td>
<td>Distance between Rear Wheels</td>
<td>0.14 ± 0.005 m</td>
</tr>
<tr>
<td>$L_a$</td>
<td>Armature Inductance</td>
<td>$(2.7 ± 0.2) \times 10^{-8}$ H</td>
</tr>
<tr>
<td>$R_a$</td>
<td>Armature Resistance</td>
<td>0.9 ± 0.15 $\Omega$</td>
</tr>
<tr>
<td>$K_g$</td>
<td>Gearbox Ratio</td>
<td>18.48</td>
</tr>
<tr>
<td>$K_b$</td>
<td>back EMF Constant</td>
<td>$(3.2 ± 0.5) \times 10^{-4}$ V/(rad/sec)</td>
</tr>
<tr>
<td>$K_t$</td>
<td>Torque Constant</td>
<td>$(3.2 ± 0.5) \times 10^{-4}$ Nm/A</td>
</tr>
<tr>
<td>$I$</td>
<td>Equivalent Moment of Inertia</td>
<td>$(2.91 ± 1.00) \times 10^{-6}$ kg $m^2$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Speed Damping Constant</td>
<td>$(7.04 ± 3.00) \times 10^{-7}$ Nms</td>
</tr>
<tr>
<td>$v_{\text{max}}$</td>
<td>Max. Speed (Enhanced Vehicle)</td>
<td>2.1 ± 0.2 m/sec</td>
</tr>
<tr>
<td>$v_{\text{max}_o}$</td>
<td>Max. Speed (Original Vehicle)</td>
<td>4.5 ± 0.2 m/sec</td>
</tr>
<tr>
<td>$E$</td>
<td>Efficiency - Power Out / Power In</td>
<td>0.15</td>
</tr>
<tr>
<td>$a_{\text{max}}$</td>
<td>Max. Acceleration (Enhanced)</td>
<td>2.5 ± 0.5 m/sec$^2$</td>
</tr>
<tr>
<td>$\omega_{\text{max}}$</td>
<td>Max. Angular Wheel Velocity</td>
<td>46 ± 4 rad/sec</td>
</tr>
<tr>
<td>$\tau_{\text{max}}$</td>
<td>Max. Torque (Zero RPM)</td>
<td>0.048 ± 0.05 Nm</td>
</tr>
<tr>
<td>$e_{\text{max}}$</td>
<td>Max. Motor Voltage</td>
<td>5.25 ± 0.05 V</td>
</tr>
</tbody>
</table>

Table 3.2: Thunder Tumbler Nominal Parameter Values with Uncertainty
3.4 Differential-Drive Robot Kinematics

Figure 3.1 can be used to understand the kinematics of a differential-drive ground robot [17].

The point that the robot rotates about at a given instant in time is called the instantaneous center of curvature (ICC) [17]. If \((x, y)\) denotes the planar inertial coordinate of the robot and \(\theta\) denotes the direction of the robot’s longitudinal body axis with respect to the \(x\)-axis, then we obtain the following nonlinear kinematic model:

\[
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix} =
\begin{bmatrix}
\cos \theta \\
\sin \theta \\
0
\end{bmatrix} v +
\begin{bmatrix}
0 \\
0 \\
1
\end{bmatrix} \omega
\]

(3.1)

where

\[
v = \sqrt{\dot{x}^2 + \dot{y}^2}
\]

(3.2)

denotes the translational speed of the robot and \(\omega = \dot{\theta}\) denotes its angular speed.
Within the above very simple (and intuitive) model, \( v \) can \( \omega \) can be thought of as inputs (or controls). This is not intuitive - especially to a controls person. Why? As discussed within Chapter 1, \( v \) and \( \omega \) cannot be instantaneously generated because of real-world mass-inertia effects (at least not without infinite forces!). In practice, \( v \) and \( \omega \) are generated by applying voltages to the left and right wheel DC motors. This will be become evident below when we discuss the motor dynamics and their relationship to the above model.

At this point, it is instructive to relate the \((v, \omega)\) to the angular velocities \((\omega_L, \omega_R)\) of the left and right rear wheels. Why? The idea here, is that if we can precisely control \((\omega_L, \omega_R)\), then we will be able to precisely control \((v, \omega)\). The desired relationships are as follows:

\[
v = \left[ \frac{r(\omega_R + \omega_L)}{2} \right] \quad \omega = \left[ \frac{r(\omega_R - \omega_L)}{d_w} \right]
\]

(3.3)

where \( r \) denotes the wheel radius and \( d_w \) denotes the distance between the rear wheels. Both \( r \) and \( d_w \) are assumed to be constant. Within Figure 3.1, the point vehicle coordinate \((x, y)\) is located on the vehicle’s longitudinal body axis directly in between the two rear wheels.

To derive the above relationships, we proceed as follows. Let \( v_l \) and \( v_r \) denote the left and right wheel translational speeds along the ground. If \( R \) denotes the “signed” distance from the \((x, y)\) coordinate of the vehicle to the ICC, then it follows that

\[
(R + d_w/2)\omega = v_r \quad (R - d_w/2)\omega = v_l
\]

(3.4)

From these equations, it follows (after some algebra) that

\[
R = \frac{d_w}{2} \left[ \frac{v_l + v_r}{v_r - v_l} \right] \quad \omega = \left[ \frac{v_r - v_l}{d_w} \right]
\]

(3.5)

Next, we note that

\[
v = R\omega \quad v_l = r\omega_L \quad v_r = r\omega_R
\]

(3.6)
Substituting $v_l = r\omega_L$ and $v_r = r\omega_R$ into $\omega = \frac{v_r - v_l}{d_w}$, yields the relation $\omega = \frac{r(\omega_R - \omega_L)}{d_w}$.

Substituting $R = \frac{v}{\omega}$ and $\omega = \frac{v_r - v_l}{d_w}$ into $R = \frac{d_w}{2} \frac{v_r + v_l}{v_r - v_l}$ yields the relation $v = \frac{v_r + v_l}{2}$.

Substituting $v_l = r\omega_L$ and $v_r = r\omega_R$ into this relation then yields the desired result $v = \frac{r(\omega_R + \omega_L)}{2}$. This completes the derivation.

It is convenient to rewrite the above relations in vector-matrix form as follows:

$$
\begin{bmatrix}
v \\
\omega
\end{bmatrix} = M
\begin{bmatrix}
\omega_R \\
\omega_L
\end{bmatrix}
M = \begin{bmatrix}
\frac{r}{2} & \frac{r}{2} \\
\frac{r}{d_w} & -\frac{r}{d_w}
\end{bmatrix}
$$

(3.7)

Again, the importance of the above relation stems from the fact that if we can control $(\omega_L, \omega_R)$ well, then we shall see that we will be able to control $(v, \omega)$ well - the latter being the prime directive of this chapter.

### 3.5 Differential-Drive Robot Dynamics

In order to more accurately represent the system, we consider a dynamical model - one that captures mass-inertia effects. The following intuitive representation of the model comes from [22]:

$$
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & 0 \\
\sin \theta & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
v \\
\omega
\end{bmatrix}
$$

(3.8)

$$
\begin{bmatrix}
\dot{v} \\
\dot{\omega}
\end{bmatrix} =
\begin{bmatrix}
\frac{1}{m} & 0 \\
0 & \frac{1}{I_z}
\end{bmatrix}
\begin{bmatrix}
F \\
\tau
\end{bmatrix}
$$

(3.9)

$$
\tan \theta = \frac{\dot{y}}{\dot{x}}
$$

(3.10)

where $F$ represents the applied translational force along the vehicle’s longitudinal body axis, $\tau$ represents the applied torque about the vertical $z$ axis passing through
the point \((x, y)\), \(m\) denotes the mass of the vehicle and \(I_z\) denotes its moment of inertia about the vertical \(z\) axis passing through the point \((x, y)\). From the above, we see that the dynamical model consists of the following five equations: three kinematic model equations within the matrix-vector equation (3.8), two Newtonian dynamical equations within the matrix-vector equation (3.9), and the no slipping (non-holonomic) constraint within equation (3.10). It should be noted that in practice, the force \(F\) and torque \(\tau\) are generated by the two DC motors on the rear wheels. This shall become evident within the subsections that follow below. It should also be noted that the above dynamical model can be derived using the classic Euler-Lagrange formulation [22] [15].

As suggested above, the kinematic model neglects dynamic mass-inertia effects. As such, the kinematic model is just an approximation to the dynamic model. As expected, and it will be shown, the kinematic model is a good approximation to the dynamical model when \((v, \omega)\) can be generated quickly. Intuitively, this occurs when \(m\) and \(I_z\) are sufficiently small (see equation (3.9)) or the inner \((v, \omega)\) loop has a sufficiently large bandwidth. This shall become evident below.

Finally, it is important to note the relationship between \((F, \tau)\) and the left-right motor torques \((\tau_L, \tau_R)\). The desired relationship is similar in form to the angular velocity relationships within equation 3.3 and is given by

\[
F = \left[ \frac{\tau_R + \tau_L}{r} \right] \\
\tau = \left[ \frac{d_w(\tau_R - \tau_L)}{2r} \right]
\]

(3.11)

Here, \(\tau_L\) and \(\tau_R\) represent the torques acting on the left and right wheels, respectively.

Next, we discuss the motor (actuator) dynamics. Ultimately, the motors are responsible for producing the wheel torques \((\tau_L, \tau_R)\) and hence the associated pair \((F, \tau)\). The latter, of course, are directly responsible for producing the vehicle speeds \((v, \omega)\).
3.5.1 DC Motor (Actuator) Dynamics with Gearbox

DC motors are widely used in robotics applications. They are the mostly widely used actuator class in mobile robots. It is important to take DC motor dynamics into account when constructing a robot’s model. There are two classes of DC motors: (1) armature-current controlled and (2) field-current controlled [52]. Within this thesis, we shall focus on the former; i.e. armature-current controlled DC motors. The dynamics for a DC motor can be visualized as shown within Figure 3.2. The associated equations are as follows:

![Figure 3.2: Visualization of DC Motor Speed-Voltage Dynamics](image)

**Armature Equation:**
\[ e_a = L_a \frac{di_a}{dt} + R_a i_a + e_b \]  \hspace{1cm} (3.12)

**back EMF Equation:**
\[ e_b = K_b K_g \omega_s \]  \hspace{1cm} (3.13)

**Torque Equation:**
\[ \tau_s = K_t K_g i_a \]  \hspace{1cm} (3.14)

**Load Equation:**
\[ I \dot{\omega}_s + \beta \omega_s = \frac{\tau_s}{K_g^2} \]  \hspace{1cm} (3.15)

**Gear Relationship:**
\[ K_g = K_{g12} K_{g23} \]  \hspace{1cm} (3.16)
Here, \( e_a \) represents the applied armature voltage. This is the control input for an armature controlled DC motor. Other relevant variables are as follows: \( i_a \) represents the armature current, \( e_b \) represents the back EMF, \( \tau_s \) represents the torque exerted by the motor on the motor shaft-load system, \( \omega_s \) represents the motor shaft angular speed. Relevant motor parameters are as follows: \( L_a \) represents the armature inductance (often negligibly small in many applications), \( R_a \) represents the armature resistance, \( K_b \) represents the back EMF motor constant, \( K_t \) represents the motor torque constant, \( K_g \) represents the gearbox ratio of motor shaft-load system \( \beta \) represents a load-motor speed rotational damping constant, and \( I \) represents the moment of inertia of the motor shaft-load system.

From the above, one can obtain the transfer function from the input voltage \( e_a \) to the angular speed \( \omega_s \):

\[
\frac{\omega_s}{e_a} = \frac{\frac{K_t}{K_g}}{(I + \beta)(L_a s + R_a) + K_t K_b}
\]

(3.17)

Given the above, some observations are in order. The motor speed transfer function is generally second order. If the armature inductance \( L_a \) is negligibly small (i.e. \( \omega_s L_a << R_a \) over the operational bandwidth), then the motor speed transfer function becomes first order. In such a case, we have the following speed-voltage transfer function approximation:

\[
\frac{\omega_s}{e_a} \approx \left[ \frac{\frac{K_t}{K_g}}{(I + \beta)(R_a) + K_t K_b} \right] \left[ \frac{R_a}{L_a s + R_a} \right]
\]

(3.18)

In such a case, the dominant motor pole becomes \( s \approx - \left( \frac{R_a \beta + K_t K_b}{R_a I} \right) = - \frac{\beta}{I} - \frac{K_t K_b}{R_a I} \) and the associated inductance pole becomes large and given by \( s \approx - \frac{R_a}{L_a} \). Given this, we see that the motor response is faster for larger \((\beta, K_t, K_b)\) and smaller \((I, R_a)\). If the armature inductance is neglected, then the speed-voltage transfer function becomes first order. Generally, \( K_t = K_b \).
Model Parameters Want to Determine. Among all the symbols defined above, finally we want the following (7) parameters for motor-wheel load system: \( R_a, L_a, K_g, K_b, K_t, I, \beta \).

- **Armature Inductance \( L_a \).** Armature inductance is usually very small in toy DC motor, we use DE5000 high performance LCR meter [37] to measure it. An LCR meter is a type of electronic test equipment used to measure the inductance (L), capacitance (C), and resistance (R) of an electronic component. In the simpler versions of this instrument the impedance was measured internally and converted for display to the corresponding capacitance or inductance value. Readings should be reasonably accurate if the capacitor or inductor device under test does not have a significant resistive component of impedance. More advanced designs measure true inductance or capacitance, as well as the equivalent series resistance of capacitors and the Q factor of inductive components. The DE-5000 is a portable, high-performance LCR meter that is full-featured yet cost effective ($140). It measures in true 4-wire Kelvin mode and rivals the capabilities and options of many of its bench counterparts. This LCR meter features automatic L-C-R selection, a Sorting mode, and selectable test frequencies. It can transfer data to a PC via a fully isolated, optical IR-USB interface. For 2000 mH range, the inductance accuracy is 0.1 mH. We measured (10) motors’ armature inductance, results are shown in Figure 3.3
Figure 3.3: dc Motor Armature Inductance of 10 ETT Motors

From the figure above, we can see measurements are close to the average result is around 270 µH.

- Armature resistance $R_a$. Armature resistance is usually very small in toy DC motor, one common way is to using stall method to measure it. Stall method means we applied constant dc voltage to the motor and stall the shaft to do the measurement of current. From equation 3.12 and 3.13, at steady state with 0 angular velocity, armature equation reduces to $e_a = R_a i_a$, then apply Ohm’s law: $R_a = \frac{V}{i_a}$. We did the measurement for 10 ETT motors, using two methods and did 5 times measurement for each motor. Results are shown in Figure 3.4
Armature Resistance of 10 Motors

<table>
<thead>
<tr>
<th>Motor Index</th>
<th>Resistance(Ohm)</th>
<th>Stall Method</th>
<th>LCR Meter</th>
<th>Stall Method Avg</th>
<th>LCR Meter Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>10</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Figure 3.4: dc Motor Armature Inductance of 10 ETT Motors

From the figure above, we can see two methods are close to each other, which justify our result. Average results of 10 motors are almost the same, specifically, stall method at 0.8933 Ω and LCR meter at 0.8874 Ω.

- **Gearbox Ratio of Motor-Wheel System $K_g$**. We decomposed the motor gearbox, disassembled gearbox is shown in Figure 3.5 and modeling our rotating system as follows. The related symbols are defined in Table 3.3.

Based on modeling of rotating systems and gears in [52],

$$n = \frac{1}{K_g} = \frac{r_1}{r_2} = \frac{N_1}{N_2} = \frac{\tau_1}{\tau_2} = \frac{\omega_1}{\omega_2} \quad (3.19)$$

where $(r_1, r_2)$ are the respective gear radii, $(N_1, N_2)$ are the respective number of teeth on each gear. We say we have an $n : 1$ or $K_g$ gear ratio between loads.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_g$</td>
<td>Gear Ratio between First and Third Load</td>
</tr>
<tr>
<td>$K_{g12}$ (n12)</td>
<td>Gear Ratio between First and Second Load</td>
</tr>
<tr>
<td>$K_{g23}$ (n23)</td>
<td>Gear Ratio between Second and Third Load</td>
</tr>
<tr>
<td>$r_1$</td>
<td>Radius of First Pinion Gear on First Load</td>
</tr>
<tr>
<td>$r_{21}$</td>
<td>Radius of Gear Connect the First and Second Load</td>
</tr>
<tr>
<td>$r_{23}$</td>
<td>Radius of Gear Connect to the Second and Third Load</td>
</tr>
<tr>
<td>$N_1$</td>
<td>Teeth Number on First Load</td>
</tr>
<tr>
<td>$N_{21}$</td>
<td>Teeth Number on Gear Connect the First and Second Load</td>
</tr>
<tr>
<td>$N_{23}$</td>
<td>Teeth number on Gear Connect the Second and Third Load</td>
</tr>
<tr>
<td>$\tau_m$</td>
<td>Torque Generated by Motor at First Load</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>Torque Exerted by the First Gear on the Second Load</td>
</tr>
<tr>
<td>$\tau_s$</td>
<td>Torque at Output Shaft Connect to the Wheel (Third Load)</td>
</tr>
<tr>
<td>$\omega_m$</td>
<td>Angular Speed of Pinion Gear at First Load</td>
</tr>
<tr>
<td>$\omega_2$</td>
<td>Angular Speed of Gears on Second Load</td>
</tr>
<tr>
<td>$\omega_s$</td>
<td>Angular Speed of the Wheel Gear(Third Load).</td>
</tr>
</tbody>
</table>

Table 3.3: Thunder Tumbler Gearbox Symbols

$\tau_1$ denote the torque exerted by the second gear on the first gear and $\tau_2$ denote the torque exerted by the first gear on the second, $(\omega_1, \omega_2)$ are the respective angular speed of each gear.

In ETT, the gearbox has two level transmission gear pairs. In the first level transmission:

$$n_{12} = \frac{1}{K_{g12}} = \frac{r_1}{r_{21}} = \frac{N_1}{N_{21}} = \frac{\tau_m}{\tau_2} = \frac{\omega_m}{\omega_2} \quad (3.20)$$
In the second level transmission:

\[ n_{23} = \frac{1}{K_{g23}} \frac{r_{23}}{r_3} = \frac{N_1}{N_{23}} = \frac{\tau_2}{\tau_s} = \frac{\omega_2}{\omega_s} \quad (3.21) \]

We have \( N_1 = 10, N_{21} = 44, N_{23} = 10 \) and \( N_3 = 42 \), then gear ratio:

\[ K_g = K_{g12}K_{g23} = \frac{N_{21}}{N_1} \frac{N_3}{N_{23}} \approx 18.48 \quad (3.22) \]

Other relation follows like below:

\[ K_g = \frac{\omega_m}{\omega_s} = \frac{\tau_s}{\tau_m} \quad (3.23) \]

After obtaining \( L_a, R_a \) and \( K_g \), we are ready for the rest \( K_b, K_t, I, \beta \). Previous result of motor armature inductance \( L_a \) shows that it can be neglected reasonably.

- **back EMF Constant** \( K_b \). From armature equation 3.12 and back EMF equation 3.13, we observed that at steady state,

\[ K_b = \frac{e_a - R_ai_a}{K_g\omega_s} \quad (3.24) \]
Given certain constant $e_a$, we measured armature current $i_a$ and angular speed of the wheel at steady state, using equation to estimate $K_b$. To fully capture the motor dynamics, we applied different voltage from 0.5V to 6V and for each voltage input we repeat the experiment twice. The estimate result is shown in Figure 3.6

![Figure 3.6: DC Motor back EMF Constant of Left and Right ETT Motors](image)

From the result, we found that $K_b$ lay down around 0.0032 V/(rad/sec) when applied armature voltage in middle range (1V 5V) and deviation was bigger at lower voltage and high voltage. This result is as expected, because static friction can have significant impact on transient behavior at motor starting stage. When applied relative high voltage (above 5V), battery internal resistance may have apparent impact on this. The figure also shows that back EMF constant $K_b$ of left and right motor are almost the same. Another group (group 2) were carried for this measurement, it corroborated the fidelity of our experiment.

- **Torque Constant $K_t$**. By assumption $K_t = K_b$. In practice, this may not be true
because of “internal phenomena”. A precise power balance can be preformed by considering all “internal phenomena” (e.g. internal losses, field energy, etc.)[52].

- **Equivalent Moment of Inertia** $I$ and **Speed Damping Constant** $\beta$. According to 3.17, we observe that the

\[
\text{Motor dc Gain} = \frac{K_i}{K_g} \frac{K_t}{R_a \beta + K_t K_b} \quad (3.25)
\]

\[
\text{Motor Dominant Pole} = \frac{R_a \beta + K_t K_b}{R_a I} \quad (3.26)
\]

Figure 3.7: DC Motor Output $\omega_s$ Response to 1.02V Step Input - Hardware

Figure 3.7 shows the hardware measured output (wheel speed) response to a input voltage about 1V. Actually we did 24 experiments from 0.6V to 5.2V voltage input, and for each voltage we did two groups of experiments to verify our measurement by comparison. Through this intensive experiments, we can fully capture the motors’ characteristic. And the big range of voltage input, we will see the nonlinear effect due to static friction at lower voltage and saturation
effect at high voltage. By this, we have better understand our model limitation and provide valuable data for future research. We processed the data set and came up with a nominal model. The nominal model will be used as foundation for inner loop design. This figure is a sample of the entire experiment process, through it we can tell that nominal model is pretty accurate at a wide range. Results from group 2 are close to group 1, which further corroborate our model. According to the experimental (hardware obtained) result shown in Figure 3.8 and Figure 3.9 the motor has a dc gain between 15-17 and dominant pole 4-6.

Figure 3.8: DC Gain Distribution of Step Response at Different Voltage Step Input
The dominant pole is obtained by settling time. For example, of $1scc = 5\tau = \frac{5}{[\text{real pole}]}$ implies a real pole is at $s = -5$ With this and the dc gain, by solving above equations (3.25)-(3.26), we got estimated $I$ and $\beta$, the result shown in Figure 3.10 and 3.11 respectively.
From the result, we found that left and right motor looks similar under same armature voltage input. The trend of parameter chaining are the same, specif-
ically the moment of inertia goes higher as applied voltage rises, while the angular speed damping coefficient goes down. From group 2, we can tell that the deviation of $\beta$ is huge, especially in lower applied voltage (less than 2V) and settles at higher voltage test range; deviation of $I$ is relatively small. Both parameters vary a lot from low voltage to high voltage. Nonlinear effect due to static friction play significant role in these two parameters, battery internal resistance may also have strong impact on this. The nonlinear modeling of friction in DC motor is extensively analyzed in literature, we will not addressed in detail in this thesis. Here we take the average value as the nominal value, $I = 2.91 \times 10^{-6} Kgm^2$, $\beta = 7.04 \times 10^{-7} Nm \cdot sec$. The dead-zone due to static friction we observed is between 0.4V - 0.6V (off-ground), saturation occurs when input voltage bigger than 5V. Future work can investigate nonlinear model, consider more complex friction model and effect of battery internal resistance.

By far, we got all the parameters needed to model DC motor dynamic characteristics at low frequency. Averaging the distribution of dc gain and dominant pole, we provided a estimated nominal numerical transfer function (from voltage to angular velocity).

$$P_{motor} = \frac{\omega_s}{e_a} = 16.246 \left[ \frac{4.685}{s + 4.685} \right]$$ (3.27)

To verify our model, we compared the time domain step response of hardware and simulation. We did the comparison for experiments from 0.6V to 5.2V voltage input, for conciseness, only response to 1.02V, 2.04V, 3.06V, 4.08V, 5.10V (PWM signal 50, 100, 150, 200, 250) were shown in Figure 3.12, Figure 3.13, Figure 3.14, Figure 3.15, and Figure 3.16.
Figure 3.12: Motor Output $\omega_s$ Response to 1.02V Step Input - Hardware and Decoupled Model

Figure 3.13: Motor Output $\omega_s$ Response to 2.04V Step Input - Hardware and Decoupled Model
Figure 3.14: Motor Output $\omega_s$ Response to 3.06V Step Input - Hardware and Decoupled Model

Figure 3.15: Motor Output $\omega_s$ Response to 4.08V Step Input - Hardware and Decoupled Model
From the above results, we see the nominal model fit well with hardware data, two groups of result matches each other. Two motors at left and right shows almost same behavior especially when input voltage between 1-4V. Till now, it’s safe to make the assumption that two DC motors are identical (input voltage under 4V). The TITO LTI ($\omega_R, \omega_L$) vehicle-motor model is assumed to be diagonal. It is natural to ask:

How can we explain the difference between the 

hardware and simulated step responses in Figure 3.12?

Differences Between Simulated and Hardware Step Responses.

Transient Differences. Consider the hardware and simulated step responses in Figure 3.12. While the transients are similar, there is a difference. The hardware response is a bit more faster than the simulated response. This difference can be explained by a higher fidelity model of the motor-wheel combination - one that incor-
oporates armature inductance. For that, we need much faster sampling rate to capture the fast pole \(-\frac{R_a}{L_a} \approx -2977\). Inaccurate measurement limited by encoder resolution could also lead to different transient speed looking. This need very high resolution commercial encoder. Future work should increase encoder resolution first and then investigate higher order model.

**Steady State Differences: Battery Internal Resistance.** First thing is the ripple of hardware. This is inherently due to encoder measurement noise at 10 Hz sampling rate, specifically the encoder resolution at this sampling rate is 1.57 rad/sec. We will discuss in more detail in last part of this chapter page 131. In the steady state, from Figure 3.17 we see that hardware measured data fit well when voltage input is below 4.7V and the deviation becomes bigger when voltage input is high.

![Figure 3.17: DC Motor Output \(\omega_s\) Response to High Voltage Step Input - Hardware and Decoupled Model](image)

Battery internal resistance (IR) may cause significant voltage drop when output
power is high. This behaves like saturation effect on motor maximum output torque. And we cannot find out detail specification for batteries we are using (Ni-MH AA Rechargeable Batteries from Amazon Basics). A similar product from Energizer shows the IR of one unit of Ni-MH AA battery is around be 0.05 - 0.15 Ω [40]. We uses 4 units as the power input to motor shield board, which could result in IR 0.2 - 0.6 Ω. Those problems will be further examined in the future.

3.5.3 Robot TITO LTI Model with Actuator Dynamics

In this section, we combine the ideas presented above in order to obtain a state space representation TITO LTI model for our differential-drive vehicle. This model, taken from [15], was used within [59] for inner-loop control design. It shall be used as the basis for our inner-loop control design as well. The TITO LTI model from motor voltages \((e_{aR}, e_{aL})\) to the wheel angular velocities \((\omega_R, \omega_L)\) can be visualized as shown within Figure 3.18. Note the cross-coupling introduced by the right-left motor torques \((\tau_R, \tau_L)\) at the input and a similar structure at the output.

![Figure 3.18: TITO LTI Robot-Motor Wheel Speed \((\omega_R, \omega_L)\) Dynamics - \(P(\omega_R, \omega_L)\)](image-url)
Figure 3.19: Differential-Drive Mobile Robot Dynamics

The associated fourth order TITO LTI state space representation \(^1\) is given by

\[
\dot{x} = Ax + Bu \quad y = Cx + Du \quad (3.28)
\]

\(^1\)Neglect friction, torque disturbance and distance between geometric center and mass center
where \( x = [v \quad \omega \quad i_{aR} \quad i_{aL}]^T \), \( y = [\omega_R \quad \omega_L]^T \), \( u = [e_{aR} \quad e_{aL}]^T \),

\[
A = \begin{bmatrix}
  -\frac{2\beta K_i^2}{mr^2} & 0 & \frac{K_t K_i}{mr} & \frac{K_t K_i}{mr} \\
  0 & -\frac{\beta K_i^2 d_w^2}{2mr^2} & \frac{K_t K_i d_w}{2La r} & -\frac{K_t K_i d_w}{2La r} \\
  \frac{-K_b K_i}{La r} & \frac{-K_b K_t d_w}{2La r} & \frac{-R_a}{La} & 0 \\
  \frac{-K_b K_i}{La r} & \frac{K_b K_t d_w}{2La r} & 0 & \frac{-R_a}{La}
\end{bmatrix}
\]

(3.29)

\[
B = \begin{bmatrix}
  0 \\
  0 \\
  \frac{1}{La} \\
  0 \\
  \frac{1}{La}
\end{bmatrix}
\]

(3.30)

\[
C = \begin{bmatrix}
  \frac{1}{r} & \frac{d_w}{2r} & 0 & 0 \\
  \frac{1}{r} & -\frac{d_w}{2r} & 0 & 0
\end{bmatrix}
\]

(3.31)

\[
D = \begin{bmatrix}
  0 \\
  0 \\
  0 \\
  0
\end{bmatrix}
\]

(3.32)

Here, \((i_{aL}, i_{aR})\) represent left and right motor armature currents, \(v\) is the vehicle’s translational velocity (directed along the direction \(\theta\)), \(\omega\) is the vehicle’s angular velocity, \((\omega_L, \omega_R)\) represent left and right vehicle wheel angular velocities, \((e_{aL}, e_{aR})\) represent left and right motor armature voltage inputs. The latter are the robot’s control inputs. Relevant system parameters are as follows: \(m\) is the vehicle mass, \(d_w\) is the distance between the wheels, \(r\) is the vehicle wheel radius, \(I_z\) is the vehicle’s moment of inertia, \(\beta\) represents a load-motor speed rotational damping constant, \(K_b\) represents a back EMF constant, \(K_t\) represents a torque constant, \(K_g\) represents the gearbox ratio of motor shaft-load system \(R_a\) represents armature resistance, and \(L_a\)
represents armature inductance (often negligibly small). It should be noted that differences in the motor properties is a practical concern. This has not been captured in the above model. It shall not be addressed within this thesis. Addressing such uncertainty will be the subject of future work.

Given the above, the associated transfer function matrix is given by

\[
P(\omega_R, \omega_L) = C(sI - A)^{-1}B + D = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}
\]

(3.33)

One can use Maple to verify that the symbolic transfer function matrix \( P(\omega_R, \omega_L) \) is symmetric and the diagonal entries are identical. This pattern is expected since the motors are identical.

**Nominal Functions.** The nominal values given in Table 3.2 shall be used to generate numerical values below. For these nominal parameter values, we obtain the following numerical TITO transfer function matrix:

\[
P(\omega_R, \omega_L) = \frac{186056(s + 2977.5)(s + 3.8722)}{(s + 2977.7)(s + 2977.2)(s + 4.2008)(s + 3.5914)}
\]

\[
14533s(s + 2981.1)
\]

(3.34)

When the inductance is neglected, we obtain the following low frequency approximations:

\[
P(\omega_R, \omega_L) \approx \begin{bmatrix} 62.411(s + 3.8675) & 4.8749s \\ 4.8749s & 62.411(s + 3.8675) \end{bmatrix}
\]

\[
\frac{1}{(s + 4.1951)(s + 3.5872)}
\]

(3.35)

This shows that the off diagonal elements are small at low frequencies.
System Approximations: $L_a$ small

In many applications, the armature inductance is very small. This is the case for the motors used within this research ($L_a \approx 265 \mu H$). When this is the case, the system poles can be approximated as follows:

\[
p_1 \approx -\left[ \frac{2K_g^2(K_bK_t + R_a\beta)}{R_a mr^2} \right] \quad p_2 \approx -\left[ \frac{K_g^2d_w^2(K_bK_t + R_a\beta)}{2R_aI_zr^2} \right] \quad p_3 \approx p_4 \approx -\frac{R_a}{L_a} \quad (3.36)
\]

The above was found using Maple by letting $L_a$ be small in the exact pole expressions. The above implies that the system is exponentially stable. This implies that any initial vehicle speed, angular velocity, armature current will be dissipated if no voltage is applied to the motor inputs.

For the nominal parameter values, the system poles are as follows:


The two high frequency (fast) poles are due to the armature inductance.

At this point, it is useful to point out zero information about specific transfer function entries. We note that $P_{11} = P_{22}$ possesses two zeros at:

\[
z_1 = \frac{-4K_g^2d_w^2(K_bK_t + R_a\beta)}{R_a r^2(d_w^2m + 4I_z)} \quad z_2 \approx p_3 = -\frac{R_a}{L_a} \quad (3.37)
\]

Similarly, when $L_a$ is small then $P_{12} = P_{21}$ possesses two zeros at:

\[
z_1 = 0 \quad z_2 \approx p_3 = -\frac{R_a}{L_a} \quad (3.38)
\]

Because the off diagonal elements possess a zero at dc (even when $L_a$ is not small), this implies that the $(\omega_R, \omega_L)$ vehicle-motor (robot) system is nearly decoupled at low frequencies. This suggests that if we have a low bandwidth control objective, then the system can be approximated by its diagonal elements. Moreover, a simple classical
(decentralized) controller should work fine. It can be shown (numerically) that generally speaking, this system has no transmission zeros. Why is this important? This is important because it roughly suggests that we may be able to use identical classical SISO (single-input single-output) controllers with sufficient lead and a sufficiently high bandwidth in order to reduce the sensitivity at low frequencies as much as we want. Here, the sufficient lead will ensure an infinite upward gain margin (ideally, of course). In practice, of course, the closed loop bandwidth will be limited by high frequency uncertainty, actuator bandwidth limitations, sensor bandwidth limitations, etc.

**Frequency Response Properties.** The singular values for the above system and the associated low frequency approximation are plotted within Figure 3.20 for the nominal parameter values given within Table 3.2. Note that the singular values at DC match one another. This is because from each input, the motor-vehicle \((\omega_R, \omega_L)\) system looks the same.

![Singular Values for TITO Model](image)

Figure 3.20: Robot Singular Values (Voltages to Wheel Speeds) - Including Low Frequency Approximation
The plot in Figure 3.20 suggests that the low frequency approximation (red, with a 20 dB/decade high frequency roll-off) is a good approximation for the system. The relatively high system gain at low frequencies will help achieve good low frequency command following and low frequency disturbance attenuation (in principle, without too much control action). The plot also suggests that as long as we keep the desired control bandwidth below say 2 rad/sec, then a large control gain will not be required in order to have good feedback properties.

To better examine the coupling in our \((\omega_R, \omega_L)\) system, we have plotted the frequency response in Figure 3.21. The figure clearly shows that the off-diagonal elements peak just below 1 rad/sec and that the coupling disappears at dc. This low frequency behavior, as well as the first order low frequency behavior of the diagonal elements, provides substantive motivation for a decentralized PI control law; i.e. the use of identical PI controllers for each motor.

![Figure 3.21: Robot Frequency Response (Voltages to Wheel Speeds) - Including Low Frequency Approximation](image)

Figure 3.21: Robot Frequency Response (Voltages to Wheel Speeds) - Including Low Frequency Approximation
Plant. The above discussion has focused on the \((\omega_R, \omega_L)\) system. It must be noted that strictly speaking, the \((\omega_R, \omega_L)\) system is not the plant for our inner-loop control system. Why? The inner-loop plant, strictly speaking, has \((v, \omega)\) as outputs since these are the variables that we wish to command! That is, we shall be specifying \((v_{ref}, \omega_{ref})\) reference commands to our inner-loop control system. The transfer function matrix for our \((v, \omega)\) system - the plant \(P\) - is as follows:

\[
P = P_{(v, \omega)} = \begin{bmatrix}
5014.7(s + 3.591)(s + 2978) & 5014.7(s + 3.591)(s + 2978) \\
61258(s + 4.201)(s + 2977) & -61258(s + 4.201)(s + 2977) \\
(s + 3.591)(s + 4.201)(s + 2977)(s + 2978)
\end{bmatrix}
\]

(3.39)

When the inductances are neglected, we obtain the following low frequency approximation:

\[
P = P_{(v, \omega)} \approx \begin{bmatrix}
1.6822(s + 3.587) & 1.6822(s + 3.587) \\
20.549(s + 4.195) & -20.549(s + 4.195)
\end{bmatrix} \frac{1}{(s + 3.587)(s + 4.195)}
\]

(3.40)

The above shows that unlike the \((\omega_R, \omega_L)\) system, the \((v, \omega)\) system (the plant) is not decoupled at low frequencies.

Plant SVD at DC. To better understand the coupling at dc, we perform a singular value decomposition (svd) for \(P\) at dc. Doing so yields the following:

\[
P(0) = P_{(v, \omega)}(0) = \begin{bmatrix}
0.401 & 0.401 \\
5.73 & -5.73
\end{bmatrix} = U\Sigma V^H
\]

(3.41)

\[
= \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix} \begin{bmatrix}
8.101 & 0 \\
0 & 0.5671
\end{bmatrix} \begin{bmatrix}
0.7071 & 0.7071 \\
0 & 0.7071
\end{bmatrix}^H
\]

(3.42)

where the unitary matrix \(U = [u_1 \ u_2]\) contains left singular vectors (output directions), \(\Sigma = diag(\sigma_1, \sigma_2)\) contains the maximum and minimum singular values, and
the unitary matrix $V = [v_1 \ v_2]$ contains right singular vectors (input directions). This svd clearly shows control coupling at dc and hence at low frequencies. More precisely, we see that the

- maximum singular value 8.101 (41.84 dB) is associated with the vehicle’s angular velocity $\omega$ ($u_1 = [\ 0 \ 1 \ ]^T$) and equal and opposite motor input voltages ($v_1 = [0.7071 \ -0.7071]^T$);

- minimum singular value 0.5671 (-11.35 dB) is associated with the vehicle’s translational velocity $v$ ($u_2 = [1 \ 0 \ ]^T$) and equal motor input voltages ($v_2 = [0.7071 \ 0.7071]^T$).

Given the above, it is natural to ask: Why is the maximum singular value associated with $\omega$ and the minimum with $v$? This follows from the fact that it is easier to turn than to move forward. We also note that the input voltage directions given above are exactly how we would expect a differential-drive system to generate $\omega$ and $v$. That is, the above svd corroborates our intuition about a differential-drive motor-vehicle system.

**Plant Singular Values.** The plant singular values and those of the low frequency approximation are shown within Figure 3.22. Here, the singular values at dc are not identical. This, fundamentally, is because we lose symmetry when we go from $(\omega_R, \omega_L)$ to $(v, \omega)$. More precisely, up above we showed (via svd at dc) that the maximum singular value at low frequencies is associated with rotation $\omega$ while the minimum singular value is associated with translation $v$.

**Plant Frequency Response.** In order to better understand and visualize the low frequency coupling issues associated with our plant, it is useful to examine the frequency response plot in Figure 3.23. Within this figure, the plant frequency response
Figure 3.22: Robot Plant Singular Values (Voltages to $v$ and $\omega$) - Including Low Frequency Approximation

is in blue and the frequency response for our low frequency approximation to the plant is in red. Again, we see that our low frequency approximation is a very good approximation for the plant. The approximation should be particularly good below about 50 rad/sec. Figure 3.23 also shows while our $(\omega_R, \omega_L)$ system is fairly decoupled at low frequencies, this is not the case for our $(v, \omega)$ plant system. While this suggests that a multivariable controller may be necessary for our plant, the work in this thesis shows that one can apply classical SISO control theory to our (fairly decoupled) $(\omega_R, \omega_L)$ system to indirectly control our $(v, \omega)$ plant system. This, and important issues discussed below, will receive greater attention in future work.
Comparisons Between Decoupled and Coupled Models. Recall the coupled TITO LTI vehicle-motor \((\omega_R, \omega_L)\) dynamical model in Chapter 3 [15]. We wish to examine whether the decoupled model is a valid approximation for the coupled model at low frequencies. Figure 3.24 and Figure 3.25 justify our assumption, and show that the decoupled model approximated the coupled model well at low frequencies. From this, we conclude that
Figure 3.24: Frequency Response for Vehicle-Motor - Coupled \((\omega_R, \omega_L)\) Model

The following is a good approximation to vehicle-motor \((\omega_R, \omega_L)\) plant:

\[
P_{[\omega_R, \omega_L]} \approx 16.246 \begin{bmatrix} 4.685 \\ s + 4.685 \end{bmatrix} \times I_{2 \times 2}
\]  \hspace{1cm} (3.43)

70
Plant Control Issues.[14] Given the above, it is natural to ask:

If the above \((v, \omega)\) system is the plant we really want to control, then why did we spend so much time on the \((\omega_R, \omega_L)\) system?

A quick and simple answer to this is that [nearly decoupled] \((\omega_R, \omega_L)\) system is much simpler than the [highly coupled] \((v, \omega)\) plant system.

To provide a more substantive answer to this, first note that the \((v, \omega)\) plant \(P = P(v, \omega)\) is related to the “angular wheel velocity plant” \(P(\omega_R, \omega_L)\) according to the following relationship:

\[
P = P(v, \omega) = MP(\omega_R, \omega_L)
\]

(3.44)

where \(M\) was defined above in equation (3.7) on page 40. From this, we can show that

if we design a good simple controller for \(P(\omega_R, \omega_L)\),

then it is likely to work well for \(P\).

Why is this? The next few pages provides support for this claim and addresses related issues that must be considered - especially if one takes the above implied \((\omega_R, \omega_L)\) control approach. Lets be specific. Suppose \(K(\omega_R, \omega_L) = k(s)I_{2 \times 2}\) where \(k(s)\) is a scalar SISO transfer function (e.g. PI controller as used below and in Chapter 4). Given this, it can be shown that if \(K(\omega_R, \omega_L)\) works well for \(P(\omega_R, \omega_L)\), then \(K = K(\omega_R, \omega_L)M^{-1}\) will work well for \(P\). Why is this?

- Critical Relationship: \((v, \omega)\) and \((\omega_R, \omega_L)\) Systems Have Identical Open Loop Singular Values. With \(K = K(\omega_R, \omega_L)M^{-1}\) as the controller for \(P\),
the new open loop transfer function matrix for the \((v, \omega)\) system is \(PK = PK_{(\omega_R, \omega_L)}M^{-1} = MP_{(\omega_R, \omega_L)}K_{(\omega_R, \omega_L)}M^{-1}\). More detailed discussion is within[14].

The following main results shown here for completeness. (Loop Singular Values for \((\omega_R, \omega_L)-(v, \omega)\) Systems are Identical. From the structure of \(P, K,\) and \(M\), it can be shown that

\[
\sigma_i[PK] = \sigma_i[P_{(\omega_R, \omega_L)}K_{(\omega_R, \omega_L)}]
\]

(3.45)

The above singular value result implies - modulo important (non-negligible) \(T_{ru}\) and \(T_{dy}\) issues to be pointed out below - that if \(K_{(\omega_R, \omega_L)}\) works well for \(P_{(\omega_R, \omega_L)}\), then \(K = K_{(\omega_R, \omega_L)}M^{-1}\) will work well for \(P = MP_{(\omega_R, \omega_L)}\).

- **Same \(L_o, S_o, T_o\) Singular Values.** More precisely, our main result in 3.45 implies that the two designs \((P, K)\) and \((P_{(\omega_R, \omega_L)}, K_{(\omega_R, \omega_L)})\) will possess the same singular values for the output/error open loop transfer function matrix \(L_o = PK\), output sensitivity \(S_o = (I + L_o)^{-1}\), and output complementary sensitivity \(T_o = I - S_o = L_o(I + L_o)^{-1}\) (i.e. at the plant output (or error)).

- **Same \(L_i, S_i, T_i\) Singular Values.** Since \(L_i = KP = PK = L_o\), it also follows that the designs will also possess the same singular values for the input/controls open loop transfer function matrix \(L_i = KP\), input sensitivity \(S_i = (I + L_i)^{-1}\), and input complementary sensitivity \(T_i = I - S_i = L_i(I + L_i)^{-1}\) (i.e. at the plant input (or controls)).

The above “almost proves” that if \((P_{(\omega_R, \omega_L)}, K_{(\omega_R, \omega_L)})\) is good, then \((P, K)\) will be good. It does prove that given the simple decentralized control structure selected,

if the singular values for \((L_o, L_i), (S_o, S_i), (T_o, T_i)\) are good for the \((\omega_R, \omega_L)\) system,

then they will be good for the \((v, \omega)\) system.
Given this, why then do we say “almost proves . . . ?” While the above is excellent, it
must be noted that the singular values for the transfer function matrices $T_{ru}$ (reference
to controls) and $T_{diy}$ (input disturbance to outputs) may differ for the two designs:
$(P_{(\omega_R,\omega_L)}, K_{(\omega_R,\omega_L)})$ and $(P, K)$. Why is this? This important (non-negligible) issue
is now explained.

- **$T_{ru}$ Differences for $(\omega_R, \omega_L)$ and $(v, \omega)$ Systems.** Simple multivariable sys-
tem algebra shows that

$$T_{ru} = K(I + PK)^{-1} = K(\omega_R, \omega_L)M^{-1}(I + MP_{(\omega_R, \omega_L)}K_{(\omega_R, \omega_L)}M^{-1})^{-1}$$

$$= K(\omega_R, \omega_L)(I + P_{(\omega_R, \omega_L)}K_{(\omega_R, \omega_L)})^{-1}M^{-1} \quad (3.46)$$

$$= T_{ru}(\omega_R, \omega_L)M^{-1}$$

This shows that the control response to reference commands can be different for the two loops.

- **$T_{diy}$ Differences for $(\omega_R, \omega_L)$ and $(v, \omega)$ Systems.** Similarly,

$$T_{diy} = (I + PK)^{-1}P = (I + MP_{(\omega_R, \omega_L)}K_{(\omega_R, \omega_L)}M^{-1})^{-1}MP_{(\omega_R, \omega_L)}$$

$$= M(I + P_{(\omega_R, \omega_L)}K_{(\omega_R, \omega_L)}M)^{-1}P_{(\omega_R, \omega_L)} \quad (3.47)$$

$$= MT_{diy}(\omega_R, \omega_L)$$

This shows that the output response to input disturbances can be different for the two loops.

In what follows, we shall plot frequency response plots to emphasize the proven simi-
larities and distinctions for the two systems under consideration: $(\omega_R, \omega_L)$ and $(v, \omega)$. While the above supports our focus above on $P_{(\omega_R, \omega_L)}$ rather than $P = P_{(v, \omega)}$ - modulo
the $T_{ru}$ and $T_{diy}$ issues illuminated above, there are still a few additional practical
implementation issues that must be pointed out. Before moving on, lets examine the
actual numerical $M$ and $M^{-1}$ for the system being considered in this chapter. From Table 3.2, we have $r = 0.05$ m and $d_w = 0.14$ m. Given this, it follows that we have

$$M = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d_w} & -\frac{r}{d_w} \end{bmatrix} = \begin{bmatrix} 0.025 & 0.025 \\ 0.3571 & -0.3571 \end{bmatrix} \quad (3.48)$$

$$M^{-1} = \begin{bmatrix} \frac{1}{r} & \frac{d_w}{2r} \\ \frac{1}{r} & -\frac{d_w}{2r} \end{bmatrix} = \begin{bmatrix} 20 & 1.4 \\ 20 & -1.4 \end{bmatrix} \quad (3.49)$$

What do these numbers tell us? The singular values of $M$ are $(0.5051, 0.0354)$. Those for $M^{-1}$ are $(28.2843, 1.9799)$. Using the relationships in equations (3.46) and (3.47), as well as the sub-multiplicative property of the $\mathcal{H}_\infty$ norm, yields the following:

$$\| T_{ru(v,\omega)} \|_{\mathcal{H}_\infty} \leq \| M^{-1} \|_{\mathcal{H}_\infty} \| T_{ru(\omega_R,\omega_L)} \|_{\mathcal{H}_\infty} = 21.40 \| T_{dy(\omega_R,\omega_L)} \|_{\mathcal{H}_\infty} \quad (3.50)$$

$$\| T_{dy(v,\omega)} \|_{\mathcal{H}_\infty} \leq \| M \|_{\mathcal{H}_\infty} \| T_{dy(\omega_R,\omega_L)} \|_{\mathcal{H}_\infty} = 0.7143 \| T_{dy(\omega_R,\omega_L)} \|_{\mathcal{H}_\infty} \quad (3.51)$$

The first inequality imply that we should be particularly concerned about the control response to reference commands for the $(v, \omega)$ system. It suggests that the $(v, \omega)$ system can have up to 21.40 times the control effort of the $(\omega_R, \omega_L)$ system (in the worst case). The second inequality tells us that the output response to input disturbances for the $(v, \omega)$ system will not be a concern if the $(\omega_R, \omega_L)$ system output response to input disturbances is satisfactory.

- **Practical Implementation Issues.** The following question is natural to ask:

If we are primarily interested in controlling $(v, \omega)$, then what can happen if we control $(\omega_R, \omega_L)$?
- **M is Well Known.** First, we note that the matrix $M$ is the key to the above discussion. If it were uncertain, then we would be particularly worried. However, $M$ is generally well known. Why? The matrix $M$ involves very well known system parameters (e.g. wheel radius $r$ and distance between wheels $d_w$). $M$ is well known. (We assume that $r$ and $d_w$ are well known constants that do not change over time. We assume, for example, that $r$ cannot decrease because of loss of air in the tires.) The issue here is therefore not uncertainty in $M$.

- **Accuracy of Measurements.** The issue here is what can we measure more accurately? $(\omega_R, \omega_L)$ or $(v, \omega)$? Within this thesis, we use optical wheel encoders (20 CPT) to measure $(\omega_R, \omega_L)$. From this, $(v, \omega)$ is computed (estimated). This is done using the relationships within equation (3.7). The fundamental point here is that if an IMU (inertial measurement unit) is used to measure $(v, \omega)$, then we may be able to get more accurate measurements and hence better inner-loop performance in comparison to the encoder-based $(\omega_R, \omega_L)$ approach taken within this thesis. Within the thesis, $\theta$ information directly obtained by the IMU built-in algorithm or using camera. This is done for our $(v, \theta)$ cruise control along a path outer-loop control law. The IMU is not used to get $v$, future work shall address how the IMU can be used to improve our inner-loop performance.
3.6 Uncertainty of Parameter

In this section, we will examine the parameter uncertainty of TITO model. Trade studies of variations in vehicle mass, vehicle moment of inertia, motor back EMF constant, motor torque constant and motor armature resistance has been discussed. Both frequency and time domains analysis are provided to give insights of single parameter uncertainty impact on the system.

3.6.1 Frequency Response Trade Studies

Mass $m$ Variations. Figure 3.26 shows that as the mass is increased, the dc gains do not change, the diagonal magnitudes get smaller at all frequencies, and the coupling at all frequencies decreases.

![Bode magnitude plots of the plant for variations of mass](image)

Figure 3.26: Bode Magnitude for Robot (Voltages to Wheel Speeds) - Mass Variations

Moment of Inertia $I_z$ Variations. Figure 3.27 shows that as the moment of inertia is increased, the dc gains do not change, the diagonal magnitudes get smaller at mid-range and high frequencies, and the coupling at low frequencies increases.
Figure 3.27: Bode Magnitude for Robot (Voltages to Wheel Speeds) - \( I \) Variations

**back EMF Constant \( K_b \) Variations.** Figure 3.28 shows that as the back EMF constant is increased, the dc gains get smaller, the diagonal magnitudes do not change at midrange and high frequencies. The off-diagonal magnitudes get smaller at low frequencies and do not change at midrange and high frequencies, and the coupling at low frequencies decreases.

Figure 3.28: Bode Magnitude for Robot (Voltages to Wheel Speeds) - \( K_b \) Variations

**Torque Constant \( K_t \) Variations.** Figure 3.29 shows that as the torque constant is
increased, the dc do not change, the diagonal magnitudes get larger at midrange and high frequencies. The off-diagonal magnitudes get smaller at low frequencies and get larger at midrange and high frequencies.

Figure 3.29: Bode Magnitude for Robot (Voltages to Wheel Speeds) - $K_t$ Variations

**Armature Resistance $R_a$ Variations.** Figure 3.30 shows that as the armature resistance $R_a$ is increased, the dc gains don’t change. The magnitudes of the diagonal elements start to get smaller with increasing $R_a$ at mid-range frequencies below 10000 rad/sec and do not change at high frequencies. The magnitudes of the off-diagonal elements get larger at low frequencies, start to get smaller with increasing $R_a$ at mid-range frequencies below 10000 rad/sec and do not change at high frequencies.
**3.6.2 Time Response Trade Studies**

**Mass** \( m \) **Variations.** Figure 3.31 contains the system wheel angular velocity responses to step voltage inputs for mass variations.

![Step response of the plant for variations of mass](image)

Figure 3.31: System Wheel Angular Velocity Responses to Step Voltages - Mass Variations

The figure shows that as the vehicle mass is increased, the system
• dc gain does not change
• settling time (bandwidth) increases (decreases)
• cross coupling decreases

The latter is a control-theoretic reason for reducing vehicle mass. (Energy savings, of course, is generally the primary reason given for wanting to reduce vehicle mass.)

**Moment of Inertia $I_z$ Variations.** Figure 3.32 contains the system wheel angular velocity responses to step voltage inputs for moment of inertia variations.

![Step response of the plant for variations of moment of inertia](image)

Figure 3.32: System Wheel Angular Velocity Responses to Step Voltages - Moment of Inertia Variations

The figure shows that as the vehicle moment of inertia is increased, the system

• dc gain does not change
• settling time (bandwidth) increases (decreases)
• cross coupling first decreases and then increases after certain range

**back EMF Constant $K_b$ Variations.** Figure 3.33 contains the system wheel angular velocity responses to step voltage inputs for back EMF constant variations.
The figure shows that as the motor back EMF constant is increased, the system

- dc gain gets smaller
- settling time (bandwidth) decreases (increases)
- cross coupling decreases

**Torque Constant \( K_t \) Variations.** Figure 3.34 contains the system wheel angular velocity responses to step voltage inputs for torque constant variations.
The figure shows that as the motor torque constant is increased, the system

- dc gain gets larger
- settling time (bandwidth) decreases (increases)
- cross coupling decreases

**Armature Resistance $R_a$ Variations.** Figure 3.35 contains the system wheel angular velocity responses to step voltage inputs for mass variations.
Figure 3.35: System Wheel Angular Velocity Responses to Step Voltages - Armature Resistance Variations

The figure shows that as the motor armature resistance is increased, the system

- dc gain gets smaller
- settling time (bandwidth) increases (decreases)
- cross coupling increases

The above trade studies give insight into the system being examined. More specifically, they are useful for understanding the impact of single parameter uncertainty. Future work can examine the structured uncertainties of multiple parameters.

3.7 Differential-Drive Robot Model with Dynamics on Ground

Till this point, the nominal plant model 3.43 looks pretty well, however it is not true when the following practical factors have been taken into consideration which is necessary in our hardware platform:

**Friction between vehicle tyres and ground.** Friction could significantly influence the model when torque constant $K_t$ of the motor is small. When $K_t$ is small, the
required torque to drive the vehicle will be increased a lot. Recall torque equation:

\[ \tau_s = K_t K_g i_a \]  

(3.52)

The torque required to overcoming friction will significantly increase armature current.

**Battery internal resistance** 4 AA Ni-MH rechargeable battery pack is used to provide power to motor shield. As discussed ahead page 59, the potential internal resistance(IR) could be as much as 0.6 Ω. This battery internal resistance could lead to significant voltage drop. Through off-ground high voltage step response, we observed this impact on our model. The step response from voltage to angular velocity at same PWM command of off-ground and on ground corroborates our assumption.

**Power dissipation of Motor Shield Board.** Motor shield board output impedance could also play important role when current is high. From datasheet of driver IC (TB6612FNG)[38] in motor shield board, we found out that output current could be saturated from drive IC see Figure 3.36.

![Figure 3.36: Target Characteristics of TB6612FNG](image.png)
Given discussion above, it motivate we tested the ground model and compare it with the off ground one. Vehicle has been given same PWM command (PWM=50 about 1.02V voltage), the result is shown in Figure 3.37.

![Off Ground / On Ground Output Response at 1V](image)

Figure 3.37: Off Ground and On Ground Step Response to 50 PWM Command (1V)

From the comparison, we can clear see the difference between off ground and on ground in step response, difference occurs in steady state as well as transient state. This somewhat support our previous assumptions, quantitative analysis will be addressed in future work. To deal with the model difference, we put forward on ground model.

3.7.1 Empirically Obtained Experimental Data for Ground Model.

By assumption of identical motor dynamics, we apply same voltage\(^2\) input to left and right motors. Through encoder data of angular wheel speed, we can obtain a step response data between voltage input and angular speed of two wheels \((\omega_R, \omega_L)\). We estimated linear translation speed by averaging \(v = \frac{(\omega_R + \omega_L)}{2}\). To fully capture the

\(^2\)Here, same voltage actually command by same PWM signal. Also notice that PWM based regulation may not reflects the true voltage as desired.
vehicle dynamics, we did this experiments for different voltage level. One sample for
PWM command at 50 (about 1V ideally) is shown in Figure 3.38.

Figure 3.38: Output $\omega_s$ Response to PWM 50 Voltage Step Input - Hardware and
Decoupled Model

3.7.2 Fitting Model to Collected Data.

After we got the step response data from last step, we use the same method as we
did for motor modeling. Obtained the settling time and dc gain of this step response
(input voltage, output angular speed of wheel) and then approximate it by first order
model with form $b\frac{a}{s+a}$. One sample of this is shown in Figure 3.39 and Figure 3.40.
Figure 3.39: On Ground Motor Fitting Left Motor - Hardware and Decoupled Model

Figure 3.40: On Ground Motor Fitting Right Motor - Hardware and Decoupled Model
These two figures shows that two motors behave similar as off ground case. The fitting first order model is pretty accurate locally (model fits certain voltage input). Then we proceed to next step, by average left and right angular speed: \( \omega_v = \frac{\omega_R + \omega_L}{2} \), where \( \omega_v \) denotes the equivalent angular speed of the longitudinal translation speed \( \omega_v = \frac{v}{r} \). Approximate numerical first order transfer function distribution (in terms of dc gain and dominant pole) is shown in Figure 3.41

![DC Gain Distribution](image1)

![Dominate Pole Distribution](image2)

Figure 3.41: On Ground Approximate Transfer Function Fitting Model Distribution of V1

Final step is to take average of dc gain distribution and dominant pole and get nominal on ground plant for V1.

\[
\frac{\omega_v}{e_a} = 11.637 \frac{0.9745}{s + 0.9745}
\]  

(3.53)

The following is a approximation to the on ground inner-loop \((\omega_R, \omega_L)\) plant for V1:

\[
P_{[\omega_R, \omega_L]} \approx 11.637 \left[ \frac{0.9745}{s + 0.9745} \right] \times I_{2 \times 2}
\]  

(3.54)
This plant take the friction and battery internal resistance effect implicitly, and fit well with hardware experimental data. From result of V1 (ETT 1) the model fits well with hardware data.

3.7.3 On Ground Nominal Model.

From above, we see on ground approximation model 3.54 fits well with hardware for V1. The following question immediately come after: does this model applies to all other 8 ETTs? Thus we tested all other 8 ETTs with same procedure and got nominal on ground model for 9 ETTs see Figure 3.42.

The following is approximation to the on ground inner-loop ($\omega_R, \omega_L$) plant:

$$P_{[\omega_R, \omega_L]} \approx 11.268 \left[ \frac{1.159}{s + 1.159} \right] \times I_{2\times2} \quad (3.55)$$

To verify this nominal model, we compared the step response and frequency response of individual ETT with the nominal model see Figure 3.43 and Figure 3.44.
Figure 3.43: Step Response of Nominal Model with other ETT Model

Figure 3.44: On Ground Nominal Model with other ETT Model

This complete modeling of on ground model. From the nominal model comparison between different ETT, we see that there is noticeable difference. This difference is inherently due to hardware difference because our thunder tumble is low cost and with loose requirement on quality control.
3.8 Summary and Conclusion

In this chapter, mathematical modeling of a differential-drive mobile robot was discussed. First motor dynamic with gearbox has been carefully developed, hardware data matched simulation of nominal model at low voltage input. A TITO LTI model incorporating motor dynamics was carefully examined. For the model of motor dynamics, gearbox and speed damping constant has been included. Then, we put forward our decoupled on ground model considering battery internal resistance, motor shield power dissipation and friction between tyres and ground. All 9 ETTs has been test and follows with a nominal ground model. At last we averaged the 9 nominal model of each ETTs, and got on ground nominal model as basis for future design.
Chapter 4

SINGLE VEHICLE CASE STUDY FOR A LOW-COST MULTI-CAPABILITY DIFFERENTIAL-DRIVE ROBOT: ENHANCED THUNDER TUMBLE (ETT)

4.1 Introduction and Overview

In this chapter, we describe how to design controller for enhanced thunder tumbler. Based on decoupled on ground model we put forward in last chapter, inner-loop control has been designed and relevant control parameter trade off has been done. Then two (2) outer-loop control law types are presented, analyzed and implemented in hardware: (1) \((v, \theta)\) cruise control along a path (using encoders and IMU) [15], (2) \((\Delta x, \theta)\) separation-direction control (using encoders, IMU/camera, and ultrasonics) [3], [6]. The underlying theory for each control law is explained and justified. Finally, the results from hardware demonstrations are presented and discussed.

4.2 Inner-Loop Speed Control Design and Implementation

In this section, we describe \((v, \omega)\) and \((\omega_R, \omega_L)\) inner-loop control design for our differential drive Thunder Tumbler. For this basic inner-loop control modality, the angular velocity of each vehicle wheel is estimated/approximated by exploiting the encoder pulse counts picked up by the two wheel encoders during a \(T\) sec sampling window. This results in the following estimate for \((\omega_R, \omega_L)\):

\[
\begin{align*}
\omega_r &\approx \frac{2\pi n_r}{20T} = 3.142 \ n_r \ (rad/sec) \\
\omega_l &\approx \frac{2\pi n_l}{20T} = 3.142 \ n_l \ (rad/sec)
\end{align*}
\]  

\(\bullet \ T = 0.1 \text{ sec (100 ms or 10 Hz) was the chosen sampling (and actuation) time.}

Future work will examine the benefits of a faster sampling/actuation frequency.
- $n_r$ is the number of counts measured by the optical encoder (photo-interrupter) on the right wheel (with 20 counts per rotation$^1$),

- $n_l$ is the number of counts measured by the optical encoder (photo-interrupter) on the left wheel (with 20 counts per rotation).

We note that as the number of black-white stripe pairs used on a wheel is increased, then the constant 3.142 would decrease. The vehicle translational and rotational velocities ($v, \omega$) are then estimated from the above ($\omega_r, \omega_l$) estimates as follows:

\[
\begin{bmatrix}
  v \\
  \omega
\end{bmatrix}
= M
\begin{bmatrix}
  \omega_R \\
  \omega_L
\end{bmatrix}
M = \begin{bmatrix}
  \frac{r}{2} & \frac{r}{2} \\
  \frac{r}{d_w} & -\frac{r}{d_w}
\end{bmatrix}
M^{-1} = \begin{bmatrix}
  \frac{1}{r} & \frac{d_w}{2r} \\
  \frac{1}{r} & -\frac{d_w}{2r}
\end{bmatrix}
\]

(4.2)

and

\[
v = \left( \frac{r(\omega_R + \omega_L)}{2} \right) \approx 0.1571 \left( \frac{n_r + n_l}{2} \right) \text{ m/sec}
\]

(4.3)

\[
\omega = \left( \frac{r(\omega_R - \omega_L)}{d_w} \right) \approx 1.122 (n_r - n_l) \text{ rad/sec}
\]

(4.4)

where $r = 0.05$ m is the radius of each wheel and $d_w = 0.14$ m is the distance between the rear wheels. The above suggests that a single missed count could result in a 0.157 m/sec translation velocity error or a 1.122 rad/sec rotational velocity error! As the number of magnets used on a wheel is increased, these errors would decrease. This analysis can be used to select the number of black-white stripe pairs needed in order to achieve a desired resolution. Future work will leverage this analysis. (On our vehicle, we could not make stable measurement for higher resolution encoder code disk because the photo-interrupter is sensitive to distance and this distance between the sensor and wheel changes severely during rotation because of low-cost chassis.)

---

$^1$This does not factor in the possibility of missed counts.
Control Design: PI with One Pole Roll-Off and Command Pre-filter. Based on the simple (decoupled first order) LTI model obtained in the previous section

\[ P_{inner} = \frac{\omega}{c} = 11.268 \left[ \frac{1.159}{s + 1.159} \right] \quad (4.5) \]

we design a PI controller with roll-off and pre-filter. The controller has the form (PI plus roll-off):

\[ K_{inner} = \frac{g(s + z)}{s} \left[ \frac{40}{s + 40} \right] \quad (4.6) \]

This \( K_{inner} \) will be used to drive each DC motor on the vehicle. Next, we use the PI controller portion \( \frac{g(s+z)}{s} \) (i.e. neglect the high frequency roll off) to place the dominant closed poles near

\[ s = -2.50 \pm j1.21 \quad (\zeta = 0.90, \ \omega_n = 2.78 \text{ rad/sec}). \quad (4.7) \]

Doing so will yield, assuming negligible impact of roll off, a settling time near 2 second and essentially no overshoot \( (100e^{-\zeta_{pi}\sqrt{1-\zeta^2}} \approx 0.15\%) \). The desired (approximate) closed loop characteristic equation is

\[ \Phi_{desired}(s) \approx s^2 + 5s + 7.716. \quad (4.8) \]

Given the above, we have \( P_{inner}K_{inner} \approx 11.268 \left[ \frac{1.159}{s + 1.159} \right] \left[ \frac{g(s+z)}{s} \right] \). This yields

\[ \Phi_{actual}(s) \approx s(s + 1.159) + 13.05g(s + z) = s^2 + (1.159 + 13.05g)s + 13.05gz. \quad (4.9) \]

Equating this to the desired characteristic equation gives us the two equations \( (1.159 + 13.05g = 5, 13.05gz = 7.716) \) in the two unknowns \( (g, z) \). Solving then yields:

\[ g \approx 0.2942 \quad z \approx 2.0091. \quad (4.10) \]

A reference command pre-filter

\[ W_{inner} = \frac{z}{s + z} \quad (4.11) \]
will ensure that the overshoot to a step reference command approximates that dictated by the second order theory. That is, we obtain the following single channel (SISO) map form commanded wheel speed to actual wheel speed:

\[
T_{ry \text{wheel speeds}} = W_{inner} \left[ \frac{P_{inner} K_{inner}}{1 + P_{inner} K_{inner}} \right] \approx \frac{7.716}{s^2 + 5s + 7.716}.
\] (4.12)

Figure 4.1: Visualization of \((v, \omega)\) and \((\omega_R, \omega_L)\) Inner-Loop Control

The inner-loop control system can be visualized as shown in Figure 4.1. \((v, \omega)\) are commanded but not directly measured. Within Figure 4.1, the matrix \(M\) is a 2 \times 2 vehicle-wheel speed map that relates the vehicle translational-rotational velocities \((v, \omega)\) to the wheel angular velocities \((\omega_R, \omega_L)\); i.e. see equation (4.2) (page 93). Although only the wheel encoder count information is fed back within the physical inner-loop hardware implementation, the system outputs \(v\) and \(\omega\) were estimated (computed) using wheel encoder counts in accordance with equations (4.3)-(4.4)

**Reference to Output** \(T_{ry} (v, \omega)\) **Map.** From Figure 4.1, it follows that one can use the relationships in equation (4.2) to get the model-based closed loop map from references \((v_{ref}, \omega_{ref})\) to outputs \((v, \omega)\). Doing so yields the following TITO LTI
(nearly decoupled at low frequencies) map:

\[ T_{ry} = MPK(I + PK)^{-1}WM^{-1} \approx \begin{bmatrix} \frac{7.716}{s^2 + 5s + 7.716} \end{bmatrix} I_{2\times2} \]  \hspace{1cm} (4.13)

where \( P \approx P_{\text{inner}}I_{2\times2} \) is a TITO LTI nearly decoupled system at low frequencies (see Chapter 3 and results above) and \( K = K_{\text{inner}}I_{2\times2} \) is a diagonal inner-loop controller.

**Inner-Loop Open Loop Singular Values:** \((v, \omega)\) and \((\omega_R, \omega_L)\) Systems. Recalling the plant control issues discussion on pages (71)-(75), we plot the open loop singular values for \( PK \) and \( MPKM^{-1} \) in Figure 4.2.

Their singular values are identical. This is expected from a simple algebraic analysis - one exploiting the structure of \( M \) and the fact that \( PK \) is symmetric (see discussion on pages (71)-(75)).

![Open Loop Singular Values](image)

**Figure 4.2:** \( PK \) and \( L_o = MPKM^{-1} \) Singular Values

The main point of Figure 4.2, and the discussion on pages (71)-(75), is that

- It does not really matter whether we design for \((v, \omega)\) or \((\omega_R, \omega_L)\) - both system will possess the same closed loop properties except for control action differences. This is not a negligible difference. Why? Control effort, like energy usage, is very important!
• That is, they will possess the same loop $L$, sensitivity $S = (I + L)^{-1}$, and complementary sensitivity $T = I - S$ singular values.

• Recall that since $PK = KP$ (see discussion on pages (71)-(75)), it follows that the singular values at the outputs/errors are the same as those at the controls/inputs for the two closed loop systems (i.e. $(\omega_R, \omega_L)$ and $(v, \omega)$). This implies that

$$
\sigma_k[L_o] = \sigma_k[L_i] \quad \sigma_k[S_o] = \sigma_i[S_i] \quad \sigma_k[T_o] = \sigma_i[T_i] \quad k = 1, 2\ldots(4.14)
$$

for both closed loop systems; (i.e. $(\omega_R, \omega_L)$ and $(v, \omega)$). Recall that:

– For the $(\omega_R, \omega_L)$ system, $L_o = PK = KP = L_i$.

– For the $(v, \omega)$ system, $L_o = MPKM^{-1}$ and $L_i = KM^{-1}MP = PK$.

For either system, $\sigma_k[L_o] = \sigma_k[L_i] \ (k = 1, 2\ldots)$. While obvious for the first $(\omega_R, \omega_L)$ system, the proof for the second $(v, \omega)$ system requires using the structure of $M$ (see equation (4.2) on page 93) and the fact that $PK$ is symmetric (see discussion on pages (71)-(75)).

**Sensitivity Singular Values (Decoupled Model).** The sensitivity singular values (at outputs/controls) for either system ($(v,\omega)$ or $(\omega_R, \omega_L)$) are plotted in Figure 4.3. As expected, the singular values are identical for the two systems: $(\omega_R, \omega_L)$ and $(v, \omega)$. Moreover, Figure 4.3 shows that the system will possess good low frequency command following as well as nominal stability robustness properties (i.e. little sensitivity peaking). Again, these properties will hold for both the $(\omega_R, \omega_L)$ and the $(v, \omega)$ closed loop systems.

**Complementary Sensitivity Singular Values (Decoupled Model).** The complementary sensitivity singular values (at outputs/controls) for system $(\omega_R, \omega_L)$ are
Figure 4.3: $S_o = (I + L_o)^{-1} = S_i$ Singular Values - Using Decoupled Model

plotted in Figure 4.4. As expected, the singular values are identical for the two systems: $(\omega_R, \omega_L)$ and $(v, \omega)$.

Figure 4.4: $T_o = L_o(I + L_o)^{-1} = T_i$ Singular Values - Using Decoupled Model

Figure 4.4 shows that the system will possess good low frequency command following as well as high frequency noise attenuation.

**Reference to Control Singular Values (Decoupled Model).** The reference to control singular values are shown in Figures 4.5-4.6. The latter shows the utility of
the command pre-filter for reducing control effort.

Figure 4.5: $T_{ru}$ Singular Values (No Pre-filter) - Using Decoupled Model

Figure 4.6: $T_{ru}W$ Singular Values (with Pre-filter) - Using Decoupled Model

Figure 4.6 shows that output disturbances with frequency content near 1-5 rad/sec can be maximally amplified with respect to the controls (by a factor of 1.48 or 3.4 dB), if they occur in the worse case direction.

Figure 4.7 shows the singular values for $T_{ru}$ (unfiltered) for the $(v, \omega)$ system.
Input Disturbance to Output $T_{diy}$ Singular Values. The input disturbance to output singular values are shown in Figures 4.8. The plot shows that input disturbances near 3 rad/sec will be maximally amplified (about 8 dB, if they occur in the worse case direction).

Figure 4.8: $T_{diy}$ Singular Values - Using Decoupled Model

Figure 4.9 shows the singular values for $T_{diy}$ for the $(v, \omega)$ system.
Step Response Analysis Using Decoupled Model: Output Responses \((\omega_R, \omega_L)\).

filtered step reference time responses is shown within Figure 4.10.

The associated decoupled unfiltered step reference time responses is shown within Figure 4.11.
Figure 4.11: Inner-Loop $[\omega_R, \omega_L]$ Unfiltered Step Response - Using Decoupled Model

**Step Response Analysis with Decoupled Model: Control Responses.** The plots in Figures 4.12-4.13 show the control (motor voltage) responses to a step reference command - filtered and unfiltered. The latter shows how the reference command pre-filter lessens control effort. Both plots show only slight cross coupling - a consequence of our well designed inner-loop control system.
Figure 4.12: Control Response to Step Command (with Pre-filter) - with Decoupled Model

Figure 4.13: Control Response to Unfiltered Step Command - with Decoupled Model
4.2.1 Frequency Domain $(g,z)$ Trade Studies

In what follows, $L = PK = KP$ denotes the open loop transfer function matrix, $S = (I + L)^{-1}$ denotes the closed loop sensitivity transfer function matrix, $T = L(I + L)^{-1}$ denotes the closed loop complementary sensitivity transfer function matrix, $KS$ denotes the transfer function matrix from (unfiltered) reference commands to controls (motor voltages), and $SP$ denotes the transfer function matrix from input disturbances to the wheel speeds. We now examine trade studies for gain $g$ and zero $z$ variations. Unless stated otherwise, all plots presented are for the $(\omega_R, \omega_L)$ system and not the $(v, \omega)$ system. When the $(v, \omega)$ system is being considered, it will be explicitly stated.

**Open Loop.** Figures 4.14-4.15 show the singular values of $L = PK$ for specific $(g, z)$ variations.

![Open Loop Singular Values](image)

**Figure 4.14:** Singular Values for $L \ (g=0.10, 0.30, 0.50, 0.70; \ z=2)$
Figure 4.15: Singular Values for $L$ ($g=0.50; z=1, 2, 3, 4$)

The following observations follow from Figures 4.14-4.15:

- Increasing $g$ increases singular values of $L$ at all frequencies.
- Increasing $z$ will increase singular values of $L$ at low frequencies (because it increases effective gain at low frequencies), but it has no impact at high frequencies.
- In Figure 4.14, we see that increasing $g$ impacts the crossovers proportionately.
- In Figure 4.15, we see that increasing $z$ doesn’t impact the crossovers much.

**Sensitivity.** Figures 4.16-4.17 contain sensitivity singular values for specific $(g, z)$ variations.
Figure 4.16: Singular Values for $S (g=0.10, 0.30, 0.50, 0.70; z=2)$

Figure 4.17: Singular Values for $S (g=0.50; z=1, 2, 3, 4)$

From Figures 4.16-4.17, we make the following observations:

- Increasing $g$ results in smaller sensitivity at low frequencies and a slightly larger peak sensitivity.
• Increasing $z$ results in smaller sensitivity at low frequencies but increases peak sensitivity slightly.
• peak sensitivities do not change much with increasing $g$
• peak sensitivities do not change much with increasing $z$.

**Complementary Sensitivity.** Figures 4.18-4.19 contain complementary sensitivity singular values for specific ($g, z$) variations.

![Complementary Sensitivity Graph](image)

Figure 4.18: Singular Values for $T$ ($g=0.10, 0.30, 0.50, 0.70; z=2$)
Figure 4.19: Singular Values for $T$ ($g=0.50; z=1, 2, 3, 4$)

From Figures 4.18-4.19, we make the following observations:

- Increasing $g$ will result in a larger bandwidth (but worse high frequency noise attenuation and larger peak complementary sensitivity; a tradeoff must be made).
- Increasing $z$ will result in slightly larger bandwidth and a larger peak complementary sensitivity. High frequency noise attenuation is the same for different $z$ values.
- Peak sensitivities increase with increasing $g$.
- Peak complementary sensitivities increase slightly with increasing $z$.

**Reference to Control (Unfiltered).** Figures 4.20-4.21 contain (unfiltered) reference to control singular values for specific $(g, z)$ variations. As the plots above, these plots are for the $(\omega_R, \omega_L)$ system. As such, they tell us what control responses result from $(\omega_R, \omega_L)$ commands - not from $(v, \omega)$ commands. This is addressed below.
Figure 4.20: Singular Values for $T_{ru}$ (g=0.10, 0.30, 0.50, 0.70; z=2) - ($\omega_R, \omega_L$) Commands

Figure 4.21: Singular Values for $T_{ru}$ (g=0.50; z=1, 2, 3, 4) - ($\omega_R, \omega_L$) Commands

From Figures 4.20 -4.21, we make the following observations when ($\omega_R, \omega_L$) commands are issued to our inner-loop control system:
• Increasing $g$ or $z$ increases the peak $T_{ru}$ at all except low frequencies.

• Increasing $g$ increases peak $T_{ru}$.

• Increasing $z$ slightly increases peak $T_{ru}$.

For completeness, Figures 4.22-4.23 contain (unfiltered) reference to control singular values for specific $(g, z)$ variations. These plots are for the actual plant $P$. As such, they tell us the control response to a $(v, \omega)$ commands - as we shall be giving in a practical $(v, \omega)$ inner-loop control implementation (see Chapter 4).

![Figure 4.22: Singular Values for $KSM^{-1}$ $(g=0.10, 0.30, 0.50, 0.70; z=2)$ - $(v, \omega)$ Commands](image)

110
Figure 4.23: Singular Values for $KSM^{-1}$ ($g=0.50; z=1, 2, 3, 4$) - $(v, \omega)$ Commands

From Figures 4.22-4.23, we make the following observations when $(v, \omega)$ commands are issued to the inner-loop control system:

- peak controls will generally be larger for $(v, \omega)$ commands versus $(\omega_R, \omega_L)$ commands (see Figures 4.20-4.21)

- peak controls will increase significantly with increasing $g$ or slightly with increasing $z$

While the above suggests that control saturation can be an issue when large unfiltered $(v, \omega)$ commands are issued, it must be noted that a reference command pre-filter (to low pass filter the derivative action of the zero in our PI controller) can help with this.

**Reference to Control (Filtered).** As discussed above, a command pre-filter can significantly help with control action. We therefore use a command pre-filter $W = \frac{s}{s + z}$ on each reference command. Figures 4.24 -4.25 contain (filtered) reference to control singular values for specific $(g, z)$ variations. As many of the prior plots, these plots
are for the \((\omega_R,\omega_L)\) system. As such, they tell us what control responses result from \((\omega_R,\omega_L)\) commands. They do not tell us about control responses from \((v,\omega)\) commands. This is addressed below.

Figure 4.24: Singular Values for \(W \cdot T_{ru} \ (g=0.10, 0.30, 0.50, 0.70; z=2)\) - \((\omega_R,\omega_L)\) Commands

\[
\begin{array}{c}
\text{Frequency (rad/s)} \\
\text{Singular Values (dB)}
\end{array}
\]

Figure 4.25: Singular Values for \(W \cdot T_{ru} \ (g=0.50; z=1, 2, 3, 4)\) - \((\omega_R,\omega_L)\) Commands

112
From Figures 4.24-4.25, we make the following observations when \((\omega_R, \omega_L)\) commands are issued to our inner-loop control system:

- Increasing \(g\) or \(z\) increases the size of \(WT_{ru}\) at all but low frequencies.
- Increasing \(g\) increases the peak \(WT_{ru}\) only slightly.
- Increasing \(z\) increases the peak \(WT_{ru}\), but it does not impact \(WT_{ru}\) at low frequencies.
- Peak \(W \cdot T_{ru}\) increases slightly as \(g\) increases.
- Peak \(W \cdot T_{ru}\) increases with increasing \(z\).

The above plots suggest that overshoot and saturation due to filtered \((\omega_R, \omega_L)\) commands reference commands should not be too much of an issue - unless, of course, very large reference commands are issued to the inner-loop control system.

**Input Disturbance to Output** \(T_{diy}\). Figures 4.26-4.27 contain input disturbance to control singular values for specific \((g, z)\) variations. As many of the plots above, these plots are for the \((\omega_R, \omega_L)\) system. As such, they tell us what \((\omega_R, \omega_L)\) responses result from input (voltage) disturbances. They do not tell us about \((v, \omega)\) responses. This is addressed below.

Figures 4.26-4.27, contain the singular value plots for \(T_{diy}\) for specific \((g, z)\) variations. Figures 4.26-4.27, we make the following observations:
Figure 4.26: Singular Values for $T_{diy}$ ($g=0.10, 0.30, 0.50, 0.70; z=2$) - $(\omega_R, \omega_L)$ Responses

Figure 4.27: Singular Values for $T_{diy}$ ($g=0.50; z=1, 2, 3, 4$) - $(\omega_R, \omega_L)$ Responses

From Figures 4.26 -4.27, we make the following observations:

- peak $T_{diy}$ decreases with increasing $g$ ($z$ has little impact on peak)
• increasing $g$ reduces $T_{d_iy}$ at all frequencies except at high frequencies

• increasing $z$ reduces $T_{d_iy}$ at low frequencies

• frequency at which peak $T_{d_iy}$ occurs increases with increasing $g$ (also with increasing $z$ but to a lesser extent)

For completeness, Figures 4.28-4.29 contain input disturbance $d_i$ to output $y = [v \ \omega]^T$ singular values for specific $(g, z)$ variations. These plots are for the actual plant $P$. As such, they tell us about the $(v, \omega)$ responses to input (voltage) disturbances $d_i$.

![MSP Singular Values](image)

**Figure 4.28: Singular Values for MSP ($g=0.50; z=1, 2, 3, 4$) - $(v, \omega)$ Responses**
Figure 4.29: Singular Values for MSP ($g=0.50; z=1, 2, 3, 4$) - $(v, \omega)$ Responses

From Figures 4.28-4.29, we make the following observations about the closed loop system containing the actual actual $(v, \omega)$ plant $P$:

- peak $T_{dy}$ has improved for actual $(v, \omega)$ plant versus $(\omega_R, \omega_L)$ system (see Figures 4.26 -4.27)
- peak $T_{dy}$ decreases with increasing $g$ ($z$ has little impact on peak)
- increasing $g$ reduces $T_{dy}$ at all frequencies except at high frequencies
- increasing $z$ reduces $T_{dy}$ at low frequencies
- frequency at which peak $T_{dy}$ occurs increases with increasing $g$ (also with increasing $z$ but to a lesser extent)

The above patterns suggests that while the actual $(v, \omega)$ plant will exhibit a worse $T_{ry}$, it will exhibit a better $T_{dy}$ than the $(\omega_R, \omega_L)$ system.
4.2.2 Time Domain \((g,z)\) Trade Studies

Output and Control Responses to Step Reference Commands for different \((g,z)\) values. Figures 4.32-4.33 contain output and control responses to unfiltered step reference commands for specific \((g,z)\) variations.

\[
\begin{align*}
\text{Figure 4.30: Output Response to Step Command} & \quad (g = 0.10, 0.30, 0.50, 0.70; z = 2)
\end{align*}
\]
Figure 4.31: Output Response to Step Command ($g = 0.50; z = 1, 2, 3, 4$)

Figure 4.32: Control Response to Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)
Figure 4.33: Control Response to Step Command \((g = 0.50; z = 1, 2, 3, 4)\)

From the above output and control responses to unfiltered step reference commands (Figures 4.32-4.33), we obtain the following observations:

- faster settling time results from increasing \(g\) or \(z\)
- increasing \(g\) will result in a larger overshoot
- increasing \(z\) will result in a slightly more overshoot within this range of variation
- increasing \(g\) will result in larger and faster control action, little impact with increasing \(z\) within this range of variation

Figures 4.36-4.37 contain output and control responses to filtered step reference commands for specific \((g, z)\) variations.
Figure 4.34: Output Response to Step Command \((g = 0.10, 0.30, 0.50, 0.70; z = 2)\)

Figure 4.35: Output Response to Step Command \((g = 0.50; z = 1, 2, 3, 4)\)
Figure 4.36: Control Response to Filtered Step Command \( g = 0.10, 0.30, 0.50, 0.70; z = 2 \)

Figure 4.37: Control Response to Filtered Step Command \( (g = 0.50; z = 1, 2, 3, 4) \)

From Figures 4.36-4.37, we obtain the following observations:

- increasing \( g \) or \( z \) will result in larger and faster control action
- command pre-filter helps with control action.
4.2.3 Inner-Loop Experimental Result

In this part, we will check our design for inner loop with hardware experimental result. We examined design parameters we discussed above ($g = 0.29$, $z = 2.01$) with simulation and hardware.

![Decoupled Inner-Loop Step Response](image)

Figure 4.38: Output Response to filtered Step Command ($\omega_{R_{ref}} = 10$, $\omega_{L_{ref}} = 10$)

For completeness, output response to $(v, \omega)$ system is shown in Figure 4.39
Figure 4.39: Output Response to filtered Step Command \((v_{ref} = 0.5, \omega_{ref} = 0)\)

Figure 4.40: Control Response to filtered Step Command \((\omega_{Rref} = 10, \omega_{Lref} = 10)\)

From the above output and control responses to unfiltered step reference commands (Figures 4.38-4.40), we obtain the following observation. Output response of hardware follows simulation and reach steady state in almost 2 seconds, overshoot about 8% is
observed. Control effort from hardware is close to simulation result in steady state, however shows much deviation at beginning. The difference comes from following factors:

- **Deadzone Effect from Stiction.** When vehicle begins to start, it need overcome static friction which is much higher than the rolling friction experienced when it runs. This cause about 0.2 seconds delay in system response, which will cause integration term build much higher than it should be and oscillation will be observed as expected.

- **Encoder Resolution.** To improve the encoder resolution, we implemented average filter in hardware. However, it still have strong impact when reference command is relative slow. This will cause oscillation in control effort and inevitable steady state error.
4.3 Outer-Loop Control Design and Implementation

4.3.1 Outer-Loop 1: \((v, \theta)\) Cruise Control Along Line - Design and Implementation

In this section, we examine \((v, \theta)\) cruise control along a line/curve. This outer-loop control law can be visualized as shown in Figure 4.41. Here, \((v, \theta)\) are commanded.

\[ v \] is calculated based on wheel encoders; see equation (4.3) on page 93. For cruise control along a line, \(v_{ref} = \text{constant}, \theta_{ref} = 0\) are commanded. For cruise control along a line, \(\theta\) is directly measured by the IMU.

The use of a proportional gain controller is justified because the map from the references \(v_{ref}\) and \(\omega_{ref}\) to the actual speeds \(v\) and \(\omega\) looks like a diagonal system \(\text{diag}(\frac{a}{s+a}, \frac{b}{s+b})\) (at low frequencies). This is a consequence of a well-designed inner-loop (see above). The outer-loop \(\theta\) controller therefore sees \(\frac{b}{s(s+a+b)}\). From classical root locus ideas, a proportional controller is therefore justified - provided that the gain is not too large. If the gain is too large, oscillations will be expected in \(\theta\). A PD controller with roll off would help with this issue.

Figures 4.42-4.43 show frequency responses for \(T_{\theta_{ref}, \theta}\) for proportional and PD outer-loop control laws. Figure 4.44-4.45 show the corresponding responses to an initial condition \(\theta_o = 0.1\) rad. The responses corroborate what was pointed out above, namely that a sufficiently large gain proportional control law can result in peaking
while a PD control law has little peaking.

Figure 4.42: $T_{\theta_{ref}\theta}$ Frequency Response for $\theta$ Outer-Loop (P Control)

Figure 4.43: $T_{\theta_{ref}\theta}$ Frequency Response for $\theta$ Outer-Loop (PD control, $K_d = 1$)

**CODE.** The Raspberry Pi C code used for implementing our $(v, \theta)$ outer-loop cruise control along a line control law can be found within Appendix A on page 242. This outer-loop uses the $(\omega_R, \omega_L) - (v, \omega)$ inner-loop control law that has been implemented using the Arduino code within Appendix C on page 294.
Figure 4.44: Cruise Control $\theta$ Response Using P Control ($\theta_o = 0.1$ rad)

Figure 4.45: Cruise Control $\theta$ Response Using PD Control ($\theta_o = 0.1$ rad)
4.3.2 Outer-Loop 2: Separation-Direction ($\Delta x, \theta$) Control - Design and Implementation

In this section, separation-direction ($\Delta x, \theta$) outer-loop control is discussed. Within [3], [6], vehicle separation modeling and longitudinal platoon control is presented. The ideas presented within [3], [6] motivate the PD ultrasonics-encoder-IMU-based separation control laws used for the separation-direction ($\Delta x, \theta$) outer-loop control within this thesis. The ideas here are also used to have multiple differential-drive vehicles following an autonomous or remotely controlled leader vehicle. ($\Delta x, \theta$) separation-direction control can be visualized as shown in Figure 4.46. Future work will examine the related control saturation prevention ideas presented within [64].

![Figure 4.46: Visualization of ($\Delta x, \theta$) Separation-Direction Control System](image)

Here, ($\Delta x, \theta$) are commanded. $\Delta x$ is measured using an ultrasonic sensor. $\theta$ is directly measured by the IMU. Here, $y$ position is not considered (assumed to be small) since the ultrasonic sensor only provides almost lineal directional information.

The use of a proportional gain controller is justified because the map from the references $v_{\text{ref}}$ and $\omega_{\text{ref}}$ to the actual speeds $(v, \omega)$ looks like a diagonal system.
diag \left( \frac{a}{s + a}, \frac{b}{s + b} \right) \) (at low frequencies). This is a consequence of a well-designed inner-loop (see above). The outer-loop \( \theta \) controller therefore sees \( \frac{b}{s(s+b)} \), and position controller sees \( \left[ \frac{b}{s(s+b)} \right] \) (since \( v_y \) is so small, integrating \( v \) results in \( x \) position). From classical root locus ideas, a proportional controller is therefore justified - provided that the gain is not too large. If the gain is too large, oscillations will be expected. A PD controller with roll off would help with this issue.

![Constant separation using P control](image)

**Figure 4.47:** Vehicle Separation Control (Proportional Control: \( K_p = 0.5, 1.0, 1.5, 2.0; \ \Delta x_o = 1 \))
Figure 4.48: Vehicle Separation Control (PD control: $K_p = 0.5, 1.0, 1.5, 2.0; K_d=1; \Delta x_o = 1$)

Figures 4.47-4.48 illustrate separation convergence of proportional and PD control. A 1m initial condition was given. The desired separation L is 0.2 m. The following observations are noted:

- For a proportional control, as we increase the value of $K_p$ the robot moves faster but the overshoot increases.
- For a PD control, the robot is able to move fast with no overshoot.

**CODE.** The Raspberry Pi C code used for implementing our ($\Delta x, \theta$) outer-loop separation-direction control control law can be found within Appendix A on page 242. This outer-loop uses the ($\omega_R, \omega_L$)-($v, \omega$) inner-loop control law that has been implemented using the Arduino code within Appendix C on page 294.
**Fundamental Limitations Due to Sensors, Actuators, Hardware/Software.**

It must be emphasized that the performance exhibited within each demonstration is fundamentally constrained by the limitations of the sensors, actuators, and hardware/software being used. Both bandwidth (speed/dynamic) and accuracy (static/ steady state) limitations are a concern in any practical embedded system implementation. Given this, it is important to acknowledge the following sensor, actuator, hardware/software limitations. We follow the same structure in [14] and only difference will be discussed in detail. Detailed discussion can be found in [14] at page 161-172.

- **Factor-of-Ten Performance Limitation Rule.** Here is a simple - commonly used - “factor of ten rule. If we can sense/actuate/compute at a maximum (“reliable” or “available”) rate $\omega$ rad/sec, then the widely used factor of 10 rule yields a maximum control bandwidth of $0.1\omega$ rad/sec or about $\frac{1}{60}\omega$ Hz.

The reason for this “factor-of-ten rule” is to adequately guard against the real-world “push-pop effects” that are observed in practical embedded system applications. Generally, as we push the bandwidth higher (improving performance at lower frequencies), we generally pay a price (e.g. increase in sensitivity) at higher frequencies. As we get closer to the maximum available bandwidth, the push-pop phenomenon gets worse [52], [69]. Future work will examine how far we can really push this rule; e.g. when can we get away with a “factor-of-five rule?”

Understanding fundamental hardware limitations is critical to understand what is realistically achievable. This is addressed for each of the following: wheel DC motors, encoders, ultrasonic sensors, camera, Arduino Uno, IMU, Raspberry Pi camera and
Raspberry Pi 3. The following is common to all demonstrations:

- **6V Brushed dc Motors.** Within this thesis, we use 6V brushed armature controlled DC motors on our Thunder Tumbler vehicles. When the applied voltage is less than 0.3V, there is a dead zone and the wheels do not spin (wheels off ground). With a 1V voltage input, the wheel which connects to the shaft runs at about 16 rad/sec (measured via encoders, wheels off ground).

- **Original Unloaded Vehicle (Not Enhanced).** [14]. The original vehicle (unloaded, unenhanced) has a mass of 0.56 kg. The maximum vehicle speed observed was around 4.5 m/sec (obtained via measuring tape and clock). The associated wheel angular velocity is \( \left( \frac{4.5}{0.05} \right) 90 \text{ rad/sec} \) (or 859 rpm). The unloaded torque-speed profile for the motor-wheel system (off the ground) can be found in [14] at page 162. the maximum motor torque as 0.048 Nm.

- **Fully Loaded (Enhanced) Vehicle.** When the vehicle is fully loaded (i.e. enhanced), the vehicle mass increases by 55% (with respect to the original mass) to 0.87kg. We observed a maximum (fully loaded) “average” vehicle acceleration of about 3\( m/sec^2 \) (obtained via measuring tape and clock). The maximum (fully loaded) vehicle speed (wheels on ground) observed was about 2.3 m/sec. The associated wheel angular velocity is \( \left( \frac{2.3}{0.05} \right) 46 \text{ rad/sec} \) (or 439 rpm). A more powerful DC motors would help. This will be investigated in future work.

- **Arduino D-to-A (Actuation).** In this thesis, the Arduino actuation rate to the motor shield is 10Hz (0.1 sec actuation interval) or about 60\( rad/sec \). Given this, the widely used factor-of-ten rule yields maximum control bandwidth of 6 rad/s. Associated with classic D-to-A actuation is a zero order hold half sample
time delay [55]. Why?

\[
ZOH(j\omega) = \frac{1 - e^{-j\omega T}}{j\omega} = e^{-j\omega 0.5 T} \frac{j 2 \sin \omega 0.5 T}{j\omega} = T e^{-j\omega 0.5 T} \left[ \frac{\sin 0.5 \omega T}{0.5 \omega T} \right]
\] (4.15)

The half sample time delay is seen in the term \(e^{-j\omega 0.5 T}\). From the following first order Pade approximation

\[
e^{-s\Delta} \approx \frac{e^{-s 0.5 \Delta}}{1 + s 0.5 \Delta} = \left[ \frac{2}{\Delta} - s \right] \left[ \frac{2}{\Delta} + s \right]^{-1}
\] (4.16)

it follows that a time delay \(\Delta\) has a right half plane (non-minimum phase) zero at \(z = \frac{2}{\Delta}\). With \(\Delta = 0.05\) (half sample time delay associated with ZOH), we get \(z = \frac{2}{0.05} = 40\). This then yields, using our factor-of-ten rule, a maximum control bandwidth of about 4 rad/s. We thus see that a maximum inner-loop control bandwidth of about 4-6 rad/sec is about all we should be willing to push without further (more detailed) modeling.

- **Arduino A-to-D (Sampling).** In this thesis, the sampling time for all experimental hardware demonstrations is 10 Hz (0.1 sec actuation interval) or about 60 rad/sec. Given this, the widely used factor-of-ten rule yields maximum control bandwidth of 6 rad/s. It should be noted that the Arduino has a 10-bit ADC \(\left(2^{10} = 1024\right)\) capability [36]. How does this impact us? This translates to about 0.1% of the maximum speed. If we associate a maximum voltage 5 V with 10 bits and a maximum speed of 3 m/sec, it follows that a 1 bit error translates into a \(\frac{3}{1024} \approx 0.003 \ m/sec\) speed error. This is not very significant so long as the speeds that our vehicles are likely to operate at are not too low. If the speed is greater than 3 cm/sec, then this 1 bit error (0.003) will represent less than 10%; 5% for speeds exceeding 6 cm/sec. Again, we’d have to travel very slowly for this 1 bit error to matter.

- **Wheel Encoder Limitations.** Encoders on a vehicle’s wheels can be used to measure wheel angular speed, wheel angular rotation, wheel translational
speed, wheel linear translation. Let’s focus on the latter because it corresponds to vehicle linear translation when moving along a straight line. For our differential-drive Thunder Tumbler vehicles, we use eight encoders on each wheel. As such, our angular resolution is \( \frac{2\pi}{20} = 18^\circ \). This amount of error seems very large. Because we could not make stable measurement for high resolution encoder code disk. We then decided to see what we could achieve with this low-cost speed-position measuring solution. A consequence of using wheel encoders for measuring distance traveled is the inevitable accumulation of dead-reckoning error. The spatial resolution associated with an 20 CPT system is

\[
x_{\text{resolution}} = r_{\text{wheel}} \theta_{\text{opt, resolution}} = (5\text{cm})\left(\frac{2\pi}{20}\right) = 0.0157 \approx 1.6 \text{ cm}.
\]

How do we use this information? Let the variable ‘counter’ denote the number of pulses that we have counted due to wheel rotation. (The count increments each time a magnet crosses the photo-interrupter.) The distance traveled at each count is

\[
\Delta x = 0.016 \times \text{counter} \text{ m}.
\]

In between counts, we have a maximum error of 1.6 cm. In [14], the author discuss the following thing in detail. (1) Given a desired \( x \), how do we issue reference commands? That is, how do we decide on the number of counts needed? (2) How can the dead reckoning error build up? (3) What control bandwidth limitations do the encoders impose? (4) What speed estimation and following issues can arise? We now address them briefly for completeness.

1. **Issuing Reference Commands: Determining Desired Count.** Let’s suppose that we wish to reach a point \( x = f x_{\text{resolution}} \) where \( f \in [0, \infty) \) is a desired position factor. We wish to minimize the following fractional error (number between 0 and 1) over all integer counter values:
\[ \left| \frac{x - x_{\text{resolution} \cdot \text{counter}}}{x} \right| = \left| \frac{fx_{\text{resolution}} - x_{\text{resolution} \cdot \text{counter}}}{fx_{\text{resolution}}} \right| = \left| \frac{f - \text{counter}}{f} \right| = 1 - \frac{\text{counter}}{f} \]  

(4.17)

\[ \left| \frac{x - x_{\text{resolution} \cdot \text{counter}}}{x} \right| = \left| \frac{fx_{\text{resolution}} - x_{\text{resolution} \cdot \text{counter}}}{fx_{\text{resolution}}} \right| = \left| 1 - \frac{\text{counter}}{f} \right| \]  

(4.18)

The optimal counter value is

\[ \text{Optimal Counter} = \text{round}(f) \]  

(4.19)

where round is the standard rounding function; e.g., \( \text{round}(0.4999) = 0 \), \( \text{round}(0.5001) = 1 \) (see Figure 4.49 on page 135). With some thought, the solution presented is quite intuitive.

![Figure 4.49: Minimizing Counter Value Versus Desired f](image)

The associated optimal percent error is given by

\[ \text{Optimal Percent Error} = 100 \left| 1 - \frac{\text{round}(f)}{f} \right| \]  

(4.20)
2. **Build Up of Dead Reckoning Error.** The term dead reckoning refers to the estimation of a new position from an old position and available measurements. Dead reckoning errors can only be corrected with an instrument that gives us better information. Such as, for example, a camera - as we use within the thesis.

3. **Bandwidth Limitations Due to Optical Encoders.** Next, we address bandwidth limitations imposed by the encoders. To address this, we ask how fast can our encoders sample the angular velocity of the wheels? This is related to the vehicle’s speed. The number of samples (or counts) per sec can obtained as follows by noting that the vehicle speed is related to the wheel angular velocity via \( v = r_{\text{wheel}} \omega \):

\[
\left( \frac{20 \text{ samples}}{\text{rev}} \right) \left( \frac{\text{rev}}{2\pi \text{ rad}} \right) \omega \frac{\text{rad}}{\text{sec}} = 63.66 \frac{\text{samples}}{\text{sec}} \tag{4.21}
\]

This rate is in Hz. We multiply by \( 2\pi \) to convert to rad/sec. Doing so
yields $400v\frac{rad}{sec}$. Using our factor-of-ten rule then gives $BW_{\text{encoder limit}} = 0.1(400v) = 40v\frac{rad}{sec}$ where the vehicle speed $v$ is measured in m/sec; i.e. we will have an associated $40v\frac{rad}{sec}$ encoder-based maximum control bandwidth constraint. We see that $40v$ will be larger than our limit $4\frac{rad}{sec}$ (due to half sample A-to-D zero order hold effect with $T = 0.1$, see above) if $v > 0.1\frac{m}{sec}$ or about $8\frac{\text{inches}}{sec}$. While this minimum speed constraint is easy to satisfy, the result suggests that we can have problems at low vehicle speeds.

**Speed Estimation and Following Issues.** Given the position error discussion above, it is natural to discuss speed issues. The vehicle speed can be estimated using equation (4.3) (page 93)

$$v = r_{\text{wheel}}\omega \approx (0.05m) \left(\frac{\text{counts}}{T\ \text{sec}}\right) \left(\frac{\text{rev}}{8\ \text{counts}}\right) \left(\frac{2\pi}{\text{rev}}\right) \quad (4.22)$$

$$= 0.1571\text{counts} \frac{m}{sec} \quad (4.23)$$

From this, we therefore see that if we give a speed command between $0.1571\text{ counts}$ and $0.1571\ (\text{counts} + 1)$, then the speed can oscillate between these two values and the associated maximum percent speed error can be $100\left(\frac{0.1571(\text{counts}+1)-0.1571\text{counts}}{0.1571\text{counts}}\right) = \left(\frac{100}{\text{counts}}\right)\%$. Given this, the error will be less than $10\%$ if $\text{counts} > 10$; i.e. $10\text{counts}$ in a $T = 0.1$ sampling window or $(0.1571)(10) = 1.57 \frac{m}{sec}$ (100 counts/sec). This is smaller than the maximum observed speed of the fully loaded vehicle (i.e. about $2.3\ \frac{m}{sec}$). When we observe only $\text{count} = 5$ counts in a $T = 0.1$ sec sampling window (or $50\frac{\text{counts}}{sec}$), the percent error rises to $20\%$. This corresponds to a speed of $(0.1571)(5) = 0.785 \frac{m}{sec}$. Since the maximum speed is $2.3\ \frac{m}{sec}$, we are likely to observe at most $\frac{2.3}{0.1571} = 14.65$ or 14-15 counts in a $T = 0.1$ sec sampling window. This will result in a maximum percent
speed error of \( \frac{100}{15} - \frac{100}{14} \) or 6.7 – 7.1%. This suggests that we can have significant speed following issues when low speed reference commands are issued. The above analysis also suggest that a single miss count will, at best, result in 6.7 – 7.1% speed percent error.

At very low speeds, a single miss count can result in a far greater error.

- **IMU Limitations.** We now summarize IMU limitations. Within this thesis, the IMU is used for measuring the orientation \( \theta \) of the differential-drive Thunder Tumbler vehicle. The IMU has a 16 bit output, it has internal built-in algorithm to calculate absolute angle. Then it follows that we will have an \( \theta \) resolution of \( \frac{360}{2^{16} - 1} \approx 0.0055^\circ \).

From a dynamic standpoint, the IMU is able to provide 100 readings per second [58], which is 100 Hz. Using our factor-of-ten rule, it follows that \( BW_{IMU_{limit}} = 0.1 \times 2\pi(100) \approx 60\text{rad/s} \). In our work, we used 10 Hz or 60 rad/sec which then gives an IMU-based maximum control bandwidth constraint of 6 rad/sec. Given the above discussions (e.g. 4 rad/sec Arduino A-to-D limitation), this is not limiting.

For completeness, other IMU properties (although not relevant to the thesis) are as follows. The IMU possesses the following range characteristics: Accelerometer - \( \pm 2, 4, 6, 8, 16\text{g} \); Gyro - \( \pm 245, 500, 2000^\circ/\text{sec} \). Given the above, possibilities for improvement include the following: (1) Combine IMU with wheel encoders (and possibly more encoders) for better estimation of speed/position, (2) Use a higher fidelity IMU together with a camera and LIDAR.

- **Camera Field of View (FOV) Limitations.** The Raspberry Pi 5MP camera FOV is limited at 53.5° horizontally and a 41.41° vertically (independent
of how it is pointed). A pan-tilt servomechanism could significantly extend the above FOVs. For a ground robot tracking a quadrotor flying overhead, the performance can be improved by putting in a pan-tilt servomechanism for the camera. In that way, the camera is able to view a larger area.

- **Camera Frame Rate Limitations.** According to the Raspberry Pi camera datasheet [57], the Raspberry Pi camera is capable of 15 fps (frames per second) at a resolution of 2592 × 1944, 30 fps at 1980 × 1080, 42 fps at 1296 × 972, and 60 fps at 640 × 480. The lowest frame rate corresponds to 15 Hz or about 90 rad/sec. Using our factor-of-ten rule, this will constrain performance to a camera-based maximum control bandwidth constraint of about 9 rad/sec. Given the above discussions (e.g. 4 rad/sec Arduino A-to-D limitation), this is not limiting. For our experiments, we used the lowest resolution and hence the highest frame rate.

- **Image Processing Limitations.** The algorithm we used for image processing within this thesis is a color filtering algorithm from [24]. The image is processed on a Raspberry Pi 3 using Opencv [24]. Given this, the relationship between frame rate and resolution is given as follows: 1 fps at a resolution of 1080 × 720 (0.78 MP), 2.5 fps at a resolution of 640 × 480 (0.31 MP), and 10 fps at a resolution of 320 × 240 (0.08 MP). The image processing has a maximum speed of 10 fps (10 Hz). Even when the resolution is lower than 320 × 240, the frame rate does not increase significantly. In the demos which involve image processing, a resolution of 320 × 240 is selected. This gives a processing speed of 10 fps or 10 Hz or about 60 rad/sec. Using our factor-of-ten rule, then gives an image processing-based maximum control bandwidth of 6 rad/sec. Given the above discussions (e.g. 4 rad/sec Arduino A-to-D zero order hold half sample limitation), this is not limiting.
• **Ultrasonic Sensor Limitations.** The SR04 ultrasonic sensor detection range is from 2 cm to 5 m, and has a maximum reflection (utilization) angle\(^2\) of 45° [68]. It can send out a 40 kHz ultrasonic signal every 200 \(\mu\)sec. Consider an object sitting 2 m away from the sensor. The time delay can be calculated as \(\Delta = \frac{2 \times 2m}{340 m/\text{sec}} \approx 0.0118 \text{ sec}\). This gives a right half plane zero at around \(z = \frac{2}{\Delta} \approx 170\). Using our factor-of-ten rule, this will constrain the control bandwidth to (about) an ultrasonics constrained maximum control bandwidth of 17 rad/sec. Given the above discussions (e.g. 4 rad/sec Arduino A-to-D zero order hold half sample limitation), this is not limiting.

• **Inner-loop Controller: PI vs PID.** For all the demos within this thesis, a PI controller is used within the inner-loop. (See discussions provided in Section 1.2 as well as in Chapter 3.) A PID controller is able to provide more phase lead than a PI controller. This could be necessary if the armature inductance is significant. However, our DC motor has an armature inductance around \(L_a \approx 0.3\text{mH}\) and an armature resistance around \(R_a \approx 1 \Omega\). This combination gives an approximate high frequency pole at \(s \approx -\frac{R_a}{L_a} = -333\). As such, it does not affect the phase at low frequencies too much. Given this, the choice of a PI controller within the thesis for inner-loop control, and not a PID controller, is justified.

\(^2\)If the ultrasonic sensor is pointed directly at a wall (i.e. 0° pointing angle - measure with respect to horizontal), it will get a clean reflection. If the angle is increased to 45°, the sensor may not get any reflection back.
**Differential-Drive Inner-Loop Control Attributes.** The following differential-drive inner-loop attributes are common to all of our demonstrations.

- Thunder Tumbler Differential-Drive Vehicle

- Inner-loop Model: A linear TITO model was examined then ground model was used for inner-loop control design [15]. The model was discussed earlier in this chapter as well as in Chapters 1 and 3.

- Inner-Loop Actuators: Two 6V DC motors - one on right wheel, one on left wheel

- Inner-Loop Sensor: Optical wheel encoders for speed

- Inner-Loop Outputs: Vehicle speed and angular rate; while these are the (computed, not measured) outputs, we actually feed back the encoder measured quantities \((\omega_R, \omega_L)\); all computations are based on equation (4.1) (page 92), equation (4.2) (page 93), equation (4.3)-(4.4) (page 93).

- **Inner-Loop Reference Commands**: Speed reference command and angular rate reference command \((v_{ref}, \omega_{ref})\)

- **Inner-Loop Controller**: PI controller with roll-off (on each motor) and reference command prefilter; PI controller is justified because the map from the references \((v_{ref}, \omega_{ref})\) to the actual speeds \((v, \omega)\) looks like a diagonal system \(\text{diag} \left( \frac{a}{s+a}, \frac{b}{s+b} \right)\) after closing the \((\omega_R, \omega_L)\) inner-loop; \((\omega_R, \omega_L)\) inner-loop is used within in each of the demonstrations discussed below.

- **Inner-Loop Limitations**: In this thesis, the actuation rate is 0.1sec, which is 10 Hz or 60 rad/sec. Our factor-of-ten rule then yields a maximum control bandwidth of 6 rad/sec. The magnetic encoder-based distance traveled along a straight line is 0.314 *counter* m.
• **Possibilities for Improvement**: More precise inner-loop sensor (e.g. higher resolution commercial optical encoder).

• **Summary of Frequency Response Data**: \( T_{ry} \) looks like unity at low frequency; No peaking; 3dB bandwidth near 2.5 rad/sec

• **Summary of Time Responses from Simulations**: zero steady state error; No overshoot; Settling time around 2 sec

• **Summary of Time Response from Hardware/Experiments**: Some oscillation was observed in experiments. This is due to wheel encoder limitations (see inner-loop limitations discussion above). Other nonlinearities to consider include: stiction in wheels, backlash in gears (due to some spacing between gears), dead-zone of motors (minimum voltage required to move the robot - loaded/unloaded Thunder Tumbler), wheel structure (smooth and soft for maximum contact on hard track vs dimpled and hard for better gripping on soft track)

• **Outer Loop Proportional Controller Limitations**. All proportional controllers designed for the \((v, \theta)\) outer-loop do not have roll off. While this may amplify high frequency sensor noise (speed noise from encoders, directional noise from IMU), it should be noted that the Arduino A-to-D zero order hold 

\[
ZOH = \frac{1-e^{-sT}}{s}
\]

provides high frequency noise attenuation. This will be examined in future work.

The following relevant theory (see Chapter 1 for more details) applies to all of our hardware demonstrations since all demonstrations involve differential-drive Thunder Tumbler vehicles:

• differential-drive robot modeling [15],[59]

• vision-based line/curve following work within [24]
vehicle separation modeling and longitudinal platoon control work within [3], [6] (presenting vehicle separation control laws)

**Key Hardware Demonstrations.** The following describes each demonstration.

1. **Cruise Control Along a Line.** An effective speed-directional \((v, \theta)\) cruise control system - one that follows speed and direction commands - is an essential cornerstone to build more complex control capabilities. It builds on the \((\omega_R, \omega_L)\) or \((v, \omega)\) inner-loop control law (discussed above).

This demonstration involves the following:

- **Outer-Loop Model:** A simplified on ground decoupled TITO LTI model is used for outer-loop control design because of a well-designed \((v, \omega)\) inner-loop control law [15]. See justification of outer-loop below.
- **Outer-Loop Sensor:** Vehicle orientation \(\theta\) directly measured by IMU/Camera is to be fed back
- **Outer-Loop Output:** Vehicle actual speed \(v\) is generated by inner-loop and orientation \(\theta\) is obtained by IMU/Camera
- **Outer-Loop Reference Command:** An orientation reference command \(\theta_{ref} = 0^\circ\) and a constant speed reference command \(v = constant\) is issued to promote motion along a straight line
- **Outer-Loop Controller:** A proportional controller (with no roll off) for vehicle orientation \(\theta\) (direction); vehicle speed is also commanded to inner-loop
  - The use of a proportional gain controller is justified because the map from the references \(v_{ref}\) and \(\omega_{ref}\) to the actual speeds \(v\) and \(\omega\) looks like a diagonal system \(diag(\frac{a}{s+a}, \frac{b}{s+b})\) (at low frequencies). The outer-loop \(\theta\) controller therefore sees \(\frac{b}{s(s+b)}\). From classical root locus ideas, a proportional controller is therefore justified - provided that the gain
is not too large. If the gain is too large, oscillations will be expected in $\theta$. A PD controller with roll off would help with this issue.

- **Performance Tradeoffs:** If the proportional gain $k_\theta$ is large, there will be oscillations in $\theta$. A PD controller with roll off would help with this issue.

- **Limitations:** $BW_{encoder_{limit}} \approx 40v$; will be greater than 4 rad/sec if $v > 0.1$ m/sec. Each vehicle stop and go can result in a 1.5 cm static position error (see discussion above).

- **Possibilities for Improvement:** A more precise inner-loop sensor (e.g. more white-black pair on optical code disk.

- **Summary of Frequency Response Data:** For $\theta$ outer-loop, large $k_p$ yields large bandwidth, large frequency response peaking; small $k_p$ yields small bandwidth, small frequency response peaking; PD control yields large bandwidth, large frequency response peaking.

- **Summary of Time Responses from Simulations:** Large $k_p$ yields fast response, overshoot; small $k_p$ yields slow response, no overshoot; PD control yields fast response, no overshoot.

---

![Cruise control along a line $v_{rel} = 0.5$ m/sec](image)

Figure 4.51: Cruise Control Along a Line
• **Summary of Time Response from Hardware/Experiments:** Experiment data is close to simulation results. Differences can be explained by considering the following possibilities: stiction in wheels, backlash in gears (due to some spacing between gears), dead-zone of motors (minimum voltage required to move the robot - loaded/unloaded Thunder Tumbler), wheel structure (smooth and soft for maximum contact on hard track vs dimpled and hard for better gripping on soft track).

• **Overview/Conclusion:** The robot is issued a reference speed command of 0.5 m/s with \( \theta_{\text{ref}} = 0^\circ \). The hardware result shows little drift from straight along the time due to dead reckoning error.

• **Relevant Theory:** Dhaouadi, et. al. (2013) [15] provides basic TITO LTI vehicle-motor \((v, \omega)\) model for inner-loop design.

2. **Separation-Direction Control.** When the outer-loop properly exploits an ultrasonic distance/range-finding sensor, vehicle-target spacing control can be achieved. Separation-direction \((\Delta x, \theta)\) outer loop and \((v, \omega)\) inner loop are involved. \(\Delta x\) is measured using ultrasonic sensor, and \(\theta\) is directly measured by IMU. Here, \(y\) position is not considered, since ultrasonic sensor only provides almost lineal directional information.

This demonstration involves the following:

• **Outer-Loop Model:** A simple (decoupled) TITO LTI on ground model is used for outer-loop control - a consequence of a well-designed inner-loop - the latter based on the TITO LTI vehicle-motor model within [15]. See control law justification below.

• **Outer-Loop Sensor:** Ultrasonic sensor for nearly-lineal separation information, vehicle angular velocity \(\omega\) measured by IMU is integrated to yield the
vehicle orientation $\theta$ to be fed back

- **Outer-Loop Outputs**: Position and orientation
- **Outer-Loop Reference Commands**: A desired separation distance and orientation ($\theta_{ref} = 0^\circ$)
- **Outer-Loop Controller**: P controller (with no roll off) or PD controller with roll off
  - The use of a proportional gain controller is justified because the map from the references $v_{ref}$ and $\omega_{ref}$ to the actual speeds $v$ and $\omega$ looks like a diagonal system $\text{diag}(\frac{a}{s+a}, \frac{b}{s+b})$ (at low frequencies). The outer-loop $\theta$ controller therefore sees $\frac{b}{s(s+b)}$, and position controller sees $\frac{a}{s(s+a)}$ (since $v_y$ is so small, integrating $v$ results in $x$ position). From classical root locus ideas, a proportional controller is therefore justified - provided that the gain is not too large. If the gain is too large, oscillations will be expected. A PD controller with roll off would help with this issue.

- **Performance Tradeoffs**: Large $k_p$ - fast with large overshoot, small $k_p$ - slow with no overshoot. PD - fast with small overshoot. These are shown in Figure 4.52, 4.53 and 4.54.
Figure 4.52: Vehicle Separation Convergence Using Proportional Control ($K_p = 1; \Delta x(0) \approx 1$)

Figure 4.53: Vehicle Separation Convergence Using Proportional Control ($K_p = 1; \Delta x(0) \approx 1$)
Figure 4.54: Vehicle Separation Convergence Using PD Control ($K_p = 1.5; K_d = 1; \Delta x(0) \approx 1$)

- **Limitations**: The HCSR04 ultrasonic sensor detection range is from 2cm to 5m, and has a minimum reflection angle of 45°. It can send out 40KHz ultrasonic signal every 200 µs. Consider an object sitting 2m away from the sensor. The time delay can be calculated as $\Delta = \frac{1}{340} \approx 0.0118s$. This gives a (non-minimum phase) right half plane zero at around $z = \frac{2}{\Delta} \approx 170$. This, in turn, imposes a maximum control bandwidth constraint of 17 rad/sec (using standard factor-of-ten rule). It should be noted that the dead-reckoning error associated with the wheel encoders does not apply here, since the separation information is measured via ultrasonic sensor and hence is not cumulative.

- **Possibilities for Improvement**: While a more precise inner-loop sensor (e.g. commercial optical rotary encoder) could help, something like a camera or Lidar or GPS can provide useful measurements that can be used to correct the accumulated dead reckoning error.

- **Summary of Frequency Response Data**: For the $\Delta x$ outer-loop, large $k_p$
yields large bandwidth, large frequency response peaking, small $k_p$ yields small bandwidth, small frequency response peaking, PD control yields large bandwidth, and small frequency response peaking.

- **Summary of Time Response from Simulation**: Large $k_p$ yields fast response, overshoot; small $k_p$ yields slow response, no overshoot; PD control yields fast response, no overshoot.

- **Summary of Time Response from Hardware/Experiment**: Data from experiments was found to be close to simulation results (see Figures 4.52-4.54). The following needs to be considered to adequately explain differences between simulation and experiment: stiction in wheels, backlash in gears (due to some spacing between gears), dead-zone of motors (minimum voltage required to move the robot - loaded/unloaded Thunder Tumbler), wheel structure (smooth and soft for maximum contact on hard track vs dimpled and hard for better gripping on soft track).

- **Overview/Conclusion**: Initially the robot is about 1 meter away from the target, and we want to keep a 20cm constant spacing through ultrasonic sensor. Proportional control and PD control is examined. It is shown that Small $k_p$ - slow with no overshoot, large $k_p$ - fast with large overshoot. PD - fast with no overshoot.

- **Relevant Theory**: Vehicle separation modeling and longitudinal platoon control work is presented within [3], [6]. Future work will examine more about the related control saturation prevention ideas presented within [64].
4.4 Summary and Conclusion

This chapter has provided a comprehensive case study for our enhanced differential-drive Thunder Tumbler vehicle. Both simulation and hardware results were presented. Many demonstrations were thoroughly discussed. All control law developments were supported by theory. Differences between hardware results and simulation results were also addressed. Particular focus was placed on the fundamental limitations impose by system components/subsystems.
Chapter 5

LONGITUDINAL CONTROL OF A PLATOON OF VEHICLES

5.1 Introduction and Overview

The subject of design and analysis of longitudinal control laws has been studied extensively from 1960s to 21st new century [1] - [13]. Numerous topics such as vehicle-follower controllers, nonlinear vehicle dynamics, automated guideway transit systems has been reported. Even though much effort has been spent on various control laws for longitudinal control of a platoon of vehicles, however, few experiments had been done due to expensive hardware. This paper will propose classical control law for platoon model of electrical vehicles. The concept of this study is to take full advantage of advances in communication and measurement, using ETT low-cost differential-drive robots to show how a platoon of vehicles can maintain constant spacing and how communication among the vehicle network will improve the performance.

Longitudinal platoon of identical vehicles layout is shown in Figure 5.1. For a change in the lead vehicles velocity, the resulting changes in the spacing can have different behavior due to different controller design. Simulation results show that through the appropriate choice of coefficients in the control law, the deviations in vehicle spacings from their respective steady-state values do not get magnified along the platoon and in fact get attenuated as one goes down the platoon.

5.2 Platoon Configuration

Figure 5.1 shows the a platoon configuration for a platoon of one leader with 4 followers [3]. The platoon is assumed to move in a straight line. The position of the

151
Figure 5.1: Platoon of 4 Vehicles

$i – th$ vehicles rear bumper with respect to a fixed reference point 0 on the roadside is denoted by $x$. The position of the lead vehicles rear bumper with respect to the same fixed reference point 0 is denoted by $x_l$ or $x_0$. Each vehicle is assigned a slot of length $L$ along the road. As shown, $\Delta$ is the deviation of the $i – th$ vehicle position from its assigned position. The subscript $i$ is used because $\Delta_i$ is measured by ultrasonic sensor located in the $i – th$ vehicle.

Given the platoon configuration in Figure 5.1, elementary geometry shows that:

for $i = 2, 3, ...$

$$\Delta_i(t) := x_{i-1}(t) - x_i(t) - L \quad (5.1)$$

$$\dot{\Delta}_i(t) := \dot{x}_{i-1}(t) - \dot{x}_i(t) \quad (5.2)$$

$$\ddot{\Delta}_i(t) := \ddot{x}_{i-1}(t) - \ddot{x}_i(t) \quad (5.3)$$

Kinematics The corresponding kinematic equations for the lead vehicle and the first vehicle are as follows:

$$\Delta_1(t) := x_1(t) - x_1(t) - L \quad (5.4)$$

$$\dot{\Delta}_1(t) := v_1(t) - \dot{x}_1(t) \quad (5.5)$$

$$\ddot{\Delta}_1(t) := a_1(t) - \ddot{x}_1(t) \quad (5.6)$$

152
5.3 Modeling for Longitudinal Platoon of Vehicles

As we discussed in 3.7.3, the on ground inner-loop \((\omega_R, \omega_L)\) plant:

\[
P_{[\omega_R, \omega_L]} \approx 11.268 \left[ \frac{1.159}{s + 1.159} \right] \times I_{2 \times 2}
\] (5.7)

will be used as basis our vehicle model for platoon. The vehicle model for longitudinal control is shown in Figure 5.2 and 5.3, where bypass signal to inner loop controller is a safety feature to realize negative control informally due to encoder limitation. When this signal is set to 1, we applied constant negative PWM command in that control period to our motor neglecting the current control signal generated by inner loop controller. More detailed discussion can be found in later section about platoon experimental result.

![Figure 5.2: Leader Model of Platoon](image1)

![Figure 5.3: i-th Vehicle Model of Platoon](image2)

In this model, we made the following assumption: all vehicles are identical, in each vehicle left and right sides are symmetric, and no wind gust. In practical, lateral
deviation was treated as disturbance and can be corrected by inner loop with IMU or camera.

5.4 Control for Longitudinal Platoon of Vehicles

In this section, different control laws for longitudinal platoon are discussed. The ideas presented within [3], [6] motivate the PID-based control law for longitudinal platoon separation control. The leader is controlled by a speed-directional \((v, \theta)\) cruise control controller with constant cruise speed. The \(i\)-th follower in the platoon control can be visualized as shown in Figure 5.4. And the platoon controller contains two modes can be visualized as shown in in Figure 5.4.

![Figure 5.4: Visualization of i-th Follower in Platoon Separation Control System](image)

Here, \(\theta_i\) are commanded to be zero. \(\Delta_i\) is compute by 5.2 relative distance is measured using an ultrasonic sensor. \(\theta\) is directly measured by the IMU or Camera. \(e_{v_{ff}}\) is the speed difference between leader and \(i-th\) follower and defined as \(e_{v_{ff}} = v_l - v_i\). \(v_l\) is obtained from wireless communication. \(th\) is predefined minimum separation distance, which is used as a switch to change controller mode from mode 1 to mode 2. Mode 1 is the normal operation mode when \(\Delta_i\) is greater than \(th\). Whereas mode 2 is a safety feature which will enable inner loop controller send negative PWM command.
to motor directly and bypass PWM command generated by inner loop controller. $K_x$ is a PID controller with respect to separation error $e_{xi}$ ($\Delta_i$), we named it $\Delta_x$ path. $K_\theta$ is a PD controller with respect to orientation error $e_\theta$. $K_{ff}$ is a PI controller with respect to speed difference between leader and follower, we named it feed-forward path (FF-path).

The use of a proportional gain controller is justified because the map from the references $v_{ref}$ and $\omega_{ref}$ to the actual speeds $(v, \omega)$ looks like a diagonal system $\text{diag} \left( \frac{a}{s+a}, \frac{b}{s+b} \right)$ (at low frequencies). This is a consequence of a well-designed inner-loop (see above). The outer-loop $\theta$ controller therefore sees $\frac{b}{s(s+b)}$, and position controller sees $\left[ \frac{b}{s(s+b)} \right]$ (since $v_y$ is so small, integrating $v$ results in $x$ position). From classical root locus ideas, a proportional controller is therefore justified - provided that the gain is not too large. If the gain is too large, oscillations will be expected. A PD controller with roll off would help with this issue. The additional control effort from communication path works like feed-forward and help reduced overshoot.

Figure 5.5: Visualization of Platoon Controller
5.5 Longitudinal Platoon Separation Controller Tradeoff Study

In this section, we examined different types of controller design. Specifically,

- Proportional Control for $\Delta x$ path
- PD Control for $\Delta x$ path
- PID Control for $\Delta x$ path
- PID Control for $\Delta x$ path
- PID Control for $\Delta x$ path + Proportional Control for FF-path
- PID Control for $\Delta x$ path + PI control for FF-path

**Proportional Control for $\Delta x$ path.** Proportional control for $\Delta x$ is used, and no leader information.

![Simulation result of platoon of 5 vehicle](image)

Figure 5.6: Simulation of Vehicle Separation Control of Platoon (Proportional Control for $\Delta x$ ($K_p = 0.5$))
Figure 5.7: Simulation of Control Response of Platoon (Proportional Control for $\Delta x$ ($K_p = 0.5$))

Figure 5.8: Simulation of Vehicle Separation Control of Platoon (Proportional Control for $\Delta x$ ($K_p = 1.0$))
Figure 5.9: Simulation of Control Response of Platoon (Proportional Control for $\Delta_x$ ($K_p = 1.0$))

Figure 5.10: Simulation of Vehicle Separation Control of Platoon (Proportional Control for $\Delta_x$ ($K_p = 2.0$))
Figure 5.11: Simulation of Control Response of Platoon (Proportional Control for $\Delta_x$ ($K_p = 2.0$))

Summary of Time Response from Simulation: From above simulation, we observed the following:

- Platoon separation reach steady state slowly
- Separation steady state error decreasing with increasing $k_p$ but always exist
- Separation oscillation increasing and get amplified along the platoon
- Control response peak and oscillation increasing with increasing $k_p$ and get amplified along the platoon

Since we don’t want separation control error get amplified and too much oscillation in separation and control. This motivates us to examine PD controller for $\Delta_x$ path.

PD Control for $\Delta_x$ path. PD control for $\Delta_x$ is used, and no leader information. Controller structure is like following:

$$K_x = \frac{g(s + z) \times 100^2}{(s + 100)^2}$$  \hspace{1cm} (5.8)
where, $z = 5$ and two roll-off is used to reduce overshoot from derivation-term. Note here $k_d = g$ and $k_p = gz$.

![Simulation result of platoon of 5 vehicle](image1)

Figure 5.12: Simulation of Vehicle Separation Control of Platoon (PD Control for $\Delta_x$ ($g = 0.2$))

![Control response of platoon of 5 vehicle](image2)

Figure 5.13: Simulation of Control Response of Platoon (PD Control for $\Delta_x$ ($g = 0.2$))
Figure 5.14: Simulation of Vehicle Separation Control of Platoon (PD Control for $\Delta x$ ($g = 0.5$))

Figure 5.15: Simulation of Control Response of Platoon (PD Control for $\Delta x$ ($g = 0.5$))

Summary of Time Response from Simulation: From above simulation, we observed the following:

- Platoon separation reach steady state faster than proportional control
- Separation steady state error getting smaller and with increasing $g$, but still exist
• Separation oscillation don’t get amplified along the platoon with small $g$

• Control response peak and oscillation get smaller with small $g$ and get attenuated along the platoon

While a bigger $g$ ($k_d$) can reach smaller steady state separation error, more oscillation is observed in transient response for separation as well as control. To reach zero steady state error in platoon separation control, PID controller for $\Delta x$ path will be examined first, then feed-forward path is introduced to reduce oscillation.
**PID Control for $\Delta x$ path.** PID control for $\Delta x$ is used, and no leader information. Controller structure is like following:

$$K_x = \frac{g(s + z)^2 \times 100^2}{s(s + 100)^2} \quad (5.9)$$

where, two zeros at $z = 1$ and two roll-off is used to reduce overshoot from derivation-term. Note here $k_d = g$, $k_p = 2gz$ and $k_i = gz^2$.

![Simulation result of platoon of 5 vehicle](image)

Figure 5.16: Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta x \ (g = 1.0)$)

163
Figure 5.17: Simulation of Control Response of Platoon (PID Control for $\Delta x$ ($g = 1.0$))

Figure 5.18: Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta x$ ($g = 2.0$))
Figure 5.19: Simulation of Control Response of Platoon (PID Control for $\Delta x (g = 2.0)$)

Summary of Time Response from Simulation: From above simulation, we observed the following:

- Platoon separation reach steady state faster than proportional control
- Separation steady state error becomes zero
- Separation oscillation get amplified along the platoon
- Control response peak and oscillation get amplified along the platoon

While a PID controller can reach zero steady state error in separation, more oscillation is observed in transient response for separation as well as control. Feed-forward path is needed to reduce oscillation.
**PID Control for $\Delta_x$ path + Proportional Control for Feed-forward.** Here, PID control for $\Delta_x$ is used, and wireless communication is used to get leader speed information and proportional control for feed-forward error $e_v = v_l - v_i$. The controller output is sum of $\Delta_x$ path and feed-forward path. Controller structure is like following:

$$K_{\text{longitudinal}} = K_x + K_{ff}$$  \hspace{1cm} (5.10)

$$K_x = \frac{g(s+z)^2 \times 100^2}{s(s+100)^2}$$  \hspace{1cm} (5.11)

$$K_{ff} = k_{pff}$$  \hspace{1cm} (5.12)

where, two zeros at $z = 1$, $g = 1$ and two roll-off is used to reduce overshoot from derivation-term. $k_{pff}$ denotes the proportional gain using at FF path. Note here $k_d = g$, $k_p = 2gz$ and $k_i = gz^2$.

![Simulation result of platoon of 5 vehicle](image)

Figure 5.20: Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x$ ($g = 1.0, z = 1.0$) and $k_{pff} = 0.5$ for FF-path)
Figure 5.21: Simulation of Control Response of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and $k_{pff} = 0.5$ for FF-path)

Figure 5.22: Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and $k_{pff} = 1.5$ for FF-path)
Summary of Time Response from Simulation: From above simulation, we observed the following:

- Platoon separation reach steady state slower than PID control
- Separation steady state error becomes zero
- Separation oscillation don’t get amplified significantly along the platoon
- Control response peak and oscillation get amplified along the platoon

Compared to a PID controller, zero steady state error in separation is also reached. Moreover, smaller oscillation is observed in transient response for separation. But control transient response still shows clear oscillation and get amplified along the platoon and cannot easily solved by increasing $k_{pff}$, which lead to using PI control in Feed-forward path.
**PID Control for $\Delta x$ path + PI Control for Feed-forward.** Here, PID control for $\Delta x$ is used, and wireless communication is used to get leader speed information and proportional control for feed-forward error $e_v = v_l - v_i$. The controller output is sum of $\Delta x$ path and feed-forward path. Controller structure is like following:

$$K_{\text{longitudinal}} = K_x + K_{ff}$$  \hspace{1cm} (5.13)

$$K_x = \frac{g(s + z)^2 \times 100^2}{s(s + 100)^2}$$  \hspace{1cm} (5.14)

$$K_{ff} = \frac{g_{ff}(s + z_{ff}) \times 100}{s(s + 100)}$$  \hspace{1cm} (5.15)

where, two zeros at $z = 1$, $g = 1$ and two roll-off is used to reduce overshoot from derivation-term. $g_{ff} = 0.5$ denotes the proportional gain using at FF path, and $z_{ff}$ denotes the zero in FF-path, one roll-off is used to reduce overshoot from zero at FF path. Note here $k_d = g$, $k_p = 2gz$ and $k_i = gz^2$ and $k_{pff} = g_{ff}z_{ff}$

Figure 5.24: Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta x$ ($g = 1.0$, $z = 1.0$) and PI Control for FF-path ($g_{ff} = 0.5$, $z_{ff} = 1.0$))
Figure 5.25: Simulation of Control Response of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and PI Control for FF-path ($g_{ff} = 0.5$, $z_{ff} = 1.0$))

Figure 5.26: Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta_x (g = 1.0, z = 1.0)$ and PI Control for FF-path ($g_{ff} = 0.5$, $z_{ff} = 2.0$))
Summary of Time Response from Simulation: From above simulation, we observed the following:

- Platoon separation reach steady state fast
- Separation steady state error becomes zero
- Separation oscillation get attenuated along the platoon
- Control response peak and oscillation get attenuated along the platoon

Compared to a last design, zero steady state error in separation is also reached and much smaller oscillation is observed in transient response for separation. Besides, control transient response also get attenuated along the platoon. Oscillatory behavior in control is observed when $z_{ff}$ it too big.

Summary of Tradeoff Study In this section, we examined five different controllers design with and without feed-forward path enabled by wireless communication. From
the discussion, we noticed that ultrasonic-based PID controller for $\Delta_x$ path is sufficient to stabilize the platoon separation control. However, increasing control effort of the platoon reveals that the size of the platoon cannot be scaled arbitrarily because of hardware saturation limitation on control. By introducing integration term in feedforward path and properly choosing the controller parameters, we can not only get monotonically decreasing separation error, but also have decreasing control response in transient.

5.6 Experimental Results for Controlled Platoon of Vehicles

In this section, we examined our controller design through three controller design: Design parameters of three different controllers are given in Table 5.1.

<table>
<thead>
<tr>
<th>Design No.</th>
<th>$\Delta_x$ Path</th>
<th>Feed-Forward Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$K_p = 1$</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>$K_p = 1.5, K_d = 0.2$</td>
<td>$K_p = 0.5$</td>
</tr>
<tr>
<td>3</td>
<td>$K_p = 1.0, K_i = 0.5, K_d = 0.3$</td>
<td>$K_p = 0.4, K_i = 0.6$</td>
</tr>
</tbody>
</table>

Table 5.1: Platoon Design Parameter

For all the experiments six(6) ETTs are used. Simulation as well as hardware will be provided to verify our design. For all simulation and hardware experiment, $\Delta_{x_{ref}} = 0.4$, and leader cruise speed is set to $0.3m/sec$. 
**Design 1.** Proportional control for $\Delta_x$ is used, and no leader information. Simulation result is shown in Figure 5.28 and 5.29

![Simulation result of platoon of 5 vehicle](image1)

Figure 5.28: Simulation of Vehicle Separation Control of Platoon (Proportional Control for $\Delta_x$ ($K_p = 1$))

![Control response of platoon of 5 vehicle](image2)

Figure 5.29: Simulation of Control Response of Platoon (Proportional Control for $\Delta_x$ ($K_p = 1$))

Hardware result of platoon of 6 vehicles is shown in Figure 5.30
Figure 5.30: Experimental Separation Response of Platoon Proportional Control for $\Delta x$ ($K_p = 1$)

- **Summary of Time Response from Simulation:** From above simulation, we see the vehicle separation get amplified along the platoon at beginning and settle down after about 15 seconds, separation oscillation is observed, and steady state error is about $0.3m$. Control response peak get higher and oscillation get larger along the platoon.

- **Summary of Time Response from Hardware/Experiment:** Hardware data was found to be close to simulation results. Compare to simulation result, slower response is observed for all vehicles. Reasons for differences from simulations: stiction in wheels, dead-zone of motors (minimum voltage required to move the robot - loaded/unloaded Thunder Tumbler).
Design 2. PD control for $\Delta_x$ and proportional control for feed-forward path (FF-Path) speed difference between leader and follower. Leader speed information is broadcasted to followers through wireless communication. Simulation result is shown in Figure 5.31 and 5.32

Figure 5.31: Simulation of Vehicle Separation Control of Platoon (PD Control for $\Delta_x$ ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$))

Figure 5.32: Simulation of Control Response of Platoon (PD Control for $\Delta_x$ ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$))
Hardware result of platoon of 5 vehicles is shown in Figure 5.33

Figure 5.33: Experimental Separation Response of Platoon (PD Control for \( \Delta x (K_p = 1.5, K_d = 0.2) \) and Proportional control for FF Path \( (K_p = 0.5) \))

- **Summary of Time Response from Simulation**: From above simulation, we see the vehicle separation does not get amplified along the platoon at beginning and settle down after about 8 seconds, no oscillation is observed, and steady state error is about 0.2m. Control response peak get smaller along the platoon and smaller oscillation is observed, no negative control is observed.

- **Summary of Time Response from Hardware/Experiment**: Hardware data was found to be close to simulation results. Compare to simulation result, higher separation error and little oscillation at beginning is shown in hardware result. Reasons for differences from simulations: stiction in wheels, backlash in gears, dead-zone of motors, wheel structure, negative control is informally applied to motor directly using bypass mechanism.
**Design 3.** PID control for $\Delta x$ and PI control for feed-forward path (FF-Path) speed difference between leader and follower. Leader speed information is broadcasted to followers through wireless communication. Simulation result is shown in Figure 5.34 and 5.35

![Simulation result of platoon of 5 vehicle](image)

**Figure 5.34:** Simulation of Vehicle Separation Control of Platoon (PID Control for $\Delta x$ ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$))

![Control response of platoon of 5 vehicle](image)

**Figure 5.35:** Simulation of Control Response of Platoon (PID Control for $\Delta x$ ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$))
Hardware result of platoon of 5 vehicles is shown in Figure 5.36

Figure 5.36: Experimental Separation Response of Platoon (PID Control for $\Delta x$ ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$))

- **Summary of Time Response from Simulation**: From above simulation, we see the vehicle separation get attenuated along the platoon and settle down after about 10 seconds, no oscillation is observed, zero steady state error is reached. Control response peak get smaller along the platoon and small oscillation is observed, no negative control is observed.

- **Summary of Time Response from Hardware/Experiment**: Hardware data was found to be close to simulation results. Compare to simulation result, higher separation error and little oscillation at beginning is shown in hardware result. Reasons for differences from simulations: stiction in wheels, backlash in gears, dead-zone of motors, wheel structure, negative control is informally applied to motor directly using bypass mechanism.
5.7 Platoon Simulation with model Uncertainty and Stiction Deadzone

In this section, we will show how platoon controller performance degrades when model variation and deadzone effect from stiction has been considered into simulation. The result is only shown for design 3 last section to show future research direction. For model uncertainty, we choose 5 ETTs ground model arbitrarily from all 9 ETTs. Stiction deadzone is randomly set to be $0.82V, 1.24V, 0.99V, 0.58V, 0.83V$.

![Control response of platoon of 5 vehicle](image)

**Figure 5.37:** Simulation of Separation Response of Platoon with model uncertainty and stiction
Compare Figure 5.37 with 5.34, we see that separation peak getting larger, negative error is occurred. Compare Figure 5.38 with 5.35, we see that control peak getting increased, more oscillation is shown. This non-ideal simulation to some extent explain the difference between simulation and hardware results. Future work should investigate this in detail.

5.8 Summary and Conclusions

Based our simplified model, different controllers has been discussed. We have shown controller performance of different designs. Contrast to the control law only rely on local information of immediately predecessor, spacing deviation will not get magnified if leader speed information is sent to followers and proportional controller is implemented for speed difference, and it get attenuated if PI controller is used for speed difference.
Chapter 6

SUMMARY AND FUTURE DIRECTIONS

6.1 Summary of Work

This thesis addressed many design, analysis, and control issues that are critical to achieve the longer-term FAME objective. The following summarizes key themes within the thesis.

1. Update Low-Cost FAME Mobile Robot Platform. In previous work of Lin [14], it was shown how off-the-shelf components could be used to build a low-cost multi-capability ground vehicle that can be used for serious robotics/FAME research. Based on his work, we carefully redesign the platform. Main improvement as following:

- **Encoder.** Optical wheel-encoders which 2.5 times accurate than magnetic based one was developed to increase measuring accuracy of speed.

- **IMU.** Absolute orientation sensor BNO055 IMU can give out vehicle orientation directly that LSM9DS0 can only get angular rate and, this can benefit outer-loop design when $\theta$ is needed.

- **Onboard Microcontroller.** Raspberry Pi 3 with more computation power than pi 2, this would benefit when image processing is needed, experimental results showed FPS increase from 8 to 10.

- **Communication Capability.** TL-WN722N WiFi USB adapter with external antenna which can support more stable communication compared
to Edimax Wi-Fi adapter in longer distance. Duplex serial communication protocol is designed, we can record more vehicle running states on SD card on Pi 3. Previously, we just have 1K EEPROM on Uno can be used for this purpose.

- **Unified Software Architecture.** All outer loops have been implemented on Pi 3, inner loop is fixed on Arduino Uno. One can switch different controllers easily by command interface written in C.

2. **FAME Architecture.** A general FAME architecture has been described one that can accommodate a large fleet of vehicles.

3. **Literature Survey.** A fairly comprehensive literature survey of relevant work was presented.

4. **Modeling.** Kinematic and dynamic models for both vehicle types were presented and analyzed to understand the full utility of each model. Motor dynamics includes gearbox transmission was developed, all modeling parameters have been carefully measured. Uncertainty of those parameters has been captured using several enhanced thunder tumbler. A TITO LTI differential-drive vehicle-motor model was first examined as the basis for inner-loop control, then has been modified as on ground model which including practical factors like friction and battery internal resistance. This on ground model becomes foundation of inner loop design, and all other works follows. Outer-loop control was facilitated by having a well-designed inner-loop - thus permitting a simple classical outer-loop design. Relevant model trade studies were conducted in order to understand the impact of vehicle parameters for differential-drive.
5. **Control for Single Vehicle.** Both inner-loop and outer-loop control designs were discussed in the context of an overall hierarchical control inner-outer loop framework. This framework lends itself to accommodate multiple modes of operation; e.g. cruise control along a line, separation control. Comprehensive inner-loop trade studies were conducted for the differential-drive class of vehicles.

6. **Control for Multiple Vehicle.** For longitudinal platoon separation control, we proposed platoon controller framework based on classical control with feed-forward path. Comprehensive platoon controller trade studies were conducted for the differential-drive class of vehicles.

7. **Hardware Demonstrations.** Many hardware demonstrations were conducted - with simulation data corroborating the hardware results. The limitations of the hardware (e.g. encoders, A-to-D, vehicle) was discussed to shed light on differences between the data sets.

6.2 **Directions for Future Research**

Future work will involve each of the following:

- **Localization.** Development of a lab-based localization system using a variety of technologies (e.g. cameras, lidar, ultrasonic, etc.). Localization is essential for multi-robots cooperating. Once each robot knows where it is and where the other robots are, more complicated robot cooperation can be performed.

- **Onboard Sensing.** Addition of multiple onboard sensors; e.g. additional ultrasonics, cameras, lidar, GPS, etc.
• **Advanced Image Processing.** Use of advanced image processing and optimization algorithms [60]

• **Multi-Vehicle Cooperation.** Cooperation between ground, air, and sea vehicles - including quadrotors, micro-air vehicles and eventually nano-air vehicles.

• **Parallel Onboard Computing.** Use of multiple processors on a robot for computationally intense work; e.g. onboard optimization and decision making.

• **Environment Mapping.** Rapid and efficient mapping of unknown and partially known areas via multiple robotic agents.

• **Modeling and Control.** More accurate dynamic models and control laws. This can include the development of multi-rate control laws that can significantly lower sampling requirements.

• **Control-Centric Vehicle Design.** Understanding when simple control laws are possible and when complex control laws are essential. This includes understanding how control-relevant specifications impact (or can drive) the design of a vehicle.
REFERENCES


[66] Pi 2 and Pi 3 Comparison, http://www.techrepublic.com/article/raspberry-pi-3-how-much-better-is-it-than-the-raspberry-pi-2/


#ifndef INCLUDE_CAMERA_H
#define INCLUDE_CAMERA_H

struct camera_para {
    int inverse;
    int th;
    int roi_x;
    int roi_y;
    int roi_width;
    int roi_height;
};

extern struct camera_para cpara;
extern volatile int camera_should_stop;

void *camera_thread(void *para);
#endif /* INCLUDE_CAMERA_H */

#define CTRL_LOOP_H

#include <pthread.h>
#include <semaphore.h>

struct h2l_vehicle_status;

enum control_mode_t {
    CTRL_OPEN_LOOP = 0,
    CTRL_WI_WR,
    CTRL_V_OMEGA,
    CTRL_V_THETA,
    CTRL_X_Y,
    CTRL_DELTA_X_THETA,
    CTRL_LINE_TRACK,
    CTRL_PLATOONING
};

enum control_flag_t {
    CTRL_NONE,
    CTRL_START,
    CTRL_STOP
};

struct vehicle_state {
    // controller state, it can be 0 (stop)
    // or 1 (running)
int ctrl_state; // running or stopped
int ctrl_flag;
int ctrl_outer_loop_mode;
int ctrl_inner_loop_mode;

// sensor measured states (low level)
uint32_t timestamp;
float imu_theta;
float camera_delta_theta;
float camera_delta_theta_p;
float imu_acx;
float imu_omega;
float encoder_wr;
float ultrasonic_delta_x;
float ultrasonic_delta_x_bk;
float ultrasonic_delta_x_filtered;

// calculated / fused states

// NOTE: these states are in XI-O-YI coordinates
double accx;
double v;
double omega;
double x;
double y;
double theta;
// new 0801 for platoon faster start
double v_dsr;
}

// Inner loop parameter (previously in mc_init)
struct ctrl_inner_para {
  float prefilter_coefficient;
  float roll_off_coefficient;
  float kp_left;
  float ki_left;
  float kd_left;
  float kp_right;
  float ki_right;
  float kd_right;
  float deadzone_threshold;
  float deadzone_saturation;
};

// These are states only used by a specific controller

// If you need to hack the controller or add a new controller, just add variables in these structure
// or create a new structure.
struct ctrl_open_loop_state {
  // desired value
  int16_t pwm_left_dsr;
  int16_t pwm_right_dsr;
};

struct ctrl_wl_wr_state {
struct ctrl_v_omega_state {
    // desired value
    double v_dsr;
    double omega_dsr;
};

struct ctrl_v_theta_state {
    // desired value
    double v_dsr;
    double theta_dsr;
    double w_l_dsr;
    double w_r_dsr;
    double u_v;
    double u_omega;
    double error_theta;
    double error_theta_p;
    double error_theta_sum_p;
    double u_omega_rf1_p;
    double u_omega_rf2_p;
    // debug
    double up_omega_out;
    double ui_omega_out;
    double ud_omega_out;
    double u_omega_wo_rf; // without roll off
    FILE *log_file;
    FILE *recent_log_file;
};

struct ctrl_x_y_state {
    // outer loop x_y control
    struct ctrl_x_y_para {
        double kp_dist;
        double ki_dist;
        double kd_dist;
    };
}
double kp_angle;
double ki_angle;
double kd_angle;
}

struct ctrl_x_y_state {
  // desired value
  double x_dsr;
  double y_dsr;
  double wl_dsr;
  double wr_dsr;
  double u_v;
  double u_omega;
  double error_dist;
  double error_dist_p;
  double error_dist_sum_p;
  // double u_v_rfl_p;
  // double u_v_rfl_p;
  double error_angle;
  double error_angle_p;
  double error_angle_sum_p;
  // double u_omega_rfl_p;
  // double u_omega_rfl_p;
  // debug
  double up_dist_out;
  double ui_dist_out;
  double ud_dist_out;
  double up_angle_out;
  double ui_angle_out;
  double ud_angle_out;
  FILE *log_file;
  FILE *recent_log_file;
};

// outer loop delta_x_theta

struct ctrl_delta_x_theta_para {
  double kp_delta_x;
  double ki_delta_x;
  double kd_delta_x;
  // NOTE: Currently only kp_theta is in usage
  double kp_theta;
  double ki_theta;
  double kd_theta;
};

struct ctrl_delta_x_theta_state {
  // desired value
  double delta_x_dsr;
  double theta_dsr;
  // previous state and current intermediate state
  double wl_dsr;
  double wr_dsr;
double u_v;
double u_omega;

double error_delta_x;
double error_delta_x_p;
double error_delta_x_pp;
double u_v_rfl_p;
double u_v_rfl2_p;
double u_v_out_p;

double error_theta;
double error_theta_p;
double error_theta_sum_p;
double u_omega_rfl_p;
double u_omega_rfl2_p;

// debug state
double up_v_out;
double ui_v_out;
double ud_v_out;
double u_v_out;
double up_omega_out;
double ui_omega_out;
double ud_omega_out;
double u_omega_out;

FILE *log_file;
FILE *recent_log_file;

};

// outer loop line track
struct ctrl_line_track_para {

double kp;
double ki;
double kd;
};

struct ctrl_line_track_state {

// desired value
double v_dsr;
double w_dsr;
double wr_dsr;
double u_v;
double u_omega;
double delta_theta;
double delta_theta_p;
double delta_theta_sum_p;
double u_omega_rfl_p;
double u_omega_rfl2_p;

// debug
double up_omega_out;
double ui_omega_out;
double ud_omega_out;
double u_omega_out;

FILE *log_file;
FILE *recent_log_file;
}

// platooning

struct ctrl_platooning_para {
    int leader_id;
    double kp_delta_x;
    double ki_delta_x;
    double kd_delta_x;
    double kp_ffleader;
    double ki_ffleader;
    double kd_ffleader;
    double ka_ffleader;
    double kp_ffpre;
    double ki_ffpre;
    double kd_ffpre;
    double ka_ffpre;
    double kp_theta;
    double ki_theta;
    double kd_theta;
};

struct ctrl_platooning_state {
    double delta_x_dsr;
    double theta_dsr;
    double v_dsr_leader;
    double v_leader;
    double v_leader_p;
    double acc_leader;
    int ctrl_flag_leader;
    double v_pre;
    double acc_pre;
    double x_leader_p;
    double x_leader;
    double wl_dsr;
    double wr_dsr;
    double u_v;
    double u_rf_p;
    double u_omega;
    double delta_x;
    double u_v_x;
    double error_delta_x;
double error_delta_x_p;
double error_delta_x_pp;
double delta_x_filtered;
double u_v_x_rf1_p;
double u_v_x_rf2_p;

// debug state (delta_x)
double up_v_x_out;
double ui_v_x_out;
double ud_v_x_out;
double u_v_x_out_p;
double u_v_x_out;

// state derived from leader
double u_v_ffleader;
double error_leader;
double error_leader_p;
// debug error_leader_pp;
double error_leader_sum_p;
double u_v_ffleader_rf_p;
double u_v_ffleader_acc;

// debug state (leader)
double up_v_ffleader_out;
double ui_v_ffleader_out;
double ud_v_ffleader_out;
double u_v_ffleader_out_p;
double u_v_ffleader_out;
double u_v_ffleader_rf_before_sat;

// state derived from previous vehicle
double u_v_ffpre;
double error_pre;
double error_pre_p;
double error_pre_pp;
double u_v_ffpre_rf_p;
double u_v_ffpre_acc;

// debug state (leader)
double up_v_ffpre_out;
double ui_v_ffpre_out;
double ud_v_ffpre_out;
double u_v_ffpre_out_p;
double u_v_ffpre_out;

// state derived from theta
double error_theta;
double error_theta_p;
double error_theta_sum_p;
double u_omega_rf1_p;
double u_omega_rf2_p;

double u_omega_out;
double ui_omega_out;
double ud_omega_out;
double u_omega_worf;
FILE *log_file;
FILE *recent_log_file;

};

// exported global variable

extern volatile int ctrl_loop_should_stop;
extern sem_t ctrl_loop_sem;
extern int ctrl_loop_current_rt;

// vehicle state

extern volatile struct vehicle_state vstate;

// states and parameter of inner loop

extern volatile struct ctrl_inner_para ctrl_inner_para;

// states and parameter of outer loop

extern volatile struct ctrl_open_loop_state ctrl_open_loop_state;
extern volatile struct ctrl_wl_wr_state ctrl_wl_wr_state;

extern volatile struct ctrl_v_omega_state ctrl_v_omega_state;
extern volatile struct ctrl_v_theta_para ctrl_v_theta_para;
extern volatile struct ctrl_v_theta_state ctrl_v_theta_state;

extern volatile struct ctrl_x_y_para ctrl_x_y_para;
extern volatile struct ctrl_x_y_state ctrl_x_y_state;

extern volatile struct ctrl_delta_x_theta_para ctrl_delta_x_theta_para;
extern volatile struct ctrl_delta_x_theta_state ctrl_delta_x_theta_state;

extern volatile struct ctrl_line_track_para ctrl_line_track_para;
extern volatile struct ctrl_line_track_state ctrl_line_track_state;

extern volatile struct ctrl_platooning_para ctrl_platooning_para;
extern volatile struct ctrl_platooning_state ctrl_platooning_state;

void update_vehicle_state(struct h2l_vehicle_status *state);
void *ctrl_loop_thread(void *para);
void ctrl_setup();
void ctrl_start();
#include <stdint.h>
#include <string.h>

struct mc_init;

#if defined H2L_PROTOCOL_H
#define H2L_PROTOCOL_H
#endif

#define MC_PROTO_HEADER_SIZE 4

#define SERIAL_STATE_INIT 0
#define SERIAL_STATE_MAGIC1 1
#define SERIAL_STATE_MAGIC2 2
#define SERIAL_STATE_PROTO 3

#define SERIAL_MAGIC1 'A'
#define SERIAL_MAGIC2 'F'

#define OPCODE_OPEN_LOOP 0x00
#define OPCODE_CTRL_WL_WR 0x10
#define OPCODE_SETUP 0xF0
#define OPCODE_START 0xF1
#define OPCODE_STOP 0xF2
#define OPCODE_DEBUG_ENABLE 0xF3
#define OPCODE_DEBUG_DISABLE 0xF4

#define OPCODE_VEHICLE_STATUS 0xE0
#define OPCODE_CTRL_STATUS_DEBUG 0xE1

struct h2l_header {
    uint8_t magic1;
    uint8_t magic2;
    uint8_t len;
    uint8_t opcode;
};

// message from HLC to LLC

struct h2l_open_loop {
    struct h2l_header header;
    int16_t v_left;
};

void ctrl_stop();
void ctrl_open_loop_set_pwm(int16_t pwm_left, int16_t pwm_right);
void ctrl_set_vomega(double v_dsr, double omega_dsr);
void ctrl_set_vtheta(double v_dsr, double theta_dsr);
void ctrl_set_x_y(double x_dsr, double y_dsr);
void ctrl_set_delta_x_theta(double delta_x_dsr, double theta_dsr);
void ctrl_set_line_track(double v_dsr);
void ctrl_set_platooning(double delta_x_dsr, double theta_dsr);

double ctrl_outer_loop_threshold_and_saturation_v
    (double In, double Th, double Min, double Max);
double ctrl_outer_loop_threshold_and_saturation_w
    (double In, double Th, double Min, double Max);

#endif /* CTRL_LOOP_H */
struct h2l wl_wr {
    struct h2l_header header;
    float wl_dsr;
    float wr_dsr;
};

struct h2l setup {
    struct h2l_header header;
    float roll_off_coefficient;
    float prefilter_coefficient;
    float kp_left;
    float ki_left;
    float kd_left;
    float kp_right;
    float ki_right;
    float kd_right;
    float deadzone_threshold;
    float deadzone_saturation;
};

// message from LLC to HLC

struct h2l vehicle_status {
    struct h2l_header header;
    uint32_t timestamp;
    float imu_theta;   // from IMU
    float imu_accx;   // from IMU
    float imu_omega;   // from IMU
    float encoder wl;
    float encoder wr;
    float ultrasonic_delta_x;   // from ultrasonic sensor
};

struct h2l ctrl status debug {
    struct h2l_header header;
    uint32_t timestamp;   // Arduino timestamp in ms
    float wl_dsr;
    float wr_dsr;
    float wl_dsr_filtered;
    float wr_dsr_filtered;

    // debug
    int16_t pwml;   // with roll off
    int16_t pwmr;
    int16_t pwml_out;  // without roll off
    int16_t pwmr_out;
    int16_t pwml_out_p;
    int16_t pwmr_out_p;
}
// helper function

static inline void set_uint16(char *buff, int offset, uint16_t value) {
    char *p = (char *)&value;
    memcpy(buff + offset, p, sizeof(value));
}

static inline void set_uint32(char *buff, int offset, uint32_t value) {
    char *p = (char *)&value;
    memcpy(buff + offset, p, sizeof(value));
}

static inline void set_float(char *buff, int offset, float value) {
    char *p = (char *)&value;
    memcpy(buff + offset, p, sizeof(value));
}

static inline uint16_t get_uint16(char *buff, int offset) {
    uint16_t ret;
    uint16_t *p = &ret;
    memcpy(p, buff + offset, sizeof(ret));
    return ret;
}

static inline uint32_t get_uint32(char *buff, int offset) {
    uint32_t ret;
    uint32_t *p = &ret;
    memcpy(p, buff + offset, sizeof(ret));
    return ret;
}

static inline float get_float(char *buff, int offset) {
    float ret;
    float *p = &ret;
    memcpy(p, buff + offset, sizeof(ret));
    return ret;
}
static inline void h2l_set_header(struct h2l_header *pheader, uint8_t len, uint8_t opcode) {
  pheader->magic1 = SERIAL_MAGIC1;
  pheader->magic2 = SERIAL_MAGIC2;
  pheader->len = len;
  pheader->opcode = opcode;
}

// send command message to LLC

// CAUTION: only call these function in control loop
int h2l_send_open_loop_msg(int16_t pwm_left, int16_t pwm_right);
int h2l_send_wl_wr_msg(float wl_dsr, float wr_dsr);
int h2l_send_setup_msg();
int h2l_send_start_msg();
int h2l_send_stop_msg();
int h2l_send_debug_enable_msg();
int h2l_send_debug_disable_msg();

// void h2l_print_vehicle_status(FILE *fd);
// struct h2l_vehicle_status *msg);
void h2l_print_ctrl_status_debug(FILE *fd,
  struct h2l_ctrl_status_debug *msg);

// h2l receiving thread
//(note that the sending thread is control loop)
void *h2l_receive_thread(void *para);

extern volatile int serial_should_stop;
extern int serial_fd;

#endif /* H2L PROTOCOL_H_ */

#ifndef PI_MC_H_
#define PI_MC_H_

struct mc_init {
  int vehicle_id;
  char tty_file_name[64];
  int tty_baud_rate;
  char log_file_name[64];
  char meta_file_name[64];
  // v2i and v2v parameter
  char manager_ip[16];
  int manager_port;
  int v2v_period;
  // general controller parameter
  int running_time; // in milliseconds

#endif /* PI_MC_H_ */
int ctrl_loop_period; // in milliseconds
int debug_mode;
double wheel_radius;
double distance_of_wheels;
};
extern struct mc_init mc_init;
extern FILE *meta_file;
extern FILE *log_file;
extern FILE *recent_log_file;
extern FILE *recent_meta_file;
#endif /* PI_MC_H */

#include <stdio.h>
#endif SERIAL_H_
#define SERIAL_H_
int serial_init(const char* serialport, int baud);
int serial_close(int fd);

// CAUTION: these send and receiving are blocking
// busy waiting
int serial_send_n_bytes(int fd, const char* buff, size_t n);
int serial_recv_n_bytes(int fd, char *buff, size_t n);
int serial_flush(int fd);

// CAUTION: this is nonblocking receive
int serial_recv_byte(int fd, char *buff);
#endif /* SERIAL_H_ */

#ifndef V2V_H_
#define V2V_H_

struct v2v_msg {
  uint32_t ts;
  uint32_t vid;
  double theta;
  double accx;
  double v;
  double omega;
  double v_dsr;
  int ctrl_flag;
};

struct v2v_msg *get_recent_msg(int vid);
void print_v2v_msg(FILE *fd, struct v2v_msg *msg);
void *v2v_sending_thread( void *para);
void *v2v_receiving_thread( void *para);
extern volatile int v2v_sending_should_stop;
extern volatile int v2v_receiving_should_stop;

extern sem_t v2v_send_sem;

#SBATCH

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>
#include <fstream>
#include <ctime>
#include <unistd.h>
#include "raspicam/raspicam_cv.h"
#include "ctrl_loop.h"
#include "pi.mc.h"
#include "camera.h"

using namespace cv;
using namespace raspicam;
using namespace std;

struct camera_para cpara;
volatile int camera_should_stop;

long get_millis() {
    struct timespec spec;
    clock_gettime(CLOCK_MONOTONIC, &spec);
    long ms = spec.tv_sec * 1000 + spec.tv_nsec / 1000000;
    return ms;
}

/**Sets a property in the VideoCapture.
 * Implemented properties:
 * CV_CAP_PROP_FRAME_WIDTH, CV_CAP_PROP_FRAME_HEIGHT,
 * CV_CAP_PROP_FORMAT: CV_8UC1 or CV_8UC3
 * CV_CAP_PROP_BRIGHTNESS: [0,100]
 * CV_CAP_PROP_CONTRAST: [0,100]
 * CV_CAP_PROP_SATURATION: [0,100]
 * CV_CAP_PROP_GAIN: (iso): [0,100]
 * CV_CAP_PROP_EXPOSURE: -1 auto. [1,100]
 * //shutter speed from 0 to 33ms
 * CV_CAP_PROP_WHITE_BALANCE_RED_V: /[1,100] -1 auto whitebalance
 * CV_CAP_PROP_WHITE_BALANCE_BLUE_U: /[1,100] -1 auto whitebalance
 * */

// LEE
// 0725 edited the original library add new set feature by
// adding two more cases in switch section in raspicam_cv.cpp

void set_picam_property(RaspiCam_Cv &picam) {
picam.set(CV_CAP_PROP_FRAME_WIDTH, 320);
picam.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
picam.set(CV_CAP_PROP_FORMAT, CV_8UC3);
    picam.set(777, 1); // HFlip
    picam.set(888, 1); // VFlip
// picam.set(CV_CAP_PROP_BRIGHTNESS, 50);
// picam.set(CV_CAP_PROP_CONTRAST, 50);
// picam.set(CV_CAP_PROP_SATURATION, 50);
// picam.set(CV_CAP_PROP_GAIN, 50);
// picam.set(CV_CAP_PROP_EXPOSURE, -1);
// picam.set(CV_CAP_PROP_WHITE_BALANCE_RED_V, -1);
}

double delta_theta bk = 0;
void *camera_thread(void *para) {
    ofstream camlog("camera.txt");
    RaspiCam_Cv picam;
    set_picam_property(picam);
    if (!picam.open()) {
        cerr << "Error opening camera" << endl;
    }
    cout << "<camera> Connected to camera =" << picam.getId() << endl;
    double frame_width = picam.get(CV_CAP_PROP_FRAME_WIDTH);
    double frame_height = picam.get(CV_CAP_PROP_FRAME_HEIGHT);
    cout << "Frame Size = " << frame_width << "x" << frame_height << endl;
    cout << "ROI Size = " << para.roi_width << "x" << para.roi_height << endl;

    Size frame_size(static_cast<int>(frame_width),
                     static_cast<int>(frame_height));
    VideoWriter writer("./camera.avi",
                       CV_FOURCC('P', 'I', 'M', '1'),
                       20, frame_size, true);
    // initialize the VideoWriter object
    if (!writer.isOpened())
        cout << "ERROR: Failed to write the frame video" << endl;
    return NULL;
}

Size roi_size(static_cast<int>(para.roi_width),
              static_cast<int>(para.roi_height));

VideoWriter roi_writer("./roi.avi",
                      CV_FOURCC('P', 'I', 'M', '1'),
                      20, roi_size, true);
    // initialize the VideoWriter object
    if (!roi_writer.isOpened())
        cout << "ERROR: Failed to write the ROI video" << endl;
    return NULL;}
// namedWindow("Control", CV_WINDOW_AUTOSIZE);
// createTrackbar("threshold", "Control", &th, 255);
long ts = get_millis();

while (!camera_should_stop) {
    Mat frame;
    Mat imgray;
    Mat imthresh;

    picam.grab();
    picam.retrieve(frame);

    // convert to grey scale
    cvtColor(frame, imgray, CV_RGB2GRAY);
    // binary threshold
    threshold(imgray, imthresh, cpara.th, 255,
             THRESH_BINARY_INV);

    // get subimage
    Rect rect(80, 20, 160, 100);
    Rect rect(cpara.roi_x, cpara.roi_y,
              cpara.roi_width, cpara.roi_height);
    Mat roi = imthresh(rect);

    // morphological opening
    // (remove small objects from the foreground)
    erode(roi, roi,
          getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
    dilate(roi, roi,
           getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));

    // morphological closing (fill small holes in the foreground)
    dilate(roi, roi,
           getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
    erode(roi, roi,
           getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));

    // calculate the moments of the thresholded image
    Moments m = moments(roi);
    double m01 = m.m01;
    double m10 = m.m10;
    double m00 = m.m00;

    double u1 = m10 / m00;
    double v1 = m01 / m00;

    // double width = cpara.roi_width;
    // double height = cpara.roi_height;

    // double delta_theta_rad = atan((u1 - width / 2) / height);
    // double delta_theta_degree = delta_theta_rad * 180 / 3.1416;

    double width = cpara.roi_width;
    double height = cpara.roi_height;

    // 32 = 20 (Look ahead) + 12 (rear wheel 2 camera)
    // vertical distance calibration 32cm
    double lat_pixel_2_cm_gain = (1.7/26.0);
    double lat_err_cm = (u1 - width / 2)* lat_pixel_2_cm_gain;
    double delta_theta_rad = -atan(lat_err_cm / 32.0);
// camera failure handling
// 0.32 is the maximum delta_theta we can get
// given 20cm look ahead distance
if (fabs(delta_theta_rad)>0.32) {
    cout << "camera off track" << endl;
delta_theta_rad = delta_theta_bk;
}
else if (u1>10 && u1<=310) {
    //update backup data
delta_theta_bk = delta_theta_rad;
}
else if (u1<10 || u1>310) {
    //cout "<camera> error " << endl;
delta_theta_rad = 0;
delta_theta_bk = 0;
} else {
}
double delta_theta_degree = delta_theta_rad * 180 / 3.1416;

// show the ROI and the image
rectange(frame, rect, Scalar(127, 127, 127));
imshow("original", frame); //show the original image
imshow("roi", roi);
long t2 = getmillis();
camlog << "FPS: " << 1000 / (t2 - ts) << ", ";
camlog << "center of mass: (" << u1 << ", " << v1;
camlog << ") delta_theta: " << delta_theta_rad << ");"
camlog << "lat_err_cm:" <<lat_err_cm<<endl;
ts = t2;
vstate.camera_delta_theta = delta_theta_rad;
Mat roi_rgb;
cvtColor(roi, roi_rgb, CV_GRAY2RGB);
writer.write(frame);
roi_writer.write(roi_rgb);
// sleep 20 ms
usleep(20000);
}
picam.release();
cout << "camera thread exiting...
return 0;
}
#include <unistd.h>
#include <sys/types.h>
#include "h2l.h"
#include "ctrl_loop.h"
#include "v2v.h"
#include "p1mc.h"

volatile int ctrl_loop_should_stop = 0;
sem_t ctrl_loop_sem;

int ctrl_loop_current_rt = 0;

// ------ vehicle state and controller state ------------
volatile struct vehicle_state vstate;

// states and parameter of inner loop
volatile struct ctrl inner para ctrl inner para;

// states and parameter of outer loop
volatile struct ctrl open loop state ctrl open loop state;
volatile struct ctrl w1 wr state ctrl w1 wr state;
volatile struct ctrl v omega state ctrl v omega state;
volatile struct ctrl v theta para ctrl v theta para;
volatile struct ctrl v theta state ctrl v theta state;
volatile struct ctrl x y para ctrl x y para;
volatile struct ctrl x y state ctrl x y state;
volatile struct ctrl delta x theta para ctrl delta x theta para;
volatile struct ctrl delta x theta state ctrl delta x theta state;
volatile struct ctrl line track para ctrl line track para;
volatile struct ctrl line track state ctrl line track state;
volatile struct ctrl platooning para ctrl platooning para;
volatile struct ctrl platooning state ctrl platooning state;

// LEE 0726 new global variable
int delta x neg error flag = 0;
int delta x neg error flag p = 0;

int started flag = 0;
int should stop flag = 0;

// controller bypass flags
// be cautious when use it
int not start bypass = 1;
int stop bypass = 0;
int too close bypass = 0;
int brake bypass = 0;

// general system flags
int vehicle status = 0;
int separation too close flag = 0;
int follower caught up flag = 0;
int leader stopped flag = 0;

long running time start ms = 0;

// store start time
long run_lapsed_time_ms = 0;
int negative_flag_cnt = 0;
// flag up when delta_x error is negative
// ----------------- vehicle state utility -----------------

void kinematics(double wl, double wr,
double *v, double *omega) {
    *v = (wl + wr) * mc_init.wheel_radius / 2;
    *omega = (wr - wl) * mc_init.wheel_radius
    / mc_init.distance_of_wheels;
}

void inverse_kinematics(double v, double omega,
double *wl, double *wr) {
    *wl = (2 * v - mc_init.distance_of_wheels * omega)
    / (2 * mc_init.wheel_radius);
    *wr = (2 * v + mc_init.distance_of_wheels * omega)
    / (2 * mc_init.wheel_radius);
}

void update_vehicle_state(struct h2l_vehicle_status *state) {
    vstate.timestamp = state->timestamp;
    vstate.imu_theta = state->imu_theta;
    vstate.imu_accx = state->imu_accx;
    vstate.imu_omega = state->imu_omega;
    vstate.encoder_wl = state->encoder_wl;
    vstate.encoder_wr = state->encoder_wr;
    vstate.ultrasonic_delta_x = state->ultrasonic_delta_x;
    // calculate / fuse states
    vstate.accx = vstate.imu_accx;
    kinematics(vstate.encoder_wl, vstate.encoder_wr,
(double *) &vstate.v,
(double *) &vstate.omega);
    // derive x and y through dead reckoning
    // FIXIT: imu_theta is absolute theta derived
    // from IMU in XI-O-YI coordinates
    static int integration_error_print_flag = 0;
    vstate.theta = vstate.imu_theta;
    // sometimes abnormal initial error happens
    // deal with them carefully
    if (fabs(vstate.x) > 100) vstate.x = 0;
    vstate.x += vstate.v * cos(vstate.theta)
    / 1000.0;
    if((vstate.x < 0 ){
        integration_error_print_flag++;
        if (1 == integration_error_print_flag)
            printf("Integration of X Error!\n x: 
%6.2f, theta: %6.2f",
                vstate.x, vstate.theta);
    }
    vstate.y += vstate.v * sin(vstate.theta)
    / 1000.0;
// outer loop utility

// PID controller absolute style
double ctrl_pid_with_d_rolloff(
    double err, double err_sum, double err_p,
    double kp, double ki, double kd,
    double ud_lpf_n, double *ud_save,
    double ud_p, double ts)
{
    double up = kp * err;
    double ui = ki * ts * err_sum;
    // Apply low pass filter for ud, so that it //
    // is less sensitive to noise.
    double alpha = 1.0 / (1.0 + ud_lpf_n * ts);
    double ud = alpha * ud_p + (1 - alpha) *
        *ud_save = ud;
    double u = up + ui + ud;
    return u;
}

// PID controller position style
// using back Euler discretization
double ctrl_pid(double err, double err_sum, double err_p,
    double kp, double ki, double kd,
    double *up_out, double *ui_out, double *ud_out,
    double ts)
{
    double up = kp * (err - err_p) / ts;
    double ui = ki * ts * err;
    double ud = kd * ((err - err_p) - (err_p - err_pp)) / ts;
    double ud_save = ud;
    double u = up + ui + ud;
    return u;
}

// PID controller incremental style using
// back Euler discretization
double ctrl_pid_inc(double up, double err,
    double err_p, double err_pp,
    double kp, double ki, double kd,
    double *up_out, double *ui_out,
    double *ud_out, double ts)
{
    double up = kp * (err - err_p);
    double ui = ki * ts * err;
    double ud = kd * ( (err - err_p) - (err_p - err_pp) ) / ts;
    double ud_temp = ud;
    double deltau = up + ui + ud;
    double u = up + deltau;
    if (up_out != NULL)
        *up_out = up;
    if (ui_out != NULL)
        *ui_out = ui;
    if (ud_out != NULL)
        *ud_out = ud;
    return u;
}
const double rf_coeff = 0.8; // backward Euler 40/(s+40)

double ctrl_roll_off_once(double u_rf_p, double u_in, double rf_coeff) {
    double u_rf = (1 - rf_coeff) * u_rf_p + rf_coeff * u_in;
    return u_rf;
}

// wr wl saturate utility function
const double wl_wr_sat_lower_bound = 0;
const double wl_wr_sat_upper_bound = 20;

// CAUTION: this is just a temporary solution
void ctrl_sat_wl_wr(double *wl, double *wr) {
    if (*wl < wl_wr_sat_lower_bound) {
        *wl = wl_wr_sat_lower_bound;
    } else if (*wl > wl_wr_sat_upper_bound) {
        *wl = wl_wr_sat_upper_bound;
    }
    if (*wr < wl_wr_sat_lower_bound) {
        *wr = wl_wr_sat_lower_bound;
    } else if (*wr > wl_wr_sat_upper_bound) {
        *wr = wl_wr_sat_upper_bound;
    }
}

// control effort restriction
double ctrl_threshold_and_saturation(double in, double th, double min, double max) {
    double out = in;
    // threshold means no response region is [-th, th]
    if (fabs(in) < th) {
        out = 0;
    } else if (in > max) {
        out = max;
    } else if (in < min) {
        out = min;
    }
    return out;
}

// outer loop controller

// outer loop framework
// 1) information gathering from sensor or communication
// 2) calculate error
// 3) pid control + roll off
// 4) control effort restriction (saturation, deadzone)
// 5) obtain wl_dsr, wr_dsr through inverse kinematics
// 6) states update
// when bypass mechanism involved things becomes
// more complicated final cmd is bypass controller
// and controlled by different bypass flags triggered
void ctrl_open_loop() {
  h2l_send_open_loop_msg(ctrl_open_loop_state.pwm_left_dsr,
                         ctrl_open_loop_state.pwm_right_dsr);
}

void ctrl_wl_wr_loop() {
  h2l_send_wl_wr_msg(ctrl_wl_wr_state.wl_dsr,
                     ctrl_wl_wr_state.wr_dsr);
}

// direct v_omega
void ctrl_v_omega_loop() {
  double wl_dsr;
  double wr_dsr;
  inverse_kinematics(ctrl_v_omega_state.v_dsr,
                      ctrl_v_omega_state.omega_dsr,
                      &wl_dsr, &wr_dsr);
  // send to inner loop:
  h2l_send_wl_wr_msg((float)wl_dsr, (float)wr_dsr);
}

// -------- (v, theta) control --------
void ctrl_v_theta_init() {
  char fn_buff[64];
  char suffix[128];
  time_t t = time(NULL);
  struct tm tm = *localtime(&t);
  snprintf(suffix, sizeof(suffix),
           "%04d_%02d-%02d_%02d_%02d",
           tm.tm_year + 1900,
           tm.tm_mon + 1,
           tm.tm_mday,
           tm.tm_hour,
           tm.tm_min,
           tm.tm_sec);
  snprintf(fn_buff, sizeof(fn_buff),
           "v_theta_%s.txt", suffix);
  ctrl_v_theta_state.log_file = fopen(fn_buff, "w+");
  ctrl_v_theta_state.recent_log_file = fopen("v_theta.txt", "w+");
}

void ctrl_v_theta_exit() {
  fclose(ctrl_v_theta_state.log_file);
  fclose(ctrl_v_theta_state.recent_log_file);
}

void ctrl_v_theta_print(FILE *fd) {
  struct timespec now;
  clock_gettime(CLOCK_MONOTONIC, &now);
}
```c
uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;

// print vehicle state first
fprintf(fd, "\t%08u, %08u, ",
    "% 6.2f, %6.2f, %6.2f, % 6.2f, %6.2f, \t\t",
    ts, vstate.timestamp,
    vstate.ultrasonic_delta_x,
    vstate.encoder_wl, vstate.encoder_wr,
    vstate.v, vstate.omega,
    vstate.x, vstate.y, vstate.theta);

// controller states
fprintf(fd, "\t% 6.2f, %6.2f, %6.2f, %6.2f,",
    ctrl_v_theta_state.wl_dsr,
    ctrl_v_theta_state.wr_dsr,
    ctrl_v_theta_state.u_v,
    ctrl_v_theta_state.u_omega,
    ctrl_v_theta_state.v_dsr,
    ctrl_v_theta_state.theta_dsr);

// error value
fprintf(fd, "\t% 6.2f, %6.2f, %6.2f, \t\t",
    ctrl_v_theta_state.error_theta,
    ctrl_v_theta_state.error_theta_p,
    ctrl_v_theta_state.error_theta_sum_p);

// debug value
fprintf(fd, "\t% 6.2f, %6.2f, %6.2f, %6.2f\n",
    ctrl_v_theta_state.up_omega_out,
    ctrl_v_theta_state.ui_omega_out,
    ctrl_v_theta_state.ud_omega_out,
    ctrl_v_theta_state.u_omega_wor);

fflush(fd);
}

void ctrl_v_theta_loop() {

    // calculate error
    // sum update before PID because
    // I_term(k) = sum(i=1\k) e(i)
    double error_theta =
        ctrl_v_theta_state.theta_dsr - vstate.theta;
    double error_theta_sum =
        double error_theta_sum_p+ error_theta;
    double error_theta_p = ctrl_v_theta_state.error_theta_p;

    double up_v_out;
    double ui_v_out;
    double ud_v_out;

    //PID controller main body
    // NOTE: v is not controlled here.
    double u_v = ctrl_v_theta_state.v_dsr;

    double up_omega_out;
    double ui_omega_out;
    double ud_omega_out;

    double u_omega_out = ctrl_pid(
        error_theta, error_theta_sum, error_theta_p,
        ctrl_v_theta_para.kp_theta,
        ctrl_v_theta_para.ki_theta,
```
ctrl_v_theta:notepad_kd_theta,
&up_omega_out, &ui_omega_out,
&ud_omega_out, mc_init.ctrl_loop_period / 1000.0);

// NOTE: If you has d-term then roll off twice,
// (low pass filter)
double u_omega_rfl = ctrl_roll_off_once
(ctrl_v_theta_state.u_omega_rfl_p,
 u_omega_out, rf_coeff);
double u_omega_rf2 = ctrl_roll_off_once
(ctrl_v_theta_state.u_omega_rf2_p,
 u_omega_rfl, rf_coeff);

// outer loop control effort restriction
// not restriction for u_v
double u_omega =
ctrl_threshold_and_saturation(u_omega_rf2, 0, -2, 2);
double wl_dsr;
double wr_dsr;
inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);

// send to inner loop;
h2l_send_wl_wr_msg((float) wl_dsr, (float) wr_dsr);

// update current states
ctrl_v_theta_state.error_theta = error_theta;
ctrl_v_theta_state.wl_dsr = wl_dsr;
ctrl_v_theta_state.wr_dsr = wr_dsr;
ctrl_v_theta_state.u_v = u_v;
ctrl_v_theta_state.u_omega = u_omega;

// update debug states
ctrl_v_theta_state.up_omega_out = up_omega_out;
ctrl_v_theta_state.ui_omega_out = ui_omega_out;
ctrl_v_theta_state.ud_omega_out = ud_omega_out;
ctrl_v_theta_state.u_omega_0_rfl = u_omega_out;

// print control state to file
ctrl_v_theta_print(ctrl_v_theta_state.log_file);
ctrl_v_theta_print(ctrl_v_theta_state.recent_log_file);

// update previous states
ctrl_v_theta_state.error_theta_p = error_theta;
ctrl_v_theta_state.error_theta_sum_p = error_theta_sum;
ctrl_v_theta_state.u_omega_rfl_p = u_omega_rfl;
ctrl_v_theta_state.u_omega_rf2_p = u_omega_rf2;

}

// ____________ (x, y) control ____________

void ctrl_x_y_init () {
    char fn_buff[64];
    char suffix[128];
    time_t t = time(NULL);
    struct tm tm = * localtime(&t);
    snprintf(suffix, sizeof(suffix),
            "%04d.%02d.%02d...%02d.%02d.%02d",
            tm.tm_year + 1900,
            tm.tm_mon + 1,
            tm.tm_mday,
tm.tm_hour, 
    tm.tm_min, 
    tm.tm_sec); 

snprintf(fn_buff, sizeof(fn_buff), "x_y_%s.txt", suffix); 

ctrl_x_y_state.log_file = fopen(fn_buff, "w+"); 
ctrl_x_y_state.recent_log_file = fopen("x_y.txt", "w+"); } 

void ctrl_x_y_exit() { 
    fclose(ctrl_x_y_state.log_file); 
    fclose(ctrl_x_y_state.recent_log_file); 
} 

void ctrl_x_y_print(FILE *fd) { 

    struct timespec now; 
    clock_gettime(CLOCK_MONOTONIC, &now); 

    uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000; 

    // print vehicle state first
    printf(fd, "%08u %08u,
        "%6.2f, %6.2f, %6.2f, %6.2f, %6.2f, \t\t",
        ts, vstate.timestamp,
        vstate.ultrasonic_delta_x, vstate.encoder_wr, vstate.encoder_wr, vstate.v, vstate.omega,
        vstate.x, vstate.y, vstate.theta); 

    printf(fd, "%6.2f, %6.2f, %6.2f, %6.2f,",
        ctrl_x_y_state.wl_dsr, ctrl_x_y_state.wr_dsr, ctrl_x_y_state.u_v, ctrl_x_y_state.u_omega,
        ctrl_x_y_state.x_dsr, ctrl_x_y_state.y_dsr); 

    printf(fd, "%6.2f, %6.2f, %6.2f, %6.2f,",
        ctrl_x_y_state.error_dist, ctrl_x_y_state.error_dist_p, ctrl_x_y_state.error_dist_sum_p,
        ctrl_x_y_state.error_angle_p, ctrl_x_y_state.error_angle_sum_p); 

    printf(fd, "%6.2f, %6.2f, %6.2f, %6.2f,",
        ctrl_x_y_state.up_dist_out, ctrl_x_y_state.ui_dist_out, ctrl_x_y_state.ud_dist_out,
        ctrl_x_y_state.up_angle_out, ctrl_x_y_state.ui_angle_out, ctrl_x_y_state.ud_angle_out); 

    fflush(fd); 

} 

void ctrl_x_y_loop() { 
    int ret; 

    
215
double phi = atan((ctrl_x_y_state.y_dsr - vstate.y) / (ctrl_x_y_state.x_dsr - vstate.x));

double error_x = ctrl_x_y_state.x_dsr - vstate.x;
double error_y = ctrl_x_y_state.y_dsr - vstate.y;

double error_dist = error_x + fabs(error_y);
double error_dist_sqrt = sqrt(error_x * error_x + error_y * error_y);

// deadzone
if (error_dist_sqrt < 0.04 && error_dist_sqrt > -0.04) {
    error_dist = 0;
}

double error_dist_sum = ctrl_x_y_state.error_dist_sum_p + error_dist;
double error_dist_p = ctrl_x_y_state.error_dist_p;

double error_angle = phi - vstate.theta;

// deadzone
if ((error_x < 0.02 && error_x > -0.02) && (error_y < 0.02 && error_y > -0.02)) {
    error_angle = 0;
}

double error_angle_sum = ctrl_x_y_state.error_angle_sum_p + error_angle;
double error_angle_p = ctrl_x_y_state.error_angle_p;

double up_dist_out;
double ui_dist_out;
double ud_dist_out;
double up_angle_out;
double ui_angle_out;
double ud_angle_out;

double u_v = ctrl_pid(error_dist, error_dist_sum, error_dist_p, ctrl_x_y_para.kp_dist, ctrl_x_y_para.ki_dist, ctrl_x_y_para.kd_dist, &up_dist_out, &ui_dist_out, &ud_dist_out, mc_init.ctrl_loop_period / 1000.0);

double u_omega = ctrl_pid(error_angle, error_angle_sum, error_angle_p, ctrl_x_y_para.kp_angle, ctrl_x_y_para.ki_angle, ctrl_x_y_para.kd_angle, &up_angle_out, &ui_angle_out, &ud_angle_out, mc_init.ctrl_loop_period / 1000.0);

// saturation
u_v = ctrl_threshold_and_saturation(u_v, 0, -0.5, 0.5);
u_omega = ctrl_threshold_and_saturation(u_omega, 0, -3, 3);

double wl_dsr;
double wr_dsr;

inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);

// send to inner loop:
ret = h2l_send_wl_wr_msg((float)wl_dsr, (float)wr_dsr);
if (ret < 0) {
    printf("<ctrl_loop> Out of control!!!\n");
// update current state
ctrl_x_y_state.error_dist = error_dist;
ctrl_x_y_state.error_angle = error_angle;
ctrl_x_y_state.wl_dsr = wl_dsr;
ctrl_x_y_state.wr_dsr = wr_dsr;
ctrl_x_y_state.u_v = u_v;
ctrl_x_y_state.u_omega = u_omega;

// update debug states
ctrl_x_y_state.up_dist_out = up_dist_out;
ctrl_x_y_state.ui_dist_out = ui_dist_out;
ctrl_x_y_state.ud_dist_out = ud_dist_out;
ctrl_x_y_state.up_angle_out = up_angle_out;
ctrl_x_y_state.ui_angle_out = ui_angle_out;
ctrl_x_y_state.ud_angle_out = ud_angle_out;

// print control state to file
ctrl_x_y_print(ctrl_x_y_state.log_file);
ctrl_x_y_print(ctrl_x_y_state.recent_log_file);

// update previous state
ctrl_x_y_state.error_dist_p = error_dist;
ctrl_x_y_state.error_dist_sum_p = error_dist_sum;
ctrl_x_y_state.error_angle_p = error_angle;
ctrl_x_y_state.error_angle_sum_p = error_angle_sum;

// ---------- (delta_x, theta) control ----------

void ctrl_delta_x_thetainit() {
    char fn_buf[64];
    char suffix[128];
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    snprintf(suffix, sizeof(suffix), "%04d_%02d_%02d_%02d_%02d_%02d",
            tm.tm_year + 1900, tm.tm_mon + 1,
            tm.tm_mday, tm.tm_hour, tm.tm_min,
            tm.tm_sec);
    snprintf(fn_buf, sizeof(fn_buf), "delta_x_theta_%s.txt", suffix);
    ctrl_delta_x_theta_state.log_file = fopen(fn_buf, "w++");
    ctrl_delta_x_theta_state.recent_log_file =
        fopen("delta_x_theta.txt", "w++");
}

void ctrl_delta_x_theta_exit() {
    fclose(ctrl_delta_x_theta_state.log_file);
    fclose(ctrl_delta_x_theta_state.recent_log_file);
}

void ctrl_delta_x_theta_print(FILE *fd) {
    struct timespec now;
}
clock_gettime(CLOCK_MONOTONIC, &now);
uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;

// print vehicle state first
fprintf(fd, "%08u, %08u, "
    "% 6.2f, %6.2f, %6.2f, "
    "% 6.2f, %6.2f, % 6.2f, %6.2f, \t\t",
    ts, vstate.timestamp,
    vstate.ultrasonic_delta_x,
    vstate.encoder_wr, vstate.encoder_wr,
    vstate.x, vstate.y, vstate.theta);

// print control variable
fprintf(fd, "%6.4f, %6.4f, %6.4f, %6.4f, "
    "%6.4f, %6.4f, \t\t",
    ctrl_delta_x_theta_state.wl_dsr,
    ctrl_delta_x_theta_state.wr_dsr,
    ctrl_delta_x_theta_state.u_v,
    ctrl_delta_x_theta_state.u_omega,
    ctrl_delta_x_theta_state.delta_x_dsr,
    ctrl_delta_x_theta_state.theta_dsr);

fprintf(fd, "%6.4f, %6.4f, %6.4f, %6.4f,"
    "%6.4f, %6.4f, \t\t",
    ctrl_delta_x_theta_state.error_delta_x,
    ctrl_delta_x_theta_state.error_delta_x_p,
    ctrl_delta_x_theta_state.error_delta_x_pp,
    ctrl_delta_x_theta_state.error_theta,
    ctrl_delta_x_theta_state.error_theta_p,
    ctrl_delta_x_theta_state.error_theta_sum_p);

fprintf(fd, "%6.4f, %6.4f, %6.4f, %6.4f,"
    "%6.4f, %6.4f, \t\t",
    ctrl_delta_x_theta_state.up_v_out,
    ctrl_delta_x_theta_state.ui_v_out,
    ctrl_delta_x_theta_state.ud_v_out,
    ctrl_delta_x_theta_state.uv_out,
    ctrl_delta_x_theta_state.up_omega_out,
    ctrl_delta_x_theta_state.ui_omega_out,
    ctrl_delta_x_theta_state.ud_omega_out);

fprintf(fd, "\%d, \%d, \%d, \%d\n",
    started_flag,
    should_stop_flag,
    not_start_bypass,
    stop_bypass);
fflush(fd);
}

// NOTE: This outer loop only control delta_x
// it can be used as platoon separation control also
// when communication is not used
void ctrl_delta_x_theta_loop() {
    // get sensor reading
    double delta_x = vstate.ultrasonic_delta_x;

    if(fabs(delta_x - vstate.ultrasonic_delta_x_bk)>= 0.1
        && started_flag == 1){
        printf("ULTRASONIC ABNORMAL!\n");
    }
delta_x = vstate.ultrasonic_delta_x bk;
}

if (0 == delta_x || delta_x >= 3 ){
    delta_x = vstate.ultrasonic_delta_x bk;
    // set limit on sensor reading
    // to prevent unexpected error
    if (delta_x >= 2) delta_x = 1;
    printf("ULTRASONIC ERROR!\n");
}

// getting delta_theta by camera
double delta_theta = vstate.camera_delta_theta;

LEE 0623 define variable and initialize them to be 0
// if not in the normal operation status,
//they will be left as 0
double error_delta_x = 0;
double error_delta_x_p = 0;
double error_delta_x_pp = 0;
double error_theta = 0;
double error_theta_sum = 0;
double error_theta_p = 0;

// CAUTION: Here we use (delta_x - delta_x_dsrr).
// This is intentional.
// i.e. if delta_x is bigger than desired value, speed up.
// LEE 0623 adding new starts status checking
// by detecting rising edge of
// delta_x error
error_delta_x = delta_x - ctrl.delta_x_theta_state.delta_x_dsrr;
error_theta = delta_theta;
if (error_delta_x <= 0){
delta_x_neg_error_flag =1;
}
else{
delta_x_neg_error_flag =0;
}

// vehicle status checking
// stop status checking
// vehicle will stop if too close //
// to predecessor or off-track
// or reach pre-defined maximum run lapse time
// only works if
// running_time_MAX_ms < (running_time=20000 in init.conf)
long run_lapsed_time_ms = 0;
if (running_time_start_ms >0 ){
    run_lapsed_time_ms =
    vstate.timestamp - running_time_start_ms;
}

// after start and getting too close
static int stop_print_flag = 0;
if(delta_x <= 0.15 && started_flag == 1){
    //too close set stop flag
    // and reset flags and start time
    should_stop_flag = 1;
    stop_bypass++;
    delta_x_neg_error_flag_p = 1;
    delta_x_neg_error_flag = 1;
started_flag = 0;
running_time_start_ms = 0;
stop_print_flag++; if(1 == stop_print_flag){
    printf("DELTA X at %5.2f STOP!\n", delta_x);
}
static int off_track_flag = 0;
if(fabs(error_theta) >= 0.30){
    // off-track
    should_stop_flag = 1;
    stop_bypass++; off_track_flag++; if(1 == off_track_flag )
        printf("OFF TRACK AND STOP\n");
}
// start status checking
if (0 == started_flag){
    // if not start yet
    ctrl_delta_x_theta_state.error_delta_x_p = 0;
    ctrl_delta_x_theta_state.error_delta_x_pp = 0;
    ctrl_delta_x_theta_state.u_v_rf1_p = 0;
    ctrl_delta_x_theta_state.u_v_rf2_p = 0;
    ctrl_delta_x_theta_state.u_v_out_p = 0;
    ctrl_delta_x_theta_state.error_theta_p = 0;
    ctrl_delta_x_theta_state.error_theta_sum_p = 0;
    ctrl_delta_x_theta_state.u_omega_rf1_p = 0;
    ctrl_delta_x_theta_state.u_omega_rf2_p = 0;
    // error at beginning cross 0 from negative to positive
    // i.e. rising edge set started_flag on
    if (1 == delta_x_neg_error_flag_p & 0 == delta_x_neg_error_flag_p){
        started_flag = 1;
        not_start_bypass = 0;
        stop_bypass = 0;
        should_stop_flag = 0;
        separation_too_close_flag = 0;
        stop_print_flag = 0;
        // store start time
        running_time_start_ms = vstate.timestamp;
        printf("Vehicle Start at %ld \n", running_time_start_ms);
    } else{
        // if not start yet, set up not start bypass
        // flag and update delta_x_neg_error_flag_p
        not_start_bypass = 1;
    }
} else{ // to distinguish start before and after
    delta_x_neg_error_flag_p = 9;
    delta_x_neg_error_flag = 9;
}
// PID controller on delta_x, which derive u_v
double u_v_out=0;
double u_omega_out=0;
double up_v_out=0;
double ui_v_out=0;
double ud_v_out=0;
double up_omega_out=0;
double ui_omega_out=0;
double ud_omega_out=0;
// control effort restriction
double lowest_cruise_speed = 0.1;
double u_v;
double u_omega = 0;
double u_v_rf1 = 0;
double u_v_rf2 = 0;
double u_omega_rf1 = 0;
double u_omega_rf2 = 0;
double kdl_local = 0;
static int cancel_kd_flag = 0;
if (run_lapsed_time_ms > 3000)
{
    kdl_local = 0;
cancel_kd_flag++;
    if(1 == cancel_kd_flag)
    {
        printf("cancel kd term \r\n");
    }
} else if (1 == delta_x_neg_error_flag )
{
    kdl_local = 0;
    if(1 == started_flag)
    {
        cancel_kd_flag++;
        if(1 == cancel_kd_flag)
        {
            printf("cancel kd term \r\n");
        }
    }
} else{
    kdl_local = ctrl_delta_x_theta_para.kd_delta_x;
}
// normal operation case checking if pass then get
// relative errors
if(0 == should_stop_flag && 1 == started_flag)
{
    // error_delta_x and error_delta_theta
    // has been computed ahead
    error_delta_x_p = ctrl_delta_x_theta_state.error_delta_x_p;
    error_delta_x_pp = ctrl_delta_x_theta_state.error_delta_x_pp;
    error_theta_sum = ctrl_delta_x_theta_state.error_theta_sum_p
                     + error_theta;
    error_theta_p = ctrl_delta_x_theta_state.error_theta_p;
}
// normal operation pid incremental
// be careful about memory ctrl states u_v_out_p
u_v_out = ctrl_pid inc(
    ctrl_delta_x_theta_state.u_v_out_p,
    error_delta_x ,
    error_delta_x_p ,
    error_delta_x_pp ,
    ctrl_delta_x_theta_para.kp_delta_x ,
    ctrl_delta_x_theta_para.ki_delta_x ,
    kdl_local ,
    &u_v_out ,
    &ui_v_out ,
    &ud_v_out ,
    mc_init.ctrl_loop_period / 1000.0);
if(fabs(u_v_out)>3)
    printf("u_v_out error at %6.2f, ");
"P:%6.2f, I:%6.2f, D:%6.2f\n",
    u_v.out, up_v.out, ui_v.out, ud_v.out);
    //u_v.out = lowest_cruise_speed;
    u_v.out = ctrl_delta_x_theta.state.u_v.out_p;
    u_omega.out = 0;
  }
  }
  u_omega_out = ctrl_pid(
    error.theta,
    error.theta_sum,
    error.theta_p,
    ctrl_delta_x_theta_para.kp.theta,
    ctrl_delta_x_theta_para.ki.theta,
    ctrl_delta_x_theta_para.kd.theta,
    &upomega.out,
    &ui_omega_out,
    &ud_omega_out,
    mc_init.ctrl_loop_period / 1000.0
  );
}

  // controller roll off on the output v
  // (low pass filter)
  u_v.rf1 = ctrl_roll_off_once
    (ctrl_delta_x_theta.state.u_v.rf1_p, u_v.out, rf.coeff);
  u_v.rf2 = ctrl_roll_off_once
    (ctrl_delta_x_theta.state.u_v.rf2_p, u_v.rf1, rf.coeff);
}

  // controller roll off on the output omega
  // (low pass filter)
  u_omega.rf1 = ctrl_roll_off_once
    (ctrl_delta_x_theta.state.u_omega.rf1_p, u_omega.out, rf.coeff);
  u_omega.rf2 = ctrl_roll_off_once
    (ctrl_delta_x_theta.state.u_omega.rf2_p, u_omega.rf1, rf.coeff);
  }

  if(fabs(u_omega_out)>100){
    printf("u_omega_out error\n");
    u_omega_out = 0;
  }

  if(error_delta_x <= -0.03 && l==started_flag){
    negative_flag_cnt ++;
    u_v = 0;
    ctrl_delta_x_theta.state.u_v.out_p = 0;
    ctrl_delta_x_theta.state.u_v.rf1_p = 0;
    ctrl_delta_x_theta.state.u_v.rf2_p = 0;
    brake_bypass = (int)(fabs(error_delta_x)/0.01);
    printf("error delta_x %" "6.2f, brake time %d: \n",
      brake_bypass, negative_flag_cnt);
  }

  if(error_delta_x >= 0.03){
    u_v = ctrl_threshold_and_saturation
    (u_v.rf2, 0, 0, 0.8);
  }

  if(error_delta_x <= lowest_cruise_speed){
u_omega=0;

// normal operation but vehicle is
too close to predecessor
// look ahead distance has been obstructed
// by predecessor
// handle this unexpected sensor failure
// delta_x less than 30cm may cause
// unexpected camera reading error
// camera look ahead distance is set to 20cm
// too close look ahead distance vehicle
// will easily lose the track
if(delta_x <= 0.30){
  separation_too_close_flag++;
  if(separation_too_close_flag == 1){
    printf("! SEPARATION TOO CLOSE SENSOR 
    "CHANGE TO IMU TEMPORARILY !\r\n");
    error_theta = 0 - vstate.imu_theta;
    u_v = 0;
    u_omega = 0;
  }else{
    separation_too_close_flag = 0;
  }
}else{
// abnormal case dealing
// should reset ctrl memory states to zero !!
  u_vout = 0;
  u_omega_out = 0;
  u_v = 0;
  u_omega = 0;
  ctrl_delta_x_theta_state.error_delta_x_p = 0;
  ctrl_delta_x_theta_state.error_delta_x_pp = 0;
  ctrl_delta_x_theta_state.u_v_rfl_p = 0;
  ctrl_delta_x_theta_state.u_v_rf2_p = 0;
  ctrl_delta_x_theta_state.u_out_p = 0;
  ctrl_delta_x_theta_state.error_theta_p = 0;
  ctrl_delta_x_theta_state.error_theta_sum_p = 0;
  ctrl_delta_x_theta_state.u_omega_rfl_p = 0;
  ctrl_delta_x_theta_state.u_omega_rf2_p = 0;
}

double wl_dsr;
double wr_dsr;

inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);

// ctrl_saturation_wl_wr(&wl_dsr, &wr_dsr);

// send to inner loop
if(wl_dsr < 0){ wl_dsr = 0;}
if(wr_dsr < 0){ wr_dsr = 0;}

// final safety to prevent backward rotation in ETT

// LEE abnormal case bypass
if(1 == not_start_bypass || stop_bypass > 0 || brake_bypass > 0){
  // send 0 reference command
  if(1 == not_start_bypass){
    h2l_send_wl_wr_msg(0, 0);
  }
}
if (1 == stop_bypass) {
    printf("STOP! send wl_dsr -1 wr_dsr -1\n");
    h2l_send_wr_msg(-1, -1);
    // printf("STOP! send wl_dsr -1 wr_dsr -1\n");
} } else {  // normal operation
    h2l_send_wr_msg((float)wl_dsr, (float)wr_dsr);
} }

// save current intermediate state, for debug
ctrl_delta_x_theta_state.wl_dsr = wl_dsr;
ctrl_delta_x_theta_state.wr_dsr = wr_dsr;
ctrl_delta_x_theta_state.u_v = u_v;
ctrl_delta_x_theta_state.u_omega = u_omega;
ctrl_delta_x_theta_state.error_delta_x = error_delta_x;
ctrl_delta_x_theta_state.error_theta = error_theta;
ctrl_delta_x_theta_state.up_v_out = up_v_out;
ctrl_delta_x_theta_state.ui_v_out = ui_v_out;
ctrl_delta_x_theta_state.ud_v_out = ud_v_out;
ctrl_delta_x_theta_state.u_v_out = u_v_out;
ctrl_delta_x_theta_state.u_omega_out = u_omega_out;
ctrl_delta_x_theta_state.up_vrf1_out = up_vrf1_out;
ctrl_delta_x_theta_state.ui_vrf1_out = ui_vrf1_out;
ctrl_delta_x_theta_state.ud_vrf1_out = ud_vrf1_out;
ctrl_delta_x_theta_state.u_vrf1_out = u_vrf1_out;
ctrl_delta_x_theta_state.up_vrf2_out = up_vrf2_out;
ctrl_delta_x_theta_state.ui_vrf2_out = ui_vrf2_out;
ctrl_delta_x_theta_state.ud_vrf2_out = ud_vrf2_out;
ctrl_delta_x_theta_state.u_vrf2_out = u_vrf2_out;
ctrl_delta_x_theta_state.error_delta_x_p = error_delta_x;
ctrl_delta_x_theta_state.error_delta_x_pp = error_delta_x;
ctrl_delta_x_theta_state.error_theta_p = error_theta;
ctrl_delta_x_theta_state.error_theta_sum_p = error_theta_sum;
ctrl_delta_x_theta_state.u_omega_rfl_p = u_omega_rfl;
ctrl_delta_x_theta_state.u_omega_rfl2_p = u_omega_rfl2;
ctrl_delta_x_theta_state.u_omega_rfl3_p = u_omega_rfl3;

// other state
delta_x_neg_error_flag_p = delta_x_neg_error_flag;
vs_state.ultrasonic_delta_x_bk = vs_state.ultrasonic_delta_x;
```c
void ctrl_line_track_init() {
    char fn_buff[64];
    char suffix[128];
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    snprintf(suffix, sizeof(suffix),
                "%04d_%02d_%02d-%02d_%02d_%02d",
                tm.tm_year + 1900,
                tm.tm_mday, tm.tm_mon+1, tm.tm_hour, tm.tm_min,
                tm.tm_sec);
    snprintf(fn_buff, sizeof(fn_buff),
                "line_track_%s.txt", suffix);
    ctrl_line_track_state.log_file
        = fopen(fn_buff, "w+");
    ctrl_line_track_state.recent_log_file
        = fopen("line_track.txt", "w+");}

void ctrl_line_track_exit() {
    fclose(ctrl_line_track_state.log_file);
    fclose(ctrl_line_track_state.recent_log_file);
}

void ctrl_line_track_print(FILE *fd) {
    struct timespec now;
    clock_gettime(CLOCK_MONOTONIC, &now);
    uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
    // print vehicle state first
    fprintf(fd, "%08u, %08u,
            "% 6.2f, % 6.2f, % 6.2f, % 6.2f, % 6.2f, %t",
            ts, vs.state.timestamp,
            vs.state.encoder wl, vs.state.encoder wr,
            vs.state.v, vs.state.x, vs.state.v_dsr
        );
    // print controller states
    fprintf(fd, "% 6.2f, % 6.2f, % 6.2f, % 6.2f, %t",
            ctrl_line_track_state.wl_dsr,
            ctrl_line_track_state.wr_dsr,
            ctrl_line_track_state.u_v,
            ctrl_line_track_state.u_omega
        );
    // debug value
    fprintf(fd, "% 6.3f, % 6.3f, % 6.3f, %d \n",
            vs.state.camera_delta_theta,
            ctrl_line_track_state.u_omega_out,
            (0 - vs.state.imu_theta),
            vs.state.ctrl_flag);
    fflush(fd);
}

void ctrl_line_track_loop() {
```
// calculate error

double delta_theta = vstate.camera_delta_theta;
double delta_theta_sum = ctrl_line_track_state.delta_theta_sum;
    + delta_theta;
double delta_theta_p = ctrl_line_track_state.delta_theta_p;

double up_v_out;
double ui_v_out;
double ud_v_out;

// PID controller main body

// NOTE: v is not controlled here.
double u_v = ctrl_line_track_state.v_dsr;
double up_omega_out;
double ui_omega_out;
double ud_omega_out;

double u_omega_out = ctrl_pid(
delta_theta, delta_theta_sum, delta_theta_p,
    ctrl_line_track.para.kp, ctrl_line_track.para.ki,
    ctrl_line_track.para.kd,
    &up_omega_out, &ui_omega_out,
    &ud_omega_out, mc_init.ctrl_loop_period / 1000.0);

// NOTE: If you has d-term then roll off, (low pass filter)
double u_omega_rfl = ctrl_roll_off_once
    (ctrl_line_track_state.u_omega_rfl_p,
    u_omega_out, rf_coeffs);
double u_omega_rf2 = ctrl_roll_off_once
    (ctrl_line_track_state.u_omega_rf2_p,
    u_omega_rfl, rf_coeffs);

// outer loop control effort restriction
// not restriction for u_v
double u_omega = ctrl_threshold_and_saturation
    (u_omega_rf2, 0, -2, 2);

double wl_dsr;
double wr_dsr;
inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);

// send to inner loop:
h2l_send_wl_wr_msg((float) wl_dsr, (float) wr_dsr);

// update current states
ctrl_line_track_state.delta_theta = delta_theta;
ctrl_line_track_state.wl_dsr = wl_dsr;
ctrl_line_track_state.wr_dsr = wr_dsr;
ctrl_line_track_state.u_v = u_v;
ctrl_line_track_state.u_omega = u_omega;

// update debug states
ctrl_line_track_state.up_omega_out = up_omega_out;
ctrl_line_track_state.ui_omega_out = ui_omega_out;
ctrl_line_track_state.ud_omega_out = ud_omega_out;
ctrl_line_track_state.u_omega_out = u_omega_out;

// print control state to file
ctrl_line_track_print
    (ctrl_line_track_state.log_file);
ctrl_line_track_print
    (ctrl_line_track_state.recent_log_file);
1306  // update previous states
1307  ctrl_line_track_state.delta_theta_p = delta_theta;
1308  ctrl_line_track_state.delta_theta_sum_p = delta_theta_sum;
1309  ctrl_line_track_state.u_omega_rfl_p = u_omega_rfl;
1310  ctrl_line_track_state.u_omega_rf2_p = u_omega_rf2;
1311  }
1312  
1313  // -------------- platooning control --------------
1314  void ctrl_platooning_init() {
1315      char fn_buff[64];
1316      char suffix[128];
1317      time_t t = time(NULL);
1318      struct tm tm = *localtime(&t);
1319      snprintf(suffix, sizeof(suffix),
1320          "%04d_%02d_%02d_%02d_%02d",
1321          tm.tm_year + 1900,
1322          tm.tm_mon + 1,
1323          tm.tm_mday,
1324          tm.tm_hour,
1325          tm.tm_min,
1326          tm.tm_sec);
1327      snprintf(fn_buff, sizeof(fn_buff),
1328          "platooning_%s.txt", suffix);
1329      ctrl_platooning_state.log_file = fopen(fn_buff, "w");
1330      ctrl_platooning_state.recent_log_file = fopen("platooning.txt", "w");
1331  }
1332
1333  void ctrl_platooning_exit() {
1334      fclose(ctrl_platooning_state.log_file);
1335      fclose(ctrl_platooning_state.recent_log_file);
1336  }
1337
1338  void ctrl_platooning_print(FILE *fd) {
1339      struct timespec now;
1340      clock_gettime(CLOCK_MONOTONIC, &now);
1341      uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
1342      fprintf(fd, "%08u %08u  
1343          "%6.4f %6.2f %6.2f %6.4f %6.4f 
1344          "%6.4f %6.4f %6.4f %6.4f  
1345          ts , vstate.timestamp ,
1346          vstate.ultrasonic_delta_x ,
1347          vstate.encoder wl ,
1348          vstate.encoder.wr ,
1349          vstate.v ,
1350          vstate.omega ,
1351          vstate.x ,
1352          vstate.y ,
1353          vstate.theta );
1354
1355      // print control variable col: 11-16
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443

f p r i n t f ( fd , ” %6.2 f , %6.2 f , %6.4 f , %6.4 f , ”
” %6.4 f , %6.4 f , \ t \ t ” ,
c t r l p l a t o o n i n g s t a t e . wl dsr ,
c t r l p l a t o o n i n g s t a t e . wr dsr ,
ctrl platooning state . u v ,
c t r l p l a t o o n i n g s t a t e . u omega ,
ctrl platooning state . delta x dsr ,
ctrl platooning state . theta dsr );
// e r r o r c o l : 17−25
f p r i n t f ( fd , ” %6.4 f , %6.4 f , %6.4 f , ”
” %6.4 f , %6.4 f , %6.4 f , ”
” %6.4 f , %6.4 f , %6.4 f , \ t \ t ” ,
ctrl platooning state . error theta ,
ctrl platooning state . error theta p ,
c t r l p l a t o o n i n g s t a t e . error theta sum p ,
ctrl platooning state . error delta x ,
ctrl platooning state . error delta x p ,
ctrl platooning state . error delta x pp ,
ctrl platooning state . error leader ,
ctrl platooning state . error leader p ,
ctrl platooning state . error leader sum p );
// c o l : 2 6 −40
f p r i n t f ( fd , ” %6.4 f , %6.4 f , %6.4 f , %6.4 f , %6.4 f , ”
” %6.4 f , \ t \ t ”
” %6.4 f , %6.4 f , %6.4 f , %6.4 f , %6.4 f , \ t ”
” %6.4 f , %6.4 f , %6.4 f , %6.4 f , %6.4 f , \ t ”
” %6.4 f , %6.4 f , %6.4 f , %6.4 f , %6.4 f , \ t ”
” %6.4 f , %6.4 f , %6.4 f , %6.4 f , \ t \ t ” ,
ctrl
ctrl
ctrl
ctrl
ctrl
ctrl

platooning
platooning
platooning
platooning
platooning
platooning

state
state
state
state
state
state

.v
.u
.u
.u
.u
.u

ctrl
ctrl
ctrl
ctrl
ctrl

platooning
platooning
platooning
platooning
platooning

state
state
state
state
state

. up v x out ,
. ui v x out ,
. ud v x out ,
. u v x out p ,
. u v x out ,

ctrl
ctrl
ctrl
ctrl
ctrl

platooning
platooning
platooning
platooning
platooning

state
state
state
state
state

.
.
.
.
.

up v ffleader out ,
ui v ffleader out ,
ud v ffleader out ,
u v ffleader out p ,
u v ffleader out ,

ctrl
ctrl
ctrl
ctrl
ctrl

platooning
platooning
platooning
platooning
platooning

state
state
state
state
state

.
.
.
.
.

up v ffpre out ,
ui v ffpre out ,
ud v ffpre out ,
u v ffpre out p ,
u v ffpre out ,

ctrl
ctrl
ctrl
ctrl

platooning
platooning
platooning
platooning

state
state
state
state

. up omega out ,
. ui omega out ,
. ud omega out ,
. u omega wo rf ) ;

leader ,
v x,
v ffleader ,
v ffleader acc ,
v ffpre ,
v ffpre acc ,

f p r i n t f ( fd , ”%d , %d , %d , %d , \ t \ t ” ,
started flag ,
should stop flag ,
not start bypass ,
stop bypass
);
f p r i n t f ( fd , ”% 6 . 2 f , % 6 . 2 f , % 6 . 2 f , \ t \ t ” ,

228


void ctrl_platooning_loop() {

    static int leader_v_dsr_print_flag = 0;
    // information gathering from sensor or communication
    // sensor
    double delta_x = vstate.ultrasonic_delta_x;
    double delta_x_filtered = 0.4;

    // // get leader info from communicaiton
    struct v2v_msg *leader_msg = get_recent_msg(ctrl_platooning_para.leader_id);
    // get msg from immediately predecessor not enable yet
    if (leader_msg == NULL) {
        printf("BUG! ! ! leader_msg = NULL, leader_id = %d\n",
               ctrl_platooning_para.leader_id);
        return;
    }
    else
    {
        ctrl_platooning_state.ctrl_flag_leader = leader_msg->ctrl_flag;
        if (fabs(leader_msg->v) > 0.001
            && 0 == started_flag){
            printf("Leader_v nonzero received ",
                   " % 6.2f!\n", leader_msg->v);
            if (fabs(leader_msg->v) < 1){
                started_flag = 1;
                not_start_bypass = 0;
                vehicle_status = 1;
                running_time_start_ms = vstate.timestamp;
                printf("Vehicle Start at %ld ms!\n",
                       "% 6.2f !\n", delta_x);
            }
            else{
                vstate.ultrasonic_delta_x bk = delta_x;
                printf("Ultrasonic bk Initialized! ",
                       "delta_x at % 6.2f !\n", delta_x);
                printf("Ultrasonic bk Initialized\n",
                       "at % 6.2f, ",
                       "replace bk with 0.4m \n",
                       vstate.ultrasonic_delta_x bk);
                vstate.ultrasonic_delta_x bk = 0.4;
            }
        }
    }
    if (delta_x > 0.01 && delta_x < 3){
        vstate.ultrasonic_delta_x bk = delta_x;
        printf("<Ultrasonic bk Initialized at" ",
               "% 6.2f !\n", delta_x);
    } else{
        vstate.ultrasonic_delta_x bk = delta_x;
        printf("Ultrasonic bk Initialized Error! ",
               "delta_x at % 6.2f !\n", delta_x);
        printf("Ultrasonic bk Initialized\n",
               "at % 6.2f, ",
               "replace bk with 0.4m \n",
               vstate.ultrasonic_delta_x bk);
        vstate.ultrasonic_delta_x bk = 0.4;
    }
    vstate.x = 0;
    vstate.y = 0;
    }
}
printf("Communication Error ?" ": v.recv: % 6.2f\n", leader_msg->v);
"

// v-dsr information getting
if (fabs(leader_msg->v_dsr - 0.001) 
& 0 == started_flag) {
    printf("Leader #v_dsr# nonzero received" ": % 6.2f\n", leader_msg->v_dsr);
    if ( fabs(leader_msg->v_dsr - 0.3) < 0.01 ) {
        printf("Vehicle Start at %ld ms !\n", running_time_start_ms);
        started_flag = 1;
        not_start_bypass = 0;
        vehicle_status = 1;

        if (delta_x > 0.01 && delta_x < 3) {
            vstate.ultrasonic_delta_x_bk = delta_x;
            printf("<Ultrasonic bk Initialized at" ": % 6.2f !>\n", delta_x);
        }
    } else {
        vstate.ultrasonic_delta_x_bk = delta_x;
        printf("Ultrasonic bk Initialized Error !" "delta_x at % 6.2f !\n", delta_x);
        printf("Ultrasonic bk Initialized" "at % 6.2f."
        " replace bk with 0.4m \n",
        vstate.ultrasonic_delta_x_bk);
        vstate.ultrasonic_delta_x_bk = 0.4;
    }
}
else {
    printf("Communication Error ?" ": v_dsr recv: % 6.2f\n", 
    leader_msg->v_dsr);
}
}
}
ctrl_platooning.state.v_dsr_leader = leader_msg->v_dsr;
}
}
if (1 == started_flag ) {
    if (delta_x >= 3 || delta_x < 0.03) {
        printf("ULTRASONIC ERROR OUT OF RANGE!" ": % 6.2f\n", delta_x);
    }
    if (vstate.ultrasonic_delta_x_bk < 0.05 || 
    vstate.ultrasonic_delta_x_bk > 2 ) {
        printf("ULTRASONIC BK OUT OF RANGE!" ": % 6.2f\n", delta_x);
        printf("change current and bk to 0.4\n");
        delta_x = 0.4;
        vstate.ultrasonic_delta_x_bk = 0.4;
    } else {
        delta_x = vstate.ultrasonic_delta_x_bk;
    }
}
else {
    // sensor reading is in the range
    if (fabs(delta_x - vstate.ultrasonic_delta_x_bk) 
    >= 0.1) {

230
if(fabs(delta_x - vstate.ultrasonic_delta_x_bk) 
  >= 0.3){
    printf("ULTRASONIC JUMP HUGE\n");
    delta_x = vstate.ultrasonic_delta_x_bk;
} else{
    if(fabs(delta_x) >= 0.30){
        printf("CAMERA ERROR\n");
        // use 0/ imu data as backup
        delta_x = 0;
        vstate.ultrasonic_delta_x_bk = delta_x_fitered;
    } else{
        delta_x_fitered = delta_x;
        vstate.ultrasonic_delta_x_bk = delta_x_fitered;
    }
}

// getting delta_theta by camera
double delta_theta = vstate.camera_delta_theta;
if(fabs(delta_theta) >= 0.30){
    printf("CAMERA ERROR\n");
    // use 0/ imu data as backup
    delta_theta = 0;
    vstate.camera_delta_theta_p = delta_theta;
}

double x_leader;
double v_leader = leader_msg->v;

if(v_leader >0.5 || v_leader < 0){
    printf("LEADER INFO BACKUP ERROR v_leader" 
        " : % 6.2f ues bk: %6.2f !\n", 
        v_leader, 
        ctrl_platooning_state.v_leader_p);
    v_leader = ctrl_platooning_state.v_leader_p;
}

v_leader = 0.5*(v_leader + ctrl_platooning_state.v_leader_p);

x_leader = ctrl_platooning_state.x_leader_p + v_leader * 0.1;

double acc_leader = leader_msg->accx;
if (vstate.v < 0 || vstate.v > 1){
    printf("vstate.v at % 6.2f\n", vstate.v);
    printf("encoder info wl: % 6.2f wr: % 6.2f\n", vstate.encoder wl, vstate.encoder wl);
    vstate.v = 0.3;
}

// current error computing
double error_delta_x = delta_x_filtered - ctrl_platooning_state.delta_x_dsr;
double error_theta = error_delta_x;
double error_leader = v_leader - vstate.v;
double error_leader_acc = acc_leader - vstate.accx;
double u_v = 0;
double u_omega = 0;

// status checking
// vehicle status checking

// stop status checking
// vehicle will stop if
// too close to predecessor
// off-track.

if (v_leader < 0.25 && 1 == started_flag){
    // leader stop check
    // printf("Leader speed less than 0.25 and started\n");
    if (run_lapsed_time_ms > 3000){
        leader_stopped_flag++;
        if (1 == leader_stopped_flag){
            printf("LEADER STOPPED after this vehicle\n");
            printf("run % 6.2f sec \n", (double)(run_lapsed_time_ms)/1000.0);
            // should_stop_flag = 1;
        }
    } else{
    }
}

// if camera is too close to object ahead
// replace theta info from imu
static int camera_too_close_flag = 0;

// double lowest_cruise_speed = 0.05;
if (delta_x <= 0.30 && 1 == started_flag){
    camera_too_close_flag++;
    if (camera_too_close_flag == 1){
        printf("camera view too close brake bypass 1\n");
        if (fabs(error_theta) < 0.30){
            printf("CAMERA Error!\n");
            delta_theta = 0;
        }
        if (brake_bypass <= 0)
            brake_bypass = 1;
    } else{
    }
} else{
}
static int off_track_stop_print_flag = 0;
if(fabs(error_theta) >= 0.25){
    // off-track
    off_track_stop_print_flag++;
    if(5 == off_track_stop_print_flag ){
        printf("OFF TRACK AND STOP offtrack_cnt %d \n", off_track_stop_print_flag);
    }
}

static int emergency_stop_print_flag = 0;
if(delta_x <= 0.10 && started_flag == 1 && delta_x > 0){
    // too close set stop flag
    // >= 0 to prevent error reading at beginning
    emergency_stop_print_flag++;
    if (brake_bypass<= 0)
        brake_bypass = 10;
    if(1 == emergency_stop_print_flag){
        printf("TOO CLOSE at %5.2f AND STOP!\n", delta_x);
    }
}

if(emergency_stop_print_flag > 0 || off_track_stop_print_flag >4){
    // started_flag = 0;
    // be caution, when stop flag is on
    // don't clear started_flag immediately
    should_stop_flag = 1;
    not_start_bypass = 0;
    stop_bypass++;;
    vehicle_status--;;
    if(-1 == vehicle_status){
        printf("VEHICLE SHOULD STOP!\n");
        printf("STOP FLAGS STATUS: emgcy:"
"%d off_track: %d \n"
emergency_stop_print_flag ,
off_track_stop_print_flag);
    }
}
if (running_time_start_ms > 0 ){
    run_lapsed_time_ms
    = vstate.timestamp - running_time_start_ms;
}

// error update
// CAUTION: Here we use (delta_x - delta_x_dsr).
// This is intentional.
double error_delta_x_p
    = ctrl_platooning_state.error_delta_x_p;
double error_delta_x_pp
    = ctrl_platooning_state.error_delta_x_pp;

double error_leader_p
    = ctrl_platooning_state.error_leader_p;
double error_leader_pp
    = ctrl_platooning_state.error_leader_pp;
double error_leader_sum = ctrl_platooning_state.error_leader_sum + error_leader;

double error_theta_sum = ctrl_platooning_state.error_theta_sum + error_theta;

double error_theta_p = ctrl_platooning_state.error_theta_p;

if (vehicle_status <= 0) {
    error_delta_x = 0;
    error_leader_acc = 0;
    error_theta = 0;
    error_delta_x_p = 0;
    error_delta_x_pp = 0;
    error_leader_sum = 0;
    error_theta_sum = 0;
    error_theta_p = 0;
}

// Derive velocity component from delta_x info.
// CAUTION: This is PID incremental style
double up_v.x_out = 0;
double ui_v.x_out = 0;
double ud_v.x_out = 0;
double u_v.x_out = 0;
double u.v.x.rf1 = 0;
double u.v.x.rf2 = 0;
double x.v.kd_local = 0;

// Derive velocity component from leader info.
// CAUTION: This is PID incremental style!!!
double up_v.ffleader_out = 0;
double ui_v.ffleader_out = 0;
double ud_v.ffleader_out = 0;
double ff_v.kp_local = ctrl_platooning_param.kp_ffleader;
double ff_v.ki_local = ctrl_platooning_param.ki_ffleader;
double ff_v.kd_local = ctrl_platooning_param.kd_ffleader;

if (vstate.v > v_leader && vstate.v > 0.3) {
    follower_catched_up_flag++;
    if (1 == follower_catched_up_flag) {
        printf("<CATCH UP!> v_leader = %.2f v: %.2f\n", v_leader, vstate.v);
        printf("ff_v PID now is:");
        printf("%6.2f, %6.2f, %6.2f \n",
            ff_v.kp_local, ff_v.ki_local, ff_v.kd_local);
    }
}

static int cancel_kd_flag = 0;
if (run_lapsed_time_ms > 3000) {
    cancel_kd_flag++;
    if (1 == cancel_kd_flag) {
        printf("cancel kd term \n");
    }
} else if (1 == follower_catched_up_flag) {
x.v.kd.local = 0;
if (1 == started.flag)
    cancel.kd.flag++;
else if (1 == cancel.kd.flag)
    }
}
x.v.kd.local = ctrl.delta.x.theta_para.kd.delta.x;
}

u.v.x.out = ctrl.pid.inc(
    ctrl.platooning.state.u.v.x.out_p,
    error.delta.x, error.delta.x_p,
    error.delta.x_pp,
    ctrl.platooning.param.kp.delta.x,
    ctrl.platooning.param.ki.delta.x,
    x.v.kd.local, &up.v.x.out, &ui.v.x.out,
    &ud.v.x.out,
    mc_init.ctrl_loop.period / 1000.0);

// controller roll off on the output v (low pass filter)
u.v.x.rf1 = ctrl.roll_off_once(
    ctrl.platooning.state.u.v.x.rf1_p,
    u.v.x.out, 2
    rf.coff);  
u.v.x.rf2 = ctrl.roll_off_once(
    ctrl.platooning.state.u.v.x.rf2_p,
    u.v.x.rf1,
    rf.coff);

// u.v.x.rf2 = u.v.x.rf1;
if (running.time.start_ms < 1000 || started.flag == 0){
    if (u.v.x.rf2 < 0){
        ctrl.platooning.state.u.v.x.rf1_p = 0;
        ctrl.platooning.state.u.v.x.rf2_p = 0;
        u.v.x.rf1 = 0;
        u.v.x.rf2 = 0;
    }
}

if (follower.cahted.up.flag > 1){
    ff.v.kp.local = 0;
}

double u.v.ffleader.out = ctrl.pid(error_leader,
    error_leader_sum,
    error_leader_p,
    ff.v.kp_local,
    ff.v.ki_local,
    ff.v.kd_local,
    &up.v.ffleader_out, &ui.v.ffleader_out,
    &ud.v.ffleader_out,
    mc_init.ctrl_loop.period / 1000.0);

double u.v.ffleader.rf = ctrl.roll_off_once(
    ctrl.platooning.state.u.v.ffleader.rf_p,
    u.v.ffleader_out,
    rf.coff);

// u.v.ffleader.rf = u.v.ffleader.out;
// bypass roll if only P control
u.v.ffleader.rf = ctrl.platooning.state.u.v.ffleader.rf_before_sat
    = u.v.ffleader.rf;

u.v.ffleader.rf = ctrl.threshold_and_saturation
    (u.v.ffleader.rf, 0, 0, 0.3);

// if (u.v.ffleader.rf)
```c

1917  double u_v_ffleader_acc = ctrl_platooning_para.ka_ffleader
1918      * error_leader_acc;
1919
1920  // merge several control efforts
1921
1922  double u_v_total = u_v_x_rf2 + u_v_ffleader_rf + u_v_ffleader_acc;
1923  double u_v_rf = ctrl_roll_off_once(ctrl_platooning_state.u_rf_p,
1924                                        u_v_total, 0.8);
1925
1926  double up_omega_out;
1927  double ui_omega_out;
1928  double ud_omega_out;
1929
1930  double u_omega_out = ctrl_pid(  
1931      error_theta,  
1932      error_theta_sum,  
1933      error_theta_p,  
1934      ctrl_platooning_para.kp_theta,  
1935      ctrl_platooning_para.ki_theta,  
1936      ctrl_platooning_para.kd_theta,  
1937      &up_omega_out,  
1938      &ui_omega_out,  
1939      &ud_omega_out,  
1940      mc_init.ctrl_loop_period / 1000.0 );
1941
1942  // NOTE: If you have d-term then roll off,
1943  //(low pass filter twice)
1944  double u_omega_rf1 = ctrl_roll_off_once(ctrl_platooning_state.u_omega_rf1_p,  
1945                                          u_omega_out, rf_coeff);
1946  double u_omega_rf2 = ctrl_roll_off_once(ctrl_platooning_state.u_omega_rf2_p,  
1947                                          u_omega_rf1, rf_coeff);
1948
1949  if(fabs(u_omega_out)>10 || fabs(u_v) >2){
1950      printf("Controller failure, uw: ",
1951             "% 6.2f, u_v: % 6.2f",u_omega_out,u_v);
1952      if(fabs(u_v) >2){
1953          u_v = ctrl_platooning_state.u_v_rf_p;
1954      }  
1955      if(fabs(u_omega_out)>10){  
1956          u_omega_out = 0;
1957      }
1958  }
1959  // control effort restriction
1960  u_v = ctrl_threshold_and_saturation(u_v_rf , 0, 0, 0.6);
1961  u_omega = ctrl_threshold_and_saturation(  
1962                                        u_omega_rf2 , 0, -2, 2);
1963  // no direction control at very low speed
1964  // if(u_v < 0.1){
1965  //   u_omega = 0;
1966  // }
1967  double wl_dsr;
1968  double wr_dsr;
1969  if(l== started_flag && 0 == leader_stopped_flag  
1970     && 0 == follower_ca_time_dsr){
```
```c
u_v = ctrl_threshold_and_saturation
    (u_v_rf, 0, 0, 0.6);

inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);

// final safety check to prevent backward rotation in ETT
if(wl_dsr < 0) { wl_dsr = 0; }
if(wr_dsr < 0) { wr_dsr = 0; }
static int stop_in_advance_flag = 0;
int h2l_status = 1;

if(leader_stopped_flag > 0 && delta_x_filtered <= 0.44 ){
    stop_in_advance_flag++;
    should_stop_flag = 1;
    if(!should_stop_flag){
        printf("STOP IN ADVANCE at delta_x \\
               : %6.2f and BRAKE at %ld\n",
               delta_x_filtered, run_lapsed_time_ms);
        brake_bypass = 10;
        h2l_status = -1;
    } else{
        h2l_status = 0;
    }
    u_v = 0;
    u_v_x_rf2 = 0;
    ctrl_platooning_state.u_v_x_out_p = 0;
    ctrl_platooning_state.u_v_x_rf1_p = 0;
    ctrl_platooning_state.u_v_x_rf2_p = 0;

} // h2l handling bypass and normal cases
if(1 == not_start_bypass){
    h2l_status = 0;
} else if(stop_bypass > 0){
    if(1 == stop_bypass)
        h2l_status = -1;
    else
        h2l_status = 0;
} else if(brake_bypass > 0){
    h2l_status = -1;
    printf("brake_bypass : %d\n", brake_bypass);
    brake_bypass--;  
} // sending lower level command according to h2l_status
if(1 == h2l_status ){
    // normal operation
    h2l_send_wl_wr_msg((float)wl_dsr, (float)wr_dsr);
} else if(0 == h2l_status){
    // turn off motor
    h2l_send_wl_wr_msg(0, 0);
    //printf("send 0 0\r\n");
} else{
    // brake action
    h2l_send_wl_wr_msg(-1, -1);
    printf("send -1 -1\r\n");
}
```
// save current intermediate state, for debug
ctrl_platooning_state.delta_x_filtered = delta_x_filtered;

ctrl_platooning_state.x_leader = x_leader;
ctrl_platooning_state.v_leader = v_leader;
ctrl_platooning_state.acc_leader = acc_leader;

ctrl_platooning_state.wl_dsr = wl_dsr;
ctrl_platooning_state.wr_dsr = wr_dsr;
ctrl_platooning_state.u_v = u_v;
ctrl_platooning_state.u_omega = u_omega;

ctrl_platooning_state.error_delta_x = error_delta_x;
ctrl_platooning_state.error_leader = error_leader;

ctrl_platooning_state.error_theta = error_theta;

ctrl_platooning_state.u_v_x = u_v_x_rf2;
ctrl_platooning_state.u_v_ffleader = u_v_ffleader_rf;
ctrl_platooning_state.u_v_ffleader_acc = u_v_ffleader_acc;

ctrl_platooning_state.up_v_x_out = up_v_x_out;
ctrl_platooning_state.ui_v_x_out = ui_v_x_out;
ctrl_platooning_state.ud_v_x_out = ud_v_x_out;
ctrl_platooning_state.u_v_x_out = u_v_x_out;

ctrl_platooning_state.up_v_ffleader_out = up_v_ffleader_out;
ctrl_platooning_state.ui_v_ffleader_out = ui_v_ffleader_out;
ctrl_platooning_state.ud_v_ffleader_out = ud_v_ffleader_out;
ctrl_platooning_state.u_v_ffleader_out = u_v_ffleader_out;

ctrl_platooning_state.up_omega_out = up_omega_out;
ctrl_platooning_state.ui_omega_out = ui_omega_out;
ctrl_platooning_state.ud_omega_out = ud_omega_out;
ctrl_platooning_state.u_omega_wo_rf = u_omega_out;

// print control state to file
ctrl_platooning_print(ctrl_platooning_state.log_file);
ctrl_platooning_print(
    ctrl_platooning_state.recent_log_file);

// update previous state
ctrl_platooning_state.u_v_rf_p = u_v_rf;

// from delta_x
ctrl_platooning_state.u_v_x_out_p = u_v_x_out;

ctrl_platooning_state.error_delta_x_p = error_delta_x;
ctrl_platooning_state.error_delta_x_pp = error_delta_x_p;

ctrl_platooning_state.u_v_x_rf1_p = u_v_x_rf1;
ctrl_platooning_state.u_v_x_rf2_p = u_v_x_rf2;

// from leader vehicle
ctrl_platooning_state.v_leader_p = v_leader;
ctrl_platooning_state.x_leader_p = x_leader;
ctrl_platooning_state.u_v_ffleader_out_p = u_v_ffleader_out;

ctrl_platooning_state.error_leader_p = error_leader;
ctrl_platooning_state.error_leader_sum_p = error_leader_sum;
ctrl_platooning_state.u_v_ffleader_rf.p = u_v_ffleader_rf;

// from theta
ctrl_platooning_state.error_theta.p = error_theta;
ctrl_platooning_state.error_theta_sum.p = error_theta_sum;
ctrl_platooning_state.u_omega_rf1.p = u_omega_rf1;
ctrl_platooning_state.u_omega_rf2.p = u_omega_rf2;
}

---

// outer loop thread

void *ctrl_loop_thread (void *para) {
    printf("<ctrl_loop> Control loop thread started.\n");
    ctrl_v_theta_init();
    ctrl_x_y.init();
    ctrl_delta_x_theta_init();
    ctrl_line_track_init();
    ctrl_platooning_init();
    while (!ctrl_loop_should_stop) {
        sem_wait(&ctrl_loop_sem);
        if (vstate.ctrl_state > 0) {
            ctrl_loop_current_rt -= mc.init.ctrl_loop_period;
            if (ctrl_loop_current_rt < 0) {
                vstate.ctrl_state = 0;
                h2l_send_stop_msg();
            }
            // printf("<ctrl_loop> iterative.\n");
            switch (vstate.ctrl_outer_loop_mode) {
            case CTRL_OPEN_LOOP: {
                ctrl_open_loop();
                break;
            }
            case CTRL_WL_WR: {
                ctrl_wl_wr_loop();
                break;
            }
            case CTRL_V_OMEGA: {
                ctrl_v_omega_loop();
                break;
            }
            case CTRL_V_THETA: {
                ctrl_v_theta_loop();
                break;
            }
            case CTRL_X_Y: {
                ctrl_x_y_loop();
                break;
            }
            case CTRL_DELTA_X_THETA: {
                ctrl_delta_x_theta_loop();
                break;
            }
            }
case CTRL_LINE_TRACK: {
    ctrl_line_track_loop();
    break;
}

case CTRL_PLATOONING: {
    ctrl_platooning_loop();
    break;
}
default:
    break;
}

if (vstate.ctrl_flag == CTRL_START) {
    h2l_send_start_msg();
    vstate.ctrl_flag = CTRL_NONE;
}

if (vstate.ctrl_flag == CTRL_STOP) {
    h2l_send_stop_msg();
    vstate.ctrl_flag = CTRL_NONE;
    vstate.ctrl_state = 0;
    printf("Stop executed!!!\n");
}

ctrl_v.theta_exit();
ctrl_x_y_exit();
ctrl_delta_x_theta_exit();
ctrl_line_track_exit();
ctrl_platooning_exit();
printf("<ctrl_loop> Control loop thread exiting...\n");
return NULL;

// ---------------- outer loop interface ----------------

// CAUTION: ctrl_setup() are called by command thread

// FIXIT: for setup, start, and stop, /
//we should put these h2l_send_xxx() /
// function call to ctrl loop thread

void ctrl_setup() {
    h2l_send_setup_msg();
}

void ctrl_start() {
    ctrl_loop_current_rt = mc_init.running_time;
    vstate.ctrl_state = 1;
    vstate.ctrl_flag = CTRL_START;
}

void ctrl_stop() {
    vstate.ctrl_flag = CTRL_STOP;
}
```c
void ctrl_open_loop_set_pwm(int16_t pwm_left_dsr, int16_t pwm_right_dsr) {
    if (vstate.ctrl_state == 0 || (vstate.ctrl_state == 1 && vstate.ctrl_outer_loop_mode == CTRL_OPEN_LOOP)) {
        ctrl_open_loop_state.pwm_left_dsr = pwm_left_dsr;
        ctrl_open_loop_state.pwm_right_dsr = pwm_right_dsr;
        vstate.ctrl_outer_loop_mode = CTRL_OPEN_LOOP;
    } else {
    }
}

void ctrl_set wl wr(double wl_dsr, double wr_dsr) {
    if (vstate.ctrl_state == 0 || (vstate.ctrl_state == 1 && vstate.ctrl_outer_loop_mode == CTRL_WL_WR)) {
        ctrl_wl_wr_state.wl_dsr = wl_dsr;
        ctrl_wl_wr_state.wr_dsr = wr_dsr;
        vstate.ctrl_outer_loop_mode = CTRL_WL_WR;
    } else {
    }
}

void ctrl_set_v_omega(double v_dsr, double omega_dsr) {
    if (vstate.ctrl_state == 0 || (vstate.ctrl_state == 1 && vstate.ctrl_outer_loop_mode == CTRL_V_OMEGA)) {
        ctrl_v_omega_state.v_dsr = v_dsr;
        ctrl_v_omega_state.omega_dsr = omega_dsr;
        vstate.ctrl_outer_loop_mode = CTRL_V_OMEGA;
    } else {
    }
}

void ctrl_set_v_theta(double v_dsr, double theta_dsr) {
    if (vstate.ctrl_state == 0 || (vstate.ctrl_state == 1 && vstate.ctrl_outer_loop_mode == CTRL_V_THETA)) {
        ctrl_v_theta_state.v_dsr = v_dsr;
        ctrl_v_theta_state.theta_dsr = theta_dsr;
        vstate.ctrl_outer_loop_mode = CTRL_V_THETA;
    } else {
    }
}

void ctrl_set_x_y(double x_dsr, double y_dsr) {
    if (vstate.ctrl_state == 0 || (vstate.ctrl_state == 1 && vstate.ctrl_outer_loop_mode == CTRL_X_Y)) {
    }
}
```
```c
    ctrl_x_y_state.x_dsr = x_dsr;
    ctrl_x_y_state.y_dsr = y_dsr;
    vstate.ctrl_outer_loop_mode = CTRL_X_Y;
    vstate.ctrl_inner_loop_mode = CTRL_WL_WR;
} else {
}

void ctrl_set_delta_x_theta(double delta_x_dsr, double theta_dsr) {
    if (vstate.ctrl_state == 0
        || (vstate.ctrl_state == 1
            && vstate.ctrl_outer_loop_mode
            == CTRL_DELTA_X_THETA)) {
        ctrl_delta_x_theta_state.delta_x_dsr = delta_x_dsr;
        ctrl_delta_x_theta_state.theta_dsr = theta_dsr;
        vstate.ctrl_outer_loop_mode = CTRL_DELTA_X_THETA;
        vstate.ctrl_inner_loop_mode = CTRL_WL_WR;
    } else {
    }
}

void ctrl_set_line_track(double v_dsr) {
    if (vstate.ctrl_state == 0
        || (vstate.ctrl_state == 1
            && vstate.ctrl_outer_loop_mode
            == CTRL_LINE_TRACK)) {
        ctrl_line_track_state.v_dsr = v_dsr;
        vstate.ctrl_outer_loop_mode = CTRL_LINE_TRACK;
        vstate.ctrl_inner_loop_mode = CTRL_WL_WR;
    } else {
    }
}

void ctrl_set_platooning(double delta_x_dsr, double theta_dsr) {
    if (vstate.ctrl_state == 0
        || (vstate.ctrl_state == 1
            && vstate.ctrl_outer_loop_mode
            == CTRL_PLATOONING)) {
        ctrl_platooning_state.delta_x_dsr = delta_x_dsr;
        ctrl_platooning_state.theta_dsr = theta_dsr;
        vstate.ctrl_outer_loop_mode = CTRL_PLATOONING;
        vstate.ctrl_inner_loop_mode = CTRL_WL_WR;
    } else {
    }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#include <unistd.h>
#include "h2l.h"
#include "serial.h"
#include "ctrl_loop.h"
#include "pi_mc.h"
```
volatile int serial_should_stop = 0;
int serial_fd = -1;

int h2l_send_open_loop_msg(int16_t pwm_left, int16_t pwm_right){
    int ret = 0;
    struct h2l_open_loop msg;
    h2l_set_header(&msg.header, sizeof(msg) - sizeof(msg.header), OPCODE_OPEN_LOOP);
    msg.v_left = pwm_left;
    msg.v_right = pwm_right;
    ret = serial_send_n_bytes(serial_fd, (char *)&msg, sizeof(msg));
    serial_flush(serial_fd);
    return ret;
}

int h2l_send_wl_wr_msg(float wl_dsr, float wr_dsr) {
    int ret = 0;
    struct h2l_wl_wr msg;
    h2l_set_header(&msg.header, sizeof(msg) - sizeof(msg.header), OPCODE_CTRL_WL_WR);
    msg.wl_dsr = wl_dsr;
    msg.wr_dsr = wr_dsr;
    ret = serial_send_n_bytes(serial_fd, (char *)&msg, sizeof(msg));
    serial_flush(serial_fd);
    return ret;
}

int h2l_send_setup_msg() {
    int ret = 0;
    struct h2l_setup msg;
    h2l_set_header(&msg.header, sizeof(msg) - sizeof(msg.header), OPCODE_SETUP);
    msg.prefilter_coefficient = ctrl_inner_para.prefilter_coefficient;
    msg.roll_off_coefficient = ctrl_inner_para.roll_off_coefficient;
    msg.kp_left = ctrl_inner_para.kp_left;
    msg.ki_left = ctrl_inner_para.ki_left;
    msg.kd_left = ctrl_inner_para.kd_left;
    msg.kp_right = ctrl_inner_para.kp_right;
    msg.ki_right = ctrl_inner_para.ki_right;
    msg.kd_right = ctrl_inner_para.kd_right;
    msg.deadzone_threshold = ctrl_inner_para.deadzone_threshold;
    msg.deadzone_saturation = ctrl_inner_para.deadzone_saturation;
    ret = serial_send_n_bytes(serial_fd, (char *)&msg, sizeof(msg));
}
```c
82     serial_flush(serial_fd);
83     // printf("setup: %d, %d\n", ret, sizeof(msg));
84     return ret;
85 }
86
87 static int h2l_send_header_only_msg(uint8_t opcode) {
88     int ret = 0;
89     struct h2l_header header;
90     h2l_set_header(&header, 0, opcode);
91     ret = serial_send_n_bytes(serial_fd, (char *)&header, sizeof(header));
92     serial_flush(serial_fd);
93     return ret;
94 }
95
96 int h2l_send_start_msg() {
97     return h2l_send_header_only_msg(OPCODE_START);
98 }
99
100 int h2l_send_stop_msg() {
101     return h2l_send_header_only_msg(OPCODE_STOP);
102 }
103
104 int h2l_send_debug_enable_msg() {
105     return h2l_send_header_only_msg(OPCODE_DEBUG_ENABLE);
106 }
107
108 int h2l_send_debug_disable_msg() {
109     return h2l_send_header_only_msg(OPCODE_DEBUG_DISABLE);
110 }
111
112 void h2l_update_vehicle_state(struct h2l_vehicle_state *msg) {
113     update_vehicle_state(msg);
114 }
115
116 void h2l_print_vehicle_status(FILE *fd) {
117     if (fd == NULL) {
118         return;
119     }
120     //fd = stdout;
121     fprintf(fd, "%08u, %5.4f, %5.4f, %5.4f, %5.2f, %5.2f, %5.3f, %t\t",
122             vstate.timestamp, vstate.imu_theta, vstate.imu_accx,
123             vstate.imu_omega, vstate.encoder wl, vstate.encoder wr,
124             vstate.ultrasonic_delta_x);
125 }
126
127 void h2l_print_ctrl_status_debug(FILE *fd, struct h2l_ctrl_status_debug *msg) {
128 ```
```c
int i;
if (fd == NULL) {
    return;
    //fd = stdout;
}

//update vstate_v_dsr;
vstate_v_dsr = (msg->wl_dsr + msg->wr_dsr)
    * mc_init_wheel_radius / 2;

h2l_print_vehicle_status(fd);

    printf(fd, "%5.2f, %5.2f, %5.2f, %5.2f, \t\t",
            msg->wl_dsr, msg->wr_dsr,
            msg->wl_dsr_filtered, msg->wr_dsr_filtered);

    printf(fd, "%4d, %4d, %4d, %4d, %4d, %4d, \t\t",
            msg->pwml, msg->pwmr, msg->pwml_out,
            msg->pwmr_out, msg->pwml_out_p, msg->pwmr_out_p);

    printf(fd, "%5.4f, %5.4f, %5.4f, %5.4f, ",
            msg->err_w1, msg->err_w1,
            msg->err_w1p, msg->err_w1pp,
            msg->err_w1_pp, msg->err_w1_pp);

for (i = 0; i < sizeof(msg->timestamps)
    / sizeof(msg->timestamps[0]); i++) {
    printf(fd, "%8u, ", msg->timestamps[i]);
}

    printf(fd, "\n");
    fflush(fd);

// ----------------- receive and dispatch -----------------

static int h2l_recv_process(char serial_buff[]) {
    int ret;
    int i;

    int opcode = serial_buff[3];
    int len = serial_buff[2];

    // receive payload
    ret = serial_recv_n_bytes(serial_fd, serial_buff
        + MCPROTO_HEADER_SIZE,

245
// for (i = 0; i < len; i++) {
// printf("%02X ", serial_buff[i]);
// }
// printf("\n");

switch (opcode) {
    case OPCODE_VEHICLE_STATUS: {
        struct h2l_vehicle_status *msg = (struct h2l_vehicle_status *) (serial_buff);
        update_vehicle_state(msg);
        break;
    }
    case OPCODE_CTRL_STATUS_DEBUG: {
        struct h2l_ctrl_status_debug *msg =
            (struct h2l_ctrl_status_debug *) (serial_buff);
        h2l_print_ctrl_status_debug(log_file, msg);
        h2l_print_ctrl_status_debug(recent_log_file, msg);
        // h2l_print_ctrl_status_debug(stdout, msg);
        break;
    }
    default: break;
}
return 0;
}

void *h2l_receive_thread(void *para) {
    int ret = 0;
    int i = 0;
    char c;
    int comm_state = SERIAL_STATE_INIT;
    char *arduino_serial_dev = mc_init.tty_file_name;
    // Open serial port
    serial_fd = serial_init(arduino_serial_dev,
                            mc_init.tty_baud_rate);
    if (serial_fd < 0) {
        printf("Unable to open serial port!!!\n");
        exit(-1);
    }
    printf("<mc_h2l_thread>
    "Serial port open successfully.\n");
    char serial_buff[128];
    while (!serial_shoudl_stop) {
        // CAUTION: This is polling every 1 ms!!!
        // receive protocol header
        ret = serial_recv_byte(serial_fd, &c);
        if (ret < 0) {
            printf("Serial read error!!!\n");
            break;
        } else if (ret == 0) {
            // handle data
        }
    }
}
```c
usleep(1000);
continue;
}

serial_buff[comm_state] = c;

// printf("%02X \n", c);

switch (comm_state) {  
  case SERIAL_STATE_INIT: {  
    if (c == SERIAL_MAGIC_1)  
      comm_state = SERIAL_STATE_MAGIC1;  
    else  
      comm_state = SERIAL_STATE_INIT;  
    break;
  }  
  case SERIAL_STATE_MAGIC1: {  
    if (c == SERIAL_MAGIC_2)  
      comm_state = SERIAL_STATE_MAGIC2;  
    else  
      comm_state = SERIAL_STATE_INIT;  
    break;
  }  
  case SERIAL_STATE_MAGIC2: {  
    comm_state = SERIAL_STATE_PROTO;  
    break;
  }  
  case SERIAL_STATE_PROTO: {  
    h2l_recv_process(serial_buff);  
    comm_state = SERIAL_STATE_INIT;  
    break;
  }  
  default: {  
    comm_state = SERIAL_STATE_INIT;  
    break;
  }  
}

printf("<mc_h2l_thread> Serial thread exiting...
");
serial_close(serial_fd);
return NULL;
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <time.h>
#include <signal.h>
#include <sys/types.h>
#include <pthread.h>
#include <semaphore.h>
#include "serial.h"
#include "h2l.h"
```
#include "v2v.h"
#include "ctrl_loop.h"
#include "camera.h"
#include "pi_mc.h"

struct mc_init mc_init;

FILE *meta_file = NULL;
FILE *recent_meta_file = NULL;
FILE *log_file = NULL;
FILE *recent_log_file = NULL;

// ---------------- config ----------------
void print_config(FILE *file) {
  fprintf(file, "vehicle_id=%d\n", mc_init.vehicle_id);
  fprintf(file, "tty_file_name=%s\n", mc_init.tty_file_name);
  fprintf(file, "log_file_name=%s\n", mc_init.log_file_name);
  fprintf(file, "meta_file_name=%s\n", mc_init.meta_file_name);
  fprintf(file, "manager=%s\n", mc_init.manager_ip, mc_init.manager_port);
  fprintf(file, "v2v_period=%d, ctrl_loop_period=%d\n", mc_init.v2v_period, mc_init.ctrl_loop_period);
  fprintf(file, "running_time=%d\n", mc_init.running_time);
  fprintf(file, "debug_mode=%d\n", mc_init.debug_mode);
  fprintf(file, "left PID controller: kp=%f, ki=%f, kd=%f\n",
          ctrl_innere_paras.kp_left, ctrl_innere_paras.ki_left, ctrl_innere_paras.kd_left);
  fprintf(file, "right PID controller: kp=%f, ki=%f, kd=%f\n",
          ctrl_innere_paras.kp_right, ctrl_innere_paras.ki_right, ctrl_innere_paras.kd_right);
  fprintf(file, "deadzone: threshold=%f, saturation=%f\n",
          ctrl_innere_paras.deadzone_threshold, ctrl_innere_paras.deadzone_saturation);
  fprintf(file, "pwm_left_init=%d, pwm_right_init=%d\n",
          ctrl_open_loop_state.pwm_left_dsr, ctrl_open_loop_state.pwm_right_dsr);
  fprintf(file, "wl_dsr_init=%f, wr_dsr_init=%f\n",
          ctrl_wl_wr_state.wl_dsr, ctrl_wl_wr_state.wr_dsr);
  fprintf(file, "v_dsr_init=%f, omega_dsr_init=%f\n",
          ctrl_vomega_state.v_dsr, ctrl_vomega_state.omega_dsr);
  fflush(file);
}

int set_config(char *key, char *value) {
  if (strcmp(key, "vehicle_id") == 0) {
    mc_init.vehicle_id = atoi(value);
  } else if (strcmp(key, "tty_file_name") == 0) {
    int len = strlen(value);
    value[len - 1] = 0;
    strncpy(mc_init.tty_file_name, value, sizeof(mc_init.tty_file_name));
  } else if (strcmp(key, "tty_baud_rate") == 0) {
    mc_init.tty_baud_rate = atoi(value);
  } else if (strcmp(key, "log_file_name") == 0) {
    int len = strlen(value);
    value[len - 1] = 0;
    strncpy(mc_init.log_file_name, value, sizeof(mc_init.log_file_name));
  }
  return 0;
}
} else if (strcmp(key, "meta_file_name") == 0) {
    int len = strlen(value);
    value[len - 1] = 0;
    strncpy(mc_init.meta_file_name, value, sizeof(mc_init.meta_file_name));
} else if (strcmp(key, "manager_ip") == 0) {
    int len = strlen(value);
    value[len - 1] = 0;
    strncpy(mc_init.manager_ip, value, sizeof(mc_init.manager_ip));
} else if (strcmp(key, "manager_port") == 0) {
    mc_init.manager_port = atoi(value);
} else if (strcmp(key, "v2v_period") == 0) {
    mc_init.v2v_period = atoi(value);
}
else if (strcmp(key, "camera_inverse") == 0) {
    cpara.inverse = atoi(value);
} else if (strcmp(key, "camera_threshold") == 0) {
    cpara.th = atoi(value);
} else if (strcmp(key, "camera_roi_x") == 0) {
    cpara.roi_x = atoi(value);
} else if (strcmp(key, "camera_roi_y") == 0) {
    cpara.roi_y = atoi(value);
} else if (strcmp(key, "camera_roi_width") == 0) {
    cpara.roi_width = atoi(value);
} else if (strcmp(key, "camera_roi_height") == 0) {
    cpara.roi.height = atoi(value);
}
else if (strcmp(key, "running_time") == 0) {
    mc_init.running_time = atoi(value);
} else if (strcmp(key, "ctrl_loop_period") == 0) {
    mc_init.ctrl_loop_period = atoi(value);
} else if (strcmp(key, "controller_mode") == 0) {
    vstate.ctrl_outer_loop_mode = atoi(value);
    vstate.ctrl_inner_loop_mode = CTRL_OPEN_LOOP;
} else
    vstate.ctrl_inner_loop_mode = CTRL_WL_WR;
} else if (strcmp(key, "debug_mode") == 0) {
    mc_init.debug_mode = atoi(value);
}
else if (strcmp(key, "wheel_radius") == 0) {
    mc_init.wheel_radius = atof(value);
} else if (strcmp(key, "distance_of_wheels") == 0) {
    mc_init.distance_of_wheels = atof(value);
}

// general controller parameter

// parameter of outer loop

// outer loop v_theta control
else if (strcmp(key, "outer_v_theta_kp_theta") == 0) {
    ctrl_v_theta_para.kp_theta = atof(value);
} else if (strcmp(key, "outer_v_theta_ki_theta") == 0) {
    ctrl_v_theta_para.ki_theta = atof(value);
} else if (strcmp(key, "outer_v_theta_kd_theta") == 0) {
    ctrl_v_theta_para.kd_theta = atof(value);
else if (strcmp(key, "outer_x_y_kp_dist") == 0) {  
    ctrl_x_y_para.kp_dist = atof(value);  
} else if (strcmp(key, "outer_x_y_kd_dist") == 0) {  
    ctrl_x_y_para.kd_dist = atof(value);  
} else if (strcmp(key, "outer_x_y_ki_dist") == 0) {  
    ctrl_x_y_para.ki_dist = atof(value);  
} else if (strcmp(key, "outer_x_y_kp_angle") == 0) {  
    ctrl_x_y_para kp_angle = atof(value);  
} else if (strcmp(key, "outer_x_y_kd_angle") == 0) {  
    ctrl_x_y_para.kd_angle = atof(value);  
} else if (strcmp(key, "outer_x_y_ki_angle") == 0) {  
    ctrl_x_y_para.ki_angle = atof(value);  
} else if (strcmp(key, "outer_x_y_lead_angle") == 0) {  
    ctrl_x_y_para.lead_angle = atof(value);  
}

else if (strcmp(key, "outer_delta_x_theta") == 0) {  
    ctrl_delta_x_theta para.kp_delta_x = atof(value);  
} else if (strcmp(key, "outer_delta_x_theta_ki_delta_x") == 0) {  
    ctrl_delta_x_theta para.ki_delta_x = atof(value);  
} else if (strcmp(key, "outer_delta_x_theta_kd_delta_x") == 0) {  
    ctrl_delta_x_theta para.kd_delta_x = atof(value);  
} else if (strcmp(key, "outer_delta_x_theta_kp_theta") == 0) {  
    ctrl_delta_x_theta para.kp_theta = atof(value);  
} else if (strcmp(key, "outer_delta_x_theta_ki_theta") == 0) {  
    ctrl_delta_x_theta para.ki_theta = atof(value);  
} else if (strcmp(key, "outer_delta_x_theta_kd_theta") == 0) {  
    ctrl_delta_x_theta para.kd_theta = atof(value);  
}

else if (strcmp(key, "outer_line_track") == 0) {  
    ctrl_line_track para.kp = atof(value);  
} else if (strcmp(key, "outer_line_track_ki") == 0) {  
    ctrl_line_track para.ki = atof(value);  
} else if (strcmp(key, "outer_line_track_kd") == 0) {  
    ctrl_line_track para.kd = atof(value);  
}

else if (strcmp(key, "outer_platooning_leader_id") == 0) {  
    ctrl_platooning para.leader_id = atoi(value);  
}  
else if (strcmp(key, "outer_platooning kp_delta_x") == 0) {  
    ctrl_platooning para.kp_delta_x = atof(value);  
} else if (strcmp(key, "outer_platooning_ki_delta_x") == 0) {  
    ctrl_platooning para.ki_delta_x = atof(value);  
} else if (strcmp(key, "outer_platooning_kd_delta_x") == 0) {  
    ctrl_platooning para.kd_delta_x = atof(value);  
} else if (strcmp(key, "outer_platooning_kp_ffleader") == 0) {  
    ctrl_platooning para.kp_ffleader = atof(value);  
} else if (strcmp(key, "outer_platooning_ki_ffleader") == 0) {  
    ctrl_platooning para.ki_ffleader = atof(value);  
} else if (strcmp(key, "outer_platooning_kd_ffleader") == 0) {  
    ctrl_platooning para.kd_ffleader = atof(value);  
} else if (strcmp(key, "outer_platooning_ka_ffleader") == 0) {  
    ctrl_platooning para.ka_ffleader = atof(value);  
}
else if (strcmp(key, "outer_platooning_kp_ffpre") == 0) {
    ctrl_platooning_para.kp_ffpre = atof(value);
} else if (strcmp(key, "outer_platooning_ki_ffpre") == 0) {
    ctrl_platooning_para.ki_ffpre = atof(value);
} else if (strcmp(key, "outer_platooning_kd_ffpre") == 0) {
    ctrl_platooning_para.kd_ffpre = atof(value);
} else if (strcmp(key, "outer_platooning_ka_ffpre") == 0) {
    ctrl_platooning_para.ka_ffpre = atof(value);
}

else if (strcmp(key, "outer_platooning_kp_theta") == 0) {
    ctrl_platooning_para.kp_theta = atof(value);
} else if (strcmp(key, "outer_platooning_ki_theta") == 0) {
    ctrl_platooning_para.ki_theta = atof(value);
} else if (strcmp(key, "outer_platooning_kd_theta") == 0) {
    ctrl_platooning_para.kd_theta = atof(value);
}

else if (strcmp(key, "prefilterCoefficient") == 0) {
    ctrl_inner_para.prefilter_coefficient = atof(value);
} else if (strcmp(key, "rollOffCoefficient") == 0) {
    ctrl_inner_para.roll_off_coefficient = atof(value);
} else if (strcmp(key, "inner_kp_left") == 0) {
    ctrl_inner_para.kp_left = atof(value);
} else if (strcmp(key, "inner_ki_left") == 0) {
    ctrl_inner_para.ki_left = atof(value);
} else if (strcmp(key, "inner_kd_left") == 0) {
    ctrl_inner_para.kd_left = atof(value);
} else if (strcmp(key, "inner_kp_right") == 0) {
    ctrl_inner_para.kp_right = atof(value);
} else if (strcmp(key, "inner_ki_right") == 0) {
    ctrl_inner_para.ki_right = atof(value);
} else if (strcmp(key, "inner_kd_right") == 0) {
    ctrl_inner_para.kd_right = atof(value);
} else if (strcmp(key, "deadzoneThreshold") == 0) {
    ctrl_inner_para.deadzone_threshold = atof(value);
} else if (strcmp(key, "deadzoneSaturation") == 0) {
    ctrl_inner_para.deadzone_saturation = atof(value);
}

else if (strcmp(key, "pwm_left_init") == 0) {
    ctrl_open_loop_state.pwm_left_dsr = atof(value);
} else if (strcmp(key, "pwm_right_init") == 0) {
    ctrl_open_loop_state.pwm_right_dsr = atof(value);
} else if (strcmp(key, "wl_dsr_init") == 0) {
    ctrl_wl_wr_state.wl_dsr = atof(value);
} else if (strcmp(key, "wr_dsr_init") == 0) {
    ctrl_wl_wr_state.wr_dsr = atof(value);
} else if (strcmp(key, "v_dsr_init") == 0) {
    ctrl_v_omega_state.v_dsr = atof(value);
} else if (strcmp(key, "omega_dsr_init") == 0) {
    ctrl_v_omega_state.omega_dsr = atof(value);
} else if (strcmp(key, "outer_v_theta_v_dsr_init") == 0) {
    ctrl_v_theta_state.v_dsr = atof(value);
} else if (strcmp(key, "outer_v_theta_theta_dsr_init") == 0) {
    ctrl_v_theta_state.theta_dsr = atof(value);
else if (strcmp(key, "outer_x_y_x dsr init") == 0) {
    ctrl_x_y_state.x_dsr = atof(value);
}
else if (strcmp(key, "outer_x_y_y dsr init") == 0) {
    ctrl_x_y_state.y_dsr = atof(value);
}
else if (strcmp(key, "outer_delta_x_theta_delta_x dsr init") == 0) {
    ctrl_delta_x_theta_state.delta_x_dsr = atof(value);
}
else if (strcmp(key, "outer_delta_x_theta_theta dsr init") == 0) {
    ctrl_delta_x_theta_state.theta_dsr = atof(value);
}
else if (strcmp(key, "outer_platooning_delta_x dsr init") == 0) {
    ctrl_platooning_state.delta_x_dsr = atof(value);
}
else if (strcmp(key, "outer_platooning_theta dsr init") == 0) {
    ctrl_platooning_state.theta_dsr = atof(value);
}
else if (strcmp(key, "outer_line_track_v dsr init") == 0) {
    ctrl_line_track_state.v_dsr = atof(value);
}

else {
    return -1;
}

return 0;

int parse_conf_file(char *conf_fn) {
    int ret = 0;
    int i;
    char *line = NULL;
    int line_count = 1;
    size_t n;
    FILE *fd;
    fd = fopen(conf_fn, "r");
    if (!fd) {
        printf("Unable to open configuration file: %s\n", conf_fn);
        return -1;
    }
    while ((ret = getline(&line, &n, fd)) != -1) {
        if (line[0] == '#' || line[0] == '
') {
            // The line is comment, do nothing.
        } else {
            char *ch = strchr(line, '=');
            if (!ch) {
                printf("Bad config file at line %d: %s\n", line_count, line);
                line_count = 1;
                return -1;
                break;
            }
            *ch = 0;
            char *key = line;
            char *value = ch + 1;
            // printf("key = %s, value = %s\n", key, value);
            if (set_config(key, value) < 0) {
345         printf("Bad parameter at line %d: %s, %s\n", 
346                line_count, key, 
347                value);
348         return -1;
349     break;
350 }
351     line_count++;
352 }
353 free(line);
354 fclose(fd);
355 return 0;
356 }
357
358 // ---------------------------------- command ----------------------------------
359 volatile int command_should_stop = 0;
360
361 int dispatch_command(char *cmd, char *argv) {
362     int ret = -1;
363     if (strcmp(cmd, "open_loop") == 0) {
364         int v_left;
365         int v_right;
366         sscanf(argv, "%d %d", &v_left, &v_right);
367         // printf("open_loop command: v_left= %d, v_right= %d\n", 
368         // v_left, v_right);
369         ctrl_open_loop_set_pwm((int16_t) v_left, 
370                                 (int16_t) v_right);
371     } else if (strcmp(cmd, "wl_wr") == 0) {
372         float wl_dsr;
373         float wr_dsr;
374         sscanf(argv, "%f %f", &wl_dsr, &wr_dsr);
375         // printf("close loop wl_wr command: wl_dsr= %f, wr_dsr= %f\n", 
376         // wl_dsr, wr_dsr);
377         ctrl_set_wl_wr(wl_dsr, wr_dsr);
378     } else if (strcmp(cmd, "v_omega") == 0) {
379         float v_dsr;
380         float omega_dsr;
381         sscanf(argv, "%f %f", &v_dsr, &omega_dsr);
382         // printf("close loop v_omega command: v_dsr= %f, 
383         // omega_dsr= %f\n", 
384         // v_dsr, omega_dsr);
385         ctrl_set_v_omega(v_dsr, omega_dsr);
386     } else if (strcmp(cmd, "v_theta") == 0) {
387         float v_dsr;
388         float theta_dsr;
389         sscanf(argv, "%f %f", &v_dsr, &theta_dsr);
390         ctrl_set_v_theta(v_dsr, theta_dsr);
391     } else if (strcmp(cmd, "x_y") == 0) {
392         float x_dsr;
393         float y_dsr;
394         sscanf(argv, "%f %f", &x_dsr, &y_dsr);
395         ctrl_set_x_y(x_dsr, y_dsr);
396     } else if (strcmp(cmd, "delta_x_theta") == 0) {
397         float delta_x_dsr;
398         float theta_dsr;
399         sscanf(argv, "%f %f", &delta_x_dsr, &theta_dsr);
400         ctrl_set_delta_x_theta(delta_x_dsr, theta_dsr);
401     }
402     return ret;
403 }
404
} else if (strcmp(cmd, "line_track") == 0) {
    float v_dsr;
    sscanf(argv, "%f", &v_dsr);
    ctrl_set_line_track(v_dsr);
}
} else if (strcmp(cmd, "platooning") == 0) {
    float delta_x_dsr;
    float theta_dsr;
    sscanf(argv, "%f %f", &delta_x_dsr, &theta_dsr);
    ctrl_set_platooning(delta_x_dsr, theta_dsr);
}
} else if (strcmp(cmd, "setup") == 0) {
    ctrl_setup();
    printf("setup command\n");
}
} else if (strcmp(cmd, "start") == 0) {
    ctrl_start();
    printf("start command\n");
}
} else if (strcmp(cmd, "stop") == 0) {
    ctrl_stop();
    printf("stop command\n");
}
} else if (strcmp(cmd, "exit") == 0) {
    command_should_stop = 1;
    serial_should_stop = 1;
    ctrl_loop_should_stop = 1;
    v2v_sending_should_stop = 1;
    v2v_receiving_should_stop = 1;
    camera_should_stop = 1;
}
} else {
    printf("Unknown command!!!\n");
}
}

return ret;

void *command_thread(void *para) {

    int ret;
    int i;
    char *line = (char *)malloc(1024);
    size_t n;
    printf("<command> Command thread started.\n");
    while (!command_should_stop) {
        // read a line from keyboard
        ret = getline(&line, &n, stdin);
        if (ret < 0) {
            printf("Unable to receive command! ret = %d(%s)\n", ret, strerror(errno));
            continue;
        }
        // remove the '\n'
        line[strlen(line) - 1] = 0;
        char *cmd = line;
        char *ch = strchr(line, ' ');

```c
if (ch != NULL) {
    *ch = 0;
    ch++;}
dispatch_command(cmd, ch);
}
free(line);
printf("<command> Command thread exiting...
return NULL;

// ---------------- timer ----------------
timer_t tid;
int timer_v2v_count = 0;
int timer_ctrl_loop_count = 0;

void handler(int signo) {
    // struct timespec now;
    // clock_gettime(CLOCK_MONOTONIC, &now);
    // printf("[%d] Diff time:%ld us\n", count++,
    // timer_diff(&now, &prev));
    // prev = now;
    timer_v2v_count++;
timer_ctrl_loop_count++;
    if (timer_v2v_count >= mc_init.v2v_period) {
        // wake up v2v thread
        sem_post(&v2v_send_sem);
timer_v2v_count = 0;
    }
    if (timer_ctrl_loop_count >= mc_init.ctrl_loop_period) {
        // wake up control thread
        sem_post(&ctrl_loop_sem);
timer_ctrl_loop_count = 0;
    }
}

void timer_init() {
    int i = 0;
    // setup signal handler
    struct sigaction sigact;
sigset_t set;
sigemptyset(&set);
sigaddset(&set, SIGALRM);
sigact.sa_flags = SA_RESTART;
sigact.sa_mask = set;
sigact.sa_handler = &handler;
    // register signal handler
    sigaction( SIGALRM, &sigact, NULL);
    // setup timer interval
    struct itimerspec tim_spec;
tim_spec.it_interval.tv_sec = 0;
tim_spec.it_interval.tv_nsec = 1000 * 1000;
tim_spec.it_value.tv_sec = 0;
```
```c
549    int spec.it_value.tv_nsec = 1000 * 1000;
550
551    // Specifying sevp as NULL is equivalent to
552    // specifying a pointer to a
553    // sigevent structure in which sigev_notify is
554    // SIGEV_SIGNAL, sigev_signo
555    // is SIGALRM, and sigev_value.sival_int is the timer ID.
556    if (timer_create(CLOCK_MONOTONIC, NULL, &t_id))
557        perror("timer_create");
559    if (timer_settime(t_id, 0, &tim_spec, NULL))
560        perror("timer_settime");
561    sem_init(&v2v_send_sem, 0, 0);
563 }
566    void timer_exit() {
567        timer_delete(t_id);
568        sem_destroy(&v2v_send_sem);
569    }
572 }
574    int main(int argc, char **argv) {
575        char fn_buff[64];
576        char suffix[128];
578        char conf_fn[1] = "init.conf";
580        if (argc == 1) {
581            time_t t = time(NULL);
582            struct tm tm = localtime(&t);
583            snprintf(suffix, sizeof(suffix),
584                "%04d_%02d_%02d_%02d_%02d_%02d",
585                tm.tm_year + 1900, tm.tm_mon + 1,
586                tm.tm_mday, tm.tm_hour, tm.tm_min,
587                tm.tm_sec);
589            } else if (argc == 2) {
592                strncpy(suffix, argv[1], sizeof(suffix));
593            } else {
595                printf("Usage: pi_mc init.conf\n");
596            }
598            // open log file and meta file
599            snprintf(fn_buff, sizeof(fn_buff), "log_%s.txt", suffix);
600            log_file = fopen(fn_buff, "w+");
602            snprintf(fn_buff, sizeof(fn_buff), "meta_%s.txt", suffix);
603            meta_file = fopen(fn_buff, "w+");
605            // Read and parse config file
606            if (parse_conf_file(conf_fn) >= 0) {
607                print_config(meta_file);
609            } else {
610                printf("<main> Error in reading config file!!!\n");
611                return -1;
612            }
613            recent_meta_file = fopen(mc_init.meta_file_name, "w+");
614            print_config(recent_meta_file);
```
recent_log_file = fopen( mci init .log_file_name , "w+" );

printf("<main> log file and meta file successfully created.\n" );

printf("<main> My VID is %d\n", mc_init.vehicle_id );

// initialize timer
timer_init ();

// create threads
pthread_t command_thread_tid;

pthread_t h2l_thread_id;

pthread_t ctrl_loop_thread_tid;

pthread_t v2v_sending_thread_tid;

pthread_t v2v_receiving_thread_tid;

pthread_t camera_thread_tid;

// keyboard command thread
pthread_create(&command_thread_tid, NULL, command_thread, NULL);

usleep(1000);

// h2l receiving thread (HLC <- LLC)
pthread_create(&h2l_thread_tid, NULL, h2l_receive_thread, NULL);

// Make sure serial port is correctly
// opened before ctrl_loop is started.
usleep(1000);

// h2l sending thread (HLC -> LLC)
pthread_create(&ctrl_loop_thread_tid, NULL,
ctrl_loop_thread, NULL);

usleep(1000);

// v2v thread
pthread_create(&v2v_sending_thread_tid, NULL,
v2v_sending_thread, NULL);

pthread_create(&v2v_receiving_thread_tid, NULL,
v2v_receiving_thread, NULL);

pthread_create(&camera_thread_tid, NULL, camera_thread, NULL);

pthread_join(command_thread_tid, NULL);

pthread_join(h2l_thread_tid, NULL);

pthread_join(ctrl_loop_thread_tid, NULL);

pthread_join(v2v_sending_thread_tid, NULL);

pthread_join(v2v_receiving_thread_tid, NULL);

pthread_join(camera_thread_tid, NULL);

// exit

timer_exit ();

fclose( meta_file );

fclose( recent_meta_file );

fclose( log_file );

fclose( recent_log_file );

printf("Exit.\n");
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <sys/types.h>

// see http://www.cmrr.umn.edu/~strupp/serial.html
// takes the string name of the serial port
// (e.g. "/dev/tty.usbserial", "COM1")
// and a baud rate (bps) and connects to
// that port at that speed and 8N1.
// opens the port in fully raw mode so you can send binary data.
// returns valid fd, or −1 on error
int serial_init(const char *serialport, int baud) {
    struct termios options;
    int fd;

    // fd = open(serialport, O_RDWR | O_NOCTTY | O_NDELAY);
    fd = open(serialport, O_RDWR | O_NOCTTY);
    // fd = open(serialport, O_RDWR | O_NONBLOCK);
    // fd = open(serialport, O_RDWR);

    if (fd == −1) {
        perror("serialport_init: Unable to open port ");
        return −1;
    }

    //int iflags = TIOCM_DTR;
    //ioctl(fd, TIOCMBS, &iflags); // turn on DTR
    //ioctl(fd, TIOCMIBC, &iflags); // turn off DTR

    if (tcgetattr(fd, &options) < 0) {
        perror("serialport_init: Couldn't get term attributes");
        return −1;
    }

    speed_t brate = baud; // let you override switch below if needed
    switch (baud) {
    case 4800:
        brate = B4800;
        break;
    case 9600:
        brate = B9600;
        break;
    #ifdef B14400
    case 14400: brate=B14400; break;
    #endif
    case 19200:
        brate = B19200;
        break;
    #ifdef B28800
    case 28800: brate=B28800; break;
    #endif
    case 38400:
        brate = B38400;
        break;
    case 57600:
        brate = B57600;
        break;
    case 115200:
        brate = B115200;
        break;
    }
break;
case 230400:
  brate = B230400;
break;
case 460800:
  brate = B460800;
break;
case 576000:
  brate = B576000;
break;
case 921600:
  brate = B921600;
break;
case 1152000:
  brate = B1152000;
break;
case 1500000:
  brate = B1500000;
break;
case 2000000:
  brate = B2000000;
break;
case 2500000:
  brate = B2500000;
break;
case 3000000:
  brate = B3000000;
break;
case 3500000:
  brate = B3500000;
break;
case 4000000:
  brate = B4000000;
break;
}  
cfsetispeed(&toptions, brate);
csetospeed(&toptions, brate);

// 8N1
toptions.c_cflag &= ~PARENB;
tooptions.c_cflag &= ~CSTOPB;
tooptions.c_cflag &= ~CSIZE;
tooptions.c_cflag |= CS8;
// no flow control

tooptions.c_cflag &= ~CRTSCTS;

// disable hang-up-on-close to avoid reset
//tooptions.c_cflag &= ~HUPCL;

// turn on READ & ignore ctrl lines

tooptions.c_cflag |= CREAD | CLOCAL;

// turn off s/w flow ctrl

tooptions.c_iflag &= ~(IXON | IXOFF | IXANY);

// make raw

tooptions.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
tooptions.c_oflag &= ~OPOST;

// see: http://unixwiz.net/techtips/termios-vmin-vtime.html
tooptions.c_cc[VMIN] = 0;
tooptions.c_cc[VTIME] = 0;
//toptions.c_cc[VTIME] = 20;
tcsetattr(fd, TCSANOW, &toptions);
if (tcsetattr(fd, TCSAFLUSH, &toptions) < 0) {
  perror("init_serialport: Couldn't set term attributes");
  return −1;
}
return fd;

//
int serial_close(int fd) {
    return close(fd);
}

int serial_recv_byte(int fd, char *buff) {
    return read(fd, buff, 1);
}

// CAUTION: blocking / busy waiting send
int serial_send_n_bytes(int fd, const char *buff, size_t n) {
    int ret = 0;
    int count = 0;
    while (count < n) {
        ret = write(fd, buff, n);
        if (ret < 0) {
            return ret;
        }
        count += ret;
    }
    return 0;
}

// CAUTION: blocking / busy waiting receive
int serial_recv_n_bytes(int fd, char *buff, size_t n) {
    int ret = 0;
    int count = 0;
    while (count < n) {
        ret = read(fd, buff + count, n - count);
        if (ret < 0) {
            return ret;
        }
        count += ret;
    }
    return count;
}

//
int serial_flush(int fd) {
    return tcflush(fd, TIOFLUSH);
}

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <time.h>
#include <signal.h>
#include <sys/types.h>
#include <pthread.h>
#include <semaphore.h>
#include <sys/socket.h>
#include <netinet/in.h>
```c
#include <arpa/inet.h>
#include "v2v.h"
#include "ctrl_loop.h"
#include "pi mc.h"

volatile int v2v_sending_should_stop = 0;
volatile int v2v_receiving_should_stop = 0;

const int nr_dst = 10;
struct sockaddr_in servaddr[10];
struct v2v_msg recent_msgs[10];

struct v2v_msg *get_recent_msg(int vid) {
    if (vid < 1 || vid > 9)
        return NULL;
    else
        return &recent_msgs[vid];
}

void print_v2v_msg_send(FILE *fd, struct v2v_msg *msg, int dst_id, int flag) {
    fprintf(fd, "[\%08d]\<\sl%ld -> %ld\>
          \<%4.4f, %4.4f, %4.4f, %4.4f, ctrl_flag %d\>\n",
            msg->ts, msg->vid, dst_id, flag,
            msg->theta, msg->accx, msg->v, msg->omega,
            msg->v_dsr, msg->ctrl_flag);
    fflush(fd);
}

void print_v2v_msg_recv(FILE *fd, struct v2v_msg *msg, int my_id) {
    struct timespec now;
    clock_gettime(CLOCK_MONOTONIC, &now);
    uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
    fprintf(fd, "[\%08d]\<[\%08d]\<\%ld -> %ld\>
          \<%4.4f, %4.4f, v: %4.4f, %4.4f, %4.4f, ctrl_flag %d\>\n",
        ts, msg->ts, msg->vid, my_id,
        msg->theta, msg->accx, msg->v, msg->omega,
        msg->v_dsr, msg->ctrl_flag);
    fflush(fd);
}

void *v2v_sending_thread(void *para) {
    int i, ret;
    struct timespec now;
    char addbuff[64];
    FILE *v2v_send_fd = fopen("v2v-send.txt", "w+");
    int sendsockfd = socket(AF_INET, SOCK_DGRAM, 0);
    for (i = 1; i < nr_dst; i++) {
        snprintf(addbuff, sizeof(addbuff), "%d.168.0.10%d", i);
        memset(&servaddr[i], 0, sizeof(servaddr[i]));
        servaddr[i].sin_family = AF_INET;
        servaddr[i].sin_addr.s_addr = inet_addr(addbuff);
        servaddr[i].sin_port = htons(32000);
```
memset(&servaddr[0], 0, sizeof(servaddr[0]));
servaddr[0].sin_family = AF_INET;
servaddr[0].sin_addr.s_addr = inet_addr(mc_init.manager_ip);
servaddr[0].sin_port = htons(32000);

printf("<v2v> V2V sending thread started.\n");

while (!v2v_sending.should_stop) {
    sem_wait(&v2v_send_sem);

clock_gettime(CLOCK_MONOTONIC, &now);
uint32_t ts = (uint32_t)(now.tv_sec * 1000UL + now.tv_nsec / 1000000);

struct v2v_msg vmsg;

vmsg.ts = ts;
vmsg.vid = (uint32_t)mc_init.vehicle_id;
vmsg.theta = vstate.imu_theta;
vmsg.accx = vstate.imu.accx;
vmsg.v = vstate.v;
vmsg.omega = vstate.omega;
vmsg.v_dsr = vstate.v_dsr;
vmsg.ctrl_flag = vstate.ctrl_flag;

if (mc_init.vehicle_id == 1) {
    // Platooning header broadcast its states
    for (i = 2; i < nr_dst; i++) {
        ret = sendto(sendsockfd, &vmsg, sizeof(vmsg),
                      MSG_DONTWAIT,
                      (struct sockaddr *)&servaddr[i],
                      sizeof(servaddr[i]));
        printf("v2v_msg_send(v2v_send_fd, &vmsg, i, ret);";
    }
} else {

    // Each vehicle sends its state to immediate successor,
    // except for the last one.
    if (mc_init.vehicle_id != nr_dst - 1) {
        int dst_id = mc_init.vehicle_id + 1;
        ret = sendto(sendsockfd, &vmsg, sizeof(vmsg),
                     MSG_DONTWAIT,
                     (struct sockaddr *)&servaddr[dst_id],
                     sizeof(servaddr[dst_id]));
        printf("v2v_msg_send(v2v_send_fd, &vmsg, dst_id, ret);";
    }
}

fclose(v2v_send_fd);

printf("<v2v> V2V sending thread exiting...\n");
return NULL;

void *v2v_receiving_thread(void *para) {

}
int ret;
int sockfd;
struct sockaddr *servaddr;
struct sockaddr *claddr;
socklen_t len;
char recv_buff[1024];

FILE *v2v_recv = fopen("v2v-recv.txt", "w+");
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(32000);
bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

// receiving function timeout every 1 second.
// so as to check v2v receiving should stop.
struct timeval tv;
tv.tv_sec = 1;
tv.tv_usec = 0;
if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO,
    &tv, sizeof(tv)) < 0) {
    perror("Error in setting receiving timeout");
}

printf("<v2v> V2V receiving thread started.\n");

while (!v2v_receiving_should_stop) {
    ret = recvfrom(sockfd, recv_buff, sizeof(recv_buff), 0,
        (struct sockaddr *)&claddr, &len);
    if (ret > 0) {
        struct v2v_msg *msg = (struct v2v_msg *) recv_buff;
        print_v2v_msg_recv(v2v_recv, msg, mc_init.vehicle_id);

        // update received state
        uint32_t ts = msg->ts;
        uint32_t id = msg->vid;
        if (id >= 1 && id <= 9) {
            // if (ts > recent_msgs[id].ts)
            recent_msgs[id] = *msg;
        }
    }
}

fclose(v2v_recv);

printf("<v2v> V2V receiving thread exiting...\n");
return NULL;
APPENDIX B

MATLAB CODE
% Load Model
P2wrwl = load ('ETT_TITO_Model_2nd_order.mat');
P4wrwl = load ('ETT_TITO_Model_4th_order.mat');
P2w = load ('ETT_Pvw_Model_2nd_order.mat');
P4w = load ('ETT_Pvw_Model_4th_order.mat');

% Robot Singular Values (Voltages to Wheel Speeds) — Including Low Frequency Approximation
fig34 = figure (34);
sigma (P4wrwl.TITO_SS,P2wrwl.TITO_SS)
hold on; grid on;
title('Singular Values for TITO Model', 'FontSize', 24);
legend('Without approximation', 'Low frequency approximation')
h_line = findobj(fig34, 'type', 'line');
h_axes = findobj(fig34, 'type', 'axes');
set(h_line, 'LineWidth', 3);
set(h_axes, 'FontSize', 15);
xlabel('Frequency', 'FontSize', 24);
ylabel('Singular Values', 'FontSize', 24);

% Robot Frequency Response (Voltages to Wheel Speeds) — Including Low Frequency Approximation
fig35 = figure (35);
bode(P4wrwl.TITO_SS,P2wrwl.TITO_SS)
hold on; grid on;
title('Frequency Response: Voltage to \omega_R, \omega_L', 'FontSize', 24);
legend('Without approximation', 'Low frequency approximation')
h_line = findobj(gcf, 'type', 'line');
h_axes = findobj(gcf, 'type', 'axes');
set(h_line, 'LineWidth', 3);
set(h_axes, 'FontSize', 15);
xlabel('Frequency', 'FontSize', 24);
ylabel('Phase', 'FontSize', 24);

% SVD analysis
% Figure 3.6: Robot Plant Singular Values (Voltages to v and w) — Including Low Frequency Approximation
fig36 = figure (36);
sigma(P4vw.TITO_SS,P2vw.TITO_SS)
hold on; grid on;
title('Singular Values for Plant', 'FontSize', 24);
legend('Without approximation', 'Low frequency approximation')
h_line = findobj(gcf, 'type', 'line');
h_axes = findobj(gcf, 'type', 'axes');
set(h_line, 'LineWidth', 3);
set(h_axes, 'FontSize', 15);
xlabel('Frequency', 'FontSize', 24);
ylabel('Singular Values', 'FontSize', 24);

% Figure 3.7: Robot Plant Frequency Response (Voltages to [v;w]) — Including Low Frequency Approximation
opts = bodeoptions;
% opts.InputLabels.FontSize = 20;
% opts.OutputLabels.FontSize = 20;

bode(P4vw.TITO_SS, P2vw.TITO_SS)
hold on; grid on;
title('Frequency Response: Voltage to \[ v, \omega \]','FontSize', 24);
legend('Without approximation','Low frequency approximation')

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'FontSize', 15);
xlabel('Frequency','FontSize', 24);
ylabel('Phase','FontSize', 24);

% % Singular Value of M
M = P4wrwl.ETT_M
M_inv = inv(M)
svd(M)
svd(M_inv)
norm(M, inf)
norm(M_inv, inf)

% % FIGURE in Chapter 4
% Robot Frequency Response (Voltages to Wheel Speeds) - Including Low Frequency Approximation
fig45 = figure(45);
hold on; grid on;
title('Frequency Response: Voltage to \[ \omega_R, \omega_L \]','FontSize', 24);
legend('Without approximation','Low frequency approximation')

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'FontSize', 15);
xlabel('Frequency', 'FontSize', 24);
ylabel('Phase', 'FontSize', 24);

% % Robot Step Response (Voltages to Wheel Speeds)
fig46 = figure(46);
hold on; grid on;
title('Step Response: Voltage to \[ \omega_R, \omega_L \]','FontSize', 24);
legend('Without approximation','Low frequency approximation')

% % Auto Print Figure after enlarge all of them
% print(fig34, 'Singular Values for TITO Model', '-depsc', '-tiff')
% print(fig34, 'Singular Values for TITO Model', '-dpng', '-r0')
a31 = -Kb*Kg/(La*r); a32 = -Kb*Kg*dw/(2*La*r); a33 = Ra/La; a34 = 0;

a41 = a31; a42 = -a32; a43 = 0; a44 = a33;

M = [r/2 r/2; r/dw -r/dw];
A = [ a11 a12 a13 a14; a21 a22 a23 a24; a31 a32 a33 a34; a41 a42 a43 a44];
B = [zeros(2,2); eye(2)]./La;
C = [inv(M) zeros(2,2)];
D = zeros(2,2);

% % Still Looks very Ugly Use Maple To Get it
% syms s
% I4 = eye(4);
% P = C/(s*I4 - A)*B + D
% pretty (simplify(P))

% Vehicle Parameters for Our Experimental Enhanced Thunder Tumbler
close all
ETTm = 0.89;
ETTbeta = 7.04e-7;
ETTbeta = 0.05;
ETTJz = 0.0051; % Iz/m*1/12*(L^2+W^2)
% Vicent 0.0051; LIN 0.0013
ETT = 18.48;
ETT_Kt = 0.0032;
ETT_Kb = 0.0032;
ETT_dw = 0.14;
ETT_Ra = 0.79;
ETT_La = 265e-6;

% beta variation
% variation_vec = [ETTbeta/4 ETTbeta/2 ETTbeta ETTbeta*2];
% titlerstr_bodemag
% 'Bode magnitude plots of the plant for variations of damping costant'
% titlerstr_step
% 'Step response of the plant for variations of damping costant'
% legendstr_cell
% {'\beta = 0.0176e-5 (Nms)'}, {'\beta = 0.0352e-5 (Nms)'}, ...
% {'\beta = 0.0704e-5 (Nms)'},
% {'\beta = 0.1408e-5 (Nms)'};

% moment of inertia Iz variation
% variation_vec = [ETT_Jz/4 ETT_Jz/2 ETT_Jz ETT_Jz*2];
% titlerstr_bodemag
% 'Bode magnitude plots of the plant
% for variations of moment of inertia'
% titlerstr_step
% 'Step response of the plant for variations of moment of inertia'
% legendstr_cell = {'I_z = 0.0013 (Kg*m^2)'}, {'I_z = 0.0026 (Kg*m^2)'}, ...
% {'I_z = 0.0051 (Kg*m^2)'}, {'I_z = 0.0102 (Kg*m^2)'};

% moment of inertia Ra variation
% variation_vec = [ETT_Ra/2 ETT_Ra ETT_Ra*2 ETT_Ra*3];
% titlerstr_bodemag
% 'Bode magnitude plots of the plant
% for variations of armature resistance';

267
\begin{verbatim}
81  \% titlestr_step
82  \%= 'Step response of the plant for variations of armature resistance';
83  \% legendstr_cell = {'R_a = 0.3950 (\Omega)', 'R_a = 0.7900 (\Omega)', ...
84  \%    'R_a = 1.5800 (\Omega)', 'R_a = 2.3700 (\Omega)'};
85
86  \% moment of inertia Kt variation
87  \% variation_vec = [ETT_Kt/2 ETT_Kt/1.5 ETT_Kt ETT_Kt*1.5];
88  \% titlestr_bodemag
89  \%= 'Bode magnitude plots of the plant for variations of torque constant';
90  \% titlestr_step
91  \%= 'Step response of the plant for variations of torque constant';
92  \% legendstr_cell = {'K_t = 0.0016 (NmA)', 'K_t = 0.0021 (NmA)',...
93  \%    'K_t = 0.0032 (NmA)', 'K_t = 0.0048 (NmA)'};
94
95  \% moment of inertia Kb variation
96  \% variation_vec = [ETT_Kb/2 ETT_Kb/1.5 ETT_Kb ETT_Kb*1.5];
97  \% titlestr_bodemag
98  \%= 'Bode magnitude plots of the plant for variations of back emf constant';
99  \% titlestr_step
100 \%= 'Step response of the plant for variations of back emf constant';
101 \% legendstr_cell = {'K_b = 0.0016 (V/(rad/sec))','K_b = 0.0021 (V/(rad/sec))',...
102 \%    'K_b = 0.0032 (V/(rad/sec))','K_b = 0.0048 (V/(rad/sec))'};
103
104 \% Mass m variation
105 \% variation_vec = [ETT_m*0.6 ETT_m*0.8 ETT_m*1 ETT_m*1.2];
106 \% titlestr_bodemag
107 \%= 'Bode magnitude plots of the plant for variations of mass';
108 \% titlestr_step
109 \%= 'Step response of the plant for variations of mass';
110 \% legendstr_cell = {'m = 0.5340 (kg)','m = 0.7120 (kg)',...
111 \%    'm = 0.8900 (kg)','m = 1.0680 (kg)'};
112
113 \% Plant_cell = cell(length(variation_vec),1);
114
115 \for ii = 1:1:4
116 \% ETT_beta = variation_vec(ii);
117 \% ETT_Iz = variation_vec(ii);
118 \% ETT_Ra = variation_vec(ii);
119 \% ETT_Kt = variation_vec(ii);
120 \% ETT_Kb = variation_vec(ii);
121 \% ETT_m = variation_vec(ii);
122
123 \end{verbatim}

268
ETT_B = double(subs(B, La, ETT_La));
ETT_C = double(subs(C, {r dw}, {ETT_x, ETT_dw}));
ETT_D = D;
TITO_SS = ss(ETT_A, ETT_B, ETT_C, ETT_D);

Plant_cell{ii} = TITO_SS;

fig10 = figure(10);
bodemag(TITO_SS);
hold on; grid on;
title(titlestr_bodemag, 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Magnitude', 'FontSize', 24);
legend(legendstr_cell);

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);
h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

fig20 = figure(20);
step(TITO_SS);
hold on; grid on;
title(titlestr_step, 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
legend(legendstr_cell);
hold on; grid on;

P_dc = ETT_C / (I4 - ETT_A) * ETT_B + ETT_D;
P_vpa = vpa(P_dc, 3)
P_vpa = vpa(P(1, 1), 3)

TF = ss2tfm(ETT_A, ETT_B, ETT_C, ETT_D);
TF_zpk = zpk(TF)

% Frequency and Step Response of couple model
% close all
% fig10 = figure(10);
% bodemag(TITO_SS);
% hold on; grid on;
% title('Frequency response of coupled model')
% fig20 = figure(20);
% step(TITO_SS);
% hold on; grid on;
% Output response of coupled model

% Frequency and Step Response of decoupled model
% first get decoupled model by removing off-diagonal element in TFM

% TF_decoupled(1,1)= TF(1,1);
% TF_decoupled(2,2)= TF(2,2);

% Inner loop trade study on TITO model

clear all
close allclc

% Load Model
P2wrwl = load('ETT_TITO_Model_2nd_order.mat');
P4wrwl = load('ETT_TITO_Model_4th_order.mat');
P2vw = load('ETT_Pvw_Model_2nd_order.mat');
P4vw = load('ETT_Pvw_Model_4th_order.mat');
P = P2wrwl.TITO_SS;
dw = 0.14;
r = 0.05;
M = [r/2, r/2, r/dw, -r/dw];
Minv = inv(M);

Ap = P.a; Bp = P.b; Cp = P.c; Dp = P.d;
s = tf('s');

% Open loop singular values, sensitivity, comp sensitivity for 4 diff values of g
w = logspace(-3,3,100);
z = 1.26;
g = [0.07, 0.1, 0.15];
z = 2;
g = [1 3 5 7] .* 0.1;

% for ii = 1:length(g)
%  for ii = 1:1:

%  K = [ g(ii)*((s + z)/s)*(40/(s+40))  0
%        0 g(ii)*((s + z)/s)*(40/(s+40)) ];

%  K = ss(K);

%  W = [ z/(s+z)  0
%        0 z/(s+z) ];

% figure(10)
sigma(K,w);
%sv = 20*log10(sv);
%semilogx(w, sv, str(i,:))
title('Controller Singular Values ', 'FontSize', 24)
xlabel('Frequency ', 'FontSize', 24)
ylabel('Singular Values ', 'FontSize', 24);
legend('g = 0.10 , z = 2', 'g = 0.30 , z = 2', 'g = 0.50 , z = 2', 'g = 0.70 , z = 2')
grid on
hold on

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

L = P*K;

figure(11)
sigma(L,w);

sv = 20*log10(sv);
semilogx(w, sv, str(i,:))
title('Open Loop Singular Values', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
legend( 'g = 0.10, z = 2', 'g = 0.30, z = 2', ...
    'g = 0.50, z = 2', 'g = 0.70, z = 2')
grid on

hold on

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

% %
% % Sensitivity
asen = L.a - L.b*L.c;
bsen = L.b;
csen = -L.c;
[row, col] = size(csen);
[row1, col1] = size(bsen);
dsen = eye(row, col1);

figure(12)
sigma(ss(asen, bsen, csen, dsen),w);

sv = 20*log10(sv);
semilogx(w, sv, str(i,:))
title('Sensitivity', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
legend( 'g = 0.10, z = 2', 'g = 0.30, z = 2', ...
    'g = 0.50, z = 2', 'g = 0.70, z = 2')
grid on

hold on

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

% %
% % Complementary sensitivity
acl = L.a - L.b*L.c;
bcl = L.b;
ccl = L.c;
dcl = L.d;
cls = ss(acl, bcl, ccl, dcl);
figure (13)
sigma (ss (acl, bcl, ccl, dcl), w);
%sv = 20* log10 (sv);
% semilogx (w, sv, str (i, :))
title ('Complementary Sensitivity ', 'FontSize', 24)
xlabel ('Frequency ', 'FontSize', 24)
ylabel ('Singular Values ', 'FontSize', 24);
grid on
hold on
legend ( 'g = 0.10, z = 2', 'g = 0.30, z = 2', 'g = 0.50, z = 2', 'g = 0.70, z = 2');

figure (14)
step (cls)

title ('Step Response for T\{ru\} (Unfiltered)', 'FontSize', 24)
xlabel ('Time ', 'FontSize', 24)
ylabel ('Angular Velocity (rad/s)', 'FontSize', 24);
grid on
hold on
legend ( 'g = 0.10, z = 2', 'g = 0.30, z = 2', 'g = 0.50, z = 2', 'g = 0.70, z = 2');

figure (15)
sigma (ss (Aru, Bru, Cru, Dru), w);
%sv = 20* log10 (sv);
% semilogx (w, sv, str (i, :))
title ('T\{ru\} (Unfiltered) ', 'FontSize', 24)
xlabel ('Frequency ', 'FontSize', 24)
ylabel ('Singular Values ', 'FontSize', 24);
grid on
hold on
legend ( 'g = 0.10, z = 2', 'g = 0.30, z = 2', 'g = 0.50, z = 2', 'g = 0.70, z = 2');

% Reference to control Tru no filter
% states [xp xk]^T
Bru = [Bp*K.d K.b ];
Cru = [-K.d*Cp  K.c ];
Dru = K.d;
Tru = ss (Aru, Bru, Cru, Dru);

figure (15)
h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);

% Control response to step command
figure(16)
step(Tru)
grid on
hold on
title('Control Response for step command (Unfiltered)', ...'
'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Voltage (V)', 'FontSize', 24)
legend(['g = 0.10, z = 2', 'g = 0.30, z = 2', ...'
  'g = 0.50, z = 2', 'g = 0.70, z = 2'])

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);

% Reference to control filtered TruW
F_True = TruW;

figure(17)
sigma(F_True,w);

%sv = 20*log10(sv);
%semilogx(w, sv, str(i,:))
title('Tr\{ru\}\cdotW', 'FontSize', 24)
grid on
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24)
hold on
legend(['g = 0.10, z = 2', 'g = 0.30, z = 2', ...'
  'g = 0.50, z = 2', 'g = 0.70, z = 2'])

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);

% Control response to filtered step command
figure(18)
step(F_True)
grid on
hold on
title('Control Response for step command with pre-filter', ...'
'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Voltage (V)', 'FontSize', 24)
legend(['g = 0.10, z = 2', 'g = 0.30, z = 2', ...'
  'g = 0.50, z = 2', 'g = 0.70, z = 2'])

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);
%% Step response with prefilter Try W

figure(19)
step([W)]
grid on
hold on
title('Step Response T_{\text{ty}} \cdot W', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Angular Velocity (rad/s)', 'FontSize', 24)
legend('{g = 0.10, z = 2}', '{g = 0.30, z = 2}', '{g = 0.50, z = 2}', '{g = 0.70, z = 2}')

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);
h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

%%

% Tdy states: [xp xk]^T
−K.b∗Cp K.a ];
[row, col] = size(K.b);
Bdiy = [
0*ones(row, 2)
];
[row1, col1] = size(K.a);
Cdiy = [Cp 0*ones(2, col1)];
Ddiy = 0*ones(2, 2);
Tdiy = ss(Adiy, Bdiy, Cdiy, Ddiy);

figure(191)
sigma(Tdiy);%

%sv = 20*log10(sv);
%semilogx(w, sv, str(i,:))
title('T_{\text{dy}}', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
legend('{g = 0.10, z = 2}', '{g = 0.30, z = 2}', '{g = 0.50, z = 2}', '{g = 0.70, z = 2}')
grid on
hold on

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);
h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

%%

% MSP Tdy for (v,w)
Tdiy_vw = Ms*Tdiy;
figure(192)
sigma(Tdiy_vw);
title('MSP Singular Values', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
grid on
hold on

legend('{g = 0.10, z = 2}', '{g = 0.30, z = 2}', '{g = 0.50, z = 2}')
h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidht', 2);
set(h_axes, 'FontSize', 15);

%%%% KSM^(-1) unfiltered Tru for (v,w)
Tru_vw = Tru*Minv;
figure(193)
sigma(Tru_vw,w);
title('KSM^{-1} Singular Values', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24)
grid on
hold on

legend('g = 0.10, z = 2', 'g = 0.30, z = 2', ... 'g = 0.50, z = 2', 'g = 0.70, z = 2', ... 'g = 0.50, z = 1', 'g = 0.50, z = 2', ... 'g = 0.50, z = 3', 'g = 0.50, z = 4')

end

%%%% Open loop singular values, sensitivity, comp sensitivity for 4 diff values of z

g = 0.5;
z = [1 2 3 4];
for jj = 1:length(z)
    K = g*((s+z(jj))/s)*(40/(s+40))
    W = [z(jj)/(s+z(jj)) 0; 0 z(jj)/(s+z(jj)) ];
end

% figure (20)
sigma(K,w);
sv = 20*log10(sv);
semilogx(w, sv, str(i,:))
title('Controller Singular Values', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
legend('g = 0.50, z = 1', 'g = 0.50, z = 2', ... 'g = 0.50, z = 3', 'g = 0.50, z = 4')
grid on
hold on

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidht', 2);
set(h_axes, 'FontSize', 15);
\[ L = P \ast K; \]

figure (21)

\begin{verbatim}
\texttt{sigma(L,w);}
\texttt{\%sv = 20*log10(sv);}
\texttt{\%semilogx(w, sv, str(i,:));}
\texttt{title(’Open Loop Singular Values’,’FontSize’, 24)}
\texttt{xlabel(’Frequency’,’FontSize’, 24)}
\texttt{ylabel(’Singular Values’,’FontSize’, 24);}  
\texttt{legend(’g = 0.50,z = 1’,’g = 0.50,z = 2’,’g = 0.50,z = 3’,’g = 0.50,z = 4’)}
\texttt{grid on}
\texttt{hold on}
\end{verbatim}

\begin{verbatim}
\begin{align*}
\texttt{h\_line = findobj(gcf, ’type’, ’line’);} \\
\texttt{set(h\_line, ’LineWidth’, 3);} \\
\texttt{h\_axes = findobj(gcf, ’type’, ’axes’);} \\
\texttt{set(h\_axes, ’linewidth’, 2);} \\
\texttt{set(h\_axes, ’FontSize’, 15);} \\
\end{align*}
\end{verbatim}

\begin{verbatim}
\% Sensitivity
\\texttt{asen = L.a - L.b*L.c;} \\
\texttt{bsen = L.b;} \\
\texttt{csen = -L.c;} \\
\texttt{[row, col] = size(csen);} \\
\texttt{[row1, col1] = size(bsen);} \\
\texttt{dsen = eye(row, col1);} \\
\texttt{figure (22)}
\texttt{sigma(ss(asen,bsen, csen, dsen),w);}
\texttt{\%sv = 20*log10(sv);}
\texttt{\%semilogx(w, sv, str(i,:));}
\texttt{title(’Sensitivity’,’FontSize’, 24)}
\texttt{xlabel(’Frequency’,’FontSize’, 24)}
\texttt{ylabel(’Singular Values’,’FontSize’, 24);}  
\texttt{legend(’g = 0.50,z = 1’,’g = 0.50,z = 2’,’g = 0.50,z = 3’,’g = 0.50,z = 4’)}
\texttt{grid on}
\texttt{hold on}
\end{verbatim}

\begin{verbatim}
\begin{align*}
\texttt{h\_line = findobj(gcf, ’type’, ’line’);} \\
\texttt{set(h\_line, ’LineWidth’, 3);} \\
\texttt{h\_axes = findobj(gcf, ’type’, ’axes’);} \\
\texttt{set(h\_axes, ’linewidth’, 2);} \\
\texttt{set(h\_axes, ’FontSize’, 15);} \\
\end{align*}
\end{verbatim}

\begin{verbatim}
\%Closed loop dynamics
\texttt{acl = L.a - L.b*L.c;} \\
\texttt{bcl = L.b;} \\
\texttt{ccl = L.c;} \\
\texttt{dcl = L.d;} \\
\texttt{cls = ss(acl,bcl,ccl,dcl);} \\
\texttt{figure (23)}
\texttt{sigma(ss(acl,bcl,ccl,dcl),w);}
\texttt{\%sv = 20*log10(sv);}
\texttt{\%semilogx(w, sv, str(i,:));}
\texttt{title(’Complementary Sensitivity’,’FontSize’, 24)}
\texttt{xlabel(’Frequency’,’FontSize’, 24)}
\texttt{ylabel(’Singular Values’,’FontSize’, 24);}  
\texttt{grid on}
\end{verbatim}

% Figure 21

The figure illustrates the open-loop singular values for different gain settings, showing the sensitivity of the system to variations. The code plots these values on a logarithmic scale, and the legend indicates the specific gain and zero settings for each curve.

% Figure 22

This figure presents the sensitivity analysis, revealing how changes in the gain and zero parameters affect the singular values of the complementary sensitivity function. The plot is again on a logarithmic scale, and the legend highlights the gain and zero values.

% Figure 23

Finally, the closed-loop dynamics are examined, with the complementary sensitivity function plotted for various gain and zero configurations. The logarithmic scale is used to visualize the system's behavior under different conditions.
hold on

legend('g = 0.50, z = 1', 'g = 0.50, z = 2', ...
      'g = 0.50, z = 3', 'g = 0.50, z = 4')

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);

%%
%% Step response unfiltered
figure(24)
step(cls)
title('Step Response for T_{ry} (Unfiltered)', ...
      'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Angular Velocity (rad/s)', 'FontSize', 24)
grid on
hold on

legend('g = 0.50, z = 1', 'g = 0.50, z = 2', ...
      'g = 0.50, z = 3', 'g = 0.50, z = 4')

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);

%%
%% Reference to control Tru no filter
% states [xp xk] \dot{\hat{T}}
Aru = [Ap-Bp*K.d*Cp Bp*K.c
      -K.b*Cp K.a ];
Bru = [Bp*K.d
      K.b ];
Cru = [-K.d*Cp K.c ];
Dru = K.d;
Tru = ss(Aru, Bru, Cru, Dru);

figure(25)
sigma(ss(Aru, Bru, Cru, Dru),w);
%sv = 20*log10(sv);
%semilogx(w, sv, str(i,:))
title('T_{ru} (Unfiltered)', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
grid on
hold on

legend('g = 0.50, z = 1', 'g = 0.50, z = 2', ...
      'g = 0.50, z = 3', 'g = 0.50, z = 4');

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);
% Control response to step command
figure(26)
step(Tru)
hold on; grid on;
title('Control Response for step command (Unfiltered)', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Voltage (V)', 'FontSize', 24)
legend('
% Reference to control filtered Tru W
F_Trw = Tru * W;
figure(27)
sigma(F_Trw, w);
%sv = 20 * log10(sv);
%semilogx(w, sv, str(i,:))
title('T_{\text{ru}} \cdot W', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24)
grid on
hold on
legend('
% Control response to filtered step command
figure(28)
step(F_Trw)
grid on
hold on

title('Control Response for step command with pre-filter', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Voltage (V)', 'FontSize', 24)
legend('
% Step response with prefilter Try W
figure(29)
step(cls * W)
grid on
hold on
title('Step Response T_{\{ry\}} cdot W ', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Angular Velocity (rad/s)', 'FontSize', 24)
legend('
g = 0.50, z = 1', 'g = 0.50, z = 2', ..., 
g = 0.50, z = 3', 'g = 0.50, z = 4')
h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);
h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'FontSize', 24);

%%%%

% Tdiy states: [xp xk]^T
Adiy = [Ap-Bp*K.d*Cp K.a]
-K.b*Cp
[row, col]=size(K.b);
Bdiy = [Bp 0*ones(row,2)];
[row1, col1]=size(K.a);
Cdiy = [Cp 0*ones(2, col1)];
Ddiy = 0*ones(2, 2);
Tdiy = ss(Adiy, Bdiy, Cdiy, Ddiy);

figure(291)
sigma(Tdiy, w);
hold on

%%%%

% MSP Tdiy for (v,w)
Tdiy_vw = M*Tdiy;
figure(292)
sigma(Tdiy_vw, w);

%%%%

% TS2
Tdiy_vw = M*Tdiy;
figure(293)
sigma(Tdiy_vw, w);

%%%%
% inner loop design
clear all
close all
clc

S_avg = load('ETT_avg.mat')
S2 = load('ETT_TITO_Model_2nd_order.mat')
M = S2.ETT_M;
Minv = inv(M);
p11 = S_avg.tf_ett;
P=tf([p11 0;
0 p11]);
figure(1)
step(P, 5)

title('Plant Step Response', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
grid on
hold on

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);
h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

end
figure(2)
bode(P)
title('Plant Frequency Response', 'FontSize', 24);
xlabel('Frequency', 'FontSize', 24);
ylabel('Phase', 'FontSize', 24);
hold on; grid on;

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'LineWidth', 2);
set(h_axes, 'FontSize', 15);

%%
g = 0.2942;
z = 2.0091;

Kinner_tf = [ g*(s+z)*100/(s*(s+100)) 0
             0 g*(s+z)*100/(s*(s+100)) ];

Kinner = ss(Kinner_tf);

Linner = P*Kinner; %open loop

asen = Linner.a - Linner.b*Linner.c; % sensitivity
bsen = Linner.b;
csen = -Linner.c;
[row, col] = size(csen);
[row1, col1] = size(bsen);
dsen = eye(row, col);
Sinner = ss(asen, bsen, csen, dsen);

acl = Linner.a - Linner.b*Linner.c;
% comp sensitivity unfiltered
bcl = Linner.b;
ccl = Linner.c;
dcl = Linner.d;
T = ss(acl, bcl, ccl, dcl);

W_tf = [ z/(s+z) 0
        0 z/(s+z) ];
W = ss(W_tf);

Try = T*W; % try = comp sensitivity filtered

% Tdiy
Adiy = [Ap-B*Kinner.d*Cp Bp*Kinner.c
        -Kinner.b*Cp Kinner.a ];
[row, col] = size(Kinner.b);
Bdiy = [ 0*ones(row,2) ];
[row, col1] = size(Kinner.a);
Cdiy = [Cp 0*ones(2, col1)];
Ddiy = 0*ones(2,2);
Tdiy = ss(Adiy, Bdiy, Cdiy, Ddiy);

Tru = Kinner*Sinner; % Tru unfiltered
Truf = Kinner*Sinner*W; % Tru filtered
Tru\_vw = Tru*Minv; \hspace{1em} \% Tru vw unfiltered
Tdiy\_vw = M*Tdiy; \hspace{1em} \% Tdiy vw

figure(3) \hspace{1em} \% open loop singular value
w = logspace(-2,3,100);
Lv\_w = Minv * Linner * M
sigma(Linner,Lw,w);
title('Open Loop Singular Values', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
legend('PK singular values', '\(M\hat{\Sigma}^{-1}\) singular values')
gridd on
hold on

h\_line = findobj(gcf, 'type', 'line');
set(h\_line, 'LineWidth', 3);
h\_axes = findobj(gcf, 'type', 'axes');
set(h\_axes, 'linestyle', '2');
set(h\_axes, 'FontSize', 15);

figure(4) \hspace{1em} \% sensitivity
sigma(Sinner,w);
title('Sensitivity Singular Values', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
legend(XXX)
gridd on
hold on

h\_line = findobj(gcf, 'type', 'line');
set(h\_line, 'LineWidth', 3);
h\_axes = findobj(gcf, 'type', 'axes');
set(h\_axes, 'linestyle', '2');
set(h\_axes, 'FontSize', 15);

figure(5) \hspace{1em} \% comp sensitivity
sigma(T,w);
title('Complementary Sensitivity Singular Values', ...
'Frequency', 'FontSize', 24)
xlabel('Frequency', 'FontSize', 24)
ylabel('Singular Values', 'FontSize', 24);
legend(XXX)
gridd on
hold on

h\_line = findobj(gcf, 'type', 'line');
set(h\_line, 'LineWidth', 3);
h\_axes = findobj(gcf, 'type', 'axes');
177    set(h_axes, 'linewidth', 2);
178    set(h_axes, 'FontSize', 15);
179
180
181    \%
182    figure(6)  \%tdiy
183    sigma(Tdiy,w);
184    title('T_{\text{diy}} Singular Values', 'FontSize', 24)
185    xlabel('Frequency', 'FontSize', 24)
186    ylabel('Singular Values', 'FontSize', 24);
187    \%    legend(XXX)
188    grid on
189    hold on
190
191    h_line = findobj(gcf, 'type', 'line');
192    set(h_line, 'LineWidth', 3);
193    h_axes = findobj(gcf, 'type', 'axes');
194    set(h_axes, 'linewidth', 2);
195    set(h_axes, 'FontSize', 15);
196
197    \%
198    figure(7)  \%tru
199    sigma(Tru,w);
200    title('T_{\text{ru}} Singular Values', 'FontSize', 24)
201    xlabel('Frequency', 'FontSize', 24)
202    ylabel('Singular Values', 'FontSize', 24);
203    \%    legend(XXX)
204    grid on
205    hold on
206
207    h_line = findobj(gcf, 'type', 'line');
208    set(h_line, 'LineWidth', 3);
209    h_axes = findobj(gcf, 'type', 'axes');
210    set(h_axes, 'linewidth', 2);
211    set(h_axes, 'FontSize', 15);
212
213    \%
214    figure(8)  \%tru_filtered
215    sigma(Truf,w);
216    title('T_{\text{ru filtered}} W Singular Values', 'FontSize', 24)
217    xlabel('Frequency', 'FontSize', 24)
218    ylabel('Singular Values', 'FontSize', 24);
219    \%    legend(XXX)
220    grid on
221    hold on
222
223    h_line = findobj(gcf, 'type', 'line');
224    set(h_line, 'LineWidth', 3);
225    h_axes = findobj(gcf, 'type', 'axes');
226    set(h_axes, 'linewidth', 2);
227    set(h_axes, 'FontSize', 15);
228
229
230
231
232
233
234
235
236
283
%% step opt setting
opt = stepDataOptions('StepAmplitude',10);

%%
figure(9)  % Try W
step(Tryf,5,'b',opt);
title('Step response (Filtered)', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
hold on; grid on;

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

%%
figure(10)  % Try
step(T,5, 'b', opt);
title('Step response (Unfiltered)', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
hold on; grid on;

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

%%
figure(11)  % Tru
step(Truf,5, 'b', opt);
title('Control response (Unfiltered)', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Voltage(V)', 'FontSize', 24);
hold on; grid on;

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

%%
figure(12)  % Tru W
step(Truf,5, 'b', opt);
title('Control response (Filtered)', 'FontSize', 24)
xlabel('Time', 'FontSize', 24)
ylabel('Voltage (V)', 'FontSize', 24);
hold on; grid on;

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);
figure (13)  \texttt{MSP}
\begin{verbatim}
sigma (T_{diy \textsc{w}}, w);
title ('MSP Singular Values', 'FontSize', 24)
xlabel ('Frequency', 'FontSize', 24)
ylabel ('Singular Values', 'FontSize', 24);
end
\end{verbatim}

figure (14)  \texttt{KSM}^{-1}
\begin{verbatim}
sigma (T_{\textsc{w}w}, w);
title ('KSM^{-1} Singular Values', 'FontSize', 24)
xlabel ('Frequency', 'FontSize', 24)
ylabel ('Singular Values', 'FontSize', 24);
\end{verbatim}

% loopsens function testing
\begin{verbatim}
K_{tf} = K_{inner\_tf};
SF = loopsens(P_{tf}, K_{tf})
T_{wrwl} = \textsc{series}(W_{tf}, SF.To)
figure (1000);
bode(T_{wrwl}, 'b^+', Try, 'r-')
\end{verbatim}

\begin{verbatim}
[num1, den1] = ss2tf(Try.a, Try.b, Try.c, Try.d, 1);
n11 = num1(1,:);
d11 = den1(1,:);
Try_{tf11} = \textsc{zpk}(\text{minreal}(tf(n11, d11)))
\end{verbatim}

\begin{verbatim}
[num2, den2] = ss2tf(Try.a, Try.b, Try.c, Try.d, 2);
n22 = num2(2,:);
d22 = den2(1,:);
Try_{tf22} = \textsc{zpk}(\text{minreal}(tf(n22, d22)))
\end{verbatim}

\begin{verbatim}
Ttw = \textsc{series}(\textsc{series}(Minv, T_{wrwl}), M)
T_{vw\_test} = M * T_{wrwl} * Minv
figure (2000);
bode(Ttw, 'b^+', T_{vw\_test}, 'r-')
\end{verbatim}

\begin{verbatim}
[num1, den1] = ss2tf(Tvw.a, Tvw.b, Tvw.c, Tvw.d, 1);
\end{verbatim}
n11 = numl(1,:);
d11 = denl(1,:);
Tvw_tf11 = zpk(minreal(tf(n11,d11)));
figure(3000);
bode(Tvw_tf11,'b-',Try_tf11,'r-')

% Outer-loops controller for single vehicle
% 2016-07-06
% P control Freq
close all
Kp_vec = [0.5 1.1.5 2];
cell_size = length(Kp_vec);
for ii=1:1:cell_size
    fig101 = figure(101);
    K = tf(Kp_vec(ii));
    S = sisotf_generator(1,P,K);
    bode(S,.Try);
    title('Frequency response of cruise control ...\theta with P control', ...'
    'FontSize', 24)
    hold on; grid on;
    add_legend_in_for_loop(' ','Kp ',Kp_vec,'float ',ii,cell_size);
    h_line = findobj(gcf, 'type', 'line');
    set(h_line, 'LineWidth', 3);
    h_axes = findobj(gcf, 'type', 'axes');
    set(h_axes, 'linewidth', 2);
    set(h_axes, 'FontSize', 15);
    xlabel('Frequency', 'FontSize', 24)
    ylabel('Phase', 'FontSize', 24);
end

% P control \theta_0 = 0.1rad
time = theta_P Resp.time;
y1 = theta_P Resp.signals.values(:,2);
y2 = theta_P Resp.signals.values(:,3);
y3 = theta_P Resp.signals.values(:,4);
y4 = theta_P Resp.signals.values(:,5);
fig103 = figure(103);
plot(time,[y1 y2 y3 y4]);
    h_line = findobj(gcf, 'type', 'line');
    set(h_line, 'LineWidth', 3);
    h_axes = findobj(gcf, 'type', 'axes');
    set(h_axes, 'linewidth', 2);
    set(h_axes, 'FontSize', 15);
    title('Cruise control \theta response using P control', ...'
    'FontSize', 24)
    xlabel('Time(seconds)', 'FontSize', 24)
    ylabel('Angle(rad)', 'FontSize', 24);
    axis([0 20 0.05 0.1])
    hold on; grid on;
    add_legend_in_for_loop(' ','Kp ',Kp_vec,'float ',ii,cell_size);
    legend('Kp=0.5 ', 'Kp=1.0 ', 'Kp=1.5 ', 'Kp=2.0 ');
% PD control Frequency

% Kp = g*z; g = Kd; K = Kd(s+Kp/Kd);
N = 40;
% g_vec = [0.3:0.1:0.6]; z = 2;
% cell_size = length(g_vec);
Kd = 1;
cell_size = length(Kp_vec);
K_pd_cell = cell(cell_size, 1);
for ii = 1:1:cell_size
    fig102 = figure(102);
    Kp = Kp_vec(ii);
    K = zpk(-z,-N,g*N);
    K = pid(Kp, 0, Kd);
    rf = tf(N, [1 N]);
    K_rf = series(K, rf);
    K_pd_cell{ii} = tf(K_rf);
    S = sisotf_generator(1,P,K_rf);
    bode(S,Try);
    title('Frequency response of cruise control ... theta with PD control ...','FontSize', 24)
    hold on; grid on;
    addlegend_in_for_loop('', 'Kp', Kp_vec, 'float', ii, cell_size);
    h_line = findobj(gcf, 'type', 'line');
    set(h_line, 'LineWidth', 3);
    h_axes = findobj(gcf, 'type', 'axes');
    set(h_axes, 'lineweight', 2);
    set(h_axes, 'FontSize', 15);
    xlabel('Frequency', 'FontSize', 24)
    ylabel('Phase', 'FontSize', 24);
end

%% PD control time response

time = theta_PDresp.time;
y1 = theta_PDresp.signals.values(:,2);
y2 = theta_PDresp.signals.values(:,3);
y3 = theta_PDresp.signals.values(:,4);
y4 = theta_PDresp.signals.values(:,5);
fig104 = figure(104);
plot(time,[y1 y2 y3 y4]);
title('Cruise control \theta response using PD control', 'FontSize', 24)
xlabel('Time(seconds)', 'FontSize', 24)
ylabel('Angle(rad)', 'FontSize', 24);
axis([0 20 -0.05 0.1])
hold on; grid on;
legend('Kp=0.5', 'Kp=1.0', 'Kp=1.5', 'Kp=2.0');

save theta_simlink theta*
Outer-loops controller for single vehicle

% separation control

% 2016-07-20

clc
close all

load('basis_for_outer_loop.mat')
speed_lower_limit = -1;

Kp_vec = [0.5 0.6 0.7 1];
cell_size = length(Kp_vec);

for ii = 1:1:cell_size
    K = tf(Kp_vec(ii));
    S = sisotf_generator(1,P,K);
    % theta Frequency Response for Outer-Loop (P Control)
    fig101 = figure(101);
    % bode(S.Try);
    % title('Frequency response of outer loop with P control',...
    % 'FontSize', 24)
    % hold on; grid on;
    % add_legend_in_for_loop('', 'Kp', Kp_vec, 'float', ii, cell_size);
    % h_line = findobj(gcf, 'type', 'line');
    % set(h_line, 'LineWidth', 3);
    % h_axes = findobj(gcf, 'type', 'axes');
    % set(h_axes, 'linewidth', 2);
    % set(h_axes, 'FontSize', 15);
    % xlabel('Frequency', 'FontSize', 24)
    % ylabel('Phase', 'FontSize', 24);
end

% PD control
Kp_vec = [0.5 1.5 2];
Kp_vec = [1 2];
Kp = g*z; % g = Kd; K = Kd(s+Kp/Kd);
N = 40;
Kd = 1; % in thesis now
Kd = [1 1];
cell_size = length(Kp_vec);
K_pd = cell(cell_size, 1);

% Frequency
for ii = 1:1:cell_size
    Kp = Kp_vec(ii);
    K = pid(Kp,0,Kd);
    rf = tf(N,[1 N]);
    K_rf = series(K,rf);
    K_pd{ii} = tf(K_rf);
    S = sisotf_generator(1,P,K_rf);
    fig102 = figure(102);
    bode(S.Try);
    % title('Frequency response of ...
    % outer loop with PD control',...
    % 'FontSize', 24)
% hold on; grid on;
% add_legend_in_for_loop ('', 'Kp', 'Kp_vec', 'float', ii, cell_size);
% h.line = findobj(gcf, 'type', 'line');
% set(h.line, 'LineWidth', 3);
% h.axes = findobj(gcf, 'type', 'axes');
% set(h.axes, 'LineWidth', 2);
% set(h.axes, 'FontSize', 15);
% xlabel('Frequency', 'FontSize', 24)
% ylabel('Phase', 'FontSize', 24);
% end

%%% P control delta_x = 0.2 delta_x0 = 1
%%% P_vec = [0.4:0.2:1]
time = separation.P_resp.time;
y1 = separation.P_resp.signals.values(:, 2);
y2 = separation.P_resp.signals.values(:, 3);
y3 = separation.P_resp.signals.values(:, 4);
y4 = separation.P_resp.signals.values(:, 5);

fig201 = figure(201);
plot(time, [y1 y2 y3 y4]);
set(h.line, 'LineWidth', 3);
set(h.axes, 'FontSize', 15); % h.axes = findobj(gcf, 'type', 'axes');
set(h.axes, 'LineWidth', 2);
set(h.axes, 'FontSize', 15);

hold on; grid on;
% add_legend_in_for_loop('', 'Kp', 'P_vec', 'float', ii, cell_size);
% legend('Kp=0.4', 'Kp=0.6', 'Kp=0.8', 'Kp=0.1');
% legend('Kp=0.5', 'Kp=1.0', 'Kp=1.5', 'Kp=2.0');
% legend('Kp=0.7', 'Kp=1.0', 'Kp=1.2', 'Kp=1.5');
% legend('Kp=0.5', 'Kp=0.6', 'Kp=0.7', 'Kp=1.0');

%%% Tru for P control
time = separation.P_resp.time;
u1 = separation.P_resp_vel.signals.values(:, 1);
u2 = separation.P_resp_vel.signals.values(:, 2);
u3 = separation.P_resp_vel.signals.values(:, 3);
u4 = separation.P_resp_vel.signals.values(:, 4);

fig301 = figure(301);
plot(time, [u1 u2 u3 u4]);
set(h.line, 'LineWidth', 3);
set(h.axes, 'FontSize', 15); % h.axes = findobj(gcf, 'type', 'axes');
set(h.axes, 'LineWidth', 2);
set(h.axes, 'FontSize', 15);

% Tru of P control
set(h.line, 'LineWidth', 3);
set(h.axes, 'FontSize', 15); % h.axes = findobj(gcf, 'type', 'axes');
set(h.axes, 'LineWidth', 2);
set(h.axes, 'FontSize', 15);

hold on; grid on;

axis([0 max(time) -0.1 2.5])
% add_legend_in_for_loop (', 'Kp', P_vec, 'float', ii, cell_size);

% legend (', 'Kp=0.4', 'Kp=0.6', 'Kp=0.8', 'Kp=0.1');
% legend (', 'Kp=0.5', 'Kp=1.0', 'Kp=1.5', 'Kp=2.0');
% legend (', 'Kp=0.5', 'Kp=0.6', 'Kp=0.7', 'Kp=1.0');

%% PD control Simulink separation

time = separation_PD.resp.time;
y1 = separation_PD.resp.signals.values (:, 2);
y2 = separation_PD.resp.signals.values (:, 3);
y3 = separation_PD.resp.signals.values (:, 4);
y4 = separation_PD.resp.signals.values (:, 5);

fig202 = figure (202);
plot (time, [y1 y2 y3 y4]);
h_line = findobj (gcf, 'type', 'line');
set (h_line, 'LineWidth', 3);
h_axes = findobj (gcf, 'type', 'axes');
set (h_axes, 'LineWidth', 2);
set (h_axes, 'FontSize', 15);

title ('Constant separation using PD control', 'FontSize', 24);
xlabel ('Time(seconds)', 'FontSize', 24);
ylabel ('Distance(m)', 'FontSize', 24);
axis ([0 20 0 1.05])
hold on; grid on;
% legend ('Kp=0.4', 'Kp=0.6', 'Kp=0.8', 'Kp=1.0');
% legend ('Kp=0.5', 'Kp=1.0', 'Kp=1.5', 'Kp=2.0');
% legend ('Kp=0.5', 'Kp=0.7', 'Kp=1.5', 'Kp=2.0');

%% Tru for PD control

time = separation_PD.resp.time;
u1 = separation_PD.resp_vel.signals.values (:, 1);
u2 = separation_PD.resp_vel.signals.values (:, 2);
u3 = separation_PD.resp_vel.signals.values (:, 3);
u4 = separation_PD.resp_vel.signals.values (:, 4);

fig302 = figure (302);
plot (time, [u1 u2 u3 u4]);
h_line = findobj (gcf, 'type', 'line');
set (h_line, 'LineWidth', 3);
h_axes = findobj (gcf, 'type', 'axes');
set (h_axes, 'LineWidth', 2);
set (h_axes, 'FontSize', 15);

title ('Tru of PD control with Kd = 1', 'FontSize', 24);
xlabel ('Time(seconds)', 'FontSize', 24);
ylabel ('Speed(m/sec)', 'FontSize', 24);
axis ([0 max(time) -0.1 2.5])
hold on; grid on;
% add_legend_in_for_loop (', 'Kp', P_vec, 'float', ii, cell_size);
% legend ('Kp=0.4', 'Kp=0.6', 'Kp=0.8', 'Kp=0.1');
% legend ('Kp=0.5', 'Kp=1.0', 'Kp=1.5', 'Kp=2.0');
% legend ('Kp=0.5', 'Kp=0.7', 'Kp=1.5', 'Kp=2.0');
close all
load ('basis_for_outer_loop.mat')
P_out = tf(P)
 translation_speed_saturation = [-2 2];
 angular_speed_saturation = translation_speed_saturation./RADIUS;
 % voltage2pwm_gain = 255/BV;
 % voltage2pwm_gain = 1;% uncomment translation cancel this time
 % Initial Conditions !!!!! DON'T FORGET
 vehicle_num = 7;% include leader
 follower_num = vehicle_num - 1;
 vel_ic_leader = 0;
 % follower initial conditions
 acc_ic = zeros(1,follower_num);
 assigned_space = 0.4;
 pos_ic = -assigned_space.*(1:1:follower_num)
 vehicle_id = 1:1:follower_num;

N = 100;
 rf = tf(N,[1 N])

%Nominal Model Variations
% Non identical model
% P_in =
% 13.06
% ______
% s + 1.159

dp_nominal = P_in.den(1)(2);
dc_nominal = P_in.num(1)(2)./dp_nominal;%11.36;

% dc_arr = [12 12 10 12 10 12];
% dp_arr = dp_nominal.*[0.7 2 1.2 2 1 2]
% dc_arr = dc_nominal*[0.9 1.1 .9 1 1.2 .1];
% dp_arr = dp_nominal*[1.2 1.5 1.2 .9 1.2 1];
% dc_arr = dc_nominal*[1 1 1 1 1 1];
% dp_arr = dp_nominal.*[1 1 1 1 1 1];

% S = load ('model_uncertain.mat');
% dc_arr = [S.dc_arr(3),S.dc_arr(4),S.dc_arr(5),...]
% S.dc_arr(7),S.dc_arr(8),S.dc_arr(9)]
% dp_arr = [S.dp_arr(3),S.dp_arr(4),S.dp_arr(5),...]
% S.dp_arr(7),S.dp_arr(8),S.dp_arr(9)]
% %
% for ii=1:1:6
% n_arr(ii) = dp_arr(ii)*dc_arr(ii);
% end

P_in1 = tf([n_arr(1)],1 dp_arr(1));
P_in2 = tf([n_arr(2)],1 dp_arr(2));
P_in3 = tf([n_arr(3)],1 dp_arr(3));
P_in4 = tf([n_arr(4)],1 dp_arr(4));
P_in5 = tf([n_arr(5)],1 dp_arr(5));
P_in6 = tf([n_arr(6)],1 dp_arr(6));

dz_nominal = 40/255*BV;
% dz_arr = dz_nominal*[1 1.5 1.2 1 1 1.4 ]
% dz_arr = dz_nominal*[1 1.5 1.2 0.7 1 2]*0;

291
% PID + PI

K_out = pid(1,0.5,0.3)
K_out = series(K_out,rf);
K_out = series(K_out,rf)
zpk(K_out)

% PI feedforward path wrt speed
z_ff = 3;
g_ff = 1.5;
g_ff = 0.4;
K_FF = zpk([-z_ff],[0 -100],[100*g_ff])
K_FF = tf(K_FF)
PI_FF_vec = [g_ff g_ff*z_ff 0]

% Retrieve data from simulink and plot
close all
%
% if you want to upadate simulation result
% Run SIMLINK model before this section
% platoon_control_no_feed_forward_0531
% Figure 10X

disp('Plotting Simulation Result...')
Time_FF = Platoon_vel_FF.time;
Vel_FF = Platoon_vel_FF.signals.values;
Pos_FF = Platoon_pos_FF.signals.values;
Delta_X_FF = Platoon_deltaX_FF.signals.values;

% lineType_cell ={'k--','b','r','g',' ',' ' , 'm'};

% Try separation of platoon
Platoon_Separation = Delta_X_FF + 0.4;
fig203 = figure(203);
plot(Time_FF,Platoon_Separation(:,1),'k--',
     Time_FF,Platoon_Separation(:,2),
     Time_FF,Platoon_Separation(:,3),
     Time_FF,Platoon_Separation(:,4),
     Time_FF,Platoon_Separation(:,5),
     Time_FF,Platoon_Separation(:,6));
title('Simulation result of platoon of 5 vehicle ', 'FontSize', 24)
xlabel('Time(seconds)', 'FontSize', 24);
ylabel('Vehicle Sepration (m)', 'FontSize', 24);
axis([0 30 0.3 0.7])
legend('\Delta_1','\Delta_2','\Delta_3','\Delta_4','\Delta_5')
h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);
h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);

% control
u1 = Platoon_u1(:,2);
```matlab
u2 = Platoon_u2(:,2);
u3 = Platoon_u3(:,2);
u4 = Platoon_u4(:,2);
u5 = Platoon_u5(:,2);
u6 = Platoon_u6(:,2);

fig300 = figure(300);
plot(Time_FF,u1,...
     Time_FF,u2,...
     Time_FF,u3,...
     Time_FF,u4,...
     Time_FF,u5)

h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);

h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 24);

hold on; grid on;

axis([0 30 -0.5 1.5])
title('Control response of platoon of 5 vehicle', 'FontSize', 24);
xlabel('Time (sec)')
ylabel('Voltage (V)')

legend('Follower 1','Follower 2','Follower 3',...
       'Follower 4','Follower 5');
```
APPENDIX C

ARDUINO CODE
// motion control (differential drive)
// Zhichao Li
// 2016-08-14
#include <Wire.h>
#include <math.h>
#include <TimerOne.h>

// Vehicle Setting Macro
#define CONFIGPLATFORM_ETT

// system state can be 0 (stopped) or 1 (running),
// i.e. motor power state
int motor_power_state = 0;

// The controller mode can be either 0 (open_loop)
// or 1 (close_loop)
int ctrl_mode = 0;

// debug flag
int debug_mode = 1;

long timestamps[8];

// function forward declaration

void serial_parse_command(int opcode);
void serial_state_machine_proceed(int c);
void serial_send_header(byte len, byte opcode);

void m_set_pwm(int v_left_new, int v_left_direction,
               int v_right_new, int v_right_direction);
void m_stop();

long encoder_left_read();
long encoder_right_read();
long encoder_left_write();
long encoder_right_write();

void ctrl_reset();
void ctrl_set_open_loop();
void ctrl_set_wl_wr(float wl_dsr_new, float wr_dsr_new);

// hardware abstraction layer

// ETT

#ifdef (CONFIGPLATFORM_ETT)

// Motor
#define MOTOR2WD_ADAFRUIT
#define LEFTMOTOR_INDEX 3
#define RIGHTMOTOR_INDEX 2

// Encoder
#define ENCODER2WD
#define ENC_LEFT_PIN_A 2
#define ENC_LEFT_PIN_B 2
#define ENC_RIGHT_PIN_A 3
#define ENC_RIGHT_PIN_B 3

// Count Per Turn of Encoder
const long ENCODER_CPT_GEARED = 80;

#endif
// IMU
#define CONFIG_IMU_BNO055
#define CONFIG_ULTRASONIC_HC_SR04
#define ULTRASONIC_TRIGGER_PIN 4
#define ULTRASONIC_ECHO_PIN 5
#define ULTRASONIC_MAX_DISTANCE 300

// Control
int PWM_MAX = 255;
int PWM_MIN = -20;
int PWM_D2A_FACTOR = 49;
int CTRL_LOOP_PERIOD = 100;
#endif // defined (CONFIG_PLATFORM_ETT)

// ----------- Device code -----------
// --------- Motor ---------
#if defined (CONFIG_MOTOR_2WD_ADAFRUIT)
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_PWMServoDriver.h"

Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *m_left = AFMS.getMotor(M_LEFT_MOTOR_INDEX);
Adafruit_DCMotor *m_right = AFMS.getMotor(M_RIGHT_MOTOR_INDEX);

void m_set_pwm(int v_left_new, int v_left_direction,
int v_right_new, int v_right_direction)
{
if (v_left_direction == 1 || v_left_direction == '+') {
m_left->setSpeed(v_left_new);
m_left->run(FORWARD);
} else if (v_left_direction == -1 || v_left_direction == 2
|| v_left_direction == '-') {
m_left->setSpeed(v_left_new);
m_left->run(BACKWARD);
} else {
m_left->run(RELEASE);
}

if (v_right_direction == 1 || v_right_direction == '+') {
m_right->setSpeed(v_right_new);
m_right->run(FORWARD);
} else if (v_right_direction == -1 || v_right_direction == 2
|| v_right_direction == '-') {
m_right->setSpeed(v_right_new);
m_right->run(BACKWARD);
} else {
m_right->run(RELEASE);
}

void m_stop() {
// LEE Release may not equal to setSpeed(0)
m_left->run(RELEASE);
m_right->run(RELEASE);
}
void m_setup() {
    AFMS.begin();
}

#define // defined (CONFIG_MOTOR_2WD_ADAFRUIT)

// ------------ Encoder ------------
#if defined (CONFIG_ENCODER_2WD)

#include <Encoder.h>

Encoder EncL(ENC_LEFT_PIN_A, ENC_LEFT_PIN_B);
Encoder EncR(ENC_RIGHT_PIN_A, ENC_RIGHT_PIN_B);

long encoder_left_read() {
    return EncL.read();
}

long encoder_right_read() {
    return EncR.read();
}

void encoder_left_write(long val) {
    EncL.write(val);
}

void encoder_right_write(long val) {
    EncR.write(val);
}

void encoder_reset() {
    EncL.write(0);
    EncR.write(0);
}

float encoder_calculate_angular_speed(long delta_tick, long delta_time) {
    return 1.0 * (delta_tick) / (delta_time / 1000.0) * 2 * PI / ENCODER_CPT_GEARED;
}

#endif // defined (CONFIG_ENCODER_2WD)

// ---------- IMU ----------
#if defined (CONFIG_IMU_BNO055)

#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>

Adafruit_BNO055 imu_bno055 = Adafruit_BNO055();
imu::Vector<3> euler_init;
imu::Vector<3> euler;
imu::Vector<3> acc;
imu::Vector<3> gyro;

#endif // defined (CONFIG_IMU_BNO055)
void imu_setup() {
  imu_bno055.begin();
  delay(1000);
  euler_init = imu_bno055.getVector(Adafruit_BNO055::VECTOR_EULER);
}

void imu_read() {
  euler = imu_bno055.getVector(Adafruit_BNO055::VECTOR_EULER);
  acc = imu_bno055.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
  gyro = imu_bno055.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
}

float imu_get_theta() {
  return -(euler.x() - euler_init.x()) * 3.14 / 180.0;
}

float imu_get_accy() {
  return -acc.y();
}

float imu_get_omega() {
  return gyro.z();
}

#endif

// 。。。。 Ultrasonic sensor 。。。。
#if defined (CONFIG_ULTRASONIC_HC_SR04)
#include <NewPing.h>

NewPing sonar(ULTRASONIC_TRIGGER_PIN,
              ULTRASONIC/ECHO_PIN, ULTRASONIC_MAX_DISTANCE);

// LEE 0702 to emulate unexpected abnormal measurement
// We backup last measurement if current us is wrong,
// replace us = us_safety_p
int us_safety_p = 0;

float ultrasonic_get_delta_x() {
  // get delta x from ultrasonic sensor
  int us = sonar.ping();
  float distance;

  if (us == 0) {
    // distance = 5;
    us = us_safety_p;
  } else {
    // CAUTION: distance is in meter, not centimeter
    distance = us / 57.0 / 100.0;
  }

  if (distance >= ULTRASONIC_MAX_DISTANCE){
    return (float)ULTRASONIC_MAX_DISTANCE;
  }
}
    return distance;
  }
#endif

// __________ controller parameter __________
// CAUTION: ctrl_loop_period is also accessed in interrupt
// CAUTION: ctrl_loop_period can not be
// changed at running time
// loop period is in ms
volatile long ctrl_loop_period = CTRL_LOOP_PERIOD;

// PID controller parameter
float prefilter_co = 0.167;
float kp_left = 0.29;
float ki_left = 0.58;
float kd_left = 0;
float kp_right = 0.29;
float ki_right = 0.58;
float kd_right = 0;
float roll_off_co = 0.8; // 40/(s+40)

// __________ control global variables __________
// robot state
float imu_theta = 0;
float imu_accy = 0;
float imu_omega = 0;
float delta_x = 0;
float wr = 0;
float wr_p = 0;
// Desired angular velocity
float wl_ds = 0;
float wr_ds = 0;
float wl_ds_filtered = 0;
float wr_ds_filtered = 0;
float wl_ds_filtered_p = 0;
float wr_ds_filtered_p = 0;
float err_wl = 0;
float err_wr = 0;
float err_wl_p = 0;
float err_wr_p = 0;
float err_wl_pp = 0;
float err_wr_pp = 0;
// controller output PWM
int pwml = 0;
int pwxr = 0;
int pwml_out = 0;
int pwmr_out = 0;
int pwnl_out_p = 0;
int pwnr_out_p = 0;
float pwnl_up;
float pwnl_ui;
float pwnl_ud;
float pwnr_up;
float pwnr_ui;
float pwnr_ud;

// control utility function
void ctrl_reset() {
    wl_dsr_filtered = 0;
    wr_dsr_filtered = 0;
    wl_dsr_filtered_p = 0;
    wr_dsr_filtered_p = 0;
    err_wl_p = 0;
    err_wr_p = 0;
    err_wl_pp = 0;
    err_wr_pp = 0;
    pwnl_out_p = 0;
    pwnr_out_p = 0;
}

// Determine direction and control saturation
void ctrl_set_pwm() {
    int pwnl_ctrl = pwnl;
    int pwnr_ctrl = pwnr;
    int pwnlMotor;
    int right_dir = 0;
    int left_dir = 0;

    pwnlMotor = pwnl_ctrl;
    if (pwnl_ctrl > PWM_MAX) {
        pwnlMotor = PWM_MAX;
    }
    if (pwnl_ctrl < PWM_MIN) {
        pwnlMotor = PWM_MIN;
    }
    pwnrMotor = pwnr_ctrl;
    if (pwnr_ctrl > PWM_MAX) {
        pwnrMotor = PWM_MAX;
    }
    if (pwnr_ctrl < PWM_MIN) {
        pwnrMotor = PWM_MIN;
    }

    if (pwnlMotor > 0) {
        left_dir = '+';
    } else if (pwnlMotor < 0) {
        pwnlMotor = -pwnlMotor;
        left_dir = '-';
    } else {
        // note that set pwm as 0 is not release motor
        left_dir = '+';
    }
}
if (pwmr_motor > 0) {
    right_dir = '+';
} else if (pwmr_motor < 0) {
    pwmr_motor = -pwmr_motor;
    right_dir = '-';
} else {
    right_dir = '+';
}

// call lower level interface
m_set_pwm(pwml_motor, left_dir, pwmr_motor, right_dir);

void ctrl_get_theta_accx_omega() {
    imu_read();
    imu_theta = imu_get_theta();
    imu_accy = imu_get_accy();
    imu_omega = imu_get_omega();
}

void ctrl_get_delta_x() {
    delta_x = ultrasonic_get_delta_x();
}

void ctrl_get_current_wl_wr() {
    long left = encoder_left_read();
    long right = encoder_right_read();
    wl = encoder_calculate_angular_speed(left, ctrl_loop_period);
    wr = encoder_calculate_angular_speed(right, ctrl_loop_period);

    // average filter
    wl = (wl + wl_p) / 2;
    wr = (wr + wr_p) / 2;
    wl_p = wl;
    wr_p = wr;

    // so that every time we get increment from last time
    encoder_left_write(0);
    encoder_right_write(0);
}

// PID controller
float ctrl_pid(float err, float err_sum, float err_p,
    float kp, float ki, float kd,
    float *up_out, float *ui_out, float *ud_out,
    float ts) {
    float u = 0;
    float up = kp * err;
    float ui = ki * ts * err_sum;
    float ud = kd * (err - err_p) / ts;
    u = up + ui + ud;
    if (*up_out != NULL)
        *up_out = up;
if (ui_out != NULL) *ui_out = ui;
if (ud_out != NULL) *ud_out = ud;
return u;
}

// PID controller Incremental Style
int ctrl_pid_inc(int u_p, float err_p, float err_pp, float kp, float ki, float kd, float *up_out, float *ui_out, float *ud_out, float ts)
{
    float up = kp * (err - err_p);
    float ui = ki * ts * err;
    float ud = kd * ((err - err_p) - (err_p - err_pp)) / ts;
    float delta_u = (up + ui + ud) * PWM_D2A_FACTOR;
    int u = u_p + (int)delta_u;
    if (up_out != NULL) *up_out = up;
    if (ui_out != NULL) *ui_out = ui;
    if (ud_out != NULL) *ud_out = ud;
    return (int)u;
}

// control inner loop, i.e. (wL, wR) control

// If error is smaller than the threshold, then set error to 0
float ctrl_deadzone_threshold(float err_in) {
    float err_out = err_in;
    if (abs(err_in) < deadzone_threshold) {
        err_out = 0;
    }
    return err_out;
}

// Saturate the control variable change, i.e. we do not allow rapid change of PWM between interactions.
int ctrl_deadzone_saturation(int u_in, int u_in_p) {
    int u_out = u_in;
    if (abs(u_in - u_in_p) > deadzone_saturation) {
        u_out = (int)(u_in + deadzone_saturation);
    }
    return u_out;
}

// roll off for controller output (plant (motor shield) input)
// u_rf = (1 - rolloff_coeff) u_rf_p + rolloff_coeff * u_in
float ctrl_output_rolloff(int u_rf_p, int u_in)
{
    float u_rf = (1 - roll_off_coeff) * u_rf_p + roll_off_coeff * u_in;
    return u_rf;
}

// Update desired angular velocity and use PID controller
// Thus control inner loop (angular velocity)
void ctrl_inner_loop() {
    if (ctrl_mode == 0) {
        // controller mode is open loop
        ctrl_set_pwm();
        return;
    }

    // restriction desire speed to positive
    // to prevent unpredictable error
    int stop_action_level = 0;
    if (wl_dsr <= 0.02 || wr_dsr <= 0.02) {
        if(wl_dsr<0 || wr_dsr<0 ){
            stop_action_level = 2;
        }else{
            stop_action_level = 1;
        }
    }
    w_l_ds_r = 0;
    w_r_ds_r = 0;
} else {
    stop_action_level = 0;
}

    w_l_ds_r_f i l t e r e d = (1 - pre f i l t e r_c o ) * w_l_d s r_f i l t e r e d_p + pre f i l t e r_c o * w_l_d s r;
    w_r_ds_r_f i l t e r e d = (1 - pre f i l t e r_c o ) * w_r_d s r_f i l t e r e d_p + pre f i l t e r_c o * w_r_d s r;

    // calculate error between measured output
    // and desire value
    err_w_l = w_l_d s r_f i l t e r e d - w_l;
    err_w_r = w_r_d s r_f i l t e r e d - w_r;

    // PID Inner loop Controller
    pwml_out = ctrl_pid_inc(pwml_out_p, err_w_l, err_w_l_p, err_w_l_pp, kp_left, ki_left, kd_left, &pwml_up, &pwml_ui, &pwml_ud, (float)ctrl_loop_period / 1000.0);
    pwmr_out = ctrl_pid_inc(pwmr_out_p, err_w_r, err_w_r_p, err_w_r_pp, kp_right, ki_right, kd_right, &pwmr_up, &pwmr_ui, &pwmr_ud, (float)ctrl_loop_period / 1000.0);

    pwml = (int) ctrl_output_rolloff (pwml_out_p, pwml_out);
    pwmr = (int) ctrl_output_rolloff (pwmr_out_p, pwmr_out);

    if(stop_action_level >0 ){
        if(stop_action_level<= 1){
            //release motor
            pwmr = 0;
            pwml = 0;
        } else {
            // take brake action
            pwmr = -5;
            pwml = -5;
        }
    }
}
void ctrl_inner_loop_update() {
    // Iteration
    pwml_out_p = pwml_out;
    pwmr_out_p = pwmr_out;
    err_wl_p = err_wl;
    err_wr_p = err_wr;
    err_wl_pp = err_wl_p;
    err_wr_pp = err_wr_p;
    wl_dsr_filtered_p = wl_dsr_filtered;
    wr_dsr_filtered_p = wr_dsr_filtered;
}

// ------------ control interface -----------
void ctrl_set_open_loop(int pwml_new, int pwmr) {
    pwml = pwml_new;
    pwmr = pwmr_new;
    ctrl_mode = 0;
}

void ctrl_set_wl_wr(float wl_dsr_new, float wr_dsr_new) {
    wl_dsr = wl_dsr_new;
    wr_dsr = wr_dsr_new;
    ctrl_mode = 1;
}

// ------------ serial protocol ------------
#define SERIAL_STATE_INIT 0
#define SERIAL_STATE_MAGIC1 1
#define SERIAL_STATE_MAGIC2 2
#define SERIAL_STATE_PROTO 3
#define SERIAL_MAGIC_1 'A'
#define SERIAL_MAGIC_2 'F'

// command sent from HLC to LLC
#define OPCODE_OPEN_LOOP 0x00
#define OPCODE_CTRL_WL_WR 0x10
#define OPCODE_PAN_TILT 0x20
#define OPCODE_SETUP 0x40
#define OPCODE_START 0xF1
#define OPCODE_STOP 0xF2
#define OPCODE_DEBUG_ENABLE 0xF3
# define OPCODE_DEBUG_DISABLE 0xF4

// state report from LLC to HLC

// define OPCODE_VEHICLE_STATUS 0xE0

// define OPCODE_CTRL_STATUS_DEBUG 0xE1

// ------------ serial utility function -------

// CAUTION: this is blocking write!!!

void serial_put_char(byte c) {
  while (Serial.write(c) <= 0)
  {
    
  }

void serial_send_header(byte len, byte opcode) {
  serial_put_char(SERIAL_MAGIC_1);
  serial_put_char(SERIAL_MAGIC_2);
  serial_put_char(len);
  serial_put_char(opcode);
  Serial.flush();
}

void serial_send_int(int i) {
  byte *cp = (byte *) &i;
  byte c0 = cp[0];
  byte c1 = cp[1];
  serial_put_char(c0);
  serial_put_char(c1);
  Serial.flush();
}

void serial_send_long(long l) {
  byte *cp = (byte *) &l;
  byte c0 = cp[0];
  byte c1 = cp[1];
  byte c2 = cp[2];
  byte c3 = cp[3];
  serial_put_char(c0);
  serial_put_char(c1);
  serial_put_char(c2);
  serial_put_char(c3);
  Serial.flush();
}

void serial_send_float(float f) {
  byte *cp = (byte *) &f;
  byte c0 = cp[0];
byte c1 = cp[1];
byte c2 = cp[2];
byte c3 = cp[3];
serial_putchar(c0);
serial_putchar(c1);
serial_putchar(c2);
serial_putchar(c3);
Serial.flush();

// CAUTION: this is blocking read!!!
int serial_get_char() {
  while (Serial.available() <= 0);
  return (char) Serial.read();
}
int serial_get_int() {
  char bytearray[2];
  bytearray[0] = serial_get_char();
  bytearray[1] = serial_get_char();
  return *((int *)bytearray);
}
long serial_get_long() {
  char bytearray[4];
  bytearray[0] = serial_get_char();
  bytearray[1] = serial_get_char();
  bytearray[2] = serial_get_char();
  bytearray[3] = serial_get_char();
  return *((long *)bytearray);
}
float serial_get_float() {
  char bytearray[4];
  bytearray[0] = serial_get_char();
  bytearray[1] = serial_get_char();
  bytearray[2] = serial_get_char();
  bytearray[3] = serial_get_char();
  return *((float *)bytearray);
}

int serial_state = SERIAL_STATE_INIT;
int serial_length = 0;

// --- dispatch serial message according to the opcode ---
void serial_parse_command(int opcode) {
  switch (opcode) {
  case OPCODE_OPEN_LOOP:
// If direction variable is + ,
// then it means rotating forward;
// If direction variable is - ,
// then it means rotating backward;
// All other direction value is stop
int pwm_left_dsr = serial_get_int();
int pwm_right_dsr = serial_get_int();
if (motor_power_state == 0 ||
    motor_power_state == 1 && ctrl_mode == 0)
    ctrl_set_open_loop(pwm_left_dsr, pwm_right_dsr);
    break;

    }

    case OPCODE_CTRL_WL_WR: {
        float wl_dsr_new = serial_get_float();
        float wr_dsr_new = serial_get_float();
        if (motor_power_state == 0 ||
            motor_power_state == 1 && ctrl_mode == 1)
            ctrl_set_wl_wr(wl_dsr_new, wr_dsr_new);
            break;
    }

    case OPCODE_SET: {
        // Setup only works if the system is stopped.
        if (motor_power_state == 0) {
            // The order must defines exactly
            // the same as h21 setup function
            prefilter_co = serial_get_float();
            roll_off_co = serial_get_float();
            kp_left = serial_get_float();
            ki_left = serial_get_float();
            kd_left = serial_get_float();
            kp_right = serial_get_float();
            ki_right = serial_get_float();
            kd_right = serial_get_float();
            deadzone_threshold = serial_get_float();
            deadzone_saturation = serial_get_float();
        }
        break;
    }

    case OPCODE_START: {
        motor_power_state = 1;
        break;
    }

    case OPCODE_STOP: {
        motor_power_state = 0;
        m_stop();
        ctrl_reset();
        break;
    }

    case OPCODE_DEBUG_ENABLE: {
debug_mode = 1;
break;
}
case OPCODE_DEBUG_DISABLE: {
    debug_mode = 0;
    break;
}

// case OPCODE Xxx: {
//     // add new function here
//     // use these comments as template
//     break;
// }
default: {
    break;
}
}

// serial state machine
void serial_state_machine_proceed(int c) {
    Serial.print(c, HEX);
    Serial.print(' ');

    switch (serial_state) {
    case SERIAL_STATE_INIT: {
        if (c == SERIAL_MAGIC_1)
            serial_state = SERIAL_STATE_MAGIC1;
        else
            serial_state = SERIAL_STATE_INIT;
        break;
    }
    case SERIAL_STATE_MAGIC1: {
        if (c == SERIAL_MAGIC_2)
            serial_state = SERIAL_STATE_MAGIC2;
        else
            serial_state = SERIAL_STATE_INIT;
        break;
    }
    case SERIAL_STATE_MAGIC2: {
        serial_length = c;
        serial_state = SERIAL_STATE_PROTO;
        break;
    }
    case SERIAL_STATE_PROTO: {
        // opcode = c
        serial_parse_command(c);
        serial_state = SERIAL_STATE_INIT;
        break;
    }
    default: {
        serial_state = SERIAL_STATE_INIT;
        break;
    }
    }
}
void serial_send_vehicle_status(long timestamp) {
    serial_send_header(28, OPCODE_VEHICLE_STATUS);
    // the first argument of send header is the total sum of payload below
    serial_send_long(timestamp);
    serial_send_float(imu_theta);
    serial_send_float(imu_accy);
    serial_send_float(imu_omega);
    serial_send_float(wl);
    serial_send_float(wr);
    serial_send_float(delta_x);
}

void serial_send_ctrl_status_debug(long timestamp) {
    int i;
    serial_send_header(112, OPCODE_CTRL_STATUS_DEBUG);
    serial_send_long(timestamp);
    serial_send_float(wl_dsr);
    serial_send_float(wr_dsr);
    serial_send_float(wl_dsr_filtered);
    serial_send_float(wr_dsr_filtered);
    serial_send_int(pwml);
    serial_send_int(pwmr);
    serial_send_int(pwml_out);
    serial_send_int(pwmr_out);
    serial_send_int(pwml_out_p);
    serial_send_int(pwmr_out_p);
    serial_send_float(err_wl);
    serial_send_float(err_wl_p);
    serial_send_float(err_wl_pp);
    serial_send_float(err_wr);
    serial_send_float(err_wr_p);
    serial_send_float(err_wr_pp);
    serial_send_float(pwml_up);
    serial_send_float(pwml_ui);
    serial_send_float(pwml_ud);
    serial_send_float(pwmr_up);
    serial_send_float(pwmr_ui);
    serial_send_float(pwmr_ud);
    for (i = 0; i < 8; i++) {
        serial_send_long(timestamps[i]);
    }
}

// interrupt

// ------------ serial sending ------------
// Timer interrupt counter.
// It is reset when new loop period comes.
int timer_counter = 0;

// Timer flag for the controller loop
volatile int timer_inner_loop_flag = 0;

// Update Encoder Register in interrupt
// when interrupt happens set Flag_TimerUpdate
// Remember to reset Encoder if Encoder CPT is high
// as counter in one sampling period
// DistMax=Longmax/Enc_CPT_MotorShaft/GearboxRatio
// *(2*pi*radius)

// CAUTION: This function is called in interrupt!!!
// Must make it short!!!
void timer_update() {
    timer_counter++;
    if (timer_counter >= ctrl_loop_period) {
        timer_inner_loop_flag = 1;
        timer_counter = 0;
    }
}

// ______________ setup ______________

void setup() {
    Serial.begin(115200); // Serial Setup
    Timer1.initialize(1000); // NOTE: period is in us
    // LEE Note may change to 10ms for ETT
    Timer1.attachInterrupt(timer_update);

    // motor setup
    m_setup();

    imu_setup();
    // in case of unexpected start state, reset everything
    m_stop();
    encoder_reset();
    ctrl_reset();
}

// ______________ loop ______________

unsigned long loop_count = 0;

void loop() {
    // Run Serial State Machine
    // CAUTION: Never block or delay or spend too much time
    if (Serial.available() > 0) {
        int c = Serial.read();
        serial_state_machine.proceed(c);
    }
    if (timer_inner_loop_flag > 0) {

loop_count++;  
unsigned long timestamp = millis();
timestamps[0] = millis() - timestamp;

// sensor data gathering
ctrl_get_current_wl_wr(); // less than 1 ms
timestamps[1] = millis() - timestamp;
ctrl_get_theta_accx_omega(); // roughly 5 - 6 ms
// (I2C bus needs time)
timestamps[2] = millis() - timestamp;
lw

ctrl_get_delta_x();
timestamps[3] = millis() - timestamp;
serial_send_vehicle_status(timestamp); // at most 1 ms
timestamps[4] = millis() - timestamp;

if (motor_power_state > 0) {
    // Run controller.
    ctrl-inner_loop(); // less than 1 ms
timestamps[5] = millis() - timestamp;
    // Set controller output to motor.
    ctrl_set_pwm();
    // roughly 3 - 4 ms on Adafruit motor shield,
timestamps[6] = millis() - timestamp;
    serial_send_ctrl_status_debug(timestamp);
    // roughly 2 - 3 ms
    ctrl-inner_loop_update();
timestamps[7] = millis() - timestamp;
}

timer-inner-loop_flag = 0;