

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. Artimeiadis
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

	Page
LIST OF TABLES	viii
LIST OF FIGURES	ix
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 AND C^4S RE- QUIREMENTS	25
2.1 Introduction and Overview	25
2.2 FAME Architecture and C^4S Requirements	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

CHAPTER	Page
3.6 Uncertainty of Parameter	76
3.6.1 Frequency Response Trade Studies	76
3.6.2 Time Response Trade Studies	79
3.7 Differential-Drive Robot Model with Dynamics on Ground	83
3.7.1 Empirically Obtained Experimental Data for Ground Model.	86
3.7.2 Fitting Model to Collected Data.....	87
3.7.3 On Ground Nominal Model.	90
3.8 Summary and Conclusion	92
4 SINGLE VEHICLE CASE STUDY FOR A LOW-COST MULTI-CAPABILITY DIFFERENTIAL-DRIVE ROBOT: ENHANCED THUNDER TUMBLE (ETT)	93
4.1 Introduction and Overview	93
4.2 Inner-Loop Speed Control Design and Implementation.....	93
4.2.1 Frequency Domain (g,z) Trade Studies	105
4.2.2 Time Domain (g, z) Trade Studies.....	118
4.2.3 Inner-Loop Experimental Result	123
4.3 Outer-Loop Control Design and Implementation	126
4.3.1 Outer-Loop 1: (v, θ) Cruise Control Along Line - Design and Implementation	126
4.3.2 Outer-Loop 2: Separation-Direction $(\Delta x, \theta)$ Control - Design and Implementation	129
4.4 Summary and Conclusion	150
5 LONGITUDINAL CONTROL OF A PLATOON OF VEHICLES	151
5.1 Introduction and Overview	151

CHAPTER	Page
5.2 Platoon Configuration	152
5.3 Modeling for Longitudinal Platoon of Vehicles	153
5.4 Control for Longitudinal Platoon of Vehicles	154
5.5 Longitudinal Platoon Separation Controller Tradeoff Study	156
5.6 Experimental Results for Controlled Platoon of Vehicles	172
5.7 Platoon Simulation with model Uncertainty and Stiction Deadzone .	179
5.8 Summary and Conclusions	181
6 SUMMARY AND FUTURE DIRECTIONS	182
6.1 Summary of Work	182
6.2 Directions for Future Research	184
REFERENCES	186
 APPENDIX	
A C CODE.....	192
B MATLAB CODE	269
C ARDUINO CODE	299

LIST OF TABLES

Table	Page
3.1 Hardware Components for Enhanced Differential-Drive Thunder Tumbler Robotic Vehicle.....	32
3.2 Thunder Tumbler Nominal Parameter Values with Uncertainty	37
3.3 Thunder Tumbler Gearbox Symbols	47
5.1 Platoon Design Parameter	172

LIST OF FIGURES

Figure	Page
1.1 Visualization of Fully-Loaded (Enhanced) Thunder Tumbler	12
1.2 Optical Wheel Encoders - RPR220 photo-interrupter Sensors on Left, code disk on Right	13
1.3 Adafruit BNO055 9DOF Inertial Measurement Unit (IMU)	13
1.4 Arduino Uno Open-Source Microcontroller Development Board	14
1.5 Adafruit Motor Shield for Arduino v2.3 - Provides PWM Signal to DC Motors	14
1.6 Raspberry Pi 3 Model B Open-Source Single Board Computer	15
1.7 Raspberry Pi 5MP Camera Module	15
1.8 TP-LINK Wireless High Gain USB Adapter	16
1.9 HC-SR04 Ultrasonic Sensor	16
1.10 Visualization of Inner- and Outer-Loop Control Laws	19
2.1 <i>FAME</i> Architecture to Accommodate Fleet of Cooperating Vehicles ...	26
3.1 Visualization of Differential-Drive Mobile Robot	38
3.2 Visualization of DC Motor Speed-Voltage Dynamics	42
3.3 dc Motor Armature Inductance of 10 ETT Motors	45
3.4 dc Motor Armature Inductance of 10 ETT Motors	46
3.5 Disassembled Gearbox in ETT	48
3.6 DC Motor back EMF Constant of Left and Right ETT Motors	49
3.7 DC Motor Output ω_s Response to 1.02V Step Input - Hardware	50
3.8 DC Gain Distribution of Step Response at Different Voltage Step Input	51
3.9 DC Gain Distribution of Step Response at Different Voltage Step Input	52
3.10 Equivalent Moment of Inertia Distribution of Left and Right Motors ...	53

Figure	Page
3.11 Equivalent Speed Damping Constant Distribution of Left and Right Motors	53
3.12 Motor Output ω_s Response to 1.02V Step Input - Hardware and Decoupled Model	55
3.13 Motor Output ω_s Response to 2.04V Step Input - Hardware and Decoupled Model	55
3.14 Motor Output ω_s Response to 3.06V Step Input - Hardware and Decoupled Model	56
3.15 Motor Output ω_s Response to 4.08V Step Input - Hardware and Decoupled Model	56
3.16 Motor Output ω_s Response to 5.10V Step Input - Hardware and Decoupled Model	57
3.17 DC Motor Output ω_s Response to High Voltage Step Input - Hardware and Decoupled Model	58
3.18 TITO LTI Robot-Motor Wheel Speed (ω_R, ω_L) Dynamics - $P_{(\omega_R, \omega_L)}$	59
3.19 Differential-Drive Mobile Robot Dynamics	60
3.20 Robot Singular Values (Voltages to Wheel Speeds) - Including Low Frequency Approximation	64
3.21 Robot Frequency Response (Voltages to Wheel Speeds) - Including Low Frequency Approximation	65
3.22 Robot Plant Singular Values (Voltages to v and ω) - Including Low Frequency Approximation	68
3.23 Robot Plant Frequency Response (Voltages to $[v, \omega]$) - Including Low Frequency Approximation	69

Figure	Page
3.24 Frequency Response for Vehicle-Motor - Coupled (ω_R, ω_L) Model	70
3.25 Vehicle-Motor Response to Unit Step Input - Coupled (ω_R, ω_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	85
3.37 Off Ground and On Ground Step Response to 50 PWM Command (1V)	85
3.38 Output ω_s Response to PWM 50 Voltage Step Input - Hardware and Decoupled Model	87
3.39 On Ground Motor Fitting Left Motor - Hardware and Decoupled Model	88

Figure	Page
3.40 On Ground Motor Fitting Right Motor - Hardware and Decoupled Model	88
3.41 On Ground Approximate Transfer Function Fitting Model Distribution of V1	89
3.42 On Ground Nominal Model	90
3.43 Step Response of Nominal Model with other ETT Model	91
3.44 On Ground Nominal Model with other ETT Model	91
4.1 Visualization of (v, ω) and (ω_R, ω_L) Inner-Loop Control	96
4.2 PK and $L_o = MPKM^{-1}$ Singular Values	97
4.3 $S_o = (I + L_o)^{-1} = S_i$ Singular Values - Using Decoupled Model	99
4.4 $T_o = L_o(I + L_o)^{-1} = T_i$ Singular Values - Using Decoupled Model	99
4.5 T_{ru} Singular Values (No Pre-filter) - Using Decoupled Model	100
4.6 $T_{ru}W$ Singular Values (with Pre-filter) - Using Decoupled Model	100
4.7 KSM^{-1} Singular Values	101
4.8 T_{diy} Singular Values - Using Decoupled Model	101
4.9 MSP Singular Values	102
4.10 Inner-Loop $[\omega_R, \omega_L]$ Filtered Step Response - Using Decoupled Model ..	102
4.11 Inner-Loop $[\omega_R, \omega_L]$ Unfiltered Step Response - Using Decoupled Model	103
4.12 Control Response to Step Command (with Pre-filter) - with Decoupled Model	104
4.13 Control Response to Unfiltered Step Command - with Decoupled Model	104
4.14 Singular Values for L ($g=0.10, 0.30, 0.50, 0.70; z=2$)	105
4.15 Singular Values for L ($g=0.50; z=1, 2, 3, 4$)	106
4.16 Singular Values for S ($g=0.10, 0.30, 0.50, 0.70; z=2$)	107

Figure	Page
4.17 Singular Values for S ($g=0.50; z=1, 2, 3, 4$)	107
4.18 Singular Values for T ($g=0.10, 0.30, 0.50, 0.70; z=2$)	108
4.19 Singular Values for T ($g=0.50; z=1, 2, 3, 4$)	109
4.20 Singular Values for T_{ru} ($g=0.10, 0.30, 0.50, 0.70; z=2$) - (ω_R, ω_L) Commands	110
4.21 Singular Values for T_{ru} ($g=0.50; z=1, 2, 3, 4$) - (ω_R, ω_L) Commands	110
4.22 Singular Values for KSM^{-1} ($g=0.10, 0.30, 0.50, 0.70; z=2$) - (v, ω) Commands	111
4.23 Singular Values for KSM^{-1} ($g=0.50; z=1, 2, 3, 4$) - (v, ω) Commands	112
4.24 Singular Values for $W \cdot T_{ru}$ ($g=0.10, 0.30, 0.50, 0.70; z=2$) - (ω_R, ω_L) Commands	113
4.25 Singular Values for $W \cdot T_{ru}$ ($g=0.50; z=1, 2, 3, 4$) - (ω_R, ω_L) Commands	113
4.26 Singular Values for T_{diy} ($g=0.10, 0.30, 0.50, 0.70; z=2$) - (ω_R, ω_L) Responses	115
4.27 Singular Values for T_{diy} ($g=0.50; z=1, 2, 3, 4$) - (ω_R, ω_L) Responses	115
4.28 Singular Values for MSP ($g=0.50; z=1, 2, 3, 4$) - (v, ω) Responses	116
4.29 Singular Values for MSP ($g=0.50; z=1, 2, 3, 4$) - (v, ω) Responses	117
4.30 Output Response to Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)	118
4.31 Output Response to Step Command ($g = 0.50; z = 1, 2, 3, 4$)	119
4.32 Control Response to Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)	119
4.33 Control Response to Step Command ($g = 0.50; z = 1, 2, 3, 4$)	120
4.34 Output Response to Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)	121
4.35 Output Response to Step Command ($g = 0.50; z = 1, 2, 3, 4$)	121

Figure	Page
4.36 Control Response to Filtered Step Command $g = 0.10, 0.30, 0.50, 0.70; z = 2)$	122
4.37 Control Response to Filtered Step Command ($g = 0.50; z = 1, 2, 3, 4$)	122
4.38 Output Response to filtered Step Command ($\omega R_{ref} = 10, \omega L_{ref} = 10$)	123
4.39 Output Response to filtered Step Command ($v_{ref} = 0.5, \omega_{ref} = 0$)	124
4.40 Control Response to filtered Step Command ($\omega R_{ref} = 10, \omega L_{ref} = 10$)	124
4.41 Visualization of Cruise Control Along a Line	126
4.42 $T_{\theta_{ref}\theta}$ Frequency Response for θ Outer-Loop (P Control)	127
4.43 $T_{\theta_{ref}\theta}$ Frequency Response for θ Outer-Loop (PD control, $K_d = 1$)	127
4.44 Cruise Control θ Response Using P Control ($\theta_o = 0.1$ rad)	128
4.45 Cruise Control θ Response Using PD Control ($\theta_o = 0.1$ rad)	128
4.46 Visualization of $(\Delta x, \theta)$ Separation-Direction Control System.....	129
4.47 Vehicle Separation Control (Proportional Control: $K_p = 0.5, 1.0, 1.5, 2.0;$ $\Delta x_o = 1$)	130
4.48 Vehicle Separation Control (PD control: $K_p = 0.5, 1.0, 1.5, 2.0; K_d=1;$ $\Delta x_o = 1$)	131
4.49 Minimizing Counter Value Versus Desired $f = \frac{x}{x_{resolution}}$	136
4.50 Minimum Percent Error Versus Desired $f = \frac{x}{x_{resolution}}$	137
4.51 Cruise Control Along a Line	145
4.52 Vehicle Separation Convergence Using Proportional Control ($K_p = 1;$ $\Delta x(0) \approx 1$)	147
4.53 Vehicle Separation Convergence Using Proportional Control ($K_p = 1;$ $\Delta x(0) \approx 1$)	148

Figure	Page
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	154
5.4 Visualization of i-th Follower in Platoon Separation Control System ...	155
5.5 Visualization of Platoon Controller	155
5.6 Simulation of Vehicle Separation Control of Platoon (Proportional Control for $\Delta_x (K_p = 0.5)$)	157
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 $\Delta_x (K_p = 1.0)$)	158
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 $\Delta_x (K_p = 2.0)$)	159
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 $\Delta_x (g = 0.2)$)	161
5.14 Simulation of Vehicle Separation Control of Platoon (PD Control for $\Delta_x (g = 0.5)$)	161

Figure	Page
5.15 Simulation of Control Response of Platoon (PD Control for Δ_x ($g = 0.5$))	162
5.16 Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 1.0$))	163
5.17 Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 1.0$))	164
5.18 Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 2.0$))	164
5.19 Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 2.0$))	165
5.20 Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_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 Δ_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 Δ_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 Δ_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 Δ_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 Δ_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 Δ_x ($g = 1.0, z = 1.0$) and PI Control for FF-path ($g_{ff} = 0.5, z_{ff} = 2.0$))	170

Figure	Page
5.27 Simulation of Control Response of Platoon (PID Control for Δ_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 Δ_x ($K_p = 1$))	173
5.29 Simulation of Control Response of Platoon (Proportional Control for Δ_x ($K_p = 1$))	173
5.30 Experimental Separation Response of Platoon Proportional Control for Δ_x ($K_p = 1$)	174
5.31 Simulation of Vehicle Separation Control of Platoon (PD Control for Δ_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 Δ_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 Δ_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 Δ_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 Δ_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
5.36 Experimental Separation Response of Platoon (PID Control for Δ_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

Figure	Page
5.37 Simulation of Separation Response of Platoon with model uncertainty and stiction	180
5.38 Simulation of Control Response of Platoon with model uncertainty and stiction.....	180

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, θ) controllable degrees of freedom is less than the three (3) total (x, y, θ) 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 θ 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, ω) 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 (ω_R, ω_L). The map from the voltages to the vehicle longitudinal and angular speeds (v, ω) 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 (ω_R, ω_L) and (v, ω) 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 (ω_R, ω_L) is a TITO LTI coupled model that is nearly decoupled (decentralized) at low frequencies; i.e. frequencies below $\frac{\beta}{I}$, where β denotes motor shaft rotational speed damping and I denotes rotational moment of inertia. (This is not true for (v, ω) .) 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, ω) or (ω_R, ω_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 β , back emf and torque constant $K_b = K_t$, armature resistance R_a . (Additional relevant details

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 (ω_R, ω_L) and (v, ω) control laws for our differential-drive vehicles. Inner-loop control law parameter trade studies are presented within Chapter 3. Both 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, θ) 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, ω) or (ω_R, ω_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}, ω_{ref}) to the actual velocities (v, ω) looks like a simple diagonal system (e.g. $diag(\frac{a}{s+a}, \frac{b}{s+b})$) 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 θ 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 θ . 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 [?], [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 text [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 basis 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 ω . These are used to design our proportional plus integral (PI) (ω_R, ω_L) or (v, ω) 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 $r_{wheel}(\frac{2\pi}{20}) \approx 0.0157$ m (where $r_{wheel} = 0.05$ 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 (2592×1944 pixel or 5 MP static images; 1080p30 (30 fps), 720p60 and

640×480 p60/90 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, θ) , (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, θ) 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 us used to implement outer-loop (v, θ) , $(\Delta x, \theta)$, 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.

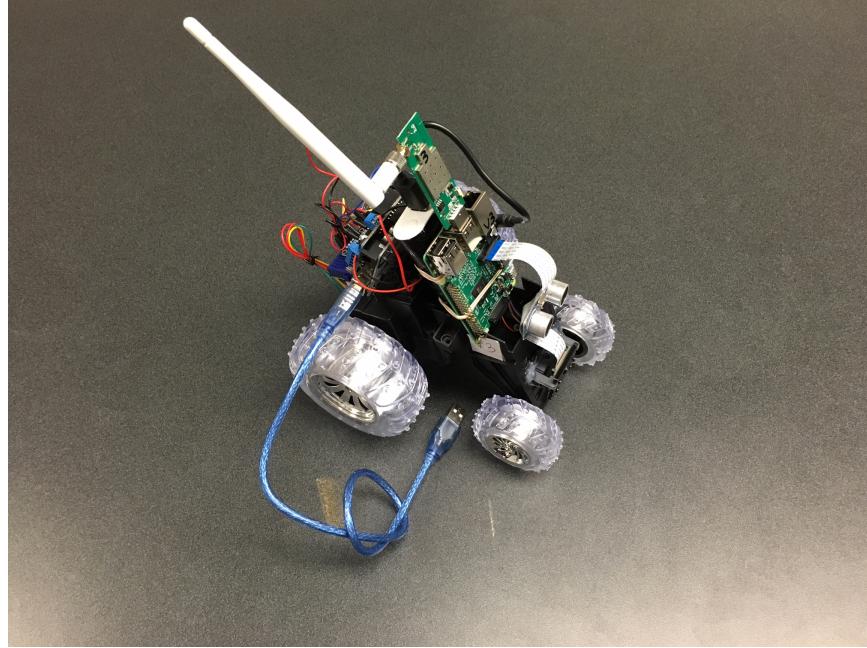


Figure 1.1: Visualization of Fully-Loaded (Enhanced) Thunder Tumbler

Each differential-drive Thunder Tumbler vehicle was augmented to provide a suite of substantive capabilities. Each augmented (“enhanced” Thunder Tumbler) vehicle costs less than \$175 but offers the capability of commercially available vehicles costing over \$500.

(1) optical wheel encoders and IMU (RPR220 photo-interrupter REFL 6mm 800nm, 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

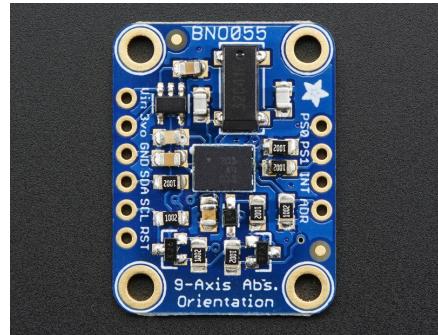


Figure 1.3: Adafruit BNO055 9DOF Inertial Measurement Unit (IMU)

(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, ω) or (ω_R, ω_L) inner-loop control and encoder-IMU-ultrasound-based cruise-position-directional-separation outer-loop control,

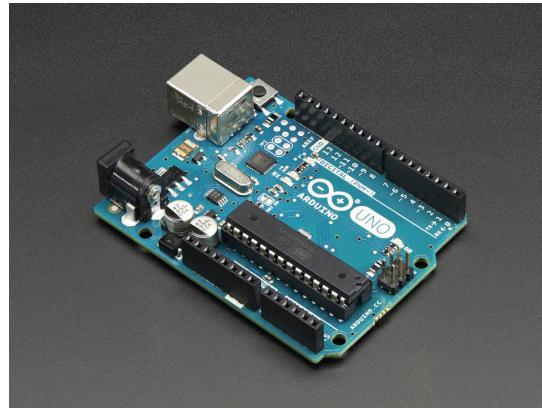


Figure 1.4: Arduino Uno Open-Source Microcontroller Development Board

(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

(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,



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

(5) a Raspberry Pi 5MP camera (2592 × 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,



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

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



Figure 1.9: HC-SR04 Ultrasonic Sensor

- 2. Why should a hierarchical inner-outer loop control architecture be used?**[14] 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, ω) or (ω_R, ω_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, ω) or (ω_R, ω_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 (ω_R, ω_L) - (v, ω) control laws. The IMU was used to implement our (v, θ) 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, ω) speed control law and three outer-loop control laws: (1) (v, θ) , (2) $\Delta x, \theta$, and (3) vehicular platoon separation control

Mathematic Name	Control Law	Visualization
(v, ω)	Inner-loop speed control (encoders)	
(v, θ)	Cruise control along a line (v- encoders, θ - IMU or camera)	
$(\Delta x, \theta)$	Separation control along a line (Δx -ultrasonic, θ -IMU)	
Δx_i	Control of longitudinal platoon of vehicles (Δx -ultrasonic, θ -IMU, Wi-Fi Adapter)	

Figure 1.10: Visualization of Inner- and Outer-Loop Control Laws

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, θ) 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. $diag(\frac{a}{s+a}, \frac{b}{s+b})$, 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 $\left[\begin{smallmatrix} a & 0 \\ 0 & \frac{a}{s(s+a)} \end{smallmatrix} \right]$; i.e. there is an addi-

tional 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 93) 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 182) 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 192) contains Arduino program files used to generate inner loop results for this thesis.
- Appendix B (page 269) contains all MATLAB mfiles used to generate the results for this thesis.
- Appendix C (page 299) 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 AND C^4S REQUIREMENTS

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 (C^4), and sensing (S) requirements needed to support a fleet of collaborating vehicles. Collectively, the C^4 and S requirements are referred to as C^4S requirements.

2.2 FAME Architecture and C^4S Requirements

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 (C^4), and sensing (C^4S) 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

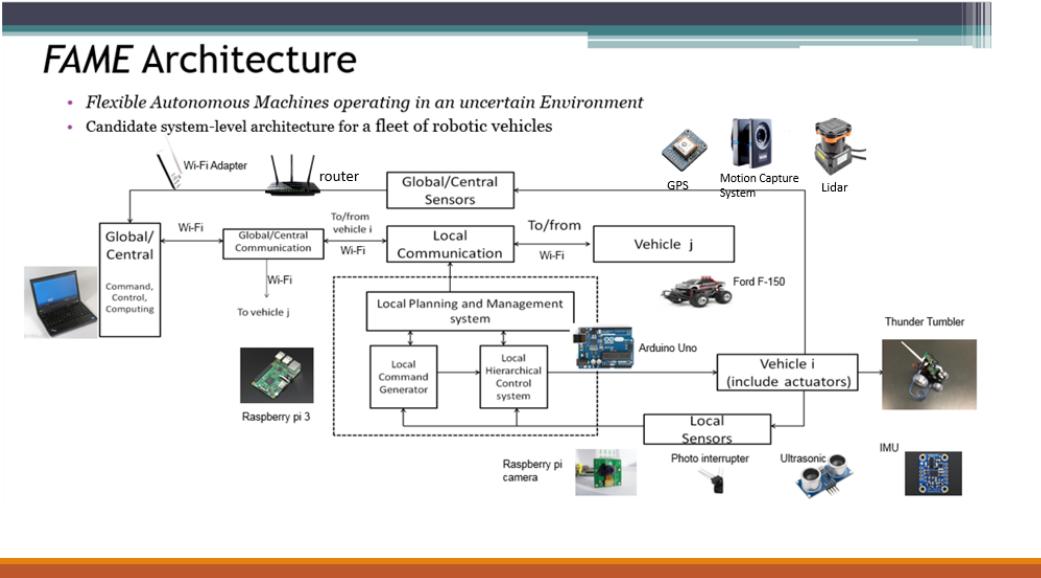


Figure 2.1: *FAME* Architecture to Accommodate Fleet of Cooperating Vehicles

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 \pm 2,4,6,8,16g. Gyro \pm 125 – 2000°/sec. Compass \pm 1300 μ T (x-,y-axis) \pm 2500 μ T (z-axis)) [58], ultrasonic range sensors (40kHz, 0.02-5 m, approximately \pm 8° directional) [68], and Raspberry Pi cameras(2592 \times 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 \times 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, ω) 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.

Product	Quantity	Price (\$)
Thunder Tumbler Vehicle	1	\$10
Raspberry Pi 3 Model B	1	\$40
Arduino Uno	1	\$25
Adafruit Motor Shield	1	\$20
Raspberry Pi 5MP Camera	1	\$20
TP-LINK TL-WN722N USB Wi-Fi Adapter	1	\$14
Camera Holder	1	\$5
HCSR04 Ultrasonic Sensor	1	\$5
EasyAcc PowerBank for Raspberry Pi	1	\$13
Power Supply for Motors	4	\$3.5
Optical Sensor Photo-Interrupter	2	\$1.5
Adafruit 9-DOF IMU BNO055	1	\$35
Miscellaneous	1	\$15
Total Price		\$219

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 (θ) 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: $\pm 2/4/6/8/16$ g linear acceleration full scale, ± 13 gauss (x-,y-axis) ± 25 gauss (z-axis)), magnetic full scale and $\pm 125\text{-}2000$ 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 SDA-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, θ) cruise control along a line (2) Outer-loop 2 (Δ_x, θ) 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 inforamtion, 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 Δx 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.

Parameter	Definition	Nominal Values
m	Mass (Enhanced Vehicle)	$0.87 \pm 0.02 \text{ kg}$
m_o	Mass (Original Vehicle)	$0.56 \pm 0.02 \text{ kg}$
I_z	Moment of Inertia	0.0051 kg m^2
r	Wheel Radius	$0.05 \pm 0.005 \text{ m}$
d_w	Distance between Rear Wheels	$0.14 \pm 0.005 \text{ m}$
L_a	Armature Inductance	$(2.7 \pm 0.2) \times 10^{-8} \text{ H}$
R_a	Armature Resistance	$0.9 \pm 0.15 \Omega$
K_g	Gearbox Ratio	18.48
K_b	back EMF Constant	$(3.2 \pm 0.5) \times 10^{-4} \text{ V/(rad/sec)}$
K_t	Torque Constant	$(3.2 \pm 0.5) \times 10^{-4} \text{ Nm/A}$
I	Equivalent Moment of Inertia	$(2.91 \pm 1.00) \times 10^{-6} \text{ kg m}^2$
β	Speed Damping Constant	$(7.04 \pm 3.00) \times 10^{-7} \text{ Nms}$
v_{max}	Max. Speed (Enhanced Vehicle)	$2.1 \pm 0.2 \text{ m/sec}$
v_{max_o}	Max. Speed (Original Vehicle)	$4.5 \pm 0.2 \text{ m/sec}$
E	Efficiency - Power Out / Power In	0.15
a_{max}	Max. Acceleration (Enhanced)	$2.5 \pm 0.5 \text{ m/sec}^2$
$\omega_{s_{max}}$	Max. Angular Wheel Velocity	$46 \pm 4 \text{ rad/sec}$
$\tau_{s_{max}}$	Max. Torque (Zero RPM)	$0.048 \pm 0.05 \text{ Nm}$
$e_{a_{max}}$	Max. Motor Voltage	$5.25 \pm 0.05 \text{ V}$

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].

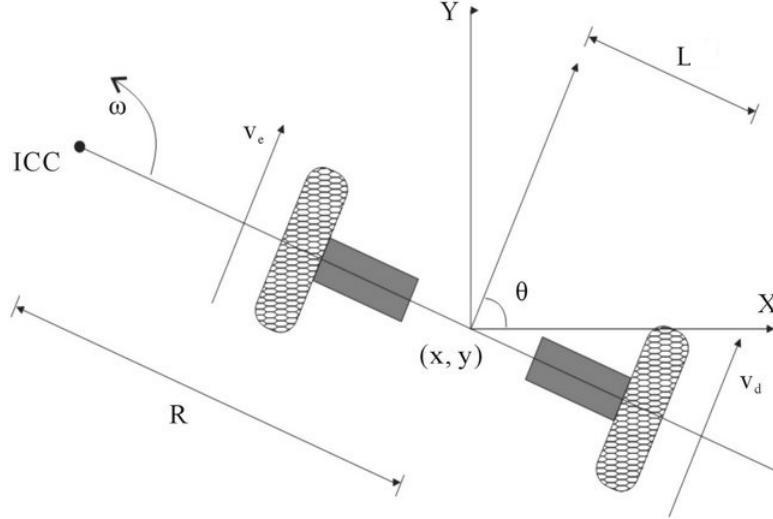


Figure 3.1: Visualization of Differential-Drive Mobile Robot

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 θ denotes the direction of the robot's longitudinal body axis with respect to the x -axis, then we obtain the following nonlinear kinematic model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta \\ \sin \theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \quad (3.1)$$

where

$$v = \sqrt{\dot{x}^2 + \dot{y}^2} \quad (3.2)$$

denotes the translational speed of the robot and $\omega = \dot{\theta}$ denotes its angular speed.

Within the above very simple (and intuitive) model, v and ω 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 ω cannot be instantaneously generated because of real-world mass-inertia effects (at least not without infinite forces!). In practice, v and ω are generated by applying voltages to the left and right wheel DC motors. This will 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, ω) to the angular velocities (ω_L, ω_R) of the left and right rear wheels. Why? The idea here, is that if we can precisely control (ω_L, ω_R) , then we will be able to precisely control (v, ω) . The desired relationships are as follows:

$$v = \left[\frac{r(\omega_R + \omega_L)}{2} \right] \quad \omega = \left[\frac{r(\omega_R - \omega_L)}{d_w} \right] \quad (3.3)$$

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

$$(R + d_w/2)\omega = v_r \quad (R - d_w/2)\omega = v_l \quad (3.4)$$

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

$$R = \frac{d_w}{2} \left[\frac{v_l + v_r}{v_r - v_l} \right] \quad \omega = \left[\frac{v_r - v_l}{d_w} \right] \quad (3.5)$$

Next, we note that

$$v = R\omega \quad v_l = r\omega_L \quad v_r = r\omega_R \quad (3.6)$$

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 $R = \frac{d_w}{2} \frac{v_l + v_r}{v_r - v_l}$ 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:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = M \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \quad M = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d_w} & -\frac{r}{d_w} \end{bmatrix} \quad (3.7)$$

Again, the importance of the above relation stems from the fact that if we can control (ω_L, ω_R) well, then we shall see that we will be able to control (v, ω) 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]:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (3.8)$$

$$\begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \frac{1}{m} & 0 \\ 0 & \frac{1}{I_z} \end{bmatrix} \begin{bmatrix} F \\ \tau \end{bmatrix} \quad (3.9)$$

$$\tan \theta = \frac{\dot{y}}{\dot{x}} \quad (3.10)$$

where F represents the applied translational force along the vehicle's longitudinal body axis, τ 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 τ 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, ω) can be generated quickly. Intuitively, this occurs when m and I_z are sufficiently small (see equation (3.9)) or the inner (v, ω) loop has a sufficiently large bandwidth. This shall become evident below.

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

$$F = \left[\frac{\tau_R + \tau_L}{r} \right] \quad \tau = \left[\frac{d_w(\tau_R - \tau_L)}{2r} \right] \quad (3.11)$$

Here, τ_L and τ_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 (τ_L, τ_R) and hence the associated pair (F, τ) . The latter, of course, are directly responsible for producing the vehicle speeds (v, ω) .

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:

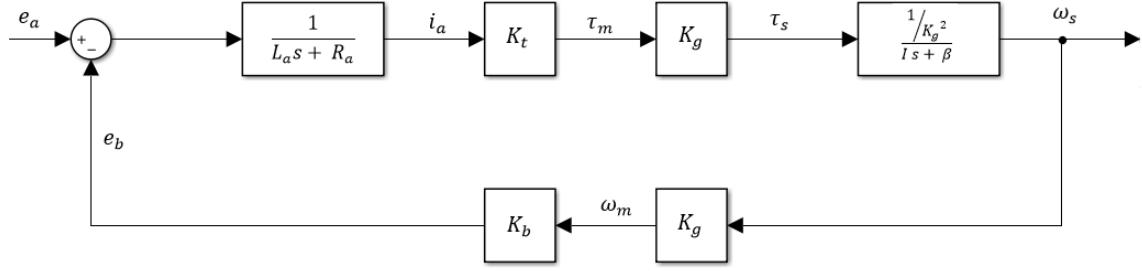


Figure 3.2: Visualization of DC Motor Speed-Voltage Dynamics

Armature Equation:

$$e_a = L_a \frac{di_a}{dt} + R_a i_a + e_b \quad (3.12)$$

back EMF Equation:

$$e_b = K_b K_g \omega_s \quad (3.13)$$

Torque Equation:

$$\tau_s = K_t K_g i_a \quad (3.14)$$

Load Equation:

$$I \dot{\omega}_s + \beta \omega_s = \frac{\tau_s}{K_g^2} \quad (3.15)$$

Gear Relationship:

$$K_g = K_{g12} K_{g23} \quad (3.16)$$

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, τ_s represents the torque exerted by the motor on the motor shaft-load system, ω_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 β 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 ω_s :

$$\frac{\omega_s}{e_a} = \left[\frac{\frac{K_t}{K_g}}{(Is + \beta)(L_a s + R_a) + K_t K_b} \right] \quad (3.17)$$

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. $\omega_s L_a \ll R_a$ 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:

$$\frac{\omega_s}{e_a} \approx \left[\frac{\frac{K_t}{K_g}}{(Is + \beta)(R_a) + K_t K_b} \right] \left[\frac{R_a}{L_a s + R_a} \right] \quad (3.18)$$

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 (β, 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, $K_t = K_b$.

3.5.2 Empirically Obtained Experimental Data for Motor-Wheel System

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 , β .

- *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

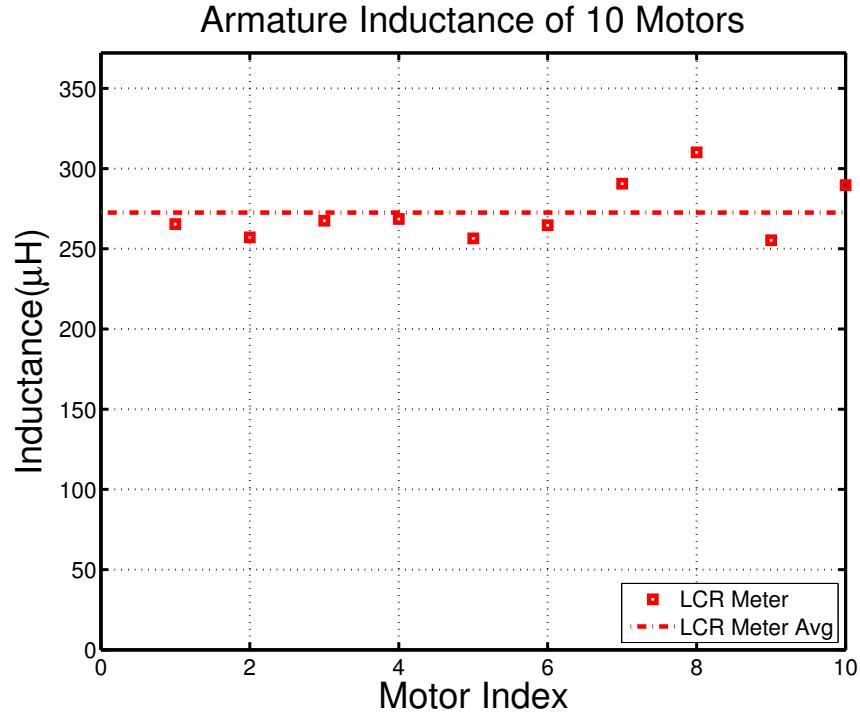


Figure 3.3: dc Motor Armature Inductance of 10 ETT Motors

From the figure above, we can see measurements are close to the average result is around $270 \mu\text{H}$.

- *Armature resistance R_a .* Armature resistance is usually very small in toy DC motor, one common way is to use 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

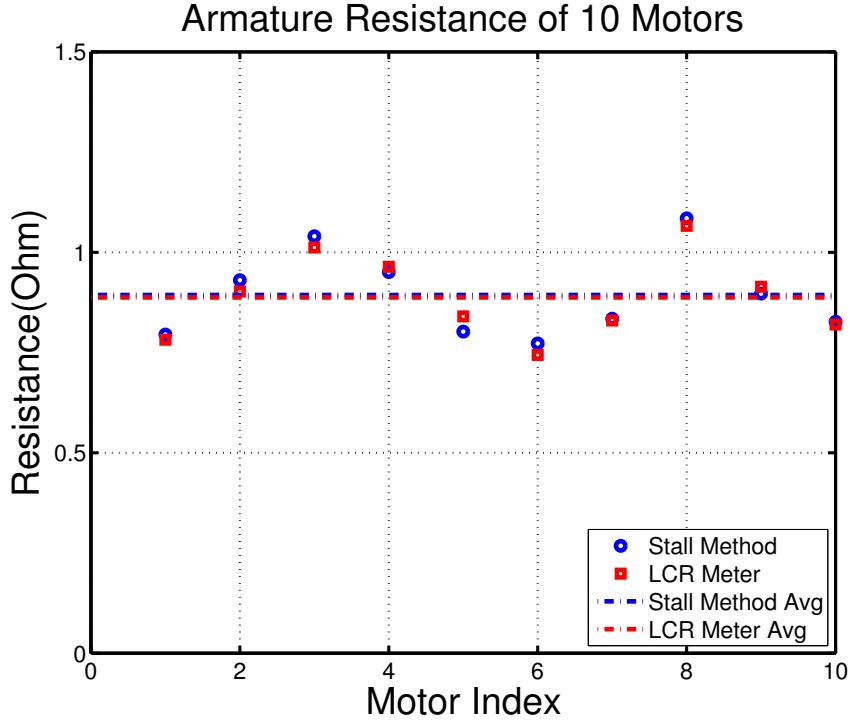


Figure 3.4: dc Motor Armature Inductance of 10 ETT Motors

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 0.8933Ω and LCR meter at 0.8874Ω .

- *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.

Parameter	Definition
K_g	Gear Ratio between First and Third Load
$K_{g12}(n12)$	Gear Ratio between First and Second Load
$K_{g23}(n23)$	Gear Ratio between Second and Third Load
r_1	Radius of First Pinion Gear on First Load
r_{21}	Radius of Gear Connect the First and Second Load
r_{23}	Radius of Gear Connect to the Second and Third Load
N_1	Teeth Number on First Load
N_{21}	Teeth Number on Gear Connect the First and Second Load
N_{23}	Teeth number on Gear Connect the Second and Third Load
τ_m	Torque Generated by Motor at First Load
τ_2	Torque Exerted by the First Gear on the Second Load
τ_s	Torque at Output Shaft Connect to the Wheel (Third Load)
ω_m	Angular Speed of Pinion Gear at First Load
ω_2	Angular Speed of Gears on Second Load
ω_s	Angular Speed of the Wheel Gear(Third Load).

Table 3.3: Thunder Tumbler Gearbox Symbols

τ_1 denote the torque exerted by the second gear on the first gear and τ_2 denote the torque exerted by the first gear on the second, (ω_1, ω_2) are the respective angular speed of each gear.

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

$$n12 = \frac{1}{K_{g12}} = \frac{r_1}{r_{21}} = \frac{N_1}{N_{21}} = \frac{\tau_m}{\tau_2} = \frac{\omega_m}{\omega_2} \quad (3.20)$$



Figure 3.5: Disassembled Gearbox in ETT

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 , β . 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,

$$K_b = \frac{e_a - R_a i_a}{K_g \omega_s} \quad (3.24)$$

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

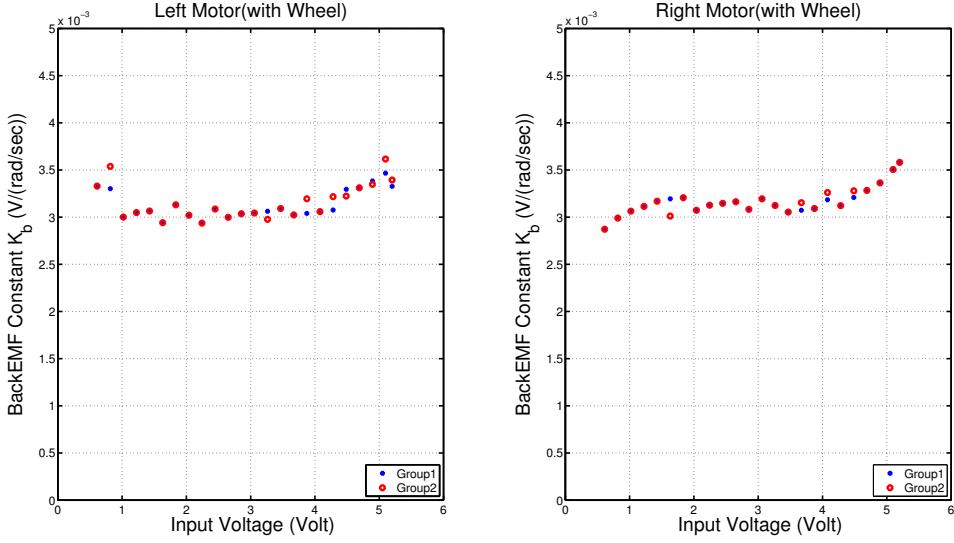


Figure 3.6: DC Motor back EMF Constant of Left and Right ETT Motors

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 β .* According to 3.17, we observe that the

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

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

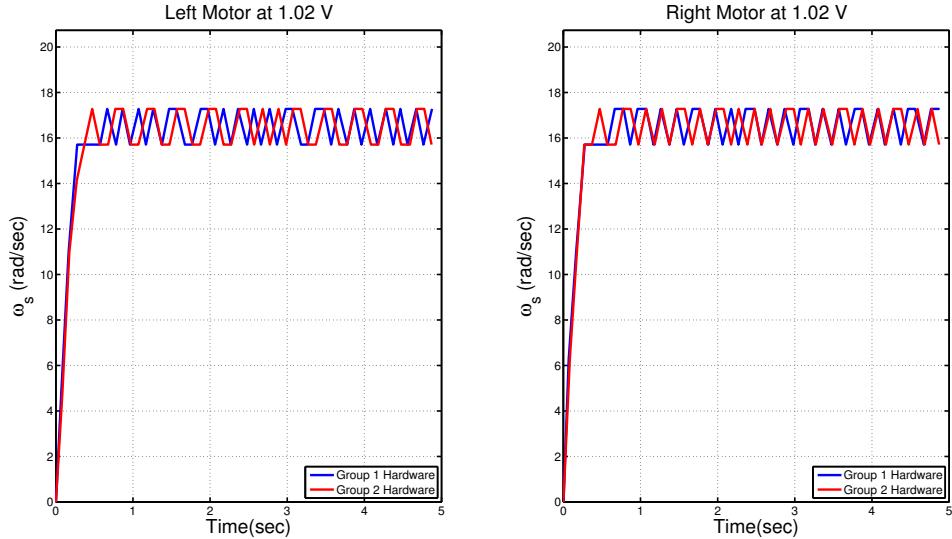


Figure 3.7: DC Motor Output ω_s Response to 1.02V Step Input - Hardware

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.

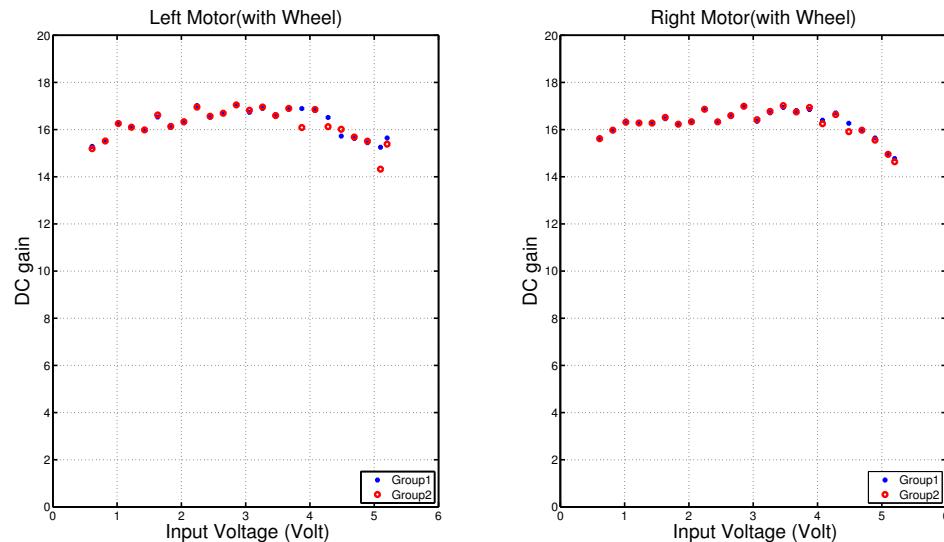


Figure 3.8: DC Gain Distribution of Step Response at Different Voltage Step Input

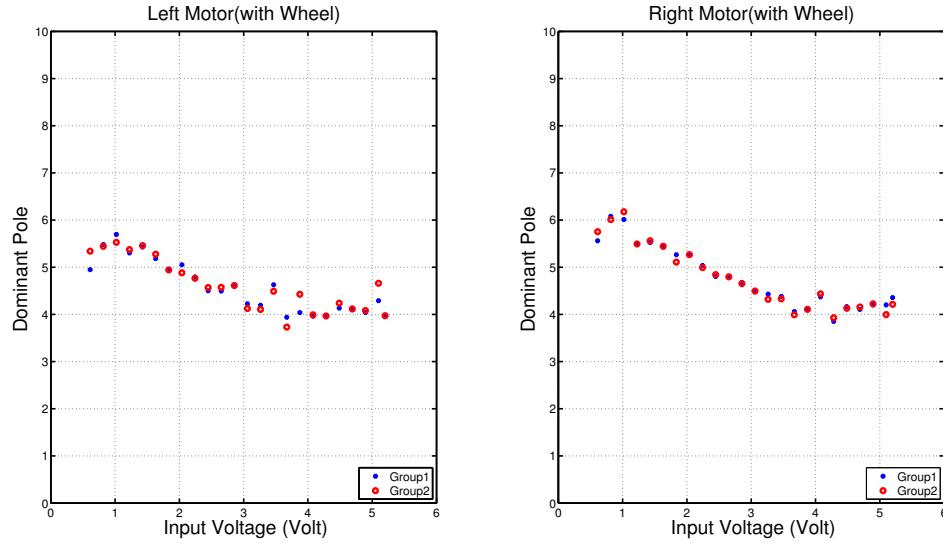


Figure 3.9: DC Gain Distribution of Step Response at Different Voltage Step Input

The dominant pole is obtained by settling time. For example, of $1sec = 5\tau = \frac{5}{|real\ pole|}$ 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 β , the result shown in Figure 3.10 and 3.11 respectively.

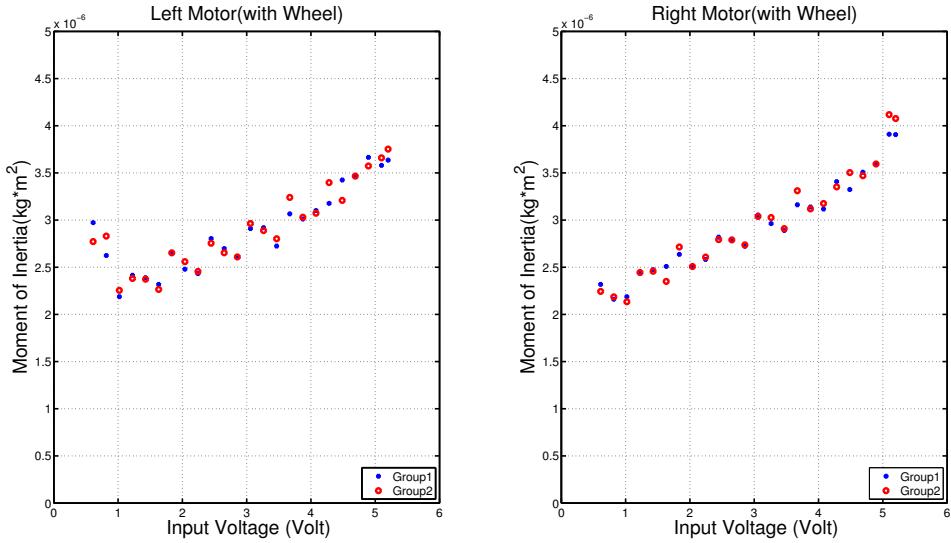


Figure 3.10: Equivalent Moment of Inertia Distribution of Left and Right Motors

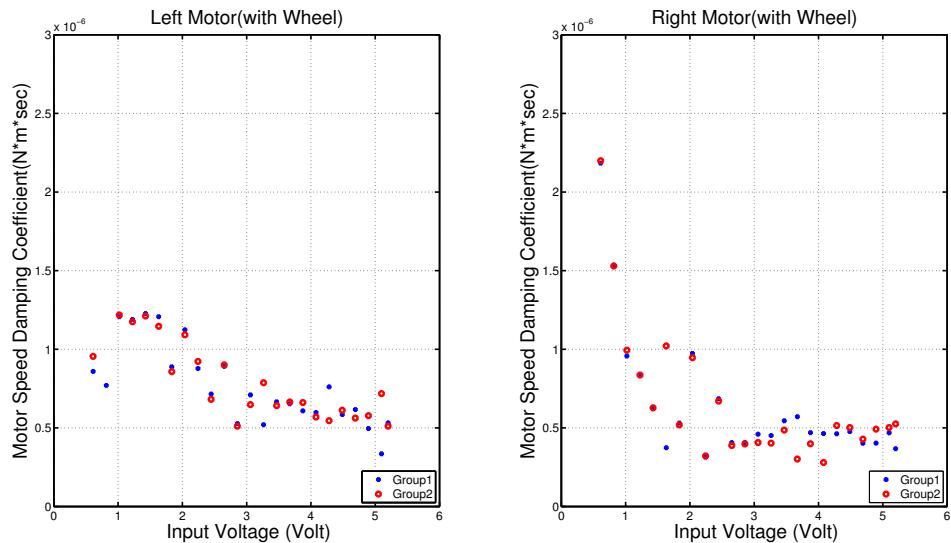


Figure 3.11: Equivalent Speed Damping Constant Distribution of Left and Right Motors

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 β 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).

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

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.

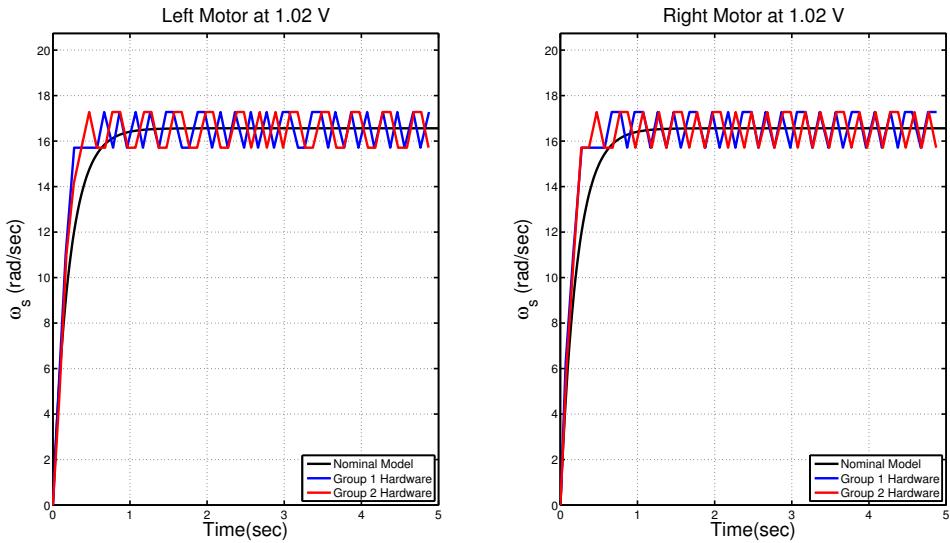


Figure 3.12: Motor Output ω_s Response to 1.02V Step Input - Hardware and Decoupled Model

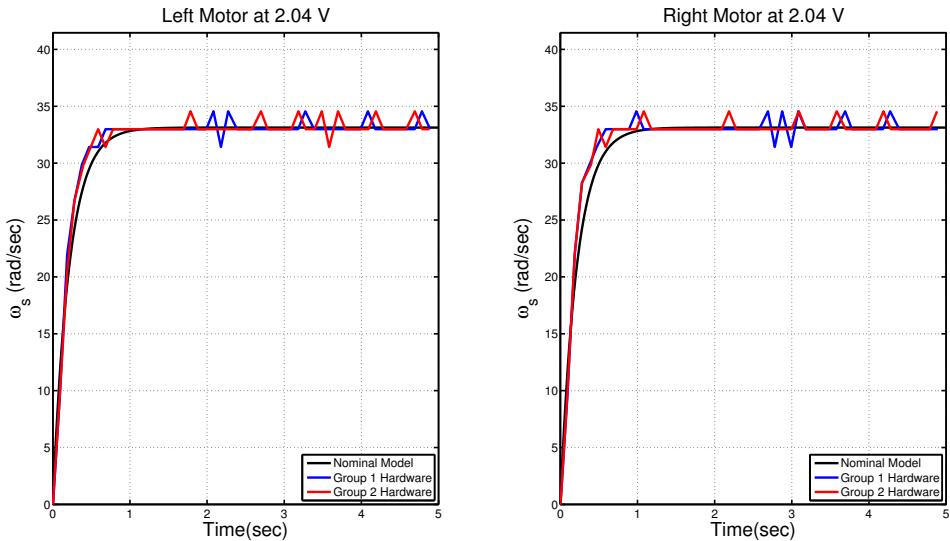


Figure 3.13: Motor Output ω_s Response to 2.04V Step Input - Hardware and Decoupled Model

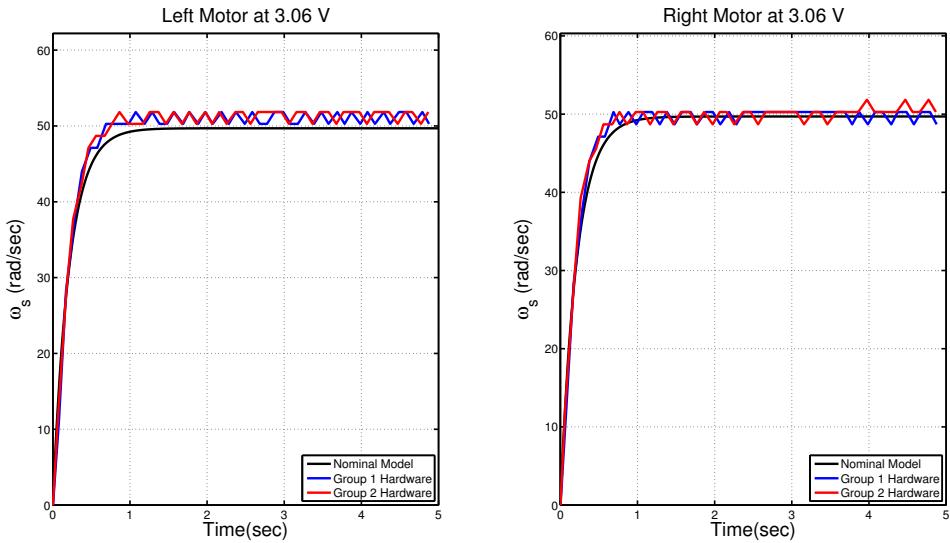


Figure 3.14: Motor Output ω_s Response to 3.06V Step Input - Hardware and Decoupled Model

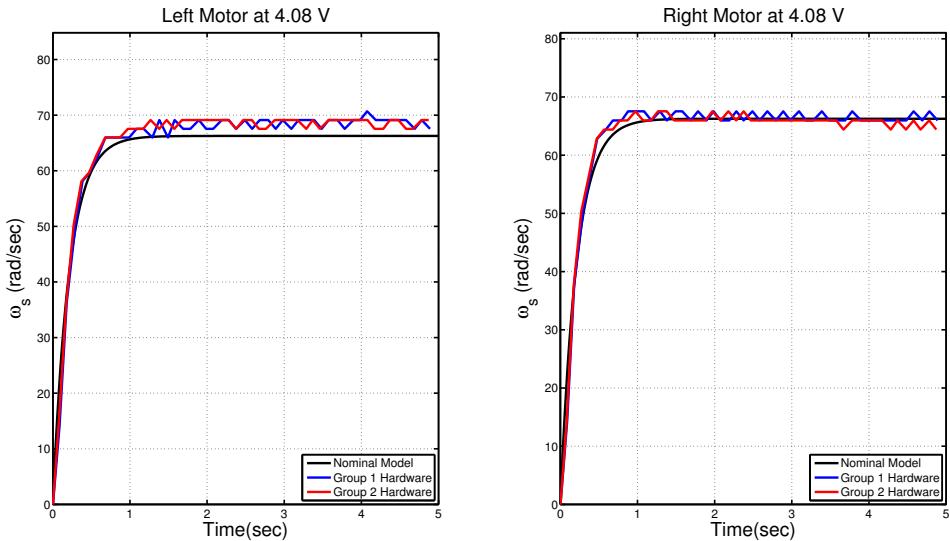


Figure 3.15: Motor Output ω_s Response to 4.08V Step Input - Hardware and Decoupled Model

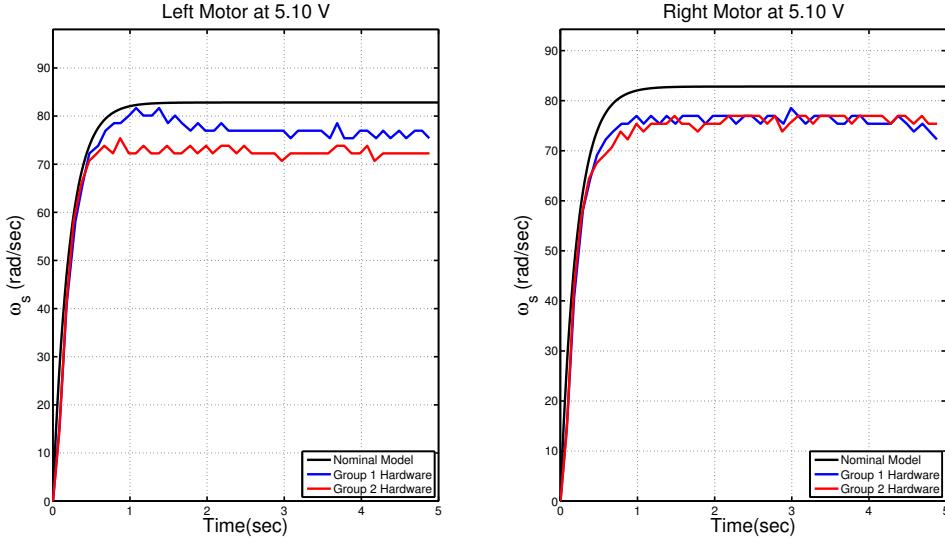


Figure 3.16: Motor Output ω_s Response to 5.10V Step Input - Hardware and Decoupled Model

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 (ω_R, ω_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-

porates 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 132. 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.

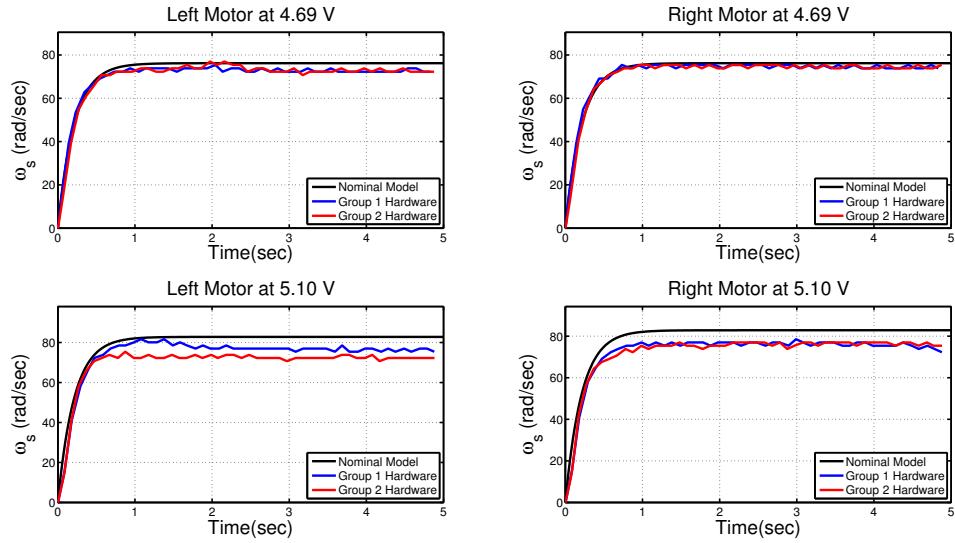


Figure 3.17: DC Motor Output ω_s Response to High Voltage Step Input - Hardware and Decoupled Model

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 $0.05 - 0.15 \Omega$ [40]. We uses 4 units as the power input to motor shield board, which could result in IR $0.2 - 0.6 \Omega$. 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 (ω_R, ω_L) can be visualized as shown within Figure 3.18. Note the cross-coupling introduced by the right-left motor torques (τ_R, τ_L) at the input and a similar structure at the output.

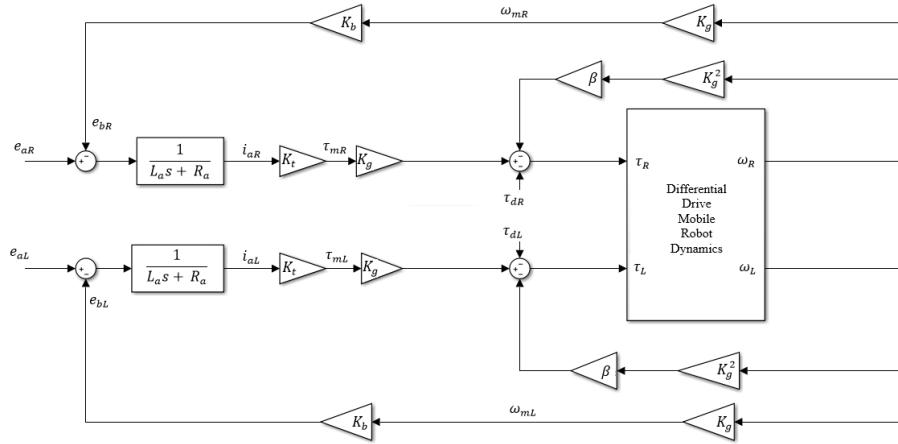


Figure 3.18: TITO LTI Robot-Motor Wheel Speed (ω_R, ω_L) Dynamics - $P_{(\omega_R, \omega_L)}$

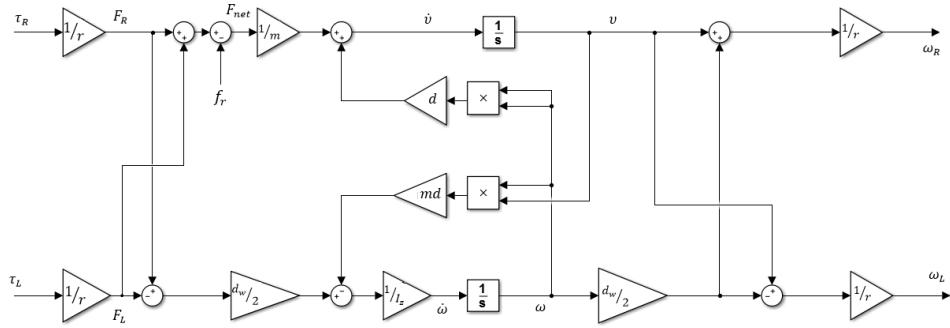


Figure 3.19: Differential-Drive Mobile Robot Dynamics

The associated fourth order TITO LTI state space representation ¹ is given by

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

¹Neglect friction, torque disturbance and distance between geometric center and mass center

where $x = [v \ \omega \ i_{aR} \ i_{aL}]^T$, $y = [\omega_R \ \omega_L]^T$, $u = [e_{aR} \ e_{aL}]^T$,

$$A = \begin{bmatrix} \frac{-2\beta K_g^2}{mr^2} & 0 & \frac{K_g K_t}{mr} & \frac{K_g K_t}{mr} \\ 0 & \frac{-\beta K_g^2 d_w^2}{2I_z r^2} & \frac{K_g K_t d_w}{2I_z r} & \frac{-K_g K_t d_w}{2I_z r} \\ \frac{-K_g K_b}{L_a r} & \frac{-K_g K_b d_w}{2L_a r} & \frac{-R_a}{L_a} & 0 \\ \frac{-K_g K_b}{L_a r} & \frac{K_g K_b d_w}{2L_a r} & 0 & \frac{-R_a}{L_a} \end{bmatrix} \quad (3.29)$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{L_a} & 0 \\ 0 & \frac{1}{L_a} \end{bmatrix} \quad (3.30)$$

$$C = \begin{bmatrix} \frac{1}{r} & \frac{d_w}{2r} & 0 & 0 \\ \frac{1}{r} & \frac{-d_w}{2r} & 0 & 0 \end{bmatrix} \quad (3.31)$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (3.32)$$

Here, (i_{aL}, i_{aR}) represent left and right motor armature currents, v is the vehicle's translational velocity (directed along the direction θ), ω is the vehicle's angular velocity, (ω_L, ω_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, β 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

$$P_{(\omega_R, \omega_L)} = C(sI - A)^{-1}B + D = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \quad (3.33)$$

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:

$$P_{(\omega_R, \omega_L)} = \frac{\begin{bmatrix} 186056(s + 2977.5)(s + 3.8722) & 14533s(s + 2981.1) \\ 14533s(s + 2981.1) & 186056(s + 2977.5)(s + 3.8722) \end{bmatrix}}{(s + 2977.7)(s + 2977.2)(s + 4.2008)(s + 3.5914)} \quad (3.34)$$

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

$$P_{(\omega_R, \omega_L)} \approx \begin{bmatrix} 62.411(s + 3.8675) & 4.8749s \\ 4.8749s & 62.411(s + 3.8675) \end{bmatrix} \frac{1}{(s + 4.1951)(s + 3.5872)} \quad (3.35)$$

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:

$$p_1 \approx - \left[\frac{2K_g^2(K_b K_t + R_a \beta)}{R_a m r^2} \right] \quad p_2 \approx - \left[\frac{K_g^2 d_w^2 (K_b K_t + R_a \beta)}{2R_a I_z r^2} \right] \quad p_3 \approx p_4 \approx - \frac{R_a}{L_a} \quad (3.36)$$

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:

$$\text{Nominal Poles: } -3.5914, -4.2008, -2977.2, -2977.7$$

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:

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

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 (ω_R, ω_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 (ω_R, ω_L) system looks the same.

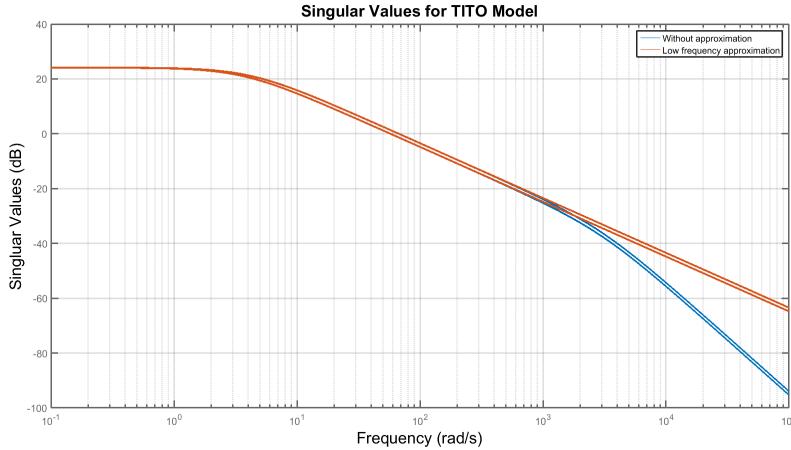


Figure 3.20: Robot Singular Values (Voltages to Wheel Speeds) - Including Low Frequency Approximation

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 (ω_R, ω_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.

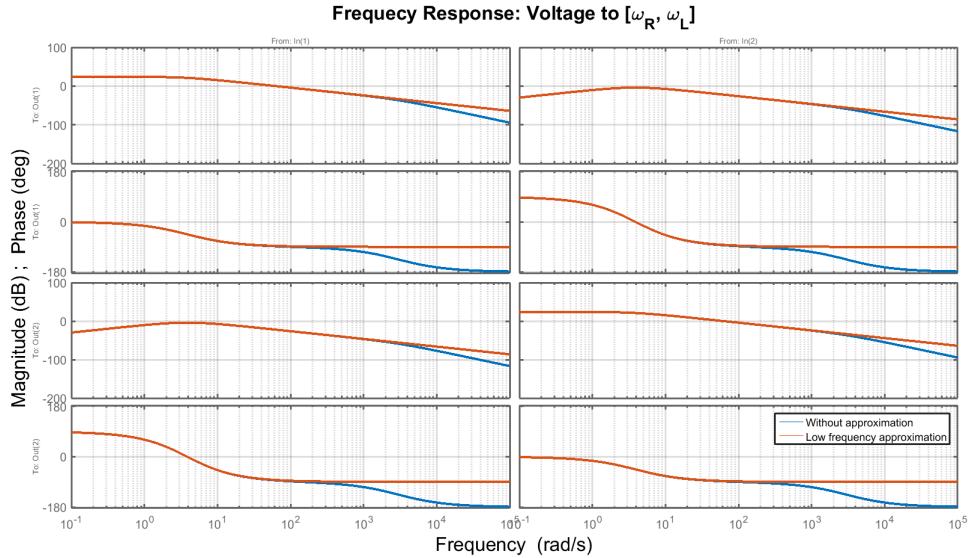


Figure 3.21: Robot Frequency Response (Voltages to Wheel Speeds) - Including Low Frequency Approximation

Plant. The above discussion has focused on the (ω_R, ω_L) system. It must be noted that strictly speaking, the (ω_R, ω_L) system is not the plant for our inner-loop control system. Why? The inner-loop plant, strictly speaking, has (v, ω) as outputs since these are the variables that we wish to command! That is, we shall be specifying (v_{ref}, ω_{ref}) reference commands to our inner-loop control system. The transfer function matrix for our (v, ω) system - the plant P - is as follows:

$$P = P_{(v,\omega)} = \frac{\begin{bmatrix} 5014.7(s + 3.591)(s + 2978) & 5014.7(s + 3.591)(s + 2978) \\ 61258(s + 4.201)(s + 2977) & -61258(s + 4.201)(s + 2977) \end{bmatrix}}{(s + 3.591)(s + 4.201)(s + 2977)(s + 2978)} \quad (3.39)$$

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

$$P = P_{(v,\omega)} \approx \begin{bmatrix} 1.6822(s + 3.587) & 1.6822(s + 3.587) \\ 20.549(s + 4.195) & -20.549(s + 4.195) \end{bmatrix} \frac{1}{(s + 3.587)(s + 4.195)} \quad (3.40)$$

The above shows that unlike the (ω_R, ω_L) system, the (v, ω) 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:

$$P(0) = P_{(v,\omega)}(0) = \begin{bmatrix} 0.401 & 0.401 \\ 5.73 & -5.73 \end{bmatrix} = U\Sigma V^H \quad (3.41)$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 8.101 & 0 \\ 0 & 0.5671 \end{bmatrix} \begin{bmatrix} 0.7071 & 0.7071 \\ -0.7071 & 0.7071 \end{bmatrix}^H \quad (3.42)$$

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 ω ($u_1 = [0 \ 1]^T$) 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 ω 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 ω 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 (ω_R, ω_L) to (v, ω) . More precisely, up above we showed (via svd at dc) that the maximum singular value at low frequencies is associated with rotation ω 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

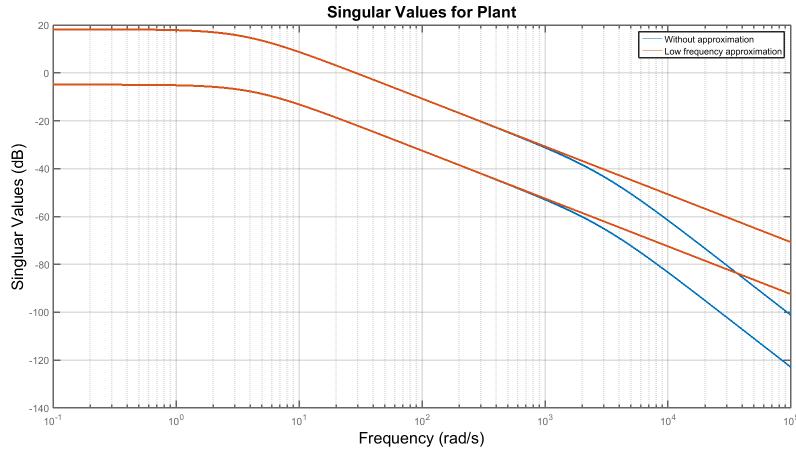


Figure 3.22: Robot Plant Singular Values (Voltages to v and ω) - Including Low Frequency Approximation

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 (ω_R, ω_L) system is fairly decoupled at low frequencies, this is not the case for our (v, ω) 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) (ω_R, ω_L) system to indirectly control our (v, ω) plant system. This, and important issues discussed below, will receive greater attention in future work.

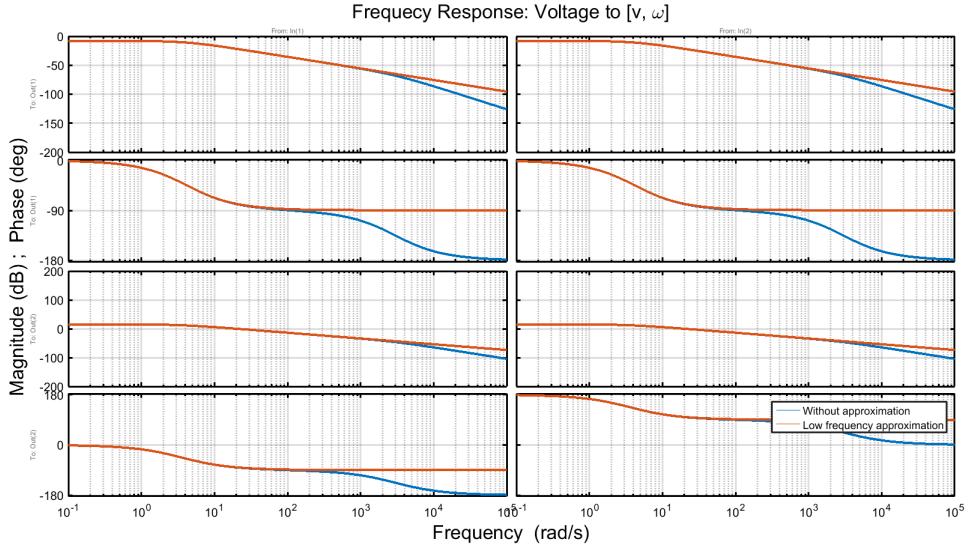


Figure 3.23: Robot Plant Frequency Response (Voltages to $[v, \omega]$) - Including Low Frequency Approximation

Comparisons Between Decoupled and Coupled Models. Recall the coupled TITO LTI vehicle-motor (ω_R, ω_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

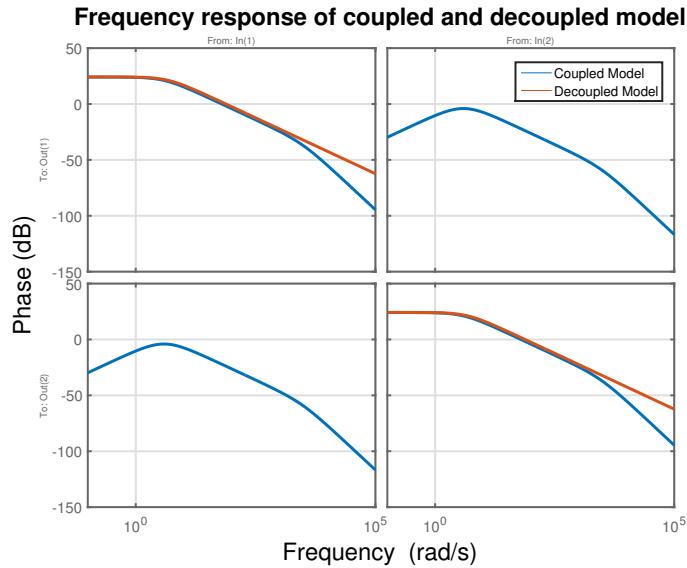


Figure 3.24: Frequency Response for Vehicle-Motor - Coupled (ω_R, ω_L) Model

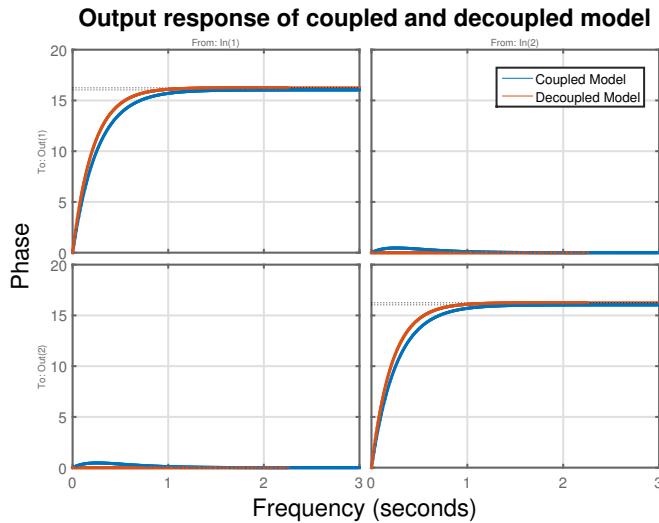


Figure 3.25: Vehicle-Motor Response to Unit Step Input - Coupled (ω_R, ω_L) Model

The following is a good approximation to vehicle-motor (ω_R, ω_L) plant:

$$P_{[\omega_R, \omega_L]} \approx 16.246 \left[\frac{4.685}{s + 4.685} \right] \times I_{2 \times 2} \quad (3.43)$$

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

If the above (v, ω) system is the plant we really want to control,
then why did we spend so much time on the (ω_R, ω_L) system?

A quick and simple answer to this is that

the (nearly decoupled) (ω_R, ω_L) system is much simpler
than the highly coupled (v, ω) plant system.

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

$$P = P_{(v, \omega)} = MP_{(\omega_R, \omega_L)} \quad (3.44)$$

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 (ω_R, ω_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, ω) and (ω_R, ω_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, ω) 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 (ω_R, ω_L) - (v, ω) Systems are Identical. From the structure of P , K , and M , it can be shown that

$$\sigma_i [PK] = \sigma_i [P_{(\omega_R, \omega_L)}K_{(\omega_R, \omega_L)}] \quad (3.45)$$

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 (ω_R, ω_L) system,

then they will be good for the (v, ω) 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 $T_{d_{iy}}$ (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 (ω_R, ω_L) and (v, ω) Systems.** Simple multivariable system algebra shows that

$$\begin{aligned} T_{ru} &= K(I + PK)^{-1} = K_{(\omega_R, \omega_L)} M^{-1} (I + MP_{(\omega_R, \omega_L)} K_{(\omega_R, \omega_L)} M^{-1})^{-1} \\ &= K_{(\omega_R, \omega_L)} (I + P_{(\omega_R, \omega_L)} K_{(\omega_R, \omega_L)})^{-1} M^{-1} \\ &= T_{ru_{(\omega_R, \omega_L)}} M^{-1} \end{aligned} \quad (3.46)$$

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

- **$T_{d_{iy}}$ Differences for (ω_R, ω_L) and (v, ω) Systems.** Similarly,

$$\begin{aligned} T_{d_{iy}} &= (I + PK)^{-1} P = (I + MP_{(\omega_R, \omega_L)} K_{(\omega_R, \omega_L)} M^{-1})^{-1} MP_{(\omega_R, \omega_L)} \\ &= M(I + P_{(\omega_R, \omega_L)} K_{(\omega_R, \omega_L)} M)^{-1} P_{(\omega_R, \omega_L)} \\ &= M T_{d_{iy}(\omega_R, \omega_L)} \end{aligned} \quad (3.47)$$

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 similarities and distinctions for the two systems under consideration: (ω_R, ω_L) and (v, ω) . While the above supports our focus above on $P_{(\omega_R, \omega_L)}$ rather than $P = P_{(v, \omega)}$ - modulo the T_{ru} and $T_{d_{iy}}$ 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

$$M = \left[\begin{array}{c|c} \frac{r}{2} & \frac{r}{2} \\ \hline \frac{r}{d_w} & -\frac{r}{d_w} \end{array} \right] = \left[\begin{array}{c|c} 0.025 & 0.025 \\ \hline 0.3571 & -0.3571 \end{array} \right] \quad (3.48)$$

$$M^{-1} = \left[\begin{array}{c|c} \frac{1}{r} & \frac{d_w}{2r} \\ \hline \frac{1}{r} & -\frac{d_w}{2r} \end{array} \right] = \left[\begin{array}{c|c} 20 & 1.4 \\ \hline 20 & -1.4 \end{array} \right] \quad (3.49)$$

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}^∞ norm, yields the following:

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

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

The first inequality imply that we should be particularly concerned about the control response to reference commands for the (v, ω) system. It suggests that the (v, ω) system can have up to 21.40 times the control effort of the (ω_R, ω_L) system (in the worst case). The second inequality tells us that the output response to input disturbances for the (v, ω) system will not be a concern if the (ω_R, ω_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, ω) ,
then what can happen if we control (ω_R, ω_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? (ω_R, ω_L) or (v, ω) ? Within this thesis, we use optical wheel encoders (20 CPT) to measure (ω_R, ω_L) . From this, (v, ω) 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, ω) , then we may be able to get more accurate measurements and hence better inner-loop performance in comparison to the encoder-based (ω_R, ω_L) approach taken within this thesis. Within the thesis, θ information directly obtained by the IMU built-in algorithm or using camera. This is done for our (v, θ) 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.

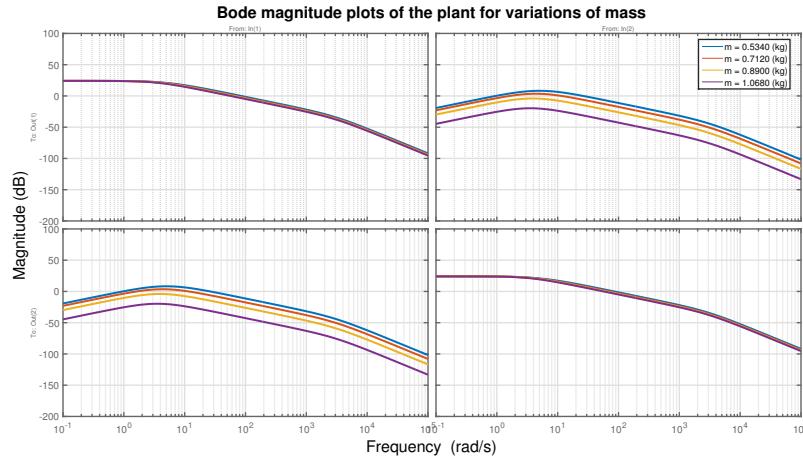


Figure 3.26: Bode Magnitude for Robot (Voltages to Wheel Speeds) - Mass Variations

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.

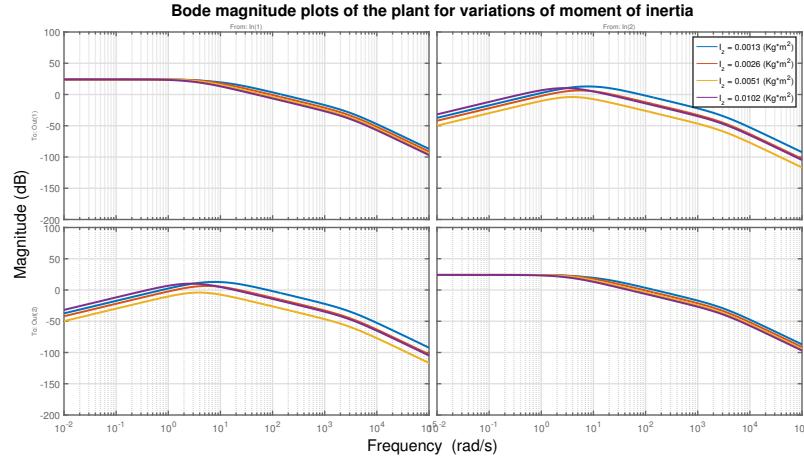


Figure 3.27: Bode Magnitude for Robot (Voltages to Wheel Speeds) - I Variations

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.

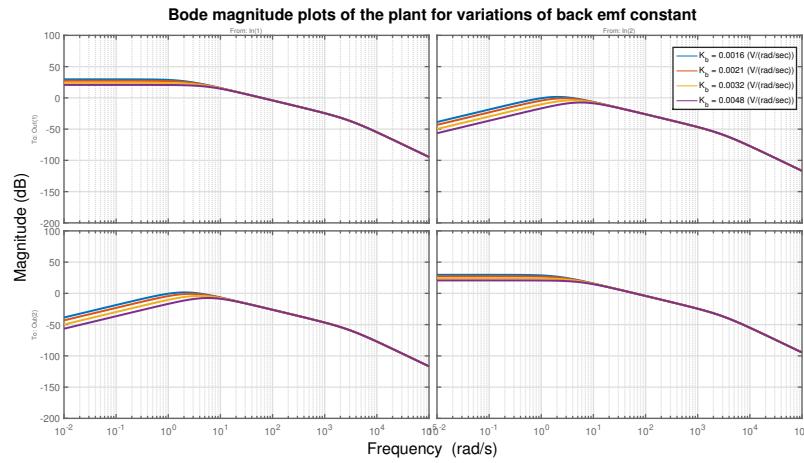


Figure 3.28: Bode Magnitude for Robot (Voltages to Wheel Speeds) - K_b Variations

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.

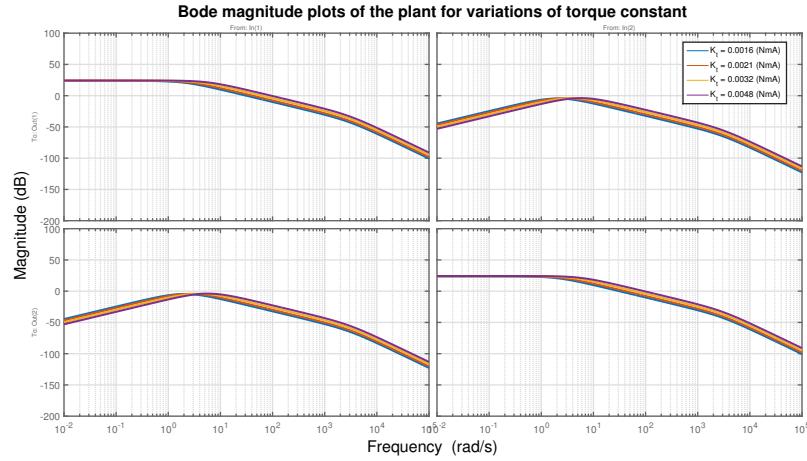


Figure 3.29: Bode Magnitude for Robot (Voltages to Wheel Speeds) - K_t Variations

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.

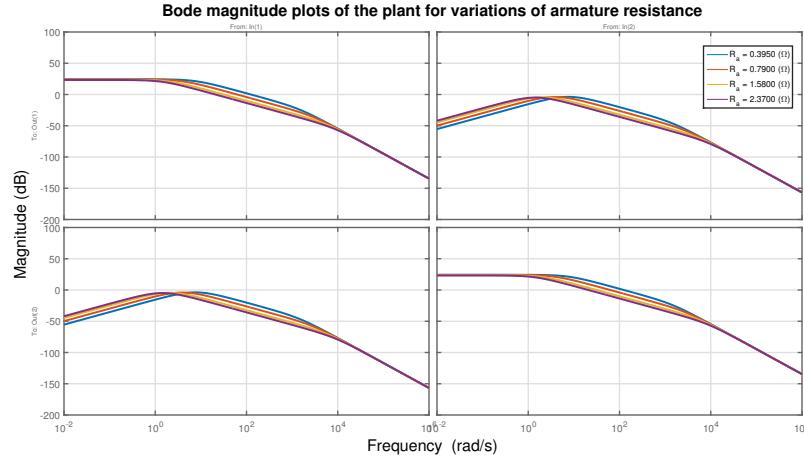


Figure 3.30: Bode Magnitude for Robot (Voltages to Wheel Speeds) - R_a Variations

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.

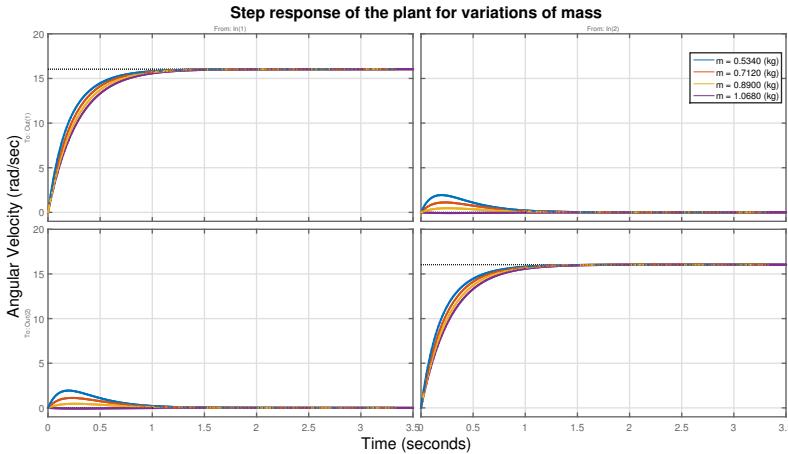


Figure 3.31: System Wheel Angular Velocity Responses to Step Voltages - Mass Variations

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.

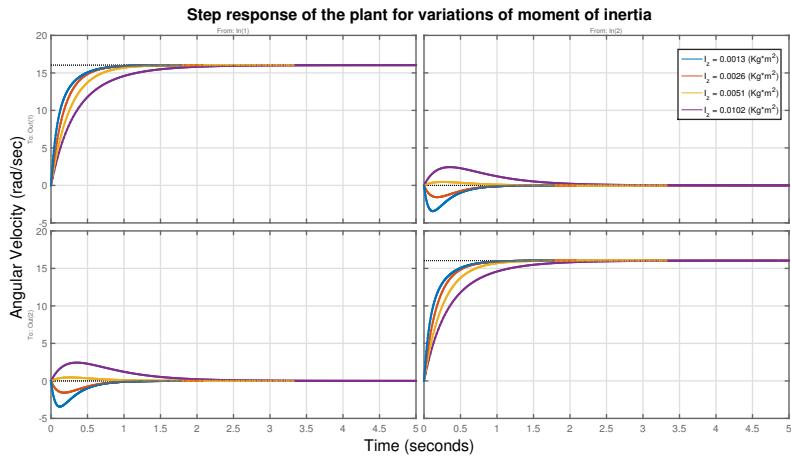


Figure 3.32: System Wheel Angular Velocity Responses to Step Voltages - Moment of Inertia Variations

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.

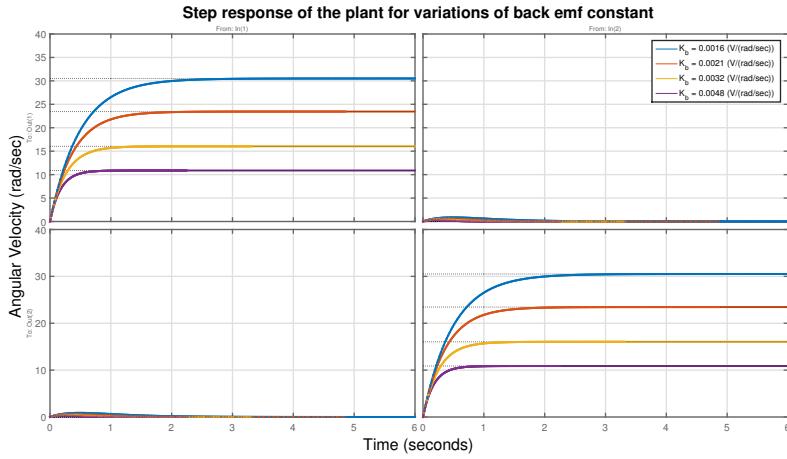


Figure 3.33: System Wheel Angular Velocity Responses to Step Voltages - back EMF Constant Variations

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.

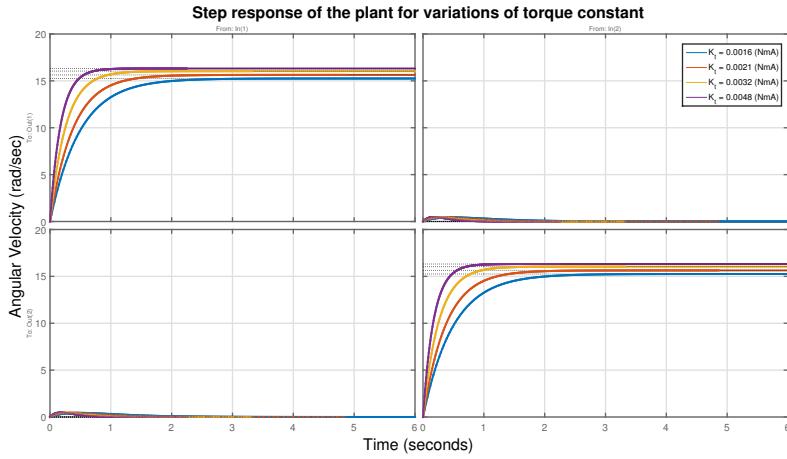


Figure 3.34: System Wheel Angular Velocity Responses to Step Voltages - Torque Constant Variations

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.

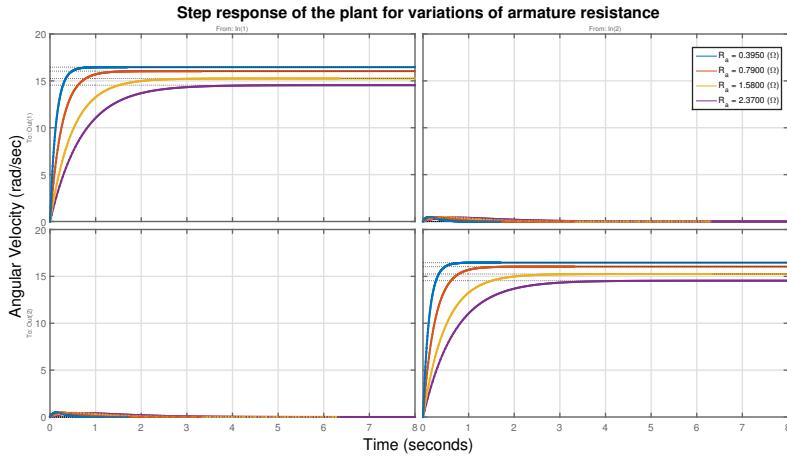


Figure 3.35: System Wheel Angular Velocity Responses to Step Voltages - Armature Resistance Variations

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:

$$\tau_s = K_t K_g i_a \quad (3.52)$$

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 0.6Ω . 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.

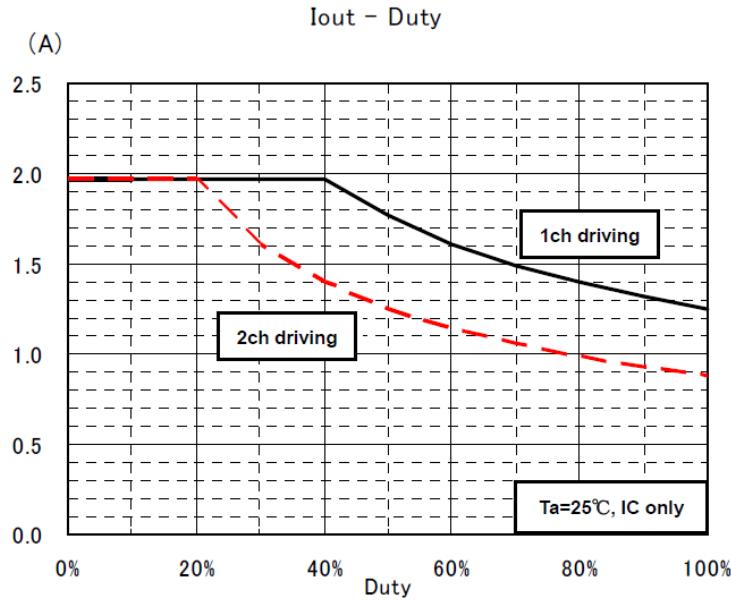


Figure 3.36: Target Characteristics of TB6612FNG

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.

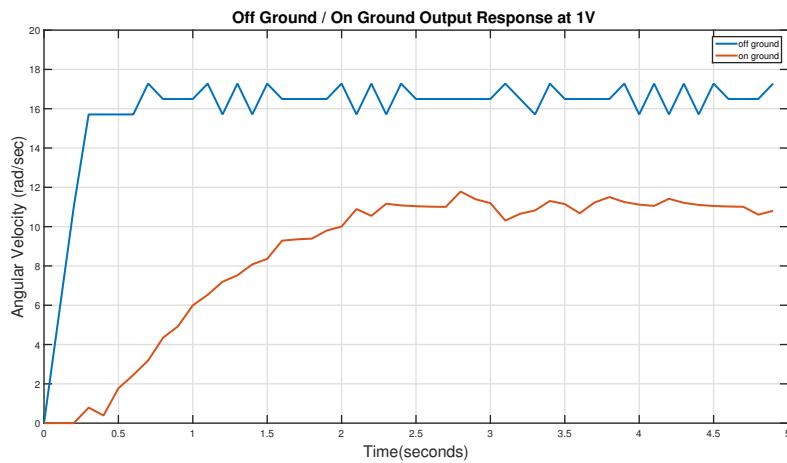


Figure 3.37: Off Ground and On Ground Step Response to 50 PWM Command (1V)

From the comparison, we can clearly see the difference between off ground and on ground in step response, difference occurs in steady state as well as transient state. This somewhat supports 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 voltage² 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 (ω_R, ω_L). We estimated linear translation speed by averaging $v = \frac{(\omega_R, \omega_L)}{2}$. To fully capture the vehicle dynamics, we did these experiments for different voltage levels. One sample for PWM command at 50 (about 1V ideally) is shown in Figure 3.38.

²Here, same voltage actually command by same PWM signal. Also notice that PWM based regulation may not reflect the true voltage as desired.

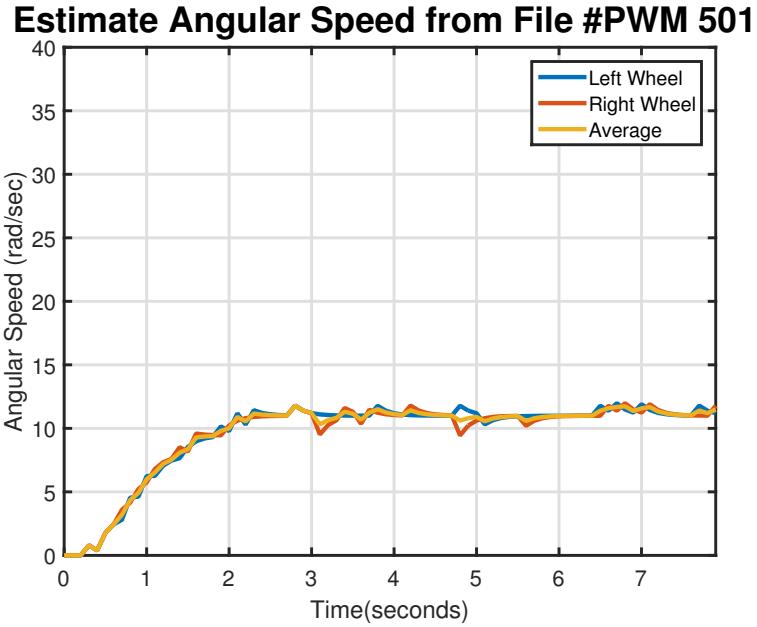


Figure 3.38: Output ω_s Response to PWM 50 Voltage Step Input - Hardware and Decoupled Model

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.

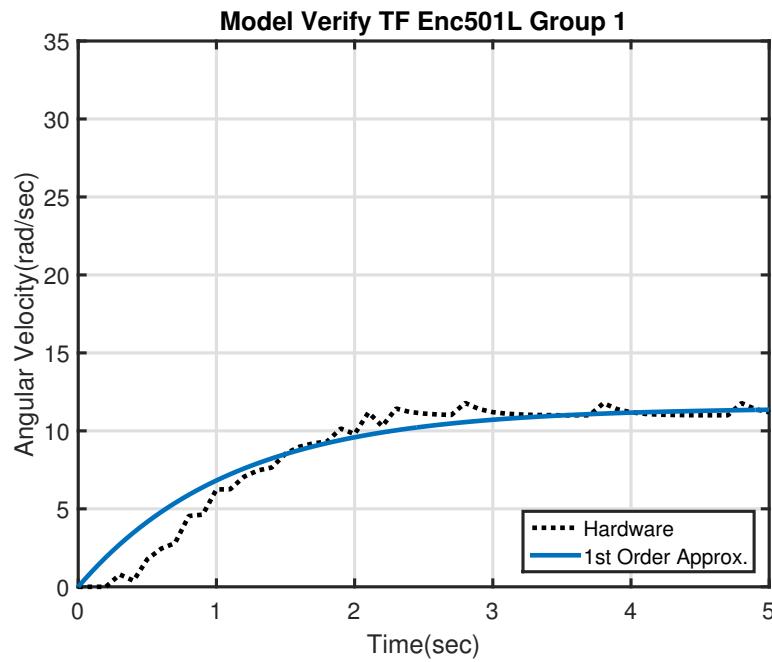


Figure 3.39: On Ground Motor Fitting Left Motor - Hardware and Decoupled Model

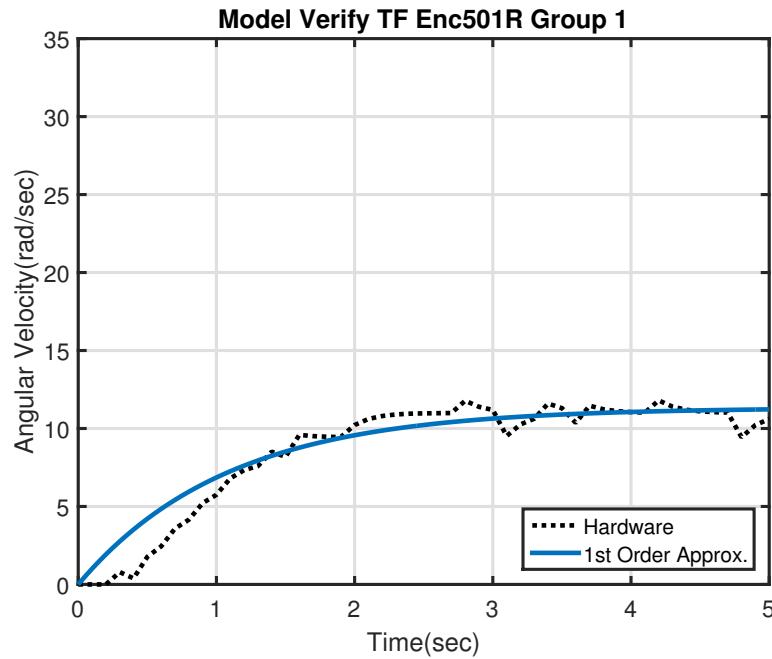


Figure 3.40: On Ground Motor Fitting Right Motor - Hardware and Decoupled Model

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 ω_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

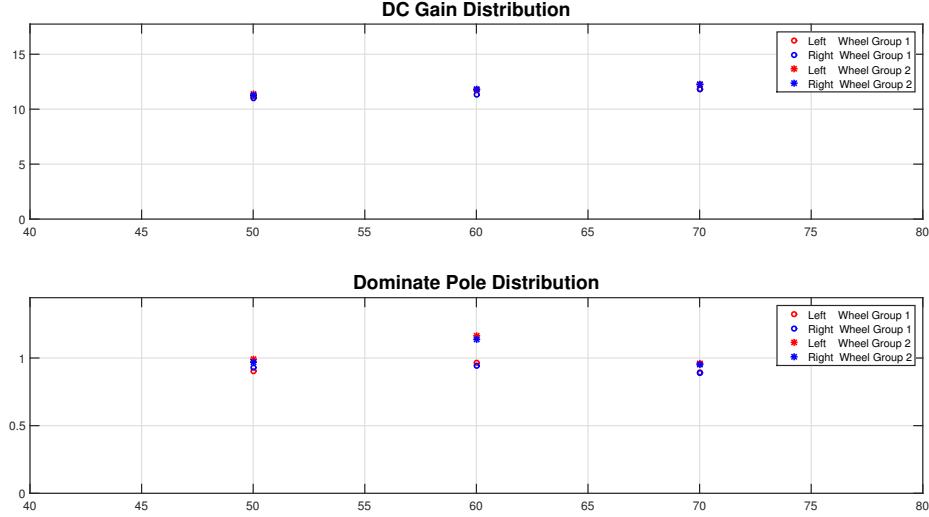


Figure 3.41: On Ground Approximate Transfer Function Fitting Model Distribution of V1

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

$$\frac{\omega_v}{e_a} = 11.637 \frac{0.9745}{s + 0.9745} \quad (3.53)$$

The following is a approximation to the on ground inner-loop (ω_R, ω_L) plant for V1:

$$P_{[\omega_R, \omega_L]} \approx 11.637 \left[\frac{0.9745}{s + 0.9745} \right] \times I_{2 \times 2} \quad (3.54)$$

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.

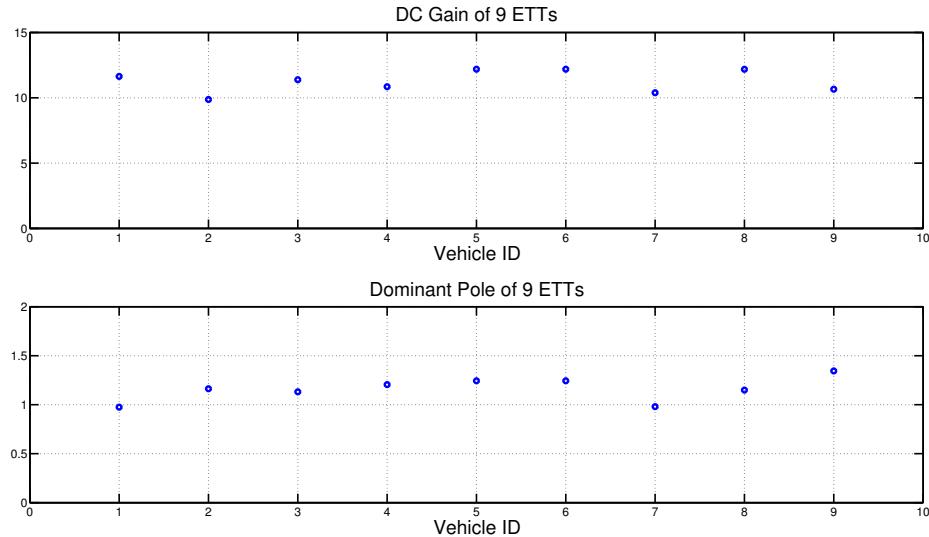


Figure 3.42: On Ground Nominal Model

The following is a approximation to the on ground inner-loop (ω_R, ω_L) plant:

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

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.

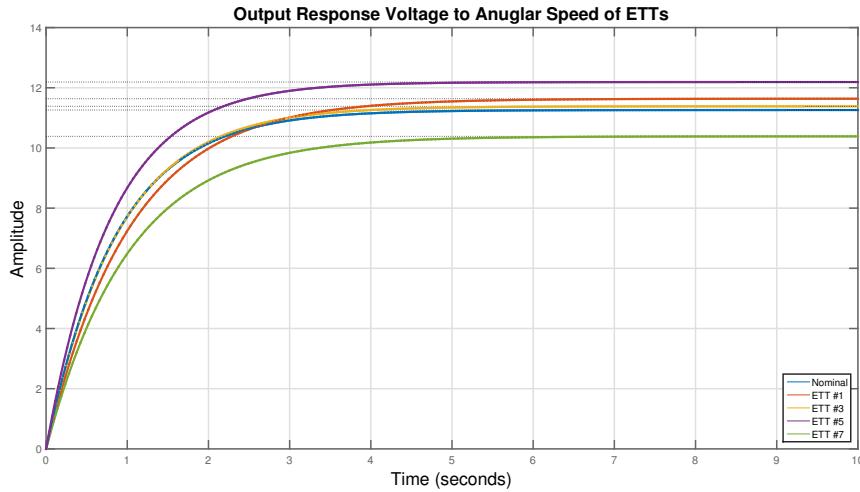


Figure 3.43: Step Response of Nominal Model with other ETT Model

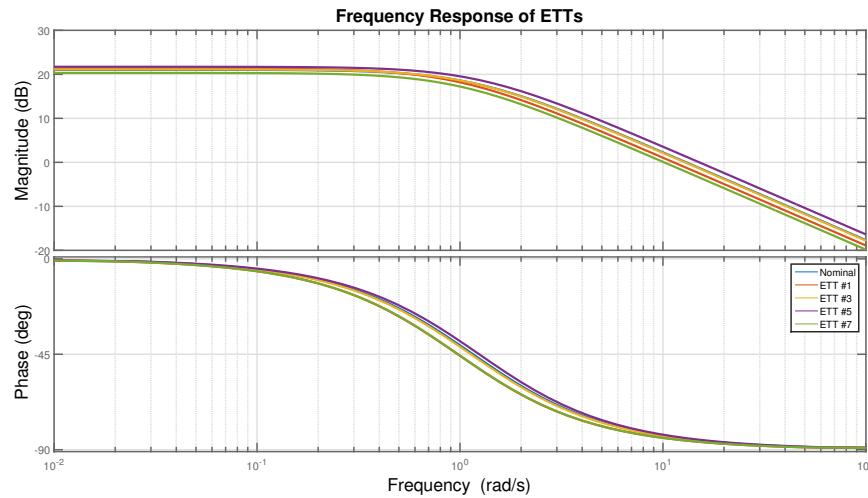


Figure 3.44: On Ground Nominal Model with other ETT Model

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, θ) 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, ω) and (ω_R, ω_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 (ω_R, ω_L) :

$$\omega_r \approx \frac{2\pi n_r}{20T} = 3.142 n_r \text{ (rad/sec)} \quad \omega_l \approx \frac{2\pi n_l}{20T} = 3.142 n_l \text{ (rad/sec)} \quad (4.1)$$

- $T = 0.1$ 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 rotation¹),
- 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, ω) are then estimated from the above (ω_r, ω_l) estimates as follows:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = M \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \quad M = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d_w} & -\frac{r}{d_w} \end{bmatrix} \quad M^{-1} = \begin{bmatrix} \frac{1}{r} & \frac{d_w}{2r} \\ \frac{1}{r} & \frac{-d_w}{2r} \end{bmatrix} \quad (4.2)$$

and

$$v = \left(\frac{r(\omega_R + \omega_L)}{2} \right) \approx 0.1571 \left(\frac{n_r + n_l}{2} \right) \text{ m/sec} \quad (4.3)$$

$$\omega = \left(\frac{r(\omega_R - \omega_L)}{d_w} \right) \approx 1.122 (n_r - n_l) \text{ rad/sec} \quad (4.4)$$

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.)

¹This 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

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

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

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

Doing so will yield, assuming negligible impact of roll off, a settling time near 2 second and essentially no overshoot ($100e^{\frac{-\zeta p_i}{\sqrt{1-\zeta^2}}} \approx 0.15\%$). 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 from commanded wheel speed to actual wheel speed:

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

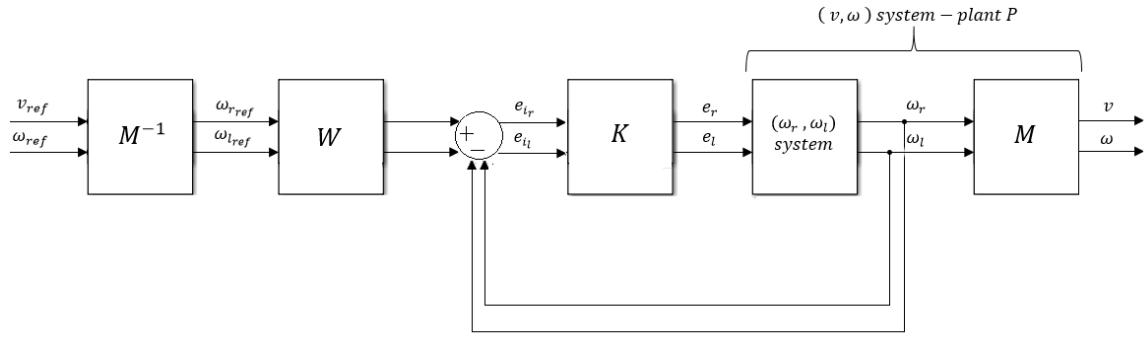


Figure 4.1: Visualization of (v, ω) and (ω_R, ω_L) Inner-Loop Control

The inner-loop control system can be visualized as shown in Figure 4.1. (v, ω) are commanded but not directly measured. Within Figure 4.1, the matrix M is a 2×2 vehicle-wheel speed map that relates the vehicle translational-rotational velocities (v, ω) to the wheel angular velocities (ω_R, ω_L) ; i.e. see equation (4.2) (page 94). Although only the wheel encoder count information is fed back within the physical inner-loop hardware implementation, the system outputs v and ω were estimated (computed) using wheel encoder counts in accordance with equations (4.3)-(4.4)

Reference to Output T_{ry} (v, ω) 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}, ω_{ref}) to outputs (v, ω) . Doing so yields the following TITO LTI

(nearly decoupled at low frequencies) map:

$$T_{ry} = MPK(I + PK)^{-1}WM^{-1} \approx \left[\frac{7.716}{s^2 + 5s + 7.716} \right] I_{2 \times 2} \quad (4.13)$$

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

Inner-Loop Open Loop Singular Values: (v, ω) and (ω_R, ω_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)).

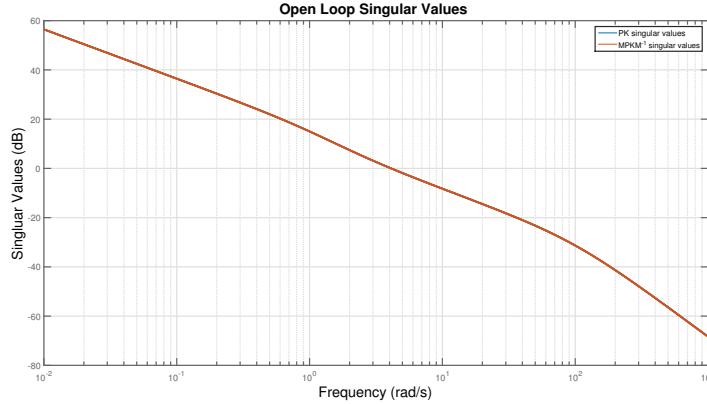


Figure 4.2: PK and $L_o = MPKM^{-1}$ Singular Values

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, ω) or (ω_R, ω_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. (ω_R, ω_L) and (v, ω)). This implies that

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

for both closed loop systems; (i.e. (ω_R, ω_L) and (v, ω)). Recall that:

- For the (ω_R, ω_L) system, $L_o = PK = KP = L_i$.
- For the (v, ω) system, $L_o = MPKM^{-1}$ and $L_i = KM^{-1}MP = PK$.

For either system, $\sigma_k[L_o] = \sigma_k[L_i]$ ($k = 1, 2 \dots$). While obvious for the first (ω_R, ω_L) system, the proof for the second (v, ω) system requires using the structure of M (see equation (4.2) on page 94) 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 (ω_R, ω_L)) are plotted in Figure 4.3. As expected, the singular values are identical for the two systems: (ω_R, ω_L) and (v, ω) . 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 (ω_R, ω_L) and the (v, ω) closed loop systems.

Complementary Sensitivity Singular Values (Decoupled Model). The complementary sensitivity singular values (at outputs/controls) for system (ω_R, ω_L) are

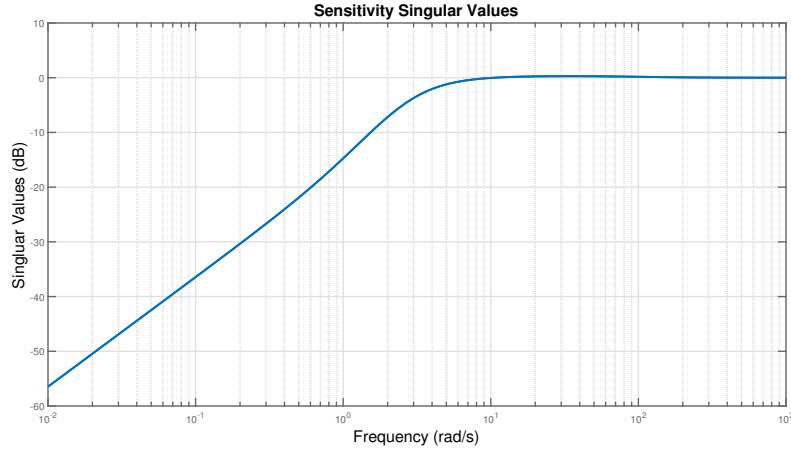


Figure 4.3: $S_o = (I + L_o)^{-1} = S_i$ Singular Values - Using Decoupled Model

plotted in Figure 4.4. As expected, the singular values are identical for the two systems: (ω_R, ω_L) and (v, ω) .

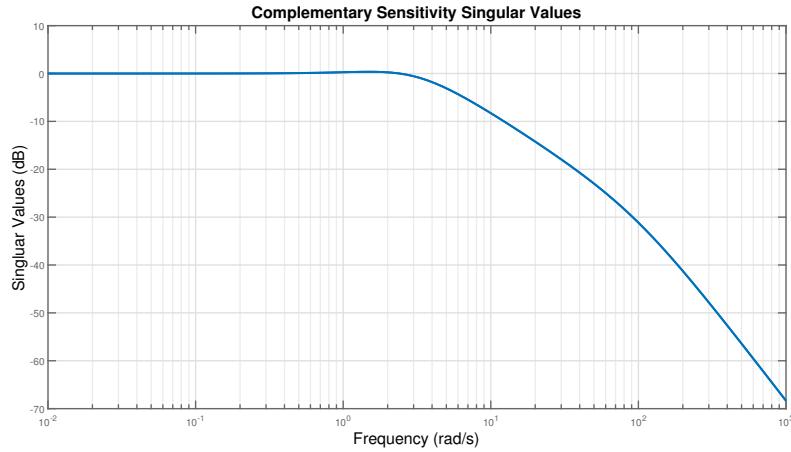


Figure 4.4: $T_o = L_o(I + L_o)^{-1} = T_i$ Singular Values - Using Decoupled Model

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.

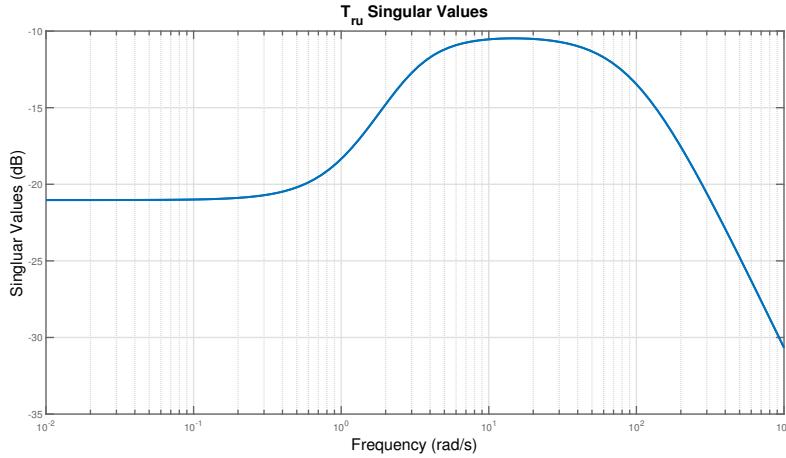


Figure 4.5: T_{ru} Singular Values (No Pre-filter) - Using Decoupled Model

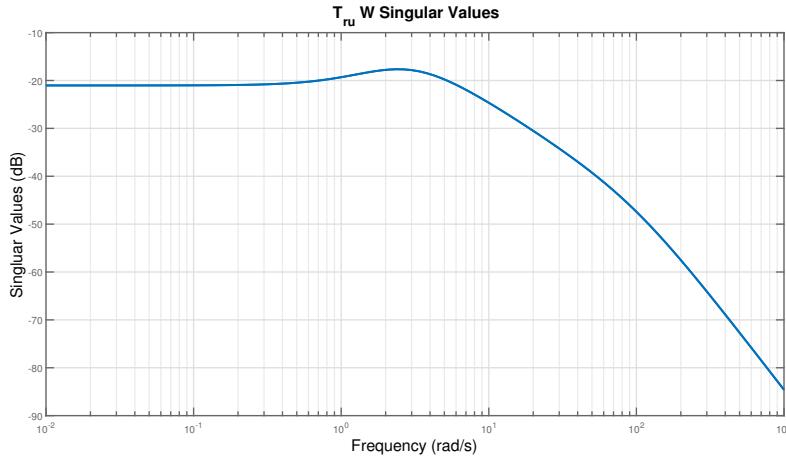


Figure 4.6: $T_{ru}W$ Singular Values (with Pre-filter) - Using Decoupled Model

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, ω) system.

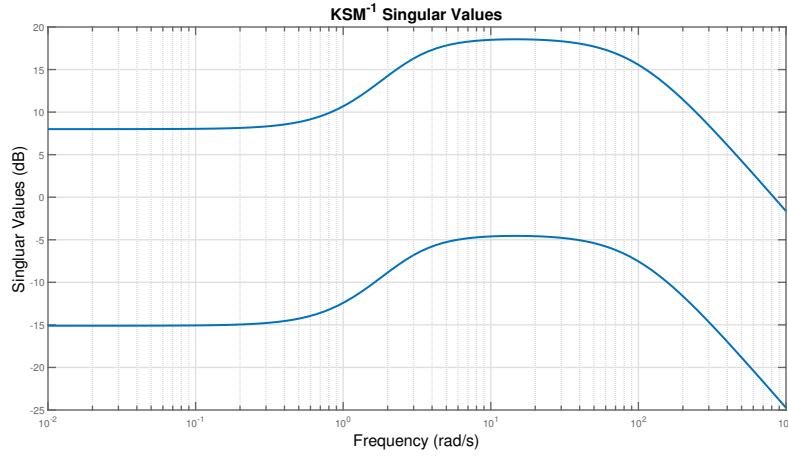


Figure 4.7: KSM^{-1} Singular Values

Input Disturbance to Output $T_{d_{iy}}$ 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).

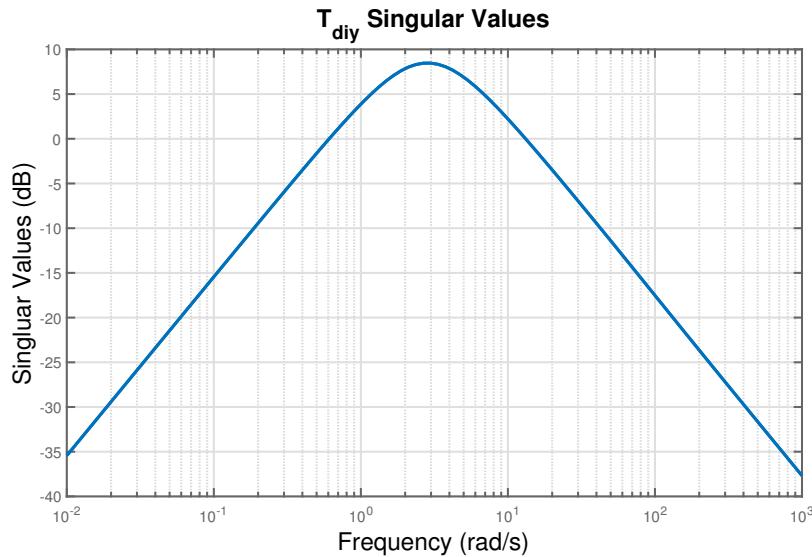


Figure 4.8: $T_{d_{iy}}$ Singular Values - Using Decoupled Model

Figure 4.9 shows the singular values for $T_{d_{iy}}$ for the (v, ω) system.

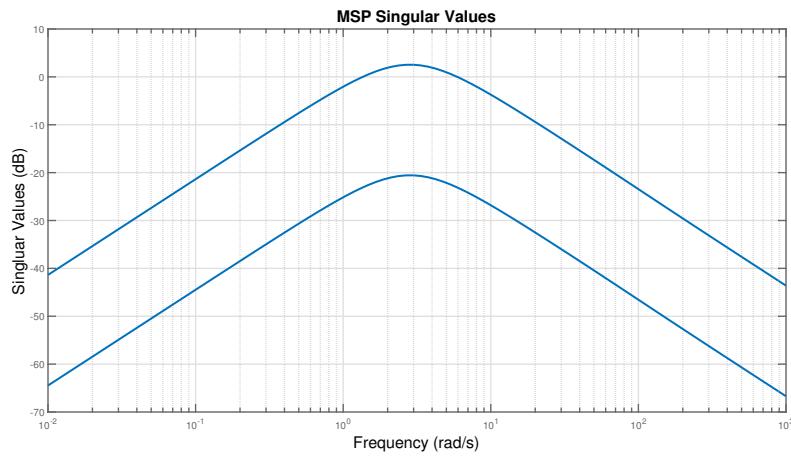


Figure 4.9: *MSP* Singular Values

Step Response Analysis Using Decoupled Model: Output Responses (ω_R, ω_L).

filtered step reference time responses is shown within Figure 4.10.

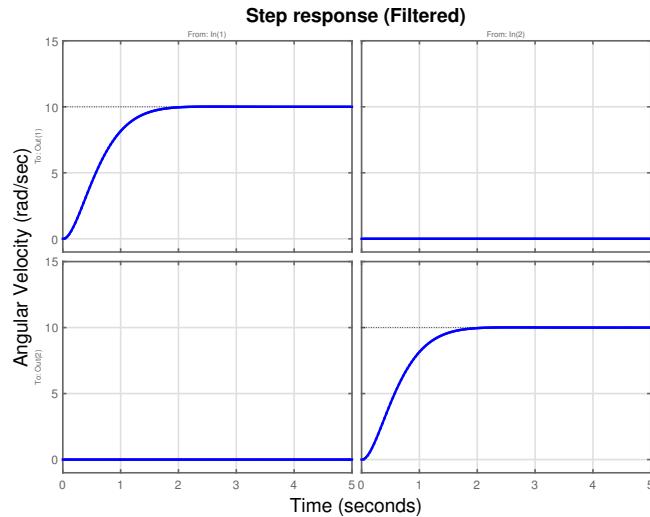


Figure 4.10: Inner-Loop $[\omega_R, \omega_L]$ Filtered Step Response - Using Decoupled Model

The associated decoupled unfiltered step reference time responses is shown within Figure 4.11.

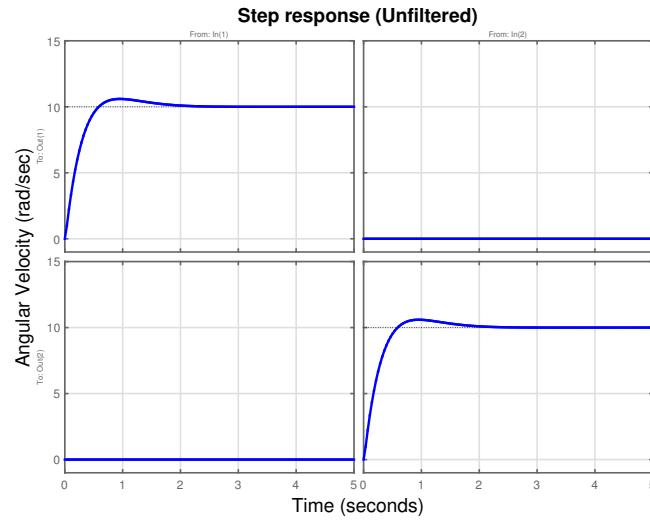


Figure 4.11: Inner-Loop $[\omega_R, \omega_L]$ Unfiltered Step Response - Using Decoupled Model

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.

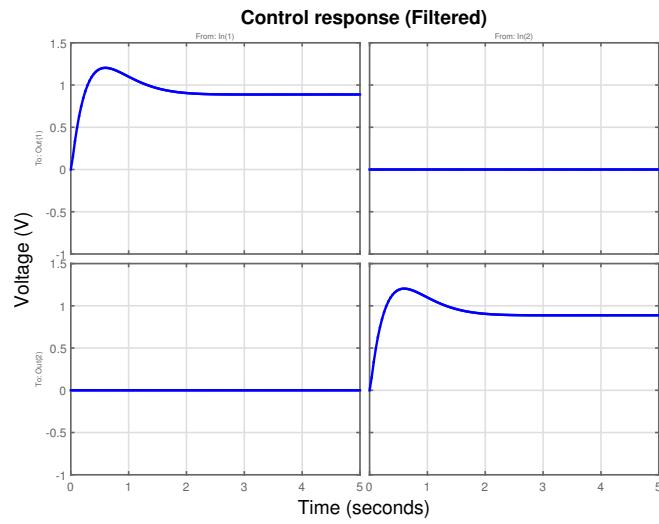


Figure 4.12: Control Response to Step Command (with Pre-filter) - with Decoupled Model

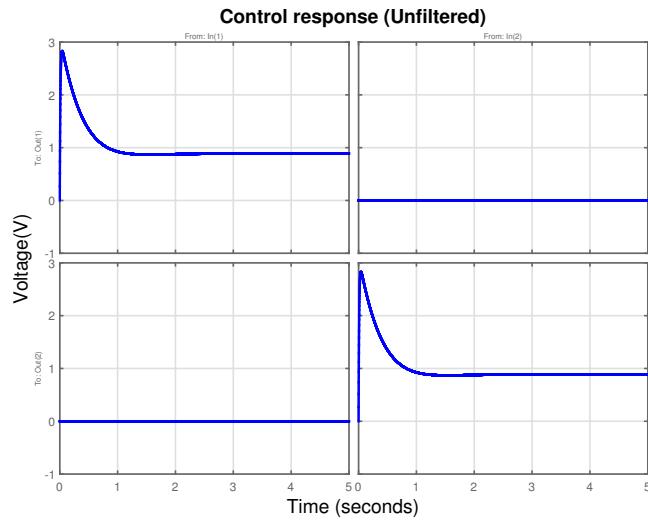


Figure 4.13: Control Response to Unfiltered Step Command - with Decoupled Model

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 (ω_R, ω_L) system and not the (v, ω) system. When the (v, ω) 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.

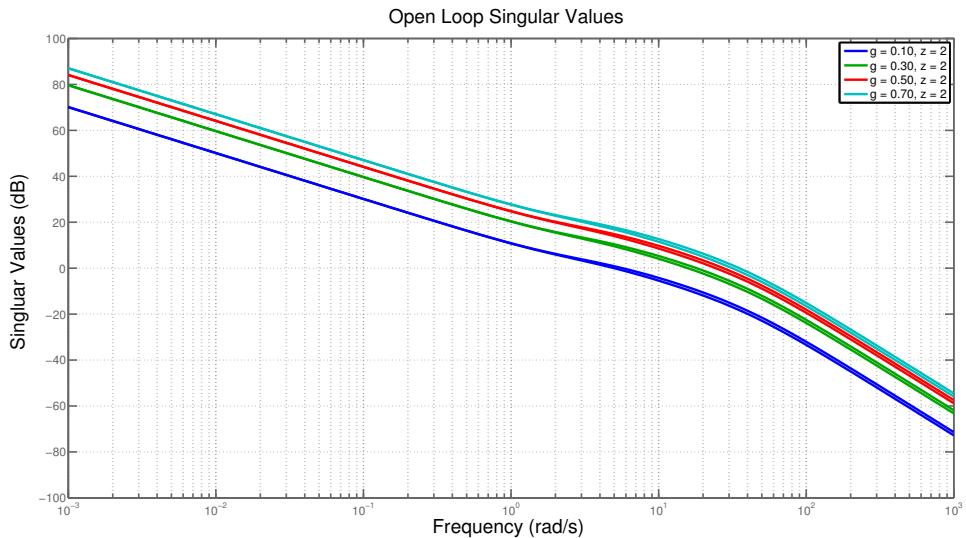


Figure 4.14: Singular Values for L ($g=0.10, 0.30, 0.50, 0.70; z=2$)

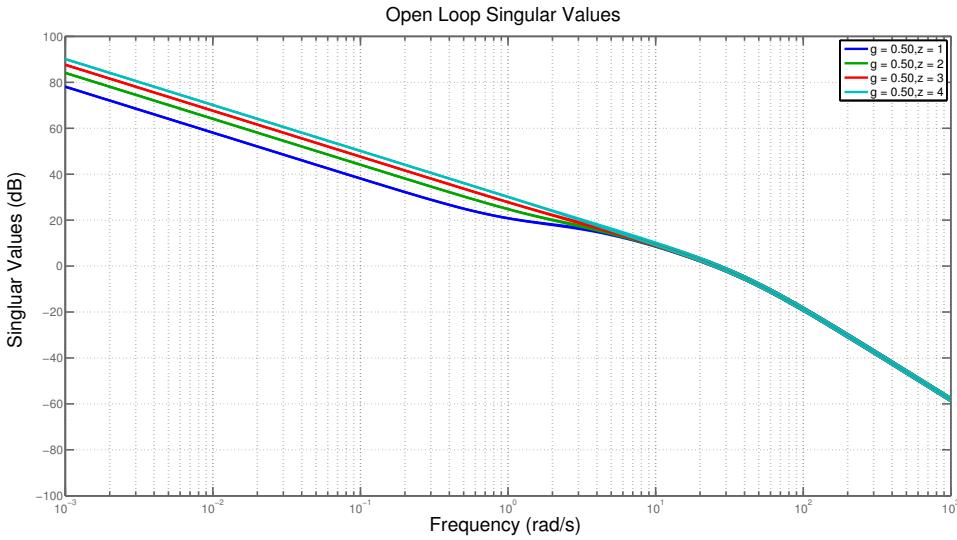


Figure 4.15: Singular Values for L ($g=0.50$; $z=1, 2, 3, 4$)

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.

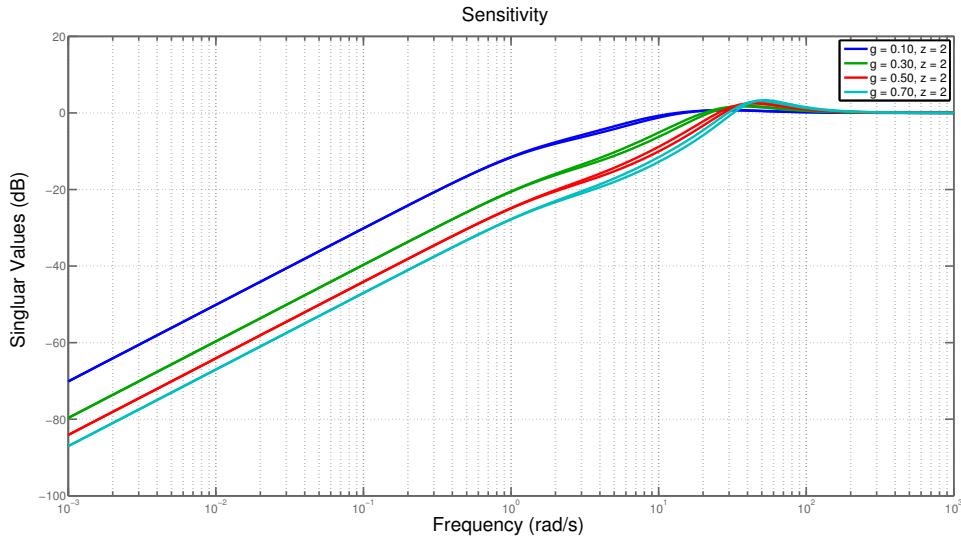


Figure 4.16: Singular Values for S ($g=0.10, 0.30, 0.50, 0.70; z=2$)

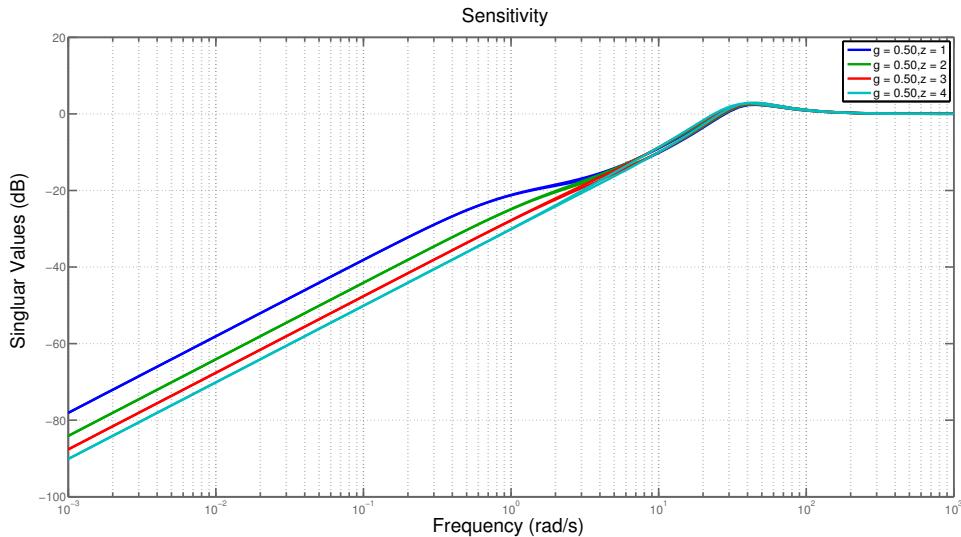


Figure 4.17: Singular Values for S ($g=0.50; z=1, 2, 3, 4$)

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.

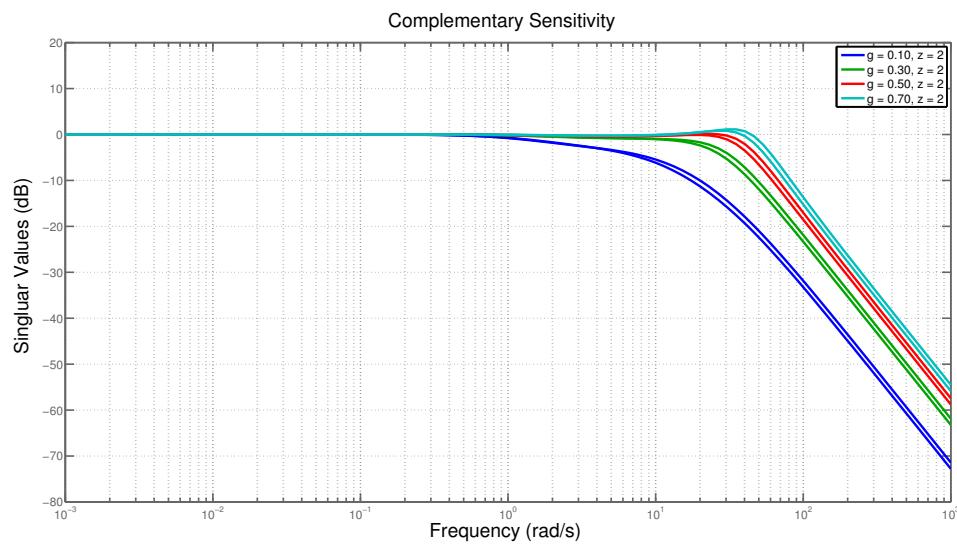


Figure 4.18: Singular Values for T ($g=0.10, 0.30, 0.50, 0.70; z=2$)

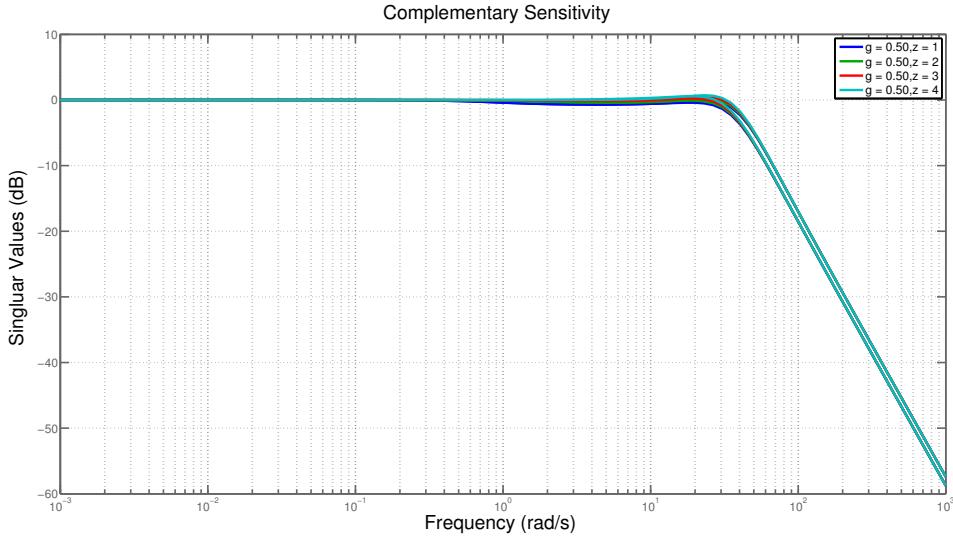


Figure 4.19: Singular Values for T ($g=0.50$; $z=1, 2, 3, 4$)

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 (ω_R, ω_L) system. As such, they tell us what control responses result from (ω_R, ω_L) commands - not from (v, ω) commands. This is addressed below.

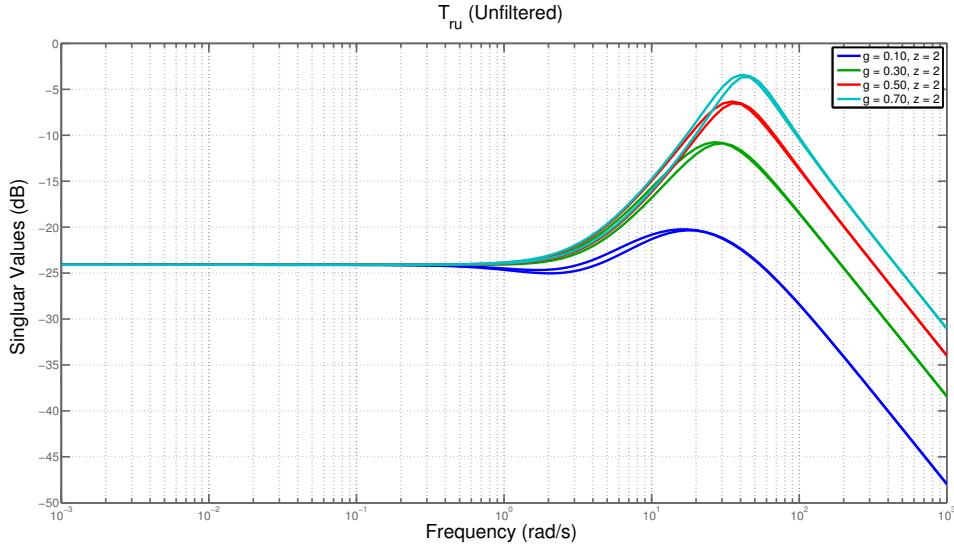


Figure 4.20: Singular Values for T_{ru} ($g=0.10, 0.30, 0.50, 0.70$; $z=2$) - (ω_R, ω_L) Commands

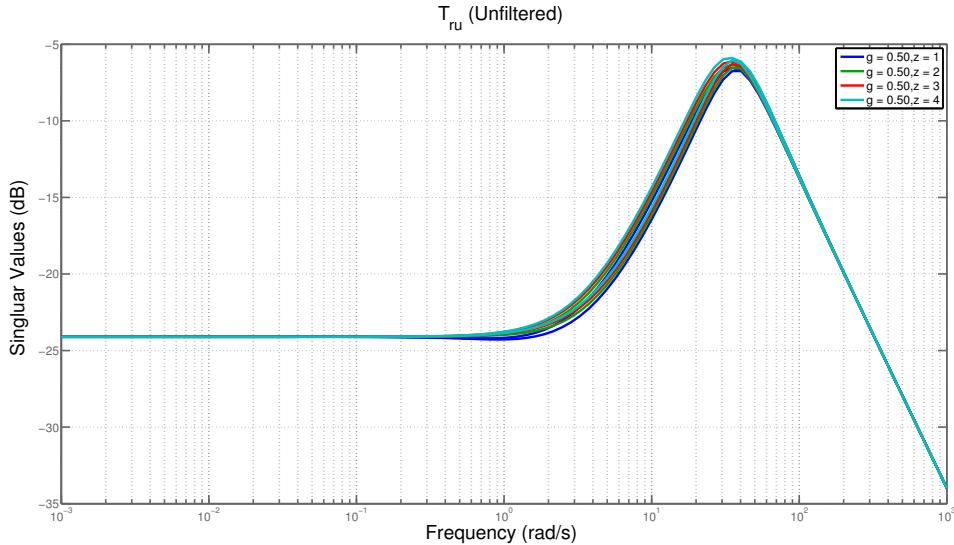


Figure 4.21: Singular Values for T_{ru} ($g=0.50$; $z=1, 2, 3, 4$)- (ω_R, ω_L) Commands

From Figures 4.20 -4.21, we make the following observations when (ω_R, ω_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, ω) commands - as we shall be giving in a practical (v, ω) inner-loop control implementation (see Chapter 4).

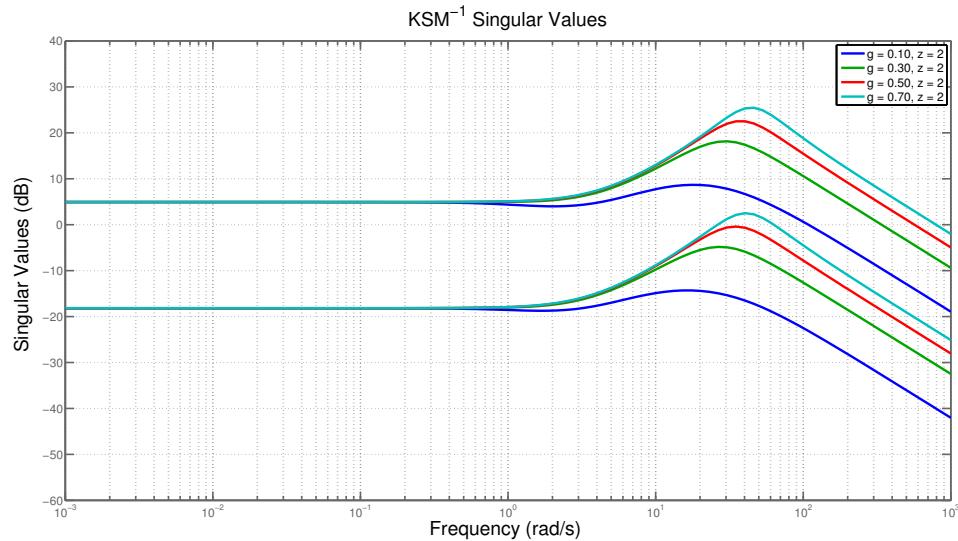


Figure 4.22: Singular Values for KSM^{-1} ($g=0.10, 0.30, 0.50, 0.70; z=2$) - (v, ω) Commands

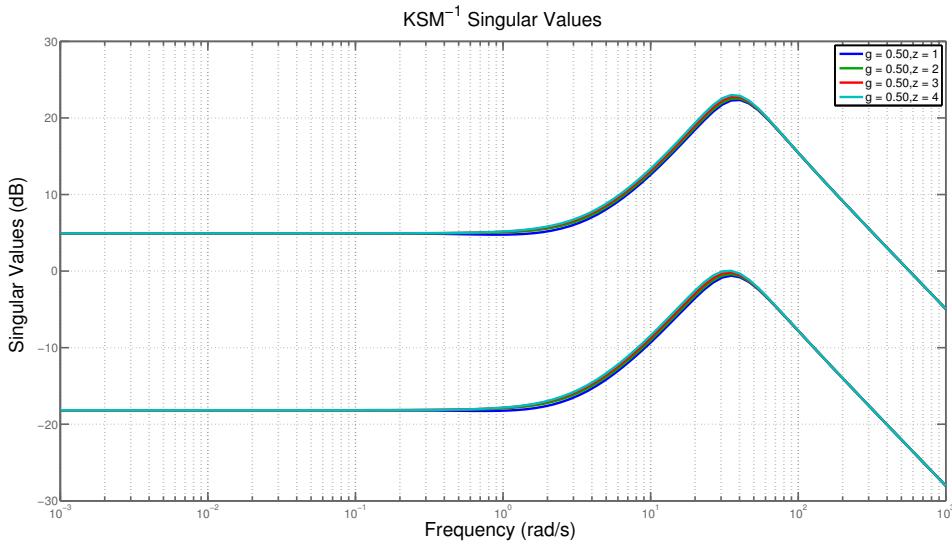


Figure 4.23: Singular Values for KSM^{-1} ($g=0.50$; $z=1, 2, 3, 4$)- (v, ω) Commands

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

- peak controls will generally be larger for (v, ω) commands versus (ω_R, ω_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, ω) 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 $W = \frac{z}{s+z}$ 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 (ω_R, ω_L) system. As such, they tell us what control responses result from (ω_R, ω_L) commands. They do not tell us about control responses from (v, ω) commands. This is addressed below.

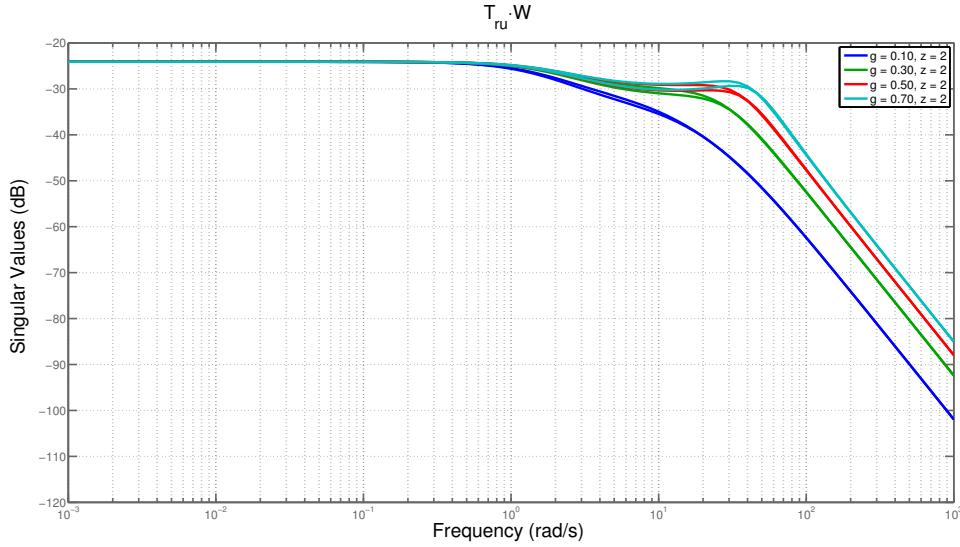


Figure 4.24: Singular Values for $W \cdot T_{ru}$ ($g=0.10, 0.30, 0.50, 0.70$; $z=2$) - (ω_R, ω_L) Commands

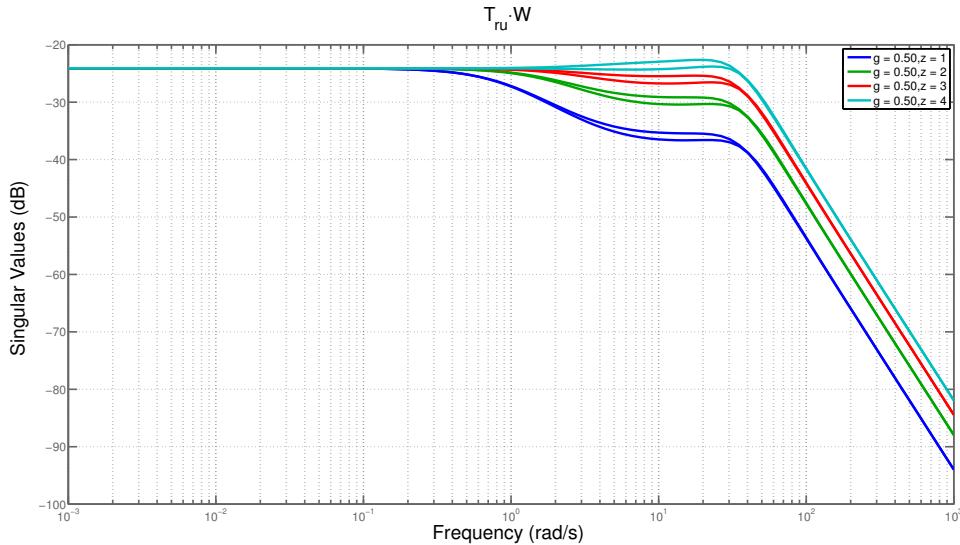


Figure 4.25: Singular Values for $W \cdot T_{ru}$ ($g=0.50$; $z=1, 2, 3, 4$) - (ω_R, ω_L) Commands

From Figures 4.24-4.25, we make the following observations when (ω_R, ω_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 (ω_R, ω_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 (ω_R, ω_L) system. As such, they tell us what (ω_R, ω_L) responses result from input (voltage) disturbances. They do not tell us about (v, ω) 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:

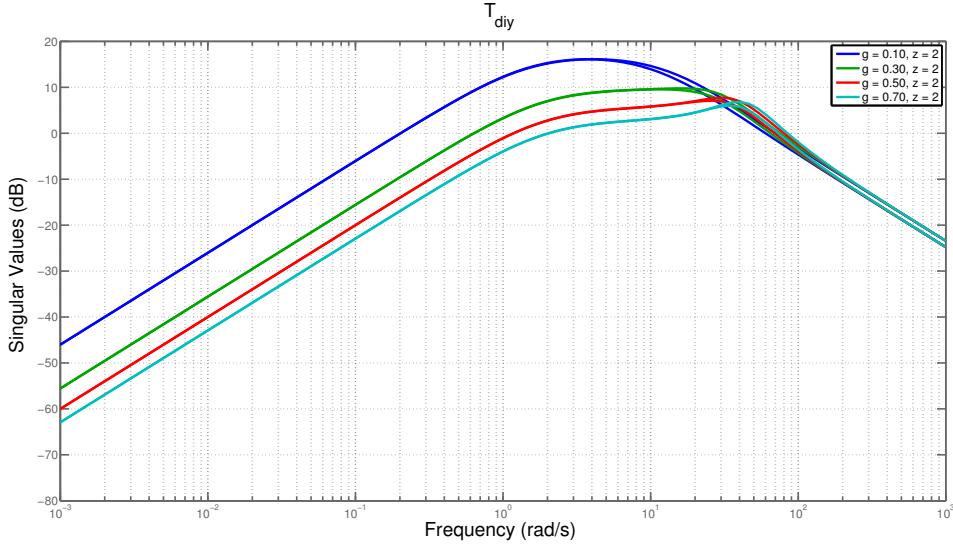


Figure 4.26: Singular Values for T_{dy} ($g=0.10, 0.30, 0.50, 0.70; z=2$) - (ω_R, ω_L) Responses

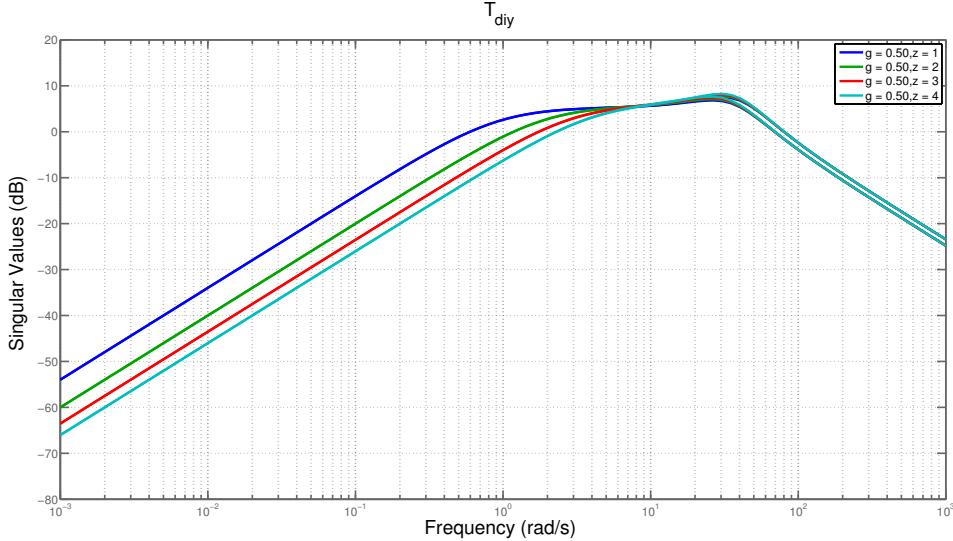


Figure 4.27: Singular Values for T_{dy} ($g=0.50; z=1, 2, 3, 4$) - (ω_R, ω_L) Responses

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

- peak T_{dy} decreases with increasing g (z has little impact on peak)

- increasing g reduces $T_{d_i y}$ at all frequencies except at high frequencies
- increasing z reduces $T_{d_i y}$ at low frequencies
- frequency at which peak $T_{d_i y}$ 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, ω) responses to input (voltage) disturbances d_i .

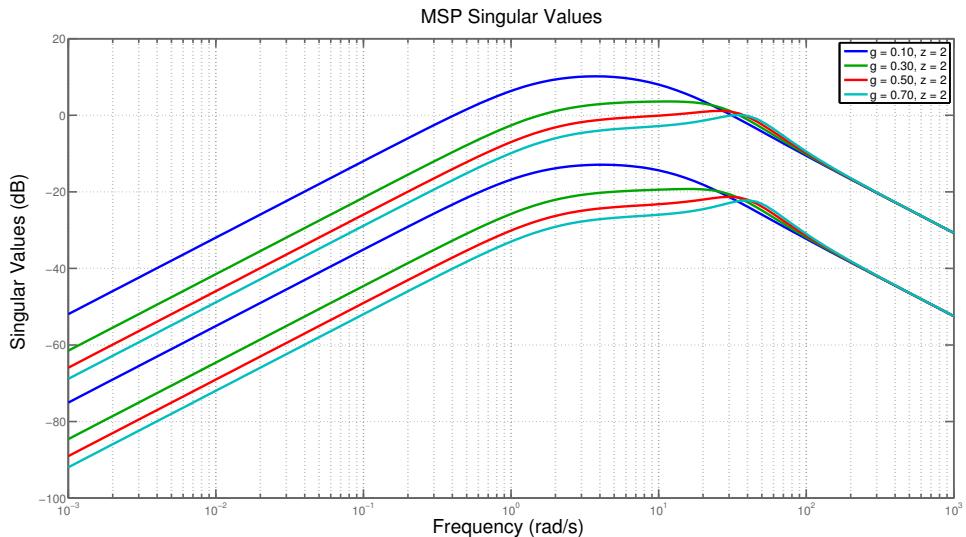


Figure 4.28: Singular Values for MSP ($g=0.50$; $z=1, 2, 3, 4$) - (v, ω) Responses

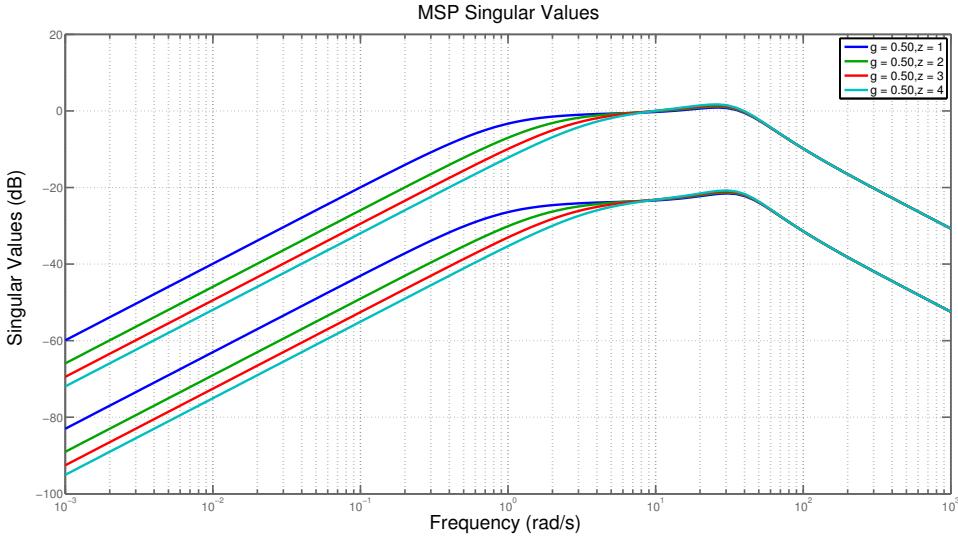


Figure 4.29: Singular Values for MSP ($g=0.50$; $z=1, 2, 3, 4$) - (v, ω) Responses

From Figures 4.28-4.29, we make the following observations about the closed loop system containing the actual plant P :

- peak $T_{d_{iy}}$ has improved for actual (v, ω) plant versus (ω_R, ω_L) system (see Figures 4.26 -4.27)
- peak $T_{d_{iy}}$ 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)

The above patterns suggest that while the actual (v, ω) plant will exhibit a worse T_{ry} , it will exhibit a better $T_{d_{iy}}$ than the (ω_R, ω_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.

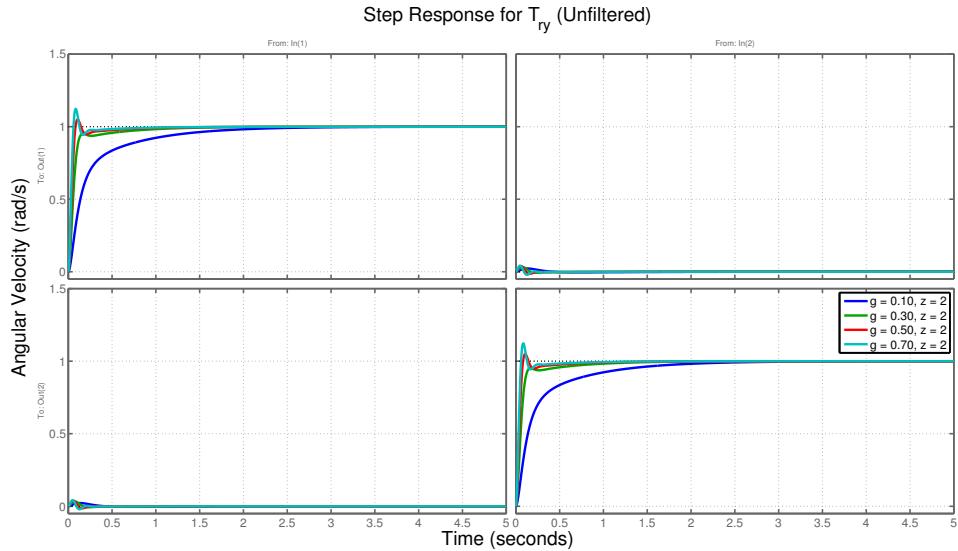


Figure 4.30: Output Response to Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)

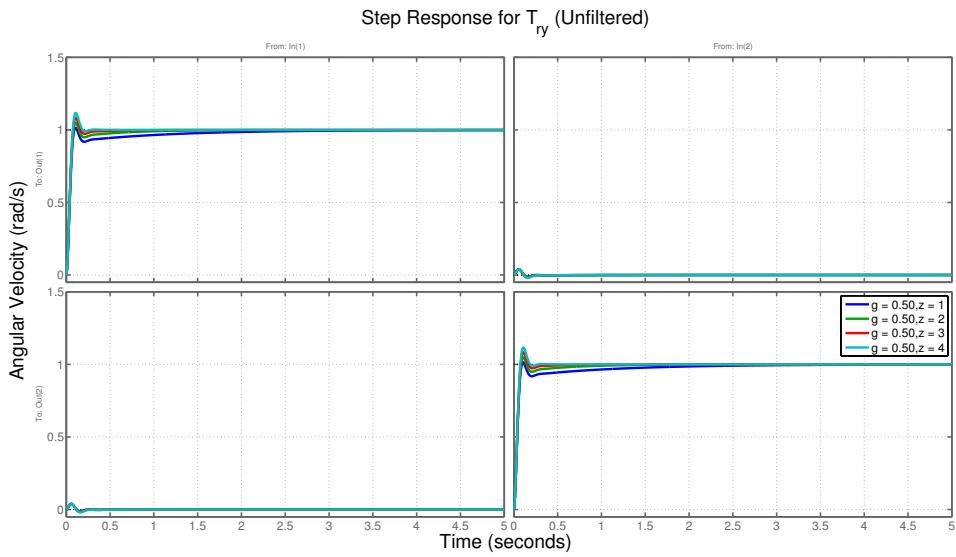


Figure 4.31: Output Response to Step Command ($g = 0.50; z = 1, 2, 3, 4$)

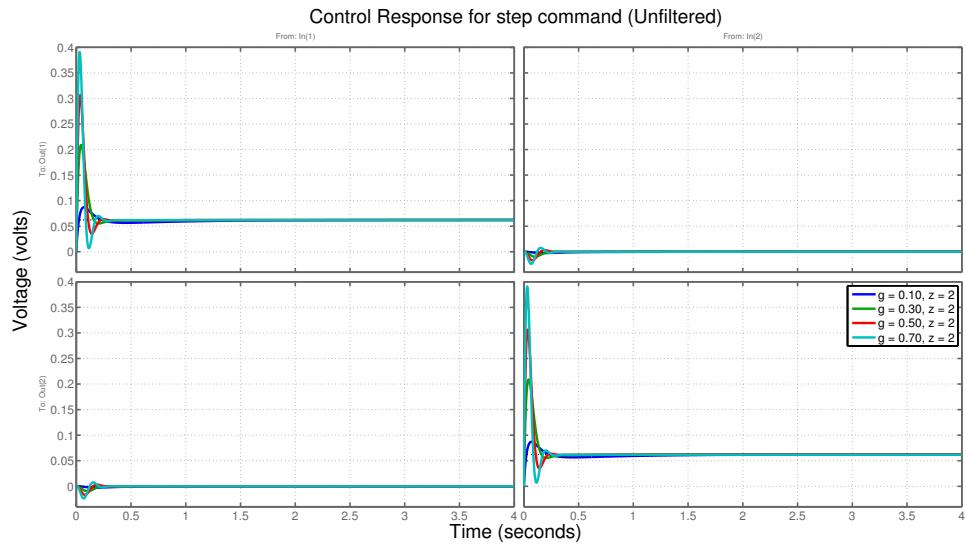


Figure 4.32: Control Response to Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)

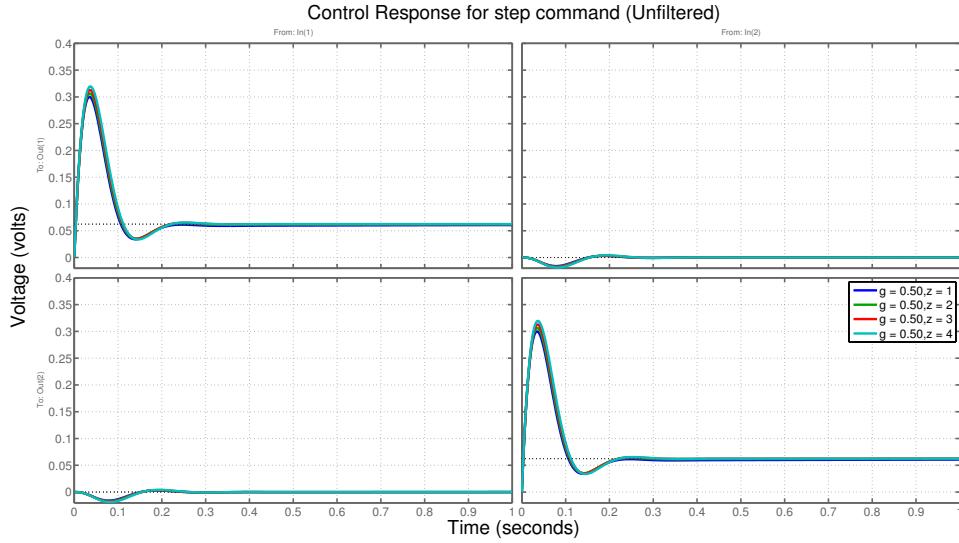


Figure 4.33: Control Response to Step Command ($g = 0.50; z = 1, 2, 3, 4$)

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.

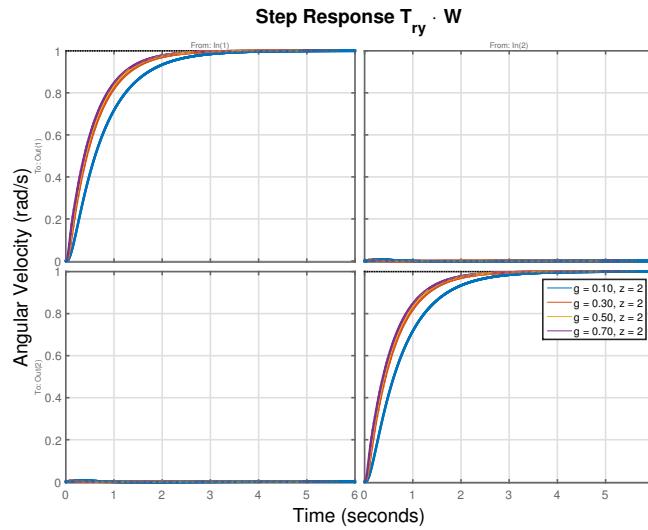


Figure 4.34: Output Response to Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)

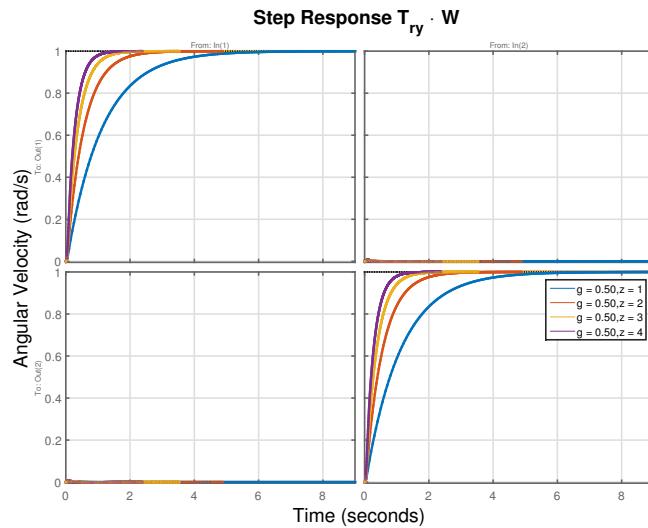


Figure 4.35: Output Response to Step Command ($g = 0.50; z = 1, 2, 3, 4$)

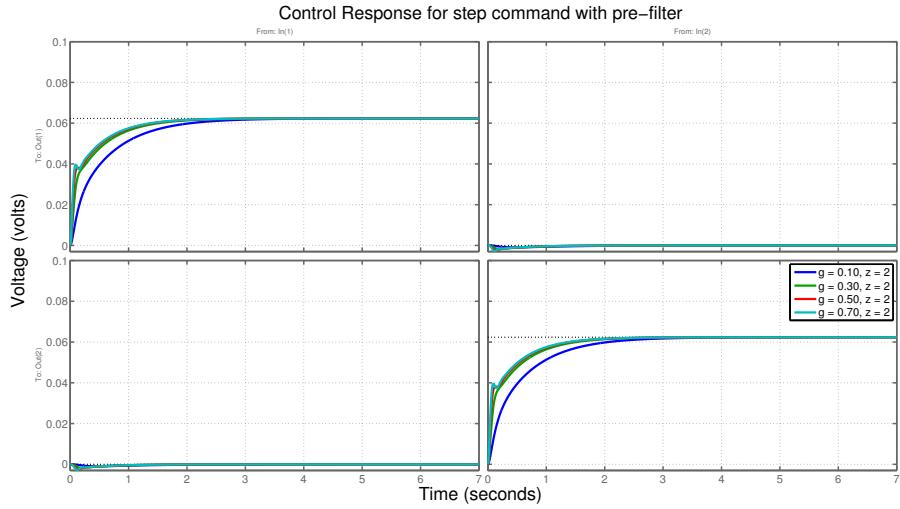


Figure 4.36: Control Response to Filtered Step Command ($g = 0.10, 0.30, 0.50, 0.70; z = 2$)

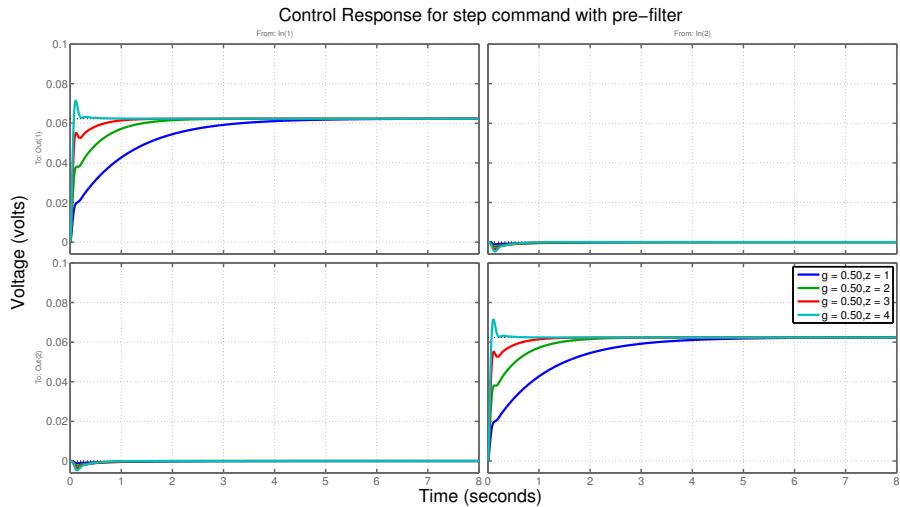


Figure 4.37: Control Response to Filtered Step Command ($g = 0.50; z = 1, 2, 3, 4$)

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.

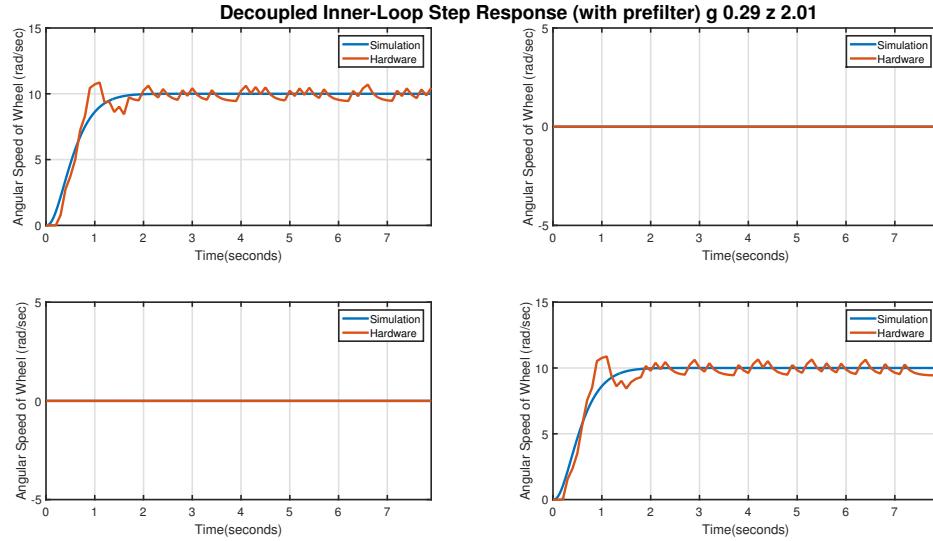


Figure 4.38: Output Response to filtered Step Command ($\omega R_{ref} = 10$, $\omega L_{ref} = 10$)

For completeness, output response to (v, ω) system is shown in Figure 4.39

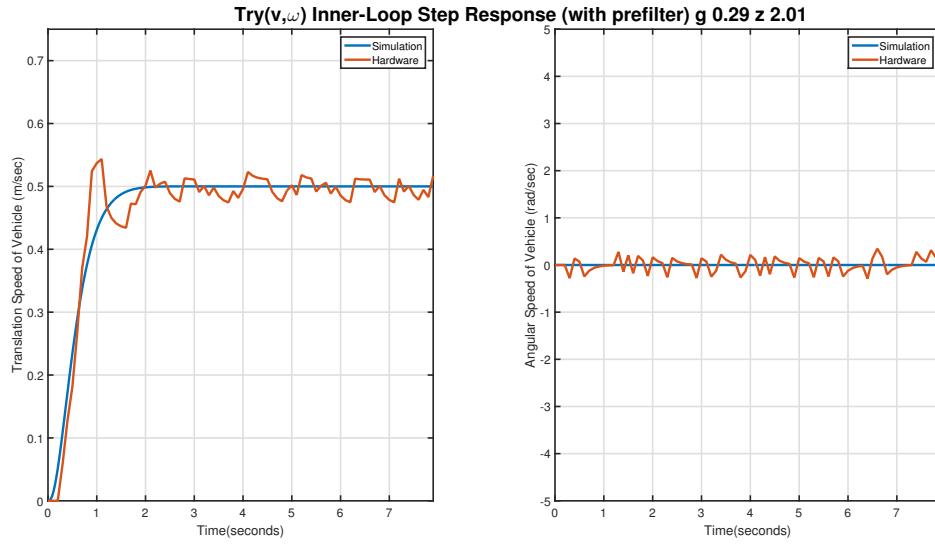


Figure 4.39: Output Response to filtered Step Command ($v_{ref} = 0.5$, $\omega_{ref} = 0$)

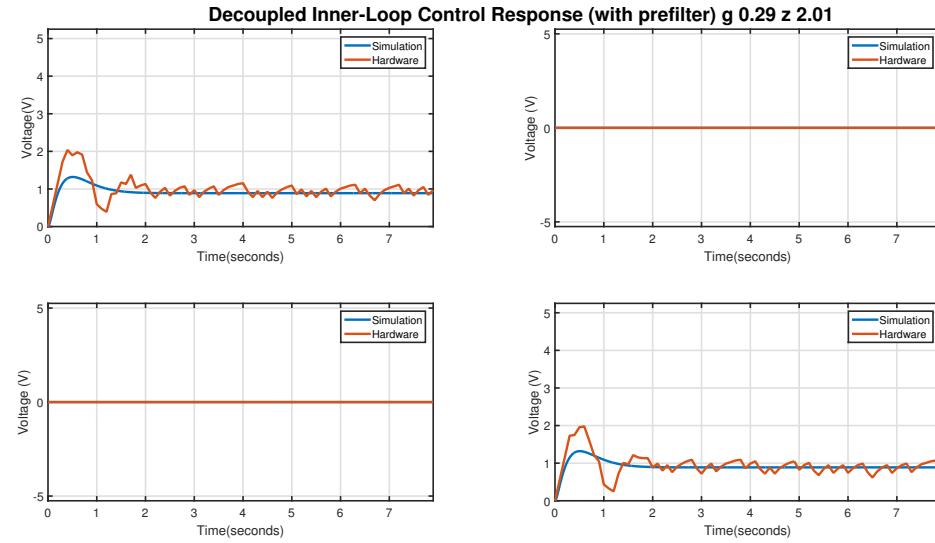


Figure 4.40: Control Response to filtered Step Command ($\omega R_{ref} = 10$, $\omega L_{ref} = 10$)

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, θ) Cruise Control Along Line - Design and Implementation

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

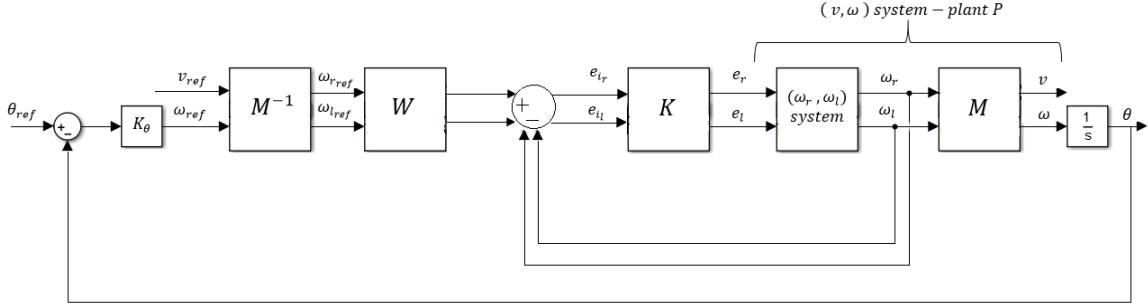


Figure 4.41: Visualization of Cruise Control Along a Line

v is calculated based on wheel encoders; see equation (4.3) on page 94. For cruise control along a line, $v_{ref} = constant, \theta_{ref} = 0$ are commanded. For cruise control along a line, θ is directly measured by the IMU.

The use of a proportional gain controller is justified because the map from the references v_{ref} and ω_{ref} to the actual speeds v and ω looks like a diagonal system $diag(\frac{a}{s+a}, \frac{b}{s+b})$ (at low frequencies). This is a consequence of a well-designed inner-loop (see above). The outer-loop θ 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 θ . 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.

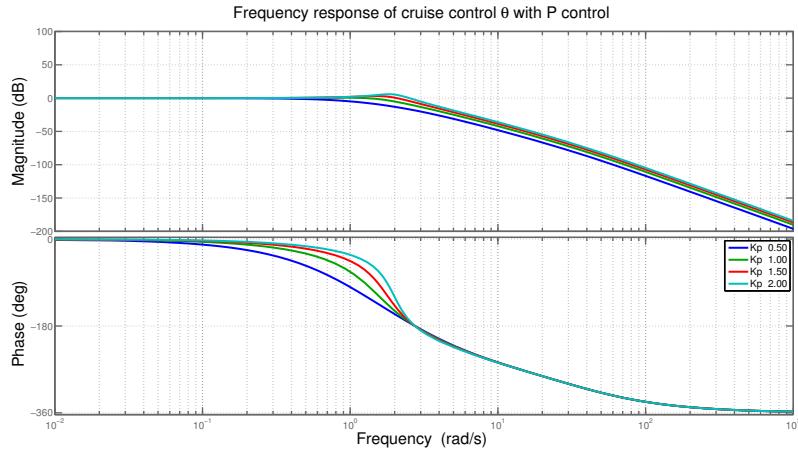


Figure 4.42: $T_{\theta_{ref}\theta}$ Frequency Response for θ Outer-Loop (P Control)

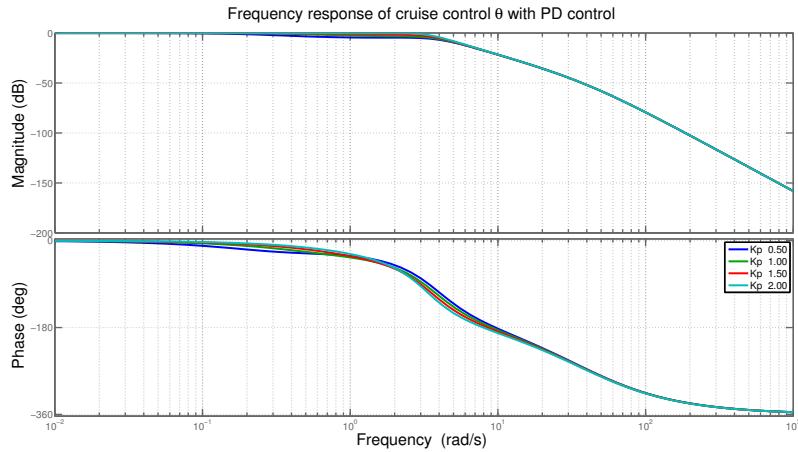


Figure 4.43: $T_{\theta_{ref}\theta}$ Frequency Response for θ Outer-Loop (PD control, $K_d = 1$)

CODE. The Raspberry Pi C code used for implementing our (v, θ) outer-loop *cruise control along a line* control law can be found within Appendix A on page 247. This outer-loop uses the (ω_R, ω_L) - (v, ω) inner-loop control law that has been implemented using the Arduino code within Appendix C on page 299.

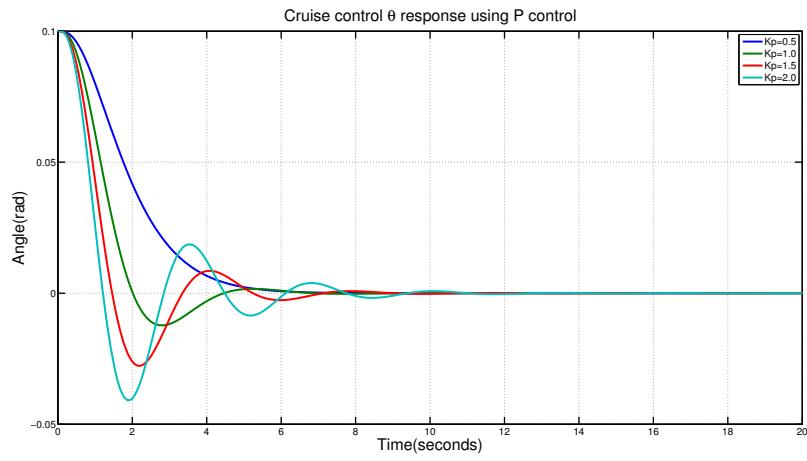


Figure 4.44: Cruise Control θ Response Using P Control ($\theta_o = 0.1$ rad)

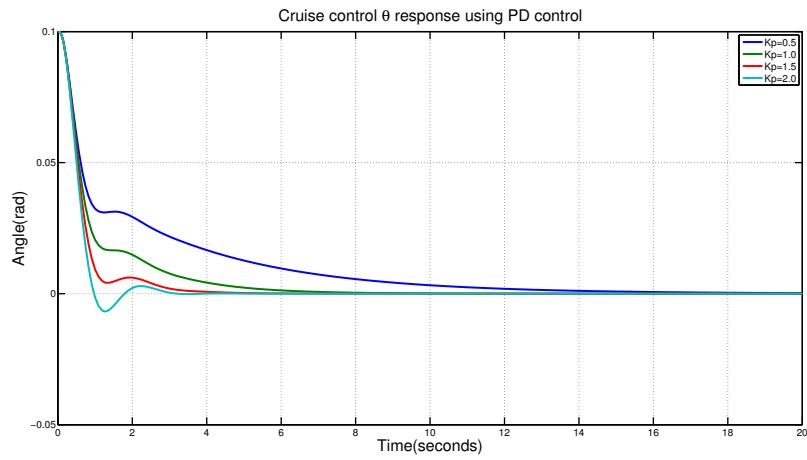


Figure 4.45: Cruise Control θ Response Using PD Control ($\theta_o = 0.1$ rad)

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].

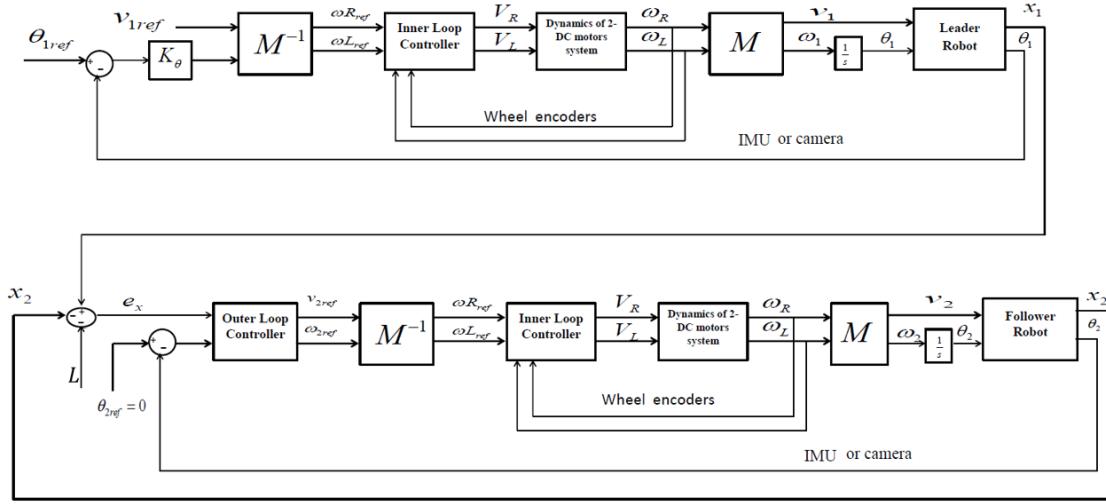


Figure 4.46: Visualization of ($\Delta x, \theta$) Separation-Direction Control System

Here, ($\Delta x, \theta$) are commanded. Δx is measured using an ultrasonic sensor. θ 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_{ref} and ω_{ref} to the actual speeds (v, ω) 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 θ 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.

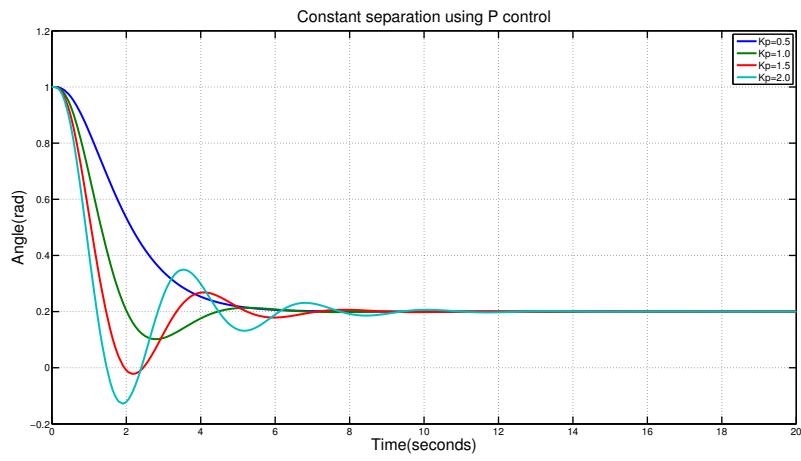


Figure 4.47: Vehicle Separation Control (Proportional Control: $K_p = 0.5, 1.0, 1.5, 2.0$; $\Delta x_o = 1$)

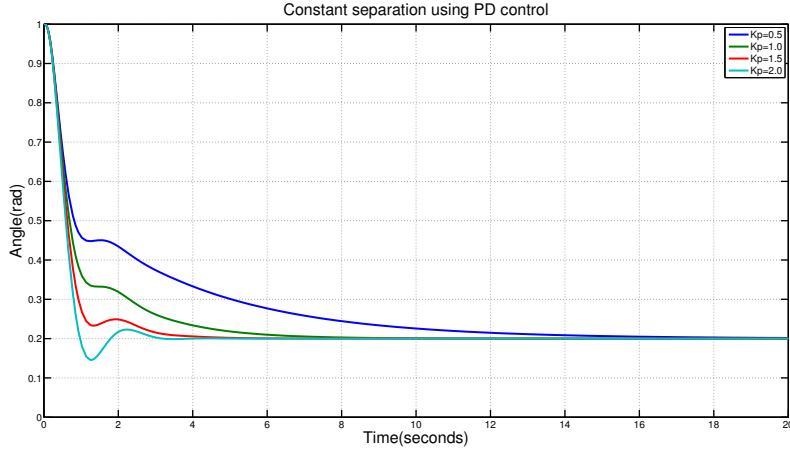


Figure 4.48: Vehicle Separation Control (PD control: $K_p = 0.5, 1.0, 1.5, 2.0; K_d=1;$
 $\Delta x_o = 1$)

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 247. 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 299.

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 ω rad/sec, then the widely used factor of 10 rule yields a maximum control bandwidth of 0.1ω 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 ($\frac{4.5}{0.05}$) 90 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 $3m/sec^2$ (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 ($\frac{2.3}{0.05}$) 46 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 $60rad/sec$. 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?

$$ZOH(j\omega) = \frac{1 - e^{-j\omega T}}{j\omega} = e^{-j\omega 0.5T} \frac{j2 \sin \omega 0.5T}{j\omega} = Te^{-j\omega 0.5T} \left[\frac{\sin 0.5\omega T}{0.5\omega T} \right] \quad (4.15)$$

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

$$e^{-s\Delta} = \frac{e^{-s0.5\Delta}}{e^{s0.5\Delta}} \approx \frac{1 - s0.5\Delta}{1 + s0.5\Delta} = \left[\frac{\frac{2}{\Delta} - s}{\frac{2}{\Delta} + s} \right] \quad (4.16)$$

it follows that a time delay Δ 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 ($2^{10} = 1024$) 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. Lets 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 $x_{resolution} = r_{wheel}\theta_{opt_{resolution}} = (5cm)(\frac{2\pi}{20}) = 0.0157 \approx 1.6$ 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 $\Delta x = 0.016 \times counter$ 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.* Lets suppose that we wish to reach a point $x = fx_{resolution}$ 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:

$$\left| \frac{x - x_{resolution} \text{counter}}{x} \right| = \left| \frac{fx_{resolution} - x_{resolution} \text{counter}}{fx_{resolution}} \right| \quad (4.17)$$

$$= \left| \frac{f - \text{counter}}{f} \right| = \left| 1 - \frac{\text{counter}}{f} \right| \quad (4.18)$$

The optimal counter value is

$$\text{Optimal Counter} = \text{round}(f) \quad (4.19)$$

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 136). With some thought, the solution presented is quite intuitive.

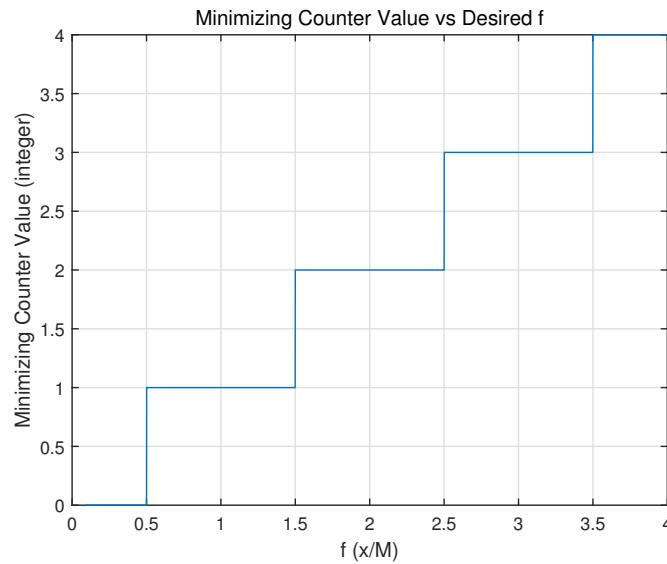


Figure 4.49: Minimizing Counter Value Versus Desired $f = \frac{x}{x_{resolution}}$

The associated optimal percent error is given by

$$\text{Optimal Percent Error} = 100 \left| 1 - \frac{\text{round}(f)}{f} \right| \quad (4.20)$$

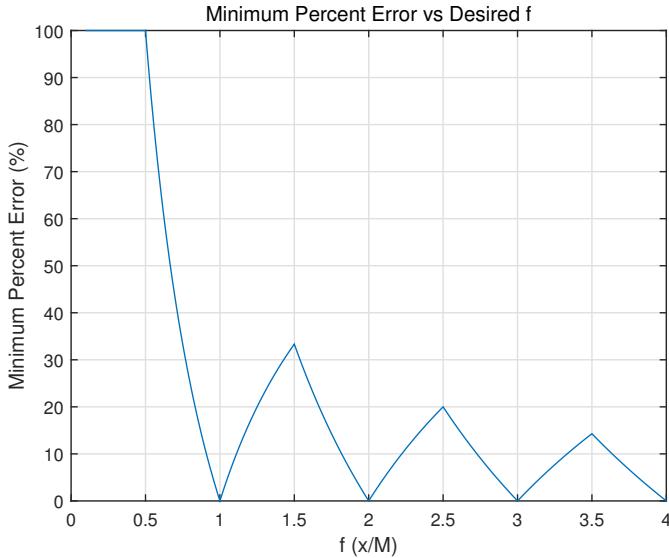


Figure 4.50: Minimum Percent Error Versus Desired $f = \frac{x}{x_{resolution}}$

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 be obtained as follows by noting that the vehicle speed is related to the wheel angular velocity via $v = r_{wheel}\omega$:

$$\left(20 \frac{\text{samples}}{\text{rev}}\right) \left(\frac{\text{rev}}{2\pi \text{ rad}}\right) \omega \frac{\text{rad}}{\text{sec}} = 63.66v \frac{\text{samples}}{\text{sec}} \quad (4.21)$$

This rate is in Hz. We multiply by 2π to convert to rad/sec. Doing so

yields $400v \frac{\text{rad}}{\text{sec}}$. Using our factor-of-ten rule then gives $BW_{encoder_{limit}} = 0.1(400v) = 40v \frac{\text{rad}}{\text{sec}}$ where the vehicle speed v is measured in m/sec; i.e. we will have an associated 40 v rad/sec encoder-based maximum control bandwidth constraint. We see that $40v$ will be larger than our limit 4 rad/sec (due to half sample A-to-D zero order hold effect with $T = 0.1$, see above) if $v > 0.1 \frac{m}{\text{sec}}$ or about $8 \frac{\text{inches}}{\text{sec}}$. 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 94)

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

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

From this, we therefore see that if we give a speed command between 0.1571 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. 10counts in a $T = 0.1$ sampling window or $(0.1571)(10) = 1.57 \frac{m}{\text{sec}}$ (100 counts/sec). This is smaller than the maximum observed speed of the fully loaded vehicle (i.e. about 2.3 m/sec). When we observe only $count = 5$ counts in a $T = 0.1$ sec sampling window (or $50 \frac{\text{counts}}{\text{sec}}$), the percent error rises to 20%. This corresponds to a speed of $(0.1571)(5) = 0.785 \frac{m}{\text{sec}}$. Since the maximum speed is 2.3 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 θ 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 θ 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 $BW_{IMU_{limit}} = 0.1 \times 2\pi(100) \approx 60rad/s$. 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 - $\pm 2, 4, 6, 8, 16g$; Gyro - $\pm 245, 500, 2000^\circ/sec$.

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) angle² of 45° [68]. It can send out a 40 kHz ultrasonic signal every 200 μ sec. Consider an object sitting 2 m away from the sensor. The time delay can be calculated as $\Delta = \frac{2 \times 2m}{340 \frac{m}{sec}} \approx 0.0118 \text{ sec}$. 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.

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

²If 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.

- 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 (ω_R, ω_L); all computations are based on equation (4.1) (page 93), equation (4.2) (page 94), equation (4.3)-(4.4) (page 94).
- *Inner-Loop Reference Commands:* Speed reference command and angular rate reference command (v_{ref}, ω_{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}, ω_{ref}) to the actual speeds (v, ω) looks like a diagonal system $diag(\frac{a}{s+a}, \frac{b}{s+b})$ after closing the (ω_R, ω_L) inner-loop; (ω_R, ω_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, θ) 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, θ) cruise control system - one that follows speed and direction commands - is an essential cornerstone to build more complex control capabilities. It builds on the (ω_R, ω_L) or (v, ω) 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, ω) inner-loop control law [15]. See justification of outer-loop below.
- *Outer-Loop Sensor:* Vehicle orientation θ directly measured by IMU/Camera is to be fed back
- *Outer-Loop Output:* Vehicle actual speed v is generated by inner-loop and orientation θ 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 θ (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 ω_{ref} to the actual speeds v and ω looks like a diagonal system $diag(\frac{a}{s+a}, \frac{b}{s+b})$ (at low frequencies). The outer-loop θ 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 θ . A PD controller with roll off would help with this issue.
- *Performance Tradeoffs:* If the proportional gain k_θ is large, there will be oscillations in θ . 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 θ 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.

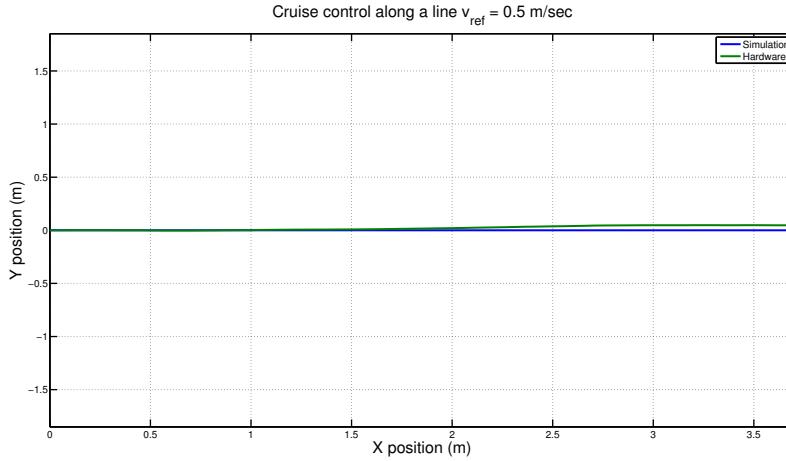


Figure 4.51: Cruise Control Along a Line

- *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 $\theta_{ref} = 0^\circ$. 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, ω) 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, ω) inner loop are involved. Δx is measured using ultrasonic sensor, and θ 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 ω measured by IMU is integrated to yield the vehicle orientation θ 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 ω_{ref} to the actual speeds v and ω looks like a diagonal system $diag(\frac{a}{s+a}, \frac{b}{s+b})$ (at low frequencies). The outer-loop θ controller therefore sees $\left[\frac{b}{s(s+a)} \right]$, 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.

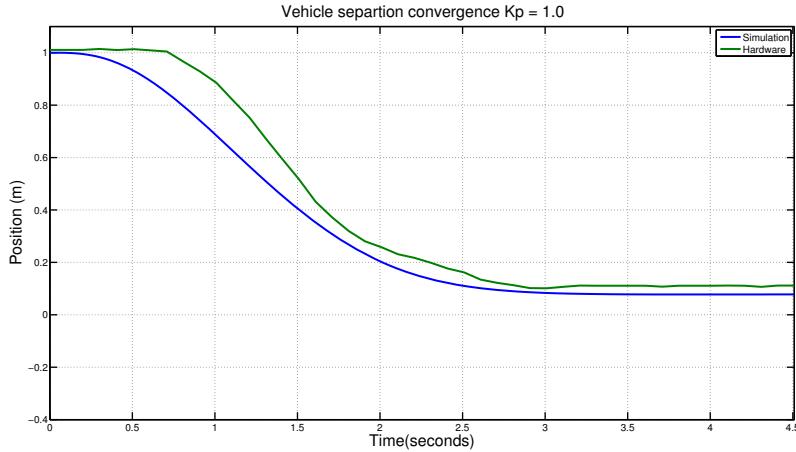


Figure 4.52: Vehicle Separation Convergence Using Proportional Control ($K_p = 1$; $\Delta x(0) \approx 1$)

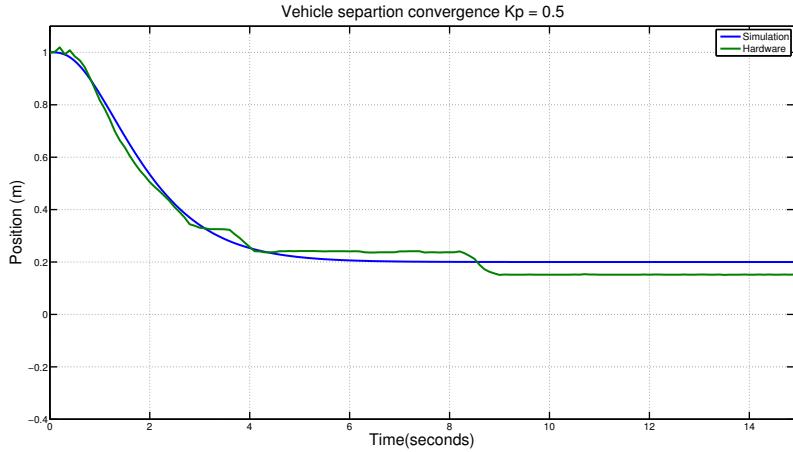


Figure 4.53: Vehicle Separation Convergence Using Proportional Control ($K_p = 1$; $\Delta x(0) \approx 1$)

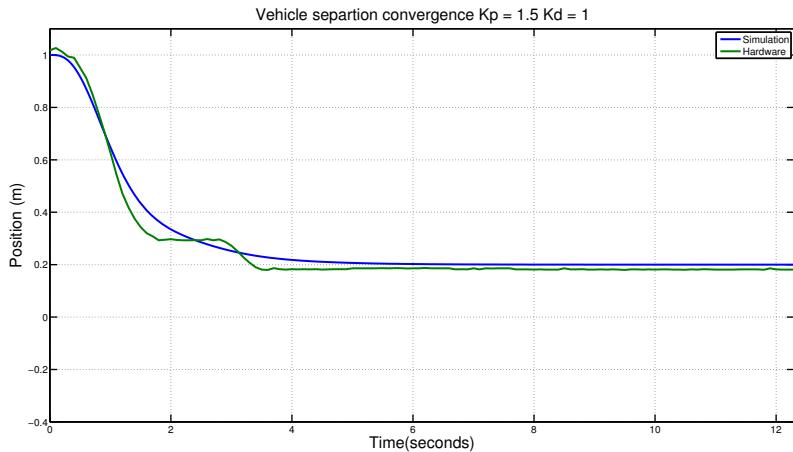


Figure 4.54: Vehicle Separation Convergence Using PD Control ($K_p = 1.5$; $K_d = 1$; $\Delta x(0) \approx 1$)

- *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 $200 \mu\text{s}$. Consider an object sitting 2m away from the sensor. The time delay can be calculated as $\Delta = \frac{4}{340} \approx 0.0118s$. 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 Δx 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.

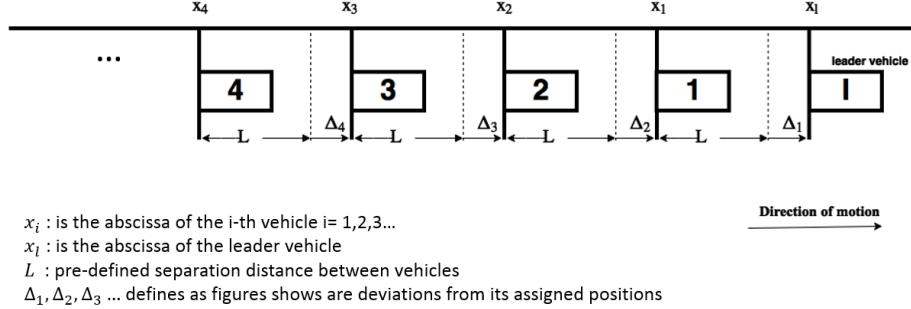


Figure 5.1: Platoon of 4 Vehicles

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 $i - th$ 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, Δ is the deviation of the $i - th$ vehicle position from its assigned position. The subscript i is used because Δ_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 $i = 2, 3, \dots$

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

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

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

Kinematics The corresponding kinematic equations for the lead vehicle and the

first vehicle are as follows:

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

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

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

5.3 Modeling for Longitudinal Platoon of Vehicles

As we discussed in 3.7.3, the on ground inner-loop (ω_R, ω_L) plant:

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

will be used to 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.

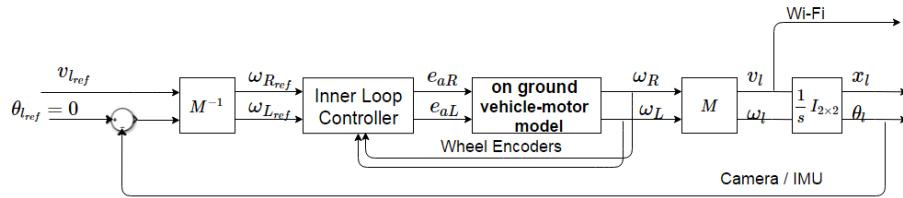


Figure 5.2: Leader Model of Platoon

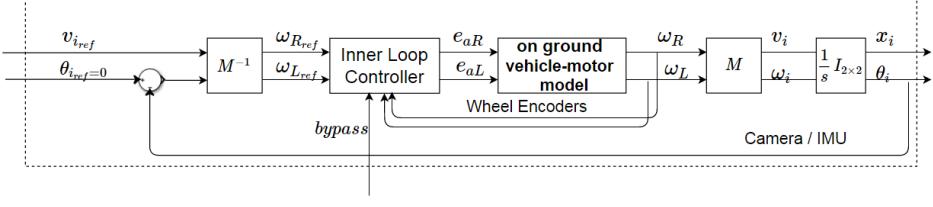


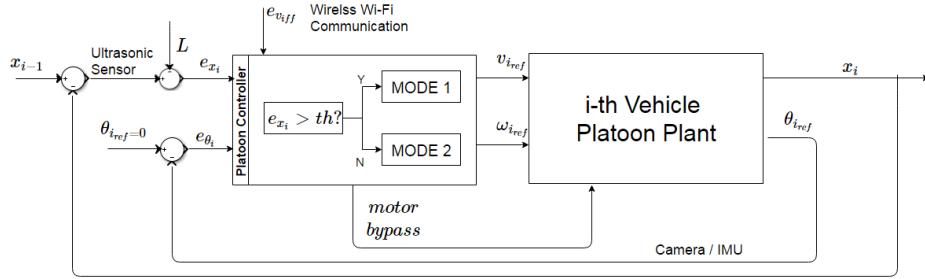
Figure 5.3: i-th Vehicle Model of Platoons

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, θ) 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.

Here, θ_i are commanded to be zero. Δ_i is compute by 5.2 relative distance is measured using an ultrasonic sensor. θ 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 Δ_i is greater than th . Whereas mode 2 is a safety feature which will



$e_{x_i} = \Delta_i$: is the deviation of the i-th vehicle
 $e_{v_{iff}}$: is the speed difference between leader and i-th vehicle

Figure 5.4: Visualization of i-th Follower in Platoon Separation Control System

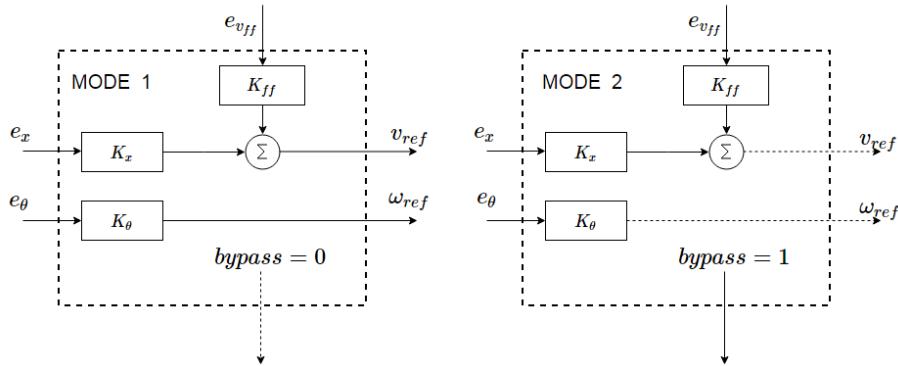


Figure 5.5: Visualization of Platoon Controller

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} (Δ_i), we named it Δ_x path . K_θ is a PD controller with respect to orientation error e_θ . 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 ω_{ref} to the actual speeds (v, ω) 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 θ 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.

5.5 Longitudinal Platoon Separation Controller Tradeoff Study

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

- Proportional Control for Δ_x path
- PD Control for Δ_x path
- PID Control for Δ_x path
- PID Control for Δ_x path
- PID Control for Δ_x path + Proportional Control for FF-path
- PID Control for Δ_x path + PI control for FF-path

Proportional Control for Δ_x path. Proportional control for Δ_x is used, and no leader information.

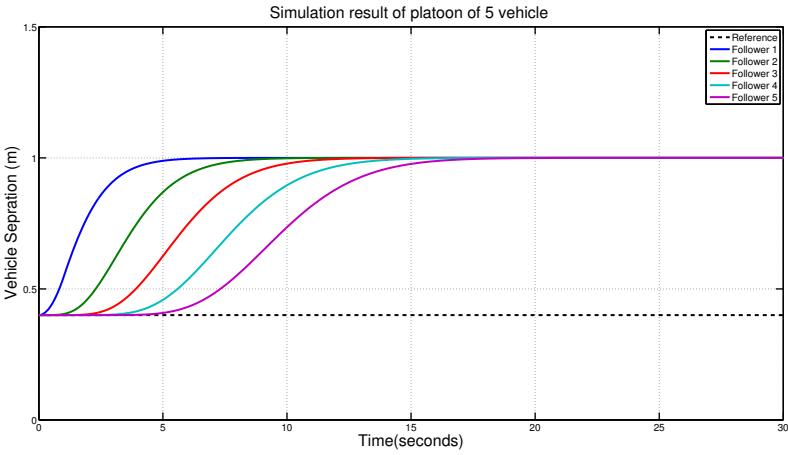


Figure 5.6: Simulation of Vehicle Separation Control of Platoon (Proportional Control for Δ_x ($K_p = 0.5$)

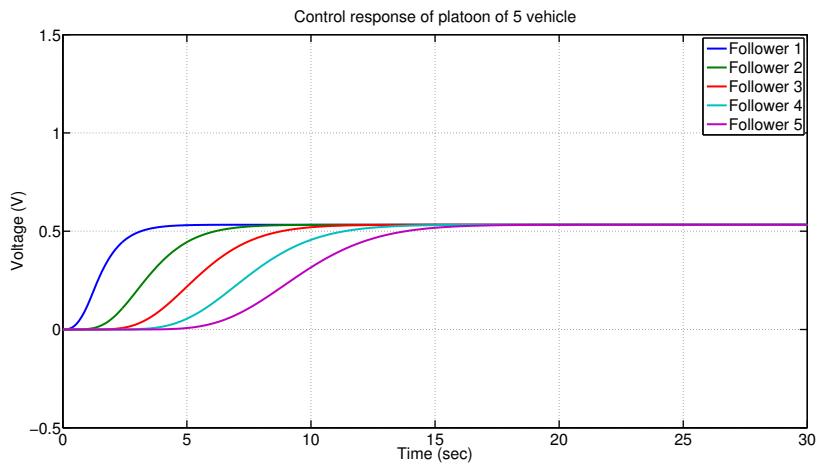


Figure 5.7: Simulation of Control Response of Platoon (Proportional Control for Δ_x ($K_p = 0.5$)

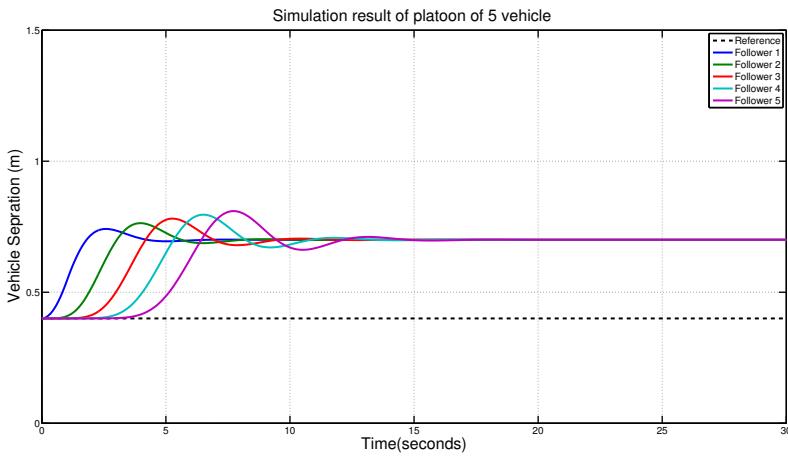


Figure 5.8: Simulation of Vehicle Separation Control of Platoon (Proportional Control for Δ_x ($K_p = 1.0$)

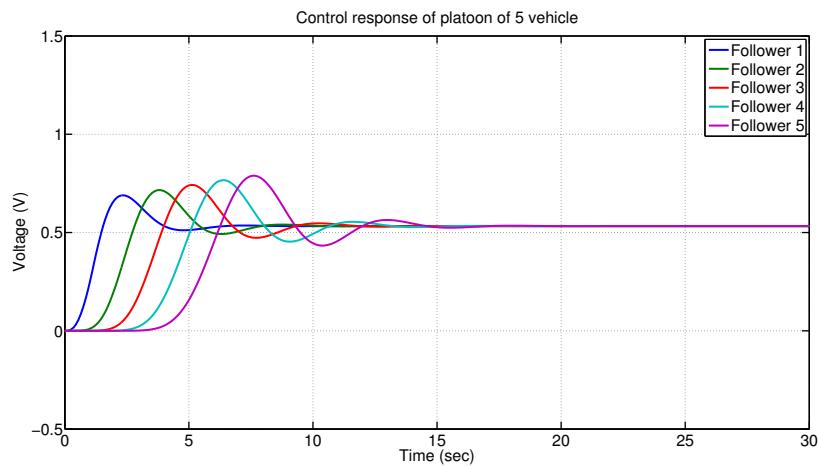


Figure 5.9: Simulation of Control Response of Platoon (Proportional Control for Δ_x ($K_p = 1.0$)

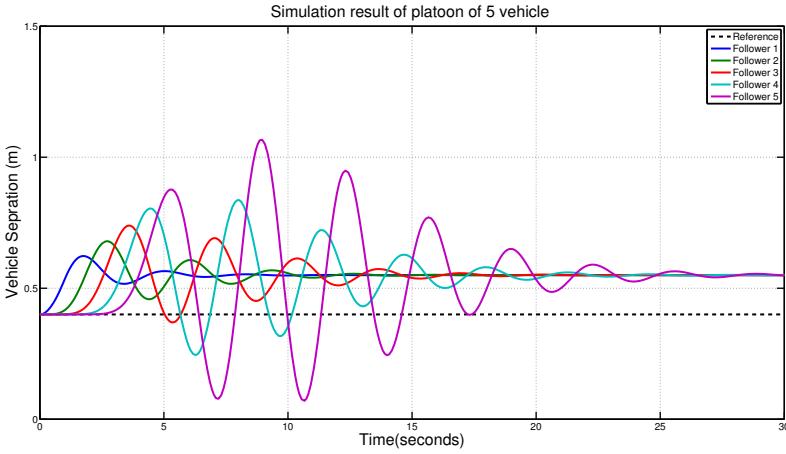


Figure 5.10: Simulation of Vehicle Separation Control of Platoon (Proportional Control for Δ_x ($K_p = 2.0$)

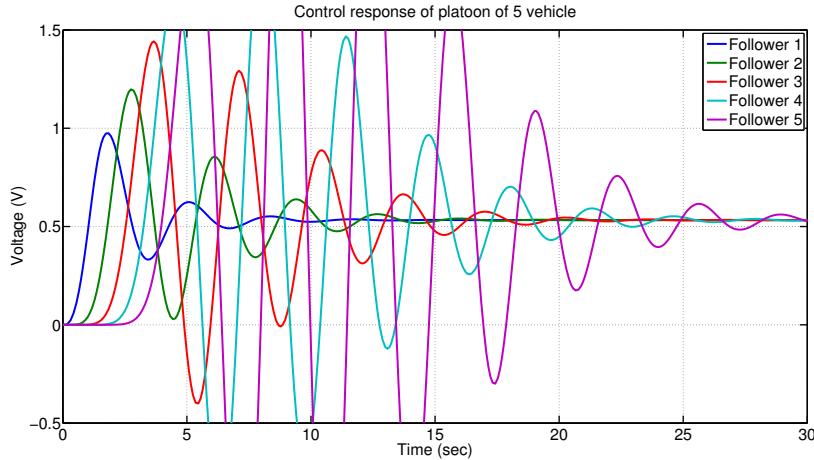


Figure 5.11: Simulation of Control Response of Platoon (Proportional Control for Δ_x ($K_p = 2.0$)

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 Δ_x path.

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

$$K_x = \frac{g(s + z) \times 100^2}{(s + 100)^2} \quad (5.8)$$

where, $z = 5$ and two roll-off is used to reduce overshoot from derivation-term.

Note here $k_d = g$ and $k_p = gz$.

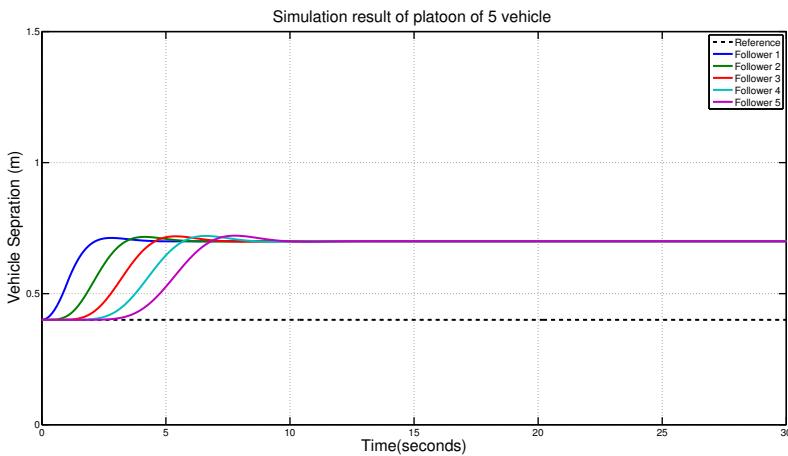


Figure 5.12: Simulation of Vehicle Separation Control of Platoon (PD Control for Δ_x ($g = 0.2$))

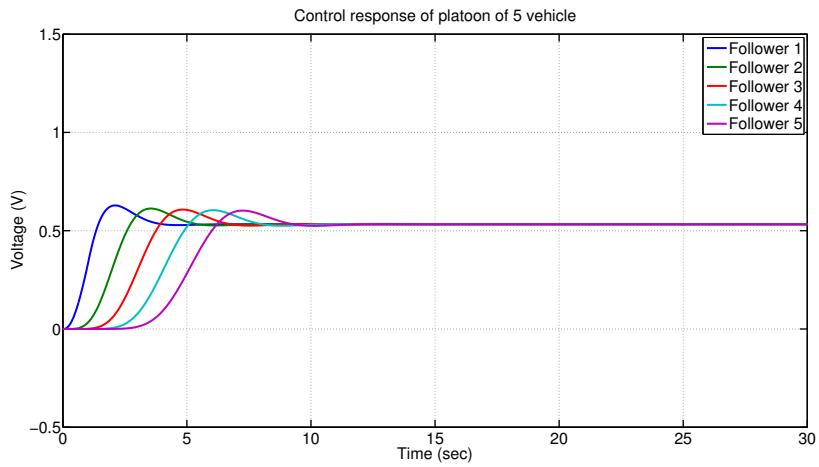


Figure 5.13: Simulation of Control Response of Platoon (PD Control for Δ_x ($g = 0.2$))

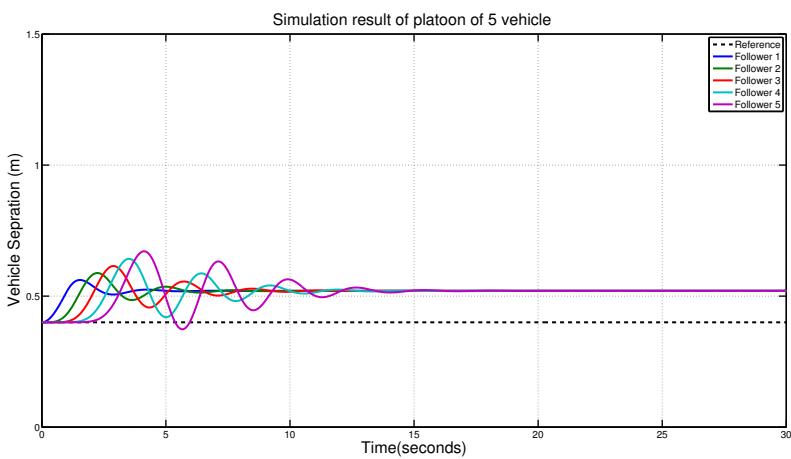


Figure 5.14: Simulation of Vehicle Separation Control of Platoon (PD Control for Δ_x ($g = 0.5$))

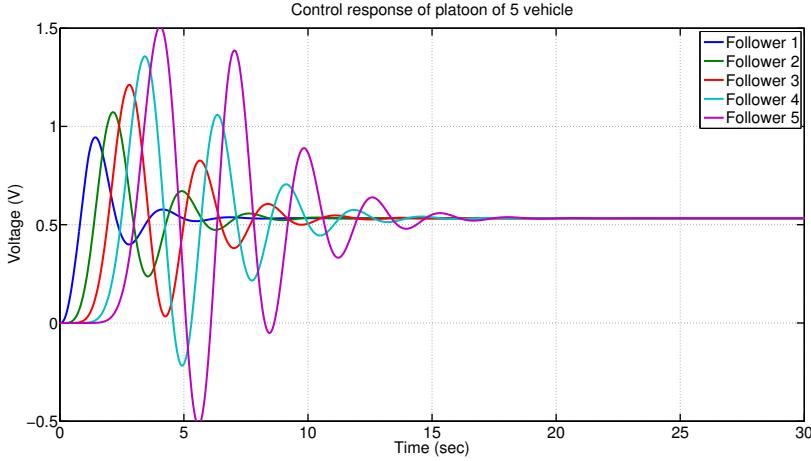


Figure 5.15: Simulation of Control Response of Platoon (PD Control for Δ_x ($g = 0.5$))

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 Δ_x path will be examined first, then feed-forward path is introduced to reduce oscillation.

PID Control for Δ_x path. PID control for Δ_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 $k_i = gz^2$.

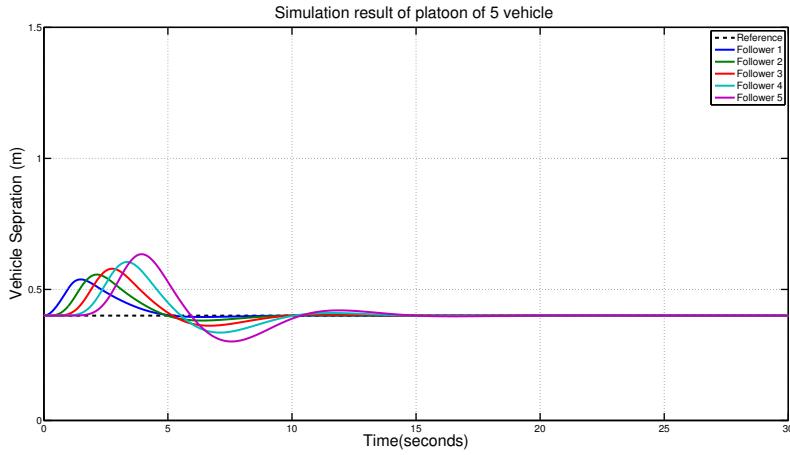


Figure 5.16: Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 1.0$))

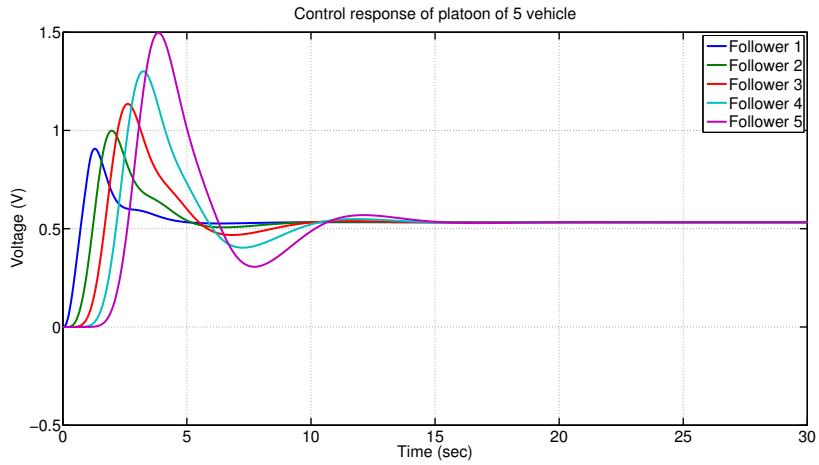


Figure 5.17: Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 1.0$))

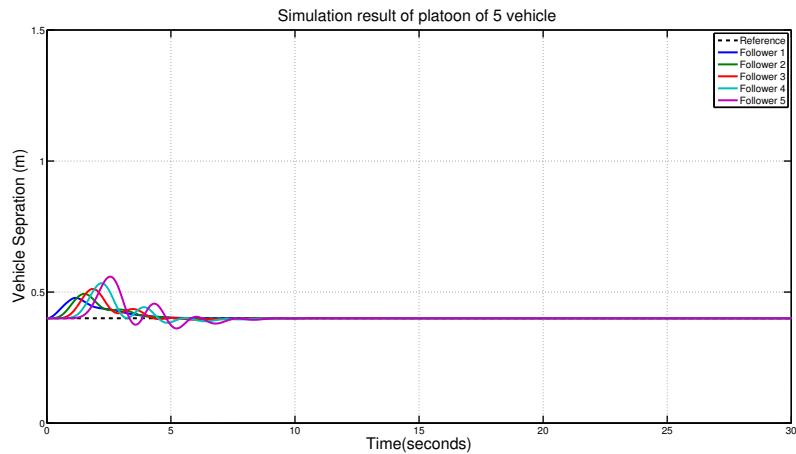


Figure 5.18: Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 2.0$))

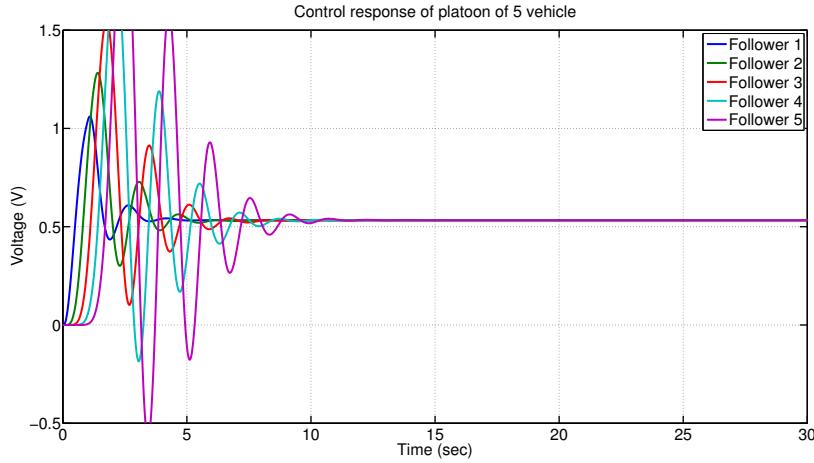


Figure 5.19: Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 2.0$))

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 Δ_x path + Proportional Control for Feed-forward.

Here, PID control for Δ_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 Δ_x path and feed-forward path. Controller structure is like following:

$$K_{longitudinal} = K_x + K_{ff} \quad (5.10)$$

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

$$K_{ff} = k_{pff} \quad (5.12)$$

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 $k_i = gz^2$.

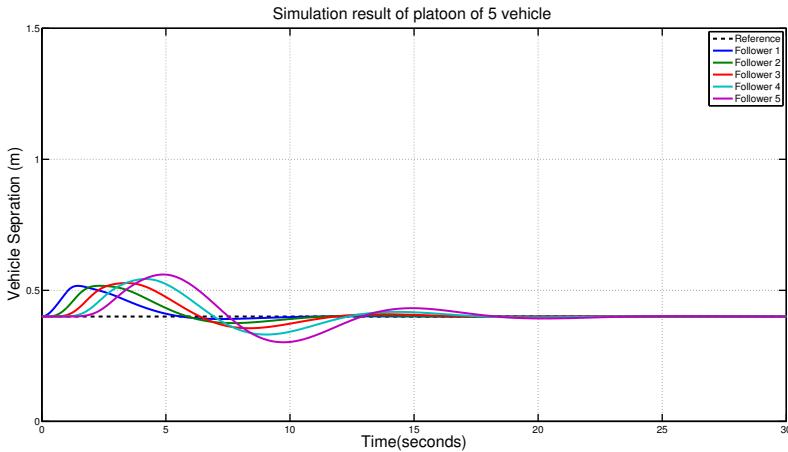


Figure 5.20: Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and $k_{pff} = 0.5$ for FF-path))

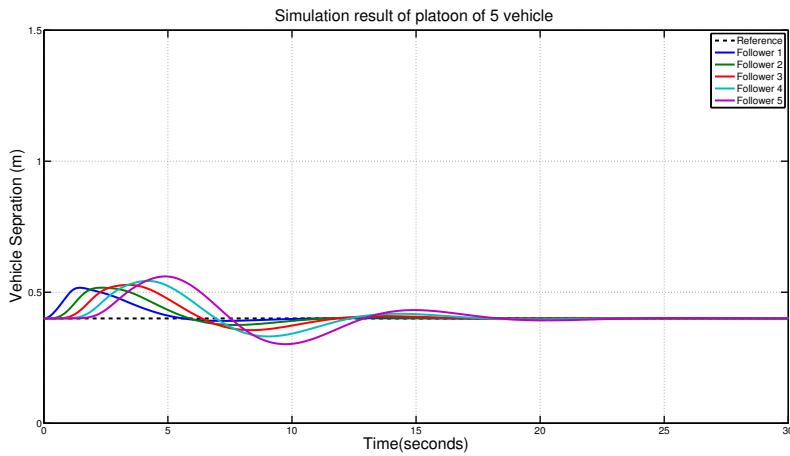


Figure 5.21: Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and $k_{pff} = 0.5$ for FF-path))

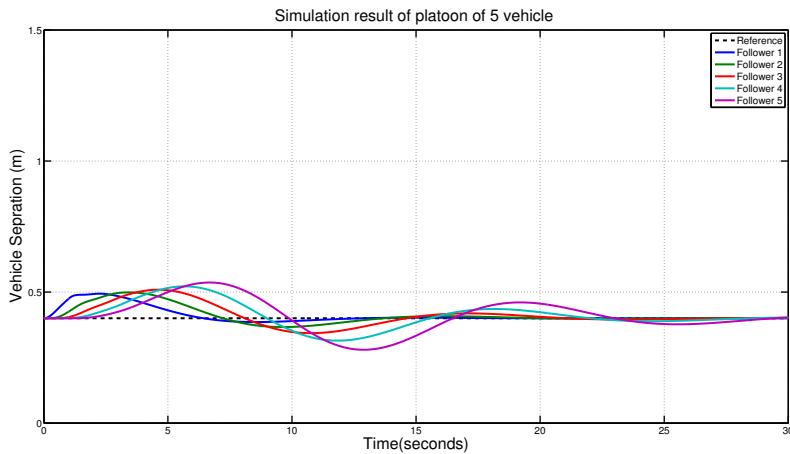


Figure 5.22: Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and $k_{pff} = 1.5$ for FF-path))

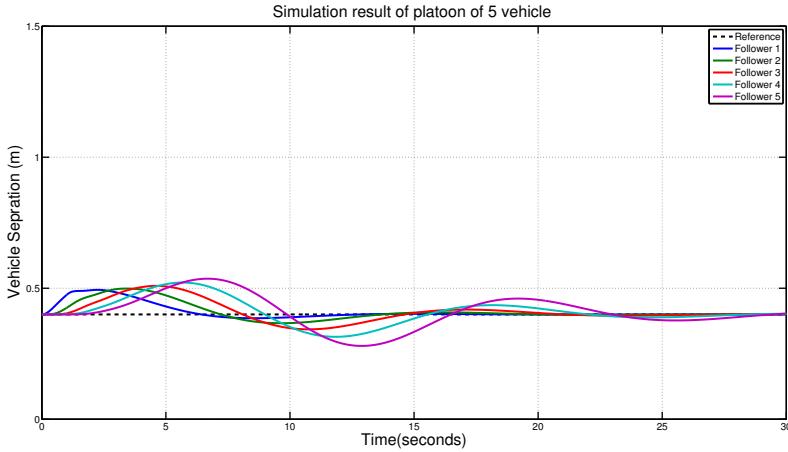


Figure 5.23: Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and $k_{pff} = 1.5$ for FF-path))

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 Δ_x path + PI Control for Feed-forward. Here, PID control for Δ_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 Δ_x path and feed-forward path. Controller structure is like following:

$$K_{longitudinal} = K_x + K_{ff} \quad (5.13)$$

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

$$K_{ff} = \frac{g_{ff}(s+z_{ff}) * 100}{s(s+100)} \quad (5.15)$$

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 $k_{pff} = g_{ff}z_{ff}$

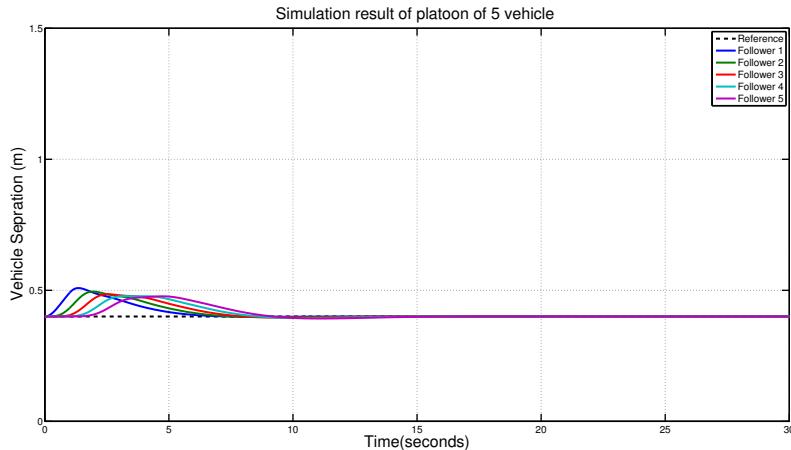


Figure 5.24: Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and PI Control for FF-path ($g_{ff} = 0.5, z_{ff} = 1.0$))

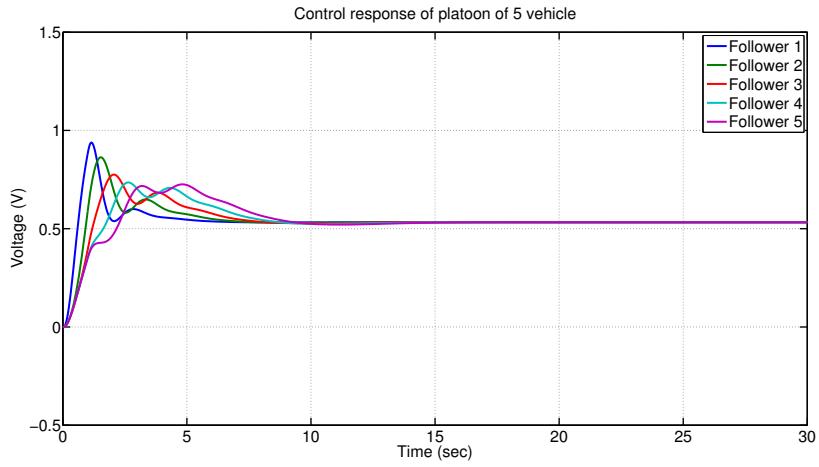


Figure 5.25: Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and PI Control for FF-path ($g_{ff} = 0.5, z_{ff} = 1.0$))

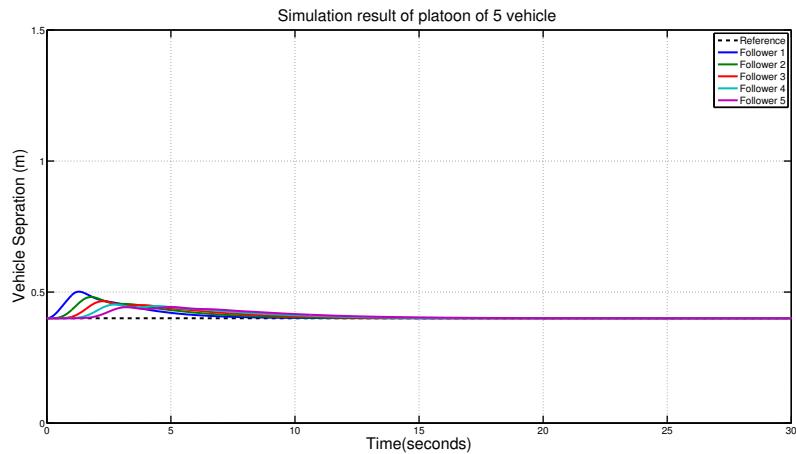


Figure 5.26: Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and PI Control for FF-path ($g_{ff} = 0.5, z_{ff} = 2.0$))

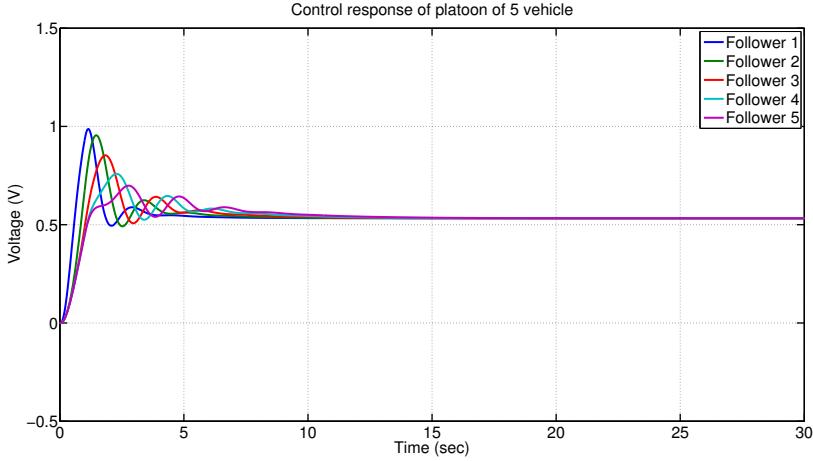


Figure 5.27: Simulation of Control Response of Platoon (PID Control for Δ_x ($g = 1.0, z = 1.0$) and PI Control for FF-path ($g_{ff} = 0.5, z_{ff} = 2.0$))

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} is 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 Δ_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 feed-forward 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.

Design No.	Δ_x Path	Feed-Forward Path
1	$K_p = 1$	N/A
2	$K_p = 1.5, K_d = 0.2$	$K_p = 0.5$
3	$K_p = 1.0, K_i = 0.5, K_d = 0.3$	$K_p = 0.4, K_i = 0.6$

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 Δ_x is used, and no leader information.

Simulation result is shown in Figure 5.28 and 5.29

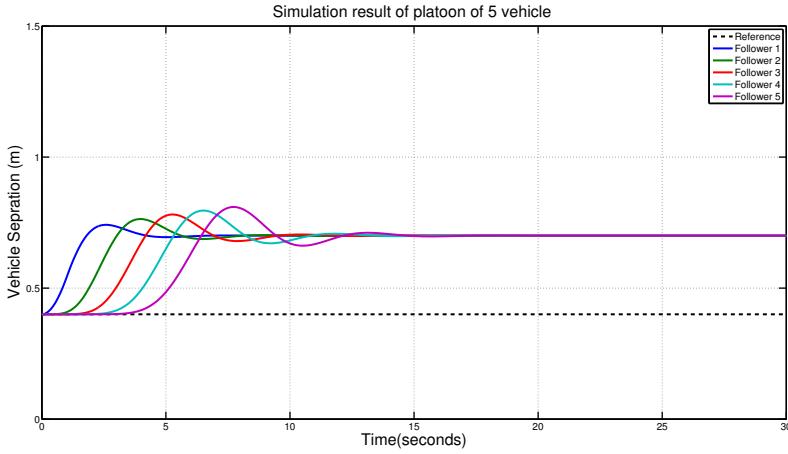


Figure 5.28: Simulation of Vehicle Separation Control of Platoon (Proportional Control for Δ_x ($K_p = 1$)

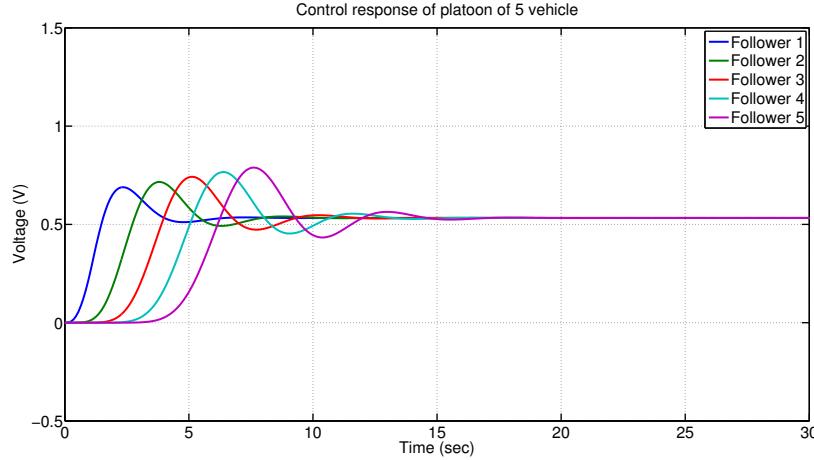


Figure 5.29: Simulation of Control Response of Platoon (Proportional Control for Δ_x ($K_p = 1$)

Hardware result of platoon of 6 vehicles is shown in Figure 5.30

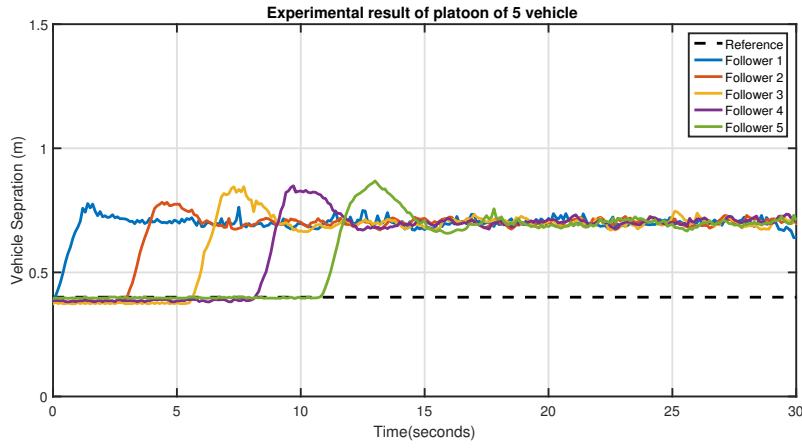


Figure 5.30: Experimental Separation Response of Platoon Proportional Control for Δ_x ($K_p = 1$)

- *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 Δ_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

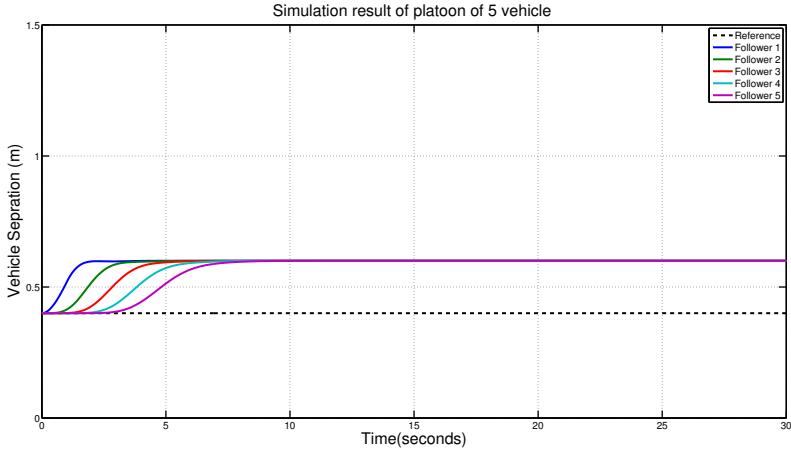


Figure 5.31: Simulation of Vehicle Separation Control of Platoon (PD Control for Δ_x ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$))

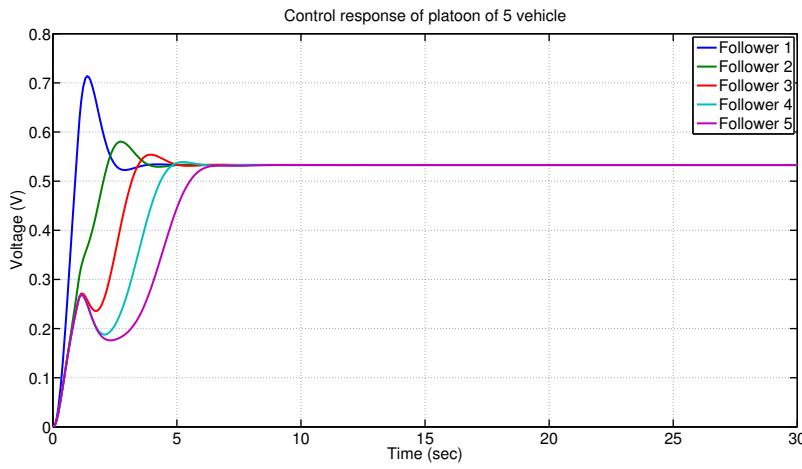


Figure 5.32: Simulation of Control Response of Platoon (PD Control for Δ_x ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$))

Hardware result of platoon of 5 vehicles is shown in Figure 5.33

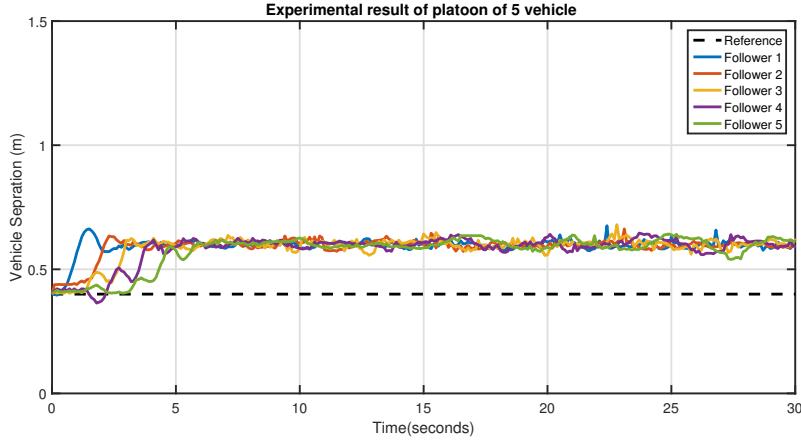


Figure 5.33: Experimental Separation Response of Platoon (PD Control for Δ_x ($K_p = 1.5, K_d = 0.2$) and Proportional control for FF Path ($K_p = 0.5$))

- *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 Δ_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

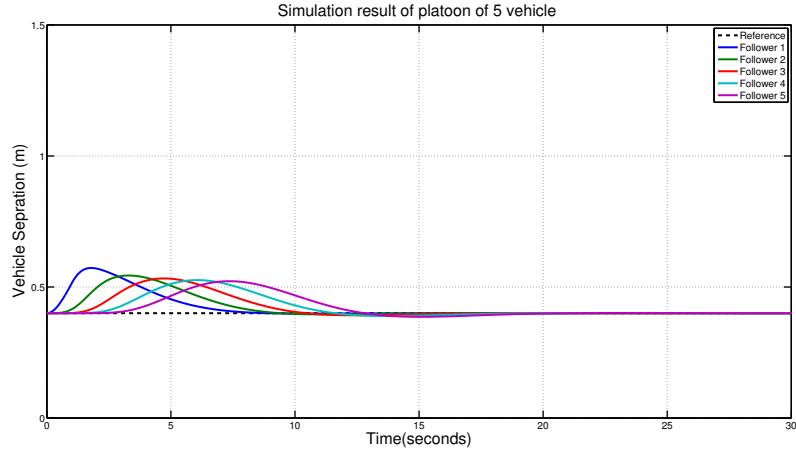


Figure 5.34: Simulation of Vehicle Separation Control of Platoon (PID Control for Δ_x ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$))

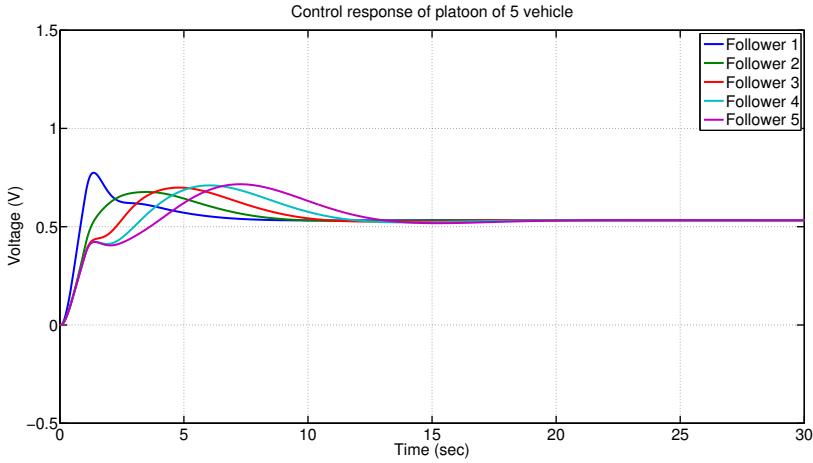


Figure 5.35: Simulation of Control Response of Platoon (PID Control for Δ_x ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$))

Hardware result of platoon of 5 vehicles is shown in Figure 5.36

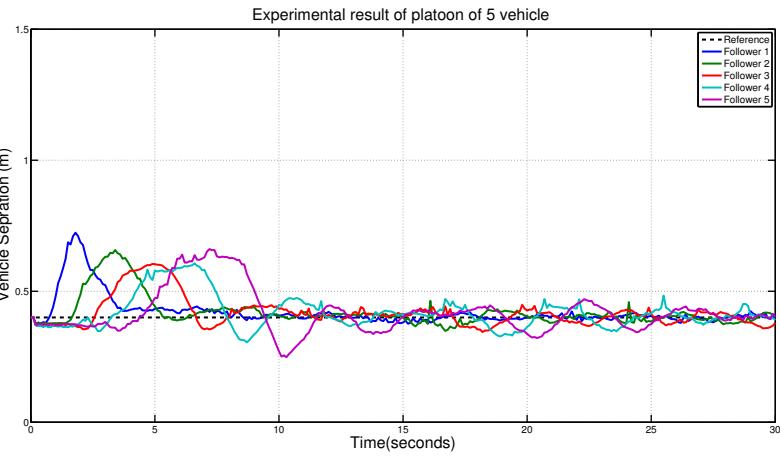


Figure 5.36: Experimental Separation Response of Platoon (PID Control for Δ_x ($K_p = 1.0, K_i = 0.5, K_d = 0.3$) and Proportional control for FF Path ($K_p = 0.4, K_i = 0.6$))

- *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.

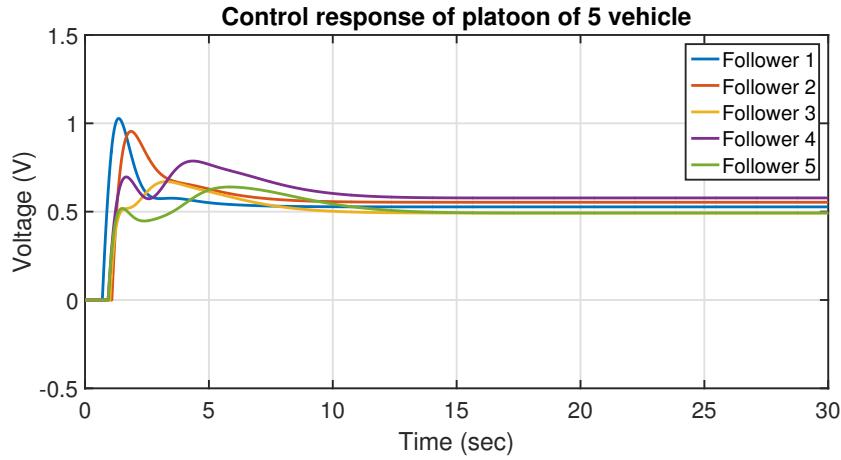


Figure 5.37: Simulation of Separation Response of Platoon with model uncertainty and stiction

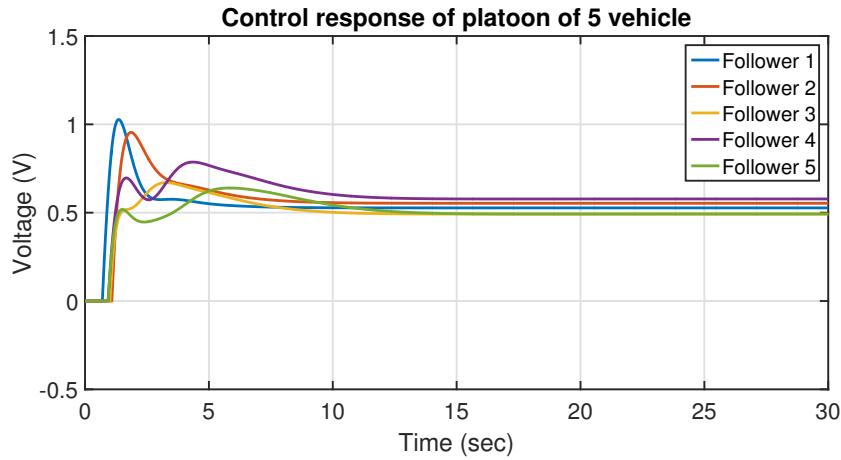


Figure 5.38: Simulation of Control Response of Platoon with model uncertainty and stiction

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 extant 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.

(a) **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 θ 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 com-

pared 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.

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

(c) **Literature Survey.** A fairly comprehensive literature survey of relevant work was presented.

(d) **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.

- (e) **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.
- (f) **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.
- (g) **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

- [1] Caudill R. J., Garrard W. L., "Vehicle-follower longitudinal control for automated transit vehicles," *Journal of Dynamic Systems, Measurement, and Control* , pp. 241-248, 1977.
- [2] Chiu, Harry Y, George B. Stupp., "Vehicle-follower control with variable-gains for short headway automated guideway transit systems," *Journal of Dynamic Systems, Measurement, and Control* , pp. 183-189, 1977.
- [3] S.E. Sheikholeslam, C. Desoer, "Longitudinal Control of a Platoon of Vehicles," *American Control Conference, IEEE*, pp. 291-296, 1990.
- [4] S.E. Sheikholeslam, C. Desoer, "Longitudinal Control Of A Platoon Of Vehicles. I, Linear Model," California Partners for Advanced Transit and Highways (PATH), 1989.
- [5] S.E. Sheikholeslam, C. Desoer, "Longitudinal control of a platoon of vehicles. III, Nonlinear model," California Partners for Advanced Transit and Highways (PATH), 1990.
- [6] S.E. Sheikholeslam, "Longitudinal control of a platoon of vehicles with no communication of lead vehicle information: a system level study," *IEEE Transactions on vehicular technology* , vol. 42, pp.546-554, 1991.
- [7] Hedrick, J. K., Mcmahon D. H., and Swaroop D., "Vehicle modeling and control for automated highway systems," California Partners for Advanced Transit and Highways (PATH), 1993.
- [8] Varaiya, P., "Smart cars on smart roads: problems of control.," *IEEE Transactions on automatic control* , vol. 38 , pp.195-207, 1993.
- [9] Hedrick, J. K., Mcmahon D. H., and Swaroop D., "Longitudinal control development for IVHS fully automated and semi-automated systems: Phase III," California Partners for Advanced Transit and Highways (PATH), 1996.
- [10] Datta N. G., and Lygeros J., "Longitudinal control of the lead car of a platoon," *IEEE Transactions On Vehicular Technology*, vol. 43 , pp.1125-1135, 1994.
- [11] Liang N. G., and Lygeros J., "Optimal adaptive cruise control with guaranteed string stability," *Vehicle System Dynamics*, vol. 32 , pp.313-330, 1999.
- [12] Stankovic, S. S., Stanojevic M. J., and Siljak D. D., "Decentralized Overlapping Control of a Platoon of Vehicles," *IEEE Transactions on Control Systems Technology* , vol. 8 , pp.816-832, 2000.
- [13] Jovanovic, M. R., Bamieh B.,, "On the ill-posedness of certain vehicular platoon control problems," *IEEE Transactions on Automatic Control* , vol. 50 , pp.1307-1321, 2005.

- [14] Lin Z., *Modeling, Design and Control of Multiple Low-Cost Robotic Ground Vehicles*, Arizona State University, MS Thesis, 2015.
- [15] Dhaouadi R. and A. A. Hatab, “Dynamic Modeling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework,” *Adv Robot Autom* 2.107, 2013.
- [16] “Rotary Electrodynamics of a DC Motor: Motor as Mechanical Capacitor,” http://www2.ece.ohio-state.edu/~passino/lab2_rotary_dynamics.pdf.
- [17] Ribeiro M. I., Lima P., “Kinematic models of mobile robots,” Instituto de Sistemas e Robotica, 2002: 1000-1049.
- [18] R. Fierro and F.L. Lewis, “Control of a nonholonomic mobile robot: backstepping kinematics into dynamics,” *Proceedings of the 34th IEEE Conference In Decision and Control*, volume 4, 1995.
- [19] Kanayama Y, Kimura Y, Miyazaki F, et al, “A stable tracking control method for an autonomous mobile robot,” *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 384-389, 1990.
- [20] Petrov P., “Modeling and adaptive path control of a differential drive mobile robot,” *Proceedings of the 12th WSEAS international Conference on Automatic Control, Modelling & Simulation*, pp. 403-408, 2010.
- [21] A.M. Bloch, M. Reyhanoglu and H. McClamorch, “Control and stabilization of nonholonomic dynamic systems,” *IEEE Transactions On Automatic Control*, 37, 11, pp. 1746-1757, 1992.
- [22] Gholipour A, Yazdanpanah M J., “Dynamic Tracking Control of Nonholonomic Mobile Robot with Model Reference Adaptation for Uncertain Parameters,” *European Control Conference (ECC)*, University of Cambridge, UK, 2003.
- [23] Pepy R., Lambert A., and Mounier H., “Path Planning using a Dynamic Vehicle Model,” *Information and Communication Technologies, ICTTA '06. 2nd. IEEE*, 1: 781-786, 2006.
- [24] Bradski G. and Kaehler A., “Learning OpenCV: Computer vision with the OpenCV library,” *O'Reilly Media, Inc.*, 2008.
- [25] Vieira, F.C., Medeiros, A.A.D., Alsina, P.J., Araujo, A.P., “Position and Orientation Control of a Two-Wheeled Differentially Driven Nonholonomic Mobile Robot,” *ICINCO Proceedings*, 7 pages, 2004.
- [26] Rodriguez, A.A., *Analysis and Design of Multivariable Feedback Control Systems*, Control3D, L.L.C., Tempe, AZ, 2002.
- [27] R. DeSantis, “Path-tracking for a tractor-trailer-like robot,” *Int. J. Rob. Research*, Vol. 13, No. 6, pp. 533-543, 1994.

- [28] B. Andrea-Novel, B. Campion, G. Bastin, “Control of nonholonomic wheeled mobile robots by state feedback linearization,” *The Int. J.*
- [29] Aguiar A. P., Dacic D. B., Hespanha J. P., et al., “Path-following or reference-tracking,” 2004.
- [30] F. del Rio, G. Jimenez, J. Sevillano, S. Vicente, A.Balcells, “A path following control for unicycle robots,” *J. Rob. Systems*, Vol. 18, No.7, pp.325-342, 2001.
- [31] M. El-Hawwary, M. Maggiore, “Global path following for unicycle and other results,” *Proc. IEEE American Contr. Conference*, pp. 3500-3505, 2008.
- [32] K. J. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning*, Instrument Society of America, Research Triangle Park, North Carolina, 1995.
- [33] Marino R, Scalzi S, Cinili F., “Nonlinear PI front and rear steering control in four wheel steering vehicles,” *Vehicle System Dynamics*, 2007, 45(12): 1149-1168.
- [34] De Silva C. W., *Sensors and actuators: control system instrumentation*, CRC Press, 2007.
- [35] FUJH T, MIZUSHIMA N. A New Approach to LQ Design: Application to the Design of Optimal Servo Systems. 1987.
- [36] Arduino Uno description, <https://www.arduino.cc/en/Main/arduinoBoardUno>
- [37] LCR meter, https://en.wikipedia.org/wiki/LCR_meter.
- [38] TB6612FNG datasheet, <https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf>.
- [39] Raspberry Pi 3 benchmarks, <https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/>.
- [40] Energizer AA Battery, <http://data.energizer.com/PDFs/nickelmetalhydrideappman.pdf>.
- [41] Raspberry Pi 3 v.s. Pi 2, <http://www.techrepublic.com/article/raspberry-pi-3-how-much-better-is-it-than-the-raspberry-pi-2/>.
- [42] Introducing the Raspberry Pi 2 - Model B - Adafruit Learning. <https://learn.adafruit.com/introducing-the-raspberry-pi-2-model-b/overview>.
- [43] Brambilla M, Ferrante E, Birattari M, et al., “Swarm robotics: a review from the swarm engineering perspective,” *Swarm Intelligence*, 2013, 7(1): 1-41.

- [44] Aung W. P., "Analysis on modeling and Simulink of DC motor and its driving system used for wheeled mobile robot," World Academy of Science, Engineering and Technology, 2007, 32: 299-306.
- [45] Lyshevski SE. Nonlinear control of mechatronic systems with permanent-magnet DC motors. *Mechatronics* 1999;9:53952.
- [46] Kara T, Eker I., "Nonlinear modeling and identification of a DC motor for bidirectional operation with real time experiments," *Energy Conversion and Management*, 2004, 45(7): 1087-1106.
- [47] "Basics of rotary encoders: Overview and new technologies," <http://machinedesign.com/sensors/basics-rotary-encoders-overview-and-new-technologies-0>
- [48] Batten C., "Control for Mobile Robots," *Maslab IAP Robotics Course*, 2005.
- [49] Singh B, Payasi R P, Verma K S, et al., "Design of Controllers PD, PI & PID for Speed Control of DC Motor Using IGBT Based Chopper," *German Journal of Renewable and Sustainable Energy Research (GJRSER)*, 2013, 1(1).
- [50] Francis B A, Wonham W M., "The internal model principle of control theory," *Automatica*, 1976, 12(5): 457-465.
- [51] Feng L, Koren Y, Borenstein J., "Cross-coupling motion controller for mobile robots," *IEEE Control Systems*, 1993, 13(6): 35-43.
- [52] Rodriguez, A., *Analysis and Design of Feedback Control Systems*, Control3D,L.L.C., Tempe, AZ, 2002.
- [53] Puttannaiah K, Echols J, and Rodriguez A, "A Generalized \mathcal{H}^∞ Control Design Framework for Stable Multivariable Plants subject to Simultaneous Output and Input Loop Breaking Specifications," American Control Conf. (ACC), IEEE, 2015.
- [54] Puttannaiah K, Echols J, Mondal K and Rodriguez A, "Analysis and Use of Several Generalized H-Infinity Mixed Sensitivity Framework for Stable Multivariable Plants Subject to Simultaneous Output and Input Loop Breaking Specifications," Accepted for publication in Conf. on Decision and Control (CDC), IEEE, 2015.
- [55] Rodriguez, A., EEE481: Computer Control Systems, course notes, 2014.
- [56] Kaplan, Elliott, and Christopher Hegarty, eds., "Understanding GPS: principles and applications." Artech house, 2005.
- [57] Raspberry Pi 5MP camera, <https://www.raspberrypi.org/products/camera-module-v2/>.

- [58] Building an Arduino-based self-balancing robot, <https://roboticdreams.wordpress.com/>
- [59] Anvari I., *Non-holonomic Differential-Drive Mobile Robot Control & Design: Critical Dynamics and Coupling Constraints*, Arizona State University, MS Thesis, 2013.
- [60] Aldaco J., *Image Processing Based Control of Mobile Robotics*, Arizona State University, MS Thesis, 2016.
- [61] R. W. Brockett, “Asymptotic stability and feedback stabilization,” in R. W. Brockett, R. S. Millman, and H. J. Sussmann, editors, *Differential Geometric Control Theory*, Birkhauser, Boston, MA, 1983
- [62] Kolmanovsky I, McClamroch N H., “Developments in nonholonomic control problems,” *IEEE Control Systems*, 1995, 15(6): 20-36.
- [63] A. Astolfi, “On the Stabilization of Nonholonomic Systems,” *Proceedings of the 33rd IEEE Conference on Decision and Control*, vol.4, no., pp. 3481-3486, vol.4, 14-16 Dec 1994.
- [64] S.C. Warnick and A.A. Rodriguez, “A Systematic Anti-windup Strategy and the Longitudinal Control of a Platoon of Vehicles with Control Saturation,” *IEEE Transactions on Vehicular Technology*, Vol. 49, No. 3, May 2000, pp. 1006-101
- [65] J.K. Hedrick and A. Girard, *Control of Nonlinear Dynamic Systems: Theory and Applications*, 2005.
- [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/>
- [67] Wi-Fi technology, <https://en.wikipedia.org/wiki/Wi-Fi> 2015
- [68] HCSR04 Ultrasonic sensor, <http://www.micropik.com/PDF/HCSR04.pdf>
- [69] G. Stein, “Respect the Unstable,” *IEEE Control Systems Magazine*, 2003.

APPENDIX A

C CODE

```

1
2 #ifndef INCLUDE_CAMERA_H_
3 #define INCLUDE_CAMERA_H_
4
5
6 struct camera_para {
7
8     int inverse;
9     int th;
10
11    int roi_x;
12    int roi_y;
13    int roi_width;
14    int roi_height;
15
16
17};
18
19
20
21 extern struct camera_para cpara;
22
23 extern volatile int camera_should_stop;
24
25
26
27
28 void *camera_thread(void *para);
29
30
31 #endif /* INCLUDE_CAMERA_H_ */

```

```

1
2
3 #ifndef CTRL_LOOP_H_
4 #define CTRL_LOOP_H_
5
6 #include <pthread.h>
7 #include <semaphore.h>
8
9 struct h2l_vehicle_status;
10
11 enum control_mode_t {
12
13     CTRL_OPEN_LOOP = 0,
14
15     CTRL_WLWR,
16     CTRL_V_OMEGA,
17
18     CTRL_V_THETA,
19     CTRL_X_Y,
20     CTRL_DELTA_X_THETA,
21
22     CTRL_LINE_TRACK,
23     CTRL_PLATOONING
24
25 };
26
27 enum control_flag_t {
28
29     CTRL_NONE,
30     CTRL_START,
31     CTRL_STOP
32
33 };
34
35 struct vehicle_state {
36
37     // controller state, it can be 0 (stop)

```

```

38     // or 1 (running)
39
40     int ctrl_state; // running or stopped
41     int ctrl_flag;
42     int ctrl_outer_loop_mode;
43     int ctrl_inner_loop_mode;
44
45     // sensor measured states (low level)
46
47     uint32_t timestamp;
48     float imu_theta;
49     float camera_delta_theta;
50     float camera_delta_theta_p;
51     float imu_accx;
52     float imu_omega;
53     float encoder_wl;
54     float encoder_wr;
55     float ultrasonic_delta_x;
56     float ultrasonic_delta_x_bk;
57     float ultrasonic_delta_x_fitered;
58
59     // calculated / fused states
60
61     // NOTE: these states are in XI-O-YI coordinates
62     double accx;
63
64     double v;
65     double omega;
66
67     double x;
68     double y;
69     double theta;
70     // new 0801 for platoon faster start
71     double v_dsr;
72
73 };
74
75 // Inner loop parameter (previously in mc_init)
76 struct ctrl_inner_para {
77
78     float prefilter_coefficient;
79     float roll_off_coefficient;
80     float kp_left;
81     float ki_left;
82     float kd_left;
83     float kp_right;
84     float ki_right;
85     float kd_right;
86     float deadzone_threshold;
87     float deadzone_saturation;
88
89 };
90
91
92
93
94 // These are states only used by a specific controller
95
96 // If you need to hack the controller or add a new controller,
97 // just add variables in these structure
98 // or create a new structure.
99
100 struct ctrl_open_loop_state {
101
102     // desired value
103     int16_t pwm_left_dsr;
104     int16_t pwm_right_dsr;
105
106
107 };

```

```

108
109 struct ctrl_wl_wr_state {
110
111     // desired value
112     double wl_dsr;
113     double wr_dsr;
114
115 };
116
117 struct ctrl_v_omega_state {
118
119     // desired value
120     double v_dsr;
121     double omega_dsr;
122
123 };
124
125
126
127 // outer loop v_theta control
128
129 struct ctrl_v_theta_para {
130
131     double kp_theta;
132     double ki_theta;
133     double kd_theta;
134
135
136 };
137
138
139 struct ctrl_v_theta_state {
140
141     // desired value
142     double v_dsr;
143     double theta_dsr;
144
145
146     double wl_dsr;
147     double wr_dsr;
148     double u_v;
149     double u_omega;
150
151     double error_theta;
152     double error_theta_p;
153     double error_theta_sum_p;
154
155     double u_omega_rf1_p;
156     double u_omega_rf2_p;
157
158     // debug
159
160     double up_omega_out;
161     double ui_omega_out;
162     double ud_omega_out;
163
164     double u_omega_wo_rf; // without roll off
165
166     FILE *log_file;
167     FILE *recent_log_file;
168
169 };
170
171
172 // outer loop x_y control
173
174 struct ctrl_x_y_para {
175
176     double kp_dist;
177     double ki_dist;

```

```

178     double kd_dist;
179
180     double kp_angle;
181     double ki_angle;
182     double kd_angle;
183
184 };
185
186 struct ctrl_x_y_state {
187
188     // desired value
189     double x_dsr;
190     double y_dsr;
191
192     double wl_dsr;
193     double wr_dsr;
194     double u_v;
195     double u_omega;
196
197     double error_dist;
198     double error_dist_p;
199     double error_dist_sum_p;
200
201 // double u_v_rf1_p;
202 // double u_v_rf2_p;
203
204     double error_angle;
205     double error_angle_p;
206     double error_angle_sum_p;
207
208 // double u_omega_rf1_p;
209 // double u_omega_rf2_p;
210
211     // debug
212
213     double up_dist_out;
214     double ui_dist_out;
215     double ud_dist_out;
216     double up_angle_out;
217     double ui_angle_out;
218     double ud_angle_out;
219
220     FILE *log_file;
221     FILE *recent_log_file;
222
223 };
224
225 // outer loop delta_x_theta
226
227 struct ctrl_delta_x_theta_para {
228
229     double kp_delta_x;
230     double ki_delta_x;
231     double kd_delta_x;
232
233     // NOTE: Currently only kp_theta is in usage
234     double kp_theta;
235     double ki_theta;
236     double kd_theta;
237
238 };
239
240 struct ctrl_delta_x_theta_state {
241
242     // desired value
243     double delta_x_dsr;
244     double theta_dsr;
245
246     // previous state and current intermediate state
247

```

```

248     double wl_dsr;
249     double wr_dsr;
250
251     double u_v;
252     double u_omega;
253
254     double error_delta_x;
255     double error_delta_x_p;
256     double error_delta_x_pp;
257
258     double u_v_rf1_p;
259     double u_v_rf2_p;
260
261     double u_v_out_p;
262
263
264     double error_theta;
265     double error_theta_p;
266     double error_theta_sum_p;
267
268     double u_omega_rf1_p;
269     double u_omega_rf2_p;
270
271     // debug state
272     double up_v_out;
273     double ui_v_out;
274     double ud_v_out;
275     double u_v_out;
276
277     double up_omega_out;
278     double ui_omega_out;
279     double ud_omega_out;
280     double u_omega_out;
281
282
283     FILE *log_file;
284     FILE *recent_log_file;
285 };
286
287
288
289 // outer loop line track
290
291 struct ctrl_line_track_para {
292
293     double kp;
294     double ki;
295     double kd;
296
297
298 };
299
300
301 struct ctrl_line_track_state {
302
303     // desired value
304     double v_dsr;
305
306     double wl_dsr;
307     double wr_dsr;
308     double u_v;
309     double u_omega;
310
311     double delta_theta;
312     double delta_theta_p;
313     double delta_theta_sum_p;
314
315     double u_omega_rf1_p;
316     double u_omega_rf2_p;
317
318     // debug

```

```

319
320     double up_omega_out;
321     double ui_omega_out;
322     double ud_omega_out;
323     double u_omega_out;
324
325     FILE *log_file;
326     FILE *recent_log_file;
327
328 };
329
330
331 // platooning
332
333 struct ctrl_platooning_para {
334
335     int leader_id;
336
337     double kp_delta_x;
338     double ki_delta_x;
339     double kd_delta_x;
340
341     double kp_ffleader;
342     double ki_ffleader;
343     double kd_ffleader;
344
345     double ka_ffleader;
346
347     double kp_ffpre;
348     double ki_ffpre;
349     double kd_ffpre;
350
351     double ka_ffpre;
352
353     // NOTE: Currently only kp_theta is in usage
354     double kp_theta;
355     double ki_theta;
356     double kd_theta;
357
358
359 };
360
361
362 struct ctrl_platooning_state {
363
364     // desired value
365     double delta_x_dsr;
366     double theta_dsr;
367
368     // previous state and current intermediate state
369     double v_dsr_leader;
370     double v_leader;
371     double v_leader_p;
372     double acc_leader;
373     int ctrl_flag_leader;
374
375
376     double v_pre;
377     double acc_pre;
378     double x_leader_p;
379     double x_leader;
380
381     double wl_dsr;
382     double wr_dsr;
383
384     double u_v;
385     double u_v_rf_p;
386     double u_omega;
387
388     // state derived from delta_x
389     double u_v_x;

```

```

390
391     double error_delta_x;
392     double error_delta_x_p;
393     double error_delta_x_pp;
394
395     double delta_x_filtered;
396
397     double u_v_x_rf1_p;
398     double u_v_x_rf2_p;
399
400     // debug state (delta_x)
401     double up_v_x_out;
402     double ui_v_x_out;
403     double ud_v_x_out;
404     double u_v_x_out_p;
405     double u_v_x_out;
406
407
408
409     // state derived from leader
410     double u_v_ffleader;
411
412     double error_leader;
413     double error_leader_p;
414     // double error_leader_pp;
415     double error_leader_sum_p;
416     double u_v_ffleader_rf_p;
417
418     double u_v_ffleader_acc;
419
420     // debug state (leader)
421
422     double up_v_ffleader_out;
423     double ui_v_ffleader_out;
424     double ud_v_ffleader_out;
425     double u_v_ffleader_out_p;
426     double u_v_ffleader_out;
427     double u_v_ffleader_rf_before_sat;
428
429
430     // state derived from previous vehicle
431     double u_v_ffpre;
432
433     double error_pre;
434     double error_pre_p;
435     double error_pre_pp;
436
437     double u_v_ffpre_rf_p;
438
439     double u_v_ffpre_acc;
440
441     // debug state (leader)
442
443     double up_v_ffpre_out;
444     double ui_v_ffpre_out;
445     double ud_v_ffpre_out;
446     double u_v_ffpre_out_p;
447     double u_v_ffpre_out;
448
449
450     // state derived from theta
451     double error_theta;
452     double error_theta_p;
453     double error_theta_sum_p;
454
455     double u_omega_rf1_p;
456     double u_omega_rf2_p;
457
458
459     double up_omega_out;
460     double ui_omega_out;

```

```

461     double ud_omega_out;
462     double u_omega_wo_rf;
463
464     FILE *log_file;
465     FILE *recent_log_file;
466
467 };
468
469
470
471
472 // exported global variable
473
474
475 extern volatile int ctrl_loop_should_stop;
476 extern sem_t ctrl_loop_sem;
477
478 extern int ctrl_loop_current_rt;
479
480 // vehicle state
481
482 extern volatile struct vehicle_state vstate;
483
484 // states and parameter of inner loop
485
486 extern volatile struct ctrl_inner_para ctrl_inner_para;
487
488 // states and parameter of outer loop
489
490 extern volatile struct ctrl_open_loop_state
491 ctrl_open_loop_state;
492
493 extern volatile struct ctrl_wl_wr_state
494 ctrl_wl_wr_state;
495
496 extern volatile struct ctrl_v_omega_state
497 ctrl_v_omega_state;
498
499 extern volatile struct ctrl_v_theta_para
500 ctrl_v_theta_para;
501 extern volatile struct ctrl_v_theta_state
502 ctrl_v_theta_state;
503
504 extern volatile struct ctrl_x_y_para ctrl_x_y_para;
505 extern volatile struct ctrl_x_y_state ctrl_x_y_state;
506
507 extern volatile struct ctrl_delta_x_theta_para
508 ctrl_delta_x_theta_para;
509 extern volatile struct ctrl_delta_x_theta_state
510 ctrl_delta_x_theta_state;
511
512 extern volatile struct ctrl_line_track_para
513 ctrl_line_track_para;
514 extern volatile struct ctrl_line_track_state
515 ctrl_line_track_state;
516
517 extern volatile struct ctrl_platooning_para
518 ctrl_platooning_para;
519 extern volatile struct ctrl_platooning_state
520 ctrl_platooning_state;
521
522
523
524
525
526
527 void update_vehicle_state( struct h2l_vehicle_status *state );
528
529 void *ctrl_loop_thread( void *para );
530
531 void ctrl_setup();

```

```

532 void ctrl_start();
533 void ctrl_stop();
534
535 void ctrl_open_loop_set_pwm(int16_t pwm_left, int16_t pwm_right);
536 void ctrl_set_wl_wr(double wl_dsr, double wr_dsr);
537 void ctrl_set_v_omega(double v_dsr, double omega_dsr);
538 void ctrl_set_v_theta(double v_dsr, double theta_dsr);
539 void ctrl_set_x_y(double x_dsr, double y_dsr);
540 void ctrl_set_delta_x_theta(double delta_x_dsr, double theta_dsr);
541 void ctrl_set_line_track(double v_dsr);
542 void ctrl_set_platooning(double delta_x_dsr, double theta_dsr);
543 double ctrl_outer_loop_threshold_and_saturation_v
544 (double In, double Th, double Min, double Max);
545 double ctrl_outer_loop_threshold_and_saturation_w
546 (double In, double Th, double Min, double Max);
547
548 #endif /* CTRL_LOOP_H */

```

```

1
2 #include <stdint.h>
3 #include <string.h>
4
5 struct mc_init;
6
7
8 #ifndef H2L_PROTOCOL_H_
9 #define H2L_PROTOCOL_H_
10
11
12
13
14 #define MC_PROTO_HEADER_SIZE 4
15
16 #define SERIAL_STATE_INIT 0
17 #define SERIAL_STATE_MAGIC1 1
18 #define SERIAL_STATE_MAGIC2 2
19 #define SERIAL_STATE_PROTO 3
20
21 #define SERIAL_MAGIC_1 'A'
22 #define SERIAL_MAGIC_2 'F'
23
24 #define OPCODE_OPEN_LOOP 0x00
25
26 #define OPCODE_CTRL_WL_WR 0x10
27
28 #define OPCODE_SETUP 0xF0
29 #define OPCODE_START 0xF1
30 #define OPCODE_STOP 0xF2
31 #define OPCODE_DEBUG_ENABLE 0xF3
32 #define OPCODE_DEBUG_DISABLE 0xF4
33
34
35 #define OPCODE_VEHICLE_STATUS 0xE0
36 #define OPCODE_CTRL_STATUS_DEBUG 0xE1
37
38
39 struct h2l_header {
40
41     uint8_t magic1;
42     uint8_t magic2;
43     uint8_t len;
44     uint8_t opcode;
45 };
46
47 // message from HLC to LLC
48
49 struct h2l_open_loop {
50

```

```

51     struct h2l_header header;
52     int16_t v_left;
53     int16_t v_right;
54 }
55 };
56
57 struct h2l_wl_wr {
58
59     struct h2l_header header;
60     float wl_dsr;
61     float wr_dsr;
62 };
63
64 struct h2l_setup {
65
66     struct h2l_header header;
67     float roll_off_coefficient;
68     float prefilter_coefficient;
69     float kp_left;
70     float ki_left;
71     float kd_left;
72     float kp_right;
73     float ki_right;
74     float kd_right;
75     float deadzone_threshold;
76     float deadzone_saturation;
77
78 };
79
80
81
82
83
84
85
86 // message from LLC to HLC
87
88 struct h2l_vehicle_status {
89
90     struct h2l_header header;
91
92     uint32_t timestamp;
93     float imu_theta;      // from IMU
94     float imu_accx;      // from IMU
95     float imu_omega;      // from IMU
96     float encoder_wl;
97     float encoder_wr;
98     float ultrasonic_delta_x;
99     // from ultrasonic sensor
100 };
101
102
103
104 struct h2l_ctrl_status_debug {
105
106     struct h2l_header header;
107     uint32_t timestamp; // Arduino timestamp in ms
108
109     float wl_dsr;
110     float wr_dsr;
111     float wl_dsr_filtered;
112     float wr_dsr_filtered;
113
114     // debug
115
116     int16_t pwml; // with roll off
117     int16_t pwmr;
118     int16_t pwml_out; // without roll off
119     int16_t pwmr_out;
120     int16_t pwml_out_p;

```

```

121     int16_t pwmr_out_p;
122
123     float err_wl;
124     float err_wr;
125     float err_wl_p;
126     float err_wr_p;
127     float err_wl_pp;
128     float err_wr_pp;
129
130     float pwml_up;
131     float pwml_ui;
132     float pwml_ud;
133     float pwmr_up;
134     float pwmr_ui;
135     float pwmr_ud;
136
137     uint32_t timestamps[8];
138 }
139
140
141
142
143 // helper function
144
145 static inline void set_uint16(char *buff, int offset, uint16_t value) {
146
147     char *p = (char *)&value;
148     memcpy(buff + offset, p, sizeof(value));
149 }
150
151 static inline void set_uint32(char *buff, int offset, uint32_t value) {
152
153     char *p = (char *)&value;
154     memcpy(buff + offset, p, sizeof(value));
155 }
156
157 static inline void set_float(char *buff, int offset, float value) {
158
159     char *p = (char *)&value;
160     memcpy(buff + offset, p, sizeof(value));
161 }
162
163
164 static inline uint16_t get_uint16(char *buff, int offset) {
165
166     uint16_t ret;
167     uint16_t *p = &ret;
168
169     memcpy(p, buff + offset, sizeof(ret));
170
171     return ret;
172 }
173
174 static inline uint32_t get_uint32(char *buff, int offset) {
175
176     uint32_t ret;
177     uint32_t *p = &ret;
178
179     memcpy(p, buff + offset, sizeof(ret));
180
181     return ret;
182 }
183
184 static inline float get_float(char *buff, int offset) {
185
186     float ret;
187     float *p = &ret;
188
189     memcpy(p, buff + offset, sizeof(ret));

```

```

190         return ret;
191     }
192 }
193
194 static inline void h2l_set_header(struct h2l_header *pheader,
195                                 uint8_t len, uint8_t opcode) {
196
197     pheader->magic1 = SERIAL_MAGIC_1;
198     pheader->magic2 = SERIAL_MAGIC_2;
199     pheader->len = len;
200     pheader->opcode = opcode;
201 }
202
203
204 // send command message to LLC
205
206 // CAUTION: only call these function in control loop
207
208 int h2l_send_open_loop_msg(int16_t pwm_left, int16_t pwm_right);
209 int h2l_send_wl_wr_msg(float wl_dsr, float wr_dsr);
210 int h2l_send_setup_msg();
211 int h2l_send_start_msg();
212 int h2l_send_stop_msg();
213 int h2l_send_debug_enable_msg();
214 int h2l_send_debug_disable_msg();
215
216
217 //void h2l_print_vehicle_status(FILE *fd,
218 // struct h2l_vehicle_status *msg);
219 void h2l_print_ctrl_status_debug(FILE *fd,
220                                 struct h2l_ctrl_status_debug *msg);
221
222 // h2l receiving thread
223 //(note that the sending thread is control loop)
224 void *h2l_receive_thread(void *para);
225
226
227
228 extern volatile int serial_should_stop;
229 extern int serial_fd;
230
231 #endif /* H2L_PROTOCOL_H_ */

1 #ifndef PI_MC_H_
2 #define PI_MC_H_
3
4 struct mc_init {
5
6     int vehicle_id;
7
8     char tty_file_name[64];
9     int tty_baud_rate;
10
11
12     char log_file_name[64];
13     char meta_file_name[64];
14
15     // v2i and v2v parameter
16
17     char manager_ip[16];
18     int manager_port;
19
20     int v2v_period;
21
22     // general controller parameter
23

```

```

24     int running_time; // in milliseconds
25
26     int ctrl_loop_period; // in milliseconds
27
28     int debug_mode;
29
30     double wheel_radius;
31     double distance_of_wheels;
32
33 };
34
35 extern struct mc_init mc_init;
36
37 extern FILE *meta_file;
38 extern FILE *log_file;
39 extern FILE *recent_log_file;
40 extern FILE *recent_meta_file;
41
42 #endif /* PI_MC_H_ */

1 #include <stddef.h>
2
3 #ifndef SERIAL_H_
4 #define SERIAL_H_
5
6
7 int serial_init(const char* serialport, int baud);
8 int serial_close(int fd);
9
10 // CAUTION: these send and receiving are blocking
11 // busy waiting
12 int serial_send_n_bytes(int fd, const char* buff, size_t n);
13 int serial_recv_n_bytes(int fd, char *buff, size_t n);
14
15 int serial_flush(int fd);
16
17 // CAUTION: this is nonblocking receive
18 int serial_recv_byte(int fd, char *buff);
19
20
21 #endif /* SERIAL_H_ */

1
2 #ifndef V2V_H_
3 #define V2V_H_
4
5
6
7
8
9 struct v2v_msg {
10
11     uint32_t ts;
12     uint32_t vid;
13     double theta;
14     double accx;
15     double v;
16     double omega;
17     double v_dsr;
18     int ctrl_flag;
19
20 };
21
22 struct v2v_msg *get_recent_msg(int vid);
23 void print_v2v_msg(FILE *fd, struct v2v_msg *msg);
24

```

```

25 void *v2v_sending_thread(void *para);
26 void *v2v_receiving_thread(void *para);
27
28 extern volatile int v2v_sending_should_stop;
29 extern volatile int v2v_receiving_should_stop;
30
31 extern sem_t v2v_send_sem;
32
33 #endif /* V2V_H_ */

1 # Configuration file for motion control program
2 # running on high level
3 # controller (HLC)
4
5 vehicle_id=5
6
7 tty_file_name=/dev/ttyACM0
8 tty_baud_rate=115200
9
10 log_file_name=xxx.txt
11 meta_file_name=yyy.txt
12
13
14 # ----- v2i and v2v parameter -----
15
16 manager_ip=192.168.0.150
17 manager_port=50000
18
19 # v2v communication period in millisecond
20 v2v_period=100
21
22 # ----- camera parameter -----
23
24 # For tracking white tape, camera_inverse=0.
25 camera_inverse=0
26 camera_threshold=67
27
28 camera_roi_x=0
29 camera_roi_y=70
30 camera_roi_width=320
31 camera_roi_height=20
32
33
34 # ----- general controller parameter -----
35
36 # running time in millisecond
37 running_time=60000
38
39
40 # control outer loop period in millisecond
41 ctrl_loop_period=100
42
43
44
45
46
47 # The controller mode can be the following:
48 # 0 (open loop)
49 # 1 (close loop, wl_wr)
50 # 2 (close loop, v_omega)
51 # 3 (close loop, v_theta)
52 # 4 (close loop, x_y)
53 # 5 (close loop, delta_x_theta)
54 # 6 (close loop, line_track)
55 # 7 (close loop, platooning)
56
57 controller_mode=7
58 debug_mode=1

```

```

59
60   wheel_radius=0.05
61   distance_of_wheels=0.14
62
63   # ----- parameter of outer loop -----
64
65   # no parameter for outer loop v_omega control
66
67   # outer loop v_theta control
68   # CAUTION: outer loop only control theta
69
70   outer_v_theta_kp_theta=3
71   outer_v_theta_ki_theta=0
72   outer_v_theta_kd_theta=4
73
74   # outer loop line_track
75   outer_line_track_kp=3
76   outer_line_track_ki=0
77   outer_line_track_kd=4
78
79
80   # outer loop x_y control
81   outer_x_y_kp_dist=1.2
82   outer_x_y_ki_dist=0
83   outer_x_y_kd_dist=0
84
85   outer_x_y_kp_angle=1.5
86   outer_x_y_ki_angle=0
87   outer_x_y_kd_angle=0
88
89   # outer loop delta_x_theta
90   outer_delta_x_theta_kp_delta_x=1
91   outer_delta_x_theta_ki_delta_x=0
92   outer_delta_x_theta_kd_delta_x=0
93
94   outer_delta_x_theta_kp_theta=3
95   outer_delta_x_theta_ki_theta=0
96   outer_delta_x_theta_kd_theta=4
97
98   # outer loop platooning
99
100  outer_platooning_leader_id=1
101
102  outer_platooning_kp_delta_x=1
103  outer_platooning_ki_delta_x=0.3
104  outer_platooning_kd_delta_x=0.3
105
106  outer_platooning_kp_ffleader=0.4
107  outer_platooning_ki_ffleader=0.6
108  outer_platooning_kd_ffleader=0
109
110  outer_platooning_ka_ffleader=0
111
112  outer_platooning_kp_ffpre=0
113  outer_platooning_ki_ffpre=0
114  outer_platooning_kd_ffpre=0
115
116  outer_platooning_ka_ffpre=0
117
118  outer_platooning_kp_theta=3
119  outer_platooning_ki_theta=0
120  outer_platooning_kd_theta=4
121
122
123   # ----- parameter of inner loop -----
124
125   # prefilter_coefficient=1
126   prefilter_coefficient=0.1667
127
128   roll_off_coefficient=0.8
129
130   inner_kp_left=0.29

```

```

131 inner_ki_left=0.58
132 inner_kd_left=0
133
134 inner_kp_right=0.29
135 inner_ki_right=0.58
136 inner_kd_right=0
137
138
139 deadzone_threshold=0
140 deadzone_saturation=0
141
142 # ----- initial value of control variable -----
143
144
145
146 # Initialization value for open loop
147 pwm_left_init=100
148 pwm_right_init=100
149
150
151 # Initialization value for outer loop
152
153 wl_dsr_init=10
154 wr_dsr_init=10
155
156 # Initialization value for outer loop
157
158 v_dsr_init=0.4
159 omega_dsr_init=0
160
161
162
163 outer_v_theta_v_dsr_init=0.4
164 outer_v_theta_theta_dsr_init=0
165
166
167 outer_x_y_x_dsr_init=1.5
168 outer_x_y_y_dsr_init=1.5
169
170
171 outer_delta_x_theta_delta_x_dsr_init=0.4
172 outer_delta_x_theta_theta_dsr_init=0
173
174
175 outer_platooning_delta_x_dsr_init=0.4
176 outer_platooning_theta_dsr_init=0
177
178 outer_line_track_v_dsr_init=0.3

```

```

1
2 #include <opencv2/core/core.hpp>
3 #include <opencv2/highgui/highgui.hpp>
4 #include <opencv2/imgproc/imgproc.hpp>
5
6 #include <iostream>
7 #include <fstream>
8
9 #include <ctime>
10
11 #include <unistd.h>
12
13 #include "raspicam/raspicam_cv.h"
14
15 #include "ctrl_loop.h"
16 #include "pi_mc.h"
17 #include "camera.h"
18
19 using namespace cv;
20 using namespace raspicam;

```

```

21  using namespace std;
22
23  struct camera_para cpara;
24
25  volatile int camera_should_stop;
26
27  long get_millis() {
28
29      struct timespec spec;
30
31      clock_gettime(CLOCK_MONOTONIC, &spec);
32
33      long ms = spec.tv_sec * 1000 + spec.tv_nsec / 1000000;
34
35      return ms;
36  }
37
38  /** Sets a property in the VideoCapture.
39  *
40  *
41  * Implemented properties:
42  *   * CV_CAP_PROP_FRAME_WIDTH, CV_CAP_PROP_FRAME_HEIGHT,
43  *   * CV_CAP_PROP_FORMAT: CV_8UC1 or CV_8UC3
44  *   * CV_CAP_PROP_BRIGHTNESS: [0,100]
45  *   * CV_CAP_PROP_CONTRAST: [0,100]
46  *   * CV_CAP_PROP_SATURATION: [0,100]
47  *   * CV_CAP_PROP_GAIN: (iso): [0,100]
48  *   * CV_CAP_PROP_EXPOSURE: -1 auto. [1,100]
49  * //shutter speed from 0 to 33ms
50  *   * CV_CAP_PROP_WHITE_BALANCE_RED_V : /
51  *   // [1,100] -1 auto whitebalance
52  *   * CV_CAP_PROP_WHITE_BALANCE_BLUE_U :
53  *   // [1,100] -1 auto whitebalance
54  *
55  */
56 // LEE
57 // 0725 edited the original library add new set feature by
58 // adding two more cases in switch section in raspicam_cv.cpp
59
60
61 void set_picam_property(RaspiCam_Cv &picam) {
62     picam.set(CV_CAP_PROP_FRAME_WIDTH, 320);
63     picam.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
64     picam.set(CV_CAP_PROP_FORMAT, CV_8UC3);
65         picam.set(777, 1); // Hflip
66         picam.set(888, 1); // Vflip
67 //     picam.set(CV_CAP_PROP_BRIGHTNESS, 50);
68 //     picam.set(CV_CAP_PROP_CONTRAST, 50);
69 //     picam.set(CV_CAP_PROP_SATURATION, 50);
70 //     picam.set(CV_CAP_PROP_GAIN, 50);
71 //     picam.set(CV_CAP_PROP_EXPOSURE, -1);
72 //     picam.set(CV_CAP_PROP_WHITE_BALANCE_RED_V, -1);
73 //     picam.set(CV_CAP_PROP_WHITE_BALANCE_BLUE_U, -1);
74 }
75
76 double delta_theta_bk = 0;
77 void *camera_thread(void *para) {
78
79     ofstream camlog("camera.txt");
80
81     RaspiCam_Cv picam;
82
83     set_picam_property(picam);
84
85     if (!picam.open()) {
86         cerr << "Error opening camera" << endl;
87     }

```

```

88
89     cout << "<camera> Connected to camera ="
90     << picam.getId() << endl;
91
92     double frame_width = picam.get(CV_CAP_PROP_FRAME_WIDTH);
93     double frame_height = picam.get(CV_CAP_PROP_FRAME_HEIGHT);
94
95     cout << "Frame Size = " << frame_width
96     << "x" << frame_height << endl;
97     cout << "ROI Size = " << cpara.roi_width
98     << "x" << cpara.roi_height << endl;
99
100    Size frame_size(static_cast<int>(frame_width),
101                    static_cast<int>(frame_height));
102
103    VideoWriter writer("./camera.avi",
104                        CV_FOURCC('P', 'I', 'M', '1'),
105                        20, frame_size, true);
106    //initialize the VideoWriter object
107    //if not initialize the VideoWriter successfully,
108    //exit the program
109    if (!writer.isOpened())
110    {
111        cout << "ERROR: Failed to write the frame video"
112        << endl;
113        return NULL;
114    }
115
116    Size roi_size(static_cast<int>(cpara.roi_width),
117                  static_cast<int>(cpara.roi_height));
118
119    VideoWriter roi_writer("./roi.avi",
120                          CV_FOURCC('P', 'I', 'M', '1'),
121                          20, roi_size, true);
122    //initialize the VideoWriter object
123    //if not initialize the VideoWriter successfully,
124    //exit the program
125    if (!roi_writer.isOpened())
126    {
127        cout << "ERROR: Failed to write the ROI video" << endl;
128        return NULL;
129    }
130
131
132    // namedWindow("Control", CV_WINDOW_AUTOSIZE);
133    // createTrackbar("threshold", "Control", &th, 255);
134
135    long ts = get_millis();
136 ;
137
138    while (!camera_should_stop) {
139
140        Mat frame;
141        Mat imgrey;
142        Mat imthresh;
143
144
145        picam.grab();
146        picam.retrieve(frame);
147
148        // convert to grey scale
149        cvtColor(frame, imgrey, CV_RGB2GRAY);
150        // binary threshold
151        threshold(imgrey, imthresh, cpara.th, 255,
152                  THRESH_BINARY_INV);
153
154        // get subimage

```

```

155 //      Rect rect(80, 20, 160, 100);
156 //      Rect rect(cpara.roi_x, cpara.roi_y,
157 //                  cpara.roi_width, cpara.roi_height);
158 Mat roi = imthresh(rect);
159
160 // morphological opening
161 // (remove small objects from the foreground)
162 erode(roi, roi,
163         getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
164 dilate(roi, roi,
165         getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
166
167 // morphological closing (fill small holes in the foreground)
168 dilate(roi, roi,
169         getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
170 erode(roi, roi,
171         getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
172
173 // calculate the moments of the thresholded image
174 Moments m = moments(roi);
175
176 double m01 = m.m01;
177 double m10 = m.m10;
178 double m00 = m.m00;
179
180 double u1 = m10 / m00;
181 double v1 = m01 / m00;
182
183 //      double width = cpara.roi_width;
184 //      double height = cpara.roi_height;
185 //
186 //      double delta_theta_rad = atan((u1 - width / 2) / height);
187 //      double delta_theta_degree = delta_theta_rad * 180 / 3.1416;
188
189 double width = cpara.roi_width;
190 double height = cpara.roi_height;
191 // 32 = 20 (Look ahead) + 12(rear wheel 2 camera)
192 // vertical distance calibration 32cm
193 double lat_pixel_2_cm_gain = (1.7/26.0);
194 double lat_err_cm = (u1 - width / 2)* lat_pixel_2_cm_gain;
195 double delta_theta_rad = -atan(lat_err_cm / 32.0);
196
197 // camera failure handling
198 // 0.32 is the maximum delta_theta we can get
199 // given 20cm look ahead distance
200 if (fabs(delta_theta_rad)>=0.32){
201     cout << "<camera> off track" << endl;
202     delta_theta_bk = delta_theta_rad;
203 }
204 else if (u1>=10 && u1<=310){
205     //update backup data
206     delta_theta_bk = delta_theta_rad;
207 }
208 else if (u1<10 || u1>310){
209     //update backup data
210     //cout << "<camera> error " << endl;
211     delta_theta_rad= 0;
212     delta_theta_bk = 0;
213 } else{
214     ;
215 }
216 double delta_theta_degree = delta_theta_rad * 180 / 3.1416;
217
218 // show the ROI and the image
219
220 //      rectangle(frame, rect, Scalar(127, 127, 127));

```

```

221 //      imshow(" original", frame); //show the original image
222 //      imshow(" roi", roi);
223
224         long t2 = get_millis();
225
226         camlog << "FPS: " << 1000 / (t2 - ts) << ", ";
227         camlog << "center of mass: (" << ul << ", " << v1;
228         camlog << ") delta_theta: " << delta_theta_rad << "("
229             << delta_theta_degree << ")");
230         camlog << " lat_err_cm: " << lat_err_cm << endl;
231
232         ts = t2;
233
234         vstate.camera_delta_theta = delta_theta_rad;
235
236         Mat roi_rgb;
237         cvtColor(roi, roi_rgb, CV_GRAY2RGB);
238
239         writer.write(frame);
240         roi_writer.write(roi_rgb);
241
242         // sleep 20 ms
243         usleep(20000);
244
245     }
246
247     picam.release();
248
249     cout << "<camera> camera thread exiting ..." << endl;
250
251     return 0;
252
253 }
254 }
```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <errno.h>
5
6 #include <math.h>
7
8 #include <unistd.h>
9 #include <sys/types.h>
10
11 #include "h2l.h"
12 #include "ctrl_loop.h"
13 #include "v2v.h"
14 #include "pi_mc.h"
15
16 volatile int ctrl_loop_should_stop = 0;
17 sem_t ctrl_loop_sem;
18
19 int ctrl_loop_current_rt = 0;
20
21 // ----- vehicle state and controller state -----
22
23 volatile struct vehicle_state vstate;
24
25 // states and parameter of inner loop
26
27 volatile struct ctrl_inner_para ctrl_inner_para;
28
29 // states and parameter of outer loop
30
31 volatile struct ctrl_open_loop_state ctrl_open_loop_state;
32
33 volatile struct ctrl_wl_wr_state ctrl_wl_wr_state;
```

```

34
35 volatile struct ctrl_v_omega_state ctrl_v_omega_state;
36
37 volatile struct ctrl_v_theta_para ctrl_v_theta_para;
38 volatile struct ctrl_v_theta_state ctrl_v_theta_state;
39
40 volatile struct ctrl_x_y_para ctrl_x_y_para;
41 volatile struct ctrl_x_y_state ctrl_x_y_state;
42
43 volatile struct ctrl_delta_x_theta_para
44 ctrl_delta_x_theta_para;
45 volatile struct ctrl_delta_x_theta_state
46 ctrl_delta_x_theta_state;
47
48 volatile struct ctrl_line_track_para
49 ctrl_line_track_para;
50 volatile struct ctrl_line_track_state
51 ctrl_line_track_state;
52
53 volatile struct ctrl_platooning_para
54 ctrl_platooning_para;
55 volatile struct ctrl_platooning_state
56 ctrl_platooning_state;
57
58 // LEE 0726 new global variable
59 int delta_x_neg_error_flag = 0;
60 int delta_x_neg_error_flag_p = 0;
61
62 int started_flag = 0;
63 int should_stop_flag = 0;
64 // controller bypass flags
65 // be cautious when use it
66 int not_start_bypass = 1;
67 int stop_bypass = 0;
68 int too_close_bypass = 0;
69 int brake_bypass = 0;
70 // general system flags
71 int vehicle_status = 0;
72 int separation_too_close_flag = 0;
73 int follower_catched_up_flag = 0;
74 int leader_stopped_flag = 0;
75
76 long running_time_start_ms = 0;
77 // store start time
78 long run_lapsed_time_ms = 0;
79 int negative_flag_cnt = 0;
80 // flag up when delta_x error is negative
81
82 // ----- vehicle state utility -----
83
84
85
86
87 void kinematics(double wl, double wr,
88 double *v, double *omega) {
89
90     *v = (wl + wr) * mc_init.wheel_radius / 2;
91     *omega = (wr - wl) * mc_init.wheel_radius
92         / mc_init.distance_of_wheels;
93 }
94
95 void inverse_kinematics(double v, double omega,
96 double *wl, double *wr) {
97
98     *wl = (2 * v - mc_init.distance_of_wheels * omega)
99         / (2 * mc_init.wheel_radius);
100    *wr = (2 * v + mc_init.distance_of_wheels * omega)
101        / (2 * mc_init.wheel_radius);
102 }

```

```

103
104 void update_vehicle_state( struct h2l_vehicle_status *state) {
105
106     vstate.timestamp = state->timestamp;
107     vstate.imu_theta = state->imu_theta;
108     vstate.imu_accx = state->imu_accx;
109     vstate.imu_omega = state->imu_omega;
110     vstate.encoder_wl = state->encoder_wl;
111     vstate.encoder_wr = state->encoder_wr;
112     vstate.ultrasonic_delta_x = state->ultrasonic_delta_x;
113
114     // calculate / fuse states
115
116     vstate.accx = vstate.imu_accx;
117
118     kinematics(vstate.encoder_wl, vstate.encoder_wr,
119     (double *) &vstate.v,
120     (double *) &vstate.omega);
121
122     // derive x and y through dead reckoning
123     // FIXIT: imu_theta is absolute theta derived
124     // from IMU in XI-O-YI coordinates
125     static int integration_error_print_flag = 0;
126     vstate.theta = vstate.imu_theta;
127     // sometimes abnormal initial error happens
128     // deal with them carefully
129     if (fabs(vstate.x) > 100) vstate.x = 0;
130     vstate.x += vstate.v * cos(vstate.theta)
131     * mc_init.ctrl_loop_period
132         / 1000.0;
133     if(vstate.x < 0 ){
134         integration_error_print_flag++;
135         if (1 == integration_error_print_flag)
136             printf("Integration of X Error!\n x:"
137             "%6.2f, theta: %6.2f",
138             vstate.x, vstate.theta);
139     }
140     vstate.y += vstate.v * sin(vstate.theta)
141     * mc_init.ctrl_loop_period
142         / 1000.0;
143
144 }
145
146 // ----- outer loop utility -----
147
148 // PID controller absolute style
149 double ctrl_pid_with_d_rolloff(
150     double err, double err_sum, double err_p,
151     double kp, double ki, double kd,
152     double ud_lpf_n, double *ud_save,
153     double ud_p, double ts) {
154
155     double up = kp * err;
156     double ui = ki * ts * err_sum;
157
158     // Apply low pass filter for ud, so that it //
159     // is less sensitive to noise.
160     double alpha = 1.0 / (1.0 + ud_lpf_n * ts);
161     double ud = alpha * ud_p + (1 - alpha) *
162         kd * (err - err_p) / ts;
163
164     *ud_save = ud;
165
166     double u = up + ui + ud;
167
168     return u;
169 }
170 // PID controller position style

```

```

171 //using back Euler discretization
172 double ctrl_pid(double err, double err_sum, double err_p,
173                 double kp, double ki, double kd,
174                 double *up_out, double *ui_out, double *ud_out,
175                 double ts) {
176
177     double up = kp * err;
178     double ui = ki * ts * err_sum;
179     double ud = kd * (err - err_p) / ts;
180     double u = up + ui + ud;
181
182     if (up_out != NULL)
183         *up_out = up;
184     if (ui_out != NULL)
185         *ui_out = ui;
186     if (ud_out != NULL)
187         *ud_out = ud;
188
189     return u;
190 }
191 // PID controller incremental style using
192 //back Euler discretization
193 double ctrl_threshold_and_saturation(double in, double th,
194                                       double min, double max);
195
196 double ctrl_pid_inc(double u_p, double err,
197                      double err_p, double err_pp,
198                      double kp, double ki, double kd,
199                      double *up_out, double *ui_out,
200                      double *ud_out, double ts) {
201
202     double up = kp * (err - err_p);
203     double ui = ki * ts * err;
204     double ud = kd * ((err - err_p) - (err_p - err_pp)) / ts;
205     double ud_temp = ud;
206
207     double deltau = up + ui + ud;
208     double u = u_p + deltau;
209
210     if (up_out != NULL)
211         *up_out = up;
212     if (ui_out != NULL)
213         *ui_out = ui;
214     if (ud_out != NULL)
215         *ud_out = ud;
216
217     return u;
218 }
219 const double rf_coeff = 0.8; // backward Euler 40/(s+40)
220
221 double ctrl_roll_off_once(double u_rf_p,
222                           double u_in, double rf_coeff) {
223
224     double u_rf = (1 - rf_coeff) * u_rf_p + rf_coeff * u_in;
225
226     return u_rf;
227 }
228
229 //wr_wl saturate utility function
230 const double wl_wr_hw_sat_lower_bound = 0;
231 const double wl_wr_hw_sat_upper_bound = 20;
232
233
234 // CAUTION: this is just a temporary solution
235 void ctrl_saturate_wl_wr(double *wl, double *wr) {
236     if (*wl < wl_wr_hw_sat_lower_bound) {
237         *wl = wl_wr_hw_sat_lower_bound;
238     } else if (*wl > wl_wr_hw_sat_upper_bound) {

```

```

239             *wl = wl_wr_hw_sat_upper_bound;
240         }
241
242         if (*wr < wl_wr_hw_sat_lower_bound) {
243             *wr = wl_wr_hw_sat_lower_bound;
244         } else if (*wr > wl_wr_hw_sat_upper_bound) {
245             *wr = wl_wr_hw_sat_upper_bound;
246         }
247     }
248
249 // control effort restriction
250 double ctrl_threshold_and_saturation(double in, double th,
251 double min,
252         double max) {
253
254     double out = in;
255
256     // threshold means no response region is [-th, th]
257     if (fabs(in) < th) {
258         out = 0;
259     } else if (in > max) {
260         out = max;
261     } else if (in < min) {
262         out = min;
263     }
264     return out;
265 }
266
267 // ----- outer loop controller -----
268
269 // outer loop framework
270 //1) information gathering from sensor or communication
271 //2) calculate error
272 //3) pid control + roll off
273 //4) control effort restriction (saturation, deadzone)
274 //5) obtain wl_dsr, wr_dsr through inverse kinematics
275 //6) states update
276 // when bypass mechanism involved things becomes
277 // more complicated final cmd is bypass controller
278 // and controlled by different bypass flags triggered
279 // by different events
280
281 void ctrl_open_loop() {
282
283     h2l_send_open_loop_msg(ctrl_open_loop_state.pwm_left_dsr,
284                           ctrl_open_loop_state.pwm_right_dsr);
285
286 }
287
288 void ctrl_wl_wr_loop() {
289
290     h2l_send_wl_wr_msg(ctrl_wl_wr_state.wl_dsr,
291                         ctrl_wl_wr_state.wr_dsr);
292
293 }
294
295 // direct v_omega
296 void ctrl_v_omega_loop() {
297
298     double wl_dsr;
299     double wr_dsr;
300
301     inverse_kinematics( ctrl_v_omega_state.v_dsr,
302                           ctrl_v_omega_state.omega_dsr,
303                           &wl_dsr, &wr_dsr);
304
305     // send to inner loop;

```

```

306     h2l_send_wl_wr_msg((float) wl_dsr, (float) wr_dsr);
307 }
308
309 // ----- (v, theta) control -----
310
311 void ctrl_v_theta_init() {
312
313     char fn_buff[64];
314     char suffix[128];
315
316     time_t t = time(NULL);
317     struct tm tm = *localtime(&t);
318
319     snprintf(suffix, sizeof(suffix),
320             "%04d.%02d.%02d.%02d.%02d",
321             tm.tm_year + 1900,
322             tm.tm_mon + 1,
323             tm.tm_mday,
324             tm.tm_hour,
325             tm.tm_min,
326             tm.tm_sec);
327
328     snprintf(fn_buff, sizeof(fn_buff),
329             "v_theta.%s.txt", suffix);
330
331     ctrl_v_theta_state.log_file = fopen(fn_buff, "w+");
332     ctrl_v_theta_state.recent_log_file
333     = fopen("v_theta.txt", "w+");
334 }
335
336 void ctrl_v_theta_exit() {
337
338     fclose(ctrl_v_theta_state.log_file);
339     fclose(ctrl_v_theta_state.recent_log_file);
340
341 }
342
343 void ctrl_v_theta_print(FILE *fd) {
344
345     struct timespec now;
346     clock_gettime(CLOCK_MONOTONIC, &now);
347
348     uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
349
350     // print vehicle state first
351     fprintf(fd, "%08u, %08u, "
352             "% 6.2f, % 6.2f, % 6.2f, "
353             "% 6.2f, % 6.2f, % 6.2f, % 6.2f, % 6.2f, \t\t",
354             ts, vstate.timestamp,
355
356             vstate.ultrasonic_delta_x,
357             vstate.encoder_wl, vstate.encoder_wr,
358             vstate.v, vstate.omega,
359             vstate.x, vstate.y, vstate.theta);
360
361     // controller states
362     fprintf(fd, "%6.2f, %6.2f, % 6.2f, % 6.2f,"
363             "\t\t% 6.2f, % 6.2f, \t\t",
364             ctrl_v_theta_state.wl_dsr,
365             ctrl_v_theta_state.wr_dsr,
366             ctrl_v_theta_state.u_v,
367             ctrl_v_theta_state.u_omega,
368
369             ctrl_v_theta_state.v_dsr,
370             ctrl_v_theta_state.theta_dsr);
371
372     // error value
373     fprintf(fd, "% 6.2f, % 6.2f, % 6.2f,\t\t",

```

```

374         ctrl_v_theta_state.error_theta ,
375         ctrl_v_theta_state.error_theta_p ,
376         ctrl_v_theta_state.error_theta_sum_p );
377
378     // debug value
379     fprintf(fd, "% .2f, % .2f, % .2f, % .2f\n",
380             ctrl_v_theta_state.up_omega_out ,
381             ctrl_v_theta_state.ui_omega_out ,
382             ctrl_v_theta_state.ud_omega_out ,
383             ctrl_v_theta_state.u_omega_wf);
384
385     fflush( fd );
386 }
387
388 void ctrl_v_theta_loop() {
389
390     // calculate error
391     // sum update before PID because
392     // I-term(k) = sum(i=1^k) e(i)
393
394     double error_theta =
395         ctrl_v_theta_state.theta_dsr - vstate.theta;
396     double error_theta_sum =
397         ctrl_v_theta_state.error_theta_sum_p+ error_theta;
398     double error_theta_p = ctrl_v_theta_state.error_theta_p;
399
400     double up_v_out;
401     double ui_v_out;
402     double ud_v_out;
403
404     //PID controller main body
405
406     // NOTE: v is not controlled here.
407     double u_v = ctrl_v_theta_state.v_dsr;
408
409     double up_omega_out;
410     double ui_omega_out;
411     double ud_omega_out;
412
413     double u_omega_out = ctrl_pid(
414         error_theta , error_theta_sum , error_theta_p ,
415         ctrl_v_theta_para.kp_theta ,
416         ctrl_v_theta_para.ki_theta ,
417         ctrl_v_theta_para.kd_theta ,
418         &up_omega_out , &ui_omega_out ,
419         &ud_omega_out , mc_init.ctrl_loop_period / 1000.0);
420
421     // NOTE: If you has d-term then roll off twice,
422     //(low pass filter)
423
424     double u_omega_rf1 = ctrl_roll_off_once
425     (ctrl_v_theta_state.u_omega_rf1_p ,
426      u_omega_out , rf_coeff);
427     double u_omega_rf2 = ctrl_roll_off_once
428     (ctrl_v_theta_state.u_omega_rf2_p ,
429      u_omega_rf1 , rf_coeff);
430
431     // outer loop control effort restriction
432     // not restriction for u_v
433     double u_omega =
434         ctrl_threshold_and_saturation(u_omega_rf2 , 0 , -2, 2);
435
436     double wl_dsr;
437     double wr_dsr;
438     inverse_kinematics(u_v , u_omega , &wl_dsr , &wr_dsr);
439
440     // send to inner loop;
441     h2l_send_wl_wr_msg(( float ) wl_dsr , ( float ) wr_dsr);
442

```

```

443     // update current states
444     ctrl_v_theta_state.error_theta = error_theta;
445     ctrl_v_theta_state.wl_dsr = wl_dsr;
446     ctrl_v_theta_state.wr_dsr = wr_dsr;
447     ctrl_v_theta_state.u_v = u_v;
448     ctrl_v_theta_state.u_omega = u_omega;
449
450     // update debug states
451     ctrl_v_theta_state.up_omega_out = up_omega_out;
452     ctrl_v_theta_state.ui_omega_out = ui_omega_out;
453     ctrl_v_theta_state.ud_omega_out = ud_omega_out;
454     ctrl_v_theta_state.u_omega_wo_rf = u_omega_out;
455
456     // print control state to file
457     ctrl_v_theta_print(ctrl_v_theta_state.log_file);
458     ctrl_v_theta_print(ctrl_v_theta_state.recent_log_file);
459
460     // update previous states
461
462     ctrl_v_theta_state.error_theta_p = error_theta;
463     ctrl_v_theta_state.error_theta_sum_p = error_theta_sum;
464
465     ctrl_v_theta_state.u_omega_rf1_p = u_omega_rf1;
466     ctrl_v_theta_state.u_omega_rf2_p = u_omega_rf2;
467
468 }
469
470 // ----- (x, y) control -----
471
472 void ctrl_x_y_init() {
473
474     char fn_buff[64];
475     char suffix[128];
476
477     time_t t = time(NULL);
478     struct tm tm = *localtime(&t);
479
480     snprintf(suffix, sizeof(suffix),
481             "%04d.%02d.%02d-%02d.%02d.%02d",
482             tm.tm_year + 1900,
483             tm.tm_mon + 1,
484             tm.tm_mday,
485             tm.tm_hour,
486             tm.tm_min,
487             tm.tm_sec);
488
489     snprintf(fn_buff, sizeof(fn_buff), "x_y_%s.txt", suffix);
490
491     ctrl_x_y_state.log_file = fopen(fn_buff, "w+");
492     ctrl_x_y_state.recent_log_file = fopen("x_y.txt", "w+");
493 }
494
495 void ctrl_x_y_exit() {
496
497     fclose(ctrl_x_y_state.log_file);
498     fclose(ctrl_x_y_state.recent_log_file);
499
500 }
501
502 void ctrl_x_y_print(FILE *fd) {
503
504     struct timespec now;
505     clock_gettime(CLOCK_MONOTONIC, &now);
506
507     uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
508
509     // print vehicle state first
510     fprintf(fd, "%08u, %08u, "

```

```

511         "% 6.2f, %6.2f, %6.2f, "
512         "% 6.2f, % 6.2f, % 6.2f, % 6.2f, % 6.2f, \t\t",
513         ts, vstate.timestamp,
514
515         vstate.ultrasonic_delta_x,
516         vstate.encoder_wl, vstate.encoder_wr,
517         vstate.v, vstate.omega,
518         vstate.x, vstate.y, vstate.theta);
519
520         fprintf(fd, "%6.2f, %6.2f, % 6.2f, % 6.2f,"
521         "\t% 6.2f, % 6.2f, \t\t",
522             ctrl_x_y_state.wl_dsr,
523             ctrl_x_y_state.wr_dsr, ctrl_x_y_state.u_v,
524             ctrl_x_y_state.u_omega,
525
526             ctrl_x_y_state.x_dsr, ctrl_x_y_state.y_dsr);
527
528         fprintf(fd, "% 6.2f, % 6.2f, % 6.2f, % 6.2f,"
529         "% 6.2f, % 6.2f, \t\t",
530             ctrl_x_y_state.error_dist,
531             ctrl_x_y_state.error_dist_p,
532             ctrl_x_y_state.error_dist_sum_p,
533             ctrl_x_y_state.error_angle,
534             ctrl_x_y_state.error_angle_p,
535             ctrl_x_y_state.error_angle_sum_p);
536
537         fprintf(fd, "% 6.2f, % 6.2f, % 6.2f, % 6.2f,"
538         "% 6.2f, % 6.2f\n",
539             ctrl_x_y_state.up_dist_out,
540             ctrl_x_y_state.ui_dist_out,
541             ctrl_x_y_state.ud_dist_out,
542             ctrl_x_y_state.up_angle_out,
543             ctrl_x_y_state.ui_angle_out,
544             ctrl_x_y_state.ud_angle_out);
545
546         fflush(fd);
547     }
548
549     void ctrl_x_y_loop() {
550
551         int ret;
552
553         double phi = atan(
554             (ctrl_x_y_state.y_dsr - vstate.y)
555             / (ctrl_x_y_state.x_dsr - vstate.x));
556
557         double error_x = ctrl_x_y_state.x_dsr - vstate.x;
558         double error_y = ctrl_x_y_state.y_dsr - vstate.y;
559
560         double error_dist = error_x + fabs(error_y);
561         double error_dist_sqrt = sqrt(error_x * error_x +
562             error_y * error_y);
563
564         // deadzone
565         if (error_dist_sqrt < 0.04 && error_dist_sqrt > -0.04) {
566             error_dist = 0;
567         }
568
569         double error_dist_sum = ctrl_x_y_state.error_dist_sum_p
570         + error_dist;
571         double error_dist_p = ctrl_x_y_state.error_dist_p;
572
573         double error_angle = phi - vstate.theta;
574
575         // deadzone
576         if ((error_x < 0.02 && error_x > -0.02)
577             && (error_y < 0.02 && error_y > -0.02)) {
578             error_angle = 0;

```

```

579     }
580
581
582     double error_angle_sum = ctrl_x_y_state.error_angle_sum_p
583     + error_angle;
584     double error_angle_p = ctrl_x_y_state.error_angle_p;
585
586     double up_dist_out;
587     double ui_dist_out;
588     double ud_dist_out;
589     double up_angle_out;
590     double ui_angle_out;
591     double ud_angle_out;
592
593     double u_v = ctrl_pid(error_dist, error_dist_sum,
594                           error_dist_p,
595                           ctrl_x_y_para.kp_dist,
596                           ctrl_x_y_para.ki_dist,
597                           ctrl_x_y_para.kd_dist,
598                           &up_dist_out, &ui_dist_out, &ud_dist_out,
599                           mc_init.ctrl_loop_period / 1000.0);
600
601     double u_omega = ctrl_pid(error_angle, error_angle_sum,
602                               error_angle_p,
603                               ctrl_x_y_para.kp_angle, ctrl_x_y_para.ki_angle,
604                               ctrl_x_y_para.kd_angle,
605                               &up_angle_out, &ui_angle_out, &ud_angle_out,
606                               mc_init.ctrl_loop_period / 1000.0);
607
608
609 // saturation
610 u_v = ctrl_threshold_and_saturation(u_v, 0, -0.5, 0.5);
611 u_omega = ctrl_threshold_and_saturation(u_omega, 0, -3, 3);
612
613 double wl_dsr;
614 double wr_dsr;
615
616 inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);
617
618 // send to inner loop;
619 ret = h2l_send_wl_wr_msg((float) wl_dsr, (float) wr_dsr);
620 if (ret < 0) {
621     printf("<ctrl_loop> Out of control!!!\n");
622 }
623
624 // update current state
625 ctrl_x_y_state.error_dist = error_dist;
626 ctrl_x_y_state.error_angle = error_angle;
627
628 ctrl_x_y_state.wl_dsr = wl_dsr;
629 ctrl_x_y_state.wr_dsr = wr_dsr;
630 ctrl_x_y_state.u_v = u_v;
631 ctrl_x_y_state.u_omega = u_omega;
632
633 // update debug states
634 ctrl_x_y_state.up_dist_out = up_dist_out;
635 ctrl_x_y_state.ui_dist_out = ui_dist_out;
636 ctrl_x_y_state.ud_dist_out = ud_dist_out;
637
638 ctrl_x_y_state.up_angle_out = up_angle_out;
639 ctrl_x_y_state.ui_angle_out = ui_angle_out;
640 ctrl_x_y_state.ud_angle_out = ud_angle_out;
641
642 // print control state to file
643 ctrl_x_y_print(ctrl_x_y_state.log_file);
644 ctrl_x_y_print(ctrl_x_y_state.recent_log_file);
645
646 // update previous state
647

```

```

648     ctrl_x_y_state.error_dist_p = error_dist;
649     ctrl_x_y_state.error_dist_sum_p = error_dist_sum;
650
651     ctrl_x_y_state.error_angle_p = error_angle;
652     ctrl_x_y_state.error_angle_sum_p = error_angle_sum;
653
654 }
655
656 // ----- (delta_x, theta) control -----
657
658 void ctrl_delta_x_theta_init() {
659
660     char fn_buff[64];
661     char suffix[128];
662
663     time_t t = time(NULL);
664     struct tm tm = *localtime(&t);
665
666     snprintf(suffix, sizeof(suffix),
667             "%04d.%02d.%02d.%02d.%02d",
668             tm.tm_year + 1900, tm.tm_mon + 1,
669             tm.tm_mday, tm.tm_hour, tm.tm_min,
670             tm.tm_sec);
671
672     snprintf(fn_buff, sizeof(fn_buff),
673             "delta_x_theta_%s.txt", suffix);
674     ctrl_delta_x_theta_state.log_file = fopen(fn_buff, "w+");
675     ctrl_delta_x_theta_state.recent_log_file
676     = fopen("delta_x_theta.txt", "w+");
677 }
678
679
680 void ctrl_delta_x_theta_exit() {
681
682     fclose(ctrl_delta_x_theta_state.log_file);
683     fclose(ctrl_delta_x_theta_state.recent_log_file);
684 }
685
686
687 void ctrl_delta_x_theta_print(FILE *fd) {
688
689     struct timespec now;
690     clock_gettime(CLOCK_MONOTONIC, &now);
691
692     uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
693
694
695     // print vehicle state first
696     fprintf(fd, "%08u %08u, "
697             "% 6.2f, % 6.2f, % 6.2f, "
698             "% 6.2f, % 6.2f, % 6.2f, % 6.2f, % 6.2f, \t\t",
699             ts, vstate.timestamp,
700
701             vstate.ultrasonic_delta_x,
702             vstate.encoder_wl, vstate.encoder_wr,
703             vstate.v, vstate.omega,
704             vstate.x, vstate.y, vstate.theta);
705
706     // print control variable
707     fprintf(fd, "%6.2f, %6.2f, %6.4f, %6.4f, "
708             "%6.4f, %6.4f, \t\t",
709             ctrl_delta_x_theta_state.wl_dsr,
710             ctrl_delta_x_theta_state.wr_dsr,
711             ctrl_delta_x_theta_state.u_v,
712             ctrl_delta_x_theta_state.u_omega,
713             ctrl_delta_x_theta_state.delta_x_dsr,
714             ctrl_delta_x_theta_state.theta_dsr
715         );

```

```

716
717     fprintf(fd , "%6.4f, %6.4f, %6.4f, %6.4f,"
718             "%6.4f, %6.4f, \t\t",
719             ctrl_delta_x_theta_state.error_delta_x ,
720             ctrl_delta_x_theta_state.error_delta_x_p ,
721             ctrl_delta_x_theta_state.error_delta_x_pp ,
722             ctrl_delta_x_theta_state.error_theta ,
723             ctrl_delta_x_theta_state.error_theta_p ,
724             ctrl_delta_x_theta_state.error_theta_sum_p
725         );
726
727     fprintf(fd , "%6.4f, %6.4f, %6.4f, %6.4f,"
728             "%6.4f, %6.4f, %6.4f,\t\t",
729             ctrl_delta_x_theta_state.up_v_out ,
730             ctrl_delta_x_theta_state.ui_v_out ,
731             ctrl_delta_x_theta_state.ud_v_out ,
732             ctrl_delta_x_theta_state.u_v_out_p ,
733             ctrl_delta_x_theta_state.u_v_out ,
734             ctrl_delta_x_theta_state.up_omega_out ,
735             ctrl_delta_x_theta_state.ui_omega_out ,
736             ctrl_delta_x_theta_state.ud_omega_out
737         );
738     fprintf(fd , "%d, %d, %d, %d\n",
739             started_flag ,
740             should_stop_flag ,
741             not_start_bypass ,
742             stop_bypass
743         );
744     fflush(fd );
745 }
746
747 // NOTE: This outer loop only control delta_x
748 // it can be used as platoon separation control also
749 // when communication is not used
750 void ctrl_delta_x_theta_loop() {
751     //get sensor reading
752
753     double delta_x = vstate.ultrasonic_delta_x ;
754
755     if(fabs(delta_x - vstate.ultrasonic_delta_x_bk)>= 0.1
756         && started_flag == 1){
757         printf("ULTRASONIC ABNORMAL!\n");
758         delta_x= vstate.ultrasonic_delta_x_bk ;
759     }
760
761     if (0 == delta_x || delta_x >= 3 ){
762         delta_x = vstate.ultrasonic_delta_x_bk ;
763         // set limit on sensor reading
764         // to prevent unexpected error
765         if (delta_x >= 2) delta_x = 1;
766         printf("ULTRASONIC ERROR!\n");
767     }
768
769
770     // getting delta_theta by camera
771     double delta_theta = vstate.camera_delta_theta ;
772
773     // LEE 0623 define variable and initialize them to be 0
774     // if not in the normal operation status ,
775     //they will be left as 0
776
777     double error_delta_x      = 0;
778     double error_delta_x_p    = 0;
779     double error_delta_x_pp   = 0;
780
781     double error_theta        = 0;
782     double error_theta_sum    = 0;
783     double error_theta_p      = 0;

```

```

784
785
786
787     // CAUTION: Here we use (delta_x - delta_x_dsr).
788     // This is intentional.
789     // i.e. if delta_x is bigger than desired value, speed up.
790
791     // LEE 0623 adding new start status checking
792     // by detecting rising edge of
793     // delta_x error
794     error_delta_x      =
795     delta_x - ctrl_delta_x_theta_state.delta_x_dsr;
796     error_theta        = delta_theta;
797
798     if (error_delta_x <= 0){
799         delta_x_neg_error_flag=1;
800     } else{
801         delta_x_neg_error_flag=0;
802     }
803
804     // vehicle status checking
805     // stop status checking
806     // vehicle will stop if too close //
807     // to predecessor or off-track
808     // or reach pre-defined maximum run_lapse time
809     // only works if
810     // running_time_MAX_ms < (running_time=20000 in init.conf)
811     long run_lapsed_time_ms = 0;
812     if (running_time_start_ms >0 ){
813         run_lapsed_time_ms =
814         vstate.timestamp - running_time_start_ms;
815     }
816
817     // after start and getting too close
818     static int stop_print_flag = 0;
819     if(delta_x <= 0.15 && started_flag == 1){
820         //too close set stop flag
821         // and reset flags and start time
822         should_stop_flag           = 1;
823         stop_bypass++;
824         delta_x_neg_error_flag_p   = 1;
825         delta_x_neg_error_flag     = 1;
826         started_flag               = 0;
827         running_time_start_ms     = 0;
828         stop_print_flag++;
829         if(1 == stop_print_flag){
830             printf("DELTA X at %5.2f STOP!\n",delta_x);
831         }
832     }
833     static int off_track_flag = 0;
834     if(fabs(error_theta) >= 0.30){
835         // off-track
836         should_stop_flag           = 1;
837         stop_bypass++;
838         off_track_flag++;
839         if(1 == off_track_flag )
840             printf("OFF TRACK AND STOP\n");
841     }
842
843     // start status checking
844     if (0 == started_flag){
845         // if not start yet
846         ctrl_delta_x_theta_state.error_delta_x_p      = 0;
847         ctrl_delta_x_theta_state.error_delta_x_pp     = 0;
848         ctrl_delta_x_theta_state.u_v_rf1_p            = 0;
849         ctrl_delta_x_theta_state.u_v_rf2_p            = 0;
850         ctrl_delta_x_theta_state.u_v_out_p           = 0;
851

```

```

852     ctrl_delta_x_theta_state.error_theta_p      = 0;
853     ctrl_delta_x_theta_state.error_theta_sum_p = 0;
854     ctrl_delta_x_theta_state.u_omega_rf1_p    = 0;
855     ctrl_delta_x_theta_state.u_omega_rf2_p    = 0;
856
857     // error at beginning cross 0 from negative to positive
858     // i.e. rising edge set started_flag on
859     if(1 == delta_x_neg_error_flag_p
860     && 0 == delta_x_neg_error_flag){
861         started_flag      = 1;
862         not_start_bypass = 0;
863         stop_bypass      = 0;
864         should_stop_flag = 0;
865         separation_too_close_flag = 0;
866         stop_print_flag  = 0;
867
868         // store start time
869         running_time_start_ms = vstate.timestamp;
870         printf("Vehicle Start at %ld !\n",
871                running_time_start_ms);
872
873     } else{
874         // if not start yet, set up not start bypass
875         // flag and update delta_x_neg_error_flag_p
876         not_start_bypass = 1;
877     }
878 } else{ // to distinguish start before and after
879     delta_x_neg_error_flag_p = 9;
880     delta_x_neg_error_flag = 9;
881 }
882
883 // PID controller on delta_x, which derive u_v
884 double u_v_out=0;
885 double u_omega_out=0;
886
887 double up_v_out=0;
888 double ui_v_out=0;
889 double ud_v_out=0;
890
891 double up_omega_out=0;
892 double ui_omega_out=0;
893 double ud_omega_out=0;
894
895
896 // control effort restriction
897 double lowest_cruise_speed = 0.1;
898
899 double u_v;
900 double u_omega = 0;
901
902 double u_v_rf1 = 0;
903 double u_v_rf2 = 0;
904
905 double u_omega_rf1 = 0;
906 double u_omega_rf2 = 0;
907
908 double kd_local = 0;
909
910
911 static int cancel_kd_flag = 0;
912 if (run_lapsed_time_ms > 3000){
913     kd_local = 0;
914     cancel_kd_flag++;
915     if(1 == cancel_kd_flag){
916         printf("cancel kd term \r\n");
917     }
918 } else if (1 == delta_x_neg_error_flag ){
919     kd_local = 0;
920     if(1 == started_flag){

```

```

921         cancel_kd_flag++;
922         if(1 == cancel_kd_flag){
923             printf("cancel kd term \r\n");
924         }
925     } else{
926         kd_local = ctrl_delta_x_theta_para.kd_delta_x;
927     }
928
929
930
931 // normal operation case checking if pass then get
932 //relative errors
933 if(0 == should_stop_flag && 1 == started_flag){
934     // error_delta_x and error_delta_theta
935     //has been computed ahead
936     error_delta_x_p =
937         ctrl_delta_x_theta_state.error_delta_x_p;
938     error_delta_x_pp =
939         ctrl_delta_x_theta_state.error_delta_x_pp;
940
941     error_theta_sum =
942         ctrl_delta_x_theta_state.error_theta_sum_p
943             + error_theta;
944     error_theta_p =
945         ctrl_delta_x_theta_state.error_theta_p;
946
947     // normal operation pid incremental
948     // be careful about memory ctrl states u_v_out_p
949     u_v_out = ctrl_pid_inc(
950         ctrl_delta_x_theta_state.u_v_out_p,
951         error_delta_x ,
952         error_delta_x_p ,
953         error_delta_x_pp ,
954         ctrl_delta_x_theta_para.kp_delta_x ,
955         ctrl_delta_x_theta_para.ki_delta_x ,
956         kd_local ,
957         &up_v_out ,
958         &ui_v_out ,
959         &ud_v_out ,
960         mc_init.ctrl_loop_period / 1000.0
961     );
962     if(fabs(u_v_out)>3){
963         printf("u_v_out error at %6.2f ,"
964             "P:%6.2f , I:%6.2f , D:%6.2f\r\n",
965             u_v_out , up_v_out , ui_v_out , ud_v_out);
966         //u_v_out = lowest_cruise_speed;
967         u_v_out = ctrl_delta_x_theta_state.u_v_out_p;
968         u_omega_out = 0;
969     }
970     u_omega_out = ctrl_pid(
971         error_theta ,
972         error_theta_sum ,
973         error_theta_p ,
974         ctrl_delta_x_theta_para.kp_theta ,
975         ctrl_delta_x_theta_para.ki_theta ,
976         ctrl_delta_x_theta_para.kd_theta ,
977         &up_omega_out ,
978         &ui_omega_out ,
979         &ud_omega_out ,
980         mc_init.ctrl_loop_period / 1000.0
981     );
982
983     // controller roll off on the output v
984     //(low pass filter)
985     u_v_rf1 = ctrl_roll_off_once
986     (ctrl_delta_x_theta_state.u_v_rf1_p , u_v_out , rf_coeff );
987     u_v_rf2 = ctrl_roll_off_once
988     (ctrl_delta_x_theta_state.u_v_rf2_p , u_v_rf1 , rf_coeff );

```

```

989
990
991 // controller roll off on the output omega
992 // (low pass filter)
993 u_omega_rf1 = ctrl_roll_off_once
994 (ctrl_delta_x_theta_state.u_omega_rf1_p ,
995 u_omega_out , rf_coeff );
996 u_omega_rf2 = ctrl_roll_off_once
997 (ctrl_delta_x_theta_state.u_omega_rf2_p ,
998 u_omega_rf1 , rf_coeff );
999
1000
1001 if( fabs(u_omega_out)>100){
1002     printf("u_omega_out error\n");
1003     u_omega_out = 0;
1004 }
1005
1006
1007 if(error_delta_x <= -0.03 && 1== started_flag){
1008     negative_flag_cnt++;
1009     u_v = 0;
1010     ctrl_delta_x_theta_state.u_v_out_p = 0;
1011     ctrl_delta_x_theta_state.u_v_rf1_p = 0;
1012     ctrl_delta_x_theta_state.u_v_rf1_p = 0;
1013     if(brake_bypass < 2)
1014         brake_bypass = (int)(fabs(error_delta_x)/0.01);
1015     printf("error delta_x %"
1016 "6.2f, brake time %d: \n",
1017             error_delta_x , brake_bypass);
1018 } else{
1019     u_v      = ctrl_threshold_and_saturation
1020     (u_v_rf2 , 0 , 0 , 0.8);
1021 }
1022
1023 if(error_delta_x >= 0.03){
1024     negative_flag_cnt = 0;
1025     brake_bypass = 0;
1026 }
1027
1028
1029 u_omega = ctrl_threshold_and_saturation
1030 (u_omega_rf2 , 0 , -2,2);
1031 if (u_v <= lowest_cruise_speed){
1032     u_omega=0;
1033 }
1034
1035
1036 // normal operation but vehicle is
1037 // too close to predecessor
1038 // look ahead distance has been obstructed
1039 // by predecessor
1040 // handle this unexpected sensor failure
1041 // delta_x less than 30cm may cause
1042 // unexpected camera reading error
1043 // camera look ahead distance is set to 20cm
1044 // too close look ahead distance vehicle
1045 // will easily lose the track
1046 if(delta_x<= 0.30){
1047     separation_too_close_flag++;
1048     if(separation_too_close_flag == 1){
1049         printf(" ! SEPARATION TOO CLOSE SENSOR "
1050 "CHANGE TO IMU TEMPORARILY !\r\n");
1051         error_theta = 0 - vstate.imu_theta;
1052     }
1053     u_v = 0;
1054     u_omega = 0;
1055 } else{

```

```

1056             separation_too_close_flag = 0;
1057         }
1058     } else{
1059         // abnormal case dealing
1060         // should reset ctrl memory states to zero !!!
1061         u_v_out      = 0;
1062         u_omega_out  = 0;
1063         u_v          = 0;
1064         u_omega       = 0;
1065
1066         ctrl_delta_x_theta_state.error_delta_x_p      = 0;
1067         ctrl_delta_x_theta_state.error_delta_x_pp     = 0;
1068         ctrl_delta_x_theta_state.u_v_rf1_p           = 0;
1069         ctrl_delta_x_theta_state.u_v_rf2_p           = 0;
1070         ctrl_delta_x_theta_state.u_v_out_p           = 0;
1071
1072         ctrl_delta_x_theta_state.error_theta_p        = 0;
1073         ctrl_delta_x_theta_state.error_theta_sum_p   = 0;
1074         ctrl_delta_x_theta_state.u_omega_rf1_p       = 0;
1075         ctrl_delta_x_theta_state.u_omega_rf2_p       = 0;
1076
1077     }
1078
1079
1080
1081     double wl_dsr;
1082     double wr_dsr;
1083
1084     inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);
1085
1086 // ctrl_saturate_wl_wr(&wl_dsr, &wr_dsr);
1087
1088 // send to inner loop
1089 if(wl_dsr < 0){ wl_dsr = 0;}
1090 if(wr_dsr < 0){ wr_dsr = 0;}
1091 // final safety to prevent backward rotation in ETT
1092 // LEE abnormal case bypass
1093
1094 if(1 == not_start_bypass || stop_bypass >0
1095 || brake_bypass > 0){
1096     // send 0 reference command
1097     if(1 == not_start_bypass){
1098         h2l_send_wl_wr_msg(0, 0);
1099     }
1100     if(1 == stop_bypass){
1101         printf("STOP! send wl_dsr -1 wr_dsr -1\n");
1102         h2l_send_wl_wr_msg(-1, -1);
1103         //printf("STOP! send wl_dsr -1 wr_dsr -1\n");
1104     }
1105     if(brake_bypass > 0){
1106         printf("BRAKE! send wl_dsr -1 wr_dsr -1\n");
1107         h2l_send_wl_wr_msg(-1, -1);
1108         brake_bypass--;
1109     }
1110 }
1111 else{// normal operation
1112     h2l_send_wl_wr_msg((float)wl_dsr, (float)wr_dsr);
1113 }
1114
1115
1116 // save current intermediate state , for debug
1117
1118 ctrl_delta_x_theta_state.wl_dsr = wl_dsr;
1119 ctrl_delta_x_theta_state.wr_dsr = wr_dsr;
1120 ctrl_delta_x_theta_state.u_v = u_v;
1121 ctrl_delta_x_theta_state.u_omega = u_omega;
1122
1123 ctrl_delta_x_theta_state.error_delta_x = error_delta_x;

```

```

1124     ctrl_delta_x_theta_state.error_theta = error_theta;
1125
1126     ctrl_delta_x_theta_state.up_v_out = up_v_out;
1127     ctrl_delta_x_theta_state.ui_v_out = ui_v_out;
1128     ctrl_delta_x_theta_state.ud_v_out = ud_v_out;
1129     ctrl_delta_x_theta_state.u_v_out = u_v_out;
1130
1131     ctrl_delta_x_theta_state.up_omega_out = up_omega_out;
1132     ctrl_delta_x_theta_state.ui_omega_out = ui_omega_out;
1133     ctrl_delta_x_theta_state.ud_omega_out = ud_omega_out;
1134     ctrl_delta_x_theta_state.u_omega_out = u_omega_out;
1135
1136     // print control state to file
1137     ctrl_delta_x_theta_print(ctrl_delta_x_theta_state.log_file);
1138     ctrl_delta_x_theta_print
1139     (ctrl_delta_x_theta_state.recent_log_file);
1140
1141     // update previous state
1142     ctrl_delta_x_theta_state.error_delta_x_p
1143     = error_delta_x;
1144     ctrl_delta_x_theta_state.error_delta_x_pp
1145     = error_delta_x_p;
1146     ctrl_delta_x_theta_state.u_v_rf1_p
1147     = u_v_rf1;
1148     ctrl_delta_x_theta_state.u_v_rf2_p
1149     = u_v_rf2;
1150     ctrl_delta_x_theta_state.u_v_out_p
1151     = u_v_out;
1152
1153     ctrl_delta_x_theta_state.error_theta_p
1154     = error_theta;
1155     ctrl_delta_x_theta_state.error_theta_sum_p
1156     = error_theta_sum;
1157     ctrl_delta_x_theta_state.u_omega_rf1_p
1158     = u_omega_rf1;
1159     ctrl_delta_x_theta_state.u_omega_rf2_p
1160     = u_omega_rf2;
1161
1162     // other state
1163     delta_x_neg_error_flag_p
1164     = delta_x_neg_error_flag;
1165     vstate.ultrasonic_delta_x_bk
1166     = vstate.ultrasonic_delta_x;
1167
1168 }
1169
1170 // ----- line track control -----
1171
1172 void ctrl_line_track_init() {
1173
1174     char fn_buff[64];
1175     char suffix[128];
1176
1177     time_t t = time(NULL);
1178     struct tm tm = *localtime(&t);
1179
1180     snprintf(suffix, sizeof(suffix),
1181             "%04d_%02d_%02d__%02d_%02d_%02d",
1182             tm.tm_year + 1900,
1183             tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min,
1184             tm.tm_sec);
1185
1186     snprintf(fn_buff, sizeof(fn_buff),
1187             "line_track_%s.txt", suffix);
1188
1189     ctrl_line_track_state.log_file
1190     = fopen(fn_buff, "w+");
1191     ctrl_line_track_state.recent_log_file
1192     = fopen("line_track.txt", "w+");

```

```

1193  }
1194
1195 void ctrl_line_track_exit() {
1196
1197     fclose(ctrl_line_track_state.log_file);
1198     fclose(ctrl_line_track_state.recent_log_file);
1199
1200 }
1201 void ctrl_line_track_print(FILE *fd) {
1202
1203     struct timespec now;
1204     clock_gettime(CLOCK_MONOTONIC, &now);
1205
1206     uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
1207
1208     // print vehicle state first
1209     fprintf(fd, "%08u, %08u, "
1210             "% 6.2f, % 6.2f, % 6.2f,% 6.2f,% 6.2f,\t\t",
1211             ts, vstate.timestamp,
1212             vstate.encoder_wl, vstate.encoder_wr,
1213             vstate.v, vstate.x, vstate.v_dsr
1214         );
1215
1216     // controller states
1217     fprintf(fd, "%6.2f, %6.2f, % 6.2f, % 6.2f,\t\t",
1218             ctrl_line_track_state.wl_dsr,
1219             ctrl_line_track_state.wr_dsr,
1220             ctrl_line_track_state.u_v,
1221             ctrl_line_track_state.u_omega
1222         );
1223
1224     // debug value
1225     fprintf(fd, "% 6.3f, % 6.3f, % 6.3f, %d \n",
1226             vstate.camera_delta_theta,
1227             ctrl_line_track_state.u_omega_out,
1228             (0-vstate.imu_theta),
1229             vstate.ctrl_flag);
1230
1231     fflush(fd);
1232 }
1233
1234 void ctrl_line_track_loop() {
1235
1236     // calculate error
1237
1238
1239     double delta_theta = vstate.camera_delta_theta;
1240     double delta_theta_sum
1241     = ctrl_line_track_state.delta_theta_sum_p
1242         + delta_theta;
1243     double delta_theta_p
1244     = ctrl_line_track_state.delta_theta_p;
1245
1246     double up_v_out;
1247     double ui_v_out;
1248     double ud_v_out;
1249
1250     //PID controller main body
1251
1252     // NOTE: v is not controlled here.
1253     double u_v = ctrl_line_track_state.v_dsr;
1254
1255     double up_omega_out;
1256     double ui_omega_out;
1257     double ud_omega_out;
1258
1259     double u_omega_out = ctrl_pid(
1260         delta_theta, delta_theta_sum, delta_theta_p,
1261

```

```

1262         ctrl_line_track_para.kp, ctrl_line_track_para.ki,
1263         ctrl_line_track_para.kd,
1264         &up_omega_out, &ui_omega_out,
1265         &ud_omega_out, mc_init.ctrl_loop_period / 1000.0);
1266
1267     // NOTE: If you has d-term then roll off, (low pass filter)
1268
1269     double u_omega_rf1 = ctrl_roll_off_once
1270     (ctrl_line_track_state.u_omega_rf1_p ,
1271      u_omega_out, rf_coeff);
1272     double u_omega_rf2 = ctrl_roll_off_once
1273     (ctrl_line_track_state.u_omega_rf2_p ,
1274      u_omega_rf1, rf_coeff);
1275
1276     // outer loop control effort restriction
1277     // not restriction for u_v
1278     double u_omega = ctrl_threshold_and_saturation
1279     (u_omega_rf2, 0, -2, 2);
1280
1281     double wl_dsr;
1282     double wr_dsr;
1283     inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);
1284
1285     // send to inner loop;
1286     h2l_send_wl_wr_msg((float) wl_dsr, (float) wr_dsr);
1287
1288     // update current states
1289     ctrl_line_track_state.delta_theta = delta_theta;
1290     ctrl_line_track_state.wl_dsr = wl_dsr;
1291     ctrl_line_track_state.wr_dsr = wr_dsr;
1292     ctrl_line_track_state.u_v = u_v;
1293     ctrl_line_track_state.u_omega = u_omega;
1294
1295     // update debug states
1296     ctrl_line_track_state.up_omega_out = up_omega_out;
1297     ctrl_line_track_state.ui_omega_out = ui_omega_out;
1298     ctrl_line_track_state.ud_omega_out = ud_omega_out;
1299     ctrl_line_track_state.u_omega_out = u_omega_out;
1300
1301     // print control state to file
1302     ctrl_line_track_print
1303     (ctrl_line_track_state.log_file);
1304     ctrl_line_track_print
1305     (ctrl_line_track_state.recent_log_file);
1306
1307     // update previous states
1308
1309     ctrl_line_track_state.delta_theta_p = delta_theta;
1310     ctrl_line_track_state.delta_theta_sum_p
1311     = delta_theta_sum;
1312
1313     ctrl_line_track_state.u_omega_rf1_p = u_omega_rf1;
1314     ctrl_line_track_state.u_omega_rf2_p = u_omega_rf2;
1315
1316 }
1317
1318 // ----- platooning control -----
1319
1320 void ctrl_platooning_init() {
1321
1322     char fn_buff[64];
1323     char suffix[128];
1324
1325     time_t t = time(NULL);
1326     struct tm tm = *localtime(&t);
1327
1328     snprintf(suffix, sizeof(suffix),
1329             "%04d_%02d_%02d__%02d_%02d",

```

```

1330         tm.tm_year + 1900,
1331         tm.tm_mon + 1,
1332         tm.tm_mday,
1333         tm.tm_hour,
1334         tm.tm_min,
1335         tm.tm_sec);
1336
1337     sprintf(fn_buff, sizeof(fn_buff),
1338             "platooning_%s.txt", suffix);
1339
1340     ctrl_platooning_state.log_file = fopen(fn_buff, "w+");
1341     ctrl_platooning_state.recent_log_file
1342     = fopen("platooning.txt", "w+");
1343 }
1344
1345 void ctrl_platooning_exit() {
1346
1347     fclose(ctrl_platooning_state.log_file);
1348     fclose(ctrl_platooning_state.recent_log_file);
1349 }
1350
1351
1352 void ctrl_platooning_print(FILE *fd) {
1353
1354     struct timespec now;
1355     clock_gettime(CLOCK_MONOTONIC, &now);
1356
1357     uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
1358
1359     // print vehicle state first col: 1-10
1360     fprintf(fd, "%08u, %08u, "
1361             "%6.4f, %6.2f, %6.2f, %6.4f, %6.4f, "
1362             "%6.4f, %6.4f, %6.4f, \t\t",
1363             ts, vstate.timestamp,
1364             vstate.ultrasonic_delta_x,
1365             vstate.encoder_wl,
1366             vstate.encoder_wr,
1367             vstate.v,
1368             vstate.omega,
1369             vstate.x,
1370             vstate.y,
1371             vstate.theta);
1372
1373     // print control variable col: 11-16
1374     fprintf(fd, "%6.2f, %6.2f, %6.4f, %6.4f, "
1375             "%6.4f, %6.4f, \t\t",
1376             ctrl_platooning_state.wl_dsr,
1377             ctrl_platooning_state.wr_dsr,
1378             ctrl_platooning_state.u_v,
1379             ctrl_platooning_state.u_omega,
1380             ctrl_platooning_state.delta_x_dsr,
1381             ctrl_platooning_state.theta_dsr);
1382
1383     // error col: 17-25
1384     fprintf(fd, "%6.4f, %6.4f, %6.4f, "
1385             "%6.4f, %6.4f, %6.4f, "
1386             "%6.4f, %6.4f, %6.4f, \t\t",
1387             ctrl_platooning_state.error_theta,
1388             ctrl_platooning_state.error_theta_p,
1389             ctrl_platooning_state.error_theta_sum_p,
1390
1391             ctrl_platooning_state.error_delta_x,
1392             ctrl_platooning_state.error_delta_x_p,
1393             ctrl_platooning_state.error_delta_x_pp,
1394
1395             ctrl_platooning_state.error_leader,
1396             ctrl_platooning_state.error_leader_p,
1397             ctrl_platooning_state.error_leader_sum_p);

```

```

1398
1399 // col :26 -40
1400 fprintf(fd , "%6.4f, %6.4f, %6.4f, %6.4f, %6.4f,"
1401 "%6.4f, \t\t"
1402 " %6.4f, %6.4f, %6.4f, %6.4f, \t"
1403 " %6.4f, %6.4f, %6.4f, %6.4f, \t"
1404 " %6.4f, %6.4f, %6.4f, %6.4f, \t"
1405 " %6.4f, %6.4f, %6.4f,\t\t",
1406
1407 ctrl_platooning_state.v_leader ,
1408 ctrl_platooning_state.u_v_x ,
1409 ctrl_platooning_state.u_v_ffleader ,
1410 ctrl_platooning_state.u_v_ffleader_acc ,
1411 ctrl_platooning_state.u_v_ffpre ,
1412 ctrl_platooning_state.u_v_ffpre_acc ,
1413
1414 ctrl_platooning_state.up_v_x_out ,
1415 ctrl_platooning_state.ui_v_x_out ,
1416 ctrl_platooning_state.ud_v_x_out ,
1417 ctrl_platooning_state.u_v_x_out_p ,
1418 ctrl_platooning_state.u_v_x_out ,
1419
1420 ctrl_platooning_state.up_v_ffleader_out ,
1421 ctrl_platooning_state.ui_v_ffleader_out ,
1422 ctrl_platooning_state.ud_v_ffleader_out ,
1423 ctrl_platooning_state.u_v_ffleader_out_p ,
1424 ctrl_platooning_state.u_v_ffleader_out ,
1425
1426 ctrl_platooning_state.up_v_ffpre_out ,
1427 ctrl_platooning_state.ui_v_ffpre_out ,
1428 ctrl_platooning_state.ud_v_ffpre_out ,
1429 ctrl_platooning_state.u_v_ffpre_out_p ,
1430 ctrl_platooning_state.u_v_ffpre_out ,
1431
1432 ctrl_platooning_state.up_omega_out ,
1433 ctrl_platooning_state.ui_omega_out ,
1434 ctrl_platooning_state.ud_omega_out ,
1435 ctrl_platooning_state.u_omega_wo_rf );
1436
1437 fprintf(fd , "%d, %d, %d, %d, \t\t",
1438 started_flag ,
1439 should_stop_flag ,
1440 not_start_bypass ,
1441 stop_bypass
1442 );
1443 fprintf(fd , "% 6.2f, % 6.2f, % 6.2f,\t\t",
1444 vstate.ultrasonic_delta_x ,
1445 vstate.ultrasonic_delta_x_bk ,
1446 ctrl_platooning_state.delta_x_filtered
1447 );
1448 fprintf(fd , "% 6.2f, % 6.2f, % 6.2f, % 6.2f,"
1449 "% 6.2f,% 6.2f, %d\n",
1450 ctrl_platooning_state.acc_leader ,
1451 vstate.imu_accx ,
1452 ctrl_platooning_state.x_leader ,
1453 vstate.x ,
1454 ctrl_platooning_state.u_v_ffleader_rf_before_sat ,
1455 ctrl_platooning_state.v_dsr_leader ,
1456 ctrl_platooning_state.ctrl_flag_leader
1457 );
1458
1459 fflush( fd );
1460 }
1461 }
1462 void ctrl_platooning_loop() {
1463
1464 static int leader_v_dsr_print_flag = 0;
1465 // information gathering from sensor or communication
1466

```

```

1467     // sensor
1468     double delta_x = vstate.ultrasonic_delta_x;
1469     double delta_x_fitlered = 0.4;
1470
1471     // //get leader info from communicaiton
1472     struct v2v_msg *leader_msg
1473     = get_recent_msg(ctrl_platooning_para.leader_id);
1474     // get msg from immediately predecessor not enable yet
1475     if (leader_msg == NULL) {
1476         printf("BUG!!! leader_msg = NULL, leader_id = %d\n",
1477               ctrl_platooning_para.leader_id);
1478         return;
1479     }
1480     else{
1481         ctrl_platooning_state.ctrl_flag_leader
1482         = leader_msg->ctrl_flag;
1483         if (fabs(leader_msg->v) >0.001
1484             && 0 == started_flag){
1485             printf("Leader #v# nonzero received"
1486                   ": % .2f\n",leader_msg->v);
1487             if(fabs(leader_msg->v) < 1){
1488                 started_flag = 1;
1489                 not_start_bypass = 0;
1490                 vehicle_status = 1;
1491                 running_time_start_ms = vstate.timestamp;
1492                 printf("Vehicle Start at %ld ms !\n",
1493                       running_time_start_ms);
1494                 if(delta_x > 0.01 && delta_x < 3){
1495                     vstate.ultrasonic_delta_x_bk = delta_x;
1496                     printf("<Ultrasonic bk Initialized at"
1497                           "% .2f !>\n", delta_x);
1498                 }else{
1499                     vstate.ultrasonic_delta_x_bk = delta_x;
1500                     printf("Ultrasonic bk Initialized Error!"
1501                           " delta_x at % .2f !\n", delta_x);
1502                     printf("Ultrasonic bk Initialized"
1503                           " at % .2f, "
1504                           " replace bk with 0.4m \n",
1505                           vstate.ultrasonic_delta_x_bk);
1506                     vstate.ultrasonic_delta_x_bk = 0.4;
1507                 }
1508                 vstate.x = 0;
1509                 vstate.y = 0;
1510             }else{
1511                 printf("Communication Error ?"
1512                   ": v_recv: % .2f\n",leader_msg->v);
1513             }
1514         }
1515         // v-dsr information getting
1516         if( fabs(leader_msg->v_dsr) > 0.001)
1517             && 0 == started_flag){
1518             printf("Leader #v_dsr# nonzero received"
1519                   ": % .2f\n",leader_msg->v_dsr);
1520             if( fabs(leader_msg->v_dsr - 0.3) < 0.01 ){
1521                 running_time_start_ms = vstate.timestamp;
1522                 printf("Vehicle Start at %ld ms !\n",
1523                   running_time_start_ms);
1524                 started_flag = 1;
1525                 not_start_bypass = 0;
1526                 vehicle_status = 1;
1527
1528                 if(delta_x > 0.01 && delta_x < 3){
1529                     vstate.ultrasonic_delta_x_bk = delta_x;
1530                     printf("<Ultrasonic bk Initialized at"
1531                           "% .2f !>\n", delta_x);
1532                 }else{
1533

```

```

1534     vstate.ultrasonic_delta_x_bk = delta_x;
1535     printf("Ultrasonic bk Initialized Error! "
1536             "delta_x at % .2f !\n", delta_x);
1537     printf("Ultrasonic bk Initialized"
1538             "at % .2f,"
1539             " replace bk with 0.4m \n",
1540             vstate.ultrasonic_delta_x_bk);
1541     vstate.ultrasonic_delta_x_bk = 0.4;
1542 }
1543
1544     vstate.x = 0;
1545     vstate.y = 0;
1546 } else{
1547     printf("Communication Error ?"
1548         ": v-dsr recv: % .2f\n",
1549             leader_msg->v-dsr);
1550 }
1551 }
1552 ctrl_platooning_state.v-dsr.leader = leader_msg->v-dsr;
1553
1554
1555 if (1 == started_flag ){
1556     if(delta_x >= 3 || delta_x < 0.03){
1558
1559         printf("ULTRASONIC ERROR OUT OF RANGE!"
1560             "% .2f\n",delta_x);
1561
1562         if(vstate.ultrasonic_delta_x_bk < 0.05 ||
1563             vstate.ultrasonic_delta_x_bk > 2 ){
1564             printf("ULTRASONIC BK OUT OF RANGE!"
1565                 "% .2f\n",delta_x);
1566             printf("change current and bk to 0.4\n");
1567             delta_x = 0.4;
1568             vstate.ultrasonic_delta_x_bk = 0.4;
1569         } else{
1570             delta_x = vstate.ultrasonic_delta_x_bk;
1571         }
1572
1573     } else{
1574         // sensor reading is in the range
1575         if(fabs(delta_x - vstate.ultrasonic_delta_x_bk)
1576             >= 0.1){
1577
1578             if(fabs(delta_x - vstate.ultrasonic_delta_x_bk)
1579                 >= 0.3){
1580                 printf("ULTRASONIC JUMP HUGE\n");
1581                 delta_x = vstate.ultrasonic_delta_x_bk;
1582
1583             } else{
1584                 // only update bk when surge
1585                 //less than certain bound
1586                 printf("ULTRASONIC NOISE!"
1587                     "delta_x now: % .2f bk: % .2f\n",
1588                     delta_x,
1589                     vstate.ultrasonic_delta_x_bk);
1590                 delta_x_filtered
1591                 = 0.5*(delta_x +
1592                     vstate.ultrasonic_delta_x_bk);
1593                 vstate.ultrasonic_delta_x_bk
1594                 = delta_x_filtered;
1595             }
1596         } else{
1597             delta_x_filtered = delta_x;
1598             vstate.ultrasonic_delta_x_bk
1599             = delta_x_filtered;
1600         }

```

```

1601         // only use filter when sensor reading in range
1602     }
1603 }
1604
1605
1606
1607
1608     // getting delta_theta by camera
1609     double delta_theta = vstate.camera_delta_theta;
1610     if (fabs(delta_theta) >= 0.30 ){
1611         printf("CAMERA ERROR!\n");
1612         // use 0/ imu data as backup
1613         // delta_theta = 0;
1614         delta_theta = vstate.camera_delta_theta_p;
1615         if(fabs(delta_theta) >= 0.30){
1616             printf("CAMERA BK Error!\n");
1617             delta_theta = 0;
1618         }
1619     }
1620     vstate.camera_delta_theta_p = delta_theta;
1621
1622
1623
1624
1625     double x_leader;
1626
1627     double v_leader = leader_msg->v;
1628
1629
1630     if(v_leader >0.5 || v_leader < 0){
1631         printf("LEADER INFO BACKUP ERROR v_leader"
1632             ": % 6.2f ues bk: %6.2f !\n",
1633             v_leader,
1634             ctrl_platooning_state.v_leader_p);
1635         v_leader = ctrl_platooning_state.v_leader_p;
1636     }
1637
1638     v_leader
1639     = 0.5*(v_leader + ctrl_platooning_state.v_leader_p);
1640
1641     x_leader
1642     = ctrl_platooning_state.x_leader_p + v_leader * 0.1;
1643
1644     double acc_leader = leader_msg->accx;
1645
1646     if(vstate.v < 0 || vstate.v > 1){
1647         printf("vstate.v at % 6.2f"
1648             "error replace by 0.3 \n",vstate.v);
1649         printf("encoder info wl: % 6.2f wr: % 6.2f\n",
1650             vstate.encoder_wl,vstate.encoder_wl);
1651         vstate.v = 0.3;
1652     }
1653
1654
1655     // current error computing
1656     double error_delta_x
1657     = delta_x_fitered - ctrl_platooning_state.delta_x_dsr;
1658     double error_theta
1659     = delta_theta;
1660     double error_leader
1661     = v_leader - vstate.v;
1662     double error_leader_acc
1663     = acc_leader - vstate.accx;
1664     double u_v
1665     = 0;
1666     double u_omega
1667     = 0;
1668
1669     // status checking
1670     // vehicle status checking
1671
1672     // stop status checking

```

```

1669     // vehicle will stop if
1670     // too close to predecessor
1671     // off-track ,
1672
1673
1674     if(v.leader < 0.25 && 1 == started_flag ){
1675         // leader stop check
1676         //printf("Leader speed less than 0.25 and started\n");
1677         if(run_lapsed_time_ms >3000){
1678             leader_stopped_flag++;
1679             if(1 == leader_stopped_flag){
1680                 printf("LEADER STOPPED after this vehicle"
1681                     "run % 6.2f sec \n",
1682                     (double)(run_lapsed_time_ms)/1000.0);
1683                 //should_stop_flag = 1;
1684             }
1685         } else{
1686             ;
1687         }
1688     }
1689
1690     // if camera is too close to object ahead
1691     // replace theta info from imu
1692     static int camera_too_close_flag = 0;
1693
1694
1695     // double lowest_cruise_speed = 0.05;
1696     if (delta_x<= 0.30 && 1 == started_flag){
1697         camera_too_close_flag++;
1698         if(camera_too_close_flag == 1){
1699             printf("camera view too close brake bypass 1\n");
1700             //error_theta = 0 - vstate.imu.theta;
1701             error_theta = ctrl_platooning_state.error_theta_p;
1702             if(fabs(delta_theta) >= 0.30){
1703                 printf("CAMERA BK Error!\n");
1704                 delta_theta = 0;
1705             }
1706             if (brake_bypass<= 0)
1707                 brake_bypass = 1;
1708         } else{
1709             ;
1710         }
1711     }
1712
1713
1714
1715
1716     static int off_track_stop_print_flag = 0;
1717     if(fabs(error_theta) >= 0.25){
1718         // off-track
1719         off_track_stop_print_flag++;
1720         if(5 == off_track_stop_print_flag ){
1721             printf("OFF TRACK AND STOP offtrack_cnt %d \n",
1722                   off_track_stop_print_flag );
1723         }
1724     }
1725
1726 }
1727
1728
1729
1730     static int emergency_stop_print_flag = 0;
1731     if(delta_x <= 0.10 && started_flag == 1 && delta_x > 0){
1732         //too close set stop flag
1733         // >= 0 to prevent error reading at beginning
1734         emergency_stop_print_flag++;
1735         if (brake_bypass<= 0)
1736             brake_bypass = 10;

```

```

1736
1737     if(1 == emergency_stop_print_flag){
1738         printf("TOO CLOSE at %5.2f AND STOP!\n",delta_x );
1739     }
1740
1741
1742
1743
1744     if(emergency_stop_print_flag > 0
1745     || off_track_stop_print_flag >4){
1746         // started_flag = 0;
1747         // be caution , when stop flag is on
1748         // don't clear started_flag immediately
1749         should_stop_flag = 1;
1750         not_start_bypass = 0;
1751         stop_bypass++;
1752         vehicle_status--;
1753         if(-1 == vehicle_status){
1754             printf("VEHICLE SHOULD STOP!\n");
1755             printf("STOP FLAGS STATUS: emgcy: "
1756                   "%d off_track: %d \n",
1757                   emergency_stop_print_flag ,
1758                   off_track_stop_print_flag );
1759         }
1760     }
1761     if (running_time_start_ms >0 ){
1762         run_lapsed_time_ms
1763         = vstate.timestamp - running_time_start_ms;
1764     }
1765
1766     // error update
1767     // CAUTION: Here we use (delta_x - delta_x_dsr).
1768     // This is intentional.
1769
1770     double error_delta_x_p
1771     = ctrl_platooning_state.error_delta_x_p;
1772     double error_delta_x_pp
1773     = ctrl_platooning_state.error_delta_x_pp;
1774
1775
1776     double error_leader_p
1777     = ctrl_platooning_state.error_leader_p;
1778     //double error_leader_pp
1779     = ctrl_platooning_state.error_leader_pp;
1780     double error_leader_sum
1781     = ctrl_platooning_state.error_leader_sum_p
1782         + error_leader;
1783
1784     double error_theta_sum
1785     = ctrl_platooning_state.error_theta_sum_p
1786         + error_theta;
1787     double error_theta_p
1788     = ctrl_platooning_state.error_theta_p;
1789
1790     if(vehicle_status <= 0){
1791         error_delta_x = 0;
1792         error_leader_acc = 0;
1793         error_theta = 0;
1794
1795         error_delta_x_p = 0;
1796         error_delta_x_pp = 0;
1797         error_leader_sum = 0;
1798
1799         error_theta_sum = 0;
1800         error_theta_p = 0;
1801     }
1802     // Derive velocity component from delta_x info.
1803     // CAUTION: This is PID incremental style
1804     double up_v_x_out = 0;

```

```

1805     double ui_v_x_out = 0;
1806     double ud_v_x_out = 0;
1807     double u_v_x_out = 0;
1808     double u_v_x_rfl1 = 0;
1809     double u_v_x_rfl2 = 0;
1810
1811     double x_v_kd_local = 0;
1812     // Derive velocity component from leader info.
1813     // CAUTION: This is PID incremental style!!!
1814     double up_v_ffleader_out=0;
1815     double ui_v_ffleader_out=0;
1816     double ud_v_ffleader_out=0;
1817
1818
1819     double ff_v_kp_local = ctrl_platooning_para.kp_ffleader;
1820     double ff_v_ki_local = ctrl_platooning_para.ki_ffleader;
1821     double ff_v_kd_local = ctrl_platooning_para.kd_ffleader;
1822
1823
1824
1825     if( vstate.v > v_leader && vstate.v > 0.3){
1826         follower_catched_up_flag++;
1827
1828         if(1 == follower_catched_up_flag){
1829             printf("<CATCH UP!> v_leader"
1830                   ": %6.2f v: %6.2f\n",
1831                   v_leader, vstate.v);
1832             printf("ff_v_ PID now is:"
1833                   "%6.2f, %6.2f, % 6.2f \n",
1834                   ff_v_kp_local,
1835                   ff_v_ki_local,
1836                   ff_v_kd_local);
1837         }
1838     }
1839
1840
1841     static int cancel_kd_flag = 0;
1842     if (run_lapsed_time_ms > 3000){
1843         x_v_kd_local = 0;
1844         cancel_kd_flag++;
1845         if(1 == cancel_kd_flag){
1846             printf("cancel kd term \r\n");
1847         }
1848     } else if (1 == follower_catched_up_flag ){
1849         x_v_kd_local = 0;
1850         if(1 == started_flag){
1851             cancel_kd_flag++;
1852             if(1 == cancel_kd_flag){
1853                 }
1854         }
1855     } else{
1856         x_v_kd_local = ctrl_delta_x_theta_para.kd_delta_x;
1857     }
1858
1859     u_v_x_out = ctrl_pid_inc(
1860         ctrl_platooning_state.u_v_x_out_p ,
1861         error_delta_x , error_delta_x_p ,
1862         error_delta_x_pp ,
1863         ctrl_platooning_para.kp_delta_x ,
1864         ctrl_platooning_para.ki_delta_x ,
1865         x_v_kd_local , &up_v_x_out , &ui_v_x_out ,
1866         &ud_v_x_out , mc_init.ctrl_loop_period / 1000.0);
1867
1868
1869     // controller roll off on the output v (low pass filter)
1870     u_v_x_rfl1 = ctrl_roll_off_once
1871     (ctrl_platooning_state.u_v_x_rfl1_p ,
1872      u_v_x_out ,

```

```

1873             rf_coeff);
1874     u_v_x_rf2 = ctrl_roll_off_once
1875     (ctrl_platooning_state.u_v_x_rf2_p ,
1876      u_v_x_rf1 ,
1877      rf_coeff);
1878 // u_v_x_rf2 = u_v_x_rf1;
1879 if(running_time_start_ms < 1000 || started_flag == 0){
1880     if (u_v_x_rf2 < 0){
1881         ctrl_platooning_state.u_v_x_out_p = 0;
1882         ctrl_platooning_state.u_v_x_rf1_p = 0;
1883         ctrl_platooning_state.u_v_x_rf2_p = 0;
1884         up_v_x_out = 0;
1885         u_v_x_rf1 = 0;
1886         u_v_x_rf2 = 0;
1887     }
1888 }
1889 if(follower_catched_up_flag > 1){
1890     ff_v_kp_local = 0;
1891 }
1892 double u_v_ffleader_out = ctrl_pid(error_leader ,
1893     error_leader_sum ,
1894     error_leader_p ,
1895     ff_v_kp_local ,
1896     ff_v_ki_local ,
1897     ff_v_kd_local ,
1898     &up_v_ffleader_out , &ui_v_ffleader_out ,
1899     &ud_v_ffleader_out ,
1900     mc_init.ctrl_loop_period / 1000.0);
1901
1902
1903 double u_v_ffleader_rf = ctrl_roll_off_once(
1904     ctrl_platooning_state.u_v_ffleader_rf_p ,
1905     u_v_ffleader_out ,
1906     rf_coeff);
1907
1908 //u_v_ffleader_rf = u_v_ffleader_out;
1909 //bypass roll if only P control
1910 ctrl_platooning_state.u_v_ffleader_rf_before_sat
1911 = u_v_ffleader_rf;
1912
1913
1914 u_v_ffleader_rf = ctrl_threshold_and_saturation
1915 (u_v_ffleader_rf , 0, 0, 0.3);
1916 //if(u_v_ffleader_rf)
1917
1918 double u_v_ffleader_acc
1919 = ctrl_platooning_para.ka_ffleader
1920     * error_leader_acc;
1921
1922
1923 // merge several control efforts
1924
1925
1926 double u_v_total
1927 = u_v_x_rf2 + u_v_ffleader_rf + u_v_ffleader_acc;
1928 double u_v_rf
1929 = ctrl_roll_off_once(ctrl_platooning_state.u_v_rf_p ,
1930     u_v_total , 0.8);
1931
1932 double up_omega_out;
1933 double ui_omega_out;
1934 double ud_omega_out;
1935
1936 double u_omega_out = ctrl_pid(
1937     error_theta ,
1938     error_theta_sum ,
1939     error_theta_p ,
1940     ctrl_platooning_para.kp_theta ,
1941     ctrl_platooning_para.ki_theta ,

```

```

1942     ctrl_platooning_para.kd_theta ,
1943     &up_omega_out ,
1944     &ui_omega_out ,
1945     &ud_omega_out ,
1946     mc_init.ctrl_loop_period / 1000.0
1947 );
1948
1949 // NOTE: If you has d-term then roll off ,
1950 // (low pass filter twice)
1951
1952 double u_omega_rf1
1953 = ctrl_roll_off_once(ctrl_platooning_state.u_omega_rf1_p,
1954     u_omega_out, rf_coeff);
1955 double u_omega_rf2
1956 = ctrl_roll_off_once(ctrl_platooning_state.u_omega_rf2_p,
1957     u_omega_rf1, rf_coeff);
1958
1959
1960 if(fabs(u_omega_out)>10 || fabs(u_v) >2){
1961     printf("Controller failure , uw:"
1962         "% 6.2f, u_v: % 6.2f\n",u_omega_out,u_v);
1963     if(fabs(u_v) >2){
1964         u_v = ctrl_platooning_state.u_v_rf_p;
1965     }
1966     if(fabs(u_omega_out)>10){
1967         u_omega_out = 0;
1968     }
1969 }
1970 // control effort restriction
1971 u_v = ctrl_threshold_and_saturation(u_v_rf, 0, 0, 0.6);
1972
1973 u_omega = ctrl_threshold_and_saturation
1974 (u_omega_rf2, 0, -2, 2);
1975
1976 // no direction control at very low speed
1977 // if(u_v < 0.1){
1978 //     u_omega = 0;
1979 //}
1980 double wl_dsr;
1981 double wr_dsr;
1982
1983 if(l== started_flag && 0 == leader_stopped_flag
1984     && 0 == follower_catched_up_flag){
1985     u_v = ctrl_threshold_and_saturation
1986 (u_v_rf, 0, 0, 0.6);
1987 }
1988
1989 inverse_kinematics(u_v, u_omega, &wl_dsr, &wr_dsr);
1990
1991 // final safety check to prevent backward rotation in ETT
1992 if(wl_dsr < 0){ wl_dsr = 0; }
1993 if(wr_dsr < 0){ wr_dsr = 0; }
1994 static int stop_in_advance_flag = 0;
1995 int h2l_status = 1;
1996
1997 if(leader_stopped_flag >0 && delta_x_filtered <= 0.44 ){
1998
1999     stop_in_advance_flag++;
2000     should_stop_flag = 1;
2001     if(l==stop_in_advance_flag){
2002         printf("STOP IN ADVANCE at delta_x"
2003             ": % 6.2f and BRAKE at %ld\n",
2004             delta_x_filtered,
2005             run_lapsed_time_ms);
2006         brake_bypass = 10;
2007         h2l_status = -1;
2008     } else{

```

```

2009         h2l_status = 0;
2010     }
2011     u_v = 0;
2012     u_v_x_rf2 = 0;
2013     ctrl_platooning_state.u_v_x_out_p = 0;
2014     ctrl_platooning_state.u_v_x_rf1_p = 0;
2015     ctrl_platooning_state.u_v_x_rf2_p = 0;
2016 }
2017 }
2018
2019 // h2l handling bypass and normal cases
2020 if(1 == not_start_bypass){
2021     h2l_status = 0;
2022 }
2023
2024 if(stop_bypass >0){
2025     if(1 == stop_bypass)
2026         h2l_status = -1;
2027     else
2028         h2l_status = 0;
2029 }
2030
2031 if(brake_bypass > 0){
2032     h2l_status = -1;
2033     printf("brake_bypass: %d\n ",brake_bypass );
2034     brake_bypass--;
2035 }
2036
2037 // sending lower level command according to h2l_status
2038 if(1 == h2l_status ){
2039     // normal operation
2040     h2l_send_wl_wr_msg(( float )wl_dsr , ( float )wr_dsr );
2041 } else if(0 == h2l_status ){
2042     // turn off motor
2043     h2l_send_wl_wr_msg(0 , 0 );
2044     //printf(" send 0 0\r\n");
2045 } else{
2046     // brake action
2047     h2l_send_wl_wr_msg(-1 , -1 );
2048     printf("send -1 -1\r\n");
2049 }
2050
2051
2052
2053
2054 // save current intermediate state , for debug
2055 ctrl_platooning_state.delta_x_filtered = delta_x_filtered ;
2056
2057 ctrl_platooning_state.x_leader      = x_leader;
2058 ctrl_platooning_state.v_leader      = v_leader;
2059 ctrl_platooning_state.acc_leader    = acc_leader;
2060
2061 ctrl_platooning_state.wl_dsr = wl_dsr;
2062 ctrl_platooning_state.wr_dsr = wr_dsr;
2063 ctrl_platooning_state.u_v = u_v;
2064 ctrl_platooning_state.u_omega = u_omega;
2065
2066 ctrl_platooning_state.error_delta_x = error_delta_x;
2067 ctrl_platooning_state.error_leader = error_leader;
2068
2069 ctrl_platooning_state.error_theta = error_theta;
2070
2071 ctrl_platooning_state.u_v_x = u_v_x_rf2;
2072 ctrl_platooning_state.u_v_ffleader = u_v_ffleader_rf;
2073 ctrl_platooning_state.u_v_ffleader_acc = u_v_ffleader_acc;
2074
2075
2076 ctrl_platooning_state.up_v_x_out = up_v_x_out;
2077 ctrl_platooning_state.ui_v_x_out = ui_v_x_out;

```

```

2078     ctrl_platooning_state.ud_v_x_out = ud_v_x_out;
2079     ctrl_platooning_state.u_v_x_out = u_v_x_out;
2080
2081     ctrl_platooning_state.up_v_ffleader_out
2082     = up_v_ffleader_out;
2083     ctrl_platooning_state.ui_v_ffleader_out
2084     = ui_v_ffleader_out;
2085     ctrl_platooning_state.ud_v_ffleader_out
2086     = ud_v_ffleader_out;
2087     ctrl_platooning_state.u_v_ffleader_out
2088     = u_v_ffleader_out;
2089
2090
2091     ctrl_platooning_state.up_omega_out = up_omega_out;
2092     ctrl_platooning_state.ui_omega_out = ui_omega_out;
2093     ctrl_platooning_state.ud_omega_out = ud_omega_out;
2094     ctrl_platooning_state.u_omega_rf = u_omega_out;
2095
2096     // print control state to file
2097     ctrl_platooning_print(ctrl_platooning_state.log_file);
2098     ctrl_platooning_print
2099     (ctrl_platooning_state.recent_log_file);
2100
2101     // update previous state
2102     ctrl_platooning_state.u_v_rf_p = u_v_rf;
2103     // from delta_x
2104
2105     ctrl_platooning_state.u_v_x_out_p = u_v_x_out;
2106
2107     ctrl_platooning_state.error_delta_x_p = error_delta_x;
2108     ctrl_platooning_state.error_delta_x_pp = error_delta_x_p;
2109
2110     ctrl_platooning_state.u_v_x_rf1_p = u_v_x_rf1;
2111     ctrl_platooning_state.u_v_x_rf2_p = u_v_x_rf2;
2112
2113     // from leader vehicle
2114     ctrl_platooning_state.v_leader_p = v_leader;
2115     ctrl_platooning_state.x_leader_p = x_leader;
2116     ctrl_platooning_state.u_v_ffleader_out_p
2117     = u_v_ffleader_out;
2118
2119     ctrl_platooning_state.error_leader_p
2120     = error_leader;
2121     ctrl_platooning_state.error_leader_sum_p
2122     = error_leader_sum;
2123
2124     ctrl_platooning_state.u_v_ffleader_rf_p
2125     = u_v_ffleader_rf;
2126
2127
2128     // from theta
2129
2130     ctrl_platooning_state.error_theta_p = error_theta;
2131     ctrl_platooning_state.error_theta_sum_p
2132     = error_theta_sum;
2133
2134     ctrl_platooning_state.u_omega_rf1_p = u_omega_rf1;
2135     ctrl_platooning_state.u_omega_rf2_p = u_omega_rf2;
2136
2137 }
2138
2139 // _____ outer loop thread _____
2140
2141 void *ctrl_loop_thread(void *para) {
2142
2143     printf("<ctrl_loop> Control loop thread started.\n");
2144
2145     ctrl_v_theta_init();
2146     ctrl_x_y_init();
2147     ctrl_delta_x_theta_init();

```

```

2148     ctrl_line_track_init();
2149     ctrl_platooning_init();
2150
2151     while (!ctrl_loop_should_stop) {
2152
2153         sem_wait(&ctrl_loop_sem);
2154
2155         if (vstate.ctrl_state > 0) {
2156
2157             ctrl_loop_current_rt -= mc_init.ctrl_loop_period;
2158             if (ctrl_loop_current_rt < 0) {
2159                 vstate.ctrl_state = 0;
2160                 h2l_send_stop_msg();
2161             }
2162
2163             // printf("<ctrl_loop> iterative.\n");
2164
2165             switch (vstate.ctrl_outer_loop_mode) {
2166
2167                 case CTRLOPENLOOP: {
2168                     ctrl_open_loop();
2169                     break;
2170                 }
2171                 case CTRLWLWR: {
2172                     ctrl_wl_wr_loop();
2173                     break;
2174                 }
2175                 case CTRLV_OMEGA: {
2176                     ctrl_v_omega_loop();
2177                     break;
2178                 }
2179                 case CTRL_V_THETA: {
2180                     ctrl_v_theta_loop();
2181                     break;
2182                 }
2183                 case CTRL_X_Y: {
2184                     ctrl_x_y_loop();
2185                     break;
2186                 }
2187                 case CTRL_DELTA_X_THETA: {
2188                     ctrl_delta_x_theta_loop();
2189                     break;
2190                 }
2191                 case CTRLLINE_TRACK: {
2192                     ctrl_line_track_loop();
2193                     break;
2194                 }
2195                 case CTRLPLATOONING: {
2196                     ctrl_platooning_loop();
2197                     break;
2198                 }
2199                 default:
2200                     break;
2201             }
2202
2203             if (vstate.ctrl_flag == CTRLSTART) {
2204
2205                 h2l_send_start_msg();
2206                 vstate.ctrl_flag = CTRLNONE;
2207
2208             }
2209
2210         }
2211
2212         if (vstate.ctrl_flag == CTRLSTOP) {
2213
2214

```

```

2215         h2l_send_stop_msg();
2216         vstate.ctrl_flag = CTRLNONE;
2217         vstate.ctrl_state = 0;
2218         printf("stop executed!!!\n");
2219     }
2220 }
2221 }
2222
2223     ctrl_v_theta_exit();
2224     ctrl_x_y_exit();
2225     ctrl_delta_x_theta_exit();
2226
2227     ctrl_line_track_exit();
2228     ctrl_platooning_exit();
2229
2230     printf("<ctrl_loop> Control loop thread exiting...\n");
2231
2232     return NULL;
2233 }
2234
2235 // _____ outer loop interface _____
2236
2237 // CAUTION: ctrl_setup() are called by command thread
2238
2239 // FIXIT: for setup, start, and stop, /
2240 // we should put these h2l_send_xxx()
2241 // function call to ctrl loop thread
2242
2243 void ctrl_setup() {
2244
2245     h2l_send_setup_msg();
2246 }
2247
2248 void ctrl_start() {
2249
2250     ctrl_loop_current_rt = mc_init.running_time;
2251     vstate.ctrl_state = 1;
2252     vstate.ctrl_flag = CTRLSTART;
2253 }
2254
2255 void ctrl_stop() {
2256
2257     vstate.ctrl_flag = CTRLSTOP;
2258 }
2259
2260 void ctrl_open_loop_set_pwm(int16_t pwm_left_dsr,
2261     int16_t pwm_right_dsr) {
2262
2263     if (vstate.ctrl_state == 0
2264         || (vstate.ctrl_state == 1
2265             && vstate.ctrl_outer_loop_mode
2266             == CTRLOPENLOOP)) {
2267         ctrl_open_loop_state.pwm_left_dsr = pwm_left_dsr;
2268         ctrl_open_loop_state.pwm_right_dsr = pwm_right_dsr;
2269         vstate.ctrl_outer_loop_mode = CTRLOPENLOOP;
2270         vstate.ctrl_inner_loop_mode = CTRLOPENLOOP;
2271     } else {
2272
2273     }
2274 }
2275
2276 void ctrl_set_wl_wr(double wl_dsr, double wr_dsr) {
2277
2278     if (vstate.ctrl_state == 0
2279         || (vstate.ctrl_state == 1
2280             && vstate.ctrl_outer_loop_mode
2281             == CTRLWLWR)) {
2282         ctrl_wl_wr_state.wl_dsr = wl_dsr;

```

```

2283         ctrl_wl_wr_state.wr_dsr = wr_dsr;
2284         vstate.ctrl_outer_loop_mode = CTRL_WLWR;
2285         vstate.ctrl_inner_loop_mode = CTRL_WLWR;
2286     } else {
2287     }
2288 }
2289 }
2290
2291 void ctrl_set_v_omega(double v_dsr, double omega_dsr) {
2292
2293     if (vstate.ctrl_state == 0
2294         || (vstate.ctrl_state == 1
2295             && vstate.ctrl_outer_loop_mode
2296             == CTRL_V_OMEGA)) {
2297         ctrl_v_omega_state.v_dsr = v_dsr;
2298         ctrl_v_omega_state.omega_dsr = omega_dsr;
2299         vstate.ctrl_outer_loop_mode = CTRL_V_OMEGA;
2300         vstate.ctrl_inner_loop_mode = CTRL_WLWR;
2301     } else {
2302     }
2303 }
2304 }
2305
2306 void ctrl_set_v_theta(double v_dsr, double theta_dsr) {
2307
2308     if (vstate.ctrl_state == 0
2309         || (vstate.ctrl_state == 1
2310             && vstate.ctrl_outer_loop_mode
2311             == CTRL_V_THETA)) {
2312         ctrl_v_theta_state.v_dsr = v_dsr;
2313         ctrl_v_theta_state.theta_dsr = theta_dsr;
2314         vstate.ctrl_outer_loop_mode = CTRL_V_THETA;
2315         vstate.ctrl_inner_loop_mode = CTRL_WLWR;
2316     } else {
2317     }
2318 }
2319 }
2320
2321 void ctrl_set_x_y(double x_dsr, double y_dsr) {
2322
2323     if (vstate.ctrl_state == 0
2324         || (vstate.ctrl_state == 1
2325             && vstate.ctrl_outer_loop_mode
2326             == CTRL_X_Y)) {
2327         ctrl_x_y_state.x_dsr = x_dsr;
2328         ctrl_x_y_state.y_dsr = y_dsr;
2329         vstate.ctrl_outer_loop_mode = CTRL_X_Y;
2330         vstate.ctrl_inner_loop_mode = CTRL_WLWR;
2331     } else {
2332     }
2333 }
2334 }
2335
2336 void ctrl_set_delta_x_theta
2337 (double delta_x_dsr, double theta_dsr) {
2338
2339     if (vstate.ctrl_state == 0
2340         || (vstate.ctrl_state == 1
2341             && vstate.ctrl_outer_loop_mode
2342             == CTRL_DELTA_X_THETA)) {
2343         ctrl_delta_x_theta_state.delta_x_dsr = delta_x_dsr;
2344         ctrl_delta_x_theta_state.theta_dsr = theta_dsr;
2345         vstate.ctrl_outer_loop_mode = CTRL_DELTA_X_THETA;
2346         vstate.ctrl_inner_loop_mode = CTRL_WLWR;
2347     } else {
2348     }
2349 }
2350 }
```

```

2351
2352 void ctrl_set_line_track(double v_dsr) {
2353
2354     if (vstate.ctrl_state == 0
2355         || (vstate.ctrl_state == 1
2356             && vstate.ctrl_outer_loop_mode
2357             == CTRL_LINE_TRACK)) {
2358         ctrl_line_track_state.v_dsr = v_dsr;
2359         vstate.ctrl_outer_loop_mode = CTRL_LINE_TRACK;
2360         vstate.ctrl_inner_loop_mode = CTRL_WLWR;
2361     } else {
2362
2363     }
2364 }
2365
2366 void ctrl_set_platooning(double delta_x_dsr ,double theta_dsr){
2367
2368     if (vstate.ctrl_state == 0
2369         || (vstate.ctrl_state == 1
2370             && vstate.ctrl_outer_loop_mode
2371             == CTRL_PLATOONING)) {
2372         ctrl_platooning_state.delta_x_dsr = delta_x_dsr ;
2373         ctrl_platooning_state.theta_dsr = theta_dsr ;
2374         vstate.ctrl_outer_loop_mode = CTRL_PLATOONING;
2375         vstate.ctrl_inner_loop_mode = CTRL_WLWR;
2376     } else {
2377
2378     }
2379
2380 }
```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4
5 #include <unistd.h>
6
7 #include "h2l.h"
8 #include "serial.h"
9 #include "ctrl_loop.h"
10 #include "pi_mc.h"
11
12
13
14 volatile int serial_should_stop = 0;
15
16 int serial_fd = -1;
17
18
19 int h2l_send_open_loop_msg(int16_t pwm_left,int16_t pwm_right){
20
21     int ret = 0;
22     struct h2l_open_loop msg;
23
24     h2l_set_header(&msg.header, sizeof(msg)
25     - sizeof(msg.header),
26                 OPCODE_OPENLOOP);
27     msg.v_left = pwm_left;
28     msg.v_right = pwm_right;
29
30     ret = serial_send_n_bytes(serial_fd ,
31     (char *)&msg, sizeof(msg));
32     serial_flush(serial_fd );
33
34     return ret;
35 }
36
```

```

37  int h2l_send_wl_wr_msg( float wl_dsr , float wr_dsr ) {
38
39      int ret = 0;
40      struct h2l_wl_wr msg;
41
42      h2l_set_header(&msg.header , sizeof(msg)
43 - sizeof(msg.header),
44             OPCODECTRLWLWR);
45      msg.wl_dsr = wl_dsr;
46      msg.wr_dsr = wr_dsr;
47
48      ret = serial_send_n_bytes(serial_fd ,
49      (char *)&msg, sizeof(msg));
50      serial_flush(serial_fd );
51
52      return ret;
53
54  }
55
56  int h2l_send_setup_msg() {
57
58      int ret = 0;
59      struct h2l_setup msg;
60
61      h2l_set_header(&msg.header , sizeof(msg)
62 - sizeof(msg.header),
63             OPCODESETUP);
64
65      msg.prefilter_coefficient
66      = ctrl_inner_para.prefilter_coefficient;
67      msg.roll_off_coefficient
68      = ctrl_inner_para.roll_off_coefficient;
69      msg.kp_left = ctrl_inner_para.kp_left;
70      msg.ki_left = ctrl_inner_para.ki_left;
71      msg.kd_left = ctrl_inner_para.kd_left;
72      msg.kp_right = ctrl_inner_para.kp_right;
73      msg.ki_right = ctrl_inner_para.ki_right;
74      msg.kd_right = ctrl_inner_para.kd_right;
75      msg.deadzone_threshold
76      = ctrl_inner_para.deadzone_threshold;
77      msg.deadzone_saturation
78      = ctrl_inner_para.deadzone_saturation;
79
80      ret = serial_send_n_bytes(serial_fd ,
81      (char *)&msg, sizeof(msg));
82      serial_flush(serial_fd );
83 // printf("setup: %d, %d\n", ret, sizeof(msg));
84
85      return ret;
86  }
87
88  static int h2l_send_header_only_msg(uint8_t opcode) {
89
90      int ret = 0;
91      struct h2l_header header;
92
93      h2l_set_header(&header , 0, opcode);
94
95      ret = serial_send_n_bytes(serial_fd ,
96      (char *)&header, sizeof(header));
97      serial_flush(serial_fd );
98
99      return ret;
100 }
101
102 int h2l_send_start_msg() {
103
104     return h2l_send_header_only_msg(OPCODESTART);
105 }
```

```

106
107 int h2l_send_stop_msg() {
108
109     return h2l_send_header_only_msg(OPCODE_STOP);
110 }
111
112 int h2l_send_debug_enable_msg() {
113
114     return h2l_send_header_only_msg(OPCODE_DEBUG_ENABLE);
115 }
116
117 int h2l_send_debug_disable_msg() {
118
119     return h2l_send_header_only_msg(OPCODE_DEBUG_DISABLE);
120 }
121
122
123
124 void h2l_update_vehicle_state
125 (struct h2l_vehicle_status *msg) {
126
127     update_vehicle_state(msg);
128
129 }
130
131
132
133 void h2l_print_vehicle_status(FILE *fd) {
134
135     if(fd == NULL) {
136         return;
137         //fd = stdout;
138     }
139     fprintf(fd, "%08u, %5.4f, %5.4f, %5.4f, %5.2f,"
140             "%5.2f, %5.3f, \t\t",
141             vstate.timestamp,
142             vstate.imu_theta, vstate.imu_accx,
143             vstate.imu_omega,
144             vstate.encoder_wl, vstate.encoder_wr,
145             vstate.ultrasonic_delta_x);
146 }
147
148 void h2l_print_ctrl_status_debug(FILE *fd,
149 struct h2l_ctrl_status_debug *msg) {
150
151     int i;
152
153     if(fd == NULL) {
154         return;
155         //fd = stdout;
156     }
157
158     //update vstate v_dsr;
159     vstate.v_dsr = (msg->wl_dsr + msg->wr_dsr)
160     * mc_init.wheel_radius / 2;
161
162     h2l_print_vehicle_status(fd);
163
164     fprintf(fd, "%5.2f, %5.2f, %5.2f, %5.2f, \t\t",
165             msg->wl_dsr, msg->wr_dsr,
166             msg->wl_dsr_filtered, msg->wr_dsr_filtered);
167
168
169     fprintf(fd, "%4d, %4d, %4d, %4d, %4d, %4d, \t\t",
170             msg->pwm_l, msg->pwm_r, msg->pwm_l_out,
171             msg->pwm_r_out,
172             msg->pwm_l_out_p, msg->pwm_r_out_p);
173

```

```

174     fprintf(fd , "%5.4f, %5.4f, %5.4f,"
175     "%5.4f, %5.4f, %5.4f, \t\t",
176     msg->err_wl , msg->err_wr ,
177     msg->err_wl_p , msg->err_wr_p ,
178     msg->err_wl_pp , msg->err_wr_pp );
179
180     fprintf(fd , "%5.4f, %5.4f, %5.4f, %5.4f,"
181     "%5.4f, %5.4f, \t\t",
182     msg->pwml_up , msg->pwml_ui , msg->pwml_ud ,
183     msg->pwmr_up , msg->pwmr_ui , msg->pwmr_ud );
184
185     for(i = 0; i < sizeof(msg->timestamps)
186         / sizeof(msg->timestamps[0]); i++) {
187
188         fprintf(fd , "%8u, " , msg->timestamps[i]);
189     }
190
191     fprintf(fd , "\n");
192
193     fflush(fd );
194 }
195
196
197
198
199
200
201
202
203 // ----- receive and dispatch -----
204
205
206
207 static int h2l_recv_process(char serial_buff[]) {
208
209     int ret;
210     int i;
211
212     int opcode = serial_buff[3];
213     int len = serial_buff[2];
214
215     // receive payload
216     ret = serial_recv_n_bytes(serial_fd , serial_buff
217     + MCLPROTO_HEADER_SIZE,
218     len);
219
220     // for(i = 0; i < len; i++) {
221     //     printf("%02X ", serial_buff[i]);
222     // }
223     // printf("\n");
224
225     switch (opcode) {
226
227     case OPCODE_VEHICLE_STATUS: {
228
229         struct h2l_vehicle_status *msg
230         = (struct h2l_vehicle_status *) (serial_buff);
231
232         update_vehicle_state(msg);
233         break;
234     }
235     case OPCODE_CTRLSTATUS_DEBUG: {
236
237         struct h2l_ctrl_status_debug *msg =
238             (struct h2l_ctrl_status_debug *)
239             (serial_buff);
240
241         h2l_print_ctrl_status_debug(log_file , msg);

```

```

242         h2l_print_ctrl_status_debug(recent_log_file , msg);
243     //      h2l_print_ctrl_status_debug(stdout , msg);
244     break;
245   }
246   default: break;
247 }
248
249   return 0;
250 }
251
252 void *h2l_receive_thread( void *para) {
253
254   int ret = 0;
255   int i = 0;
256
257   char c;
258
259   int comm_state = SERIAL_STATE_INIT;
260
261   char *arduino_serial_dev = mc_init.tty_file_name;
262
263   // Open serial port
264   serial_fd = serial_init(arduino_serial_dev ,
265   mc_init.tty_baud_rate);
266   if (serial_fd < 0) {
267     printf("Unable to open serial port!!!\n");
268     exit(-1);
269   }
270
271   printf("<mc_h2l_thread>\n"
272   "Serial port open successfully.\n");
273
274   char serial_buff[128];
275
276   while (!serial_should_stop) {
277
278     // CAUTION: This is polling every 1 ms!!!
279     // receive protocol header
280     ret = serial_recv_byte(serial_fd , &c);
281     if(ret < 0) {
282       printf("Serial read error!!!\n");
283       break;
284     } else if(ret == 0) {
285
286       usleep(1000);
287       continue;
288     }
289
290     serial_buff[comm_state] = c;
291
292     // printf("%02X ", c);
293
294     switch (comm_state) {
295     case SERIAL_STATE_INIT: {
296       if (c == SERIAL_MAGIC_1)
297         comm_state = SERIAL_STATE_MAGIC1;
298       else
299         comm_state = SERIAL_STATE_INIT;
300       break;
301     }
302     case SERIAL_STATE_MAGIC1: {
303       if (c == SERIAL_MAGIC_2)
304         comm_state = SERIAL_STATE_MAGIC2;
305       else
306         comm_state = SERIAL_STATE_INIT;
307       break;
308     }
309   }

```

```

310         case SERIAL_STATE_MAGIC2: {
311             comm_state = SERIAL_STATE_PROTO;
312             break;
313         }
314         case SERIALSTATE_PROTO: {
315             h2l_recv_process(serial_buff);
316             comm_state = SERIAL_STATE_INIT;
317             break;
318         }
319         default: {
320             comm_state = SERIAL_STATE_INIT;
321             break;
322         }
323     }
324 }
325 }
326 }
327 }
328 }
329 printf("<mc_h2l_thread> Serial thread exiting ...\\n");
330 serial_close(serial_fd);
331
332 return NULL;
333 }
334 }
```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <errno.h>
5
6 #include <unistd.h>
7 #include <time.h>
8 #include <signal.h>
9 #include <sys/types.h>
10
11 #include <pthread.h>
12 #include <semaphore.h>
13
14 #include "serial.h"
15 #include "h2l.h"
16 #include "v2v.h"
17 #include "ctrl_loop.h"
18 #include "camera.h"
19 #include "pi_mc.h"
20
21 struct mc_init mc_init;
22
23 FILE *meta_file = NULL;
24 FILE *recent_meta_file = NULL;
25 FILE *log_file = NULL;
26 FILE *recent_log_file = NULL;
27
28 // -----
29
30 void print_config(FILE *file) {
31
32     fprintf(file, "vehicle_id=%d\\n", mc_init.vehicle_id);
33     fprintf(file, "tty_file_name=%s\\n", mc_init.tty_file_name);
34     fprintf(file, "log_file_name=%s\\n", mc_init.log_file_name);
35     fprintf(file, "meta_file_name=%s\\n", mc_init.meta_file_name);
36     fprintf(file, "manager=%s:%d\\n",
37             mc_init.manager_ip, mc_init.manager_port);
38     fprintf(file, "v2v_period=%d, ctrl_loop_period=%d\\n",
39             mc_init.v2v_period,
40             mc_init.ctrl_loop_period);
```

```

41     fprintf(file , "running_time=%d\n" , mc_init.running_time);
42     fprintf(file , "debug_mode=%d\n" , mc_init.debug_mode);
43
44     fprintf(file , "left PID controller: kp=%f , ki=%f , kd=%f\n" ,
45             ctrl_inner_para.kp_left , ctrl_inner_para.ki_left ,
46             ctrl_inner_para.kd_left);
47     fprintf(file , "right PID controller: kp=%f , ki=%f , kd=%f\n" ,
48             ctrl_inner_para.kp_right , ctrl_inner_para.ki_right ,
49             ctrl_inner_para.kd_right);
50     fprintf(file , "deadzone: threshold=%f , saturation=%f\n" ,
51             ctrl_inner_para.deadzone_threshold ,
52             ctrl_inner_para.deadzone_saturation);
53
54     fprintf(file , "pwm_left_init=%d , pwm_right_init=%d\n" ,
55             ctrl_open_loop_state.pwm_left_dsr ,
56             ctrl_open_loop_state.pwm_right_dsr);
57     fprintf(file , "wl_dsr_init=%f , wr_dsr_init=%f\n" ,
58             ctrl_wl_wr_state.wl_dsr ,
59             ctrl_wl_wr_state.wr_dsr);
60     fprintf(file , "v_dsr_init=%f , omega_dsr_init=%f\n" ,
61             ctrl_v_omega_state.v_dsr ,
62             ctrl_v_omega_state.omega_dsr);
63
64     fflush(file);
65 }
66
67 int set_config(char *key , char *value) {
68
69     if (strcmp(key , "vehicle_id") == 0) {
70         mc_init.vehicle_id = atoi(value);
71     } else if (strcmp(key , "tty_file_name") == 0) {
72         int len = strlen(value);
73         value[len - 1] = 0;
74         strncpy(mc_init.tty_file_name , value ,
75                 sizeof(mc_init.tty_file_name));
76     } else if (strcmp(key , "tty_baud_rate") == 0) {
77         mc_init.tty_baud_rate = atoi(value);
78     } else if (strcmp(key , "log_file_name") == 0) {
79         int len = strlen(value);
80         value[len - 1] = 0;
81         strncpy(mc_init.log_file_name , value ,
82                 sizeof(mc_init.log_file_name));
83     } else if (strcmp(key , "meta_file_name") == 0) {
84         int len = strlen(value);
85         value[len - 1] = 0;
86         strncpy(mc_init.meta_file_name , value ,
87                 sizeof(mc_init.meta_file_name));
88     } else if (strcmp(key , "manager_ip") == 0) {
89         int len = strlen(value);
90         value[len - 1] = 0;
91         strncpy(mc_init.manager_ip , value ,
92                 sizeof(mc_init.manager_ip));
93     } else if (strcmp(key , "manager_port") == 0) {
94         mc_init.manager_port = atoi(value);
95     } else if (strcmp(key , "v2v_period") == 0) {
96         mc_init.v2v_period = atoi(value);
97     }
98
99     // camera configuration
100    else if (strcmp(key , "camera_inverse") == 0) {
101        cpara.inverse = atoi(value);
102    } else if (strcmp(key , "camera_threshold") == 0) {
103        cpara.th = atoi(value);
104    } else if (strcmp(key , "camera_roi_x") == 0) {
105        cpara.roi_x = atoi(value);

```

```

106     } else if (strcmp(key, "camera_roi_y") == 0) {
107         cpara.roi_y = atoi(value);
108     } else if (strcmp(key, "camera_roi_width") == 0) {
109         cpara.roi_width = atoi(value);
110     } else if (strcmp(key, "camera_roi_height") == 0) {
111         cpara.roi_height = atoi(value);
112     }
113
114
115
116     else if (strcmp(key, "running_time") == 0) {
117         mc_init.running_time = atoi(value);
118     } else if (strcmp(key, "ctrl_loop_period") == 0) {
119         mc_init.ctrl_loop_period = atoi(value);
120     } else if (strcmp(key, "controller_mode") == 0) {
121         vstate.ctrl_outer_loop_mode = atoi(value);
122         if (vstate.ctrl_outer_loop_mode == CTRL_OPENLOOP)
123             vstate.ctrl_inner_loop_mode = CTRL_OPENLOOP;
124         else
125             vstate.ctrl_inner_loop_mode = CTRL_WLWR;
126     } else if (strcmp(key, "debug_mode") == 0) {
127         mc_init.debug_mode = atoi(value);
128     }
129
130 // vehicle configuration
131     else if (strcmp(key, "wheel_radius") == 0) {
132         mc_init.wheel_radius = atof(value);
133     } else if (strcmp(key, "distance_of_wheels") == 0) {
134         mc_init.distance_of_wheels = atof(value);
135     }
136
137 // general controller parameter
138
139 // parameter of outer loop
140
141 // outer loop v_theta control
142     else if (strcmp(key, "outer_v_theta_kp_theta") == 0) {
143         ctrl_v_theta_para.kp_theta = atof(value);
144     } else if (strcmp(key, "outer_v_theta_ki_theta") == 0) {
145         ctrl_v_theta_para.ki_theta = atof(value);
146     } else if (strcmp(key, "outer_v_theta_kd_theta") == 0) {
147         ctrl_v_theta_para.kd_theta = atof(value);
148     }
149
150 // outer loop x_y control
151     else if (strcmp(key, "outer_x_y_kp_dist") == 0) {
152         ctrl_x_y_para.kp_dist = atof(value);
153     } else if (strcmp(key, "outer_x_y_ki_dist") == 0) {
154         ctrl_x_y_para.ki_dist = atof(value);
155     } else if (strcmp(key, "outer_x_y_kd_dist") == 0) {
156         ctrl_x_y_para.kd_dist = atof(value);
157     } else if (strcmp(key, "outer_x_y_kp_angle") == 0) {
158         ctrl_x_y_para.kp_angle = atof(value);
159     } else if (strcmp(key, "outer_x_y_ki_angle") == 0) {
160         ctrl_x_y_para.ki_angle = atof(value);
161     } else if (strcmp(key, "outer_x_y_kd_angle") == 0) {
162         ctrl_x_y_para.kd_angle = atof(value);
163     }
164
165 // outer loop delta_x_theta
166     else if (strcmp(key, "outer_delta_x_theta_kp_delta_x") == 0) {
167         ctrl_delta_x_theta_para.kp_delta_x = atof(value);
168     } else if (strcmp(key, "outer_delta_x_theta_ki_delta_x") == 0) {
169         ctrl_delta_x_theta_para.ki_delta_x = atof(value);
170     } else if (strcmp(key, "outer_delta_x_theta_kd_delta_x") == 0) {

```

```

171     ctrl_delta_x_theta_para.kd_delta_x = atof(value);
172 } else if (strcmp(key, "outer_delta_x_theta_kp_theta") == 0) {
173     ctrl_delta_x_theta_para.kp_theta = atof(value);
174 } else if (strcmp(key, "outer_delta_x_theta_ki_theta") == 0) {
175     ctrl_delta_x_theta_para.ki_theta = atof(value);
176 } else if (strcmp(key, "outer_delta_x_theta_kd_theta") == 0) {
177     ctrl_delta_x_theta_para.kd_theta = atof(value);
178 }
179
180 // outer loop line track
181 else if (strcmp(key, "outer_line_track_kp") == 0) {
182     ctrl_line_track_para.kp = atof(value);
183 } else if (strcmp(key, "outer_line_track_ki") == 0) {
184     ctrl_line_track_para.ki = atof(value);
185 } else if (strcmp(key, "outer_line_track_kd") == 0) {
186     ctrl_line_track_para.kd = atof(value);
187 }
188
189
190
191 // outer loop platooning
192 else if (strcmp(key, "outer_platooning_leader_id") == 0) {
193     ctrl_platooning_para.leader_id = atoi(value);
194 }
195
196
197 else if (strcmp(key, "outer_platooning_kp_delta_x") == 0) {
198     ctrl_platooning_para.kp_delta_x = atof(value);
199 } else if (strcmp(key, "outer_platooning_ki_delta_x") == 0) {
200     ctrl_platooning_para.ki_delta_x = atof(value);
201 } else if (strcmp(key, "outer_platooning_kd_delta_x") == 0) {
202     ctrl_platooning_para.kd_delta_x = atof(value);
203 }
204
205 else if (strcmp(key, "outer_platooning_kp_ffleader") == 0) {
206     ctrl_platooning_para.kp_ffleader = atof(value);
207 } else if (strcmp(key, "outer_platooning_ki_ffleader") == 0) {
208     ctrl_platooning_para.ki_ffleader = atof(value);
209 } else if (strcmp(key, "outer_platooning_kd_ffleader") == 0) {
210     ctrl_platooning_para.kd_ffleader = atof(value);
211 } else if (strcmp(key, "outer_platooning_ka_ffleader") == 0) {
212     ctrl_platooning_para.ka_ffleader = atof(value);
213 }
214
215 else if (strcmp(key, "outer_platooning_kp_ffpre") == 0) {
216     ctrl_platooning_para.kp_ffpre = atof(value);
217 } else if (strcmp(key, "outer_platooning_ki_ffpre") == 0) {
218     ctrl_platooning_para.ki_ffpre = atof(value);
219 } else if (strcmp(key, "outer_platooning_kd_ffpre") == 0) {
220     ctrl_platooning_para.kd_ffpre = atof(value);
221 } else if (strcmp(key, "outer_platooning_ka_ffpre") == 0) {
222     ctrl_platooning_para.ka_ffpre = atof(value);
223 }
224
225 else if (strcmp(key, "outer_platooning_kp_theta") == 0) {
226     ctrl_platooning_para.kp_theta = atof(value);
227 } else if (strcmp(key, "outer_platooning_ki_theta") == 0) {
228     ctrl_platooning_para.ki_theta = atof(value);
229 } else if (strcmp(key, "outer_platooning_kd_theta") == 0) {
230     ctrl_platooning_para.kd_theta = atof(value);
231 }
232
233
234
235

```

```

236     // parameter of inner loop
237
238     else if (strcmp(key, "prefilter_coefficient") == 0) {
239         ctrl_inner_para.prefilter_coefficient = atof(value);
240     } else if (strcmp(key, "roll_off_coefficient") == 0) {
241         ctrl_inner_para.roll_off_coefficient = atof(value);
242     } else if (strcmp(key, "inner_kp_left") == 0) {
243         ctrl_inner_para.kp_left = atof(value);
244     } else if (strcmp(key, "inner_ki_left") == 0) {
245         ctrl_inner_para.ki_left = atof(value);
246     } else if (strcmp(key, "inner_kd_left") == 0) {
247         ctrl_inner_para.kd_left = atof(value);
248     } else if (strcmp(key, "inner_kp_right") == 0) {
249         ctrl_inner_para.kp_right = atof(value);
250     } else if (strcmp(key, "inner_ki_right") == 0) {
251         ctrl_inner_para.ki_right = atof(value);
252     } else if (strcmp(key, "inner_kd_right") == 0) {
253         ctrl_inner_para.kd_right = atof(value);
254     } else if (strcmp(key, "deadzone_threshold") == 0) {
255         ctrl_inner_para.deadzone_threshold = atof(value);
256     } else if (strcmp(key, "deadzone_saturation") == 0) {
257         ctrl_inner_para.deadzone_saturation = atof(value);
258     }
259
260     // initial value
261
262     else if (strcmp(key, "pwm_left_init") == 0) {
263         ctrl_open_loop_state.pwm_left_dsr = atoi(value);
264     } else if (strcmp(key, "pwm_right_init") == 0) {
265         ctrl_open_loop_state.pwm_right_dsr = atoi(value);
266     } else if (strcmp(key, "wl_dsr_init") == 0) {
267         ctrl_wl_wr_state.wl_dsr = atof(value);
268     } else if (strcmp(key, "wr_dsr_init") == 0) {
269         ctrl_wl_wr_state.wr_dsr = atof(value);
270     } else if (strcmp(key, "v_dsr_init") == 0) {
271         ctrl_v_omega_state.v_dsr = atof(value);
272     } else if (strcmp(key, "omega_dsr_init") == 0) {
273         ctrl_v_omega_state.omega_dsr = atof(value);
274     } else if (strcmp(key, "outer_v_theta_v_dsr_init") == 0) {
275         ctrl_v_theta_state.v_dsr = atof(value);
276     } else if (strcmp(key, "outer_v_theta_theta_dsr_init") == 0) {
277         ctrl_v_theta_state.theta_dsr = atof(value);
278     } else if (strcmp(key, "outer_x_y_x_dsr_init") == 0) {
279         ctrl_x_y_state.x_dsr = atof(value);
280     } else if (strcmp(key, "outer_x_y_y_dsr_init") == 0) {
281         ctrl_x_y_state.y_dsr = atof(value);
282     } else if (strcmp(key, "outer_delta_x_theta_delta_x_dsr_init")
283     == 0) {
284         ctrl_delta_x_theta_state.delta_x_dsr = atof(value);
285     } else if (strcmp(key, "outer_delta_x_theta_theta_dsr_init")
286     == 0) {
287         ctrl_delta_x_theta_state.theta_dsr = atof(value);
288     } else if (strcmp(key, "outer_platooning_delta_x_dsr_init")
289     == 0) {
290         ctrl_platooning_state.delta_x_dsr = atof(value);
291     } else if (strcmp(key, "outer_platooning_theta_dsr_init")
292     == 0) {
293         ctrl_platooning_state.theta_dsr = atof(value);
294     } else if (strcmp(key, "outer_line_track_v_dsr_init")
295     == 0) {
296         ctrl_line_track_state.v_dsr = atof(value);
297     }
298
299     else {
300         return -1;

```

```

301     }
302     return 0;
303 }
304
305 int parse_conf_file(char *conf_fn) {
306
307     int ret = 0;
308     int i;
309     char *line = NULL;
310     int line_count = 1;
311     size_t n;
312
313     FILE *fd;
314
315     fd = fopen(conf_fn, "r");
316     if (!fd) {
317         printf("Unable to open configuration file: %s\n", conf_fn);
318         return -1;
319     }
320
321     while ((ret = getline(&line, &n, fd)) != -1) {
322
323         if (line[0] == '#' || line[0] == '\n') {
324             // The line is comment, do nothing.
325
326         } else {
327
328             char *ch = strchr(line, '=');
329             if (!ch) {
330                 printf("Bad config file at line %d: %s\n",
331                     line_count, line);
332                 return -1;
333                 break;
334             }
335
336             *ch = 0;
337             char *key = line;
338             char *value = ch + 1;
339
340             // printf("key = %s, value = %s\n", key, value);
341
342             if (set_config(key, value) < 0) {
343                 printf("Bad parameter at line %d: %s, %s\n",
344                     line_count, key,
345                     value);
346                 return -1;
347                 break;
348             }
349         }
350     }
351     line_count++;
352
353     free(line);
354     fclose(fd);
355     return 0;
356 }
357
358 // -----
359
360 // -----
361 volatile int command_should_stop = 0;
362
363 int dispatch_command(char *cmd, char *argv) {
364
365     int ret = -1;
366
367
368

```

```

369     if (strcmp(cmd, "open_loop") == 0) {
370         int v_left;
371         int v_right;
372         sscanf(argv, "%d %d", &v_left, &v_right);
373     //     printf(" open_loop command: v_left=%d, v_right=%d\n",
374     //            v_left, v_right);
375         ctrl_open_loop_set_pwm((int16_t) v_left,
376             (int16_t) v_right);
377
378     } else if (strcmp(cmd, "wl_wr") == 0) {
379         float wl_dsr;
380         float wr_dsr;
381         sscanf(argv, "%f %f", &wl_dsr, &wr_dsr);
382     //     printf(" close loop wl_wr command: wl_dsr=%f, wr_dsr=%f\n",
383     //            wl_dsr, wr_dsr);
384         ctrl_set_wl_wr(wl_dsr, wr_dsr);
385
386     } else if (strcmp(cmd, "v_omega") == 0) {
387         float v_dsr;
388         float omega_dsr;
389         sscanf(argv, "%f %f", &v_dsr, &omega_dsr);
390     //     printf(" close loop v_omega command: v_dsr=%f,
391 //omega_dsr=%f\n",
392     //            v_dsr, omega_dsr);
393         ctrl_set_v_omega(v_dsr, omega_dsr);
394
395     } else if (strcmp(cmd, "v_theta") == 0) {
396         float v_dsr;
397         float theta_dsr;
398         sscanf(argv, "%f %f", &v_dsr, &theta_dsr);
399         ctrl_set_v_theta(v_dsr, theta_dsr);
400
401     } else if (strcmp(cmd, "x_y") == 0) {
402         float x_dsr;
403         float y_dsr;
404         sscanf(argv, "%f %f", &x_dsr, &y_dsr);
405         ctrl_set_x_y(x_dsr, y_dsr);
406
407     } else if (strcmp(cmd, "delta_x_theta") == 0) {
408         float delta_x_dsr;
409         float theta_dsr;
410         sscanf(argv, "%f %f", &delta_x_dsr, &theta_dsr);
411         ctrl_set_delta_x_theta(delta_x_dsr, theta_dsr);
412
413     } else if (strcmp(cmd, "line_track") == 0) {
414         float v_dsr;
415         sscanf(argv, "%f", &v_dsr);
416         ctrl_set_line_track(v_dsr);
417
418     } else if (strcmp(cmd, "platooning") == 0) {
419         float delta_x_dsr;
420         float theta_dsr;
421         sscanf(argv, "%f %f", &delta_x_dsr, &theta_dsr);
422         ctrl_set_platooning(delta_x_dsr, theta_dsr);
423
424     } else if (strcmp(cmd, "setup") == 0) {
425
426         ctrl_setup();
427         printf(" setup command\n");
428
429     } else if (strcmp(cmd, "start") == 0) {
430
431         ctrl_start();
432         printf(" start command\n");
433
434     } else if (strcmp(cmd, "stop") == 0) {
435

```

```

436         ctrl_stop();
437         printf("stop command\n");
438     } else if (strcmp(cmd, "exit") == 0) {
439
440         command_should_stop = 1;
441         serial_should_stop = 1;
442         ctrl_loop_should_stop = 1;
443         v2v_sending_should_stop = 1;
444         v2v_receiving_should_stop = 1;
445         camera_should_stop = 1;
446
447     } else {
448
449         printf("Unknown command!!!\n");
450     }
451
452     return ret;
453 }
454
455 void *command_thread(void *para) {
456
457     int ret;
458     int i;
459     char *line = (char *)malloc(1024);
460     size_t n;
461
462     printf("<command> Command thread started.\n");
463
464     while (!command_should_stop) {
465
466         // read a line from keyboard
467         ret = getline(&line, &n, stdin);
468         if (ret < 0) {
469             printf("Unable to receive command! ret = %d(%s)\n",
470                   ret, strerror(errno));
471             continue;
472         }
473
474         // remove the '\n'
475         line[strlen(line) - 1] = 0;
476
477         char *cmd = line;
478
479         char *ch = strchr(line, ' ');
480         if (ch != NULL) {
481             *ch = 0;
482             ch++;
483         }
484
485         dispatch_command(cmd, ch);
486
487     }
488     free(line);
489
490     printf("<command> Command thread exiting...\n");
491
492     return NULL;
493 }
494
495 // -----
496
497 timer_t t_id;
498 int timer_v2v_count = 0;
499 int timer_ctrl_loop_count = 0;
500
501 void handler(int signo) {
502

```

```

504 // struct timespec now;
505 //
506 // clock_gettime(CLOCK_MONOTONIC, &now);
507 // printf("[%d] Diff time:%ld us\n", count++,
508 // timerdiff(&now, &prev));
509 // prev = now;
510
511 timer_v2v_count++;
512 timer_ctrl_loop_count++;
513
514 if (timer_v2v_count >= mc_init.v2v_period) {
515     // wake up v2v thread
516     sem_post(&v2v_send_sem);
517     timer_v2v_count = 0;
518 }
519 if (timer_ctrl_loop_count >= mc_init.ctrl_loop_period) {
520     // wake up control thread
521     sem_post(&ctrl_loop_sem);
522     timer_ctrl_loop_count = 0;
523 }
524 }
525
526 void timer_init() {
527     int i = 0;
528
529     // setup signal handler
530     struct sigaction sigact;
531     sigset(SIGALRM, &sigact);
532
533     sigemptyset(SIGALRM, &sigact.sa_handler);
534     sigaddset(SIGALRM, &sigact.sa_handler);
535
536     sigact.sa_flags = SA_RESTART;
537     sigact.sa_mask = set;
538     sigact.sa_handler = &handler;
539
540     // register signal handler
541     sigaction(SIGALRM, &sigact, NULL);
542
543     // setup timer interval
544     struct itimerspec tim_spec;
545     tim_spec.it_interval.tv_sec = 0;
546     tim_spec.it_interval.tv_nsec = 1000 * 1000;
547     tim_spec.it_value.tv_sec = 0;
548     tim_spec.it_value.tv_nsec = 1000 * 1000;
549
550     // Specifying sevp as NULL is equivalent to
551     // specifying a pointer to a
552     // sigevent structure in which sigev_notify is
553     // SIGEV_SIGNAL, sigev_signo
554     // is SIGALRM, and sigev_value.sival_int is the timer ID.
555     if (timer_create(CLOCK_MONOTONIC, NULL, &t_id))
556         perror("timer_create");
557
558     if (timer_settime(t_id, 0, &tim_spec, NULL))
559         perror("timer_settime");
560
561     sem_init(&v2v_send_sem, 0, 0);
562
563 }
564
565 void timer_exit() {
566     timer_delete(t_id);
567
568     sem_destroy(&v2v_send_sem);
569
570 }
571

```

```

572 }
573
574 int main(int argc, char **argv) {
575
576     char fn_buff[64];
577     char suffix[128];
578
579     char conf_fn[] = "init.conf";
580
581     if (argc == 1) {
582         time_t t = time(NULL);
583         struct tm tm = *localtime(&t);
584
585         snprintf(suffix, sizeof(suffix),
586                 "%04d.%02d.%02d.%02d.%02d",
587                         tm.tm_year + 1900, tm.tm_mon + 1,
588                         tm.tm_mday, tm.tm_hour, tm.tm_min,
589                         tm.tm_sec);
590
591     } else if (argc == 2) {
592         strcpy(suffix, argv[1], sizeof(suffix));
593     } else {
594         printf("Usage: pi_mc init.conf\n");
595     }
596
597     // open log file and meta file
598
599
600     snprintf(fn_buff, sizeof(fn_buff), "log.%s.txt", suffix);
601     log_file = fopen(fn_buff, "w+");
602
603     snprintf(fn_buff, sizeof(fn_buff), "meta.%s.txt", suffix);
604     meta_file = fopen(fn_buff, "w+");
605
606     // Read and parse config file
607     if (parse_conf_file(conf_fn) >= 0) {
608         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);
615
616     recent_log_file = fopen(mc_init.log_file_name, "w+");
617
618     printf("<main> log file and meta file successfully created.\n");
619
620     printf("<main> My VID is %d\n", mc_init.vehicle_id);
621
622     // initialize timer
623     timer_init();
624
625     // create threads
626
627     pthread_t command_thread_tid;
628
629     pthread_t h2l_thread_tid;
630     pthread_t ctrl_loop_thread_tid;
631
632     pthread_t v2v_sending_thread_tid;
633     pthread_t v2v_receiving_thread_tid;
634
635     pthread_t camera_thread_tid;
636
637     // keyboard command thread
638     pthread_create(&command_thread_tid, NULL, command_thread, NULL);

```

```

639     usleep(1000);
640
641     // h2l receiving thread (HLC <- LLC)
642     pthread_create(&h2l_thread_tid , NULL, h2l_receive_thread , NULL);
643
644     // Make sure serial port is correctly
645     //opened before ctrl_loop is started.
646     usleep(1000);
647
648     // h2l sending thread (HLC -> LLC)
649     pthread_create(&ctrl_loop_thread_tid , NULL,
650                   ctrl_loop_thread , NULL);
651
652     usleep(1000);
653
654     // v2v thread
655     pthread_create(&v2v_sending_thread_tid , NULL,
656                   v2v_sending_thread , NULL);
657     pthread_create(&v2v_receiving_thread_tid , NULL,
658                   v2v_receiving_thread , NULL);
659
660     pthread_create(&camera_thread_tid , NULL, camera_thread , NULL);
661
662     pthread_join(command_thread_tid , NULL);
663
664     pthread_join(h2l_thread_tid , NULL);
665     pthread_join(ctrl_loop_thread_tid , NULL);
666
667     pthread_join(v2v_sending_thread_tid , NULL);
668     pthread_join(v2v_receiving_thread_tid , NULL);
669
670     pthread_join(camera_thread_tid , NULL);
671
672     // exit
673
674     timer_exit();
675
676     fclose(meta_file);
677     fclose(recent_meta_file);
678     fclose(log_file);
679     fclose(recent_log_file);
680
681     printf("Exit.\n");
682
683     return EXIT_SUCCESS;
684 }

```

```

1 #include <stdio.h>
2
3 #include <unistd.h>
4 #include <fcntl.h>
5 #include <termios.h>
6
7 #include <sys/types.h>
8
9 // see http://www.cmrr.umn.edu/~strupp/serial.html
10
11 // takes the string name of the serial port
12 // (e.g. "/dev/tty.usbserial","COM1")
13 // and a baud rate (bps) and connects to
14 // that port at that speed and 8N1.
15 // opens the port in fully raw mode so you can send binary data.
16 // returns valid fd, or -1 on error
17 int serial_init(const char* serialport , int baud) {

```

```

18     struct termios toptions;
19     int fd;
20
21 //  fd = open(serialport, O_RDWR | O_NOCTTY | O_NDELAY);
22 //  fd = open(serialport, O_RDWR | O_NOCTTY);
23 //fd = open(serialport, O_RDWR | O_NONBLOCK);
24 //fd = open(serialport, O_RDWR);
25
26 if (fd == -1) {
27     perror("serialport_init: Unable to open port ");
28     return -1;
29 }
30
31 //int iflags = TIOCM_DTR;
32 //ioctl(fd, TIOCMBIS, &iflags);      // turn on DTR
33 //ioctl(fd, TIOCMBIC, &iflags);      // turn off DTR
34
35 if (tcgetattr(fd, &toptions) < 0) {
36     perror("serialport_init: Couldn't get term attributes");
37     return -1;
38 }
39 speed_t brate = baud; // let you override switch below if needed
40 switch (baud) {
41 case 4800:
42     brate = B4800;
43     break;
44 case 9600:
45     brate = B9600;
46     break;
47 #ifdef B14400
48     case 14400: brate=B14400; break;
49 #endif
50     case 19200:
51         brate = B19200;
52         break;
53 #ifdef B28800
54     case 28800: brate=B28800; break;
55 #endif
56     case 38400:
57         brate = B38400;
58         break;
59     case 57600:
60         brate = B57600;
61         break;
62     case 115200:
63         brate = B115200;
64         break;
65     case 230400:
66         brate = B230400;
67         break;
68     case 460800:
69         brate = B460800;
70         break;
71     case 576000:
72         brate = B576000;
73         break;
74     case 921600:
75         brate = B921600;
76         break;
77     case 1152000:
78         brate = B1152000;
79         break;
80     case 1500000:
81         brate = B1500000;
82         break;
83     case 2000000:
84         brate = B2000000;
85         break;
86     case 2500000:
87         brate = B2500000;

```

```

88         break;
89     case 3000000:
90         brate = B3000000;
91         break;
92     case 3500000:
93         brate = B3500000;
94         break;
95     case 4000000:
96         brate = B4000000;
97         break;
98     }
99     cfsetispeed(&toptions, brate);
100    cfsetospeed(&toptions, brate);
101
102    // 8N1
103    toptions.c_cflag &= ~PARENB;
104    toptions.c_cflag &= ~CSTOPB;
105    toptions.c_cflag &= ~CSIZE;
106    toptions.c_cflag |= CS8;
107    // no flow control
108    toptions.c_cflag &= ~CRTSCTS;
109
110    // disable hang-up-on-close to avoid reset
111    // toptions.c_cflag &= ~HUPCL;
112
113    // turn on READ & ignore ctrl lines
114    toptions.c_cflag |= CREAD | CLOCAL;
115    // turn off s/w flow ctrl
116    toptions.c_iflag &= ~(IXON | IXOFF | IXANY);
117
118    // make raw
119    toptions.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG );
120    toptions.c_oflag &= ~OPOST;
121
122    // see: http://unixwiz.net/techtips/termios-vmin-vtime.html
123    toptions.c_cc [VMIN] = 0;
124    toptions.c_cc [VTIME] = 0;
125    // toptions.c_cc [VTIME] = 20;
126
127    tcsetattr(fd, TCSANOW, &toptions);
128    if (tcsetattr(fd, TCSAFLUSH, &toptions) < 0) {
129        perror("init_serialport: Couldn't set term attributes");
130        return -1;
131    }
132
133    return fd;
134 }
135
136 //
137 int serial_close(int fd) {
138     return close(fd);
139 }
140
141 int serial_recv_byte(int fd, char *buff) {
142
143     return read(fd, buff, 1);
144 }
145
146 // CAUTION: blocking / busy waiting send
147 int serial_send_n_bytes(int fd, const char* buff, size_t n) {
148
149     int ret = 0;
150     int count = 0;
151
152     while(count < n) {
153         ret = write(fd, buff, n);
154         if (ret < 0) {
155             return ret;

```

```

156         }
157         count += ret;
158     }
159     return 0;
160 }
161 // CAUTION: blocking / busy waiting receive
162 int serial_recv_n_bytes(int fd, char *buff, size_t n) {
163
164     int ret = 0;
165     int count = 0;
166
167     while (count < n) {
168         ret = read(fd, buff + count, n - count);
169         if (ret < 0) {
170             return ret;
171         }
172         count += ret;
173     }
174
175     return count;
176 }
177
178 /**
180 int serial_flush(int fd) {
181     return tcflush(fd, TCIOFLUSH);
182 }

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <errno.h>
5
6 #include <unistd.h>
7 #include <time.h>
8 #include <signal.h>
9 #include <sys/types.h>
10
11 #include <pthread.h>
12 #include <semaphore.h>
13
14 #include <sys/socket.h>
15 #include <netinet/in.h>
16 #include <arpa/inet.h>
17
18 #include "v2v.h"
19 #include "ctrl_loop.h"
20 #include "pi_mc.h"
21
22 sem_t v2v_send_sem;
23
24 volatile int v2v_sending_should_stop = 0;
25 volatile int v2v_receiving_should_stop = 0;
26
27 const int nr_dst = 10;
28 struct sockaddr_in servaddr[10];
29
30 struct v2v_msg recent_msgs[10];
31
32 struct v2v_msg *get_recent_msg(int vid) {
33
34     if (vid < 1 || vid > 9)
35         return NULL;
36     else
37         return &recent_msgs[vid];
38 }
39

```

```

40 void print_v2v_msg_send(FILE *fd, struct v2v_msg *msg,
41 int dst_id, int flag) {
42
43
44     fprintf(fd, "[%08d]<%1d -> %1d><%1d>\n"
45             " (%.4f, %.4f, %.4f, %.4f, %.4f, ctrl_flag %d)\n",
46             msg->ts, msg->vid, dst_id, flag,
47             msg->theta, msg->accx, msg->v, msg->omega,
48             msg->v_dsr, msg->ctrl_flag);
49     fflush(fd);
50 }
51
52 void print_v2v_msg_recv(FILE *fd, struct v2v_msg *msg, int my_id) {
53
54     struct timespec now;
55     clock_gettime(CLOCK_MONOTONIC, &now);
56
57     uint32_t ts = now.tv_sec * 1000 + now.tv_nsec / 1000000;
58
59     fprintf(fd, "[%08d][%08d]<%1d -> %1d>\n"
60             " (%.4f, %.4f, v: %.4f, %.4f, %.4f, ctrl_flag %d\n",
61             ts, msg->ts, msg->vid, my_id,
62             msg->theta, msg->accx, msg->v, msg->omega,
63             msg->v_dsr, msg->ctrl_flag);
64     fflush(fd);
65 }
66
67 void *v2v_sending_thread(void *para) {
68
69     int i, ret;
70     struct timespec now;
71
72     char addbuff[64];
73
74     FILE *v2v_send_fd = fopen("v2v-send.txt", "w+");
75
76     int sendsockfd = socket(AF_INET, SOCK_DGRAM, 0);
77
78     for (i = 1; i < nr_dst; i++) {
79         snprintf(addbuff, sizeof(addbuff), "192.168.0.10%d", i);
80         memset(&servaddr[i], 0, sizeof(servaddr[i]));
81         servaddr[i].sin_family = AF_INET;
82         servaddr[i].sin_addr.s_addr = inet_addr(addbuff);
83         servaddr[i].sin_port = htons(32000);
84     }
85
86     memset(&servaddr[0], 0, sizeof(servaddr[0]));
87     servaddr[0].sin_family = AF_INET;
88     servaddr[0].sin_addr.s_addr = inet_addr(mc_init.manager_ip);
89     servaddr[0].sin_port = htons(32000);
90
91     printf("<v2v> V2V sending thread started.\n");
92
93     while (!v2v_sending_should_stop) {
94
95         sem_wait(&v2v_send_sem);
96
97
98         clock_gettime(CLOCK_MONOTONIC, &now);
99         uint32_t ts = (uint32_t)(now.tv_sec * 1000UL
100                     + now.tv_nsec / 1000000);
101
102         struct v2v_msg vmsg;
103
104         vmsg.ts = ts;
105         vmsg.vid = (uint32_t)mc_init.vehicle_id;

```

```

107         vmsg.theta = vstate imu_theta;
108         vmsg.accx = vstate imu_accx;
109         vmsg.v = vstate v;
110         vmsg.omega = vstate omega;
111         vmsg.v_dsr = vstate v_dsr;
112         vmsg.ctrl_flag = vstate ctrl_flag;
113
114     if (mc_init.vehicle_id == 1) {
115
116         // Platooning header broadcast its states
117         for (i = 2; i < nr_dst; i++) {
118             ret = sendto(sendsockfd, &vmsg, sizeof(vmsg),
119                         MSG_DONTWAIT,
120                         (struct sockaddr *) &servaddr[i],
121                         sizeof(servaddr[i]));
122             print_v2v_msg_send(v2v_send_fd, &vmsg, i, ret);
123         }
124     } else {
125         // Features not enable yet
126         // Each vehicle sends its state to immediate successor,
127         // except for the last one.
128         if (mc_init.vehicle_id != nr_dst - 1) {
129             int dst_id = mc_init.vehicle_id + 1;
130             ret = sendto(sendsockfd, &vmsg, sizeof(vmsg),
131                         MSG_DONTWAIT,
132                         (struct sockaddr *) &servaddr[dst_id],
133                         sizeof(servaddr[dst_id]));
134             print_v2v_msg_send(v2v_send_fd, &vmsg, dst_id, ret);
135         }
136     };
137
138 }
139
140
141 }
142
143 fclose(v2v_send_fd);
144
145 printf("<v2v> V2V sending thread exiting...\n");
146
147 return NULL;
148 }
149
150 void *v2v_receiving_thread(void *para) {
151
152     int ret;
153     int sockfd;
154     struct sockaddr_in servaddr;
155     struct sockaddr_in cliaddr;
156     socklen_t len;
157     char recv_buff[1024];
158
159     FILE *v2v_recv = fopen("v2v-recv.txt", "w+");
160
161     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
162
163     memset(&servaddr, 0, sizeof(servaddr));
164     servaddr.sin_family = AF_INET;
165     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
166     servaddr.sin_port = htons(32000);
167     bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
168
169     // receiving function timeout every 1 second,
170     // so as to check v2v_receiving_should_stop.
171     struct timeval tv;
172     tv.tv_sec = 1;
173     tv.tv_usec = 0;

```

```

174     if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO,
175         &tv, sizeof(tv)) < 0) {
176         perror("Error in setting receiving timeout");
177     }
178     printf("<v2v> V2V receiving thread started.\n");
179
180     while (!v2v_receiving_should_stop) {
181
182         ret = recvfrom(sockfd, recv_buff, sizeof(recv_buff), 0,
183                         (struct sockaddr *) &cliaddr, &len);
184         if (ret > 0) {
185             struct v2v_msg *msg = (struct v2v_msg *) recv_buff;
186
187             print_v2v_msg_recv(v2v_recv, msg, mc_init.vehicle_id);
188
189             // update received state
190             uint32_t ts = msg->ts;
191             uint32_t id = msg->vid;
192             if (id >= 1 && id <= 9) {
193                 if (ts > recent_msgs[id].ts)
194                     recent_msgs[id] = *msg;
195             }
196         }
197     }
198
199     fclose(v2v_recv);
200
201
202     printf("<v2v> V2V receiving thread exiting ... \n");
203
204     return NULL;
205 }

```

APPENDIX B
MATLAB CODE

```

1 clear all;
2 close all;
3 clc
4
5 %% Load Model
6 P2wrwl = load('ETT_TITO_Model_2nd_order.mat');
7 P4wrwl = load('ETT_TITO_Model_4th_order.mat');
8 P2vw = load('ETT_Pvw_Model_2nd_order.mat');
9 P4vw = load('ETT_Pvw_Model_4th_order.mat');
10
11 %% Robot Singular Values (Voltages to Wheel Speeds) -
12 %% Including Low Frequency Approximation
13
14 fig34 = figure(34);
15 sigma(P4wrwl.TITO_SS,P2wrwl.TITO_SS)
16 hold on;grid on;
17 title('Singular Values for TITO Model','FontSize', 24);
18 legend('Without approximation','Low frequency approximation')
19 h_line = findobj(fig34, 'type', 'line');
20 set(h_line, 'LineWidth', 3);
21 h_axes = findobj(fig34, 'type', 'axes');
22 set(h_axes, 'linewidth', 2);
23 set(h_axes, 'FontSize', 15);
24 xlabel('Frequency','FontSize', 24)
25 ylabel('Singluar Values', 'FontSize', 24);
26 %
27 % print(fig34,'Singular Values for TITO Model','-depsc','-tiff ')
28 % print(fig34,'Singular Values for TITO Model','-dpng','-r0')
29
30
31 %% Robot Frequency Response (Voltages to Wheel Speeds) -
32 %% Including Low Frequency Approximation
33 fig35 = figure(35);
34 bode(P4wrwl.TITO_SS,P2wrwl.TITO_SS)
35 hold on;grid on;
36 title('Frequency Response: Voltage to [\omega_R, \omega_L]',...
37 'FontSize',24);
38 legend('Without approximation','Low frequency approximation')
39 h_line = findobj(gcf, 'type', 'line');
40 set(h_line, 'LineWidth', 3);
41 h_axes = findobj(gcf, 'type', 'axes');
42 set(h_axes, 'linewidth', 2);
43 set(h_axes, 'FontSize', 15);
44 xlabel('Frequency','FontSize', 24)
45 ylabel(' Phase', 'FontSize', 24);
46
47
48
49
50 %% SVD analysis
51 % Figure 3.6: Robot Plant Singular Values (Voltages to v and w) -
52 % Including Low Frequency Approximation
53 fig36 = figure(36);
54 sigma(P4vw.TITO_SS,P2vw.TITO_SS)
55 hold on;grid on;
56 title('Singular Values for Plant','FontSize', 24);
57 legend('Without approximation','Low frequency approximation')
58 h_line = findobj(gcf, 'type', 'line');
59 set(h_line, 'LineWidth', 3);
60 h_axes = findobj(gcf, 'type', 'axes');
61 set(h_axes, 'linewidth', 2);
62 set(h_axes, 'FontSize', 15);
63 xlabel('Frequency','FontSize', 24)
64 ylabel('Singluar Values', 'FontSize', 24);
65 %% Figure 3.7: Robot Plant Frequency Response (Voltages to [v;w]) -
66 %% Including Low Frequency Approximation

```

```

67 fig37 = figure(37);
68
69 opts = bodeoptions;
70 % opts.InputLabels.FontSize = 20;
71 % opts.OutputLabels.FontSize = 20;
72
73
74 bode(P4vw.TITO_SS,P2vw.TITO_SS)
75 hold on;grid on;
76 title('Frequency Response: Voltage to [v, \omega]', 'FontSize', 24);
77 legend('Without approximation', 'Low frequency approximation')
78 h_line = findobj(gcf, 'type', 'line');
79 set(h_line, 'LineWidth', 3);
80 h_axes = findobj(gcf, 'type', 'axes');
81 set(h_axes, 'LineWidth', 2);
82 set(h_axes, 'FontSize', 15);
83 xlabel('Frequency', 'FontSize', 24)
84 ylabel('Phase', 'FontSize', 24);
85
86 %% Singular Value of M
87 M = P4wrwl.ETT_M
88 M_inv = inv(M)
89 svd(M)
90 svd(M_inv)
91
92 norm(M, inf)
93
94 norm(M_inv, inf)
95
96
97 %% FIGURE in Chapter 4
98 % Robot Frequency Response (Voltages to Wheel Speeds) -
99 % Including Low Frequency Approximation
100 fig45 = figure(45);
101 bode(P4wrwl.TITO_SS,P2wrwl.TITO_SS)
102 hold on;grid on;
103 title('Frequency Response: Voltage to [\omega_R, \omega_L]');
104 legend('Without approximation', 'Low frequency approximation')
105 %% Robot Step Response (Voltages to Wheel Speeds)
106 fig46 = figure(46);
107 step(P4wrwl.TITO_SS,P2wrwl.TITO_SS)
108 hold on;grid on;
109 title('Step Response: Voltage to [\omega_R, \omega_L]');
110 legend('Without approximation', 'Low frequency approximation')
111
112 %% Auto Print Figure after enlarge all of them
113 % print(fig34,'Singular Values for TITO Model','-depsc','-tiff')
114 % print(fig34,'Singular Values for TITO Model','-dpng','-r0')
115
116 % print(fig35,'Singular Values for TITO Model','-depsc','-tiff')
117 % print(fig35,'Singular Values for TITO Model','-dpng','-r0')

```

```

1 clear all
2 clc
3 syms m beta r Iz Kg Kt Kb dw Ra La
4 a11 = -2*beta*Kg^2/(m*r^2);
5 a12=0;
6 a13 = Kg*Kt/(m*r);
7 a14 = a13;
8
9 a21=0;
10 a22 = -beta*Kg^2*dw^2/(2*Iz*r^2);
11 a23 = Kg*Kt*dw/(2*Iz*r);
12 a24 = a23;
13

```

```

14 a31=-Kb*Kg/(La*r);
15 a32=-Kb*Kg*dw/(2*La*r );
16 a33=Ra/La;
17 a34=0;
18
19 a41=a31 ;
20 a42=-a32 ;
21 a43=0;
22 a44=a33 ;
23
24 M=[r/2 r/2;r/dw -r/dw];
25 A=[ a11 a12 a13 a14;
26 a21 a22 a23 a24;
27 a31 a32 a33 a34;
28 a41 a42 a43 a44];
29 B=zeros(2,2);eye(2)./La];
30 C=[inv(M) zeros(2,2)];
31 D=zeros(2,2);
32 % Still Looks very Ugly Use Maple To Get it
33 % syms s
34 % I4=eye(4);
35 % P=C/(s*I4-A)*B+D
36 % pretty(simplify(P))
37
38
39 %% Vehicle Parameters for Our Experimental Enhanced Thunder Tumbler
40 close all
41 ETT_m=0.89 ;
42 ETT_beta=7.04e-7;
43 ETT_r=0.05 ;
44 ETT_Iz=0.0051;%1/12*ETT_m*(0.19^2+0.18^2);% Iz=m*1/12*(L^2+W^2)
45 % Vicent 0.0051; LIN 0.0013
46 ETT_Kg=18.48 ;
47 ETT_Kt=0.0032 ;
48 ETT_Kb=0.0032 ;
49 ETT_dw=0.14 ;
50 ETT_Ra=0.79 ;
51 ETT_La=265e-6;
52
53 % beta variation
54 % variation_vec = [ETT_beta/4 ETT_beta/2 ETT_beta ETT_beta*2];
55 % titlestr_bodemag
56 %='Bode magnitude plots of the plant for variations of damping costant';
57 % titlestr_step
58 %='Step response of the plant for variations of damping costant';
59 % legendstr_cell
60 %={'\beta = 0.0176e-5 (Nms)', '\beta = 0.0352e-5 (Nms)', ...
61 % '\beta = 0.0704e-5 (Nms)',
62 % '\beta = 0.1408e-5 (Nms)'};
63
64
65 % moment of inertia Iz varation
66 % variation_vec = [ETT_Iz/4 ETT_Iz/2 ETT_Iz ETT_Iz*2];
67 % titlestr_bodemag
68 %='Bode magnitude plots of the plant
69 %for variations of moment of inertia';
70 % titlestr_step
71 %='Step response of the plant for variations of moment of inertia';
72 % legendstr_cell = {'I_z = 0.0013 (Kg*m^2)', 'I_z = 0.0026 (Kg*m^2)', ...
73 % 'I_z = 0.0051 (Kg*m^2)', 'I_z = 0.0102 (Kg*m^2)'};
74
75
76 % moment of inertia Ra varation
77 % variation_vec = [ETT_Ra/2 ETT_Ra ETT_Ra*2 ETT_Ra*3];
78 % titlestr_bodemag
79 %='Bode magnitude plots of the plant
80 % for variations of armature resistance';

```

```

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
87 % moment of inertia Kt varation
88 % variation_vec = [ETT_Kt/2 ETT_Kt/1.5 ETT_Kt ETT_Kt*1.5];
89 % titlestr_bodemag
90 %<= 'Bode magnitude plots of the plant for variations of torque constant';
91 % titlestr_step
92 %<= 'Step response of the plant for variations of torque constant';
93 % legendstr_cell = {'K_t = 0.0016 (NmA)', 'K_t = 0.0021 (NmA)', ...
94 %                   'K_t = 0.0032 (NmA)', 'K_t = 0.0048 (NmA)'};
95
96
97
98
99 % moment of inertia Kb varation
100 % variation_vec = [ETT_Kb/2 ETT_Kb/1.5 ETT_Kb ETT_Kb*1.5];
101 % titlestr_bodemag
102 %<= 'Bode magnitude plots of the plant for
103 % variations of back emf constant';
104 % titlestr_step
105 %<= 'Step response of the plant for variations of back emf constant';
106 % legendstr_cell
107 %<= {'K_b = 0.0016 (V/(rad/sec))', 'K_b = 0.0021 (V/(rad/sec))', ...
108 %                   'K_b = 0.0032 (V/(rad/sec))', 'K_b = 0.0048 (V/(rad/sec))'};
109
110
111 % Mass m varation
112
113 variation_vec = [ETT_m*0.6 ETT_m*0.8 ETT_m*1 ETT_m*1.2];
114 titlestr_bodemag
115 = 'Bode magnitude plots of the plant for variations of mass';
116 titlestr_step
117 = 'Step response of the plant for variations of mass';
118 legendstr_cell = {'m = 0.5340 (kg)', 'm = 0.7120 (kg)', ...
119 %                   'm = 0.8900 (kg)', 'm = 1.0680 (kg)'};
120
121
122 Plant_cell= cell(length(variation_vec),1);
123
124
125
126
127 for ii= 1:1:4
128 %     ETT_beta = variation_vec(ii);
129 %     ETT_Iz = variation_vec(ii);
130 %     ETT_Ra = variation_vec(ii);
131 %     ETT_Kt = variation_vec(ii);
132 %     ETT_Kb = variation_vec(ii);
133 %     ETT_m = variation_vec(ii);
134
135 ETT_M = subs(M,{r,dw},{ETT_r,ETT(dw)});
136 ETT_A=subs(A,{m,beta r Iz Kg Kt Kb dw Ra La },...
137 {ETT.m,ETT_beta,ETT.r,ETT.Iz, ...
138 ETT.Kg,ETT.Kt,ETT.Kb,ETT.dw,ETT.Ra,ETT.La});
139
140 % Eigenvalues For the Characteristic Equations
141 ETT_eig=vpa(eig(ETT_A),3)
142
143
144 ETT_M = double(subs(M,{r dw},{ETT_r,ETT(dw)}));
145 ETT_Minv = inv(ETT_M);
146
147 ETT_A=double(ETT_A);

```

```

148 ETT_B=double( subs(B,La,ETT_La));
149 ETT_C=double( subs(C,{ r dw},{ ETT_r,ETT_dw})); 
150 ETT_D=D;
151 TITO_SS=ss (ETT_A,ETT_B,ETT_C,ETT_D);%
152
153
154 Plant_cell{ ii } = TITO_SS;
155
156 fig10 = figure (10);
157 bodemag(TITO_SS);
158 hold on; grid on;
159 title(titlestr_bodemag , 'FontSize' , 24)
160 xlabel('Frequency' , 'FontSize' , 24)
161 ylabel('Magnitude' , 'FontSize' , 24);
162 legend(legendstr_cell);
163
164 h_line = findobj(gcf , 'type' , 'line');
165 set(h_line , 'LineWidth' , 3);
166 h_axes = findobj(gcf , 'type' , 'axes');
167 set(h_axes , 'linewidth' , 2);
168 set(h_axes , 'FontSize' , 15);
169
170
171 fig20 = figure (20);
172 step(TITO_SS);
173 hold on; grid on;
174 title(titlestr_step , 'FontSize' , 24)
175 xlabel('Time' , 'FontSize' , 24)
176 ylabel('Angular Velocity (rad/sec)' , 'FontSize' , 24);
177 legend(legendstr_cell);
178 hold on;grid on;
179 h_line = findobj(gcf , 'type' , 'line');
180 set(h_line , 'LineWidth' , 3);
181 h_axes = findobj(gcf , 'type' , 'axes');
182 set(h_axes , 'linewidth' , 2);
183 set(h_axes , 'FontSize' , 15);
184
185
186
187 end
188
189
190 %
191 I4=eye (4);
192 P_dc=ETT.C/(I4-ETT.A)*ETT.B+ETT.D;
193 P_dc=vpa(P_dc,3)
194 %
195 % % % P11=vpa(P(1,1),3)
196
197 %
198 TF=ss2tfm (ETT_A,ETT_B,ETT_C,ETT_D);
199 % TF=zpk(TF)
200 TF_zpk=zpk(TF)
201
202 %% Frequency and Step Response of couple model
203
204 %
205 % close all
206 % fig10 = figure (10);
207 % bodemag(TITO_SS);
208 % hold on; grid on;
209 % title('Frequency response of coupled model')
210 %
211 % fig20 = figure (20);
212 % step(TITO_SS);
213 % hold on; grid on;

```

```

214 % title('Output response of coupled model')
215 %
216 %
217 % Frequency and Step Response of decoupled model
218 % first get decoupled model by removing off-diagonal element in TFM
219
220 % TF_decoupled(1,1)= TF(1,1);
221 % TF_decoupled(2,2)= TF(2,2);

1 % Inner loop trade study on TITO model
2
3 clear all
4 close all
5 clc
6
7 %% Load Model
8 P2wrwl = load('ETT_TITO_Model_2nd_order.mat');
9 P4wrwl = load('ETT_TITO_Model_4th_order.mat');
10 P2vw = load('ETT_Pvw_Model_2nd_order.mat');
11 P4vw = load('ETT_Pvw_Model_4th_order.mat');
12
13 P = P2wrwl.TITO_SS;
14
15 dw = 0.14;
16 r = 0.05;
17
18 M = [r/2      r/2
19          r/dw   -r/dw];
20 Minv = inv(M);
21
22
23 Ap = P.a; Bp = P.b; Cp = P.c; Dp = P.d;
24
25 s = tf('s');
26
27 % Open loop singular values, sensitivity, comp sensitivity for 4 diff
28 % values of g
29 w = logspace(-3,3,100);
30
31 % z = 1.26;
32 % g = [0.07, 0.1, 0.15];
33 z = 2;
34 g = [1 3 5 7].*0.1;
35 %%
36 for ii=1:length(g)
37 %% for ii=1:1:1
38 K = [ g(ii)*((s + z)/s)*(40/(s+40))    0
39                  0           g(ii)*((s + z)/s)*(40/(s+40))];
40
41 K = ss(K);
42
43 W = [z/(s+z)    0
44          0        z/(s+z) ];
45
46 %%
47 figure(10)
48 sigma(K,w);
49 %sv = 20*log10(sv);
50 %semilogx(w, sv, str(i,:))
51 title('Controller Singular Values ', 'FontSize', 24)
52 xlabel('Frequency', 'FontSize', 24)
53 ylabel('Singular Values', 'FontSize', 24);
54 legend('g = 0.10 , z = 2', 'g = 0.30 , z = 2', ...
55       'g = 0.50 , z = 2', 'g = 0.70 , z = 2')
56 grid on

```

```

57      hold on
58
59
60      h_line = findobj(gcf, 'type', 'line');
61      set(h_line, 'LineWidth', 3);
62      h_axes = findobj(gcf, 'type', 'axes');
63      set(h_axes, 'linewidth', 2);
64      set(h_axes, 'FontSize', 15);
65
66 %%%
67      L = P*K;
68
69      figure(11)
70      sigma(L,w);
71      %sv = 20*log10(sv);
72      %semilogx(w, sv, str(i,:))
73      title('Open Loop Singular Values ', 'FontSize', 24)
74      xlabel('Frequency', 'FontSize', 24)
75      ylabel('Singular Values', 'FontSize', 24);
76      legend('g = 0.10 , z = 2', 'g = 0.30 , z = 2',...
77             'g = 0.50 , z = 2', 'g = 0.70 , z = 2')
78      grid on
79      hold on
80
81
82      h_line = findobj(gcf, 'type', 'line');
83      set(h_line, 'LineWidth', 3);
84      h_axes = findobj(gcf, 'type', 'axes');
85      set(h_axes, 'linewidth', 2);
86      set(h_axes, 'FontSize', 15);
87
88 %%%
89      % Sensitivity
90      asen = L.a - L.b*L.c;
91      bsen = L.b;
92      csen = -L.c;
93      [row, col] = size(csen);
94      [row1, col1] = size(bsen);
95      dsen = eye(row, col1);
96
97      figure(12)
98      sigma(ss(asen, bsen, csen, dsen), w);
99      %sv = 20*log10(sv);
100     %semilogx(w, sv, str(i,:))
101     title('Sensitivity ', 'FontSize', 24)
102     xlabel('Frequency', 'FontSize', 24)
103     ylabel('Singular Values', 'FontSize', 24);
104     legend('g = 0.10 , z = 2', 'g = 0.30 , z = 2',...
105           'g = 0.50 , z = 2', 'g = 0.70 , z = 2')
106
107     hold on
108     grid on
109
110     h_line = findobj(gcf, 'type', 'line');
111     set(h_line, 'LineWidth', 3);
112     h_axes = findobj(gcf, 'type', 'axes');
113     set(h_axes, 'linewidth', 2);
114     set(h_axes, 'FontSize', 15);
115
116 %%%
117
118      %Complementary sensitivity
119      acl = L.a - L.b*L.c;
120      bcl = L.b;
121      ccl = L.c;
122      dcl = L.d;
123      cls = ss(acl, bcl, ccl, dcl);

```

```

124
125 figure(13)
126 sigma(ss(acl,bcl,ccl,dcl),w);
127 %sv = 20*log10(sv);
128 %semilogx(w, sv, str(i,:))
129 title('Complementary Sensitivity ', 'FontSize', 24)
130 xlabel('Frequency ', 'FontSize', 24)
131 ylabel('Singular Values ', 'FontSize', 24);
132
133 grid on
134 hold on
135 legend('g = 0.10 , z = 2 ', 'g = 0.30 , z = 2 ', ...
136 'g = 0.50 , z = 2 ', 'g = 0.70 , z = 2 ');
137
138 h_line = findobj(gcf, 'type', 'line');
139 set(h_line, 'LineWidth', 3);
140 h_axes = findobj(gcf, 'type', 'axes');
141 set(h_axes, 'linewidth', 2);
142 set(h_axes, 'FontSize', 15);
143
144 %%
145 % Step response unfiltered
146 figure(14)
147 step(cls)
148
149
150 title('Step Response for T_{ry} (Unfiltered)', 'FontSize', 24)
151 xlabel('Time ', 'FontSize', 24)
152 ylabel('Angular Velocity (rad/s) ', 'FontSize', 24)
153 grid on
154 hold on
155
156 legend('g = 0.10 , z = 2 ', 'g = 0.30 , z = 2 ', ...
157 'g = 0.50 , z = 2 ', 'g = 0.70 , z = 2 ')
158
159 h_line = findobj(gcf, 'type', 'line');
160 set(h_line, 'LineWidth', 3);
161 h_axes = findobj(gcf, 'type', 'axes');
162 set(h_axes, 'linewidth', 2);
163 set(h_axes, 'FontSize', 15);
164
165 %%
166 % Reference to control Tru no filter
167 % states [xp xk]^T
168 Aru = [Ap-Bp*K.d*Cp Bp*K.c
169 -K.b*Cp K.a];
170
171 Bru = [Bp*K.d
172 K.b];
173 Cru = [-K.d*Cp K.c];
174 Dru = K.d;
175 Tru = ss(Aru, Bru, Cru, Dru);
176
177 figure(15)
178 sigma(ss(Aru,Bru,Cru, Dru),w);
179 %sv = 20*log10(sv);
180 %semilogx(w, sv, str(i,:))
181 title(' T_{ru} (Unfiltered) ', 'FontSize', 24)
182 xlabel('Frequency ', 'FontSize', 24)
183 ylabel('Singular Values ', 'FontSize', 24);
184
185 grid on
186 hold on
187 legend('g = 0.10 , z = 2 ', 'g = 0.30 , z = 2 ', ...
188 'g = 0.50 , z = 2 ', 'g = 0.70 , z = 2 ')
189

```

```

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 % Control response to step command
199 figure(16)
200 step(Tru)
201
202 grid on
203 hold on
204 title('Control Response for step command (Unfiltered)', ...
205 'FontSize', 24)
206 xlabel('Time', 'FontSize', 24)
207 ylabel('Voltage (V)', 'FontSize', 24)
208 legend('g = 0.10, z = 2', 'g = 0.30, z = 2', ...
209 'g = 0.50, z = 2', 'g = 0.70, z = 2')
210
211 h_line = findobj(gcf, 'type', 'line');
212 set(h_line, 'LineWidth', 3);
213 h_axes = findobj(gcf, 'type', 'axes');
214 set(h_axes, 'linewidth', 2);
215 set(h_axes, 'FontSize', 15);
216
217 %%%
218 % Reference to control filtered Tru*W
219 F_Tru = Tru*W;
220
221
222 figure(17)
223 sigma(F_Tru,w);
224 %sv = 20*log10(sv);
225 %semilogx(w, sv, str(i,:))
226 title(' T_{ru}\cdot\dot{W} ', 'FontSize', 24)
227 grid on
228 xlabel('Frequency', 'FontSize', 24)
229 ylabel('Singular Values', 'FontSize', 24)
230 hold on
231 legend('g = 0.10, z = 2', 'g = 0.30, z = 2', ...
232 'g = 0.50, z = 2', 'g = 0.70, z = 2')
233 h_line = findobj(gcf, 'type', 'line');
234 set(h_line, 'LineWidth', 3);
235 h_axes = findobj(gcf, 'type', 'axes');
236 set(h_axes, 'linewidth', 2);
237 set(h_axes, 'FontSize', 15);
238
239 %%%
240 % Control response to filtered step command
241 figure(18)
242 step(F_Tru)
243 grid on
244 hold on
245 title('Control Response for step command with pre-filter', ...
246 'FontSize', 24)
247 xlabel('Time', 'FontSize', 24)
248 ylabel('Voltage (V)', 'FontSize', 24)
249 legend('g = 0.10, z = 2', 'g = 0.30, z = 2', ...
250 'g = 0.50, z = 2', 'g = 0.70, z = 2')
251 h_line = findobj(gcf, 'type', 'line');
252 set(h_line, 'LineWidth', 3);
253 h_axes = findobj(gcf, 'type', 'axes');
254 set(h_axes, 'linewidth', 2);
255 set(h_axes, 'FontSize', 15);

```

```

256
257 %%%
258 % Step response with prefilter Try W
259 figure(19)
260 step(clsc*W)
261 grid on
262 hold on
263 title('Step Response T_{ry} \cdot W', 'FontSize', 24)
264 xlabel('Time', 'FontSize', 24)
265 ylabel('Angular Velocity (rad/s)', 'FontSize', 24)
266
267 legend('g = 0.10, z = 2', 'g = 0.30, z = 2', ...
268 'g = 0.50, z = 2', 'g = 0.70, z = 2')
269 h_line = findobj(gcf, 'type', 'line');
270 set(h_line, 'LineWidth', 3);
271 h_axes = findobj(gcf, 'type', 'axes');
272 set(h_axes, 'linewidth', 2);
273 set(h_axes, 'FontSize', 15);
274
275 %%%
276 %% Tdiy states: [xp xk]^T
277 Adiy = [Ap-Bp*K.d*Cp Bp*K.c
278 -K.b*Cp K.a];
279 [row,col]=size(K.b);
280 Bdiy = [
281 Bp
282 0*ones(row,2)];
283 [row1,col1]=size(K.a);
284 Cdiy = [Cp 0*ones(2,col1)];
285 Ddiy = 0*ones(2,2);
286 Tdiy = ss(Adiy,Bdiy,Cdiy,Ddiy);
287
288 figure(191)
289 sigma(Tdiy,w);
290 %sv = 20*log10(sv);
291 %semilogx(w, sv, str(i,:))
292 title(' T_{diy} ', 'FontSize', 24)
293 xlabel('Frequency', 'FontSize', 24)
294 ylabel('Singular Values', 'FontSize', 24);
295
296 legend('g = 0.10, z = 2', 'g = 0.30, z = 2', ...
297 'g = 0.50, z = 2', 'g = 0.70, z = 2')
298 grid on
299 hold on
300
301 h_line = findobj(gcf, 'type', 'line');
302 set(h_line, 'LineWidth', 3);
303 h_axes = findobj(gcf, 'type', 'axes');
304 set(h_axes, 'linewidth', 2);
305 set(h_axes, 'FontSize', 15);
306
307 %%%
308 %% MSP Tdiy for (v,w)
309 Tdiy_vw = M*Tdiy;
310 figure(192)
311 sigma(Tdiy_vw,w);
312 title(' MSP Singular Values', 'FontSize', 24)
313 xlabel('Frequency', 'FontSize', 24)
314 ylabel('Singular Values', 'FontSize', 24)
315
316 grid on
317 hold on
318
319 legend('g = 0.10, z = 2', 'g = 0.30, z = 2', ...
320 'g = 0.50, z = 2', 'g = 0.70, z = 2')

```

```

322
323     h_line = findobj(gcf, 'type', 'line');
324     set(h_line, 'LineWidth', 3);
325     h_axes = findobj(gcf, 'type', 'axes');
326     set(h_axes, 'linewidth', 2);
327     set(h_axes, 'FontSize', 15);
328
329 %% KSM^-1 unfiltered Tru for (v,w)
330 Tru_vw = Tru*Minv;
331 figure(193)
332 sigma(Tru_vw,w);
333 title('KSM^{-1} Singular Values', 'FontSize', 24)
334 xlabel('Frequency', 'FontSize', 24)
335 ylabel('Singular Values', 'FontSize', 24)
336
337 grid on
338 hold on
339
340 legend('g = 0.10 , z = 2','g = 0.30 , z = 2',...
341           'g = 0.50 , z = 2','g = 0.70 , z = 2')
342 h_line = findobj(gcf, 'type', 'line');
343 set(h_line, 'LineWidth', 3);
344 h_axes = findobj(gcf, 'type', 'axes');
345 set(h_axes, 'linewidth', 2);
346 set(h_axes, 'FontSize', 15);
347
348 end
349
350
351 %% Open loop singular values, sensitivity,
352 %% comp sensitivity for 4 diff
353 %% values of z
354
355 g = 0.5;
356 z = [1 2 3 4];
357
358 for jj=1:1:length(z)
359 %% for jj=1:1:1
360     K = [ g*((s + z(jj))/s)*(40/(s+40))    0;
361             0                  g*((s + z(jj))/s)*(40/(s+40))];
362
363     K = ss(K);
364
365     W = [ z(jj)/(s+z(jj))    0
366             0      z(jj)/(s+z(jj)) ];
367
368 %% 
369 figure(20)
370 sigma(K,w);
371 %%sv = 20*log10(sv);
372 %%semilogx(w, sv, str(i,:))
373 title('Controller Singular Values ', 'FontSize', 24)
374 xlabel('Frequency', 'FontSize', 24)
375 ylabel('Singluar Values ', 'FontSize', 24)
376
377 legend('g = 0.50 ,z = 1','g = 0.50 ,z = 2',...
378           'g = 0.50 ,z = 3','g = 0.50 ,z = 4')
379 grid on
380 hold on
381
382
383 h_line = findobj(gcf, 'type', 'line');
384 set(h_line, 'LineWidth', 3);
385 h_axes = findobj(gcf, 'type', 'axes');
386 set(h_axes, 'linewidth', 2);
387 set(h_axes, 'FontSize', 15);

```

```

388 %%%
389 L = P*K;
390
391 figure(21)
392 sigma(L,w);
393 %sv = 20*log10(sv);
394 %semilogx(w, sv, str(i,:))
395 title('Open Loop Singular Values ', 'FontSize', 24)
396 xlabel('Frequency', 'FontSize', 24)
397 ylabel('Singular Values', 'FontSize', 24);
398 legend('g = 0.50 ,z = 1','g = 0.50 ,z = 2',...
399 'g = 0.50 ,z = 3','g = 0.50 ,z = 4')
400 grid on
401 hold on
402
403
404 h_line = findobj(gcf, 'type', 'line');
405 set(h_line, 'LineWidth', 3);
406 h_axes = findobj(gcf, 'type', 'axes');
407 set(h_axes, 'linewidth', 2);
408 set(h_axes, 'FontSize', 15);
409 %%%
410 % Sensitivity
411 asen = L.a - L.b*L.c;
412 bsen = L.b;
413 csen = -L.c;
414 [row, col] = size(csen);
415 [row1, col1] = size(bsen);
416 dsen = eye(row, col1);
417
418 figure(22)
419 sigma(ss(asen, bsen, csen, dsen),w);
420 %sv = 20*log10(sv);
421 %semilogx(w, sv, str(i,:))
422 title('Sensitivity ', 'FontSize', 24)
423 xlabel('Frequency', 'FontSize', 24)
424 ylabel('Singular Values', 'FontSize', 24);
425 legend('g = 0.50 ,z = 1','g = 0.50 ,z = 2',...
426 'g = 0.50 ,z = 3','g = 0.50 ,z = 4')
427
428 hold on
429 grid on
430
431 h_line = findobj(gcf, 'type', 'line');
432 set(h_line, 'LineWidth', 3);
433 h_axes = findobj(gcf, 'type', 'axes');
434 set(h_axes, 'linewidth', 2);
435 set(h_axes, 'FontSize', 15);
436
437 %%%
438
439 %Closed loop dynamics
440 acl = L.a - L.b*L.c;
441 bcl = L.b;
442 ccl = L.c;
443 dcl = L.d;
444 cls = ss(acl, bcl, ccl, dcl);
445
446 figure(23)
447 sigma(ss(acl, bcl, ccl, dcl),w);
448 %sv = 20*log10(sv);
449 %semilogx(w, sv, str(i,:))
450 title('Complementary Sensitivity ', 'FontSize', 24)
451 xlabel('Frequency', 'FontSize', 24)
452 ylabel('Singular Values', 'FontSize', 24);
453
454 grid on

```

```

455      hold on
456
457      legend( 'g = 0.50 ,z = 1' , 'g = 0.50 ,z = 2' , ...
458          'g = 0.50 ,z = 3' , 'g = 0.50 ,z = 4' )
459
460      h_line = findobj(gcf, 'type', 'line');
461      set(h_line, 'LineWidth', 3);
462      h_axes = findobj(gcf, 'type', 'axes');
463      set(h_axes, 'linewidth', 2);
464      set(h_axes, 'FontSize', 15);
465
466
467  %%%
468  % Step response unfiltered
469  figure(24)
470  step(cls)
471
472
473  title('Step Response for T_{ry} (Unfiltered)', ...
474      'FontSize', 24)
475  xlabel('Time', 'FontSize', 24)
476  ylabel('Angular Velocity (rad/s)', 'FontSize', 24)
477  grid on
478  hold on
479
480  legend( 'g = 0.50 ,z = 1' , 'g = 0.50 ,z = 2' , ...
481          'g = 0.50 ,z = 3' , 'g = 0.50 ,z = 4' )
482
483  h_line = findobj(gcf, 'type', 'line');
484  set(h_line, 'LineWidth', 3);
485  h_axes = findobj(gcf, 'type', 'axes');
486  set(h_axes, 'linewidth', 2);
487  set(h_axes, 'FontSize', 15);
488 %%%
489
490  % Reference to control Tru no filter
491  % states [xp xk]^T
492  Aru = [Ap-Bp*K.d*Cp  Bp*K.c
493          -K.b*Cp           K.a];
494
495  Bru = [Bp*K.d
496          K.b];
497  Cru = [-K.d*Cp  K.c];
498  Dru = K.d;
499  Tru = ss(Aru,Bru,Cru,Dru);
500
501  figure(25)
502  sigma(ss(Aru,Bru,Cru, Dru),w);
503  %sv = 20*log10(sv);
504  %semilogx(w, sv, str(i,:))
505  title(' T_{ru} (Unfiltered) ', 'FontSize', 24)
506  xlabel('Frequency', 'FontSize', 24)
507  ylabel('Singluar Values', 'FontSize', 24);
508
509  grid on
510  hold on
511
512  legend( 'g = 0.50 ,z = 1' , 'g = 0.50 ,z = 2' , ...
513          'g = 0.50 ,z = 3' , 'g = 0.50 ,z = 4' );
514
515  h_line = findobj(gcf, 'type', 'line');
516  set(h_line, 'LineWidth', 3);
517  h_axes = findobj(gcf, 'type', 'axes');
518  set(h_axes, 'linewidth', 2);
519  set(h_axes, 'FontSize', 15);
520

```

```

521 %%%
522
523 % Control response to step command
524 figure(26)
525 step(Tru)
526 hold on; grid on;
527 title('Control Response for step command (Unfiltered)', ...
528 'FontSize', 24)
529 xlabel('Time', 'FontSize', 24)
530 ylabel('Voltage (V)', 'FontSize', 24)
531
532 legend('g = 0.50 ,z = 1','g = 0.50 ,z = 2',...
533 'g = 0.50 ,z = 3','g = 0.50 ,z = 4')
534 h_line = findobj(gcf, 'type', 'line');
535 set(h_line, 'LineWidth', 3);
536 h_axes = findobj(gcf, 'type', 'axes');
537 set(h_axes, 'linewidth', 2);
538 set(h_axes, 'FontSize', 15);
539
540
541 %%%
542 % Reference to control filtered Tru*W
543 F_Tru = Tru*W;
544
545 figure(27)
546 sigma(F_Tru,w);
547 %sv = 20*log10(sv);
548 %semilogx(w, sv, str(i,:))
549 title(' T_{ru}\cdot W ', 'FontSize', 24)
550 xlabel('Frequency', 'FontSize', 24)
551 ylabel('Singular Values', 'FontSize', 24)
552 grid on
553 hold on
554
555 legend('g = 0.50 ,z = 1','g = 0.50 ,z = 2',...
556 'g = 0.50 ,z = 3','g = 0.50 ,z = 4')
557 h_line = findobj(gcf, 'type', 'line');
558 set(h_line, 'LineWidth', 3);
559 h_axes = findobj(gcf, 'type', 'axes');
560 set(h_axes, 'linewidth', 2);
561 set(h_axes, 'FontSize', 15);
562
563
564 %%%
565 % Control response to filtered step command
566 figure(28)
567 step(F_Tru)
568 grid on
569 hold on
570 title('Control Response for step command with pre-filter',...
571 'FontSize', 24)
572 xlabel('Time', 'FontSize', 24)
573 ylabel('Voltage (V)', 'FontSize', 24)
574 legend('g = 0.50 ,z = 1','g = 0.50 ,z = 2',...
575 'g = 0.50 ,z = 3','g = 0.50 ,z = 4')
576 h_line = findobj(gcf, 'type', 'line');
577 set(h_line, 'LineWidth', 3);
578 h_axes = findobj(gcf, 'type', 'axes');
579 set(h_axes, 'linewidth', 2);
580 set(h_axes, 'FontSize', 15);
581
582
583 %%%
584 % Step response with prefilter Try W
585 figure(29)
586 step(c1s*W)

```

```

587     grid on
588     hold on
589     title('Step Response T_{ry} \cdot W ', 'FontSize', 24)
590     xlabel('Time', 'FontSize', 24)
591     ylabel('Angular Velocity (rad/s)', 'FontSize', 24)
592
593     legend('g = 0.50,z = 1','g = 0.50,z = 2',...
594           'g = 0.50,z = 3','g = 0.50,z = 4')
595     h_line = findobj(gcf, 'type', 'line');
596     set(h_line, 'LineWidth', 3);
597     h_axes = findobj(gcf, 'type', 'axes');
598     set(h_axes, 'linewidth', 2);
599     set(h_axes, 'FontSize', 15);
600
601 %%%
602 % Tdiy states: [xp xk]^T
603 Adiy = [Ap-Bp*K.d*Cp Bp*K.c
604          -K.b*Cp K.a];
605 [row,col]=size(K.b);
606 Bdiy = [
607          Bp
608          0*ones(row,2)];
609 [row1,col1]=size(K.a);
610 Cdiy = [Cp 0*ones(2,col1)];
611 Ddiy = 0*ones(2,2);
612 Tdiy = ss(Adiy,Bdiy,Cdiy,Ddiy);
613
614 figure(291)
615 sigma(Tdiy,w);
616 grid on
617 hold on
618 %sv = 20*log10(sv);
619 %semilogx(w, sv, str(i,:))
620 title(' T_{diy} ', 'FontSize', 24)
621 xlabel('Frequency', 'FontSize', 24)
622 ylabel('Singular Values', 'FontSize', 24);
623
624
625 legend('g = 0.50,z = 1','g = 0.50,z = 2',...
626           'g = 0.50,z = 3','g = 0.50,z = 4')
627
628 h_line = findobj(gcf, 'type', 'line');
629 set(h_line, 'LineWidth', 3);
630 h_axes = findobj(gcf, 'type', 'axes');
631 set(h_axes, 'linewidth', 2);
632 set(h_axes, 'FontSize', 15);
633
634 %%
635 % MSP Tdiy for (v,w)
636 Tdiy_vw = M*Tdiy;
637 figure(292)
638 sigma(Tdiy_vw,w);
639 title(' MSP Singular Values ', 'FontSize', 24)
640 xlabel('Frequency', 'FontSize', 24)
641 ylabel('Singular Values', 'FontSize', 24)
642
643 grid on
644 hold on
645
646 legend('g = 0.50,z = 1','g = 0.50,z = 2',...
647           'g = 0.50,z = 3','g = 0.50,z = 4')
648
649 h_line = findobj(gcf, 'type', 'line');
650 set(h_line, 'LineWidth', 3);
651 h_axes = findobj(gcf, 'type', 'axes');
652 set(h_axes, 'linewidth', 2);

```

```

653      set(h_axes, 'FontSize', 15);
654  %%%
655  % KSM^-1 unfiltered Tru for (v,w)
656  Tru_vw = Tru*Minv;
657  figure(293)
658  sigma(Tru_vw,w);
659  title('KSM^{-1} Singular Values', 'FontSize', 24)
660  xlabel('Frequency', 'FontSize', 24)
661  ylabel('Singular Values', 'FontSize', 24)
662  grid on
663  hold on
664
665
666  legend('g = 0.50,z = 1','g = 0.50,z = 2',...
667  'g = 0.50,z = 3','g = 0.50,z = 4')
668  h_line = findobj(gcf, 'type', 'line');
669  set(h_line, 'LineWidth', 3);
670  h_axes = findobj(gcf, 'type', 'axes');
671  set(h_axes, 'linewidth', 2);
672  set(h_axes, 'FontSize', 15);
673
674
675 end

1 %% inner loop design
2 clear all
3 close all
4 clc
5
6
7 %%
8 S_avg = load('ETT_avg.mat')
9 S2 = load('ETT_TITO_Model_2nd_order.mat')
10
11 M = S2.ETT_M;
12 Minv = inv(M);
13
14 p11 = S_avg.tf_ett;
15 P_tf = [p11 0;
16 0 p11];
17 P = ss(P_tf); %(wr,wl) system
18 Ap = P.a; Bp = P.b; Cp = P.c; Dp = P.d;
19
20
21 s = tf('s');
22
23 %%
24 figure(1)
25 step(P, 5)
26
27
28 title('Plant Step Response', 'FontSize', 24)
29 xlabel('Time', 'FontSize', 24)
30 ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
31
32 grid on
33 hold on
34
35
36 h_line = findobj(gcf, 'type', 'line');
37 set(h_line, 'LineWidth', 3);
38 h_axes = findobj(gcf, 'type', 'axes');
39 set(h_axes, 'linewidth', 2);
40 set(h_axes, 'FontSize', 15);
41 %%
42

```

```

43
44 figure(2)
45 bode(P)
46 title('Plant Frequency Response', 'FontSize', 24);
47 xlabel('Frequency', 'FontSize', 24)
48 ylabel('Phase', 'FontSize', 24);
49 hold on; grid on;
50
51
52 h_line = findobj(gcf, 'type', 'line');
53 set(h_line, 'LineWidth', 3);
54 h_axes = findobj(gcf, 'type', 'axes');
55 set(h_axes, 'linewidth', 2);
56 set(h_axes, 'FontSize', 15);
57
58
59
60
61 %%%
62
63 g = 0.2942;
64 z = 2.0091;
65
66 Kinner_tf = [ g*(s+z)*100/(s*(s+100)) 0
67 0 g*(s+z)*100/(s*(s+100)) ];
68
69 Kinner = ss(Kinner_tf);
70
71 Linner = P*Kinner; %open loop
72
73
74 asen = Linner.a - Linner.b*Linner.c; % sensitivity
75 bsen = Linner.b;
76 csen = -Linner.c;
77 [row,col] = size(csen);
78 [row1,col1] = size(bsen);
79 dsen = eye(row,col1);
80 Sinner = ss(asen,bsen,csen,dsen);
81
82 acl = Linner.a - Linner.b*Linner.c;
83 % comp sensitivity unfiltered
84 bcl = Linner.b;
85 ccl = Linner.c;
86 dcl = Linner.d;
87 T = ss(acl,bcl,ccl,dcl);
88
89
90 W_tf = [z/(s+z) 0
91 0 z/(s+z)];
92 W = ss(W_tf);
93
94 Try = T*W; % try = comp sensitivity filtered
95
96 % Tdiy
97 Adiy = [Ap-Bp*Kinner.d*Cp Bp*Kinner.c
98 -Kinner.b*Cp Kinner.a];
99 [row,col]=size(Kinner.b);
100 Bdiy = [
101 Bp
102 0*ones(row,2)];
103 [row1,col1]=size(Kinner.a);
104 Cdiy = [Cp 0*ones(2,col1)];
105 Ddiy = 0*ones(2,2);
106 Tdiy = ss(Adiy,Bdiy,Cdiy,Ddiy);
107
108 Tru = Kinner*Sinner; % Tru unfiltered
109 Truf = Kinner*Sinner*W; % Tru filtered

```

```

110
111 Tru_vw = Tru*Minv; % Tru vw unfiltered
112 Tdiy_vw = M*Tdiy; % Tdiy vw
113
114
115 %%%
116 figure(3) % open loop singular value
117 w = logspace(-2,3,100);
118 Lvw = Minv * Linner * M
119 sigma(Linner,Lvw,w);
120 title('Open Loop Singular Values ', 'FontSize', 24)
121 xlabel('Frequency', 'FontSize', 24)
122 ylabel('Singluar Values', 'FontSize', 24);
123 legend('PK singular values', 'MPKM{-1} singular values')
124 grid on
125 hold on
126
127
128 h_line = findobj(gcf, 'type', 'line');
129 set(h_line, 'LineWidth', 3);
130 h_axes = findobj(gcf, 'type', 'axes');
131 set(h_axes, 'linewidth', 2);
132 set(h_axes, 'FontSize', 15);
133
134
135
136
137
138
139
140 %%%
141
142 figure(4) %sensitivity
143 sigma(Sinner,w);
144 title('Sensitivity Singular Values', 'FontSize', 24)
145 xlabel('Frequency', 'FontSize', 24)
146 ylabel('Singluar Values', 'FontSize', 24);
147 %% legend(XXX)
148 grid on
149 hold on
150
151
152 h_line = findobj(gcf, 'type', 'line');
153 set(h_line, 'LineWidth', 3);
154 h_axes = findobj(gcf, 'type', 'axes');
155 set(h_axes, 'linewidth', 2);
156 set(h_axes, 'FontSize', 15);
157
158
159
160 %%%
161
162 figure(5) %comp sensitivity
163
164 sigma(T,w);
165 title('Complementary Sensitivity Singular Values',...
166 'FontSize', 24)
167 xlabel('Frequency', 'FontSize', 24)
168 ylabel('Singluar Values', 'FontSize', 24);
169 %% legend(XXX)
170 grid on
171 hold on
172
173
174 h_line = findobj(gcf, 'type', 'line');
175 set(h_line, 'LineWidth', 3);
176 h_axes = findobj(gcf, 'type', 'axes');

```

```

177 set(h_axes, 'LineWidth', 2);
178 set(h_axes, 'FontSize', 15);
179
180
181
182 %%%
183 figure(6) %tdiy
184
185 sigma(Tdiy,w);
186 title('T_{diy} Singular Values', 'FontSize', 24)
187 xlabel('Frequency', 'FontSize', 24)
188 ylabel('Singluar Values', 'FontSize', 24);
189 %% legend(XXX)
190 grid on
191 hold on
192
193
194 h_line = findobj(gcf, 'type', 'line');
195 set(h_line, 'LineWidth', 3);
196 h_axes = findobj(gcf, 'type', 'axes');
197 set(h_axes, 'LineWidth', 2);
198 set(h_axes, 'FontSize', 15);
199
200
201
202
203
204 %%%
205 figure(7) %tru
206
207 sigma(Tru,w);
208 title(' T_{ru} Singular Values ', 'FontSize', 24)
209 xlabel('Frequency', 'FontSize', 24)
210 ylabel('Singluar Values', 'FontSize', 24);
211 %% legend(XXX)
212 grid on
213 hold on
214
215
216 h_line = findobj(gcf, 'type', 'line');
217 set(h_line, 'LineWidth', 3);
218 h_axes = findobj(gcf, 'type', 'axes');
219 set(h_axes, 'LineWidth', 2);
220 set(h_axes, 'FontSize', 15);
221
222
223
224
225 %%%
226 figure(8) %tru_filtered
227 sigma(Truf,w);
228 title(' T_{ru} W Singular Values ', 'FontSize', 24)
229 xlabel('Frequency', 'FontSize', 24)
230 ylabel('Singluar Values', 'FontSize', 24);
231 %% legend(XXX)
232 grid on
233 hold on
234
235
236 h_line = findobj(gcf, 'type', 'line');
237 set(h_line, 'LineWidth', 3);
238 h_axes = findobj(gcf, 'type', 'axes');
239 set(h_axes, 'LineWidth', 2);
240 set(h_axes, 'FontSize', 15);
241
242
243

```

```

244 %% step opt setting
245 opt = stepDataOptions('StepAmplitude',10);
246 %%
247 figure(9) % Try W
248 step(Try,5, 'b', opt);
249 title('Step response (Filtered)', 'FontSize', 24)
250 xlabel('Time', 'FontSize', 24)
251 ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
252 hold on;grid on;
253
254 h_line = findobj(gcf, 'type', 'line');
255 set(h_line, 'LineWidth', 3);
256 h_axes = findobj(gcf, 'type', 'axes');
257 set(h_axes, 'linewidth', 2);
258 set(h_axes, 'FontSize', 15);
259
260
261
262 %%
263 %%
264 figure(10) % Try
265 step(T,5, 'b', opt)
266 title('Step response (Unfiltered)', 'FontSize', 24)
267 xlabel('Time', 'FontSize', 24)
268 ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
269 hold on;grid on;
270
271 h_line = findobj(gcf, 'type', 'line');
272 set(h_line, 'LineWidth', 3);
273 h_axes = findobj(gcf, 'type', 'axes');
274 set(h_axes, 'linewidth', 2);
275 set(h_axes, 'FontSize', 15);
276
277
278 %%
279 %%
280
281 figure(11) % Tru
282 step(Tru,5, 'b', opt)
283 title('Control response (Unfiltered)', 'FontSize', 24)
284 xlabel('Time', 'FontSize', 24)
285 ylabel('Voltage(V)', 'FontSize', 24);
286 hold on;grid on;
287
288 h_line = findobj(gcf, 'type', 'line');
289 set(h_line, 'LineWidth', 3);
290 h_axes = findobj(gcf, 'type', 'axes');
291 set(h_axes, 'linewidth', 2);
292 set(h_axes, 'FontSize', 15);
293
294
295 %%
296 %%
297 figure(12) % Tru W
298 step(Truf,5, 'b', opt)
299 title('Control response (Filtered)', 'FontSize', 24)
300 xlabel('Time', 'FontSize', 24)
301 ylabel('Voltage (V)', 'FontSize', 24);
302 hold on;grid on;
303
304 h_line = findobj(gcf, 'type', 'line');
305 set(h_line, 'LineWidth', 3);
306 h_axes = findobj(gcf, 'type', 'axes');
307 set(h_axes, 'linewidth', 2);
308 set(h_axes, 'FontSize', 15);
309

```

```

310
311 %%%
312 figure(13)      %MSP
313 sigma(Tdiy_vw,w);
314 title('MSP Singular Values', 'FontSize', 24)
315 xlabel('Frequency', 'FontSize', 24)
316 ylabel('Singluar Values', 'FontSize', 24);
317 % legend(XXX)
318 grid on
319 hold on
320
321
322 h_line = findobj(gcf, 'type', 'line');
323 set(h_line, 'LineWidth', 3);
324 h_axes = findobj(gcf, 'type', 'axes');
325 set(h_axes, 'linewidth', 2);
326 set(h_axes, 'FontSize', 15);
327
328
329
330 %%
331 figure(14)      %KSM^ {-1}
332 sigma(Tru_vw,w);
333 title('KSM^ {-1} Singular Values', 'FontSize', 24)
334 xlabel('Frequency', 'FontSize', 24)
335 ylabel('Singluar Values', 'FontSize', 24);
336 % legend(XXX)
337 grid on
338 hold on
339
340
341 h_line = findobj(gcf, 'type', 'line');
342 set(h_line, 'LineWidth', 3);
343 h_axes = findobj(gcf, 'type', 'axes');
344 set(h_axes, 'linewidth', 2);
345 set(h_axes, 'FontSize', 15);
346
347 %%
348 % loopsens function testing
349 K_tf = Kinner_tf;
350 SF = loopsens(P_tf, K_tf)
351 Twrwl = series(W_tf, SF.To)
352 % figure(1000);
353 % bode(Twrwl, 'b^', Try, 'r-')
354
355 %%
356 [num1,den1] = ss2tf(Try.a, Try.b, Try.c, Try.d, 1);
357
358 n11 = num1(1,:);
359 d11 = den1(1,:);
360 Try_tf11 = zpk(minreal(tf(n11,d11)))
361
362 [num2,den2] = ss2tf(Try.a, Try.b, Try.c, Try.d, 2);
363
364 n22 = num2(2,:);
365 d22 = den2(1,:);
366 Try_tf22 = zpk(minreal(tf(n22,d22)))
367
368 %%
369 Tvw = series(series(Minv, Twrwl), M)
370 Tvw_test = M* Twrwl * Minv
371 figure(2000);
372 bode(Tvw, 'b^', Tvw_test, 'r-')
373
374 %%
375 [num1,den1] = ss2tf(Tvw.a, Tvw.b, Tvw.c, Tvw.d, 1);

```

```

376
377 n11 = num1(1,:);
378 d11 = den1(1,:);
379 Tvw_tf11 = zpk(minreal(tf(n11,d11)))
380
381
382 figure(3000);
383 bode(Tvw_tf11, 'b^', Try_tf11, 'r-')

1
2 % Outer-loops controller for single vehicle
3 % 2016-07-06
4 % P control Freq
5 close all
6 Kp_vec = [0.5 1 1.5 2];
7 cell_size = length(Kp_vec);
8
9 for ii=1:1:cell_size
10    fig101 = figure(101);
11    K = tf(Kp_vec(ii));
12    S = siso_tf_generator(1,P,K);
13    % theta Frequency Response for Outer-Loop (P Control)
14    bode(S.Try);
15    title('Frequency response of cruise control ...
16          \theta with P control',...
17          'FontSize', 24)
18    hold on; grid on;
19    add_legend_in_for_loop('','Kp ',Kp_vec,'float ',ii ,cell_size);
20
21    h_line = findobj(gcf, 'type', 'line');
22    set(h_line, 'LineWidth', 3);
23    h_axes = findobj(gcf, 'type', 'axes');
24    set(h_axes, 'linewidth', 2);
25    set(h_axes, 'FontSize', 15);
26
27    xlabel('Frequency', 'FontSize', 24)
28    ylabel(' Phase', 'FontSize', 24);
29
30 end
31 %% P control theta_0 = 0.1rad
32
33 time = theta_P_resp.time;
34 y1 = theta_P_resp.signals.values(:,2);
35 y2 = theta_P_resp.signals.values(:,3);
36 y3 = theta_P_resp.signals.values(:,4);
37 y4 = theta_P_resp.signals.values(:,5);
38
39 fig103 = figure(103);
40 plot(time,[y1 y2 y3 y4]);
41 h_line = findobj(gcf, 'type', 'line');
42 set(h_line, 'LineWidth', 3);
43 h_axes = findobj(gcf, 'type', 'axes');
44 set(h_axes, 'linewidth', 2);
45 set(h_axes, 'FontSize', 15);
46
47 title('Cruise control \theta response using P control',...
48      'FontSize', 24)
49 xlabel('Time(seconds)', 'FontSize', 24)
50 ylabel('Angle(rad)', 'FontSize', 24);
51 axis([0 20 -0.05 0.1])
52 hold on; grid on;
53 %      add_legend_in_for_loop('','Kp ',Kp_vec,'float ',ii ,cell_size);
54 legend('Kp=0.5 ','Kp=1.0 ','Kp=1.5 ','Kp=2.0 ');
55
56

```

```

57 %% PD control Frequency
58
59 % Kp = g*z;% g = Kd; K = Kd(s+Kp/Kd);
60 N = 40;
61 % g_vec = [0.3:0.1:0.6]; z = 2;
62 % cell_size = length(g_vec);
63 Kd = 1;
64 cell_size = length(Kp_vec);
65 K_pd_cell = cell(cell_size,1);
66 for ii=1:1:cell_size
67     fig102 = figure(102);
68     Kp = Kp_vec(ii);
69     % K = zpk(-z,-N,g*N);
70     K = pid(Kp,0,Kd);
71     rf = tf(N,[1 N]);
72     K_rf = series(K,rf);
73     K_pd_cell{ii} = tf(K_rf);
74     S = siso_tf_generator(1,P,K_rf);
75     bode(S.Try);
76     title('Frequency response of cruise control ... \theta with PD control',...
77           'FontSize', 24)
78     hold on; grid on;
79     add_legend_in_for_loop(' ', 'Kp ', Kp_vec, 'float', ii, cell_size);
80
81     h_line = findobj(gcf, 'type', 'line');
82     set(h_line, 'LineWidth', 3);
83     h_axes = findobj(gcf, 'type', 'axes');
84     set(h_axes, 'linewidth', 2);
85     set(h_axes, 'FontSize', 15);
86
87     xlabel('Frequency', 'FontSize', 24)
88     ylabel(' Phase', 'FontSize', 24);
89
90 end
91
92
93
94
95 %% PD control time response
96 time = theta_PD.resp.time;
97 y1 = theta_PD.resp.signals.values(:,2);
98 y2 = theta_PD.resp.signals.values(:,3);
99 y3 = theta_PD.resp.signals.values(:,4);
100 y4 = theta_PD.resp.signals.values(:,5);
101
102 fig104 = figure(104);
103 plot(time,[y1 y2 y3 y4]);
104
105
106 title('Cruise control \theta response using PD control',...
107       'FontSize', 24)
108 xlabel('Time(seconds)', 'FontSize', 24)
109 ylabel('Angle(rad)', 'FontSize', 24);
110 axis([0 20 -0.05 0.1])
111 hold on; grid on;
112 legend('Kp=0.5 ', 'Kp=1.0 ', 'Kp=1.5 ', 'Kp=2.0 ');
113 h_line = findobj(gcf, 'type', 'line');
114 set(h_line, 'LineWidth', 3);
115 h_axes = findobj(gcf, 'type', 'axes');
116 set(h_axes, 'linewidth', 2);
117 set(h_axes, 'FontSize', 15);
118
119
120 save theta_simlink theta*

```

```

1 % Outer-loops controller for single vehicle
2 % separation control
3 % 2016-07-20
4
5
6
7 %%%
8 clc
9 close all
10
11 %%
12 load('basis_for_outer_loop.mat')
13 speed_lower_limit = -1;
14
15 %%
16 Kp_vec =[0.5 0.6 0.7 1];
17 cell_size = length(Kp_vec);
18 %
19 for ii=1:1:cell_size
20
21 K = tf(Kp_vec(ii));
22 S = siso_tf_generator(1,P,K);
23 % theta Frequency Response for Outer-Loop (P Control)
24 %
25 % fig101 = figure(101);
26 % bode(S.Try);
27 % title('Frequency response of outer loop with P control',...
28 % 'FontSize', 24)
29 % hold on; grid on;
30 % add_legend_in_for_loop('','Kp ',Kp_vec,'float ',ii ,cell_size );
31 %
32 % h_line = findobj(gcf, 'type', 'line');
33 % set(h_line, 'LineWidth', 3);
34 % h_axes = findobj(gcf, 'type', 'axes');
35 % set(h_axes, 'linewidth', 2);
36 % set(h_axes, 'FontSize', 15);
37 %
38 xlabel('Frequency ','FontSize', 24)
39 ylabel(' Phase ','FontSize', 24);
40 end
41
42
43
44 %% PD control
45 Kp_vec =[0.5 1 1.5 2];
46 % Kp_vec =[1 2];
47 % Kp = g*z;% g = Kd; K = Kd(s+Kp/Kd);
48 N = 40;
49 Kd = 1; % in thesis now
50 cell_size = length(Kp_vec);
51 K_pd_cell = cell(cell_size ,1);
52
53 % Frequency
54 for ii=1:1:cell_size
55
56 Kp = Kp_vec(ii);
57 K = pid(Kp,0 ,Kd);
58 rf = tf(N,[1 N]);
59 K_rf = series(K, rf);
60 K_pd_cell{ii} = tf(K_rf);
61 S = siso_tf_generator(1,P,K_rf);
62 %
63 % fig102 = figure(102);
64 % bode(S.Try);
65 % title('Frequency response of ...
66 % outer loop with PD control',...
% 'FontSize', 24)

```

```

67 %      hold on; grid on;
68 %      add_legend_in_for_loop ('', 'Kp ', Kp_vec, 'float ', ii , cell_size);
69 %
70 %      h_line = findobj(gcf, 'type', 'line ');
71 %      set(h_line, 'LineWidth', 3);
72 %      h_axes = findobj(gcf, 'type', 'axes ');
73 %      set(h_axes, 'linewidth', 2);
74 %      set(h_axes, 'FontSize', 15);
75 %
76 %      xlabel('Frequency', 'FontSize', 24)
77 %      ylabel(' Phase', 'FontSize', 24);
78
79 end
80
81
82
83 %% P control delta_x = 0.2 delta_x0 = 1
84 % P_vec = [0.4:0.2:1]
85 % time = separation_P_resp.time;
86 y1 = separation_P_resp.signals.values(:,2);
87 y2 = separation_P_resp.signals.values(:,3);
88 y3 = separation_P_resp.signals.values(:,4);
89 y4 = separation_P_resp.signals.values(:,5);
90
91 fig201 = figure(201);
92 plot(time,[y1 y2 y3 y4]);
93 h_line = findobj(gcf, 'type', 'line ');
94 set(h_line, 'LineWidth', 3);
95 h_axes = findobj(gcf, 'type', 'axes ');
96 set(h_axes, 'linewidth', 2);
97 set(h_axes, 'FontSize', 15);
98
99 title('Constant separation using P control',...
100      'FontSize', 24)
101 xlabel('Time(seconds)', 'FontSize', 24)
102 ylabel('Distance(m)', 'FontSize', 24);
103
104 hold on; grid on;
105 %      add_legend_in_for_loop ('', 'Kp ', P_vec, 'float ', ii , cell_size);
106 %      legend('Kp=0.4 ','Kp=0.6 ','Kp=0.8 ','Kp=0.1 ');
107 %      legend('Kp=0.5 ','Kp=1.0 ','Kp=1.5 ','Kp=2.0 ');
108 %      legend('Kp=0.7 ','Kp=1.0 ','Kp=1.2 ','Kp=1.5 ');
109 %      legend('Kp=0.5 ','Kp=0.6 ','Kp=0.7 ','Kp=1.0 ');
110
111
112 %% Tru for P control
113 time = separation_P_resp.time;
114 u1 = separation_P_resp_vel.signals.values(:,1);
115 u2 = separation_P_resp_vel.signals.values(:,2);
116 u3 = separation_P_resp_vel.signals.values(:,3);
117 u4 = separation_P_resp_vel.signals.values(:,4);
118
119 fig301 = figure(301);
120 plot(time,[u1 u2 u3 u4]);
121 h_line = findobj(gcf, 'type', 'line ');
122 set(h_line, 'LineWidth', 3);
123 h_axes = findobj(gcf, 'type', 'axes ');
124 set(h_axes, 'linewidth', 2);
125 set(h_axes, 'FontSize', 15);
126
127 title('Tru of P control',...
128      'FontSize', 24)
129 xlabel('Time(seconds)', 'FontSize', 24)
130 ylabel('Speed(m/sec)', 'FontSize', 24);
131 axis([0 max(time) -0.1 2.5])
132 hold on; grid on;

```

```

133 %      add_legend_in_for_loop (' ', 'Kp ', P_vec , 'float ', ii , cell_size);
134 %      legend('Kp=0.4 ','Kp=0.6 ','Kp=0.8 ','Kp=0.1 ');
135 %      legend('Kp=0.5 ','Kp=1.0 ','Kp=1.5 ','Kp=2.0 ');
136 %      legend('Kp=0.5 ','Kp=0.7 ','Kp=1.5 ','Kp=2.0 ');
137 %      legend('Kp=0.5 ','Kp=0.6 ','Kp=0.7 ','Kp=1.0 ');
138
139
140
141
142 %% PD control Simulink separation
143 time = separation_PD_resp.time;
144 y1 = separation_PD_resp.signals.values (:,2);
145 y2 = separation_PD_resp.signals.values (:,3);
146 y3 = separation_PD_resp.signals.values (:,4);
147 y4 = separation_PD_resp.signals.values (:,5);
148
149 fig202 = figure(202);
150 plot(time ,[y1 y2 y3 y4]);
151 h_line = findobj(gcf, 'type', 'line');
152 set(h_line, 'LineWidth', 3);
153 h_axes = findobj(gcf, 'type', 'axes');
154 set(h_axes, 'linewidth', 2);
155 set(h_axes, 'FontSize', 15);
156
157 title('Constant separation using PD control',...
158 'FontSize', 24)
159 xlabel('Time(seconds)', 'FontSize', 24)
160 ylabel('Distance(m)', 'FontSize', 24);
161 axis([0 20 0 1.05])
162 hold on; grid on;
163 %      legend('Kd=0.4 ','Kd=0.6 ','Kd=0.8 ','Kd=1.0 ');
164 legend('Kp=0.5 ','Kp=1.0 ','Kp=1.5 ','Kp=2.0 ');
165 %      legend('Kp=0.5 ','Kp=0.7 ','Kp=1.5 ','Kp=2.0 ');
166
167
168
169 %% Tru for PD control
170 time = separation_PD_resp.time;
171 u1 = separation_PD_resp_vel.signals.values (:,1);
172 u2 = separation_PD_resp_vel.signals.values (:,2);
173 u3 = separation_PD_resp_vel.signals.values (:,3);
174 u4 = separation_PD_resp_vel.signals.values (:,4);
175
176 fig302 = figure(302);
177 plot(time ,[u1 u2 u3 u4]);
178 h_line = findobj(gcf, 'type', 'line');
179 set(h_line, 'LineWidth', 3);
180 h_axes = findobj(gcf, 'type', 'axes');
181 set(h_axes, 'linewidth', 2);
182 set(h_axes, 'FontSize', 15);
183
184 title('Tru of PD control with Kd = 1',...
185 'FontSize', 24)
186 xlabel('Time(seconds)', 'FontSize', 24)
187 ylabel('Speed(m/sec)', 'FontSize', 24);
188 axis([0 max(time) -0.1 2.5])
189 hold on; grid on;
190 %      add_legend_in_for_loop (' ', 'Kp ', P_vec , 'float ', ii , cell_size);
191 %      legend('Kp=0.4 ','Kp=0.6 ','Kp=0.8 ','Kp=0.1 ');
192 legend('Kp=0.5 ','Kp=1.0 ','Kp=1.5 ','Kp=2.0 ');
193 %      legend('Kp=0.5 ','Kp=0.7 ','Kp=1.5 ','Kp=2.0 ');

```

1 % Outer-loops controller for Platoon
2 % 2016-08-04 Variations

```

3 close all
4 load('basis_for_outer_loop.mat')
5 P_out = tf(P)
6 tranlation_speed_saturation = [-2 2];
7 angular_speed_saturation = tranlation_speed_saturation./RADIUS;
8 % voltage2pwm_gain = 255/BV;
9 voltage2pwm_gain = 1;% uncomment translation cancel this time
10 % Initial Conditions !!!!! DON'T FORGET
11 vehicle_num = 7;% include leader
12 follower_num = vehicle_num - 1;
13 vel_ic_leader = 0;
14
15 % follower initial conditions
16 acc_ic = zeros(1,follower_num);
17 assigned_space = 0.4;
18 pos_ic = -assigned_space.* (1:1:follower_num)
19 vehicle_id = 1:1:follower_num;
20
21 N = 100;
22 rf = tf(N,[1 N])
23
24
25 %Nominal Model Variations
26
27 % Non identical model
28 % P_in =
29
30 %      13.06
31 %
32 %   s + 1.159
33
34 dp_nominal = P_in.den{1}(2);
35 dc_nominal = P_in.num{1}(2)./dp_nominal;%11.36 ;
36
37 % dc_arr      = [12 12 10 12 10 12];
38 % dp_arr      = dp_nominal.*[0.7 2 1.2 2 1 2]
39
40 % dc_arr      = dc_nominal*[0.9 1.1 0.9 1 1.2 1];
41 % dp_arr      = dp_nominal.*[1.2 1.5 1.2 0.9 1.2 1];
42
43 dc_arr      = dc_nominal*[1 1 1 1 1 1];
44 dp_arr      = dp_nominal.*[1 1 1 1 1 1];
45
46 % S = load('model_uncertain.mat');
47 %
48 % dc_arr      = [ S.dc_arr(3) , S.dc_arr(4) , S.dc_arr(5) , ...
49 %                  S.dc_arr(7) , S.dc_arr(8) , S.dc_arr(9) ]
50 % dp_arr      = [ S.dp_arr(3) , S.dp_arr(4) , S.dp_arr(5) , ...
51 %                  S.dp_arr(7) , S.dp_arr(8) , S.dp_arr(9) ]
52
53 %
54 for ii=1:1:6
55     n_arr(ii) = dp_arr(ii)*dc_arr(ii);
56 end
57
58 P_in1 = tf([ n_arr(1) ],[1 dp_arr(1)]);
59 P_in2 = tf([ n_arr(2) ],[1 dp_arr(2)]);
60 P_in3 = tf([ n_arr(3) ],[1 dp_arr(3)]);
61 P_in4 = tf([ n_arr(4) ],[1 dp_arr(4)]);
62 P_in5 = tf([ n_arr(5) ],[1 dp_arr(5)]);
63 P_in6 = tf([ n_arr(6) ],[1 dp_arr(6)]);
64
65
66 dz_nominal = 40/255*BV;
67
68 % dz_arr = dz_nominal*[1 1.5 1.2 1 1 1.4 ]
69 dz_arr = dz_nominal*[1 1.5 1.2 0.7 1 2]*0;

```

```

70
71
72
73 %% -----PID + PI -----
74
75 %
76 K_out = pid(1,0.5,0.3)
77 K_out = series(K_out,rf);
78 K_out = series(K_out,rf)
79 zpk(K_out)
80 % PI feedforward path wrt speed
81 % z_ff = 3;
82 % g_ff = 1;
83 z_ff = 1.5;
84 g_ff = 0.4;
85
86 K_FF = zpk([-z_ff],[0 -100],[100*g_ff])
87 K_FF = tf(K_FF)
88 PI_FF_vec = [g_ff g_ff*z_ff 0]
89
90
91
92 %% Retrieve data from simulink and plot
93 close all
94 % title_str = 'Platoon of 7 Vehicle without Feed Forward Information';
95 % title(title_str);
96
97 % if you want to upadate simulation result
98 % Run SIMLINK model before this section
99 % platoon_control_no_feed_forward_0531
100 % Figure 10X
101 disp('Plotting Simulation Result... ')
102 Time_FF = Platoon_vel_FF.time;
103 Vel_FF = Platoon_vel_FF.signals.values;
104 Pos_FF = Platoon_pos_FF.signals.values;
105 Delta_X_FF = Platoon_deltaX_FF.signals.values;
106 % lineType_cell={'k-','b','r','g','','','m'};
107
108
109 % Try separation of platoon
110
111 Platoon_Separation = Delta_X_FF + 0.4;
112 fig203 = figure(203);
113 plot(Time_FF,Platoon_Separation(:,1), 'k--', ...
114      Time_FF,Platoon_Separation(:,2), ...
115      Time_FF,Platoon_Separation(:,3), ...
116      Time_FF,Platoon_Separation(:,4), ...
117      Time_FF,Platoon_Separation(:,5), ...
118      Time_FF,Platoon_Separation(:,6));
119
120 title('Simulation result of platoon of 5 vehicle ', 'FontSize', 24)
121 xlabel('Time(seconds)', 'FontSize', 24)
122 ylabel('Vehicle Separation (m)', 'FontSize', 24);
123 hold on; grid on;
124 % legend('\Delta_1','\Delta_2','\Delta_3','\Delta_5')
125 axis([0 30 0.3 0.7])
126 legend('Reference','Follower 1','Follower 2','Follower 3',...
127        'Follower 4','Follower 5')
128 h_line = findobj(gcf, 'type', 'line');
129 set(h_line, 'LineWidth', 3);
130 h_axes = findobj(gcf, 'type', 'axes');
131 set(h_axes, 'linewidth', 2);
132 set(h_axes, 'FontSize', 15);
133
134 % control
135
136 u1 = Platoon_u1(:,2);

```

```

137 u2 = Platoon_u2(:,2);
138 u3 = Platoon_u3(:,2);
139 u4 = Platoon_u4(:,2);
140 u5 = Platoon_u5(:,2);
141 u6 = Platoon_u6(:,2);
142
143 fig300 = figure(300);
144 plot(Time_FF,u1, ...
145 Time_FF,u2, ...
146 Time_FF,u3, ...
147 Time_FF,u4, ...
148 Time_FF,u5)
149 h_line = findobj(gcf, 'type', 'line');
150 set(h_line, 'LineWidth', 3);
151 h_axes = findobj(gcf, 'type', 'axes');
152 set(h_axes, 'linewidth', 2);
153 set(h_axes, 'FontSize', 24);
154
155 hold on; grid on;
156 axis([0 30 -0.5 1.5])
157 title('Control response of platoon of 5 vehicle', 'FontSize', 24);
158 xlabel('Time (sec)')
159 ylabel('Voltage (V)')
160
161 legend('Follower 1','Follower 2','Follower 3',...
162 'Follower 4','Follower 5');

```

APPENDIX C
ARDUINO CODE

```

1 // motion control (differential drive)
2 // Zhichao Li
3 // 2016-08-14
4 #include <Wire.h>
5 #include <math.h>
6 #include <TimerOne.h>
7
8 // Vehicle Setting Macro
9 #define CONFIG_PLATFORM_ETT
10
11
12 // system state can be 0 (stopped) or 1 (running),
13 // i.e. motor power state
14 int motor_power_state = 0;
15
16 // The controller mode can be either 0 (open_loop)
17 // or 1 (close_loop)
18 int ctrl_mode = 0;
19
20 // deug flag
21 int debug_mode = 1;
22
23 long timestamps[8];
24
25 // ----- function forward declaration -----
26
27 void serial_parse_command(int opcode);
28 void serial_state_machine_proceed(int c);
29 void serial_send_header(byte len, byte opcode);
30
31 // hardware abstraction layer
32 void m_set_pwm(int v_left_new, int v_left_direction,
33                 int v_right_new, int v_right_direction);
34 void m_stop();
35
36 long encoder_left_read();
37 long encoder_right_read();
38 long encoder_left_write();
39 long encoder_right_write();
40
41 // control interface
42 void ctrl_reset();
43 void ctrl_set_open_loop();
44 void ctrl_set_wl_wr(float wl_dsr_new, float wr_dsr_new);
45
46 // ----- hardware dependent code -----
47
48 // ----- ETT -----
49
50 #if defined (CONFIG_PLATFORM_ETT)
51
52 // Motor
53 #define CONFIG_MOTOR_2WD_ADAFRUIT
54
55 #define M_LEFT_MOTOR_INDEX 3
56 #define M_RIGHT_MOTOR_INDEX 2
57
58 // Encoder
59 #define CONFIG_ENCODER_2WD
60
61 #define ENC_LEFT_PIN_A 2
62 #define ENC_LEFT_PIN_B 2
63 #define ENC_RIGHT_PIN_A 3
64 #define ENC_RIGHT_PIN_B 3
65
66 // Count Per Turn of Encoder
67 const long ENCODER_CPT_GEARDED = 80;
68

```

```

69 // IMU
70 #define CONFIG_IMU_BNO055
71
72 #define CONFIG_ULTRASONIC_HC_SR04
73
74 #define ULTRASONIC_TRIGGER_PIN 4
75 #define ULTRASONIC_ECHO_PIN 5
76 #define ULTRASONIC_MAX_DISTANCE 300
77
78 // Control
79 int PWMMAX = 255;
80 int PWMLMIN = -20;
81
82 int PWM_D2A_FACTOR = 49;
83
84 int CTRLLOOP_PERIOD = 100;
85
86 #endif // defined (CONFIG_PLATFORM_ETT)
87
88 // ----- Device code -----
89
90 // ----- Motor -----
91
92
93
94 #if defined (CONFIG_MOTOR_2WD_ADAFRUIT)
95
96 #include <Adafruit_MotorShield.h>
97 #include "utility/Adafruit_PWM_Servo_Driver.h"
98
99 Adafruit_MotorShield AFMS = Adafruit_MotorShield();
100 Adafruit_DCMotor *m_left = AFMS.getMotor(M_LEFT_MOTOR_INDEX);
101 Adafruit_DCMotor *m_right = AFMS.getMotor(M_RIGHT_MOTOR_INDEX);
102
103 void m_set_pwm( int v_left_new,
104                  int v_left_direction,
105                  int v_right_new,
106                  int v_right_direction) {
107     if (v_left_direction == 1 || v_left_direction == '+') {
108         m_left->setSpeed(v_left_new);
109         m_left->run(FORWARD);
110     } else if (v_left_direction == -1 || v_left_direction == '-'){
111         m_left->setSpeed(v_left_new);
112         m_left->run(BACKWARD);
113     } else {
114         m_left->run(RELEASE);
115     }
116 }
117
118 if (v_right_direction == 1 || v_right_direction == '+') {
119     m_right->setSpeed(v_right_new);
120     m_right->run(FORWARD);
121 } else if (v_right_direction == -1
122            || v_right_direction == 2
123            || v_right_direction == '-'){
124     m_right->setSpeed(v_right_new);
125     m_right->run(BACKWARD);
126 } else {
127     m_right->run(RELEASE);
128 }
129
130 }
131
132 void m_stop() {
133 // LEE Release may not equal to setSpeed(0)
134     m_left->run(RELEASE);
135     m_right->run(RELEASE);

```

```

136  }
137
138 void m_setup() {
139
140     AFMS.begin();
141 }
142
143 #endif // defined (CONFIG_MOTOR_2WD_ADAFRUIT)
144
145
146
147 // ----- Encoder -----
148
149 #if defined (CONFIG_ENCODER_2WD)
150
151 // Encoder
152
153 #include <Encoder.h>
154
155 Encoder EncL(ENC_LEFT_PIN_A, ENC_LEFT_PIN_B);
156 Encoder EncR(ENC_RIGHT_PIN_A, ENC_RIGHT_PIN_B);
157
158 long encoder_left_read() {
159     return EncL.read();
160 }
161
162 long encoder_right_read() {
163     return EncR.read();
164 }
165
166 void encoder_left_write(long val) {
167     EncL.write(val);
168 }
169
170 void encoder_right_write(long val) {
171     EncR.write(val);
172 }
173
174 void encoder_reset() {
175
176     EncL.write(0);
177     EncR.write(0);
178 }
179
180 float encoder_calculate-angular-speed(long delta_tick,
181 long delta_time){
182
183     return 1.0 * (delta_tick) / (delta_time / 1000.0)
184             * 2 * PI / ENCODER_CPT_GEARED;
185 }
186
187 #endif // defined (CONFIG_ENCODER_2WD)
188
189 // ----- IMU -----
190
191 #if defined (CONFIG_IMU_BNO055)
192
193 #include <Adafruit_Sensor.h>
194 #include <Adafruit_BNO055.h>
195
196 Adafruit_BNO055 imu_bno055 = Adafruit_BNO055();
197
198 imu::Vector<3> euler_init;
199
200 imu::Vector<3> euler;
201 imu::Vector<3> acc;
202 imu::Vector<3> gyro;
203

```

```

204 void imu_setup() {
205     imu_bno055.begin();
206     delay(1000);
207     euler_init = imu_bno055.getVector(
208         Adafruit_BNO055::VECTOR_EULER);
209 }
210
211 void imu_read() {
212     euler = imu_bno055.getVector(Adafruit_BNO055::VECTOR_EULER);
213     acc = imu_bno055.getVector(
214         Adafruit_BNO055::VECTOR_LINEARACCEL);
215     gyro = imu_bno055.getVector(
216         Adafruit_BNO055::VECTOR_GYROSCOPE);
217 }
218
219 float imu_get_theta() {
220     return -(euler.x() - euler_init.x()) * 3.14 / 180.0;
221 }
222
223 float imu_get_accy() {
224     return -acc.y();
225 }
226
227 float imu_get_omega() {
228     return gyro.z();
229 }
230
231 #endif
232
233 // ————— Ultrasonic sensor —————
234
235 #if defined (CONFIG_ULTRASONIC_HC_SR04)
236
237 #include <NewPing.h>
238
239 NewPing sonar(ULTRASONIC_TRIGGER_PIN,
240                 ULTRASONIC_ECHO_PIN, ULTRASONIC_MAX_DISTANCE);
241
242
243 // LEE 0702 to emliniate unexpected abnormal measurement
244 // We backup last measurement if current us is wrong,
245 // replace us = us_safety_p
246 int us_safety_p = 0;
247 float ultrasonic_get_delta_x() {
248
249     // get delta x from ultrasonic sensor
250     int us = sonar.ping();
251     float distance;
252
253     if(us == 0) {
254         //distance = 5;
255         us = us_safety_p;
256     } else {
257         // CAUTION: distance is in meter, not centimeter
258         distance = us / 57.0 / 100.0;
259     }
260     if (distance >= ULTRASONIC_MAX_DISTANCE){
261         return (float)ULTRASONIC_MAX_DISTANCE;
262     }
263 }
```

```

271     return distance;
272 }
273
274 #endif
275
276 // ----- controller parameter -----
277
278 // CAUTION: ctrl_loop_period is also accessed in interrupt
279 // CAUTION: ctrl_loop_period can not be
280 // changed at running time
281 // loop period is in ms
282 volatile long ctrl_loop_period = CTRLLOOP_PERIOD;
283
284
285 // PID controller parameter
286
287 float prefilter_co = 0.167;
288
289 float kp_left = 0.29;
290 float ki_left = 0.58;
291 float kd_left = 0;
292
293 float kp_right = 0.29;
294 float ki_right = 0.58;
295 float kd_right = 0;
296
297 float roll_off_co = 0.8; // 40/(s+40)
298
299 // ----- control global variables -----
300
301 // robot state
302
303 float imu_theta = 0;
304 float imu_accy = 0;
305 float imu_omega = 0;
306
307 float delta_x = 0;
308
309 // Current angular velocity
310 float wl = 0;
311 float wr = 0;
312
313 float wl_p = 0;
314 float wr_p = 0;
315
316 // Desired angular velocity
317 float wl_dsr = 0;
318 float wr_dsr = 0;
319
320 float wl_dsr_filtered = 0;
321 float wr_dsr_filtered = 0;
322
323 float wl_dsr_filtered_p = 0;
324 float wr_dsr_filtered_p = 0;
325
326 float err_wl = 0;
327 float err_wr = 0;
328
329 float err_wl_p = 0;
330 float err_wr_p = 0;
331
332 float err_wl_pp = 0;
333 float err_wr_pp = 0;
334
335 // controller output PWM
336 int pwml = 0;
337 int pwmr = 0;
338
339 int pwml_out = 0;
340 int pwmr_out = 0;

```

```

341
342 int pwml_out_p = 0;
343 int pwmr_out_p = 0;
344
345 float pwml_up;
346 float pwml_ui;
347 float pwml_ud;
348
349 float pwmr_up;
350 float pwmr_ui;
351 float pwmr_ud;
352
353
354 // _____ control utility function _____
355
356 void ctrl_reset() {
357
358     wl_dsr_filtered = 0;
359     wr_dsr_filtered = 0;
360
361     wl_dsr_filtered_p = 0;
362     wr_dsr_filtered_p = 0;
363
364     err_wl_p = 0;
365     err_wr_p = 0;
366
367     err_wl_pp = 0;
368     err_wr_pp = 0;
369
370     pwml_out_p = 0;
371     pwmr_out_p = 0;
372 }
373
374
375
376 //Determine direction and control saturation
377 void ctrl_set_pwm() {
378
379     int pwml_ctrl = pwml;
380     int pwmr_ctrl = pwmr;
381     int pwml_motor;
382     int pwmr_motor;
383     int right_dir = 0;
384     int left_dir = 0;
385
386     pwml_motor = pwml_ctrl;
387     if (pwml_ctrl > PWMMAX) {
388         pwml_motor = PWMMAX;
389     }
390     if (pwml_ctrl < PWMLMIN) {
391         pwml_motor = PWMLMIN;
392     }
393     pwmr_motor = pwmr_ctrl;
394     if (pwmr_ctrl > PWMMAX) {
395         pwmr_motor = PWMMAX;
396     }
397     if (pwmr_ctrl < PWMLMIN) {
398         pwmr_motor = PWMLMIN;
399     }
400
401     if (pwml_motor > 0) {
402         left_dir = '+';
403     } else if (pwml_motor < 0) {
404         pwml_motor = -pwml_motor;
405         left_dir = '-';
406     } else {
407         // note that set pwm as 0 is not release motor
408         left_dir = '+';
409     }
410

```

```

411     if (pwmr_motor > 0) {
412         right_dir = '+';
413     } else if (pwmr_motor < 0) {
414         pwmr_motor = -pwmr_motor;
415         right_dir = '-';
416     } else {
417         right_dir = '+';
418     }
419
420     // call lower level interface
421     m_set_pwm(pwml_motor, left_dir, pwmr_motor, right_dir);
422 }
423
424
425 void ctrl_get_theta_accx_omega() {
426
427     imu_read();
428     imu_theta = imu_get_theta();
429     imu_accy = imu_get_accy();
430     imu_omega = imu_get_omega();
431 }
432
433 void ctrl_get_delta_x() {
434
435     delta_x = ultrasonic_get_delta_x();
436
437 }
438
439 // compute angular velocity of each wheel
440 //through Encoder Measurement
441 void ctrl_get_current_wl_wr() {
442
443     long left = encoder_left_read();
444     long right = encoder_right_read();
445
446     wl = encoder_calculate_angular_speed(left, ctrl_loop_period);
447     wr = encoder_calculate_angular_speed(right, ctrl_loop_period);
448
449     // average filter
450     wl = (wl + wl_p) / 2;
451     wr = (wr + wr_p) / 2;
452
453     wl_p = wl;
454     wr_p = wr;
455
456     // reset encoder value,
457     // so that every time we get increment from last time
458     encoder_left_write(0);
459     encoder_right_write(0);
460
461 }
462
463
464 // PID controller
465 float ctrl_pid(float err, float err_sum, float err_p,
466                 float kp, float ki, float kd,
467                 float *up_out, float *ui_out, float *ud_out,
468                 float ts) {
469
470     float u = 0;
471     float up = kp * err;
472     float ui = ki * ts * err_sum;
473     float ud = kd * (err - err_p) / ts;
474
475     u = up + ui + ud;
476
477     if (up_out != NULL)
478         *up_out = up;

```

```

479     if(ui_out != NULL)
480         *ui_out = ui;
481     if(ud_out != NULL)
482         *ud_out = ud;
483
484     return u;
485 }
486
487 // PID controller Incremental Style
488 int ctrl_pid_inc(int u_p,
489                     float err, float err_p, float err_pp,
490                     float kp, float ki, float kd,
491                     float *up_out, float *ui_out, float *ud_out,
492                     float ts) {
493
494     float up = kp * (err - err_p);
495     float ui = ki * ts * err;
496     float ud = kd * ((err - err_p) - (err_p - err_pp)) / ts;
497
498     float delta_u = (up + ui + ud) * PWM_D2A_FACTOR;
499
500     int u = u_p + (int)delta_u;
501
502     if(up_out != NULL)
503         *up_out = up;
504     if(ui_out != NULL)
505         *ui_out = ui;
506     if(ud_out != NULL)
507         *ud_out = ud;
508
509     return (int)u;
510 }
511
512 // ----- control inner loop , i.e. (wL, wR) control -----
513
514 // If error is smaller than the threshold , then set error to 0
515 float ctrl_deadzone_threshold(float err_in) {
516
517     float err_out = err_in;
518     if (abs(err_in) < deadzone_threshold) {
519         err_out = 0;
520     }
521     return err_out;
522 }
523
524 // Saturate the control variable change , i.e. we do not allow
525 // rapid change of PWM between interations.
526 int ctrl_deadzone_saturation(int u_in, int u_in_p) {
527
528     int u_out = u_in;
529     if (abs(u_in - u_in_p) > deadzone_saturation) {
530         u_out = (int)(u_in + deadzone_saturation);
531     }
532     return u_out;
533 }
534
535
536 // roll off for controller output (plant (motor shield ) input)
537 // u_rf = (1-rf_coeff) u_rf_p + rf.coeff * u_in
538 float ctrl_output_rolloff (int u_rf_p, int u_in){
539     float u_rf = (1 - roll_off_co)* u_rf_p + roll_off_co * u_in;
540     return u_rf;
541 }
542
543
544
545 // Update desired angular velocity and use PID controller
546 // Thus control inner loop (angular velocity )

```

```

547
548     void ctrl_inner_loop() {
549
550         if (ctrl_mode == 0) {
551
552             // controller mode is open loop
553             ctrl_set_pwm();
554             return;
555         }
556         // restriction desire speed to positive
557         // to prevent unpredictable error
558         int stop_action_level = 0;
559         if (wl_dsr <= 0.02 || wr_dsr <= 0.02) {
560             if(wl_dsr<0 || wr_dsr <0 ){
561                 stop_action_level = 2;
562             } else{
563                 stop_action_level = 1;
564             }
565
566
567             wl_dsr = 0;
568             wr_dsr = 0;
569         } else{
570             stop_action_level = 0;
571         }
572
573         wl_dsr_filtered = (1 - prefilter_co) * wl_dsr_filtered_p
574             + prefilter_co * wl_dsr;
575         wr_dsr_filtered = (1 - prefilter_co) * wr_dsr_filtered_p
576             + prefilter_co * wr_dsr;
577
578         //calculate error between measured output
579         //and desire value
580         err_wl = wl_dsr_filtered - wl;
581         err_wr = wr_dsr_filtered - wr;
582
583
584         // PID Inner loop Controller
585
586         pwml_out = ctrl_pid_inc(pwml_out_p,
587             err_wl, err_wl_p, err_wl_pp,
588             kp_left, ki_left, kd_left,
589             &pwml_up, &pwml_ui, &pwml_ud,
590             (float)ctrl_loop_period / 1000.0);
591         pwmr_out = ctrl_pid_inc(pwmr_out_p,
592             err_wr, err_wr_p, err_wr_pp,
593             kp_right, ki_right, kd_right,
594             &pwmr_up, &pwmr_ui, &pwmr_ud,
595             (float)ctrl_loop_period / 1000.0);
596
597
598         pwml = (int) ctrl_output_rolloff (pwml_out_p, pwml_out);
599         pwmr = (int) ctrl_output_rolloff (pwmr_out_p, pwmr_out);
600
601         if(stop_action_level >0 ){
602             if(stop_action_level<= 1){
603                 //release motor
604                 pwmr = 0;
605                 pwml = 0;
606             } else{
607                 //take brake action
608                 pwmr = -5;
609                 pwml = -5;
610             }
611         }
612
613     }
614

```

```

615  }
616
617
618 void ctrl_inner_loop_update() {
619     // Iteration
620     pwml_out.p = pwml_out;
621     pwmr_out.p = pwmr_out;
622     err_wl_p = err_wl;
623     err_wr_p = err_wr;
624     err_wl_pp = err_wl_p;
625     err_wr_pp = err_wr_p;
626     wl_dsr_filtered_p = wl_dsr_filtered;
627     wr_dsr_filtered_p = wr_dsr_filtered;
628
629
630 }
631
632
633 // ----- control interface -----
634
635 void ctrl_set_open_loop(int pwml_new, int pwmr_new) {
636
637     pwml = pwml_new;
638     pwmr = pwmr_new;
639
640     ctrl_mode = 0;
641 }
642
643 void ctrl_set_wl_wr(float wl_dsr_new, float wr_dsr_new) {
644
645     wl_dsr = wl_dsr_new;
646     wr_dsr = wr_dsr_new;
647
648     ctrl_mode = 1;
649 }
650
651
652
653
654
655
656
657
658 // ----- serial protocol -----
659
660 #define SERIAL_STATE_INIT      0
661 #define SERIAL_STATE_MAGIC1    1
662 #define SERIAL_STATE_MAGIC2    2
663 #define SERIAL_STATE_PROTO    3
664
665 #define SERIAL_MAGIC_1 'A'
666 #define SERIAL_MAGIC_2 'F'
667
668 // command sent from HLC to LLC
669
670 #define OPCODE_OPEN_LOOP        0x00
671
672 #define OPCODE_CTRL_WL_WR      0x10
673
674 //#define OPCODE_PAN_TILT       0x20
675 //
676 //#define OPCODE_IMU_CONFIG     0x40
677 //#define OPCODE_IMU_RESET      0x41
678
679
680
681 #define OPCODE_SETUP            0xF0
682 #define OPCODE_START           0xF1
683 #define OPCODE_STOP             0xF2
684 #define OPCODE_DEBUG_ENABLE     0xF3

```

```

685 #define OPCODE_DEBUG_DISABLE           0xF4
686
687
688 // state report from LLC to HLC
689
690 #define OPCODE_VEHICLE_STATUS         0xE0
691 #define OPCODE_CTRLSTATUS_DEBUG        0xE1
692
693
694 // _____ serial utility function _____
695
696 // CAUTION: this is blocking write!!!
697
698 void serial_put_char(byte c) {
699     while (Serial.write(c) <= 0)
700         ;
701 }
702
703 void serial_send_header(byte len, byte opcode) {
704     serial_put_char(SERIAL_MAGIC_1);
705     serial_put_char(SERIAL_MAGIC_2);
706     serial_put_char(len);
707     serial_put_char(opcode);
708     Serial.flush();
709 }
710
711 void serial_send_int(int i) {
712     byte *cp = (byte *) &i;
713
714     byte c0 = cp[0];
715     byte c1 = cp[1];
716
717     serial_put_char(c0);
718     serial_put_char(c1);
719     Serial.flush();
720 }
721
722 void serial_send_long(long l) {
723     byte *cp = (byte *) &l;
724
725     // byte c0 = (byte)(l & 0x000000FF);
726     // byte c1 = (byte)((l & 0x0000FF00) >> 8);
727     // byte c2 = (byte)((l & 0x00FF0000) >> 16);
728     // byte c3 = (byte)((l & 0xFF000000) >> 24);
729
730     byte c0 = cp[0];
731     byte c1 = cp[1];
732     byte c2 = cp[2];
733     byte c3 = cp[3];
734
735     serial_put_char(c0);
736     serial_put_char(c1);
737     serial_put_char(c2);
738     serial_put_char(c3);
739     Serial.flush();
740 }
741
742 void serial_send_float(float f) {
743     byte *cp = (byte *) &f;
744
745     byte c0 = cp[0];

```

```

752     byte c1 = cp[1];
753     byte c2 = cp[2];
754     byte c3 = cp[3];
755
756     serial_putchar(c0);
757     serial_putchar(c1);
758     serial_putchar(c2);
759     serial_putchar(c3);
760     Serial.flush();
761 }
762
763 // CAUTION: this is blocking read !!!
764
765 int serial_get_char() {
766
767     while (Serial.available() <= 0)
768         ;
769     return (char) Serial.read();
770
771 }
772 int serial_get_int() {
773
774     char bytarray[2];
775
776     bytarray[0] = serial_get_char();
777     bytarray[1] = serial_get_char();
778
779     return *((int *) bytarray);
780 }
781
782 long serial_get_long() {
783
784     char bytarray[4];
785
786     bytarray[0] = serial_get_char();
787     bytarray[1] = serial_get_char();
788     bytarray[2] = serial_get_char();
789     bytarray[3] = serial_get_char();
790
791     return *((long *) bytarray);
792 }
793
794 float serial_get_float() {
795
796     char bytarray[4];
797
798     bytarray[0] = serial_get_char();
799     bytarray[1] = serial_get_char();
800     bytarray[2] = serial_get_char();
801     bytarray[3] = serial_get_char();
802
803     return *((float *) bytarray);
804 }
805
806
807 int serial_state = SERIAL_STATE_INIT;
808 int serial_length = 0;
809
810 // — dispatch serial message according to the opcode —
811
812
813 void serial_parse_command(int opcode) {
814
815     switch (opcode) {
816
817         case OPCODE_OPEN_LOOP: {
818

```

```

819         // If direction variable is + ,
820         //then it means rotating forward;
821         // If direction variable is - ,
822         //then it means rotating backward;
823         // All other direction value is stop
824         int pwm_left_dsr = serial_get_int();
825         int pwm_right_dsr = serial_get_int();
826         if (motor_power_state == 0 ||
827             motor_power_state == 1
828             && ctrl_mode == 0)
829             ctrl_set_open_loop(pwm_left_dsr , pwm_right_dsr);
830
831         break;
832     }
833
834     case OPCODE_CTRL_WLWR: {
835
836         float wl_dsr_new = serial_get_float();
837         float wr_dsr_new = serial_get_float();
838
839         if (motor_power_state == 0 ||
840             motor_power_state == 1
841             && ctrl_mode == 1)
842             ctrl_set_wl_wr(wl_dsr_new , wr_dsr_new);
843         break;
844     }
845
846     case OPCODE_SETUP: {
847
848         // Setup only works if the system is stopped.
849         if (motor_power_state == 0) {
850
851             // The order must defines exatly
852             //the same as h2l setup funciton
853             prefilter_co = serial_get_float();
854             roll_off_co = serial_get_float();
855             kp_left = serial_get_float();
856             ki_left = serial_get_float();
857             kd_left = serial_get_float();
858             kp_right = serial_get_float();
859             ki_right = serial_get_float();
860             kd_right = serial_get_float();
861             deadzone_threshold = serial_get_float();
862             deadzone_saturation = serial_get_float();
863
864         }
865
866         break;
867     }
868     case OPCODE_START: {
869
870         motor_power_state = 1;
871
872         break;
873     }
874
875     case OPCODE_STOP: {
876
877         motor_power_state = 0;
878         m_stop();
879         ctrl_reset();
880
881         break;
882     }
883
884     case OPCODE_DEBUG_ENABLE: {
885

```

```

886         debug_mode = 1;
887
888     break;
889 }
890 case OPCODE_DEBUG_DISABLE: {
891
892     debug_mode = 0;
893
894     break;
895 }
896
897 //    case OPCODE_XXX: {
898 //
899 //        // add new function here
900 //        // use these comments as template
901 //        break;
902 //    }
903 default: {
904
905     break;
906 }
907
908 }
909
910 }
911
912 // serial state machine
913 void serial_state_machine_proceed(int c) {
914
915 //    Serial.print(c, HEX);
916 //    Serial.print(' ');
917
918 switch (serial_state) {
919     case SERIAL_STATE_INIT: {
920         if (c == SERIAL_MAGIC_1)
921             serial_state = SERIAL_STATE_MAGIC1;
922         else
923             serial_state = SERIAL_STATE_INIT;
924         break;
925     }
926     case SERIAL_STATE_MAGIC1: {
927         if (c == SERIAL_MAGIC_2)
928             serial_state = SERIAL_STATE_MAGIC2;
929         else
930             serial_state = SERIAL_STATE_INIT;
931         break;
932     }
933     case SERIAL_STATE_MAGIC2: {
934         serial_length = c;
935         serial_state = SERIAL_STATE_PROTO;
936         break;
937     }
938     case SERIAL_STATE_PROTO: {
939
940         // opcode = c
941         serial_parse_command(c);
942
943         serial_state = SERIAL_STATE_INIT;
944         break;
945     }
946     default: {
947         serial_state = SERIAL_STATE_INIT;
948         break;
949     }
950
951 }
952
953 }

```

```

954 // ----- serial sending -----
955
956 void serial_send_vehicle_status(long timestamp) {
957     serial_send_header(28, OPCODE_VEHICLE_STATUS);
958     // the first argument of send header is the
959     // total sum of payload below
960     serial_send_long(timestamp);
961
962     serial_send_float(imu_theta);
963     serial_send_float(imu_accy);
964     serial_send_float(imu_omega);
965     serial_send_float(wl);
966     serial_send_float(wr);
967     serial_send_float(delta_x);
968
969 }
970
971
972
973 void serial_send_ctrl_status_debug(long timestamp) {
974     int i;
975
976     serial_send_header(112, OPCODE_CTRLSTATUS_DEBUG);
977
978     serial_send_long(timestamp);
979
980     serial_send_float(wl_dsr);
981     serial_send_float(wr_dsr);
982     serial_send_float(wl_dsr_filtered);
983     serial_send_float(wr_dsr_filtered);
984
985     serial_send_int(pwml);
986     serial_send_int(pwmr);
987     serial_send_int(pwml_out);
988     serial_send_int(pwmr_out);
989     serial_send_int(pwml_out_p);
990     serial_send_int(pwmr_out_p);
991
992     serial_send_float(err_wl);
993     serial_send_float(err_wl_p);
994     serial_send_float(err_wl_pp);
995     serial_send_float(err_wr);
996     serial_send_float(err_wr_p);
997     serial_send_float(err_wr_pp);
998
999
1000    serial_send_float(pwml_up);
1001    serial_send_float(pwml_ui);
1002    serial_send_float(pwml_ud);
1003    serial_send_float(pwmr_up);
1004    serial_send_float(pwmr_ui);
1005    serial_send_float(pwmr_ud);
1006
1007    for(i = 0; i < 8; i++) {
1008        serial_send_long(timestamps[i]);
1009
1010    }
1011
1012 }
1013
1014
1015
1016
1017
1018 // ----- interrupt -----
1019

```

```

1020 // Timer interrupt counter.
1021 // It is reset when new loop period comes.
1022 int timer_counter = 0;
1023
1024 // Timer flag for the controller loop
1025 volatile int timer_inner_loop_flag = 0;
1026
1027 // Update Encoder Register in interrupt
1028 // when interrupt happens set Flag_TimerUpdate
1029 // Remember to reset Encoder if Encoder CPT is high
1030 // then you can use Enc*Ticks
1031 // as counter in one sampling period
1032 // DistMax=Longmax/Enc_CPT_MotorShaft/GearboxRatio
1033 //*(2*pi*radius)
1034
1035 // CAUTION: This function is called in interrupt!!!
1036 // Must make it short!!!
1037 void timer_update() {
1038
1039     timer_counter++;
1040
1041     if (timer_counter >= ctrl_loop_period) {
1042         timer_inner_loop_flag = 1;
1043         timer_counter = 0;
1044     }
1045
1046 }
1047
1048 // ----- setup -----
1049
1050 // forward declaration
1051 void timer_update();
1052
1053 void setup() {
1054
1055     Serial.begin(115200); //Serial Setup
1056
1057     Timer1.initialize(1000); // NOTE: period is in us
1058     // LEE Note may change to 10ms for ETT
1059     Timer1.attachInterrupt(timer_update);
1060
1061     // motor setup
1062     m_setup();
1063
1064     imu_setup();
1065     // in case of unexpected start state, reset everything
1066     m_stop();
1067     encoder_reset();
1068     ctrl_reset();
1069
1070 }
1071
1072 // ----- loop -----
1073
1074 unsigned long loop_count = 0;
1075
1076 void loop() {
1077
1078     // Run Serial State Machine
1079     // CAUTION: Never block or delay or spend too much time
1080     if (Serial.available() > 0) {
1081         int c = Serial.read();
1082         serial_state_machine_proceed(c);
1083     }
1084
1085     if (timer_inner_loop_flag > 0) {
1086

```

```

1087     loop_count++;
1088
1089     unsigned long timestamp = millis();
1090     timestamps[0] = millis() - timestamp;
1091
1092     // sensor data gathering
1093     ctrl_get_current_wl_wr(); // less than 1 ms
1094     timestamps[1] = millis() - timestamp;
1095     ctrl_get_theta_accx_omega(); // roughly 5 ~ 6 ms
1096     //(I2C bus needs time)
1097     timestamps[2] = millis() - timestamp;
1098
1099     ctrl_get_delta_x();
1100     timestamps[3] = millis() - timestamp;
1101
1102     serial_send_vehicle_status(timestamp); // at most 1 ms
1103     timestamps[4] = millis() - timestamp;
1104
1105     if (motor_power_state > 0) {
1106
1107         // Run controller.
1108         ctrl_inner_loop(); // less than 1 ms
1109         timestamps[5] = millis() - timestamp;
1110
1111         // Set controller output to motor.
1112         ctrl_set_pwm();
1113         // roughly 3 ~ 4 ms on Adafruit motor shield,
1114         timestamps[6] = millis() - timestamp;
1115
1116         serial_send_ctrl_status_debug(timestamp);
1117         // roughly 2 ~ 3 ms
1118         ctrl_inner_loop_update();
1119         timestamps[7] = millis() - timestamp;
1120
1121     }
1122     timer_inner_loop_flag = 0;
1123 }
1124
1125
1126 }
```