Modeling and Control for Vision Based Rear Wheel Drive Robot

and Solving Indoor SLAM Problem Using LIDAR

by

Xianglong Lu

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

Approved July 2016 by the
Graduate Supervisory Committee:

Armando A. Rodriguez, Chair
Spring Berman
Panagiotis Artemiadis

ARIZONA STATE UNIVERSITY

August 2016

ABSTRACT

To achieve 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, control objectives for rear-wheel drive ground vehicles. Toward this ambitious goal, several critical objectives are addressed. One central objective of the thesis was to show how to build low-cost multi-capability robot platform that can be used for conducting *FAME* research.

A TFC-KIT car chassis was augmented to provide a suite of substantive capabilities. The augmented vehicle (FreeSLAM) costs less than \$500 but offers the capability of commercially available vehicles costing over \$2000.

All demonstrations presented involve rear-wheel drive FreeSLAM robot. The following summarizes the key hardware demonstrations presented and analyzed: (1) Cruise (v, θ) control along a line, (2) Cruise (v, θ) control along a curve, (3) Planar (x, y) Cartesian Stabilization for rear wheel drive vehicle, (4) Finish the track with camera pan tilt structure in minimum time, (5) Finish the track without camera pan tilt structure in minimum time, (6) Vision based tracking performance with different cruise speed, (7) Vision based tracking performance with different camera fixed look-ahead distance, (8) Vision based tracking performance with different delay from vision subsystem, (9) Manually remote controlled robot to perform indoor SLAM, (10) Autonomously line guided robot to perform indoor SLAM.

For most cases, hardware data is compared with, and corroborated by, model-based simulation data. In short, the thesis uses low-cost self-designed rear-wheel drive robot to demonstrate many capabilities that are critical in order to reach the longer-term *FAME* goal.

Dedicated to my parents

ACKNOWLEDGMENTS

First, and most of all, I would like to thank Dr. Armando Antonio Rodriguez, for his expertise, assistance, guidance, and patience throughout the process of writing this thesis. Without his help and inspiration, this paper would not have been possible. I would like to thank my committee members, Dr. Panagiotis Artemiadis and Dr. Spring Berman, for their support, suggestions, and encouragement. Besides, I take this opportunity to express my gratitude to all of the EE and CS faculty members for their help and support.

Secondly, I also thank my parents for their encouragement, support and attention. I can not finish this program without their support.

Last but not least, I would like to take this opportunity to extend my sincere gratitude to my partners who supported me through this venture, especially, Zhichao Li, Jesus Aldaco, Venkatraman Renganathan and Duo Lv. Their great help and inspiration, have been sincerely appreciated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION AND OVERVIEW OF WORK

1.1 Introduction and Motivation

In recent years, with the improvement of economy and society, road capacity and traffic safety are becoming serious problems. Heavy driving work and fatigue driving are two key reasons causing traffic accidents. In this case, how to improve traffic safety has become a fatal social issue. These problems have motivated new researches and applications, for example, the self-driving vehicles, which can achieve better road capacity and safer driving by using control and SLAM algorithms, etc.

As the evolution of electromechanical and computing technologies continue to accelerate, the possible applications continue to grow. This accelerated growth is observed within the robotics research. New technologies (e.g. Arduino, Raspberry Pi with compatible interfaces, software and actuators/sensors) now permit young hobbyists and researchers to perform very complicated tasks - tasks that would have required great hardware/programming expertise just a few years ago. Within this thesis, current off-the-shelf technologies (e.g. Arduino, Raspberry Pi, commercially available chassis kit) are exploited to develop low-cost ground vehicles that can be used for multi-vehicle robotics research. Short-term, the goal is to develop several low cost ground vehicle platforms that can be used for multi-vehicle robotics research. This goal is intended as a first step toward the longer-term goal of achieving a fleet of *Flexible Autonomous Machines operating in an uncertain Environment (FAME)*. Such a fleet can involve multiple ground and air vehicles that work collaboratively to accomplish coordinated tasks. 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 addresses modeling, design and control issues associated with ground-based robotic vehicle. Particularly, LIDAR was used to implement Simultaneous Localization And Mapping ($SLAM$) algorithm (hector mapping) to perform indoor robot localization and mapping. Toward the longer-term $FAME$ goal, several critical objectives are addressed. One central objective of the thesis was to show how to use low-cost chassis kit and convert it into somewhat "intelligent" multi-capability robotic platform that can be used for conducting $FAME$ research. This thesis focuses on a rear-wheel drive robot (called FreeSLAM). Kinematic and dynamical models are examined. Rear-wheel drive means that the speed of the rear wheels are the same and controlled by a single dc motor (in our case two motors are treated identically and issued same voltage command). 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. 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 control, control along a line/path, finish the track in minimum time, etc). Once the basic control issues are addressed, the vision-based lateral model is explained in detail. According to this model, three key parameters will greatly influence the tracking performance: robot cruise speed, fixed look-ahead distance and delay from vision subsystem. Each case above was well tested and discussed. Hector Mapping, which is one of the popular SLAM approaches to solve indoor SLAM problem was well discussed and implemented. Extended Kalman Filter is optimal filter to estimate the robotic pose (X , Y position and orientation) under Gaussian noise. Once we have those information, we can represent the 2D grid map of the unknown environment using laser scan data.

To draw a brief conclusion, this chapter attempts to provide a fairly comprehensive literature survey - one that summarizes relevant literature and how it has been used. This is then used as the basis for outlining the central contributions of the thesis.

1.2 Literature Servery: Robotics-controls and SLAM approaches

In an effort to emphasize on the state of ground robotics vehicle modeling, hardware, design, control and SLAM basic approaches, the following topically literature survey is offered. In short, the following works are most relevant for the developments within this thesis:

- Tricycle-model vehicle steering control problem (presenting kinematic model) work within: [2]

- Rear-wheel drive vehicle modeling work within [3], (presenting dynamical model), addressing the affects of robot cruise speed V_x, vision subsystem look-ahead distance L and delay of vision subsystem T_d. Nominal parameters for the simulation in Chapter 3 was taken from [3]

- Camera based vision-based line/curve following work within both [3] and [14]

- Robot Operating System (ROS) architecture (ROS nodes, publisher and subscriber protocols and catkin working space etc.) within: [12]

- Extended Kalman Filter algorithm (EKF, implemented for filter the Gaussian noise for depth sensors) within: [1]

- Rao-Blackwellized Particle Filters algorithm (PF, for reducing non-Gaussian noise in SLAM problems) within: [6]

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

- Rear-Wheel Drive Robot Modeling

Within this thesis, rear-wheel drive ground vehicle(Self-Designed FreeSLAM Robot) represents a central focus of the work. Here, rear-wheel drive means the car's driven wheels - i.e., the wheels that receive power from the engine (DC motors) - are the ones in back, and those two front wheels, are responsible for steering only. As such, two rear wheels are of the same speed. Nominally, we assume that the motors are identical. The motor inputs (vehicle controls) are voltages. Speed of the robot v_x depends on the applied voltages and the steering servo controls the direction (ψ in the following chapters).

Kinematic Model

A kinematic model for rear-wheel drive robot (ignoring dynamic mass-inertia effects) is presented within [3]. Within this kinematic model, it is assumed that the translational and angular velocities (v_x,$\dot{\psi}$) of the robot are realized instantaneously. Of course, it is not real 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 requires infinite acceleration and force. In short, the kinematic model is less accurate than a dynamic model (which includes acceleration constraining mass-inertia effects).

Dynamic Model

A dynamic model can take the torques applied to the robot wheels as inputs (controls) to the system. This is done within [3]. The model presents within these works incorporates dynamic (acceleration constraining) mass-inertia effects as well as friction, wheel slippage etc. Given this, it is obvious that a dynamic model generally gives a much more accurate model of the vehicle robot.

Simultaneous localization and mapping

In robotic mapping, simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. While this initially appears to be a chicken-and-egg problem there are several algorithms known for solving it, at least approximately, in tractable time for certain environments. Popular approximate solution methods include the particle filter and extended Kalman filter. SLAM algorithms are tailored to the available resources, hence not aimed at perfection, but at operational compliance. Published approaches are employed in self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers, newly emerging domestic robots and even inside the human body.

Extended Kalman Filter (EKF)

In robotics, EKF SLAM is a class of algorithms which utilizes the extended Kalman filter (EKF) for simultaneous localization and mapping (SLAM). Typically, EKF SLAM algorithms are feature based, and use the maximum likelihood algorithm for data association. For the past decade, the EKF SLAM has not been the major method for SLAM, until the introduction of FastSLAM.[1]

Associated with the EKF is the Gaussian noise assumption, which significantly impairs EKF SLAM's ability to deal with uncertainty. With greater amount of uncertainty in the posterior, the linearization in the EKF fails. In this thesis, EKF is used to estimate the current states of the vehicle robot, which are X, Y (current position) and (current orientation), those states are estimated and used to design controllers and solve localization problems.

Rao-Blackwellized Particle Filters algorithm (PF algorithm)

Recently Rao-Blackwellized particle filters have been introduced as effective means to solve the simultaneous localization and mapping (SLAM) problem. This approach uses a particle filter in which each particle carries an individual map of the environment. Accordingly, a key question is how to reduce the number of particles. We present adaptive techniques to reduce the number of particles in a Rao-Blackwellized particle filter for learning grid maps. We propose an approach to compute an accurate proposal distribution taking into account not only the movement of the robot but also the most recent observation. This drastically decrease the uncertainty about the robot's pose in the prediction step of the filter. Furthermore, we apply an approach to selectively carry out re-sampling operations which seriously reduces the problem of particle depletion.

1.3 Frameworks

ROS(The Robot Operating System)

Robot Operating System (ROS) is a collection of software frameworks for robot software development, (see also Robotics middle-ware) providing operating system-like functionality on a heterogeneous computer cluster. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. Despite the importance of reactivity and low latency in robot control, ROS, itself, is not a Real-time OS, though it is possible to integrate ROS with real-time code.

Both the language-independent tools and the main client libraries (C++, Python, LISP) are released under the terms of the BSD license, and as such are open source software and free for both commercial and research use. The majority of other packages are licensed under a variety of open source licenses. These other packages implement commonly used functionality and applications such as hardware drivers, robot models, data types, planning, perception, simultaneous localization and mapping, simulation tools, and other algorithms.

The main ROS client libraries (C++, Python, LISP) are geared toward a Unix-like system, primarily because of their dependence on large collections of open-source software dependencies. For these client libraries, Ubuntu Linux is listed as ”Supported” while other variants such as Fedora Linux, Mac OS X, and Microsoft Windows are designated ”Experimental” and are supported by the community. The native Java ROS client library, rosjava, however, does not share these limitations and has enabled ROS-based software to be written for the Android OS. rosjava has also enabled ROS to be integrated into an officially-supported MATLAB toolbox which can be used on Linux, Mac OS X, and Microsoft Windows. A JavaScript client library, roslibjs has also been developed which enables integration of software into a ROS system via any standards-compliant web browser.

ROS used in this thesis is the latest version: ROS JADE.

Self-Designed Rear Wheel Drive SLAM Robot Enhancement

As discussed above, and within the thesis, the rear wheel drive vehicle platform is augmented with the following:

(1) *Visualization of Full-Loaded(Enhanced) Real Rheel Drive FreeSLAM Robot - Vision Mode*

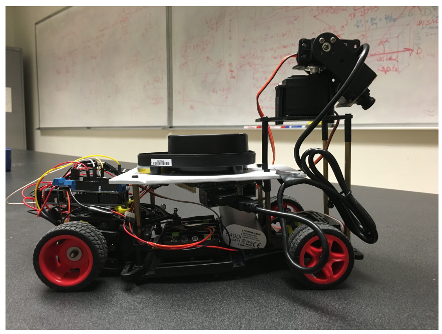

Figure 1.1: Side View of Self-Designed FreeSLAM Robot

FreeSLAM Robot: Vision Mode

Rear Wheel Drive, UAV Tracking, Camera vision sensing, Depth sensors

(2) *Visualization of Full-Loaded(Enhanced) Real Rheel Drive FreeSLAM Robot - Scan Mode*

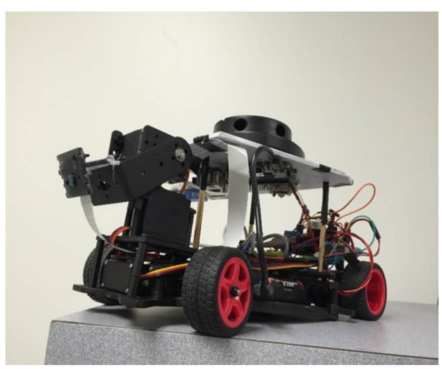

Figure 1.2: FreeSLAM Robot Scan Mode

FreeSLAM Robot : Scan Mode

High Accuracy LIDAR Sensing, Fixed Pan Servo, Less Speed for not Losing Landmarks

(3) *Duo's Differential Drive Robot*

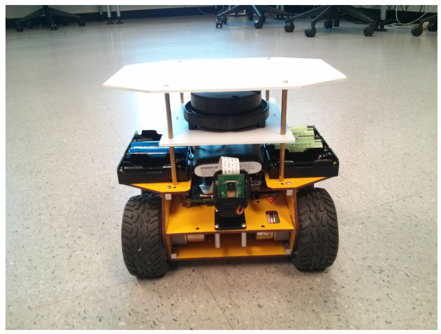

Figure 1.3: Duo's Differential Robot

(4) *360 Degree RP Lidar*

The RPLIDAR 360 Laser Scanner is a low cost 360 degree 2D scanner (LIDAR) solution. It preforms 360 degree laser scanning with more than 6 meters distance detection range. The produced 2D point cloud data can be used in mapping, localization (SLAM) and object/ environment modeling. RPLIDAR emits a modulated infrared laser signal and the laser signal is then reflected by the object to be detected. The returning signal is sampled by vision acquisition in RPLIDAR and the DSP embedded in RPLIDAR starts processing the sample data, output distance value and angle value between the object and the RPLIDAR. Through processing the sample data is output through a communication interface.

Figure 1.4: 360 Degree RP LiDAR

Description:

- 360 laser scanner development kit with omnidirectional laser scan
- High speed laser triangulation vision system
- Ideal sensor for robot localization mapping
- User configurable scan rate (rotation speed) via PWM signal

Features: Omnidirectional laser scan, User configurable scan rate via the motor PWM signal, Plug Play using included USB cable, No coding job required, SLAM ready, 5.5hz (2000 sample/sec), 6 meters measurement range, Obstacle avoidance, mapping localization, navigation sensor.

Specifications: Distance range: 0.2 - 6m, Angular range: 0°-360°, Distance resolution <0.5mm (1 percent of the distance), Angular resolution: $\leq 1°$, Sample duration: 0.5 milliseconds, Sample frequency: $\geq$2000Hz, Scan rate: 5.5Hz, M2.5 x 15mm standoffs.

Optical: Laser wavelength: 785 nanometer, Laser power: 3 milliwatt, Pulse length: 110 microsecond.

Applications:

- Robot localization mapping (SLAM)
- 3D modeling
- Obstacle avoidance and security
- Multitouch and human interaction

(5) *Adafruit 9DOF Inertial Measurement Unit (IMU)*

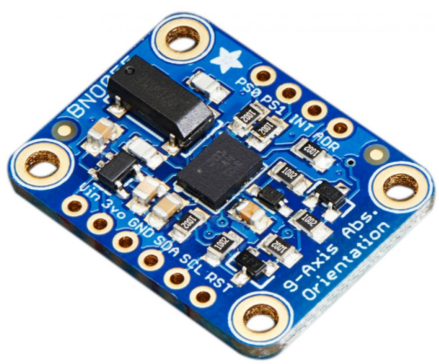

Figure 1.5: Adafruit 9DOF Inertial Measurement Unit (IMU)

Figure 1.5 is a visualization for the 9 DOF (degree of freedom) IMU we were using: BNO055. The BNO055 can output the following sensor data:
Absolute Orientation (100Hz), Angular Velocity Vector (100Hz), Acceleration Vector (100Hz), Magnetic Field Strength Vector (20Hz), Linear Acceleration Vector (100Hz), Gravity Vector (100Hz) and Temperature(1Hz). The gyroscope and accelerometer are the two sensors we have used the most.

(6) *an Arduino Uno open-source microcontroller development board* (16MHZ ATmega328 processor, 32KB Flash Memory, 14 digital I/O pins, 6 analog inputs, \$25,

see Figure 1.6) for both encoder-IMU-based speed (v, ω) or (v_x, δ_f) inner-loop control and encoder-IMU-ultrasound-based cruise-position-directional-separation outer-loop control

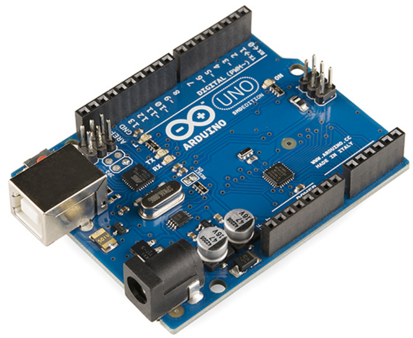

Figure 1.6: Arduino Uno Open-Source Microcontroller Development Board

(7) *an Arduino motor shield* (see Figure 1.7) for inner-loop motor PWM[1] speed control,

[1]PWM or *pulse width modulation* is a method for generating a desired dc voltage level from a larger positive dc reference voltage. The reference voltage is switched on and off via FETs to produce a high frequency PWM (or square-wave like) signal. The FET inputs are controlled to adjust the duty cycle of the PWM signal. When the PWM signal is low pass filtered, the desired dc voltage is obtained (with some ripple). When the motor shield drives a dc motor, the motor-load moment of inertia as well as the motor's armature inductance will provide sufficient low pass filtering so that the resulkting ripple is negligibly small. Given the above, complementary paired FETs can be used to produce negative dc voltages.

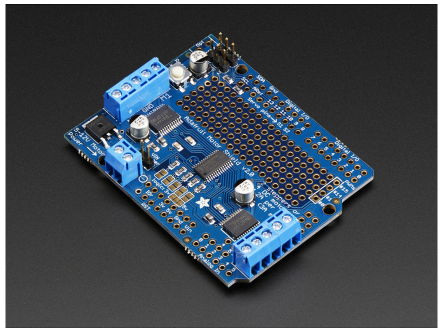

Figure 1.7: Adafruit Motor Shield for Arduino v2.3 - Provides PWM Signal to DC Motors

(8) *a Raspberry Pi III Model B single board computer* (A 1.2GHz 64-bit quad-core ARMv8 CPU, 802.11n Wireless LAN, Bluetooth 4.1, 1GB RAM 4 USB ports 40 GPIO pins, (like raspberry Pi II Model B), see Figure 1.8) for more demanding vision-based cruise-position-directional outer-loop control,

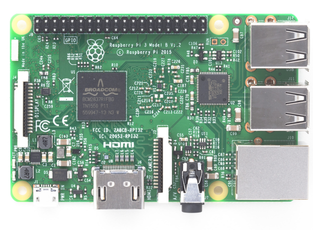

Figure 1.8: Raspberry Pi 3 Model B Open-Source Single Board Computer

(9) *Linux USB camera* (2592 $\times$ 1944 pixel or 5 MP static images; 1080p30 (30 fps), 720p60 and 640x480p60/90 MPEG-4 video, see Figure 1.9) for outer-loop cruise-position-directional control,

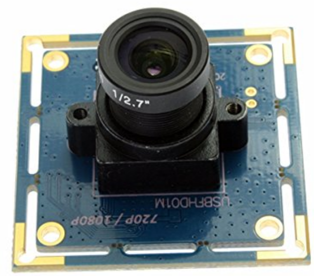

Figure 1.9: Raspberry Pi 5MP Camera Module

(10) *Wireless Communication between Raspberry Pi and PC*

FreeSLAM robot is able to establish wireless communication with host PC through ssh. This is done via WiFi - a wireless local area network based on the IEEE 802.11 (2.4, 5 GHz) standard. More precisely, PC can send commands to Pi and get data back wirelessly. see Figure 1.10) which serves as a transmitter on the robot.

Figure 1.10: EDIMAX WiFi Adapter - Enables Video Link from Robot to Central Laptop

Data will be sent to a remotely situated (< 30 m) TPLINK TL-WDR3500 wireless router (600 Mbps total bandwidth, 300 Mbps for 2.4GHz, 300 Mbps for 5GHz). The router transmits the radio signal to a wireless adapter on the nearby (< 30 m) laptop.

(11) *Mallofusa 2 DOF Pan Tilt with Mg995 Servos Sensor Mount* Each servo of this pan tilt has a scan range of 120 degrees with a 0.1 degree accuracy. This pan tilt is designed for line tracking, object tracking, video streaming, sensor fusion and extra.

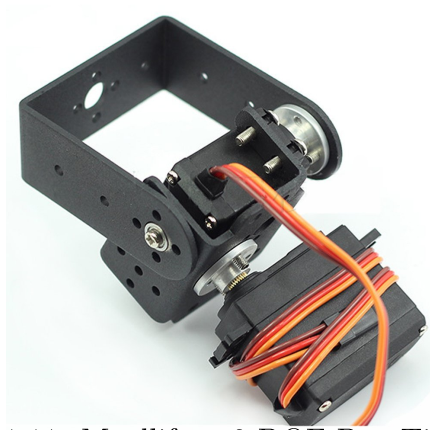

Figure 1.11: Moallifusa 2 DOF Pan Tilt Servos

(12) *USB to serial UART bridge*

SparkFun has a line of USB to serial UART bridge products designed to allow a user to communicate with a serial UART through a common USB port. It is harder to find computers with serial UART ports on them these days, but super common to find serial devices. Many of the official Arduino and clones share a common interface. This interface is essentially the 6 pin Single-In-Line (SIL), 0.1 pitch version of FTDIs TTL-232R cables.

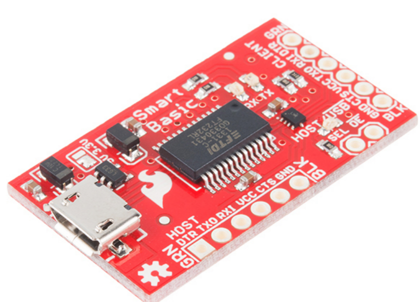

Figure 1.12: Spark Fun UART Chip

The key change from the FTDI cables to our Arduino compatible boards is that we swapped pin 6 from RTS to DTR. This change was required to match Arduinos method of resetting the ATmega328P using the DTR signal.

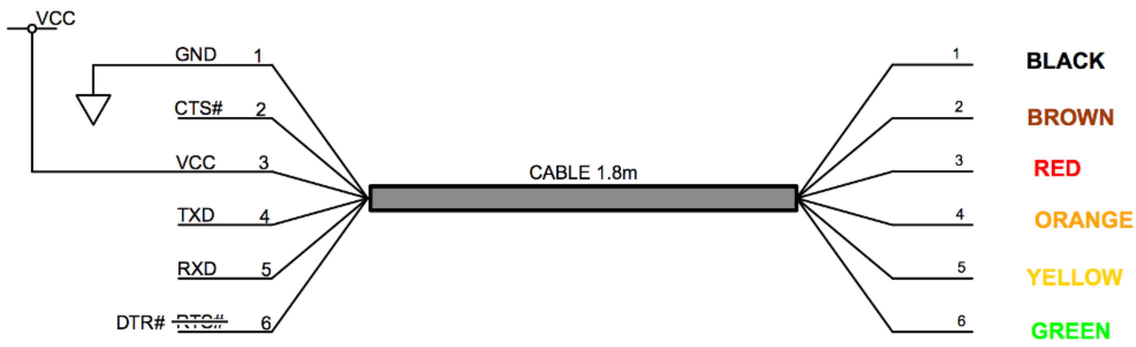

Figure 1.13: Spark Fun UART Pin Connections

Component	**Price**
Chassis and Motors	\$180
Futaba S3003 Servo	\$10
Arduino Uno	\$25
Adafruit Motor Shield	\$20
Raspberry Pi 2	\$40
WiFi adapter	\$25
Adafruit 9DOF IMU	\$20
Pi camera	\$20
Neato xv11 LIDAR	\$80
5V external battery for Raspberry Pi	\$20
Hitachi 18650 battery for motor	\$30
Total Price	**\$470**

Table 1.1: Bill of Material of FreeSLAM Robot

1.4 Organization of Thesis

The remainder of the thesis is organized as follows.

- Chapter 2 presents an overview for a general $FAME$ architecture describing candidate technologies (e.g. sensing, communications, computing, actuation).
- Chapter 3 describes modeling and control issues for rear-wheel drive (RWD) ground vehicles. The ideas presented here using an academic (numerical) system provide a foundation for the work in Chapter 4.
- Chapter 4 presents system-theoretic as well as hardware results for our FreeSLAM ground robotic vehicle. Many demonstrations are described. This chapter contains the main work that was conducted.
- Chapter 5 describes one of the most popular SLAM algorithm - Hector Mapping. Hector Mapping requires LIDAR scan data only to estimated robot pose. Extended Kalman Filter implementation (to reduce Gaussian Noise) is well discussed.
- Chapter 6 summarized another SLAM approach - gmapping. For gmapping, sensor fusion of LIDAR scan data and odometry data (IMU and encoders) is required. Partical Filter is introduced in the case that input and observation noise are not Gaussian distribution.
- Chapter 7 talks about general future works and researches.

1.5 Summary and Conclusions

In this chapter, we described a general (candidate) *FAME* architecture for a fleet of cooperating robotic vehicles. This self-designed rear-wheel drive robot - FreeSLAM robot can be a part of it. Besides, how we enhanced the robot is well addressed. In the following chapters, as we introduced before, both simulation and hardware implementation for modeling and controller design of the robot will be discussed in the following chapters.

Chapter 2

OVERVIEW OF GENERAL FAME ARCHITECTURE & C^4S REQUIREMENTS

2.1 Introduction and Overview

In this chapter, we describe a general architecture for our general *FAME* research. 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^4S 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[1]. 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.

[1]Here the term robotic vehicle can refer to a ground, air, space, sea or underwater vehicle.

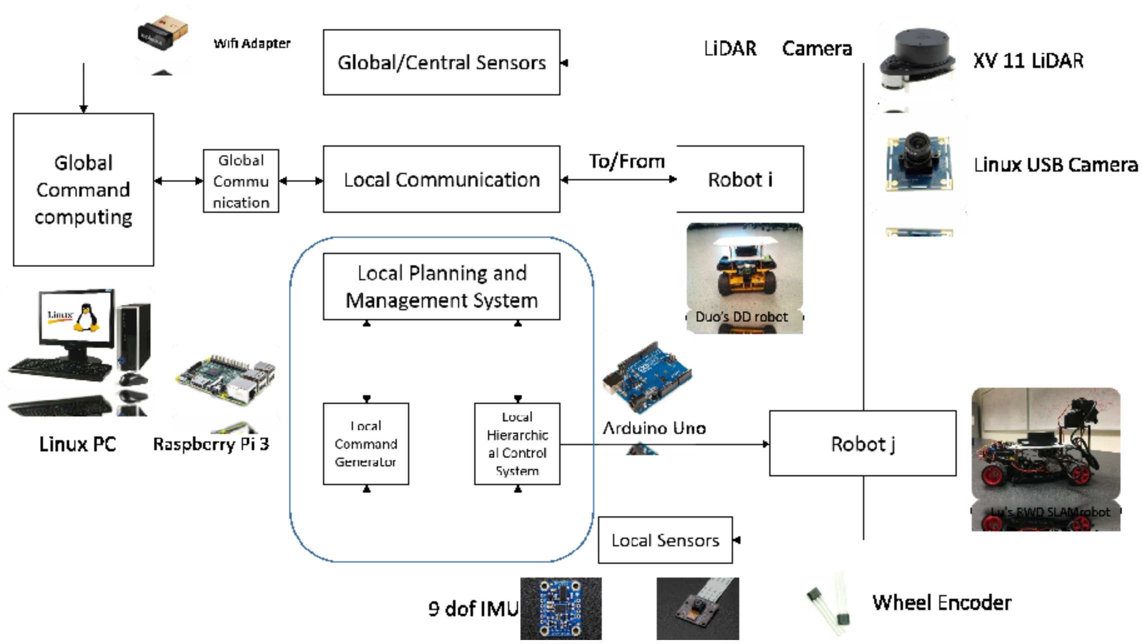

Figure 2.1: *FAME* Architecture to Accommodate of Fleet of Cooperating Vehicles

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

- **Global/Central Communications.** In order to communicate with members of the fleet, a suite of communication devices must be available to central command. Such devices can include (wideband) spread spectrum transmitters/receivers, WiFi/Bluetooth adapters, etc. Within this thesis, we use (wideband) spread spectrum transmitters/receivers and WiFi adapters.

- **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. rear-wheel drive robot - FreeSLAM robot).

- **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, micro-controller or suite of computers/micro-controllers. Within this thesis, we primarily exploit Arduino Uno micro-controller (16MHZ ATmega328 processor, 32KB Flash Memory, 14 digital I/O pins, 6 analog inputs, $25) [19]and Raspberry Pi II (900 MHz quad-core ARM Cortex-A7 CPU, 1GB SDRAM, 40 GPIO pins, camera interface, $35) [20] 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 Raspberry Pi II IDLE), 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 magnetic encoders(A3144 Hall effect sensor, VELLEMAN 8 mm $\times$ 3 mm magnet, 12 per wheel), IMUs to measure vehicle rotation (9DOF, Accelerometer $\pm$ 2,4,6,8,16g. Gyro $\pm$ $245, 500, 2000°/sec$. Compass $\pm$ 1.3 to $\pm$ 8.1 Gauss) [27], ultrasonic range sensors (40kHz, 0.02-3 m, approximately $\pm 8°$ directional), and Raspberry Pi cameras(2592 $\times$ 1944, 30 fps, 150 MPs, MPEG-4) [24]. 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) to send locally obtained Raspberry Pi camera video (2592 $\times$ 1944, 30 fps, 150 MPs, MPEG-4) [24] 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 self-designed rear-wheel drive FreeSLAM robot is used as a member of the fleet.

Chapter 3

VISION BASED COMPLETE LATERAL MODEL STUDIES AND SIMULATION

3.1 Introduction and Overview

Recent interest in self-driving car system and current advances in real-time image processing provide a suitable testbed for employing the visual information extracted from image sequences in the feedback loop of the control system.

Once including the vision part in the whole feedback control system, various strategies for controller design system are presented. We investigate the choice of the look-ahead distance L, which varies with the longitudinal velocity and is affected by the quality of the offset estimations. Several controller design techniques and closed loop simulations are presented.

The purpose of this chapter is to illustrate fundamental modeling and control design methods for a rear-wheel drive (RWD) robotic ground vehicle. This is achieved by presenting relevant model trade studies and then illustrating the design of an inner-loop (v_x, $\dot{\psi}$) speed and direction control law and associated trade-offs. Such a control law is generally the basis for any outer-loop control law.

3.2 Vision Based Complete Lateral Model Studies and Simulations

The dynamic model of the vehicle is described by a detailed 6-DOF nonlinear model. This model is too complex and not suitable for controller design. Due to the possibility decoupling of longitudinal and lateral dynamics, a linearized model of the lateral vehicle dynamics is used for controller design. Besides, closed loop simulations take into account the full nonlinear dynamic model of the vehicle.

3.2.1 Nonlinear Model

When physical parameters of the tires (tire pressure, road, tire surface condition) are fixed and cornering forces are determined solely by tire normal force, tire slip angle and tire slip ratio. In the simplified setting the tire normal force generated from the tire can be approximated by:

$$F_y = c\alpha \tag{3.1}$$

The quantity c characterizes the tire cornering capabilities and is referred to as $corneringstiffness$, and α is the tire slip angle between the orientation of the tire and its velocity. While c^* is the effective value of the cornering stiffness and μ is the road adhesiveness parameter. This relationship is captured by

$$c = \mu c^* \tag{3.2}$$

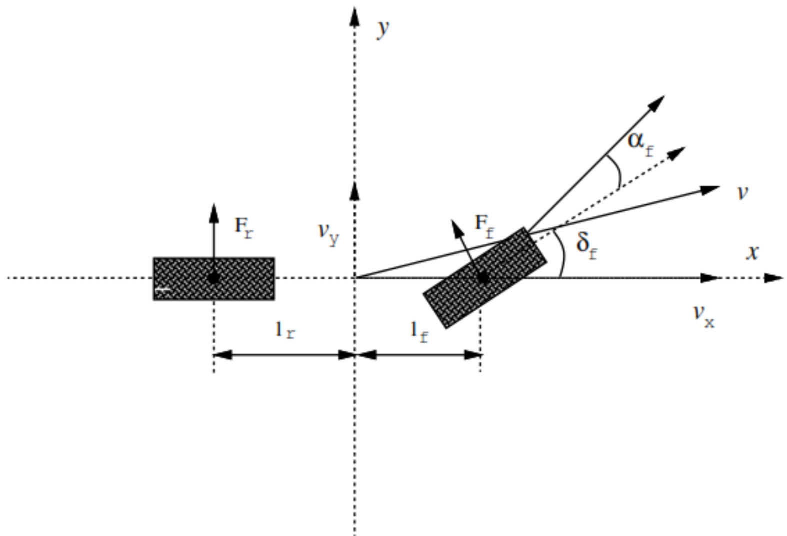

Figure 3.1: Kinematic Behavior of the Bicycle Model

The velocity $v = (v_x, v_y)$ expressed in inertial vehicle frame and the yaw rate of the vehicle $\dot{\psi}$ characterizes the motion of the vehicle. The forces acting on the front and rear wheels are F_f and F_r. Side slip angles are denoted α_f, α_r and those are the angles between the current steering angle and the vehicle's current orientation. Besides, the steering angle of the front wheel is δ_f, the distance of the axles to the center of the gravity of the vehicle are l_f and l_r.

The kinematic behavior of the vehicle which is shown above is approximated by the bicycle model, with two front and rear wheels lumped together. Lateral dynamics can be linearized by current longitudinal velocity. The net lateral force and the net torque acting on the center of the gravity of the vehicle are:

$$F = F_f + F_r \tag{3.3}$$

$$\tau = F_f l_f + F_r l_r \tag{3.4}$$

The variables and additional parameters in the model are:

v_x denotes longitudinal speed

α_f,α_r side slips angle between steering angle and front and rear tire velocities

vehicle yaw angle

δ_f front wheel steering angle

δ commanded steering angle

m total mass of the vehicle

I_ψ total inertia of the vehicle around centre of gravity

l_f, l_r distance of the front and rear axles from the CG l distance between the front and the rear axle $l_f + l_r$ $c_f.c_r$ cornering stiffness of the front and rear tires

Nominal Parameters

Here in the simulations, parameters are taken from [3].

The values of the parameters of the particular model used in simulations are: **m = 1573kg,** I_{psi} **=** $2753kgm_2$ **,** l_f **= 1.137m,** l_r **= 1.530m,** c_f **= 2x60000 N/rad,** c_r **= 2x50000 N/rad.** The cornering stiffness is doubled since the two tires are lumped together. The individual normal forces acting at the front and rear tires are:

$$F_f = c_f \alpha_f \tag{3.5}$$

$$F_r = c_r \alpha_r \tag{3.6}$$

where slide slip angles α_f and α_r between the steering angle and the tire velocity are:

$$\alpha_f = \delta - arctan(\frac{v_y + l_f\dot{\psi}}{v_x}) \approx \delta - \frac{v_y + l_f\dot{\psi}}{v_x} \tag{3.7}$$

$$\alpha_r = -arctan(\frac{v_y - l_r\dot{\psi}}{v_x}) \approx \frac{-v_y + l_r\dot{\psi}}{v_x} \tag{3.8}$$

The net lateral force F and the net torque τ at the center of gravity are:

$$F = ma = m({}_y + v_x\dot{\psi}) = F_f + F_r \tag{3.9}$$

$$\tau = I_\psi \ddot{\psi} = F_f l_f - F_r l_r \tag{3.10}$$

the lateral dynamics have the following form:

$$\begin{bmatrix} \dot{v}_y \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} -\frac{c_f + c_r}{m v_x} & -v_x + \frac{c_r l_r - c_f l_f}{m v_x} \\ \frac{-l_f c_f + l_r c_r}{I_\psi v_x} & -\frac{l_f^2 c_f + l_r c_r}{I_\psi v_x} \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{c_f}{m} \\ \frac{l_f c_f}{I_\psi} \end{bmatrix} \delta_f$$

3.3 Vision Dynamics

The equations capturing the evolution of the measurements extracted from images (Implementing OpenCV in Raspberry Pi Camera) are as follows:

$$\dot{y}_L = v\varepsilon_L - v_y - \dot{\psi}L \tag{3.11}$$

$$\dot{\varepsilon}_L = vK_L - \dot{\psi} \tag{3.12}$$

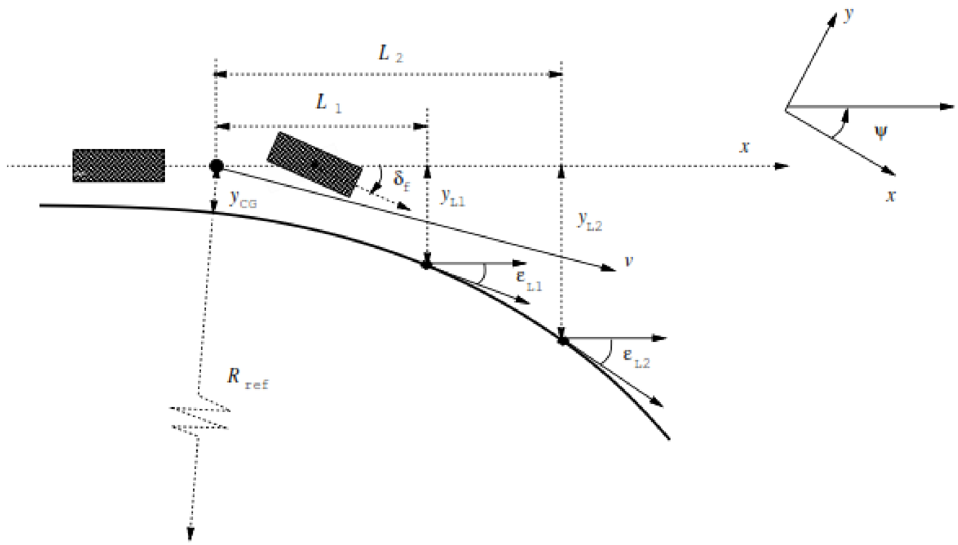

Figure 3.2: Visualization of Vision Dynamics

The vision system estimates the offset from the center line y_L and the angle between the road tangent and heading of the vehicle ε_L at some look-ahead distance L.

The additional parameters and measurements of the vision system are:

- y_L the offset from the center-line at the look-ahead distance
- ε_L the angle between the tangent to the road and the orientation of the vehicle with respect to the road
- L look ahead distance at which the measurements are taken
- K_L is the disturbance

3.4 Vision Subsystem Based Complete Lateral Model

Combining the vehicle lateral dynamics with the vision dynamics.

$$\dot{x} = Ax + Bu + E\omega \tag{3.13}$$

$$y = Cx + Du + F\omega \tag{3.14}$$

The state $x = [v_y, \dot{\psi}, y_L, \varepsilon_L]^T$ and control input $u = \delta_f$, and disturbance $\omega = K_L$. Here is the state space equations for the complete dynamic model:

$$\begin{bmatrix} \dot{v}_y \\ \ddot{\psi} \\ \dot{y}_L \\ \dot{\varepsilon}_L \end{bmatrix} = \begin{bmatrix} -\frac{c_f+c_r}{mv_x} & -v_x + \frac{c_r l_r - c_f l_f}{mv_x} & 0 & 0 \\ \frac{-l_f c_f + l_r c_r}{I_\psi v_x} & -\frac{l_f^2 c_f + l_r^2 c_r}{I_\psi v_x} & 0 & 0 \\ -1 & -L & 0 & v_x \\ 0 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \\ y_L \\ \varepsilon_L \end{bmatrix} + \begin{bmatrix} \frac{c_f}{m} \\ \frac{lfc_f}{I_\psi} \\ 0 \\ 0 \end{bmatrix} \delta_f + \begin{bmatrix} 0 \\ 0 \\ 0 \\ v_x \end{bmatrix} K_L$$

There are two subsystems in this whole complete model. The first one is the on-board vehicle sensors subsystem, where inertial sensors (9 DOF IMU and encoders) are used for measuring lateral acceleration $\ddot{y} = (\dot{v}_y + v_x\dot{\psi})$ and the yaw rate $\dot{\psi}$. Meanwhile, the vision subsystem estimates y_L and ε_L. The road curvature K_L is working as a exogenous disturbance signal.

The output equations have following form:

$$y = \begin{bmatrix} -\frac{c_f+c_r}{mv_x} & \frac{c_r l_r - c_f l_f}{mv_x} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \\ y_L \\ \varepsilon_L \end{bmatrix} + \begin{bmatrix} \frac{c_f}{m} \\ 0 \\ 0 \\ 0 \end{bmatrix} \delta_f$$

The block diagram of the overall camera vision based lateral system is showed in Figure 2.4.

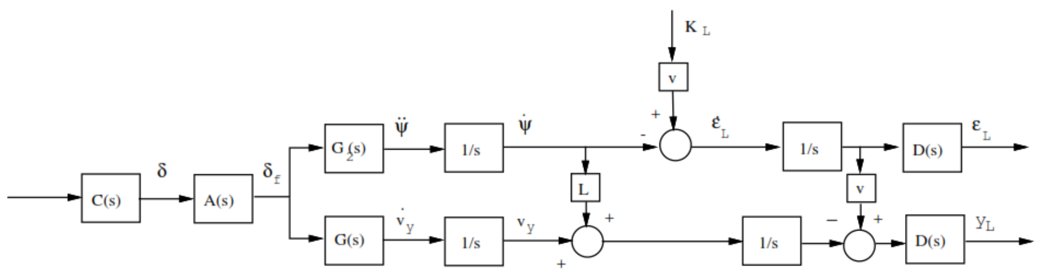

Figure 3.3: The Block Diagram of the Overall Vision Based Lateral System

There are two important transfer functions: The first one is $V_1(s)$ between the front wheel steering angle δ_f and y_L and the second one is $V_2(s)$ between δ_f and ε_L. The transfer functions $V_1(s)$ and $V_2(s)$ share a denominator $P(s)$:

$$P_s = s^2(s^2 v_x^2 m I_\psi + s v_x (I_\psi(c_f + c_r) + m(c_f l_f^2 + c_r l_r^2)) + c_f c_r l^2 + m v_x^2 (c_f l_f + c_r l_r)) \quad (3.15)$$

and the visualization of those two important transfer functions are:

$$V_1(s) = \frac{y_L}{\delta_f} = \frac{s^2 v_x^2 c_f I_\psi + s v_x c_r c_f (l_f l_r + l_r^2) + c_r c_f v_x^2 l + L(s^2 v_x^2 c_f l_f m + s v_x c_r c_f l)}{P(s)} \quad (3.16)$$

$$V_2(s) = \frac{\varepsilon_L}{\delta_f} = \frac{s^2 c_f l_f m v_x^2 + s c_f c_r v_x l}{P(s)} \quad (3.17)$$

Where δ_f denotes the front wheel steering angle, y_L is the offset from the center line at the look-ahead distance, ε_L is the angle between the tangent to the road and the orientation of the vehicle with respect to the road and L is the camera look-ahead distance.

3.5 Frequency Domain System Analysis

Under the situation that: vehicle cruise speed $V_x = 20m/s$ and fixed look-ahead distance $L = 15m$.

$$P = 1.732e09s^4 + 2.436e10s^3 + 2.675e11s^2 \tag{3.18}$$

$$V_1 = \frac{y_L}{\delta_f} = \frac{819.7s^2 + 6108s + 7390}{s^4 + 14.06s^3 + 154.4s^2} \tag{3.19}$$

$$V_2 = \frac{\varepsilon_L}{\delta_f} = \frac{49.56s + 369.5}{s^3 + 14.06^2 + 154.4s} \tag{3.20}$$

The core of the analysis lies in the understanding of the behavior of the vehicle at various speeds (the complex nonlinear model can be linearized at different cruise speed V_x), under various road conditions. Then, we analysed how different look-ahead distance L affects the dynamic behavior of the vehicle. Besides, the delay of vision subsystem is very important too.

In the following subsection, we study the system close loop performance by analyzing root locus and bode plot. When we apply a P controller (K = 1) to the plant V_1 and V_2, then we have $L = PK$, which is the open loop transfer function. Last, we can draw bode plots and root locus for the open loop transfer functions and we know the closed loop dynamics then.

3.5.1 Analysis of Model at Different Cruise Speed V_x

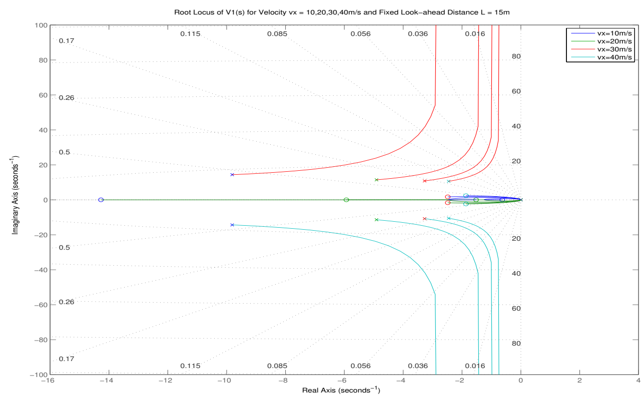

Figure 3.4: Root Locus of V1(s) for Varying Cruise Speed V_x and Fixed Look-Ahead Distance L = 15m

Figure 3.4: As the root locus of $V1_s$ shows, overall, the double integrator at the origin corresponds to the integration action between lateral acceleration and position at the look-ahead. The two poles and zeros in the left half plane characterize the vehicle dynamics.

By increasing the cruise speed V_x, both two poles and two zeros in the left half plane are moving towards to the imaginary axis.

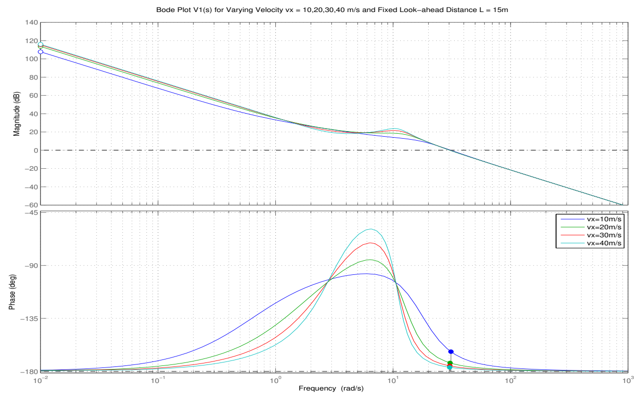

Figure 3.5: Bode Plot of V1(s) for Varying Cruise Speed V_x and Fixed Look-Ahead Distance L = 15m

Figure 3.5: Bode plot V1(s) for varying cruise speed V_x = 10, 20, 30, 40 m/s with a fixed camera look-ahead distance and no vision subsystem delay. It shows that increasing the cruise speed V_x will decrease the Phase Margin (PM). Under the condition that cruise speed V_x = 40m/s (maximum speed in the plot), the Phase Margin (PM) is only 3.42 degrees which is not good.

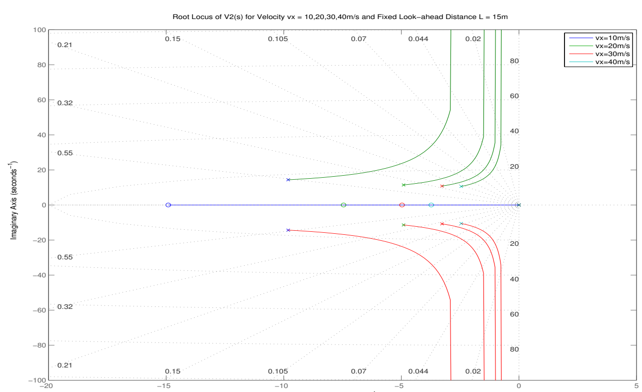

Figure 3.6: Root Locus of V2(s) for Varying Cruise Speed V_x and Fixed Look-Ahead Distance L = 15m

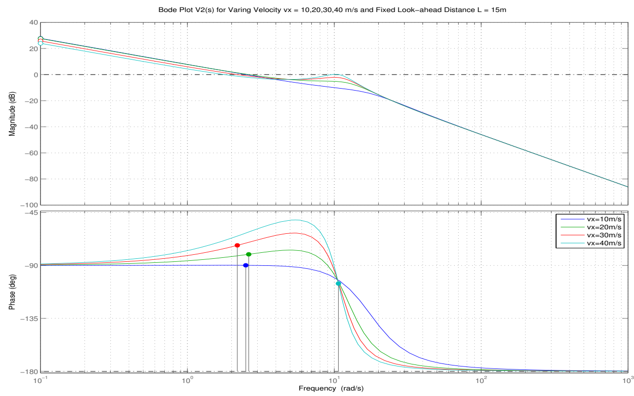

Figure 3.7: Bode Plot of V2(s) for Varying Cruise Speed V_x and Fixed Look-Ahead Distance L = 15m

3.5.2 *Analysis of Model at Different Look-Ahead Distance L*

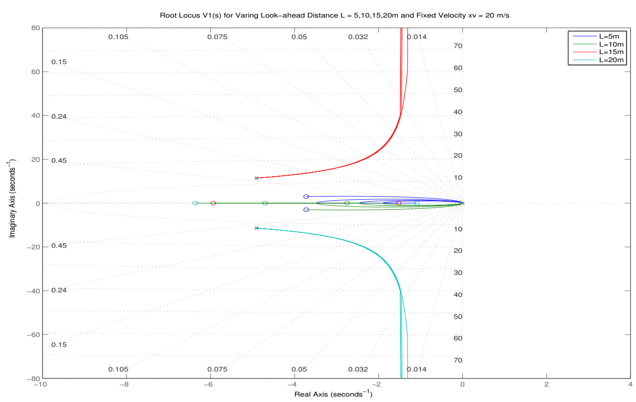

Figure 3.8: Root Locus of V1(s) for Varying Look-Ahead Distance L and Fixed Cruise Speed V_x

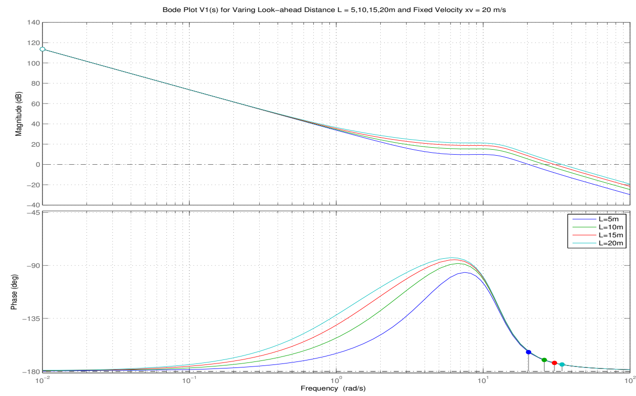

Figure 3.9: Bode Plot of V1(s) for Varying Look-Ahead Distance L and Fixed Cruise Speed V_x

Figure 3.9: Bode plot $V_1(s)$ for varying look-ahead distance L = 5, 10, 15, 20 m at $V_x = 20$m/s without delay. As the plot represents, increasing the look-ahead distance L adds substantial phase lead at the crossover frequencies.

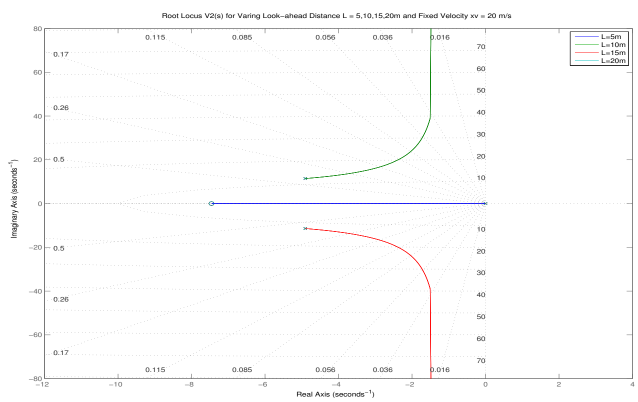

Figure 3.10: Root Locus of V2(s) for Varying Look-Ahead Distance L and Fixed Cruise Speed V_x

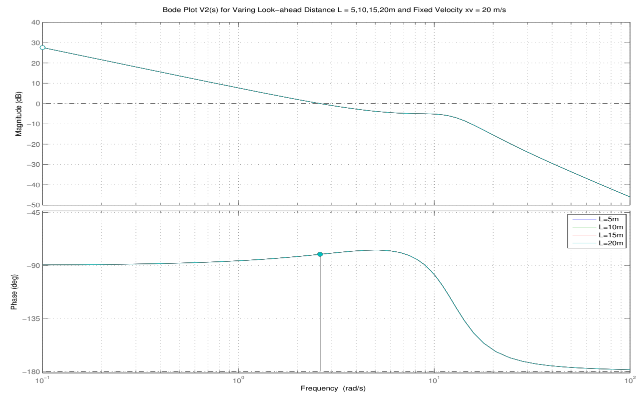

Figure 3.11: Bode Plot of V2(s) for Varying Look-Ahead Distance L and Fixed Cruise Speed V_x

3.5.3 Analysis of Camera Vision Delay Issues

One important parameter which will effect the overall system is the delay associated with the latency of visual processing. As shown in the overall system block diagram, the component is a pure time delay element $e^{-T_d s}$ representing the latency T_d of the vision subsystem. This delay component becomes:

$$D(s) = e^{-T_d s} \approx \frac{2 - T_d s}{2 + T_d s} \tag{3.21}$$

$V_1(s)D(s)$ demonstrate the effect of vision subsystem latency.

Under certain condition:

$$D(s) = \frac{-0.15s + 2}{0.15s + 2} \tag{3.22}$$

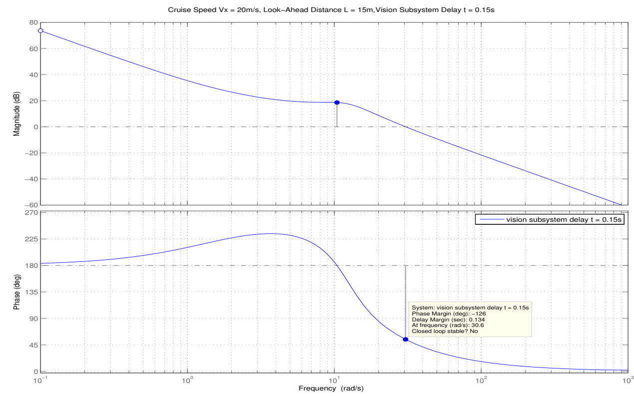

Figure 3.12: Bode Plot of V1(s)D(s) for Cruise Speed V_x = 20m/s, Look-Ahead Distance L = 15m and Vision Subsystem Delay t = 0.15s

In this Situation, the Phase Margin (PM) is $-126°$ which shows that the open loop system $V_1 D(s)$ is unstable due to the 0.15s vision subsystem latency.

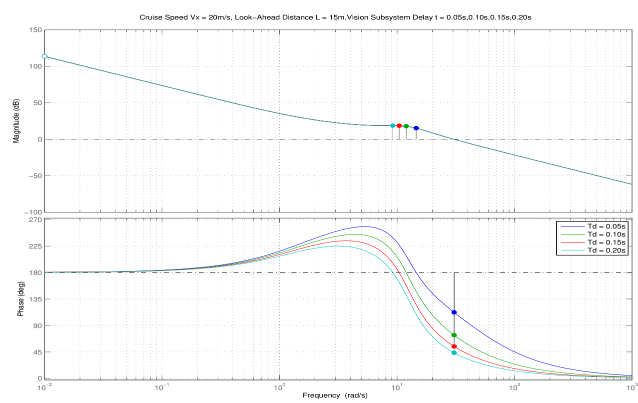

Figure 3.13: Bode Plot of V1(s)D(s) for Cruise Speed V_x = 20m/s, Look-Ahead Distance L = 15m and Varying Vision Subsystem Delay t = 0.05s, 0.10s, 0.15s, 0.20s

Figure 3.13: The presence of the delay adds an additional phase lag over the whole range of frequencies. In this case, the phase margin (PM) of all circumstances will diminishes and the systems are becoming unstable.

3.6 Summary and Conclusion

Within this Chapter 3, we discussed the vision based lateral complete model, simulation results were well presented and analysed. In the following chapter, hardware implementations will be introduced and compared to the simulation results.

Chapter 4

CASE STUDY FOR MODELING, CONTROL AND IMPLEMENT OF A SELF-DESIGNED REAR WHEEL DRIVE TESTBED : FREESLAM ROBOT

4.1 Introduction and Overview

In this chapter, we describe how to significantly enhance a self-designed rear-wheel drive FreeSLAM vehicle with the capabilities described in Chapter 1. i.e.magnetic wheel encoders for estimating translational/rotational speeds and distances, IMU for vehicle posture θ estimation, camera for directional information, xv 11 hacked LIDAR for depth information, Wi-Fi adapter for wireless communication between PC and Raspberry Pi, Arduino for less intense computations, Raspberry Pi II for more intense (e.g. video based) computations. Both modeling and control issues are addressed. A TITO LTI vehicle-motor model is used as the basis for designing (v,$\dot{\psi}$) inner-loop control laws. Two outer-loop control law types are presented, analyzed and implemented in hardware: (1) (v,θ) cruise control - track following (using camera, encoder and IMU), (2) planar (x, y) Cartesian stabilization (using encoders and IMU). Once the basic control issues are addressed, the vision-based lateral model is explained in detail. According to this model, three key parameters will greatly influence the tracking performance: robot cruise speed, fixed look-ahead distance and delay from vision subsystem. Each case above was well tested and discussed. The underlying theory for each control law is explained and justified. Finally, the results from our many hardware demonstrations are presented and discussed.

4.2 Hardware Limitations

Understanding fundamental hardware limitations is critical to understand what is realistically achievable. This is addressed for each of the following: xv 11 hacked LIDAR, encoders, Raspberry pi camera, Arduino Uno, IMU, and Raspberry Pi II. The following is common to all hardware implementations for our rear-wheel drive robot.

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

$$ZOH(j\omega) = \frac{1 - e^{-j\omega T}}{j\omega} = e^{-j\omega 0.5T} \frac{j2\sin\omega 0.5T}{j\omega} = Te^{-j\omega 0.5T}\left[\frac{\sin 0.5\omega T}{0.5\omega T}\right] \quad (4.1)$$

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

$$e^{-s\Delta} = \frac{e^{-s0.5\Delta}}{e^{s0.5\Delta}} \approx \frac{1 - s0.5\Delta}{1 + s0.5\Delta} = \left[\frac{\frac{2}{\Delta} - s}{\frac{2}{\Delta} + s}\right] \quad (4.2)$$

  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 . 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.** In this thesis, 12 small magnets and one hall effect sensor are used to serve as an self-designed encoder. 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 $\frac{2\pi}{12} = \frac{\pi}{6}$ or 30°. This amount of error seems very large. Because we could not fit more magnets on the wheel, we maxed out at eight. 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 12 magnet system is $x_{resolution} = r_{wheel}\theta_{mag_{resolution}} = (2.4cm)(\frac{2\pi}{12}) \approx 1.31$ 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 Hall effect sensor.) The distance traveled at each count is $\Delta x = 0.0131 \times counter$ m.

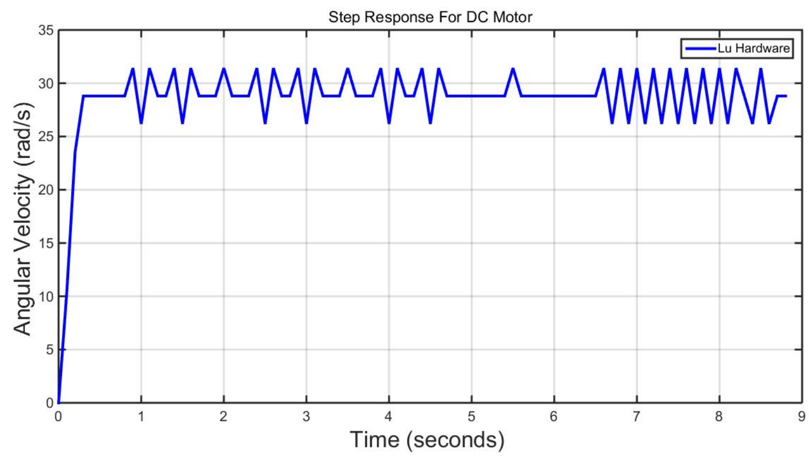

Figure 4.1: Encoder Resolution Before Average Filter Implementation

Original Encoder Resolution

Average angular velocity is 28.8 rad/s and peak to peak ripple is 5.2 rad/s.

After implementing average filter (a signal processing method)

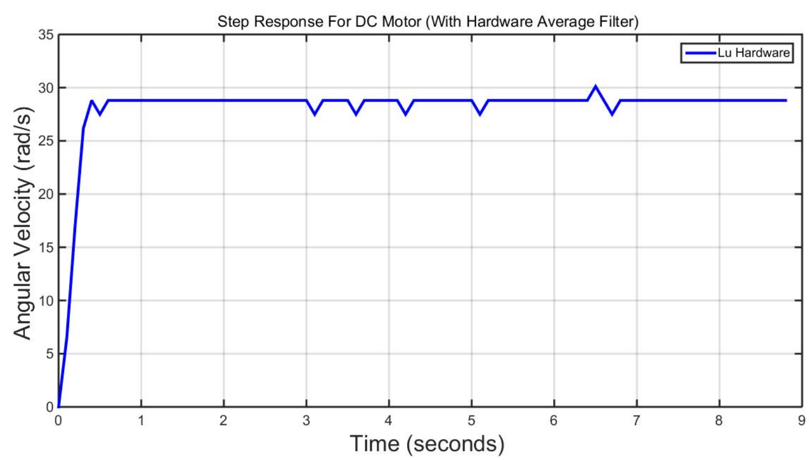

Figure 4.2: Encoder Resolution After Average Filter Implementation

Filtered Encoder Resolution

After implementing the average filter, we can make the following observations that peak to peak ripple is 2.6 rad/s, which has been greatly reduced.

Estimation of Vehicle-Motor Model Parameters. The dc motor parameters were estimated by iterating between experiments and model-based time simulations. Motor armature inductance L_a was neglected. Armature resistance R_a was measured using Ohm's law: $R_a = \frac{V}{I_a}$. Settling time, steady state speed and armature current were used to solve for two parameters: angular speed damping β, back emf and torque constant $K_b = K_t$. The transfer function from armature voltage control input to angular shaft velocity for a dc motor-load combination is given by:

$$\frac{\omega}{V} = \left[\frac{\frac{K_m}{R_a I}}{s + \frac{R_a b + K_b K_m}{R_a I}} \right] \tag{4.3}$$

From this, we observe that the

$$\text{Motor DC Gain} = \frac{K_t}{K_t K_b + R_a b} \tag{4.4}$$

$$\text{Motor Dominant Pole} = \frac{R_a b + K_b K_t}{R_a I} \tag{4.5}$$

Motor Model for FreeSLAM rear wheel drive robot is RN 260-c.

Here are the parameters:

Motor(Actuator) transfer function:

$$\frac{\Omega(s)}{U_a(s)} = \frac{K_t}{L_a J s^2 + s(L_a B + R_a J) + K_e K_t + R_a B}$$

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 represents the torque exerted

Table 4.1: RN 260 Motor Dynamics

	Current (A)	Speed (rpm)	Torque (g*cm)	Voltage (V)
No Load	0.13	10000	0	4.5
Max Efficiency	0.51	7950	18	4.5
Max Output	1.07	5000	44	4.5
Stall	2	0	88	4.5

by the motor on the motor shaft-load system, ω 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_e represents the back emf motor constant, K_t represents the motor torque constant, b represents a load-motor speed rotational damping constant, and I represents the moment of inertia of the motor shaft-load system.

- R_a **Armature Resistance**

$$U_a = E_a + I_a + R_a \tag{4.6}$$

$$P_1 = U_a I_a \tag{4.7}$$

$$P_M = E_a I_a \tag{4.8}$$

$$R_a = \frac{P_1 - P_M}{I_a^2} \tag{4.9}$$

- L_a **Armature Inductor**

$$L_a = 0.2mH \tag{4.10}$$

- K_t **motor torque constant and** K_e **motor back EMF constant**

$$T_e = K_t I_a \tag{4.11}$$

$$I_a = 1.07A \tag{4.12}$$

$$T_e = 44g \cdot cm \tag{4.13}$$

$$= 0.0043N \cdot m \tag{4.14}$$

$$K_e = K_t \tag{4.15}$$

Off Ground Motor Dynamics Comparison Between Hardware and Simulation Result

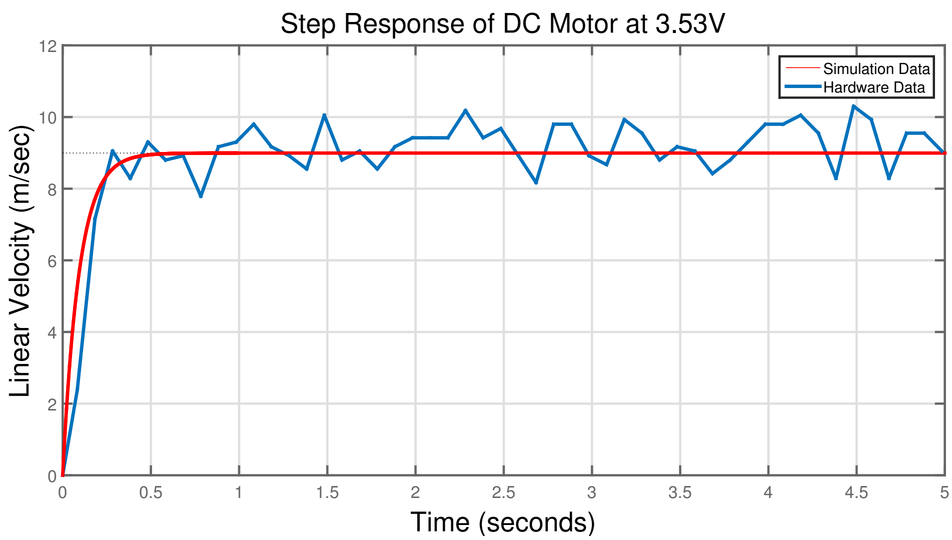

Figure 4.3: Off Ground Motor Dynamics Comparison

From figure 4.3, we can make the following observations:

When the input voltage of DC motor is 3.53V, we can measure the steady state of wheel linear velocity which is 9 m/sec. Motor dynamics can be estimated as a standard first order system. Settling time T_s of the system is 0.3 seconds with a step response peak-peak ripple of 2.4 m/s.

DC Motor Dynamics (first order plant)

$$P_{motor} = \frac{27.1}{s + 10.64} \tag{4.16}$$

Table 4.2: FreeSLAM Robot Nominal Parameter Values and Characteristics

Parameters	Definition	Nominal Values
m	Fully Loaded Mass	1.47kg
m_0	Mass (Not Loaded)	0.83kg
I	Moment of Inertia (Estimated using Cube)	0.0015kgm2
r	Wheel Radius	0.024m
d_w	Distance Btw 2 Rear Wheels	0.134m
L_a	Armature Inductance	0.2mH (neglected)
R_a	Armature Resistance	2.523Ω
K_b	Back EMF Constant	0.004V/(rad/sec)
K_t	Torque Constant	0.004Nm/A
v_{max}	Max. Observed Speed (Enhanced Vehicle)	5m/s
v_{max0}	Max. Observed Speed (Original vehicle)	7.2m/s
e_{amax}	Max. Motor Voltage	7.2V
a_{max}	Max. Accel. (Enhanced)	3.2m/sec2
$\omega_{wheelmax}$	Max. Angular Vel. (Enhanced)	208.3 rad/sec

Table 4.3: Front Wheel Steer Angle δ_f Accuracy

Front Wheels Steering to The Right					
Arduino Servo PWM Command Increasement	Previous Steering Angle	Current Steering Angle	Actual Angle Increasement	Error	Percentage Error
N/A	N/A	5.9375	N/A	N/A	N/A
+5	5.9375	11.0625	5.125	+0.125	2.5%
+5	11.0625	16.0000	4.9375	-0.0625	1.25%
+5	16.0000	20.9375	4.9375	-0.0625	1.25%
+5	20.9375	25.8125	4.875	-0.125	2.5%
Front Wheels Steering to The Left					
N/A	N/A	-5.1250	N/A	N/A	
-5	-5.1250	-10.6875	-5.5625	-0.5625	11.25%
-5	-10.6875	-15.5625	-4.875	+0.125	2.5%
-5	-15.5625	-20.4375	-4.875	+0.125	2.5%
-5	-20.4375	-25.5625	-5.125	-0.125	2.5%
Avg. Steering Angle Error		3.28%			

Front Wheel Steering Angle Accuracy

To detect the accuracy of the front wheel steering inner loop, BON055 IMU is used to test the accuracy of response to Arduino servo command. Due to hardware limitations, range of steering angle is $-30° \sim +30°$ (+ denotes steering to the right).

4.4 Case Study for Vehicle Longitudinal Model and Linearized Lateral Model

Within this section , we address modeling, analysis and control design for rear wheel drive SLAM robot. Both kinematic and nonlinear models are examined. Nominal model parameters were accurately measured for FreeSLAM robot. The nonlinear dynamical model is a three degree-of-freedom (dof) sixth order model that ignores actuator (DC motor) dynamics.The linear model is fourth order if the two position variables(X,Y) are removed from the model.The dynamical model is linearized about constant translational speed conditions. The goal is to understand the model to develop speed dependent cruise control laws. The studies presented shall serve as the basis for future cruise control system designs and hardware implementations. Linearization about a constant speed (i.e. uniform rectilinear motion) results in decoupled longitudinal and lateral dynamics. As we will mention in Chapter 5, all those motions we applied to the robot, are called control data in the observatory model, they can be represented as:

$$u_{t_1:u_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, \cdots, u_{t_2} \tag{4.17}$$

The linear longitudinal model (throttle to longitudinal speed v_x) is first order, stable and minimum phase. It is easy to control and trivial. The linear lateral model (steering angle to yaw angle) is third order and it is a little bit harder to control. Model characteristics were analyzed as a function of speed (for future cruise control developments). The (steering angle to yew rate $\dot{\psi}$) linear lateral model is stable for all speeds because the vehicle exhibits rear-wheel-dominated concerning($l_f c_f < l_r c_r$). Given this, it follows that the linear lateral dynamics (steering angle to yaw) are marginally stable for any speed (due to an integrator to generate yaw from yaw rate).

The longitudinal input is applied longitudinal force F(can be thought of as equivalent to throttle). The associated output is speed. The lateral input is the front wheel steering angle δ_f. The associated output is yaw angle ψ. A PI controller (with roll-off and a command pre-filter) was used for lateral angle (directional control). Control law parameters were selected at each speed in order to achieve a 5 seconds speed settling time and 2.5 seconds yaw settling time both with less than 7 percent overshoot to step reference commands. With this implementation, we then show how the control law parameters change as a function of speed again, for future cruise control law developments.

In short, the chapter presents results that will be useful for future cruise control law developments, for example, robot accurately line tracking and simultaneously localization and mapping and path planning etc.

4.5 Description of Nonlinear Model for Rear-Wheel Drive (RWD) Robot

Within this section, we examine two models for the rear wheel drive vehicle (FreeSLAM robot). The first is an ideal kinematic model one that neglects mass-inertia effects. The second one is a more accurate dynamics model that captures mass-inertia effects. It is the latter dunamics model that will be used to conduct relevant speed dependent linear trade studies within this section.

4.5.1 Kinematic Model of FreeSLAM Robot

This section describes a kinematic model for my rear wheel drive FreeSLAM robot. Being a kinematic model, it ignores mass-inertia effects. Many of the equations of motion developed from this point forward will be based upon a simplification in which both the front and rear wheels of the vehicle are lumped together to form a single front and a single rear tire. This simplification is often referred to as a single $bicyclemodel$. The latter of these names belies the utility of this approach. One can find more complicated models which include roll and pitch dynamics. Such models are often used only for simulation.The *bicyclemodel* is more useful for analysis and control law development. Consider Figure 4.4. Within this figure, a body-fixed coordinate system is affixed to the vehicle's rear axle.

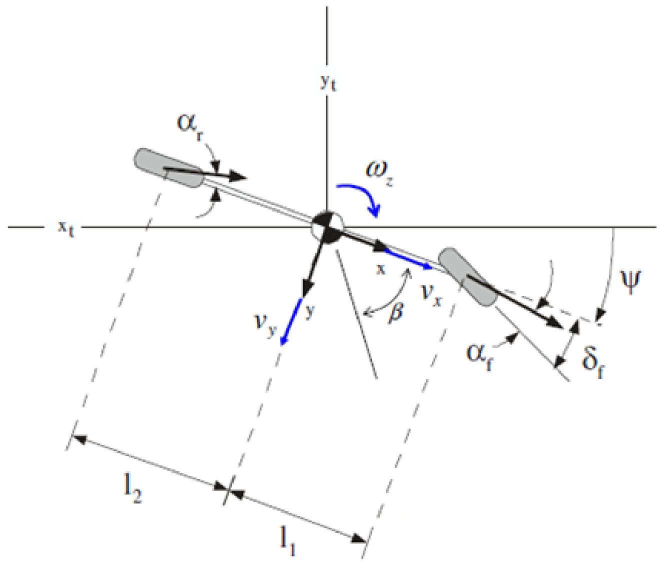

Figure 4.4: Visualization of Kinematic Model for RWD Robot (The Bicycle Model)

The vehicle's kinematics are as follows.

$$\dot{x} = vcos\Psi$$

$$\dot{y} = vsin\Psi$$

$$\dot{\Psi} = \frac{vtan\Psi}{L}$$

where:

- Ψ is the vehicle angle with respect to the X - axis
- $v_x = \dot{x}$ and $v_y = \dot{y}$ are the x and y projections of v.

- L is the distance between the front and rear wheels.

- δ is the front wheel steering angel

4.5.2 Nonlinear Dynamics Model for FreeSLAM Rear Wheel Drive Robot

Nominal model parameters were measured. The following defines key model variables.

- v_x denotes longitudinal speed

- α_f,α_r side slips angle between steering angle and front and rear tire velocities

- ψ vehicle yaw angle

- δ_f front wheel steering angle

- δ commanded steering angle

- m total mass of the vehicle

- I total inertia of the vehicle around centre of gravity

The nonlinear (single track) dynamics model is used within this thesis. Nominal model parameters were accurately measured for FreeSLAM robot. The nonlinear vehicle model is described by the following dynamics equations:

$$m(\dot{v_x} - v_y r) = -c_a {v_x}^2 + f_{lf} cos\delta_f + f_{lr} - f_{sf} sin\delta_f \tag{4.18}$$

$$m(\dot{v_y} + v_x r) = f_{sf} cos\delta_f + f_{sr} + f_{lf} sin\delta_f \tag{4.19}$$

$$I\dot{r} = l_f f_{sf} cos\delta_f - l_r f_{sr} + l_f f_{lf} sin\delta_f \tag{4.20}$$

and the following represents front and rear slip angles:

$$\alpha_f = \delta_f - (\frac{v_y + l_f r}{v_x}) \tag{4.21}$$

$$\alpha_r = -(\frac{v_y - l_r r}{v_x}) \tag{4.22}$$

Additional relationships that are useful are the following:

$$a_y = \dot{v_y} + r v_x \tag{4.23}$$

$$v_y = v sin\beta \tag{4.24}$$

The longitudinal and lateral models can be described as the following forth order matrix.

$$\begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{-2v_x c_a}{m} & 0 & 0 & 0 \\ 0 & -\frac{c_f + c_r}{m v_x} & 0 & -v_x + \frac{c_r l_r - c_f l_f}{m v_x} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-l_f c_f + l_r c_r}{I v_x} & 0 & -\frac{l_f^2 c_f + l_r^2 c_r}{I v_x} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{1}{m} & 0 \\ 0 & \frac{c_f}{m} \\ 0 & 0 \\ 0 & \frac{l_f c_f}{I} \end{bmatrix} \begin{bmatrix} F \\ \delta_f \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \psi \\ \dot{\psi} \end{bmatrix}$$

It is a decoupled TITO LTI system for the following reasons: we can observe the zeros in both first column and first row in matrix A, which means longitudinal state v_x is not influencing the 3 lateral states (v_y, ψ and $\dot{\psi}$). In the same way, those three lateral states are not influencing the longitudinal state (v_x). According to the analysis above, we can make a brief conclusion that it is a decoupled TITO LTI System.

The values of the parameters of the FreeSLAM model used in simulations are : m = 1.47kg , I = 0.0015 kg$\cdot$ m^2 , front wheel stiffness $c_f = 0.0368\ N/rad$, rear wheel stiffness $c_r = 0.0368\ N/rad$, the concerning stiffness is increased by factor 2 since the two tires are lumped together.

However, we have to state here that both longitudinal and lateral plant are not that accurate for the following three reasons:

- When we calculate the moment of inertia of the robot, it has been estimated as a cube
- front and rear wheel rotary stiffness (c_f and c_r) are under estimation
- the state space neglects static friction of the ground

Since the system dynamic estimation is not that accurate, we'll introduce System Identify method in the following subsection by introducing on-ground test. Comparison between the estimated model and System ID based model will be well explained.

4.6 Analysis of Linearized Model

Longitudinal Model(first order)

$$P_{longitudinal} = \frac{v_x}{F} = \frac{0.6803}{(s + 0.1116)} \tag{4.25}$$

Lateral Model (third order)

when equilibrium linear velocity v_e is 0.1 m/s

$$P_{Lateral} = \frac{\dot{\psi}}{\delta_f} = \frac{0.368(s + 0.484)}{(s + 1.007)(s + 0.457)} \tag{4.26}$$

Longitudinal Dynamics

Bode frequency response plot for the longitudinal plant as we change the equilibrium speed v_x in increments of 0.1 m/sec. We make the following observations:

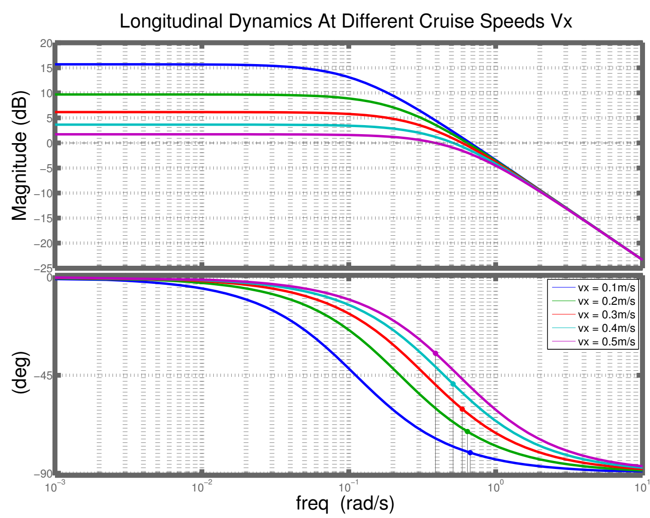

Figure 4.5: Longitudinal Dynamics at Different Cruise Speed Vx

Lateral Dynamics

Bode frequency response plot for the lateral plant as we change the equilibrium speed v_x in increments of 0.1 m/sec. We make the following observations:

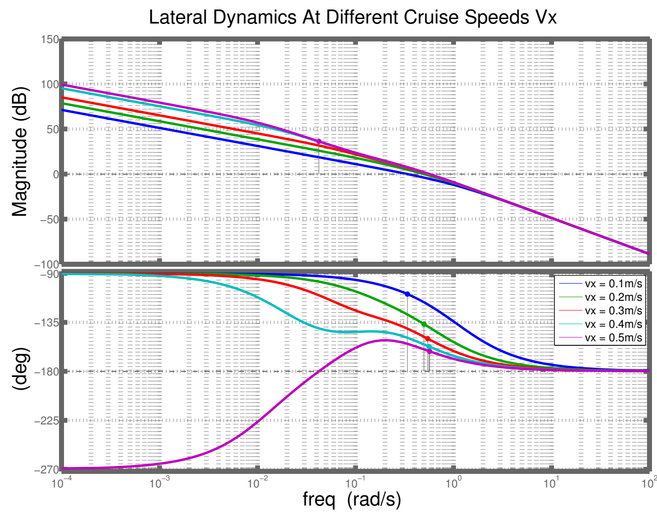

Figure 4.6: Lateral Dynamics at Different Speed Vx

Speed Dependent Pole Movement.

Bode frequency response plot for the lateral plant as we change the equilibrium speed v_x in increments of 0.1 m/sec. We make the following pole movement observations:

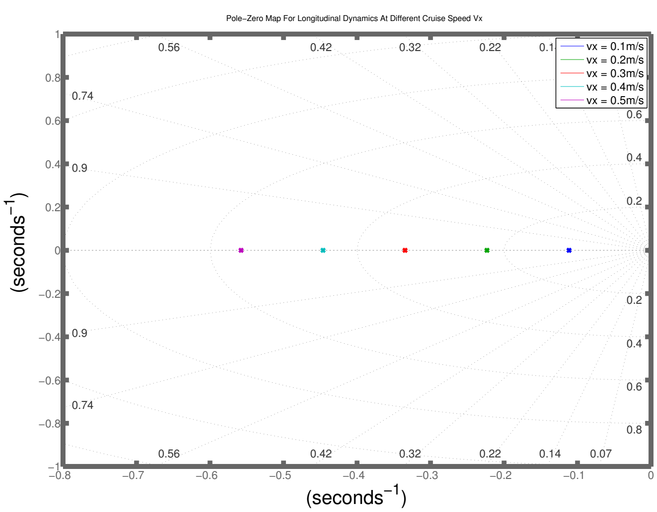

Figure 4.7: Pole-Zero Map For Longitudinal Dynamics at Different Cruise Speed Vx

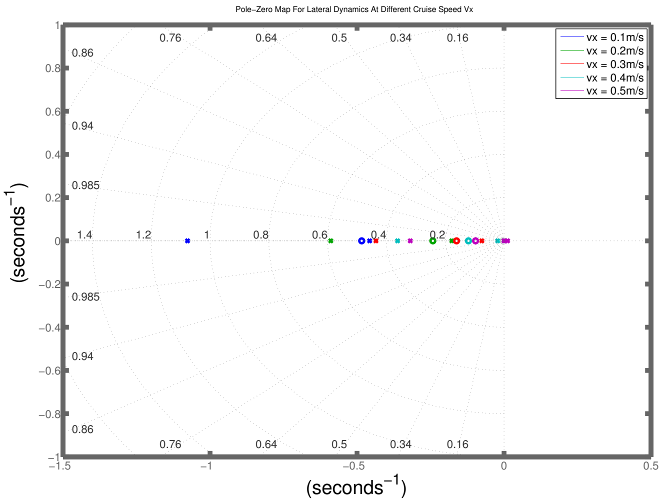

Figure 4.8: Pole-Zero Map For Lateral Dynamics at Different Cruise Speed Vx

4.6.1 Longitudinal Inner Loop Controller Design

Inner Loop Controller 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.

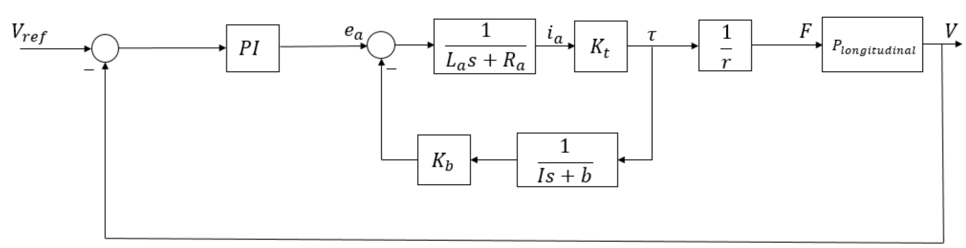

Figure 4.9: Block Diagram for Longitudinal Model Inner Loop Control

Longitudinal Plant:

$$P_{long} = \frac{V_x}{F} = \left[\frac{0.6803}{(s+0.1116)}\right] \tag{4.27}$$

Then combining the motor dynamics we have obtained in Chapter 3

$$\frac{F}{e_a} = 0.215\left[\frac{(s+14.53)}{(s+16.67)}\right] \tag{4.28}$$

we get the final longitudinal inner loop plant:

$$P_{long_inner} = P_{long}\frac{F}{e_a} = 0.146\left[\frac{(s+14.53)}{(s+0.1116)(s+16.67)}\right] \tag{4.29}$$

As we can see here, the dominant pole is ($s = -0.1116$) and the fast pole ($s = -16.67$) comes from the motor dynamics.

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

$$K_{inner} = \frac{g(s+z)^m}{s}\left[\frac{100}{s+100}\right]^m \tag{4.30}$$

Because the rear wheel drive vehicle will have the same rear wheel speed, K_{inner} will be the same for driving two DC motors.

Then, we are going to design for a phase margin (PM) of 60 deg and unity-gain crossover frequency (ω_g) of 3 rad/sec. The open loop transfer function L is given by

$$L = P_{long_inner}K_{inner} = \frac{g(s+z)^m}{s}\left[\frac{0.146(s+14.53)}{(s+0.1116)(s+16.67)}\right]\left[\frac{100}{s+100}\right]^m \tag{4.31}$$

According to the phase margin $PM = 180^\circ + \angle L(j\omega_g)$, we can compute the z value, i.e.

$$PM = 180^\circ - 90^\circ + mtan^{-1}(\frac{\omega_g}{z}) + tan^{-1}(\frac{\omega_g}{14.53}) - tan^{-1}(\frac{\omega_g}{0.1116}) - tan^{-1}(\frac{\omega_g}{16.67}) - mtan^{-1}(\frac{\omega_g}{100}) = 60^\circ \tag{4.32}$$

As a result

$$tan^{-1}(\frac{3}{z}) = 58.12^{\circ} \tag{4.33}$$

$$z = 1.87 \tag{4.34}$$

Now after getting z, we obtain g by knowing that $|L(j\omega_g)| = 1$.

$$\frac{0.146g\sqrt{\omega_g^2 + z^2}\sqrt{\omega_g^2 + 14.53^2}}{\omega_g\sqrt{\omega_g^2 + 0.1116^2}\sqrt{\omega_g^2 + 16.67^2}} = 1 \tag{4.35}$$

$$g = 19.9 \tag{4.36}$$

This values of g and z yields

$$\Phi_{actual}(s) \approx s(s + 0.1116)(s + 16.67) + 0.146g(s + z)(s + 14.53) \tag{4.37}$$

A reference command pre-filter

$$W = \frac{z}{s + z} \tag{4.38}$$

The final g and z we have chosen are $g = 11.68$, $z = 02.02$.

The pre-filter W will ensure that the overshoot to a step reference command approximates that dictated by the second order theory.

4.6.2 On Ground Longitudinal Model

Actually, there is a slightly difference between the actual vehicle longitudinal on ground model with the model we have calculated.

Here is the on-ground longitudinal plant, we can see that the hardware result and simulation result are matched:

$$P_{long} = \frac{v_x}{e_a} = \frac{0.3274}{(s + 1.176)} \tag{4.39}$$

Longitudinal Plant e_a to v_x Step Response

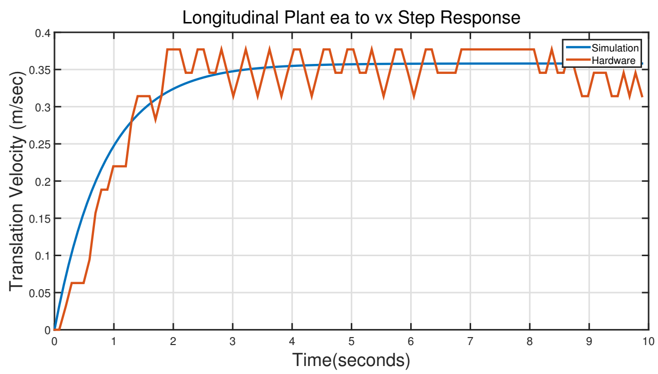

Figure 4.10: Longitudinal Plant ea to vx Step Response

- steady state is 0.35 m/s
- peak-peak ripple is 0.06 m/s

When we design the controller, settling time is set to 2 seconds and damping ratio is set to 0.9 (omega n is set to 2.78 rad/s and overshoot is around 0.15%) , PI controller parameters are g = 11.68 and z = 2.02.

Then we have T_{ry}:

$$T_{ry} = WPK(1 + PK)^{-1} \tag{4.40}$$

$$Try = \frac{7.716}{s^s + 5s + 7.716} \tag{4.41}$$

Finally we do the longitudinal inner loop performance studies:

T_{ry} (V_{ref} to V) Hardware and Simulation Result

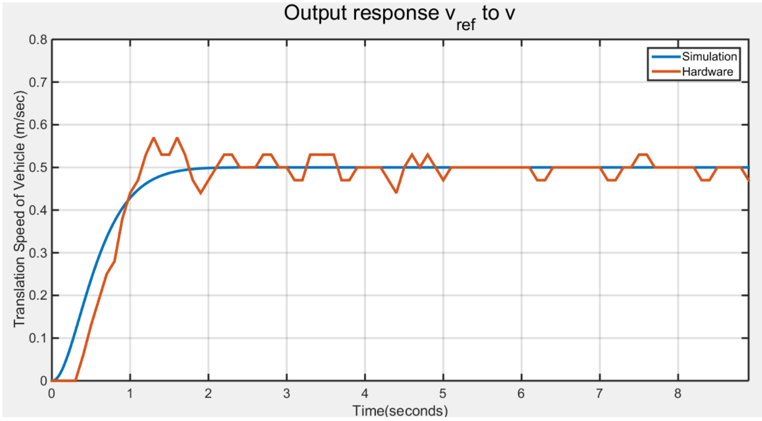

Figure 4.11: T_{ry} (V_{ref} to V) Hardware and Simulation Result

- steady state is $0.5m/s$, which is desired linear velocity
- peak-peak ripple is 0.06m/s

T_{ru} (V_{ref} to DC motor input voltage e_a) Hardware and Simulation Result

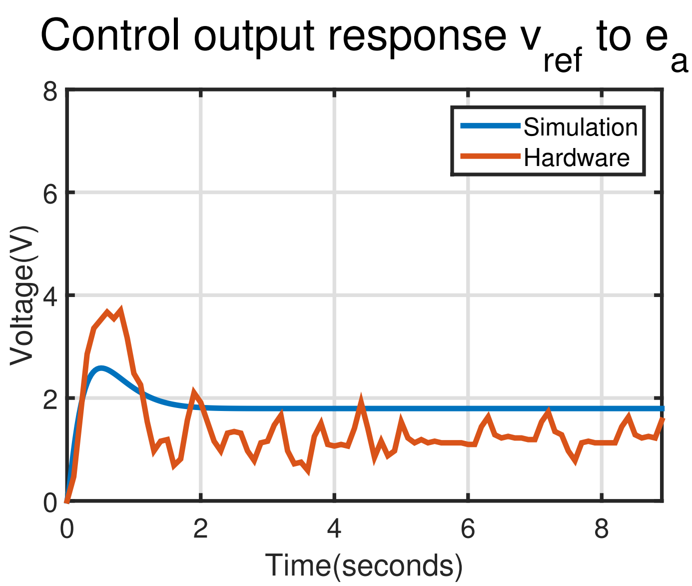

Figure 4.12: Longitudinal Plant ea to vx Step Response

- steady state of hardware result is 1.7 V

- peak-peak ripple of hardware result is 0.7 V

- simulation and hardware result are matched

We can draw a brief conclusion from the plots above that the simulation and hardware results are matched well.

4.6.3 Longitudinal Model Inner Loop PI Controller Trade Studies

In what follows, $L = PK = KP$ denotes the open loop transfer function, $S = (1+L)^{-1}$ denotes the closed loop sensitivity transfer function. $T = L(1+l)^{-1}$ denotes the closed loop complementary sensitivity transfer function, KS denotes the transfer function from (unfiltered) reference commands to controls (DC motor voltages e_a), and SP denotes the transfer function from input disturbances to the wheel speeds. We now examine trade studies for gain g and zero z variations.

From Reference Command to Output T_{ry}: Magnitude Responses

When g is varied (g is from 1 to 17, $z = 0.5$), one obtains the closed loop T_{ry} magnitude responses in Figure 4.13 and Figure 4.14 contains magnitude responses for z variations ($g = 9$ and z is from 0.3 to 0.7). In this case, the pre-filter has been implemented.

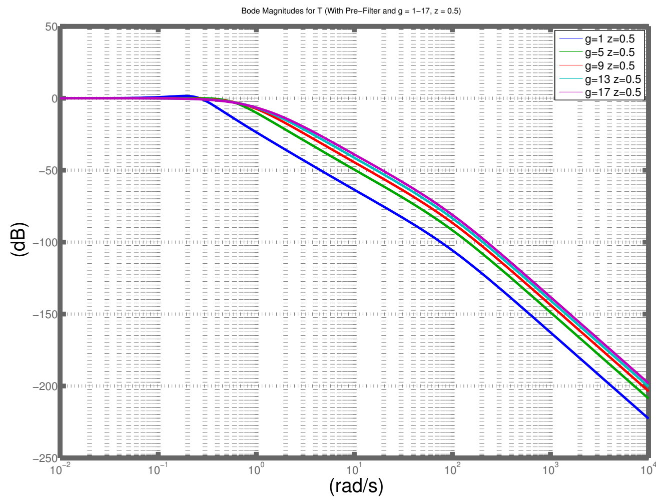

Figure 4.13: Bode Magnitudes for T_{ry} (With Pre-Filter and g = 1-17, z = 0.5)

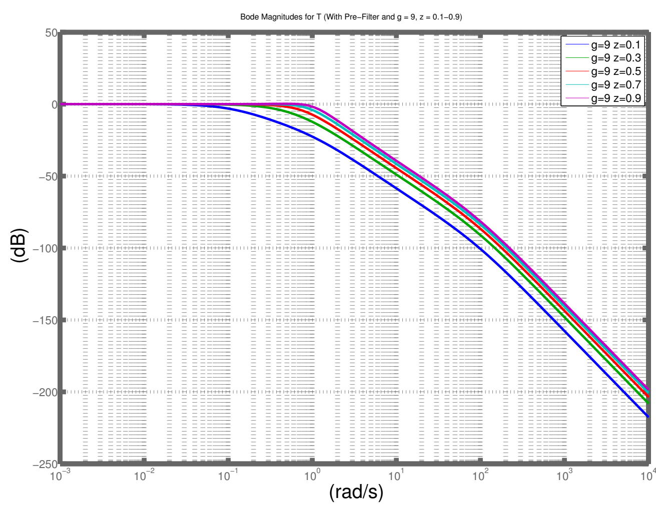

Figure 4.14: Bode Magnitudes for T_{ry} (With Pre-Filter and g = 9, z = 0.1-0.9)

From Figure 4.13 & 4.14, we observe the following:

- System bandwidth increases with increasing g or z
- Increasing z increases all peak magnitudes, peak magnitudes do not increase with a increasing g

Open Loop L Analysis

Figures 4.15 & 4.16 show the bode plots of $L = PK$ for specific (g, z) variations.

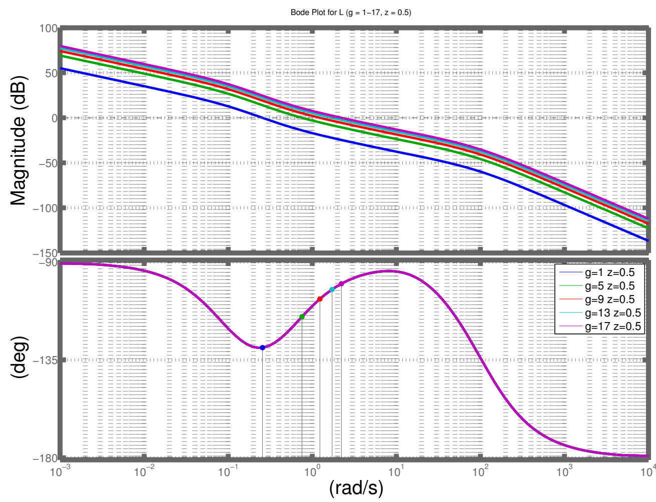

Figure 4.15: Bode Magnitudes for L and g = 1-17, z = 0.5

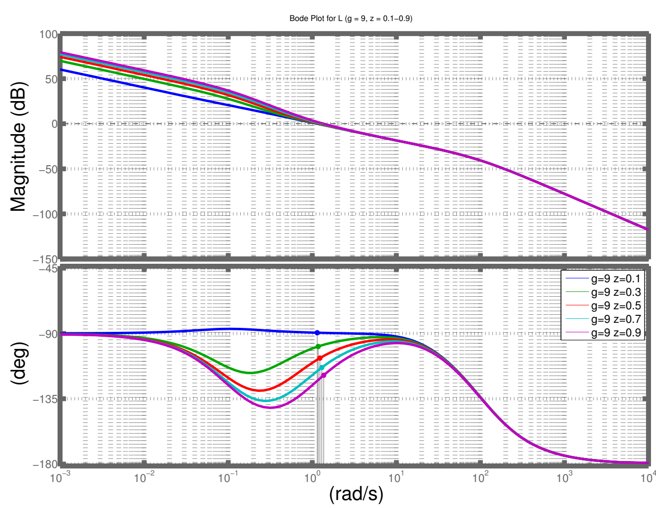

Figure 4.16: Bode Magnitudes for T (With Pre-Filter and g = 1-17, z = 0.5)

We observe that low frequency reference command r will be followed, low frequency output disturbances d_o will be attenuated and high frequency sensors noise n

will be attenuated too. Besides, the phase margin increases as the g is increasing.

Sensitivity (Longitudinal Decoupled Model)

Figures 4.17 & 4.18 contain sensitivity S bode-magnitude values for specific (g , z) variations.

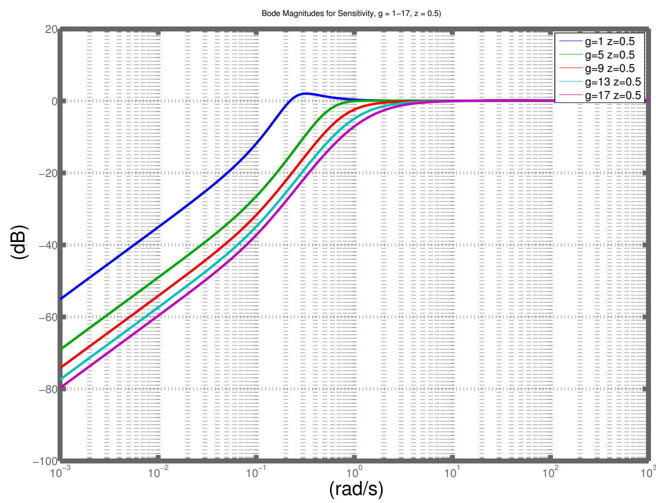

Figure 4.17: Bode Magnitudes for Sensitivity, g = 1-17, z = 0.5

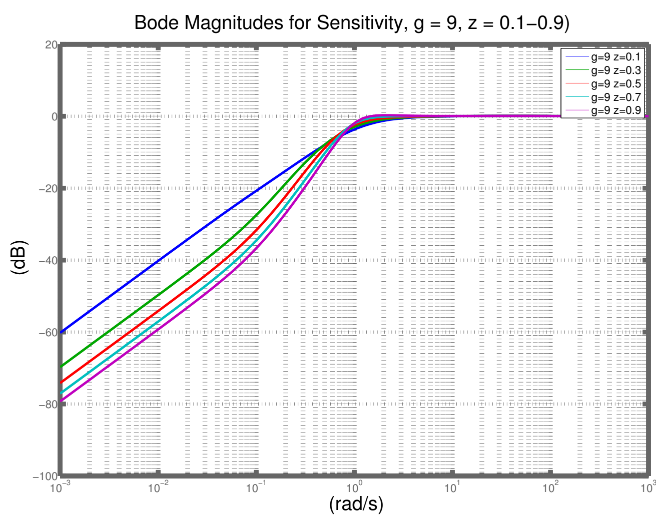

Figure 4.18: Bode Magnitudes for Sensitivity, g = 9, z = 0.1-0.9

From Figure 4.17 & 4.18, we make the following observations:

- Increasing g results in smaller sensitivities at low frequencies and a slightly larger peak sensitivity.

- Increasing z results in smaller sensitivity at low frequencies but increases peak sensitivities somewhat (since it gives "less lead near crossover").

Complementary Sensitivity

Figures 4.19 & 4.20 contain complementary sensitivity bode magnitude values for specific (g, z) variations.

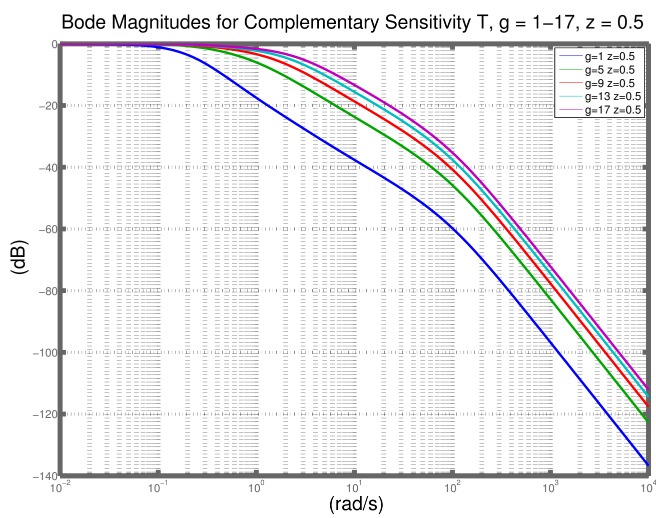

Figure 4.19: Bode Magnitudes for Complementary Sensitivity T, g = 1-17, z = 0.5

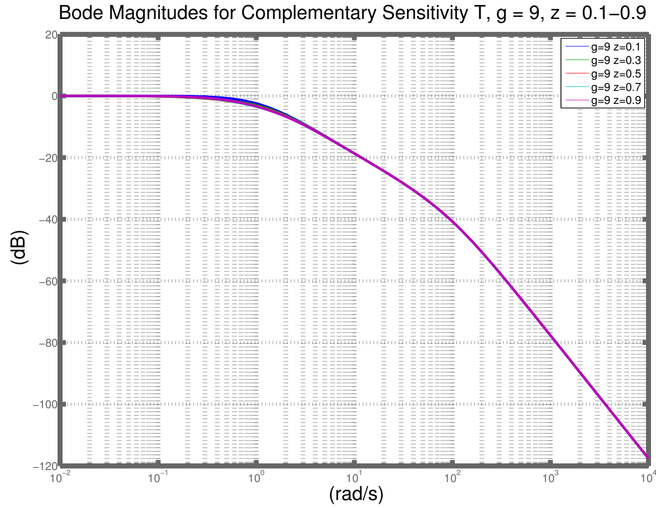

Figure 4.20: Bode Magnitudes for Complementary Sensitivity T, g = 9, z = 0.1-0.9

- Increasing g will result in a larger bandwidth and a smaller peak complementary sensitivity T, (but worse high frequency noise attenuation; a trade-off here must be made).

- Increasing z will result in larger bandwidth and a larger peak complementary sensitivity T. High frequency noise attenuation is the same for different z values.

Reference to Control (Unfiltered)

Figure 4.21 & 4.22 contain (unfiltered) reference to control bode magnitude plot for specific (g, z) variations. As the plots above, these plots are for the reference cruise speed v_x to DC Motor input voltage e_a longitudinal speed control system. As such, they tell us what control responses result from desired $\omega_{rearwheel}$ commands. This is addressed below.

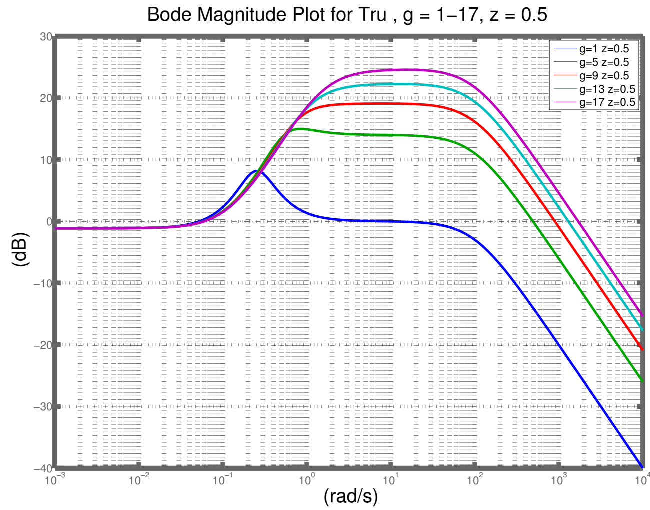

Figure 4.21: Bode Magnitude plot for Tru , g = 1-17, z = 0.5

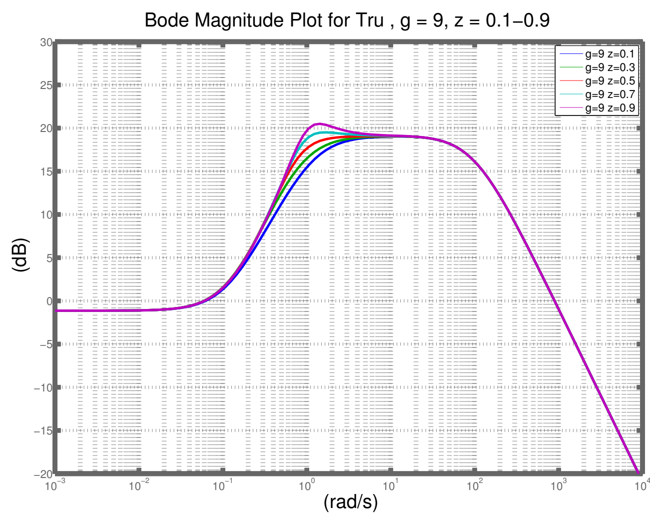

Figure 4.22: Bode Magnitude plot for Tru , g = 9, z = 0.1-0.9

- Increasing g or z increases the peak T_{ru} at all except low frequencies.
- Increasing g increases peak T_{ru}
- Increasing z increases peak T_{ru}

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 the reference command. Figures 4.23 & 4.24 contain (filtered) reference to control bode magnitudes for specific (g, z) variations. Here the reference command is desired robot cruise speed v_x and the control value stands for the input voltage e_a to the rear wheel DC motors.

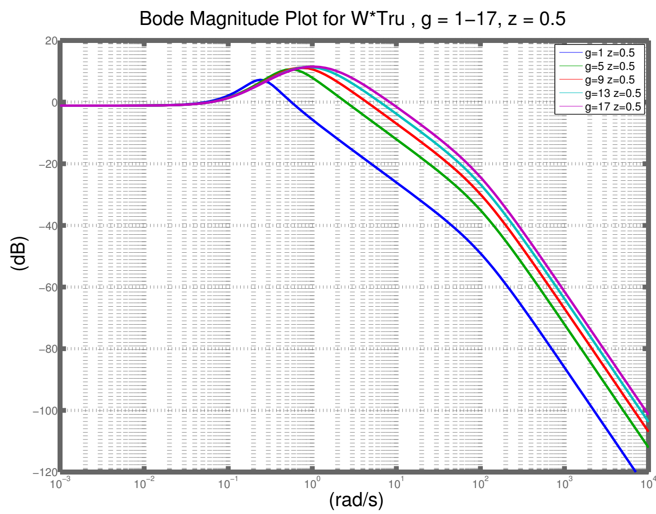

Figure 4.23: Bode Magnitude plot for TruW , g = 1-17, z = 0.5

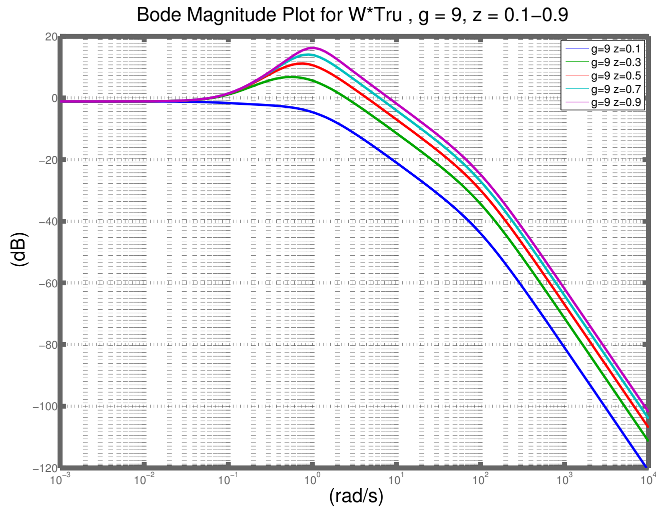

Figure 4.24: Bode Magnitude plot for TruW , g = 9, z = 0.1-0.9

- 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.

The above plots suggest that overshoot and saturation due to filtered v_x commands reference command 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 4.25 & 4.26 contain input disturbance to control singular values for specific (g, z) variations. As such, they tell us what cruise speed v_x responses result from input (DC motor input voltage e_a) disturbances.

Figures 4.25 & 4.26 contain the bode magnitude values for T_{diy} for specific (g, z) variations. We make the following observations:

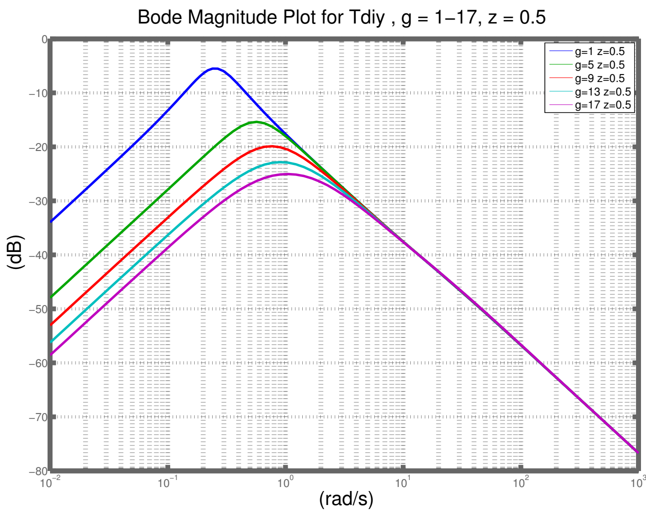

Figure 4.25: Bode Magnitude plot for Tdiy , g = 1-17, z = 0.5

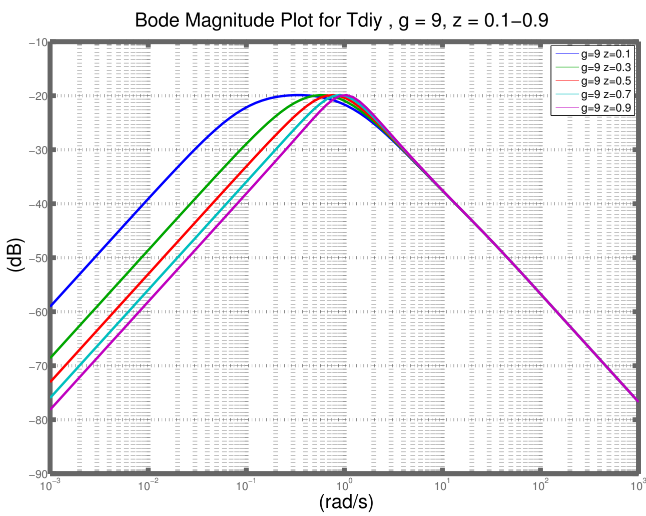

Figure 4.26: Bode Magnitude plot for Tdiy , g = 9, z = 0.1-0.9

From Figures 4.25 & 4.26, we make the following observations.

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

- increasing z reduces T_{diy} at low frequencies.
- frequency at which peak T_{diy} occurs increases with increasing g (also with increasing z but to a lesser extant)

4.6.4 Lateral Inner Loop Controller Design

Lateral Inner Loop Controller Design: PI With One Pole Roll-Off and Command Pre-filter base on the lateral model we've gotten in section 4.3.

Robot Lateral Model Inner Loop Controller Design

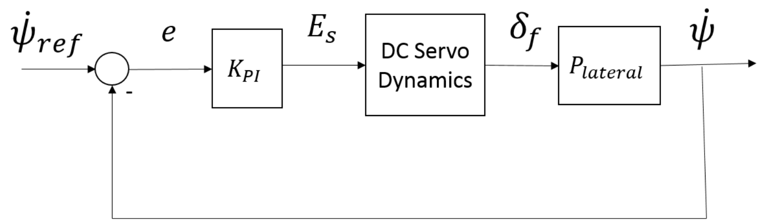

Figure 4.27: Block Diagram for Robot Lateral Model Inner Loop Control

Front Wheels Steering DC Servo Dynamics

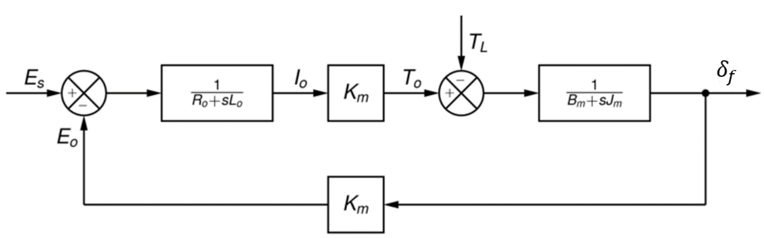

Figure 4.28: Front Wheels Steering DC Servo Dynamics

The Plot above is a complete model for the robot lateral dynamics. Actually we can control the steering angle of the DC Servo directly using Arduino Uno *servo.write* digital write command. In other word, we can control the parameter front wheel steering angle δ_f directly, as a result, DC motor dynamics was not carefully analysed in this chapter. Besides, because the response for front wheel steering DC Servo Dynamics is fast, the *ServoDynamics* block can be estimated as a constant number block 1.

Robot Lateral Plant:

when v_x is 0.1 m/s

$$P_{Lateral} = \frac{\dot{\psi}}{\delta_f} = \left[\frac{0.368(s+0.484)}{(s+1.077)(s+0.457)}\right] \tag{4.42}$$

Due to the integrator down there, it is not appropriate for us to implement a PI controller in this case. There are basically two ways to design the inner loop controller for this lateral model: The first choice is using a simple PI controller and the second option is implementing a model-based phase-lead compensator.

Let's talk about the simple PI controller (with high frequency roll-off and pre-filter) design first. The PI controller has the form:

$$K_{lateral} = \frac{g(s+z)^m}{s}\left[\frac{100}{100+s}\right]^{m+1} \tag{4.43}$$

Then, we are going to design for a phase margin (PM) of 60 deg and unity-gain crossover frequency (ω_g) of 5 rad/sec. The open loop transfer function L is given by

$$L = P_{lateral}K_{lateral} = \frac{g(s+z)^m}{s}\left[\frac{0.368(s+0.484)}{(s+1.077)(s+0.457)}\right]\left[\frac{100}{100+s}\right]^{m+1} \tag{4.44}$$

In this case, $K_p = g$ and $K_i = gz$

Lastly, we compute the ideal g and z here. In my design, $g = 18$, $z = 1.2$.

$$W = \frac{z}{s+z} \tag{4.45}$$

Here, pre-filter W will ensure that the overshoot to a step reference command approximates that dictated by the second order theory.

4.6.5 Lateral Model Inner Loop PI Controller frequency and Time Domain Studies

In what follows, $L = PK = KP$ denotes the open loop transfer function, $S = (1+L)^{-1}$ denotes the closed loop sensitivity transfer function. $T = L(1+l)^{-1}$ denotes the closed loop complementary sensitivity transfer function, KS denotes the transfer function from (unfiltered) reference commands to controls (front wheel steering angle δ_f), and SP denotes the transfer function from input disturbances to the wheel speeds. We now examine studies for this system in both frequency and time domain.

Open Loop L Frequency Domain Analysis

Figure 4.29 show the bode plot for $L = PK$ for designed g and z.

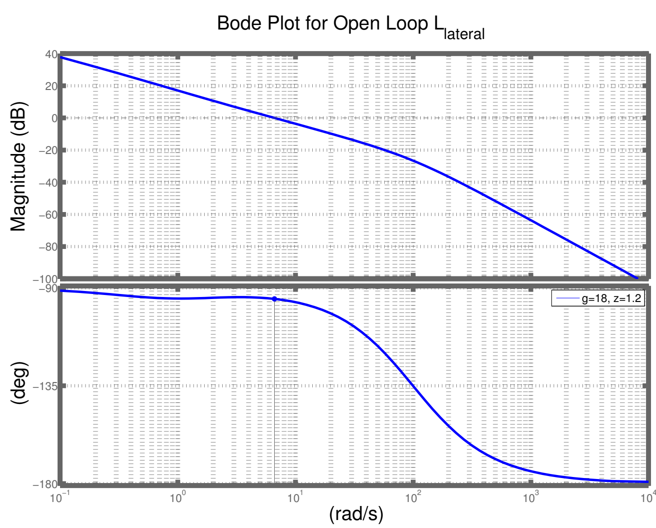

Figure 4.29: Bode Plot for Open Loop $L_{lateral}$

From Figure 4.29, we observe the following:

We observe that low frequency reference command r will be followed, low frequency output disturbances do will be attenuated and high frequency sensors noise n will be attenuated too.

With the PI controller $g = 18$ and $z = 1.2$, the crossover frequency of open loop L is 6.63 rad/s with a phase margin (PM) equals 84.9°. This means the open loop L is stable and the system is relatively faster than the longitudinal open loop system, which reflect the hardware performances.

T_{ry} without a pre-filter W

Figure 4.30 shows the frequency response for T_{ry} without a pre-filter $(\frac{1.2}{s+1.2})$. System should be fast but not that robust like the system with a pre-filter.

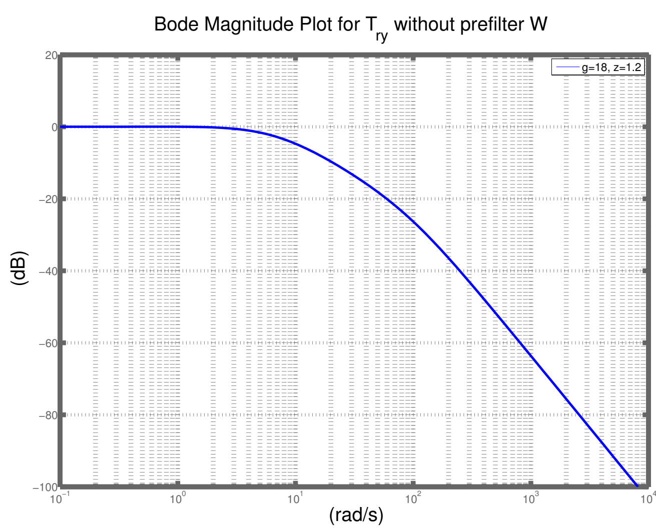

Figure 4.30: Bode Magnitude Plot for T_{ry} without Prefilter W

From Figure 4.24, we can observe that the $(-3dB)$ bandwidth is 7.26 rad/s.

T_{ry} with a pre-filter W

Figure 4.31 shows the frequency response for T_{ry} with implementing a pre-filter $(\frac{1.2}{s+1.2})$. As expected, the system is more robust but the low frequency pole (comes with the pre-filter W) reduces the bandwidth. Please see time domain analysis section for more details.

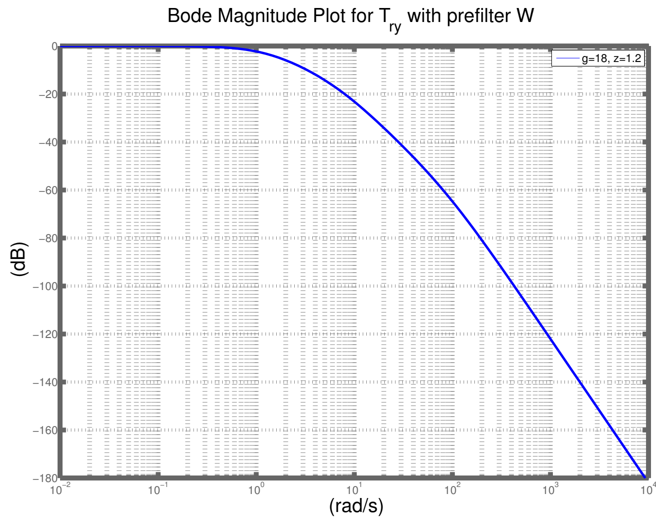

Figure 4.31: Bode Magnitude Plot for T_{ry} with Pre-Filter W

4.6.6 Time Domain Analysis for Robot Lateral Model

Step Response for T_{ry} without pre-filter W

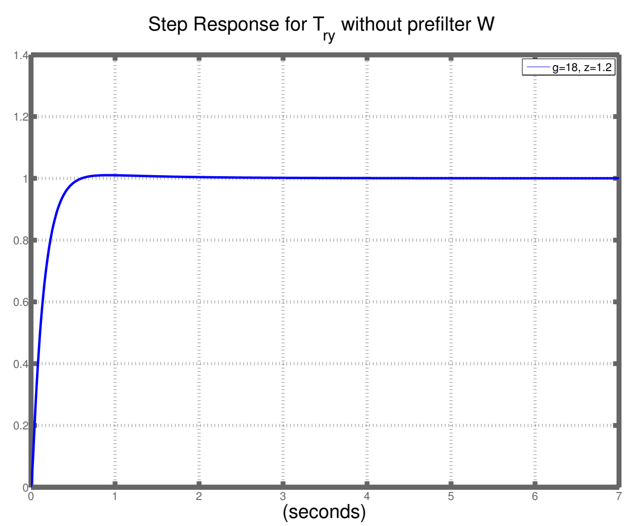

Figure 4.32: Step Response for T_{ry} without Pre-Filter W

As we observe from Figure 4.32, the output angular velocity $\dot{\psi}$ follows reference command $\dot{\psi}_{ref}$ very well with a 1 % overshoot and 1.2s settling time.

Step Response for T_{ry} with pre-filter W

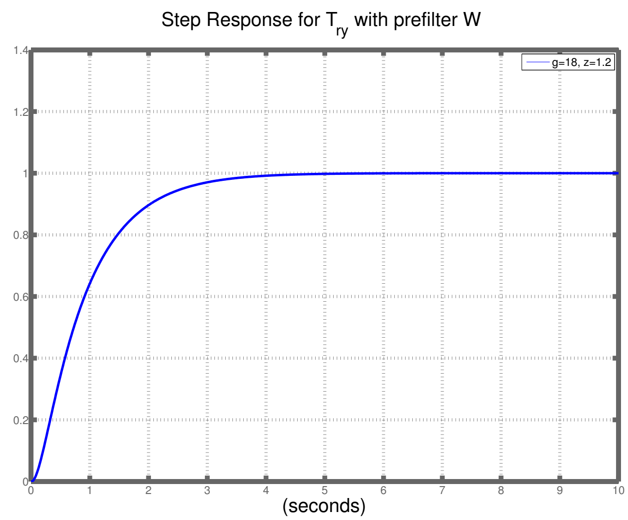

Figure 4.33: Step Response for T_{ry} with Pre-Filter W

As we observe from Figure 4.33 , the output angular velocity $\dot{\psi}$ follows reference command $\dot{\psi}_{ref}$ very well with no overshoot and a relatively larger settling time, which is 5.9 seconds. Compared to the step response for T_{ry} without a pre-filter W, the system is slower but more robust (no overshoot).

4.6.7 On Ground Lateral Model

Actually, there is a slightly difference between the actual vehicle on ground lateral model with the numerical model we have calculated. Here is the on-ground lateral plant, we can see that the hardware result and simulation result are matched:

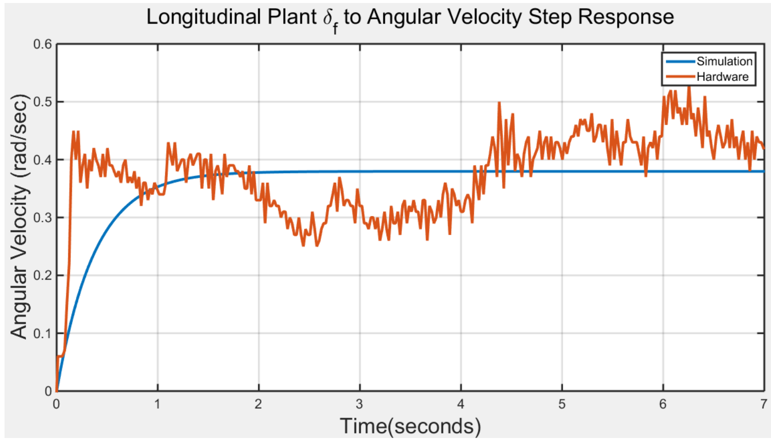

Figure 4.34: On Ground Lateral Plant

- Through system ID method (robot on-ground test), the linearized lateral plant can be estimated as a first order system
- step response steady state of hardware result is 0.38 rad/sec
- peak-peak ripple of hardware result is 0.27 rad/sec

To design the PI controller (for rapid response and zero steady state error), we set the desired settling time Ts = 1.5s (ω_n is set to 3.8 rad/s which is less than 4 rad/s ZOH bandwidth limitation). Then, set damping ratio to 0.886 (which means the step response of the system will roughly have a 0.4 % overshoot).

Here we design the PI controller: g = 1.38 z = 3.53.

Finally, we have T_{ry} (ω_{ref} to ω) here:

$$T_{ry} = \frac{14.8}{s^2 + 6.67s + 14.8} \tag{4.46}$$

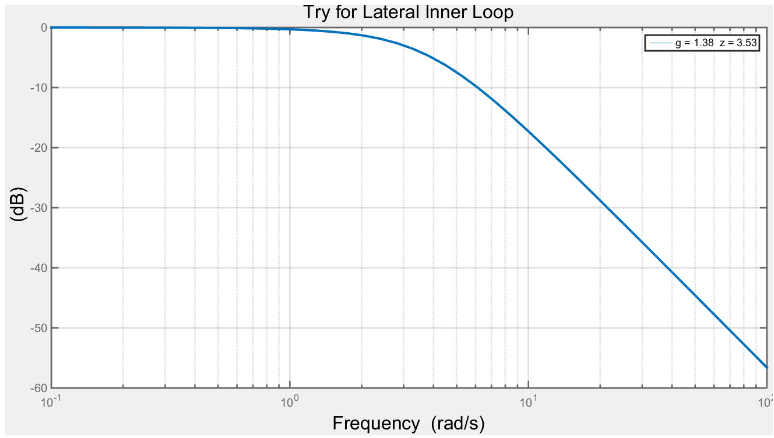

Figure 4.35: Lateral On Ground Inner Loop Try

And then T_{ru}, which is the ω_{ref} to steer angle δ_f response.

$$T_{ru} = \frac{5.12(s + 2.66)}{s^2 + 6.67s + 14.82} \tag{4.47}$$

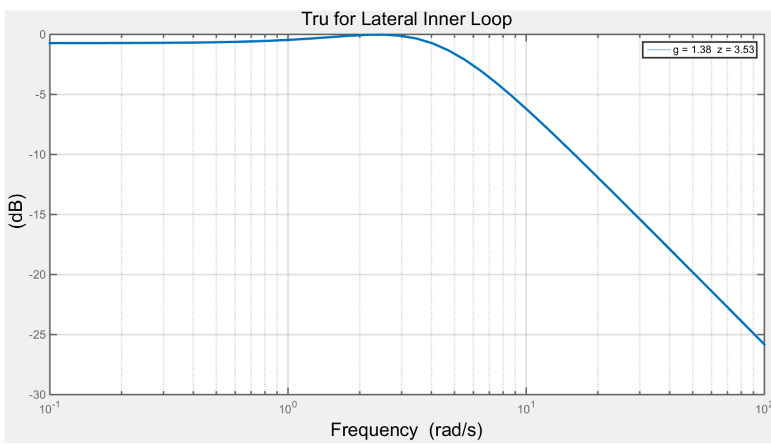

Figure 4.36: Lateral On Ground Inner Loop Tru

4.7 Outer Loop: (v, θ) Cruise Control Along Line - Design and Implementation

In this section, we examine (v, θ) cruise control along a line. This outer-loop control law can be visualized as shown in Figure 4.37.

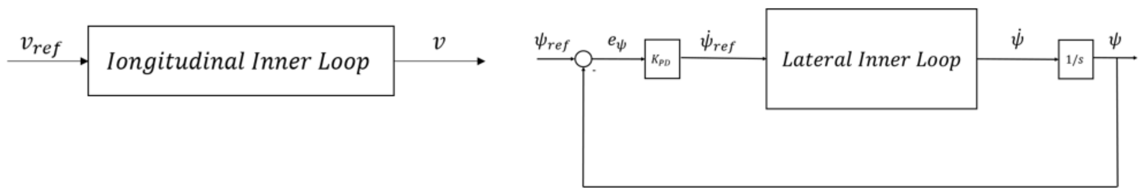

Figure 4.37: Visualization of Cruise Control Along a Line

Here, (v, θ) are commanded. v is calculated based on wheel encoders. For cruise control along a line, $v_{ref} = constant, \omega_{ref} = 0$ are commanded. For cruise control along a line, θ is calculated based on integrating ω measured by the IMU (i.e. $\theta = \theta_{previous} + \omega T$, $T = 0.1$ sec).

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.

Figure 4.38 shows both simulation and hardware implementation results for robot going along a straight line. As we can observe, the trajectory error increases while robot goes further. This error majorly comes from dead reckoning error.

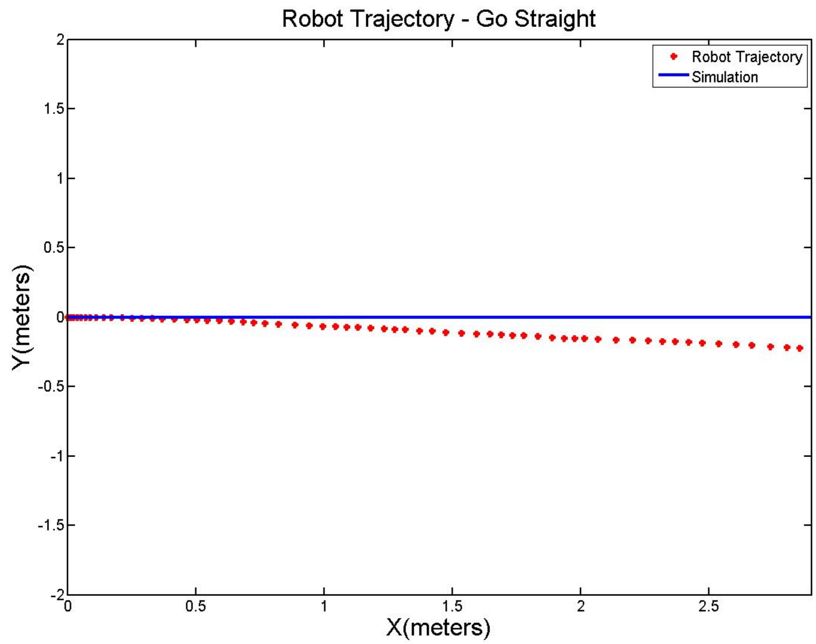

Figure 4.38: Robot Trajectory - Go Along a Line

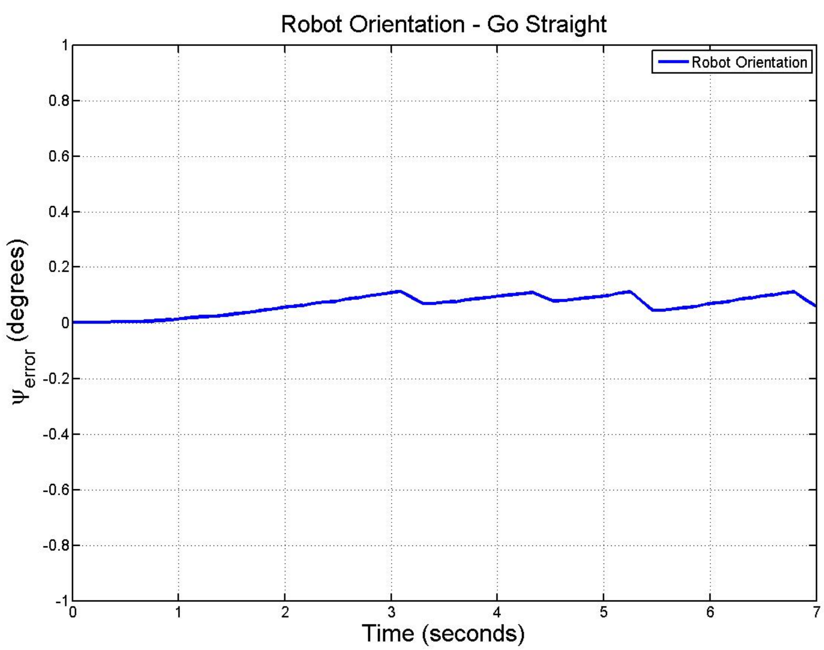

Figure 4.39: Orientation Error - Go Alone a Line

4.8 Outer Loop: Planar (x, y) Cartesian Stabilization - Design and Implementation

In this section, we discuss the planar (x, y, θ) outer-loop control law. It can be visualized as shown in Figure 4.40.

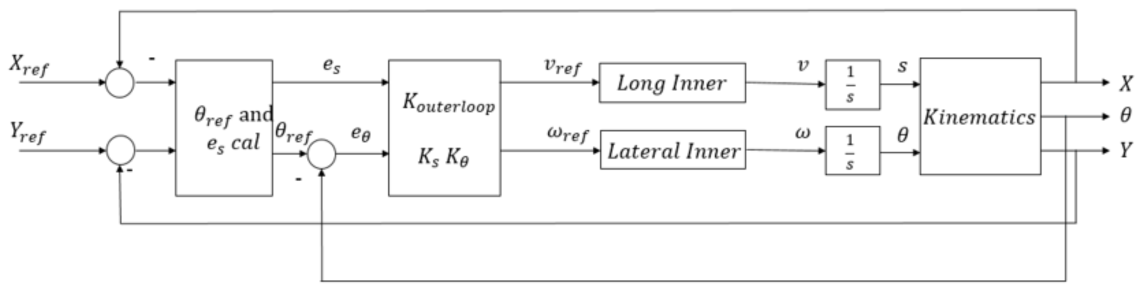

Figure 4.40: Visualization of Planar (xy) Cartesian Stabilization Control System

Here, θ is calculated based on ω information from IMU (i.e. $\theta = \theta_{previous} + \omega T$, $T = 0.1$ sec). X and Y position is estimated using dead reckoning based on wheel encoders. That is, $x = x_{previous} + v_x T$, $y = y_{previous} + v_y T$, $v_x = v \cos\theta$, $v_y = v \sin\theta$;

The nonlinear kinematic model can be usefully rewritten in terms of angular and linear displacements. For this transformed system, a simple control law $v = k_s e_s$, $\omega = k_\theta e_\theta$ results in an error dynamics matrix (after linearization) that is Hurwitz when $k_\theta > k_s > 0$ A drawback of this control law (consistent with the Brockett 1983 result is that it can only get the system arbitrarily close to the desired $(x_{ref}, y_{ref}, \theta_{ref})$. To precisely achieve the objective, one would have to switch control laws. These ideas are used to motivate a simple proportional control law for the planar (x, y) outer-loop position control that was implemented for the rear-wheel drive vehicle.

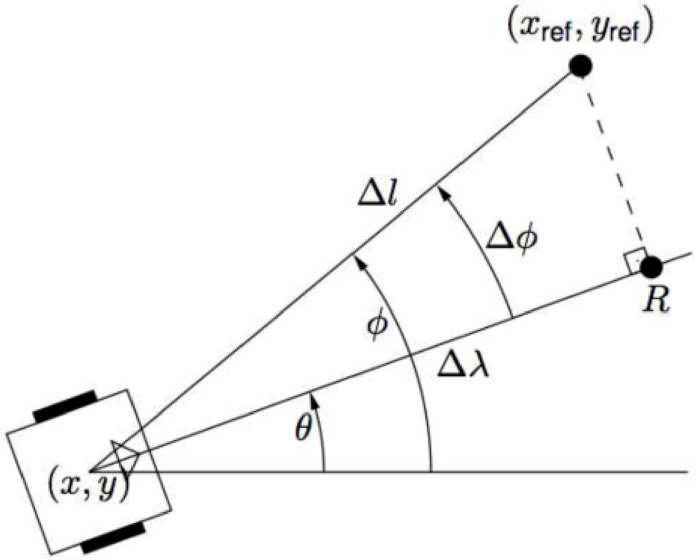

Figure 4.41: Visualization of Longitudinal Distance to Target $e_s = \Delta\lambda$ and Angular Error $e_\theta = \Delta\phi$

It is now useful to present some of the key ideas Cartesian stabilization. Let $e_s = \Delta\lambda$ denote the projection of the vehicle-to-target vector onto the longitudinal body axis of the vehicle. ϕ is defined as the angle which binds (x_{ref}, y_{ref}) and (x, y). It is called the *pointing angle*.

From Figure 4.41, we have:

$$\phi = \tan^{-1}\left(\frac{y_{ref} - y}{x_{ref} - x}\right) \tag{4.48}$$

$$e_\theta = \phi - \theta \tag{4.49}$$

$$e_s = \Delta\lambda = \Delta l \cos \Delta\phi \tag{4.50}$$

The structure of the control law is as follows - a proportional control law:

$$v = k_s e_s \qquad\qquad \omega = k_\theta e_\theta \tag{4.51}$$

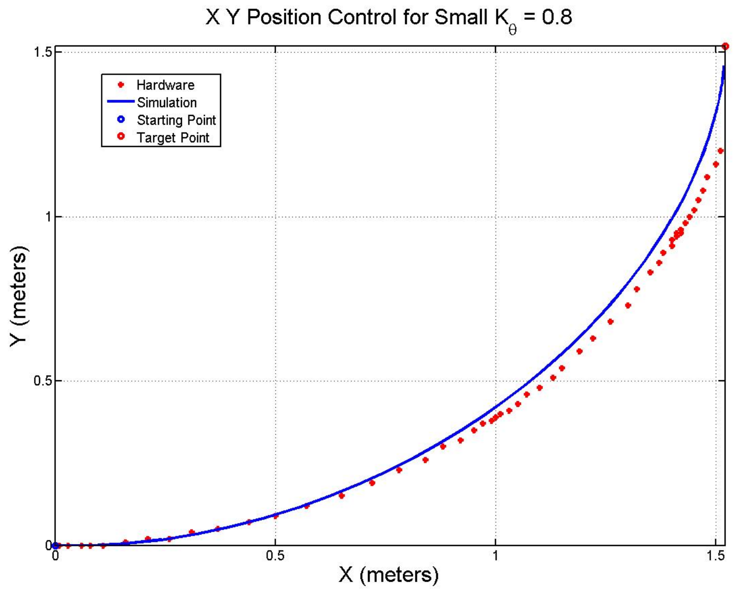

Figure 4.42: Robot Position Control in xy Plane - Cartesian Stabilization (small K_θ $= 0.8$

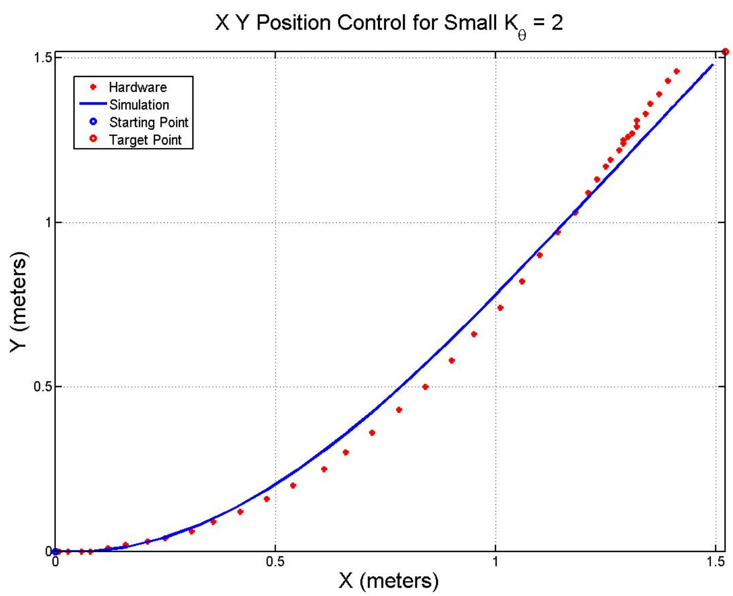

Figure 4.43: Robot Position Control in xy Plane - Cartesian Stabilization (large K_θ $= 2$

From Figure 4.42 and 4.43, we can make the following observations:

- With small K_θ, the trajectory is less directionally aggressive.
- With large K_θ, robot moves more directly towards the target

4.9 Outer Loop Vision Based (v_x, θ) Control - Finish the Oval Track

Block Diagram For Black Line Guidance Robot Lateral Model Outer Loop Design

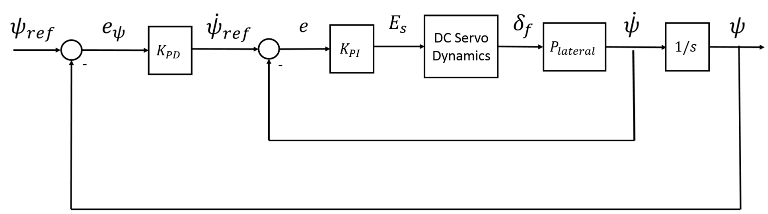

Figure 4.44: Visualization for Vision Based Outer Loop Control System Block Diagram

Vision subsystem is feeding back e_ψ

In this case, e_ψ denotes the angle deviation between the black track (center of gravity of the black area in camera's region of interest) and the orientation of the robot. In this case, we can obtain e_ψ directly from vision subsystem.

$$e_\psi = \psi_{ref} - \psi \tag{4.52}$$

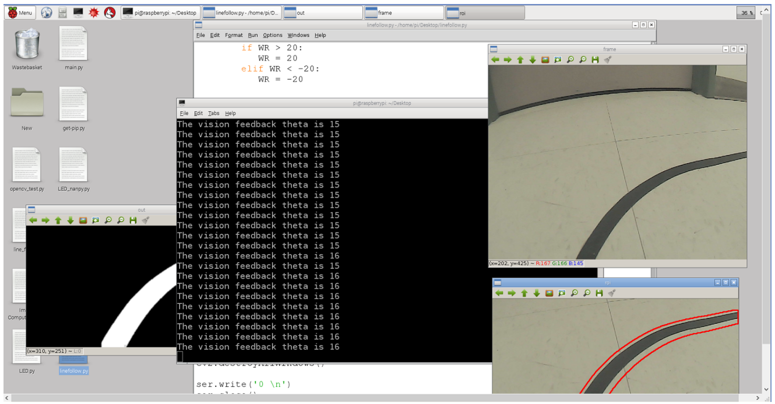

Figure 4.45: Feedback Black Line Tracking Error in Degrees

Simplified Block Diagram for Vision Based Lateral Outer Loop Control

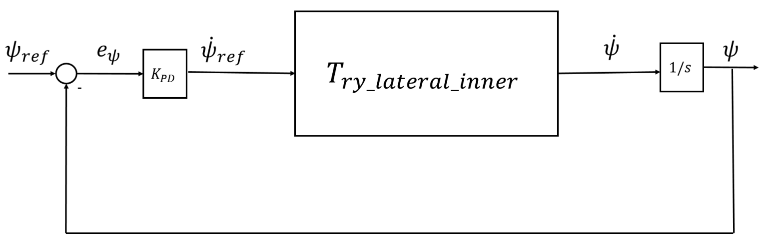

Figure 4.46: Simplified Block Diagram for Vision Based Lateral Outer Loop Control

Transfer Functions

Transfer Function from $\dot{\psi}_{ref}$ to $\dot{\psi}$ (without pre-filter W):

$$T_{ry} = \frac{\dot{\psi}}{\dot{\psi}_{ref}} = \left[\frac{662.4(s+1.2)(s+0.484)}{(s+92.88)(s+6.94)(s+1.229)(s+0.4857)}\right] \tag{4.53}$$

$$= \frac{662.4s^2+1115s+384.7}{s^4+101.5s^3+816.3s^3+1165s+384.7} \tag{4.54}$$

Plant for Outer Loop Controller Design

$$
\begin{aligned}
T_{plant} &= \frac{\psi}{\dot{\psi}_{ref}} && (4.55)\\
&= \left[\frac{662.4(s+1.2)(s+0.484)}{s(s+92.88)(s+6.94)(s+1.229)(s+0.4857)}\right] && (4.56)
\end{aligned}
$$

Known the plant we have, we can now design a plant based outer loop controller. First, we put the inverse of the plant in controller K

$$
K = GAIN\frac{(s+92.88)(s+6.94)(s+1.229)(s+0.4857)}{(s+1.2)(s+0.484)} \tag{4.57}
$$

Here we design a P controller with roll-off and pre-filter. The controller has the form (P plus 3rd order roll-off):

$$
K_{outer} = g\frac{(s+92.88)(s+6.94)(s+1.229)(s+0.4857)}{(s+1.2)(s+0.484)}\left[\frac{100}{s+100}\right]^3 \tag{4.58}
$$

$$
K_{outer} \approx 100g(s+6.94)\left[\frac{100}{s+100}\right]^2 \tag{4.59}
$$

Because we are using P controller here, notice that $K_p = g$. Actually, after pole-zero cancellation, the controller K can be approximated to be a standard PD controller.

In what follows, $L = PK = KP$ denotes the open loop transfer function, $S = (1+L)^{-1}$ denotes the closed loop sensitivity transfer function. $T = L(1+l)^{-1}$ denotes the closed loop complementary sensitivity transfer function, KS denotes the transfer function from (unfiltered) reference commands $\dot{\psi}_{ref}$ to controls ψ_{ref} (which is the reference command for lateral inner loop model) , and SP denotes the transfer function from input disturbances to the robot orientation ψ. Because we are using P controller here, we now only examine trade studies for gain g variations.

First, let us analysis the open loop transfer function $L = PK$. When g is varied (g is from 0.001-0.005), we make the following observations:

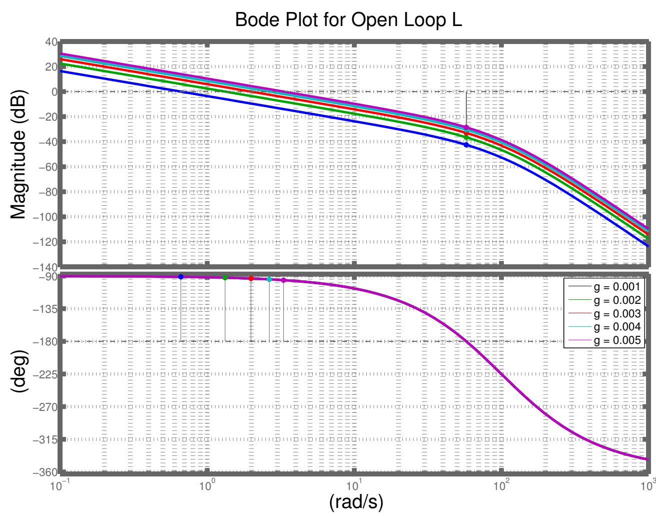

Figure 4.47: Bode Plot for Open Loop L

From Figure 4.47, we can make the following analyses:

- when controller gain $g = 0.001$, we have a proper $0dB$ crossover frequency which is around 0.662 rad/s. Using the relationship $f = 2\pi\omega$, the open loop system has a frequency of 4.16 Hz, which matches the hardware bandwidth limitations we have mentioned in the obvious chapter (pi camera vision subsystem has a maximum bandwidth of 8.1 Hz, this is the main restriction here).

- The bandwidth of the system increases while the gain g is increasing

From Reference Command to Output T_{ry}: Magnitude Responses When g is varied (g is from 0.001-0.005), Figure 1.32 obtains the closed loop T_{ry} bode magnitude responses.

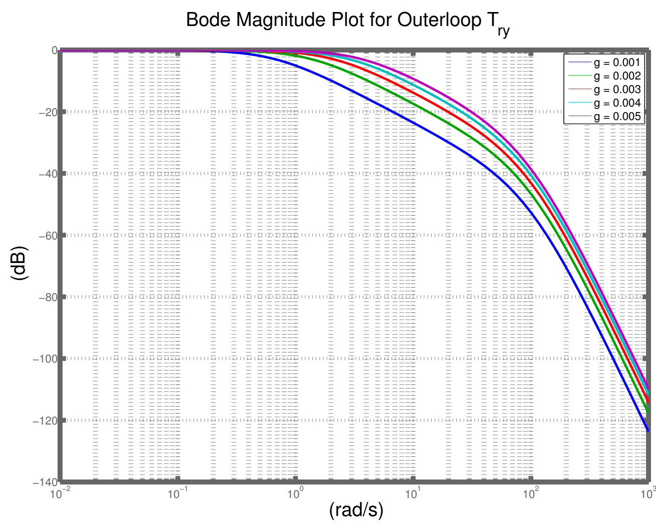

Figure 4.48: Bode Magnitude Plot for Outerloop T_{ry}

From figure 4.48, we can make the following observations:

- System bandwidth increases with a increasing g
- when g is 0.001, system has bandwidth of 0.67 rad/s, which matches the hardware result

From Reference Command to Output T_{ry}: Time domain step Responses When g is varied (g is from 0.001-0.005), one obtains the closed loop T_{ry} step responses in Figure 1.33.

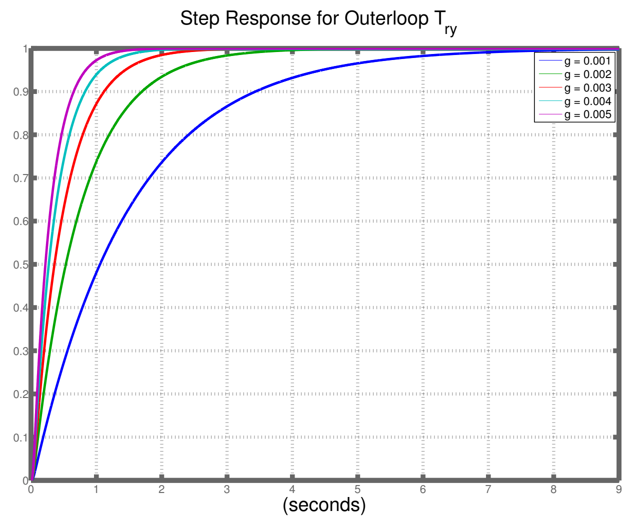

Figure 4.49: Step Response for Outerloop T_{ry}

From the step response Figure 4.49, here we make the following observations:

- With controller proportional gain g increasing, the step responses do not have any overshot
- Settling time decreases as the gain g increasing
- With a g of value 0.001, settling time to with 10% is 3.2s

From Reference Command ψ_{ref} to control $\dot{\psi}_{ref}$ - T_{ru}: Magnitude Responses

When g is varied (g is from 0.001-0.005), Figure 4.50 obtains the closed loop T_{ru} bode magnitude responses.

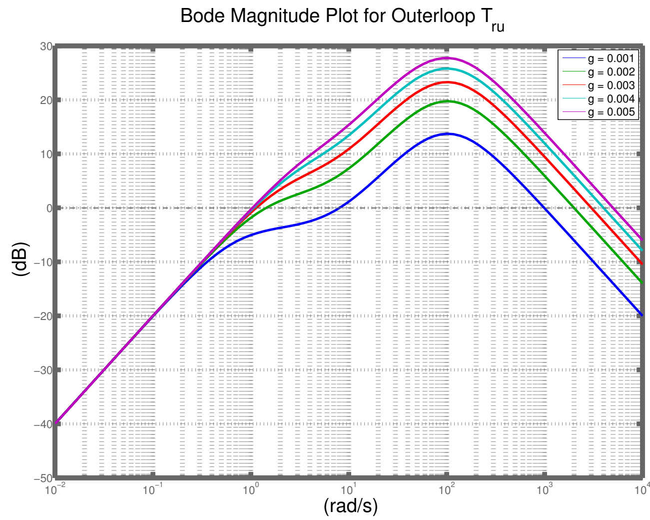

Figure 4.50: Bode Magnitude Plot for Outerloop T_{ru}

From the step response Figure 4.50, here we make the following observations:

- Increasing gain g will increase the peak T_{ru} at all except low frequencies
- Increasing g increases peak T_{ru}: 13.7 (g =0.001), 27.7dB (g = 0.005)

Sensitivity Figures 4.51 contains sensitivity bode magnitude values for specific g variations.

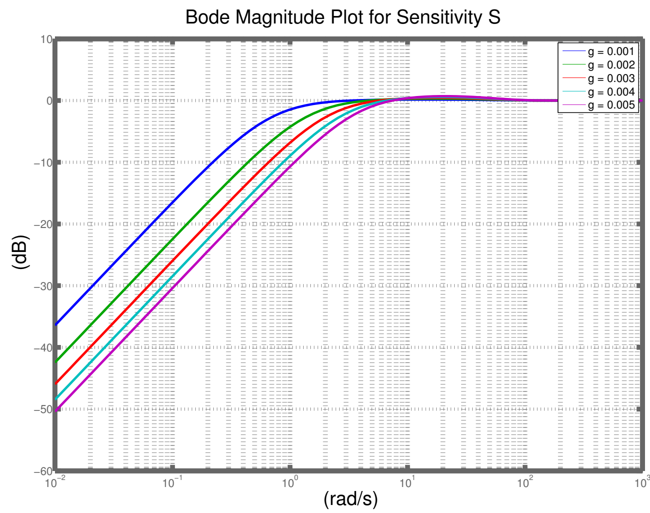

Figure 4.51: Bode Magnitude Plot for Sensitivity S

From the sensitivity S bode magnitude values, here we make the following observations:

- Increasing g results in smaller sensitivity at low frequencies and a slightly larger peak sensitivity.
- peak sensitivities do not change much with increasing g: 0.705 dB (g = 0.005), 0.155 dB ($g = 0.001$)

4.9.2 On Ground Lateral Model Outer Loop Controller Design

This outer loop design is based on on ground lateral model inner loop design. After implementing PI controller, lateral inner loop T_{ry} has two complex poles which are near real pole ($s = -3.3$). To simplify the problem, we estimate lateral inner loop T_{ry} as standard first order system. In the aspect of outer loop, outer loop plant can be estimated as lateral inner loop T_{ry} with an integrator.

$$T_{plantest} \approx \frac{3.3}{s(s+3.3)} \tag{4.60}$$

To meet the need of rapid response of the system, we used root locus approach to design a PD controller. We put a zero at z = -2. Here is the PD controller: Kp = 1.2, Kd = 0.6 (which means g = 1.2 z = 2).

After closing the loop, we have T_{ry} and T_{ru} for closed lateral outer loop system:

$$T_{ry} = \frac{1.98(s+2)}{(s+0.9)(s+4.375)} \tag{4.61}$$

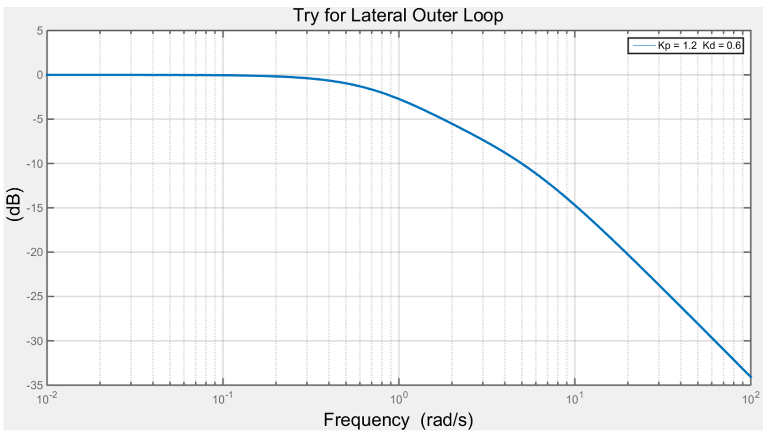

Figure 4.52: T_{ry} for Lateral Outer Loop

- ω_n of outer loop is around 0.8 rad/s, which is smaller than inner loop bandwidth

$$T_{ru} = \frac{0.6(s+3.3)(s+2)}{(s+0.9)(s+4.375)} \tag{4.62}$$

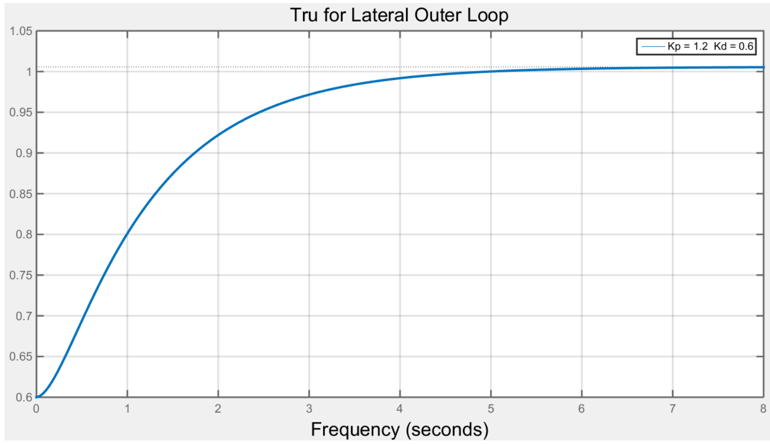

Figure 4.53: T_{ru} for Lateral Outer Loop

- steady state of T_{ru} is 1.08
- settling time of simulated T_{ru} is 5 seconds (which means this step response is slow). However, with this big settling time, robot can still finish track following tasks, so the outer loop comptroller design is successful

4.10 Complete Lateral Model for FreeSLAM Robot - Lateral Model with Pi Camera Vision Subsystem

First of all, let us recall the complete model we've mentioned in Chapter 3.

$$\dot{x} = Ax + Bu + E\omega \tag{4.63}$$

$$y = Cx + Du + F\omega \tag{4.64}$$

The state $x = [v_y, \dot{\psi}, y_L, \varepsilon_L]^T$ and control input $u = \delta_f$, and disturbance $\omega = K_L$. Here is the state space equations for the complete dynamic model:

$$\begin{bmatrix} \dot{v}_y \\ \ddot{\psi} \\ \dot{y}_L \\ \dot{\varepsilon}_L \end{bmatrix} = \begin{bmatrix} -\frac{c_f+c_r}{mv_x} & -v_x + \frac{c_r l_r - c_f l_f}{mv_x} & 0 & 0 \\ \frac{-l_f c_f + l_r c_r}{I_\psi v_x} & -\frac{l_f^2 c_f + l_r^2 c_r}{I_\psi v_x} & 0 & 0 \\ -1 & -L & 0 & v_x \\ 0 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \\ y_L \\ \varepsilon_L \end{bmatrix} + \begin{bmatrix} \frac{c_f}{m} \\ \frac{lfc_f}{I_\psi} \\ 0 \\ 0 \end{bmatrix} \delta_f + \begin{bmatrix} 0 \\ 0 \\ 0 \\ v_x \end{bmatrix} K_L$$

There are two subsystems in this whole complete model. The first one is the on-board vehicle sensors subsystem, where inertial sensors (9 DOF IMU and encoders) are used for measuring lateral acceleration $\ddot{y} = (\dot{v}_y + v_x\dot{\psi})$ and the yaw rate $\dot{\psi}$. Meanwhile, the vision subsystem estimates y_L and ε_L. The road curvature K_L is working as a exogenous disturbance signal.

The output equations have following form:

$$y = \begin{bmatrix} -\frac{c_f+c_r}{mv_x} & \frac{c_r l_r - c_f l_f}{mv_x} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \\ y_L \\ \varepsilon_L \end{bmatrix} + \begin{bmatrix} \frac{c_f}{m} \\ 0 \\ 0 \\ 0 \end{bmatrix} \delta_f$$

4.11 Plot Analysis

Like what we have done in Chapter 3, the core of the Matlab plot analysis lies in the understanding of the behavior of the vehicle at various speeds (the complex nonlinear model can be linearized at different cruise speed V_x), under various road conditions. Then, we analysed how different look-ahead distance L affects the dynamic behavior of the vehicle. Besides, the delay of vision subsystem is very important too.

4.11.1 Main Open Loop Transfer Functions

Here are our FreeSLAM Robot's working condition when it's performing wireless mapping: ideally, the cruise speed of robot vx is 0.1 m/s with a fixed pi camera look-ahead distance L which is roughly 0.1 m (10cm). Besides, in the situation we're talking about here, the process delay of camera vision subsystem is not taken into consideration.

So in this situation, the main transfer functions are:

Transfer function $V_1(s)$ and $V_2(s)$ are sharing the same denominator $P(s)$.

$$P_f s(s) = 0.0002205s^4 + 0.0003381s^3 + 0.0002708s^2 \tag{4.65}$$

$$V_{1fs}(s) = \frac{y_L}{\delta_f} = \frac{0.06183s^2 + 0.04275s + 0.01781}{s^4 + 1.534s^3 + 1.228s^2} \tag{4.66}$$

$$V_{2fs}(s) = \frac{\varepsilon_L}{\delta_f} = \frac{0.368s + 0.1781}{s^3 + 1.534^2 + 1.228s} \tag{4.67}$$

According to the transfer functions above, in the next secsection, we are going to talk about the Matlab Plot Analysis.

Robot cruise speed V_x

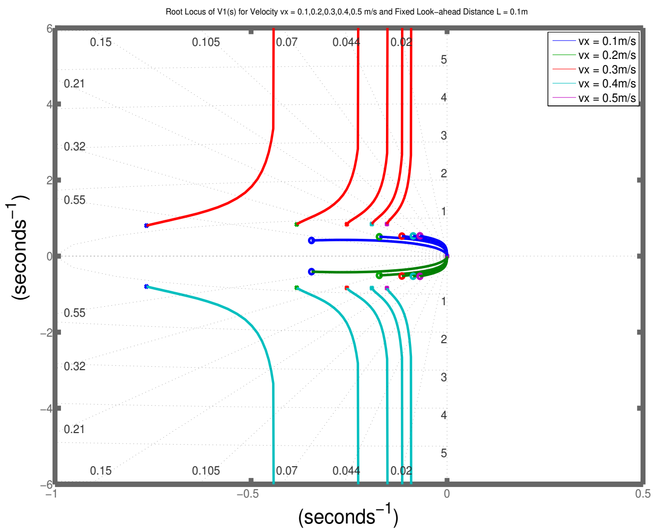

Figure 4.54: Root Locus of V1(s) for Varying Cruise Speed V_x and Fixed Look-Ahead Distance L = 0.1m

Figure 4.54 Analysis: As the root locus of $V1(s)$ shows, overall, the double integrator at the origin corresponds to the integration action between lateral acceleration and position at the look-ahead. The two poles and zeros in the left half plane characterize the vehicle dynamics.

By increasing the cruise speed V_x, both two poles and two zeros in the left half plane are moving towards to the imaginary axis.

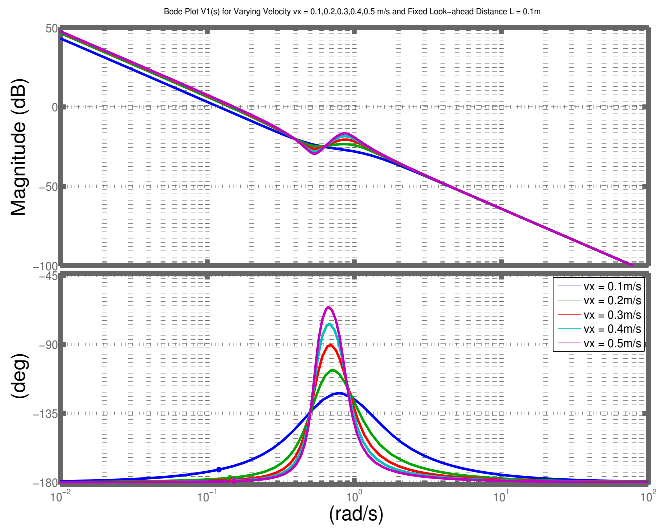

Figure 4.55: Bode Plot of V1(s) for Varying Cruise Speed V_x and Fixed Look-Ahead Distance L = 0.1m

Figure 4.55: Bode plot V1(s) for varying cruise speed $V_x = 0.1,0.2,0.3,0.4$ and 0.5 m/s with a fixed camera look-ahead distance 0.1m and no vision subsystem delay. It shows that increasing the cruise speed V_x will decrease the Phase Margin (PM). Under the condition that cruise speed $V_x = 0.5$m/s (maximum speed in the plot), the Phase Margin (PM) is only 0.774 degrees which is not good.

Hardware Result

We collect the cruise speed V of robot by using encoders (0.06 m/s resolution) and orientation of robot by 9 dof IMU (0.01 rad resolution).

By using real-wheel drive kinematic model,

$$V_x = V * cos(\theta); \tag{4.68}$$

$$V_y = V * sin(\theta); \tag{4.69}$$

I integrated the V_x and V_y Speeds to get the position information (X,Y). then plot (X,Y) to get the real trajectory plot as follows:

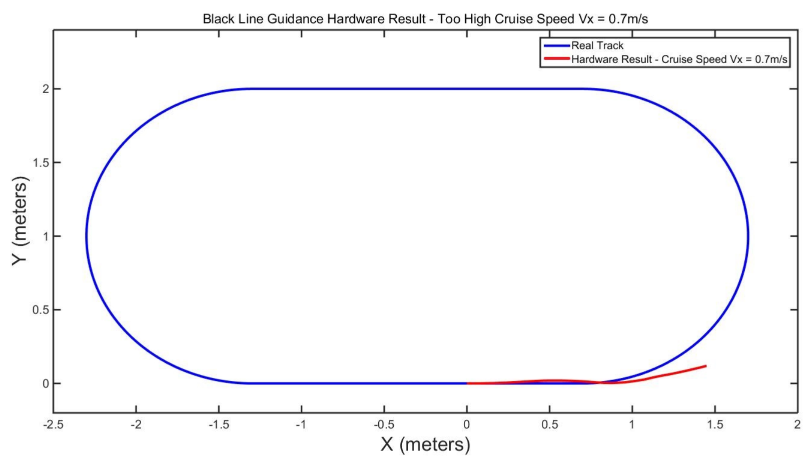

Figure 4.56: Robot Goes Off the Track Due to Too High Speed

As Figure 4.56 shows, with a commanded cruise speed of 0.7 m/s, robot goes off the track because of the too high commanded cruise speed V_x.

Camera Fixed Look-Ahead Distance L

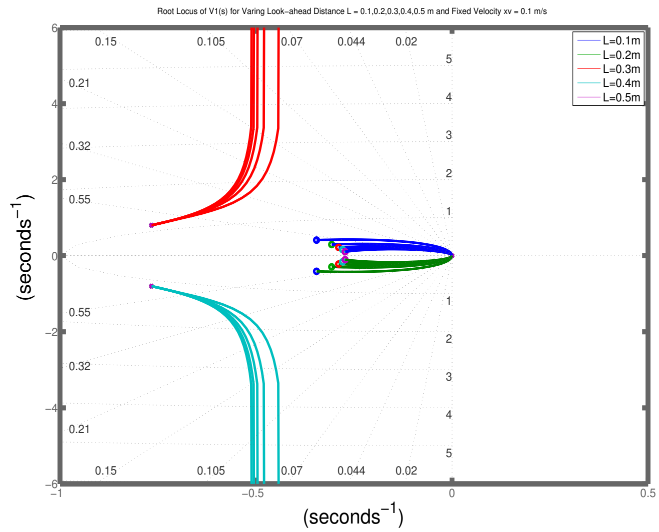

Figure 4.57: Root Locus of V1(s) for Varying Look-Ahead Distance L and Fixed Cruise Speed $V_x = 0.1\ m/s$

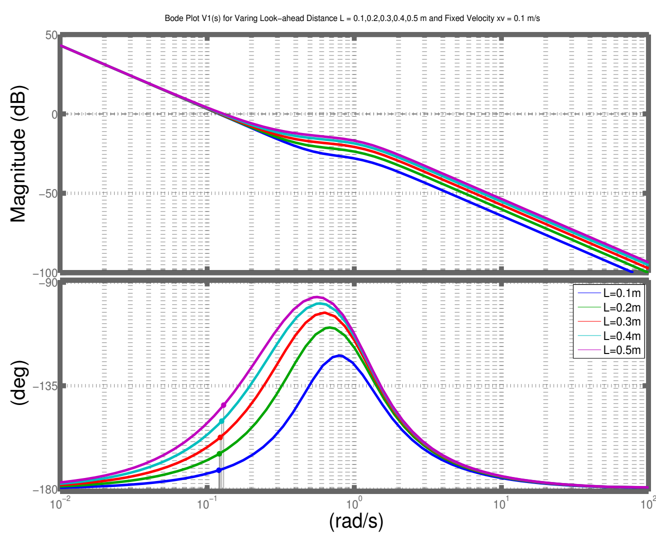

Figure 4.58: Bode Plot of V1(s) for Varying Look-Ahead Distance L and Fixed Cruise Speed $V_x = 0.1\ m/s$

Figure 4.58 Analysis: As we can observe here, the farther the pi camera looks, the more Phase Margin (PM) the vision lateral system will increase, as a result, the whole system is becoming more stable. With a cruise longitudinal speed $v_x = 0.1m/s$ and camera look-ahead distance is $0.5m$, the lateral system's phase margin will increase to 36.6 degrees which is good to our system.

Hardware Result Here we generate the trajectory of robot when it is applied a small camera look-ahead distance L = 0.1m.

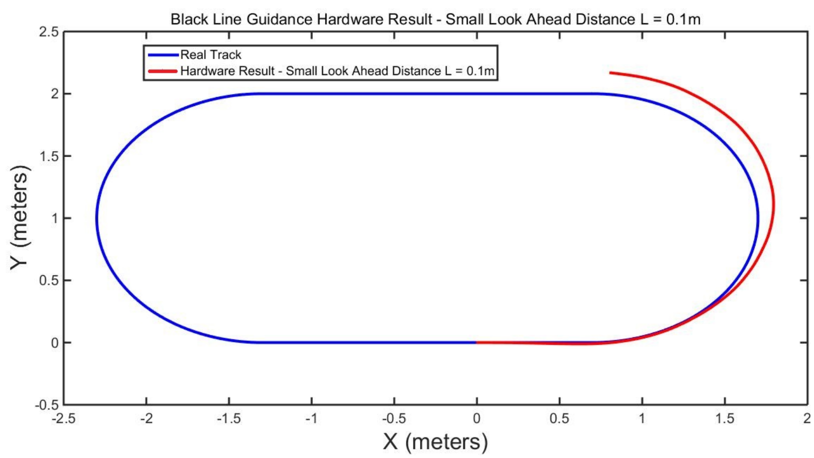

Figure 4.59: Trajectory of Robot When Small L is Applied

As we can see, robot has the ability to follow the track well at the very beginning, but it goes off the track in the end because of the too small camera look-ahead distance L.

Delay from Vision Subsystem T_d

As we have mentioned in the simulations, one important parameter which will effect the overall system is the delay associated with the latency of visual processing. As shown in the overall system block diagram, the component is a pure time delay element $e^{-T_d s}$ representing the latency T_d of the vision subsystem. Using $Pade Approximation$ and this delay component becomes:

$$D(s) = e^{-T_d s} \approx \frac{2 - T_d s}{2 + T_d s} \tag{4.70}$$

$V_1(s)D(s)$ demonstrate the effect of vision subsystem latency.

Under certain condition:

$$D(s) = \frac{-0.5s + 2}{0.5s + 2} \tag{4.71}$$

And here is the nominal transfer function (using nominal parameters):

$$V_1(s)D(s) = \frac{y_L}{\delta_f} = \frac{-0.06183s^3 + 0.2046s^2 + 0.1532s + 0.07124}{s^5 + 5.534s^4 + 7.362s^3 + 4.912s^2} \tag{4.72}$$

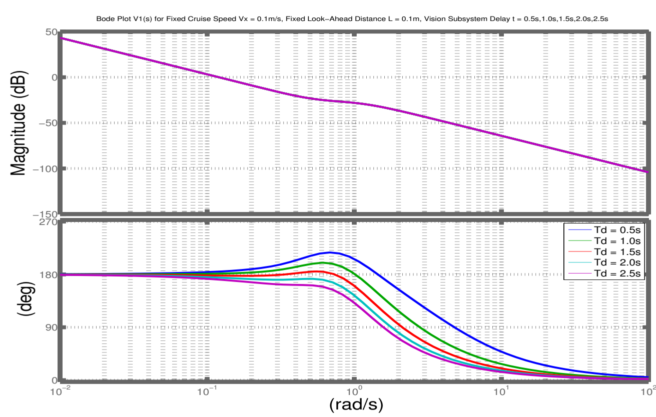

Figure 4.60: Bode Plot of V1(s)D(s) for Cruise Speed V_x = 20m/s, Look-Ahead Distance L = 15m and Vision Subsystem Delay t = 0.15s

We studied $V1(s)D(s)$ under the following situation, we fixed the cruise speed vx with a fixed camera look-ahead distance and vary the delay of vision subsystem. As we can observe above, at beginning, when the delay is as small as 0.5 seconds, the plant has a phase margin of 4.82 degrees. However, when we start increase the delay, the system is becoming unstable. For example, when delay has been increased to 2.5 seconds, the phase margin of the system is -8.82 degrees, which shows that the plant is unstable.

Hardware Result Applying a delay $T_d = 0.1s$ for vision subsystem delay, we can make the following observations:

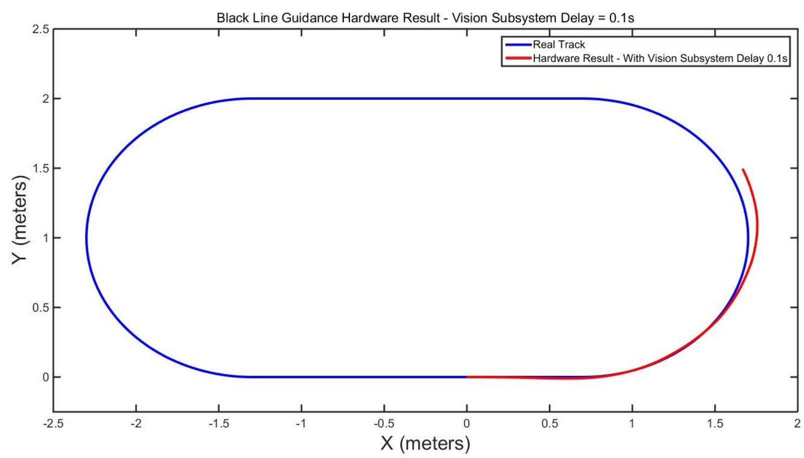

Figure 4.61: Trajectory of Robot When Vision Delay is 0.1s

Implementing delay in vision subsystem means that we are lowering the lateral outer loop frequency. In this case, by applying a delay of $0.1s$, lateral outer loop frequency has dropped from 7.5 Hz to 4.29Hz. That's the main reason why robot goes off the track.

When we increase the delay T_d from 0.1s to 0.15s, we can make the following observations:

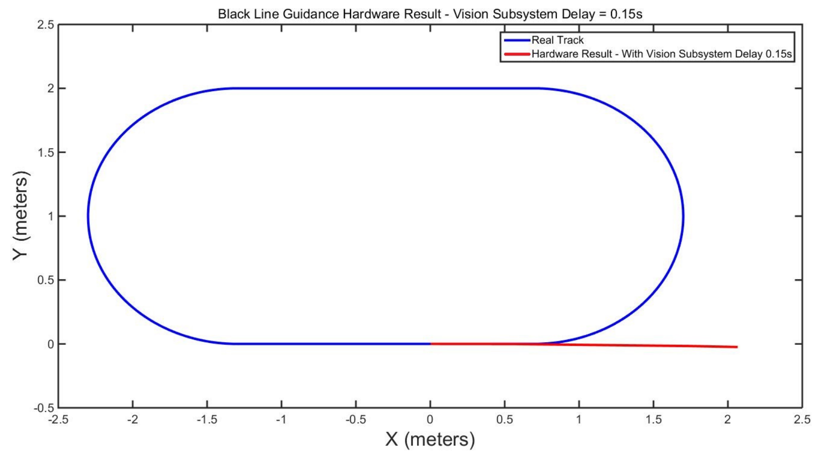

Figure 4.62: Trajectory of Robot When Vision Delay is 0.15s

To be more specific, when we apply 0.15s delay to the vision system, the lateral outer loop frequency has dropped from 7.5Hz to 3.53Hz, which makes the system stability worse (phase margin is negative). That's the main reason why robot goes off the track at the very beginning.

4.12 Finish the Track in Minimum Time - With/Without Pan Servo

As we know, camera losing track is the one of the key reasons that cause robot loses the track. Here we introduce a pan-tilt structure. In this case, under the circumstances that robot is trying to turn sharp curves (ψ_{error}) is too large, the pan servo (controlled by P controller) will pan the camera to reduce ψ_{error} to make the system remain stable.

The followings are the trajectories for robot finish the track without and with pan servo.

Robot Finish the Track without Pan Servo

Robot Finish the Track without Pan Servo with a minimum time of 24.3 seconds.

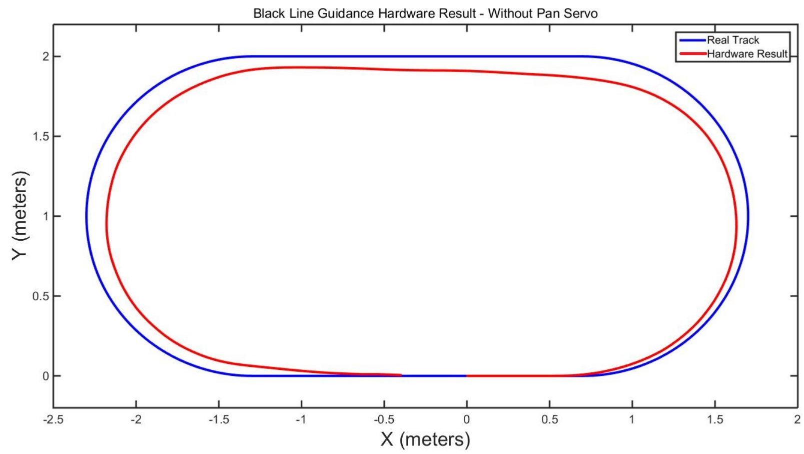

Figure 4.63: Robot Finish the Track without Pan Servo in 24s

Here is the plot for ψ_{error} changing:

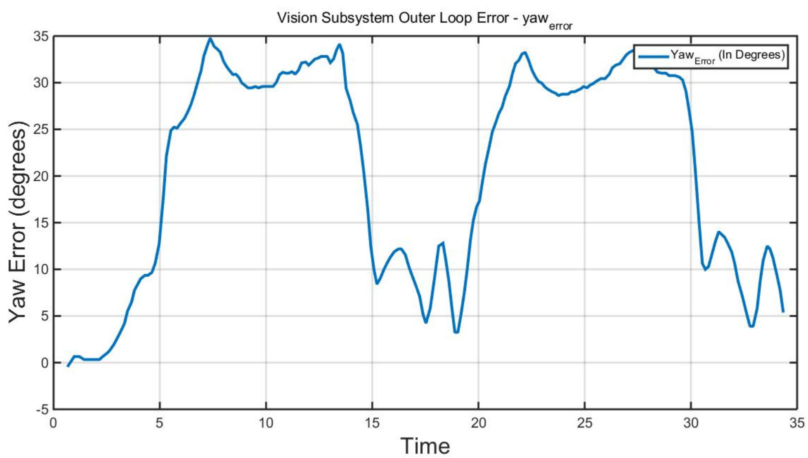

Figure 4.64: ψ_{error} Changing with Time without Implementing Pan Servo

we can make the following observations: when robot is trying to turn sharp turns, the ψ_{error} obtained from vision subsystem can reach a maximum of 35 degrees. This phenomenon can cause track losing easily. so a pan servo implementation is necessary.

Robot Finish the Track with Pan Servo with a minimum time of 19.8 seconds.

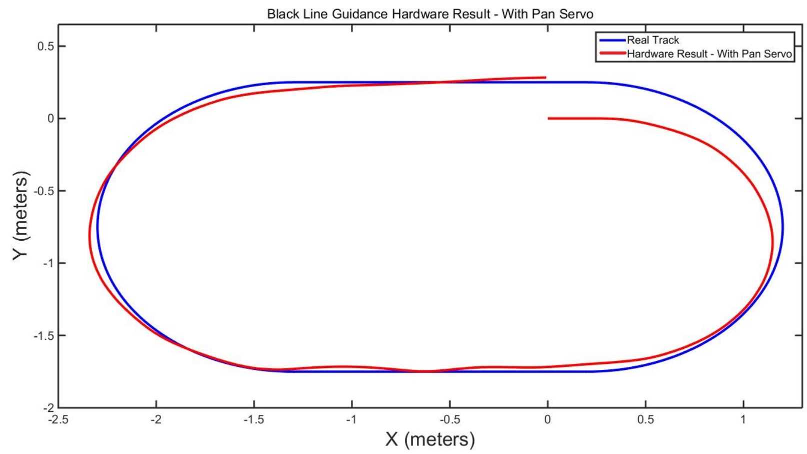

Figure 4.65: Robot Finish the Track with Pan Servo in 20s

Here is the plot for ψ_{error} changing and pan servo steering performance along with time:

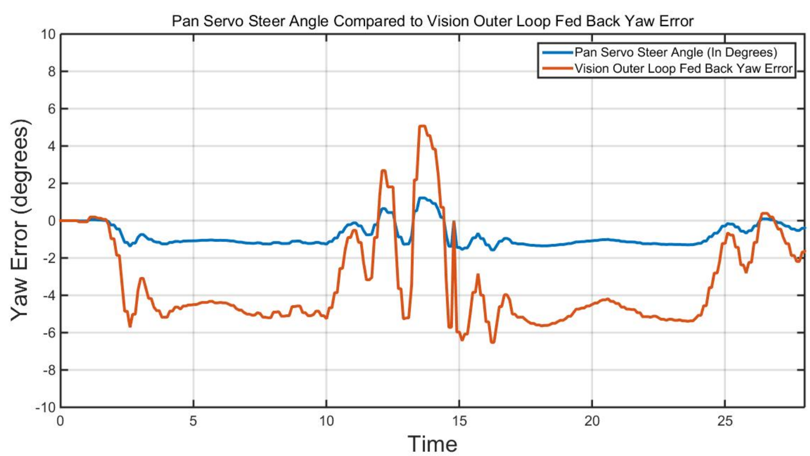

Figure 4.66: Yaw Error and Pan Servo Steer Changing with Time with Implementing Pan Servo

We can make the following observations according to the plot above. After implementing the pan servo, ψ_{error} can be easily controlled to around 0 degrees (with a max ripple of 3 degrees). To draw a brief conclusion, the pan structure contributes to the stabilization of the whole vision based system.

4.13 Summary and Conclusion

This chapter has provided a comprehensive case study for our enhanced rear-wheel drive FreeSLAM 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

SLAM WITH LIDAR SCAN DATA ONLY - HECTOR MAPPING

5.1 Introduction to SLAM (Simultaneous localization and mapping)

Definition of SLAM problem SLAM is the abbreviation for Simultaneous Localization And Mapping.

Mapping is the problem of integrating the information gathered with the robot's sensors into a given representation. It can be described by the question "What does the world look like?" Central aspects in mapping are the representation of the environment and the interpretation of sensor data. In contrast to this, localization is the problem of estimating the pose of the robot relative to a map. In other words, the robot has to answer the question, "Where am I?" Typically, one distinguishes between pose tracking, where the initial pose of the vehicle is known, and global localization, in which no a priory knowledge about the starting position is given.

Simultaneous localization and mapping (SLAM) is therefore defined as the problem of building a map while at the same time localizing the robot within that map. In practice, these two problems cannot be solved independently of each other. Before a robot can answer the question of what the environment looks like given a set of observations, it needs to know from which locations these observations have been made. At the same time, it is hard to estimate the current position of a vehicle without a map. Therefore, SLAM is often referred to as a chicken and egg problem: A good map is needed for localization while an accurate pose estimate is needed to build a map.

So, why is SLAM problem hard?

It's a chicken and egg problem

- a map is needed to localize the robot
- a pose estimate is needed to build a map

Mathematical Expression of SLAM Problem

To estimate the pose and the map of a mobile robot at the same time

$$p(x, m|z, u) \tag{5.1}$$

where x denotes the estimated pose of the robot, m is the grid map. z represents the observations (in this case it is the LIDAR scan data) and u denotes controls.

5.2 System Overview

The ability to learn a model of the environment and to localize itself is one of the most important abilities of truly autonomous robots able to operate within real world environments. In this chapter, we present a flexible and scalable system for solving the SLAM (Simultaneous Localization and Mapping) problem that has successfully been used on unmanned ground vehicles (UGV). Our approach uses the ROS jade operating system as middle-ware and is available as open source software. It honors the API of the the ROS navigation stack and thus can easily be interchanged with other SLAM approaches available in the ROS ecosystem.

- **360 RP LiDAR**

The RPLIDAR 360 Laser Scanner is a low cost 360 degree 2D scanner (LIDAR) solution. It preforms 360 degree laser scanning with more than 6 meters distance detection range. The produced 2D point cloud data can be used in mapping, localization (SLAM) and object/ environment modeling.

RPLIDAR emits a modulated infrared laser signal and the laser signal is then reflected by the object to be detected. The returning signal is sampled by vision acquisition in RPLIDAR and the DSP embedded in RPLIDAR starts processing the sample data, output distance value and angle value between the object and the RPLIDAR. Through processing the sample data is output through a communication interface.

- **ROS**

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

As a result, ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work, as is described throughout this site.

- **ROS node**

A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provide a graphical view of the system, and so on.

5.3 Hector SLAM Approach

5.3.1 Hector SLAM Requirements

Generally speaking, Hector SLAM includes the following four aspects: (1) Map the unknown environment, (2) Localize robot simultaneously, (3) Real-time capable, (4) Saving GeoTiff maps.

5.3.2 Hector Mapping-ROS API

The main SLAM node I am using is Hector Mapping.

- **Main inputs**

There are two main inputs:the first is wireless transported LiDAR scan data on the "/scan" topic. The second is transformed data via node tf.

- **Main outputs**

There are two main outputs: the first is map on the "/map" topic, while the other one is real-time position of robot in the map(using tf node "map"→"odom" transform)

5.3.3 Whole picture of Hector SLAM

Figure 5.1 shows ROS nodes connections and communication. When robot performs indoor SLAM, ROS nodes, topics and services are well visualized in the following figure:

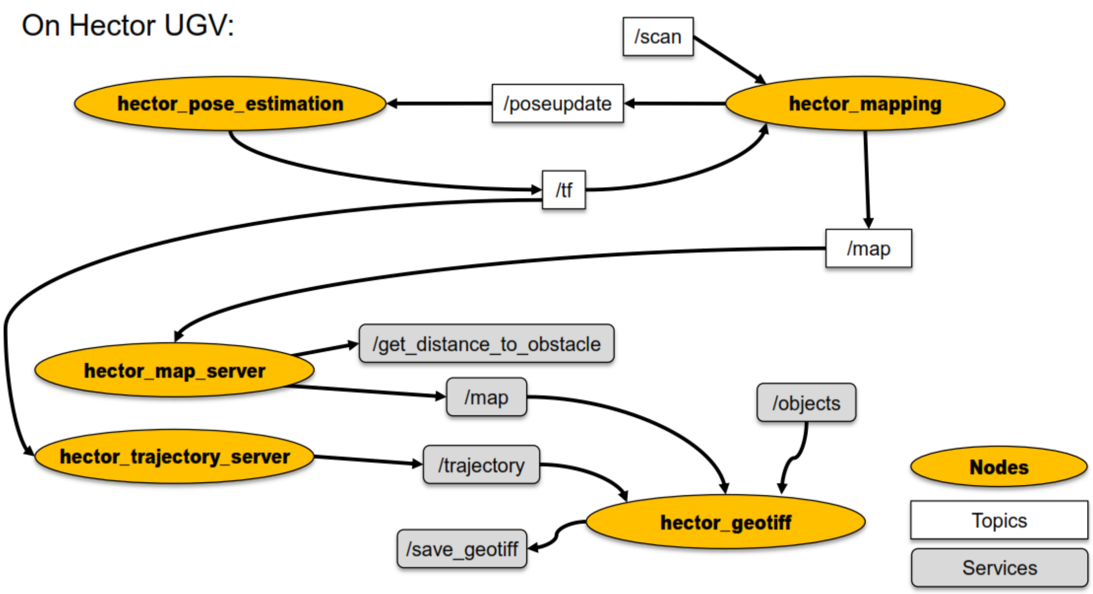

Figure 5.1: Big Picture Of Hector SLAM

5.3.4 Coordinate Frames

map

The coordinate frame called map is a world fixed frame, with its Z-axis pointing upwards. The pose of a mobile platform, relative to the map frame, should not significantly drift over time. The map frame is not continuous, meaning the pose of a mobile platform in the map frame can change in discrete jumps at any time.

In a typical setup, a localization component constantly re-computes the robot pose in the map frame based on sensor observations, therefore eliminating drift, but causing discrete jumps when new sensor information

arrives.

The map frame is useful as a long-term global reference, but discrete jumps make it a poor reference frame for local sensing and acting.

odom

The coordinate frame called odom is a world-fixed frame. The pose of a mobile platform in the odom frame can drift over time, without any bounds. This drift makes the odom frame useless as a long-term global reference. However, the pose of a robot in the odom frame is guaranteed to be continuous, meaning that the pose of a mobile platform in the odom frame always evolves in a smooth way, without discrete jumps.

In a typical setup the odom frame is computed based on an odometry source, such as wheel odometry, visual odometry or an inertia measurement unit.

The odom frame is useful as an accurate, short-term local reference, but drift makes it a poor frame for long-term reference.

base_link

The coordinate frame called base_link is rigidly attached to the mobile robot base. The base_link can be attached to the base in any arbitrary position or orientation; for every hardware platform there will be a different place on the base that provides an obvious point of reference. Note that REP 103 [1] specifies a preferred orientation for frames.

Relations Between Frames

We have chosen a tree representation to attach all coordinate frames in a robot system to each other. Therefore each coordinate frame has one parent coordinate frame, and any number of child coordinate frames. The frames described in this REP are attached as follows:

map − > odom − > base_link

The map frame is the parent of odom, and odom is the parent of base_link. Although intuition would say that both map and odom should be attached to base_link, this is not allowed because each frame can only have one parent.

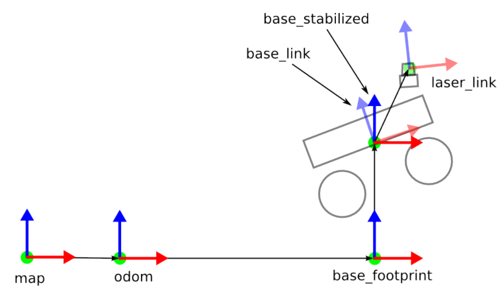

Figure 5.2: Big Picture Of Hector SLAM

- "/odom" frame is not needed, which is mainly for compatibility with ROS gmapping
- "/base_stabilized" frame is needed for transformation of LIDAR data
- height estimation is not trivial

5.4.1 SLAM Problem Model and Parameters Definition

What have been given

The robot's controls

$$u_{1:T} = u_1, u_2, u_3..., u_T \tag{5.2}$$

Here we introduce the $Standard\ OdometryModel$. Say we have a robot moving from $(\bar{x}, \bar{y}, \bar{\theta})$ to $(\bar{a}', \bar{y}', \bar{\theta}')$, we are using $(\bar{x}, \bar{y}, \bar{\theta})$ here because the (x, y) coordinates and the orientation of the robot are estimated.

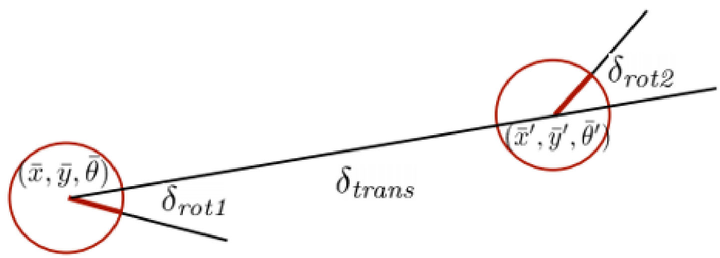

Figure 5.3: Standard Odometry Model

Then here we have the $Odometry\ Information$ $u = (\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ and the following equations:

$$\delta_{trans} = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2} \tag{5.3}$$

$$\delta_{rot1} = atan2(\bar{y} - \bar{y}', \bar{x}' - \bar{x}) - \bar{\theta} \tag{5.4}$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1} \tag{5.5}$$

Robot Observations

$$z_{1:T} = z_1, z_2, z_3..., z_T \tag{5.6}$$

Here we have the observation or sensor (encoder, IMU or LIDAR) model with the robot's pose.

$$p(z_t \mid x_t) \tag{5.7}$$

What do we want finally: map of the environment m.

Path (Trajectory) of the robot

$$x_{0:T} = x_1, x_2.x_3..., x_T \tag{5.8}$$

Then finally we estimate the robot's trajectory and the grid map.

$$p(x_{0:T}, m_{1:T}, u_{1:T}) \tag{5.9}$$

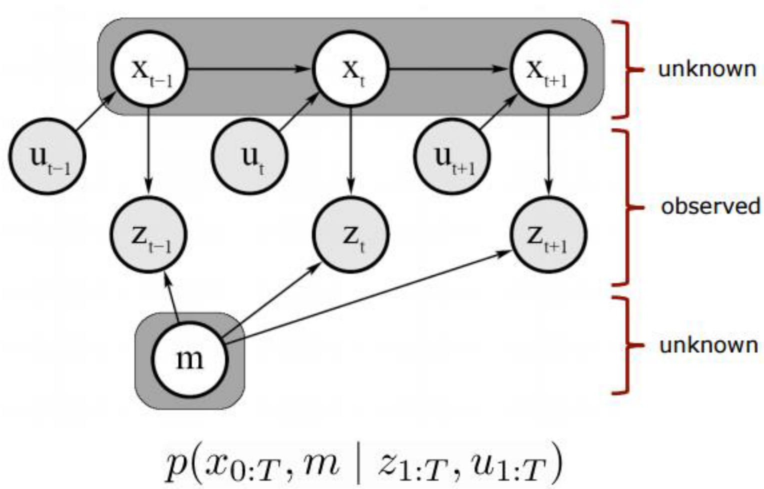

Figure 5.4: Graphic Model of SLAM Problem Approach

Platform Full 3D State

We define the navigation coordinate system as a right handed system having the origin at the starting point of the platform with the z axis pointing upwards and the x axis pointing into the yaw direction of the platform at beginning. The full 3D state is represented by

$$x = (\Omega^T + p^T + v^T)^T \tag{5.10}$$

where $\Omega = (\phi, \vartheta, \psi)^T$ are row, pitch and yaw angles, $P = (p_x, p_y, p_z)^T$ and $v = (v_x, v_y, v_z)^T$ are the position and velocity of the platform expressed in the navigation frame.

5.4.2 Extended Kalman Filter Implementation in Hector Mapping

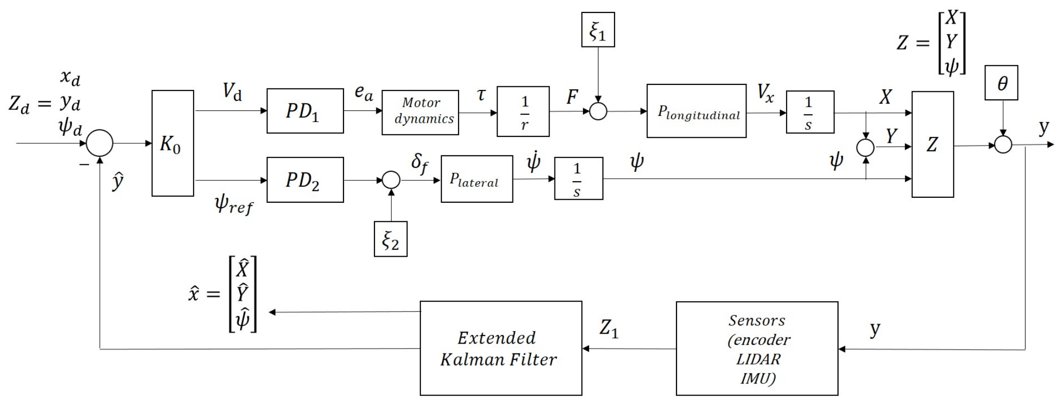

Figure 5.5: Complete Model with Extended Kalman Filter Implementation

The dynamic model can be described as the following:

$$X(k) = AX(k-1) + BU(k) + \Delta(k) \tag{5.11}$$

$$Z(k) = HX(k) + \Theta(k) \tag{5.12}$$

In those equations, besides the parameters (X, Z, U) we have mentioned above, A and B are system parameters,H is the observation system parameter, in this case, they are metrics. Δ and are the noise of process (input noise) and observation (output noise), in assumption they are Gaussian noise and their covariances are Q and R.

Kalman Filter Equations can be represented in the following equations:

$$X(k \mid k-1) = AX(k-1 \mid k-1) + BU(k) \tag{5.13}$$

$$P(k \mid k-1) = AP(k-1 \mid k-1)A^{'} + Q \tag{5.14}$$

$$X(k \mid k) = X(k \mid k-1) + Kg(k)(Z(k) - HX(k \mid k-1)) \tag{5.15}$$

$$Kg(k) = P(k \mid k-1)H^{'}/(HP(k \mid k-1)H^{'} + R) \tag{5.16}$$

$$P(k \mid k) = (I - Kg(k)H)P(k \mid k-1) \tag{5.17}$$

Analysis:

As we can see from the equations above, Kalman Gain $K_g(k)$ increases with a decreasing observe noise covariance R. When the current estimation error covariance decreases, $K_g(k)$ gets larger. To draw a brief conclusion, Kalman gain $K_g(k)$ represents the weight of observe information (for example the laser scan range information from LIDAR or the angular velocity detected by wheel encoders) during the update process. In this case, when the observe noise is smaller, $Z(k)$ gets bigger and $HX(k \mid k-1)$ gets smaller.

Kalman Filter Algorithms Process

1. Set initial system state $X(0)$ and its error covariance $P(0)$, then set noise covariance Q_0, R_0.

2. Using the equations above, calculate estimated state of the system $X(k \mid k-1)$ and estimated covariance $P(k \mid k-1)$

3. According to the updated equations, calculate the kalman gain and updated state estimation $X(k \mid k-1)$ and $P(k \mid k-1)$

4. repeat step (2) and step (3).

5.4.3 Vectors Used in EKF Implementation

robot current state x

Robot's state can be shown as the following vector:

$$x = \begin{bmatrix} X \\ Y \\ \Psi \end{bmatrix} \tag{5.18}$$

To recall the vehicle's kinematics

$$\dot{x} = vcos\Psi$$

$$\dot{y} = vsin\Psi$$

$$\dot{\Psi} = \frac{vtan\Psi}{L}$$

So to analysis this discrete system with a fixed sampling time Δt, the increasement of robot's odometry ($\Delta X\ \Delta Y\ \Delta\Psi$) are:

$$\Delta X = vcos\Psi\Delta t \tag{5.19}$$

$$\Delta Y = vsin\Psi\Delta t \tag{5.20}$$

$$\Delta\Psi = \frac{vtan\Psi}{L}\Delta t \tag{5.21}$$

All the information above is supposed to be generate by the wheel encoder (Hall Effect Sensor with 10 small magnets) and IMU (BON055).

Then, we can have a updated state $x^{'}$:

$$x^{'} = \begin{bmatrix} X^{'} \\ Y^{'} \\ \Psi^{'} \end{bmatrix} = x + \begin{bmatrix} \Delta X = vcos\Psi\Delta t \\ \Delta Y = vsin\Psi\Delta t \\ \Delta\Psi = \frac{vtan\Psi}{L}\Delta t \end{bmatrix}$$

(5.22)

The Kalman Gain Kg The Kalman gain Kg is computed to find out how much we should trust the observed landmarks and as such how much we want to gain from the new information they provide. If we can see from the odometry reading that the robot was moved 2cm to the left, according to the observed landmarks we'll use the Kalman Gain Kg to find out how much we should trust the LIDAR range readings. Finally, it may turn out to be 1 cm cause we do not the landmarks completely. If the range measurement device is really bad compared to the odometry performance of the robot, the Kalman Gain will decrease, otherwise it will increase.

The Jacobian of the measurement model H

The Jacobian of the measurement model is closely related to the measurement model. The measurement model defines how to compute an expected range and bearing of the measurements (observed landmark positions). It is done using the following formula:

$$\begin{bmatrix} RangeBearing \end{bmatrix} = \begin{bmatrix} \sqrt{(\lambda_x - x)^2 + (\lambda_y - y)^2} + \theta_{range} \\ tan^{-1}(\frac{\lambda_y - y}{\lambda_x - x}) - \psi + \theta_{angle} \end{bmatrix} \tag{5.23}$$

where λx is the x position of the landmark, x is the current estimated robot x position, λy is the y position of the landmark and y is the current estimated robot y position. ψ is the robot's yaw angle and θ is the LIDAR data observe noise.

This will give us the predicted measurement of the range and bearing to the landmark. The Jacobian of this matrix with respect to x, y, and θ, then the H is:

$$\begin{bmatrix} \frac{x-\lambda_x}{r} & \frac{y-\lambda_y}{r} & 0 \\ \frac{\lambda_y - y}{r^2} & \frac{\lambda_x - x}{r^2} & -1 \end{bmatrix} \tag{5.24}$$

To draw a brief conclusion, H shows us how much the range and bearing changes as x, y and θ changes.

The Jacobian of the prediction model: A Matrix

Like H, the Jacobian of the prediction model is closely related to the prediction model, of course, so lets go through the prediction model first. The prediction model defines how to compute an expected position of the robot given the old position and the control input.

Jacobian A yieldingL

$$A = \begin{bmatrix} 1 & 0 & -\Delta t sin\theta \\ 0 & 1 & \Delta t cos\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{5.25}$$

The SLAM specific Jacobians : J_{xr} and J_z

When doing SLAM there are some Jacobians which are only used in SLAM. This is of course in the integration of new features, which is the only step that differs from regular state estimation using EKF. The first is J_{xr}. It is basically the same as the jacobian of the prediction model, except that we start out without the rotation term. It is the jacobian of the prediction of the landmarks, which does not include prediction of theta, with respect to the robot state [x, y, theta] from X

$$J_{xr} = J_{xr} = \begin{bmatrix} 1 & 0 & -\Delta t sin\theta \\ 0 & 1 & \Delta t cos\theta \end{bmatrix} \tag{5.26}$$

The jacobian J_z is also the jacobian of the prediction model for the landmarks, but this time with respect to [range, bearing]. This in turn yields:

$$J_z = \begin{bmatrix} cos(\theta + \Delta\theta) & -\Delta t sin(\theta + \Delta\theta) \\ sin(\theta + \Delta\theta) & \Delta t cos(\theta + \Delta\theta) \end{bmatrix} \tag{5.27}$$

5.4.4 2D SLAM Visualization in RVIZ

In this section, we describe how to draw the 2D map of unknown environment using hector SLAM. Bilinear filtering, as the major algorithm, is used to solve this problem.

- Map Access - Grid Map

Grid Map

1. Grid maps are a discretization of the environment into free and occupied cells
2. Mapping with known robot poses is easy. So within this thesis, SLAM approach can be divided into to to steps: The first step is navigation: we estimate the pose of robot accurately by using majorly 2 kinds of low-pass filters(Kalman Filter for Gaussian noise and Particle Filter, which is a beyes filter, for filtering non-Gaussian noise) The localization and mapping problem are combined but

we're supposed to solve localization problem first and then focus on mapping problem.

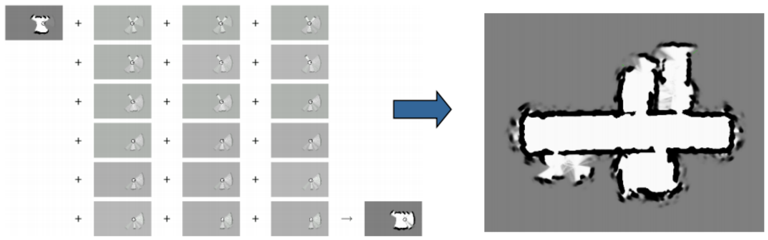

Figure 5.6: 2D Grid Map

grid map is used to represent arbitrary environments. Because LIDAR platform can exhibit6 DOF motion, the scan has to be transformed into a local stabilized coordinate frame usingthe estimated attitude of the LIDAR system.
In this case, the scan is converted into a point cloud of scan endpoints. This point cloud can be preprocessed, for example by down-sampling the number of points or removal of outliers.

Given a continuous map coordinate P_m, the occupancy value $M(P_m)$ as well as the gradient $\bigtriangledown M(P_m) = (\frac{\partial M}{\partial x}(P_m), \frac{\partial M}{\partial y}(P_m))$ can be approximated by using the four closest integer P_{00}, P_{01}, P_{10} and P_{11}. Linear interpolation along the $xaxis$ and $yaxis$ then yields:

$$M(P_m) \approx \frac{y-y_0}{y_1-y_0}(\frac{x_1-x}{x_1-x_0}M(P_{11}) + \frac{x_1-x}{x_1-x_0}M(P_{01})) + \frac{y_1-y}{y_1-y_0}(\frac{x-x_0}{x_1-x_0}M(P_{10}) + \frac{x_1-x}{x_1-x_0}M(P_{00})) \tag{5.28}$$

The derivatives can be approximated by:

$$\frac{\partial M}{\partial x} \approx \frac{y-y_0}{y_1-y_0}(M(P_{11}) - M(P_{01})) + \frac{y_1-y}{y_1-y_0}(M(P_{10}) - M(P_{00})) \tag{5.29}$$

$$\frac{\partial M}{\partial y} \approx \frac{x - x_0}{x_1 - x_0}(M(P_{11}) - M(P_{10})) + \frac{x_1 - x}{x_1 - x_0}(M(P_{01}) - M(P_{00})) \tag{5.30}$$

In this situation, we should point out that the sample points cells are situated on a regular grid with distance 1 (in map coordinates) from each other, so in this case $\frac{y-y_0}{y_1-y_0}$ and $\frac{x-x_0}{x_1-x_0}$ in the equations above approximately equal 1, which simplifies the presented equations for the gradient approximation.

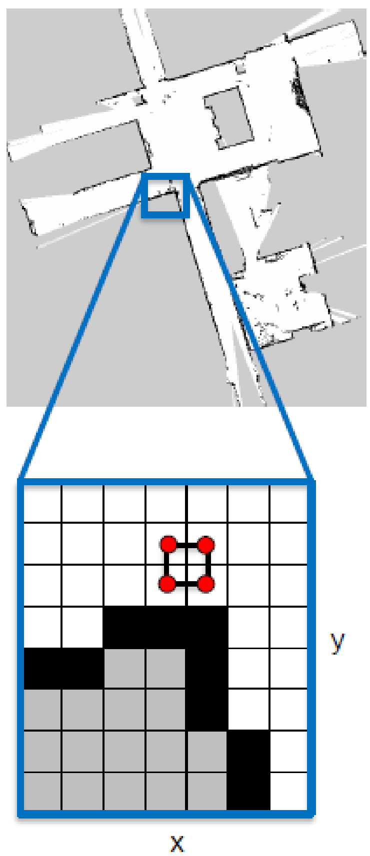

Figure 5.7: Bilinear Filtering Part 1

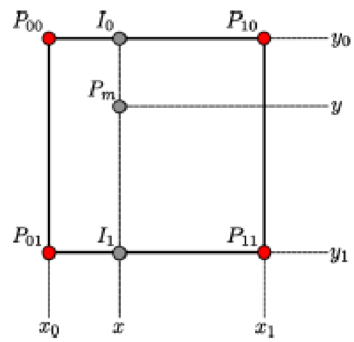

Figure 5.8: Bilinear Filtering of Occupancy Grid Map

To draw a brief conclusion, in hector SLAM, 2D map is represented by a 2D grid holding probability P_{xy} of cell occupancy. It should be noticed that this probability is represented by log odds. Obviously, This method does have pros and cons : the advantage is that this method is relatively fast, meanwhile the cons is the result is only approximate which can not be really accurate.

5.4.5 Hector Mapping Node Implementation

- *ls /dev ttyUSB0*

  To specify ttyUSB0 (UART connecting to vx 11 LIDAR) in Linux

- *roscore*

  start pre-requisites of a ROS-based system

- *sudo chmod 666 /dev/ttyUSB0*

  enable USB0 (LIDAR data reading)

- *cd catkin_ws*

  path to catkin workspace

- *source devel/setup.bash*

  source the setup file

- *rosrun xv_11_laser_driver neato_laser _publisher _port:*

  =devttyUSB0 _firmware_version:=2

  run LIDAR scan data and the publisher generated

- *rostopic echo /scan*

  visualization of LIDAR raw data

Initialize the final ROS launch file

- *cd* $catkin_w s/$

  path to catkin workspace

- *cd xv11-hector-slam-roslaunch-master/*

  cd to ROS launch file

- *roslaunch wireless mapping.launch*

  run ROS hector mapping launch file

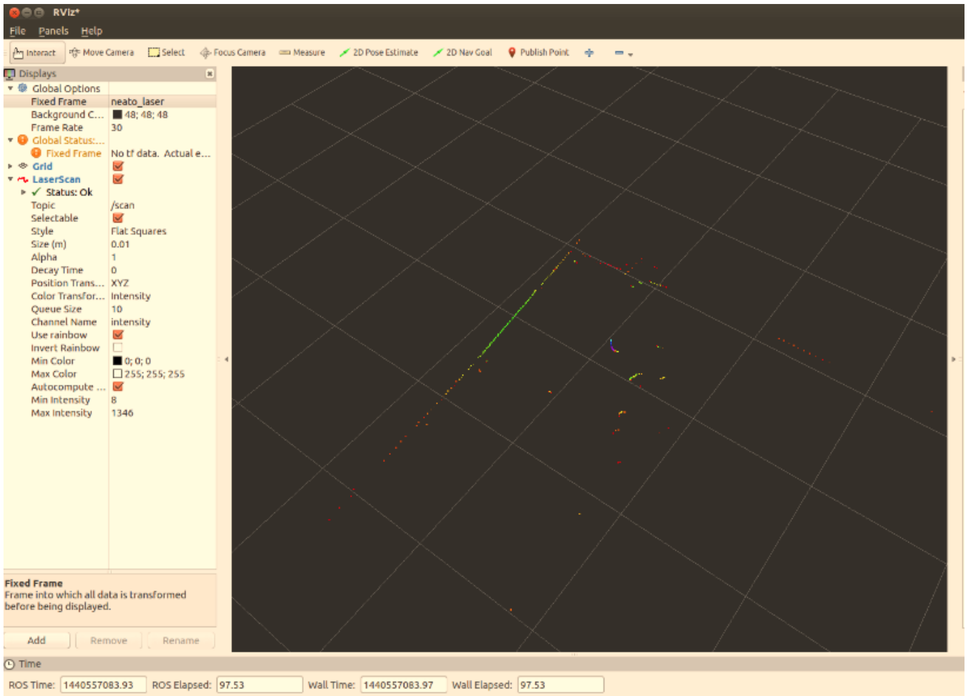

Figure 5.9: LIDAR Point Cloud Feature Detect

This experiment was held in Center Point Computer Science building. Obviously in the plot, there are features(walls) and some discrete features(like my legs and some obstacles on the ground).

We can observe different colors of those features, those colors stand for the laser intensities. Intensity only affect the color of the point, and the intensity channel uses 4 values to compute the final color of the point: (1) Min Intensity mini, (2)Max Intensity max, (3)Min Color minc, (4) Max Color maxc.

For each point, to compute the color value, we first compute a normalized intensity value based on min_i and max_i:

$$norm\_i = (i - min\_i)/(max\_i - min\_i) \tag{5.31}$$

Then to compute the color from that normalized intensity:

$$final\_c = (norm\_imax\_c) + ((1 - norm\_i)min\_c) \tag{5.32}$$

5.5 EKF SLAM Implementation Results and Analysis

Manually Remote Controlled Robot to Perform Indoor SLAM

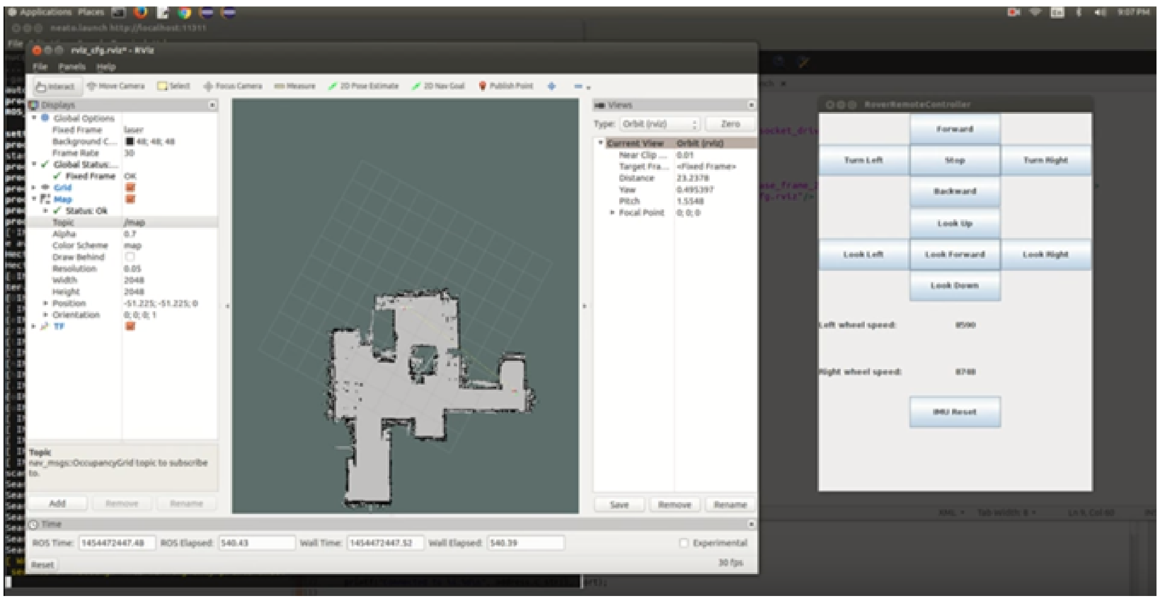

Figure 5.10: Unknown Environment 2D Map Representation

Description Robot map the room (10 meters length and 9.2 meters width) in 2 minutes. SLAM has been done wireless and a GUI was implemented. In this GUI, you can control the robot manually like going straight (just click the forward bottom) and turns (by clicking left/right turn bottoms). Besides, as we can see, the wheel encoder readings have been presented on the bottom of the GUI.

For full access to the indoor SLAM demo video, it has been uploaded to youtube:

https://www.youtube.com/watch?v=750z3U4tSAA

Autonomously line guided robot to perform indoor SLAM Besides robot can be controlled manually to perform SLAM, autonomously line guided robot performing indoor SLAM is of great importance.

First, we build a self-designed indoor area to perform SLAM, here is what this area looks like:

Figure 5.11: Self Designed Area for Mapping

Comparison between generated 2D grid map and real floor plan of that area.

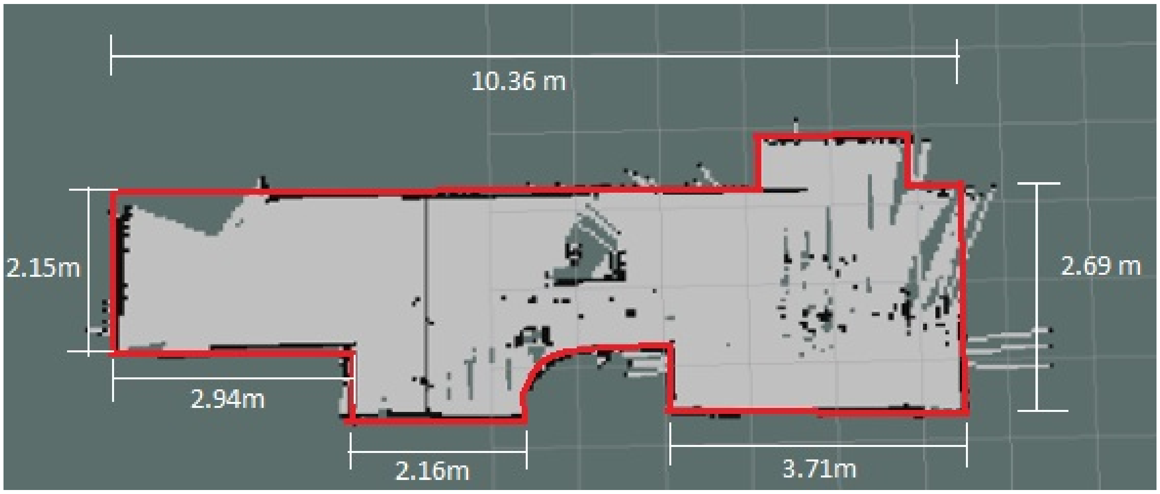

Figure 5.12: Comparison Between Generated Map and Real Floor Plan

Robot finished mapping the area in 38 seconds.

Map Accuracy

- Horizontal Accuracy

  Length of the real mapped area is $30.48cm \times 34 = 1036.32cm = 10.36m$. In generated 2D grid map, the length of the map is $9.80m$. In this case, the horizontal accuracy is:

$$\text{map horizontal accuracy} = \frac{10.36m - 9.80m}{10.36m} \times 100\% = 5.40\% \tag{5.33}$$

- Vertical Accuracy

  Width pf the real mapped area is $30.48cm \times 8{+}25cm = 268.8cm \approx 2.69m$. In generated 2D grid map, the length of the map is $2.77m$. In this case, the vertical accuracy is:

$$\text{map vertical accuracy} = \frac{2.77m - 2.69m}{2.69m} \times 100\% = 2.97\% \tag{5.34}$$

Relationship Between Nodes

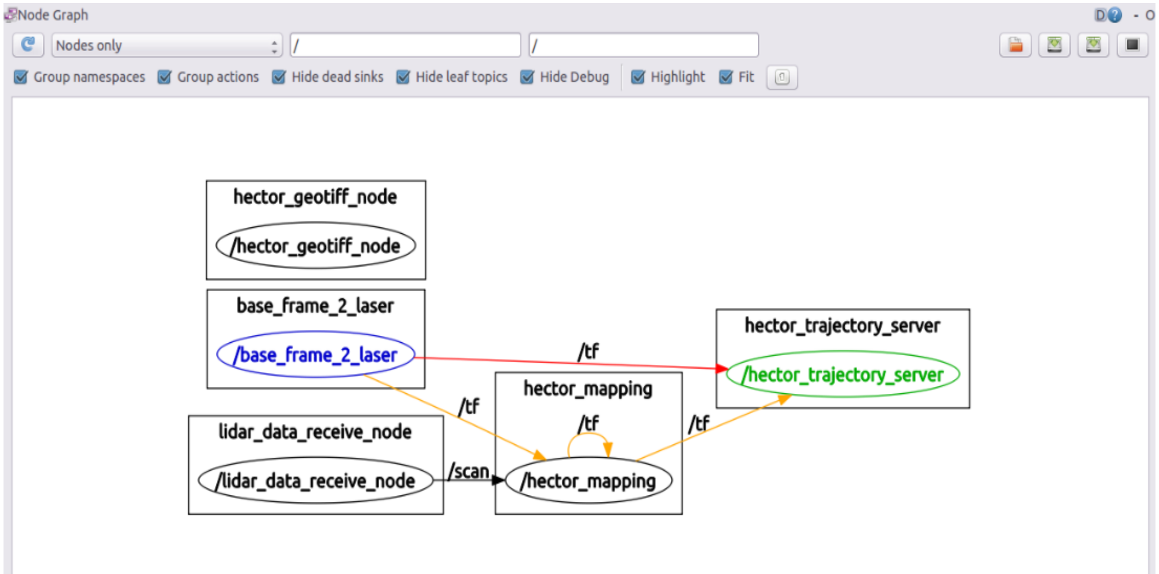

Figure 5.13: Node rqt Graph

Description Figure 5.13 shows the relationship between different running nodes. This rqt graph was generated when I was using robot to wirelessly map the room GWC 379C.

The basic data flow is :

1. Wirelessly received date stored in ROS TOPIC $/scan$
2. Change ROS TOPIC $/scan$ to a ROS PUBLISHER, data stored in this topic was broadcasting to all the running nodes
3. After receiving the published LIDAR data packages, node $hector_mapping$ is responsible for 2D map representation and node $hector_trajectory$ is calculation the estimated real time pose of robot using EKF
4. All these nodes are connected by node tf

Connections between ROS frames

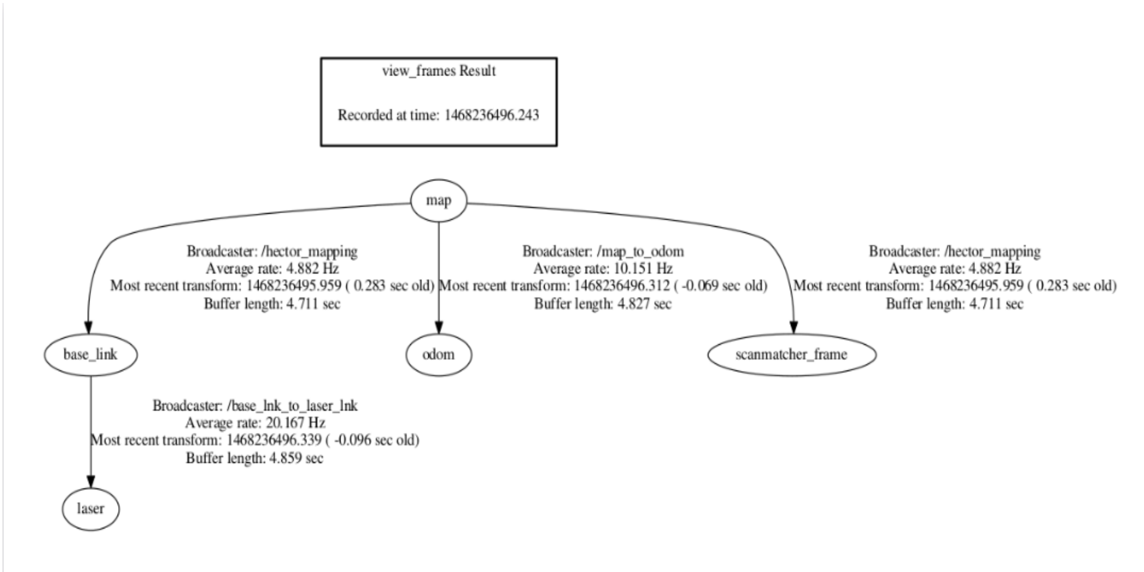

Figure 5.14: ROS tf Frames

tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

In this case, the $tfframe$ graph shows the connections between different frames ROS was using while the robot was performing indoor SLAM.

Wireless SLAM in Room GWC 379C (5x3meters Room)

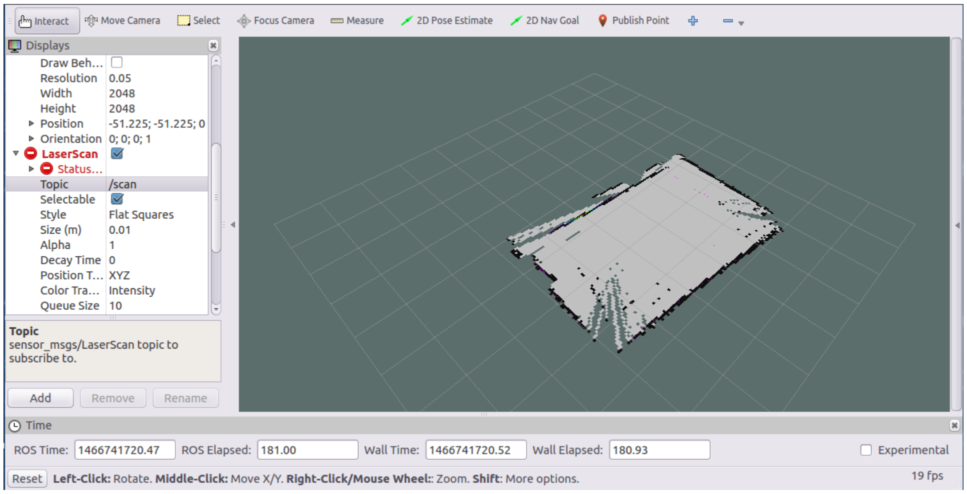

Figure 5.15: Wireless SLAM in Room GWC 379C (5x3meters Room)

Description. 2D grid map for room GWC 379C (5x3 meters) was generated in 18 seconds. Most errors and mismatches came from wireless LIDAR data transmission package loss. Using better router and WiFi adapter or replace TCP/IP protocol may solve the problem.

When LIDAR scan frequency is too low

Figure 5.16 shows the result when robot perform SLAM on 4th floor of Center Point Computer Science Engineering building. After mapped the room (on the right), robot went through the door and then went alone a hall way. Finally, it made a left turn and mapped the rest small room (which is on the left).

The second step is that robot made a sharp U-turn and started re-mapping the area. To be more precise, robot turned 180° in 2 seconds (a sharp U-turn). The generated maps (before and after sharp U-turn) are not matched.

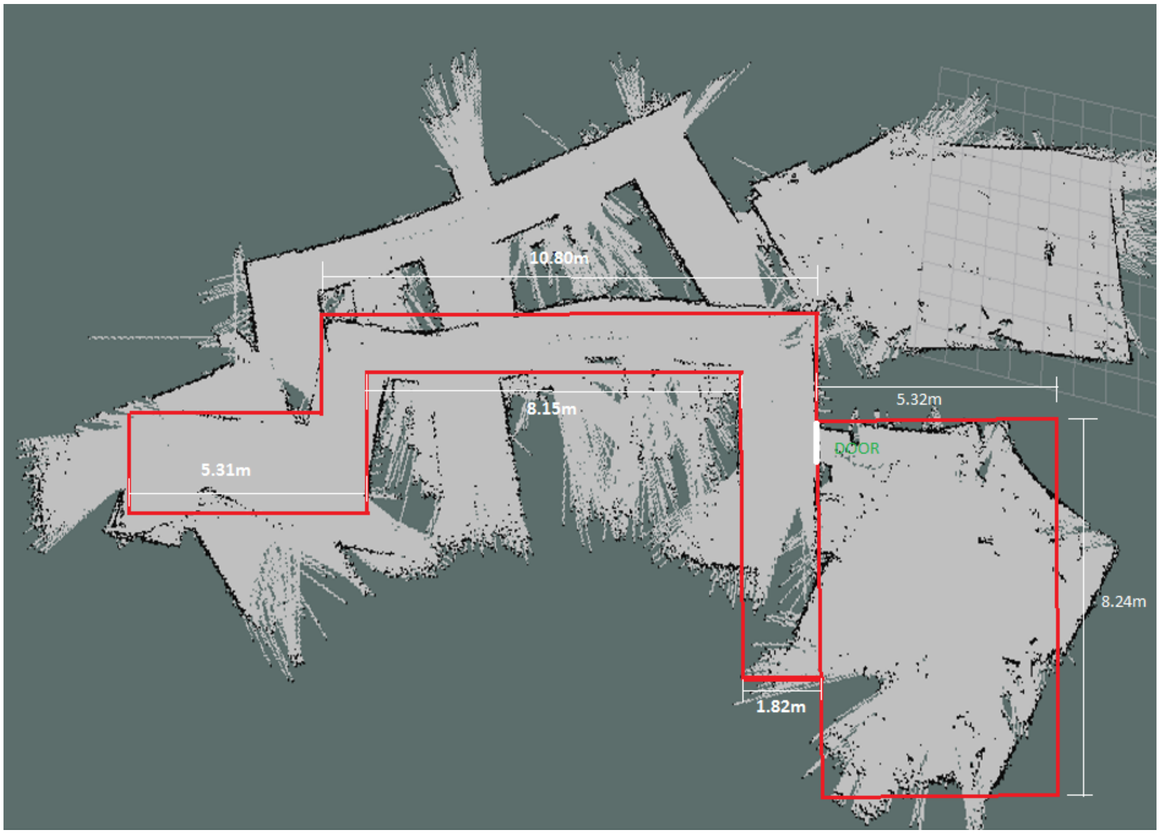

Figure 5.16: LIDAR Scan Frequency is Too Low

5.6 Summary and Conclusion

In this chapter, we well discussed how to implement Hector Mapping algorithm using ROS to perform indoor unknown environment mapping. Detailed setup instructions were provided. Both manually remote controlled robot to perform indoor SLAM and autonomously line guided robot to perform indoor SLAM have been well explained.

Real floor plan has been compared to the 2D grip map we have generated. To draw a brief conclusion, our FreeSLAM robot has the ability to perform SLAM in indoor unknown environment.

Chapter 6

SLAM WITH SENSOR FUSION OF ODOMETRY AND LIDAR SCAN DATA - GMAPPING

6.1 Introduction and Overview

When Input and Observation Noises are Non-Gaussian

Recently Rao-Blackwellized particle filters have been introduced as effective means to solve the simultaneous localization and mapping (SLAM) problem. This approach uses a particle filter in which each particle carries an individual map of the environment. Accordingly, a key question is how to reduce the number of particles. We present adaptive techniques to reduce the number of particles in a Rao- Blackwellized particle filter for learning grid maps. We propose an approach to compute an accurate proposal distribution taking into account not only the movement of the robot but also the most recent observation. This drastically decrease the uncertainty about the robot's pose in the prediction step of the filter. Furthermore, we apply an approach to selectively carry out re-sampling operations which seriously reduces the problem of particle depletion.

To draw a brief conclusion of particle filter:

what is a particle filter Briefly, particle filter is a Bayes Filter. Besides, it's a way to efficiently represent non-Gaussian distribution. As mentioned in Chapter 4, Kalman Filter is the best low pass filter when input noises are Gaussian. So in the case that those input noises are non-Gaussian, particle filter can be a way to choose from those low-pass filters.

6.2 Detailed Modeling for Gmapping SLAM Approach

Definitions

- f - motion equation
- u - control inputs
- w - input noise
- g - observation equation
- y - observation data
- n - observation noise

Motion Model

The motion model describes the relative motion of the robot:

$$p(x_t \mid x_{t-1}, u_t) \tag{6.1}$$

Estimated Robot Pose includes X and Y position information and robot's orientation ψ

$$Pose : x_t = [x, y, \psi]_k \tag{6.2}$$

Motion Equation f:

$$x_{k+1} = x_k + \Delta x_k + w_k \tag{6.3}$$

In this chapter, we assume that both input noise (w_k) and observation noise (n_k) are Non-Gaussian noise. As a result, Extended Kalman Filter will not be a good choice. The most common Non-Gaussian noise is salt and pepper noise.

Observation Model

The observation or sensor model relates measurements with the robot's estimated pose:

$$p(z_t \mid x_t) \tag{6.4}$$

$L_k = [L_{k,x}, L_{k,y}]$ is a 2D landmark. Landmark can be selected automatically by computer, usually it is supposed to be a corner or a object observed in the mapping area.

$$L_k = [L_{k,x}, L_{k,y}]_k \tag{6.5}$$

Observation Equation g:

$$\begin{bmatrix} r \\ \theta \end{bmatrix}_k = \begin{bmatrix} \sqrt{\|x_k - L_k\|^2} \\ tan^{-1} \frac{L_{k,y} - x_{k,y}}{L_{k,x} - x_{k,x}} \end{bmatrix} + n_k \tag{6.6}$$

Obviously the observation equation above is non-linear.

6.3 Probabilistic Laws

Mathematical Expression of SLAM

To recall what has been mentioned in Chapter 4, the representation of SLAM is as follows:

$$p(x, m|z, u) \tag{6.7}$$

where x denotes the pose of the robot, m is the grip map and x represents the observations and movements.

Environment Measurement Data

$$z_{t_1:t_2} = z_{t_1}, z_{t_1+1}, z_{t_1+2}, \cdots, z_{t_2} \tag{6.8}$$

denotes the set of all measurements acquired from time t_1 to time t_2, for $t_1 \leq t_2$

Control Data

An alternative source of control data are *odometers*.

We will denote sequences of control data by

$$u_{t1:u_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, \cdots, u_{t_2} \tag{6.9}$$

Probabilistic Generative Laws

The evolution of state and measurements is governed by probabilistic laws. In general, the state x_t is generated from the state x_{t-1}. Hence, the probabilistic law characterizing the evolution of state by a probabilistic distribution of the following form:

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) \tag{6.10}$$

We assume that the robot executes a control action u_1 first, and then takes a measurement z_1.

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t) \tag{6.11}$$

This property is called *conditional independence*. It states that certain variables are independent of others if one knows that values of a third group of variables, the conditioning variables.

$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t) \tag{6.12}$$

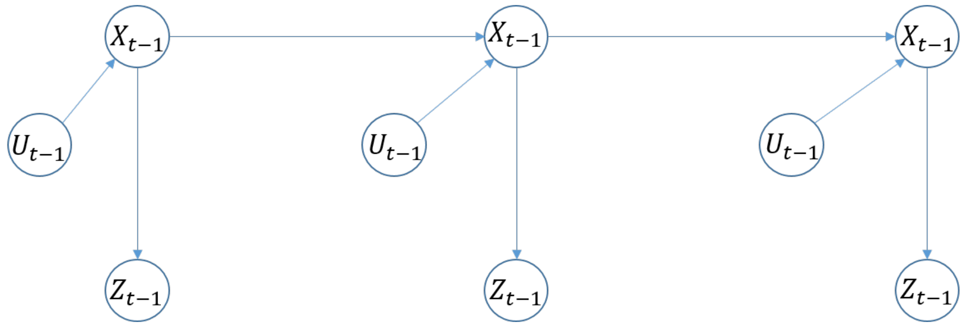

Figure 6.1: The Dynamic Bayes Network that Characterized the Evolution of Controls, States, and Measurements

This property shows that the state x_t is sufficient to predict the measurement z_1. Any other variables, such as past measurements, controls and states, is irrelevant if x_t is complete.

6.4 Sample Base Localization

The basic principle of implementing particle filter in Gmapping node: First, to set the state hypotheses (which are the "particles"). Of course we can set the number of the particles. the more particles we set, the more accuracy of state estimation we'll get. but meanwhile it will require more computational complicity and time. Second step, is that we use the combination of LIDAR processed data and odometry data to find those Survival-of-the-fittest particles, which are the accurate estimation of real time robot's pose.

Set of weighted samples

$$S = \left\{ < s^{(i)}, w^{(i)} > | i = 1, 2, ...N \right\} \tag{6.13}$$

In the equation above, $s^{(i)}$ denotes the state hypothesis and $w^{(i)}$ means the Importance weight of each particle.

The samples represent the posterior

$$P(x) = \sum_{i=1}^{N} w_i \cdot \delta_{s(i)}(x) \tag{6.14}$$

From Sampling to a Particle Filter

- Set of samples describes the posterior
- Updates are based on actions (control of DC motors and steering servo) and observations (LIDAR observations and encoder, IMU readings)

Three sequential steps:

1. Sampling from the proposal distribution (Bayes filter: prediction step)
2. Compute the particle weight (Bayes filter: correction step)
3. Resampling

Monte-Carlo Localization

- For each motion δ (each movement of robot) do sampling: Generate from each sample in a new sample according to the motion model.

$$x^{(i)} \leftarrow x^{(i)} + \Delta' \tag{6.15}$$

- For each observation(LIDAR data and odometry readings) do:Weight the samples with the observation likelihood

$$w^{(i)} \leftarrow p(z|m, x^{(i)}) \tag{6.16}$$

- Re-sampling

As a result, using all the information we've used above, we can basically conclude the particle filter solution to the SLAM problem:

1. Use a particle filter to represent potential trajectories of the robot

2. Each particle carries its own map

3. Each particle survives with a probability proportional to the likelihood of the observations relative to its own map

4. We have a joint posterior about the poses of the robot and the map

6.5 Summary and Conclusion

As we have discussed in Chapter 5, Extended Kalman Filter is one of the best filter under Gaussian noise. In this chapter, we introduced another filter: Particle Filter (PF) which may have better performance under LIDAR measurement non-Gaussian noise. More detailed algorithms and implementations will be addressed in future works and researches.

Chapter 7

SUMMARY AND FUTURE DIRECTIONS

7.1 Summary of Work

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

1. **Self-Designed Rear Wheel Drive FAME Mobile Robot Platform.** In Lin's thesis, 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 research. In this thesis, more expensive and selected components were used. While our enhanced FreeSLAM robot (with 360 RP LIDAR, wheel encoders, a 9 dof IMU, Arduino Uno, Raspberry PI III, camera, video WiFi link, pan-tilt servo) cost less than 610, it offer the capabilities of a self-driving robot costing more than 3000. Instructions for enhancement/building were included (see Appendix).

2. **FAME Architecture.** A general $FAME$ architecture has been described one that can accommodate a large fleet of vehicles and those can finish tasks cooperatively. For example, platooning of a fleet of robots and a group of robots build a map of a room together.

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

4. **Modeling** Kinematic and dynamic models for rear wheel drive robot were presented and analyzed to understand the full utility of each model. A nonlinear dynamical model (with motor dynamics) for the rear-wheel drive was used to conduct linear trade studies whose are useful for the development of cruise controllers.

5. **Control** Both inner-loop and outer-loop control designs were discussed in the context of of an overall hierarchical control inner-outer loop framework. This framework lends itself to accommodate multiple modes of operations; e.g. cruise control along a line/curve, position control along a line/curve, planar xy-Cartesian stabilization, etc.

   A great deal of effort was spent on discussion fundamental performance limitations. Attention was spent on static (steady state, accuracy related) limitations as well as dynamics (bandwidth) limitations. Encoder, IMU, camera (wireless vedio streaming and), and A-to-D (zero order hold half sample) limitations were particularly emphasized. This shall be very useful to researchers pursing future $FAME$ developments.

7.2 Directions for Future Research

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

- **On-board Sensing.** Addition of multiple on board sensors; e.g. additional ultrasonics, depth sensors(Kinect), 3D LIDAR, GPS & cameras.

- **Advanced Image Processing.** Use of advanced image processing and optimization algorithms; e.g. Implementations of OpenCV and OpenGL and vision based mapping and localization.

- **Multi-Vehicle Cooperation.** Cooperation between ground, air, and sea vehicles - including quadrotors, micro-air vehicles; e.g. nano-air vehicles landing on large ground robot, platooning of a fleet of ground robots and multi-robots solving indoor and outdoor SLAM problems.

- **Parallel On-board Computing.** Use of multiple processors on a robot for computationally intense work; e.g. multi-robots solving indoor unknown environment mapping, they have the ability to communicate to each other and divide the grid map of the room into some categories, which can significantly save time.

- **3D Unknown Environment Reconstruction.** In this thesis, the 2D indoor unknown environment mapping was welly discussed. In the future, we can achieve 3D indoor and outdoor unknown environment reconstruction using 3D LIDAR, depth sensors and cameras.

- **Modelling and Control.** More accurate dynamic models and controls 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 the design of a vehicle robot.

REFERENCES

[1] Stefan Kohlbrecher, Johannes Meyer, et al, “A Flexible and Scalable SLAM System with Full 3D Motion Estimation,” *Proceedings of the 2011 IEEE International Symposium on Safety,Security and Rescue Robotics* ,769-6, 2011.

[2] A.HEMAMI, “Steering control problem formulation of low-speed tricycle-model vehicles,” *International Journal of Control*, 61:4, 783-790.

[3] Jana Kosecka, “Vision-Based Lateral Control of Vehicles: Look-ahead and Delay Issues”, 1996.

[4] Safdar Zaman, Wolfgang Slany, Gerald Steinbauer, ”ROS-based Mapping, Localization and Autonomous Navigation using a Pioneer 3-DX Robot and their Relevant Issues” *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International*, 2011.

[5] Quigley, Morgan, et al. “ROS: an open-source Robot Operating System.” ,” *ICRA workshop on open source software*, Vol. 3. No. 3.2. 2009.

[6] Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard. “Improved techniques for grid mapping with rao-blackwellized particle filters.” *Robotics, IEEE Transactions,* on 23.1 (2007): 34-46.

[7] Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard. “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling,” *Robotics and Automation, 2005. ICRA 2005.* , Proceedings of the 2005 IEEE International Conference on. IEEE, 2005.

[8] Fierro, Rafael, and Frank L. Lewis. “Control of a nonholonomic mobile robot: backstepping kinematics into dynamics,” *Decision and Control, 1995.,* Proceedings of the 34th IEEE Conference on. Vol. 4. IEEE, 1995.

[9] Kmmerle, Rainer, et al. “On measuring the accuracy of SLAM algorithms.” Autonomous Robots 27.4 (2009): 387-407.

[10] Ganeshmurthy, M. S., and G. R. Suresh. “Path planning algorithm for autonomous mobile robot in dynamic environment,” *Signal Processing, Communication and Networking (ICSCN),*, 2015 3rd International Conference on. IEEE, 2015.

[11] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. “Probabilistic robotics,” *MIT press,*, 2005.

[12] Martinez, Aaron, and Enrique Fernndez, “ Learning ROS for robotics programming,” *Packt Publishing Ltd*, 2013.

[13] Riisgaard, Sren, and Morten Rufus Blas. “SLAM for Dummies,” *A Tutorial Approach to Simultaneous Localization and Mapping* ., 22.1-127 (2003): 126. APA

[14] Gary Bradski, Adrian Kaehler. "Learning OpenCV Computer Vision with the OpenCV Library." , O'Reilly Media

[15] Rodriguez, A.A., *Analysis and Design of Multivariable Feedback Control Systems*, Control3D, L.L.C., Tempe, AZ, 2002.

[16] Rodriguez, A.A., *Linear Systems: Analysis and Design*, Control3D,L.L.C., Tempe, AZ, 2002.

[17] Susnea, I., Filipescu, A., Vasiliu, G., et al., "Path following, real-time, embedded fuzzy control of a mobile platform wheeled mobile robot," *IEEE International Conference on Automation and Logistics (ICAL)*, pp. 268-272, 2008.

[18] K. J. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning*, Instrument Society of America, Research Triangle Park, North Carolina, 1995.

[19] Arduino Uno description, https://www.arduino.cc/en/Main/arduinoBoardUno.

[20] Introducing the Raspberry Pi 2 - Model B - Adafruit Learning. https://learn.adafruit.com/introducing-the-raspberry-pi-2-model-b/overview

[21] Rodriguez, A., EEE481: Computer Control Systems, course notes, 2014.

[22] Amir M Y, Abbass V., "Modeling of quadrotor helicopter dynamics," International Conference on Smart Manufacturing Application, ICSMA 2008, IEEE 2008: 100-105.

[23] Kaplan, Elliott, and Christopher Hegarty, eds., "Understanding GPS: principles and applications." Artech house, 2005.

[24] Raspberry Pi 5MP camera datasheet. https://www.raspberrypi.org/documentation/hardware/camera.md

[25] Research robots, http://www.mobilerobots.com/ResearchRobots.aspx

[26] Lidar, https://en.wikipedia.org/wiki/Lidar

[27] 9 dof IMU BNO055 library https://github.com/adafruit/AdafruitBNO055

[28] Rodriguez, A.A., *Analysis and Design of Multivariable Feedback Control Systems*, Control3D, L.L.C., Tempe, AZ, 2002.

[29] Rodriguez, A.A., *Linear Systems: Analysis and Design*, Control3D,L.L.C., Tempe, AZ, 2002.

APPENDIX A

MATLAB CODE

```
%************DC Motor Dynamics Simulation***********

%parameters
Ra = 2.523; %ohm resistant
La = 0;% armature inductance, which is neglected
Kt = 0.004; % torque constant
Kb = 0.004; %Back EMF constant
J = 2.96*10^-6; % Load moment of inertia
B = 4.3*10^-5; % Damping constant

% Here is the transfer function from Ea to angular velocity
s = tf('s');
Simulation = Kt/(La*J*s^2+(La*B+Ra*J)*s+Kt*Kb+Ra*B);
%step(Simulation,5,'r');

%Transfer function from Ea to Tau
H1 = (1/(La*s+Ra))*Kt;
H2 = (Kb*(1/(J*s+B)));
H3 = H1/(1+H1*H2);
zpk(H3)

%************END DC Motor Dynamics Simulation END***********
```

```
%*****State Space Representation for*****
%******Vision Based Complete Lateral Model******
close all
clear all
clc

% Parameters of Lateral Dynamics
SIM_cf=120000; %lb/rad  stiffness of front wheel
SIM_cr=100000; %lb/rad  stiffness of rear wheel
SIM_I_psi = 2753; %slug ft^2
SIM_m = 1573; %slugs  , 1 slug = 14.593903 kg
SIM_ca = 1.44; %aerodynamics drag coefficient
SIM_lr = 1.53; %distance from rear axle to cg
SIM_lf = 1.137; %distance from front axle to cg
SIM_l=SIM_lr+SIM_lf; %full length of the vehicle
SIM_L = 15;  % 15m look ahead distance by raspberry pi camera
SIM_vx = 20; % m/s cruise speed of the robot
SIM_Td = 0.15; %s vision subsystem delay

%*********State Space Representation

syms m I_psi lf lr l cf cr L vx s

% Matrix A
a11 = -(cf+cr)/(m*vx);
a12 = -vx + (cr*lr - cf*lf)/(m*vx);
a13 = 0;
a14 = 0;

a21 = (-lf*cf + lr*cr)/(I_psi*vx);
a22 = -((lf^2)*cf + (lr^2)*cr)/(I_psi*vx);
a23 = 0;
a24 = 0;
```

```
a31 = -1;
a32 = -L;
a33 = 0;
a34 = vx;

a41 = 0;
a42 = -1;
a43 = 0;
a44 = 0;

A=[ a11 a12 a13 a14;
    a21 a22 a23 a24;
    a31 a32 a33 a34;
    a41 a42 a43 a44];

%***********
%Matrix B

b11 = cf/m;
b21 = (lf*cf)/I_psi;
b31 = 0;
b41 = 0;

B = [ b11
      b21
      b31
      b41];

%*************
%Matrix C
c11 = -(cf+cr)/(m*vx);
c12 = (cr*lr - cf*lf)/(m*vx);
c13 = 0;
c14 = 0;

c21 = 0;
c22 = 1;
c23 = 0;
c24 = 0;

c31 = 0;
c32 = 0;
c33 = 1;
c34 = 0;

c41 = 0;
c42 = 0;
c43 = 0;
c44 = 1;

C=[ c11 c12 c13 c14;
    c21 c22 c23 c24;
    c31 c32 c33 c34;
    c41 c42 c43 c44];

%***************
```

```
%Matrix D
d11 = cf/m;
d21 = 0;
d31 = 0;
d41 = 0;

D = [d11
     d21
     d31
     d41];
 %%
 %Plant Simbolic
 X = (C/(s*eye(4)-A)*B+D);

 I4=eye(4);
 P=C/(s*I4-A)*B+D;
pretty(simplify(P))
 %Plug in numbers

 %*******Simulation model

 SIM_A = double(subs(A,{cf cr m vx lr lf I_psi L },...
 {SIM_cf SIM_cr SIM_m SIM_vx SIM_lr SIM_lf SIM_I_psi SIM_L}));

 SIM_B = double(subs(B,{cf m lf I_psi},...
 {SIM_cf SIM_m SIM_lf SIM_I_psi}));

 SIM_C = double(subs(C,{cf cr m vx lr lf},...
 {SIM_cf SIM_cr SIM_m SIM_vx SIM_lr SIM_lf}));

 SIM_D = double(subs(D,{cf m},{SIM_cf SIM_m}));

 SIM_SS=ss(SIM_A,SIM_B,SIM_C,SIM_D);
%%
%   SIM_X = (SIM_C/(s*eye(4)-SIM_A)*SIM_B+SIM_D);
%%
 SIM_SS(3,1)
 figure(1);
 bode(SIM_SS(3,1))
 grid on;
 hold on;
```

```
%******Longitudinal Inner Loop PI Controller Trade Study*****

clc
%PI Controller Parameters
z = 0.5;
g = 9;

%Varying g and z

%for g = 1:4:17
for z = 0.1:0.2:0.9

Ki = 4.5;
g = Kp;
```

```
z = Ki/Kp;

s = tf('s');
%winit = -1;
%wfin = 2;
%nwpts = 300;
%w = logspace(winit,wfin,nwpts);

K = ((g*(s+z))/s)*(100/(s+100));
%The PI controller with high-freq roll-off

W = z/(s+z); %The pre-filter

% Longitudinal Plant Representation

P = 0.146*(s+14.53)/((s+0.1116)*(s+16.67));

%Form Open Loop Singular Values
L = P * K;

%Open Loop Frequency Response
figure(1);
%bode(L)

%Form Closed Loop Transfer Functions
 figure(2)

 tr2y = W*L/(1+L);      % Try
 bodemag(tr2y)
 %step(tr2y,50)
 hold on
 grid

 figure(3)
   tru = K/(1+L);        % Tru without W
   tru_W = W*K/(1+L);    % Tru with W
   bodemag(tru_W)
   bodemag(tru_filter)
 step(tr2u);

 %bodemag(tr2u)
 hold on;
 grid on;
%
    S = 1/(1+L);         %Sensitivity
    bodemag(S)

%
    T = L/(1-L);            %Complementary Sensitivity
    bodemag(T)

   %Implementing the Filter:

figure(4)
step(tr2y)
hold on
grid
```

```matlab

figure(5)
step(tr2u)
hold on
grid

 tdiy = P/(1+L);      % Tdiy
 bodemag(tdiy)

 tdi2u = -PK/(1+PK); % Tdiu

 %Determine Closed Loop Poles
 clp_long = pole(1/(1+PK));

 %Stability Robustness
 allmargin(PK);

%Closed Loop Transfer Functions
zpk_tr2y = minreal(zpk(tr2y));
zpk_tr2u = minreal(zpk(tr2u));
zpk_tdi2y = minreal(zpk(tdi2y));
zpk_tdi2u = minreal(zpk(tdi2u));
%
%Plots Settings
    grid on;
    set( findobj(gca,'type','line'), 'LineWidth', 2);
    h = findobj(gcf, 'type', 'line');
    set(h, 'LineWidth', 3);
    a = findobj(gcf, 'type', 'axes');
    set(a, 'linewidth', 6);
    set(a, 'FontSize', 14);
    xlabel('', 'FontSize', 24);
    ylabel('', 'FontSize', 24);
    hold on;
end

%Trade Studies Titles and Legends

%Try changing g
title('Frequency Response T (With Pre-Filter & g = 1-17, z = 0.5)')
legend('g=1 z=0.5','g=5 z=0.5','g=9 z=0.5','g=13 z=0.5','g=17 z=0.5')

%Try changing z
title ('Bode Magnitudes for T (With Pre-Filter and g = 9, z = 0.1-0.9)')
legend ('g=9 z=0.1', 'g=9 z=0.3', 'g=9 z=0.5', 'g=9 z=0.7', 'g=9 z=0.9')

%L changing g
 title ('Bode Plot for L (g = 1-17, z = 0.5)')
 legend ('g=1 z=0.5', 'g=5 z=0.5', 'g=9 z=0.5', 'g=13 z=0.5', 'g=17 z=0.5')

%L changing z
   title ('Bode Plot for L (g = 9, z = 0.1-0.9)')
   legend ('g=9 z=0.1', 'g=9 z=0.3', 'g=9 z=0.5', 'g=9 z=0.7', 'g=9 z=0.9')

%S changing g
title ('Bode Magnitudes for Sensitivity, g = 1-17, z = 0.5')
legend ('g=1 z=0.5', 'g=5 z=0.5', 'g=9 z=0.5', 'g=13 z=0.5', 'g=17 z=0.5')
```

```
%S changing z
 title ('Sensitivity, g = 9, z = 0.1-0.9', 'FontSize', 24)
 legend ('g=9 z=0.1', 'g=9 z=0.3', 'g=9 z=0.5', 'g=9 z=0.7', 'g=9 z=0.9')

%T changing g
title ('Complementary Sensitivity T, g = 1-17, z = 0.5', 'FontSize', 24)
legend ('g=1 z=0.5', 'g=5 z=0.5', 'g=9 z=0.5', 'g=13 z=0.5', 'g=17 z=0.5')

%T changing z
title ('Complementary Sensitivity T, g = 9, z = 0.1-0.9', 'FontSize', 24)
legend ('g=9 z=0.1', 'g=9 z=0.3', 'g=9 z=0.5', 'g=9 z=0.7', 'g=9 z=0.9')

Tru without prefilter changing g
title ('Bode Magnitude Plot for Tru , g = 1-17, z = 0.5', 'FontSize', 24)
legend ('g=1 z=0.5', 'g=5 z=0.5', 'g=9 z=0.5', 'g=13 z=0.5', 'g=17 z=0.5')

Tru without prefilter changing z
 title ('Bode Magnitude Plot for Tru , g = 9, z = 0.1-0.9', 'FontSize', 24)
 legend ('g=9 z=0.1', 'g=9 z=0.3', 'g=9 z=0.5', 'g=9 z=0.7', 'g=9 z=0.9')

Tru_W with pre-filter changing g
title ('Bode Magnitude Plot for W*Tru , g = 1-17, z = 0.5', 'FontSize', 24)
legend ('g=1 z=0.5', 'g=5 z=0.5', 'g=9 z=0.5', 'g=13 z=0.5', 'g=17 z=0.5')

Tru_W with pre-filter changing z
title (' W*Tru , g = 9, z = 0.1-0.9', 'FontSize', 24)
legend ('g=9 z=0.1', 'g=9 z=0.3', 'g=9 z=0.5', 'g=9 z=0.7', 'g=9 z=0.9')

Tdiy changing g
title ('Bode Magnitude Plot for Tdiy , g = 1-17, z = 0.5', 'FontSize', 24)
legend ('g=1 z=0.5', 'g=5 z=0.5', 'g=9 z=0.5', 'g=13 z=0.5', 'g=17 z=0.5')

Tdiy changing z
title ('Bode Magnitude Plot for Tdiy , g = 9, z = 0.1-0.9', 'FontSize', 24)
legend ('g=9 z=0.1', 'g=9 z=0.3', 'g=9 z=0.5', 'g=9 z=0.7', 'g=9 z=0.9')
```

```
%***Onground Model Longitudinal PI Controller Implementation***

% PI control mode verify
clear all
% close all
clc
% Load Nominal Model
%  input voltage 2 angular speed
 Plant = tf([0.3274],[1 1.176]);
%% Global Variable List
% create_global_vars_list
Ts =0.1;
RUNTIME = 8.9;
BV = 8;

% Always check what inside

% PI control parameter loading
% Prepare PID table in advance
```

```
% put your (g,z) trade off study here

% in matrix each col corresponding to one z value
% in matrix each row corresponding to one g value
% plots in column-wise from reshaped(mat,#row*#col,1)

% ts = 1.5
% g_vec = [0.073]; z_vec = [0.194/0.073]; raw_file_name = 'out1_2.mat';
%   g_vec = [0.096]; z_vec = [0.519/0.096]; raw_file_name = 'out2_2.mat';
 g_vec = [11.68]; z_vec = [2.02];

g_len = length(g_vec);
z_len = length(z_vec);

S_PI= PID_Table_Generator_0702(g_vec,z_vec);
%% Prepare Data and Initializing For Later Mfiles
% Hardware data loading
% with prefilter
% Check and Edit This File Before you run following Code
% G_wF = DataImport_Log_PID_0707(raw_file_name)

Hw= [0  0
  0.00  15
  0.00  53
  0.00  91
  0.06  107
  0.13  112
  0.19  117
  0.25  113
  0.28  118
  0.38  101
  0.44  79
  0.47  72
  0.53  49
  0.57  31
  0.53  37
  0.53  38
  0.57  22
  0.53  26
  0.47  50
  0.44  67
  0.47  61
  0.50  49
  0.53  37
  0.53  31
  0.50  42
  0.50  43
  0.50  42
  0.53  31
  0.53  25
  0.50  36
  0.50  37
  0.47  47
  0.47  53
  0.53  31
  0.53  23
  0.53  24
```

```
    0.53  19
    0.47  40
    0.47  48
    0.50  35
    0.50  34
    0.50  35
    0.50  34
    0.47  45
    0.44  61
    0.50  45
    0.53  27
    0.50  37
    0.53  28
    0.50  31
    0.47  49
    0.50  39
    0.50  36
    0.50  38
    0.50  36
    0.50  37
    0.50  36
    0.50  36
    0.50  36
    0.50  36
    0.50  35
    0.50  35
    0.47  46
    0.47  52
    0.50  41
    0.50  39
    0.50  40
    0.50  39
    0.50  39
    0.50  38
    0.50  38
    0.47  49
    0.47  55
    0.50  43
    0.50  41
    0.53  31
    0.53  25
    0.50  36
    0.50  37
    0.50  36
    0.50  36
    0.50  36
    0.50  36
    0.47  46
    0.47  52
    0.50  41
    0.50  39
    0.50  40
    0.50  39
    0.47  50];

hw_time = 0:0.1:8.9;
G_wF.time        = hw_time;
```

```
G_wF.PWMR        = Hw(:,2);
G_wF.LinearV     = Hw(:,1);

disp('Hardware data loading finished ')

%% Generate Simulation Transfer Functions
%Try(wR_ref,wL_ref)->(wR,wL)
%Tru(wR_ref,wL_ref)->(ea_R,ea_L)
% reference cmd; v_ref = 0.5;
reference_factor = 0.5;
pwm_voltage_factor = 255/BV;
set_size = length(S_PI.PI_cell);

% Generate Transfer Functions for different g z
% Store them in Transfer Function Cell
% Creat Cells
Try_cell_wF = cell(set_size,1);
Tru_cell_wF = cell(set_size,1);

for ii =1
    g = S_PI.gz_cell{ii}(1)
    z = S_PI.gz_cell{ii}(2)
    P = Plant
    K = tf([g g*z],[1 0]);
    rf = tf([100],[1 100]);
%      rf = tf(1);
    K = series(K,rf)
    W = tf([z],[1 z]); % pre-filter
    H = tf([20],[1 20]);
    S = siso_tf_generator_0702(W,P,K,H);

%      S = siso_tf_generator_0702(W,P,K,H);
    disp('          ')
    dispstr = sprintf('********** Start with g %.3f and z %.3f', g,z);
    disp(dispstr);
    K = zpk(K)
    zpk(S.L)
    S.Try;
    zpk(minreal(S.Tru))

%      num_wF = Try_wF.num{1};
%      den_wF = Try_wF.den{1};
%      disp('damping coefficient calculation ')
%      xi = den(2)/(2*sqrt(den(3))) % damping coefficient calculation
%      overshoot = exp(-xi*pi/sqrt(1-xi^2))*100

    Try_cell_wF{ii} = S.Try;
    Tru_cell_wF{ii} = S.Tru;
    S_wF = stepinfo(S.Try);    % Get Steady State Info

    g_list(ii)    = g;
    z_list(ii)    = z;

    dispstr = sprintf('########## End with g %.3f and z %.3f',g,z);
%      disp(dispstr);
    dispstr = sprintf('%s Peak Value %.3f',dispstr,S_wF.Peak);
    disp(dispstr);
```

```
    disp(' ');
    disp(' ')
end
disp('Simulation Transfer Functions Ready ')

%% Simulation and Hardware Comparison Try(v)

for ii =1
    fig = figure(ii+20);

    [Y_Sim_wF,T_Sim_wF] = step(Try_cell_wF{ii},RUNTIME);
    Y_Sim_wF = Y_Sim_wF* reference_factor; % output v simulation

    % hardware
    time_wF          = G_wF.time;
    LinearV          = G_wF.LinearV;
    zero_output = zeros(length(time_wF),1);

    plot (T_Sim_wF,Y_Sim_wF,time_wF,LinearV);

    h_line = findobj(gcf, 'type', 'line');
    set(h_line, 'LineWidth', 3);
    h_axes = findobj(gcf, 'type', 'axes');
    set(h_axes, 'linewidth', 2);
    set(h_axes, 'FontSize', 15);

    hold on;grid on;
    title ('Output response v_{ref} to v','FontSize',24);

    legend('Simulation','Hardware','Location','NorthEast');
    xlabel('Time(seconds)');
    ylabel('Translation Speed of Vehicle (m/sec)');
    axis([0 max(time_wF) 0 0.8]);
end

%% Simulation and Hardware Comparison Tru
    PWM2Voltage_Gain = BV./255;
   % Tru(wR,wL)
    % input: linear velocity referece
    % output voltage

for ii =1
    fig = figure(ii+30);

    [Y_Sim_wF,T_Sim_wF] = step(Tru_cell_wF{ii},RUNTIME);
    Y_Sim_wF = Y_Sim_wF* reference_factor;

    % hardware
    time_wF     = G_wF.time;
    eaR          = G_wF.PWMR.*PWM2Voltage_Gain;
    zero_output = zeros(length(time_wF),1);

    plot (T_Sim_wF,Y_Sim_wF,time_wF,eaR);

    h_line = findobj(gcf, 'type', 'line');
    set(h_line, 'LineWidth', 3);
```

```
    h_axes = findobj(gcf, 'type', 'axes');
    set(h_axes, 'linewidth', 2);
    set(h_axes, 'FontSize', 15);

    hold on;grid on;
    title ('Control output response v_{ref} to e_a','FontSize',24);

    legend('Simulation','Hardware','Location','NorthEast');
    xlabel('Time(seconds)');
    ylabel('Voltage(V)');
    axis([0 max(time_wF) 0 BV]);

end
```

```
%*****Lateral Model Inner Loop PI Controller Trade Study*****
%%
clear all
%%
Kp = 18;
Ki = 21.6;
s = tf('s');
%K = Kp + Ki/s; %The PI controller
%controller parameters
g = 18;
z = 1.2;
%for
K = ((g*(s+z))/s)*(100/(s+100)); %The PI controller with
                                 %high-freq roll-off
W = z/(s+z); %The pre-filter

% lateral inner loop Plant representation

P = (0.368*(s+0.484))/((s+1.077)*(s+0.457));

%Form Open Loop Transfer Function
L = P * K;

%Open Loop Frequency Response
 figure(1)
 %bode(L);
%
% hold on
% grid
% %xlabel('Frequency (rad/sec)')
% %ylabel('Magnitude (dB)')
% title('lateral Plant Open Loop Magnitude and Phase Response')
%
% %Form Closed Loop Transfer Functions
% figure(2)

  T_ry = L/(1+L);        %without pre-filter
  zpk(minreal(T_ry))
  T_ry_W = W*L/(1+L);    % Try
  step(T_ry,7)
  bodemag(T_ry_W)
  step(T_ry_W,10)
```

```

  bode(tr2y)
  hold on;
  grid on;

 tr2u = K/(1+PK);       % Tru
 figure(3)
 bode(tr2u)
 hold on
 grid

 tdi2y = P/(1+PK);      % Tdiy
 tdi2u = -PK/(1+PK);    % Tdiu

grid on;
    set( findobj(gca,'type','line'), 'LineWidth', 2);
    h = findobj(gcf, 'type', 'line');
    set(h, 'LineWidth', 3);
    a = findobj(gcf, 'type', 'axes');
    set(a, 'linewidth', 6);
    set(a, 'FontSize', 14);
    xlabel('', 'FontSize', 24);
    ylabel('', 'FontSize', 24);
    hold on;

%end
% open loop L bode
 title ('Bode Plot for Open Loop L_{lateral}', 'FontSize', 24)
 legend ('g=18, z=1.2')

%close loop Try without prefilter
title (' T_{ry} without prefilter W', 'FontSize', 24)
legend ('g=18, z=1.2')

 %Try without prefilter
title ('Step Response for T_{ry} without prefilter W', 'FontSize', 24)
legend ('g=18, z=1.2')


%Try bode with a pre-filter
title ('Bode Magnitude Plot for T_{ry} with prefilter W', 'FontSize', 24)
legend ('g=18, z=1.2')

%Try step response with a pre-filter W
title ('Step Response for T_{ry} with prefilter W', 'FontSize', 24)
legend ('g=18, z=1.2')
```

```
%********On Ground Vehicle Lateral Model*******
%%
load('lateral_model.mat')
%%

umax    = 8.2;
% PWM     = 40;

w_vec   = yaw_diff;
```

```
T_hw     = time;
u_vec    = 20/180*pi.*ones(length(time), 1);% delta_f

w_vec = [0; w_vec];
T_hw  = [0; T_hw];
u_vec = [0; u_vec];

plant_lat =  tf([2.892],[1 2.659]);
%
radius = 0.024;

[Y_sim,T_sim] = step(plant_lat, max(T_hw));
Y_sim = Y_sim .* max(u_vec);

fig1 = figure(1);
plot(T_sim,Y_sim,T_hw,w_vec)
legend('Simulation','Hardware')
title(' \delta_f to Angular Velocity Step Response', 'FontSize', 24)
xlabel('Time(seconds)','FontSize', 24)
ylabel('Angular Velocity (rad/sec)', 'FontSize', 24);
hold on;grid on;
%
h_line = findobj(gcf, 'type', 'line');
set(h_line, 'LineWidth', 3);
h_axes = findobj(gcf, 'type', 'axes');
set(h_axes, 'linewidth', 2);
set(h_axes, 'FontSize', 15);
```

```
%**************Onground Lateral Model PI Controller Implementation*******
%%
clear all
clc
close all
%%

%% longitudinal
plant_long = tf(0.3274,[1 1.176]);
% PI inner loop v
ts = 2;
zeta = 0.9; % almost no overshoot

% inner loop longitudinal
[S1 S2 S3] = Innerloop_design_standard2nd_System(plant_long,ts,zeta);

 wn = 2.7778;   %Lateral Inner Loop Bandwidth

 Mp = 0.0015;

 g = 11.6799;   % Choose Kp and Ki

 z = 2.0178;

% No outerloop for longitudinal
%% lateral

plant_lat = tf(2.892,[1 2.659]);
```

```
% function S = siso_tf_generator(W,P,K)

% inner loop P
S4 = siso_tf_generator(1,plant_lat,2);

% outer loop PD

Integrator = tf([1],[1 0]);

plant_lat_out = series(S4.Try,Integrator);

g_vec =[1 2 3];

z = 3;

N = 100;

[K_cell,pid_cell] = platoon_pd_controller_fixed_zero_0707 ...
(plant_lat_out,g_vec,z,N,1)

% Select Kp = 6; Kd = 2;

%%
K_lat_out = K_cell{2};

S5 = siso_tf_generator(1,plant_lat_out,K_lat_out);
%Plotting Try
S5.Try;
```

```
%***********Lateral Outer Loop PD Controller Trade Studies*********
%%
clear all;
s = tf('s');
K = Kp + Ki/s; %The PI controller
z = 6;
controller parameters
g = 18;
%%

for g = 0.001:0.001:0.005

%K is the Inverse of the Plant with Gain
K = g*(s+92.88)*(s+6.94)*(s+1.229)*(s+0.4857)/ ...
((s+1.2)*(s+0.484))*(100/(s+100))^3;


W = z/(s+z); %The pre-filter

%Lateral Inner Loop Plant Representation

P = (662.4*(s+1.2)*(s+0.484))/(s*(s+92.88)*(s+6.94)*(s+1.229)*(s+0.4857));

%Form Open Loop Transfer Function
L = P * K;

%Open Loop Frequency Response
```

```
 figure(1)
 %bode(L);

hold on
grid
xlabel('Frequency (rad/sec)')
ylabel('Magnitude (dB)')
title('lateral Plant Open Loop Magnitude and Phase Response')

%Form Closed Loop Transfer Functions
figure(2)

  T_ry = L/(1+L);        %without pre-filter
  S = 1/(1+L);           %Sensitivity
  bodemag(S)
  bodemag(T_ry)
  step(T_ry)

  zpk(minreal(T_ry))
  T_ry_W = W*L/(1+L);    % Try
  step(T_ry,7)
  bodemag(T_ry_W)
  step(T_ry_W,10)

  bode(tr2y)
  hold on
  grid

  T_ru = K/(1+L);     % Tru
  figure(3)
  bodemag(T_ru);
  step(T_ru);
  hold on;
  grid;

  tdi2y = P/(1+PK);    % Tdiy
  tdi2u = -PK/(1+PK); % Tdiu

grid on;
    set( findobj(gca,'type','line'), 'LineWidth', 2);
    h = findobj(gcf, 'type', 'line');
    set(h, 'LineWidth', 3);
    a = findobj(gcf, 'type', 'axes');
    set(a, 'linewidth', 6);
    set(a, 'FontSize', 14);
    xlabel('', 'FontSize', 24);
    ylabel('', 'FontSize', 24);
    hold on;

end
%Open Loop L
 title ('Bode Plot for Open Loop L ', 'FontSize', 24)
 legend ('g = 0.001','g = 0.002','g = 0.003','g = 0.004','g = 0.005')

%Try outerloop
title ('Bode Magnitude Plot for Outerloop T_{ry} ', 'FontSize', 24)
legend ('g = 0.001','g = 0.002','g = 0.003','g = 0.004','g = 0.005')
```

```
%step response Try
title ('Step Response for Outerloop T_{ry} ', 'FontSize', 24)
legend ('g = 0.001','g = 0.002','g = 0.003','g = 0.004','g = 0.005')

%Tru Bode
 title ('Bode Magnitude Plot for Outerloop T_{ru} ', 'FontSize', 24)
 legend ('g = 0.001','g = 0.002','g = 0.003','g = 0.004','g = 0.005')

%Sensitivity S
 title ('Bode Magnitude Plot for Sensitivity S ', 'FontSize', 24)
 legend ('g = 0.001','g = 0.002','g = 0.003','g = 0.004','g = 0.005')
```

```
%*************Go Along a Line Outer Loop (v, theta) Control*************
%*************Hardware Simulation Analysis**************

data_get = csvread ('v_yaw_servo.txt');
%%
V                      =   data_get(:,1)./2;
yaw                    =   data_get(:,2).*2.*pi./180;
Td  = 0.100;          %sampling time
X_p = 0;
Y_p = 0;
X = [];
Y = [];
V_x = V .* cos(yaw);
V_y = V .* sin(yaw);
%%
figure(1);
time = linspace(0,7,69);
%plot(time,speed);
plot(time,-yaw);
axis([0 7 -1 1]);
grid on;
hold on;
title('Robot Orientation - Go Straight','FontSize', 24);
legend('Robot Orientation');
xlabel('Time (seconds)','FontSize', 24)
ylabel('\psi_{error} (degrees)', 'FontSize', 24);
%%
figure(2);
for n = 1:1:69
    X(n) = X_p + V_x(n) .* Td;
    X_p = X(n);

    Y(n) = Y_p + V_y(n) .*Td;
    Y_p = Y(n);
end;
    plot(X,Y,'r*');
    hold on;
    x1 = linspace(0,2.9,10);
    y1 = linspace(0,0,10);
    plot(x1,y1);
    hold on;
    axis([0 2.9 -2 2]);
  %axis equal;
```

```
 %hold on;

%%
     h_line = findobj(gcf, 'type', 'line');
     set(h_line, 'LineWidth', 3);
     h_axes = findobj(gcf, 'type', 'axes');
     set(h_axes, 'linewidth', 2);
     set(h_axes, 'FontSize', 15);

%%
title('Robot Trajectory - Go Straight','FontSize', 24);
legend('Robot Trajectory','Simulation');
xlabel('X(meters)','FontSize', 24)
ylabel('Y(meters)', 'FontSize', 24);
```

```
% ********Planar XY Cartesian Stabilization for Real Wheel Drive********
% ********Simulation and Hardware Result Match*********
%%
 clear
 clc
%%
 X_Y_get = csvread ('xy_raw_data.txt');
 X     =    X_Y_get(:,1);
 Y     =    X_Y_get(:,2);

 plot(X,Y,'*r');
 hold on;

 %%
 xr=[];
 yr=[];

 %ks=1.5;
 ktheta=2;%controller for x,y,angle
 w=[];w(1)=0; %initial angular velocity rad/s
 v=[];v(1)=0; %initial linear velocity m/s
 wc=[]; %ellipse w
 vc=[]; %ellipse v
 theta(1)=0; %iniatial robot angle

 x(1)=0; % initial condition
 y(1)=0; % initial condition

 xreal(1)=0;
 yreal(1)=0;

 for ks=0.55;
 % when i increase by 1, meaning one loop time
 %for ktheta=5:5:15
 for i=1:1:70;

 %p(i)=rx*ry/sqrt(ry?*cos(thetar(i))?+ ...
 % rx?*sin(thetar(i))?);
 xr(i)=1.52;
 yr(i)=1.52;

```

```
x(i+1)=x(i)+v(i)*0.1*cos(theta(i));
y(i+1)=y(i)+v(i)*0.1*sin(theta(i));
theta(i+1)=theta(i)+w(i)*0.1;
thetaR(i)=atan2((yr(i)-y(i+1)),(xr(i)-x(i+1)));

e=[xr(i)-x(i+1),yr(i)-y(i+1),thetaR(i)-theta(i)];
es=sqrt(e(1)^2+e(2)^2)*cos(e(3));
w(i+1)=ktheta*e(3);
v(i+1)=ks*es;

xreal(i+1)=xreal(i)+v(i)*0.1*cos(theta(i));
yreal(i+1)=yreal(i)+v(i)*0.1*sin(theta(i));

end
%plot(xreal,yreal)
%plot(xreal)
%hold on
end
%end

plot(xreal,yreal)
axis([0,1.52,0,1.52])
%figure(2)
%plot(xreal)
hold on
grid on
%plot(yreal)
plot(0,0,'bo')
plot(1.52,1.52,'ro')

title('X Y Position Control for Small K_{\theta} = 2','FontSize', 24)
legend('Hardware','Simulation','Starting Point', ...
'Target Point')

%%
    h_line = findobj(gcf, 'type', 'line');
    set(h_line, 'LineWidth', 3);
    h_axes = findobj(gcf, 'type', 'axes');
    set(h_axes, 'linewidth', 2);
    set(h_axes, 'FontSize', 15);
    xlabel('X (meters)','FontSize', 24)
    ylabel('Y (meters)', 'FontSize', 24);
%%
```

```
%************Hardware Data Visualization for Different Cases***********
%************With/Without Pan Servo*************
%****Different Cruise Speed, Camera Fixed Look-Ahead L****
%****and Vision Subsystem Delay***

%Use csvread command to get raw data
clear all;
%%
X_Y_get = csvread ('X_Y_position.txt');
%%
V     =    0.5.* X_Y_get(:,1);
yaw   =    X_Y_get(:,2);
```

```
Td  = 0.100;         %sampling time

%X_p = zeros(285,1);
%Y_p = zeros(285,1);
X_p = 0;
Y_p = 0;
X = [];
Y = [];

V_x = V .* cos(yaw);
V_y = V .* sin(yaw);
%%
for n = 1:1:285
    X(n) = X_p + V_x(n) .* Td;
    X_p = X(n);

    Y(n) = Y_p + V_y(n) .*Td;
    Y_p = Y(n);
end;
    plot(X,Y);
    hold on;

%%
%Plot oval
r =1 ;
theta=linspace(pi/2,pi*3/2,100);
x1=r*cos(theta)-1;
y1=r*sin(theta)+1;

 plot(x1,y1);
 hold on;

 x2 = linspace(-1,1,100);
 y2 = linspace(2,2,100);
 plot(x2,y2);
 hold on;

 x3 = linspace(-1,1,100);
 y3 = linspace(0,0,100);
 plot(x3,y3);
 hold on;

theta=linspace(-pi/2,pi/2,100);
x4=r*cos(theta)+1;
y4=r*sin(theta)+1;
 plot(x4,y4);
 hold on;

 %%
 figure(2);
 X_sim = [x1',x2',x3',x4'];
 Y_sim = [y1',y2',y3',y4'];
 plot(X_sim-0.3,Y_sim,'b',X,Y,'r');
 axis([-2.5 2 -0.2 2.2]);
 hold on;

```

```
    h_line = findobj(gcf, 'type', 'line');
    set(h_line, 'LineWidth', 3);
    h_axes = findobj(gcf, 'type', 'axes');
    set(h_axes, 'linewidth', 2);
    set(h_axes, 'FontSize', 15);

title('Black Line Guidance Hardware Result - Without Pan Servo');
legend('Real Track','Hardware Result');
xlabel('X (meters)','FontSize', 24)
ylabel('Y (meters)', 'FontSize', 24);
```

APPENDIX B

CPP CODE

```cpp
//Author: Xianglong Lu and Duo Lv
//This is a Wireless LIDAR data Receiver Cpp Code
//Through TCP Socket

#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <ros/ros.h>
#include <sensor_msgs/LaserScan.h>
#include <std_msgs/UInt16.h>

using namespace std;

struct lidar_data {

    int32_t rpm[360];
    int32_t ranges[360];
    int32_t intensities[360];

};

void laser_poll(int sockfd, sensor_msgs::LaserScan
    *scan, std_msgs::UInt16 *rpms) {

    int ret;
    int i;

    int offset = 0;
    struct lidar_data data;

    memset(&data, 0, sizeof(struct lidar_data));

    while(offset < sizeof(struct lidar_data)) {

        ret = recv(sockfd, (char *)&data + offset,
            sizeof(struct lidar_data) - offset, 0);
        if(ret <= 0) {
            break;
        } else {
            offset += ret;
        }
    }


    int32_t rpm = data.rpm[0];
    rpms->data = rpm;

    scan->angle_min = 0.0;
    scan->angle_max = 2.0*M_PI;
    scan->angle_increment = (2.0*M_PI/360.0);
    scan->time_increment = (rpm == 0 ? (1.0 / 360.0) :
```

```
        (60.0 / rpm / 360.0));
    scan->scan_time = (rpm == 0 ? 1 : 60.0 / rpm);
    scan->range_min = 0.06;
    scan->range_max = 5.0;
    scan->ranges.reserve(360);
    scan->intensities.reserve(360);

    for(i = 0; i < 360; i++) {
        float range = (data.ranges[i] < 0 ? 0 :
            data.ranges[i] / 1000.0);
        scan->ranges.push_back(range);
        scan->intensities.push_back(data.intensities[i]);
    }

//    printf("Scan received!\n");
}


int main(int argc, char **argv) {

    // roscpp init

    ros::init(argc, argv, "xv_11_lidar_socket_driver");
    ros::NodeHandle n;
    ros::NodeHandle priv_nh("~");

    printf("xv_11_lidar_socket_driver started.\n");

    // configuration parameter
    string address;
    int port;
    string frame_id;

    priv_nh.param("address", address,
        string("192.168.1.2"));
    priv_nh.param("port", port, 5000);
    priv_nh.param("frame_id", frame_id,
        string("xv_11_lidar"));


    // connect socket
    int sockfd = -1;
    struct sockaddr_in serv_addr;

    if ((sockfd = socket(AF_INET, SOCK_STREAM,
        0)) < 0) {
        printf("Unable to create socket.\n");
        return -1;
    }

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);

    if (inet_pton(AF_INET, address.c_str(),
        &serv_addr.sin_addr) <= 0) {
        printf("Invalid server address\n");
```

```
        return -1;
    }

    if (connect(sockfd, (struct sockaddr *)
        &serv_addr, sizeof(serv_addr))
            < 0) {
        printf("Connect failed.\n");
        return -1;
    }

    printf("Connected to %s:%d\n",
        address.c_str(), port);


    // publisher
    ros::Publisher laser_pub =
    n.advertise<sensor_msgs::LaserScan>("scan",
            1000);
    ros::Publisher motor_pub =
    n.advertise<std_msgs::UInt16>("rpms", 1000);


    while (n.ok()) {
      sensor_msgs::LaserScan scan;
      std_msgs::UInt16 rpms;
      scan.header.frame_id = frame_id;
      scan.header.stamp = ros::Time::now();
      laser_poll(sockfd, &scan, &rpms);
      laser_pub.publish(scan);
      motor_pub.publish(rpms);

    }



    return 0;
}
```

```
<?xml version="1.0"?>

//Establishing Connecting Between
//LIDAR and Linux PC through USB

<launch>
  <node pkg="xv_11_laser_driver"
  type="neato_laser_publisher" name="xv_11_node">

    <!--<param name="port"
    value="/dev/tty.usbserial-A9UXLBBR"/>-->

    <param name="port" value="/dev/ttyUSB0"/>
    <param name="firmware_version" value="2"/>
    <param name="frame_id" value="laser"/>
  </node>

  <node pkg="tf" type="static_transform_publisher"
```

```
  name="base_frame_2_laser"
  args="0 0 0 0 0 0 /base_frame /laser 100"/>

  <node pkg="rviz" type="rviz"
  name="rviz" args="-d rviz_cfg.rviz"/>

  <include file="default_mapping.launch"/>
  <include file="/home/jeffery/catkin_ws/
  src/hector_slam/hector_geotiff/launch/
  geotiff_mapper.launch"/>

</launch>
```

```
//This is a ROS launch file which setup
//key parameters like map_frame, base_frame etc
//Resolution and Other Key parameters of the map
//can be changed here

<?xml version="1.0"?>
<launch>

  <node pkg="hector_mapping" type="hector_mapping"
  name="hector_mapping" output="screen">
    <param name="use_sim_time" value="false"/>
    <param name="pub_map_odom_transform" value="true"/>
    <param name="map_frame" value="map"/>
    <param name="base_frame" value="base_frame"/>
    <param name="odom_frame" value="base_frame"/>
    <param name="fixed_frame" value="laser"/>

    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>

    <param name="map_resolution" value="0.050"/>
    <param name="map_size" value="2048"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_pub_period" value="1.0" />
    <param name="map_multi_res_levels" value="2" />

    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.7" />
    <param name="map_update_distance_thresh" value="0.2"/>
    <param name="map_update_angle_thresh" value="0.9" />
    <param name="laser_max_dist" value="6" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />

    <param name="advertise_map_service" value="true"/>
    <param name="scan_subscriber_queue_size" value="5"/>
    <param name="scan_topic" value="scan"/>
    <param name="tf_map_scanmatch_transform_frame_name"
    value="scanmatcher_frame" />
  </node>

</launch>
```

APPENDIX C

C CODE

```c
//Author: Duo Lv, Xianglong Lu
//This C Code is running on Raspberry Pi
//It's dealing with LIDAR RPM Counting, LIDAR
//Raw Data Analysis and TCP Socket (Wireless SLAM)


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <stdint.h>

#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

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

// uncomment this to debug reads
//#define SERIALPORTDEBUG

// takes the string name of the serial
port (e.g. "/dev/tty.usbserial","COM1")
// and a baud rate (bps) and connects
to that port at that speed and 8N1.
// opens the port in fully raw mode
so you can send binary data.
// returns valid fd, or -1 on error

int serialport_init(const char* serialport, int baud) {
    struct termios toptions;
    int fd;

    //fd = open(serialport, O_RDWR | O_NOCTTY | O_NDELAY);
    fd = open(serialport, O_RDWR | O_NONBLOCK);

    if (fd == -1) {
        perror("serialport_init: Unable to open port ");
        return -1;
    }

    //int iflags = TIOCM_DTR;
    //ioctl(fd, TIOCMBIS, &iflags);     // turn on DTR
    //ioctl(fd, TIOCMBIC, &iflags);    // turn off DTR

    if (tcgetattr(fd, &toptions) < 0) {
        perror("serialport_init: Couldn't
            get term attributes");
        return -1;
    }
```

```
    speed_t brate = baud; // let you override
    switch below if needed
    switch (baud) {
    case 4800:
        brate = B4800;
        break;
    case 9600:
        brate = B9600;
        break;
#ifdef B14400
        case 14400: brate=B14400; break;
#endif
    case 19200:
        brate = B19200;
        break;
#ifdef B28800
        case 28800: brate=B28800; break;
#endif
    case 38400:
        brate = B38400;
        break;
    case 57600:
        brate = B57600;
        break;
    case 115200:
        brate = B115200;
        break;
    }
    cfsetispeed(&toptions, brate);
    cfsetospeed(&toptions, brate);

    // 8N1
    toptions.c_cflag &= ~PARENB;
    toptions.c_cflag &= ~CSTOPB;
    toptions.c_cflag &= ~CSIZE;
    toptions.c_cflag |= CS8;
    // no flow control
    toptions.c_cflag &= ~CRTSCTS;

    //toptions.c_cflag &= ~HUPCL; // disable
    hang-up-on-close to avoid reset

    toptions.c_cflag |= CREAD | CLOCAL;
    // turn on READ & ignore ctrl lines
    toptions.c_iflag &= ~(IXON | IXOFF | IXANY);

    // turn off s/w flow ctrl
    toptions.c_lflag &= ~(ICANON | ECHO |
        ECHOE | ISIG);
    // make raw
    toptions.c_oflag &= ~OPOST; // make raw

    // see: http://unixwiz.net/techtips
    /termios-vmin-vtime.html
    toptions.c_cc[VMIN] = 0;
    toptions.c_cc[VTIME] = 0;
    //toptions.c_cc[VTIME] = 20;
```

```c

    tcsetattr(fd, TCSANOW, &toptions);
    if (tcsetattr(fd, TCSAFLUSH, &toptions)
     < 0) {
        perror("init_serialport: Couldn't
            set term attributes");
        return -1;
    }

    return fd;
}

//
int serialport_close(int fd) {
    return close(fd);
}

//
int serialport_write(int fd, char b) {
    int n = write(fd, &b, 1);
    if (n != 1)
        return -1;
    return 0;
}

//
int serialport_write_buff(int fd, const
    char* buff, int n) {
    int ret = write(fd, buff, n);
    if (ret != n) {
        perror("serialport_write: couldn't
            write whole string\n");
        return -1;
    }
    return 0;
}

//
int serialport_read_until(int fd, char* buf,
    char until, int buf_max,
        int timeout) {
    char b[1];  // read expects an array,
    so we give it a 1-byte array
    int i = 0;
    do {
        int n = read(fd, b, 1);
        // read a char at a time
        if (n == -1)
            return -1;    // couldn't read
        if (n == 0) {
            usleep(1 * 1000);
            // wait 1 msec try again
            timeout--;
            continue;
        }
#ifdef SERIALPORTDEBUG
        printf("serialport_read_until:
```

```
        i=%d, n=%d b='%c'\n",i,n,b[0]);
        // debug
#endif
        buf[i] = b[0];
        i++;
    } while (b[0] != until && i < buf_max
     && timeout > 0);

    buf[i] = 0;  // null terminate the string
    return 0;
}

//
int serialport_flush(int fd) {
    sleep(2); //required to make flush
    work, for some reason
    return tcflush(fd, TCIOFLUSH);
}

#define NR_PACKET 90
#define READING_PER_PACKET 4

#define LIDAR_BUFF_SIZE 4

// must be 22 bytes
struct lidar_serial_packet {

    uint8_t start;
    uint8_t index;
    uint16_t speed;
    uint8_t data[16];
    uint16_t checksum;

};

struct lidar_data {

    int32_t rpm[NR_PACKET *
        READING_PER_PACKET];
    int32_t distance[NR_PACKET *
        READING_PER_PACKET];
    int32_t sig_strength[NR_PACKET *
        READING_PER_PACKET];
};

struct lidar_data lidar_buff[LIDAR_BUFF_SIZE];
int lidar_producer_index = 0;
int lidar_consumer_index = 0;

sem_t lidar_sem;
int lidar_listener = 0;


int get_distance(struct lidar_serial_packet *p,
    int index) {

    if (index < 0 || index > 3)
```

```
        return -1;

    uint8_t *cp = (uint8_t *) &(p->data[index * 4]);

    if ((cp[1] & 0x80) > 0)
        return -1;
    else
        return (cp[1] & 0x3F) << 8 | cp[0];

}

int get_sig_strength(struct lidar_serial_packet *p,
    int index) {

    if (index < 0 || index > 3)
        return -1;

    uint8_t *cp = (uint8_t *) &(p->data[index * 4]);

    return cp[3] << 8 | cp[2];

}

int get_sig_warning(struct lidar_serial_packet *p,
    int index) {

    if (index < 0 || index > 3)
        return -1;

    uint8_t *cp = (uint8_t *) &(p->data[index * 4]);

    return cp[1] & 0x40;

}

void dump_packet(struct lidar_serial_packet *p) {

    int i = 0;
    uint8_t *cp = (uint8_t *) p;

    for (i = 0; i < sizeof(struct
        lidar_serial_packet); i++) {
        printf("%02x ", cp[i]);
    }

    printf("\n");

}

int skip_read(int fd) {

    int i;
    int ret;
    unsigned char c;
    int distance = 0;

    do {
```

```c

        ret = read(fd, &c, 1);
        if (ret < 0) {
            printf("Unable to read from
                serial port!\n");
            return -1;
        }

        if (ret == 0) {
            sleep(1);
            continue;
        }

//      printf("%02x ", c);

        i++;
        if (c == 0xFA) {
            distance = i;
            i = 0;
        }

    } while (distance != sizeof
        (struct lidar_serial_packet));

    for (i = 0; i < sizeof(struct
        lidar_serial_packet) - 1; i++) {
        ret = read(fd, &c, 1);
        if (ret < 0) {
            printf("Unable to read from
                serial port!\n");
            return -1;
        }
//      printf("%02x ", c);
    }

//  printf("\n");

    return 0;
}

int read_packet(int fd, struct lidar_serial_packet *p) {

    int ret;
    int offset = 0;

    while (offset != sizeof(struct lidar_serial_packet)) {
        ret = read(fd, (char *) p + offset,
                sizeof(struct lidar_serial_packet) - offset);
        if (ret < 0) {
            printf("Unable to read from serial port!\n");
            return -1;
        } else if (ret == 0) {
            sleep(1);
            continue;
        } else {
            offset += ret;
        }
```

```c
    }

    return 0;

}


void *lidar_serial_thread(void *para) {

    int i = 0;
    char *serial_dev = "/dev/ttyUSB0";

    // open serial port

    if (para != NULL) {
        serial_dev = para;
    }

    int fd = serialport_init(serial_dev, 115200);
    if (fd < 0) {
        printf("Unable to open serial port \"%s\".\n", serial_dev);
    }

    // read packet

    struct lidar_serial_packet packet;

    skip_read(fd);

    while (1) {

        read_packet(fd, &packet);

        if (packet.start != 0xFA) {
            printf("Unexpected protocol header!\n");
            dump_packet(&packet);
            skip_read(fd);
            continue;
        }

        int index = packet.index - 0xA0;
        float speed = packet.speed / 64.0;


        printf("packet %d at RPM %.2f: ", index, speed);

        for (i = 0; i < 4; i++) {

            int distance = get_distance(&packet, i);
            int sig_strength = get_sig_strength(&packet, i);
            int warning = get_sig_warning(&packet, i);

            if (warning == 0) {
                printf("%d (%d)\t", distance, sig_strength);
            } else {
                printf("%d (%d W)\t", distance, sig_strength);
            }
```

```c

            if(warning >= 0)
                sig_strength = -sig_strength;

            lidar_buff[lidar_producer_index].
            rpm[index * READING_PER_PACKET + i] = speed;
            lidar_buff[lidar_producer_index].
            distance[index * READING_PER_PACKET + i] = distance;
            lidar_buff[lidar_producer_index].
            sig_strength[index * READING_PER_PACKET + i] = sig_strength;

        }

        if(index == NR_PACKET - 1) {
            lidar_producer_index = (lidar_producer_index + 1)
            % LIDAR_BUFF_SIZE;
            sem_post(&lidar_sem);
        }

        printf("\n");
    }

    close(fd);

}

void *lidar_socket_thread(void *para) {

    int error = 0;
    int ret;
    socklen_t len;
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr, client_addr;

    char client_addr_str[16];

    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Unable to create socket.\n");
        return NULL;
    }

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(192.168.0.1);
    serv_addr.sin_port = htons(5000);

    if (bind(listenfd, (struct sockaddr*)
        &serv_addr, sizeof(serv_addr)) < 0) {
        printf("Unable to bind to local
            listening socket %s:%d\n",
                inet_ntoa(serv_addr.sin_addr),
                ntohs(serv_addr.sin_port));
        return NULL;
    }

    if (listen(listenfd, 10) < 0) {
        printf("Unable to listen on %s:%d\n",
```

```c
            inet_ntoa(serv_addr.sin_addr),
                serv_addr.sin_port);
        return NULL;
    }

    printf("Listening started.\n");



    while (1) {
        connfd = accept(listenfd, (struct sockaddr*)
            &client_addr, &len);
        if(connfd < 0) {
            printf("Accept failed.\n");
            puts(strerror(errno));
            return NULL;
        }
        inet_ntop(AF_INET, &client_addr.sin_addr,
            client_addr_str,
                sizeof(client_addr_str));
        printf("Accept connection from %s.\n", client_addr_str);

        while (error == 0) {

            if(lidar_consumer_index == lidar_producer_index)
                sem_wait(&lidar_sem);

            int offset = 0;
            char *p = (char *)&lidar_buff[lidar_consumer_index];

            while(offset < sizeof(struct lidar_data)) {
                ret = send(connfd, p + offset, sizeof
                    (struct lidar_data) - offset, 0);
                if(ret < 0) {
                    error = 1;
                    break;
                }
            }

            lidar_consumer_index = (lidar_consumer_index + 1)
            % LIDAR_BUFF_SIZE;

            printf("Data sent.\n");

        }

        printf("%s disconnected.\n", client_addr_str);
        error = 0;
        close(connfd);
    }

    return NULL;
}


int main(int argc, char **argv) {

```

```c


    sem_init(&lidar_sem, 0, 0);

    pthread_t lidar_socket_thread_tid;
    pthread_t lidar_serial_thread_tid;


    pthread_create(&lidar_socket_thread_tid, NULL,
        lidar_socket_thread, NULL);
    pthread_create(&lidar_serial_thread_tid, NULL,
        lidar_serial_thread, NULL);


    pthread_join(lidar_socket_thread_tid, NULL);
    pthread_join(lidar_serial_thread_tid, NULL);

    sem_destroy(&lidar_sem);


    return EXIT_SUCCESS;
}
```

APPENDIX D

PYTHON CODE

```

Author: Xianglong Lu 16/4/14

# stands for detailed commends

#Hardware: Raspberry Pi 2/3 + Raspberry
#pi camera module

#The goals of this code are:
#1.turning the image from color to grey, extract
#the black line by adjusting the threshold,
#then we find the line. Finally we calculate the
#mass center of this line in camera's region of interest,
# Then calculate angle between the oriantation
#of the robot and mass center of black line.

#2. Feedback the thetae to arduino (through serial
#   port) for the purpose of controlling the
#robot's steering servo. In this case, robot can
#roughly track this black line.

#3. roughly calculate fps of the pi camera, which
#is super important


#import all the modules it needs. time and math
#modules have been installed already.
#cv2 and numpy are ready when you installed
#opencv. Other modules are supposed to be installed
#in python individually.


import cv2
from numpy import linalg as LA
import numpy as np
import io
import picamera
import serial
import matplotlib.pyplot as plt
import pylab as plab
import time
import math

#Here we start the code
#In this case, raspberry pi and arduino uno
#are communicating through serial port (the blue
    #USB cable). Here we define port name is
#ttyACM0 and baud rate is 9600

ser = serial.Serial('/dev/ttyACM0', 9600)
ser.write('0 \n')

# Here we are trying to get the image(vedio
    #stream or just jpeg images). The following lines
#help us get the images or vedio stream and
#store them in frame. This this the way to setup
# a pi camera
```

```
def getImage():
        cap.capture(stream, format = 'jpeg', use_video_port = True)
        frame = np.fromstring(stream.getvalue(), dtype = np.uint8)
        stream.seek(0)
        frame = cv2.imdecode(frame,1)
        return frame

cap = picamera.PiCamera()

#flip the image horizontally or vertically if necessary

cap.vflip = True
cap.hflip = True
#resolution is set to 320x240 to get higher fps
cap.resolution = (320,240)


stream = io.BytesIO()

end = '\n'
comma = ','

#initialize angle thetae
#thetae is the angle between the oriantation
#of the robot and mass center of black line
#in region of interest

thetae_p = 0


#main function begins:

while(1):

    #start counting time(for fps calculating)
    start_time = time.time()

    frame = getImage()
    frame1 = np.array(frame)
    #roi mean the region of interest, we do not
    # need the whole frame of what camera captures
    # we just need the area that we are interested in

    roi = frame1[50:100,50:200]
    #first number is horizontal
    #second number is vertical

    # Convert BGR to GRAY
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

    # Threshold the HSV image to get only blue colors
    ret, output2 = cv2.threshold(gray, 100,
        255, cv2.THRESH_BINARY_INV)
    output2 = cv2.GaussianBlur(output2,(5,5),0)
    #roi = cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)
    #output2 = cv2.inRange(roi,np.array((10,26,33)),
```

```
        #np.array((10,26,35)))
    erode = cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
    dilate = cv2.getStructuringElement(cv2.MORPH_RECT,(6,6))

    # Erode and dilate
    output2 = cv2.erode(output2, erode, iterations = 3)
    output2 = cv2.dilate(output2, dilate, iterations = 5)

    #output2 is the contour
    cv2.imshow('out', output2)
    # Finding contours
    _,contours,_= cv2.findContours(output2,cv2.
        RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)



    #More Info: http://stackoverflow.com/questions/16538774/
    #ealing-with-contours-and-bounding-rectangle-in-
    #opencv-2-4-python-2-7

    areas = [cv2.contourArea(c) for c in contours]

    #if not not areas:

        max_index = np.argmax(areas)
        cnt = contours[max_index]

        cv2.drawContours(roi, [cnt], 0, (0,0,255), 2)

        m1 = cv2.moments(contours[max_index])

        #To calculate the mass center of black line

        u1 = int(m1['m10']/m1['m00'])
        #u1 is mass center horizontal
        v1 = int(m1['m01']/m1['m00'])
        #v1 is mass center vertical

        #u1 v1 can be printed here(in python command line)
        str1 = "u1 is %d"%u1
        str2 = "v1 is %d"%v1
        #print str1
        #print str2


        # To calculate thetae every iteration
        thetae = int((math.atan2(u1-320,150))*180/3.1416)


        # update thetae
        thetae_p = thetae


        #print thetae
        print "The vision feedback theta is %d"%(thetae)
```

```
        sthetae = str(thetae)
        string = sthetae + end
        ser.write(string)



    else:
        ser.write('0 \n')
        print 'a'

    #show frame and roi windows in real time

    cv2.imshow('frame',frame)
    cv2.imshow('roi', roi)

    # It's a way to roughly calculate fps of the
    #camera,using pi 3 and pi camera, it should
    #be around 10Hz.

    print("%s Hertz"%(1/(time.time() - start_time)))


    #wait esc to kill all process
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()

ser.write('0 \n')
ser.close()
cap.close()
```

```
# HSV Color Filtering

import cv2
import numpy as np
import picamera
import io

def getImage():
        cap.capture(stream, format = 'jpeg', use_video_port = True)
        frame = np.fromstring(stream.getvalue(), dtype = np.uint8)
        stream.seek(0)
        frame = cv2.imdecode(frame,1)
        return frame

def nothing(x):
    pass


cap = picamera.PiCamera()
cap.vflip = True
cap.hflip = True
cap.resolution = (320,240)
```

```python
cap.contrast = 0
cap.saturation = 0

stream = io.BytesIO()


cv2.namedWindow('result')

hmin,smin,vmin = 100,100,100
hmax,smax,vmax = 100,100,100


cv2.createTrackbar('hmin', 'result', 0, 179, nothing)
cv2.createTrackbar('smin', 'result', 0, 255, nothing)
cv2.createTrackbar('vmin', 'result', 0, 255, nothing)

cv2.createTrackbar('hmax', 'result', 0, 179, nothing)
cv2.createTrackbar('smax', 'result', 0, 255, nothing)
cv2.createTrackbar('vmax', 'result', 0, 255, nothing)

while (1):
    frame = getImage()

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    hmin = cv2.getTrackbarPos('hmin','result')
    smin = cv2.getTrackbarPos('smin','result')
    vmin = cv2.getTrackbarPos('vmin','result')


    hmax = cv2.getTrackbarPos('hmax','result')
    smax = cv2.getTrackbarPos('smax','result')
    vmax = cv2.getTrackbarPos('vmax','result')


    lower_blue = np.array([hmin,smin,vmin])
    upper_blue = np.array([hmax,smax,vmax])

    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    result = cv2.bitwise_and(frame, frame, mask = mask)

    cv2.imshow('result', result)

    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break

cap.close()
cv2.destroyAllWindows()
```

APPENDIX E

ARDUINO CODE

```
//This Arduino Code is for Longitudinal Inner Loop
//which is (Vdsr, V) Control

#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_PWMServoDriver.h"
#include <math.h>
#include <Encoder.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *rightMotor = AFMS.getMotor(2);
Adafruit_DCMotor *leftMotor = AFMS.getMotor(1);

Encoder re(3,3);

double wR;
double wRp = 0;
double wRn;
double RdVal = 0;
double Radius = 0.024;

double vd = 0.5;
double vd_p = 0;
double vdf;
double vdf_p = 0;

double CR;
double CR_p = 0;
double CR_pp = 0;

double Rerror;
double Rerror_p = 0;
double Rerror_pp = 0;

int PWMR;

double kp = 11.68;
double ki = 23.36;

double alpha = 100;
double h = ki/kp;

long R;
long R_last = 0;
unsigned long Time = 0;
unsigned long sample_time = 100;
double td = 0.100; //

void setup()
{

  AFMS.begin();
  Serial.begin(9600);
  leftMotor->setSpeed(0);
  rightMotor->setSpeed(0);
  leftMotor->run(FORWARD);
  rightMotor->run(FORWARD);
```

```

  leftMotor->run(RELEASE);
  rightMotor->run(RELEASE);

  delay(1000);
}

void loop()
{
  if (millis()<10000)
  {
      if(millis() - Time > sample_time)
        {
           Time = millis();
           GetSpeeds();
        }
  }

  else
  {
    rightMotor->setSpeed(0);
    leftMotor->setSpeed(0);
  }

}

void GetSpeeds()
{
   //Prefilter
   vdf = ( (td*h)*vd + (td*h)*vd_p - (td*h - 2)*vdf_p)/(2 + td*h);

   vdf_p = vdf;
   vd_p = vd;

   R = re.read();
   RdVal = (double)( R - R_last)/(td);
   wR = RdVal*2*3.14159/48;
   wRn = (wR + wRp)/2.0;

   wRp = wR;

   Rerror = vdf - wRn*Radius;
   //Controller
   CR = ((alpha*td*td*ki+2*alpha*td*kp)*Rerror +
   (2*alpha*td*td*ki)*Rerror_p + (alpha*td*td*ki-2*alpha*td*kp)
   *Rerror_pp + 8*CR_p -
   (4-2*alpha*td)*CR_pp)/(2*alpha*td + 4);

   CR_pp = CR_p;
   CR_p = CR;

   Rerror_pp = Rerror_p;
   Rerror_p = Rerror;

   PWMR = int(255.0*CR/7.8);

     if (PWMR>=255) {PWMR=255;}
```

```
    else if (PWMR<=0) {PWMR=0;}

    leftMotor->setSpeed(PWMR);
    leftMotor->run(FORWARD);
    rightMotor->setSpeed(PWMR);
    rightMotor->run(FORWARD);

    R_last = R;

   Serial.print("   ");
   Serial.print( wRn*Radius); //
   Serial.print("   ");
   Serial.println(  PWMR); //

}
```

```
//Robot (v,theta) control
//Go Along a line

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_MotorShield.h>
#include <Servo.h>
#include <math.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
//#include <Encoder.h>
#include "utility/Adafruit_PWMServoDriver.h"

#define Center 20

Servo steer_servo;

Adafruit_BNO055 bno = Adafruit_BNO055();

Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *M1 = AFMS.getMotor(1);
Adafruit_DCMotor *M2 = AFMS.getMotor(2);
#define MAG_OUTPUT 3

int wheelServo;
int count;
int offset;
double radius = 0.024;
double L_r = 0.134;
double pi = 3.14159;
double vx;                          //robot cruise speed
double vx_p=0;
double vx_filtered;
double theta;

unsigned long timeold;
imu::Vector<3> euler_init;
imu::Vector<3> euler;

```

```
double servo_kp = 2;
double servo_kd = 2;
double theta_p = 0;
double start_time;

void setup() {
  bno.begin();
  steer_servo.attach(9);
  steer_servo.write(Center);

  delay(2000);

  Serial.begin(115200); // Initialize serial port to
  //send and receive at 115200 baud
  AFMS.begin();

  pinMode(MAG_OUTPUT, INPUT_PULLUP);// turn on inside
  //pull-up resistor
  attachInterrupt(MAG_OUTPUT-2, pulseCNT, RISING);

  double start_time = millis();
}

double pd_theta(double err, double err_p, double Ts,
  double Kp, double Kd){
double u = Kp*err+ Kd*(err-err_p)/Ts;
  // add roll off later
return u;
}

void loop() {
  if(millis()-start_time<10000){

     get_speed();
     float theta_raw = get_theta();
     theta = double(theta_raw * 180/3.14);

     if(theta<=360 && theta >= 330)
       {theta = theta - 360;
}
     Serial.print(vx_filtered);
     Serial.print("  ,  ");
     Serial.print(theta);
     //Serial.print("  ,  ");
     //Serial.println(wheelServo);

  if(abs(theta<30.00)){
      M1->run(FORWARD);
      M2->run(FORWARD);
      M1->setSpeed(55);
      M2->setSpeed(55);
    }

    else
    {   M1->run(RELEASE);
        M2->run(RELEASE);
      }
```

```

    int wheelServo = Center;

    if (abs(theta) > 3){
      int u =  pd_theta(theta,theta_p,0.1,servo_kp,servo_kd);
      wheelServo = Center + u;

      //Serial.print(" , ");
      //Serial.println(wheelServo);
      if(abs(u)>30)
      {u = 0;}
      else{
      steer_servo.write(wheelServo);
    }
  }

    else{
      steer_servo.write(Center);
    }

      Serial.print("  ,  ");
      Serial.println(wheelServo);

    theta_p = theta;

     delay(100);
   }
   else{
   M1->run(RELEASE);
   M2->run(RELEASE);
   }
}

   void get_speed(){
   vx = (((double)count/12.0)*2.0*pi*radius)*
   1000.0/(millis()-timeold);
   vx_filtered = (vx + vx_p)/2;
   timeold = millis();
   count = 0;
   vx_p = vx;
   }

   void pulseCNT(){
    //Serial.println(count);
    count++;
    //Each rotation, this interrupt function is run twice
  }

   float get_theta(){
   euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
   return (euler.x() - euler_init.x()) * 3.14 / 180.0;

}
```

```
// Outer Loop XY Using IMU

```

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <math.h>
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_PWMServoDriver.h"
#include <math.h>
#include <Encoder.h>
#include <TimerOne.h>
#include <SPI.h>
#include <Servo.h>
/*  ---------------Hardware Setting--------------- */
// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *MotorR = AFMS.getMotor(2);
Adafruit_DCMotor *MotorL = AFMS.getMotor(1);


const double xy_eps = 0.15; // xy satisfying error region

// PID Setting
// Outer Loop P controller Two P controller for theta and dist
// Kp_theta > Kp_dist To be Stable
const double OuterLP_PID_Kp_theta=10;
const double OuterLP_PID_Kp_dist =0.3;


// Inner Loop PI controller Incremental Method
const double InnerLP_PID_Kp =11.68;
const double InnerLP_PID_Ki =23.36; //This number is
//independent of Ts Ki_c=Ki/Ts;
const double Prefilter_Coeff=0.167;


// Servo Setting

Servo myservo;
Servo panservo;
#define servo_center 10
#define panservo_center 83
#define servo_offset_limit 30
double u_wheelServo;
int wheelServo ;
int panServo;

//IMU object and Global Variable
Adafruit_BNO055 bno = Adafruit_BNO055();
imu::Vector<3> euler_init;
imu::Vector<3> euler;

// PWM Control Related Terms
const int PWM_Intitial=0;
const int PWM_UpperLimit=150;
const int PWM_LowerLimit=0;

// Encoder relevant variables for computing speed
```

```
Encoder EncR(3,3);


const int Enc_CPT=48;// Count Per Turn of Encoder
volatile  long EncR_Ticks=0;// counter for Right wheel Encoder
volatile   int Flag_TimerUpdate =1;// Timer Flag to
//Control iterative Every Loop
long Timer_Counter=0;

long EncR_Ticks_p=0; //RTick Record of last update


// Vehicle Basic Parameters
const double WheelRadius =0.024;




/* ----------------- Software Setting ----------------- */
// General Parameters
const double Time_SamplingTime=0.1;// sampling time
//of timer1 period in seconds
const long   Time_StopTimeMS  =7000;
const long   SerialTimeOutMs  =10000;
const int ledPIN=13;
long StartRunTime=0;


/* ------------ Other Function Global Variables ---------- */
// ctrl_inner_loop Global Variables

double wR=0;// angular velocity of right wheel


double v_dsr=0;// set this variable up
double v;
double v_dsr_filtered_p=0;// pre-filter
double Err_v_p=0;

int PWMR_p = 0;
int PWML_p = 0;
// OuterLP Global Vars


double x_dsr=1.52;//
double y_dsr=1.52;
double x=0;
double y=0;
double theta=0;

double x_p=0;
double y_p=0;

/* ------------ Other Function Global Variables ----------- */
//Regulated outputs measured/estimated
//double LinearV; //linear speed of the vehicle
//double AngularV;//angular velocity of the vehicle
```

```
//w.r.t. instantaneous ICC

int TaskFinished_Flag = 0;
// stop the motor and halt when current time reaches TimeMS
void test_stop(long TimeMS){
  if(millis()-StartRunTime>TimeMS || TaskFinished_Flag==1){
        MotorL->run(RELEASE);     // turn off motor
        MotorR->run(RELEASE);
 // digitalWrite(ledPIN,LOW);// turn off led light to
        //indicate test is over
    while(1);
  }
}

// CTRL_XY Global Variables
double LinearV_dsr=0;
double AngularV_dsr=0;

double LinearV=0;
double AngularV=0;
double theta_dsr  =0;


/* ---------- Other Function Global Variables ----------- */
// Iteration according to Timer Flag
// Run Different Main Function According to mode setting
long time_old = 0;
void iterative(int mode){
  if(millis()-time_old > 100){
    enc_update();
    time_old = millis();
  }
  if (Flag_TimerUpdate){
    // Main Function Upadates
    get_wheel_speed();
    switch(mode){
      case 0:
        ctrl_inner_loop();
        break;
      case 1:
      //   CTRL_LineTracking();
        break;
      case 2:
         ctrl_planar_stabilization();
        break;
      default:
        // if nothing else matches, do the default
        // default is optional
      break;
    }
    Flag_TimerUpdate=0;
  }
}
// Update Encoder Register in interrupt
// when interrupt happens set Flag_TimerUpdate
void enc_update(){
  EncR_Ticks  = EncR.read();
```

```
  Flag_TimerUpdate=1;// set flag for this
  Timer_Counter++;
  //Serial.println("timer_update");

  // Remember to reset Encoder if Encoder CPT is high
}

// PWM Saturation Setting
int pwm_saturation(int PWM_In){
  int PWM_Out=PWM_In;
  if(PWM_In>PWM_UpperLimit){PWM_Out=PWM_UpperLimit;}
  if(PWM_In<PWM_LowerLimit){PWM_Out=PWM_LowerLimit;}
  return PWM_Out;
}

// Motor Setting (Replace by Duo)
int motor_set_pwm(int PWML_In,int PWMR_In){
   int PWML_Out,PWMR_Out;
   PWML_Out=pwm_saturation(PWML_In);
   PWMR_Out=pwm_saturation(PWMR_In);
   MotorL->setSpeed(PWML_Out);
   MotorR->setSpeed(PWMR_Out);
}



// compute angular velocity of each wheel through
//Encoder Measurement
//input  : EncR_Ticks,
//output : v
void get_wheel_speed(){
//  Serial.print(" enc increment: ");
//  Serial.print(EncR_Ticks-EncR_Ticks_p);
  // floating point operation on Arduino might cause
  //real-time performance problem
  wR=1.0*(EncR_Ticks-EncR_Ticks_p)/Time_SamplingTime*2*3.1416/Enc_CPT;
   // Iteration
  EncR_Ticks_p =EncR_Ticks;
  v = wR*WheelRadius;

}


// PI controller Incremental Style
//input: u_p,e,e_p (Very Like to put u_p inside but seems difficult )
//output: u
int ctrl_pi_controller(int u_p,double Error, double Error_p){

  int u=0; // control input
  double BV = 7.8;
  double Kp=InnerLP_PID_Kp;
  double Ki=InnerLP_PID_Ki;
  double Ts=Time_SamplingTime;
  double delta_u = (Kp*(Error - Error_p) + Ki*Ts*Error)* (255.0/BV);

  u = u_p + (int)delta_u;
  return u;
```

```
}


void servo_steer(){

       wheelServo = servo_center + u_wheelServo;
       myservo.write(wheelServo);
      // servo saturation control
        if(wheelServo > servo_center+30 )
          {wheelServo= servo_center+30;  myservo.write(wheelServo);}
        if(wheelServo < servo_center-30 )
          {wheelServo= servo_center-30;  myservo.write(wheelServo);}

  //      Serial.print(" wheelServo  ");
  //      Serial.print(wheelServo);
  //      Serial.print(" , ");
      //  myservo.write(wheelServo);
}


// Update desired angular velocity and use PI
//controller to generate control input
// Thus control inner loop (angular velocity )
// input: v_dsr
// output:v
void ctrl_inner_loop(){

  double  v_dsr_filtered=Prefilter_Coeff*v_dsr+
  (1-Prefilter_Coeff)*v_dsr_filtered_p;
   //Error Variables  error between measured output
  //and desire value
  double Err_v = v_dsr_filtered - v;

  // PI Inner loop Controller for v

  int  PWML=ctrl_pi_controller(PWML_p,Err_v,Err_v_p);
  int  PWMR=ctrl_pi_controller(PWMR_p,Err_v,Err_v_p);

  // Set Control Input to Motor
    //motor_set_pwm(55,55);
    motor_set_pwm(PWML,PWMR);

    // Iteration
    PWMR_p=PWMR;
    PWML_p=PWML;

    v_dsr_filtered_p = v_dsr_filtered;

  // inner loop for lateral w
    servo_steer();

}



void imu_setup() {

```

```
    bno.begin();
    delay(2000);
    euler_init =   bno.getVector(Adafruit_BNO055::VECTOR_EULER);
//  imu_bno055.begin();
}

// Get Theta From IMU
double imu_get_theta(){
  euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
  double raw_degree = euler.x();
  if (raw_degree >= 180)
    return (-(360-raw_degree)*3.14/180);
  else
    return (raw_degree*3.14/180);


   // Serial.println(raw);


}


// Restriction Control Action
//input:  In,Th(Threshold),Output UpperLimit Output LowerLimit
//output: Out
double dead_zone_saturation(double In,double Th, double Min,double Max){
  double Out=In;
  if(fabs(In) <= Th) {Out=0;} // threshold Means No response region is [-Th Th]
  else if (In > Max){Out=Max;}
  else if (In < Min){Out=Min;}
  return Out;
}


// Drive Robot to Desired Position with Desired Orientation
// input x_dsr, y_dsr,
// output x   , y    , theta
void ctrl_planar_stabilization(){


  double Kp_theta  = OuterLP_PID_Kp_theta;
  double Kp_dist   = OuterLP_PID_Kp_dist;

  double Ts       = Time_SamplingTime;
  double phi      = atan2((y_dsr-y_p),(x_dsr-x_p));
  // test this function first

//  double  atan2 (double __y, double __x)  // arc tangent of y/x
  theta = imu_get_theta();
  double Err_dist = sqrt(pow((x_dsr-x_p),2)+pow((y_dsr-y_p),2));

  if (Err_dist < xy_eps){
    TaskFinished_Flag=1;
    Serial.println("Task Finished Reach Target!");
    MotorL->setSpeed(0);
    MotorR->setSpeed(0);
    MotorL->run(BACKWARD);     // turn on motor
```

```
    MotorR->run(BACKWARD);
    delay(300);
    test_stop(Time_StopTimeMS);
  }
  double delta_phi= phi+theta;
  double Err_s    = Err_dist*cos(delta_phi);
  //Serial.print(" Err_dist ");
  //Serial.print(Err_dist);
  //Serial.print(" phi: ");
  //Serial.print(phi);
  //Serial.print(" Err_s ");
  //Serial.print(Err_s);
  //Serial.print(" , ");
  //Serial.print(" theta ");
  //Serial.print(theta);
  //Serial.print(" delta_phi ");
  //Serial.print(delta_phi);

  // P Control of Distance and Theta
  AngularV_dsr= dead_zone_saturation((Kp_theta*delta_phi)
    , 0,    -30,   30 );
  v_dsr = dead_zone_saturation((Kp_dist *Err_dist)
    ,  0,     0.3,     0.75 );

  //Serial.print(" v_dsr ");
  //Serial.println(v_dsr);
  //v_dsr = 0;
  u_wheelServo = AngularV_dsr;
  ctrl_inner_loop();
  //motor_set_pwm(45,45);

  //Serial.print(" u_wheelServo ");
  //Serial.println(u_wheelServo);

  // Dead Reckoning, Ts=0.1; May Cause Problem if Running Fast
  //Serial.print(" v: ");
  //Serial.print(v);
  x = x_p+(v*cos(-theta)*Ts);
  y = y_p+(v*sin(-theta)*Ts);

  //Serial.print("  x  ");
  Serial.print(x);
  Serial.print(",");
  Serial.println(y);

  //Iteration
  x_p=x;
  y_p=y;
  #if defined(DEBUG_FLAG) && defined(CTRL_POSITION_DISP)
    //Serial.print("x_p");
    //Serial.print(x_p);
    //Serial.print("y_p");
    //Serial.println(y_p);
  #endif
}

void setup(){
```

```
    myservo.attach(9);
    //panservo.attach(10);
    myservo.write(servo_center);
    //panservo.write(panservo_center);

    imu_setup();
    StartRunTime=millis();// record start time in mS
    Serial.begin(115200);
    AFMS.begin();  // create with the default frequency 1.6KHz
    // Set the speed to start, from 0 (off) to 255 (max speed)
    MotorL->setSpeed(PWM_Intitial);
    MotorR->setSpeed(PWM_Intitial);
    MotorL->run(FORWARD);     // turn on motor
    MotorR->run(FORWARD);
  //  Timer1.initialize(500000);//Timer1.initialize(microseconds);
    //Set timer 100ms
 //   Timer1.attachInterrupt(enc_update); //

}

void loop(){
  test_stop(Time_StopTimeMS);
    // void CTRL_InnerLoop(double LinearV_dsr, double AngularV_dsr)
  iterative(2); // Run in Mode 2-XY
  //myservo.write(20);
  //Serial.println("changing servo");
  //delay(1000);
  //myservo.write(6);
  //delay(1000);
}
```

```
//Main Arduino Code for Track Following

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_MotorShield.h>
#include <Servo.h>
#include <math.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
//#include <Encoder.h>
#include "utility/Adafruit_PWMServoDriver.h"

#define servo_center 6
#define panservo_center 83
#define servo_offset_limit 30
Adafruit_BNO055 bno = Adafruit_BNO055();

Servo myservo;
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *M1 = AFMS.getMotor(1);
Adafruit_DCMotor *M2 = AFMS.getMotor(2);
//Encoder R(3, 3);
#define MAG_OUTPUT 3

```

```

         //rear wheel velovities
int count;
double radius = 0.024;
double L_r = 0.134;
double pi = 3.14159;
double vx;         //robot cruise speed
double vx_p=0;
double vx_filtered;
unsigned long time_stamp = 0;


int sampling_time_ms = 100;


imu::Vector<3> euler_init;
imu::Vector<3> euler;

//Servo servo;
Servo servo_pan;
//Servo servo_tilt;

double servo_kp = 0.6;
double servo_kd = 0.05;
double theta_cam_p = 0;
double K_pan = 0.12;

int wheelServo ;
int panServo;

const int NUMBER_OF_FIELDS = 2; // how many
//comma separated fields we expect
int fieldIndex = 0;            // the current
//field being received
double values[NUMBER_OF_FIELDS];   // array
//holding values for all the fields
double theta_cam;
double theta_imu;
double timeold;

int FLAG; // normal case on track
int sign;


// function declaration
void get_speed();
void pulseCNT();
void servo_steer();
void longitudinal_operation();
void status_print();
double imu_get_theta();
double pd_theta(double err, double err_p,
  double Ts, double Kp, double Kd);



// pid controller for steer servo
```

```
  double pd_theta(double err, double err_p,
    double Ts, double Kp, double Kd){
  double u = Kp*err+ Kd*(err-err_p)/Ts;
  // add roll off later
  return u;
}

void longitudinal_operation(){
    if(abs(theta_cam)<50 && FLAG == 1){
    // release motors if theta_cam is too large or lose track

      if(abs(theta_cam) > 20){
      M1->run(FORWARD);
      M2->run(FORWARD);
      M1->setSpeed(35);
      M2->setSpeed(35);
      }

      else{
      M1->run(FORWARD);
      M2->run(FORWARD);
      M1->setSpeed(32);
      M2->setSpeed(32);
      }
    }
    else{
      M1->run(RELEASE);
      M2->run(RELEASE);
    }
}
void status_print(){
   // Serial.print(" vx: ");
    Serial.print((millis()-timeold)/1000);
    Serial.print(" , ");
    Serial.print(vx_filtered/2);
    Serial.print(" , ");
    //Serial.print(" theta_imu: ");
    Serial.print(theta_imu);
    Serial.print(" , ");
    Serial.print(theta_cam);
    Serial.print(" , ");
    Serial.println(K_pan*theta_cam);
}
void servo_steer(){
      //servo PD
      if(abs(theta_cam)>20)
      {

        int i = K_pan*theta_cam;
      panServo = panservo_center - i;

      if(panServo > panservo_center + 20)
        {panServo= panservo_center + 20;  servo_pan.write(panServo);}
      if(panServo < panservo_center - 20 )
        {panServo= panservo_center + 20;  servo_pan.write(panServo);}

      servo_pan.write(panServo);
```

```
  }
      else{
      servo_pan.write(panservo_center);
      }


      if (abs(theta_cam) > 5){ // set deadzone
        for straight line
        //if(wheelServo > servo_center+30 )
         //{wheelServo= servo_center+30;}
        //if(wheelServo < servo_center-30 )
       //{wheelServo= servo_center-30;}
        double u =  pd_theta(theta_cam,theta_cam_p,
          0.1,servo_kp,servo_kd);
        wheelServo = servo_center - u;
        myservo.write(wheelServo);
      }

      if (abs(theta_cam) <= 5){
      myservo.write(servo_center);

      }
      // servo saturation control
        if(wheelServo > servo_center+30 )
         {wheelServo= servo_center+30;  myservo.write(wheelServo);}
        if(wheelServo < servo_center-30 )
          {wheelServo= servo_center-30;  myservo.write(wheelServo);}
        //myservo.write(wheelServo);
      //}
    // iteration of theta_cam
    theta_cam_p = theta_cam;

}

void get_speed(){
  vx = (((double)count/12.0)*2.0*pi*radius)*
  1000.0/sampling_time_ms;
  vx_filtered = (vx + vx_p)/2;
  count = 0;
  vx_p = vx;
}

void pulseCNT(){
   //Serial.println(count);
   count++;
   //Each rotation, this interrupt function is
   //run once in rising edge
 }

double imu_get_theta(){
  euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
  double raw =  (euler.x() - euler_init.x()) * 3.14 / 180.0;
  if (raw < 6.28)
    return raw;
  else
    return (raw - 6.28);
}
```

```

void setup(){
    myservo.attach(9);
    servo_pan.attach(10);
    myservo.write(servo_center);
    servo_pan.write(panservo_center);
    bno.begin();
    delay(2000);
    euler_init =   bno.getVector(Adafruit_BNO055::VECTOR_EULER);

    Serial.begin(115200); // Initialize serial port
    //to send and receive at 115200 baud
    AFMS.begin();
    pinMode(MAG_OUTPUT, INPUT_PULLUP);// turn on
    //inside pull-up resistor
    attachInterrupt(MAG_OUTPUT-2, pulseCNT, RISING);
    //M1->setSpeed(25);
    //M2->setSpeed(25);
    //M1->run(FORWARD);
    //M2->run(FORWARD);
    timeold = millis();
}
void loop()
{     //servo_pan.write(90);

   if( Serial.available())
  {
    char ch = Serial.read();
    if(ch >= '0' && ch <= '9') // is this an ascii digit
    // between 0 and 9?
    {
// yes, accumulate the value if the fieldIndex is within range
// additional fields are not stored
if(fieldIndex < NUMBER_OF_FIELDS)
{
  values[fieldIndex] = (values[fieldIndex] * 10) + (ch - '0');
}
     }
    else if (ch == ',')  // comma is our separator, so
      //move on to the next field
       {
    values[fieldIndex] = values[fieldIndex] * sign;
    fieldIndex++;   // increment field index
    sign = 1;
}
else if (ch== '-')
{
    sign = -1;
}
else
{
  // any character not a digit or comma ends the
  // acquisition of fields
  // in this example it's the newline character sent
  //by the Serial Monitor
  values[fieldIndex] = values[fieldIndex] * sign; //last number
  // print each of the stored fields
```

```
  theta_cam = values[0]/100; //get degree error
  FLAG = values[1]; //get FLAG

  for(int i=0; i < min(NUMBER_OF_FIELDS, fieldIndex+1); i++)
  {
    //Serial.println(values[i]);
    values[i] = 0; // set the values to zero, ready
    //for the next message
  }
  fieldIndex = 0;  // ready to start over
  sign = 1;
  //robot(Rtarget,Ltarget);
}
//theta_cam = values[0]; //get degree error
//FLAG = values[1]; //get FLAG

        //Serial.print("serial ch: ");
  //Serial.print(ch);
  //Serial.print("value0: ");
  //Serial.print(values[0]);
  //Serial.print(" value1: ");
  //Serial.println(values[1]);

    }

  if(millis()-time_stamp >= sampling_time_ms){// inner loop
        //Serial.print("inner loop run");
    //Serial.print(" theta_cam: ");
    //Serial.print(theta_cam);
    //Serial.print(" FLAG: ");
    //Serial.println(FLAG);

    // update and print status
    get_speed();
    time_stamp = millis(); //update current time


    theta_imu = imu_get_theta();
    //status_print();

    // robot control
    longitudinal_operation();

    servo_steer();

//    status_print();
    //
}
}
```