ABSTRACT

Title of Document: IMPLEMENTATION OF KALMAN FILTER TO TRACKING CUSTOM FOUR-WHEEL DRIVE FOUR-WHEEL-STEERING ROBOTIC PLATFORM

Michael Stanley

Master of Science, 2010

Directed By: Professor Christopher C. Davis

Department of Electrical and Computer Engineering

Vehicle tracking is an important component of autonomy in the robotics field, requiring integration of hardware and software, and the application of advanced algorithms. Sensors are often plagued with noise and require filtering. Additionally, no single sensor is sufficient for effective tracking. Data from multiple sensors is needed in order to perform effective tracking. The Kalman Filter provides a convenient and efficient solution for filtering and fusing sensor data as well as estimating noise error covariances. Consequently, it has been essential in tracking algorithms since its introduction in 1960.

This thesis presents an application of the Kalman filter to tracking of a custom four-wheel-drive four-wheel-steering vehicle using a limited sensor suite. Sensor selection is discussed, along with the characteristics of the sensor noise as related to meeting the requirements of the Kalman filter for guaranteeing optimality. The filter requires the development of a dynamical model, which is derived using empirical data methods and evaluated. Tracking results are presented and compared to unfiltered data.

IMPLEMENTATION OF KALMAN FILTER TO TRACKING CUSTOM FOUR-WHEEL DRIVE FOUR-WHEEL-STEERING ROBOTIC PLATFORM

By

Michael Stanley

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2010

Advisory Committee:

Professor Christopher C. Davis, Chair
Professor Gilmer L. Blankenship
Adjunct Associate Professor Gregory A. Schultz

© Copyright by

Michael Stanley

2010

Dedication

To my family, whose unwavering support and motivation have encouraged me to pursue my every dream.

Acknowledgments

My experience with the University of Maryland over the last six years of undergraduate and graduate degrees has enriched my life beyond my expectations. The facilities and excellent people have provided me with many stimulating and educational opportunities. I would like to start by thanking Dr. Blankenship for providing me such opportunity in both undergraduate and graduate research and study. His advice and instruction have been essential to my studies, and I am indebted to him for all his help. Techno-Sciences and TRX Systems have both been so helpful as well, and I cannot thank them enough for giving me so many real world experiences with engineering. Many thanks go to Dr. Teolis; she has been very kind and extremely helpful both professionally in TRX Systems, and in helping guide the direction of this thesis. I would not have been able to complete this research without the support of the members of the Autonomous Systems Lab (ASL), where I have worked for the last 3 years. Working with Jared Napora, Jaymit Patel and John Karvounis in the lab and on hardware, software, and algorithm development has been very educational and enjoyable. John has been a key player in helping me complete this research, while being a great friend. I am truly thankful for all he has done in the last few months. I would like to acknowledge certain members of the University of Maryland Formula SAE racing team for challenging my methods, along with advisor Gregory Schultz for providing me with an opportunity to apply my studies to real-world applications during my undergraduate degree. Mike Cook and Vegard Hamso have both been great friends as well as excellent colleagues providing great insight to difficult research problems and helping me implement my ideas. These experiences have proven invaluable to my maturation as an engineer. I am certain I would not have been able to achieve this milestone of academia without the support of those mentioned herein, as well as many others.

Table of Contents

Dedication ii

Acknowledgments iii

Table of Contents v

List of Tables viii

List of Figures ix

Chapter 1: Introduction, Purpose and Chapter Summary 1

1.1. Introduction 1

1.2. Plans, Methods and Approach 2

1.3. Chapter Summary 3

Chapter 2: Robot Hardware and Sensors 5

2.1. Robot overview 5

2.2. Sensors and their characteristics 9

2.2.1. Gaussian Distribution 10

2.2.2. Rate Gyro 12

2.2.3. Wheel Speed Sensors 15

Chapter 3: Theory and Application 23

3.1. Steering Dynamics 23

3.1.1. Wheel Odometry 26

3.2. System Identification 30

3.2.1. Selecting a Model Structure 34

3.2.2. Making a MIMO ARX Model 36

3.2.3. Evaluating Models 42

3.3. Kalman Filter 47

3.3.1. Background and Governing Equations 48

3.3.2. Incorporating the dynamical model into the filter 54

3.3.3. Sensor fusion 57

3.4. Software 59

3.4.1. ASL Framework 59

3.4.2. MATLAB .NET deployment 61

Chapter 4: Tracking Results 63

Chapter 5: Future Work 67

5.1. Camera as a sensor 68

5.2. Optic Flow and Image Translation 70

Appendix A: CAD 75

A.1. Platform CAD Drawings 75

Appendix B: Kalman filter MATLAB Code 78

Bibliography 79

List of Tables

Table 2-1: Robot Components 6

Table 2-2: Quadrature Encoder State Transition Table [11] 22

Table 3-1: Comparison of INU and Satellite sensor features [21].............................. 57

List of Figures

Figure 2-1: Tamiya TXT-1 [4] 5

Figure 2-2: Modified Tamiya TXT-1 robot with all sensors present. Side view 7

Figure 2-3: Modified Tamiya TXT-1 robot with all sensors present. Corner view 8

Figure 2-4: Gaussian distribution [7] 11

Figure 2-5: TRX INU 13

Figure 2-6: Histogram of Gyro Data taken from straight line test 14

Figure 2-7: Optical Encoder Functional Depiction [9] 15

Figure 2-8: E4P Optical encoder components 16

Figure 2-9: E4P Encoder shown on robot stub axle 17

Figure 2-10: Quadrature Encoder Pattern [9] 18

Figure 2-11: Wheel speed sensor noise 19

Figure 2-12: Data Acquisition Board Layout 20

Figure 2-13: Data Acquisition Board Schematic 21

Figure 2-14: Data Acquisition Board... 21

Figure 3-1: Instant Center of Rotation (ICR) Concept [12].. 25

Figure 3-2: Four wheel Parallel Steer [12] 26

Figure 3-3: Tire deformation [16].. 29

Figure 3-4: Black Box Model 32

Figure 3-5: Identification Toolbox Startup 37

Figure 3-6: Linear Parametric Models Window 39

Figure 3-7: ARX Model Structure Selection Window 41

Figure 3-8: Model Output Window showing the model output for $\boldsymbol{d\Theta}$ from three models. 43

Figure 3-9: Model output residuals for channel $\boldsymbol{d\Theta}$ 44

Figure 3-10: Model output for channel $\boldsymbol{V}$ 45

Figure 3-11: Model output residuals for channel $\boldsymbol{V}$. 46

Figure 3-12: Process-Estimator.. 48

Figure 3-13: The ongoing Kalman filter cycle [20]... 51

Figure 3-14: ASL Software Framework Hierarchy [3].. 60

Figure 4-1: Smooth Plotted path with indicated start and stop points. Units in meters
.... 65

Figure 4-2: Wiggly Plotted path with indicated start and stop points. Units in meters
.... 65

Figure 4-3: Wiggle and turnaround plotted path with indicated start and stop points.
Units in meters 66

Figure 5-1: Rolling Shutter [25] 70

Figure 5-2: Testing Hallway 71

Figure 5-3: Optic Flow imagery of moving keyboard 72

Figure 5-4: Image template probability map of sliding keyboard test 73

Figure 5-5: Align TREX Hobby helicopter [27]... 74

Figure A-1: Platform Modifications CAD 76

Figure A-2: Custom stub-axle for wheel speed sensor application 77

Chapter 1: Introduction, Purpose and Chapter Summary

1.1. Introduction

Methods to improve autonomous navigation are quickly becoming a significant research topic, attracting students and research companies alike [1]. Between the need to perform hazardous or mundane tasks, and the desire to simplify our every-day lives; autonomous systems, whether they be augmenting user interaction; or exhibiting complete self-reliance, deliver useful application of our ever-more capable computation platforms to the hardware systems we design.

The first step towards building an autonomous system is being able to estimate the current state of the system, in our case a robotic platform's position. In this thesis, we employ a Kalman filter to perform on-board state estimation, which is equivalent to developing a tracking system for our robot.

Inertial sensors are often used in tracking applications when very high data rates are required or when other means for tracking such as GPS are either unavailable or impractical. In this research, the robotic platform is generally operating indoors, so access to GPS or other global tracking methods is not possible. Even outdoors, where GPS is accessible, robot motion is on such a small scale that cost-effective GPS units would be useless in tracking. Most current commercially-available, high-quality GPS receivers are only accurate to about three meters [2], and individually give poor information regarding heading unless moving large distances in the same direction. This precludes their use for tracking a highly maneuverable platform such as the four-wheel steering (4WS) robot under present study.

The research presented in this thesis is supported by the University of Maryland Autonomous Systems Laboratory (ASL). Projects in this facility include wireless mesh network systems as presented in [3], distributed sensing systems, simultaneous localization and mapping (SLAM), and autonomous helicopter and ground systems. In the ASL, numerous hardware, software, and sensor assets have been developed to support these various projects, which are leveraged for this research. Many research topics focus on searching buildings or path-optimization or similar, but these rely on at least some form of self-positional awareness which this thesis attempts to attain for the vehicle under research. This research is directly applicable to an ongoing study by the ASL in distributed SLAM, by affording the study a means of robotic vehicle tracking.

1.2. Plans, Methods and Approach

The goal of this thesis is to apply a Kalman filter to track a robotic platform using a limited sensor suite. Since no external absolute sensors such as GPS or fixed reference point tracking methods are employed, the current system is not expected to have perfect accuracy over long distances. Nonetheless, using only on-board measurements of wheel encoder and inertial sensors, the Kalman filter approach demonstrated herein exhibits superior performance in tests compared to model-only and simple sensor-based tracking methods.

First, the selection and modification of a suitable robot is discussed. Actuators and sensors added to the robot will then be addressed, along with the hardware necessary to interface with them and perform high-level algorithms. A low-level data acquisition device for sensor interface developed by ASL will be summarized in the Wheel Speed Sensors section, along with the algorithms on-board to decode the sensor data.

The design process behind implementing the Kalman filter for tracking includes sensor characterization, system identification, and incorporation of software tools. Kalman filter implementation will be achieved using the MATLAB software package. Access to these methods in the primary coding environment is achieved using the MATLAB deployment tools, which will be briefly discussed in the MATLAB .NET deployment section of this thesis.

The Kalman filter and other ancillary methods are implemented entirely using an off-the-shelf, capable Core 2 Duo computer. The software architecture used to allow embedding of the computer device is discussed along with Software. Microsoft development products are used and the primary coding environment is in C#.

1.3. Chapter Summary

Chapter 2 introduces the platform on which the research is being conducted. A list of all components incorporated into the vehicle is given, several pictures of the robot are provided, and the sensors added to the vehicle are characterized.

The next chapter discusses the theory behind the work, along with the implementation itself. The necessary vehicle dynamics are discussed, along with basic methods of odometry. Having these dynamics and tracking methods in mind, the overall character of the robot is uncovered by means of system identification; a mathematical description of the behavior of the robot. This is followed by the main topic of the thesis, the Kalman filter. Background information and relevant equations are described, followed by the full implementation of the filter using the model defined. The argument for sensor fusion is posed, followed by the relevant changes necessary to the Kalman filter.

Chapter 4 presents basic evaluation of the results comparing the filters' performance to simpler methods of tracking. Finally, Chapter 5 introduces ideas for future improvement of the work performed, along with concepts for other applications of the presented technology. Methods to integrate a camera to the sensor suite for additional feedback are investigated and tested.

Chapter 2: Robot Hardware and Sensors

2.1. Robot overview

This thesis deals specifically with wheeled robotics, and for the purpose of scalability of the algorithms: is decidedly chosen to be a simple four-wheeled car. The wheeled platform itself; rather than being designed from the ground up, was built upon a commercially available off-the-shelf remote controlled car; the TXT-1 four-wheel-drive (4WD), four-wheel-steering (4WS) rock crawler truck. The availability of components for the RC market lends itself to ease of modification of said platform to suit our specific needs. The car in its original form is shown in Figure 2-1.

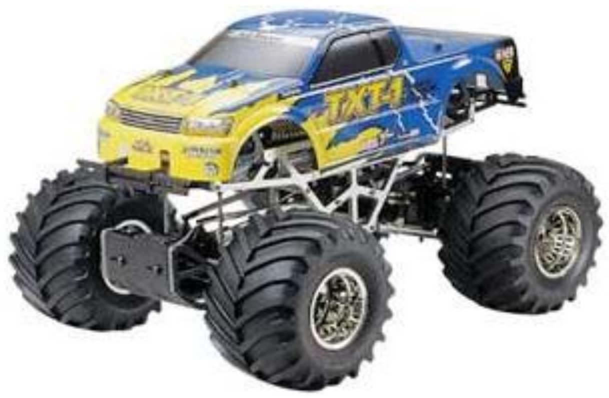


Figure 2-1: Tamiya TXT-1 [4]

Modification is needed in order to support sensors, data gathering devices, computing platforms, and control interfaces. The relevant components are listed in Table 2-1.

Quantity	Component
1	Tamiya TXT-1 Monster Truck 4WD
1	Set of Custom ASL Platform and Brackets
1	AOpen Core 2 Duo miniPC (windows XP Pro)
1	CarNetix CNX-1900 DC-DC Regulator
1	Novak Goat Crawler ESC
1	Parallax Servo Controller
1	ASL Sensor Board V2
1	Logitech QuickCam Pro 9000 Webcam
2-4	11.1V 6600mAh Li-Ion Battery Pack
2	8.4V 4200mAh Battery Pack
4	US Digital E4P Miniature Optical Encoder Kit
1	LynxMotion Pan-Tilt Kit [5]
2	Futaba HS-5645MG digital high torque servo [6]
1	TRX Systems Inertial Navigation Unit (INU)

Table 2-1: Robot Components

To support the devices not normally found on RC cars, some custom machining was performed. An aluminum platform and brackets support the pan-tilt device (which holds the camera), the AOpen miniPC Core 2 Duo computer, and the DC-DC power converter, which powers everything but the Novak Motor controller and the servo actuators. Other components are present, but not discussed for their irrelevance to this research. CAD drawings of this platform and brackets are shown in the Platform CAD

Drawings appendix. Images of the completed robotic platform are shown in Figure 2-2 and Figure 2-3.

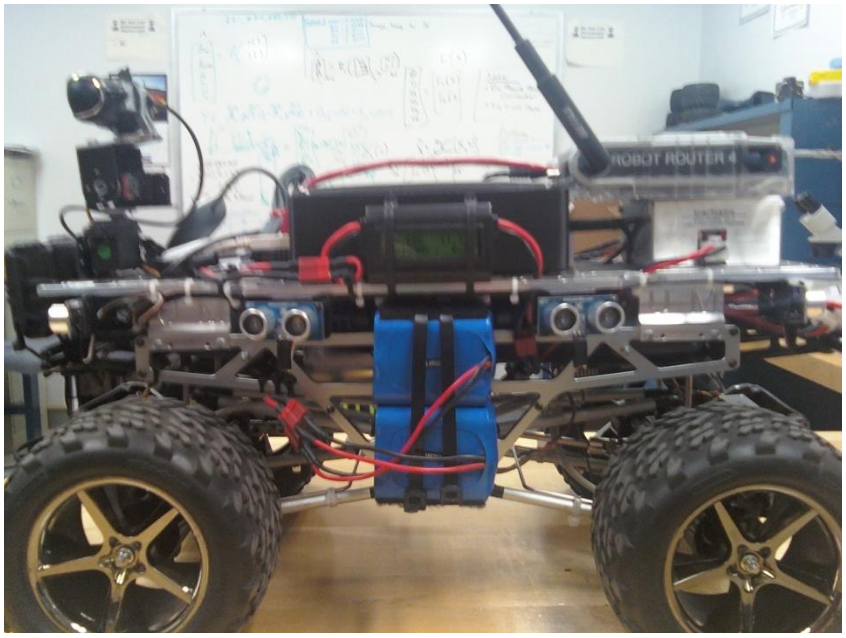


Figure 2-2: Modified Tamiya TXT-1 robot with all sensors present. Side view

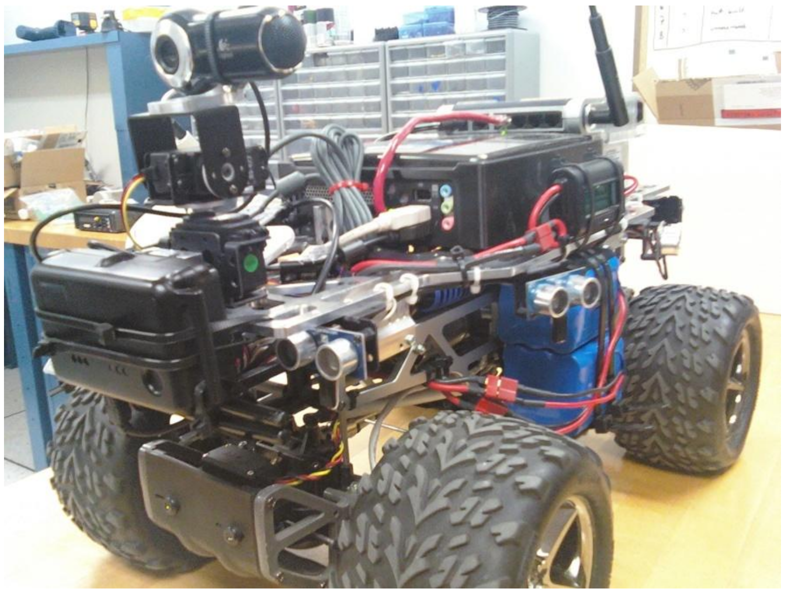


Figure 2-3: Modified Tamiya TXT-1 robot with all sensors present. Corner view

The Novak Goat crawler Electronic Speed Control (ESC) and Futaba HS-5645MG servos impact the behavior of the robot. The Goat is responsible for driving the stock motors, and hence the propulsion of the robot, while the servos control steering, front and rear. Control of these actuators is achieved through serial command of the Parallax Servo controller, and their response will be discussed in the System Identification portion of this document.

Sensors used in this thesis include the TRX INU, and the US Digital E4P optical encoder. Use of the Logitech QuickCam webcam is discussed for future application, but was not implemented in the present work. Data gathering of encoder data was performed using the custom ASL Sensor Board V2, as discussed with the sensors themselves in the Wheel Speed Sensors section.

2.2. Sensors and their characteristics

A sensor is a device that measures a physical quantity and converts it into a signal, read by an observer or instrument. Sensors come in all shapes and sizes, and measure countless quantities. The sensors used in this thesis all measure quantities of physical motion (displacement or rotation). Quadrature wheel encoders are used to detect individual wheel speeds, and from these measurements, velocity and relative heading are calculated. A MicroElectroMechanical Systems (MEMs) gyroscope in an Inertial Navigation Unit (INU) is used in an attempt to correct the errors arising from wheel slip and tire deformation discussed in the Steering Dynamics section to improve the measurement of vehicle heading. Imaging-based algorithms are also briefly discussed in the Future Work chapter for their potential application of further improving the estimate of vehicle heading, and potentially for vehicle velocity corrections.

The noise of a sensor is critical in implementing a filter. To discover the nature of the noise, a controlled experiment is executed. This is achieved by driving the vehicle on a straight path at a constant speed and gathering data. Since the sensors behave in a linear manner, deviation from this scenario does not present appreciable changes in the noise. If this were not true, sensor covariance values change over time, mapped by the mean of the sensor value. Noise profiles are created by subtracting the means from the data and plotting histograms.

2.2.1. Gaussian Distribution

Data returned from sensors can be thought of as the true parameter value with noise superposed upon it. The sensor noise can be characterized by a probability distribution. It is important to know the nature of this distribution when using a filter for the data to ensure all assumptions made in the derivation of the filter are satisfied. The Kalman filter, for instance, provides an optimal recursive estimate of the state of a linear system with the assumption that the sensor noise is *Gaussian-zero-mean*, so it is important for the sensors to exhibit this behavior.

The Gaussian distribution (otherwise known as the Normal Distribution) is a continuous probability distribution that often gives a good description of data that clusters around the mean. The central limit theorem also provides that under certain conditions the sum of a number of random variables with finite means and variances approach a normal distribution as the number of random variables increase [7]. This implies that data may start to appear *Gaussian* after some time even if the noise for a particular measurement may not be Gaussian. The Gaussian distribution is completely characterized by its mean and standard deviation (μ and σ, respectively). The equation for the Gaussian distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} * e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

This is often abbreviated:

$$N(\mu, \sigma^2)$$

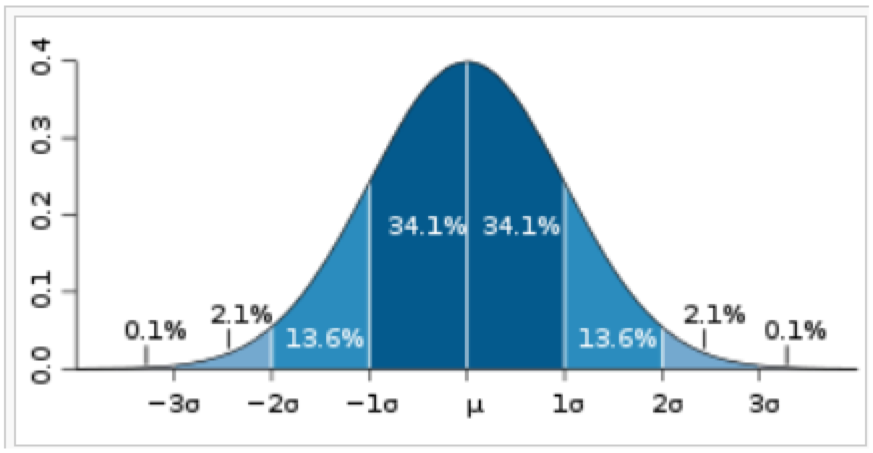


Figure 2-4: Gaussian distribution [7]

When a sensor exhibits a Gaussian profile, Figure 2-4 shows how frequent the data may appear in certain regions. The dark blue region is within one standard deviation from the mean. For a variable with a normal distribution, values will fall in within one standard deviations of the mean about 68% of the time, two standard deviations from the mean (medium and dark blue) about 95% of the time, and three standard deviations (light, medium, and dark blue) from the mean about 99.7%of the time [7]. By plotting a histogram of data, one can compare the frequency of data appearing in these regions, and hence the similarity of the noise to the Gaussian distribution.

The family of Gaussian distribution is closed under linear transformation. That is, if X_i are *independent* Gaussian random variables with means μ_i and standard deviations ${\sigma_i}^2$, then their scaled sum $X = \sum_i a_i X_i$ is Gaussian with parameters $\mu = \sum_i a_i \mu_i$, and $\sigma^2 = \sum_i {a_i}^2 {\sigma_i}^2$. This is an important fact in deriving the constituents of the covariance matrices found in the Kalman filtering section.

As will be seen below, raw measurements of a single sensor are not necessarily used directly as input to the Kalman filter. Scaling of sensor data or multiplying two sensors' data together results in a different noise profile than the original. Properties of the Gaussian distribution can be used to find the new process covariance in this case.

2.2.2. Rate Gyro

Measuring heading is difficult with wheel speed sensors alone, as discussed in the Wheel Odometry section. *Yaw rate* sensors may then be employed to either improve upon this, or replace the measurement entirely; as integrating angular rate yields vehicle angle. To this purpose, an INU is applied for its internal *gyro* (a device that measure yaw rate) sensor. Information returned from the INU is parsed and yaw information alone is used from the INU to serve for heading correction.

TRX Systems provides an integrated INU package with full 6 degrees of freedom (6-DOF) through the use of Analog Devices ADIS16355 [8]. This is a highly functional device, capable of resolving 300°/s in rotation and ±10g's of acceleration on all axes. The TRX INU (shown Figure 2-5: TRX INU) performs sophisticated calibration routines, and sends full state information representing the INU's rotation and acceleration over

USB. The device returns yaw as a cumulative measure, showing complete yaw relative to the device's startup angle. Since the dynamics of the system are generating yaw velocity (as will be seen in the System Identification section), the difference is taken of this data to effectively use the INU as a single axis gyro.

Figure 2-5: TRX INU

The noise profile of this yaw sensor exhibits behavior very close to Gaussian-zero-mean while sitting still. Additional disturbances are present when in use on the robot, and must be included in the total characterization. For this reason, data is taken in the manner described in the opening to this section, to encompass all sources of noise. These yaw disturbances are due in part to tire non-uniformity, drive-train friction, and longitudinal acceleration through solid axle suspensions causing unequal vehicle roll front and rear. Gyro drift would manifest itself as a mean-shift, but the algorithms in the TRX INU effectively remove this. The data in Figure 2-6 shows a histogram of the gyro data in the case of the vehicle being driven in a straight line for sensor characterization.

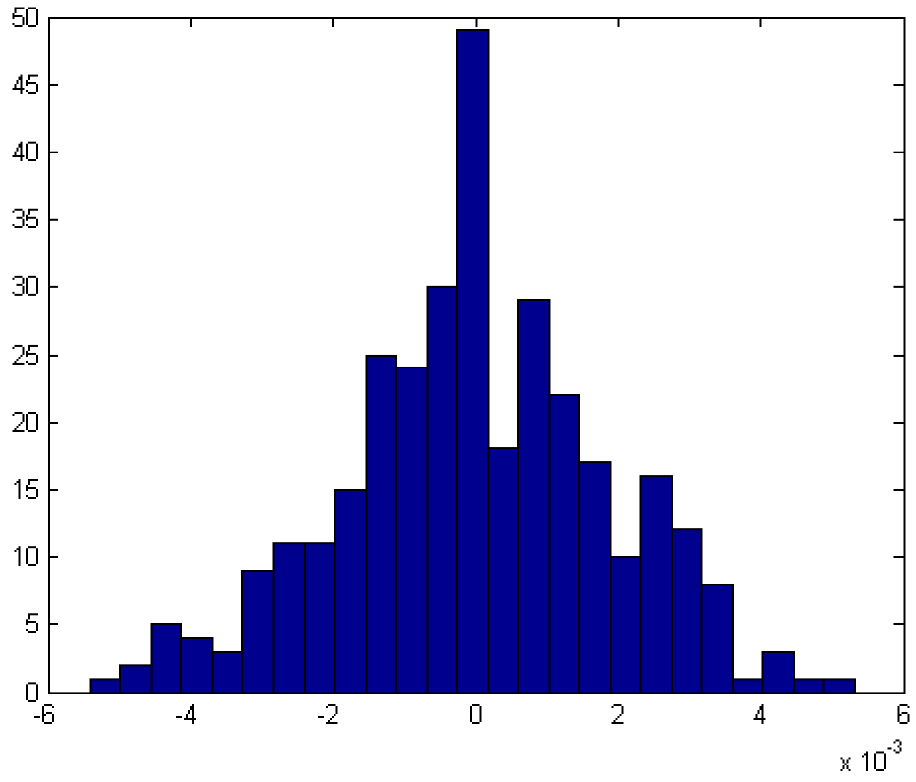


Figure 2-6: Histogram of Gyro Data taken from straight line test

Empirical measurement of the variance using MATLAB shows that:

$$\sigma^2 = 3.5 * 10^{-6}$$

This will be used in the Kalman Filter section as the covariance of the yaw-sensor observation.

2.2.3. Wheel Speed Sensors

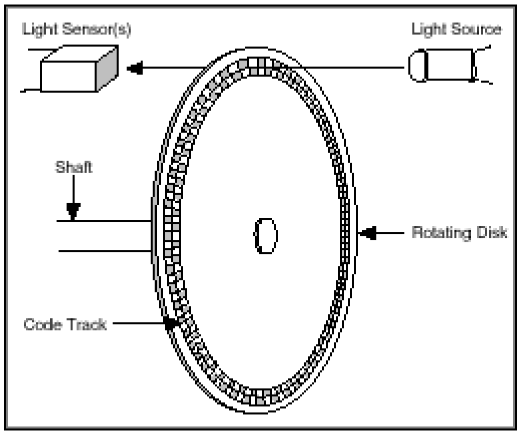


Figure 2-7: Optical Encoder Functional Depiction [9]

Measurement of individual wheel speed is convenient for the purpose of position and velocity feedback of the vehicle. Tachometer type measurement devices exist for accurate velocity measurement, but are often unwieldy and expensive. Furthermore, since we are interested in measuring the total distance covered by each wheel, accurate positional feedback is decidedly the best choice, in order to avoid accumulated velocity integration errors. For continuous rotation applications (as is the case with a drive-wheel), a rotary encoder is really the only option. Encoders may be magnetic (operating on the "Hall effect"), capacitive, or optical. They may be analog or digital. For small applications as is the case with our 4WD robotic platform, digital optical encoders are predominantly used. Digital Optical Encoders have three major elements: a coded wheel, which is a pattern of 'teeth', a light source, and a measurement device. The wheel may be *reflective* such that its pattern either reflects or absorbs the source, or *transmissive* such that the source is either transmitted or occluded. Figure 2-7 shows a basic optical encoder, with a light source being intermittently occluded by some rotary pattern. This is a transmissive type optical encoder. Our encoder is very similar in design, only that the source and receiver are on the same side of the coded wheel (reflective type). An image of the encoder used for our robot is shown taken apart in Figure 2-8. The optical emitter and detector exist on the PCB shown second from the left, followed by the coded wheel second from the right. The encoder is then shown mounted on the robotic platform in Figure 2-9. Stub-axle components were custom machined in order to support the use of this sensor on our robot, as is shown in the Platform CAD Drawings appendix.

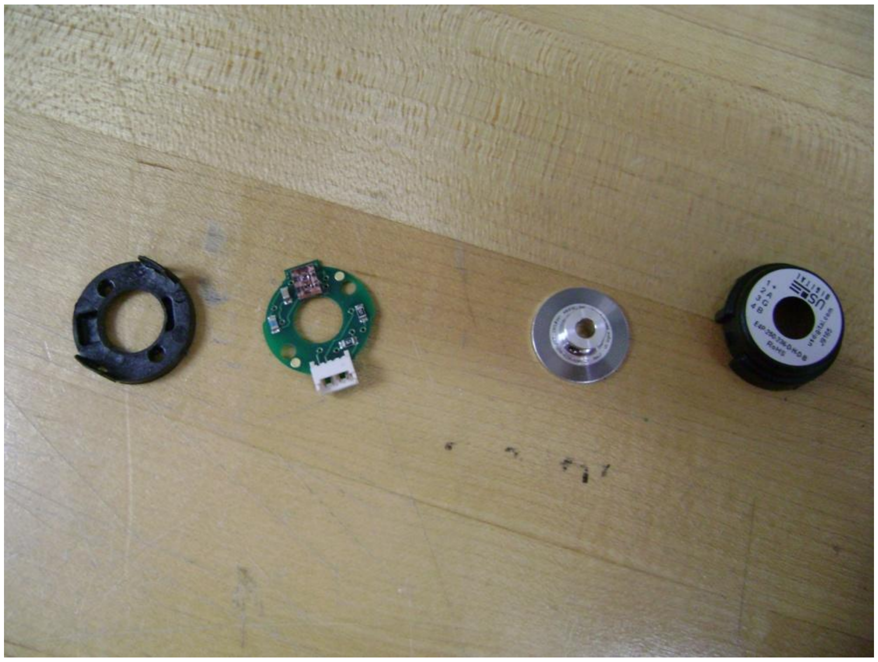


Figure 2-8: E4P Optical encoder components

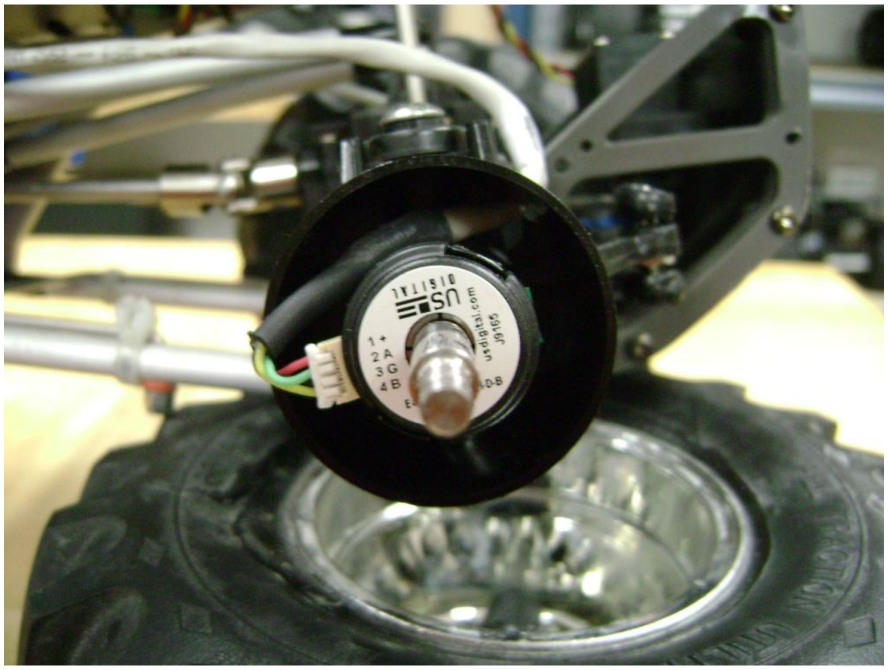


Figure 2-9: E4P Encoder shown on robot stub axle

The encoder works by emitting light from the source, having that light be absorbed or reflected by the particular position of the wheel and then measured by one or more detectors. That information is transformed to a digital code to be analyzed by a microcontroller that determines where the wheel is in its rotation, how fast it is moving, and which direction it is rotating.

The design of the coded wheel is very important in determining its operation. The pattern on the wheel determines the resolution of the encoder (how many cycles per rotation), the ability to measure direction, and whether it is a relative or absolute encoder. Absolute digital type encoders produce a unique digital code for each distinct angle of the shaft [10], while relative encoders only show relative motion to the initialized state. The resolution is an obvious feature; the number of outputs generated per rotation. The ability to measure direction requires a pattern that is either entirely unique for every angular position of the wheel (i.e. an absolute encoder) where the direction is obvious, or a cleverly designed periodic pattern. This pattern is common in encoder designs, and when it is employed, the device is called a *Quadrature Encoder*. Using two code tracks with sectors positioned 90° out of phase (Figure 2-10) the two output channels of the quadrature encoder indicate both position and direction of rotation. If A leads B, for example, the disk is rotating in a clockwise direction. If B leads A, then the disk is rotating in a counter-clockwise direction. Therefore, by monitoring both the number of pulses and the relative phase of signals A and B, you can track both the position and direction of rotation [9].

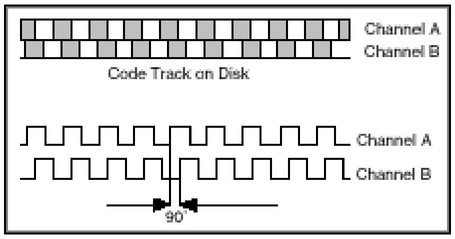


Figure 2-10: Quadrature Encoder Pattern [9]

While with our robotic platform it is possible to determine direction by other means (such as a measurement from the image frame), it is markedly more straightforward to put this task on the same sensor by selecting the appropriate device. The device used in this research is a relative, 1000 counts/revolution, quadrature encoder.

The nature of the noise of these sensors is congruent with requirements posed below in the Kalman Filter section. A histogram of the noise is shown below in Figure 2-11. Variance of the sensor is calculated to be:

$$\sigma^2 = 5.3 * 10^{-7}$$

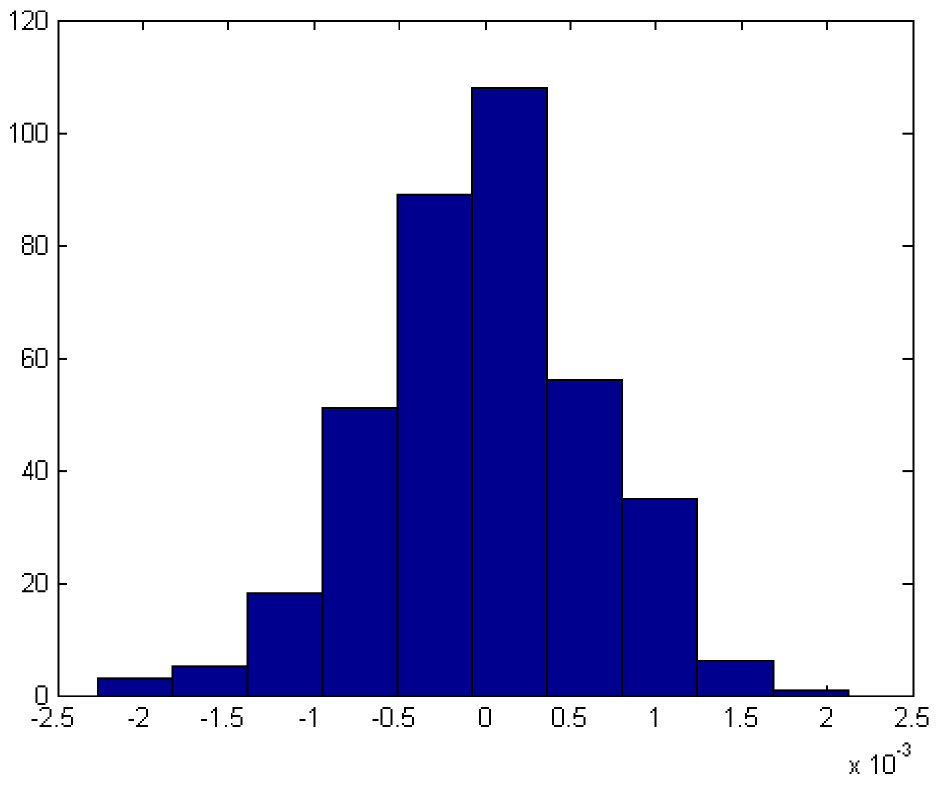


Figure 2-11: Wheel speed sensor noise

In order to use these encoders at the level of software the algorithms are written, some interface must be used. Therefore, members of ASL developed a USB data acquisition device, capable of interpreting the signals from the encoder and sending each individual wheels' speed and direction. This board was designed in Eagle and manufactured in-house. The design and an image of the board are shown in Figure 2-12, Figure 2-13, and Figure 2-14. Interface to the primary computer is through USB, using a message structure of our own definition. The messaging is done synchronously at 1/15th of a second, based on its CPU clock, both to maintain real-time measurements and to ease the integration of data process at the higher level of abstraction in C# on the primary computer.

The core computing platform of this data acquisition device is the microchip DsPIC30F6012A microcontroller. This is a convenient device for signal acquisition, basic algorithms, and communications.

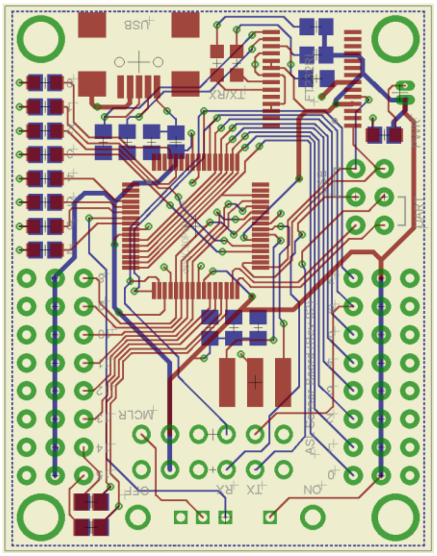


Figure 2-12: Data Acquisition Board Layout

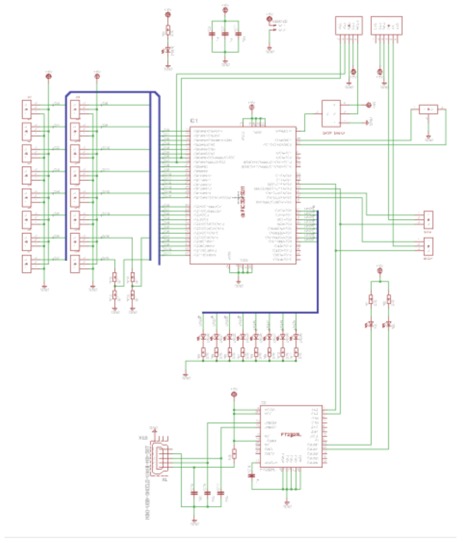


Figure 2-13: Data Acquisition Board Schematic

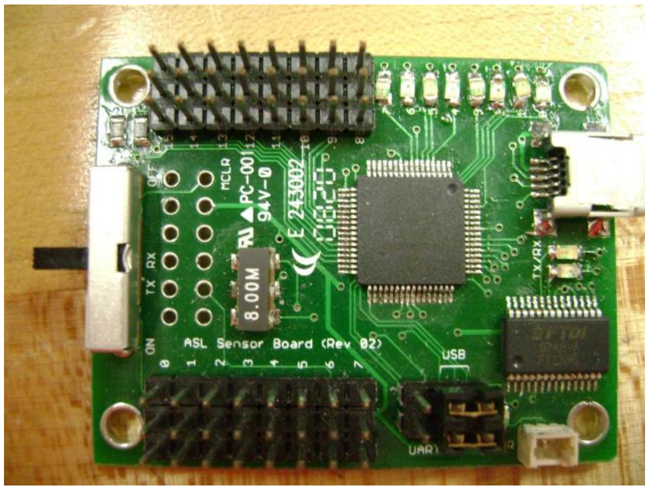


Figure 2-14: Data Acquisition Board

The device observes a change of state of the pins from each wheel encoder, and decodes the state information returned in channels A and B by the *state transition table* (similar to a *truth table*) in Table 2-2. The code required to interpret this information is written in C as a lookup table, appending the beginning state to the final state as a hybrid

4-bit word. This interprets a change from one state to another as a clockwise, counterclockwise, no change, or error event (as indicated as CW, cCW, NC, Err; respectively in the table). The message sent from the data acquisition device is the sum of these events per time step, such that the net effect is total clockwise rotations per time-step.

Previous	**Current**			
(A; B)	0, 0	0, 1	1, 0	1, 1
0, 0	NC	CW	cCW	Err
0, 1	cCW	NC	Err	CW
1, 0	CW	Err	NC	cCW
1, 1	Err	cCW	CW	NC

Table 2-2: Quadrature Encoder State Transition Table [11]

Chapter 3: Theory and Application

3.1. Steering Dynamics

The vehicle has two control inputs: steering (or wheel angle) and motor torque. Of the two, the dynamics associated with steering control are more complex. The test vehicle used in this research is a modified remote-controlled Tamiya TXT-1 truck, with front and rear axle steering control (four-wheel steer). To simplify steering behavior and control the location about which the vehicle will rotate, the front and rear axles are steered equally but in opposite directions. The following discussion explains the importance of this configuration.

First, consider the most general case of four wheel independent steering, as shown in Figure 3-1. Under low speed conditions (negligible lateral acceleration), the vehicle will turn about the intersection point of the four vectors drawn perpendicular to the wheels. The intersection point is termed the instant center of rotation (ICR). For the more common front wheel steering case, the ICR lies along the axis of the rear axle, and the turn radius depends on the mean steering angle of the front wheels defined as the Ackerman angle, $\delta_{mean} = L/R$, where L is the wheelbase and R is the turn radius.

Four wheel steering, used in this research, improves steering capability by reducing the turn radius and improving yaw response. When the rear wheels turn in the opposite direction of the front wheels, the ICR is shifted towards the vehicle centerline, enabling the vehicle to turn tighter for a given steer angle. Hence, four-wheel steering effectively reduces the necessary Ackerman angle for a desired turn radius.

Parallel steering was utilized on the current research vehicle to simplify steering geometry. This means the wheels of each axle turn the same amount, producing tire *scrub* (sideslip) during a turn [12]. The case of four-wheel parallel steering is illustrated in Figure 3-2. Scrub results because the two wheel tracks (outside and inside of the turn) have distinct ICRs. The true ICR is between the individual track ICRs.

During higher speed conditions all four wheels will experience tire slip and the wheel velocity vectors will differ from their respective steering vectors. The ICR will now occur at the intersection of the vectors perpendicular to the slip vectors. Vehicle weight properties, suspension and tire spring stiffness, and acceleration levels will cause each wheel slip vector to be different, complicating estimation of the ICR for control.

Given these basic steering principles, four-wheel parallel steering with some additional constraints was utilized to simplify ICR estimation for control and improve steering performace (over basic front-wheel steering). The front and rear wheels were constrained to steer with equal angles but in opposite directions front to rear. Consequently, at low speeds the ICR always occurs at mid-wheelbase and at the midpoint of the two individual track ICRs. Even at higher speeds with limited slip present, the ICR will remain reasonably close to the estimate.

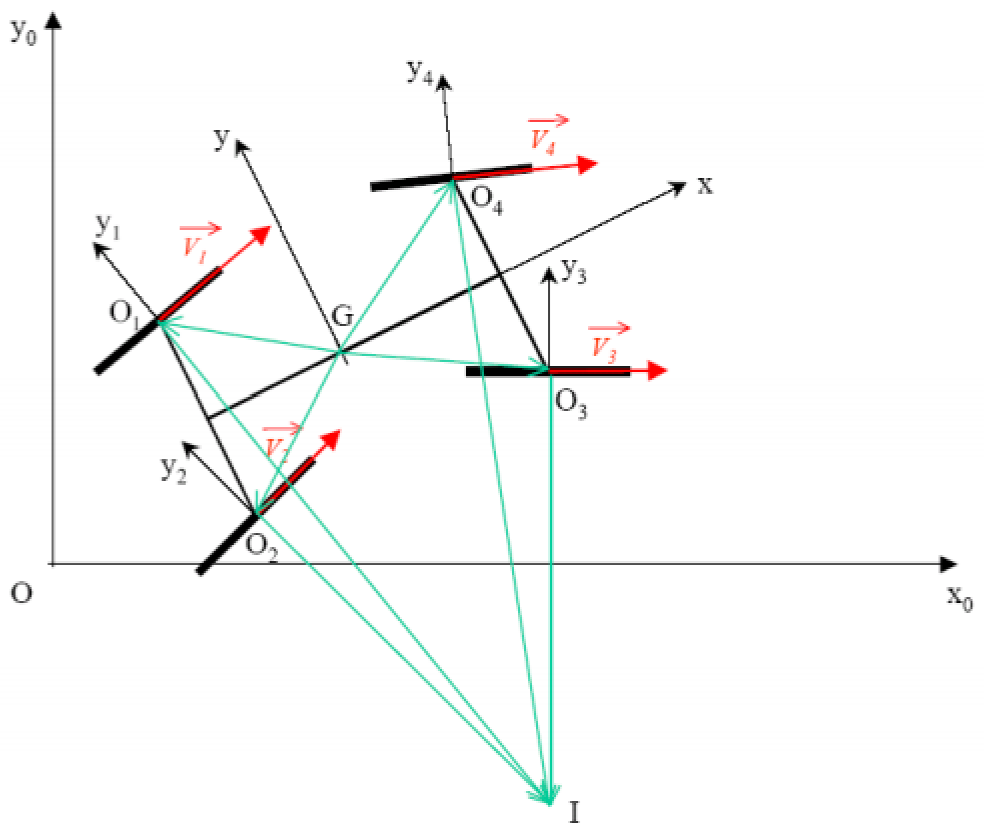


Figure 3-1: Instant Center of Rotation (ICR) Concept [13]

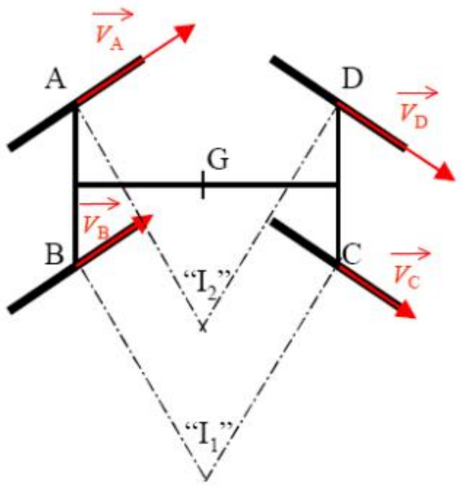


Figure 3-2: Four wheel Parallel Steer [13]

Lastly, four-wheel drive vehicles require three differentials due to differences in wheel track paths and velocities to minimize slip and reduce powertrain wear, A differential is placed between each wheel on an axle and a third placed between the two axles. With equal steering front to rear, the third differential can be eliminated, as is the case in the Tamiya TXT-1.

3.1.1. Wheel Odometry

Odometry uses the data from the movement of actuators to estimate change in position [14]. It is often applied to ground robotics for tracking and mapping purposes, as is the challenge of this thesis. Encoder-based odometry requires some knowledge of the vehicle platform. Specifically, how the motion of each of the wheels relates to motion of the vehicle. As described in the Steering Dynamics section above, it is not a completely straightforward relation with tire circumference as some might think. The tires travel paths of varying lengths and often slip. With analysis, the varying path lengths and some of the causes of slipping can be understood and used to glean even more information about the path of the vehicle.

Tracking of a vehicle on a plane requires its current position, in addition to some relative motion. A motion in X and a motion in Y (Cartesian vector displacement) could characterize that relative motion. The information garnered from our wheel speed sensors can be used to generate this. As discussed above, the paths tracked by the inside tires are different from those tracked by the outside tires. In fact, it is always the case that when in a turn, the outside tires will travel further than the inside. This fact can be used to uncover how the vehicle is rotating (turning). The distance traveled is just the average of the inside and outside tires. In principle, only two tires will be required to generate this information (one inside and one outside), but due to the errors induced by slip discussed above, all four tires are consulted in order to minimize the magnitude of the errors.

A vehicle having steered tires (when steering is limited to less than 90 degrees) cannot move in the X direction independently of its motion in the Y direction. Vehicles such as cars have a plane of symmetry, and are propelled in such a way that the guiding forces act along the plane of symmetry, with a very small component of the force used to change direction. It can therefore be assumed that there will always be a point on the vehicle such that the velocity of that point is always along the plane of symmetry (tangentially in our case). That point in our vehicle will be the halfway point between the front and rear tires [15]. Such vehicles are called non-holonomic, implying that the current position (state) will depend on the path taken to achieve it. The equations used to generate the relative motion in X and Y are given by:

$$dX = V * \cos \Theta$$

$$dY = V * \sin \Theta$$

Where theta is calculated assuming the inside and outside paths are piecewise circular. This results in a simple equation for theta shown below. At each time-step, $d\Theta$ and V are calculated by:

$$V = (V_{fr} + V_{fl} + V_{br} + V_{bl})/4$$

$$\mathrm{d}\Theta = ((V_{fl} + V_{bl}) - (V_{fr} + V_{br}))/(2 * T)$$

Where:

V – Velocity of the Vehicle

V_{fr} – Velocity of the front right wheel

V_{fl} – Velocity of the front left wheel

V_{br} – Velocity of the back right wheel

V_{bl} – Velocity of the back left wheel

T – Track-width of the vehicle

In reality, as discussed in the Steering Dynamics section, the TXT-1 will not be able to steer without some slip of the tires and the Instant Center may move further or closer to the vehicle along the normal to its axis, changing the result for Θ. This is one source of error for our estimate.

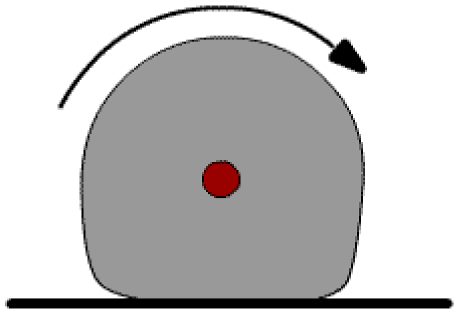


Figure 3-3: Tire deformation [16]

In addition to tire slip, the tire may also deform with load in the road-normal direction as shown in Figure 3-3. The change in effective diameter has severe implications in a single tire's ability to return accurate speed information. Effects on vehicle velocity are not as profound as the effects on Θ due to the averaging over all four tires. Θ is inherently more sensitive to this effect. This does not mean that our estimate of the equation for theta is entirely unusable; rather, that knowledge of the effective *Instant Center* is not without error. It is for these reasons that tracking by wheel odometry alone in the case of the vehicle under research is unreliable. A schema to improve upon this will be discussed in the Kalman Filter Sensor fusion section.

3.2. System Identification

Physical systems in the broadest sense are an interconnection of components, devices, or subsystems. A *system* can be viewed as a process in which input signals are transformed by the system or cause the system to respond in some way resulting in other signals as outputs [17]. *Models* are mathematical interpretations of the evolution of the state of the system by external inputs, along with the outputs generated by it. The model may depend only in present inputs, but generally may depend on past inputs and outputs and even future inputs. A system that *anticipates* future inputs is called *non-causal.* For the purposes of real-time tracking (or any type of real-time filtering for that matter), *causality* is required.

The system in the context of this thesis is the Tamiya TXT-1 truck robot, having as its inputs (u) the steering angle request and motor torque. The outputs for tracking can be phrased in several ways. The output of the system model could be position expressed directly in X and Y coordinates. This is a perfectly acceptable model. It is non-linear; however, and will complicate the estimation problem. Instead, to mirror and complement the information returned from the wheel speed sensors (for the purpose of sensor fusion as described in the Kalman Filter section), the outputs (y) were selected to be $d\Theta$ (differential change in heading) and V (the vehicle velocity). Tracking information is calculated from this data as total heading, it is the cumulative sum of $d\Theta$. X and Y are calculated as shown in the Wheel Odometry section.

The process by which a model is created (or perhaps “discovered,” depending on the philosophical view) is called *System Identification.* A model may be developed purely by theory, as many are; however, not all the parameters of the system may be known. Often errors arise due to inaccuracy in measuring the parameters, or factors arising in the physical system that were not anticipated while deriving the model. Frequently the structure of the model is known, but the parameters may not be known or may be difficult to measure.

The approach taken herein focuses on the experimental discovery of the model by recording inputs and outputs and fitting a model to them. When it is possible and convenient to write down a differential or difference equation to model the system, but difficult to discover parameters, a model based on the differential or difference equation may be used as a starting point for experimental model discovery. Such Models are called *grey-box* models. In the case where such an equation is not used as a starting point and the model is created by data alone, the model is called a *black-box* model. Figure 3-4: depicts such a black box model, where inputs are transformed to outputs by some unknown structure.

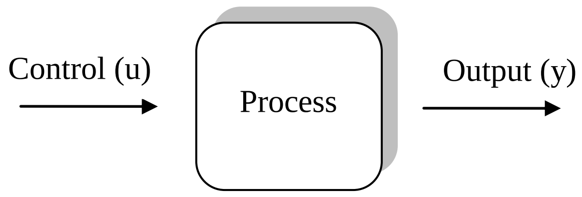


Figure 3-4: Black Box Model

The Matlab software package provides tools for the identification of systems (the *process* portion of Figure 3-4). The tool used in this thesis is the *Identification Toolbox.* This toolbox contains methods for normalizing and filtering input and output data, along with the *estimation* tools for constructing models as well as means to evaluate the models created. The process of system identification requires that you:

1) Measure the input and output signals from your system in time or frequency domain
2) Select a model structure
3) Apply an estimation method to estimate the values for the adjustable parameters allowed by the candidate model structure
4) Evaluate the estimated model to see if the model is adequate in describing the dynamics of the system sufficiently

[18]

This process may be (and in all probability *will* be) executed numerous times, where steps 1, 3 and 4 may be repeated with different data in order to improve the model. Considering this model will be implemented on a digital computer gathering data in a time-quantized manner, the nature of the model will be that of a *discrete time dynamic model*.

Choice of the input data is of paramount importance; clearly, if the steering was left at zero and the vehicle was driven in a straight line at a constant speed, the model would be useless in determining any dynamic behavior such as turning and accelerating. For this reason, input-output data was generated by driving the vehicle in such a way as to capture sufficient dynamic behavior to develop an appropriate model. Output data (y) is both the wheel encoder and gyro data. Ideally, measurement of absolute truth should be performed using some external reference, since the noise present in sensor data can further complicate the system identification. Since no practical means of performing this are available, the gyro is used as a ground truth sensor for $d\Theta$ for the purposes of model identification. The encoder-returned V is used as ground-truth for velocity.

Sinusoidal inputs sweeping frequency and magnitude are generally adequate in sufficiently exciting the vehicle dynamics. Roughly sinusoidal inputs are generated manually utilizing the XBOX controller, supplying both independent excitation to motor and steering, and coupled excitation (accelerating and decelerating while turning). Justifying this process is the success of the model in accurately representing outputs generated by inputs different from those supplied in the process of System Identification (generally a-periodic). This evaluation process is discussed in more detail in the Evaluating Models section.

3.2.1. Selecting a Model Structure

The Matlab identification toolbox provides tools for estimating models of numerous types. These include:

1) Linear Parametric Models
2) Process Models
3) Nonlinear Models
4) Spectral Models
5) Correlation Models

The process of selecting a model structure and order may be very simple if the model structure is already known as in the case of grey-box modeling. In the case of black-box modeling; however, it is not so clear. The optimal model is the simplest one that is able to recreate your outputs adequately. Having some knowledge about the system may assist in selecting this.

When an input is sent to the platform ([Motor Value, Steering Value]), the system evolves by altering its velocity and relative change in heading ($[V, d\Theta]$). The change in velocity is created by the motor value requesting a torque of the TXT-1 motor and the motor responding by accelerating the platform. In a simple motor model, integration is needed to capture the resulting velocity as a function of the motor input, so it is clear that past information will be required. Steering is usually found by direct measurement of the steering position. Our Tamiya TXT-1 platform utilizes hobby servos for steering, which do not report such information. We send a request to the servo to move to a position, and since these servos do not report their current position in time; some integration must be performed for steering as well since the servo takes some time to respond. Because of these facts, a history of the *inputs* will be required. In addition, due to the steering responding differently at different speeds and vertical loads, a history of *outputs* will likely be required for the change in heading. These facts, along with the linear mathematical structure of the Kalman Filter lead us to choose a Linear Parametric Model for evaluation since we may control the depth of memory for both the inputs and outputs. The depth of memory will depend purely on evaluation; that which produces the best model with the least order is the best solution.

The Identification Toolbox provides several structures within the Linear Parametric Models type. ARMAX and Box-Jenkins models attempt to explicitly model the noise [18]. Since the implementation of our model is in the context of the Kalman filter, where the noise is assumed Gaussian, there is no interest in introducing the added complexity of either of these models. Output Error (OE) models only support single output systems, so it is also precluded from use because we are modeling Velocity and change in relative heading (two outputs). This leaves State Space and ARX (autoregressive). Autoregressive models can be transformed into a state space model as will be investigated in the implementation of the Kalman Filter, so for exercising more control over the choice of state variables, it is decided to work with ARX models for the present development.

The form of the ARX model for MIMO systems is as follows:

$$A_0 y_k + A_1 y_{k-1} + \cdots + A_n y_{k-n} = B_0 u_k + B_1 u_{k-1} + \ldots + B_m u_{k-m} + e(t)$$

Equation 3-1

Where A_i represent square matrices with order equal to the number of outputs and B_i represent matrices of order $j * k$. In the case of the tracking problem presented $j = k = 2$ (two inputs and two outputs, respectively). The parameter A_0 is the identity matrix by convention for the purpose of implementation.

3.2.2. Making a MIMO ARX Model

The toolbox can be used at the MATLAB command prompt, or run using the GUI provided. Evaluation tools within the GUI allow rapid development and comparison of models, so it is used exclusively, rather than employing the command line interface.

To open the toolbox, enter:

```
>>ident
```

at the Matlab command prompt. This will open the GUI shown in Figure 3-5.

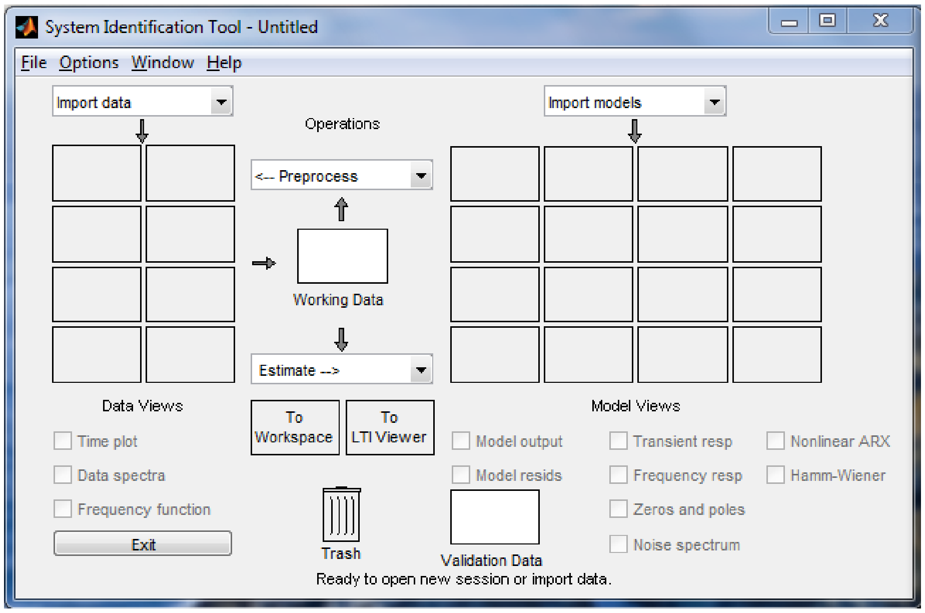


Figure 3-5: Identification Toolbox Startup

Assuming the input and output data from the given experiment are available in the MATLAB workspace, importing them into the toolbox is very straightforward. This is done by clicking the *Import Data* drop-down menu and selecting *Time domain Data*. The input and output data relevant to this discussion is a MIMO system arranged as follows:

Inputs:

$$u = [Motor; Steering]$$

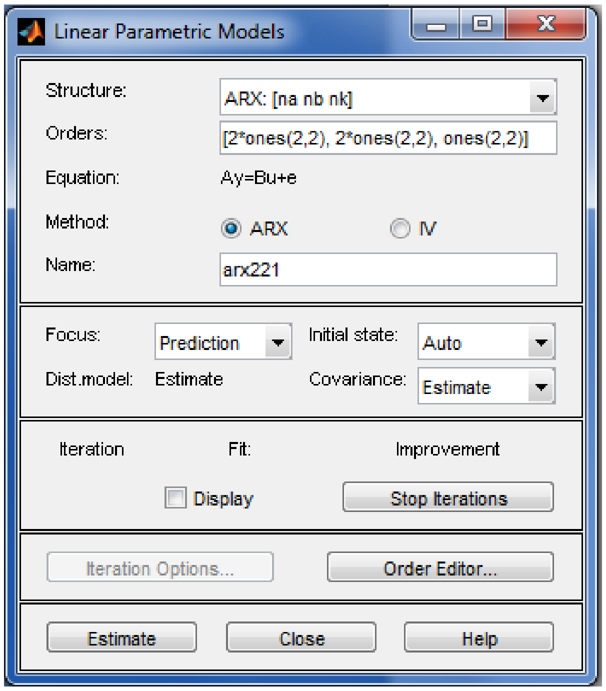
Outputs:

$$y = [V;\ d\Theta]$$

as introduced in the Robot overview section.

Once this data is entered into the Import Data GUI, it is available in one of the left frames as the name given in the data import GUI. This data is shown in bold if it is currently selected. To view the data in a time-plot, check the *time-plot* checkbox at the bottom left of the System Identification Tool GUI. If this is the first data entered into the toolbox, it will appear in the *Working Data* and *Validation Data* panes in the middle, and bottom of the GUI, respectively.

The data shown in the *Working Data* pane is that which will be used to perform the system identification. Changing the selected data set requires dragging the desired data set onto this pane. Click the *Estimate* drop down menu and select *Linear Parametric Models* (see Figure 3-6).



Structure: ARX: [na nb nk]

Orders: [2*ones(2,2), 2*ones(2,2), ones(2,2)]

Equation: Ay=Bu+e

Method: [✓ ARX] IV

Name: arx221

Focus: Prediction

Initial state: Auto

Dist.model: Estimate

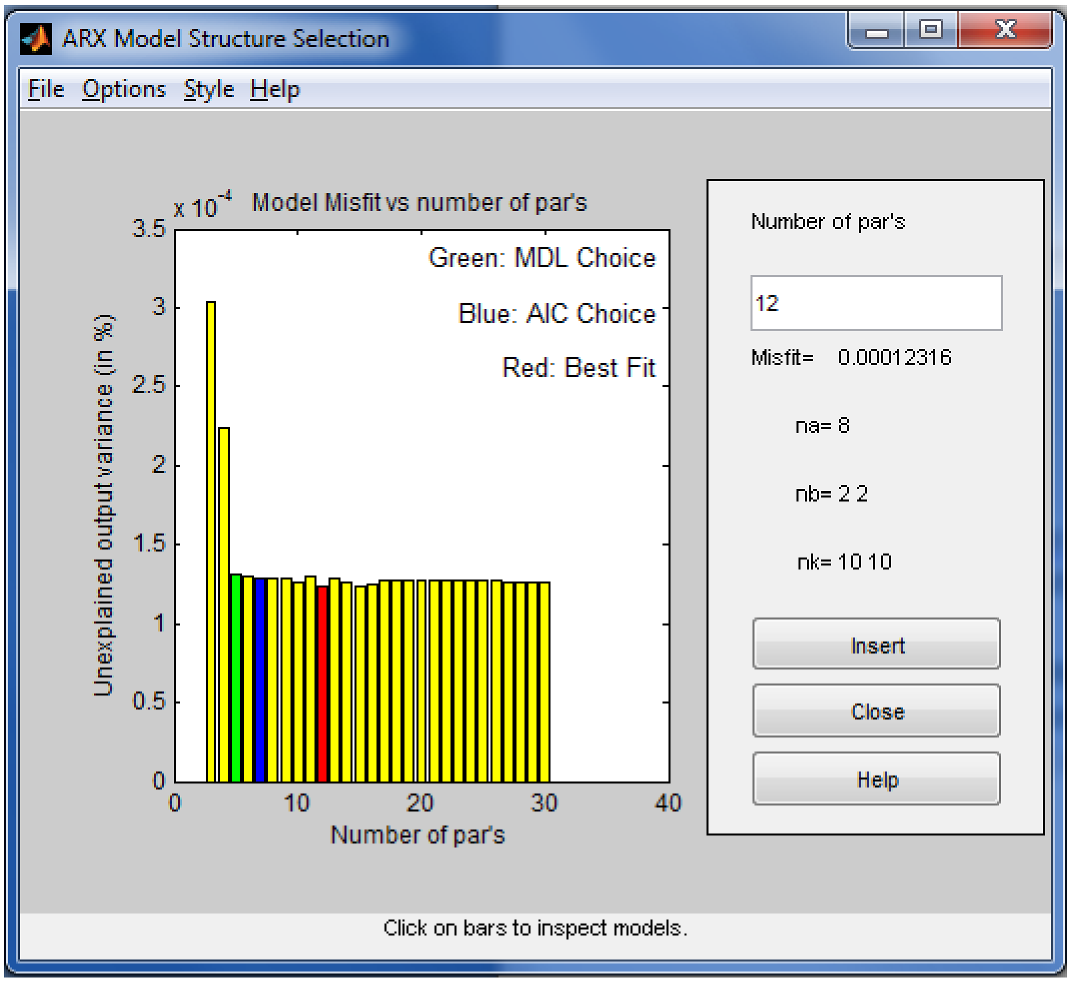
Covariance: Estimate

Figure 3-6: Linear Parametric Models Window

Here the order of the model can be chosen either directly in the *Orders* text field, or within the order editor. The order shown is [2 2 1], representing the history of inputs, outputs, and the delay in input-output response, respectively. The first number corresponds to n in Equation 3-1. The second plus the third number is m in the same equation. The last number (the delay) is the number of timesteps of the input ignored for generating the current output. For example, if the system took three time steps to respond to an input (delay = 2) and m was five, B_0 through B_2 would be zero, and B_3 through B_5 would be non-zero model parameters. In general, the orders may be specifically defined per input and output channel by defining matrices other than the matrix of *ones* as shown in the *Orders* field. In this thesis, uniform depth over inputs and outputs is used.

For single output systems, a convenient method exists for evaluating multiple orders to discover a suitable order for the system. By entering an order parameter as a vector such as $[p\colon q]$ and clicking *Estimate*, the toolbox brings up the Figure 3-7: ARX Model Structure Selection Window, showing *unexplained output variance* for models beginning with order parameter p and ending with q. This window can assist in selecting model order. For multiple output systems, it is useful to apply this test to each output individually in order to gain some insight in discovering the complete model order. If the system is treated as a single output system for each of its outputs and this test shows similar results for each, then the model order for the MIMO system is likely similar.

Once the evaluation has been performed and a model has been deemed unsatisfactory by the methods shown below, a model may be improved or discarded and re-attempted. If the order of the model is not deemed the cause of the error, more data should be used to improve the model. This is done by performing another iteration of model estimation by selecting *by an initial model* in the model structure drop down menu in the Linear Parametric Models Window.



12

Misfit= 0.00012316

na= 8

nb= 2 2

nk= 10 10

Click on bars to inspect models.

Figure 3-7: ARX Model Structure Selection Window

At this point, MIMO model estimation (system identification) is ready to be performed more explicitly using an appropriate model order. Clicking *Estimate* in the Figure 3-6: Linear Parametric Models Window will create a model shown as an image on the right side of the Figure 3-5: Identification Toolbox Startup window. Checking the boxes below the model will display the model outputs, residuals (errors with auto and cross-correlation views), model transient response, and others. Model outputs and model residuals are used in evaluating the model's performance below.

3.2.3. Evaluating Models

Multiple models of the system were developed in this research, and three are presented below to assist in understanding the process. It is important to evaluate a model with different data than was used to generate it. Once preliminary model estimation has been performed, another data set should be used to evaluate the model. This can be done by dragging and dropping a different data set onto the *Validation data* pane in the Identification Toolbox Startup menu.

Two windows are critical to judging a model's performance: the *model output*, and *model resids* (residuals). Checking the Model Output box brings up the window shown in Figure 3-8 (shown with a few of the models created for evaluation in this thesis).

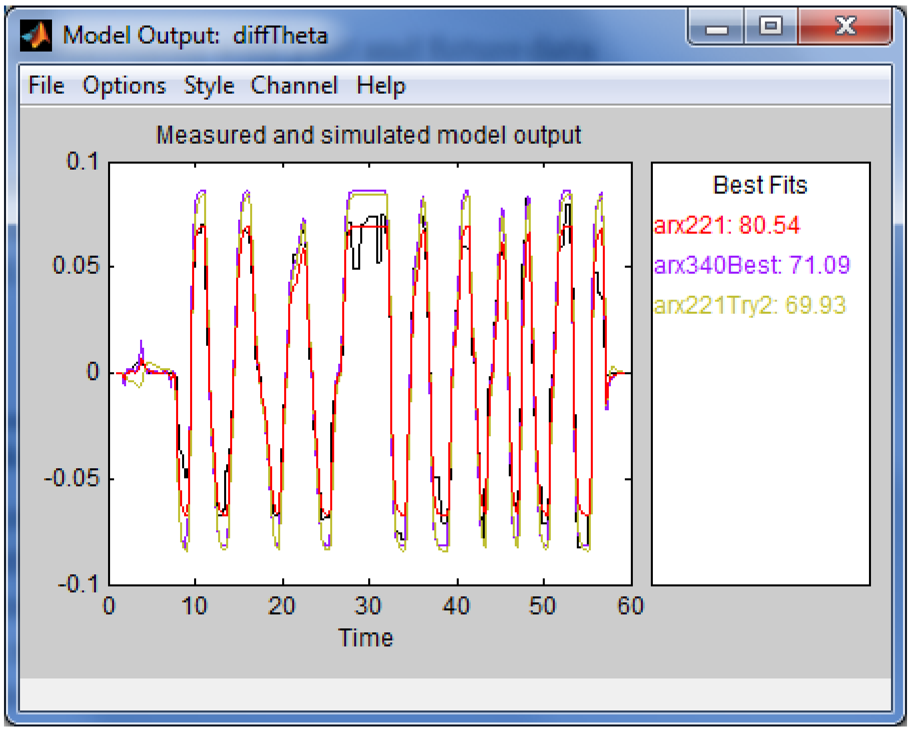


Figure 3-8: Model Output Window showing the model output for $d\Theta$ from three models.

The model output window shows the *fit percentage* to the right for each model selected. The models are named to reflect their order as discussed above. Performance for this data is relatively straightforward to judge. Clearly, the red model (arx221) fits the ground-truth sensor for $d\Theta$ (from the gyro) the best, while not modeling the noise, yet another model with the same order (arx221Try2) overshoots the data. A model's performance may improve or worsen with the addition of data to the estimation process. Generally, improvement comes by using data that exhibits additional excitation of the dynamics.

The second window relevant to judging performance is the Model output residuals window. If the model accurately represents the dynamics system, the error will not be correlated with the inputs. Noise will inevitably be present, but true noise in the output will not be highly correlated with ordered input data. Said in another way, if the system did nothing, and absolutely no output was generated (clearly the extreme case of poorly modeling the dynamics), the error would be the output of the system itself which should be highly correlated with the input data. These facts imply the cross correlation for input and output residuals should be minimal for a model to be accurate.

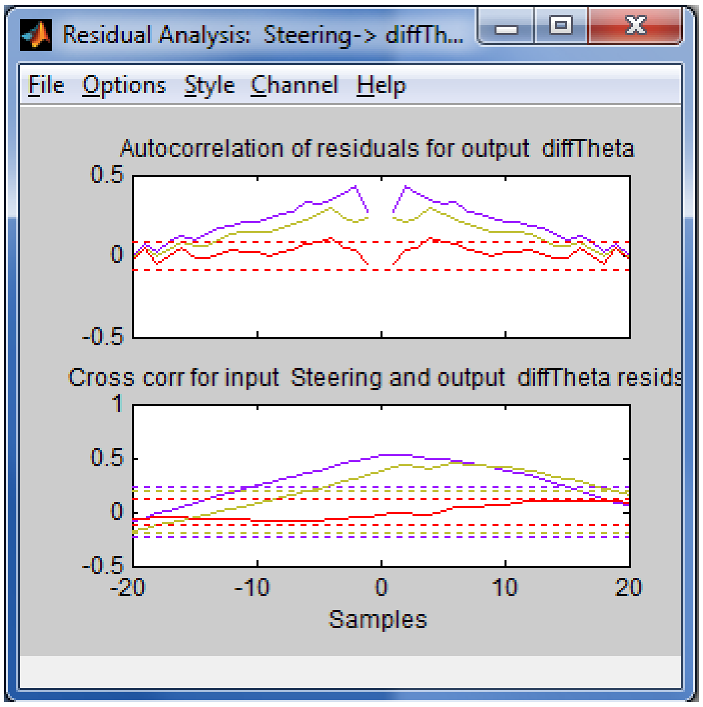


Figure 3-9: Model output residuals for channel $d\Theta$

The fit percentage is clearly the best for the red model (ARX221), as it reasonably captures the character of the ground truth sensor. This process has only been discussed for one output: $d\Theta$. Examining the same information for the other channel (V), together with the discussion above yields a decision for the best model. Images of the residuals and model output for *V* are shown in Figure 3-10, and Figure 3-11.

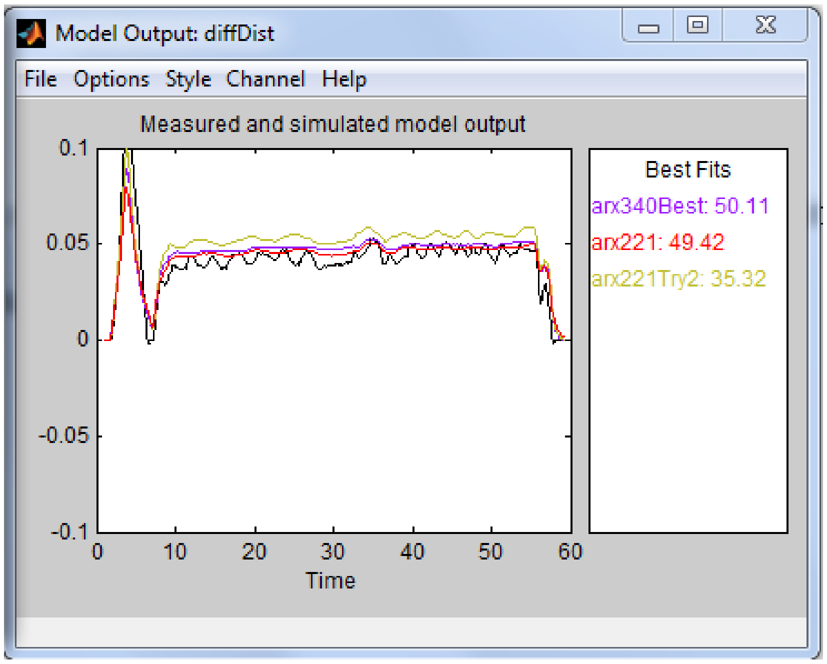


Figure 3-10: Model output for channel *V*

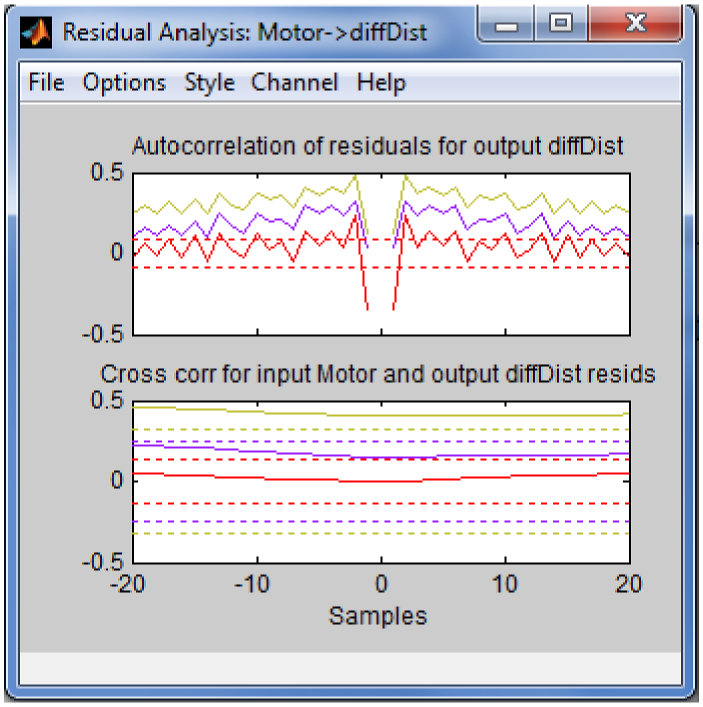


Figure 3-11: Model output residuals for channel V.

This is the result of system identification after performing multiple iterations on numerous data sets. These results show that the models performance is better at modeling the behavior for $d\Theta$ than for V. Considering the discussion in the Wheel Odometry section regarding the expected accuracy of the measurements, this is acceptable, as the wheel encoders return much better quality data for V than for $d\Theta$. Further discussion regarding this point will be had in the Kalman Filter chapter.

The model producing the best results with the lowest order was ARX221. Parameters for the model are shown here:

$$A_1 = \begin{matrix} -1.5391 & .0062 \\ .4391 & -.7789 \end{matrix}$$

$$A_2 = \begin{matrix} .5869 & -.0053 \\ -.4258 & .0032 \end{matrix}$$

$$B_0 = \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$$

$$B_1 = \begin{matrix} -.0090 & .0001 \\ .0030 & -.0047 \end{matrix}$$

$$B_2 = \begin{matrix} -.0002 & -.0001 \\ -.0057 & -.0142 \end{matrix}$$

3.3. Kalman Filter

A mathematical model of a physical system is never perfect in capturing every nuance of its character. A linear system can theoretically be modeled using only knowledge of inputs and outputs; however, real systems are rarely truly linear, and often have many un-modeled nonlinearities. These nonlinearities can be slip and deformation of the tires, and can be affected by un-modeled inputs or disturbances. Due to these nonlinearities, depending purely on the inputs of a system to determine its behavior is rarely a good practice. Even when the model is presumed to be perfect, the system may be buffeted by some noise in the process or the measurements not covered in the model. The Kalman filter solves this problem, in the sense that it allows for inaccuracies of the model as long as the sensor data is of sufficient quality to correct it.

The Kalman Filter is a state *Estimator*, meaning, it estimates the true state of a system based on measurements and inputs. The filter produces estimates of the true parameter values based on measurements and their associated calculated values by predicting a value by employing a model, estimating the uncertainty of the predicted value, and computing a weighted average of the predicted value and the measured value [19]. A diagram of the Kalman filter (KF) application is shown in Figure 3-12. The inputs to the filter are the observations from sensor data, along with the control inputs to the process. Internal to the filter is the vehicle model.

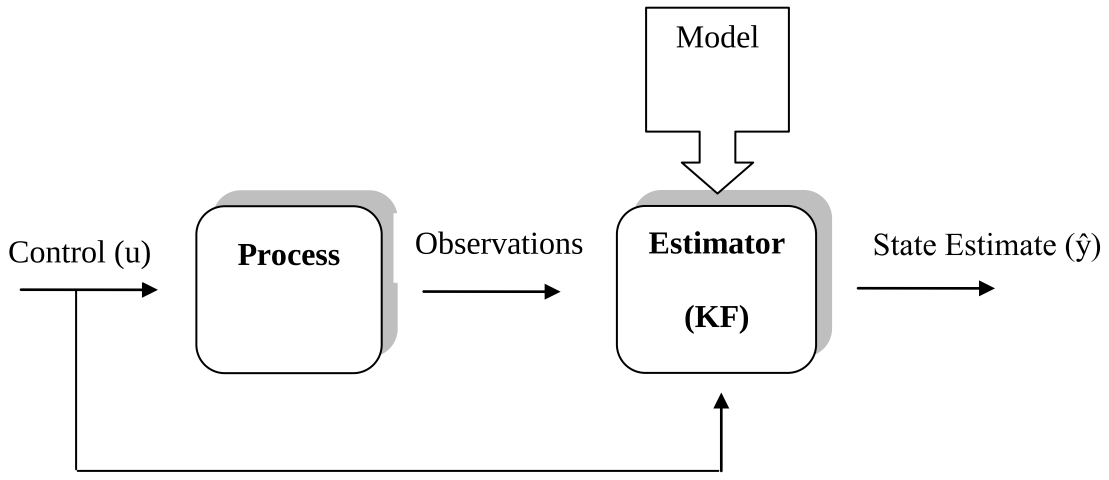


Figure 3-12: Process-Estimator

3.3.1. Background and Governing Equations

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to *estimate* the actual state of a process, in a way that minimizes the mean of the squared error. The necessity of the filter to estimate the actual state is often dictated by the inability to measure the state directly with some degree of accuracy. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown [20]. It does require some knowledge of the statistical nature of the data in the system.

In principle, the Kalman filter may have applications both to discrete-time systems as well as continuous-time systems (where such a filter is named the “Kalman-Bucy Filter [19]), but the continuous domain is not to be discussed herein. Our system is treated in discrete-time due to the nature of the digital computing methods used, and the arrival of sensor data in a time-quantized manner.

The purpose of the Kalman filter is to estimate the state $x \in \mathbb{R}^{\mathrm{n}}$ of a discrete-time process that is perturbed by some noise. The nature of that noise is crucial in the definition of the filter, and in its performance. All sensor and process noise is presumed to be zero-mean-Gaussian $N(0, \sigma)$. The system being filtered has inherent system properties described by a model. The evolution of the state is governed by the following discrete time stochastic difference equation:

$$x_k = \mathrm{A}_{\mathrm{k}}\, x_{k-1} + B_k\, u_{k-1} + w_{k-1}$$

Sensor data is used to correct this state, but measurement of this process may not necessarily be performed in the same space as the state itself. It is also assumed to have had some noise introduced in the form of sensor noise. We define the measurement or otherwise called the *observation* of $z_k \in \mathbb{R}^n$ as:

$$z_k = H_k\, x_k + v_k$$

Where:

x_k – State vector at time-step k. Dimension $(n \times 1)$

u_k – Control input vector at time-step k. Dimension m

z_k— Measurement. Dimension $(p \times 1)$

A_k – Discrete time transition matrix that relates x_{k-1} to x_k . Dimension $(n \times n)$

B_k – Control input to state vector matrix. Dimension $(n \times m)$

H_k—Relates the measurement z to the state x. Dimension $(p \times n)$

w_k— Process Gaussian White Noise (GWN). Probability distribution is Normal:

$P(w_k) \sim N(0, Q(k))$

v_k— Measurement GWN. $P(v) \sim N\big(0, R(k)\big)$

$Q(k)$ – Process Noise Covariance (PNC) matrix. Varying with time-step in principle

$R(k)$— Measurement Noise Covariance (MNC) matrix. Similar to PNC

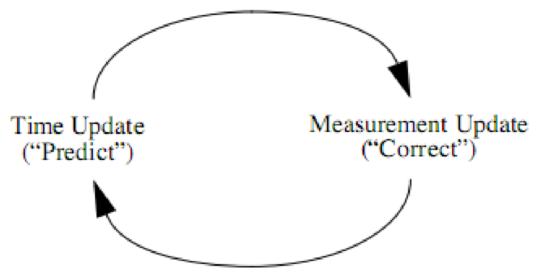


Figure 3-13: The ongoing Kalman filter cycle [20]

For the purpose of implementation, a "predictor-corrector" form is employed, where the Kalman filter is divided into two distinct stages: the *Prediction* stage and the *Update* stage (as shown in Figure 3-13). The *time update* projects the current state estimate in time. The m*easurement update* adjusts the projected update by an actual measurement at that time [20]. The prediction stage is where the dynamics of the system come into play, by predicting what the system *should* be seeing. This is followed by updating the process with sensor data (when it is available). The ability of these two stages to be partitioned forms an important implication. If sensor data is not available for a stretch of time, or if two or more sensors are operating at different frequencies, the filter may continue to operate, giving an estimate of the true state.

The prediction stage consists of the dynamics themselves, along with the incorporation of another matrix that represents the degree to which the state can be trusted. Every time the system *predicts* what the state should be, it also predicts how accurate the state estimation is. On a prediction, the state covariance matrix increases, while every time a sensor updates the state, it is reduced.

The filtering equations themselves are presented below with annotation as to their significance:

Prediction Stage:

Predicted State	$\hat{x}_{k\|k-1} = F_k \hat{x}_{k-1\|k-1} + B_k U_k$
Predicted estimate covariance	$P_{k\|k-1} = F_k P_{k-1\|k-1} {F_k}^T + Q_k$

Update Stage

Innovation (error)	$\tilde{y}_k = z_k - H_k \hat{x}_{k\|k-1}$
Innovation Covariance	$S_k = H_k P_{k\|k-1} {H_k}^T + R_k$
Kalman Gain (weighting factor)	$K_k = P_{k\|k-1} {H_k}^T {S_k}^{-1}$
Updated state estimate	$\hat{x}_{k\|k} = \hat{x}_{k\|k-1} + K_k \tilde{y}_k$
Updated estimate covariance	$P_{k\|k} = (I - K_k H_k) P_{k\|k-1}$

[19]

While the derivations themselves will not be presented in this thesis, the implementation of these equations is very straightforward in MATLAB.

Certain initializations are required for the state vector and matrices defined above. If the filter began operating with the wrong assumptions, many time-steps may occur before the filter starts reflecting what the internal state truly is, due to the internal dynamics. Since the tracking example presented assumes the system is initialized while the vehicle is standing still, the state vector is initialized to zero.

In the application of the Kalman filter to odometry, using just one sensor such as the wheel speeds, the observation matrix is as follows:

$$H_k = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

This means that we are observing the current $d\Theta$ and V since the state vector is:

$$X_k = \begin{matrix} V_k \\ d\Theta_k \\ V_{k-1} \\ d\Theta_{k-1} \end{matrix}$$

The innovation portion of the filter is:

$$\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1}$$

This is written more explicitly as:

$$\tilde{y}_k = \begin{matrix} V_{k_{Observed}} \\ d\Theta_{k_{Observed}} \end{matrix} - \begin{matrix} V_k \\ d\Theta_k \end{matrix}$$

The measurement covariance matrix (MNC) is generated by *a-priori* measurements of statistical information based on sensor characteristics. Variance of the sensor data was defined in the Sensors and their characteristics section. The wheel speed sensor data then underwent some functional transformation as defined in the Wheel Odometry section. Deriving the observation variances according to the properties of the Gaussian distribution:

$$V_{k_{var}} = \frac{1}{4}\sigma^2{}_{wheel}$$

$$d\Theta_{k_{var}} = \frac{1}{T^2}\sigma^2{}_{wheel}$$

The MNC is defined as

$$R_k = \begin{matrix} V_{k_{var}} & 0 \\ 0 & d\Theta_{k_{var}} \end{matrix}$$

The PNC matrix is defined somewhat by experimental tuning. By trusting the internal process more, the model is used more in determining the output of the filter. The matrix that seems to give the best results for tracking is:

$$Q_k = \begin{matrix} 10^{-4} & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 10^{-2} & 0 \\ 0 & 0 & 0 & 10^{-7} \end{matrix}$$

3.3.2. Incorporating the dynamical model into the filter

The equations formulated above utilize vectors for state variables (both for the state vector X_k and input vector U_k), and matrices to implement the model. The ability of MATLAB to work so seamlessly with matrices and vectors lends itself to the application of the Kalman filter. In addition, the tools present in MATLAB for the process of system identification allow effortless incorporation of the model into the code written for the Kalman filter. For these reasons, MATLAB is used to implement the filter, and will be integrated into the algorithms written in C# on the host computer.

As stated in the System Identification section, the ARX model can be transformed to the classical *state space* implementation:

$$x_{k+1} = F_k X_k + B_k U_k$$

By careful choice of states based on the model order, we can use this model. This transformation requires first, rearranging the equation as shown:

$$A_0 y_k = B_0 u_k + B_1 u_{k-1} + \dots + B_m u_{k-m} - (A_1 y_{k-1} + \cdots + A_n y_{k-n}) + e(t)$$

This transformation is followed by constructing a state vector and input vector with the dimensions satisfying the orders shown above. This state vector must not only dimensionally make sense, but also contain the necessary history of outputs to the order n. The input vector u is defined similarly but to satisfy order m, of course.

$$X_k{}^{nx1} = \begin{matrix} y_{k-1} \\ y_{k-2} \\ \vdots \\ y_{k-n} \end{matrix}$$

$$U_k{}^{mx1} = \begin{matrix} u_k \\ u_{k-1} \\ \vdots \\ u_{k-m} \end{matrix}$$

The *order* of the model is defined in terms of n and m shown in the equation above. These represent the depth of history of inputs and generated outputs required to satisfy the model's design.

Finally, F_k and B_k must be appropriately constructed to include all $A_i{}'s$ and $B_i{}'s$ defined in the model.

$$F_k{}^{nxn} = \begin{matrix} -A_1 & -A_2 & \cdots & -A_n \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{matrix}$$

$$B_k{}^{nxm} = \begin{matrix} B_1 & \cdots & B_m \\ 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{matrix}$$

The matrix F_k ensures that at every iteration, the current output will be moved one time-step into the past. Likewise, every previous output will be shifted back one time-step by the appearance of the shifted identity matrix. B_k is used to incorporate the input to the system, and appends zeros below the first line to remain consistent with the number of states introduced to the system. Clearly, since the input does not affect previous states, this dimensionally makes sense.

3.3.3. Sensor fusion

Often a single sensor is capable of resolving every detail of interest in a dynamical system, but not always. Sensors often suffer from noise, and it is common for one sensor to have less noise in certain regions of use than another. Additionally, some sensors are more capable of resolving certain regions or aspects of the dynamics of the system as in the case of GPS-Inertial Navigation (INU) integration. Table 3-1illustrates the advantages of the two sensors in different regions of use.

	Advantages	Disadvantages
INU	a. High Data rate b. Provides both translational and rotational data	1) Unbounded errors 2) Knowledge of gravity is required
GPS	Errors are bounded	1) Low data rate 2) No attitude information

Table 3-1: Comparison of INU and Satellite sensor features [21]

This is cause for the field of Sensor Fusion. This may be as simple as ignoring one sensor when it is in its mode of poorest operation and instead, trusting another. There are more advanced techniques. Should the co*variance* of one sensor increase, while another sensor's covariance decreases, this would be a reason to weight the sensor with the lowest covariance more. The Kalman filter provides a convenient methodology for doing so, by means of the *innovation* portion of the *update* stage of the filter. Very few modifications are needed to perform sensor fusion with the Kalman filter. Only the matrices H_k and R_k, along with the vector of observations z_k must be changed.

The nature of integration between two sensors is spoken of in terms of *coupling.* There are uncoupled, loosely coupled, tightly coupled, and deep integration. The details of these are beyond the scope of this discussion, but the integration of the system presented is that of a *loosely coupled* integration.

Though the filter is operating on additional sensor data, the structure of the process remains the same. Figure 3-12: Process-Estimator still represents the filter's use, only the number of observations made changes (along with some internal filter matrices as discussed below).

The matrix H_k was originally chosen to transform the state into the space in which the observations were made. When more than one observation is made for the same state vector element, H_k accommodates by simply providing that same vector element to the *innovation* stage of the filter [22]. Given that the observation is:

$$z_k = \begin{matrix} ws\_V \\ ws\_d\Theta \\ gyro\_d\Theta \end{matrix}$$

The new H_k becomes:

$$H_k = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

Making the innovation equation:

$$\tilde{y}_k = \begin{matrix} ws\_V_{k_{Observed}} \\ ws\_d\Theta_{k_{Observed}} \\ gyro\_d\Theta_{k_{observed}} \end{matrix} - \begin{matrix} V_k \\ d\Theta_k \\ d\Theta_k \end{matrix}$$

The new observation covariance matrix increases dimensions and incorporates the variance seen by the added sensor.

$$R_k = \begin{matrix} ws\_V\_Cov & 0 & 0 \\ 0 & ws\_d\Theta\_Cov & 0 \\ 0 & 0 & gyro\_d\Theta\_Cov \end{matrix}$$

3.4. Software

3.4.1. ASL Framework

The need to integrate the data from many different types of sensors is good cause for writing as high a level code as possible. It is for this reason that the .NET environment is chosen as the primary coding environment. Additionally, the varied plugins available for .NET further justifies its use.

The ASL Software Framework provides a hierarchical structure for application development. Every application that utilizes the ASL Framework consists of one or more algorithms as well as a platform. The platform specification consists of all sensor, actuator, and communication devices currently deployed on the testing platform. This specification could include one or more webcams, a servo controller, and/or the peer-to-peer communication network manager. Algorithms, on the other hand, consist of multiple interconnected processing components defined as functional units [3]. A *functional unit* is used to implement the Kalman filter itself by calling the relevant MATLAB subroutines, while other functional units handle the data gathering and graphical mapping routines. The diagram in Figure 3-14 shows the overall structure of the ASL Framework.

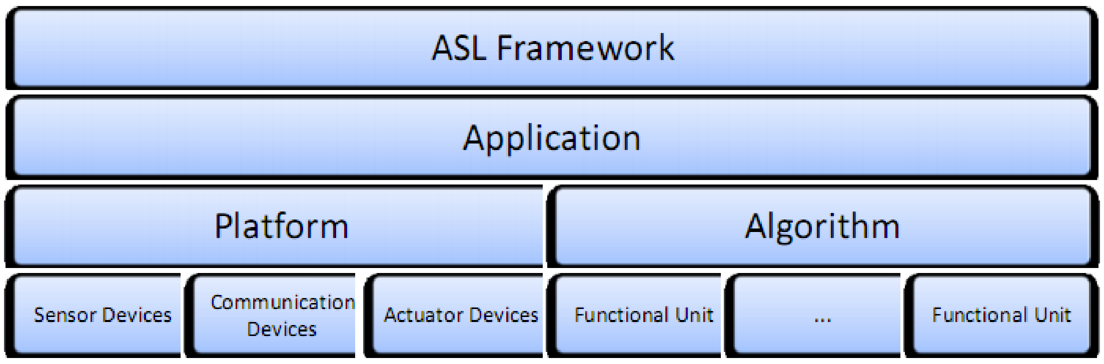


Figure 3-14: ASL Software Framework Hierarchy [3]

The framework is designed such that the algorithms written may be somewhat platform agnostic, but in this case knowledge of the platform model as well as those of the sensors is needed in the Kalman filter; however, the same procedure for system identification could be used on another vehicle and applied identically to the process discussed herein.

3.4.2. MATLAB .NET deployment

The Kalman filter discussed is very straightforward to implement in MATLAB, but not nearly so in C#. Fortunately, MATLAB provides the means for C# to call functions written natively in MATLAB directly. This is achieved through .NET deployment through the MATLAB Builder NE package.

MATLAB Builder NE creates MATLAB based .NET components for royalty-free deployment on desktop machines or Web servers. As a result, you can integrate your MATLAB applications into your organization's .NET programs. The builder creates the components by encrypting MATLAB functions and generating a .NET wrapper around them [23]. The process behind creating the .NET .dll is entirely GUI-based in the MATLAB and very straightforward. Once the functions are written in MATLAB, the Builder NE is opened either by clicking the MATLAB start menu and selecting MATLAB → MATLAB Builder NE → deployment tool, or typing

```
>>deploytool
```

at the MATLAB command line. From this point, all that is needed is to include the function files in the build, create a name for the class, and select BUILD. The ASL Framework uses the .dll created from this process and allows function calls in MATLAB that are defined as below:

```
function [Y, Xnew, Pnew] = KalmanFilter(Zk, Xold, Pold, Xcov, Zcov,
bigA, bigB, bigH, inputs)
```

to be accessed in C# in the manner shown below:

```
double[] outputs = myKalmanFilter.KalmanFilter(Zk, inputs);
```

Here Xnew and Pnew are returned internally to the class `myKalmanFilter` along with the necessary covariance matrices. These matrices may be modified at any time by another method within the `myKalmanFilter` class. The outputs returned here are simply V and $d\Theta$.

Chapter 4: Tracking Results

The output of the Kalman filter is an estimate of the robots velocity V, and rate of change of heading, $d\Theta$. In order to produce a map of the path of the robot, integration is performed and the equations in the Wheel Odometry section are employed. In addition to mapping the output from the Kalman filter, for comparison, the behaviors predicted by the model alone (using purely inputs to generate V and $d\Theta$), as well as encoder-only odometry alone are computed. Once the integration is performed for each of these, a vector of X's and Y's are held in the form of a list in C# where they will be plotted.

Examining the performance of the Kalman filter is done by comparing the plotted track generated using the filter, to the path generated by the two simpler methods. In each experiment, the starting and final position of the vehicle should be identical. The following plots shown all contain four markers. The light green marker is the starting point of the robot, while the blue, red, and dark green markers show the final position as predicted by the Kalman filter, the model by itself, and the wheel speed sensors by themselves, respectively. These plots were generated by driving the vehicle in varying manners followed by returning to the original position. From this data, a total cumulative displacement error can be easily seen.

Figure 4-1 shows these paths when the robot is driven very gently, assuring that the limits of the dynamics of the system as reported in the system identification section are obeyed. It is clear from the plot that the model by itself is very capable of tracking the robot in certain circumstances. Mapping purely by the wheel speed sensors it is clearly shown to be problematic. Significant steering errors can be seen in the gross misestimate of vehicle heading, resulting in a final position error of ~1.4m out of 30 meters versus less than .2m for both the Kalman Filter and the model based estimate.

The following figure, Figure 4-2 shows a case where the robot was driven in a more haphazard manner. This excites more of the dynamics of the vehicle, as well as likely inducing tire-slip. This is a clear cause for the application of the Kalman filter as presented. The Kalman filtered and sensor fused result has a total error on the order of .1 meter, while purely model-based tracking is close to 2.5 meters and pure wheel speed odometry is closer to 5 meters over a course of ~36 meters.

The final image, Figure 4-3, shows the estimated path of the robot for a highly disorganized path of travel. In addition to multiple high-speed steering changes, the vehicle makes three complete circles on the return trip.

It is important to note the consistency of the performance. In all three plots, the robot drives different distances, performs varying maneuvers, and yet the Kalman filter still manages to estimate the return position with consistent accuracy. This is not a perfect result, as clearly there is still error in the estimated path. There are indelible causes for error that a dead-reckoning solution will not be able to fix. Further improvements to the methods posed are discussed in Future Work.

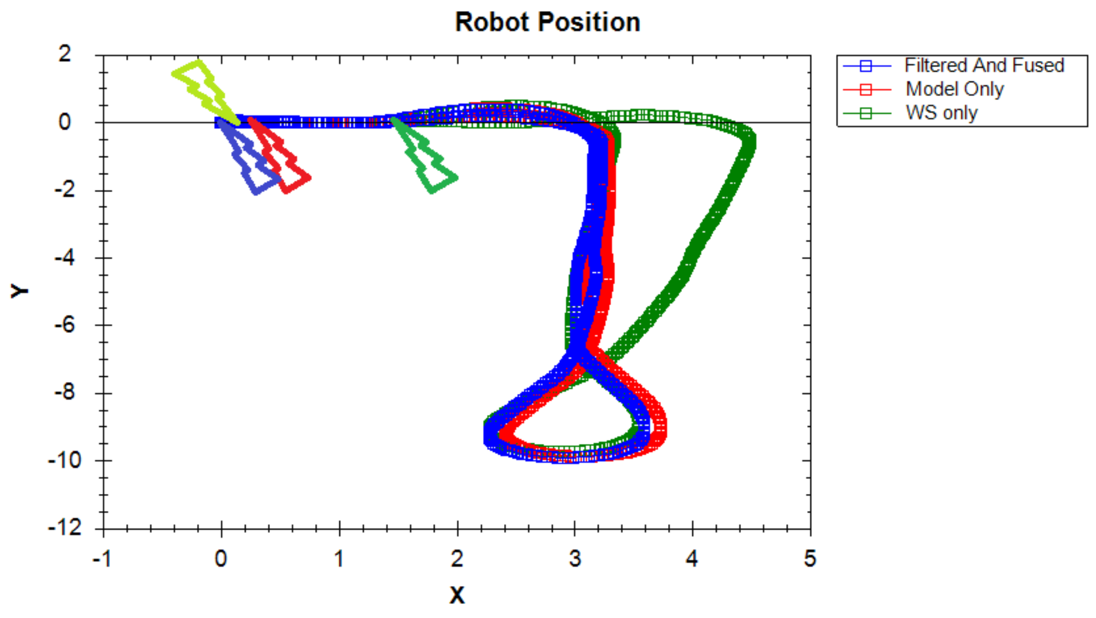


Figure 4-1: Smooth Plotted path with indicated start and stop points. Units in meters

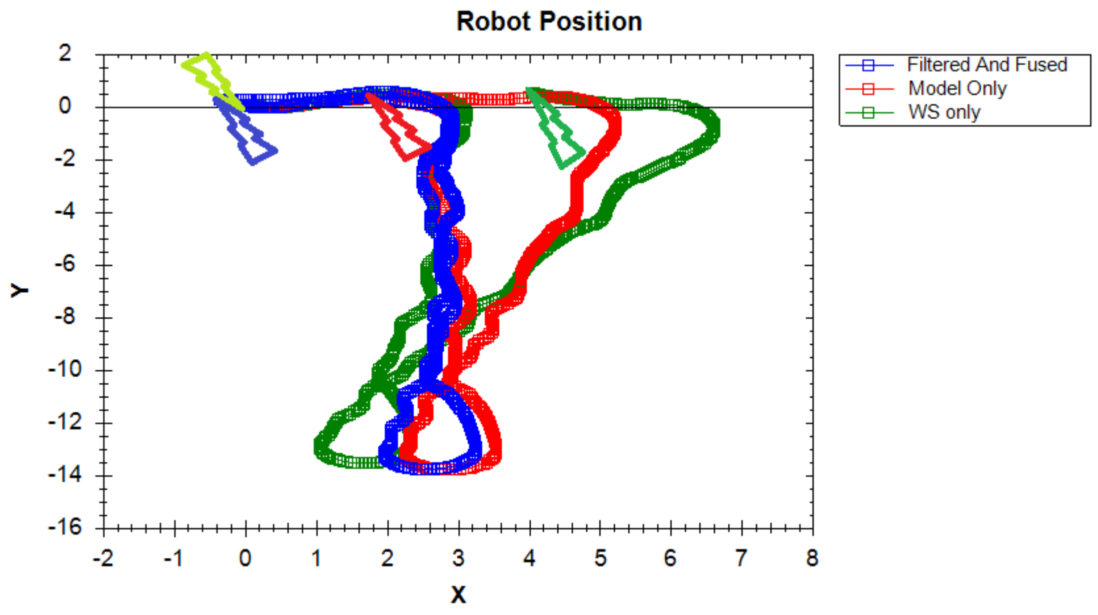


Figure 4-2: Wiggly Plotted path with indicated start and stop points. Units in meters

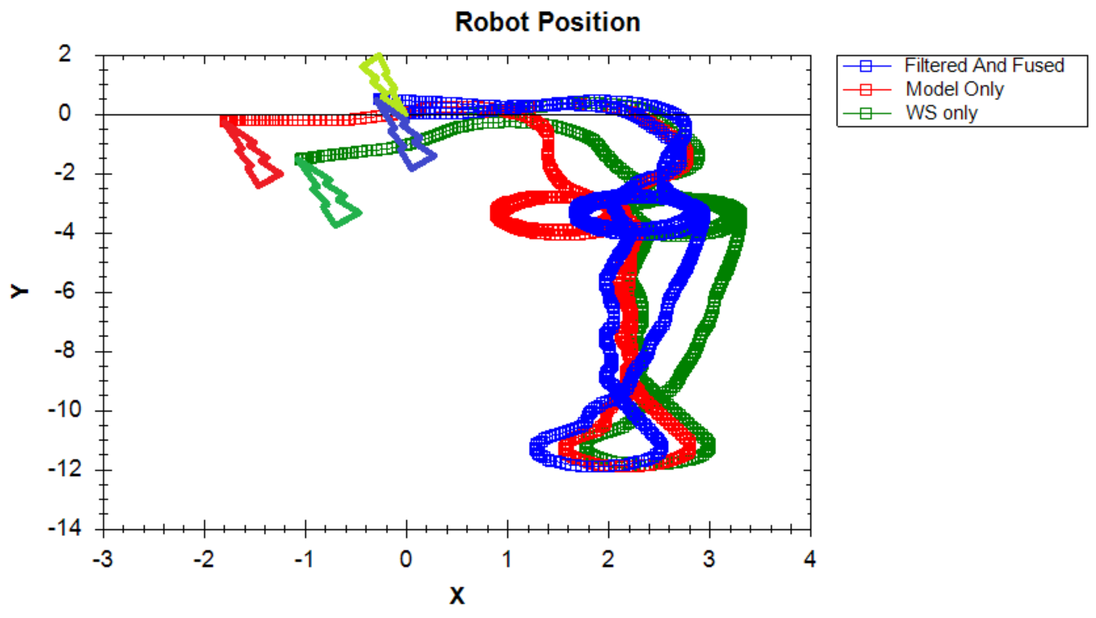


Figure 4-3: Wiggle and turnaround plotted path with indicated start and stop points. Units in meters

Chapter 5: Future Work

Towards the goal of autonomous navigation, this thesis demonstrated the ability to significantly improve tracking (state estimation) performance over raw sensor measurements using a Kalman Filter. A logical next step is to implement a closed loop control that allows the robot to move towards a set of given waypoints completely autonomously in a GPS denied environment.

Several issues arise in closing the loop. The first and probably the most problematic for long term operation of the system is that the sensors selected are all body reference frame sensors. Because of this, the robot has no global reference and no method of correcting an error in location once it starts on its course. While the Kalman filter performed well in the tests as shown, these were short duration tests and accumulation of error could be an issue in longer tests.

To address this, other global reference sensors may be added, for example an optical sensor. Optical sensors provide capability to not only to locate but also to confirm waypoints visually. They may also provide a means to autonomously select waypoints (or landmarks) and generate their own building map, which is the topic of another ASL lab thesis.

As discussed further below, cameras can also be used to correct the estimate for heading and velocity using ego motion algorithms to supplement both inertial and wheel encoder measurements as an additional body reference frame sensor. There has been much work in both wheel odometry and visual odometry (VO), while only limited attention has been paid to the intersection of the two approaches [24].

Certainly, improvements to our developed wheel encoder tracking methods posed in this thesis are also possible. In the Wheel Odometry section, the equations for heading were derived assuming a piecewise circular path, and ideal steering geometry. Performing accurate steering calibrations can improve the data returned by the encoders for heading.

5.1. Camera as a sensor

While not a motion sensor by itself, algorithms are commonly used to employ cameras as motion detectors. Specific to this application, *ego-motion* is the type of sensing of interest. Visual odometry is a type of ego-motion, where vehicle displacement and rotation may be measured in full or in part by imaging. Ego-motion is the estimation of a camera's motion relative to rigidly place objects; in this case, the scenery passed by the robot. This can be derived from instantaneous velocity measurements in the form of *optic flow*, or more positional-based measurements by observing the translation of the image frame or of one particular object identified by algorithms that are more capable.

Imaging is achieved through use of an off the shelf web-cam device. Webcams are so-called because of their colloquial use for holding face-to-face conversations over the internet. Most webcams utilize CMOS imager arrays due to their low cost. Such is the case with our webcam. The choice to use a webcam stems from how inexpensive they are, in conjunction with their ease of interface with our software framework. The web cameras provide images to the framework at a rate of 30fps, though not all cameras are equal in this respect. Our images suffer from an artifact caused by a *rolling shutter,* something most inexpensive CMOS cameras employ. A rolling shutter samples the image sensor array in a serial manner, rather than taking a complete "snapshot" at one time instance. Considering the sensor to be a matrix of measurement points, one entire row is sampled starting with the uppermost (top of figure) and continuing until the last row (bottom of figure). For slowly varying images, or slow moving objects, a clear image that accurately depicts the subject will be produced. However, the rolling shutter causes inaccuracy when measuring dynamic events such as the case on a moving robot where ego-motion is employed. If a solid vertical object is introduced into the frame, and the camera turns quickly, causing the object to move rapidly from one side to the camera frame to the other, the final camera image after a complete shutter sequence would show the object tilting to one side. Figure 5-1 shows the effects of a rolling shutter when the camera is panning to the right. The image to the left is the original object location, while the image on the right shows the result when scanned by a rolling shutter (scanning vertically downward).

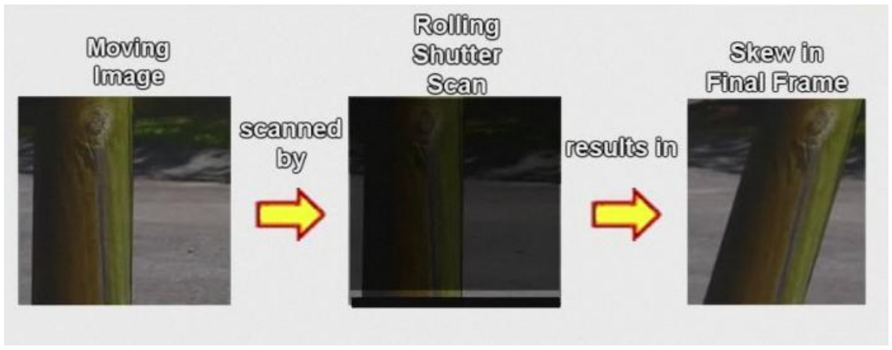


Figure 5-1: Rolling Shutter [25]

This artifact is called *skew*. This is problematic for ego motion for two reasons. Firstly, images that have translated in a highly dynamic manner may look different in their new frame when compared to the previous and may require an unknown eigenvalue transformation to compare to the previous frame. Secondly, considering that the image skews, the different parts of the object of interest appear to move at different velocities. This is essentially the same point as the first, but it is important to note the implications of the different analyses. This presents a problem when measuring optic flow or optic translation information as will be discussed below.

5.2. Optic Flow and Image Translation

Two methods are proposed for using a camera for longitudinal and yaw velocity feedback to further aid localization: optic flow and image translation. Methods to garner this information from video information exist in the OpenCV computer vision library. Emgu provides code written in .NET access to algorithms within the C-based OpenCV (**Open** Source **C**omputer **V**ision) library [26]. These methods are not employed, but have been researched for use on future platforms. Unfortunately, the quality of images is insufficient in the tests performed to justify the use of these data as sensors in this thesis. Cause for the poor quality is due in part to the image quality of the camera and partly due to the surroundings of the robot during testing (see Figure 5-2: Testing Hallway. Completely white walls, solid lines, and periodic features do not support good results from either optic flow or image translation.

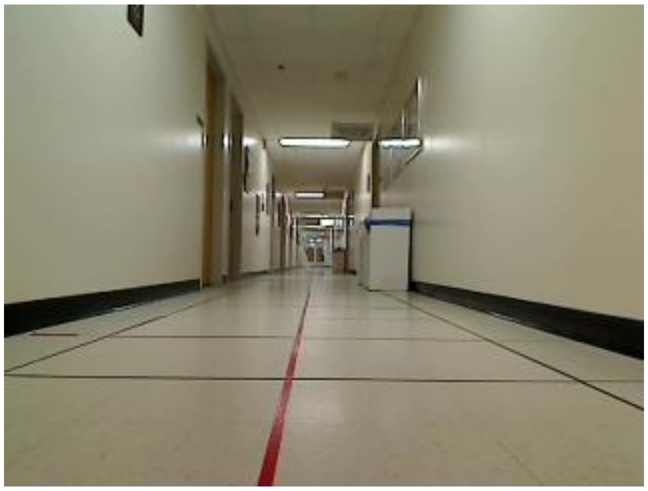


Figure 5-2: Testing Hallway

The concept of Optic Flow involves identifying points that remain unique between sequential frames in time, followed by measuring the displacement of those points between frames. This returns a velocity for every point tracked. Assuming many points may be tracked in an image, a relatively accurate result for velocity can be achieved. Figure 5-3: Optic Flow imagery of moving keyboard is an example of optic flow applied to a video of a camera moving over a keyboard. The points being tracked are denoted by red circles, and the vectors represent the direction and magnitude of the velocity of the points between sequential frames.

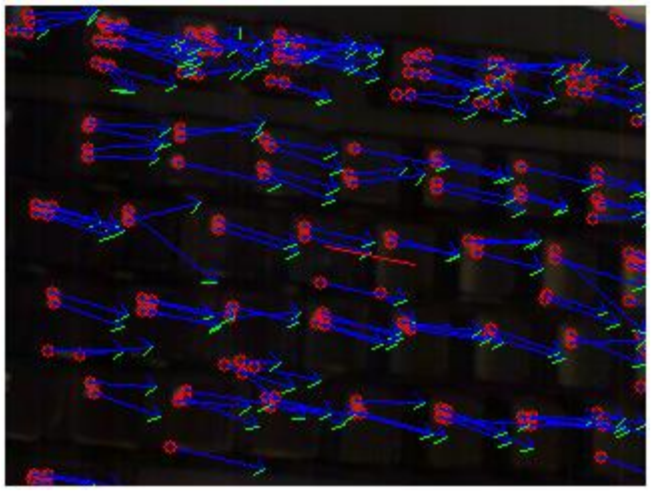


Figure 5-3: Optic Flow imagery of moving keyboard

Image translation is a term given to the measured translation of an image or portion of an image in a frame. OpenCV provides a method called template matching that reports the probability of a tested image being located at a specific location within a reference frame by sliding the tested image over the reference frame and comparing the similarities. Figure 5-4 shows the results from the same sliding keyboard test. Bright spots indicate high probabilities, indicating the object is moving by the Cartesian displacement of the bright spot from the center of the frame.

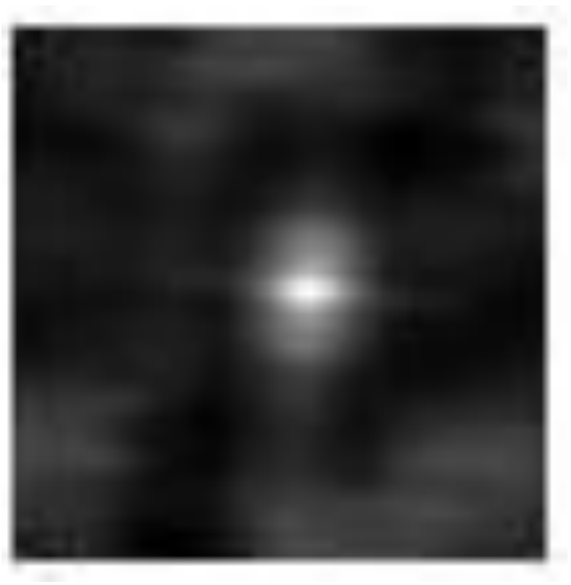


Figure 5-4: Image template probability map of sliding keyboard test

These two methods are very computationally demanding, causing the need to use lower-resolution images or smaller regions of interest. Simultaneous application of both these methods is unfeasible on the computer used in this research because the computer is not fast enough.

The concept of applying the Kalman filter to tracking of a ground platform is not a new one, but is applied for the purposes of understanding and supplementing additional research at the Autonomous Systems Lab. The same methods for model creation and Kalman filter application can be applied to many types of vehicles for land, sea, and air operation. The models may be more complex, or perhaps non-linear, but there are still ways of implementing the Kalman filter for this. The Extended Kalman filter and the Unscented Kalman filter are both used for applying the Kalman filter to non-linear systems. These are of interest to the ASL lab, and will be implemented on the Align

TREX Helicopter pictured in Figure 5-5 for stabilization and full autonomy

Figure 5-5: Align TREX Hobby helicopter [27]

Appendix A: CAD

A.1. Platform CAD Drawings

The Tamiya TXT-1 has two parallel vertical frame rails (shown in grey in Figure A-1) that form the primary structure of the remote control truck. In order to support the additional sensors and computing platform, an aluminum plate is attached to the top of these rails (shown as yellow in the image for clarity) by means of six custom brackets (shown in blue and red). The brackets raise the platform above the moving components of the truck. Sensor and computer mounting brackets are depicted in green in the image, and are mounted to the platform by means of #6-32 tapped holes at regular intervals.

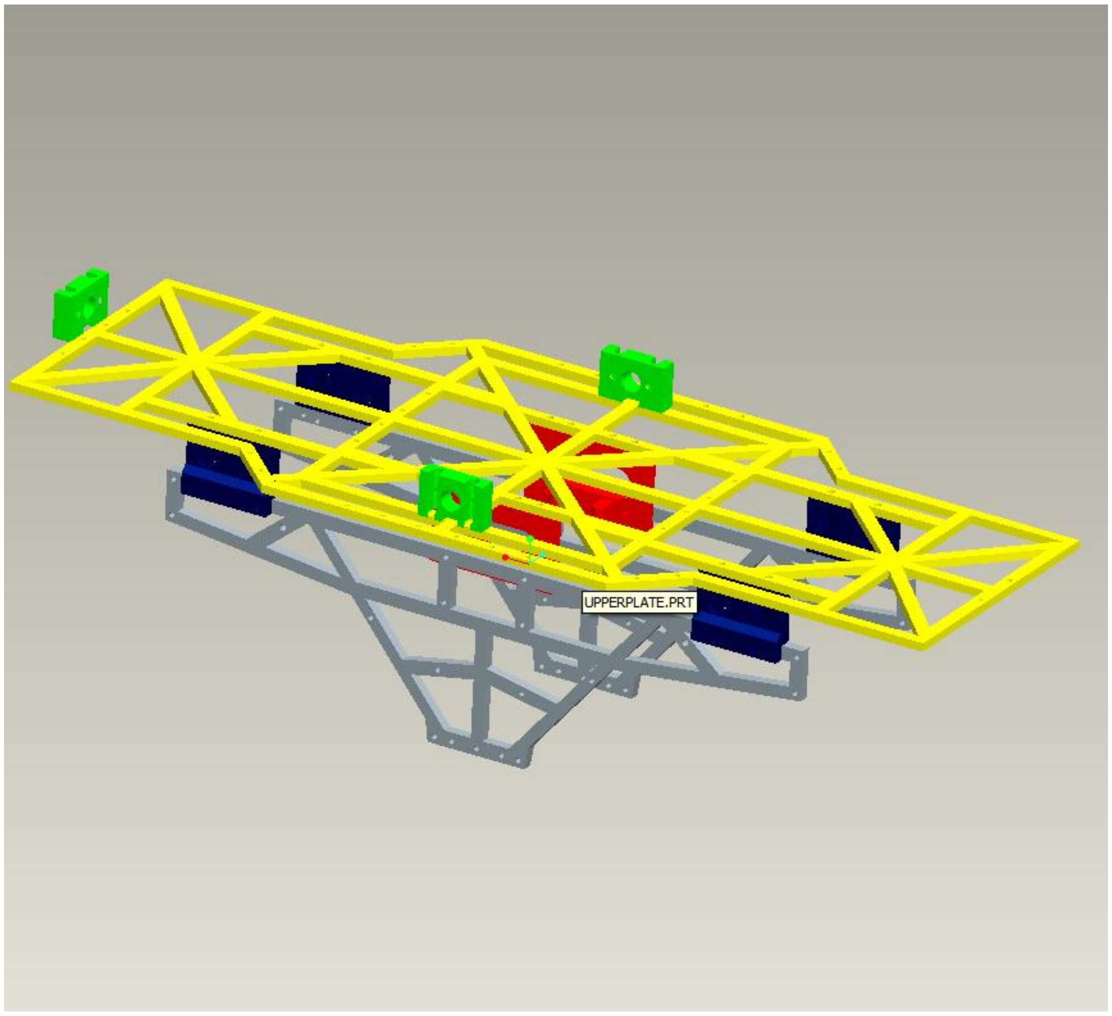


Figure A-1: Platform Modifications CAD

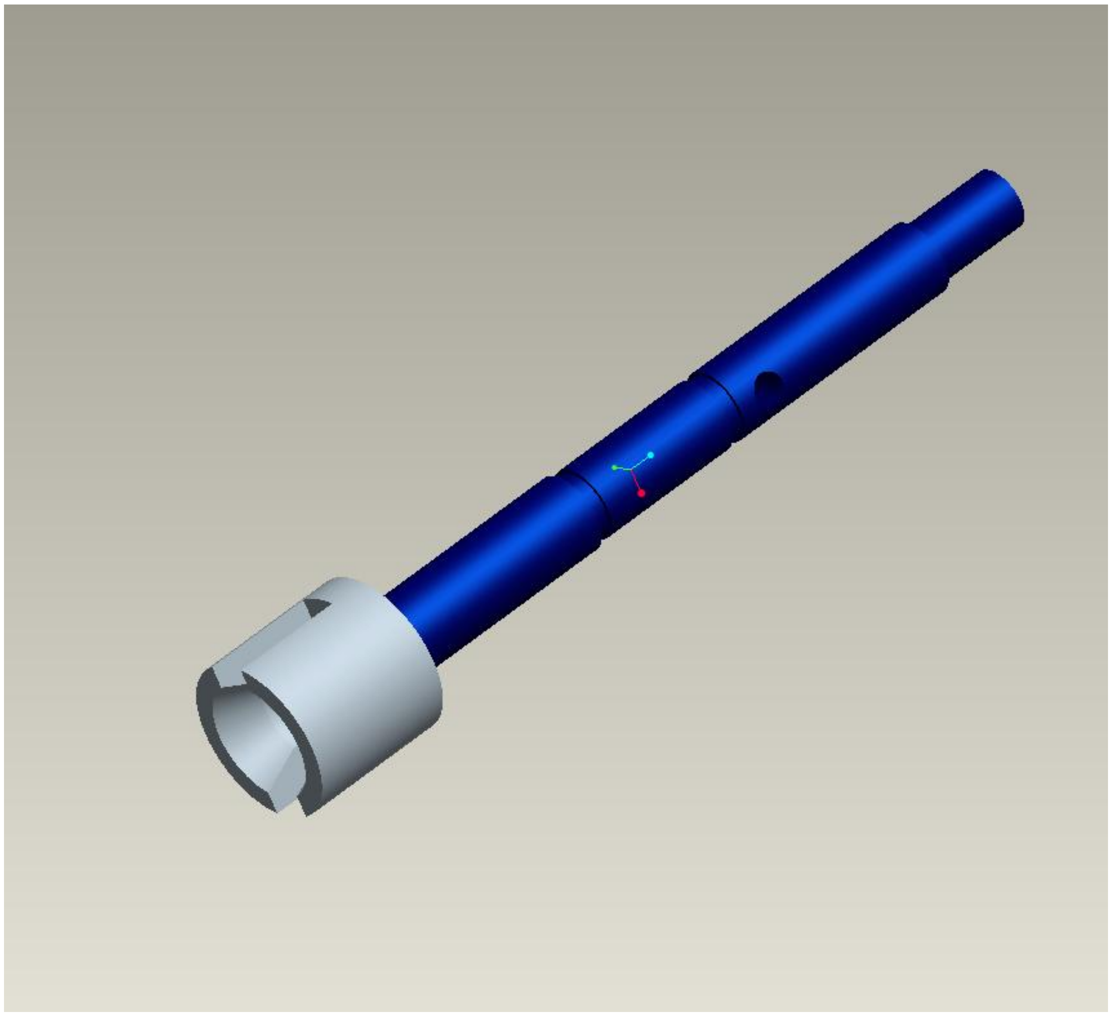


Figure A-2: Custom stub-axle for wheel-speed sensor application

Appendix B: Kalman filter MATLAB Code

```
function [Xnew, Pnew] = myKalman(Zk, Xold, Pold, Xcov, Zcov, Unew, Hk,
Bk, Fk)

%% Gather inputs
Qk = Xcov;  % State covariance
Rk = Zcov;  % Observation covariance
Uk = Unew;

%% Predict Stage
Xpred = Fk * Xold + Bk * Uk;        % Internal Dynamics.
Ppred = Fk * Pold * Fk' + Qk;       % Prediction of estimate covariance

%% Support Calculations (part of the update stage)
Yk = Zk - Hk * Xpred;               % Find residuals (error)
Sk = Hk * Ppred * Hk' + Rk;         % Innovation covariance
Kk = Ppred * Hk' * Sk^-1;           % Calc Kalman Gain

%% Correction stage (final part of the update stage)
Xnew = Xpred + Kk * Yk;             % Updated State estimate
Pnew = (eye(length(Hk)) - Kk * Hk) * Ppred; % updated estimate
covariance

end
```

Bibliography

[1] Junmin Wang and Joe Steiber, "Autonomous Ground Vehicle Control System for High-Speed and Safe Operation," in *American Control Conference*, Seattle, Washington, 2008, p. 6.

[2] Garmin. (2010) Garmin| What is WAAS? [Online]. http://www8.garmin.com/aboutGPS/waas.html

[3] Jared Napora, "IMPLEMENTATION, EVALUATION, AND APPLICATIONS OF MOBILE MESH NETWORKS FOR PLATFORMS IN MOTION," University of Maryland, College Park, Master's Thesis 2009.

[4] Tamiya TXT-1 Monster Truck. [Online]. http://www.towerhobbies.com/products/tamiya/tamc0280.html

[5] LynxMotion. [Online]. http://www.lynxmotion.com/Product.aspx?productID=287&CategoryID=61

[6] Tower Hobbies Servo. [Online]. http://www3.towerhobbies.com/cgi-bin/wti0001p?&I=LXUZ80

[7] Normal Distribution. [Online]. http://en.wikipedia.org/wiki/Normal_distribution

[8] High-Precision Tri-Axis Inertial Sensor. [Online]. http://www.analog.com/en/other-products/multi-chip/adis16355/products/product.html

[9] National Instruments. Using Quadrature Encoders with E Series DAQ Boards. [Online]. http://zone.ni.com/devzone/cda/tut/p/id/4623

[10] (2008) Rotary Encoder Wiki. [Online]. http://en.wikipedia.org/wiki/Rotary_encoder

[11] Input Sensors for Position Change. [Online]. http://www.sxlist.com/techref/io/sensor/pos/enc/quadrature.htm

[12] Thomas D Gillispie, *Fundamentals of Vehicle Dynamics*, 1st ed. Warrendale, PA, USA: Society of Automotive Engineers, INC, 1992. [Online]. http://www.emgu.com/wiki/index.php/Main_Page

[13] Shailesh Lakkad, "Modeling and Simulation of Steering Systems for Autonomous Vehicles," Florida State University, Master's Thesis 2004.

[14] Odometry Wiki. [Online]. http://en.wikipedia.org/wiki/Odometry

[15] Haim Baruh, *Analytical Dynamics*.: McGraw-Hill, 1999.

[16] Ron Kurtus. (2008, August) Rolling Friction. [Online]. http://www.school-for-champions.com/science/friction_rolling.htm

[17] Alan V. Oppenheim, Alan S. Willskey, and S. Hamad Nawab, *Signals and Systems*, 2nd ed. Boston, MA, USA: Prentice Hall, 1997.

[18] Lennart Ljung. (2010, March) Matlab System Identification Toolbox. [Online]. http://www.mathworks.com/access/helpdesk/help/toolbox/ident/

[19] (2010, April) Kalman Filter Wiki. [Online]. http://en.wikipedia.org/wiki/Kalman_filter

[20] Greg Welch and Gary Bishop, "An Introduction to the Kalman Filter," University of North Carolina at Chapel Hill, Chapel Hill, NC, 2006.

[21] David Titterton and John Weston, *Strapdown Inertial Navigation Technology*, 2nd ed. Reston, VA, USA: IEEE, AIAA, 2004.

[22] Richard M. Murray. (2008, Feb) Richard M. Murray Wiki. [Online]. http://www.cds.caltech.edu/~murray/courses/cds110/wi08/L8-1_fusion.pdf

[23] [Online]. http://www.mathworks.com/products/netbuilder/

[24] Yann LeCun and Mathew Grimes, "Efficient Off-Road Localization Using Visually Corrected Odometry," in *International Conference on Robotics and Automation*, 2009, p. 1.

[25] Barry Green. SENSOR ARTIFACTS AND CMOS ROLLING SHUTTER. [Online]. http://www.dvxuser.com/jason/CMOS-CCD/

[26] (2009, October) EMGU CV. [Online]. www.emgu.com

[27] [Online]. www.align.com.tw/