

OPTIMUM PATH TRACKING OF AN  
INDEPENDENTLY STEERED FOUR WHEELED MOBILE ROBOT

A Thesis  
Submitted to the Graduate Faculty  
of the  
North Dakota State University  
of Agriculture and Applied Science

By

Jonathan Richard Nistler

In Partial Fulfillment  
for the Degree of  
MASTER OF SCIENCE

Major Department:  
Mechanical Engineering

March 2012

Fargo, North Dakota

North Dakota State University  
Graduate School

---

**Title**

Optimum Path Tracking of an

---

Independently Steered Four Wheeled Mobile Robot

---

**By**

Jonathan Richard Nistler

---

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

---

SUPERVISORY COMMITTEE:

Dr. Majura Selekwa

---

Chair

Dr. Annie Tangpong

---

Dr. Mariusz Ziejewski

---

Dr. Jing Shi

---

Approved:

March 30<sup>th</sup>, 2012

---

Date

Dr. Alan Kallmeyer

---

Department Chair

## ABSTRACT

This thesis studies the navigational control problem for an independently steered and driven four-wheeled ground robotic vehicle, a subset of the larger problem of controlling self-reconfigurable robotic vehicles. A reconfigurable vehicle is kinematically modeled and simplified using a fixed suspension. A combined steering and speed control scheme is proposed that coordinates steering angles by remembering the path of the front axle, given by navigational sensors.

Simulation and experimental results are provided to validate the proposed algorithms. One simulation demonstrates the performance of the controller, while the second compares the maneuverability of the proposed steering algorithm with existing methods. The first experiment compares the performance of the proposed algorithm with existing algorithms on a variety of preprogrammed paths, and the second compares the performance by reactively constructing the path in real time using sensors. Results show the proposed algorithm outperformed others consistently by rates up to 40%, depending on path geometry.

## ACKNOWLEDGEMENTS

I would like to thank first and foremost my advisor, Dr. Majura Selekwa. His vigilant guidance and encouragement have made this thesis possible. He imparted to me not only knowledge of this subject matter, but life lessons also. He has taken on many roles to be a champion of my education as well as a great friend.

I would also like to thank my graduate committee, including Dr. Annie Tangpong, Dr. Mariusz Ziejewski, and Dr. Jing Shi, without whose direction and contributions, this thesis would not have been possible. I am grateful to NDSU, the Mechanical Engineering Department, and Dr. Alan Kallmeyer for the opportunity and financial means necessary to complete this degree. The department faculty and staff, as well as my fellow graduate students have been invaluable resources in this pursuit.

I would like to thank my parents, who have always pushed me to my potential. I know that they have made immeasurable sacrifices to get me to this point in my life. I would also like to thank my brothers and sister, extended family and all of my in-laws who have not only been supportive but proud of my studies. Their encouragement has been the fuel to keep me up late into the night and at school on weekends.

I am deeply indebted to my wife, Brianna, who has been more than patient with me throughout my entire post-secondary education. Her support and understanding have left her alone on the couch or at the dinner table too many times. She has been my advocate, ally, sounding board and confidante on whom I can always rely, and for this I am grateful.



## TABLE OF CONTENTS

ABSTRACT .....	iii
ACKNOWLEDGEMENTS .....	iv
LIST OF TABLES .....	x
LIST OF FIGURES .....	xi
LIST OF APPENDIX FIGURES.....	xiii
<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>1.1. THE GOAL OF THE RESEARCH.....</b>	<b>1</b>
<b>1.2. HISTORICAL DEVELOPMENTS IN POWERED GROUND VEHICLES.....</b>	<b>1</b>
<b>1.2.1. THE EVOLUTION OF STEERING TECHNOLOGIES.....</b>	<b>2</b>
<b>1.2.2. THE EVOLUTION OF POWER TRAIN CONFIGURATIONS.....</b>	<b>3</b>
<b>1.3. EMERGENCE AND FEATURES OF ALL WHEEL STEER AND ALL WHEEL DRIVE VEHICLES .....</b>	<b>5</b>
<b>1.4. WHEELED GROUND ROBOTIC VEHICLES.....</b>	<b>10</b>
<b>1.4.1. CLASSIFICATION OF FIXED-MORPHOLOGY ROBOTIC VEHICLES.....</b>	<b>11</b>
<b>1.4.2. SELF-RECONFIGURABLE ROBOTS .....</b>	<b>18</b>
<b>1.5. STATEMENT OF THE RESEARCH PROBLEM.....</b>	<b>20</b>
<b>2. DYNAMIC MODELING OF WHEELED ROBOTS.....</b>	<b>22</b>
<b>2.1. MODELING OF A FOUR WHEELED RECONFIGURABLE ROBOT WITH A PRISMATICALLY ARTICULATED SUSPENSION .....</b>	<b>22</b>
<b>2.1.1. KINEMATIC CONSTRAINTS.....</b>	<b>22</b>
<b>2.1.2. THE DYNAMIC MODEL.....</b>	<b>25</b>
<b>2.2. MODEL REDUCTION FOR A FIXED SUSPENSION 4WS/4WD ROBOT.....</b>	<b>30</b>
<b>2.2.1. MODEL REDUCTION APPROACH.....</b>	<b>30</b>
<b>2.2.2. THE DYNAMIC MODEL.....</b>	<b>32</b>

3.	DEVELOPMENT OF AN OPTIMAL CONTROL ALGORITHM FOR AWS/AWD PATH TRACKING .....	35
3.1.	LITERATURE REVIEW .....	35
3.2.	THE PROPOSED APPROACH .....	38
3.2.1.	DEFINITION OF THE KINEMATIC CONSTRAINTS .....	38
3.2.2.	INCORPORATION OF THE KINEMATIC CONSTRAINTS INTO THE PATH TRACKING CONTROLLER .....	43
4.	SIMULATION RESULTS .....	46
4.1.	NUMERICAL VALIDATION OF THE DECENTRALIZED CONTROLLER.....	48
4.2.	SIMULATED PATH TRACKING RESULTS.....	52
4.2.1.	FRONT WHEEL STEER – <i>2WF</i> FEATURES.....	56
4.2.2.	FRONT AXLE PATH TRACKING – <i>4FM</i> FEATURES.....	57
4.2.3.	CENTER OF GRAVITY TRACKING – <i>4CG</i> FEATURES.....	57
4.2.4.	FRONT AND REAR AXLE PATH TRACKING – <i>4FR</i> FEATURES.....	59
4.2.5.	PERFORMANCE METRIC AND RESULTS.....	60
5.	IMPLEMENTATION OF THE REDUCED MODEL ALGORITHM ON BIBOT-1 VEHICLE.....	64
5.1.	VEHICLE DESCRIPTION .....	64
5.1.1.	BACKGROUND HISTORY .....	64
5.1.2.	CONTROL HARDWARE AND SOFTWARE.....	65
5.1.2.1.	HARDWARE DESCRIPTION .....	65
5.1.2.2.	SOFTWARE DESCRIPTION .....	67
5.2.	PERCEPTION AND ODOMETRY SYSTEMS DESIGN .....	69
5.2.1.	SONAR PERCEPTION SYSTEM .....	69
5.2.2.	ODOMETRY SYSTEM.....	69
5.3.	ODOMETRY AND CONTROL SOFTWARE .....	71

5.3.1. VCU CODE SCHEME.....	71
5.3.2. WCU CODE SCHEME .....	75
6. EXPERIMENTAL RESULTS.....	78
6.1. PERFORMANCE RESULTS ON PASSIVE PRE-PROGRAMMED PATHS.....	78
6.1.1. EXPERIMENTAL SETUP AND PROCEDURE .....	80
6.1.2. EXPERIMENTAL RESULTS.....	82
6.2. PERFORMANCE RESULTS ON REACTIVE OBSTACLE DEFINED PATHS.....	84
6.2.1. EXPERIMENTAL SET UP AND PROCEDURE .....	85
6.2.2. EXPERIMENTAL RESULTS.....	88
6.3. SOURCES OF ERROR.....	90
7. CONCLUDING REMARKS.....	92
7.1. FUTURE WORK.....	93
8. REFERENCES .....	95
APPENDIX A. MATLAB STEERING SIMULATION CODE .....	104
A.1. FEEDBACK CONTROLLER SIMULATION .....	104
A.1.1. CONTROL1.M .....	104
A.2. 2WS CONFIGURATION.....	107
A.2.1. ANIMATE.M.....	107
A.3. AWS STEERING CONFIGURATION, FRONT WHEEL TRACKING .....	113
A.3.1. ANIMATE.M.....	113
A.4. AWS STEERING CONFIGURATION, CENTER OF GRAVITY TRACKING .....	120
A.4.1. ANIMATE.M.....	120
A.5. AWS STEERING CONFIGURATION, FRONT AND REAR WHEEL TRACKING.....	127
A.5.1. ANIMATE.M.....	127

<b>A.6. CODE COMMON TO ALL SIMULATIONS.....</b>	<b>134</b>
<b>A.6.1. CAR.M.....</b>	<b>134</b>
<b>A.6.2. CURVY.M.....</b>	<b>134</b>
<b>A.6.3. LINE(1,2,3,4).M.....</b>	<b>135</b>
<b>A.6.4. ORIGINAL.M.....</b>	<b>136</b>
<b>A.6.5. ROTATEME.M.....</b>	<b>137</b>
<b>A.6.6. TRANSLATEME.M.....</b>	<b>138</b>
<b>A.6.7. UTURN.M.....</b>	<b>138</b>
<b>A.6.8. WHEEL(1,2,3,4).M.....</b>	<b>139</b>
<b>A.6.9. ZIGZAG.M.....</b>	<b>139</b>
<b>APPENDIX B. FULL SIMULATION RESULTS.....</b>	<b>141</b>
<b>B.1. FRONT WHEEL STEER – 2WF.....</b>	<b>141</b>
<b>B.2. FRONT AXLE TRACKING – 4FM.....</b>	<b>144</b>
<b>B.3. CENTER OF GRAVITY TRACKING – 4CG.....</b>	<b>147</b>
<b>B.4. FRONT AND REAR AXLE TRACKING – 4FM.....</b>	<b>150</b>
<b>APPENDIX C. PREPROGRAMMED PATH FULL EXPERIMENTAL RESULTS.....</b>	<b>153</b>
<b>C.1. FRONT WHEEL STEER – 2WF.....</b>	<b>153</b>
<b>C.2. CENTER OF GRAVITY TRACKING – 4CG.....</b>	<b>156</b>
<b>C.3. FRONT AND REAR AXLE PATH TRACKING – 4FR.....</b>	<b>159</b>
<b>APPENDIX D. SENSORY NAVIGATION FULL EXPERIMENTAL RESULTS.....</b>	<b>162</b>
<b>D.1. FRONT AXLE TRACKING – 4FM.....</b>	<b>162</b>
<b>D.2. FRONT AND REAR AXLE PATH TRACKING – 4FR.....</b>	<b>165</b>

<b>APPENDIX E. INDIVIDUAL WHEEL UNIT CONTROLLER C CODE (4X)</b> .....	168
<b>E.1. WCU_MAIN.C</b> .....	168
<b>E.2. WCU_DRIVERS.C</b> .....	178
<b>E.3. WCU_DRIVERS.H</b> .....	185
<b>E.4. WCU_TRAPS.C</b> .....	187
<b>APPENDIX F. CENTRAL VEHICLE BODY CONTROLLER C CODE</b> .....	191
<b>F.1. VCU_MAIN.C</b> .....	191
<b>F.2. VCU_DRIVERS.C</b> .....	209
<b>F.3. VCU_DRIVERS.H</b> .....	221
<b>F.4. VCU_TRAPS.C</b> .....	223

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
1.1 - DIFFERENTIALLY DRIVEN ROBOTS.....	14
1.2 - CAR-LIKE ROBOTS.....	15
1.3 - XY ROBOTS .....	16
1.4 - ALL-WHEEL-STEER ROBOTS.....	17
1.5 - OMNIDIRECTIONAL ROBOTS .....	18
4.1 - VEHICLE SIMULATION PARAMETERS.....	46
4.2 - FEATURES OF COMPARED STEERING ALGORITHMS.....	53
4.3 - WHEEL DEVIATION FROM IDEAL PATH .....	62
6.1 - RMS DEVIATION FROM CONTROL PATH.....	82

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1.1 - THE FRONT AND REAR WHEEL PATHS OF A TYPICAL PASSENGER VEHICLE [14] .....	4
1.2 - AN ILLUSTRATION DEMONSTRATING THE REQUIREMENTS OF THE ICR [18].....	6
1.3 - AN OPEN DIFFERENTIAL WHICH ALLOWS LEFT AND RIGHT AXLES TO ROTATE AT DIFFERENT SPEEDS [20].....	7
1.4 - EXAMPLE SCENARIO WHERE 4WD SYSTEMS CAN BE IMMOBILIZED EVEN WITH TRACTION AT TWO WHEELS [21] .....	8
1.5 - A LIMITED SLIP DIFFERENTIAL THAT USES AN ELECTRONICALLY ACTIVATED CLUTCH PACK (YELLOW, LEFT) [22] .....	9
1.6 - EXAMPLE SCENARIO WHERE AWD SYSTEMS DIVERT POWER FROM WHEELS WITH NO TRACTION TO WHEELS WITH TRACTION [21] .....	10
1.7 - A MECANUM WHEEL [37] .....	13
1.8 - SEVERAL ARTICULATING SELF-RECONFIGURABLE ROBOTS [53] [51] [54] .....	19
1.9 - THE BIBOT-2 SELF-RECONFIGURABLE ROBOT EXTENDED (LEFT) AND RETRACTED (RIGHT) .....	20
2.1 - THE CONFIGURATION THE COORDINATE SYSTEM FOR A ROBOTIC VEHICLE.....	23
2.2 - FORCES AND TORQUES ACTING ON A WHEEL.....	26
2.3 - FORCES ACTING ON THE VEHICLE BODY.....	27
2.4 - PARAMETERS FOR A TYPICAL FOUR-WHEEL-STEERED VEHICLE .....	31
3.1 - TWO TYPICAL CONFIGURATIONS OF FOUR-WHEEL-STEERED VEHICLES .....	36
3.2 - THE LATERAL STABILITY OF TYPE (A) VEHICLES IS COMPROMISED IN TIGHT CORNERS .....	36
3.3 - PARAMETERS FOR A TYPICAL FOUR WHEEL STEERED VEHICLE .....	39
3.4 - A LAYOUT OF THE PROPOSED CONTROL SCHEME.....	44
4.1 - MATLAB SIMULATION FOR AN ARBITRARY PATH SHOWING CENTRODES AND CG PATH .....	47
4.2 - SIMULATION TRAJECTORY.....	49

4.3 - SIMULATED TRACTION TORQUE .....	49
4.4 - SIMULATED STEERING TORQUE.....	50
4.5 - SIMULATED AND THEORETICAL WHEEL SPEED.....	51
4.6 - SIMULATED AND THEORETICAL WHEEL ANGLE .....	51
4.7 – SAMPLE SIMULATION OUTPUT OF $2WF$ ON S-CURVE PATH.....	55
4.8 – ILLUSTRATION OF THE REAR WHEELS ‘LAGGING’ THE FRONT WHEELS .....	56
4.9 – $4CG$ TRACKING REQUIRES EXTREME ANGLES FOR RELATIVELY SMOOTH PATHS (LEFT) AND NEARLY INSTANTANEOUS ROTATION FOR NON-CONTINUOUS PATHS (RIGHT).....	58
4.10 – $4FR$ DOESN’T REQUIRE AS EXTREME OF ANGLES AS $4CG$ (LEFT) AND VERY CLOSELY APPROXIMATES IDEAL WHEEL TRACK (RIGHT).....	59
4.11 – VISUAL REPRESENTATION OF MANEUVERABILITY METRIC, BLACK DOTTED LINES REPRESENT IDEAL WHEEL PATHS EQUIVALENT TO HALF WHEEL TRACK ALONG THE PATH NORMAL.....	61
4.12 – COMPARISON OF PATH TRACKING STEERING CONFIGURATIONS.....	62
5.1 – THE BIBOT-1 WHEELED ROBOTIC VEHICLE EXPERIMENTAL PLATFORM.....	64
5.2 – HARDWARE CONNECTIONS TO VEHICLE CONTROL UNIT .....	66
5.3 - THE INDEPENDENT ODOMETER FOR MEASURING DISTANCE TRAVELED .....	70
5.4 – LOGIC FLOWCHART OF VEHICLE CONTROL UNIT WITH PERIPHERALS.....	74
5.5 – LOGIC CONTROL SCHEME OF WHEEL CONTROL UNIT.....	77
6.1 – CONTROL CURVES FOR EXPERIMENT 1 .....	79
6.2 – THE SAND TRACER MOUNTED TO THE AXLE TRACKER (LEFT) RECORDS THE MOTION OF THE VEHICLE (RIGHT) .....	81
6.3 – RMS DEVIATION OF BIBOT-1 FROM PRESCRIBED PATH .....	83
6.4 - $90^\circ$ TURN USING PVC PIPE TO CREATE CONFIGURABLE CORRIDORS .....	85
6.5 – EXPERIMENTAL PATHS FOR SENSORY NAVIGATION.....	86
6.6 – THE SONAR (BLUE) INTERPRETS SHALLOW ANGLES (GREEN) AS ROUND CORNERS AND LARGE ANGLES (RED) AS CUSPS.....	87
6.7 – PERFORMANCE COMPARISON OF BIBOT-1 NAVIGATING A CORRIDOR.....	89



## LIST OF APPENDIX FIGURES

<u>Figure</u>	<u>Page</u>
B.1 – 2WF U-TURN NO VEHICLE.....	141
B.2 – 2WF U-TURN WITH VEHICLE .....	141
B.3 – 2WF ZIG-ZAG NO VEHICLE.....	142
B.4 – 2WF ZIG-ZAG WITH VEHICLE .....	142
B.5 – 2WF S-CURVE NO VEHICLE.....	143
B.6 – 2WF S-CURVE WITH VEHICLE .....	143
B.7 – 4FM U-TURN NO VEHICLE .....	144
B.8 – 4FM U-TURN WITH VEHICLE.....	144
B.9 – 4FM ZIG-ZAG NO VEHICLE .....	145
B.10 – 4FM ZIG-ZAG WITH VEHICLE .....	145
B.11 – 4FM S-CURVE NO VEHICLE .....	146
B.12 – 4FM S-CURVE WITH VEHICLE .....	146
B.13 – 4CG U-TURN NO VEHICLE .....	147
B.14 – 4CG U-TURN WITH VEHICLE.....	147
B.15 – 4CG ZIG-ZAG NO VEHICLE .....	148
B.16 – 4CG ZIG-ZAG WITH VEHICLE.....	148
B.17 – 4CG S-CURVE NO VEHICLE .....	149
B.18 – 4CG S-CURVE WITH VEHICLE .....	149
B.19 – 4FM U-TURN NO VEHICLE .....	150
B.20 – 4FM U-TURN WITH VEHICLE .....	150
B.21 – 4FM ZIG-ZAG NO VEHICLE .....	151

B.22 – 4FM ZIG-ZAG WITH VEHICLE .....	151
B.23 – 4FM S-CURVE NO VEHICLE .....	152
B.24 – 4FM S-CURVE WITH VEHICLE .....	152
C.1 – 2WF TRACKING U-TURN.....	153
C.2 – 2WF TRACKING ZIG-ZAG.....	154
C.3 – 2WF TRACKING S-CURVE.....	155
C.4 – 4CG TRACKING U-TURN .....	156
C.5 – 4CG TRACKING ZIG-ZAG .....	157
C.6 – 4CG TRACKING S-CURVE .....	158
C.7 – 4FR TRACKING U-TURN .....	159
C.8 – 4FR TRACKING ZIG-ZAG .....	160
C.9 – 4FR TRACKING S-CURVE .....	161
D.1 – 4FM TRACKING U-TURN.....	162
D.2 – 4FM TRACKING 90° TURN.....	163
D.3 – 4FM TRACKING S-CURVE.....	164
D.4 – 4FR TRACKING U-TURN.....	165
D.5 – 4FR TRACKING 90° TURN.....	166
D.6 – 4FR TRACKING S-CURVE.....	167

# 1. INTRODUCTION

## 1.1. THE GOAL OF THE RESEARCH

The overall goal of this research is to find answers to the problem of controlling four wheeled independent drive, reconfigurable, articulated robotic vehicles with prismatic wheel support, because of their suitability for applications in terrains that are difficult to traverse by normal vehicles and humans. The effort of this thesis takes a step in that direction by developing a simple approach for addressing the complexity of the kinematic constraints for fixed suspension four wheeled vehicles in which all wheels are driven and steered (AWD/AWS). Although the proposed approach was tested on the target *fixed* suspension four wheeled AWD/AWS, it is hoped that by extension this method can be applied to the more general AWD/AWS *reconfigurable* robots with prismatic wheel supports.

Since typical reconfigurable robotic vehicles targeted by this research will have AWD/AWS capabilities, this chapter recalls how the technology of typical wheeled ground vehicles has evolved. The next section provides an overview of key developments in power train and vehicle steering technology over the years, along with their advantages and disadvantages.

## 1.2. HISTORICAL DEVELOPMENTS IN POWERED GROUND VEHICLES

Most of the powered ground vehicles throughout human history have been automobiles; they have significantly shaped our society, leading expeditions on the moon (1971) [1], forming motorsports such as NASCAR (1948) [2], and currently, a driving

economic force related to finite petroleum commodities [3]. The first commercially available gasoline powered automobile, which took the form of a tricycle, was produced by Karl Benz in 1885 [4]. In an effort to form a more stable platform, similar to horse-drawn carriages, in 1886, Gottlieb Daimler produced a four wheeled gasoline powered vehicle with a coupled front wheel steering system accompanied with other innovative advancements including a hot tube ignition system [5]. Modern cars still take this historical form of a front-wheel-steered, four wheeled vehicle, but they have undergone transformations in the form of power distribution and steering configurations.

### **1.2.1. THE EVOLUTION OF STEERING TECHNOLOGIES**

As automobiles improved, an entirely new field of study based on vehicle dynamics was borne; the initial focus was the kinematics of vehicle suspensions. Many factors were observed to play into vehicle handling including camber control, castor alignment, steer angle coordination, and weight transfer. New elements such as the double wishbone linkage, MacPherson struts, and independent suspensions were created to manage these parameters. Effort was also placed in the development of four-bar linkages that can closely exhibit Ackermann steering angles [6]. Further evolution of the automobile brought commercially available all-wheel-steer (AWS) technology to vehicles in the 1980's [7]. Early skidpad and slalom tests on AWS vehicles showed that this technology increased lateral acceleration of the vehicle, which made them more maneuverable with decreased turning radius at low speeds, when turning front and rear wheels in the opposite direction [8] [9]. Furthermore, at high speeds, AWS vehicles have been shown to have decreased

phase lag and increased stability when steering front and rear wheels in the same direction [7] [10]. This property is advantageous during high speed lane change maneuvers.

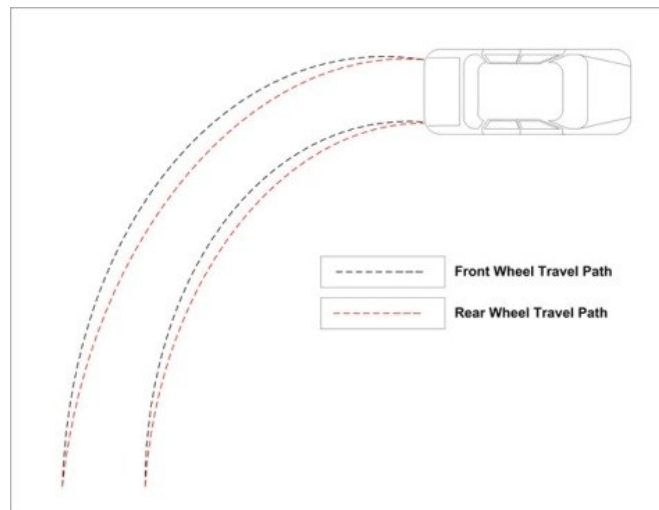
### **1.2.2. THE EVOLUTION OF POWER TRAIN CONFIGURATIONS**

In addition to developments in steering technologies, there were also new developments in power train technology which resulted in various power train configurations. From the early vehicles where rear-wheel-drive (RWD) power trains dominated, commercial automobiles have evolved in various power train configurations such as the four-wheel-drive systems (4WD) that appeared for the first time in the Spyker-60 vehicles in 1903 [11], and the front-wheel-drive (FWD) configuration that appeared for the first time in the Cord L-29 vehicles in 1929 [12]. Each power train configuration offers its own advantages and disadvantages.

FWD cars are more stable for an average driver, exhibiting predictable understeer, where the minimum cornering radius increases with increased speed. They also have minimal space requirements, but require mechanically complex transmission components and may exhibit torque steer; the tractive forces at the wheels cause a torque that counters the driver steering effort in an attempt to straighten the wheels.

RWD vehicles allow a balance of power and control that many advanced drivers prefer, however, they can exhibit oversteer, whereby the tractive forces at the back wheels cause a greater sideslip than the front wheels. When this condition happens, the vehicle becomes unstable, displaying unwanted characteristics such as drifting, 'fish-tailing', or a spinout. RWD cars dominated the market until the 1970's when FWD became economical and practical.

4WD systems thrive in low friction environments such as snow and mud where traction is at a minimum. The redundant drive system is quite robust in accelerating vehicles because it can provide power to any or all wheels to the point of traction failure. However, the mechanical coupling between the front and rear axles at the locking transfer case requires the front and rear differentials to spin at matching speeds. This indirectly implies that the average speed of the front wheels must be the same as the average speed of the rear wheels. On dry pavement, the speed match requirement can lead to increased driveline and tire wear because the front wheels take a longer path than the rear wheels, illustrated in Figure 1.1, which leads to unstable cornering dynamics such as ‘wheel hopping’ or tire skidding [13].



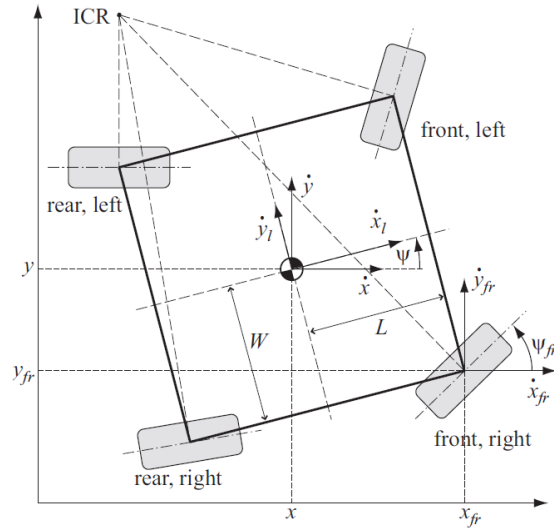
**FIGURE 1.1 - THE FRONT AND REAR WHEEL PATHS OF A TYPICAL PASSENGER VEHICLE [14]**

Until recently, 4WD vehicles have been most common as a selectable setting for low traction environments. Now, many systems use integrated Anti-lock Braking System wheel speed sensors to detect low traction environments. The system will then automatically

engage the 4WD system; once in 4WD, the cornering instabilities will continue, although it is expected that the system will disengage when traction is regained. This can be particularly troublesome when terrain is rapidly changing such as turning from an icy parking lot to a dry road. To overcome these issues, recently developed all-wheel-drive systems, AWD, which are conceptually different than 4WD, are described in the next section.

### **1.3. EMERGENCE AND FEATURES OF ALL WHEEL STEER AND ALL WHEEL DRIVE VEHICLES**

Requirements for better performance and stability, seem to be leading the automotive industry to seek unconventional approaches in the form of power distribution and maneuvering by eventually combining AWD and AWS features on a single vehicle [15]. While automotive designers knew that 4WD could maximize the power exerted at the wheels, at times, using AWS, it was difficult to satisfy Descartes' principle of rigid body motion about the Instantaneous Center of Rotation, ICR [16], [17], which is illustrated in Figure 1.2.



**FIGURE 1.2 - AN ILLUSTRATION DEMONSTRATING THE REQUIREMENTS OF THE ICR [18]**

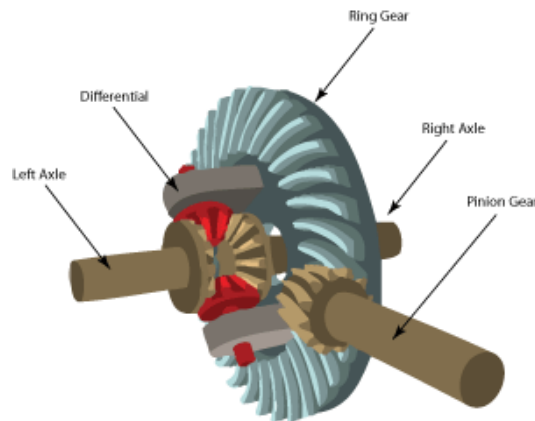
The condition requires the steering angles to be coordinated such that the normals of all wheels intersect at a single point, the ICR. In FWS vehicles, the ICR lies on the projection of the rear rigid axle, which is easily satisfied; this property has been widely known for many years, first described by Darwin (1758), later by Lankenspurger (1810) and finally patented by Ackermann (1818) [19]. However, this condition is rarely met, since the coupled steering linkages, while closely approximate, never maintain true Ackermann angles [6]. As such, sideslip is commonly induced on the tire, diminishing the total tractive ability of the tire; this makes it desirable to mechanically unlink the steered tires.

Since the ICR principle also governs the speeds of various points on the body such that the tangential speed at each wheel,  $V_i$ , is proportional to its radial distances,  $r_i$ , from the ICR,



$$V_i = \left(\frac{r_i}{r_G}\right) V_G, \quad (1.1)$$

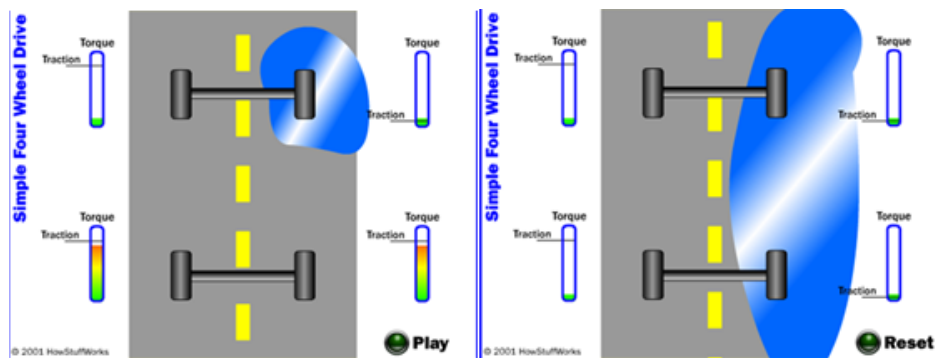
where  $r_G$  is the radial distance from the center of mass to the ICR, and  $V_G$  is the vehicle speed at its center of mass. Therefore, when cornering, there is always some speed difference between the inside wheels and outside wheels, which can cause excessive tire wear. Although early vehicles had wheels that were locked together in a live axle, this approach was soon discarded by introducing differential gearboxes between the wheel axles to account for wheel speed differences during cornering maneuvers. The typical structure of these differential gearboxes is illustrated in Figure 1.3; it allows tires on the same axle to spin at different speeds when powered by a single source, through the use of a planetary gear set.



**FIGURE 1.3 – AN OPEN DIFFERENTIAL WHICH ALLOWS LEFT AND RIGHT AXLES TO ROTATE AT DIFFERENT SPEEDS [20]**

While this simple mechanism handles the majority of vehicle driving demands, it fails when one of the wheels loses traction, because the reaction to the other wheel also disappears. Thus, the tire with no traction will simply spin, while the opposing wheel comes to a standstill.

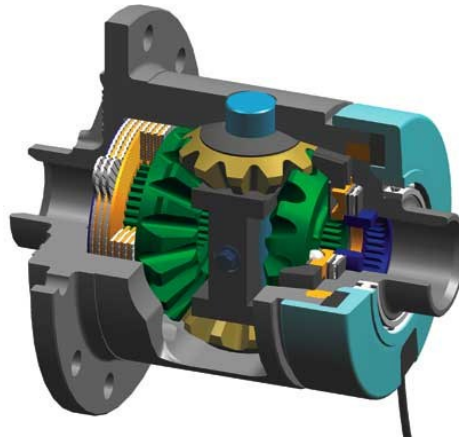
4WD power train technology was meant to address the problem of traction loss by one wheel. Vehicles of this sort have wheels on each axle connected to each other via a differential, and the axles are coupled to each other by a locking transfer case. However, this arrangement also requires that the front and rear differentials spin at the same speed which complicates the ICR condition on cornering events because the rear wheels take a shorter path than the front wheels, previously illustrated in Figure 1.1. Moreover, the properties of the open differential imply that in the worst case scenario for a 4WD vehicle, if one of the front wheels and one of the back wheels lose traction simultaneously, the vehicle will be immobilized, even though two tires still have tractive ability as demonstrated in Figure 1.4. The pitfalls of 4WD vehicles such as instable cornering, increased driveline wear and the inability to power all four wheels under various environments led to the development of all-wheel-drive, AWD, vehicles.



**FIGURE 1.4 – EXAMPLE SCENARIO WHERE 4WD SYSTEMS CAN BE IMMOBILIZED EVEN WITH TRACTION AT TWO WHEELS [21]**

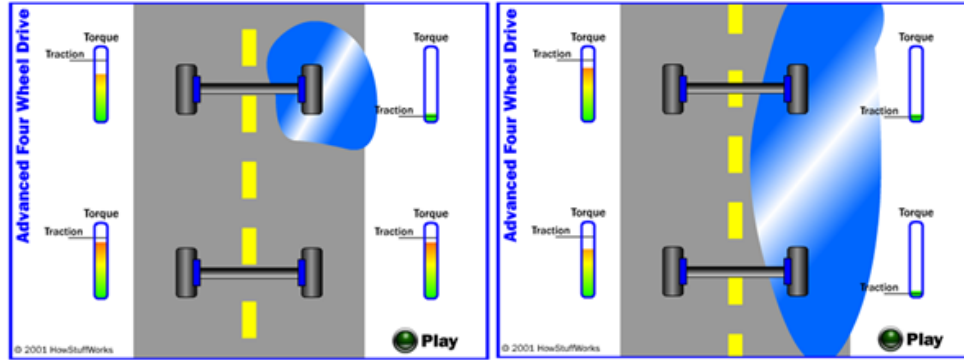
In AWD vehicles, power is optimized about all four wheels, by using limited slip differentials such as TorSen (Torque Sensing) and Haldex technologies that allow power to be automatically diverted on the fly to any wheels with traction. They may use viscous

couplings, pneumatic, mechanical or electronic engagement of clutch packs identified in Figure 1.5, below.



**FIGURE 1.5 – A LIMITED SLIP DIFFERENTIAL THAT USES AN ELECTRONICALLY ACTIVATED CLUTCH PACK (YELLOW, LEFT) [22]**

These approaches minimize the deviation of handling characteristics in cornering events as well as inhibit excessive drive train wear because they decouple the tires in high traction environments such as dry pavement [23]. However, when the vehicle senses that one wheel has lost traction, the differential is engaged and both wheels will spin at the same speed. This allows all of the locomotive torque to be used at the wheel with traction, until the other wheel has recovered as can be seen in Figure 1.6. While this discussion has focused on the left/right traction, the same principle can be applied front/rear, by using a total of three limited slip differentials. Some driving enthusiasts tune the torque distribution to induce combinations of FWD and RWD handling characteristics. These advanced differentials, allow the vehicle to maintain traction in nearly any environment, minimizing drive train and tire wear, ‘tune’ performance characteristics, and eliminate undesirable cornering behavior.



**FIGURE 1.6 – EXAMPLE SCENARIO WHERE AWD SYSTEMS DIVERT POWER FROM WHEELS WITH NO TRACTION TO WHEELS WITH TRACTION [21]**

Steer and acceleration by wire systems are becoming ever more prevalent in the automobile industry. The future of these vehicles is expected to involve the natural symbiosis of electric or gas/electric hybrid vehicles where it becomes viable to steer, brake, and power each wheel with genuine independence through the use of more economically feasible individual electric wheel motors [24].

#### **1.4. WHEELED GROUND ROBOTIC VEHICLES**

As improvements in automobile technologies grew, so too did the interest in autonomous ground robotic vehicles. The first wheeled ground robotic vehicle is generally accepted to be that of Dr. William Grey Walter, known as the *Machina Speculatrix* (a.k.a. Elmer and Elsie), developed in 1948 [25]. Wheeled ground robotic vehicles have a reasonably extensive history with robot platforms such as the JPL Rover [26] and Hilare [27] developed in the 1970's, the Kludge [28], Terragator [29] and Neptune [30] in the 1980's, the Mars Sojourner [31] in the 1990's and the Shrimp [32] in the 2000's, to name a few.

The small scale of ground robotic vehicles allows investigators to tinker with innovative wheel and steering configurations that would not be suitable or practical in the automobile industry. From each of these configurations, new modeling and control methods also have to be developed. Today, wheeled ground robot vehicles can be grouped into two classes: fixed morphology robots and self-reconfigurable robots. Fixed morphology robotic vehicles are characterized by a fixed center of mass, while self-reconfigurable robots can change the positions of their center of mass. Much of the past work on robotic vehicles focused on fixed morphology robots because of their resemblance to common automobiles; developments on self-reconfigurable robots have been very slow, and these vehicles are still at a stage of infancy.

#### **1.4.1. CLASSIFICATION OF FIXED-MORPHOLOGY ROBOTIC VEHICLES**

Campion et al. devised a system for classifying robots upon which kinematic models could be derived and studied [33]. The five classes denote the degree of mobility and steerability,  $(\delta_m, \delta_s)$ , of the robot derived from the wheel types and their configurations. While the following authors assume a 2-D environment, the future of robots will involve modeling and locomotion in 3-D space [32].

Because wheels are the main trait by which robots are classified, it is important to understand their abilities. Multiple authors [16] [33] [34] have devised classes of wheels which can be summarized into four groups: fixed, steered, self-aligning, and omnidirectional. Each of these can then be ranked by their mobility (i.e. whether they are driven or free). For discussion's sake, each wheel is treated as an ideally rigid torus with

only one point of contact, allowing the wheel to be rotated about the point of contact with zero skid, an unlikely assumption in practical applications.

Fixed wheels have only the ability to roll about an axle, and prevent the vehicle from moving laterally with respect to the tire, allowing only longitudinal motion. Steered wheels are those where the rotation about the toe axis of a wheel is controlled in addition to rolling ability. Self-aligning wheels, often called castor wheels, are similar to steered wheels in their ability to rotate, but it is not in a controllable manner. The rotation is governed by a self-aligning torque generated because the axis of rotation does not pass through the point of contact. Their purpose is to add stability to the platform as their orientation is not controlled and generally not driven. Omnidirectional wheels can take on various forms but are physically equivalent to a steered wheel. They offer the same three degrees of freedom including any translation in the  $XY$  plane as well as rotation about the point of contact. Two common forms of the omni-wheel are the ball wheel and the Mecanum [35], or Swedish [33] wheel. An example of a ball wheel is the wheel in a computer mouse (although this is not actuated) or the ball employed on the Ballbot robot at Carnegie-Mellon [36]. These balls are in contact with two non-parallel actuated rollers that rotate the ball about any axis in the  $XY$  plane. The Mecanum wheel consists of a disk of rollers that are oriented off axis and has the ability to move in both the lateral and longitudinal direction as the wheel rotates, shown in Figure 1.7.



**FIGURE 1.7 – A MECANUM WHEEL [37]**


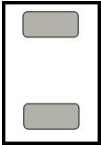

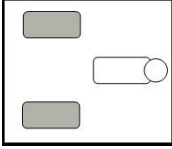

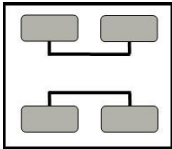
In 2-D planar space, rigid body vehicles have three possible degrees of freedom including longitudinal translation, lateral translation, and body orientation. Based on these three possible degrees of freedom, a number of wheel configurations can control one, two, or even all three modes of travel. These configurations can be classified based on the number of controllable degrees of freedom they provide the vehicle.

Wheeled robots offering only one controllable degree of freedom offer little maneuverability and have not been studied extensively as they must often be constrained through other means such as rail or wire. One example of a robot like this is the retrieval cart for many mini-load Automatic Storage and Retrieval Systems (AS/RS), which rides on a rail to locate the retrieval mechanism in front of the appropriate column of product.

Robotic vehicles offering two degrees of freedom were the typical launching point for today's research and still appear in significant numbers as they are easily constructed and controlled while offering considerable maneuverability. Robots that make up this category can be further divided into three categories: differentially driven, car-like robots, and XY vehicles.

Differentially driven vehicles have two wheels parallel to each other who share a common axle projection. The wheels are powered at different speeds and directions. The speed differential causes the robot to change orientation, while the average speed of the wheels controls the robots longitudinal velocity. Table 1.1 shows a summary of example vehicles that are differentially driven. The Segway personal transporter has only two wheels, creating an unstable platform. It drives the two wheels differentially to balance the robot as well as propel it. The Nomad Scout is driven by similar means but uses a third castor wheel to create a stable platform. The iRobot ATRV Jr. uses four wheels; the left wheels are coupled to each other and the right wheels are coupled to each other. These ‘skid steer’ vehicles are mechanically robust and easy to control but do not adhere to the previous requirement of the ICR; there must be some allowance for wheel skid causing excessive tire wear or ground disturbance.

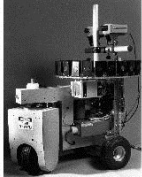
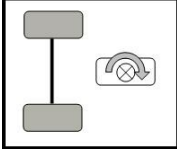
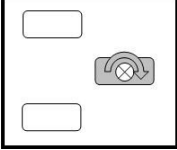

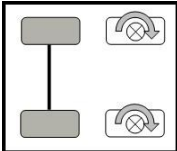
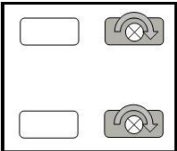
**TABLE 1.1 - DIFFERENTIALLY DRIVEN ROBOTS**

Number of Wheels	Picture	Diagram	Example
2			Segway Personal Transporter [38]
3			Nomad Scout [39]
4			iRobot ATRV Jr. [40]



Car-like robots are so named because of their likeness to passenger vehicles; examples of these robots are shown in Table 1.2. They have one or a set of steered wheels and one or a set of fixed wheels. Either the fixed wheels or steered wheels may be driven as might be found in a front wheel drive or rear wheel drive vehicle. If there are multiple steered wheels, they are often mechanically coupled through a linkage to maintain appropriate Ackermann steering geometry. Robots of this type are very abundant and developments on these platforms are more easily adopted by the automotive industry because of the hardware similarities.

**TABLE 1.2 - CAR-LIKE ROBOTS**


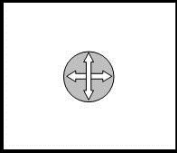

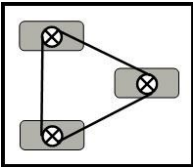
Number of Wheels	Picture	Diagram	Example
3		 Or 	NEPTUNE [41]
4		 Or 	Stanley [42]

The final group of robots with two degrees of freedom capability is the *XY* Robots. They differ from the other subgroups because they control their *XY* motion but have no control over their orientation. The Ballbot is an *XY* robot developed at Carnegie Mellon University that balances on one omnidirectional wheel which is actuated by two non-parallel driven rollers [36]. Other robots exist, such as the Denning MRV-2, with multiple

wheels whose steering mechanisms are coupled through means such as a belt or chain. One motor controls the steering orientation for all wheels while another motor drives all wheels at the same speed; this mechanism is called ‘Synchro-drive’.

Robotic vehicles offering three degrees of freedom were the next evolution in robotic ground vehicles. They make use of multiple actuators to control all three degrees of freedom offered by planar space. The increased controllable degrees of freedom allow greater maneuverability; the robot can orient and position itself through obstacles and spaces that might be otherwise impossible for a similar robot. These robots can be further divided into two categories: AWS and omnidirectional.


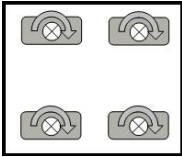

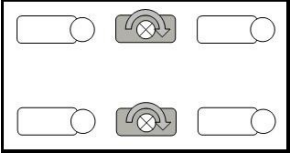
**TABLE 1.3 – XY ROBOTS**

Number of Wheels	Picture	Diagram	Example
1			Ballbot [36]
3			Denning MRV-2 [43]

AWS robots are the most direct solution to enable the third degree of freedom and can be distinguished because all of the driven wheels are also steered. They offer the same two degrees of freedom as car-like robots with the addition of lateral translation. The wheel steering angle may be mechanically coupled, as in many AWS passenger vehicles, or

independently steered. In the latter case, consistency with the Instantaneous Center of Rotation concept must be maintained to ensure there is no wheel slippage. The HERMES-III robot at Oak Ridge National Labs has two independently steered and driven wheels to adjust its orientation as well as position. The BIBOT-1 platform at North Dakota State University has four independently steered and independently driven in wheel motors.


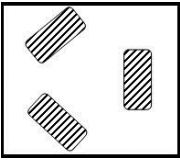

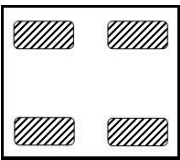
**TABLE 1.4 - ALL-WHEEL-STEER ROBOTS**

Number of Wheels	Picture	Diagram	Example
4			BIBOT-1
6			HERMES-III [44]

The final group of robots is the omnidirectional robots. They make use of the Mecanum or Swedish wheels and the motions of the robot depend on the coordination of the wheels. It can be argued that the steering response time of an omnidirectional vehicle is faster than that of an all-wheel-steered vehicle because the wheels do not have to be reoriented to change the velocity vector. The kinematics of omnidirectional wheels are more complex than those of traditionally steered vehicles because the degrees of freedom are cross-linked among all wheels. One example, the Rovio robot, uses three omnidirectional wheels with rollers that are perpendicular to the primary axis of rotation.

The URANUS robot, developed at Carnegie Mellon University, uses wheels with rollers that are misaligned 45° to the primary axis of rotation. The velocity vector generated at each wheel depends on the speed and direction of all other wheels. Although omnidirectional robots have been identified that allow control over all three degrees of motion in a planar environment, this class requires that all four wheels have traction in all directions to be able to control all three degrees of freedom; the actuators are critically interlinked and can prove to be unreliable or immobilized in rugged terrain where traction may be varying. Thus, for these applications, focus should be applied to the AWS robot class.

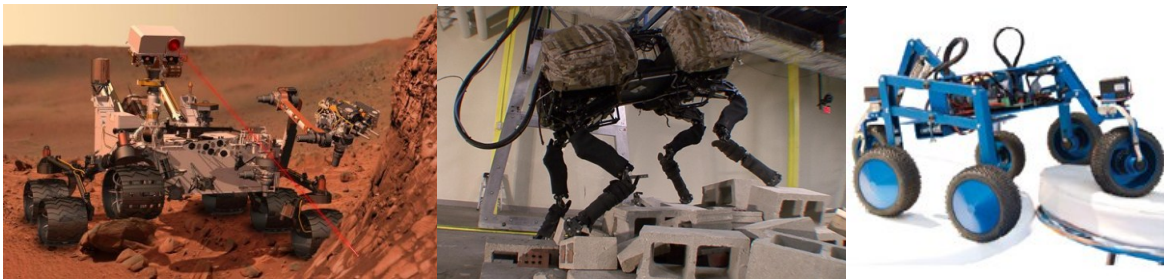
**TABLE 1.5 - OMNIDIRECTIONAL ROBOTS**

Number of Wheels	Picture	Diagram	Example
3			Rovio [45]
4			URANUS [46]

#### 1.4.2. SELF-RECONFIGURABLE ROBOTS

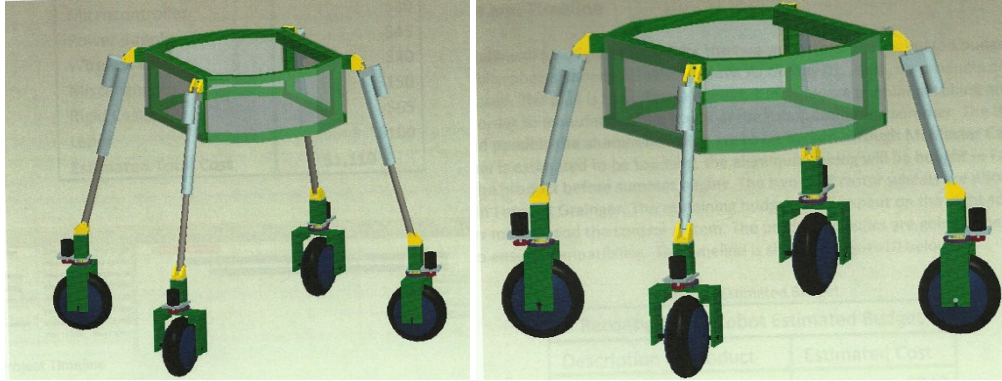
Each of the fixed morphology robot configurations presented in the previous subsection is suitable for particular terrains. As such, there does not exist one ultimate robot for all environments. Often, the construction of these robots sacrifice maneuverability for reduced complexity and control hardware. In other cases, designers find the additional degree of freedom necessary for the robot to navigate its native

environment. In an effort to develop highly maneuverable and rugged robots that can traverse all types of possible terrains, robotics research effort is now trending towards self-reconfigurable robots [47] [48] [49]. The popular effort has been focusing on articulated robots which use revolute articulated suspensions like the Mars Rover *Curiosity* [50], the *Big Dog* [51] and the *Shrimp* [52], shown in Figure 1.8.



**FIGURE 1.8 – SEVERAL ARTICULATING SELF-RECONFIGURABLE ROBOTS [53] [51] [54]**

Unfortunately, articulated reconfigurable robots with prismatic articulated suspensions have not been studied equally as well, and no justifiable reason has ever been published against such robots. This research is part of a long term effort to study prismatic self-reconfigurable robots such as the BIBOT-2 shown in Figure 1.9. It is hoped that prismatically articulated reconfigurable robots will display better maneuvering performance on highly irregular and non-flat terrains. The future of these robots will include non-planar environments in which the wheels or other articulators must be able to traverse obstacles and other difficult terrain.



**FIGURE 1.9 – THE BIBOT-2 SELF-RECONFIGURABLE ROBOT EXTENDED (LEFT) AND RETRACTED (RIGHT)**

### **1.5. STATEMENT OF THE RESEARCH PROBLEM**

This thesis is one of the steps in the major research program on controlling the motion of reconfigurable robotic vehicles equipped with prismatic wheel supports. It addresses the problem of satisfying the kinematic rigid body constraint for optimally controlling the path-tracking performance of AWS/AWD robotic vehicles. The research intends to develop a simple way of incorporating kinematic constraints in the control algorithm; as such, it has four objectives as follows:

1. To develop a comprehensive method of defining the kinematic rigid body constraints for wheeled ground vehicles using path geometry information.
2. To develop a way of incorporating the rigid kinematic constraints in the control algorithm for autonomous ground robotic vehicles.
3. To verify that the methods defined in the first two objectives above can be applied for optimally controlling the AWS/AWD vehicle.
4. To document the results.

The thesis is divided into seven chapters. The next chapter presents a dynamic model for the intended reconfigurable robotic vehicle along with its governing kinematic constraints; the chapter closes by reducing this model and its constraints to an AWS/AWD robotic model. Chapter 3 presents an approach that was developed to define the kinematic constraints using path geometry and incorporating these constraints in the robot control algorithm; the chapter closes by presenting a simple control algorithm that constrains the robot to track its path using the path geometrical data. Verification of the applicability of the developed kinematic constraint equations along with the proposed controller is discussed in Chapters 4 through 6. Numerical simulation results are discussed in Chapter 4 while experimental results that were obtained using an in-house AWS/AWD robotic vehicle are discussed in Chapters 5 and 6. Concluding remarks and recommendations for extension of the proposed algorithm to the general articulated AWS/AWD robot with prismatic wheel supports are outlined in Chapter 7.

## 2. DYNAMIC MODELING OF WHEELED ROBOTS

The most comprehensive study for kinematic models of wheeled ground vehicles was constructed by Patrick Muir and Charles Neuman [34]. Models for all classes of robotic vehicle discussed in the previous section are based on this Muir-Neuman study. This project also used the Muir-Neuman approach to develop a kinematic model applicable to a four wheeled AWD/AWS robotic vehicle with a prismatically articulated wheel configuration. This model was later reduced to be applicable to a fixed morphology AWD/AWS vehicle for the purpose of developing and validating the vehicle control algorithm. This chapter presents the development of such a model using standard methods of analytical dynamics [16] [55] [56] [57] [58]. Section 2.1 presents both the necessary kinematic constraints and the dynamic model for a four wheel reconfigurable articulated robot. The model developed in Section 2.1 is then reduced to be applicable to a four wheeled AWS/AWD fixed suspension robot.

### 2.1. MODELING OF A FOUR WHEELED RECONFIGURABLE ROBOT WITH A PRISMATICALLY ARTICULATED SUSPENSION

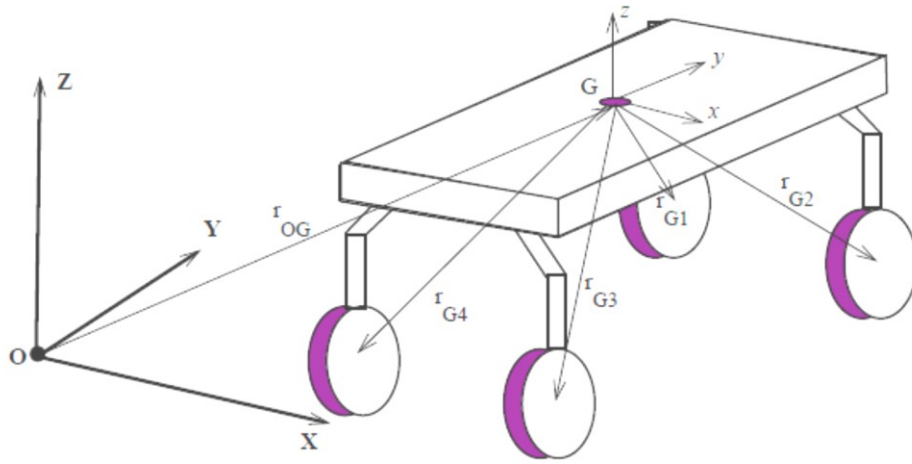
#### 2.1.1. KINEMATIC CONSTRAINTS

The model developed in this section considers a general case of robotic motion in 3-D environments involving paths,  $\vec{p}(X, Y, Z)$ , defined in the 3-D inertial frame  $XYZ$ . As a generic platform, the vehicle under study is assumed to have four wheels, not necessarily arranged symmetrically, each free to assume any coordinate position relative to the vehicle's center of mass,  $G$ ; Figure 2.1 shows the framework for this development. The



motion of the vehicle is defined by the path,  $\overrightarrow{r_{OG}}$ , tracked by its center of mass  $G$ , as well as the velocity,  $\overrightarrow{V_G}$ , and acceleration,  $\overrightarrow{a_G}$ , of this center of mass where

$$\overrightarrow{r_{OG}} \triangleq \begin{pmatrix} X_G \\ Y_G \\ Z_G \end{pmatrix}, \quad \overrightarrow{V_{OG}} \triangleq \begin{pmatrix} V_{GX} \\ V_{GY} \\ V_{GZ} \end{pmatrix}, \quad \overrightarrow{a_{OG}} \triangleq \begin{pmatrix} a_{GX} \\ a_{GY} \\ a_{GZ} \end{pmatrix}. \quad (2.1)$$



**FIGURE 2.1 - THE CONFIGURATION THE COORDINATE SYSTEM FOR A ROBOTIC VEHICLE**

The wheels support the vehicle body through actuated prismatic joints; hence, the position of each wheel relative to the vehicle's center of mass can be changed independently. It is assumed that frame  $xyz$  is attached to the vehicle's center of mass,  $G$ , where the  $y$ -axis is in the direction of the vehicle heading. The Euler angles of the moving frame  $xyz$  with respect to the global frame  $XYZ$ , are  $\theta_x$ ,  $\theta_y$ , and  $\theta_z$ ; therefore, the relative position  $(x', y', z')$  of any part of the vehicle can be expressed in the  $XYZ$  frame by using a linear coordinate transformation,  $J_R$ , such that [55] [56] [57]

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \overline{r_{OG}} + J_R \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}. \quad (2.2)$$

Assuming sequential  $X - Y - Z$  rotations, the matrix  $J_R$  is defined as

$$J_R \triangleq \begin{bmatrix} C(\theta_y)C(\theta_z) & S(\theta_x)S(\theta_y)C(\theta_z) - C(\theta_x)S(\theta_z) & S(\theta_x)S(\theta_z) + C(\theta_x)S(\theta_y)C(\theta_z) \\ C(\theta_y)S(\theta_z) & C(\theta_x)C(\theta_z) + S(\theta_x)S(\theta_y)S(\theta_z) & C(\theta_x)S(\theta_y)S(\theta_z) - S(\theta_x)C(\theta_z) \\ -S(\theta_y) & S(\theta_x)C(\theta_y) & C(\theta_x)C(\theta_y) \end{bmatrix}, \quad (2.3)$$

where  $C(x)$  corresponds to  $\cos(x)$ , and  $S(x)$  corresponds to  $\sin(x)$ . Elementary mechanics show that the respective velocity vector,  $\overline{V}_i$ , and acceleration vector,  $\overline{a}_i$ , of any point,  $\overline{r_{xyz}}$ , on the vehicle as seen from the origin of the  $XYZ$  inertial frame are

$$\overline{V}_i = \overline{V}_G + \overline{\Omega} \times \overline{r_{xyz}} + J_R \dot{\overline{r_{xyz}}}, \quad (2.4)$$

$$\overline{a}_i = \overline{a}_G + \overline{\Omega} \times (\overline{\Omega} \times \overline{r_{xyz}}) + \dot{\overline{\Omega}} \times \overline{r_{xyz}} + 2\overline{\Omega} \times \dot{\overline{r_{xyz}}} + J_R \ddot{\overline{r_{xyz}}}, \quad (2.5)$$

where  $\overline{\Omega}$  is the rotational velocity vector of the frame  $xyz$  in frame  $XYZ$  defined as

$$\overline{\Omega} \triangleq \begin{pmatrix} \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \end{pmatrix}. \quad (2.6)$$

Since the motion of the vehicle depends on its wheels, then each wheel,  $i$ , must move at velocity,  $\overline{V}_i$ , and acceleration,  $\overline{a}_i$ , as in equations (2.4) and (2.5) for the vehicle to track a particular path,  $\overline{r_{OG}}$ , at the prescribed velocity,  $\overline{V}_G$ , and acceleration,  $\overline{a}_G$ , without slippage.

Most often, robot navigation control problems are concerned with satisfying velocity requirements only, however as a rigid body, the vehicle must also satisfy the instantaneous center of rotation while in motion; i.e. there must always be a single and unique point in space,  $C$ , such that

$$\vec{V}_i = \vec{\Omega} \times (\vec{r}_{C|i}), \quad (2.7)$$

for all wheels, where  $\vec{r}_{C|i}$  is the relative position vector of wheel  $i$  from the instantaneous center of rotation,  $C$ . The locus of this point is also known to be the centrode of the motion. This is the one constraint that all robotic vehicle control algorithms must satisfy; its complexity increases with the number of controllable parameters of the wheel. Certain wheel configurations such as pivoting axles, as in a child's toy wagon or the vehicle in Figure 3.2, satisfy this constraint easily, however, the varying arrangement of wheels on reconfigurable vehicles make it extremely difficult to fulfill this condition.

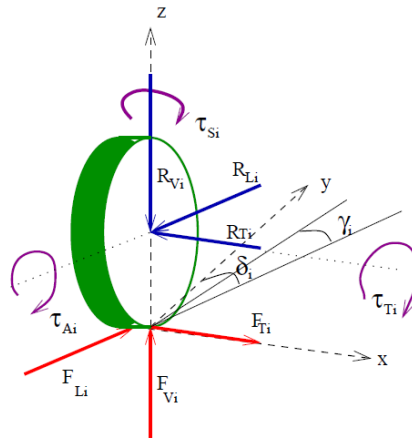
Equation (1.1) shows that the wheel velocity,  $\vec{V}_i$ , depends on the location of the wheel relative to the center of mass,  $\vec{r}_{i|G}$ , the rate of change of this location,  $\dot{\vec{r}}_{i|G}$ , and the vehicle orientation with respect to the  $XYZ$  frame,  $J_R$ , the rate of change of the path direction,  $\vec{\Omega}$ , and the desired vehicle velocity,  $\vec{V}_G$ . Since the vehicle orientation and the rate of change of the path direction are determined by the path structure and cannot be controlled by the robot, the motion control system can only coordinate the wheel position, the rate of change of this wheel position and the wheel velocity in a variety of ways to meet the desired vehicle velocity.

### 2.1.2. THE DYNAMIC MODEL

In formulating the dynamic model for the robot, the general dynamic case of a robot negotiating a corner is considered. If each wheel is turning at angle  $\delta_i$  to complete the turn, then it is acted upon by a total of six forces and three torques:

1. Ground Forces - The longitudinal force,  $F_{Li}$ , lateral force,  $F_{Ti}$ , and vertical force,  $F_{Vi}$
2. Vehicle Body Reactions - The longitudinal force,  $R_{Li}$ , lateral force,  $R_{Ti}$ , and vertical force,  $R_{Vi}$
3. Wheel Torques - The steering torque,  $\tau_{Si}$ , the traction torque,  $\tau_{Ti}$ , and the wheel self-aligning torque,  $\tau_{Ai}$

As the robot negotiates a turn, each wheel experiences a side slip angle,  $\gamma_i$ . Figure 2.2 shows the free body diagram of the wheel for this general case.



**FIGURE 2.2 – FORCES AND TORQUES ACTING ON A WHEEL**

If the wheels are assumed to roll without slip, remaining in contact with the ground and  $r_w$  is the wheel radius, then

$$F_{Li} = R_{Li}, \quad F_{Ti} = R_{Ti}, \quad F_{Vi} = R_{Vi}, \quad (2.8)$$

and the rolling speed,  $V_i$ , of the wheel becomes

$$V_i = r_w \dot{\phi}_i, \quad (2.9)$$

where  $\dot{\phi}_i$  is its rotational speed. If  $I_{wr}$  and  $D_{wr}$  are the wheel moment of inertia and the viscous damping of the wheel against rotation, then the traction torque is related to the wheel speed through

$$\tau_{Ti} = F_{Ti} r_w = \frac{I_{wr}}{r_w} \dot{V}_i + \frac{D_{wr}}{r_w} V_i. \quad (2.10)$$

The lateral tire force,  $F_{Ti}$ , is related to the vertical ground force,  $F_{Vi}$ , by

$$F_{Ti} = F_{Vi} \mu(\gamma_i), \quad (2.11)$$

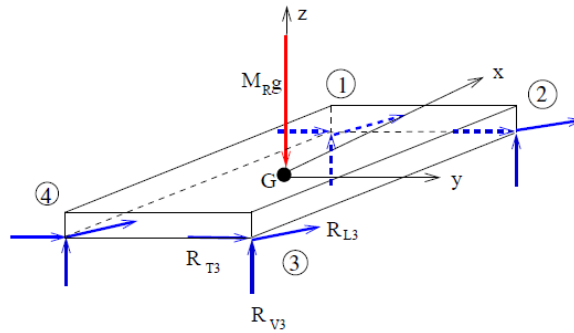
where  $\mu(\gamma_i)$  is the friction coefficient for a particular side slip angle,  $\gamma_i$ . The friction coefficient can be estimated using either the extended Burckhardt formula [59]

$$\mu(\gamma_i) = [C_1 - C_1 \exp(-C_2 \gamma_i) - C_3 \gamma_i] \exp(-C_2 \gamma_i V_i), \quad (2.12)$$

or the Pacejka formula [60]

$$\mu(\gamma_i) = C_1 \sin(C_2 \tan^{-1}(C_3 \gamma_i - C_4 (C_3 \gamma_i - \tan^{-1}(C_3 \gamma_i)))), \quad (2.13)$$

with  $C_1, C_2, C_3$  and  $C_4$  as constants that depend on the tire material and tire-ground contact conditions.



**FIGURE 2.3 – FORCES ACTING ON THE VEHICLE BODY**

The self-aligning torque,  $\tau_{Ai}$ , and the lateral forces on each wheel are related as

$$\tau_{Ai} = F_{Ti}r_{si}, \quad (2.14)$$

with  $r_{si}$  being the height from the ground to the wheel strut support. If  $I_{ws}$ ,  $D_{ws}$  and  $K_{ws}$  are the respective steering moment of inertia, damping coefficient and stiffness, then the steering torque is related to the steering angle,  $\delta_i$ , through

$$\tau_{si} = I_{ws}\ddot{\delta}_i + D_{ws}\dot{\delta}_i + K_{ws}\delta_i. \quad (2.15)$$

As illustrated in Figure 2.3, the vehicle body is acted on by a total of four triples of wheel reactions ( $R_{Li}$ ,  $R_{Ti}$ ,  $R_{Vi}$ ) for  $i = 1, 2, \dots, 4$ , and its own weight,  $M_R g$ , where  $M_R$  is the mass of the robot body. These forces satisfy Newton's equation of linear motion

$$\sum \vec{F} = M_R \ddot{\vec{r}}_{OG}, \quad (2.16)$$

as

$$\begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} = M_R \begin{bmatrix} \ddot{X}_G \\ \ddot{Y}_G \\ \ddot{Z}_G \end{bmatrix}, \quad (2.17)$$

where

$$\begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} \triangleq \begin{bmatrix} 0 \\ 0 \\ -M_R g \end{bmatrix} + J_R \sum_{i=1}^4 \begin{bmatrix} R_{Li} \sin(\delta_i) + R_{Ti} \sin(\delta_i) \\ R_{Li} \cos(\delta_i) + R_{Ti} \cos(\delta_i) \\ R_{Vi} \end{bmatrix}. \quad (2.18)$$

Additionally, these forces also satisfy the corresponding equation of rotational motion as

$$\begin{bmatrix} \tau_{\theta X} \\ \tau_{\theta Y} \\ \tau_{\theta Z} \end{bmatrix} \triangleq \sum_{i=1}^4 [\vec{r}_{i|G} \times (\vec{R}_{Ti} + \vec{R}_{Li} + \vec{R}_{Vi})] = I_R \dot{\vec{\Omega}}, \quad (2.19)$$

where  $\tau_{\theta X}$ ,  $\tau_{\theta Y}$  and  $\tau_{\theta Z}$  are components of the rotation torque on the body, and  $I_R$  is the moment of inertia of the robot body about its center of mass. Therefore, the generalized non-conservative force vector acting on the vehicle can be defined as

$$Q = [F_X, F_Y, F_Z, \tau_{\theta X}, \tau_{\theta Y}, \tau_{\theta Z}, R_{L1}, R_{T1}, R_{V1}, R_{L2}, R_{T2}, R_{V2}, \dots \dots R_{L3}, R_{T3}, R_{V3}, R_{L4}, R_{T4}, R_{V4}, \tau_{T1}, \tau_{T2}, \tau_{T3}, \tau_{T4}, \tau_{S1}, \tau_{S2}, \tau_{S3}, \tau_{S4}]^T. \quad (2.20)$$

Also, it is possible to define the generalized coordinate vector,  $q \in \mathbb{R}^{26}$ , as

$$q = [X_G, Y_G, Z_G, \theta_x, \theta_y, \theta_z, x_1, y_1, z_1, x_2, y_2, z_2, \dots \dots x_3, y_3, z_3, x_4, y_4, z_4, \varphi_1, \varphi_2, \varphi_3, \varphi_4, \delta_1, \delta_2, \delta_3, \delta_4]^T, \quad (2.21)$$

which leads to a Lagrangian,  $\mathcal{L}$ , of the form

$$\mathcal{L} = \frac{1}{2} \left[ M_R (V_{GX}^2 + V_{GY}^2 + V_{GZ}^2) + I_R (\dot{\theta}_x^2 + \dot{\theta}_y^2 + \dot{\theta}_z^2) - 2M_R g \Delta Z \dots \dots + \sum_{i=1}^4 \left( I_{wr} \dot{\varphi}_i^2 + I_{ws} \dot{\delta}_i^2 - K_{ws} \delta_i^2 - 2 \int_0^{\delta_i} D_{ws} \delta \dot{\delta} d\delta \right) \right], \quad (2.22)$$

where  $\Delta Z$  is the vertical displacement of the robot body from a predefined neutral position when the vehicle is on level ground. The standard Euler-Lagrange equation

$$\frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{q}_j} \right) - \frac{\partial \mathcal{L}}{\partial q_j} = Q_j, \quad j = 1, 2, \dots, 26, \quad (2.23)$$

for this system, results in 26 equations of motion that can be expressed in state space form as

$$\dot{x} = \mathcal{F}(x, u), \quad (2.24)$$

where the control vector,  $u \in \mathbb{R}^{20}$ , comprises of all actuated wheel forces and torques as

$$u = [F_{ex1}, F_{ey1}, F_{ez1}, F_{ex2}, F_{ey2}, F_{ez2}, F_{ex3}, F_{ey3}, F_{ez3}, \dots \dots F_{ex4}, F_{ey4}, F_{ez4}, \tau_{T1}, \tau_{T2}, \tau_{T3}, \tau_{T4}, \tau_{S1}, \tau_{S2}, \tau_{S3}, \tau_{S4}]^T. \quad (2.25)$$

The wheel extension forces,  $F_{exi}, F_{eyi}, F_{ezi}$ ,  $i = 1, 2, \dots, 4$ , are responsible for extending and retracting the wheel from position  $G$ . The state vector,  $x \in \mathbb{R}^{26}$ , is the same as the generalized coordinate vector,  $q$ .

At this stage, only routine steps are required to derive the dynamic system (2.24). This is a standard model structure for a  $26 \times 20$  nonlinear dynamic system that can be controlled using a variety of nonlinear control methods such as feedback linearization, sliding mode control, and many others, likely at a high computational cost because of its size. Despite its size, this model offers one advantage that it enables the robot to reconfigure itself into various stable configurations consistent with the path profile,  $\vec{p}(X, Y, Z)$ , by dynamically changing the position of the center of gravity as long as the kinematic constraints in equation (2.7) are satisfied. With the ability to dynamically control the position of its center of gravity, it is hoped that the robotic vehicle can maneuver in many difficult terrains as biological systems do.

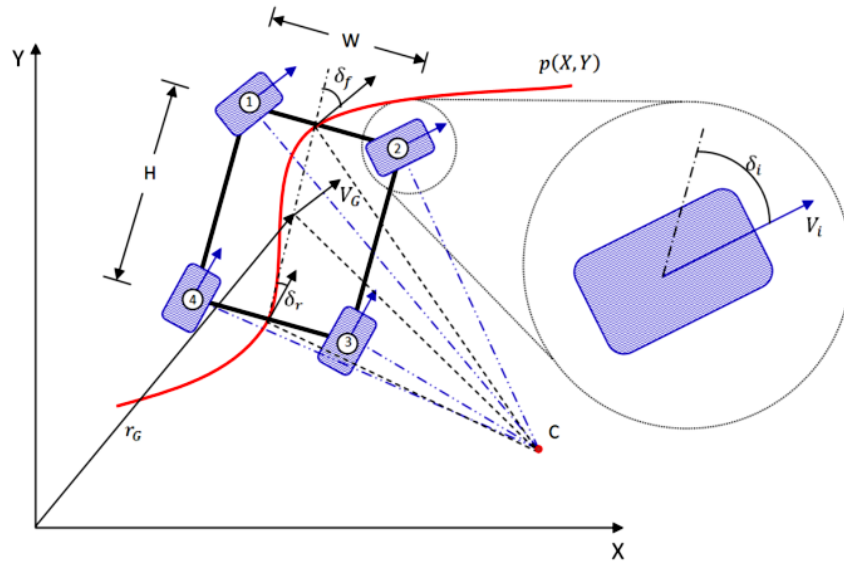
## 2.2. MODEL REDUCTION FOR A FIXED SUSPENSION 4WS/4WD ROBOT

### 2.2.1. MODEL REDUCTION APPROACH

Out of complexity of the model for the reconfigurable robot system discussed in the previous section, this research was focused on finding ways to approach the problem by first studying a four wheeled AWS/AWD robot system with non-articulated wheels, otherwise referred to as a fixed suspension. This section simplifies the model in Section



2.1.2 to be applicable for a fixed suspension AWS/AWD vehicle; parts of this section have been published by the author and coworkers in [61] [62]. It was assumed that the robot is traversing a flat plane and the vehicle is a rigid body. Under this assumption, there always exists a single point,  $C$ , at a position,  $\vec{r}_{OC}$ , in the inertial frame,  $XY$ , serving as its instantaneous center of rotation, as illustrated in Figure 2.4. Each wheel,  $i$ , has a speed,  $V_i$ , at an angle,  $\delta_i$ , to the vehicle direction, where  $\vec{p}(X, Y)$  is the path vector function. The ultimate goal of the trajectory control problem is to coordinate individual wheel velocities,  $\vec{V}_i$ , allowing the robot to track the desired trajectory,  $\vec{r}_{OG}$ , at a prescribed center of mass velocity,  $\vec{V}_G$ , without wheel slippage.



**FIGURE 2.4 - PARAMETERS FOR A TYPICAL FOUR-WHEEL-STEERED VEHICLE**

Therefore, the vehicle motion can be described as a pure rotation about the instantaneous center of rotation,  $C$ , at any instant in time satisfying

$$\vec{V}_G = \vec{\Omega} \times (\vec{r}_{OC} - \vec{r}_{OG}), \quad (2.26)$$

where  $\bar{\Omega}$  is the angular velocity of the vehicle about  $C$ . The velocities at the wheels can then be derived using

$$\bar{V}_i = \begin{bmatrix} V_i \sin(\theta - \delta_i) \\ V_i \cos(\theta - \delta_i) \end{bmatrix} = \bar{\Omega} \times [\bar{r}_{OC} - (\bar{r}_{OG} + \bar{r}_{Gi})], \quad i = 1, 2, \dots, 4, \quad (2.27)$$

where  $\bar{r}_{Gi}$  is the position vector of wheel  $i$  from the vehicle center of mass, and  $\theta$  is the angular orientation of the vehicle in the  $XY$  reference frame. In this case, the centre of vehicle motion can go through any point in the  $XY$  plane relative to the vehicle. As will be shown later, the fact that point  $C$  can be anywhere in the  $XY$  plane relative to the vehicle still makes this problem difficult to solve in the same way as the general reconfigurable vehicle discussed in the previous section.

### 2.2.2. THE DYNAMIC MODEL

To articulate the dynamic equations for the fixed suspension robot, it is assumed that the wheels roll without slipping, with each wheel speed,  $V_i$ , given by equation (2.9). Each wheel is acted on by a total of four forces and three torques:

1. The longitudinal,  $F_{li}$ , and lateral,  $F_{Ti}$ , ground forces
2. The longitudinal,  $R_{li}$ , and lateral,  $R_{Ti}$ , vehicle support reactions
3. The steering torque,  $\tau_{Si}$ , the traction torque,  $\tau_{Ti}$ , and the wheel self-aligning torque,  $\tau_{Ai}$  (which is taken to be negligible).

Therefore, the generalized coordinate vector  $q \in \mathbb{R}^{11}$  for the robotic system reduces from the one in equation (2.21) to

$$q = [X_G, Y_G, \theta, \varphi_1, \varphi_2, \varphi_3, \varphi_4, \delta_1, \delta_2, \delta_3, \delta_4]^T, \quad (2.28)$$

and the corresponding generalized force vector as also reduces from equation (2.20) to

$$Q = [F_x, F_y, \tau_\theta, \tau_{T1}, \tau_{T2}, \tau_{T3}, \tau_{T4}, \tau_{S1}, \tau_{S2}, \tau_{S3}, \tau_{S4}]^T. \quad (2.29)$$

The potential energy due to the steering damping and the vehicle suspension system can be ignored, which reduces equation (2.22) of the Lagrangian,  $\mathcal{L}$ , to

$$\mathcal{L} = \frac{1}{2} \left[ M_R V_{GX}^2 + M_R V_{GY}^2 + I_R \omega^2 + \sum_{i=1}^4 (I_{wr} \dot{\phi}_i^2 + I_{ws} \dot{\delta}_i^2) \right]. \quad (2.30)$$

When the standard Euler-Lagrange equation (2.23) is employed on (2.30), the resulting equations of motion can be expressed as

$$\begin{bmatrix} \dot{X}_G \\ V_{GX} \\ \dot{Y}_G \\ V_{GY} \\ \dot{\theta} \\ \dot{\omega} \\ \dot{\delta}_1 \\ \dot{\beta}_1 \\ \dot{\delta}_2 \\ \dot{\beta}_2 \\ \dot{\delta}_3 \\ \dot{\beta}_3 \\ \dot{\delta}_4 \\ \dot{\beta}_4 \end{bmatrix} = \begin{bmatrix} V_{GX} \\ F_x/M_R \\ V_{GY} \\ F_y/M_R \\ \omega \\ \tau_\theta/I_R \\ \beta_1 \\ \frac{\tau_{S1} - f_s}{I_{ws}} \\ \beta_2 \\ \frac{\tau_{S2} - f_s}{I_{ws}} \\ \beta_3 \\ \frac{\tau_{S3} - f_s}{I_{ws}} \\ \beta_4 \\ \frac{\tau_{S4} - f_s}{I_{ws}} \end{bmatrix}, \quad (2.31)$$

where  $f_s$  is the steering friction torque at the tire.

In the state space form of equation (2.24), the control vector,  $u \in \mathbb{R}^8$ , involves the wheel torques only as

$$u = [\tau_{T1}, \tau_{T2}, \tau_{T3}, \tau_{T4}, \tau_{S1}, \tau_{S2}, \tau_{S3}, \tau_{S4}]^T, \quad (2.32)$$

and the state vector,  $x \in \mathbb{R}^{14}$ , is defined as

$$x = [X_G, V_{GX}, Y_G, V_{GY}, \theta, \omega, \delta_1, \beta_1, \delta_2, \beta_2, \delta_3, \beta_3, \delta_4, \beta_4]^T, \quad (2.33)$$

where  $\beta_i = \dot{\delta}_i$ . The balance of longitudinal forces on each wheel satisfy the no slip condition, such that

$$R_{Li} = F_{Li} = \frac{1}{r_w} \left( \tau_{Ti} - \frac{I_{wr} \dot{V}_i}{r_w} \right) \quad i = 1, 2, \dots, 4, \quad (2.34)$$

where  $V_i$ ,  $i = 1, 2, \dots, 4$ , are the wheel velocities and the lateral forces satisfy the self-aligning torque assumption.

With this model, the control problem reduces to that of determining the steering torque,  $\tau_{Si}$ , and the traction torque,  $\tau_{Ti}$ , for each individual wheel such that the vehicle tracks the desired path. The next chapter focuses on solving this control problem for the fixed suspension vehicle with four wheeled independently driven and independently steered vehicle.

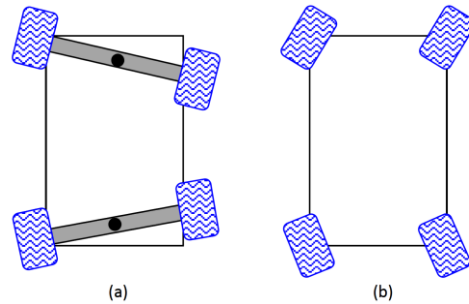
### 3. DEVELOPMENT OF AN OPTIMAL CONTROL ALGORITHM FOR AWS/AWD PATH TRACKING

#### 3.1. LITERATURE REVIEW

As stated in Chapter 1, the high maneuverability characteristics of four steered wheels have increasingly attracted interest in these vehicles for a long time. Studies on four wheel steering vehicles started in the mid-eighties by the automotive industry [8] [10] [63] [64]. These vehicles are known to be highly overactuated, which provided greater control over the motion of the robot but also make them difficult to control due to the demand of the rigid body kinematic constraint, equation (2.27), discussed in Chapter 2. Overactuation provides more robustness in the system by creating redundancy, since failure of one actuator does not disable the system. One notable instance of this was the Mars rover *Spirit*, which had one of six wheels mechanically fail but was still able to traverse the sandy terrain because of the redundant drive system [65].

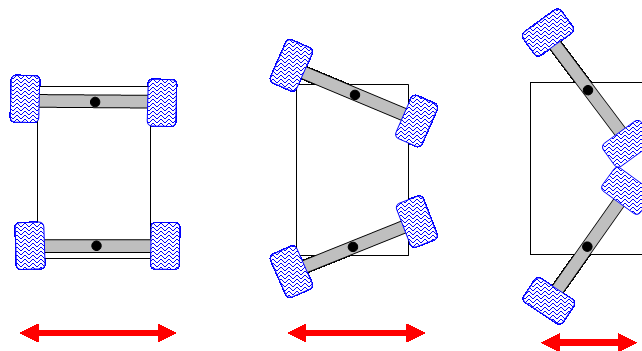
In robotics, the maneuverability problem was addressed by development of two classes of robots, namely the AWS/AWD robots and differentially driven omnidirectional vehicles equipped with omni-wheels [66] [67] [68]. The latter class successfully introduced highly maneuverable omnidirectional robots using three steered wheels with negligible slippage [68] [69] [70] [71]. However, the former class was not successful, particularly due to the large number of parameters that need to be coordinated by the controller.

Today, AWS robots are implemented in one of the two ways as illustrated in Figure 3.1: robots in which the steered wheels are mechanically coupled as illustrated in Figure 3.1 (a), and robots that have independently steered wheels illustrated in Figure 3.1 (b).



**FIGURE 3.1 - TWO TYPICAL CONFIGURATIONS OF FOUR-WHEEL-STEERED VEHICLES**

Structurally, kinematic constraints for robots that have mechanically coupled pairs of steered wheels are guaranteed to be satisfied, since only two steering angles are required and the wheel speeds are in well-defined ratios. Many significant results in robotics have been reported based on vehicles with this wheel configuration [72] [73] [74] [75] [76].



**FIGURE 3.2 - THE LATERAL STABILITY OF TYPE (A) VEHICLES IS COMPROMISED IN TIGHT CORNERS**

An important shortcoming of these vehicles is that the distance from wheel to mass center varies with steering angle. Consequently, the base of the vehicle decreases laterally as the steering radius decreases. During highly dynamic maneuvers requiring tight turning radii, the lateral stability of the vehicle may be compromised to the point of vehicle inversion. For a type (a) vehicle to accomplish a zero turn radius maneuver, the lateral base dimension reduces to zero.

It has been more challenging to control type (b) vehicles with independently steered wheels while maintaining rigid body kinematics constraints along a variety of trajectories. Control challenges posed by independently steered wheels on AWS/AWD vehicles have persistently been difficult to address. Alternative steering approaches that have been proposed to handle these systems focus on alleviating the effects of wheel slippage by convenient robot structures such as circular or square [77] [78] and sometimes by using omnidirectional wheels [55] [79] [80] [81] [82]. Other proposed methods limit the steering angles to small values that reduce wheel slippage [83] [84].

The easiest way to locate the ICR for AWS/AWD vehicles has been by constraining it to the perpendicular bisector of the robot's longitudinal centerline. The rear wheel angles become a reflection of the front wheels. However, this approach limits the robot's flexibility to track different path geometries. In effort to avoid this inflexibility, other approaches use predictive methods to estimate the vehicle yaw rate [85] [86] [87] [88] [89] and use estimated yaw rates in establishing the center of rotation position. However, since the vehicle yaw rate depends on its speed, this approach of estimating the yaw rate using the vehicle speeds and use the estimated value to determine future speeds of the vehicle is one

way causal, with the latter being a result of the former; determining the wheel velocities based on estimated yaw rates may be prone to considerable errors. In addition to this possibility of introducing errors, the resulting control will inevitably be relatively complex due to the need for yaw measurement instrumentation.

This research proposed a steering approach that aligns the front and rear axles to always track the path center. The next section describes the proposed approach and an example use in path tracking. Some of the information in these sections has been published by the author and coworkers in references [62] [61].

## 3.2. THE PROPOSED APPROACH

### 3.2.1. DEFINITION OF THE KINEMATIC CONSTRAINTS

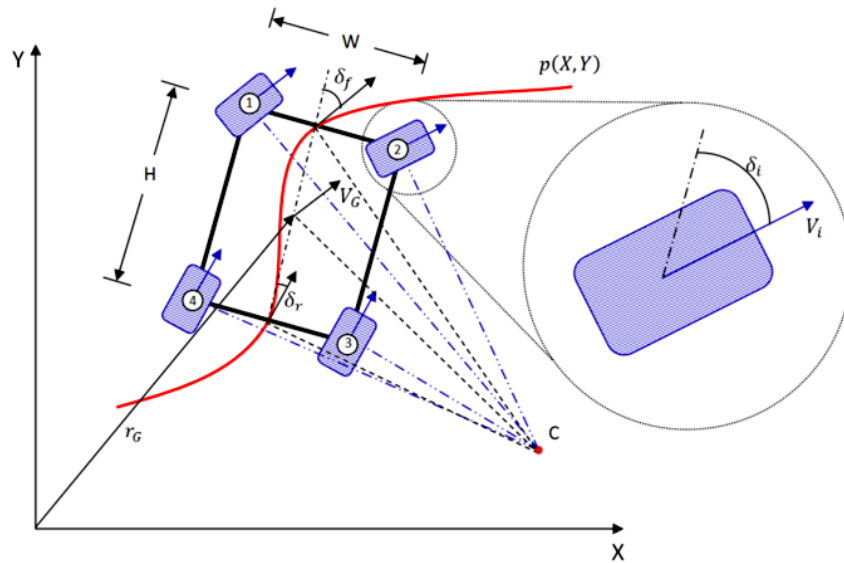
It was shown in Chapter 2 that a vehicle with the framework of Figure 3.3 satisfies the rigid body kinematic constraint only if the wheel velocities,  $\vec{V}_i$ , and steering angles,  $\delta_i$ , satisfy

$$\vec{V}_i = \begin{bmatrix} V_i \sin(\theta - \delta_i) \\ V_i \cos(\theta - \delta_i) \end{bmatrix} = \vec{\Omega} \times \vec{r}_{OG}, \quad i = 1, 2, \dots, 4, \quad (3.1)$$

where  $\vec{r}_{Gi}$  is the position vector of wheel  $i$  from the vehicle center of mass, and  $\theta$  is the angular orientation of the vehicle in the  $XY$  reference frame. The velocity,  $\vec{V}_i$ , of each wheel can be described by the wheel speed,  $V_i$ , and its steering angle,  $\delta_i$ , consistent with equation (3.1) with an instantaneous center of rotation, located at  $C$ . Since the center of rotation, can be at any position in the  $XY$  plane relative to the vehicle, the kinematic constraint problem requires proper values for the wheel speeds and angles that satisfy equation (3.1).



Two possible approaches for the vehicle to track the 2-D path,  $\vec{p}(X, Y)$  are to either track the path of its mass center or track the paths of its wheels. Both methods allow the wheels to adopt any velocities (speeds and steering angles) as long as the rigid body kinematic constraint in equation (2.26) is maintained.



**FIGURE 3.3 - PARAMETERS FOR A TYPICAL FOUR WHEEL STEERED VEHICLE**

Tracking the mass center allows an envelope about the mass center to be easily calculated and predicted. The inclusion envelope is useful in avoidance of positive obstacles (i.e. those that protrude above the plane of travel); however, its primary disadvantage is that it requires much more path area than the alternative method when avoiding negative obstacles (i.e. crevices).

The wheel tracking approach allows the vehicle to avoid wheel path intersection with negative obstacles. The primary advantage is that it enables the vehicle to maneuver with fewer constraints, since it allows the vehicle body to protrude over negative obstacles while ensuring that the wheels do not encounter them thereby allowing for a tighter turn.

However, it can pilot the vehicle into positive obstacles that are close to but not on the wheel path by the same principle. This is especially evident when negotiating corners, since there is no control on the path of the mass center.

An ideal method would be to couple these methods to exploit advantages of both, however, the scope of this research was limited to the wheel tracking approach only. In this approach, restrictions are applied that the deviation of  $G$  from its path remains within acceptable bounds,  $\lambda$ , such that

$$\|\overline{r_{OC}} - \vec{p}(X, Y)\| \leq \lambda, \quad (3.2)$$

where  $\|\cdot\|$  denotes the Euclidean norm. Therefore, the enclosed angles,  $\sigma$ , at the corners of all allowable traversable paths must satisfy

$$\lambda \geq \frac{1}{2}H \tan \frac{1}{2}\sigma. \quad (3.3)$$

If  $V_f, V_G$ , and  $V_r$  are the respective front, center, and rear speeds of the vehicle along the longitudinal centerline in directions  $\delta_f, \delta_G$ , and  $\delta_r$  as shown in Figure 3.3, then kinematic and rigid body constraints on the centerline require

$$V_f \cos \delta_f = V_G \cos \delta_G = V_r \cos \delta_r, \quad (3.4)$$

and

$$V_f \sin \delta_f - V_r \sin \delta_r = 2(V_G \sin \delta_G - V_r \sin \delta_r) = H, \quad (3.5)$$

where  $\omega$  is the yaw rate of the vehicle as it negotiates a corner. Similarly, kinematic and rigid body constraints along the front and rear axles require

$$V_1 \sin \delta_1 = V_2 \sin \delta_2 = V_f \sin \delta_f, \quad (3.6)$$

$$V_3 \sin \delta_3 = V_4 \sin \delta_4 = V_r \sin \delta_r, \quad (3.7)$$

$$V_1 \cos \delta_1 - V_2 \cos \delta_2 = 2(V_f \cos \delta_f - V_2 \sin \delta_2) = W\omega, \quad (3.8)$$

$$V_4 \cos \delta_4 - V_3 \cos \delta_3 = 2(V_r \cos \delta_r - V_3 \sin \delta_3) = W\omega. \quad (3.9)$$

The research seeks to find a controller that determines the wheel speeds,  $V_i$ ,  $i = 1, 2, \dots, 4$ , and the drive angles  $\delta_i$ ,  $i = 1, 2, \dots, 4$ , by using the path geometry information ( $\delta_f$  and  $\delta_r$ ) to satisfy the desired vehicle speed,  $V_G$ . By constraining all angles  $\delta_i$ ,  $i = f, r, G, 1, 2, \dots, 4$ , such that  $|\delta_i| \leq \frac{\pi}{2}$ , then from (3.20) and standard rigid body geometric constraints, it was found that the rigid body constraints reduce to the following wheel constraints<sup>1</sup>

$$\delta_1 = \cot^{-1} \left( \cot \delta_f + \frac{W}{2H} \cot \delta_f [\tan \delta_f - \tan \delta_r] \right), \quad (3.10)$$

$$V_1 = \frac{V_G \tan \delta_f \csc \delta_1}{\sqrt{1 + \frac{1}{4} (\tan \delta_f + \tan \delta_r)^2}}, \quad (3.11)$$

$$\delta_2 = \cot^{-1} \left( \cot \delta_f - \frac{W}{2H} \cot \delta_f [\tan \delta_f - \tan \delta_r] \right), \quad (3.12)$$

$$V_2 = \frac{V_G \tan \delta_f \csc \delta_2}{\sqrt{1 + \frac{1}{4} (\tan \delta_f + \tan \delta_r)^2}}, \quad (3.13)$$

$$\delta_3 = \cot^{-1} \left( \cot \delta_r - \frac{W}{2H} \cot \delta_r [\tan \delta_f - \tan \delta_r] \right), \quad (3.14)$$

---

<sup>1</sup> Note that in these equations,  $\lim_{\delta_x, \delta_y \rightarrow 0} \left[ \frac{\sin \delta_x}{\sin \delta_y} \sec \delta_x \right] = 1$

$$V_3 = \frac{V_G \tan \delta_r \csc \delta_3}{\sqrt{1 + \frac{1}{4} (\tan \delta_f + \tan \delta_r)^2}} \quad (3.15)$$

$$\delta_4 = \cot^{-1} \left( \cot \delta_r + \frac{W}{2H} \cot \delta_r [\tan \delta_f - \tan \delta_r] \right), \quad (3.16)$$

$$V_4 = \frac{V_G \tan \delta_r \csc \delta_4}{\sqrt{1 + \frac{1}{4} (\tan \delta_f + \tan \delta_r)^2}} \quad (3.17)$$

Ideally, these constraint equations are similar to those of [87] and [88], however, these constraints are defined using available real time information on the vehicle speed,  $V_G$ , and the path geometry  $(\delta_f, \delta_r)$ , only instead of estimating the vehicle yaw rate,  $\omega$ , of which itself depends on these wheel speeds and angles. To implement these constraints, the controller must know the path angles,  $\delta_f$  and  $\delta_r$ . This information could be established using a path following sensor setup such as painted line or buried wire on predefined paths. The general and more versatile way is to use navigation sensors to determine the path direction angle at the front axle,  $\delta_f$ . However, similar corresponding sensors for determining the path direction angle at the rear axle,  $\delta_r$ , are not viable. If the orientation angle of the vehicle,  $\theta$ , and the path gradient angle,  $\rho$ , in the inertial frame,  $XY$ , are known where

$$\rho = \tan^{-1} \left( \frac{dY}{dX} \right), \quad (3.18)$$

the path direction angles at the front axle,  $\delta_f$ , and at the rear axle,  $\delta_r$ , of the vehicle satisfy

$$\delta_f = \frac{\pi}{2} - (\theta + \rho_f), \quad (3.19)$$

$$\delta_r = \frac{\pi}{2} - (\theta + \rho_r), \quad (3.20)$$

where  $\rho_f$  and  $\rho_r$  are the path gradient angles at the front and rear axles, respectively.

Therefore to determine  $\delta_r$ , a finite path history function,  $M$ , must be created to monitor and store the front path gradient angle,  $\rho_f$ , such that

$$M(X_f, Y_f) = \rho_f, \quad (3.21)$$

$$\rho_r = M(X_f - H\sin\theta, Y_f - H\cos\theta), \quad (3.22)$$

at any front axle path coordinates,  $(X_f, Y_f)$ , and vehicle orientation,  $\theta$ . The path gradient angle at the rear axle will be used to determine the rear axle orientation,  $\delta_r$ , from (3.20).

This requires the vehicle to start in a straight line motion until the rear of the vehicle reaches the starting point of the front, or the memory function be primed with some initial path. With the path angles known, it is straightforward to see that the  $X$  and  $Y$  velocity components at the vehicles center of mass become

$$V_{GX} = \frac{V_G(\tan\delta_f + \tan\delta_r)}{\sqrt{4 + (\tan\delta_f + \tan\delta_r)^2}} \quad (3.23)$$

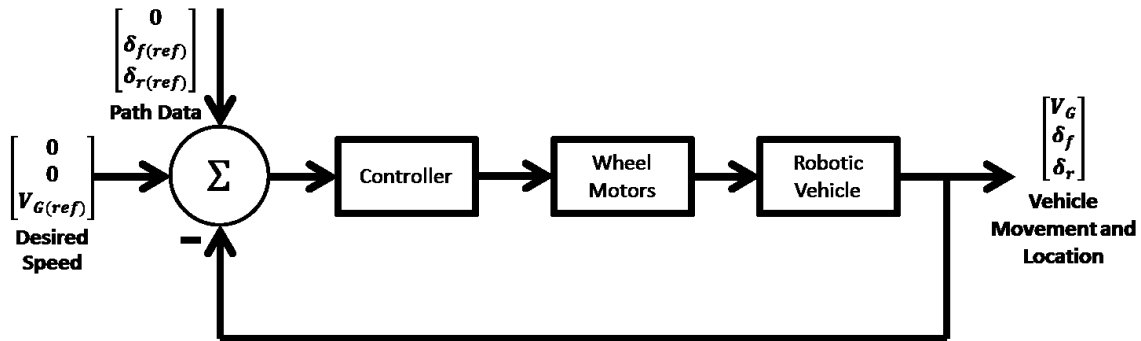
and

$$V_{GY} = \frac{2V_G}{\sqrt{4 + (\tan\delta_f + \tan\delta_r)^2}} \quad (3.24)$$

### 3.2.2. INCORPORATION OF THE KINEMATIC CONSTRAINTS INTO THE PATH TRACKING CONTROLLER

From the model developed in Chapter 2 and the wheel drive constraints developed in equations (3.10)-(3.17), the path tracking control problem reduces to finding steering

and traction wheel torques such that the vehicle moves at the desired speed,  $V_G$ , and it remains aligned to the path,  $\vec{p}(X, Y)$ , according to its geometrical parameters,  $\delta_f$  and  $\delta_r$ . In a feedback control structure, the desired vehicle speed and the path geometry define the controller reference points,  $V_{G(ref)}$ ,  $\delta_{f(ref)}$  and  $\delta_{r(ref)}$ . Figure 3.4 shows a simplified structure for the resulting feedback control system.



**FIGURE 3.4 - A LAYOUT OF THE PROPOSED CONTROL SCHEME**

One advantage of this controller is that it does not require knowledge of the current location of the vehicle  $(X, Y)$ . If the vehicle speed,  $\overline{V_G}$ , is prescribed, then only the path parameter  $\delta_f$  is required as a continuous input, as  $\delta_r$  is determined via the path history function and (3.20). The path direction,  $\delta_f$ , can be obtained through path sensors such as sonar or laser range finders in the navigation instrumentation.

Due to the complexity and size of the controller that results from (2.31), the approach proposed for this problem was to divide and decentralize the control problem into four simpler individual wheel controllers. The robot is then treated as an arrangement of independent bodies centered at the wheel locations. The robot mass is assumed to be evenly distributed among the wheels; such that (2.34) yields

$$\left(\frac{M_R}{4} + m_w\right) \dot{V}_i = \frac{1}{r_w} \left( \tau_{Ti} - \frac{I_{wr} \dot{V}_i}{r_w} \right), \quad (3.25)$$

where  $m_w$  is the mass of the wheel. This, along with the corresponding steering equations for each wheel, leads to the individual body dynamics at the wheel as

$$\begin{bmatrix} \dot{V}_i \\ \dot{\delta}_i \\ \dot{\beta}_i \end{bmatrix} = \begin{bmatrix} 4r_w \tau_{Ti} / (M_R r_w^2 + 4(I_{wr} + m_w r_w^2)) \\ \beta_i \\ \tau_{Si} - D_{ws} \beta_k / I_{ws} \end{bmatrix}, \quad (3.26)$$

where  $D_{ws}$  is the steering damping coefficient. Now, the task becomes finding  $\tau_{Ti}$  and  $\tau_{Si}$  such that the wheel motion satisfies the reference velocity,  $V_{i(ref)}$ , and steer angle,  $\delta_{i(ref)}$ , compatible with  $[V_{G(ref)}, \delta_{f(ref)}, \delta_{r(ref)}]^T$ . This approach made it possible to apply any modern multivariable feedback control algorithm to solve this problem. Although various control algorithms were numerically experimented with and found to work well as anticipated, the experimental part of the research focused on using a simple decoupled proportional controller, discretized as

$$\begin{bmatrix} \tau_{Ti}[j+1] \\ \tau_{Si}[j+1] \end{bmatrix} = \begin{bmatrix} {}^P K_T [V_{i(ref)} - V_i[j]] \\ {}^P K_S [\delta_{i(ref)} - \delta_i[j]] \end{bmatrix}, \quad (3.27)$$

where  ${}^P K_T$  and  ${}^P K_S$  are the traction and steering proportional gains, respectively, at time step,  $j$ . This controller is simple, not highly dependent on model accuracy, and easy to program in a microcontroller. Variables used are easily measurable (i.e. the velocity at each wheel) negating the need for advanced sensors such as yaw rate gyroscopes and estimation computation. It should be noted that the linear form of (3.26) can be uncontrollable, therefore linearizing methods may not be suitable for this case.

## 4. SIMULATION RESULTS

To demonstrate the viability of the control algorithm proposed in Chapter 3, a numerical simulation was first carried out. Once the algorithm was deemed viable, it was pursued on the experimental platform, the BIBOT-1, as will be described in the next chapters. The simulated vehicle parameters are shown in Table 4.1.

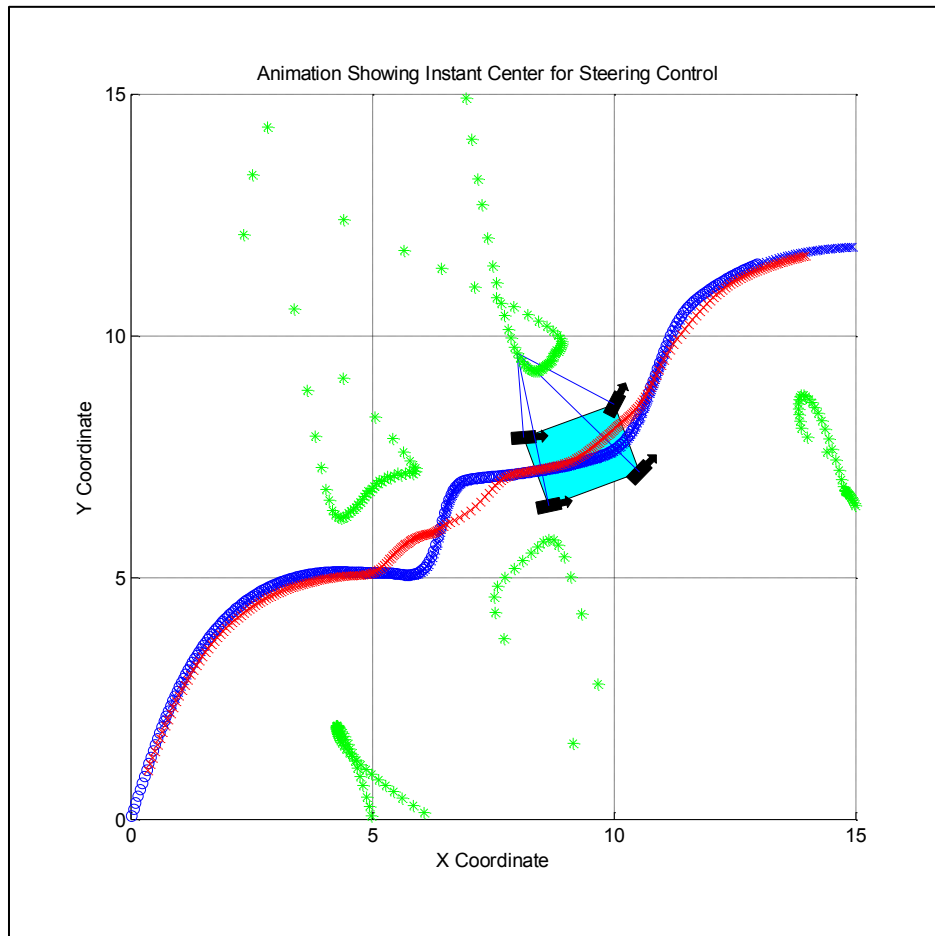
**TABLE 4.1 - VEHICLE SIMULATION PARAMETERS**

Quantity	Value	Units
$M_R$	50.0	$Kg$
$I_R$	5.5	$Kg \cdot m^2$
$H$	1.0	$m$
$W$	0.75	$m$
$I_{wr}$	0.025	$Kg \cdot m^2$
$I_{ws}$	0.009	$Kg \cdot m^2$
$r_w$	0.085	$m$
$m_w$	3.5	$Kg$
$D_{ws}$	0.5	$\frac{N \cdot m}{rad/s}$

The simulation was carried out in two phases. The first phase was to examine the numerical accuracy of the proposed decentralized control algorithm on generating the desired wheel torques to track the desired paths; Section 4.1 describes how this phase was implemented and its results. The second phase examined the performance of the two point path tracking steering approach to navigate different paths in comparison with other steering methods; results of this phase are presented in Section 4.2.



Each simulation is animated using a 'patch()' function, available in MATLAB, for each wheel and the vehicle body. The vehicle is modeled as a rectangle with a wheelbase and wheel track similar to those of the BIBOT-1. The coordinates for the vertices of the shapes are set up such that rotation and translation transformation matrices can be used on the vertices. By continually drawing, erasing, transforming and drawing again, an animation is created that is crude in detail but represents the motion of the vehicle and wheels along the sample path quite well. Figure 4.1 shows a snapshot of one animation which includes the centrede, as well as the path of the center of gravity,  $G$ .



**FIGURE 4.1 – MATLAB SIMULATION FOR AN ARBITRARY PATH SHOWING CENTRODES AND CG PATH**

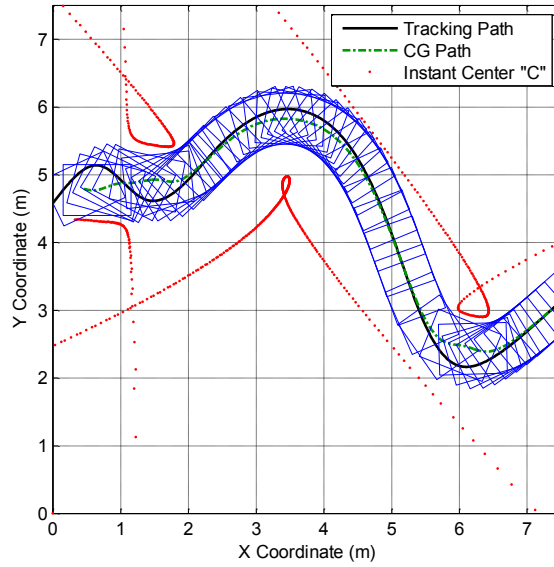
#### 4.1. NUMERICAL VALIDATION OF THE DECENTRALIZED CONTROLLER

The controller in (3.27) was numerically simulated in the MATLAB environment; the code for this simulation can be found in Appendix Section A.1. Different paths were simulated with different vehicle speeds ranging from  $0.5 \text{ m/s}$  to  $2.0 \text{ m/s}$  consistent with the speed limits of BIBOT-1, and the controller gains were tuned to  $^K P_T = 120$  and  $^K P_S = 10$ . In all simulations, the front axle path angle,  $\delta_f$ , was estimated using the path gradient while rear axle path angle,  $\delta_r$ , was calculated using the memory function (3.21)-(3.22).

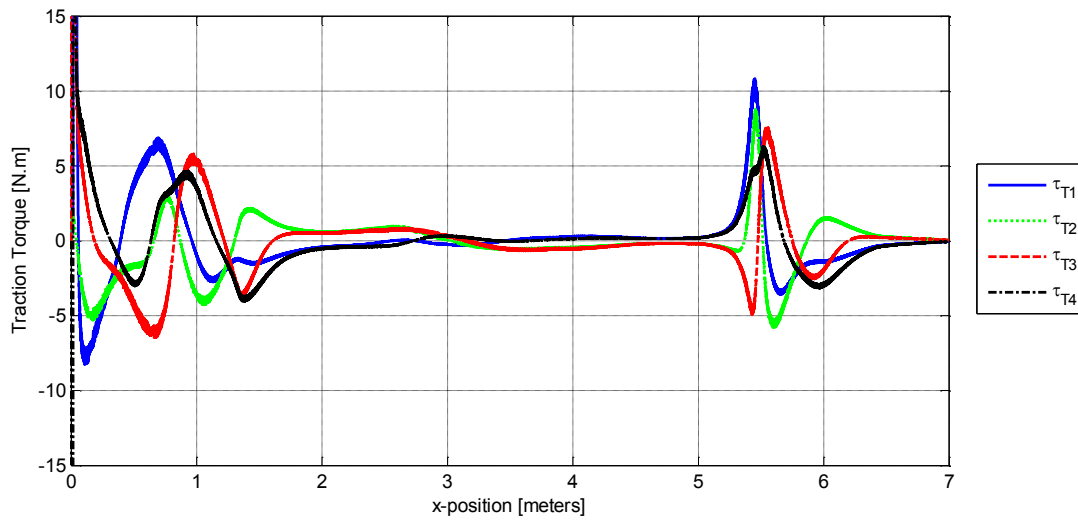
Figure 4.2 - Figure 4.4 show sample results obtained from this simulation on a single path at a sampling interval of  $1 \text{ ms}$  using a vehicle speed  $V_G = 1 \text{ m/s}$ . Figure 4.2 shows the simulated trajectory with time lapse representations of the vehicle motion. The centrodes are also displayed in red; this trajectory changes from near left to infinity-left and then jumps to infinity-right and transitions to near right. The infinite asymptote represents pure translation where all wheel speeds,  $V_i$ , as well as angles,  $\delta_i$ , are equal. The path of the vehicle CG is also shown in green; it is noted that this path does not match the tracked path, which is useful when avoiding negative obstacles but can be a hindrance in the presence of positive obstacles.

Wheel torques that were generated by the controller for this sample simulation are shown in Figure 4.3 and Figure 4.4. In all simulated paths, the magnitudes of the traction wheel torques increased whenever wheel speed changed rapidly, but was near zero when wheel speed was constant. Steering torques show similar curves, and the largest deviation from zero was observed as the vehicle was negotiating sharp curves. Both steering and

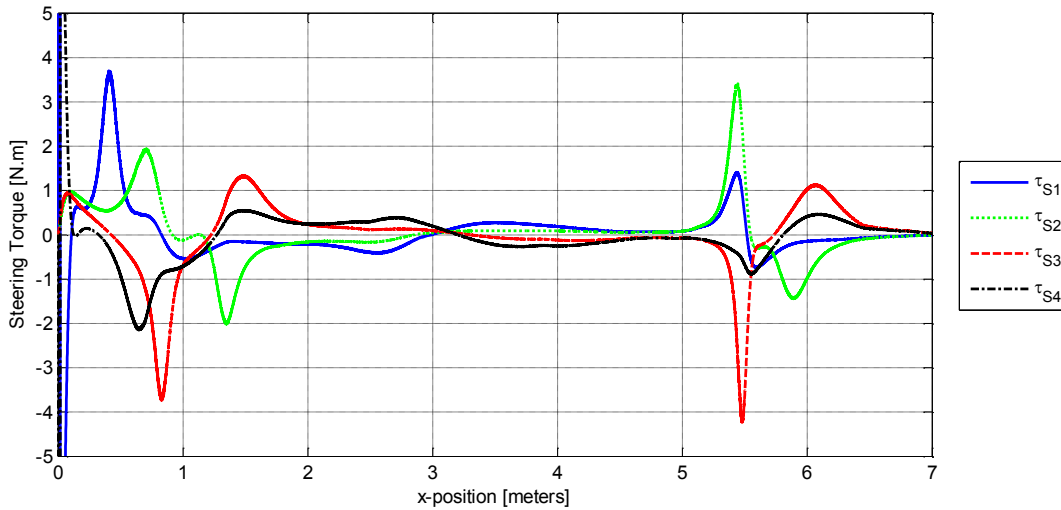
traction torques for all simulations examined were within reasonable limits of  $\pm 15 \text{ N} \cdot \text{m}$  consistent with torque limits for both steering and traction motors on BIBOT-1.



**FIGURE 4.2 - SIMULATION TRAJECTORY**



**FIGURE 4.3 - SIMULATED TRACTION TORQUE**

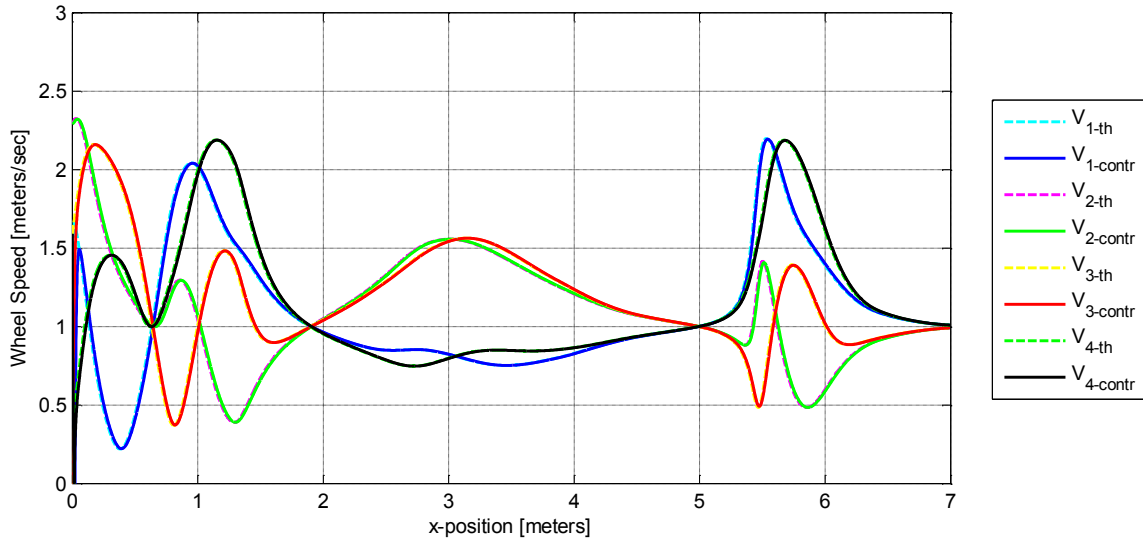


**FIGURE 4.4 - SIMULATED STEERING TORQUE**

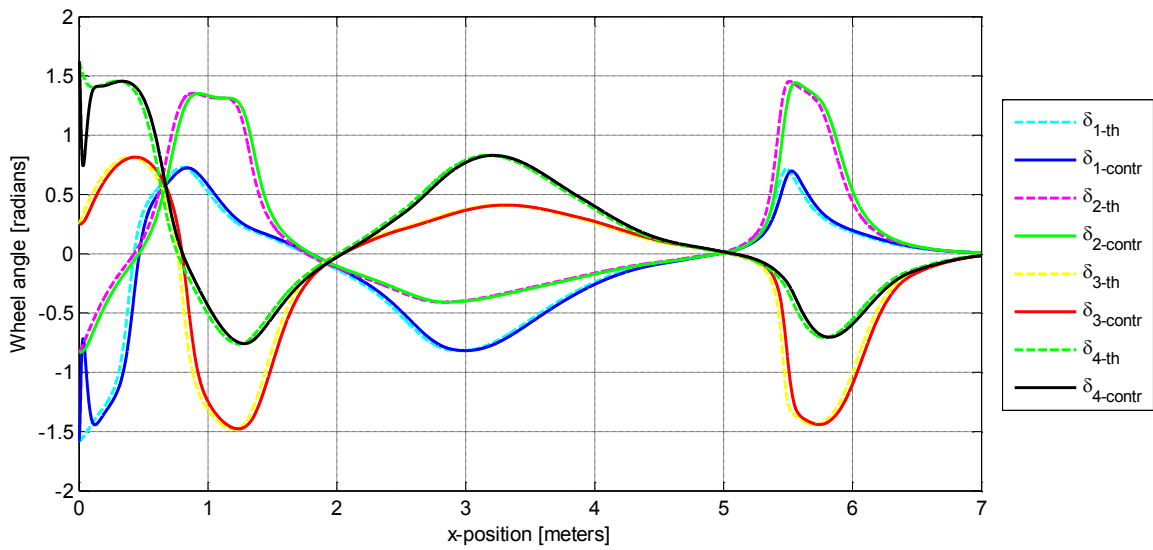
Figure 4.5 shows the controlled wheel speeds compared to the expected theoretical speeds that would satisfy the instantaneous center constraint. As can be seen from this figure, the controlled speeds are virtually equal to the expected theoretical speeds. In a sample of 6 simulated paths, the recorded maximum deviation between the controlled speeds and the theoretical speeds was negligibly small at  $\pm 0.001 \text{ m/s}$ , which happened only when the robot was making sharp turns at high speeds. Because of the effectiveness of the controller, the results in Figure 4.5 appear as though only one curve is representing each wheel when actually both the theoretical and controlled curves are displayed.

Similarly, Figure 4.6 compares the controlled steering angle to the expected angle. The controlled steering angles shown in Figure 4.6 are in close agreement with the expected theoretical angles. There was however a time lag between the theoretical angles and the controlled angles whenever the robot was assumed to be negotiating turns; in these cases where the theoretical angles would appear to be ahead of the controlled angles

by one sample interval of  $1\text{ ms}$ , which made the deviation between the controlled angles and theoretical angles to be a bit high with a maximum recorded value of  $\pm 0.1\text{ radians}$ .



**FIGURE 4.5 - SIMULATED AND THEORETICAL WHEEL SPEED**



**FIGURE 4.6 - SIMULATED AND THEORETICAL WHEEL ANGLE**

## 4.2. SIMULATED PATH TRACKING RESULTS

The second simulation was carried out to evaluate the performance of the proposed two point tracking steering algorithm in tracking arbitrary paths and compare this performance with other path tracking steering algorithms. In these simulations, the proposed algorithm, **4WS Front and Rear tracking (4FR)**, was compared with three other methods: conventional **2 Wheel Front axle steering (2WF)**, **4WS Front axle tracking with rear axle Mirror (4FM)**, **4WS Center of Gravity tracking (4CG)**. The key features of each steering algorithm are summarized in Table 4.2. The MATLAB code used in this simulation is listed in Appendix Sections A.2-A.6.

Although passenger vehicle regulatory agencies have standard testing paths such as the j-turn, fishhook, and double lane change paths [90], there are no such standard paths for wheeled mobile robots. Researchers working on mobile robots tend to design arbitrary paths that will best demonstrate the concept that they are trying to propose. Some paths are designed for highly dynamic maneuvers such as lane change or skid pads, others try to show obstacle avoidance through highly cluttered segments. Several authors have used paths such as a straight line, a sinusoid [91], a quarter circle [92], arbitrary curves [93] or zig-zags [94] of varying shapes and sizes. Gupta used two routes, the Zig-Zag and U-turn, in previous work on the BIBOT-1 [95]. Based on these paths, three new paths were created, scaled appropriately to the experimental space,  $6m \times 9m$ . These included the Zig-Zag, the U-Turn, and the S-Curve, which is essentially a sinusoid with the addition of the straight line at the beginning and end for proper priming of the path memory function.

TABLE 4.2 – FEATURES OF COMPARED STEERING ALGORITHMS

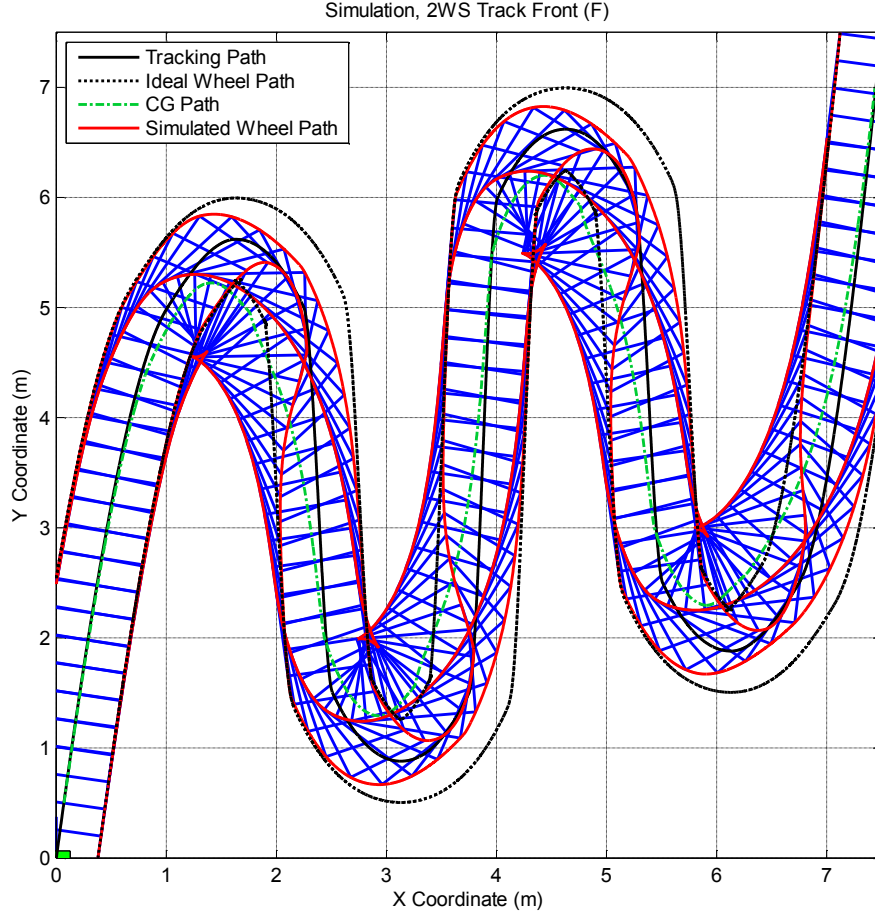
	Diagram	Location of ICR	Path, Relative to Vehicle
2WF		On projection of rear axle	<ul style="list-style-type: none"> <li>• Center of front axle lies on path</li> <li>• <math>\delta_f</math> is tangential to path at this point</li> <li>• <math>\delta_r = 0</math></li> </ul>
4CG		On longitudinal perpendicular bisector of vehicle	<ul style="list-style-type: none"> <li>• Center of gravity, <math>G</math>, lies on path</li> <li>• <math>\vec{V}_G</math> is tangential to path at <math>G</math></li> <li>• <math>\delta_r = -\delta_f</math></li> </ul>
4FM		On longitudinal perpendicular bisector of vehicle	<ul style="list-style-type: none"> <li>• Center of front axle lies on path</li> <li>• <math>\delta_f</math> is tangential to path at this point</li> <li>• Center of rear axle not constrained to path</li> <li>• <math>\delta_r = -\delta_f</math></li> </ul>
4FR		Free to lie anywhere in the XY plane	<ul style="list-style-type: none"> <li>• Center of front axle lies on path</li> <li>• <math>\delta_f</math> is tangential to path at this point</li> <li>• Center of rear axle lies on path</li> <li>• <math>\delta_r</math> is tangential to path at this point</li> <li>• <math>\delta_f \neq \delta_r</math></li> </ul>

It is assumed that these three paths contain all elements of arbitrary real world paths, including round curves, straight lines, corners and sharp cusps. Logically, if the robot can navigate these sample paths then it should be able to navigate any arbitrary path that can be formed by an amalgamation of these simple paths. Additionally, these simple path shapes can be easily modeled as a continuous function of the  $X$  and  $Y$  coordinates, with unique lateral or longitudinal trajectories that are easy to simulate numerically. This made analysis easier because MATLAB can process the data as functions using predefined tools. Once the path function is established and discretized, a number of parameters can be extracted from it, which include the position of the front and rear axles, the path slope, vehicle orientation angle, and the path angles at the front and rear axles.

The full simulation results are shown in Appendix B; for documentation purposes, in the simulations shown, the wheels and centrodes are not shown. Instead, only the wheel paths are plotted, as these were used in the metric to measure the relative maneuverability of the particular steering algorithm. Figure 4.7 is a sample depiction of the path tracking results; it shows the tracked path, the paths that would be tracked by an ideal path tracking system, the paths tracked by the controller, and the trajectory of the center of gravity. A total of 24 scenarios were simulated as included in Appendix B; this section discusses the results obtained from this simulation as well as provide a numerical summary of the performance comparisons.

Dead reckoning methods are used to determine the new location of the vehicle.





**FIGURE 4.7 – SAMPLE SIMULATION OUTPUT OF 2WF ON S-CURVE PATH**

The location of the front axle is congruent with the succeeding discrete point on the path and the change in distance,  $\Delta$ , is given by

$$\Delta = \sqrt{(x_f[j-1] - x_f[j])^2 + (y_f[j-1] - y_f[j])^2}, \quad (4.1)$$

where  $x_f$  and  $y_f$  are the x and y location of the front axle tracer at each time step,  $j$ . The wheelbase,  $H$ , and intermediary values  $\alpha$  and  $\beta$ , given by

$$\alpha = (\Delta^2 + H^2 - 2\Delta H \cos(\pi - \delta_f[j-1])) \quad (4.2)$$

and

$$\beta = \left| \sin^{-1} \frac{|\Delta \sin(\delta_f[j-1])|}{\alpha} \right| \quad (4.3)$$

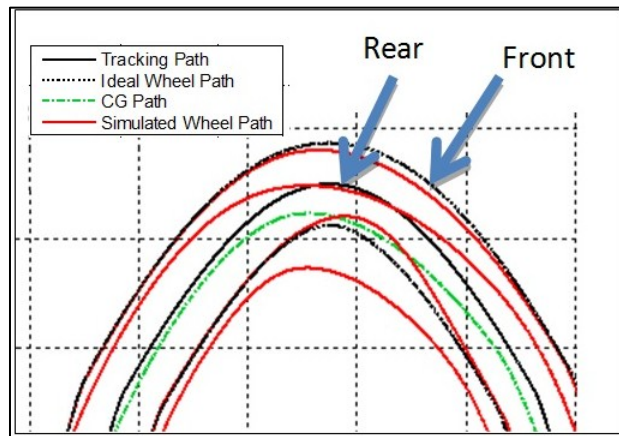
are used to determine the current vehicle heading given by

$$\theta[j] = \theta[j-1] + \sin^{-1} \frac{|\alpha \sin(\delta_f[j-1] + \beta)|}{H} - \delta_f[j-1]. \quad (4.4)$$

Subsections 4.2.1 to 4.2.4 discuss some of the observed general features that distinguish these steering algorithms.

#### 4.2.1. FRONT WHEEL STEER – 2WF FEATURES

The front wheel steering (2WF) mode is commonly found in passenger vehicles. In this case, the rear wheels are kept at an angle of  $0^\circ$  relative to the vehicle orientation while the front wheels are steered using an Ackermann steering geometry. The equations developed in Chapter 3.2.2 may still be used; however, the rear angle,  $\delta_r$ , remains at 0. Simulation results from this configuration are shown in Figure B.1 - Figure B.6 in Appendix B.



**FIGURE 4.8 – ILLUSTRATION OF THE REAR WHEELS ‘LAGGING’ THE FRONT WHEELS**

One noticeable characteristic of this steering algorithm is that the rear wheel paths always lag the front wheel paths by cutting towards the inside of the curve, taking a shorter path than the front wheels as is illustrated in Figure 4.8.

#### 4.2.2.FRONT AXLE PATH TRACKING – 4FM FEATURES

The second steering configuration was the front axle tracking with mirrored rear angle (*4FM*), which is typical of the methods proposed in [96] [97] [98]. This method is representative of a case presented in Section 6.2 where  $\delta_f$  is computed from the gradient of the path at the front of the vehicle given by a sonar array, and  $\delta_r$  is a negation of  $\delta_f$ . This implies that the ICR always lies on the centerline of the vehicle, but offers smaller turn radii than *2WF*. A key feature of this method is that the CG experiences zero sideslip (i.e. relative to the vehicle heading, there is zero lateral movement). The results of this simulation are shown in Figure B.7 - Figure B.12.

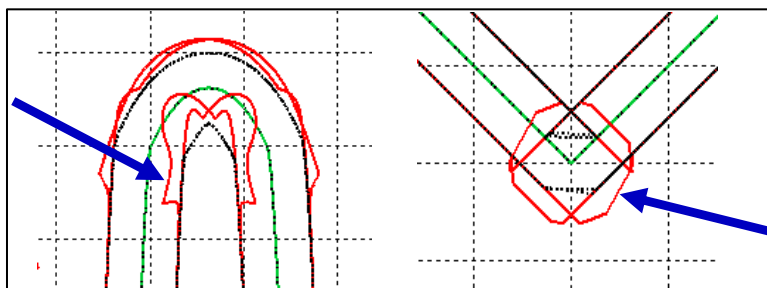
#### 4.2.3.CENTER OF GRAVITY TRACKING – 4CG FEATURES

The third steering configuration was to track the center of gravity along the given path (*4CG*). The center of gravity seems like a natural tracking point for a predefined track such as a buried wire or line following robot. The key feature of this method, like *4FM* is that it offers zero sideslip of the vehicle CG, as the velocity,  $\vec{V}_G$ , is always tangential to the path. Although it seems impractical to track a vehicle on the fly using these means, the maneuverability of this steering configuration was simulated. In a real world situation, using an adaptable navigation system such as laser or sonar range finders, the *4FM* algorithm would be better suited for path navigation. However, these sensors typically lie at the front of the vehicle and are therefore aware of the location of the front axle instead of

the center of gravity. This configuration also constrains the ICR to the perpendicular bisector as the rear wheels mirror the front wheels.

The simulation of *4CG* steering was relatively easy. First, the slope of the path is calculated using finite difference methods, then the vehicle is translated and rotated such that  $G$  coincides with the path center and the vehicle heading is tangential to the path at  $G$ . From  $G$ , the path is searched forward and backward until a distance of half of the wheelbase,  $H$ , is reached and the front and rear path trackers can be located. The slope of the path, in the  $XY$  frame, is calculated at these points and the vehicle heading,  $\theta$ , is subtracted from front and rear slopes, resulting in  $\delta_f$  and  $\delta_r$ , for that time step.

There are two notable characteristics of this path tracking approach as illustrated in Figure 4.9. First, it engages extreme steering angles to negotiate corners, even on relatively smooth continuous paths to maintain the tangential requirement. Second, it momentarily stops and rotates the vehicle by some angle to negotiate discontinuous paths, such as the Zig-Zag path with sharp cusps. This would be difficult to implement on the fly where this maneuver may be necessary such as in the case of turning a corner in an urban environment, for a constant velocity,  $V_G$ .

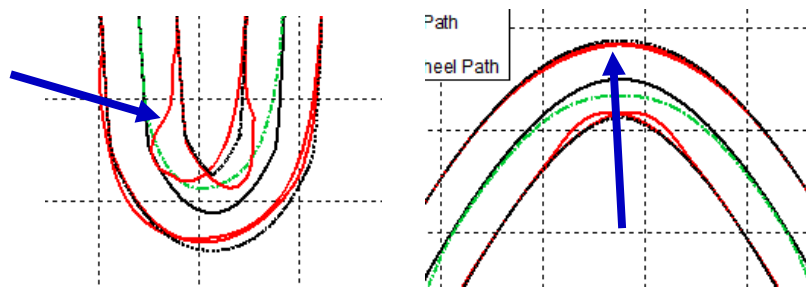


**FIGURE 4.9 – *4CG* TRACKING REQUIRES EXTREME ANGLES FOR RELATIVELY SMOOTH PATHS (LEFT) AND NEARLY INSTANTANEOUS ROTATION FOR NON-CONTINUOUS PATHS (RIGHT)**

#### 4.2.4. FRONT AND REAR AXLE PATH TRACKING – 4FR FEATURES

The final steering configuration simulated was the one proposed in this research. Both the center of the front and rear axle track the center of the prescribed path. The simulation starts with the rear path tracker on the path. The path is then searched forward to the distance equal to the wheelbase,  $H$ , of the vehicle which locates the front path tracker. The  $XY$  slope of the path at both the front and rear of the vehicle was calculated using finite difference methods. The vehicle heading,  $\theta$ , was subtracted from the slopes with the resultant being the path angles,  $\delta_f$  and  $\delta_r$ .

At first glance, this method seems to be superior to the others. It is obvious that this method does not require as extreme of wheel angles as  $4CG$ . It is also quite evident that the wheels approximate the ideal wheel path much closer than both the  $2WF$  and  $4FM$  demonstrated in Figure 4.10. One should expect the numeric metrics to favor this method based solely on inspection.

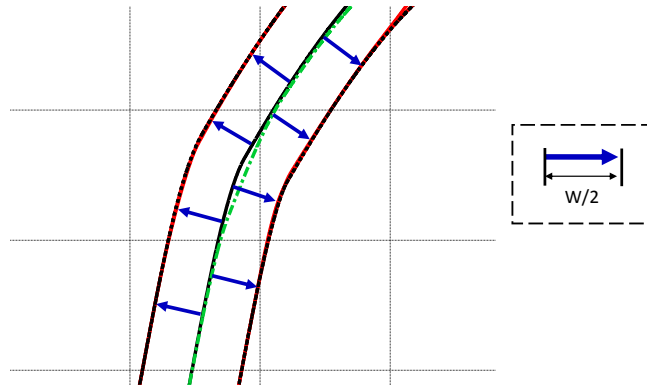


**FIGURE 4.10 – 4FR DOESN'T REQUIRE AS EXTREME OF ANGLES AS 4CG (LEFT) AND VERY CLOSELY APPROXIMATES IDEAL WHEEL TRACK (RIGHT)**

#### 4.2.5. PERFORMANCE METRIC AND RESULTS

A performance metric to numerically compare the various steering modes is required. Normally in vehicle maneuverability comparisons, common metrics are turn radii, skidpad measurements (essentially how much centripetal force is induced) or lap times. None of these measurements was viable because some of the compared steering configurations are capable of implementing zero radius turning. The skidpad and lap times are not relevant; these methods generally ignore errors from body roll or wheel slip because in this scope they are intended for low speed maneuvers, the typical operating environment of most wheeled mobile robots. Thus, a new metric had to be developed. The research used the mean value,  $\mu$ , of the Euclidean norm or the root mean square of the vehicle's normal deviation from the path center as a performance metric. To compute this metric, the locus of points at half the road width are plotted. The deviation  $\epsilon_i$  of the vehicle's CG from the path center at each sampled point,  $j$ , is calculated and used to determine the metric as

$$\mu = \sqrt{\frac{1}{n} \sum_{j=1}^n \epsilon_j^2} . \quad (4.5)$$



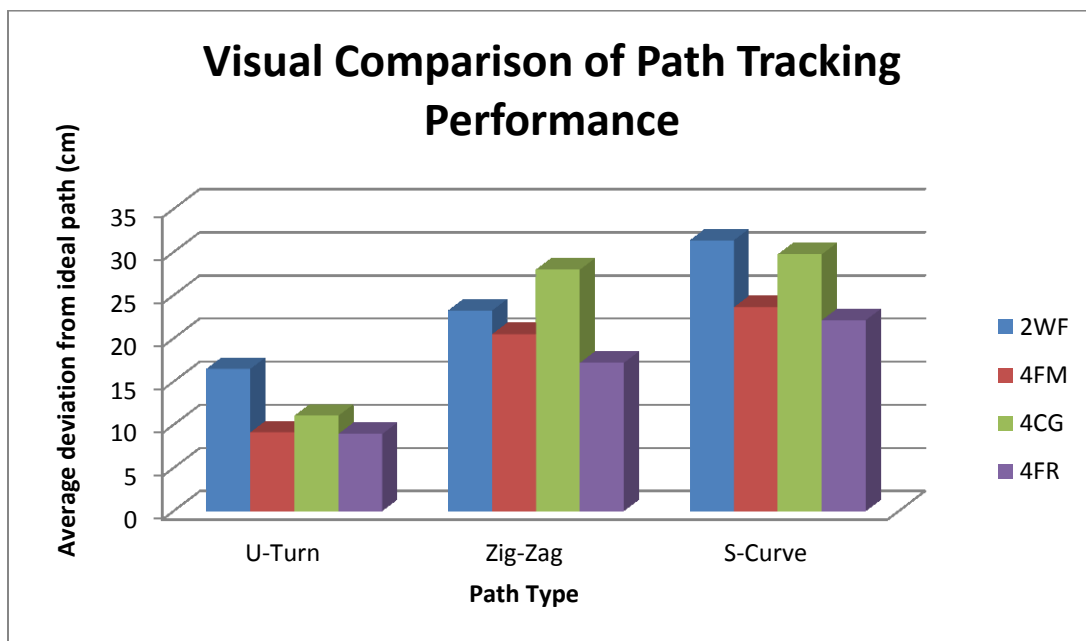
**FIGURE 4.11 – VISUAL REPRESENTATION OF MANEUVERABILITY METRIC, BLACK DOTTED LINES REPRESENT IDEAL WHEEL PATHS EQUIVALENT TO HALF WHEEL TRACK ALONG THE PATH NORMAL**

This metric can be thought of being analogous to the additional area of asphalt needed to pave the road for the robot to go around an assortment of obstacles; it is primarily a measure of the deviance of the wheels from the minimum possible paved area. A robot with two fixed, articulating axles, as shown in Figure 3.2, would be able to traverse any path perfectly and have an ideal score with the proposed metric. However, as was pointed out before, a vehicle of this sort will have variable relative distances to the CG making control difficult and has a continually changing base and consequently less lateral stability.

The Euclidean distance is calculated between chosen reference points along the path and the actual location of the wheel at each time step. These obtrusions were averaged over the number of points taken in the simulation to compute the average deviations for the various steering algorithms as presented in Table 4.3 and in Figure 4.12.

**TABLE 4.3 – WHEEL DEVIATION FROM IDEAL PATH**

	U-Turn (cm)	Zig-Zag (cm)	S-Curve (cm)
<b>2FS</b>	16.48	23.23	31.37
<b>4FM</b>	9.13	20.49	23.61
<b>4CG</b>	11.04	27.97	29.74
<b>4FR</b>	8.93	17.18	22.11



**FIGURE 4.12 – COMPARISON OF PATH TRACKING STEERING CONFIGURATIONS**

Several conclusions can be drawn from the simulation data. In general, the paths that are more involved with the most turns also have the most deviations for all steering configurations. Secondly, for nearly all paths, 4WS modes performs better than the 2WS mode, with less average path deviation, justifying the need for the additional actuators. Finally, the proposed method, *4FR*, shows superior performance to all other modes for every path. It performs with a relative deviation 20-40% better than *4CG*, dependent on



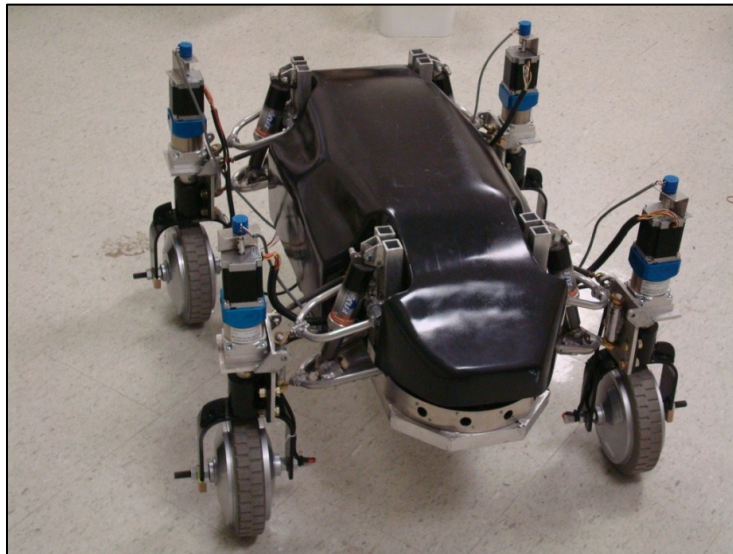
path geometry. However, the *4FM* mode is only marginally worse for two of the paths with a relative deviation error of 2-16% poorer, indicating that it may be a close contender.

These simulations provide a good basis and will be confirmed through experimentation on the mobile robot platform, the BIBOT-1, in the next Chapters.

## 5. IMPLEMENTATION OF THE REDUCED MODEL ALGORITHM ON BIBOT-1 VEHICLE

### 5.1. VEHICLE DESCRIPTION

The experimental test bed, BIBOT-1, was an AWD/AWS four wheeled mobile robot, shown in Figure 5.1. The chassis measures 94 cm long by 25.5 cm wide. The wheel track is 66 cm while the wheel base is 58.5 cm. It is equipped with four wheel units, each independently controlled, with an in-wheel tractive wheel hub motor as well as a stepper motor for steering. This makes the robot a completely independently all-wheel-steered and independently all-wheel-driven vehicle.



**FIGURE 5.1 – THE BIBOT-1 WHEELED ROBOTIC VEHICLE EXPERIMENTAL PLATFORM**

#### 5.1.1. BACKGROUND HISTORY

Development of the mechanical structure of this robot began in 2006 by a team of NDSU mechanical engineering senior students as a capstone design project [99]. The

electronic control hardware was initially designed by a team of NDSU senior electrical engineering students as a subsequent capstone design project [100]. This initial electronic design schema was deemed inadequate for a number of reasons and had to be redesigned.

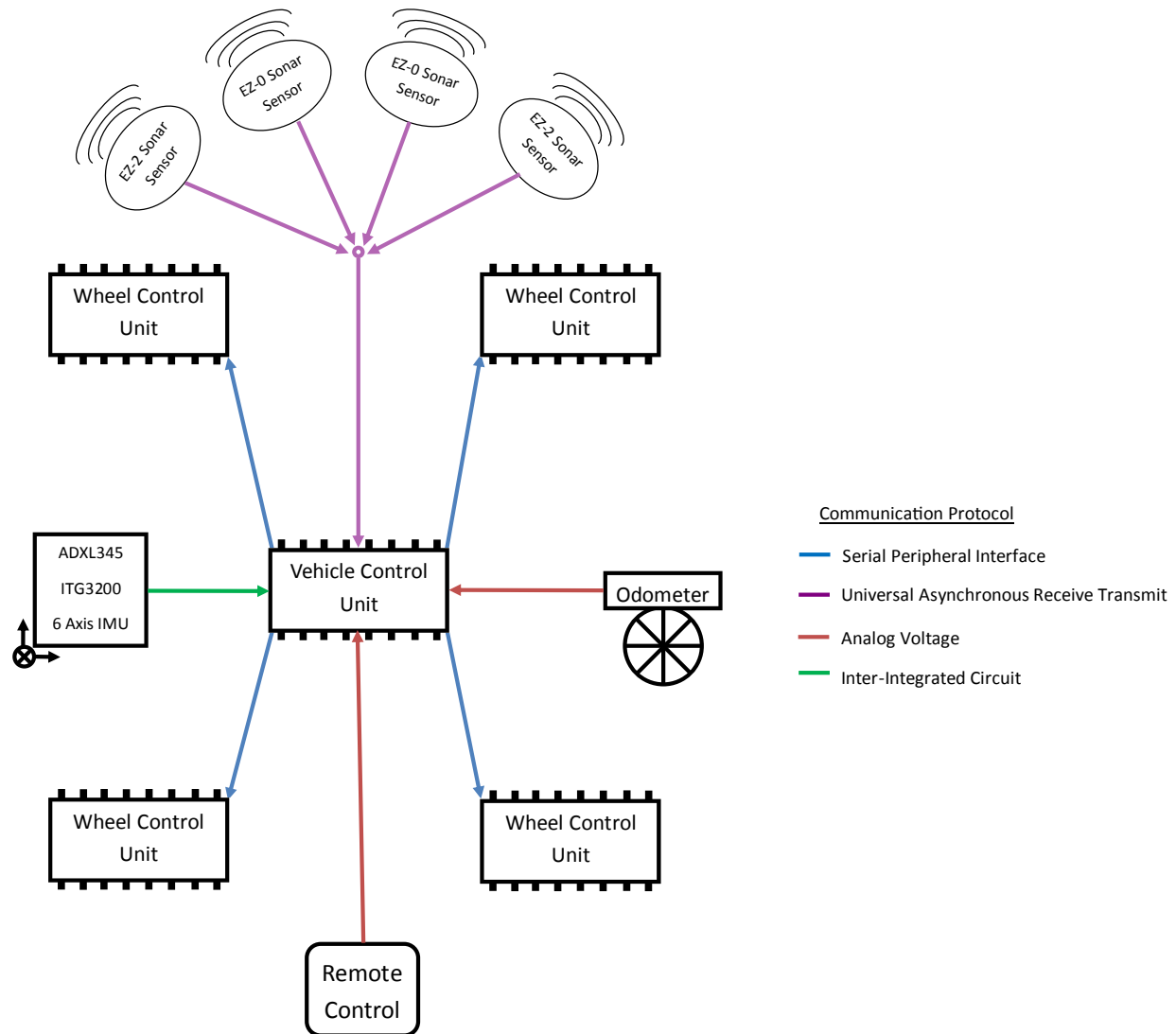
The redesign of the robot hardware and control architecture became the thesis subject matter of the student who preceded this work, Nikhil Gupta [95]. The robot electronics were designed with a structure that allows both centralized and decentralized control architectures to be implemented. There is one Vehicle Control Unit, VCU, and four Wheel Control Units, WCU. The VCU is a central microcontroller that determines the vehicle motion; it was designed with the ability to gather information from an array of sensors and determine the vehicle's current position and velocity. After collecting the vehicle information, the VCU transmits that information to all of the WCUs. Each WCU is free to calculate the necessary speed and steering direction of the wheel in any manner depending on the selected steering algorithm. This structure allows the robot actuators to be infinitely scalable, because the information processing requirements are distributed to the WCUs; therefore, any additional WCUs do not cause further demand from the VCU.

## **5.1.2. CONTROL HARDWARE AND SOFTWARE**

### ***5.1.2.1. HARDWARE DESCRIPTION***

The VCU and WCU are powered by five dsPIC33FJMC128 Microchip™ PIC microcontrollers which communicate to each other via Serial Peripheral Interface, SPI, protocol. There is an individual microcontroller for each of the four WCUs as well as one on the VCU as can be seen in the communication architecture blueprint, shown in Figure 5.2. A total of three communication protocols plus two analog signals are used on the system in

transmitting information between either the VCU or the WCUs and their peripherals such as body sensors and wheel sensors as well as for intercommunication between the VCU and the WCUs.



**FIGURE 5.2 – HARDWARE CONNECTIONS TO VEHICLE CONTROL UNIT**

In this structure, all information passed from the VCU to each WCU is identical. The code on each WCU is specific for that wheel to interpret and transform the VCU information based on its relative position on the vehicle. The WCU steers the wheel through a stepper

motor via a low level PIC10F that generates the grey code for the stepper driver. One drawback is that there is no control over the speed of the stepper because the PIC10F has only two inputs; one corresponds to steering direction while the other generates the motor drive pulses. The stepper is connected to the wheel through a gearbox, which increases the steering torque and reduces the steering rate. Each stepper motor has a 50 k $\Omega$  multiple turn potentiometer for feedback control of steer angle positioning. Because the potentiometers are connected directly to the stepper motors instead of the wheel, the controller has increased resolution over the steering angle for the fixed resolution of the ADC module on the microcontroller through the gear reduction.

Finally, the wheel is driven by a 24 volt DC hub motor, which allows the wheel to exert a tractive torque without the need for a challenging transmission. This makes for a mechanically simple and robust vehicle propulsion system. The hub motors are driven through custom pulse width modulated H-bridge drivers, and the steer motors are controlled through custom unipolar stepper drivers, all designed and implemented by Gupta [95]. An optional joystick is included for tethered teleoperation, this joystick is connected to the VCU's microcontroller analog to digital converter channels, which interprets the voltages as direction and speed signals; it is used to control the robot for manual positioning.

#### *5.1.2.2. SOFTWARE DESCRIPTION*

When this portion of the research was initiated, the hardware was almost complete and robust, but the control firmware left much to be desired. Although, it provided a crude groundwork for what was eventually redeveloped. In order to use this robot for the

intended experiments, it was necessary to fully develop this firmware. The development of the vehicle control software was the most labor intensive part of this project. It involved reevaluating the current software, familiarizing with the development environment, and learning details of microcontroller features to create an easily understandable, expandable, modular architecture.

The initial software for the project was developed in Microchip™ MPLAB v8.63 Integrated Development Environment. The project was later migrated to a beta platform called MPLABX, which is based on Java, with a much cleaner interface, allowing for java commands when using features like code folding. This makes the structure of the programs much more intuitive and modularized. The user must manually configure a “project”. This includes information based on the model, compiler and any configuration options available to the microcontroller. Each project contains header files, which define prototypes, several source files, which create the executable code, and finally a trap file, which determines error handling procedures. To streamline development, this task uses two projects only: one for the VCU and another for the WCU, of which source code can be found in Appendix E and Appendix F, respectively.

Each project was organized with two main source code files: the *XCU\_Main.c* and the *XCU\_Drivers.c* files. The *XCU\_Main.c* is the primary source file that contains the repeated functions that will be executed when the microcontroller is first powered. These functions are those that will make calls to the sensors, send and receive information, and perform the necessary calculations to manipulate data in a way that can control the actuators. The *XCU\_Drivers.c*, is a secondary file that contains all subfunctions which are primarily used to

initialize communication modules, map and configure all microcontroller I/O channels, and set up options, timers and interrupts.

## **5.2. PERCEPTION AND ODOMETRY SYSTEMS DESIGN**

At the beginning of the experimentation phase, the robot did not have sufficient sensors as would be required for autonomous navigation; particularly perception and odometry sensors were absent. As part of the control system design, these sensors had to be added to the robot structure. The following subsections describe how the perception and odometry system was designed and incorporated to the robot.

### **5.2.1. SONAR PERCEPTION SYSTEM**

An array of four sonar sensors was added to the front bumper of the robot. These sensors are arranged in pairs labeled EZ-0 and EZ-2. The sonars communicate with the control firmware via the Universal Asynchronous Receiver Transmitter, UART, protocol. The purpose of these sensors is to determine the range to the nearest object, which is required in obstacle avoidance and corridor navigation maneuvers.

### **5.2.2. ODOMETRY SYSTEM**

An odometry system was designed and added to the robot for use in both robot localization and motion control. It consists of a 3-axis ADXL345 inertial measurement unit (IMU) and a wheel encoder. The ADXL345 is equipped with a set of 3-axis accelerometers capable of measuring linear accelerations up to 16 G's, and a set of 3-axis gyroscopes capable of measuring angular velocities up to 2000°/s. This IMU can measure a total of six axes and communicates with the VCU via the Inter-Integrated Circuit, I<sup>2</sup>C, protocol. Since the hub motors of the vehicle have no built-in encoders, an additional makeshift

independent wheel encoder odometer was added to the robot. This system consists of a 14 cm diameter castor wheel with a 32 division radial black and white pattern, as can be seen in Figure 5.3. An analog phototransistor mounted in the bracket detects the wheel patterns and sends pulse signals to the VCU indicating changes in the wheel positions, with a resolution of 1.37 cm/pulse.



**FIGURE 5.3 - THE INDEPENDENT ODOMETER FOR MEASURING DISTANCE TRAVELED**

Together, these odometry sensors gather information, which is read and interpreted by the VCU to generate vehicle motion data which is in turn communicated to the WCU for coordination of the 8 independent motors in accordance with the steering algorithm implemented.

A GPS system was designed but was not implemented to the robot not only out of time limitations but also due to its poor indoor performance, which made it impractical. It is however recommended to implement this GPS for future outdoor applications.



### 5.3. ODOMETRY AND CONTROL SOFTWARE

#### 5.3.1. VCU CODE SCHEME

As the microcontroller is first powered, it calls the 'main()' function. This function chooses one of the available steering modes, *2WF*, *4CG*, *4FM*, or *4FR*. Independent steer angles use equations (3.10)-(3.17) to determine the steer angle and velocity of each wheel unit. After choosing the steering mode, it initializes the microcontroller operating environment by setting up the computational speed, which is *40 MHz*, and setting up I/O and all communication channels. Finally, the function enters into an endless loop, in which it reads all onboard sensors, determines the desired vehicle speed  $V_G$ , and the path angles  $\delta_f$  and  $\delta_r$ . This information is constantly sent to WCU at variable rates as follows: selected steering mode is transmitted using the SPI module at a rate of *5 MHz*, the desired vehicle speed,  $V_G$ , as well as  $\delta_f$  and  $\delta_r$  are sent using Timer 4 interrupt at intervals of *50 ms*.

The VCU microcontroller reads the wheel encoder odometer using a real time interrupt, which is triggered every time the encoder wheel voltage level detected by the phototransistor changes. In response to this interrupt, the 'main()' function increments the distance traveled by the robot's center of mass by *1.37 cm*, corresponding to the circumferential arc length of the wheel between the black and white bands of the encoder.

Sonar sensors are connected to the microcontroller through the UART serial module, which must be initialized in order for the sensors to be read. These sensors are read by the microcontroller triggered by the Timer 5 interrupt at a rate of *20 Hz*. The sensor works such that when the RX communication line is brought high for *20  $\mu s$* , the sensor ranges and returns the distance in inches via an ASCII representation. This data is

captured by the communication module, checked for errors and stored in the *RangeX* variable; if the data is good, the microcontroller passes to next sonar sensor until all sonar sensors are read. The sonar sensors connections to the microcontroller is such that although each has its own separate RX lines, they all share a common TX line, which minimizes the pin requirement for 4 sensors. The control software is designed to be able to distinguish signals from different sensors. The data from the sonar sensors is used to calculate  $\delta_f$  using a simple lever rule,

$$\delta_f = K_s \frac{r_l - r_r}{r_l + r_r}, \quad (5.1)$$

where  $r_l$  is the distance to the leftmost obstacle,  $r_r$  is the distance to the rightmost obstacle and  $K_s$  is the steering gain, which was an experimentally determined number that would steer the wheels away from the obstacle by an appropriate angle. In the case of mirrored steering angles this number was simply negated and assigned to  $\delta_r$ .

The IMU is connected to the microcontroller through the I<sup>2</sup>C bus; therefore, to read the IMU, the I<sup>2</sup>C module also must be initialized. Then, a function called ‘*initIMU()*’ is called which checks the active axes, collects 100 samples on each axis, and calibrates the axes ready for use. The IMU is read by the microcontroller as a response to Timer 2 interrupt which runs at a rate of 50 *Hz* as configured in the ‘*initIMU()*’ function. Every time the IMU is sampled, the units are converted to the CGS/radians unit systems and integrated to give the respective acceleration, velocity and position or orientation [101].

The tether joystick is connected to one of the 10 bit ADC ports, which is read as a response to Timer 3 interrupts occurring at a rate of 8 *kHz*. The tether signal is configured

with maximum of 3.3V compatible with dsPIC microcontrollers voltage level; this signal is discretized into 1024 levels compatible with 10 bits of the ADC module. These numbers are then manipulated into desired vehicle speeds and path angles.

Figure 5.4 shows the logic flowchart of *VCU\_Main.c* for navigating through a corridor using the *4FM* algorithm with a path memory function. The VCU gathers information from the sonars and uses that information to calculate  $\delta_f$ . The three timer modules regulate the sonar ranging, IMU sampling and SPI transmission on a prescribed schedule. Every time the comparator interrupts, the current angle at the front wheels is stored, with compensation from the angular orientation. To implement a path memory function, proposed in Chapter 3, the absolute angle,  $\rho_f$ , of the wheels relative to their global orientation was stored in an array, and retrieved again for computing  $\delta_r$  after the rear axle tracer had traversed the distance of the vehicle, where

$$\rho_f[j] = \delta_f + \theta, \quad (5.2)$$

and  $\theta$  is the absolute orientation of the vehicle relative to the global coordinate system. The rear angle,  $\delta_r$  is retrieved by subtracting the body orientation from the number that is read from the array, i.e.,

$$\delta_r = \rho_f[j - n] - \theta. \quad (5.3)$$

Where  $n$  is the number of steps from the front position to the rear of the vehicle. This method of storing the front steering angle with respect to the global coordinate system, allows the center of the rear axle track the center of the front axle very precisely.

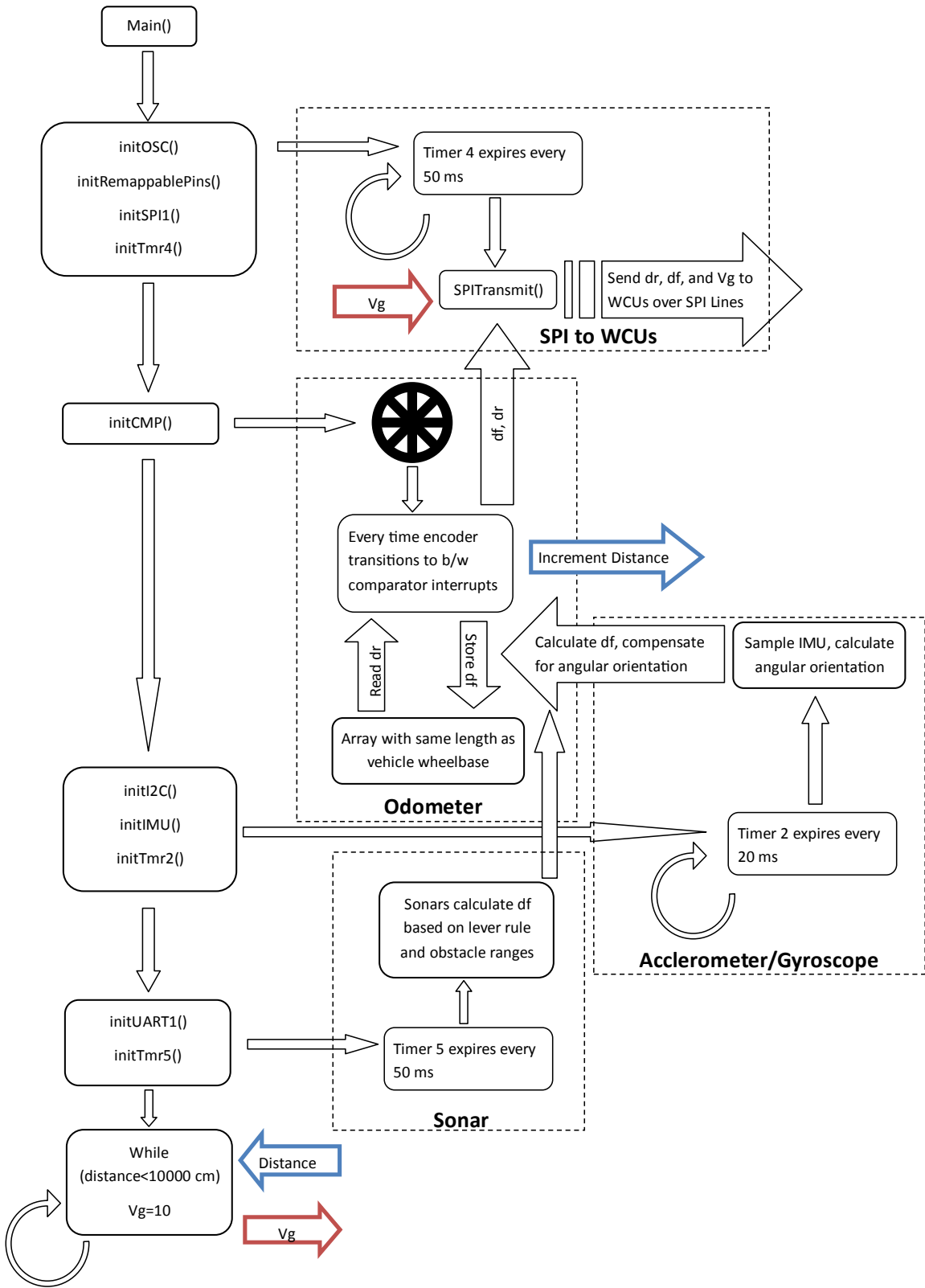


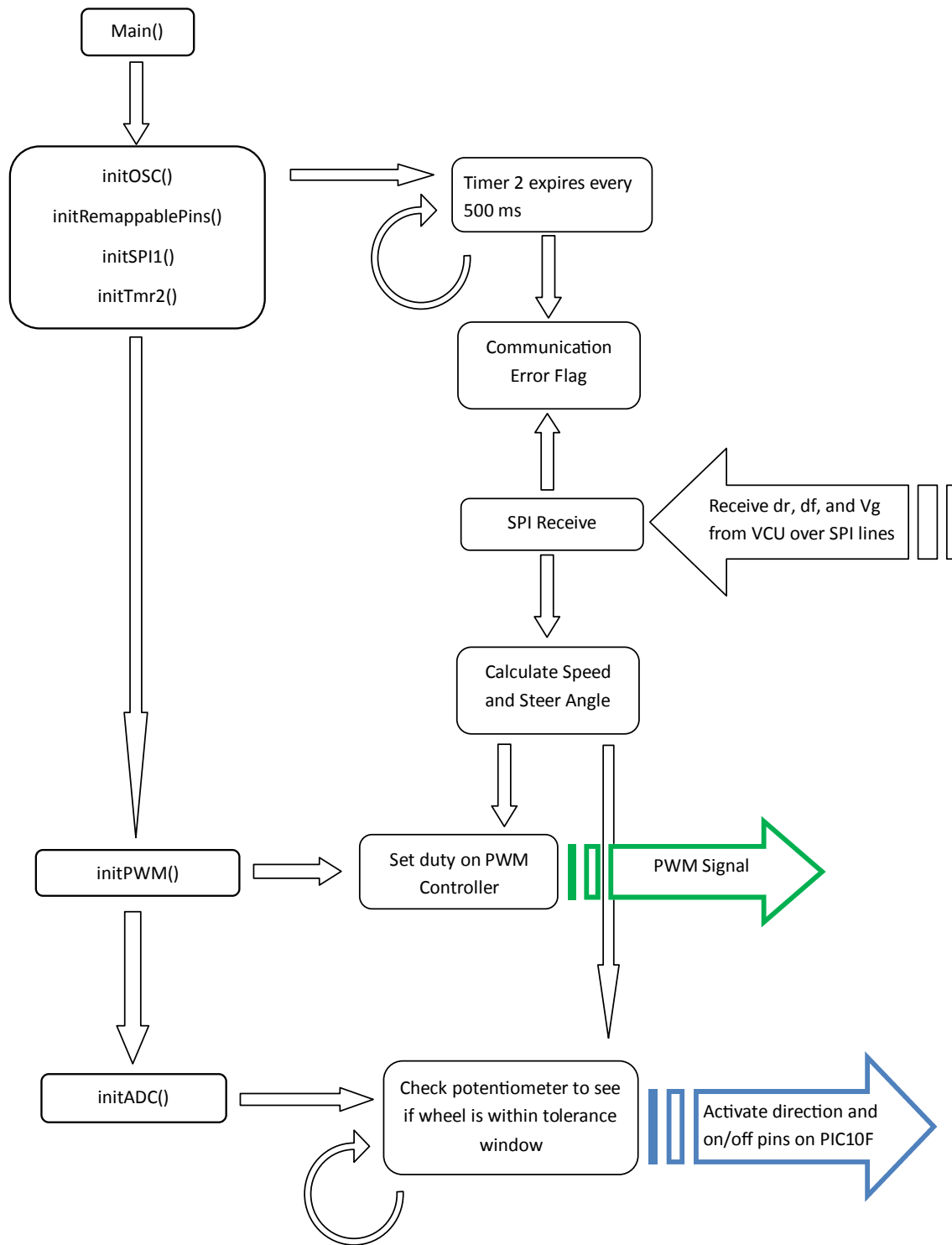
FIGURE 5.4 – LOGIC FLOWCHART OF VEHICLE CONTROL UNIT WITH PERIPHERALS

### 5.3.2. WCU CODE SCHEME

The WCU control scheme shares many common features with the VCU. As the microcontroller is first powered, it goes through an initialization process by setting up the clock frequency at 40 MHz, and initialize the I/O channels similar to the VCU. Again the SPI module for communication with the VCU is initialized with the same configuration as that of the VCU except that the WCU module is set to operate in the slave mode as opposed to the master mode for the VCU. This means that the VCU will initialize all information transfers by the chip select line and will maintain the clock cycling. Also as before, the ADC is initialized, scanning on two channels at a rate of 8 kHz. The WCU uses only one of these ADC channels for steering feedback of the potentiometer mounted to the steering stepper motor. Finally, the pulse-width-modulation unit is initialized running at an operating frequency of 40 kHz, which is used to control the speed of the DC hub motors.

The SPI communication between the VCU and WCU is coordinated at the WCU through a communication error semaphore flag and a real time interrupt which is set to run at 500 ms intervals. At each real time interrupt occurrence, the timer sets a communication error flag, which is cleared whenever the SPI receives data from the VCU. Since the VCU transmits at a much higher frequency than the frequency at which the WCU reads the data, the semaphore flag should always be reset in time before the WCU reads the data. However, as a safety measure against any possible communication error between the VCU and the WCUs, the WCU is set to ignore the data if its real time interrupt occurs before the semaphore flag is reset.

Once all modules have been appropriately initialized, the WCU program enters the main 'while()' loop. In this loop, the steering angle and wheel speed are calculated based on the three parameters passed from the VCU, i.e., the vehicle speed,  $V_G$ , and the path direction angles,  $\delta_f$  and  $\delta_r$ . The speed controller is a simple function that sets the duty cycle of the PWM module. Although a speed feedback from the wheel is required, the hub motors used on this robot do not have the required feedback tachometers, so this speed was calibrated to be proportional to the voltage applied to it, assuming that robot runs on flat surface. The steer controller, however, uses a feedback sensor in the potentiometer connected to the stepper motor; this controller sends drive signals to the dedicated PIC10F microcontroller that in turn pulses the stepper driver. The function completes when the wheel has entered a prescribed orientation tolerance window or upon the reception of a different value from the VCU. The allowance for steering interruption allows the VCU to transmit two extreme or polar angles to a wheel and it does not have to wait for the wheel to reach the first commanded angle before attempting to steer to the next commanded angle, allowing for a much more responsive reaction. Figure 5.5 is a visualization of the logic used in the WCU code to control the two motors upon reception of control information from the VCU. Although each wheel module is at a different position on the vehicle relative to other wheels, each implementing different control algorithms as specified by equations (3.10)-(3.17), all wheels share the majority of code components except the computations of the steering angle and wheel speed. As such, only one *WCU\_Main.c* program is written to handle all four wheels, and local control calculations were implemented accordingly.



**FIGURE 5.5 - LOGIC CONTROL SCHEME OF WHEEL CONTROL UNIT**

## 6. EXPERIMENTAL RESULTS

After numerically validating the proposed control algorithm, the last activity in this research was to experimentally demonstrate the viability of the proposed control algorithm on a robot platform. This chapter describes the experimental results obtained by implementing the proposed algorithm on BIBOT-1 as described in the previous chapter. Two sets of experiments were conducted as described in Sections 6.1 and 6.2. The first set evaluates the passive performance of the proposed algorithm on preprogrammed path maps, and the second set evaluates active performance of the algorithm on obstacle defined paths. In the first set, the robot is programmed with a path map and is required to track that path, while in the second set, the robot is required to follow a particular path defined by obstacles, so that by using onboard sensors, the robot avoids hitting these obstacles and hence tracks the desired path. Both sets use three paths, similar to those used in the simulation because of their key characteristics as well as historical precedents.

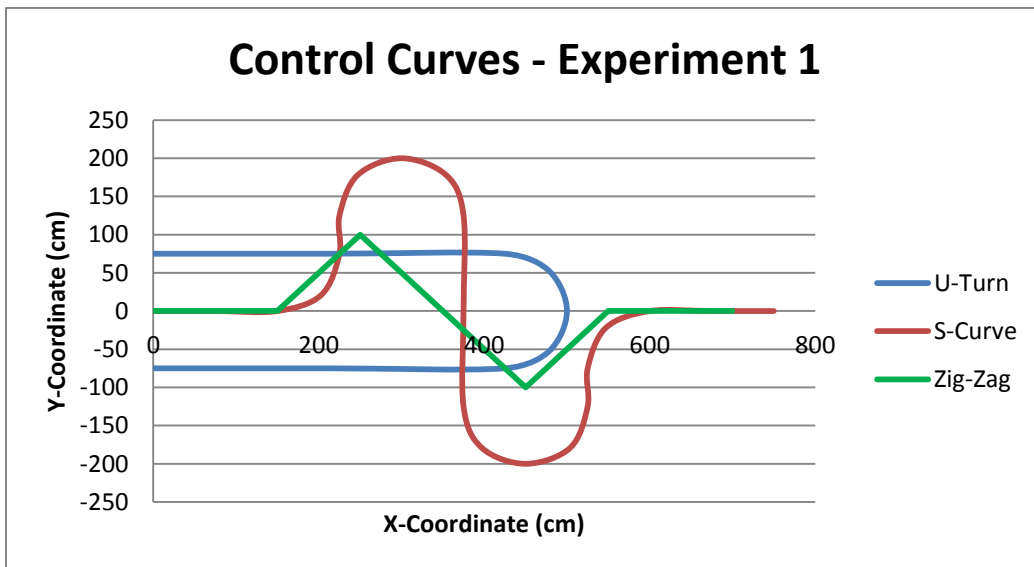
### 6.1. PERFORMANCE RESULTS ON PASSIVE PRE-PROGRAMMED PATHS

This part of the experimentation was intended to validate the results and trends of the simulation. It was designed to circumvent as many external errors as possible, especially errors due to perception sensors. As will be shown in the next set of experiments, these sensors were the primary sources of errors in robot navigation. Because these errors can overshadow the performance differences among compared steering algorithms, they need to be attenuated if the simulation is to be validated. The best way to do this is to minimize the navigational information collected by the robot by preprogramming a mapped path into the VCU's microcontroller memory. As the robot



traverses the path, the steering angles are read from the internal memory instead of being interpreted from the sonar sensors. The performance of each algorithm was evaluated using a root-mean-square deviation of the robot path from the desired path, which is equivalent to the statistical standard deviation of the robot path. Comparison trends are able to be extracted from this data and verify the simulation results.

The three paths tested include a U-Turn, a Zig-Zag path, and an S-Curve, seen in Figure 6.1. These paths are, however, scaled appropriately fit in the available experimentation space, which is limited to  $6m \times 9m$ .



**FIGURE 6.1 – CONTROL CURVES FOR EXPERIMENT 1**

The three methods compared in this experiment were *2WF*, *4CG* and *4FR*. For this experiment, *4FM* was not used because of its similarity to *4CG*. The only difference between the two algorithms is the location of the tracer point relative to the vehicle. For this type of experimentation, *4FM* does not seem appropriate because it is based on using sensor data

gathered the front sonar array, which was disabled for this experiment. This *4FM* configuration however, was addressed in the second set of these experiments. The path tracking performance of the robot was measured using the same metric as that used in the simulations of Chapter 4.

### 6.1.1. EXPERIMENTAL SETUP AND PROCEDURE

In this experimentation, the paths are preprocessed using an adaptation of the simulation done in MATLAB. The steering angles,  $\delta_f$  and  $\delta_r$ , are calculated as a function of the vehicle CG distance along the path, using dead reckoning methods similar to those described in (4.1) - (4.4) with a resolution of 1 cm. These angles are stored as arrays of between 1000 and 2500 points consistent with the length of the path in centimeters. The arrays are loaded as lookup tables into the flash memory of the VCU microcontroller, but are not included in the supplied code because of their sheer length. A path function was created where the odometer interrupt caused the function to load the appropriate values of  $\delta_f$  and  $\delta_r$ , for the respective position. To provide sufficient time for the microcontroller to process these path arrays, the vehicle was kept at a constant speed of 0.5 m/s throughout all paths and across all steering methods.

In order to allow the robot make instantaneous stops and turns when negotiating sharp corners on the Zig-Zag path under the *4CG* tracking steering as discussed before, the path function was defined in segments. The robot would follow one segment to its end where it would stop and turn by a prescribed angle as monitored by the gyroscope before following the next segment. This process is repeated by the robot for each turn to the end of the path. Although this approach worked well, it was prone to angle measurement errors

from the gyroscope which were not experienced by other algorithms. This may have contributed to decreased path tracking performance for this run.

Two paper funnels were and attached to the path tracking studs, mounted at the center of the front and rear axles, as shown in Figure 6.2. In each run, the funnels were loaded with different colored sand that was then laid in two lines as the robot traversed the path, marking the path of the front axle as well as the rear axle. This created a much more robust method of recording the robot path compared to other methods such as tracing the path using a marking pen. The sand is not pressure or time sensitive, consistently leaves an easily distinguishable line for later acquisition, and is quickly cleaned with a broom and a dustpan to reset each run. The  $XY$  coordinates of the path tracked by the robot were collected by counting 9" floor tiles from a predefined origin. The procedure was repeated three times for each of the three paths and three steering configurations for a total of 27 runs.



**FIGURE 6.2 – THE SAND TRACER MOUNTED TO THE AXLE TRACKER (LEFT) RECORDS THE MOTION OF THE VEHICLE (RIGHT)**

### 6.1.2. EXPERIMENTAL RESULTS

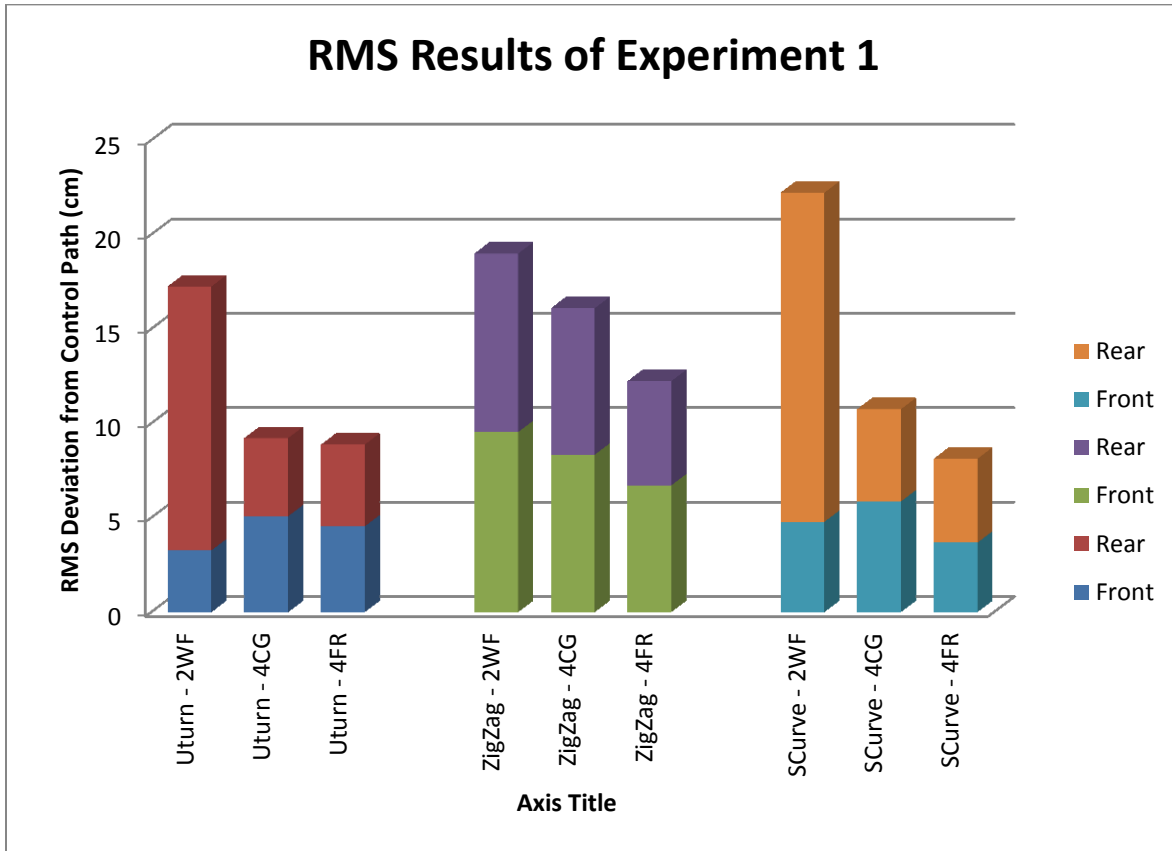
Once the results had been recorded from the robot's tracked path into  $XY$  coordinates, they were entered into both MS-Excel for visualization as well as MATLAB for data analysis. Although the full collection of raw results are attached as Appendix C, the major observations are discussed in the following section. To compare the steering algorithms over the different courses, a similar metric,  $\mu$ , used in numerical simulation was also used here. First, the locus of the path center was discretized into a regular array in MATLAB, then at each of these points, the experimental curve was scanned from beginning to end and a nearest neighbor, also known as a Fréchet distance [102], was determined as a representation of the path deviation,  $\epsilon$ . The RMS value of the path deviations for the whole path is the metric,  $\mu$ , as defined in equation (4.5). The results obtained for all methods and all paths are summarized in Table 6.1 and in Figure 6.3.

**TABLE 6.1 – RMS DEVIATION FROM CONTROL PATH**

<b>Steering Algorithm</b>	<b>U-Turn (cm)</b>	<b>Zig-Zag (cm)</b>	<b>S-Curve (cm)</b>
<b><i>2WF</i></b>	17.25	18.98	22.21
<b><i>4CG</i></b>	9.21	16.09	10.75
<b><i>4FR</i></b>	8.87	12.23	8.12

Several observations are made from these results. As had been observed in the numerical simulation, the *2WF* steering configuration results in paths wherein the rear wheels 'lag' the front wheels. Additionally, it is also observed that the robot attempted to navigate the path with extreme steering angles in the *4CG* steering mode. These

observations are in agreement with those observed in numerical simulation results of Chapter 4.



**FIGURE 6.3 – RMS DEVIATION OF BIBOT-1 FROM PRESCRIBED PATH**

Similarly, the observed RMS errors are, in general, on the same order of magnitude as those predicted by the simulation. The trends in relative performances of various steering methods also match quite well. For all three paths, both AWS configurations perform better than the conventional car-like *2WF*. These results confirm the worthiness of using all wheel steering in comparison to the conventional car-like front wheel steering. Furthermore, comparison of the two AWS modes shows that the proposed *4FR* steering algorithm performed relatively 4-25% better than the *4CG* tracking method for every path.

## 6.2. PERFORMANCE RESULTS ON REACTIVE OBSTACLE DEFINED PATHS

The final set of experiments demonstrated the practical implementation of the path tracking algorithm used in a real world environment where the path is defined using obstacles. This environment requires the robot to create a path map in real time using its onboard odometry system, which includes the IMU and the sonar sensor array. This real time map is the used by the robot to navigate through a simulated corridor of varying shapes and sizes. This part of the experiment tested only the two AWS steering method: the *4FR* and *4FM* tracking algorithms. The *2WF* and the *4CG* were not tested because the *2WF* was deemed inferior from all previous results and the *4CG* configuration was assumed to be equivalent to the *4FM*. Recall that the *4FM* algorithm is similar to the *4CG* method because both constrain their ICR along the longitudinal perpendicular bisector of the robot.

Both methods that were tested use front sonar sensors to determine the angle at the front wheels,  $\delta_f$ . The difference between the two methods is that the *4FR* method is the only one that uses the gyroscope to determine the absolute vehicle body orientation. The body orientation is consequently used to determine the absolute angle of the path using the memory function, leading to values of the rear path angle,  $\delta_r$ , as the rear axle crosses that point. This on-the-fly path memory function allows the rear wheels to approximately track an identical path as the front wheels, thus avoiding any obstacles. Of the two, the *4FM* algorithm was however deemed more convenient for this experimental setup because all wheel orientations are defined based on the front steer angle,  $\delta_f$ , which is given only by the sonar array located at the front of the vehicle. The *4FM* method uses the sonar array to find  $\delta_f$  and negates it to determine  $\delta_r$ . This implies that the ICR lies on the normal to the vehicle

heading which passes through the vehicle CG. This is a simpler approach because it does not require a gyroscope.

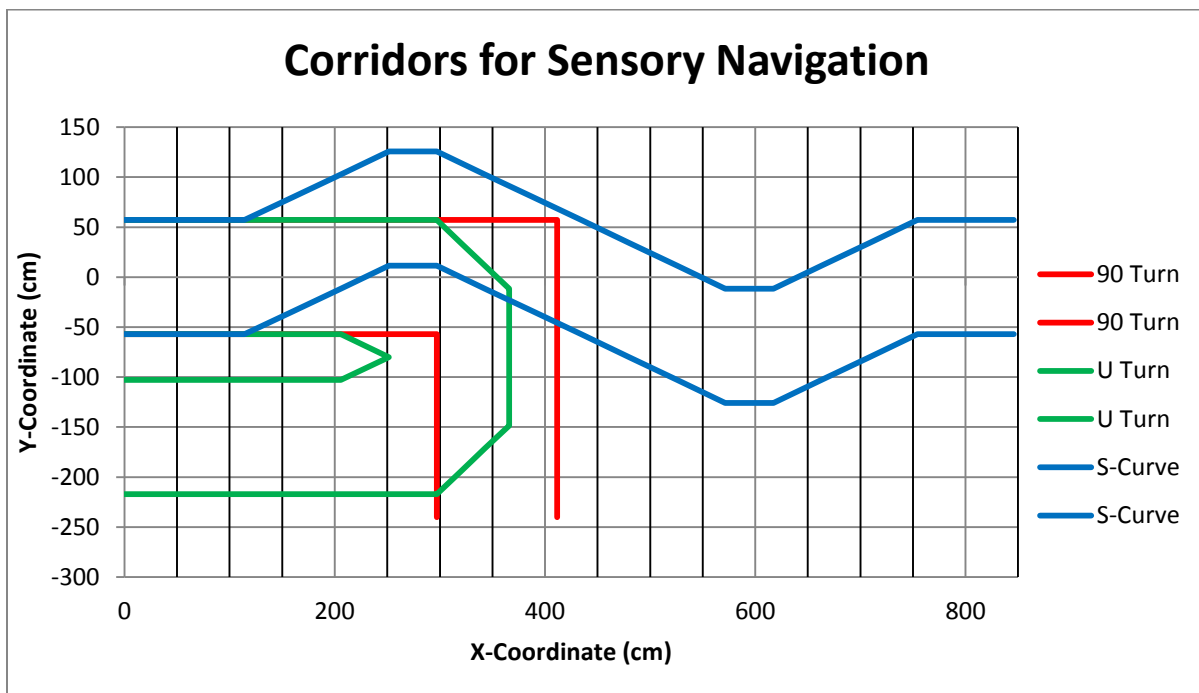
### 6.2.1. EXPERIMENTAL SET UP AND PROCEDURE

For each of the runs, a corridor was set up using pieces of 4" Ø PVC pipe cut to 24" lengths, spaced at 18" apart as shown in Figure 6.4. These PVC pipes provided adequate 'targets' for the sonar to range and were spaced closely enough that the robot interprets them as continuous walls. The front path angle,  $\delta_f$ , was computed from sonar measurements using the lever rule of equation (5.1). Because the vehicle velocity was set constant, a saturation condition was introduced so that the wheels would not 'flip' and attempt a zero radius turn. This maneuver was avoided because it takes considerably more time than standard maneuvering, as the wheels must rotate 180°, which isn't practical if a constant velocity is desired.



**FIGURE 6.4 - 90° TURN USING PVC PIPE TO CREATE CONFIGURABLE CORRIDORS**

Three paths were tested, including a U-Turn, a 90° corner, and the S-Curve simulating an obstruction or obstacle; all paths are shown in Figure 6.5. For this experiment, a 90° corner was supplanted for the Zig-Zag. The key characteristic of the Zig-Zag path is the cusps or sharp corners, that vehicles of large aspect ratios normally have a difficult time navigating. To assess the robot's ability to navigate these corners, it was desirable to have the robot interpret the boundary as having sharp corners.

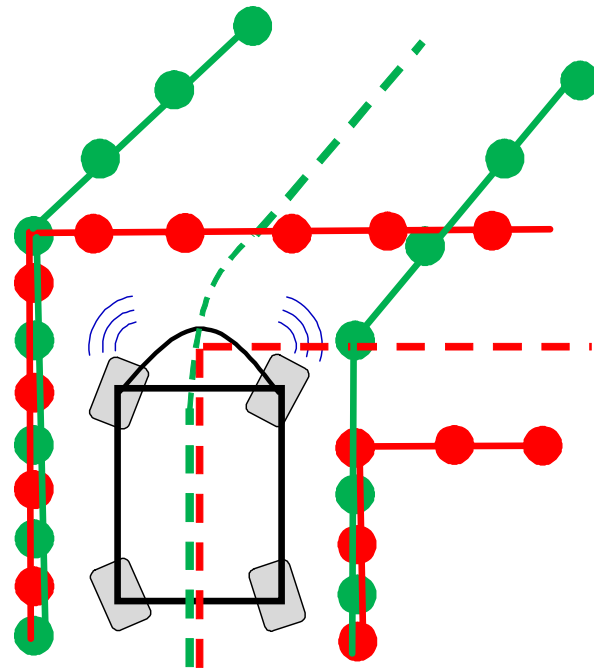


**FIGURE 6.5 – EXPERIMENTAL PATHS FOR SENSORY NAVIGATION**

Because the robot uses the sonar sensors, smaller angles are interpreted as rounder corners. Therefore to generate a noticeably discontinuous curve, only larger angles could be used; this concept is demonstrated in Figure 6.6. This is due to the fact that when the vehicle reaches the corner, to the sonar, the boundary abruptly disappears, causing the robot to fully steer in that direction until the boundary can be ranged again. For these



reasons, a 90° corner was chosen; ideally, this would have been made into a full Zig-Zag course but was not feasible due to the size of the experimental setting as well as the limited availability of corridor boundary targets. The 90° corner did successfully cause the robot to replicate the behavior exhibited when crossing the cusps in Experiment 1.



**FIGURE 6.6 – THE SONAR (BLUE) INTERPRETS SHALLOW ANGLES (GREEN) AS ROUND CORNERS AND LARGE ANGLES (RED) AS CUSPS**

Once the boundaries had been established and the appropriate steering algorithm loaded onto the VCU, the robot traversed the path, again marking the front and rear axle tracers with sand. The coordinates of the tracked path were recorded manually using the tile spacing on the floor as a reference. Any interference with the PVC corridor was noted as reflected in the results, shown in Appendix D. Again, this procedure was repeated three times for each path and steering configuration for a total of 18 runs.

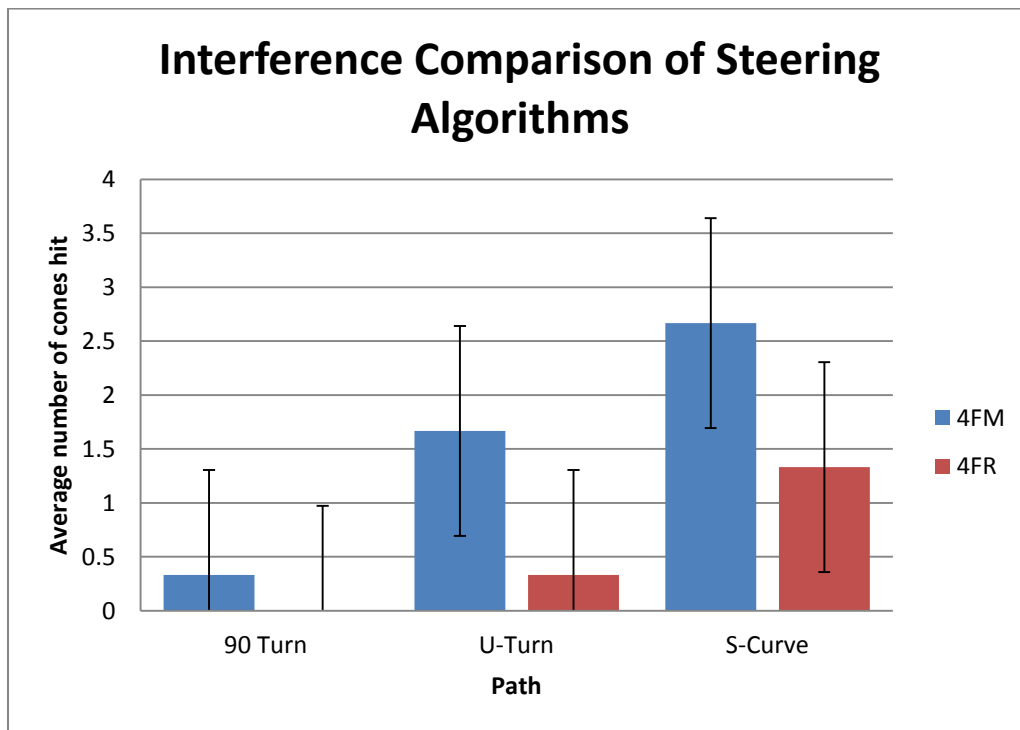
### 6.2.2. EXPERIMENTAL RESULTS

For this experiment, the relative performance measure of the robot was based upon the ability to navigate the tight corridor without any boundary interference. The exact path taken by the robot was less important compared to how often it was unable to avoid the obstacles. Therefore, the performance metric was chosen to be the number the boundary defining pipes that were interfered with by the robot throughout the navigation of the course. The pipes were regularly spaced and the robot performed well enough that only a marginal number of pipes were ever knocked over, yet there was enough of a discrepancy between the algorithms to draw a sound conclusion. The full results are shown in Appendix D and a summary of these results and observations is noted in this section.

From a qualitative standpoint, the robot seemed to perform better while it was using the *4FR* method than when it was tracking the front axle alone (*4FM*). Although, the *4FM* configuration was not as susceptible to sensor drift or error accumulation as the *4FR* algorithm because it did not rely upon the gyroscope. In the initial development of the experimental procedure, the corridor was much wider than the final one. In the wider case, both algorithms performed flawlessly. In fact, it was somewhat difficult to tell which algorithm was used based on the resulting path alone. For a wide corridor, the *4FM* is a much simpler, method that performs as good as the *4FR*, for a vehicle of this aspect ratio.

However, as the corridor was narrowed, the pitfalls of the *4FM* method became more evident. Because the wheels simply mirror each other, and the robot is only directing its front wheels, it is 'unaware' of where the rear of the vehicle is in relation to the path boundaries. Primarily, in cornering events, the rear end of the robot would swing in the

opposite direction; with a narrow corridor, this would usually cause interference. The *4FR* method however was able to navigate these boundaries with ease. The robot had the assurance of ‘knowing’ that if the front axle could pass through the boundary, the path memory function would ensure navigation of the rear end through that same gateway. The advantages of the *4FR* over *4FM* were exposed in all tight corners. It is expected that for a vehicle of a large aspect ratio, i.e. a wheelbase much larger than the wheel track, the disparity between the *4FR* and *4FM* would be even greater.



**FIGURE 6.7 – PERFORMANCE COMPARISON OF BIBOT-1 NAVIGATING A CORRIDOR**

By quantitative methods, the *4FR* performed 2-3 times better than the *4FM* as shown by the results in Figure 6.7. For the 90° corner the *4FR* method performed flawlessly; and in the other two paths it consistently performed better than *4FM*. Diagrams of the tracked paths, show the ‘bundles’, which indicate the path of the front and rear axles,

are much more tightly 'packed' for the *4FR* algorithm. This is an indicator that the controller did an effective job of navigating the rear axle along the same path as the front axle. For *4FM* tracking, the front and rear paths have little correlation. The swinging out of the rear axle is especially evident in the U-Turn route, Figure D.1. The results reflect exactly what was observed; in narrow routes the *4FR* method is superior. This however, comes with the price of increased instrumentation and susceptibility to error accumulation and drift.

### **6.3. SOURCES OF ERROR**

Throughout the experimentation, there are several sources of error. One of the largest sources of error is position drift from accumulation of smaller errors. Because the robot has no way to update its absolute position, all errors accumulate until the robot has completed its path. It was initially thought that the robot could use the 6-axis inertial measurement unit as a means of navigation. However after some simple experimentation, it was realized that this was not feasible. Any noise in the accelerometer is integrated twice causing an exponential growth of position error. These errors were observed to be on the order of kilometers after only a minute of operation, even after doing extensive calibration and introducing better error correction steps [101]. Many modern navigation systems typically use a Kalman filter to combine information from all axes and in some cases a GPS to eliminate noise and consequently drift. These filters and navigational systems themselves often become the subject of PhD dissertations. In the limited scope of time, it was not viable to use a positioning system like this, so dead reckoning methods were combined with a odometer for most applications.

The experiment was performed using sand as a path tracking medium. The sand may have contributed to wheel slip, causing the robot to misalign as it was navigating the paths. Furthermore, the mechanical platform itself displayed momentary backlashes in the steering gearboxes, which could cause large amounts of steering errors that may even outweigh the discrepancies between steering methods. Although these backlash errors were very significant in the early stages of the experimentation, they were resolved through properly adjusting the hidden set screws that adjust the gearbox backlash.

Finally, there may have been error due to the data collection methods. Because the sand was laid on a tile floor, these increments became the most convenient method of measurement. The tile spacing was 9", thus most of the data for the paths is collected at resolution of 9" intervals. At each interval, an attempt was made to estimate the intersection of the line with the tile to the nearest inch. This coarse resolution may have been too granular to fully represent the individual paths.

## 7. CONCLUDING REMARKS

This thesis has investigated path tracking control algorithms for a four wheeled robotic vehicle with independently driven and independently steered wheels (AWS/AWD). It was part of the larger problem of path tracking of self-reconfigurable robotic vehicles with prismatically articulated suspension for use in highly irregular terrains that are difficult to traverse by normal vehicles and humans. Controlling these vehicles is still a challenging problem because of their overactuation, which makes it difficult to satisfy the instantaneous center of rotation for rigid bodies.

After a brief history of past work in automobile and robotic kinematic developments including different drive train and steering configurations and their implications, the research began by studying the general four wheeled independently driven, independently steered robotic vehicle with prismatic articulated suspension. Both the kinematics and dynamics of this vehicle were studied; it was found that the kinematic dynamic model was complex with a dimension of  $26 \times 20$  and the rigid body kinematic constraint was even more complex. The problem is simplified by reducing this system to an independent AWS/AWD vehicle with a fixed suspension, which reduces the dimension of the system to  $14 \times 8$ . A new set of kinematic constraints was developed to constrain all steered wheels of the vehicle to remain at the center of the path while satisfying the instantaneous center of rotation condition. These constraints are defined using the vehicle speed, and the geometry of the path as well as that of the vehicle. These constraints were then applied in developing an optimal path control algorithm that minimizes the vehicle deviation from the path center in a decentralized control architecture.

The proposed algorithm is tested by both numerical simulation and experimentally using an in-house BIBOT-1 robot. Details of the numerical simulation and the experimentation are discussed, along with the observed results. The performance of the proposed path tracking algorithm are compared with other path tracking algorithms in a variety of paths. In general the proposed algorithm, *4FR*, outperforms all other methods that were used in this study with fewer path tracking errors. Although the experimental results are likely to have been affected by a variety of sensor uncertainties, especially for those algorithms that relied heavily on sensor data, results obtained from the using the proposed algorithm support the need to have overactuation, whose advantages are most distinguishable in highly demanding paths, where the margin for error is very small.

### **7.1. FUTURE WORK**

Throughout the experimentation, several opportunities for improvement were observed. The experimental platform would be most dramatically improved by an absolute positioning system such as GPS for outdoor navigation or a system like the STARGAZER [103] for indoor localization. The weakness of these systems is that they require beacons such as satellites or previously dispersed light signals. This makes the system impractical for unfamiliar indoor or extraterrestrial environments. However, the dead reckoning system used in the experiments as well as the inertial navigation system initially proposed are much too vulnerable to positioning error accumulation to be feasible for distances of more than 30 meters.

Another improvement to the robot would be individual wheel encoders. Currently, the robot sets the duty of the pulse width modulation controller for the hub motors. It is

therefore assumed that the wheel speed is equally linearly proportional to this duty for all wheels, when in fact this is not the case. The individual wheel encoders would allow the WCUs to employ a feedback controller for the wheel speed which would further reduce errors.

The last improvement would be for the steering motor controller to be upgraded to a more flexible microcontroller. Currently, the microcontroller only allows for the input of two signals, namely on/off and direction. This makes fine positioning of the wheels difficult and doesn't allow for variable speeds of the steering motor. Although it is a feedback control system, this limits the parameters that can be used to rigorously control the steering motor, which may further reduce the errors.

The next step forward in the research would be to test this steering algorithm in conjunction with an articulating suspension on a self-reconfigurable robot such as the BIBOT-2. The 3D environment would pose new challenges that must be addressed before this technology will be suitable for operational use. However, at this time, the research has the potential for being developed into an extraterrestrial rover, an emergency response tool or a domestic assistant. The possibilities are endless for a self-reconfigurable autonomous robotic vehicle with 3 dimensional navigational capabilities.



## 8. REFERENCES

- [1] R. Orloff, *Apollo By the Numbers: A Statistical Reference*, NASA, 2000.
- [2] G. Fielden, *The Streamline Hotel and the Birth of NASCAR*, Lincolnwood: Publications International, 2005.
- [3] Office of Technology Assessment, *Saving energy in U.S. transportation*, Washington D.C.: Congress of the United States, 1994.
- [4] G. N. Georgano, *Cars: Early and Vintage, 1886-1930*, London: Grange-Universal, 1985.
- [5] L. Weeks, *The History of the Automobile and its Inventors*, Bremen: TEC Books, 2010.
- [6] P. A. Simionescu and D. Beale, "Optimum synthesis of the four-bar function generator in its symmetric embodiment: the Ackermann steering linkage," *Mechanism and Machine Theory*, vol. 37, no. 12, pp. 1487-1504, 2002.
- [7] S. Sano, Y. Furukawa and S. Shiraishi, "Four Wheel Steering System with Rear Wheel Steer Angle Controlled as a Function of Steering Wheel Angle," Society of Automotive Engineers, Honda R&D Co., 1986.
- [8] J. C. Whitehead, "Four Wheel Steering: Maneuverability and High Speed Stabilization," Society of Automotive Engineers, Davis, 1988.
- [9] S. Sano, Y. Furukawa and Y. Oguchi, "Effect of Vehicle Response Characteristics and Driver Skill Level on Task Performance and Subjective Rating," in *Proceedings of the Eighth International Technical Conference on Experimental Safety Vehicles*, Wolfsburg, 1980.
- [10] T. Takiguchi, N. Yasuda, S. Furutani, H. Kanazawa and H. Inoue, "Improvement of Vehicle Dynamics by Vehicle-Speed-Sensing Four-Wheel Steering System," Society of Automotive Engineers, Detroit, 1986.
- [11] Spyker Co., "Excerpt from Spyker Press Release," [Online]. Available: <http://www.seriouswheels.com/cars/top-1903-Spyker-60-HP.htm>. [Accessed 11 4 2011].
- [12] D. B. Wise, "Cord: The Apex of a Triangle," *World of Automobiles*, vol. 4, pp. 435-437, London. 1974.

- [13] M. Hoeck and C. Gasch, "The influence of various 4WD driveline configurations on handling and traction," [Technical Report] Society of Automotive Engineers, 1999.
- [14] General Automotive Servicenter, [Online]. Available: <http://www.genautoinc.com/differentialservice.cfm>. [Accessed 18 September 2011].
- [15] Lexus, "GS 350F Sport," [Online]. Available: <http://www.lexus.com/models/GS/performance/fsport.html#p>. [Accessed 7 March 2013].
- [16] J. Alexander and J. Maddocks, "On the Kinematics of Wheeled Mobile Robots," *The International Journal of Robotic Research*, vol. 8, no. 5, pp. 15-27, 1989.
- [17] J. C. Alexander and M. J., "On the Maneuvering of Vehicles," *SIAM Journal of Applied Mathematics*, vol. 48, no. 1, pp. 38-51, 1988.
- [18] J. Ploeg, J. P. M. Vissers and H. Nijmeijer, "Control Design for an Overactuated Wheeled Mobile Robot," in *4th IFAC Symposium on Mechatronic Systems*, Helmond, 2006.
- [19] D. King-Hele, "Erasmus Darwin's Improved Design for Steering Carriages--And Cars," *Notes and Records of The Royal Society*, vol. 56, no. 1, pp. 41-62, London.
- [20] Diandra, "Stock Car Science," [Online]. Available: <http://www.stockcarscience.com/blog/index.php/differentials>. [Accessed 20 October 2011].
- [21] K. Nice, "How Stuff Works," Discovery Company, [Online]. Available: <http://auto.howstuffworks.com/four-wheel-drive4.htm>. [Accessed 12 October 2011].
- [22] N. Stafford, "Off-Road Adventures," [Online]. Available: <http://www.oramagazine.com/pastissues/0410-issue/041008t-ected-qt.html>. [Accessed 20 November 2011].
- [23] W. Peschke, "A Viscous Coupling in the Drive Train of an All-Wheel-Drive Vehicle," SAE Technical Paper, Detroit, 1986.
- [24] S.-i. Sakai, "Motion Control in an Electric Vehicle with Four Independently Driven In-Wheel Motors," *IEEE/ASME Transactions on Mechatronics*, vol. 4, no. 1, pp. 9-16, 1999.
- [25] W. G. Walter, "An electromechanical animal," *Dialectica*, vol. 4, pp. 42-49, 1950.

- [26] R. Choate and L. D. Jaffe, "Science aspects of a remotely controlled mars surface roving vehicle," in *Proceedings of the First National Conference, Remotely manned systems: Exploration and operation in space*, Pasadena, CA, 1972.
- [27] G. Giralt, R. Sobek and R. Chatila, "A multi-level planning and navigation system for a mobile robot: A first approach to HILARE," in *6th Internatinoal Joint Conference on Artificial Intelligence*, Tokyo, Japan, 1979.
- [28] J. M. Holland, "Rethinking Robot Mobility," *Robotics Age*, vol. 7, no. 1, pp. 26-30, January 1985.
- [29] R. Wallace and e. al, "First Results in Robot Road-Following," in *Proceedings of the IJCAI*, Los Angeles, CA, 1985.
- [30] G. Podnar, K. Dowling and M. Blackwell, "A Functional Vehicle for Autonomous Mobile Robot Research," Pittsburgh, PA, 1983.
- [31] M. Bajracharya, M. W. Maimone and D. Helmick, "Autonomy for Mars rovers: past, present, and future," *IEEE Computer Society*, vol. 41, no. 12, pp. 44-50, December 2008.
- [32] R. Siegwart, P. Lamon, T. Estier, M. Lauria and R. Piguet, "Innovative design for wheeled locomotion in rough terrain," *Robotics and Autonomous Systems*, vol. 40, no. 2-3, pp. 151-162, 2002.
- [33] G. Campion, G. Bastin and D.-N. Brigitte, "Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 47-62, 1996.
- [34] P. F. Muir and C. P. Neuman, "Kinematic Modeling of Wheeled Mobile Robots," *Journal of Robotic Systems*, vol. 4, no. 2, pp. 281-340, 1987.
- [35] O. Diegel, A. Badve, G. Bright, J. Potgieter and S. Tlale, "Improved Mecanum Wheel Design for Omni-directional Robots," in *Australiasian Conference on Robotics and Automation*, Auckland, 2002.
- [36] J. Goerner, R. Hollis, G. Kantor, K. Karthikeyan, B. Kim, M. Kumagai, T. Lauwers, J. Xian Leong, A. Mampetta, U. Nagarajan, S. Nidhiry, K. Rivard, E. Schearer and L. Yi, "Dynamically-Stable Mobile Robots in Human Environments," Carnegie Mellon, 9 August 2006. [Online]. Available: <http://www.msl.ri.cmu.edu/projects/ballbot/>. [Accessed 18 March 2011].
- [37] "Robot Shop," RobotShop Distribution, [Online]. Available: <http://www.robotshop.com/100mm-left-mecanum-wheel.html>. [Accessed 14 October 2011].

- [38] Segway Inc., "Segway PT," 2012. [Online]. Available: <http://www.segway.com/compatibility/?adid=puma>. [Accessed 1 February 2012].
- [39] F. C. A. Groen, B. J. A. Krose, S. t. Hagen, N. Vlassis and R. Bunschoten, "Intelligent Systems Lab Amsterdam," 25 January 2010. [Online]. Available: <http://www.science.uva.nl/research/isla/themes/learning/rwcp/>. [Accessed 1 May 2011].
- [40] NASA, "Jet Propulsion Laboratory," California Institute of Technology, [Online]. Available: <http://www-robotics.jpl.nasa.gov/systems/system.cfm?System=4#clifford>. [Accessed 1 May 2011].
- [41] H. Moravec, "Mobile Robot Lab," Carnegie Mellon University Robotics Institute, [Online]. Available: [http://www.ri.cmu.edu/research\\_project\\_detail.html?project\\_id=48&menu\\_id=261](http://www.ri.cmu.edu/research_project_detail.html?project_id=48&menu_id=261). [Accessed 1 May 2011].
- [42] S. Thrun, "Stanford Racing," Stanford University, 14 August 2006. [Online]. Available: <http://cs.stanford.edu/group/roadrunner//old/index.html>. [Accessed 1 May 2011].
- [43] Denning Branch International, "Products," 16 January 2005. [Online]. Available: <http://www.southcom.com.au/~robot/products.html>. [Accessed 5 May 2011].
- [44] R. Bischoff and V. Graefe, "HERMES - an intelligent Humanoid Robot Designed and Tested for Dependability," *Springer Tracts in Advanced Robotics*, vol. 5, pp. 64-74, 2003.
- [45] Wowee Electronics, "Products - Rovio," 2011. [Online]. Available: <http://www.wowwee.com/en/products/tech/telepresence/rovio/rovio>. [Accessed 1 May 2011].
- [46] G. Podnar and M. Blackwell, "Carnegie Mellon Computer Science," 1985. [Online]. Available: <http://www.cs.cmu.edu/afs/cs/user/gwp/www/robots/Uranus.html>. [Accessed 1 May 2011].
- [47] M. Yim, D. Duff and K. Roufas, "Polybot: A modular, reconfigurable robot," in *IEEE International Conference on Robotics and Automation*, San Fransisco, CA, 2000.
- [48] S. Murata and H. Kurokawa, "Self Reconfigurable Robots," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 71-78, March 2007.

- [49] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins and G. Chirikjian, "Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]," *IEEE Robotics and Automation*, vol. 14, no. 1, pp. 43-52, March 2007.
- [50] C. R. Weisbin, D. Lavery and G. Rodriguez, "Robotic Technology for Planetary Missions into the 21st Century," Jet Propulsion Lab, NASA, Pasadena, 1997.
- [51] Boston Dynamics, "Big Dog," [Online]. Available: [http://www.bostondynamics.com/robot\\_bigdog.html](http://www.bostondynamics.com/robot_bigdog.html). [Accessed 20 November 2011].
- [52] R. Siegwart and I. R. Nourbakhsh, *Autonomous Mobile Robots*, Cambridge: Massachusetts Institute of Technology, 2004.
- [53] NASA, "Commercial Rovers," [Online]. Available: <http://www-robotics.jpl.nasa.gov/systems/system.cfm?System=4#clifford>. [Accessed 20 November 2011].
- [54] BlueBotics, "Shrimp III," [Online]. Available: [http://www.bluebotics.com/solutions/Shrimp/Shrimp\\_web.pdf](http://www.bluebotics.com/solutions/Shrimp/Shrimp_web.pdf). [Accessed 20 November 2011].
- [55] P. F. Muir and C. P. Neuman, "Kinematic Modeling for Feedback Control of an Omnidirectional Wheeled Mobile Robot," in *1987 IEEE International Conference on Robotics and Automation*, Pittsburgh, 1987.
- [56] H. Baruh, *Analytical Dynamics*, Boston: McGraw-Hill International, 1998.
- [57] M. Tarokh, L. Mireles and G. McDermott, "Two approaches to kinematics modelling of articulated rovers," San Diego State University, 2005.
- [58] A. Betourne and G. Champion, "Dynamic modelling and control design of a class of omnidirectional mobile robots," in *IEEE International Conference of Robotics and Automation*, Minneapolis, 1996.
- [59] M. Burckhardt and J. Reimpell, "Fahrwerktechnik: Radschlupf-Regelsysteme," Germany, 1993.
- [60] H. Pacejka and E. Bakker, "The magic formula tyre model," *Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility*, vol. 21, no. 1, pp. 1-18, 1992.
- [61] M. F. Selekwa and J. Nistler, "Path Tracking Control of Four Wheel Independently Steered Ground Robotic Vehicles," in *IEEE International Conference on Decision and Control / European Control Conference*, Orlando, 2011.

- [62] J. Nistler and M. F. Selekwa, "Constrained Path Tracking Control of Four Wheel Steered, Four Wheel Drive Autonomous Ground Robots," in *Florida Conference on Recent Advances in Robotics*, Gainesville, 2011.
- [63] S. Sano, "Four Wheel Steering System with Rear Wheel Steer Angle Controlled as a Function of Steering Wheel Angle," Society of Automotive Engineers, Honda R&D Co., 1986.
- [64] Y. Furukawa, N. Yuhara, S. Sano, H. Takeda and Y. Matushita, "A Review of Four-Wheel Steering Studies from the Viewpoint of Vehicle Dynamics and Control," *Vehicle System Dynamics*, vol. 18, no. 1-3, pp. 151-186, 1989.
- [65] C. P. Leger, T.-O. Whiteys, J. R. Wright, S. A. Maxwell and R. G. Bonitz, "Mars Exploration Rover Surface Operations: Driving Spirit at Gusev Crater," *IEEE conference on Systems, Man, Cybernetics*, vol. 2, pp. 1815-1822, October 2005.
- [66] J. Agull, S. Cardona and J. Vivancos, "Kinematics of vehicles with directional sliding wheels," *Mechanism and Machine Theory*, vol. 22, no. 4, pp. 295-301, 1987.
- [67] J. Agullo, S. Cardona and J. Vivancos, "Dynamics of Vehicles with Directionally Sliding Wheels," *Mechanism and Machine Theory*, vol. 24, no. 1, pp. 53-60, 1989.
- [68] R. Balakrishna and A. Ghosal, "Modeling of slip for wheeled mobile robots," *Robotics and Automation*, vol. 11, no. 1, pp. 126-132, 1995.
- [69] Campion, "Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 47-62, 1996.
- [70] K. Kanjanawanishkul and Z. A, "Path following for an omnidirectional mobile robot based on model predictive control," in *IEEE International Conference on Robotics and Automation*, Kobe, 2009.
- [71] J. Vazquez and M. Velasco-Villa, "Computed -Torque Control of an Omnidirectional Mobile Robot," in *IEEE 4th International Conference on Electrical and Electronics Engineering*, Mexico City, 2007.
- [72] N. Matsumoto and M. Tomizuka, "Vehicle lateral velocity and yaw rate control with two independent control inputs," *Journal of Dynamic System Measurements and Control*, vol. 114, no. 4, pp. 606-614, 1992.
- [73] J. Ackermann, "Robust Yaw Damping of Cars with Front and Rear Wheel Steering," *IEEE Transactions on Control Systems Technology*, vol. 1, no. 1, pp. 15-20, 1993.
- [74] J. Borenstein, "Control and Kinematic Design of Multi-Degree-of-Freedom Mobile Robots with Compliant Linkage," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 1, pp. 21-35, 1995.

- [75] X. Gao, B. McVey and R. Tokar, "Robust controller design of four wheel steering systems using u-synthesis techniques," in *Proceedings of the 34th IEEE Conference on Decision and Control*, New Orleans, 1995.
- [76] M. A. Vilaplana, O. Mason, D. J. Leith and W. E. Leithead, "Control of Yaw Rate and Sideslip in 4-wheel Steering Cars with Actuator Constraints," *Switching and Learning Feedback Systems*, vol. 3355, no. Lecture Notes in Computer Science, R. Murray-Smith and R. Shorten, Eds. Springer Berlin/Heidelberg, pp. 201-222, 2005.
- [77] M. Wada, "Virtual Link Model for Redundantly Actuated Holonomic Omnidirectional Mobile Robots," in *IEEE International Conference on Robotics and Automation*, Orlando, 2006.
- [78] M. Lauria, I. Nadeua, P. Lepage, Y. Morin, P. Giguere, F. Gagnon, D. Letourneau and F. Michaud, "Design and control of a four steered wheeled mobile robot," in *32nd Annual IEEE Conference on Industrial Electronics*, Paris, 2006.
- [79] K. S. Byun and S. J. Kim, "Design of a Four-Wheeled Omnidirectional Mobile Robot with Variable Wheel Arrangement Mechanism," in *IEEE International Conference on Robotics and Automation*, 2002.
- [80] C. Leng and Q. Cao, "Velocity analysis of omnidirectional mobile robot and system implementation," in *IEEE International Conference on Automation Science and Engineering*, Shanghai, 2006.
- [81] C. C. Tsai, Z. Wu, C. Wang and M. F. Hisu, "Adaptive dynamic motion controller design for a four-wheeled omnidirectional mobile robot," in *2010 International Conference on System Science and Engineering*, Taipei, 2010.
- [82] O. Purwin and R. D'Andrea, "Trajectory generation and control for four wheeled omnidirectional vehicles," *Robotics and Autonomous Systems*, vol. 54, no. 1, pp. 13-22, 2006.
- [83] W. Langson and A. Alleyne, "Multivariable bilinear vehicle control using steering and individual wheel torques," in *Proceedings of the 1997 American Control Conference*, Albuquerque, 1997.
- [84] Y. Yavin, "Modelling the motion of a car with four steerable wheels," *Mathematical and Computer Modelling*, vol. 38, no. 10, pp. 1029-1036, 2003.
- [85] C. Connette, C. Parlitz, M. Hagele and A. Verl, "Singularity avoidance for over-actuated, pseudo-omnidirectional, wheel mobile robots," in *IEEE International Conference on Robotics and Automation*, Kobe, 2009.

- [86] M. Makatchev, J. J. McPhee, S. K. Tso and L. S. Y. T, "System design, modelling and control of a four-wheel-steering mobile robot," in *Proceedings of the 19th Chinese Control Conference*, 2000.
- [87] H. Itoh and A. Oida, "Dynamic analysis of turning performance of 4wd-4ws tractor on paved road," *Journal of Terramechanics*, vol. 27, no. 2, pp. 125-143, 1990.
- [88] H. Itoh, A. Oida and M. Yamazaki, "Numerical simulation of a 4wd-4ws tractor turning in a rice field," *Journal of Terramechanics*, vol. 36, no. 2, pp. 91-115, 1999.
- [89] S. T. Peng, J. J. Sheu and C. C. Chang, "On One Approach to Constraining Wheel Slip for the Autonomous Control of 4WS4WD Vehicle," in *IEEE International Conference on Control Applications*, Taipei, 2004.
- [90] P. Boyd, "Light Vehicle Rollover Presentation," 2001. [Online]. Available: <http://icsw.nhtsa.gov/cars/problems/studies/NASRoll/>. [Accessed 15 March 2012].
- [91] E. Maalouf, M. S. Saad and S. H, "A higher level path tracking controller for a four-wheel differentially steered mobile robot," *Robotics and Autonomous Systems*, vol. 54, no. 1, pp. 23-33, 2006.
- [92] R. DeSantis, "Path-tracking for a Tractor-Trailer-like Robot: Communication," *The International Journal of Robotics Research*, vol. 13, no. 6, pp. 533-544, 1994.
- [93] J. Normey-Rico, I. Alcalá, J. Gomez-Ortega and E. Camacho, "Mobile robot path tracking using a robust PID controller," *Control Engineering Practice*, vol. 9, no. 11, pp. 1209-1214, 2001.
- [94] A. Lipton and S. Kagami, "Real-World Path Following for a Monocular Vision Based Autonomous Mobile Robot in a 'Remote-Brain' Environment," psu.edu, Citeseer, Clayton, Australia.
- [95] N. Gupta, "Implementation of Particle Model Control Approach to a Fixed Axle UGV," North Dakota State University, Fargo, 2009.
- [96] G. Reina, L. Ojeda and A. B. J. Milella, "Wheel Slippage and Sinkage Detection for Planetary Rovers," *IEEE/ASME Transactions on Mechatronics*, vol. 11, no. 2, pp. 185-195, 2006.
- [97] D. Wang, "Trajectory Planning for a Four-Wheel-Steering Vehicle," in *IEEE International Conference on Robotics and Automation*, Seoul, 2001.
- [98] G. Ishigami and Miwa, "Steering Trajectory Analysis of Planetary Exploration Rovers Based on All-Wheel Dynamics Model," Department of Aerospace Engineering, Tohoku University, Sendai, 2005.



- [99] J. Ehlen, D. Henderson, L. Schraw, K. Watson, C. Nelson and S. Wala, "Design of an AWD AWS Autonomous Vehicle," North Dakota State University (unpublished), Fargo, 2006.
- [100] V. Singh, A. Vander Vorst, A. Zuther and B. Hest, "AWD AWS Autonomous Robotic Vehicle," NDSU (unpublished), Fargo, 2008.
- [101] J. Nistler and M. F. Selekwa, "Gravity Compensation in Accelerometer Measurements for Robot Navigation on Inclined Surfaces," *Procedia Computer Science*, vol. 6, pp. 413-418, 2011.
- [102] T. Eiter and H. Mannila, "Computing Discrete Frechet Distance," CiteSeerX, Wien, 1994.
- [103] Hagisonic, January 2012. [Online]. Available: <http://www.hagisonic.com/>.

## APPENDIX A. MATLAB STEERING SIMULATION CODE

### A.1. FEEDBACK CONTROLLER SIMULATION

#### A.1.1. CONTROL1.M

```
load angles.mat;
N=length(CGx);
Steps=2;
TotalTime=sum((diff(CGx).^2+diff(CGy).^2).^(0.5));
DT=TotalTime/(Steps*N);
vg=1;
V1=v1(1);
D1=d1(1);
V2=v2(1);
D2=d2(1);
V3=v3(1);
D3=d3(1);
V4=v4(1);
D4=d4(1);
BT1=0;
BT2=0;
BT3=0;
BT4=0;
MR=50.0;
rw=0.085;
Iwr=0.025;
Iws=0.009;
mw=3.5;
W=0.75;
H=1.0;
Tt1=0;
Tt2=0;
Tt3=0;
Tt4=0;
Ts1=0;
Ts2=0;
Ts3=0;
Ts4=0;

for k=1:N
    DF=df(k);
    DR=dr(k);

    dref1=acot(cot(DF)+(W/(2*H))*cot(DF)*(tan(DF)-tan(DR)));
    dref2=acot(cot(DF)-(W/(2*H))*cot(DF)*(tan(DF)-tan(DR)));
    dref3=acot(cot(DR)-(W/(2*H))*cot(DR)*(tan(DF)-tan(DR)));
    dref4=acot(cot(DR)+(W/(2*H))*cot(DR)*(tan(DF)-tan(DR)));
    Vref1=vg*tan(DF)*csc(dref1)/sqrt(1+.25*(tan(DF)+tan(DR))^2);
    Vref2=vg*tan(DF)*csc(dref2)/sqrt(1+.25*(tan(DF)+tan(DR))^2);
    Vref3=vg*tan(DR)*csc(dref3)/sqrt(1+.25*(tan(DF)+tan(DR))^2);
```

```

Vref4=vg*tan(DR)*csc(dref4)/sqrt(1+.25*(tan(DF)+tan(DR))^2);
PKT=120;
PKS=230;
for n=1:Steps
    TT=PKT*(Vref1-V1(length(V1)));
    Tt1=[Tt1, TT];
    Vn=V1(length(V1));
    V1=[V1, Vn+DT*4*rw*TT/(MR*rw^2+4*(Iwr+mw*rw^2))];
    TT=PKT*(Vref2-V2(length(V2)));
    Tt2=[Tt2, TT];
    Vn=V2(length(V2));
    V2=[V2, Vn+DT*4*rw*TT/(MR*rw^2+4*(Iwr+mw*rw^2))];
    TT=PKT*(Vref3-V3(length(V3)));
    Tt3=[Tt3, TT];
    Vn=V3(length(V3));
    V3=[V3, Vn+DT*4*rw*TT/(MR*rw^2+4*(Iwr+mw*rw^2))];
    TT=PKT*(Vref4-V4(length(V4)));
    Tt4=[Tt4, TT];
    Vn=V4(length(V4));
    V4=[V4, Vn+DT*4*rw*TT/(MR*rw^2+4*(Iwr+mw*rw^2))];

    bn=BT1(length(BT1));
    TS=PKS*(dref1-D1(length(D1)));

    Ts1=[Ts1, TS];
    BT1=[BT1, bn+DT*TS/Iws];

    bn=BT2(length(BT2));
    TS=PKS*(dref2-D2(length(D2)));

    Ts2=[Ts2, TS];
    BT2=[BT2, bn+DT*TS/Iws];

    bn=BT3(length(BT3));
    TS=PKS*(dref3-D3(length(D3)));

    Ts3=[Ts3, TS];
    BT3=[BT3, bn+DT*TS/Iws];

    bn=BT4(length(BT4));
    TS=PKS*(dref4-D4(length(D4)));

    Ts4=[Ts4, TS];
    BT4=[BT4, bn+DT*TS/Iws];

    dn=D1(length(D1));
    D1=[D1, dn+DT*BT1(length(BT1))];

    dn=D2(length(D2));
    D2=[D2, dn+DT*BT2(length(BT2))];

    dn=D3(length(D3));
    D3=[D3, dn+DT*BT3(length(BT3))];

```

```

        dn=D4(length(D4));
        D4=[D4,dn+DT*BT4(length(BT4))];
    end
    %Integration
end;
XM=linspace(0,7,length(d1));
XN=linspace(0,7,length(D1));
figure(1)
    plot(XM,d1,'b-',XM,d2,'g-',XM,d4,'r-',XM,d3,'k-');
    legend('\delta_{1-th}','\delta_{2-th}','\delta_{3-th}','\delta_{4-th}')
    ylabel('Wheel angle [radians]');
    xlabel('x-position [meters]');
    grid
    figure(2)
    plot(XM,v1,'b-',XM,v2,'g-',XM,v4,'r-',XM,v3,'k-');
    legend('V_{1-th}','V_{2-th}','V_{3-th}','V_{4-th}')
    ylabel('Wheel Speed [meters/sec]');
    xlabel('x-position [meters]');
    grid

figure(3)
    plot(XM,d1,'c--',XN,D1,'b-',XM,d2,'m--',XN,D2,'g-',XM,d4,'y--',XN,D3,'r-',
    XM,d3,'gr--',XN,D4,'k-');
    legend('\delta_{1-th}','\delta_{1-contr}','\delta_{2-th}','\delta_{2-
    contr}','\delta_{3-th}','\delta_{3-contr}','\delta_{4-th}','\delta_{4-contr}')
    ylabel('Wheel angle [radians]');
    xlabel('x-position [meters]');
    grid
    figure(4)
    plot(XM,vg*v1,'c--',XN,V1,'b-',XM,vg*v2,'m--',XN,V2,'g-',XM,vg*v4,'y--',XN,V3,'r-',
    XM,vg*v3,'gr--',XN,V4,'k-');
    legend('V_{1-th}','V_{1-contr}','V_{2-th}','V_{2-contr}','V_{3-th}','V_{3-
    contr}','V_{4-th}','V_{4-contr}')
    ylabel('Wheel Speed [meters/sec]');
    xlabel('x-position [meters]');
    grid
    figure(5);
    plot(XN,Tt1,'b-',XN,Tt2,'g:',XN,Tt3,'r--',XN,Tt4,'k-.');
    legend('\tau_{T1}','\tau_{T2}','\tau_{T3}','\tau_{T4}');
    ylabel('Traction Torque [N.m]');
    xlabel('x-position [meters]');
    grid
    figure(6);
    plot(XN,Ts1,'b-',XN,Ts2,'g:',XN,Ts3,'r--',XN,Ts4,'k-.');
    legend('\tau_{S1}','\tau_{S2}','\tau_{S3}','\tau_{S4}');
    ylabel('Steering Torque [N.m]');
    xlabel('x-position [meters]');
    grid
    figure(7)
    plot(XM,d1,'b-',XM,d2,'g-',XM,d4,'r-',XM,d3,'k-');
    legend('\delta_{1-th}','\delta_{2-th}','\delta_{3-th}','\delta_{4-th}')
    ylabel('Wheel angle [radians]');
    xlabel('x-position [meters]');

```

grid

## A.2. 2WS CONFIGURATION

### A.2.1. ANIMATE.M

```
function animate(option)

%=====
% Author: Jon Nistler
% Date: September 25, 2009
% For: HW3

%=====
global carHandle slopef sloper x y
global x2f y2f x1r y1r hnow drnow
global line1Handle
global line2Handle
global line3Handle
global line4Handle
global wheel1Handle
global wheel2Handle
global wheel3Handle
global wheel4Handle
global w1xpos w1ypos w1rot phi1 w1speed
global w2xpos w2ypos w2rot phi2 w2speed
global w3xpos w3ypos w3rot phi3 w3speed
global w4xpos w4ypos w4rot phi4 w4speed

%=====

global base track sizeindex heading
global rearx reary frontx fronty ICx ICy
base=1; %wheelbase of vehicle
track=base*.75;
velocity=1;

cardiag=(((track/2)^2+base^2)^.5);
carang=atan((track/2)/base);

points=500; %number of points to use
pathend=15;

%%initialize on first call
if nargin<1
    option = 'initialize';
end;

%=====
% initialize
%=====

if strcmp(option,'initialize')
```

```

tic
%initialize the program
clf; % clears the figure window
hold on; hold off; % clean up any previous holds
% create the figure window
fig1 = figure(1); % get the handle to a new figure window
set(fig1,'Position',[50,50,600,600],...
      'NumberTitle','off','Name',...
      'AWS Robot Animation');
set(gca,'drawmode','fast');
set(fig1,'Backingstore','off');

%=====
% Menu
%=====
set(gcf,'MenuBar','none');

%File menu
file_menu = uimenu(fig1,'label','File');
print_option = uimenu(file_menu,'Label','Print',...
                      'Callback','printdlg');
export_option = uimenu(file_menu,'Label','Export',...
                       'Callback','print -djpeg');
exit_option = uimenu(file_menu,'Label','Exit','Callback','exit');

>Action menu
action_menu = uimenu(fig1,'Label','Actions');
animate_option = uimenu(action_menu,'Label','Animate',...
                        'Callback','animate(''animate'')');

%Initialize coordinate system
plot(0,0,'k','linewidth',1.5)
hold on
plot(0,0,'r-','linewidth',1.5)
plot(0,0,'c.')
legend('Tracking Path','CG Path','Instant Center Path','Location','NorthWest')
axis ([0 7.5 0 7.5]);
axis square
axis('on');
grid('on');
title('Animation Showing Instant Center for Steering Control');
xlabel('X Coordinate');
ylabel('Y Coordinate');

%Initialize 'gate' graphics
carHandle=0;
car(translate(0,0));

wheel1Handle=0;
wheel1(translate(0,0));

wheel2Handle=0;
wheel2(translate(0,0));

```

```

wheel3Handle=0;
wheel3(translate(0,0));

wheel4Handle=0;
wheel4(translate(0,0));

line1Handle=0;
line1(0,0,0,0);

line2Handle=0;
line2(0,0,0,0);

line3Handle=0;
line3(0,0,0,0);

line4Handle=0;
line4(0,0,0,0);

% =====
% Calculation of Coordinates
% =====
x=linspace(0,7.5,points+1); %curve to follow
y=curvy(x);
%   x=.5.*x;
%   y=.5.*y;
dx=x(2)-x(1);
%first find point that is closest to car distance away
for i=2:1:points;
    min=inf;
    length=((x(i)-x(1))^2+(y(i)-y(1))^2)^.5;
    diff=base-length;

    if diff<min
        min=diff;
        start=i;
        sizeindex=points-start;
    end

    if diff<0 %break when get further than car length
        break
    end
end

%Establish front XY coordinates
count=1;
for i=start:points;

    frontx(count)=x(i);
    fronty(count)=y(i);
    slopef(count)=(y(i+1)-y(i-1))/(2*dx);

    %find where rear of vehicle is

```

```

min=inf;%reinitialize min for minimum search

for k=1:points; %go through all points behind the front to see which is
closest to the car length
    length=((x(i)-x(i-k))^2+(y(i)-y(i-k))^2)^.5;
    diff=base-length;

    if diff<0 %break when get further than car length
        break
    end

    if diff<min
        min=diff;
        rearx(count)=x(i-k);
        reary(count)=y(i-k);
        sloper(count)=(y(i-k+1)-y(i-1-k))/(2*dx);
    end
end

count=count+1;

end

% Rear starting x and y point
rearx(1)=x(1);
reary(1)=y(1);

%Now find initial heading angle of car
heading(1)=(fronty(1)-reary(1))/(frontx(1)-rearx(1));
%Find first df
df(1)=atan(sloper(1))-atan(heading(1));
dr(1)=0;

for i=2:sizeindex

    dfa=abs(df(i-1));
    disp=((frontx(i)-frontx(i-1))^2+(fronty(i)-fronty(i-1))^2)^.5; %pt to pt disp
    B=asin((disp*sin(dfa))/base);
    heading(i)=tan(atan(heading(i-1))+sign(df(i-1))*abs(B));
    df(i)=atan(sloper(i))-atan(heading(i));
    dr(i)=0;

end

% %%% Calculate Wheel Angles
for n=1:sizeindex;

    % %Wheel 1
    w1xpos(n)=frontx(n)-(track/2)*sin(pi-atan(heading(n)));
    w1ypos(n)=fronty(n)-(track/2)*cos(pi-atan(heading(n)));
    d1(n)=acot(cot(df(n))-(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));
    w1rot(n)=atan(heading(n))+d1(n);%d1(n);
    %

```



```

% %Wheel 2
w2xpos(n)=frontx(n)+(track/2)*sin(pi-atan(heading(n)));
w2ypos(n)=fronty(n)+(track/2)*cos(pi-atan(heading(n)));
d2(n)=acot(cot(df(n))+(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));

w2rot(n)=atan(heading(n))+d2(n);%d2(n);
%
% %Wheel 3
w3xpos(n)=frontx(n)-cardiag*cos(atan(heading(n))+carang);
w3ypos(n)=fronty(n)-cardiag*sin(atan(heading(n))+carang);
d3(n)=acot(cot(dr(n))+(track/(2*base))*cot(dr(n))*(tan(df(n))-tan(dr(n))));
w3rot(n)=atan(heading(n))+d3(n);%d3(n);
%
% %Wheel 4
w4xpos(n)=frontx(n)-cardiag*cos(atan(heading(n))-carang);
w4ypos(n)=fronty(n)-cardiag*sin(atan(heading(n))-carang);
d4(n)=acot(cot(dr(n))-(track/(2*base))*cot(dr(n))*(tan(df(n))-tan(dr(n))));
w4rot(n)=atan(heading(n))+d4(n);%d4(n);
% Center of Gravity
CGx(n)=frontx(n)-base/2*cos(atan(heading(n)));
CGy(n)=fronty(n)-base/2*sin(atan(heading(n)));
end

Error=0;
for k=1:sizeindex;
    %Find ideal xy coord
    w1xi(k)=frontx(k)-(track/2)*sin(atan(slopef(k)));
    w1yi(k)=fronty(k)+(track/2)*cos(atan(slopef(k)));
    w2xi(k)=frontx(k)+(track/2)*sin(atan(slopef(k)));
    w2yi(k)=fronty(k)-(track/2)*cos(atan(slopef(k)));
    w4xi(k)=rearx(k)-(track/2)*sin(atan(sloper(k)));
    w4yi(k)=reary(k)+(track/2)*cos(atan(sloper(k)));
    w3xi(k)=rearx(k)+(track/2)*sin(atan(sloper(k)));
    w3yi(k)=reary(k)-(track/2)*cos(atan(sloper(k)));
    w1e(k)=((w1xpos(k)-w1xi(k))^2+(w1ypos(k)-w1yi(k))^2)^.5;
    w2e(k)=((w2xpos(k)-w2xi(k))^2+(w2ypos(k)-w2yi(k))^2)^.5;
    w3e(k)=((w3xpos(k)-w3xi(k))^2+(w3ypos(k)-w3yi(k))^2)^.5;
    w4e(k)=((w4xpos(k)-w4xi(k))^2+(w4ypos(k)-w4yi(k))^2)^.5;
    Error=Error+w1e(k)+w2e(k)+w3e(k)+w4e(k);
end
Error=Error/(points*4)

% plot(rearx,reary,'b','linewidth',1.5)
plot(frontx,fronty,'k','linewidth',1.5)
plot(w1xpos,w1ypos,'g','linewidth',1.5)
plot(w2xpos,w2ypos,'g','linewidth',1.5)
plot(w3xpos,w3ypos,'g','linewidth',1.5)
plot(w4xpos,w4ypos,'g','linewidth',1.5)
plot(w1xi,w1yi,'b','linewidth',1.5)
plot(w2xi,w2yi,'b','linewidth',1.5)
plot(w3xi,w3yi,'b','linewidth',1.5)
plot(w4xi,w4yi,'b','linewidth',1.5)
plot(x,y,'k','linewidth',1.5)
plot(CGx,CGy,'r-','linewidth',1.5)

```

```

legend('Tracking Path','CG Path','Location','NorthWest')

%
% %%% Calculate Wheel Speeds
%
% w1speed = velocity*W1R./CGR;
% w2speed = velocity*W2R./CGR;
% w3speed = velocity*W3R./CGR;
% w4speed = velocity*W4R./CGR;

% %%% Second Figure to show Wheel Information
%
% fig2 = figure(2); % get the handle to a new figure window
% set(fig2,'Position',[75,75,625,625],...
%     'NumberTitle','off','Name',...
%     'Wheel Information')
% subplot(2,1,1)
% plot(rearx,phi1,'b')
% axis([0,pathend/2,-90,90]);
% hold on
% title('Wheel Angles vs. Postion')
% xlabel('Position')
% ylabel('Wheel Angle with Respect to Vehicle (Degrees)')
% plot(rearx,phi2,'m')
% plot(rearx,phi3,'r')
% plot(rearx,phi4,'g')
% hold off
%
% subplot(2,1,2)
% plot(rearx,w1speed,'b')
% axis([0,pathend/2,0,2]);
% hold on
% title('Wheel Speed vs. Postion')
% xlabel('Position')
% ylabel('Wheel Speed')
% plot(rearx,w2speed,'m')
% plot(rearx,w3speed,'r')
% plot(rearx,w4speed,'g')
% hold off
%
figure(2)
plot(w1e)
hold on
plot(w2e)
plot(w3e)
hold on
plot(w4e)
%
% =====
% Animation
% =====
elseif strcmp(option,'animate');
    %sizeindex = number of frames in animation

```

```

for index = 1:sizeindex;
    % if index==150
    % pause;
    % end
    %Car Body
    car(translate(frontx(index),fronty(index))*rotate(atan(heading(index))));
% draw the car in new posn

    %%Wheel 1
    wheel1(translate(w1xpos(index),w1ypos(index))*rotate(w1rot(index)));
    %
    %%Wheel 2
    wheel2(translate(w2xpos(index),w2ypos(index))*rotate(w2rot(index)));
    %
    %%Wheel 3
    wheel3(translate(w3xpos(index),w3ypos(index))*rotate(w4rot(index)));
    %%
    %%Wheel 4
    wheel4(translate(w4xpos(index),w4ypos(index))*rotate(w4rot(index)));

    drawnow; % flush the buffer
    %pause(0.01); % slow it down a little
end

hold off
end; %animate program

```

### A.3. AWS STEERING CONFIGURATION, FRONT WHEEL TRACKING

#### A.3.1. ANIMATE.M

```

function animate(option)

%=====
% Author: Jon Nistler
% Date: September 25, 2009
% For: HW3

%=====
global carHandle slopef sloper x y
global x2f y2f x1r y1r hnow drnow
global line1Handle
global line2Handle
global line3Handle
global line4Handle
global wheel1Handle
global wheel2Handle
global wheel3Handle
global wheel4Handle
global w1xpos w1ypos w1rot phi1 w1speed
global w2xpos w2ypos w2rot phi2 w2speed
global w3xpos w3ypos w3rot phi3 w3speed
global w4xpos w4ypos w4rot phi4 w4speed

```

```

%=====

global base track sizeindex heading
global rearx reary frontx fronty ICx ICy
base=1; %wheelbase of vehicle
track=base*.75;
velocity=1;

cardiag=(((track/2)^2+base^2)^.5);
carang=atan((track/2)/base);

points=5000; %number of points to use
pathend=15;

%%initialize on first call
if nargin<1
option = 'initialize';
end;

%=====
% initialize
%=====

if strcmp(option,'initialize')
tic
%initialize the program
clf; % clears the figure window
hold on; hold off; % clean up any previous holds
% create the figure window
fig1 = figure(1); % get the handle to a new figure window
set(fig1,'Position',[50,50,600,600],...
'NumberTitle','off','Name',...
'AWS Robot Animation');
set(gca,'drawmode','fast');
set(fig1,'Backingstore','off');

%=====
% Menus
%=====
set(gcf,'MenuBar','none');

%File menu
file_menu = uimenu(fig1,'label','File');
print_option = uimenu(file_menu,'Label','Print',...
'Callback','printdlg');
export_option = uimenu(file_menu,'Label','Export',...
'Callback','print -djpeg');
exit_option = uimenu(file_menu,'Label','Exit','Callback','exit');

>Action menu
action_menu = uimenu(fig1,'Label','Actions');
animate_option = uimenu(action_menu,'Label','Animate',...

```

```

'Callback','animate(''animate'')');

%Initialize coordinate system
plot(0,0,'k','linewidth',1.5)
hold on
plot(0,0,'r-.','linewidth',1.5)
plot(0,0,'c.')
legend('Tracking Path','CG Path','Instant Center Path','Location','NorthWest')
axis ([0 7.5 0 7.5]);
axis square
axis('on');
grid('on');
title('Animation Showing Instant Center for Steering Control');
xlabel('X Coordinate');
ylabel('Y Coordinate');

%Initialize 'gate' graphics
carHandle=0;
car(translate(0,0));

wheel1Handle=0;
wheel1(translate(0,0));

wheel2Handle=0;
wheel2(translate(0,0));

wheel3Handle=0;
wheel3(translate(0,0));

wheel4Handle=0;
wheel4(translate(0,0));

line1Handle=0;
line1(0,0,0,0);

line2Handle=0;
line2(0,0,0,0);

line3Handle=0;
line3(0,0,0,0);

line4Handle=0;
line4(0,0,0,0);

% =====
% Calculation of Coordinates
% =====
x=linspace(0,pathend,points+1); %curve to follow
y=pathdefinition2(x);
x=.5.*x;
y=.5.*y;
dx=x(2)-x(1);
%first find point that is closest to car distance away
for i=2:1:points;

```

```

min=inf;
length=((x(i)-x(1))^2+(y(i)-y(1))^2)^.5;
diff=base-length;

if diff<min
    min=diff;
    start=i;
    sizeindex=points-start;
end

if diff<0 %break when get further than car length
    break
end

end

%Establish front XY coordinates
count=1;
for i=start:points;

    frontx(count)=x(i);
    fronty(count)=y(i);
    slopef(count)=(y(i+1)-y(i-1))/(2*dx);

    min=inf;%reinitialize min for minimum search
    for k=1:points; %go through all points behind the front to see which is
closest to the car length
        length=((x(i)-x(i-k))^2+(y(i)-y(i-k))^2)^.5;
        diff=base-length;

        if diff<0 %break when get further than car length
            break
        end

        if diff<min
            min=diff;
            rearmx(count)=x(i-k);
            reary(count)=y(i-k);
            sloper(count)=(y(i-k+1)-y(i-k-1))/(2*dx);
        end
    end
    count=count+1;

end

% Rear starting x and y point
rearmx(1)=x(1);
reary(1)=y(1);

%Now find initial heading angle of car
heading(1)=(fronty(1)-reary(1))/(frontx(1)-rearmx(1));

%Find first df
df(1)=atan(slopef(1))-atan(heading(1)); %% Is this backward?

```

```

%Find first dr
sloper(1)=(y(2)-y(1))/(dx);

dr(1)=-1*df(1);

for i=2:sizeindex

    dfa=abs(df(i-1));
    disp=((frontx(i)-frontx(i-1))^2+(fronty(i)-fronty(i-1))^2)^.5; %pt to pt
disp
    l2=((disp^2+base^2-2*disp*base*cos(pi-dfa))^0.5);
    D=abs(asin((disp*sin(dfa))/l2));
    E=pi-abs(asin((l2*sin(dfa+D))/base));
    B=real(pi-E-dfa);
    heading(i)=atan(atan(heading(i-1))+sign(df(i-1))*abs(B));
    df(i)=atan(sloper(i))-atan(heading(i));
    dr(i)=-1*df(i);

end

% %%% Calculate Wheel Angles
for n=1:sizeindex;

% %Wheel 1
w1xpos(n)=frontx(n)-(track/2)*sin(pi-atan(heading(n)));
w1ypos(n)=fronty(n)-(track/2)*cos(pi-atan(heading(n)));
d1(n)=acot(cot(df(n))-(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));
w1rot(n)=atan(heading(n))+d1(n);%d1(n);
%
% %Wheel 2
w2xpos(n)=frontx(n)+(track/2)*sin(pi-atan(heading(n)));
w2ypos(n)=fronty(n)+(track/2)*cos(pi-atan(heading(n)));
d2(n)=acot(cot(df(n))+(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));

w2rot(n)=atan(heading(n))+d2(n);%d2(n);
%
% %Wheel 3
w3xpos(n)=frontx(n)-cardiag*cos(atan(heading(n))+carang);
w3ypos(n)=fronty(n)-cardiag*sin(atan(heading(n))+carang);
d3(n)=acot(cot(dr(n))+(track/(2*base))*cot(dr(n))*(tan(df(n))-tan(dr(n))));
w3rot(n)=atan(heading(n))+d3(n);%d3(n);
%
% %Wheel 4
w4xpos(n)=frontx(n)-cardiag*cos(atan(heading(n))-carang);
w4ypos(n)=fronty(n)-cardiag*sin(atan(heading(n))-carang);
d4(n)=acot(cot(dr(n))-(track/(2*base))*cot(dr(n))*(tan(df(n))-tan(dr(n))));
w4rot(n)=atan(heading(n))+d4(n);%d4(n);
% Center of Gravity
CGx(n)=frontx(n)-base/2*cos(atan(heading(n)));
CGy(n)=fronty(n)-base/2*sin(atan(heading(n)));
end
Error=0;
for k=1:sizeindex;
%Find ideal xy coord

```

```

        w1xi(k)=frontx(k)-(track/2)*sin(atan(slopef(k)));
        w1yi(k)=fronty(k)+(track/2)*cos(atan(slopef(k)));
        w2xi(k)=frontx(k)+(track/2)*sin(atan(slopef(k)));
        w2yi(k)=fronty(k)-(track/2)*cos(atan(slopef(k)));
        w4xi(k)=rearx(k)-(track/2)*sin(atan(sloper(k)));
        w4yi(k)=reary(k)+(track/2)*cos(atan(sloper(k)));
        w3xi(k)=rearx(k)+(track/2)*sin(atan(sloper(k)));
        w3yi(k)=reary(k)-(track/2)*cos(atan(sloper(k)));
w1e(k)=((w1xpos(k)-w1xi(k))^2+(w1ypos(k)-w1yi(k))^2)^.5;
w2e(k)=((w2xpos(k)-w2xi(k))^2+(w2ypos(k)-w2yi(k))^2)^.5;
w3e(k)=((w3xpos(k)-w3xi(k))^2+(w3ypos(k)-w3yi(k))^2)^.5;
w4e(k)=((w4xpos(k)-w4xi(k))^2+(w4ypos(k)-w4yi(k))^2)^.5;
Error=Error+w1e(k)+w2e(k)+w3e(k)+w4e(k);
end
Error=Error/(points*4)

% plot(rearx,reary,'b','linewidth',1.5)
plot(frontx,fronty,'g','linewidth',1.5)
plot(w1xpos,w1ypos,'g','linewidth',1.5)
plot(w2xpos,w2ypos,'g','linewidth',1.5)
plot(w3xpos,w3ypos,'g','linewidth',1.5)
plot(w4xpos,w4ypos,'g','linewidth',1.5)
plot(w1xi,w1yi,'b','linewidth',1.5)
plot(w2xi,w2yi,'b','linewidth',1.5)
plot(w3xi,w3yi,'b','linewidth',1.5)
plot(w4xi,w4yi,'b','linewidth',1.5)
plot(x,y,'k','linewidth',1.5)
plot(CGx,CGy,'r-.','linewidth',1.5)
legend('Tracking Path','CG Path','Location','NorthWest')

%
% %%% Calculate Wheel Speeds
%
% w1speed = velocity*W1R./CGR;
% w2speed = velocity*W2R./CGR;
% w3speed = velocity*W3R./CGR;
% w4speed = velocity*W4R./CGR;

% %%% Second Figure to show Wheel Information
%
% fig2 = figure(2); % get the handle to a new figure window
% set(fig2,'Position',[75,75,625,625],...
%     'NumberTitle','off','Name',...
%     'Wheel Information')
% subplot(2,1,1)
% plot(rearx,phi1,'b')
% axis([0,pathend/2,-90,90]);
% hold on
% title('Wheel Angles vs. Postion')
% xlabel('Position')
% ylabel('Wheel Angle with Respect to Vehicle (Degrees)')
% plot(rearx,phi2,'m')
% plot(rearx,phi3,'r')
% plot(rearx,phi4,'g')

```



```

% hold off
%
% subplot(2,1,2)
% plot(rearx,w1speed,'b')
% axis([0,pathend/2,0,2]);
% hold on
% title('Wheel Speed vs. Postion')
% xlabel('Position')
% ylabel('Wheel Speed')
% plot(rearx,w2speed,'m')
% plot(rearx,w3speed,'r')
% plot(rearx,w4speed,'g')
% hold off
%
figure(2)
plot(w1e)
hold on
plot(w2e)
plot(w3e)
plot(w4e)
%
% =====
% Animation
% =====
elseif strcmp(option,'animate');
%sizeindex = number of frames in animation

for index = 1:sizeindex;
%   if index==150
%       pause;
%   end
%Car Body
car(translate(frontx(index),fronty(index))*rotateme(atan(heading(index)))); % draw
the car in new posn

% %Wheel 1
wheel1(translate(w1xpos(index),w1ypos(index))*rotateme(w1rot(index)));
%
% %Wheel 2
wheel2(translate(w2xpos(index),w2ypos(index))*rotateme(w2rot(index)));
%
% % %Wheel 3
wheel3(translate(w3xpos(index),w3ypos(index))*rotateme(w4rot(index)));
% %
% % %Wheel 4
wheel4(translate(w4xpos(index),w4ypos(index))*rotateme(w4rot(index)));

drawnow; % flush the buffer
%pause(0.01); % slow it down a little
end

hold off
end; %animate program

```

## A.4. AWS STEERING CONFIGURATION, CENTER OF GRAVITY TRACKING

### A.4.1. ANIMATE.M

```
unction animate(option)
%=====
% Author: Jon Nistler
% Date: September 25, 2009
% For: HW3
%=====
% Key Variables:
% gateHandle : points to the "step" graphical object
% gateLength : length of the step (arbitrary right now)
% x,y : location of the gate

%=====
global carHandle slopef sloper x y CGx CGy dfdx df2dx2
global line1Handle
global line2Handle
global line3Handle
global line4Handle
global wheel1Handle
global wheel2Handle
global wheel3Handle
global wheel4Handle
global w1xpos w1ypos w1rot phi1 w1speed
global w2xpos w2ypos w2rot phi2 w2speed
global w3xpos w3ypos w3rot phi3 w3speed
global w4xpos w4ypos w4rot phi4 w4speed
global w1e w2e w3e w4e

%=====

global base track sizeindex heading
global rearx reary frontx fronty ICx ICy
base=1; %wheelbase of vehicle
track=base*.75;
velocity=1;

cardiag=(((track/2)^2+(base/2)^2)^.5);
carang=atan((track/2)/(base/2));

points=5000; %number of points to use
pathend=15;

%%initialize on first call
if nargin<1
option = 'initialize';
end;

%=====
% initialize
%=====
```

```

if strcmp(option,'initialize')
tic
%initialize the program
clf; % clears the figure window
hold on; hold off; % clean up any previous holds
% create the figure window
fig1 = figure(1); % get the handle to a new figure window
set(fig1,'Position',[50,50,600,600],...
    'NumberTitle','off','Name',...
    'AWS Robot Animation');
set(gca,'drawmode','fast');
set(fig1,'Backingstore','off');

%=====
% Menus
%=====
set(gcf,'MenuBar','none');

%File menu
file_menu = uimenu(fig1,'label','File');
print_option = uimenu(file_menu,'Label','Print',...
    'Callback','printdlg');
export_option = uimenu(file_menu,'Label','Export',...
    'Callback','print -djpeg');
exit_option = uimenu(file_menu,'Label','Exit','Callback','exit');

%Action menu
action_menu = uimenu(fig1,'Label','Actions');
animate_option = uimenu(action_menu,'Label','Animate',...
    'Callback','animate(''animate'')');

%Initialize coordinate system
plot(0,0,'k','linewidth',1.5)
hold on
plot(0,0,'k-.','linewidth',1.5)
plot(0,0,'c.')
legend('Tracking Path','CG Path','Instant Center Path','Location','NorthWest')
axis ([0 7.5 0 7.5]);
axis square
axis('on');
grid('on');
title('Animation Showing Instant Center for Steering Control');
xlabel('X Coordinate');
ylabel('Y Coordinate');

%Initialize 'gate' graphics
carHandle=0;
car(translate(0,0));

wheel1Handle=0;
wheel1(translate(0,0));

wheel2Handle=0;

```

```

wheel2(translate(0,0));

wheel3Handle=0;
wheel3(translate(0,0));

wheel4Handle=0;
wheel4(translate(0,0));

line1Handle=0;
line1(0,0,0,0);

line2Handle=0;
line2(0,0,0,0);

line3Handle=0;
line3(0,0,0,0);

line4Handle=0;
line4(0,0,0,0);

% =====
% Calculation of Coordinates
% =====
x=linspace(0,7.5,points+1); %curve to follow
y=curvy2(x);
dx=x(2)-x(1);

%first find point that is closest to car distance away
count=0;
min=inf;%reinitialize min for minimum search
for i=1:points; %go through all points ahead of rear to see
    %which is closest to the car length
    length=((x(i)-x(1))^2+(y(i)-y(1))^2)^.5;
    diff=base/2-length;

    if diff<min
        min=diff;
        start=i;
    end

    if diff<0 %break when get further than car length
        break
    end
end
min=inf;
for i=points:-1:1; %go through all points ahead of rear to see
    %which is closest to the car length
    length=((x(i)-x(points))^2+(y(i)-y(points))^2)^.5;
    diff=base/2-length;

    if diff<min
        min=diff;
        ending=i;
    end
end

```

```

        if diff<0 %break when get further than car length
            break
        end
    end
end
sizeindex=ending-start;

%Now find slope at all center points
count=start;
for k=1:sizeindex;
    CGx(k)=x(count);
    CGy(k)=y(count);
    %use central diff approx
    dfdx(k)=(y(count+1)-y(count-1))/(2*dx);
    %dfdx(k)=(y(count-2)+8*y(count+1)-8*y(count-1)-y(count+2))/(12*dx);
    %Now find radius of curvature at all center points
    %use central diff approx
    d2fdx2(k)=(y(count-1)-2*y(count)+y(count+1))/(dx^2);
    R(k)=(1+(dfdx(k))^2)^(3/2)/(d2fdx2(k));

    min=inf;
    for i=1:points
        %find rearx and reary and boundary
        length=((x(count-i)-x(count))^2+(y(count-i)-y(count))^2)^.5;
        diff=base/2-length;

        if diff<0 %break when get further than car length
            break
        end

        if diff<min
            min=diff;
            rearx(k)=x(count-i);
            reary(k)=y(count-i);
            sloper(k)=(y(count-i+1)-y(count-i-1))/(2*dx);
        end
    end
end
min=inf;
for i=1:points
    %find frontx and fronty and boundary
    length=((x(count+i)-x(count))^2+(y(count+i)-y(count))^2)^.5;
    diff=base/2-length;

    if diff<0 %break when get further than car length
        break
    end

    if diff<min
        min=diff;
        frontx(k)=x(count+i);
        fronty(k)=y(count+i);
        slopef(k)=(y(count+i+1)-y(count+i-1))/(2*dx);
    end
end

```

```

    end

    count=count+1;
end

%Now find coordinates of ICR
for l=1:sizeindex;

    %Define Vars
    heading(l)=dfdx(l);
    %atan(-1/dfdx(l))
%    ICx(l)=CGx(l)-R(l)*cos(heading(l));
%    ICy(l)=CGy(l)-R(l)*sin(heading(l));

end

for k=1:sizeindex;
%Find ideal xy coord
    w1xi(k)=frontx(k)-(track/2)*sin(atan(slopef(k)));
    w1yi(k)=fronty(k)+(track/2)*cos(atan(slopef(k)));
    w2xi(k)=frontx(k)+(track/2)*sin(atan(slopef(k)));
    w2yi(k)=fronty(k)-(track/2)*cos(atan(slopef(k)));
    w4xi(k)=rearx(k)-(track/2)*sin(atan(sloper(k)));
    w4yi(k)=reary(k)+(track/2)*cos(atan(sloper(k)));
    w3xi(k)=rearx(k)+(track/2)*sin(atan(sloper(k)));
    w3yi(k)=reary(k)-(track/2)*cos(atan(sloper(k)));

    end

Error=0;
% %%% Calculate Wheel Angles
for n=1:sizeindex;
df(n)=atan((base/2)/R(n));
dr(n)=-1*df(n);

% %Wheel 1
w1xpos(n)=CGx(n)+cardiag*cos(atan(heading(n))+carang);
w1ypos(n)=CGy(n)+cardiag*sin(atan(heading(n))+carang);
d1(n)=acot(cot(df(n))-(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));
w1rot(n)=atan(heading(n))+d1(n);
%
% %Wheel 2
w2xpos(n)=CGx(n)+cardiag*cos(atan(heading(n))-carang);
w2ypos(n)=CGy(n)+cardiag*sin(atan(heading(n))-carang);
d2(n)=acot(cot(df(n))+(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));
w2rot(n)=atan(heading(n))+d2(n);
%
% %Wheel 3
w3xpos(n)=CGx(n)+cardiag*cos(atan(heading(n))+pi+carang);
w3ypos(n)=CGy(n)+cardiag*sin(atan(heading(n))+pi+carang);
d3(n)=acot(cot(dr(n))-(track/(2*base))*cot(dr(n))*(tan(dr(n))-tan(df(n))));
w3rot(n)=atan(heading(n))+d3(n);

```

```

%
% %Wheel 4
w4xpos(n)=CGx(n)+cardiag*cos(pi+atan(heading(n))-carang);
w4ypos(n)=CGy(n)+cardiag*sin(pi+atan(heading(n))-carang);
d4(n)=acot(cot(dr(n))+(track/(2*base))*cot(dr(n))*(tan(dr(n))-tan(df(n))));
w4rot(n)=atan(heading(n))+d4(n);

w1e(n)=((w1xpos(n)-w1xi(n))^2+(w1ypos(n)-w1yi(n))^2)^.5;
w2e(n)=((w2xpos(n)-w2xi(n))^2+(w2ypos(n)-w2yi(n))^2)^.5;
w3e(n)=((w3xpos(n)-w3xi(n))^2+(w3ypos(n)-w3yi(n))^2)^.5;
w4e(n)=((w4xpos(n)-w4xi(n))^2+(w4ypos(n)-w4yi(n))^2)^.5;
Error=Error+w1e(n)+w2e(n)+w3e(n)+w4e(n);
end
Error=Error/(points*4)

plot(x,y,'k','linewidth',1.5)
plot(CGx,CGy,'k-.','linewidth',1.5)
plot(w1xpos,w1ypos,'g','linewidth',1.5)
plot(w2xpos,w2ypos,'g','linewidth',1.5)
plot(w3xpos,w3ypos,'g','linewidth',1.5)
plot(w4xpos,w4ypos,'g','linewidth',1.5)
plot(w1xi,w1yi,'b','linewidth',1.5)
plot(w2xi,w2yi,'b','linewidth',1.5)
plot(w3xi,w3yi,'b','linewidth',1.5)
plot(w4xi,w4yi,'b','linewidth',1.5)
plot(ICx,ICy,'c.')
legend('Tracking Path','CG Path','Instant Center Path','Location','NorthWest')

%
% %%% Calculate Wheel Speeds
% % % Wheel speeds are simply a ratio of lengths to instant center
%
% W1R = ((w1xpos-ICx).^2+(w1ypos-ICy).^2).^5; %Distance to each of the wheels from
IC
% W2R = ((w2xpos-ICx).^2+(w2ypos-ICy).^2).^5;
% W3R = ((w3xpos-ICx).^2+(w3ypos-ICy).^2).^5;
% W4R = ((w4xpos-ICx).^2+(w4ypos-ICy).^2).^5;
%
% w1speed = velocity*W1R./R;
% w2speed = velocity*W2R./R;
% w3speed = velocity*W3R./R;
% w4speed = velocity*W4R./R;

% %%% Second Figure to show Wheel Information
%
% fig2 = figure(2); % get the handle to a new figure window
% set(fig2,'Position',[75,75,625,625],...
%     'NumberTitle','off','Name',...
%     'Wheel Information')
% subplot(2,1,1)
% plot(rearx,phi1,'b')
% axis([0,pathend/2,-90,90]);
% hold on

```

```

% title('Wheel Angles vs. Postion')
% xlabel('Position')
% ylabel('Wheel Angle with Respect to Vehicle (Degrees)')
% plot(rearx,phi2,'m')
% plot(rearx,phi3,'r')
% plot(rearx,phi4,'g')
% hold off
%
% subplot(2,1,2)
% plot(rearx,w1speed,'b')
% axis([0,pathend/2,0,2]);
% hold on
% title('Wheel Speed vs. Postion')
% xlabel('Position')
% ylabel('Wheel Speed')
% plot(rearx,w2speed,'m')
% plot(rearx,w3speed,'r')
% plot(rearx,w4speed,'g')
% hold off

figure(2)
plot(w1e)
hold on
plot(w2e)
plot(w3e)
plot(w4e)

% =====
% Animation
% =====
elseif strcmp(option,'animate');
%sizeindex = number of frames in animation

for index = 1:sizeindex;

    %Car Body
    car(translate(CGx(index),CGy(index))*rotateme((pi/2)+atan(heading(index)))); % draw
the car in new posn
%line1(CGx(index),CGy(index),ICx(index),ICy(index));

% %Wheel 1
wheel1(translate(w1xpos(index),w1ypos(index))*rotateme(w1rot(index)));
% % %line1(w1xpos(index),w1ypos(index),ICx(index),ICy(index));
% %
% % %Wheel 2
wheel2(translate(w2xpos(index),w2ypos(index))*rotateme(w2rot(index)));
% % % %line2(w2xpos(index),w2ypos(index),ICx(index),ICy(index));
% % %
% % % %Wheel 3
wheel3(translate(w3xpos(index),w3ypos(index))*rotateme(w3rot(index)));
% % % %line3(w3xpos(index),w3ypos(index),ICx(index),ICy(index));
% % %
% % % %Wheel 4
wheel4(translate(w4xpos(index),w4ypos(index))*rotateme(w4rot(index)));

```



```

% %line4(w4xpos(index),w4ypos(index),ICx(index),ICy(index));

drawnow; % flush the buffer
pause(0.01); % slow it down a little
end

hold off
end; %animate program

```

## A.5. AWS STEERING CONFIGURATION, FRONT AND REAR WHEEL TRACKING

### A.5.1. ANIMATE.M

```

function animate(option)
%=====
% Author: Jon Nistler
% Date: September 25, 2009
% For: HW3
%=====
% Key Variables:
% gateHandle : points to the "step" graphical object
% gateLength : length of the step (arbitrary right now)
% x,y : location of the gate

%=====
global carHandle slopef sloper x y
global line1Handle
global line2Handle
global line3Handle
global line4Handle
global wheel1Handle
global wheel2Handle
global wheel3Handle
global wheel4Handle
global w1xpos w1ypos w1rot phi1 w1speed
global w2xpos w2ypos w2rot phi2 w2speed
global w3xpos w3ypos w3rot phi3 w3speed
global w4xpos w4ypos w4rot phi4 w4speed

%=====

global base track sizeindex heading
global rearx reary frontx fronty ICx ICy
base=1; %wheelbase of vehicle
track=base*.75;
velocity=1;

cardiag=((track/2)^2+base^2)^.5;
carang=atan((track/2)/base);

points=1001; %number of points to use
pathend=7.5;

```

```

%%initialize on first call
if nargin<1
option = 'initialize';
end;

%=====
% initialize
%=====

if strcmp(option,'initialize')
tic
%initialize the program
clf; % clears the figure window
hold on; hold off; % clean up any previous holds
% create the figure window
fig1 = figure(1); % get the handle to a new figure window
set(fig1,'Position',[50,50,600,600],...
'NumberTitle','off','Name',...
'AWS Robot Animation');
set(gca,'drawmode','fast');
set(fig1,'Backingstore','off');

%=====
% Menus
%=====
set(gcf,'MenuBar','none');

%File menu
file_menu = uimenu(fig1,'label','File');
print_option = uimenu(file_menu,'Label','Print',...
'Callback','printdlg');
export_option = uimenu(file_menu,'Label','Export',...
'Callback','print -djpeg');
exit_option = uimenu(file_menu,'Label','Exit','Callback','exit');

>Action menu
action_menu = uimenu(fig1,'Label','Actions');
animate_option = uimenu(action_menu,'Label','Animate',...
'Callback','animate(''animate'')');

%Initialize coordinate system
plot(0,0,'k','linewidth',1.5)
hold on
plot(0,0,'k-.','linewidth',1.5)
plot(0,0,'c.')
legend('Tracking Path','CG Path','Instant Center Path','Location','NorthWest')
axis ([0 7.5 0 7.5]);
axis square
axis('on');
grid('on');
title('Animation Showing Instant Center for Steering Control');
xlabel('X Coordinate');

```

```

ylabel('Y Coordinate');

%Initialize 'gate' graphics
carHandle=0;
car(translate(0,0));

wheel1Handle=0;
wheel1(translate(0,0));

wheel2Handle=0;
wheel2(translate(0,0));

wheel3Handle=0;
wheel3(translate(0,0));

wheel4Handle=0;
wheel4(translate(0,0));

line1Handle=0;
line1(0,0,0,0);

line2Handle=0;
line2(0,0,0,0);

line3Handle=0;
line3(0,0,0,0);

line4Handle=0;
line4(0,0,0,0);

% =====
% Calculation of Coordinates
% =====
x=linspace(0,7.5,points+1); %curve to follow
y=zigzag(x);

dx=x(2)-x(1);
%See where to end path
min=inf;
    for j=(points-1):-1:1; %go through all points ahead of rear to see
        %which is closest to the car length
        length=((x(points)-x(j))^2+(y(points)-y(j))^2)^.5;
        diff=base-length;

        if diff<min
            min=diff;
            ending=j;
            sizeindex=ending;
        end
        if diff<0 %break when get further than car length
            break
        end
    end

end

```

```

%first find point that is closest to car distance away
for i=1:ending;

    rearx(i)=x(i); %Store rear position values
    reary(i)=y(i);
    indexr(i)=i;

    min=inf;%reinitialize min for minimum search

    for j=i+1:points; %go through all points ahead of rear to see
        %which is closest to the car length
        length=((x(j)-x(i))^2+(y(j)-y(i))^2)^.5;
        diff=base-length;

        if diff<min
            min=diff;
            frontx(i)=x(j);
            fronty(i)=y(j);
            slopef(i)=(y(j+1)-y(j-1))/(2*dx);
        end

        if diff<0 %break when get further than car length
            break
        end

    end
end

%Now find slope at all rear points
for k=1:sizeindex;
    if k==1 %use forward difference
        sloper(k)=(y(k+1)-y(k))/(dx);
    else %use central diff approx
        sloper(k)=(y(k+1)-y(k-1))/(2*dx);
    end
end

%Now find intersection of normals
for l=1:sizeindex;
    %transform variables so they are easy to use
    %eq's for lines are (ICy-B)=S(ICx-A) & (ICy-D)=T(ICx-C)

    %Define Vars
    A=rearx(l);
    B=reary(l);
    C=frontx(l);
    D=fronty(l);
    S=-1/sloper(l);%Vector normal to slopes
    T=-1/slopef(l);

    ICx(l)=(T*C-S*A+B-D)/(T-S);
    ICy(l)=S*(ICx(l)-A)+B;
end

```

```

    %line([ICx(1) frontx(1)],[ICy(1) fronty(1)])
    %line([ICx(1) rearx(1)],[ICy(1) reary(1)])
end

%Now find heading angle of car and path of CG
for m=1:sizeindex;
    heading(m)=atan2((fronty(m)-reary(m)),(frontx(m)-rearx(m)));
    CGx(m)=rearx(m)+base/2*cos(heading(m));
    CGy(m)=reary(m)+base/2*sin(heading(m));
end

%%%% Calculate Wheel Angles
for n=1:sizeindex;
    df(n)=atan(slopec(n))-atan(heading(n));
    dr(n)=atan(sloper(n))-atan(heading(n));

%Wheel 1
    w1xpos(n)=rearx(n)+cardiag*cos(heading(n)+carang);
    w1ypos(n)=reary(n)+cardiag*sin(heading(n)+carang);
    d1(n)=acot(cot(df(n))-(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));
    w1rot(n)=atan(heading(n))+d1(n);%d1(n);

%Wheel 2
    w2xpos(n)=rearx(n)+cardiag*cos(heading(n)-carang);
    w2ypos(n)=reary(n)+cardiag*sin(heading(n)-carang);
    d2(n)=acot(cot(df(n))+(track/(2*base))*cot(df(n))*(tan(df(n))-tan(dr(n))));
    w2rot(n)=atan(heading(n))+d2(n);%d2(n);

%Wheel 3
    w3xpos(n)=rearx(n)+(track/2)*sin(pi-heading(n));
    w3ypos(n)=reary(n)+(track/2)*cos(pi-heading(n));
    d3(n)=acot(cot(dr(n))+(track/(2*base))*cot(dr(n))*(tan(df(n))-tan(dr(n))));
    w3rot(n)=atan(heading(n))+d3(n);%d3(n);

%Wheel 4
    w4xpos(n)=rearx(n)-(track/2)*sin(pi-heading(n));
    w4ypos(n)=reary(n)-(track/2)*cos(pi-heading(n));
    d4(n)=acot(cot(dr(n))-(track/(2*base))*cot(dr(n))*(tan(df(n))-tan(dr(n))));
    w4rot(n)=atan(heading(n))+d4(n);%d4(n);

end

Error=0;
for k=1:sizeindex;
    %Find ideal xy coord
    w1xi(k)=frontx(k)-(track/2)*sin(atan(slopec(k)));
    w1yi(k)=fronty(k)+(track/2)*cos(atan(slopec(k)));
    w2xi(k)=frontx(k)+(track/2)*sin(atan(slopec(k)));
    w2yi(k)=fronty(k)-(track/2)*cos(atan(slopec(k)));
    w4xi(k)=rearx(k)-(track/2)*sin(atan(sloper(k)));
    w4yi(k)=reary(k)+(track/2)*cos(atan(sloper(k)));
    w3xi(k)=rearx(k)+(track/2)*sin(atan(sloper(k)));
    w3yi(k)=reary(k)-(track/2)*cos(atan(sloper(k)));
    w1e(k)=((w1xpos(k)-w1xi(k))^2+(w1ypos(k)-w1yi(k))^2)^.5;

```

```

w2e(k)=((w2xpos(k)-w2xi(k))^2+(w2ypos(k)-w2yi(k))^2)^.5;
w3e(k)=((w3xpos(k)-w3xi(k))^2+(w3ypos(k)-w3yi(k))^2)^.5;
w4e(k)=((w4xpos(k)-w4xi(k))^2+(w4ypos(k)-w4yi(k))^2)^.5;
Error=Error+w1e(k)+w2e(k)+w3e(k)+w4e(k);
end
Error=Error/(points*4)

% plot(rearx,reary,'k','linewidth',1.5)
% plot(frontx,fronty,'k','linewidth',1.5)
plot(x,y,'k','linewidth',1.5)
plot(CGx,CGy,'k-.','linewidth',1.5)
plot(ICx,ICy,'c.')
plot(w1xpos,w1ypos,'g','linewidth',1.5)
plot(w2xpos,w2ypos,'g','linewidth',1.5)
plot(w3xpos,w3ypos,'g','linewidth',1.5)
plot(w4xpos,w4ypos,'g','linewidth',1.5)
plot(w1xi,w1yi,'b','linewidth',1.5)
plot(w2xi,w2yi,'b','linewidth',1.5)
plot(w3xi,w3yi,'b','linewidth',1.5)
plot(w4xi,w4yi,'b','linewidth',1.5)
legend('Tracking Path','CG Path','Instant Center Path','Location','NorthWest')

%%% Calculate Wheel Speeds
% Wheel speeds are simply a ratio of lengths to instant center
% CGR = (((frontx+rearx)/2)-ICx).^2 + (((fronty+reary)/2)-ICy).^2).^5; %Distance
from CG to IC
%
% W1R = ((w1xpos-ICx).^2+(w1ypos-ICy).^2).^5; %Distance to each of the wheels from
IC
% W2R = ((w2xpos-ICx).^2+(w2ypos-ICy).^2).^5;
% W3R = ((w3xpos-ICx).^2+(w3ypos-ICy).^2).^5;
% W4R = ((w4xpos-ICx).^2+(w4ypos-ICy).^2).^5;
%
% w1speed = velocity*W1R./CGR;
% w2speed = velocity*W2R./CGR;
% w3speed = velocity*W3R./CGR;
% w4speed = velocity*W4R./CGR;

% %%% Second Figure to show Wheel Information
%
% fig2 = figure(2); % get the handle to a new figure window
% set(fig2,'Position',[75,75,625,625],...
%     'NumberTitle','off','Name',...
%     'Wheel Information')
% subplot(2,1,1)
% plot(rearx,phi1,'b')
% axis([0,pathend/2,-90,90]);
% hold on
% title('Wheel Angles vs. Postion')
% xlabel('Position')
% ylabel('Wheel Angle with Respect to Vehicle (Degrees)')
% plot(rearx,phi2,'m')
% plot(rearx,phi3,'r')
% plot(rearx,phi4,'g')

```

```

% hold off
%
% subplot(2,1,2)
% plot(rearx,w1speed,'b')
% axis([0,pathend/2,0,2]);
% hold on
% title('Wheel Speed vs. Postion')
% xlabel('Position')
% ylabel('Wheel Speed')
% plot(rearx,w2speed,'m')
% plot(rearx,w3speed,'r')
% plot(rearx,w4speed,'g')
% hold off

    figure(2)
    plot(w1e)
    hold on
    plot(w2e)
    plot(w3e)
    hold on
    plot(w4e)

% =====
% Animation
% =====
elseif strcmp(option,'animate');
%sizeindex = number of frames in animation
for index=1:sizeindex
    %line([ICx(1) frontx(1)],[ICy(1) fronty(1)])
    %line([ICx(1) rearx(1)],[ICy(1) reary(1)])

    %Car Body
    car(translate(rearx(index),reary(index))*rotateme(heading(index))); % draw the car
    in new posn

%Wheel 1
wheel1(translate(w1xpos(index),w1ypos(index))*rotateme(w1rot(index)));
%line1(w1xpos(index),w1ypos(index),ICx(index),ICy(index));

%Wheel 2
wheel2(translate(w2xpos(index),w2ypos(index))*rotateme(w2rot(index)));
%line2(w2xpos(index),w2ypos(index),ICx(index),ICy(index));

%Wheel 3
wheel3(translate(w3xpos(index),w3ypos(index))*rotateme(w3rot(index)));
%line3(w3xpos(index),w3ypos(index),ICx(index),ICy(index));

%Wheel 4
wheel4(translate(w4xpos(index),w4ypos(index))*rotateme(w4rot(index)));
%line4(w4xpos(index),w4ypos(index),ICx(index),ICy(index));

drawnow; % flush the buffer
%pause(0.01); % slow it down a little

```

```
end
```

```
hold off
```

```
end; %animate program
```

## A.6. CODE COMMON TO ALL SIMULATIONS

### A.6.1. CAR.M

```
% This program is the primitive to draw the car
% it takes in an instance transformation matrix
%=====
% Author: Jon Nistler
% Date: October 6, 2009
% For: Project 1
%=====
% Key Variables:
% cylinderHandle : points to the "cylinder" graphical object
% cylinderPoints : coordinates to draw the arm
% itf : instance transformation matrix passed from 'animate'
%=====

function car(itf)
global carHandle carPoints initcar base track

initcar=translateme(-2,-2);

if (carHandle==0) %put the figure at the location given
    carPoints= [track/2      track/2      -track/2      -track/2;
                -base/2      base/2      base/2      -base/2;
                1 1 1 1 ];
    newPoints=initcar*carPoints;
    carHandle=patch(newPoints(1,:),newPoints(2:,:), 'w');

else %modify the figure by redrawing it at the location given
    newPoints=itf*carPoints;
    set(carHandle,'XData',newPoints(1,:), 'YData',newPoints(2:,:));
end
```

### A.6.2. CURVY.M

```
function [y] = curvy(x)
%determines definition of path that vehicle will take
y1 = [
0
6-1
6.62-1
6-1
1.5
0.88
1.5
6
6.62
6
```



```
1.5+1
0.88+1
1.5+1
7.5
]
```

```
x1 =[
0
1.26
1.88
2.5
2.51+.25;
3.13+.25;
3.75+.25;
3.76+0.5;
4.38+.5;
5+0.5;
5.01+.75;
5.63+.75;
6.25+.75;
7.5;
]
```

```
x1 =[
0
1.26- .25;
1.88- .25;
2.5- .25;
2.51;
3.13;
3.75;
3.76+0.25;
4.38+.25;
5+0.25;
5.01+.5;
5.63+.5;
6.25+.5;
7.5;
]
```

```
cf=fit(x1,y1,'pchipinterp');
```

```
[m,n]=size(x);
for i=1:n
y(i)=cf(x(i));
end

end
```

### A.6.3. LINE(1,2,3,4).M

```
% This program is the primitive to draw the line
```

```

% it takes in an
% instance transformation matrix
%=====
% Author: Jon Nistler
% Date: October 6, 2009
% For: Project 1
%=====
% Key Variables:
% cylinderHandle : points to the "cylinder" graphical object
% cylinderPoints : coordinates to draw the arm
% itf : instance transformation matrix passed from 'animate'
%=====

function line1(x1,y1,x2,y2)
global line1Handle

if (line1Handle==0) %put the figure at the location given
    line1Handle=line([0 0],[0 1],'color','k');

else %modify the figure by redrawing it at the location given
set(line1Handle,'XData',[x1 x2],'YData',[y1 y2]);
end

```

#### A.6.4. ORIGINAL.M

```

function [y] = original(x)
%determines definition of path that vehicle will take
x1=[0.19
0.565
1.002
1.73
2.395
3.319
4.111
4.861
5.514
6.018
6.284
6.403
6.432
6.6
6.974
7.579
8.568
9.694
10.451
10.761
11.215
11.878
12.98
14.383
15.789
16.625];

```

```

y1=[0.583
1.692
2.728
3.886
4.52
4.973
5.108
5.097
5.087
5.176
5.375
5.839
6.34
6.765
6.972
7.091
7.227
7.508
8.151
8.917
10.111
10.873
11.48
11.806
11.836
11.722];
x1=x1./2;
y1=y1./2;

cf=fit(x1,y1,'smoothingspline');

[m,n]=size(x);
for i=1:n
y(i)=cf(x(i));
end

end

```

#### A.6.5. ROTATEME.M

```

% This program is the the basic matrix
% transformation to rotate the figures by 'degrees'
% A positive rotation is counterclockwise
%=====
% Author: Jon Nistler
% Date: October 6, 2009
% For: Project 1
%=====
% Key Variables:
% degrees : angle of rotation in degrees
% rads : angle of rotation in radians
% itf : instance transformation matrix passed to 'animate'
%=====
function itf=rotateme(radians);

```

```

rads=radians;%-1*degrees/180*pi;
itf = [cos(rads) -sin(rads) 0;
       sin(rads) cos(rads) 0;
       0 0 1];

```

#### A.6.6. TRANSLATE.M

```

% This program is the the basic matrix
% transformation to translate the figures
%=====
% Author: Jon Nistler
% Date: October 6, 2009
% For: Project 1
%=====
% Key Variables:
% deltax : change in x coordinate
% deltax : change in y coordinate
% itf : instance transformation matrix passed to 'animate'
%=====

```

```

function itf = translate(deltax,deltay);

```

```

itf=[1 0 deltax; 0 1 deltax; 0 0 1];

```

#### A.6.7. UTURN.M

```

function [y] = uturn(x)
%determines definition of path that vehicle will take

```

```

y1 =[
    0
    1
    4.63
    6.5
    4.63
    1
    0

```

```

]

```

```

x1 =[
    0
    1
    1.88
    3.75
    5.62
    6.5
    7.5
]

```

```

cf=fit(x1,y1,'pchipinterp');

```

```

[m,n]=size(x);
for i=1:n
y(i)=cf(x(i));
end

```

end

#### A.6.8. WHEEL(1,2,3,4).M

```
% Author: Jon Nistler
% Date: October 6, 2009
% For: Project 1
%=====
% Key Variables:
% cylinderHandle : points to the "cylinder" graphical object
% cylinderPoints : coordinates to draw the arm
% itf : instance transformation matrix passed from 'animate'
%=====

function wheel1(itf)
global wheel1Handle wheel1Points base
wheelwidth=base/8;
wheelheight=wheelwidth*2;
%global lengthL lengthA alphas alphar

%initial alpha in deployed position
%alphar=atan((lengthA*sin(radians(175)))/(lengthL+lengthA*cos(radians(175))));
%initial alpha in retracted position
%alphad=atan((lengthA*sin(radians(30)))/(lengthL+lengthA*cos(radians(30))));
%initcylinderd=translateme(61,34+lengthL)*rotateme(degrees(alphas));%initial position
deployed
%initcylinderr=translateme(61,34+lengthL)*rotateme(degrees(alphar));%initial position
retracted

if (wheel1Handle==0) %put the figure at the location given
    wheel1Points=    [-wheelheight/2 wheelheight/2 wheelheight/2 -wheelheight/2;
                    wheelwidth/2 wheelwidth/2 -wheelwidth/2 -wheelwidth/2;
                    1 1 1 1];

%    wheel1Points=    [-wheelheight/2 wheelheight/2 wheelheight/2 3*wheelheight/4
%    3*wheelheight/4 wheelheight ...
%    3*wheelheight/4 3*wheelheight/4 wheelheight/2 wheelheight/2 -wheelheight/2;
%    wheelwidth/2 wheelwidth/2 wheelwidth/4 wheelwidth/4
%    wheelwidth/2 0 -wheelwidth/2 -wheelwidth/4 ...
%    -wheelwidth/4 -wheelwidth/2 -wheelwidth/2;
%    1 1 1 1 1 1 1 1 1];

    newPoints=wheel1Points;%initcylinderd*cylinderPoints;
    wheel1Handle=patch(newPoints(1,:),newPoints(2,:), 'b');

else %modify the figure by redrawing it at the location given
newPoints=itf*wheel1Points;
set(wheel1Handle, 'XData', newPoints(1,:), 'YData', newPoints(2,:));
end
```

#### A.6.9. ZIGZAG.M

```
function [y] = zigzag(x)
```

```
%determines definition of path that vehicle will take
x1 =[
    0
    2
    4
    6
    7
    7.5
];

y1 =[
    2
    4
    2
    4
    3
    3
];

cf=fit(x1,y1,'linearinterp');

[m,n]=size(x);
for i=1:n
y(i)=cf(x(i));
end

end
```

## APPENDIX B. FULL SIMULATION RESULTS

### B.1. FRONT WHEEL STEER – 2WF

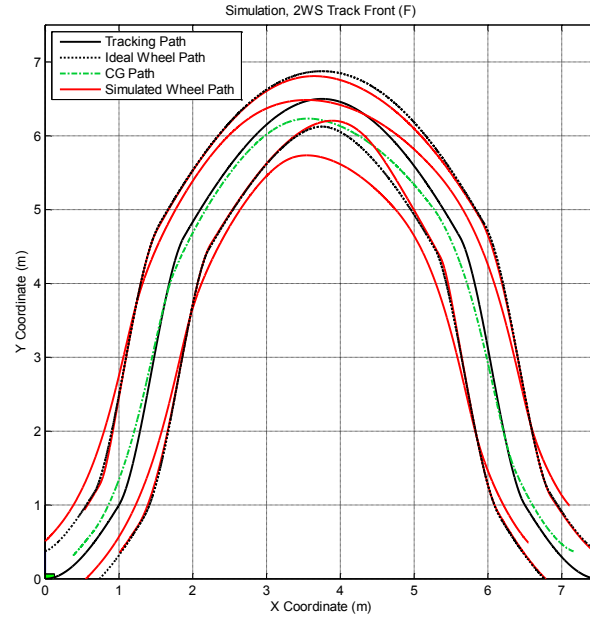


FIGURE B.1 – 2WF U-TURN NO VEHICLE

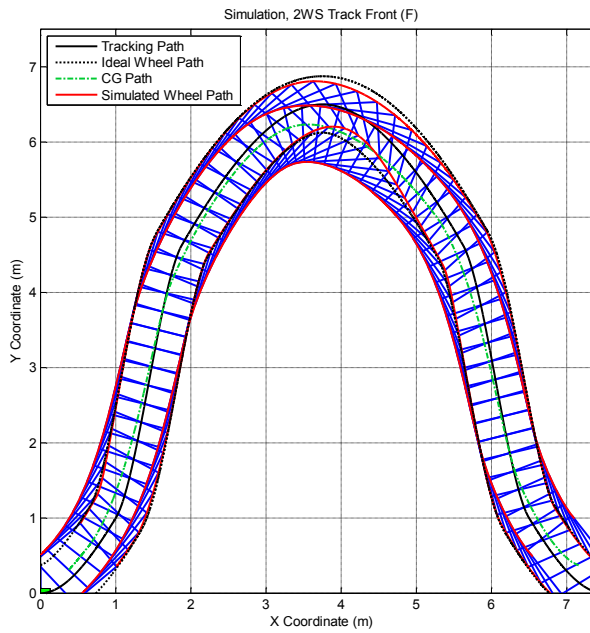
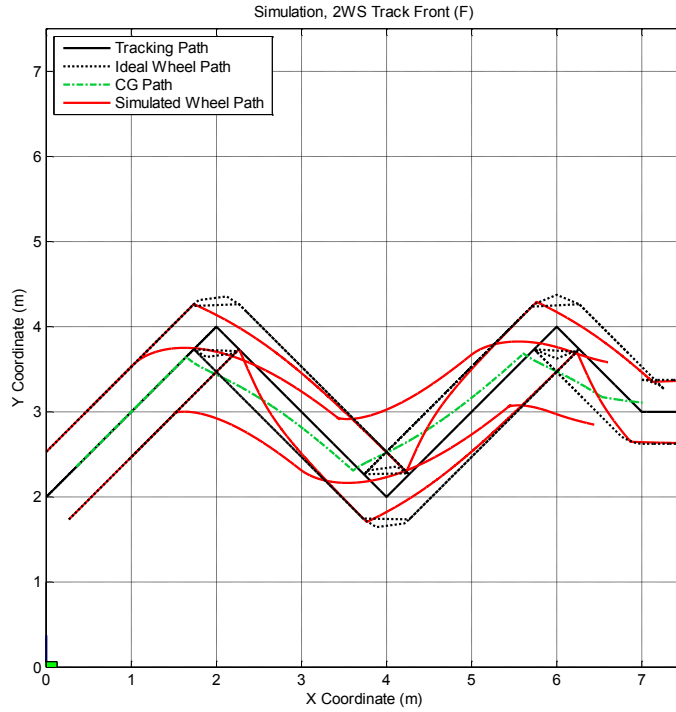
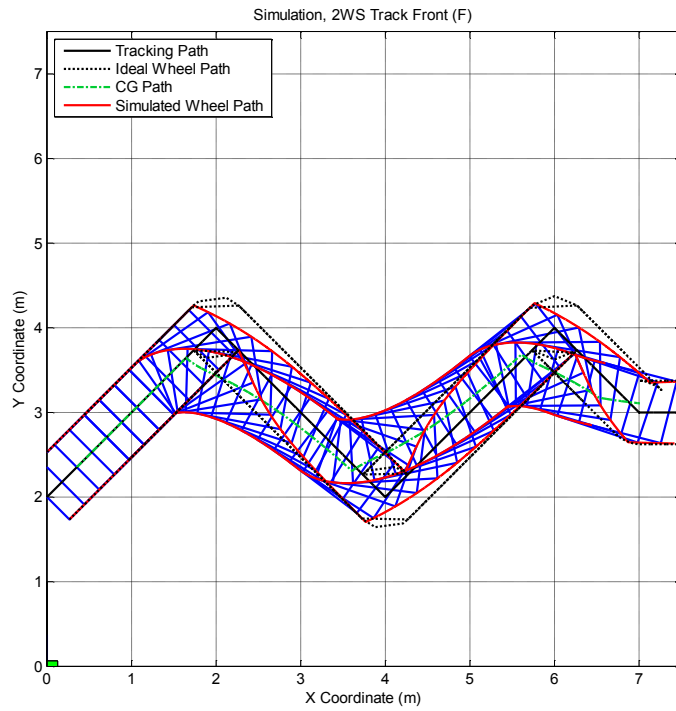


FIGURE B.2 – 2WF U-TURN WITH VEHICLE

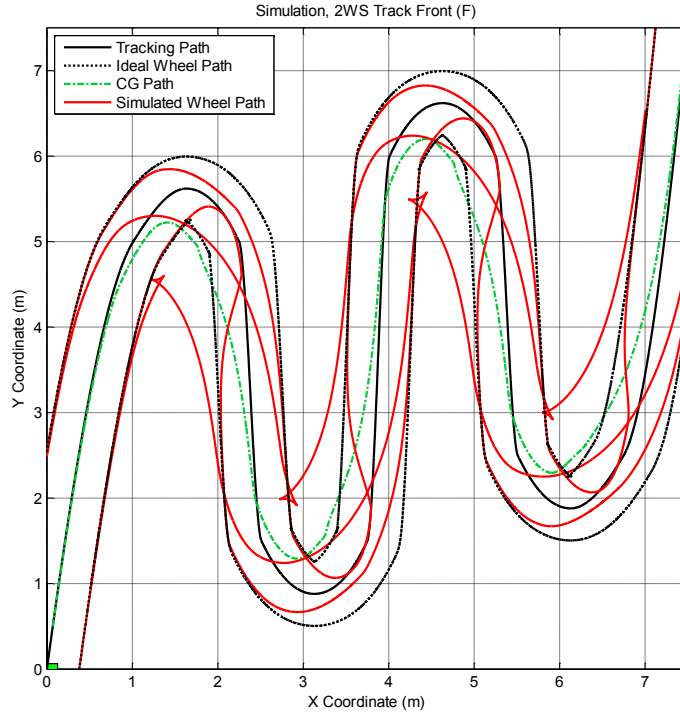


**FIGURE B.3 – 2WF ZIG-ZAG NO VEHICLE**

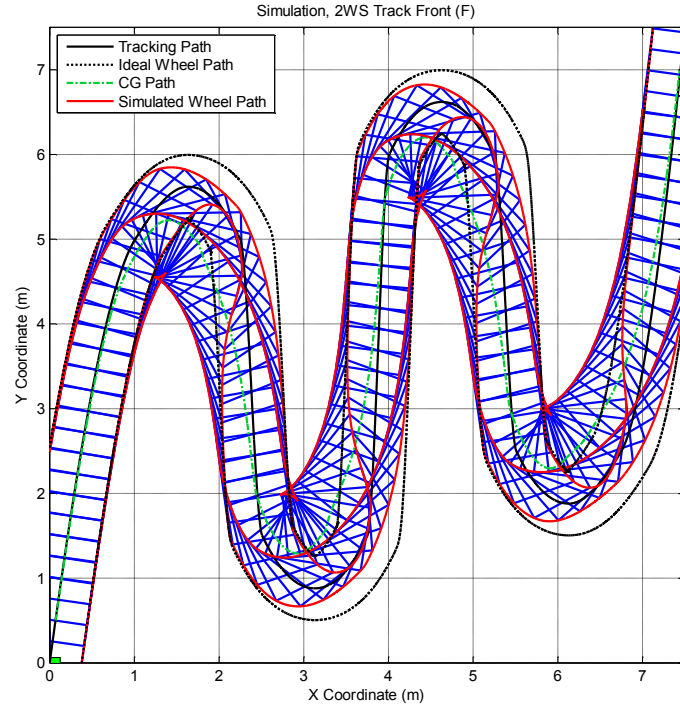


**FIGURE B.4 – 2WF ZIG-ZAG WITH VEHICLE**





**FIGURE B.5 – 2WF S-CURVE NO VEHICLE**



**FIGURE B.6 – 2WF S-CURVE WITH VEHICLE**

## B.2. FRONT AXLE TRACKING - 4FM

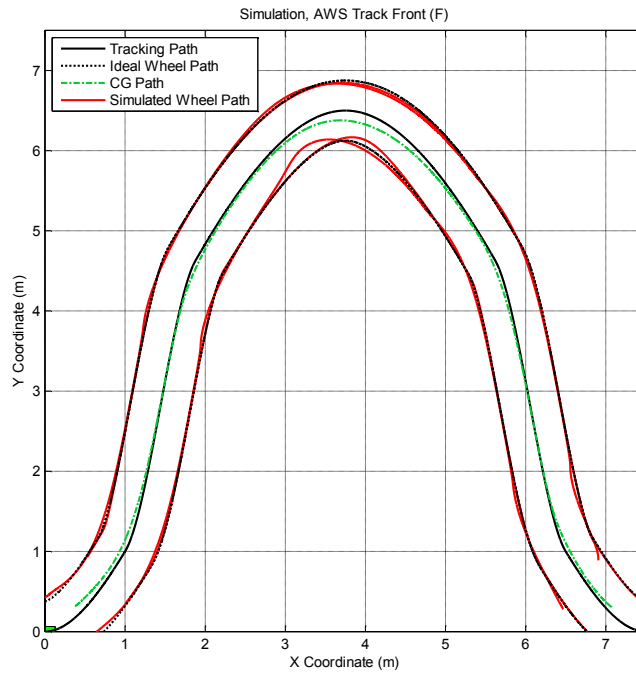


FIGURE B.7 - 4FM U-TURN NO VEHICLE

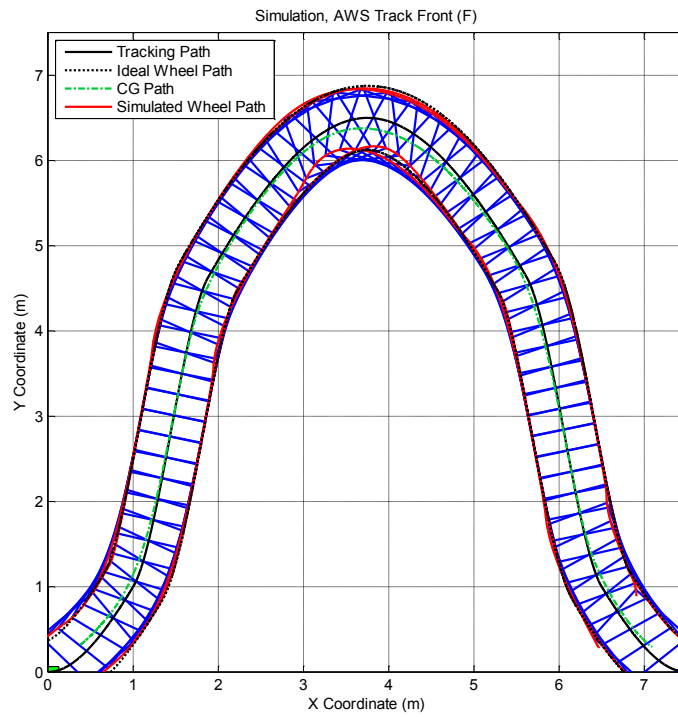
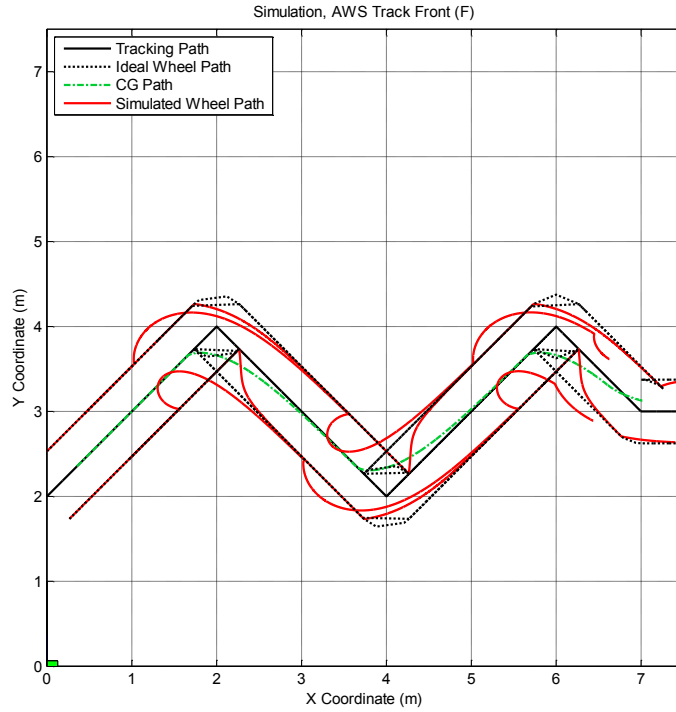
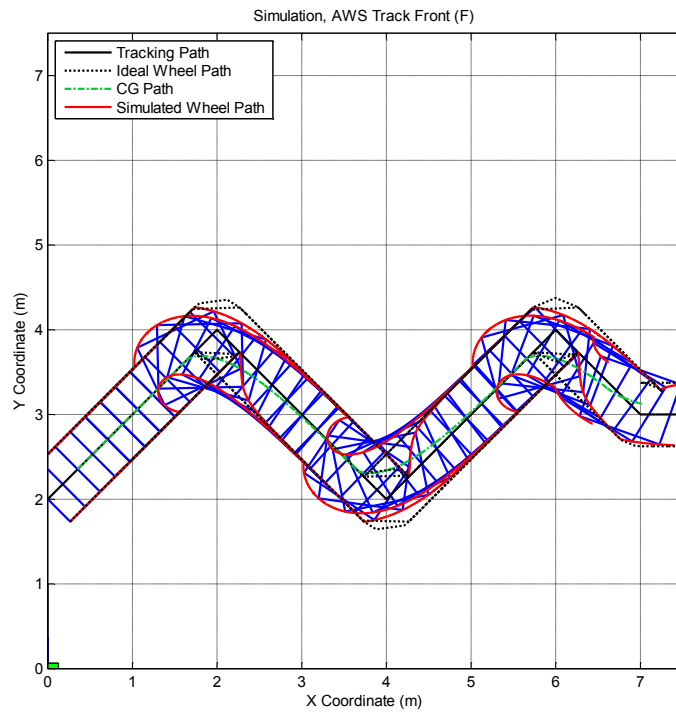


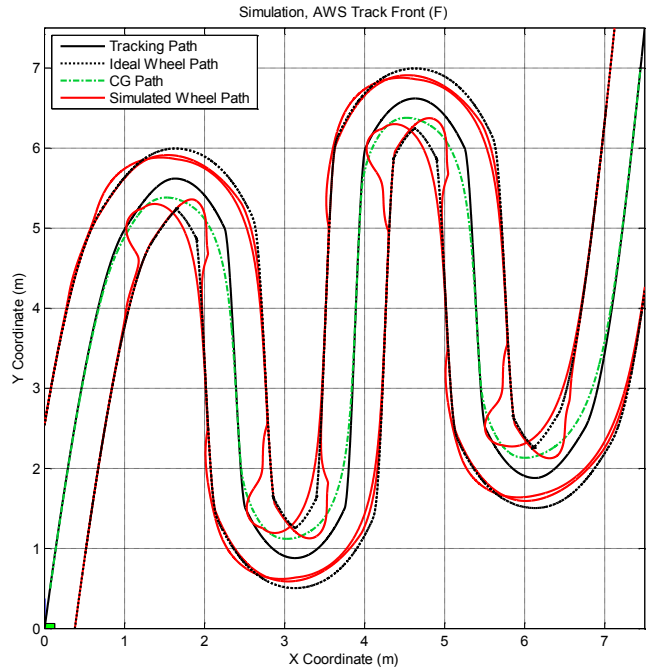
FIGURE B.8 - 4FM U-TURN WITH VEHICLE



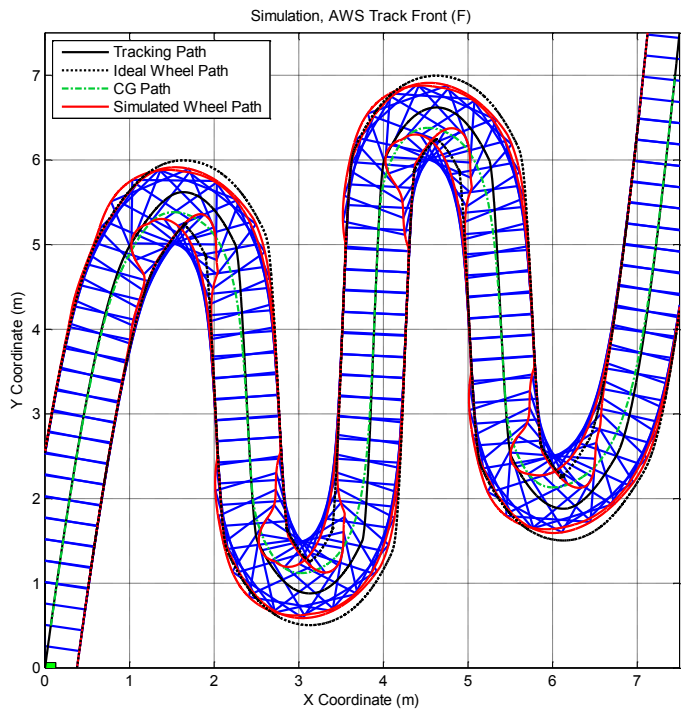
**FIGURE B.9 – 4FM ZIG-ZAG NO VEHICLE**



**FIGURE B.10 – 4FM ZIG-ZAG WITH VEHICLE**



**FIGURE B.11 – 4FM S-CURVE NO VEHICLE**



**FIGURE B.12 – 4FM S-CURVE WITH VEHICLE**

### B.3. CENTER OF GRAVITY TRACKING - 4CG

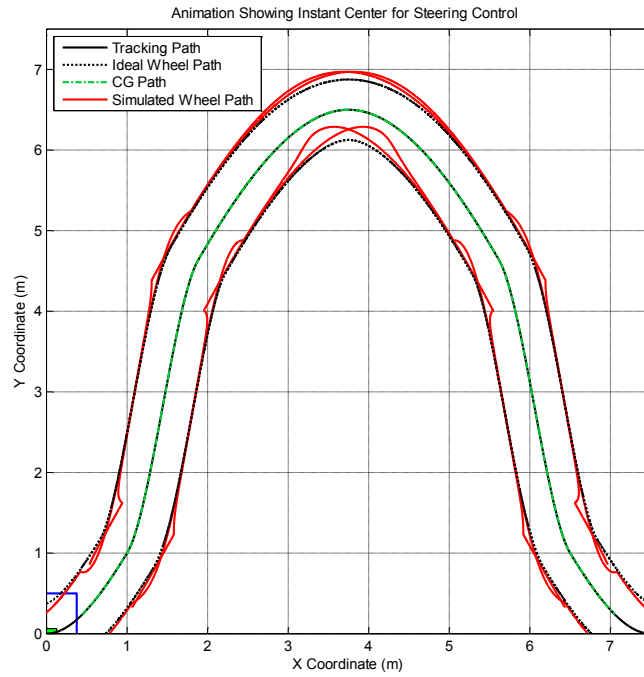


FIGURE B.13 - 4CG U-TURN NO VEHICLE

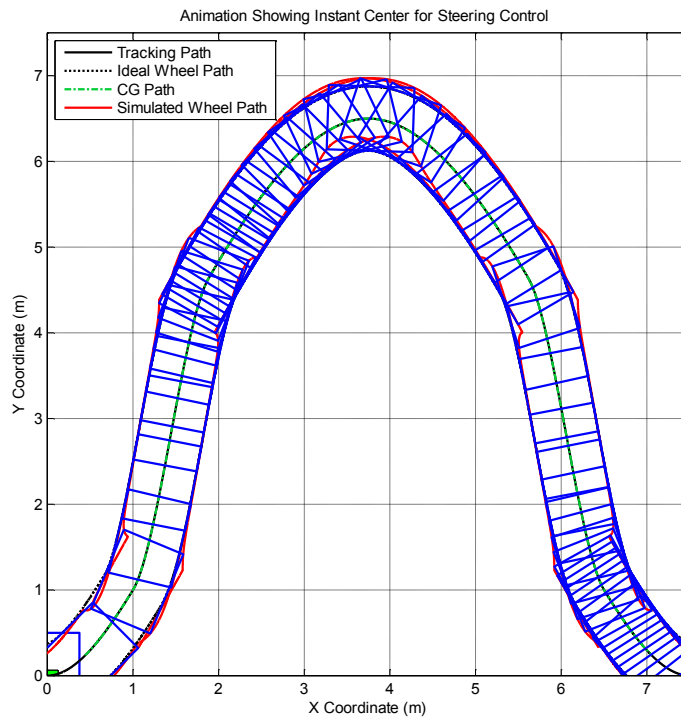
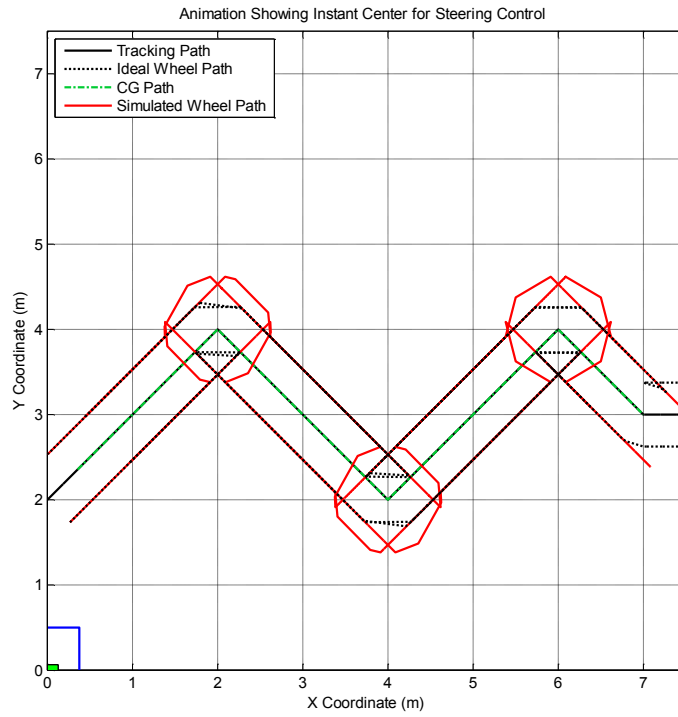
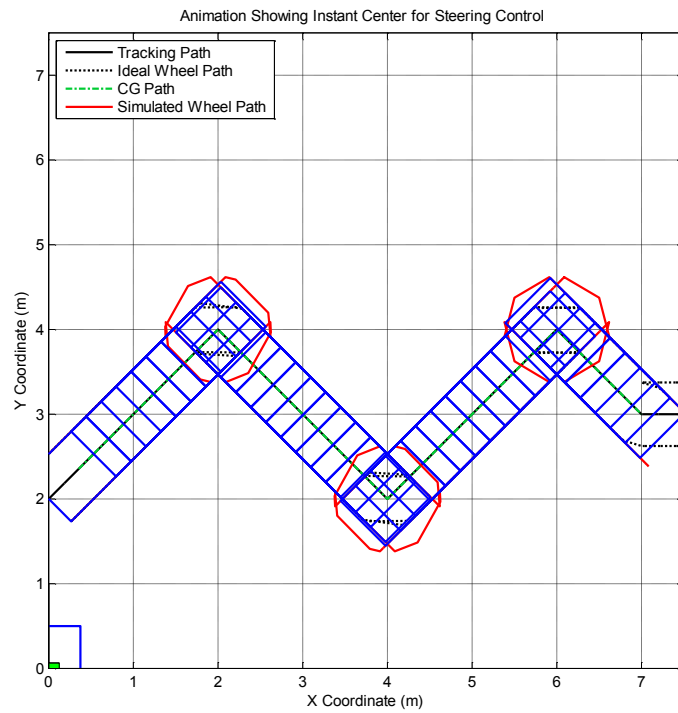


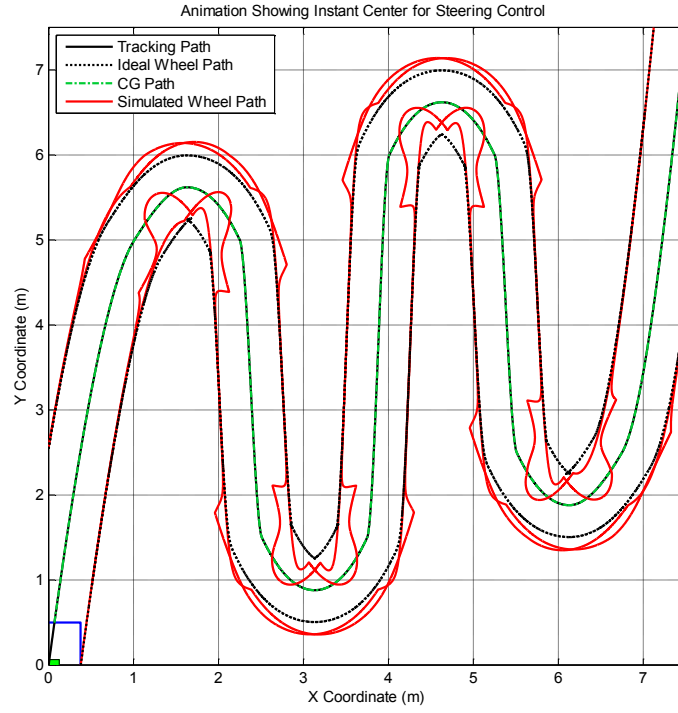
FIGURE B.14 - 4CG U-TURN WITH VEHICLE



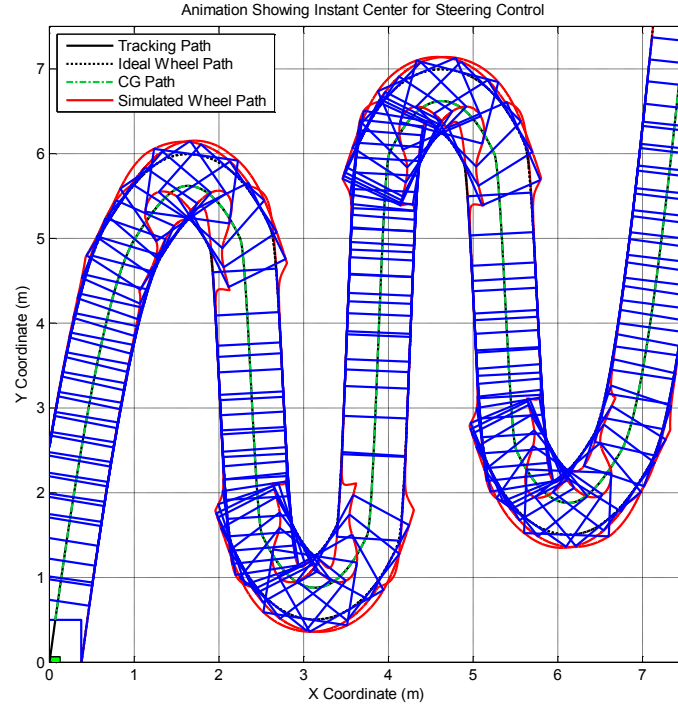
**FIGURE B.15 - 4CG ZIG-ZAG NO VEHICLE**



**FIGURE B.16 - 4CG ZIG-ZAG WITH VEHICLE**



**FIGURE B.17 - 4CG S-CURVE NO VEHICLE**



**FIGURE B.18 - 4CG S-CURVE WITH VEHICLE**

#### B.4. FRONT AND REAR AXLE TRACKING – 4FM

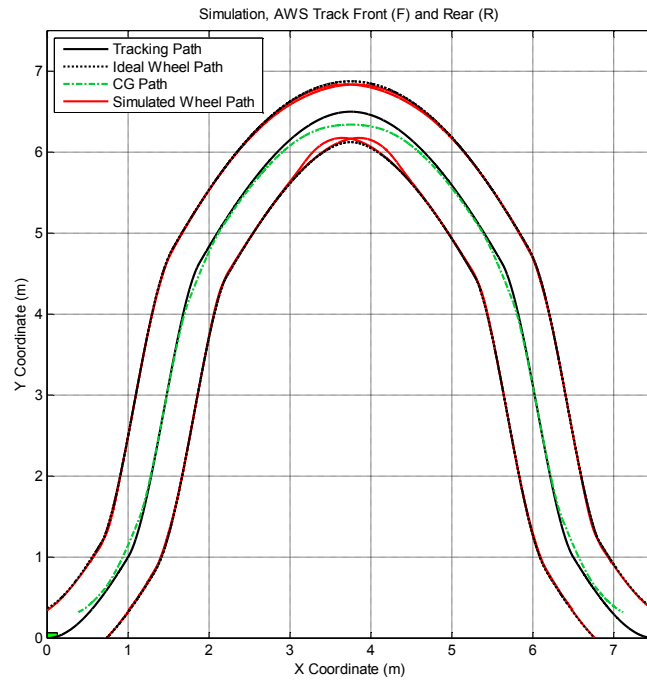


FIGURE B.19 – 4FM U-TURN NO VEHICLE

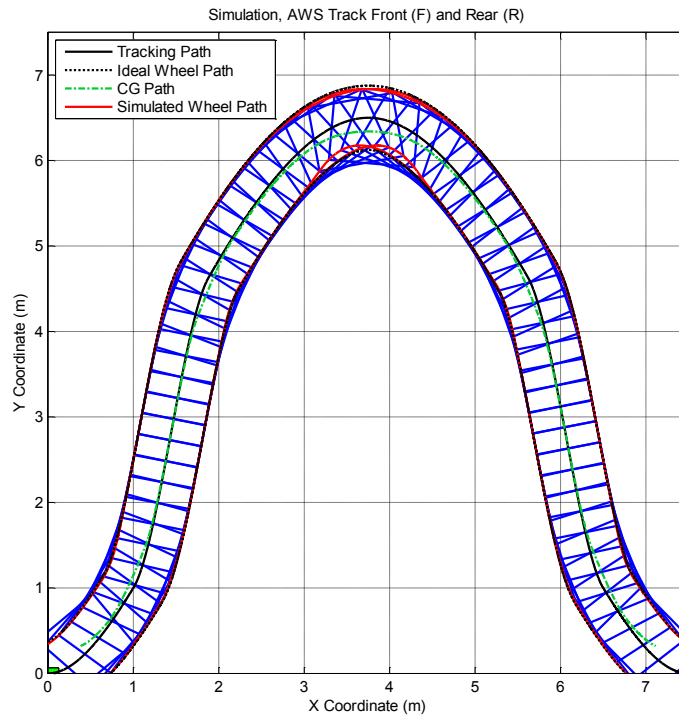
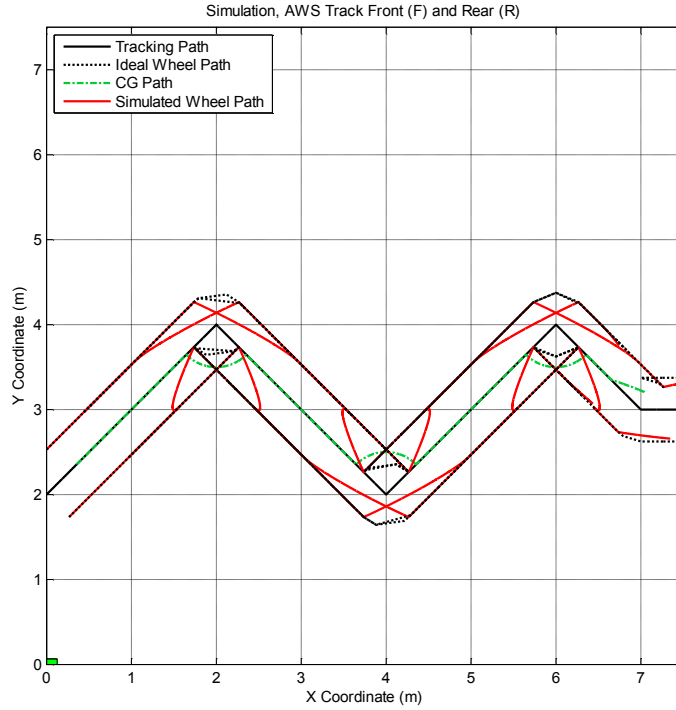
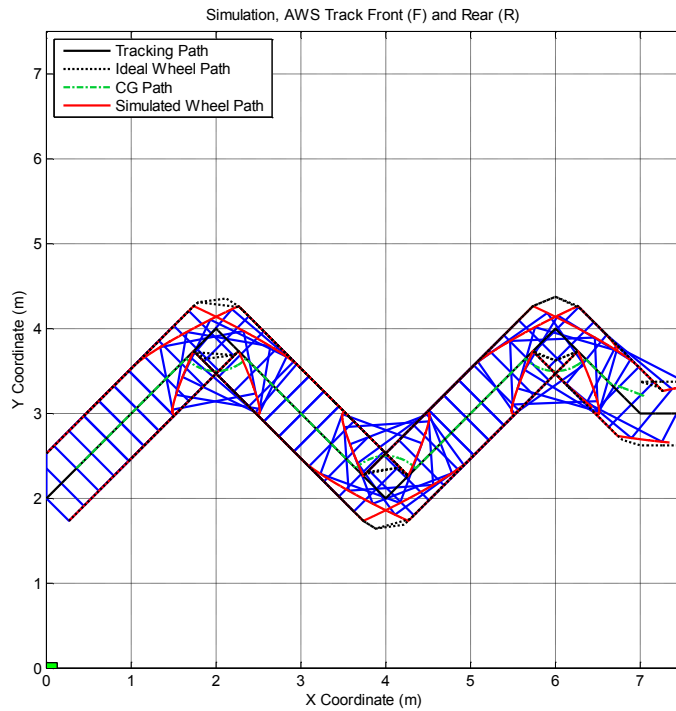


FIGURE B.20 – 4FM U-TURN WITH VEHICLE

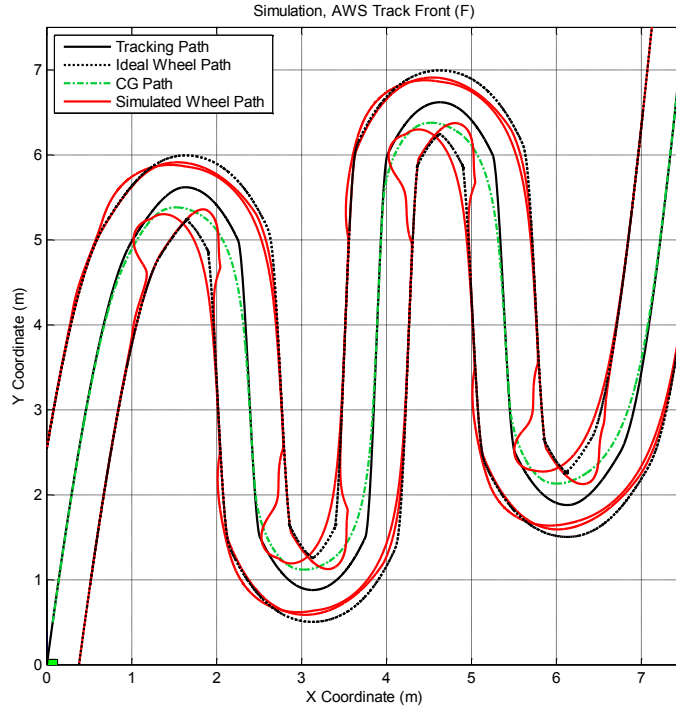




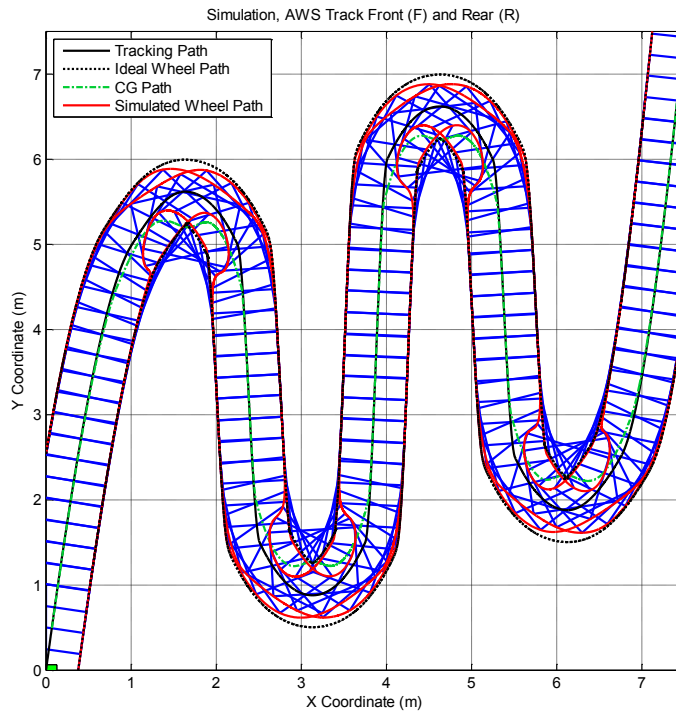
**FIGURE B.21 – 4FM ZIG-ZAG NO VEHICLE**



**FIGURE B.22 – 4FM ZIG-ZAG WITH VEHICLE**



**FIGURE B.23 – 4FM S-CURVE NO VEHICLE**



**FIGURE B.24 – 4FM S-CURVE WITH VEHICLE**

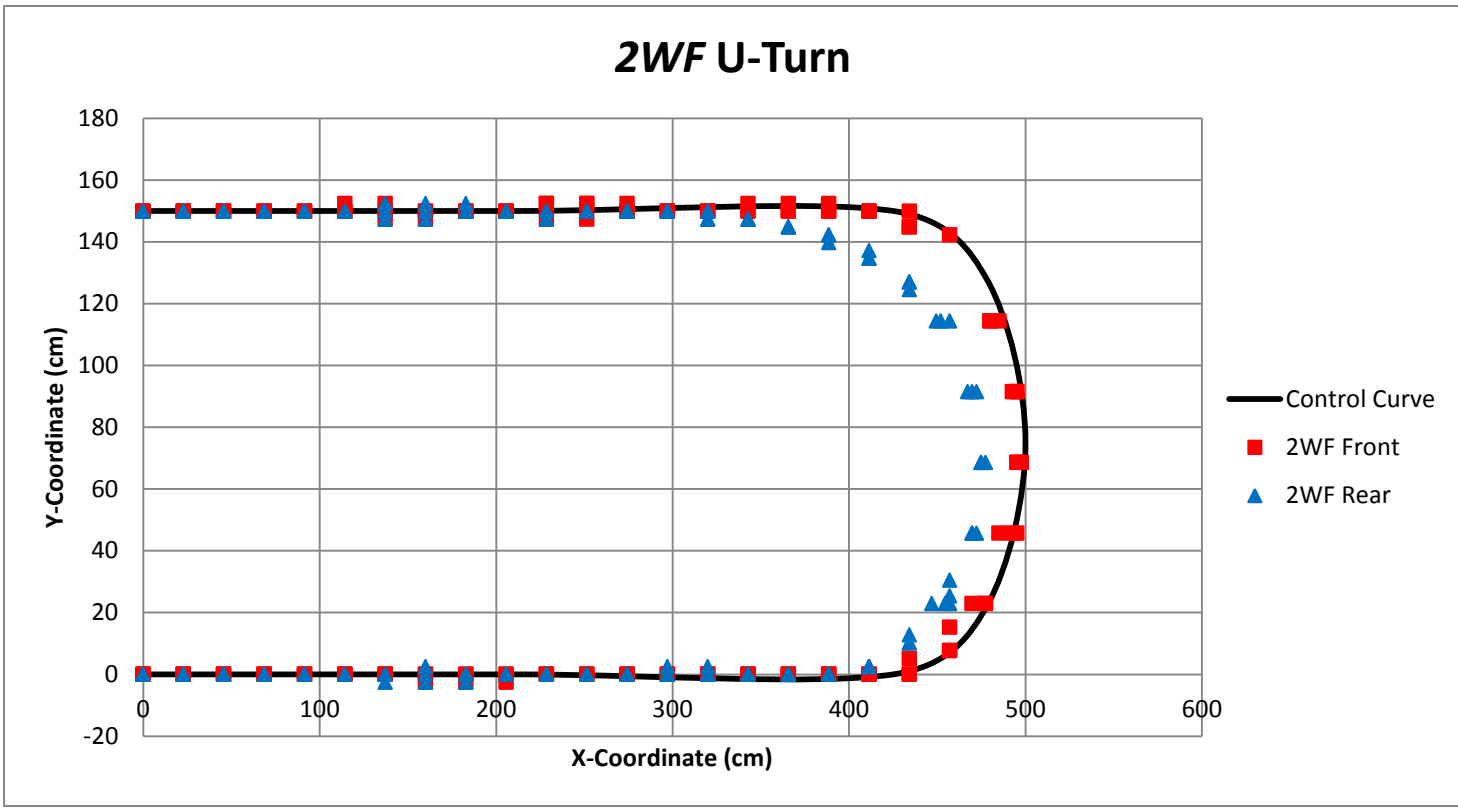


FIGURE C.1 – 2WF TRACKING U-TURN

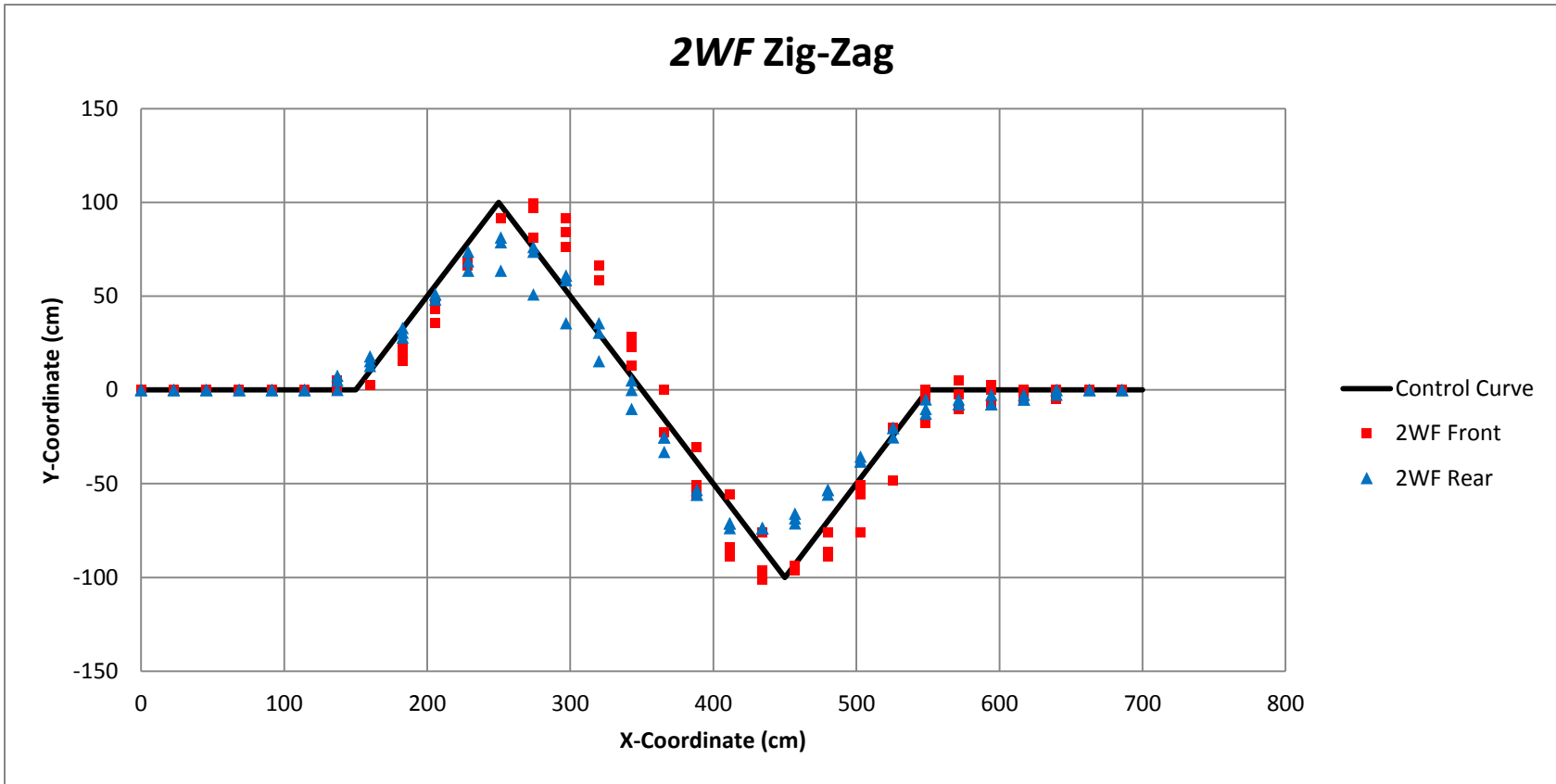


FIGURE C.2 – 2WF TRACKING ZIG-ZAG

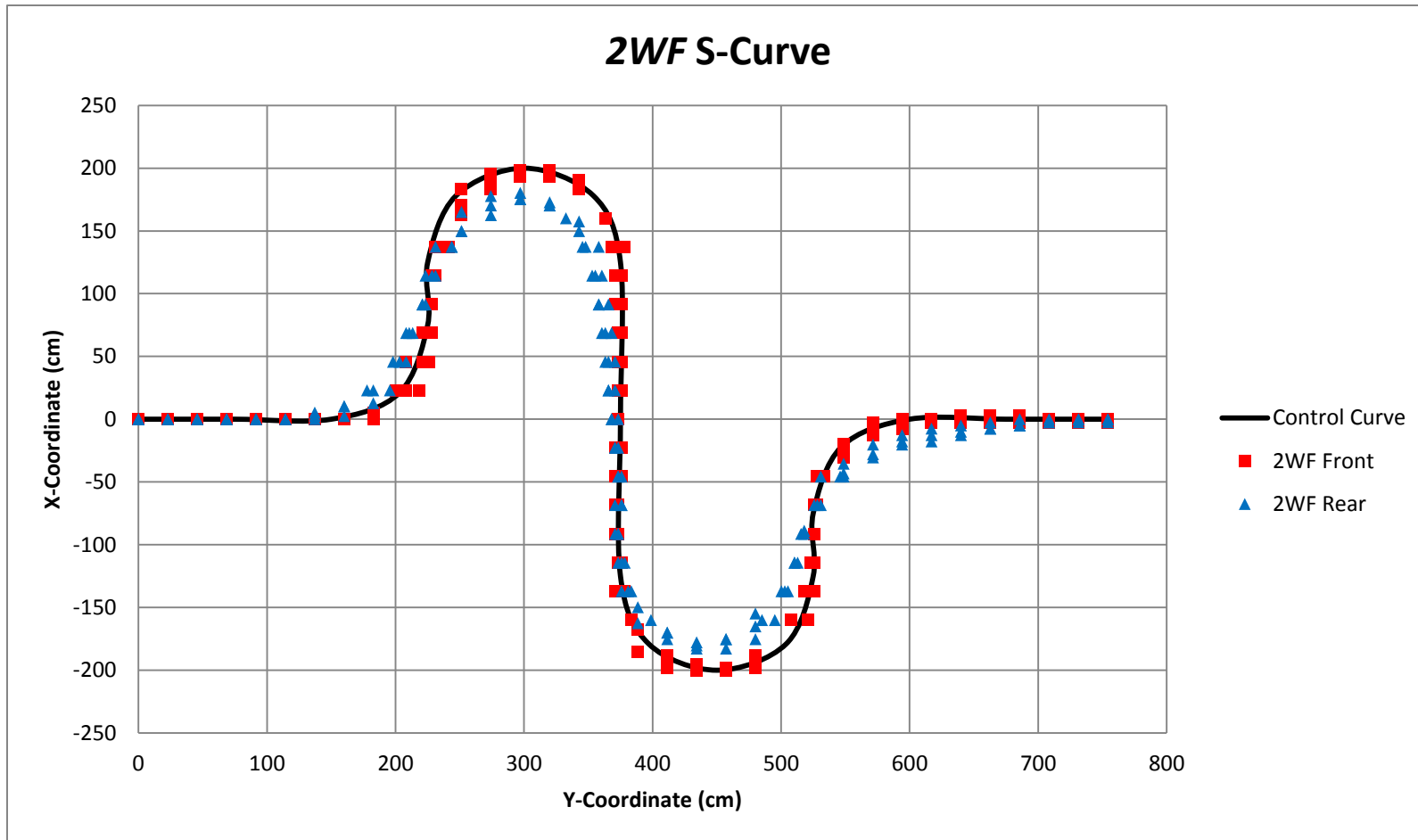


FIGURE C.3 – 2WF TRACKING S-CURVE

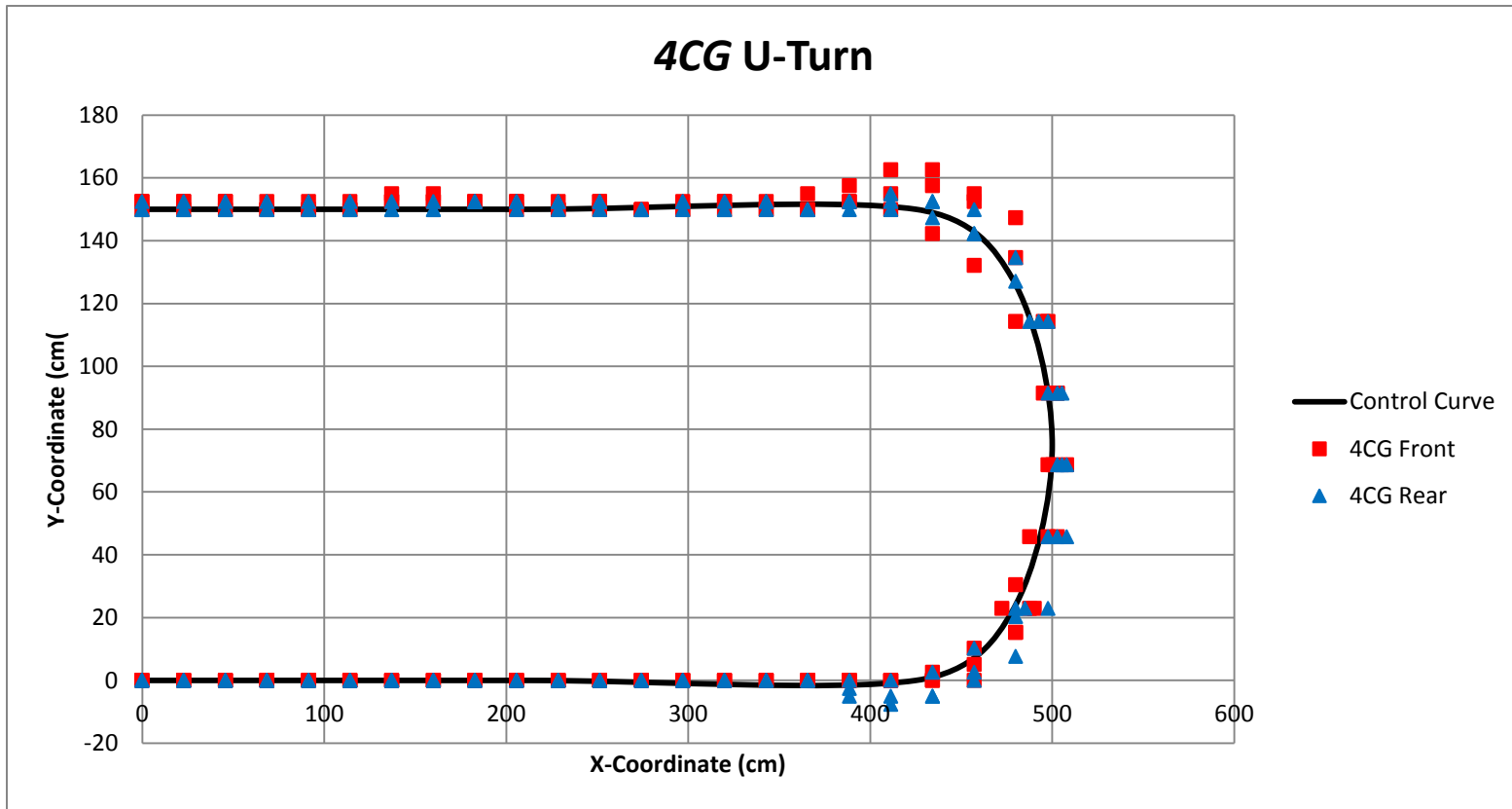


FIGURE C.4 - 4CG TRACKING U-TURN

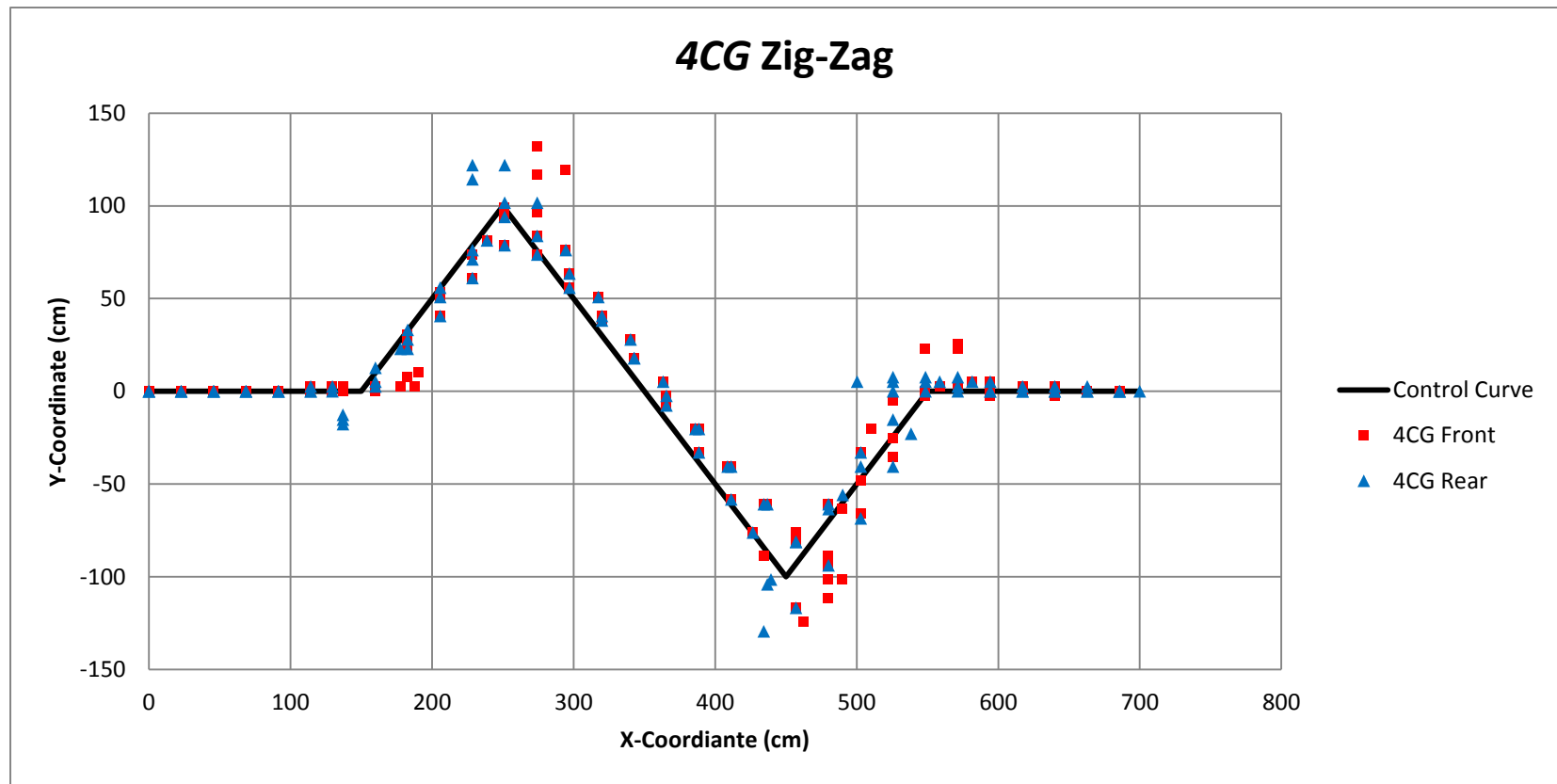


FIGURE C.5 - 4CG TRACKING ZIG-ZAG

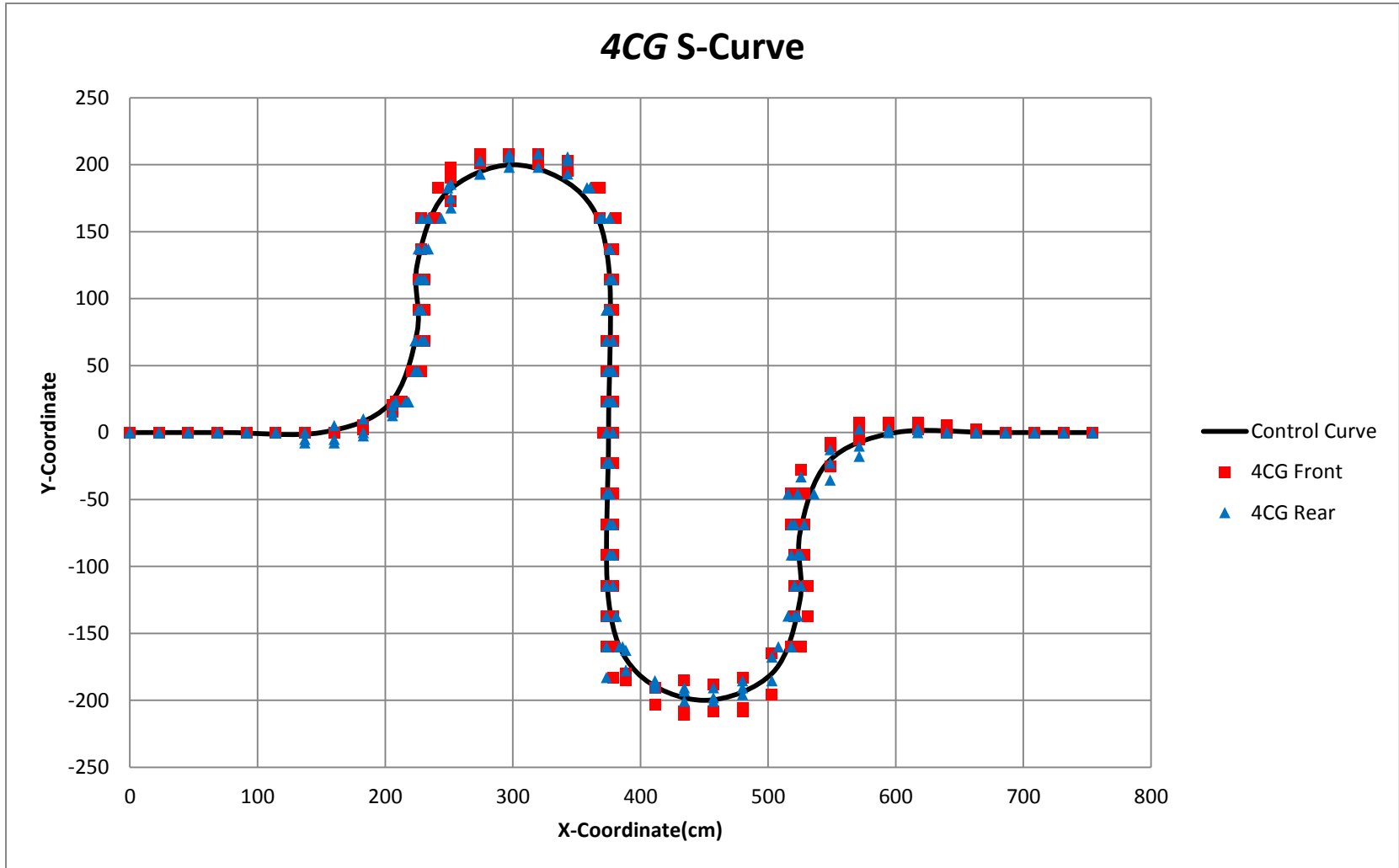


FIGURE C.6 – 4CG TRACKING S-CURVE



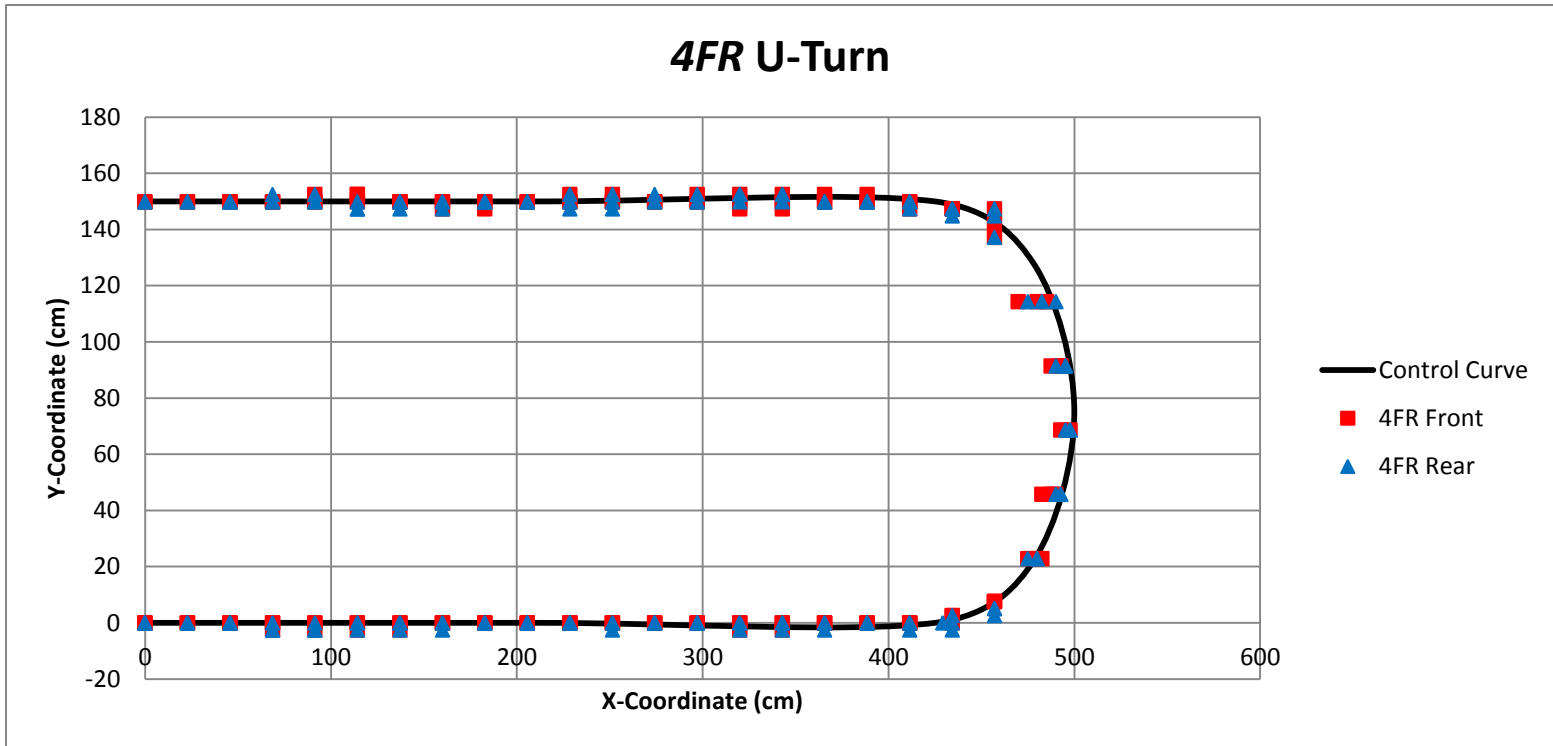


FIGURE C.7 - 4FR TRACKING U-TURN

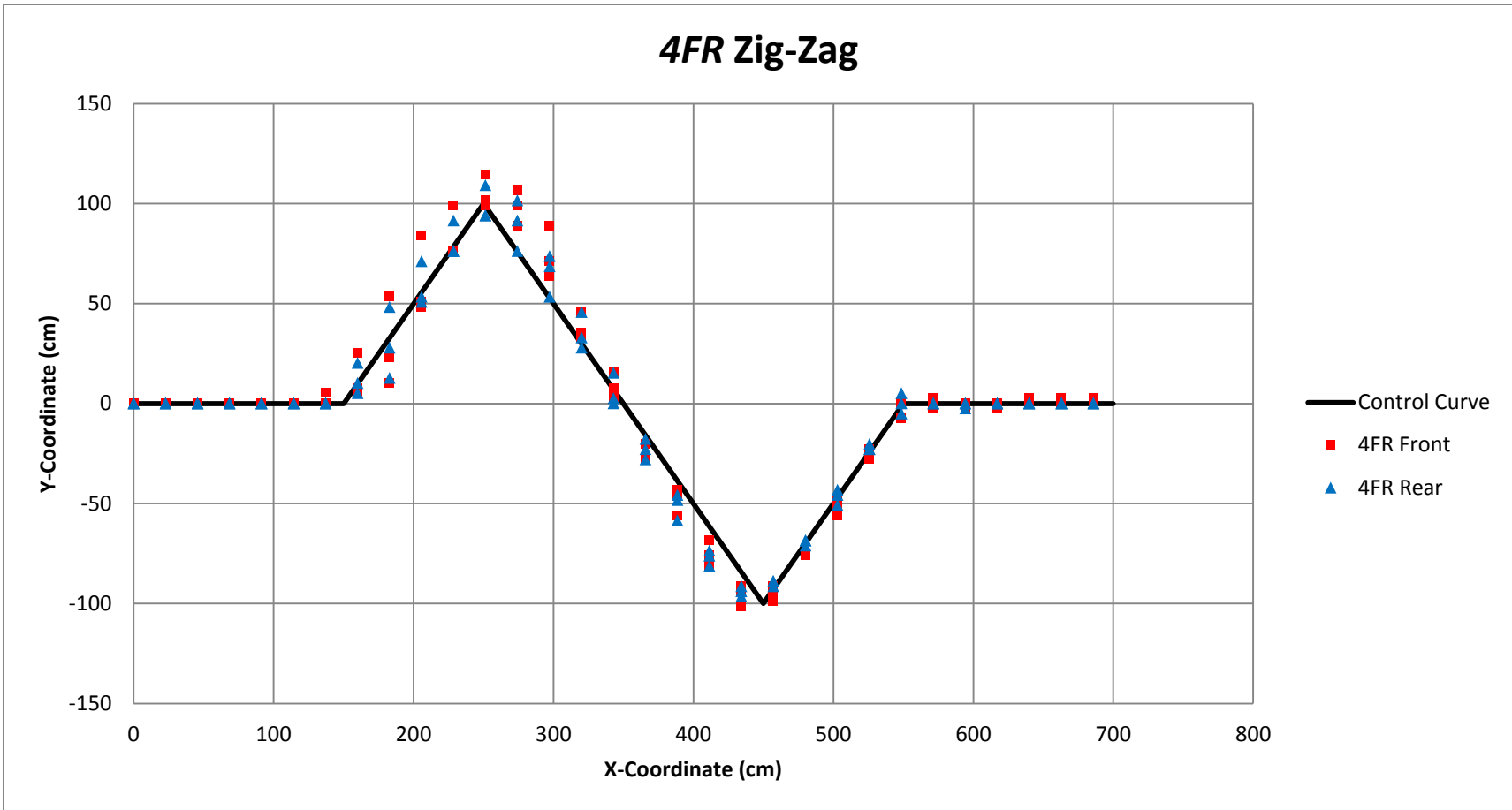


FIGURE C.8 – 4FR TRACKING ZIG-ZAG

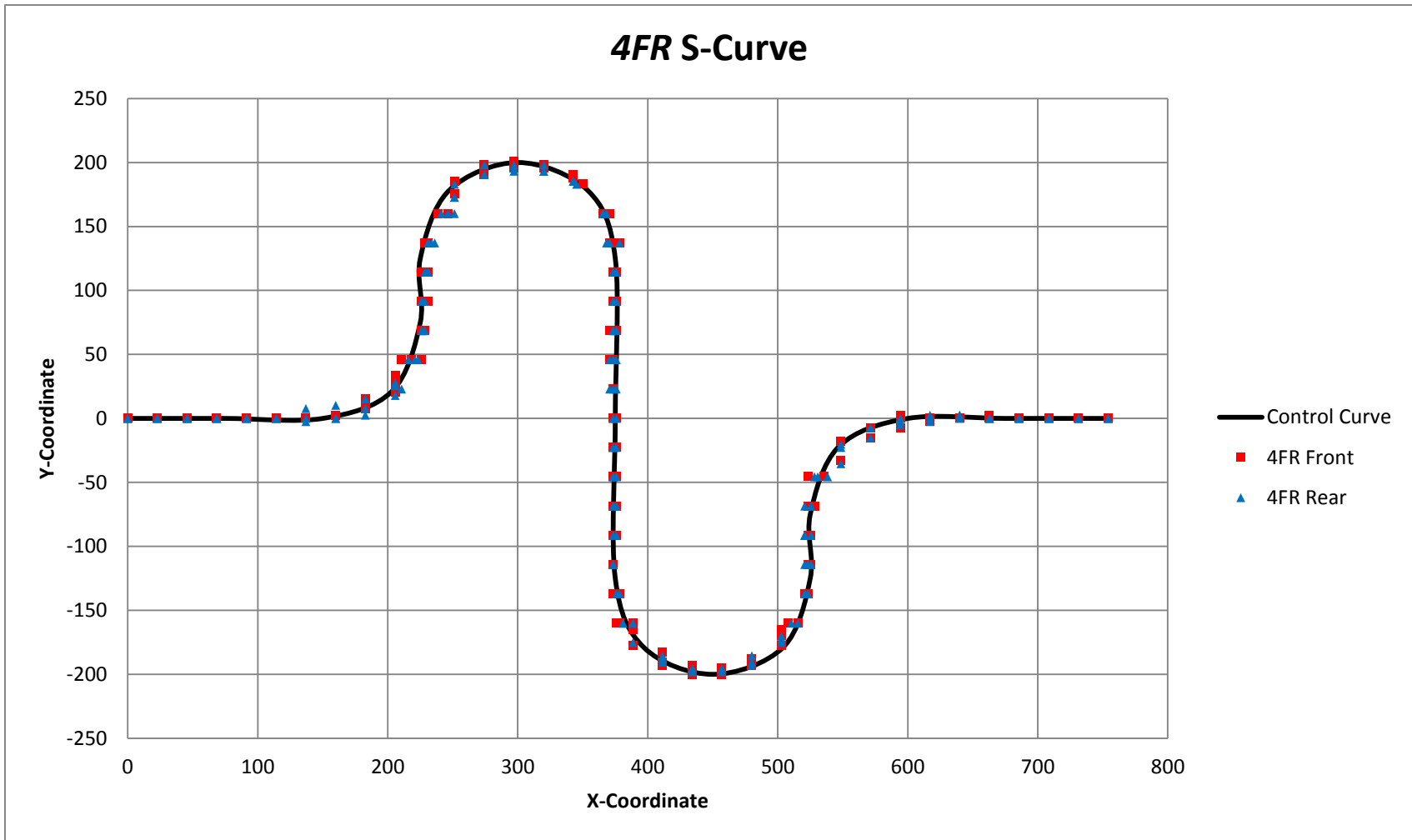


FIGURE C.9 – 4FR TRACKING S-CURVE

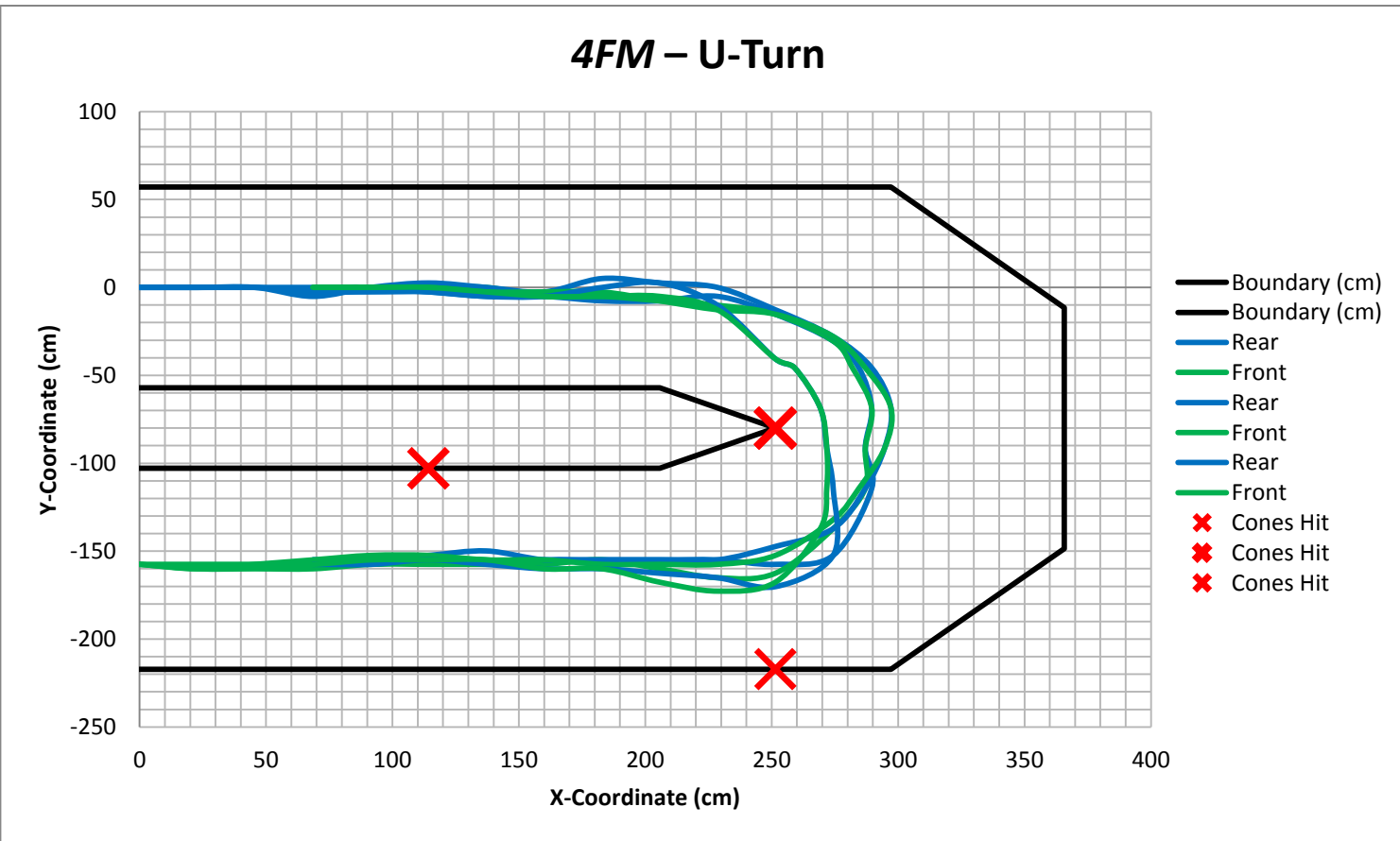


FIGURE D.1 - 4FM TRACKING U-TURN

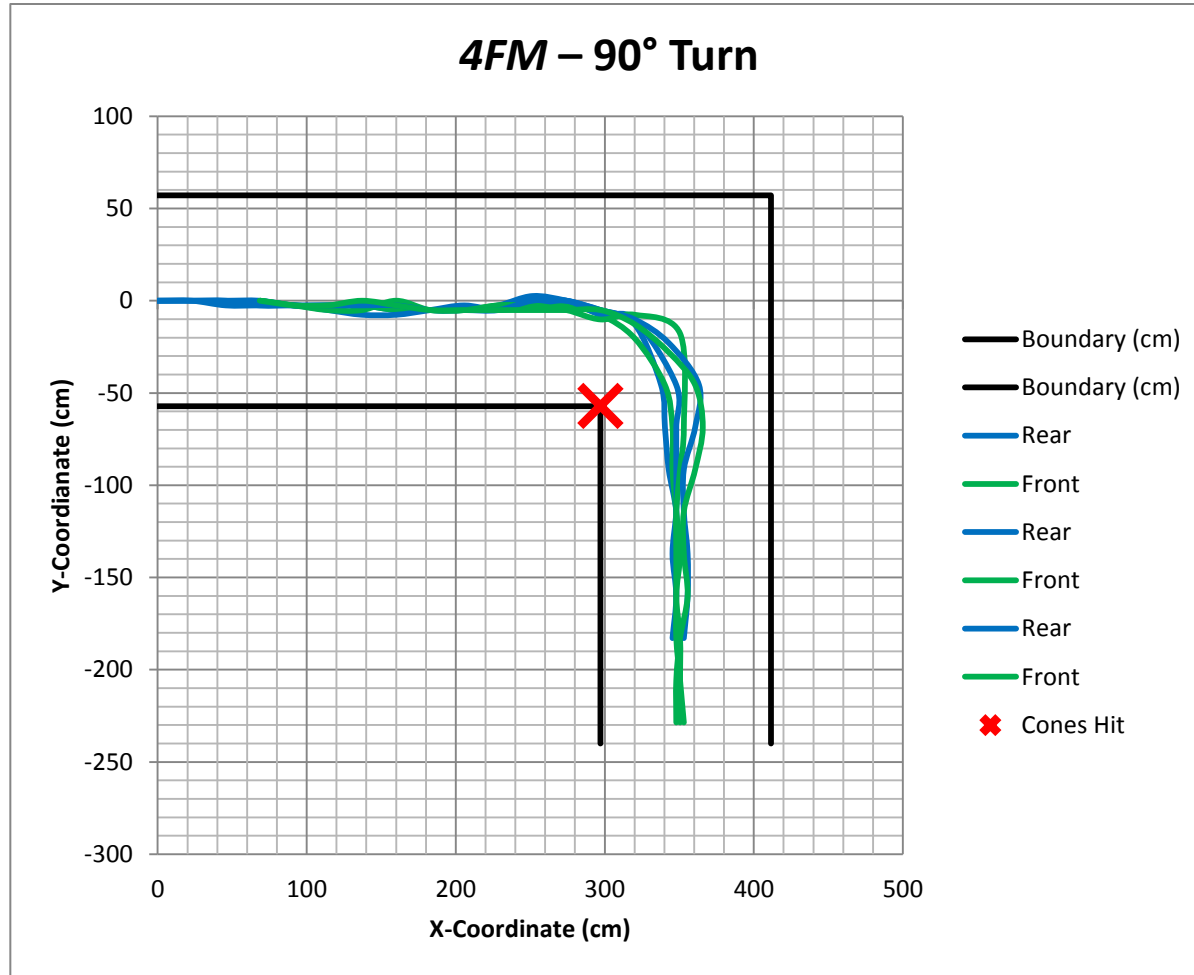


FIGURE D.2 – 4FM TRACKING 90° TURN

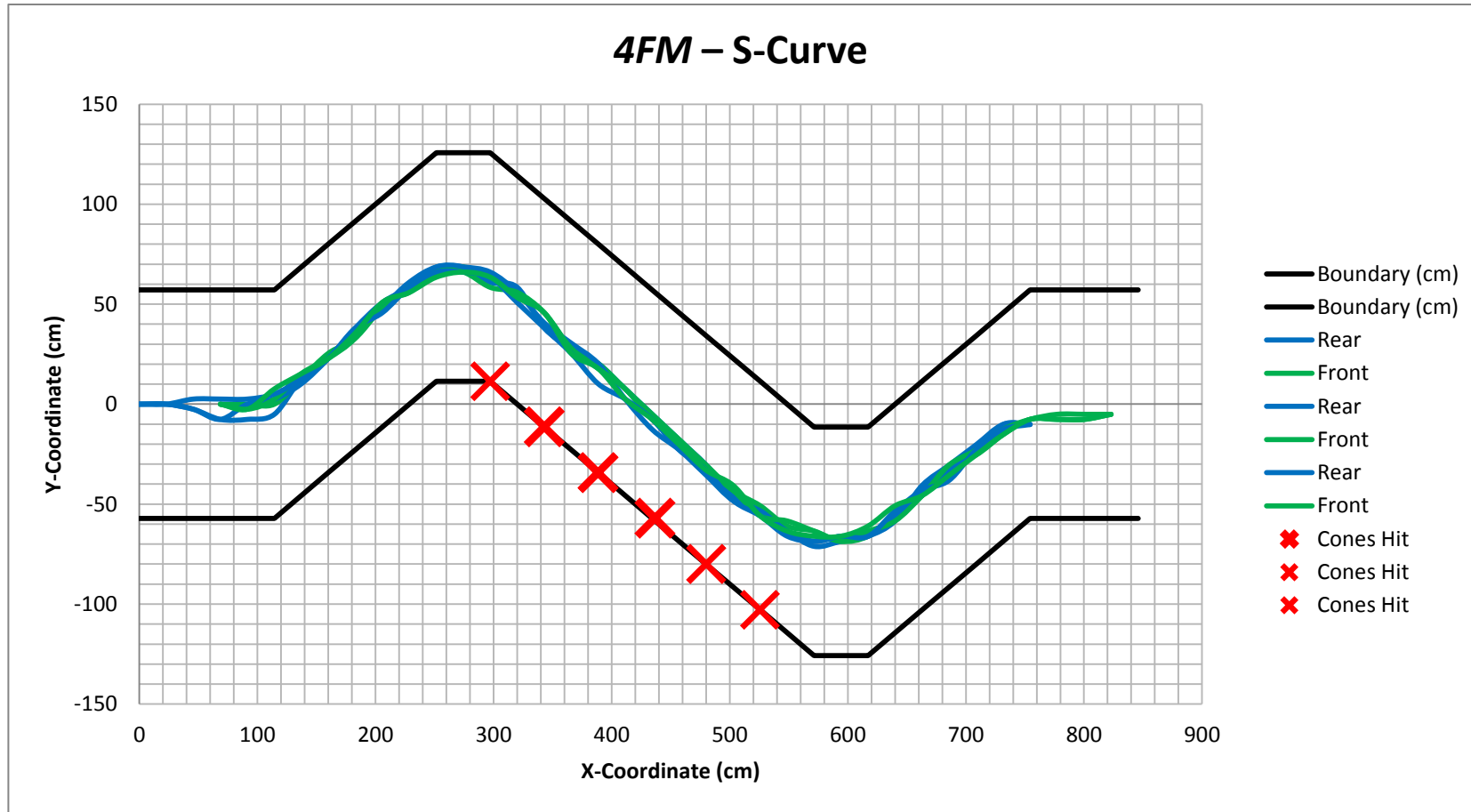


FIGURE D.3 – 4FM TRACKING S-CURVE

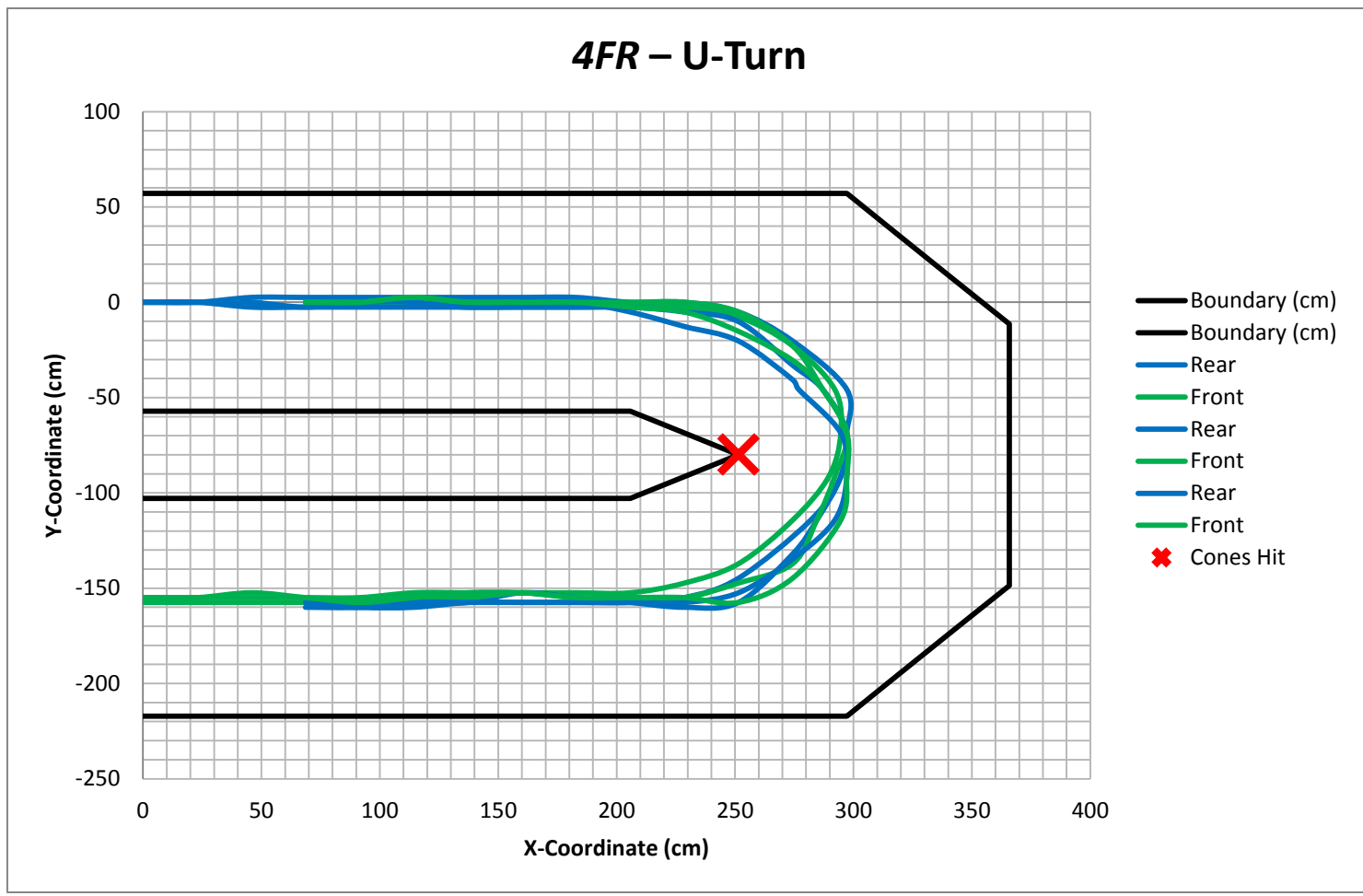


FIGURE D.4 - 4FR TRACKING U-TURN

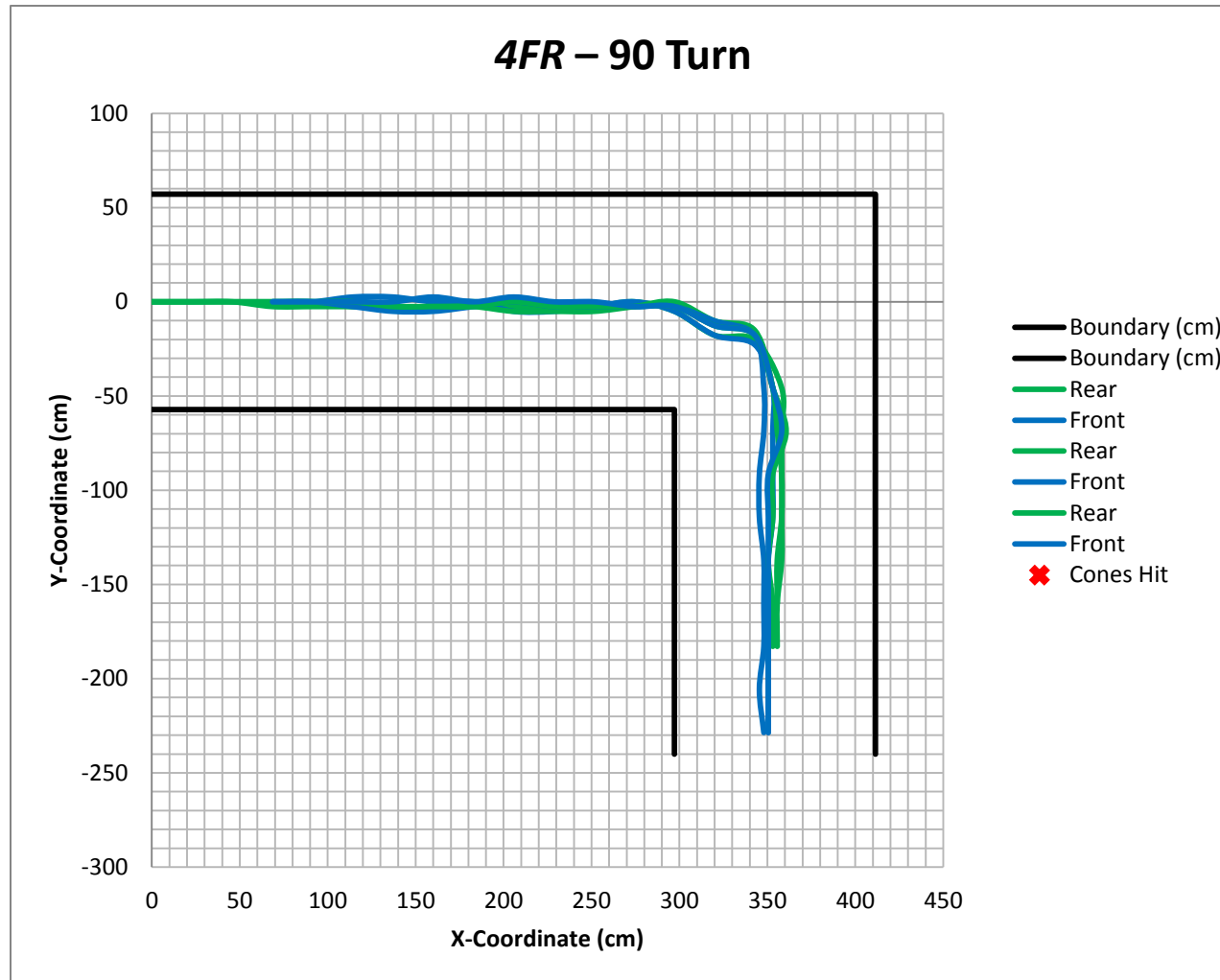


FIGURE D.5 – 4FR TRACKING 90° TURN



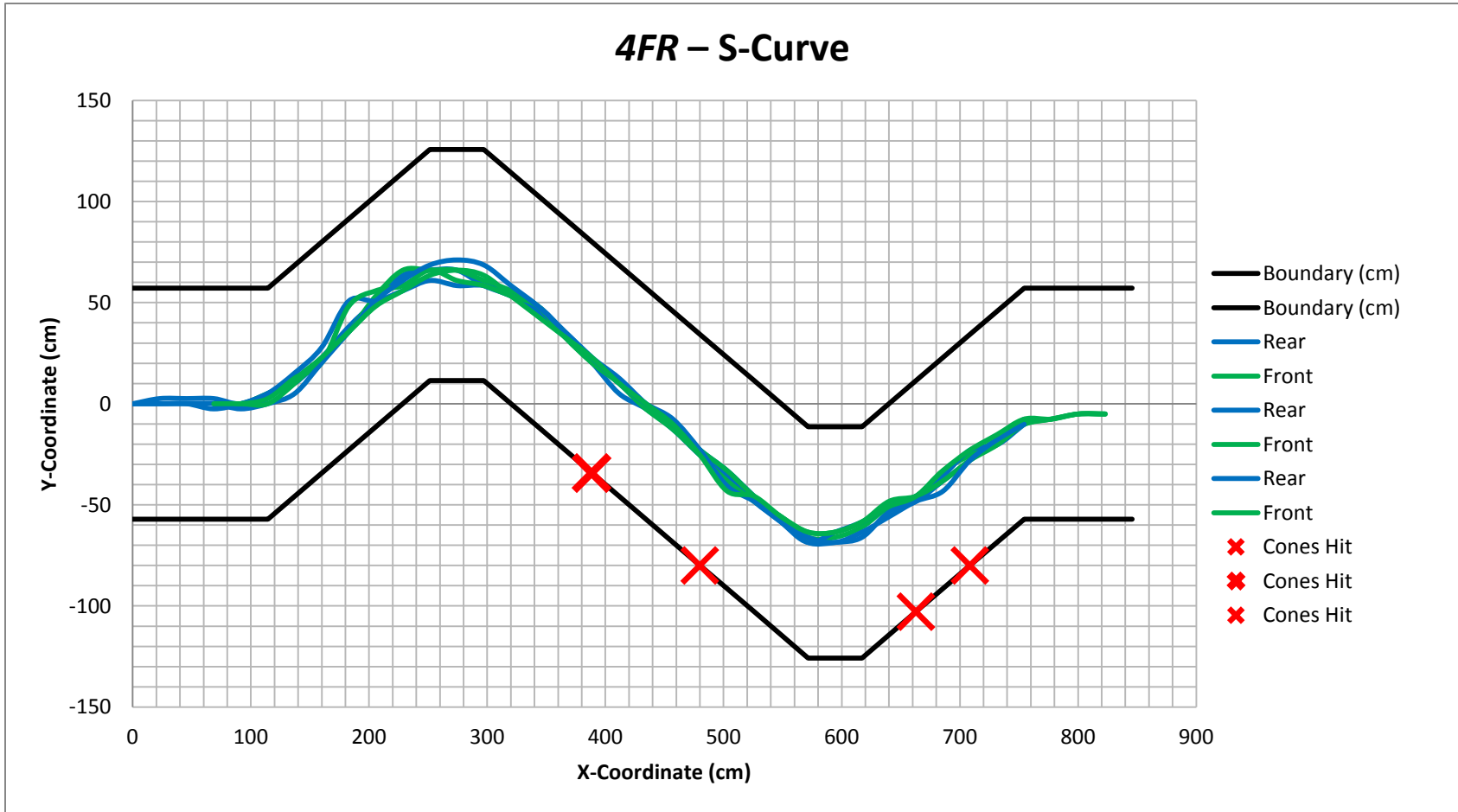


FIGURE D.6 – 4FR TRACKING S-CURVE

## APPENDIX E. INDIVIDUAL WHEEL UNIT CONTROLLER C CODE (4X)

### E.1. WCU\_MAIN.C

```
//~~~~~ WCU_Main.c ~~~~~

//<editor-fold defaultstate="collapsed" desc="Program Heading">
/*****
 * 2005 Microchip Technology Inc.
 *
 * FileName:      WCU_Main.c
 * Dependencies:  Header (.h) files if applicable, see below
 * Processor:     dsPIC33FJ128MC802
 * Compiler:      MPLAB C30 v3.00 or higher
 * Module:        Local Wheel Control Unit
 *
 *
 * REVISION HISTORY:
 *~~~~~
 * Author          Date      Comments on this revision
 *~~~~~
 * Jon Nistler     05/26/11  Revision based on Nikhil Gupta's code
 *~~~~~
 *
 * ADDITIONAL NOTES:
 * This code is tested on target board with dsPIC33FJ128MC802 controller
 * The Processor starts with the Internal oscillator without PLL enabled
 * and then the Clock is switched to PLL Mode.
 *****/
//</editor-fold>

//<editor-fold defaultstate="collapsed" desc="uC Pin Configuration">
// Needs to be updated
//
//          =====
//      Steering Pot - AN1 -|1      28|- Capacitor1?
//          - AN2 -|2      27|- Capacitor1?
//      Serial Clock - SPI1 -|3      26|- PWM1L1 - ??
//      Not Connected - x -|4      25|- Not Connected - x
//      Serial Data In - SPI1 -|5      24|- Not Connected - x
//      Slave Select - SPI1 -|6      23|- PWM1H2 - Tractive DC Motor PWM
//          Ground - Vdd -|7      22|- Not Connected - x
//      Not Connected - x -|8      21|- Not Connected - x
//      OscillatorPin1 - OSC1 -|9      20|- Capacitor 2?
//      OscillatorPin2 - OSC2 -|10     19|- Capacitor 2?
//          -|11     18|- LATB9 - Steering Brake
//Steering Left/Right - RA4 -|12     17|- LATB8 - Steering Speed Control?
//          +3.3v - Vss -|13     16|- Not Connected - x
//          -|14     15|- LATB6 - Steering Motor Brake
//          =====
//</editor-fold>
```

```

//<editor-fold defaultstate="collapsed" desc="Definitions and Libraries">
#include "p33FJ128MC802.h"
#include "math.h"
#include "WCU_Drivers.h"

_FGS(GWRP_OFF & GCP_OFF);
_FOSCSEL(FNOSC_FRC);
_FOSC(FCKSM_CSECMD & IOL1WAY_OFF & OSCIOFNC_OFF & POSCMD_HS);
_FWDT(FWDTEN_OFF);
_FPOR(PWMPIN_OFF & HPOL_ON & LPOL_ON)
//</editor-fold>

//<editor-fold defaultstate="collapsed" desc="Global Variables">
// Declare Global Variables
double df;
double dr;
double Vg;

double old_df = 0.0; //hold angles to check if there has been a change
double old_dr = 0.0;

int Mode = 1; //Determines steering mode of robot
int RemoteEnable=1; //Doesn't allow robot to break steering while loop

int PWMZero; //Duty which represents zero velocity
int ErrorFlag = 1; //When this is set, an error has occurred and the unit
//has lost contact with the VCU set speed and
char breaksteer = 0; //Steering while loop will remain until it gets a new variable

double steering_angle = 0.0; // Radians

int speed = 0; // 0-100%, will be calculated from Vg
int old_speed = 0;

//</editor-fold>

//**** Define Wheel Unit ****
int unit = 2; // Unit: 1=FL, 2=FR, 3=RR, 4=RL

#define MaxDuty 800 //Define the maximum duty of the DC wheel motor
//100=min, 800=max (forward)
#define PotWindow 3 //Error window (in bits) for steer controller (plus or minus)
//~3 bits/degree

//===== Speed and Angle Calculation =====
void Calc_Speed_n_Angle(float Vg1, float df1, float dr1)
{ //Takes in Vg (units of duty 0-100%), df (radians), and dr(radians)
  //and adjusts global variables steering_angle and speed.
  float W = 26.0;
  float H = 23.0;

  double wheelangle;
  double wheelspeed;

```

```

//<editor-fold defaultstate="collapsed" desc="AWS Calculated">
// Each wheel represents a different case
if (Mode == 1) //AWS Calculated Angles
{
    switch (unit) {
        // Unit: 1=FL, 2=FR, 3=RR, 4=RL
        // Equations from Selekwa CDC Paper

        //===== Front Left =====
    case 1:
        if (df1 == 0) {
            wheelangle = 0.0;
            wheelspeed = Vg1;
        } else {
            wheelangle = (atanf(1.0 / ((1.0 / tanf(df1))+(W / (2.0 * H))*(1.0
/ tanf(df1))*(tanf(df1) - tanf(dr1)))));
            wheelspeed = ((Vg1 * (tanf(df1) / sin(wheelangle))) / (sqrt(1.0 +
0.25 * (pow((tanf(df1) + tanf(dr1)), 2.0))))));
        }
        break;

        //===== Front Right =====
    case 2:
        if (df1 == 0) {
            wheelangle = 0.0;
            wheelspeed = Vg1;
        } else {
            wheelangle = (atanf(1.0 / ((1.0 / tanf(df1))-(W / (2.0 * H))*(1.0
/ tanf(df1))*(tanf(df1) - tanf(dr1)))));
            wheelspeed = ((Vg1 * (tanf(df1) / sin(wheelangle))) / (sqrt(1.0 +
0.25 * (pow((tanf(df1) + tanf(dr1)), 2.0))))));
        }
        break;

        //===== Rear Right =====
    case 3:
        if (dr1 == 0) {
            wheelangle = 0.0;
            wheelspeed = Vg1;
        } else {
            wheelangle = (atanf(1.0 / ((1.0 / tanf(dr1))-(W / (2.0 * H))*(1.0
/ tanf(dr1))*(tanf(df1) - tanf(dr1)))));
            wheelspeed = ((Vg1 * (tanf(dr1) / sin(wheelangle))) / (sqrt(1.0 +
0.25 * (pow((tanf(df1) + tanf(dr1)), 2.0))))));
        }
        break;

        //===== Rear Left =====
    case 4:
        if (dr1 == 0) {
            wheelangle = 0.0;
            wheelspeed = Vg1;
        }
    }
}

```

```

        } else {
            wheelangle = (atanf(1.0 / ((1.0 / tanf(dr1))+(W / (2.0 * H))*(1.0
/ tanf(dr1))*(tanf(df1) - tanf(dr1)))));
            wheelspeed = ((Vg1 * (tanf(dr1) / sin(wheelangle))) / (sqrt(1.0 +
0.25 * (pow((tanf(df1) + tanf(dr1)), 2.0)))));
        }
        break;

        //===== Default =====
default:
    wheelangle = 0.0;
    wheelspeed = 0.0;
    break;
    }
}
//</editor-fold>
//<editor-fold defaultstate="collapsed" desc="AWS Mirror">
if (Mode == 2)//AWS Mirrored Angles
{
    switch (unit)
    {
        // Unit: 1=FL, 2=FR, 3=RR, 4=RL
        // Equations from Selekwa CDC Paper

        //===== Front Left =====
    case 1:
        wheelangle = df1;
        wheelspeed = Vg1;
        break;

        //===== Front Right =====
    case 2:
        wheelangle = df1;
        wheelspeed = Vg1;
        break;

        //===== Rear Right =====
    case 3:
        wheelangle = dr1;
        wheelspeed = Vg1;
        break;

        //===== Rear Left =====
    case 4:
        wheelangle = dr1;
        wheelspeed = Vg1;
        break;

        //===== Default =====
    default:
        wheelangle = 0.0;
        wheelspeed = 0.0;
    }
}

```

```

        break;
    }
}
//</editor-fold>
//<editor-fold defaultstate="collapsed" desc="FWS Calculated">
if (Mode == 3)//FWS Calculated Angles
{
    switch (unit) {
        dr1 = 0;
        // Unit: 1=FL, 2=FR, 3=RR, 4=RL
        // Equations from Selekwa CDC Paper

        //===== Front Left =====
    case 1:
        if (df1 == 0) {
            wheelangle = 0.0;
            wheelspeed = Vg1;
        } else {
            wheelangle = (atanf(1.0 / ((1.0 / tanf(df1))+ (W / (2.0 * H))*(1.0
/ tanf(df1))*(tanf(df1) - tanf(dr1)))));
            wheelspeed = ((Vg1 * (tanf(df1) / sin(wheelangle))) / (sqrt(1.0 +
0.25 * (pow((tanf(df1) + tanf(dr1)), 2.0))))));
        }
        break;

        //===== Front Right =====
    case 2:
        if (df1 == 0) {
            wheelangle = 0.0;
            wheelspeed = Vg1;
        } else {
            wheelangle = (atanf(1.0 / ((1.0 / tanf(df1))- (W / (2.0 * H))*(1.0
/ tanf(df1))*(tanf(df1) - tanf(dr1)))));
            wheelspeed = ((Vg1 * (tanf(df1) / sin(wheelangle))) / (sqrt(1.0 +
0.25 * (pow((tanf(df1) + tanf(dr1)), 2.0))))));
        }
        break;

        //===== Rear Right =====
    case 3:
        wheelangle = 0.0;
        wheelspeed = Vg1;
        break;

        //===== Rear Left =====
    case 4:
        wheelangle = 0.0;
        wheelspeed = Vg1;
        break;

        //===== Default =====
    default:

```

```

        wheelangle = 0.0;
        wheelspeed = 0.0;
        break;
    }
}
//</editor-fold>
//<editor-fold defaultstate="collapsed" desc="FWS Same">
if (Mode == 4)//FWS Same Angles
{
    switch (unit) {
        dr1 = 0;
        // Unit: 1=FL, 2=FR, 3=RR, 4=RL
        // Equations from Selekwa CDC Paper

        //===== Front Left =====
    case 1:
        wheelangle = df1;
        wheelspeed = Vg1;
        break;

        //===== Front Right =====
    case 2:
        wheelangle = df1;
        wheelspeed = Vg1;
        break;

        //===== Rear Right =====
    case 3:
        wheelangle = 0.0;
        wheelspeed = Vg1;
        break;

        //===== Rear Left =====
    case 4:
        wheelangle = 0.0;
        wheelspeed = Vg1;
        break;

        //===== Default =====
    default:
        wheelangle = 0.0;
        wheelspeed = 0.0;
        break;
    }
}
//</editor-fold>

steering_angle = wheelangle;
speed = wheelspeed;
}

//===== SPI Decoder =====
void Decode(unsigned int varstring)

```

```

{
    // varstring is a 16 bit string,
    // 1st 8 bits = address
    // 2nd 8 bits = value

    char address;
    signed char value;

    // 1st split string into two characters
    address = varstring >> 8;
    value = varstring;

    // Assign global variables

    //Steer Mode
    //0x01=AWS, Calculated Steer Angles
    //0x02=AWS, Mirrored Steer Angles
    //0x03=FWS, Calculated Steer Angles
    //0x04=FWS, Same Steer Angle
    if (address == 'M') {
        Mode = value;
    }

    //Remote Enable will not allow steer controller to move on until it has
    //reached its setpoint
    if (address == 'X')
    {
        RemoteEnable = value;
    }

    // Change Vg to percent, i.e. 0,33,66,100%
    if (address == 'V') {
        Vg = value;
    }

    //Convert df & dr to radians, max is pi/2
    if (address == 'F') {
        df = ((value * 1.0 / 128.0)*(3.14159 / 2.0));
    }
    if (address == 'R') {
        dr = ((value * 1.0 / 128.0)*(3.14159 / 2.0));
    }

    if(RemoteEnable==0)
    {
        if ((df != old_df) || (dr != old_dr))
        {
            breaksteer = 1;//Angles are new so break out of steering while loop}
        }
        old_df = df;
        old_dr = dr;
    }
}

```



```

//===== Wheel Motor Controller =====
void Speed_Controller(signed int x1)
{
    //Takes in a percentage (0-100 integer) and sets the wheel duty
    //Duty Range = 250-1850 | 250=full backwards; 1850=full forward; 1080=stop

    if (x1 != 0) //If speed is not zero calculate duty and turn off motor brake
    {
        unsigned int duty;
        duty = (((x1 * 1.0) / 100.0)*(MaxDuty)) + PWMZero);
        //duty=(((x1*1.0)/100.0)*(MaxDuty-1080))+1080);

        if (duty > (PWMZero + MaxDuty)) //Check for too high of duty
        {
            duty = (PWMZero + MaxDuty);
        }

        if (duty < (PWMZero - MaxDuty)) //Check for too low of duty
        {
            duty = (PWMZero - MaxDuty);
        }

        LATBbits.LATB9 = 1; // Motor brake off
        P1DC2 = duty; // Apply calculated duty

    } else //Otherwise turn on motor brake and duty to 0
    {
        LATBbits.LATB9 = 0; // Motor brake on
        P1DC2 = 0; // Duty = 0
    }
}

//===== Steer Motor Controller =====
void Steer_Controller(float y1)
{ // Takes in an angle (radians) loops steer motor until it is
  // within the error window
  // Direction may need to be reversed

  int pot_reading = 0;
  int des_steer;
  signed int steer_error = 30;
  // y1=desired steering angle in radians
  // convert to pot "points" using the following conversion
  // 242 = 90 deg CW , 787 = 90 deg CCW , 512 = Center
  // 3 bits/deg (approximately)

  des_steer = ((-1.0 * (y1 / (3.14159 / 2.0))*270) + 512); //Desired steer in bits

  pot_reading = (ain0Buff[0] + ain0Buff[1]) / 2; //Reads potentiometer
  steer_error = (pot_reading - des_steer);
}

```

```

if ((abs(steer_error)) > (PotWindow * 2))
{
    while (((abs(steer_error)) > (PotWindow)) && (breaksteer == 0))
    {
        pot_reading = (ain0Buff[0] + ain0Buff[1]) / 2; //Reads potentiometer
        steer_error = (pot_reading - des_steer); //Pot Reading - Desired Steer

        if (steer_error < 0) //Steer counterclockwise

            // Use a buffer zone of +10 to -10 so motor isn't constantly jerky
            {
                LATBbits.LATB6 = 0; // Steer motor brake off
                LATBbits.LATB8 = 1; // Steer motor on
                LATAbits.LATA4 = 0; // Direction = counterclockwise
            }
        else if (steer_error > 0) //Steer clockwise
        {
            LATBbits.LATB6 = 0; // Steer motor brake off
            LATBbits.LATB8 = 1; // Steer motor on
            LATAbits.LATA4 = 1; // Direction = clockwise
        }
        // Motor is within error window so turn it off
        LATBbits.LATB6 = 1; // Steer motor brake on
        LATBbits.LATB8 = 0; // Steer motor off
        LATAbits.LATA4 = 0; // Direction = counterclockwise
    }
    breaksteer = 0; //reset break flag for steer controller
}

//===== Timer 2 Interrupt =====
void __attribute__((interrupt, no_auto_psv)) _T2Interrupt(void)
{ //If module loses communication with central controller, command to zero
  //steer and zero speed.
  if (ErrorFlag != 0) {
      Speed_Controller(0);
      Steer_Controller(0);
  }

  ErrorFlag = 1; // Set Error Flag to get reset before this triggers
  IFS0bits.T2IF = 0; // Clear Timer5 Interrupt Flag
}

//===== SPI Interrupt =====
void __attribute__((interrupt, no_auto_psv)) _SPI1Interrupt(void)
{ // SPI Interrupt service routine
  ErrorFlag = 0; //Communication Established, No error
  int read = 0;
  read = SPI1BUF;
  Decode(read); //Send 16 bit string to be decoded into speed, df, and dr
  IFS0bits.SPI1IF = 0; // Clear the SPI1 Interrupt Flag
}

```

```

/*=====
                                     Main
=====*/
int main(void)
{
    int temp;

    initOSC(); // Initialize Oscillator to 40 MHz
    initRemappablePins(); // Intialize Remappable Peripheral Pins
    initSPI1(); // Intialize SPI module
    initADC1(); // Initialize ADC module
    initPWM1(); // Intialize PWM moudel

    temp = SPI1BUF; // Read SPI Buffer to clear it
    Speed_Controller(0);
    Steer_Controller(0);
    initTmr2(); // Start 1 Hz error check

    while (1)
    {
        Calc_Speed_n_Angle(Vg, df, dr);

        Steer_Controller(steering_angle);

        if (speed != old_speed)
        {
            Speed_Controller(speed);
            old_speed = speed;
        }
    }
    return (0);
}

```

## E.2. WCU\_DRIVERS.C

```
//~~~~~ WCU_Drivers.c ~~~~~

//<editor-fold defaultstate="collapsed" desc="Program Heading">
/*****
 * 2005 Microchip Technology Inc.
 *
 * FileName:      WCU_Drivers.c
 * Dependencies:  Header (.h) files if applicable, see below
 * Processor:     dsPIC33FJ128MC802
 * Compiler:      MPLAB C30 v3.00 or higher
 * Module:        Wheel Control Unit
 *
 * REVISION HISTORY:
 *~~~~~
 * Author          Date      Comments on this revision
 *~~~~~
 * Jon Nistler     05/26/11  Revision based on Nikhil Gupta's Code
 * Jon Nistler     06/27/11  Switched Everything to MPLabX and aggregated
 *                      all drivers into one
 *~~~~~
 *
 * ADDITIONAL NOTES: This file is for setup of peripheral modules on WCU
 *****/
//</editor-fold>

#include "p33FJ128MC802.h"
#include "WCU_Drivers.h"

#define SAMP_BUFF_SIZE 2 // Size of the input buffer per analog input
#define NUM_CHS2SCAN 2 // Number of channels enabled for channel scan

//===== Oscillator =====

void initOSC(void) {
    /* Configure Oscillator to operate the device at 40Mhz
       Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
       Fosc= 20M*40/(2*4)=80Mhz for 8M input clock */
    PLLFBD = 30; // M=32 p. 144 in datasheet
    CLKDIVbits.PLLPOST = 0; // N1=2
    CLKDIVbits.PLLPRE = 2; // N2=4

    // clock switch to incorporate PLL, builtin functions p. 168 in C30 guide
    _builtin_write_OSCCONH(0x03); // Initiate Clock Switch to Primary
    // Oscillator with PLL (NOSC=0b011)
    _builtin_write_OSCCONL(0x01); // Start clock switching
    while (OSCCONbits.COSC != 0b011) {
    }; // Wait for Clock switch to occur
    while (OSCCONbits.LOCK != 1) {
    }; // Wait for PLL to lock
}

//===== Initialize Remappable Pins =====
```

```

void initRemappablePins(void) {

    //REMAPPABLE PINS CONFIGURATION
    __builtin_write_OSCCONL(OSCCON & ~(1 << 6)); //Unlock the registers
    //Configure SPI1 Port for SLAVE mode
    AD1PCFGL = 0x00FF; //analogue pins configured as digital IO
    RPINR20 = 0x0002; //SCK1 Input is associated to pin RP00 (bit 8-12) (pin 4 of
dsPIC)
    //SDI1 input is associated to pin RP02 (bit 0-4) (pin 6 of dsPIC)
    // RPOR0=0x0700; //SDO1 output associated to pin RP01 (pin 5 of the
dsPIC)
    RPINR21 = 0x0003; //SS1 associated to RP03 (pin 7 of dsPIC)
    __builtin_write_OSCCONL(OSCCON | (1 << 6)); //Lock the registers

    TRISBbits.TRISB0 = 1; //pin RB0/RP0 is configured as input for clk
    TRISBbits.TRISB1 = 1; // pin RB1/RP1 is configured as an ON/OFF
    TRISBbits.TRISB2 = 1; //pin RB2/RP2 is configured as input for data (MOSI)
    TRISBbits.TRISB3 = 1; // SS1 input

    // SETTING TRISx REGISTER FOR OUTPUT
    TRISAbits.TRISA4 = 0;

    TRISBbits.TRISB4 = 0;
    TRISBbits.TRISB5 = 0;
    TRISBbits.TRISB6 = 0;
    TRISBbits.TRISB7 = 0;
    TRISBbits.TRISB8 = 0;
    TRISBbits.TRISB9 = 0;
    TRISBbits.TRISB10 = 0;
    TRISBbits.TRISB11 = 0;
    TRISBbits.TRISB13 = 0;
    TRISBbits.TRISB14 = 0;

    //SETTING TRISx REIGISTERS FOR INPUT

    //Redundant, already done in RPInit
    TRISBbits.TRISB0 = 1; //B0, B2, and B3 used for SPI
    TRISBbits.TRISB2 = 1;
    TRISBbits.TRISB3 = 1;

    // INTIALIZING PORTx VALUE TO ZERO

    PORTAbits.RA4 = 0;

    PORTBbits.RB3 = 0;
    PORTBbits.RB4 = 0;
    PORTBbits.RB5 = 0;
    PORTBbits.RB6 = 1; // Turn on steer motor brake
    PORTBbits.RB7 = 0;
    PORTBbits.RB8 = 0;
    PORTBbits.RB9 = 0; // Turn on wheel motor brake

```

```

    PORTBbits.RB10 = 0;
    PORTBbits.RB13 = 0;
    PORTBbits.RB14 = 0;
}
//===== Initialize Serial Peripheral Interface =====

void initSPI1(void) {

    //SPI1 configuration (p. 227)
    IFS0bits.SPI1IF = 0; // Clear SPI1 interrupt flag
    IEC0bits.SPI1IE = 0; // Disable interrupt

    SPI1CON1bits.PPRE = 0b10; //Primary prescaler 4:1
    SPI1CON1bits.SPRE = 0b110; //Secondary prescaler 2:1, gives final frequency of 5
MHz
    SPI1CON1bits.MSTEN = 0; //1 = Master mode
    SPI1CON1bits.CKP = 0; //0 = Idle state for clock is a low level; active state is
a high level
    SPI1CON1bits.SSEN = 1; //1 = SS1 (Slave Select) Pin input enabled
    SPI1CON1bits.CKE = 1; //1 = Serial output data changes on transition from active
//clock state to Idle clock state (see bit 6)
    SPI1CON1bits.SMP = 0; //0 = Input data sampled at middle of data output time
    SPI1CON1bits.MODE16 = 1; //1 = Communication is word-wide (16 bits)
    SPI1CON1bits.DISSDO = 1; //1 = Disable SD01 Pin

    SPI1STATbits.SPIROV = 0; //make sure the overflow flag is cleared
    SPI1STATbits.SPIEN = 1; //enable SPI1 module

    IFS0bits.SPI1IF = 0; // make sure the SPI interrupt flag is cleared
    IPC2bits.SPI1IP = 4; // Interrupt priority 4 (high)
    IEC0bits.SPI1IE = 1; // Interrupt enabled
}
//===== Initialize Analog to Digital Converter =====

void initADC1(void) {
    // p. 275

    AD1CON1bits.AD12B = 0; // 10-bit ADC operation
    AD1CON1bits.FORM = 0; // Data Output Format: Integer
    AD1CON1bits.SSRC = 2; // Sample Clock Source: GP Timer starts conversion
    AD1CON1bits.ASAM = 1; // ADC Sample Control: Sampling begins immediately after
conversion

    AD1CON2bits.CSCNA = 1; // Scan Input Selections for CH0+ during Sample A bit
    AD1CON2bits.CHPS = 0; // Converts CH0

    AD1CON3bits.ADRC = 0; // ADC Clock is derived from Systems Clock
    AD1CON3bits.ADCS = 63; // ADC Conversion Clock Tad=Tcy*(ADCS+1)= (1/40M)*64 =
1.6us (625Khz)
    // ADC Conversion Time for 10-bit Tc=12*Tad = 19.2us

    AD1CON2bits.SMPI = (NUM_CHS2SCAN - 1); // 2 ADC Channel is scanned

```

```

//AD1CSSH/AD1CSSL: A/D Input Scan Selection Register

AD1CSSLbits.CSS0 = 1; // Enable AN0 for channel scan
AD1CSSLbits.CSS1 = 1; // Enable AN1 for channel scan

//AD1PCFGH/AD1PCFGL: Port Configuration Register
AD1PCFGLbits.PCFG0 = 0; // AN0 as Analog Input
AD1PCFGLbits.PCFG1 = 0; // AN1 as Analog Input

IFS0bits.AD1IF = 0; // Clear the A/D interrupt flag bit
IPC3bits.AD1IP = 2;
IEC0bits.AD1IE = 1; // Enable A/D interrupt
AD1CON1bits.ADON = 1; // Turn on the A/D converter

    initTmr3();
}
//===== Initialize Timer 2 (Steer_Controller) =====

void initTmr2() {
    //Timer 2 is setup to time-out every 0.5 ms (2 kHz Rate).

    //Timer 2 initialization
    T2CONbits.TON = 0; // Disable Timer
    T2CONbits.TCS = 0; // Select internal instruction cycle clock
    T2CONbits.TGATE = 0; // Disable Gated Timer mode
    T2CONbits.TCKPS = 0b11; // Select 1:256 Prescaler
    T2CONbits.T32 = 0; // Disable 32 bit

    TMR2 = 0x00; // Clear timer register

    //Period = 40000000/(256*IMU_sample_rate)
    PR2 = 0xFFFF; // Load the period value (maximum)

    IPC1bits.T2IP = 3; //Interrupt priority 3
    IFS0bits.T2IF = 0; // Clear Timer5 Interrupt Flag
    IEC0bits.T2IE = 1; // Enable Timer5 interrupt

    T2CONbits.TON = 1; // Start Timer
}
//===== Initialize Timer 3 (ADC) =====

void initTmr3() {
    /*Timer 3 is setup to time-out every 125 microseconds (8Khz Rate). As a result,
    * the module will stop sampling and trigger a conversion on every Timer3
    * time-out, i.e., Ts=125us.*/

    TMR3 = 0x0000;
    PR3 = 4999;
    IFS0bits.T3IF = 0;
    IEC0bits.T3IE = 0;

    //Start Timer 3

```

```

    T3CONbits.TON = 1;
}
//=== Initialize Pulse Width Modulation (Tractive DC Motor) ==

void initPWM1(void) {
    P1TCONbits.PTMOD = 0b00; // Free running mode
    P1TCONbits.PTCKPS = 0b00; // Input clock period= 1 Tcy
    P1TCONbits.PTOPS = 0b00; // Output post scale is 1:1
    P1TCONbits.PTSIDL = 0; // Runs in idle mode

    P1TPER = 999; // Pwm frequency of 40 Khz

    PWM1CON1bits.PMOD1 = 0; // Pwm1 pair 1 is in independent mode **Actually in
complementary mode
    PWM1CON1bits.PMOD2 = 0; // Pwm1 pair 2 is in independent mode **Actually in
complementary mode

    PWM1CON1 = 0; // Disable all PWM Pins
    PWM1CON1bits.PEN2H = 1; // Enable Pwm1H2 pin is set acitve
    PWM1CON1bits.PEN1L = 1; // Enable Pwm1L1 pin is set acitve

    PWM1CON2bits.IUE = 1; // Immediate updates for PWM

    P1DTCN1bits.DTAPS = 0b00; // Dead time for unit A is Tcy
    P1DTCN1bits.DTBPS = 0b00; // Dead time for unit B is Tcy
    P1DTCN1bits.DTA = 0; // Dead time for unit A is '0'
    P1DTCN1bits.DTB = 20; // Dead time for unit B is '20'

    P1DTCN2bits.DTS1A = 0; // Dead time for the PWM1L1 active signal is provided by
unit A
    P1DTCN2bits.DTS1I = 0; // Dead time for the PWM1L1 inactive signal is provided
by unit A
    P1DTCN2bits.DTS2A = 1; // Dead time for the PWM1H2 active signal is provided by
unit B
    P1DTCN2bits.DTS2I = 0; // Dead time for the PWM1H2 inactive signal is provided
by unit A

    P1OVDCONbits.POVD1L = 1; // Output on PWM1L1 is controlled by the PWM generator
    P1OVDCONbits.POVD2H = 1; // Output on PWM1H2 is controlled by the PWM generator
    P1DC1 = 1400; // Duty cycle of PWM1L1 (2%) ??
    P1DC2 = 1023; // Duty cycle of PWM1H2 (1.5%) ??

    PWM2CON1 = 0;
    P1TCONbits.PTEN = 1; // PWM module is ON

    switch (unit) {
        // Unit: 1=FL, 2=FR, 3=RR, 4=RL

        case 1:
        {
            PWMZero = 1085;
        }
    }
}

```



```

        break;

    case 2:
    {
        PWMZero = 1000; //962
    }
    break;

    case 3:
    {
        PWMZero = 1077;
    }
    break;

    case 4:
    {
        PWMZero = 1013;
    }
    break;
}
}
//===== ADC Interrupt Service Routine =====

int ain0Buff[SAMP_BUFF_SIZE];
int ain1Buff[SAMP_BUFF_SIZE];
int scanCounter = 0;
int sampleCounter = 0;

void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt(void) {

    switch (scanCounter) { // If this is the first scan put value in buffer 0
    case 0:
        ain0Buff[sampleCounter] = ADC1BUF0;
        break;
        // On second scan put value in buffer 1
    case 1:
        ain1Buff[sampleCounter] = ADC1BUF0;
        break;

    default:
        break;

    }

    scanCounter++; // Increments scan counter

    if (scanCounter == NUM_CHS2SCAN) //Checks to see if we have reached desired
samples
    {
        scanCounter = 0; // Reset scan counter
        sampleCounter++;
    }
}

```

```
    if (sampleCounter == SAMP_BUFF_SIZE) //Checks to see if we have reached desired
samples
        sampleCounter = 0;

    IFS0bits.AD1IF = 0; // Clear the ADC1 Interrupt Flag
}
```

### E.3. WCU\_DRIVERS.H

```
/*
 * 2005 Microchip Technology Inc.
 *
 * FileName:          VCU_Drivers.h
 * Dependencies:      Other (.h) files if applicable, see below
 * Processor:         dsPIC33FJ128MC802
 * Compiler:          MPLAB C30 v3.00 or higher
 *
 *
 * REVISION HISTORY:
 * ~~~~~~
 * Author           Date       Comments on this revision
 * ~~~~~~
 * Jonathan Nistler 10/12/11   Release used during experimentation
 * ~~~~~~*
 * ADDITIONAL NOTES:
 *
 * ~~~~~~*/
#ifndef __VCU_DRIVERS_H__
#define __VCU_DRIVERS_H__

//definitions
#define Accel 0xA6 //Write address for acclerometer and gyro on IMU
#define Gyro  0xD0

#define X_Accel 0x32 //Read registers for various axes
#define Y_Accel 0x34
#define Z_Accel 0x36
#define X_Gyro 0x1D
#define Y_Gyro 0x1F
#define Z_Gyro 0x21

//Enable axes by changing these to 1
#define Enable_X_Accel 0
#define Enable_Y_Accel 0
#define Enable_Z_Accel 0
#define Enable_X_Gyro 0
#define Enable_Y_Gyro 0
#define Enable_Z_Gyro 1

extern void delay32(unsigned long cycles);
// External Functions
extern void initOSC(void);
extern void initRemappablePins(void);
extern void initSPI1(void);
extern void initSPI2(void);
extern void initI2C(void);
extern void initUART1(void);
```

```

extern void initADC1(void);
extern void initIMU(void);
extern void initCMP(void);
extern void initTmr2();
extern void initTmr3();
extern void initTmr4();
extern void initTmr5();
extern void __attribute__((__interrupt__)) _ADC1Interrupt(void);

extern void StartI2C(void);
extern void StopI2C(void);
extern void RestartI2C(void);
extern void WriteI2C(unsigned char byte);
extern unsigned int ReadI2C(void);
extern void IdleI2C(void);
extern void ACKI2C(void);
extern void NACKI2C(void);
extern void IMUwrite(unsigned char control, unsigned char address, unsigned char
data);
extern int IMUread(unsigned char control, unsigned char address);
extern signed int AccelRead(unsigned char address);
extern signed int GyroRead(unsigned char address);
extern void __delay32(unsigned long cycles);

extern int ain0Buff[2];
extern int ain1Buff[2];

extern float X_Accel_Offset;
extern float Y_Accel_Offset;
extern float Z_Accel_Offset;
extern float X_Gyro_Offset;
extern float Y_Gyro_Offset;
extern float Z_Gyro_Offset;

#endif

```

#### E.4. WCU\_TRAPS.C

```
/*
*****
* 2005 Microchip Technology Inc.
*
* FileName:        traps.c
* Dependencies:    Header (.h) files if applicable, see below
* Processor:       dsPIC33FJ128MC802
* Compiler:        MPLAB C30 v3.00 or higher
*
*
* REVISION HISTORY:
* ~~~~~~
* Author          Date      Comments on this revision
* ~~~~~~
* NIKHIL G.       08/19/09   First release of source file
* ~~~~~~
*
* ADDITIONAL NOTES:
* 1. This file contains trap service routines (handlers) for hardware
*    exceptions generated by the dsPIC33F device.
* 2. All trap service routines in this file simply ensure that device
*    continuously executes code within the trap service routine. Users
*    may modify the basic framework provided here to suit to the needs
*    of their application.
*
*****/

#include "p33FJ128MC802.h"

void __attribute__((__interrupt__)) _OscillatorFail(void);
void __attribute__((__interrupt__)) _AddressError(void);
void __attribute__((__interrupt__)) _StackError(void);
void __attribute__((__interrupt__)) _MathError(void);
void __attribute__((__interrupt__)) _DMACError(void);

void __attribute__((__interrupt__)) _AltOscillatorFail(void);
void __attribute__((__interrupt__)) _AltAddressError(void);
void __attribute__((__interrupt__)) _AltStackError(void);
void __attribute__((__interrupt__)) _AltMathError(void);
void __attribute__((__interrupt__)) _AltDMACError(void);

/*
Primary Exception Vector handlers:
These routines are used if INTCON2bits.ALTIVT = 0.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/
```

```

void __attribute__((interrupt, no_auto_psv)) _OscillatorFail(void)
{
    INTCON1bits.OSCFAIL = 0;          //Clear the trap flag

    LATBbits.LATB9 = 0;              // Motor brake on
    P1DC2 = 0;                       // Duty = 0
    LATBbits.LATB6 = 1; // Steer motor brake on
    LATBbits.LATB8 = 0; // Steer motor off
    LATAbits.LATA4 = 0; // Direction = counterclockwise
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AddressError(void)
{
    INTCON1bits.ADDRERR = 0;         //Clear the trap flag
    LATBbits.LATB9 = 0;              // Motor brake on
    P1DC2 = 0;                       // Duty = 0
    LATBbits.LATB6 = 1; // Steer motor brake on
    LATBbits.LATB8 = 0; // Steer motor off
    LATAbits.LATA4 = 0; // Direction = counterclockwise
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _StackError(void)
{
    INTCON1bits.STKERR = 0;          //Clear the trap flag
    LATBbits.LATB9 = 0;              // Motor brake on
    P1DC2 = 0;                       // Duty = 0
    LATBbits.LATB6 = 1; // Steer motor brake on
    LATBbits.LATB8 = 0; // Steer motor off
    LATAbits.LATA4 = 0; // Direction = counterclockwise
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _MathError(void)
{
    INTCON1bits.MATHERR = 0;         //Clear the trap flag
    LATBbits.LATB9 = 0;              // Motor brake on
    P1DC2 = 0;                       // Duty = 0
    LATBbits.LATB6 = 1; // Steer motor brake on
    LATBbits.LATB8 = 0; // Steer motor off
    LATAbits.LATA4 = 0; // Direction = counterclockwise
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _DMACError(void)
{
    INTCON1bits.DMACERR = 0;         //Clear the trap flag
    LATBbits.LATB9 = 0;              // Motor brake on
    P1DC2 = 0;                       // Duty = 0
    LATBbits.LATB6 = 1; // Steer motor brake on
    LATBbits.LATB8 = 0; // Steer motor off
    LATAbits.LATA4 = 0; // Direction = counterclockwise
    while (1);
}

```

```
}
```

```
/*  
Alternate Exception Vector handlers:  
These routines are used if INTCON2bits.ALTIVT = 1.  
All trap service routines in this file simply ensure that device  
continuously executes code within the trap service routine. Users  
may modify the basic framework provided here to suit to the needs  
of their application.  
*/
```

```
void __attribute__((interrupt, no_auto_psv)) _AltOscillatorFail(void)  
{  
    INTCON1bits.OSCFAIL = 0;  
    LATBbits.LATB9 = 0; // Motor brake on  
    P1DC2 = 0; // Duty = 0  

```

```
void __attribute__((interrupt, no_auto_psv)) _AltAddressError(void)  
{  
    INTCON1bits.ADDRERR = 0;  
    LATBbits.LATB9 = 0; // Motor brake on  
    P1DC2 = 0; // Duty = 0  
    LATBbits.LATB6 = 1; // Steer motor brake on  
    LATBbits.LATB8 = 0; // Steer motor off  
    LATAbits.LATA4 = 0; // Direction = counterclockwise  
    while (1);  
}
```

```
void __attribute__((interrupt, no_auto_psv)) _AltStackError(void)  
{  
    INTCON1bits.STKERR = 0;  
    LATBbits.LATB9 = 0; // Motor brake on  
    P1DC2 = 0; // Duty = 0  
    LATBbits.LATB6 = 1; // Steer motor brake on  
    LATBbits.LATB8 = 0; // Steer motor off  
    LATAbits.LATA4 = 0; // Direction = counterclockwise  
    while (1);  
}
```

```
void __attribute__((interrupt, no_auto_psv)) _AltMathError(void)  
{  
    INTCON1bits.MATHERR = 0;  
    LATBbits.LATB9 = 0; // Motor brake on  
    P1DC2 = 0; // Duty = 0
```

```

        LATBbits.LATB6 = 1; // Steer motor brake on
        LATBbits.LATB8 = 0; // Steer motor off
        LATAbits.LATA4 = 0; // Direction = counterclockwise
        while (1);
    }

void __attribute__((interrupt, no_auto_psv)) _AltDMACError(void)
{
    INTCON1bits.DMACERR = 0; //Clear the trap flag
    LATBbits.LATB9 = 0; // Motor brake on
    P1DC2 = 0; // Duty = 0
    LATBbits.LATB6 = 1; // Steer motor brake on
    LATBbits.LATB8 = 0; // Steer motor off
    LATAbits.LATA4 = 0; // Direction = counterclockwise
    while (1);
}

```



## APPENDIX F. CENTRAL VEHICLE BODY CONTROLLER C CODE

### F.1. VCU\_MAIN.C

```
//~~~~~ VCU_Main.c ~~~~~

//<editor-fold defaultstate="collapsed" desc="Program Heading">
/*****
* 2005 Microchip Technology Inc.
*
* FileName:          VCU_Main.c
* Dependencies:     Header (.h) files if applicable, see below
* Processor:        dsPIC33FJ128MC802
* Compiler:         MPLAB C30 v3.00 or higher
* Module:           Vehicle body controller
*
*
* REVISION HISTORY:
*~~~~~
* Author          Date      Comments on this revision
*~~~~~
* Jon Nistler     5/23/11    Revision based on Nikhil Gupta's Code
*~~~~~
*
* ADDITIONAL NOTES:
* This code is tested on target board with dsPIC33FJ128MC802 controller
* The Processor starts with the Internal oscillator without PLL enabled
* and then the Clock is switched to PLL Mode.
*****/
//</editor-fold>

//<editor-fold defaultstate="collapsed" desc="uC Pin Configuration">
//
//          =====
//          uC Reset MCLR-|1      28|-AVdd
//          Remote Pot 1 - AN1-|2    27|-AVss
//          Remote Pot 2 - AN2-|3    26|-RB15 - Sonar Sensor 4 Slave Select
//          Serial Clock - SPI1-|4   25|-RB14 - Sonar Sensor 3 Slave Select
//Odometer Comparator C2IN+ -|5     24|-RB13 - Sonar Sensor 2 Slave Select
//          Serial Data Out - SPI1-|6  23|-RB12 - Sonar Sensor 1 Slave Select
//          Slave Select - SPI1-|7   22|-UART1 - Rx Sonar Sensor Data
//          Ground - Vss-|8         21|-
//          OscillatorPin1 - OSC1-|9  20|-Vcap - Capacitor??
//          OscillatorPin2 - OSC2-|10 19|-Vss - Ground (Accel Gnd)
//          Ground - RB4-|11        18|-SDA1 - Serial Data Accel/Gyro (I2C)
//          +3.3v - RA4-|12         17|-SCL1 - Serial Clock Accel/Gyro (I2C)
//          +3.3v - Vdd-|13         16|-
//          -|14                    15|-
//          =====
//</editor-fold>
```

```

    //<editor-fold defaultstate="collapsed" desc="Definitions and Libraries">
// Includes
#include "p33FJ128MC802.h"
#include "VCU_Drivers.h"
#include "math.h"
#include <stdio.h>
#include <stdlib.h>

//Definitions
#define Sonar1 LATBbits.LATB12
#define Sonar2 LATBbits.LATB13
#define Sonar3 LATBbits.LATB14
#define Sonar4 LATBbits.LATB15

#define NumberofSonars 4

#define ON 1
#define OFF 0

#define linear_units 981.0 //9.81=meters, 981.0=cm, etc
#define angular_units 57.3 //57.3= 180/pi =radians, 1.0=degrees

// Builtin functions
_FGS(GWRP_OFF & GCP_OFF);
_FOSCESEL(FNOSC_FRC); //Fast RC Oscillator
_FOSC(FCKSM_CSECMD & IOL1WAY_OFF & OSCIOFNC_OFF & POSCMD_HS);
_FWDT(FWDTEN_OFF);
_FPOR(PWMPIN_OFF & HPOL_ON & LPOL_ON)

    //</editor-fold>

    //<editor-fold defaultstate="collapsed" desc="Global Variables">
//===== Global Variables =====
int ADCcount=0; //Track initialization of ADC

signed int Vg=0; //percent -100 to 100
signed int df=0; //integer -127 to 127 = -pi/2 to pi/2
signed int dr=0; //integer -127 to 127
signed int Mode=1;
signed int RemoteEnable=1;

int CurrentSensor=1; //Keeps track of which sensor is being sampled
int UARTdatacount=1; //To track which character the UART is sending

int Range1=255;
int Range2=255;
int Range3=255;
int Range4=255;
int max;

float X_Accel_Offset;
float Y_Accel_Offset;

```

```

float Z_Accel_Offset;
float X_Gyro_Offset;
float Y_Gyro_Offset;
float Z_Gyro_Offset;

int IMUcount2=0;

int X_Accel_Sum=0;
int Y_Accel_Sum=0;
int Z_Accel_Sum=0;
int X_Gyro_Sum=0;
int Y_Gyro_Sum=0;
int Z_Gyro_Sum=0;

float X_Accel_Avg=0;
float Y_Accel_Avg=0;
float Z_Accel_Avg=0;
float X_Gyro_Avg=0;
float Y_Gyro_Avg=0;
float Z_Gyro_Avg=0;

float X_Acceleration = 0; //cm/s^2
float X_Speed = 0; //cm/s
float X_Position = 0; //cm
float Y_Acceleration = 0; //cm/s^2
float Y_Speed = 0; //cm/s
float Y_Position = 0; //cm
float Z_Acceleration = 0; //cm/s^2
float Z_Speed = 0; //cm/s
float Z_Position = 0; //cm

float X_Angular_Speed = 0; //cm/s^2
float X_Angular_Position = 0; //cm/s
float Y_Angular_Speed = 0; //cm/s^2
float Y_Angular_Position = 0; //cm/s
float Z_Angular_Speed = 0; //cm/s^2
float Z_Angular_Position = 0; //cm/s

float Time=0;
long double distance = 0.1; //cm

signed int headingint=0; //Store heading in same units as df,dr
float heading = 0; //radians
float old_heading = 0;

signed int FWAngOTF[50]={0}; //Used to store front wheel angle on the fly
int OTFindex = 1;
//</editor-fold>

//**** Put Paths Here ****
// Example:
#define pathlength 1000
const signed char dfmem[pathlength]={0};

```

```

const signed char drmem[pathlength]={0};
//****

//===== Encode =====
int Encode(char address,signed int value)
{
int varstring;
varstring=(address<<8)|(0x00FF&value); //Address 8MSB, Value 8LSB
return (varstring);
}

//===== Remote =====
void Remote(void)
{
//Samples ADC on two channels and converts those to
//signals for speed and turning

int joystick1;
int joystick2;

joystick1 = (ain0Buff[0]+ain0Buff[1])/2; //Grab average of
joystick2 = (ain1Buff[0]+ain1Buff[1])/2; //two values from ADC

df=((joystick1*1.0-512.0)/512.0)*90);
dr = (-1 * df); //AWS

//Speed Control
Vg =((joystick2*1.0-512.0)/512.0)*70);

if(abs(Vg)<5)
{Vg=0;}

}

//===== Range Pulse =====
void RangePulse(void)
{

// Sends a pulse on portB to the sonar range finders
// must be held high for at least 20 us. Range finders
// are pulsed in order

// When data comes back, the interrupt drives UARTData function to
//capture the value and index CurrentSensor

switch (CurrentSensor) {
// Sensor 1 - Far Left
case 1:
Sonar1 = ON;
break;
// Sensor 2 - Near Left
case 2:
Sonar2 = ON;
break;
// Sensor 3 - Near Right

```

```

        case 3:
            Sonar3 = ON;
            break;
            // Sensor 4 - Far Right
        case 4:
            Sonar4 = ON;
            break;
    }
    __delay32(2000); // Delay
    Sonar1 = OFF;
    Sonar2 = OFF;
    Sonar3 = OFF;
    Sonar4 = OFF;
}
//===== ASCII to Number =====
int ascii2num(int ascii)
{
    // Takes in a character in ASCII format and converts
    // it to a number
    char num;

    switch (ascii) {
        case '0':num = 0;
            break;
        case '1':num = 1;
            break;
        case '2':num = 2;
            break;
        case '3':num = 3;
            break;
        case '4':num = 4;
            break;
        case '5':num = 5;
            break;
        case '6':num = 6;
            break;
        case '7':num = 7;
            break;
        case '8':num = 8;
            break;
        case '9':num = 9;
            break;
        default: num = 0;
            break;
    }
    return num;
}
//===== Range Data =====
void RangeData(int RangeDistance2)
{
    //Upon transmission of 5th character from sonar sensor
    //RangeData assigns the distance to one of four vars
    //representing the distance of that sonar sensor

```

```

switch (CurrentSensor)
{
    // 1st Sensor - Far Left
    case 1:
        Range1 = RangeDistance2;
        break;
    // 2nd Sensor - Near Left
    case 2:
        Range2 = RangeDistance2;
        break;

    // 3rd Sensor - Near Right
    case 3:
        Range3 = RangeDistance2;
        break;
    // 4th Sensor - Far Right
    case 4:
        Range4 = RangeDistance2;
        break;
}
}
//===== UART Data =====
void UARTData(int Character)
{
    //Takes in a character sent from the UART interrupt
    //writes to five vars, when last var is written,
    //the 2nd 3rd and 4th are converted to a distance

    //Data comes in from the sensor in the form of
    // 9600 baud, 8 bit, no parity, 1 stop bit
    // 'R' '#' '#' '#' 'Carraige Return' (ASCII 13)
    // where ### is the distance in inches

    int Data1;
    int Data2;
    int Data3;
    int Data4;
    int Data5;
    int RangeDistance;

    switch (UARTdatacount)
    {
        // 1st Data - Should be ascii 'R'
        case 1:
            if(Character=='R')
            {
                Data1 = Character;
            }
            else
            {
                UARTdatacount=0; //If synchronization gets off keep resetting
                                //until next range reading comes in
            }
        }
    }

```

```

    }
    break;

    // 2nd Data - Should be ascii '#'
case 2:
    Data2 = Character;
    break;

    // 3rd Data - Should be ascii '#'
case 3:
    Data3 = Character;
    break;

    // 4th Data - Should be ascii '#'
case 4:
    Data4 = Character;
    break;

    // 5th Data - Should be ascii 'Carriage Return' (13)
case 5:
    Data5 = Character;
    UARTdatacount = 0;
    if (Data5 == 0x000D)
    {
        RangeDistance =
((ascii2num(Data2))*100)+((ascii2num(Data3))*10)+((ascii2num(Data4))*1);
        RangeData(RangeDistance);
        CurrentSensor=CurrentSensor+1; //Data is good, clear to go on to next
sensor
        Data1=0;
        Data2=0;
        Data3=0;
        Data4=0;
        Data5=0;
        if (CurrentSensor==(NumberOfSonars+1)) //Reset CurrentSensor when
overflow
            {CurrentSensor=1;}
    }
    break;
}
UARTdatacount = UARTdatacount + 1;
}
//===== IMU Measurement =====
void SampleIMU(void)
{
    // Every 20ms Timer2 expires and calls SampleIMU
    // SampleIMU samples the specified axes of the acclerometer
    // and gyroscope. It then updates the values for X,Y, and Z
    // acceleration and angular velocity. It then integrates to find
    // velocity, position, and orientation.

    // Enable or disable axes in VCU_Drivers.h

```

```

//====Accelerometer====
// X-Axis
if(Enable_X_Accel)
{
    X_Acceleration = ((AccelRead(X_Accel)-
X_Accel_Offset)*linear_units/256.0); //cm/s^2
    X_Speed = X_Speed+X_Acceleration*0.02; //cm/s
    X_Position = X_Position+X_Speed*0.02; //cm
}

// Y-Axis
if(Enable_Y_Accel)
{
    Y_Acceleration = (AccelRead(Y_Accel)-Y_Accel_Offset)*linear_units/256.0;
//cm/s^2
    Y_Speed = Y_Speed+Y_Acceleration*0.02; //cm/s
    Y_Position = Y_Position+Y_Speed*0.02; //cm
}

// Z-Axis
if(Enable_Z_Accel)
{
    Z_Acceleration = (AccelRead(Z_Accel)-Z_Accel_Offset)*linear_units/256.0;
//cm/s^2
    Z_Speed = Z_Speed+Z_Acceleration*0.02; //cm/s
    Z_Position = Z_Position+Z_Speed*0.02; //cm
}

//====Gyroscope====
// X-Axis
if(Enable_X_Gyro)
{
    X_Angular_Speed = (GyroRead(X_Gyro)-
X_Gyro_Offset)*1.0/(14.375*angular_units); //rad/s
    X_Angular_Position = X_Angular_Position+X_Angular_Speed*0.02; //rads
}

// Y-Axis
if(Enable_Y_Gyro)
{
    Y_Angular_Speed = (GyroRead(Y_Gyro)-
Y_Gyro_Offset)*1.0/(14.375*angular_units); //rad/s
    Y_Angular_Position = Y_Angular_Position+Y_Angular_Speed*0.02; //rads
}

// Z-Axis
if(Enable_Z_Gyro)
{
    Z_Angular_Speed = (GyroRead(Z_Gyro)-
Z_Gyro_Offset)*1.0/(14.375*angular_units); //rad/s
    Z_Angular_Position =
fmodf((Z_Angular_Position+Z_Angular_Speed*0.02),6.283); //rads
}

```



```

}
//===== Follow Target =====
void Follow(void)
{
    max = 300;
    int sens = 1;

    if (Range1 < max)
    {
        sens = 1;
        max = Range1;
    }
    if (Range2 < max)
    {
        sens = 2;
        max = Range2;
    }

    if (Range3 < max)
    {
        sens = 3;
        max = Range3;
    }
    if (Range4 < max)
    {
        sens = 4;
        max = Range4;
    }

/*
    switch (sens) {
        case 1:
            df = -20;
            dr = 20;
            break;
        case 2:
            df = 20;
            dr = -20;
            break;
        case 3:
            df = 10;
            dr = -10;
            break;
        case 4:
            df = -20;
            dr = 20;
            break;
    }
*/

    if(max>18)
    {Vg=15;}

```

```

        else
        {
            if(max<12)
            {Vg=-15;}
            else
            {Vg = 0;}
        }
df=0;
dr=0;
}
//===== Obstacle Avoidance =====
void Avoid(void)
{
    df=(((Range1*1.0-Range4*1.0)/(Range1*1.0+Range4*1.0))*100;

    if(df<-70)
    {df=-70;}
    if(df>70)
    {df=70;}
}
//===== On The Fly path memory function =====
void OTFPathMem()
{
    float dfF;
    float drF;
    float A;
    float B;
    float C;
    float l;
    float m;
    float theta;
    int turn;

    int OTFindexr;
    //Driven by comparator(odometer) interrupt

    if((df==0&&dr==0)||((df==dr))
    {theta=0;}
    else
    {//Convert df, dr to radians

dfF=fabs(df*1.571/128.0);
drF=fabs(dr*1.571/128.0);

//Figure out geometry

//First Check Signs
//Same Sign

if(((df>0)&&(dr>0))||((df<0)&&(dr<0)))
{

```

```

    if(dfF>drF)//df>dr
    {
        A=1.571-dfF;
        B=1.571+drF;
        turn=1;//Same sign as df
    }
    else//dr>df
    {
        A=1.571+dfF;
        B=1.571-drF;
        turn=-1;//Opposite sign as df
    }
}
else //Opposite sign
{
A=1.571-dfF;
B=1.571-drF;
turn=1;//Same sign as df
}

if(df<0)
{turn=-1*turn;}

C=3.1416-B-A;
m=fabs(58.42*sinf(B)/sinf(C));
l=sqrtf(fabs(853.22+powf(m,2)-(58.42*m*cosf(A))));

//Change in heading
theta=atanf(1.3744/l);
}

heading=old_heading+theta*turn;

/*
if(heading>3.14)
{heading=3.14;}
if(heading<-3.14)
{heading=-3.14;}
*/

old_heading=heading;

headingint = (signed int)(floorf((heading*127.0/1.5707)));

//Store absolute angle at front wheels
FWAngOTF[OTFindex]=(df+headingint);//127 or 254
//FWAngOTF[OTFindex]=0;

OTFindexr=OTFindex+1;
if (OTFindexr==44)
{OTFindexr=1;}

dr=FWAngOTF[OTFindexr]-headingint;// the path

```

```

    OTFindex=OTFindex+1;
    if(OTFindex == 44)
    {OTFindex=1;}

    if(dr<-90)
    {dr=-90;}
    if(dr>90)
    {dr=90;}
}
//===== Path Programming =====
void Path()
{
    //Driven by comparator(odometer) interrupt

    int pathindexint;

    pathindexint=floor(distance);

    df=-1*dfmem[pathindexint]; //assign df and dr according to distance along
    dr=-1*drmem[pathindexint]; // the path

    //dr=0;
    //dr=-1*df;
}
//===== SPI Transmit =====
void SPITransmit(void)
{
    //Transmits these values every 50ms, driven by Timer4
    int TRANSMIT;

    //Speed
    TRANSMIT = Encode('V',Vg); //Vg, -100 to 100
    SPI1BUF = TRANSMIT;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(1000);

    //Front Angle
    TRANSMIT = Encode('F',df); //df -127 to 127
    SPI1BUF = TRANSMIT;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(1000);

    //Rear Angle
    TRANSMIT = Encode('R',dr); //dr -127 to 127
    SPI1BUF = TRANSMIT;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(1000);
}
//===== UART Interrupt (Sonar) =====
void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{

```



```

int main(void)
{
    ////////////////Steer Mode////////////////////
        Mode=1;           //
    //1=AWS, Calculated Steer Angles //
    //2=AWS, Mirrored Steer Angles  //
    //3=FWS, Calculated Steer Angles //
    //4=FWS, Same Steer Angle       //
    ///////////////////////////////////////////////////
    ///////////////////////////////////////////////////
    ///////////////////////////////////////////////////
    //Remote/////////////////////////
    //Uses stabilizing while loop///
        RemoteEnable=0;    //
    //When disabled will make the vehicle
    //more responsive but noisier
    //(got rid of this on wheel unit program
    // but haven't yet reprogrammed them
    ///////////////////////////////////////////////////

    //<editor-fold defaultstate="collapsed" desc="Initialization">
    // Initialize local main variables
    int TRANSMIT2;
    int i; //counting variables
    int j;

    initOSC();           //Initialize Oscillator
    initRemappablePins(); //Initialize Remappable Pins
    initSPI1();          //Initialize SPI1

    //First pass
    TRANSMIT2 = SPI1BUF; // Read SPI Buffer to clear it

    Vg=0;           // Reinitialize variables
    df=0;
    dr=0;
    SPITransmit();

    // wait for some time
    for (i = 0; i < 10000; i++) {
        for (j = 0; j < 3000; j++) {
        }
    }

    TRANSMIT2 = Encode('M',Mode); //Enter Steer Mode Here
    SPI1BUF = TRANSMIT2;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(2000);

    // Won't be needed once WCU's are reprogrammed
    TRANSMIT2 = Encode('X',RemoteEnable); // Stabilizing while loop
    SPI1BUF = TRANSMIT2;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(2000);

```

```

initTmr4();          //Initialize Timer 4 (begins SPI transmission)

//===== Odometer
//initCMP();        //Initialize Comparator (Odometer)

//===== Sonar Sensors
//initUART1();      //Initialize UART1 (Sonar)
//__delay32(40000000); //Delay for a second so all sonars stop
//initTmr5();       //Initialize Timer 5 (begins ranging sonar)

// wait for some time
for (i = 0; i < 10000; i++) {
    for (j = 0; j < 5000; j++) {
    }
}

//===== Remote Control
initADC1();         //Initialize AtoD Converter, used by remote control

//===== Inertial Measurement Unit
//initI2C();        //Initialize I2C, used by Accelerometer (IMU)
//initIMU();        //Initialize Inertial Measurement Unit (Accel & Gyro)
//initTmr2();       //Initialize Timer 2 (begin inertial measurement)

//</editor-fold>

if (0)
//<editor-fold defaultstate="collapsed" desc="Track G zigzag path">
{
    //Straight
    Vg=0;
    df=0;
    dr=0;
    __delay32(40000000);
    distance=0;
    while(distance<123.6)
    {Vg=15;}
    //Turn Left
    Vg=0;
    df=-118;
    dr=118;
    __delay32(80000000);
    Z_Angular_Speed = 0; //rad/s
    Z_Angular_Position = 0; //rads
    while(Z_Angular_Position<0.52359)
    {Vg=1;}
    //Straight
    Vg=0;
    df=0;
    dr=0;
    __delay32(80000000);
    distance=0;
}

```

```

while(distance<200.0)
{Vg=15;}
//Turn Right
Vg=0;
df=118;
dr=-118;
__delay32(80000000);
Z_Angular_Speed = 0; //rad/s
Z_Angular_Position = 0; //rads
while(Z_Angular_Position>-1.0472)
{Vg=1;}
//Straight
Vg=0;
df=0;
dr=0;
__delay32(80000000);
distance=0;
while(distance<400.0)
{Vg=15;}
//Turn Left
Vg=0;
df=-118;
dr=118;
__delay32(80000000);
Z_Angular_Speed = 0; //rad/s
Z_Angular_Position = 0; //rads
while(Z_Angular_Position<1.0472)
{Vg=1;}
//Straight
Vg=0;
df=0;
dr=0;
distance=0;
__delay32(80000000);
while(distance<200.0)
{Vg=15;}
//Turn Right
Vg=0;
df=118;
dr=-118;
__delay32(80000000);
Z_Angular_Speed = 0; //rad/s
Z_Angular_Position = 0; //rads
while(Z_Angular_Position>-0.52359)
{Vg=1;}
//Straight
Vg=0;
df=0;
dr=0;
__delay32(80000000);
distance=0;
while(distance<123.6)
{Vg=15;}

```



```

    Vg=0;
    while(1){}
}
//</editor-fold>

while (1)
//<editor-fold defaultstate="collapsed" desc="Remote Control Loop">
//Remote Control Function
{
    Remote();

    for (i=1;i<20;i++)
    {j=1;
    j=2;}

}
//</editor-fold>

while (0)
//<editor-fold defaultstate="collapsed" desc="Experiment 1">
//Experiment 1 - Use preprogrammed path to determine df and dr
{
    if (distance<(pathlength-1))
    {Vg=15;}
    else
    {
    Vg=0;
    df=0;
    dr=0;
    }
}
//</editor-fold>

while (0)
//<editor-fold defaultstate="collapsed" desc="Experiment 2">
//Experiment 2 - Use on board sensors to determine df, odometer dr, and Vg
{
    Avoid();

/*
    if((Range2>15)&&(Range3>15))
    {Vg=5;}
    else
    {Vg = 0;}
*/

    //dr=-1*df;

    Vg=5;

    __delay32(10000000);

```

```
}  
    //</editor-fold>  
  
    //<editor-fold defaultstate="collapsed" desc="Catchall">  
    // Too Far!!!  
    // Catchall  
    while(1)  
    {  
        Vg=0;  
        df=0;  
        dr=0;  
    }  
    //</editor-fold>  
  
    return(0);  
}
```

## F.2. VCU\_DRIVERS.C

```

/*****
* 2005 Microchip Technology Inc.
*
* FileName:      VCU_Drivers.c
* Dependencies:  Header (.h) files if applicable, see below
* Processor:     dsPIC33FJ128MC802
* Compiler:      MPLAB  C30 v3.00 or higher
*
*
* REVISION HISTORY:
* ~~~~~
* Author      Date      Comments on this revision
* ~~~~~
* Jon Nistler  05/26/11  Revision based on Nikhil Gupta's Code
* ~~~~~
*
* ADDITIONAL NOTES: This file is for setup of ADC module.
*****/
#include "p33FJ128MC802.h"
#include "VCU_Drivers.h"

#define SAMP_BUFF_SIZE      2      // Size of the input buffer per analog input
#define NUM_CHS2SCAN        2      // Number of channels enabled for channel
scan

//int Xzero[2000];
//===== Oscillator Initialization =====
//<editor-fold defaultstate="collapsed" desc="Oscillator Setup">
void initOSC(void)
{
    /* Configure Oscillator to operate the device at 40Mhz
       Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
       Fosc= 20M*32/(2*4)=80Mhz for 20M input clock */
    PLLFBD = 30; // M=32 p. 144 in datasheet
    CLKDIVbits.PLLPOST = 0; // N1=2
    CLKDIVbits.PLLPRE = 2; // N2=4
    // clock switch to incorporate PLL, builtin functions p. 168 in C30 guide
    __builtin_write_OSCCONH(0x03); // Initiate Clock Switch to Primary
    // Oscillator with PLL (NOSC=0b011)
    __builtin_write_OSCCONL(0x01); // Start clock switching
    while (OSCCONbits.COSC != 0b011) {
    }; // Wait for Clock switch to occur
    while (OSCCONbits.LOCK != 1) {
    }; // Wait for PLL to lock
}

//</editor-fold>
//===== Remappable Pin Initialization =====
//<editor-fold defaultstate="collapsed" desc="Remappable Pins">
void initRemappablePins(void)
{
    // Data direction for pins

```

```

AD1PCFGLbits.PCFG0 = 0; // AN0 as Analog Input
AD1PCFGLbits.PCFG1 = 0; // AN1 as Analog Input
AD1PCFGLbits.PCFG2 = 1; // Digital Input
AD1PCFGLbits.PCFG3 = 0; // AN3 as Analog Input
AD1PCFGLbits.PCFG4 = 1; // Digital Input
AD1PCFGLbits.PCFG5 = 1; // Digital Input

PWM1CON1 = 0x0000;
PWM2CON1 = 0x0000;

////////// Remappable Pin Config //////////
__builtin_write_OSCCONL(OSCCON & 0xbf); //clear the bit 6 of OSCCONL to unlock
Pin Re-map
// Configure SPI1 Port for MASTER mode (p. 162-166 datasheet)
RPINR20bits.SDI1R = 0b11111; // SDI1 input is associated to Vss
RPOR0bits.RP0R = 0b01000; //remappable pin RP00 (pin 4 of the dsPIC) is
associated to SCK
RPOR1bits.RP2R = 0b00111; //RP02=SDO1 are output (p.164 & 184)
RPOR1bits.RP3R = 0b01001; //RP03=SS (Slave Select)

// I2C Module, pins are automatically configured when module is enabled
// Pin 18, RP9 is SDA1 for data
// Pin 17, RP8 is SCL1 for clocking

// Configure UART Module
RPINR18 = 0x000B; // UART Rx is associated to pin 22 RP11

__builtin_write_OSCCONL(OSCCON | 0x40); //Locks IOLOCK in OSCCONL register

// Inputs
TRISBbits.TRISB1 = 1;

// Outputs

TRISBbits.TRISB0 = 0; // pin RB0/RP0 (pin 4) for SClock1
TRISBbits.TRISB2 = 0; // pin RB2/RP2 (pin 6) for data (MOSI)
TRISBbits.TRISB3 = 0; // pin RB3/RP3 (pin 7) for SS1 (slave select)

TRISBbits.TRISB11 = 0; // pin RB11/RP11 (pin 22) UART Rx
TRISBbits.TRISB12 = 0; // pin RB12/RP12 (pin 23) Sonar Sensor 1 Tx
TRISBbits.TRISB13 = 0; // pin RB13/RP13 (pin 24) Sonar Sensor 2 Tx
TRISBbits.TRISB14 = 0; // pin RB14/RP14 (pin 25) Sonar Sensor 3 Tx
TRISBbits.TRISB15 = 0; // pin RB15/RP15 (pin 22) Sonar Sensor 4 Tx
LATB=0;
}
//</editor-fold>
//===== Comparator Initialization =====
//<editor-fold defaultstate="collapsed" desc="Comparator Setup">
void initCMP(void)
{
    //CMCONbits.C1NEG = 1; //Connected to V+

```

```

//CMCONbits.C1POS = 0; // Internal Vref

CMCONbits.C2NEG = 1; //Connected to V+
CMCONbits.C2POS = 0; // Internal Vref

CVRCONbits.CVRSS =0; // Device Supply
CVRCONbits.CVR = 7; //VRef = 1.5v

CVRCONbits.CVREN =1; //Internal Voltage On
//CMCONbits.C1EN = 1; //Enable Comparator
CMCONbits.C2EN = 1; //Enable Comparator

CMCONbits.C1EVT= 0; //Clear Comparator 1 Event
CMCONbits.C2EVT= 0; //Clear Comparator 2 Event

IPC4bits.CMIP = 5; //High Priority 5
IFS1bits.CMIF =0; //Clear Flag
IEC1bits.CMIE =1; //Enable Interrupt
}
//</editor-fold>
//===== Serial Peripheral Interface Initialization =====
//<editor-fold defaultstate="collapsed" desc="SPI1 Setup">
void initSPI1(void)
{
    //SPI1 configuration (p. 227)
    IFS0bits.SPI1IF = 0; // Clear SPI1 interrupt flag
    IEC0bits.SPI1IE = 0; // Disable interrupt

    SPI1CON1bits.PPRE = 0b10; //Primary prescaler 4:1
    SPI1CON1bits.SPRE = 0b110; //Secondary prescaler 2:1, gives final frequency of 5
MHz
    SPI1CON1bits.MSTEN = 1; //1 = Master mode
    SPI1CON1bits.CKP = 0; //0 = Idle state for clock is a low level; active state is
a high level
    SPI1CON1bits.SSEN = 1; //0 = SSx pin not used by module. Pin controlled by port
function.
    SPI1CON1bits.CKE = 1; //1 = Serial output data changes on transition from active
clock state to Idle clock state (see bit 6)
    SPI1CON1bits.SMP = 0; //0 = Input data sampled at middle of data output time
    SPI1CON1bits.MODE16 = 1; //1 = Communication is word-wide (16 bits)

    SPI1STATbits.SPIROV = 0; //make sure the overflow flag is cleared
    SPI1STATbits.SPIEN = 1; //enable SPI1 module
}
//</editor-fold>
//===== Inertial Measurement Unit Initialization =====
//<editor-fold defaultstate="collapsed" desc="IMU Setup">
void initIMU(void)
{
    int IMUcount=0;
    int IMUtotal=0;

    //See ADXL345 and ITG3200 data sheets for settings

```

```

IMUwrite(Accel,0x2C,0b00001001); //Data output 50 Hz, Bandwidth 25 Hz
IMUwrite(Accel,0x31,0b00000000); //Disable Self Test, Full Resolution,
//Justify, and Range = 2g
IMUwrite(Accel,0x38,0b00011111); //Disable FIFO, FIFO buffer=32
IMUwrite(Accel,0x2D,0b00001000); //Enable measurement

IMUwrite(Gyro,0x15,0b00000000);
IMUwrite(Gyro,0x16,0b00011101); //Full Scale Range, Low Pass Filter 10Hz
IMUwrite(Gyro,0x17,0b00000000); //Disable all interrupts on chip
IMUwrite(Gyro,0x3E,0b00000001); //Normal mode, X Gyro clock source

//Need to zero all enabled axes
// Enable or disable axes in VCU_Drivers.h

//====Accelerometer====
// X-Axis
if(Enable_X_Accel)
{
    for(IMUcount=0;IMUcount<1000;IMUcount++)
    {
        //Xzero[IMUcount]=AccelRead(X_Accel);
        //IMUtotal=IMUtotal+Xzero[IMUcount];

        IMUtotal=IMUtotal+AccelRead(X_Accel);
        __delay32(800000);
    }
    X_Accel_Offset=IMUtotal*1.0/1000.0;
    IMUtotal=0;
}

// Y-Axis
if(Enable_Y_Accel)
{
    for(IMUcount=0;IMUcount<50;IMUcount++)
    {
        IMUtotal=IMUtotal+AccelRead(Y_Accel);
        __delay32(400000);
    }
    Y_Accel_Offset=IMUtotal*1.0/50.0;
    IMUtotal=0;
}

// Z-Axis
if(Enable_Z_Accel)
{
    for(IMUcount=0;IMUcount<50;IMUcount++)
    {
        IMUtotal=IMUtotal+AccelRead(Z_Accel);
        __delay32(400000);
    }
    Z_Accel_Offset=IMUtotal*1.0/50.0;
    IMUtotal=0;
}

```

```

//====Gyroscope====
// X-Axis
if(Enable_X_Gyro)
{
    for(IMUcount=0;IMUcount<50;IMUcount++)
    {
        IMUtotal=IMUtotal+GyroRead(X_Gyro);
        __delay32(400000);
    }
    X_Gyro_Offset=IMUtotal*1.0/50.0;
    IMUtotal=0;
}

// Y-Axis
if(Enable_Y_Gyro)
{
    for(IMUcount=0;IMUcount<50;IMUcount++)
    {
        IMUtotal=IMUtotal+GyroRead(Y_Gyro);
        __delay32(400000);
    }
    Y_Gyro_Offset=IMUtotal*1.0/50.0;
    IMUtotal=0;
}

// Z-Axis
if(Enable_Z_Gyro)
{
    for(IMUcount=0;IMUcount<100;IMUcount++)
    {
        IMUtotal=IMUtotal+GyroRead(Z_Gyro);
        __delay32(800000);
    }
    Z_Gyro_Offset=IMUtotal*1.0/100.0;
    IMUtotal=0;
}

}
//</editor-fold>
//===== Inter-Integrated Circuit Initialization =====
//<editor-fold defaultstate="collapsed" desc="I2C Setup">
void initI2C(void)
{
    //I2C configuration
    I2C1BRG=395;//95 = 400k
    I2C1CON=0x1000; // Use all default values for I2C module
    I2C1RCV = 0x0000;
    I2C1TRN = 0x0000;
    I2C1CONbits.I2CEN=1; //Enable module
}
//</editor-fold>
//===== Inter-Integrated Circuit Commands =====

```

```

    //<editor-fold defaultstate="collapsed" desc="I2C Commands">
void StartI2C(void)
{
    //This function generates an I2C start condition

    I2C1CONbits.SEN = 1;        //Generate Start Condition
    while (I2C1CONbits.SEN);    //Wait for Start Condition
}

void StopI2C(void)
{
    //This function generates an I2C stop condition

    I2C1CONbits.PEN = 1;        //Generate Stop Condition
    while (I2C1CONbits.PEN);    //Wait for Stop
}

void RestartI2C(void)
{
    //This function generates an I2C Restart condition

    I2C1CONbits.RSEN = 1;        //Generate Restart
    while (I2C1CONbits.RSEN);    //Wait for restart
}

void WriteI2C(unsigned char byte)
{
    //This function transmits the byte passed to the function
    //while (I2C1STATbits.TRSTAT); //Wait for bus to be idle
    I2C1TRN = byte;              //Load byte to I2C1 Transmit buffer
    while (I2C1STATbits.TBF);    //wait for data transmission
}

unsigned int ReadI2C(void)
{
    I2C1CON=(I2C1CON & 0xFF20);
    I2C1CONbits.RCEN = 1;        //Enable Master receive
    while(!I2C1STATbits.RBF);    //Wait for receive buffer to be full
    return(I2C1RCV);             //Return data in buffer
}

void IdleI2C(void)
{
    while (I2C1STATbits.TRSTAT); //Wait for bus Idle
}

void ACKI2C(void)
{
    // Generates an Acknowledge.
    I2C1CONbits.ACKDT = 0;        //Set for ACK
    I2C1CONbits.ACKEN = 1;
    while(I2C1CONbits.ACKEN);    //wait for ACK to complete
}

```



```

}

void NACKI2C(void)
{
    //Generates a NO Acknowledge on the Bus
    I2C1CONbits.ACKDT = 1;        //Set for NotACK
    I2C1CONbits.ACKEN = 1;
    while(I2C1CONbits.ACKEN);    //wait for ACK to complete
    I2C1CONbits.ACKDT = 0;        //Set for NotACK
}

void IMUwrite(unsigned char control, unsigned char address, unsigned char data)
{
    IdleI2C();        //wait for bus Idle
    StartI2C();      //Generate Start Condition
    WriteI2C(control); //Write Control Byte (A6,A7,D0,or D1)
    IdleI2C();      //wait for ACK
    WriteI2C(address); //Write register address
    IdleI2C();      //wait for ACK
    WriteI2C(data); //write data
    IdleI2C();      //wait for ACK
    StopI2C();      //Generate Stop Condition
}

int IMUread(unsigned char control, unsigned char address)
{
    int I2Cdata;

    IdleI2C();        //wait for bus Idle
    StartI2C();      //Generate Start Condition
    WriteI2C(control); //Write Control Byte
    IdleI2C();      //wait for ACK
    WriteI2C(address); //Write start address
    IdleI2C();      //wait for bus Idle

    RestartI2C();    //Generate restart condition
    WriteI2C((control+1)); //Write control byte for read
    IdleI2C();      //wait for bus Idle

    I2Cdata=ReadI2C(); //read data from I2C buffer
    NACKI2C();        //Send Not ACK
    StopI2C();        //Generate Stop

    return(I2Cdata); //return Data
}

signed int AccelRead(unsigned char address)
{
    //Reads 2 consecutive registers from IMU (Axis High, Axis Low) and
    //concatenates them

    unsigned int Acceldata2a;
    unsigned int Acceldata2b;
    signed int Acceldata;
}

```

```

    IdleI2C();        //wait for bus Idle
    StartI2C();       //Generate Start Condition
    WriteI2C(0xA6);  //Write Control Byte
    IdleI2C();        //wait for ACK
    WriteI2C(address); //Write start address
    IdleI2C();        //wait for bus Idle

    RestartI2C();     //Generate restart condition
    WriteI2C((0xA7)); //Write control byte for read
    IdleI2C();        //wait for bus Idle

    Acceldata2a=ReadI2C(); //read data (LSB) from I2C buffer
    ACKI2C();           //Send ACK
    Acceldata2b=ReadI2C(); //read data (MSB) from I2C buffer
    NACKI2C();          //Send Not ACK
    StopI2C();         //Generate Stop

    //Concatenate data
    Acceldata2b=Acceldata2b<<8;
    Acceldata=Acceldata2b+Acceldata2a;

    return(Acceldata); //return Data
}

signed int GyroRead(unsigned char address)
{
    //Reads 2 consecutive registers from IMU (Axis Low, Axis High) and
    //concatenates them
    //Reverse of gyro, high/low

    unsigned int Gyrodata2a;
    unsigned int Gyrodata2b;
    signed int Gyrodata;

    IdleI2C();        //wait for bus Idle
    StartI2C();       //Generate Start Condition
    WriteI2C(0xD0);  //Write Control Byte
    IdleI2C();        //wait for ACK
    WriteI2C(address); //Write start address
    IdleI2C();        //wait for bus Idle

    RestartI2C();     //Generate restart condition
    WriteI2C((0xD1)); //Write control byte for read
    IdleI2C();        //wait for bus Idle

    Gyrodata2a=ReadI2C(); //read data (LSB) from I2C buffer
    ACKI2C();           //Send ACK
    Gyrodata2b=ReadI2C(); //read data (MSB) from I2C buffer
    NACKI2C();          //Send Not ACK
    StopI2C();         //Generate Stop

    //Concatenate data
    Gyrodata2a=Gyrodata2a<<8;
    Gyrodata=Gyrodata2b+Gyrodata2a;
}

```

```

        return(Gyrodata);          //return Data
    }
    //</editor-fold>
//== Universal Asynchronous Receiver/Transmitter Initialization ==
    //<editor-fold defaultstate="collapsed" desc="UART1 Setup">
void initUART1(void)
{
    //UART1 configuration
    U1MODEbits.STSEL = 0; // 1-stop bit
    U1MODEbits.PDSEL = 0; // No Parity, 8-data bits
    U1MODEbits.BRGH = 0; //Low Speed
    U1MODEbits.URXINV = 1; //Idle state is low
    U1MODEbits.ABAUD = 0; // Auto-Baud Disabled
    U1MODEbits.UEN = 0; //Disable CTS and RTS
    U1MODEbits.USIDL = 0; //Continue in idle mode

    U1BRG = 259; // BAUD Rate Setting for 9600

    U1STAbits.URXISEL = 0b00; //Interrupt when buffer has only 1 char
    U1STAbits.UTXEN = 0; // Disable UART Tx

    IPC2bits.U1RXIP = 0b010; // Interrupt priority 2
    IFS0bits.U1RXIF = 0; // clear RX interrupt flag
    IEC0bits.U1RXIE = 1; // Enable UART Rx interrupt
    IEC0bits.U1TXIE = 0; // Disable UART Tx interrupt

    U1MODEbits.UARTEN = 1; // Enable UART
}
    //</editor-fold>
//===== Analog to Digital Initialization =====
    //<editor-fold defaultstate="collapsed" desc="ADC Setup">
void initADC1(void)
{
    // p. 275

    AD1CON1bits.AD12B = 0; // 10-bit ADC operation
    AD1CON1bits.FORM = 0; // Data Output Format: Integer
    AD1CON1bits.SSRC = 2; // Sample Clock Source: GP Timer starts conversion
    AD1CON1bits.ASAM = 1; // ADC Sample Control: Sampling begins immediately after
conversion

    AD1CON2bits.CSCNA = 1; // Scan Input Selections for CH0+ during Sample A bit
    AD1CON2bits.CHPS = 0; // Converts CH0

    AD1CON3bits.ADRC = 0; // ADC Clock is derived from Systems Clock
    AD1CON3bits.ADCS = 63; // ADC Conversion Clock Tad=Tcy*(ADCS+1)= (1/40M)*64 =
1.6us (625Khz)
        // ADC Conversion Time for 10-bit Tc=12*Tad = 19.2us

    AD1CON2bits.SMPI = (NUM_CHS2SCAN-1); // 2 ADC Channel is scanned

    //AD1CSSH/AD1CSSL: A/D Input Scan Selection Register

```

```

AD1CSSLbits.CSS0=1; // Enable AN0 for channel scan
AD1CSSLbits.CSS1=1; // Enable AN1 for channel scan

//AD1PCFGH/AD1PCFGL: Port Configuration Register
AD1PCFGLbits.PCFG0 = 0; // AN0 as Analog Input
AD1PCFGLbits.PCFG1 = 0; // AN1 as Analog Input

IPC3bits.AD1IP = 1; //Interrupt Priority 1
IFS0bits.AD1IF = 0; // Clear the A/D interrupt flag bit
IEC0bits.AD1IE = 1; // Enable A/D interrupt

AD1CON1bits.ADON = 1; // Turn on the A/D converter
}
//</editor-fold>
//===== Timer2 (Accelerometer) Initialization =====
//<editor-fold defaultstate="collapsed" desc="Timer2 Setup">
void initTmr2()
{
//Timer 2 is setup to time-out every 0.5 ms (2 kHz Rate).

//Timer 2 initialization
T2CONbits.TON = 0; // Disable Timer
T2CONbits.TCS = 0; // Select internal instruction cycle clock
T2CONbits.TGATE = 0; // Disable Gated Timer mode
T2CONbits.TCKPS = 0b11; // Select 1:256 Prescaler
T2CONbits.T32 = 0; // Disable 32 bit

TMR2 = 0x00; // Clear timer register

//Period = 4000000/(256*IMU_sample_rate)
PR2 = 3125; // Load the period value

IPC1bits.T2IP = 3; //Interrupt priority 3
IFS0bits.T2IF = 0; // Clear Timer5 Interrupt Flag
IEC0bits.T2IE = 1; // Enable Timer5 interrupt

T2CONbits.TON = 1; // Start Timer
}
//</editor-fold>
//===== Timer3 (ADC) Initialization =====
//<editor-fold defaultstate="collapsed" desc="Timer3 Setup">
void initTmr3()
{
/*Timer 3 is setup to time-out every 125 microseconds (8Khz Rate). As a result, the
module
will stop sampling and trigger a conversion on every Timer3 time-out, i.e.,
Ts=125us.*/

TMR3 = 0x0000;
PR3 = 4999;
IFS0bits.T3IF = 0;

```

```

    IEC0bits.T3IE = 0;

    //Start Timer 3
    T3CONbits.TON = 1;
}
//</editor-fold>
//===== Timer4 (SPI) Initialization =====
//<editor-fold defaultstate="collapsed" desc="Timer4 Setup">
void initTmr4()
{
    //Timer 4 is setup to time-out every 100 ms (10 Hz Rate).

    //Timer4 initialization
    T4CONbits.TON = 0; // Disable Timer
    T4CONbits.TCS = 0; // Select internal instruction cycle clock
    T4CONbits.TGATE = 0; // Disable Gated Timer mode
    T4CONbits.TCKPS = 0b11; // Select 1:256 Prescaler
    T4CONbits.T32 = 0; // Disable 32 bit

    TMR4 = 0x00; // Clear timer register

    PR4 = 15620; // Load the period value

    IPC6bits.T4IP = 4; // Set high interrupt priority of 4
    IFS1bits.T4IF = 0; // Clear Timer5 Interrupt Flag
    IEC1bits.T4IE = 1; // Enable Timer5 interrupt

    T4CONbits.TON = 1; // Start Timer
}
//</editor-fold>
//===== Timer5 (Sonar) Initialization =====
//<editor-fold defaultstate="collapsed" desc="Timer5 Setup">
void initTmr5()
{
    //Timer 5 is setup to time-out every 50 ms (20 Hz Rate).
    //This is near maximum because of the time delay between when the sensor
    //is pulsed and when it returns data ~0.045 s

    //Timer5 initialization
    T5CONbits.TON = 0; // Disable Timer
    T5CONbits.TCS = 0; // Select internal instruction cycle clock
    T5CONbits.TGATE = 0; // Disable Gated Timer mode
    T5CONbits.TCKPS = 0b10; // Select 1:64 Prescaler

    TMR5 = 0x00; // Clear timer register

    PR5 = 31250; // Load the period value

    IPC7bits.T5IP = 2; // Set interrupt priority of 2
    IFS1bits.T5IF = 0; // Clear Timer5 Interrupt Flag
    IEC1bits.T5IE = 1; // Enable Timer5 interrupt
}

```

```

    T5CONbits.TON = 1; // Start Timer
}
//</editor-fold>
//===== ADC Interrupt Service Routine =====
//<editor-fold defaultstate="collapsed" desc="ADC ISR">
// initialize variables
int ain0Buff[SAMP_BUFF_SIZE];
int ain1Buff[SAMP_BUFF_SIZE];
int scanCounter=0;
int sampleCounter=0;

//ISR
void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt(void)
{
    switch (scanCounter)
    {
    // If this is the first scan put value in buffer 0
    case 0:
        ain0Buff[sampleCounter]=ADC1BUF0;
        break;
    // On second scan put value in buffer 1
    case 1:
        ain1Buff[sampleCounter]=ADC1BUF0;
        break;

    default:
        break;
    }

    scanCounter++; // Increments scan counter

    if(scanCounter==NUM_CHS2SCAN) //Checks to see if we have reached desired samples
    {
        scanCounter=0; // Reset scan counter
        sampleCounter++;
    }

    if(sampleCounter==SAMP_BUFF_SIZE) //Checks to see if we have reached desired
samples
        sampleCounter=0;

    IFS0bits.AD1IF = 0; // Clear the ADC1 Interrupt Flag
}
//</editor-fold>

```

### F.3. VCU\_DRIVERS.H

```
/*
 * 2005 Microchip Technology Inc.
 *
 * FileName:          VCU_Drivers.h
 * Dependencies:      Other (.h) files if applicable, see below
 * Processor:         dsPIC33FJ128MC802
 * Compiler:          MPLAB C30 v3.00 or higher
 *
 *
 * REVISION HISTORY:
 * ~~~~~~
 * Author           Date       Comments on this revision
 * ~~~~~~
 * Jonathan Nistler 10/12/11    Last Release of this file
 * ~~~~~~*
 * ADDITIONAL NOTES:
 *
 * ~~~~~~*/
#ifndef __VCU_DRIVERS_H__
#define __VCU_DRIVERS_H__

//definitions
#define Accel 0xA6 //Write address for acclerometer and gyro on IMU
#define Gyro  0xD0

#define X_Accel 0x32 //Read registers for various axes
#define Y_Accel 0x34
#define Z_Accel 0x36
#define X_Gyro 0x1D
#define Y_Gyro 0x1F
#define Z_Gyro 0x21

//Enable axes by changing these to 1
#define Enable_X_Accel 0
#define Enable_Y_Accel 0
#define Enable_Z_Accel 0
#define Enable_X_Gyro 0
#define Enable_Y_Gyro 0
#define Enable_Z_Gyro 1

extern void delay32(unsigned long cycles);
// External Functions
extern void initOSC(void);
extern void initRemappablePins(void);
extern void initSPI1(void);
extern void initSPI2(void);
extern void initI2C(void);
extern void initUART1(void);
```

```

extern void initADC1(void);
extern void initIMU(void);
extern void initCMP(void);
extern void initTmr2();
extern void initTmr3();
extern void initTmr4();
extern void initTmr5();
extern void __attribute__((__interrupt__)) _ADC1Interrupt(void);

extern void StartI2C(void);
extern void StopI2C(void);
extern void RestartI2C(void);
extern void WriteI2C(unsigned char byte);
extern unsigned int ReadI2C(void);
extern void IdleI2C(void);
extern void ACKI2C(void);
extern void NACKI2C(void);
extern void IMUwrite(unsigned char control, unsigned char address, unsigned char
data);
extern int IMUread(unsigned char control, unsigned char address);
extern signed int AccelRead(unsigned char address);
extern signed int GyroRead(unsigned char address);
extern void __delay32(unsigned long cycles);

extern int ain0Buff[2];
extern int ain1Buff[2];

extern float X_Accel_Offset;
extern float Y_Accel_Offset;
extern float Z_Accel_Offset;
extern float X_Gyro_Offset;
extern float Y_Gyro_Offset;
extern float Z_Gyro_Offset;

#endif

```



#### F.4. VCU\_TRAPS.C

```

/*****
*  2005 Microchip Technology Inc.
*
* FileName:      traps.c
* Dependencies:  Header (.h) files if applicable, see below
* Processor:     dsPIC33FJ128MC802
* Compiler:      MPLAB  C30 v3.00 or higher
*
*
* REVISION HISTORY:
* ~~~~~
* Author          Date      Comments on this revision
* ~~~~~
* NIKHIL G.       08/19/09  First release of source file
* ~~~~~
*
* ADDITIONAL NOTES:
* 1. This file contains trap service routines (handlers) for hardware
*    exceptions generated by the dsPIC33F device.
* 2. All trap service routines in this file simply ensure that device
*    continuously executes code within the trap service routine. Users
*    may modify the basic framework provided here to suit to the needs
*    of their application.
*
*****/

#include "p33FJ128MC802.h"

void __attribute__((__interrupt__)) _OscillatorFail(void);
void __attribute__((__interrupt__)) _AddressError(void);
void __attribute__((__interrupt__)) _StackError(void);
void __attribute__((__interrupt__)) _MathError(void);
void __attribute__((__interrupt__)) _DMACError(void);

void __attribute__((__interrupt__)) _AltOscillatorFail(void);
void __attribute__((__interrupt__)) _AltAddressError(void);
void __attribute__((__interrupt__)) _AltStackError(void);
void __attribute__((__interrupt__)) _AltMathError(void);
void __attribute__((__interrupt__)) _AltDMACError(void);
void exit_SPI(void);
/*
Primary Exception Vector handlers:
These routines are used if INTCON2bits.ALTIVT = 0.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/

```

```

void __attribute__((interrupt, no_auto_psv)) _OscillatorFail(void)
{
    INTCON1bits.OSCFAIL = 0;        //Clear the trap flag
    exit_SPI();
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AddressError(void)
{
    INTCON1bits.ADDRERR = 0;        //Clear the trap flag
    exit_SPI();
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _StackError(void)
{
    INTCON1bits.STKERR = 0;        //Clear the trap flag
    exit_SPI();
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _MathError(void)
{
    INTCON1bits.MATHERR = 0;        //Clear the trap flag
    exit_SPI();
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _DMACError(void)
{
    INTCON1bits.DMACERR = 0;        //Clear the trap flag
    exit_SPI();
    while (1);
}

/*
Alternate Exception Vector handlers:
These routines are used if INTCON2bits.ALTIVT = 1.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/

void __attribute__((interrupt, no_auto_psv)) _AltOscillatorFail(void)
{
    INTCON1bits.OSCFAIL = 0;
    exit_SPI();
    while (1);
}

```

```

void __attribute__((interrupt, no_auto_psv)) _AltAddressError(void)
{
    INTCON1bits.ADDRERR = 0;
    exit_SPI();
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltStackError(void)
{
    INTCON1bits.STKERR = 0;
    exit_SPI();
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltMathError(void)
{
    INTCON1bits.MATHERR = 0;
    exit_SPI();
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltDMACError(void)
{
    INTCON1bits.DMACERR = 0;           //Clear the trap flag
    exit_SPI();
    while (1);
}

void exit_SPI(void)
{
    //While Stabilizing
    SPI1BUF = 0x5801;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(1000);

    //Speed
    SPI1BUF = 0x5600;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(1000);

    //Front Angle
    SPI1BUF = 0x461E;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(1000);

    //Rear Angle
    SPI1BUF = 0x52E2;
    while (SPI1STATbits.SPITBF); //Wait for buffer to empty
    __delay32(1000);
}

```