Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Development of a Control Systems Platform for an Autonomous Soft-Car

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan vid Linköpings universitet av

Pratish Ray

LiTH-ISY-EX--15/4908--SE
Linköping November 2015

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköping University
S-581 83 Linköping, Sweden

Linköpings tekniska högskola
Institutionen för systemteknik
581 83 Linköping

Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Development of a Control Systems Platform for an Autonomous Soft-Car

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan vid Linköpings universitet av

Pratish Ray

LiTH-ISY-EX--15/4908--SE
Linköping November 2015

Handledare: **Kristoffer Lundahl**
ISY, Linköpings universitet

Examinator: **Jan Åslund**
ISY, Linköpings universitet

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköping University
S-581 83 Linköping, Sweden

Linköpings tekniska högskola
Institutionen för systemteknik
581 83 Linköping

Linköping, 23 October, 2015

Presentationsdatum 2015-11-06 **Publiceringsdatum (elektronisk version)**	**Institution och avdelning** Institutionen för systemteknik, Fordonssystem Department of Electrical Engineering, Vehicular Systems

URL för elektronisk version
http://www.ep.liu.se

Publikationens titel
Development of a Control Systems Platform for an Autonomous Soft-Car

Författare
Pratish Ray

Sammanfattning

Balloon Cars (Soft Cars) used for active-system testing are presently statically following one specific path. Autonomous GPS driven balloon cars can be extremely useful if made to be accurate. Volvo group's long term plan involves usage of such cars to improve active-safety systems. This report presents comprehensive details about the development of the control system of the soft-car.

Software development was preceded by purchase of parts. Detailed descriptions of the balloon car hardware components are presented. Various hardware components were interconnected to form a CAN Bus network. This thesis work describes CAN networks in general and with reference to the developed Linux based software system.

The navigation system is presented in the thesis report using a 'bottom up' approach. First the Low-level functions and variables are described that directly control the hardware component. They are called by high level functions that are subsequently explained. The high level functions include accurate turning and an automated lane change process.

The high level functions may be sequentially called and to accurately follow a specific path. Various lines and points may be used in a user defined Cartesian coordinate system. Details about the performance of various high level functions as well as of a demonstration of a pattern are presented along with the limitations of the overall system.

Nyckelord
Autonomous, GPS, Control System, CAN Bus, Lane Change, Balloon Car

Abstract

Balloon Cars (Soft Cars) used for active-system testing are presently statically following one specific path. Autonomous GPS driven balloon cars can be extremely useful if made to be accurate. Volvo group’s long term plan involves usage of such cars to improve active-safety systems. This report presents comprehensive details about the development of the control system of the soft-car.

Software development was preceded by purchase of parts. Detailed descriptions of the balloon car hardware components are presented. Various hardware components were interconnected to form a CAN Bus network. This thesis work describes CAN networks in general and with reference to the developed Linux based software system.

The navigation system is presented in the thesis report using a ‘bottom up’ approach. First the Low-level functions and variables are described that directly control the hardware component. They are called by high level functions that are subsequently explained. The high level functions include accurate turning and an automated lane change process.

The high level functions may be sequentially called and to accurately follow a specific path. Various lines and points may be used in a user defined Cartesian coordinate system. Details about the performance of various high level functions as well as of a demonstration of a pattern are presented along with the limitations of the overall system.

Acknowledgments

I would like to thank my supervisor at LiU, Kristoffer Lundahl, and my supervisor at Volvo GTT, Magnus Ström for their help and support during this thesis. I would also like to thank my examiner, Jan Åslund, for making the thesis and its results possible. Special thanks to other employees at Volvo, Robert Laxing, Mike Bilette and Jean Philippe Abeillon for their help during the thesis. I would like to thank Maziar Shalbaf for mechanical support and other assistance during the master thesis.

Linköping, October 2015

Pratish Ray

Contents

1 Introduction 1

1.1 Problem Specification 1

1.2 Prior Development 2

1.2.1 Summary of Soft Car Components 2

1.3 Development Plan and Limitations 2

1.4 Thesis Outline 3

2 System Overview 4

2.1 Raspberry Pi 2 5

2.2 Peak CAN interface 6

2.3 Arduino Mega 2560 6

2.4 Linear Actuators 7

2.5 Motor 8

2.6 GPS System 8

2.6.1 Communication with System 9

3 CAN Bus 10

3.1 Architecture Fundamentals 10

3.1.1 CAN Abstraction Layers 11

3.1.2 CAN Physical Layer 12

3.2 CAN Messages 12

3.2.1 Idle 12

3.2.2 SOF (Start of Frame) 13

3.2.3 Identifier 13

3.2.4 DLC (Data Length Code) 13

3.2.5 Data 13

3.2.6 CRC (Cyclic Redundancy Check) 13

3.2.7 Acknowledgement 13

3.2.8 EOF (End of Frame) 13

3.3 Soft Car CAN Network 13

3.4 SocketCAN 14

3.4.1 Usage 16

3.4.2 Read/Write CAN Frame 17

4 User Control 18

4.1 Parallel Processing 18
4.1.1 Interaction between Cores 20
4.2 User Interface 20
4.2.1 Manual Control (Lower Functions) 21
4.2.2 GPS Correction Features 22
4.2.3 Higher Functions 22
4.2.4 Path Following 22
5 Control System 23
5.1 Steering (Potentiometer) 23
5.1.1 Limitation 25
5.2 Significant Variables 26
5.2.1 GPS Data Variables 26
5.2.2 CAN Data Variables 26
5.2.3 Coordinate System Variables 27
5.2.4 Potentiometer Value Variables 27
5.2.5 Tuning Parameters 27
5.2.6 Motor and Potentiometer Reference Variables 28
5.3 Correction Functions 28
5.3.1 GPS Corrected Velocity 28
5.3.2 Heading Correction 32
5.4 Higher Functions 38
5.4.1 Turning by Delta/Absolute Heading 38
5.5 Lane Change 40
5.5.1 Mechanism 41
5.6 Pattern 44
5.6.1 Construction of Pattern 44
5.6.2 Pattern for Demonstration 45
6 Results 47
7 Conclusion 52
7.1 Limitations and Further Development 52
8 References 54

1 Introduction

Active safety systems consist of technology designed to avoid collisions or reduce collision severity prior to an impact. Active safety system supports, assists or alerts the driver if a critical situation ahead is detected, for example Panic Brake Assist, Forward Collision Warning or Lane Keeping Support. When these systems are still in the development and verification state, a reliable tool is required in order to obtain adequate testing in order to validate them. The quest from the initiators at Volvo Group Truck Technologies (GTT) was to bring up a concept for how to sustainably and robustly build a reliable and light target car able to withstand lighter impacts and be remotely controlled.

Volvo GTT developed a Soft Car that is desired to be able to reduce the risks during testing, increase the efficiency of testing, be operable from a distance and be able to provide more repeatability to the testing scenarios. The soft car aims to be robust in order to forgive lighter impacts with the test truck during testing, electrically powered, and remotely controlled to be able to safely perform a test, reliable to not break down during normal testing scenarios and naturally manoeuvrable.

Prior to the development of GPS based soft cars, these collision and near-miss scenarios were tested using static ‘balloon cars’ most often travelling on rails. As accurate as it may be, these balloon cars could only drive on one particular path (that assigned by the rail track). Soft Car development is crucial if and only if the final result is both accurate and repeatable (Snider, 2009)**.**

1.1 Problem Specification

The objective of the master thesis was to build a controllable navigation system that utilised live GPS information to accurately enable the soft car to follow a predetermined path. The soft car was required to follow this path with an accuracy of four centimetres. It was also required that this navigation system could be remotely accessed and controlled for making live changes, data post-processing, tuning parameters in real-time etc. It was recommended to build the control system on a medium that was easily transferable since the primary goal for the company was to be able to utilise the technology in other systems.

Initially it was assumed that by the end of the master thesis, a full-prototype would be ready for usage. However, it was discovered and declared that there needed to be additions and substitutions in terms of hardware. Due to a time constraint because of purchase and installation of new parts, Volvo realised that a full prototype would not be possible and it was recommended to focus on making the soft car as accurate as possible while ensuring that the primary goal was the utilisation of the navigational system in a different medium in the future. The initial accuracy requirement (of 4 cm) was no longer a goal to meet.

The chronological set of objectives may be best described as below:

- Configuration of Motor and Steering Actuators: Following simple paths
- Purchase and Installation of new Hardware
- Mounting of Global Positioning System and Communication module
- Improvement of path following accuracy: Finding the root cause of errors
- Usage of GPS Data for building a robust control system

1.2 Prior Development

A separate student thesis (Larsson, et al., 2013) was devoted to the construction of the soft car that was capable of being driven using a remote control. This soft car had an Arduino microcontroller and along with a CAN shield, utilised remotely transmitted CAN signals in navigating the soft car. Although this soft car was mechanically functional, the control system was not sufficiently powerful and certain changes (described in the Raspberry Pi 2) had to be made before proceeding with the development of the control system. A brief summary of the components is shown in the following sub section below a CAD drawing of the soft car as seen in figure 1.

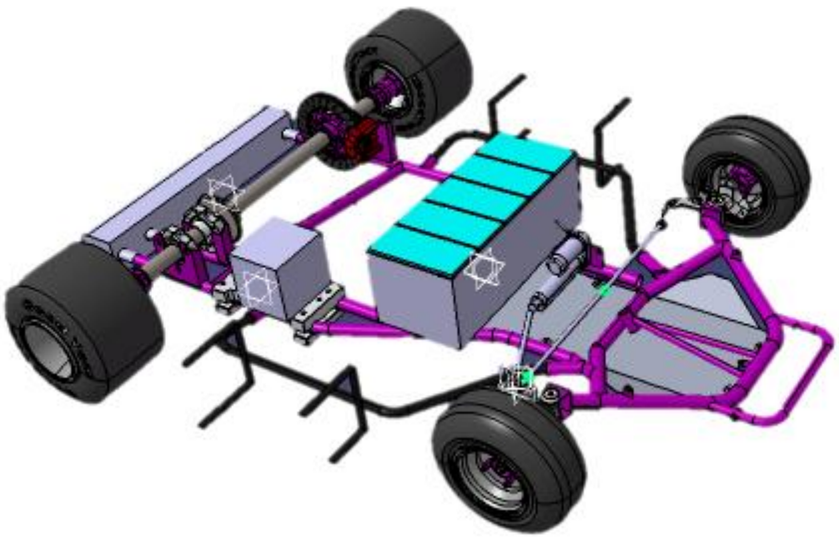


Figure 1: The concept soft car design (Larsson, et al., 2013).

1.2.1 Summary of Soft Car Components

1. **Motor:** A Brushless Direct Current motor (5 kW, 48 Volts) was used due to its low weight and power production per unit weight. The Motor is also weather resistant with IP65 classification.
2. **Chassis:** A Gillard TG14 Go-Kart Chassis was used for construction and contributing factors to the selection of the make were its geometry, weight and suitable size.
3. **Power Supply:** Lithium Ion ($LiFePO_4$) batteries were used to supply DC Current at 48 Volts. A Battery Management System (BMS) was also required in order to limit the charging and discharging of the batteries. A number of H Bridges were also required to lower the voltage to 24 Volts and 12 Volts.
4. **Steering:** Transmotec DLA series actuators (12 Volt DC) were used for steering purposes.
5. **Control System:** An Arduino mega2560 microcontroller was used in combination with CAN and motor shields to control the motor and linear actuators respectively.
6. **Remote Control:** A CAN based remote control was used to communicate with the Arduino through the CAN shield.

1.3 Development Plan and Limitations

In order to construct a fully functioning GPS driven navigation system, it was essential to increase the computational power of the system. The need for inter-communication and synchronisation between the components of the soft car was paramount. Thus, it was necessary to make structural changes to the soft car design before initiating the development of the autonomous soft car.

After due testing, it was also realised that the actuators used for braking were insufficiently powerful and also slow. Utilising the motor as the brake for testing purposes as well as having a separate emergency brake was considered to be a better idea.

It was also decided to abandon the remote control since it was targeted to have an SSH connection to the system at all times possessing all the control features thereby limiting the importance of the Remote Control.

Since the company also laid emphasis on the importance of ensuring that the technology was transferable, it was decided to build the system on a Linux environment which is becoming increasingly popular in embedded systems development (Abbot, 2006). Due to the extensive usage of CAN communication in the automotive sector (Lawrenz, 2013), it was chosen as the medium for device inter-communication.

The need for obtaining 'reasonable' accuracy without using GPS signals was desirable in order to just 'fine tune' the path-following using external feedback.

1.4 Thesis Outline

Chapter 1 is an introduction to the problem and a description of the prior work on the soft car.

Chapter 2 is an overview of the Soft car system hardware. It describes all the physical devices and how they are connected to one another.

Chapter 3 is about the CAN Bus Network. It is a theoretical description of the CAN Standard and also describes SocketCAN, an implementation of CAN through network sockets.

Chapter 4 is about the Global Positional System used in the soft car, Racelogic VBox 3i.

Chapter 5 describes the User Control of the soft car system. It elaborates on external communication to the soft car system, its parallel processing mechanism and the user-interface.

Chapter 6 describes the Control System of the soft car and is the most important Chapter.

Chapter 7 describes the Performance of the System.

Chapter 8 highlights the limitations of the soft car and presents suggestions for future development.

2 System Overview

This section provides a complete description of all the used components along with details about how they are connected. Since it is important to keep the soft car as simple as possible, it was attempted to minimise the number of components used without compromising on the performance requirement. This was also important for the company because they require future development of the soft car to be seamless.

The components that are directly part of the control system include a Raspberry Pi 2, an Arduino Mega 2560, Linear Actuators, a Compact Power Motor (Power Pack CPM90-45-4000-L) and a Racelogic Vbox 3i. The Raspberry Pi 2 acts as the 'Master' that controls the Input/output from and to all the other components. All other nodes are 'slaves' to the 'master'. Shown below are two figures that describe how the components are connected.

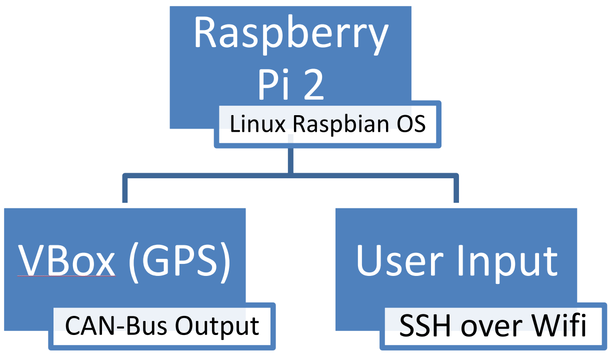


Figure 2: Hardware Interconnection (Input Side).

Figure 2 shows that the Raspberry Pi 2 is master to the input components which include the VBox GPS Data Logger and the user input. The GPS data logger obtains live information from GPS and GLONASS Satellites that are processed by the control system. The users of the system are always connected to the system and can pass commands to and receive information from the navigation system.

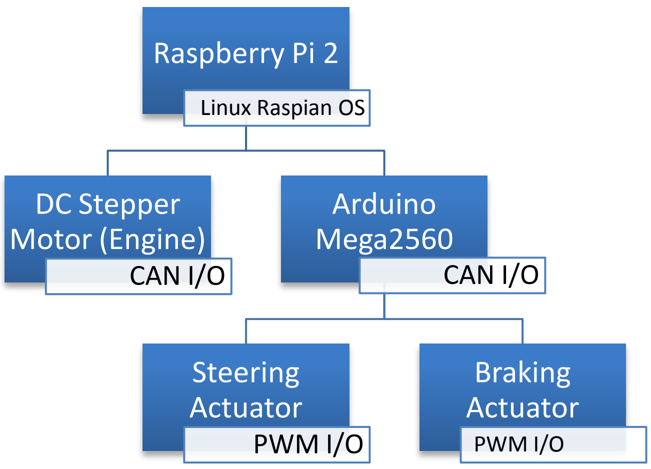


Figure 3: Hardware Interconnection (Output Side).

The Raspberry is also the master on the output side as seen in Figure 3. After processing information from the user and the GPS, it passes CAN messages to the motor and the Arduino board to accomplish the desired movement of the soft car. The Arduino board in turn passes PWM signals to the actuator through its motor shield. The motor also passes information back to its master about its status that describes the phase current being passed, its speed in rpm and whether it has applied emergency braking or not.

The linear actuators contract and expand on positive and negative PWM signals respectively. It also possesses an internal potentiometer that is used to estimate its approximate position by the control system. The accuracy of it was limited and it had been a limiting constraint for the performance of the navigation system.

2.1 Raspberry Pi 2

As mentioned before, the prior development resulted in a remote controlled car capable of reading remote controlled CAN signals using an Arduino to navigate. This system was realised to be extremely weak in terms of speed and performance. Due to the requirement of advanced features (like parallel processing) and simple logging, it was essential to strengthen the computational power of the system. A Raspberry Pi 2 (from now on referred to as RPi) is a credit-card sized single board computer and it was purchased to enable the development in Linux and most of the code was written in C.

A major advantage in the new version of this single board computer (Pi 2) as opposed to the original Pi was the availability of 4 USB ports (Cleevely, 2015) as can be seen in Figure 4. This was useful because it facilitated its simultaneous connection to the Arduino board, a Peak CAN USB transceiver and a WiFi module (WiPi).

Raspberry's default Operating System, Raspbian (Wheezy- May 2015) was selected as the Operating System due to the extensive availability of online community support and the ease of installation. The Wi-Pi (WiFi module) was connected to system based on the recommendation of the company after it was realised that 4G support for the system was limited in the testing area and the facility was not always available for development.

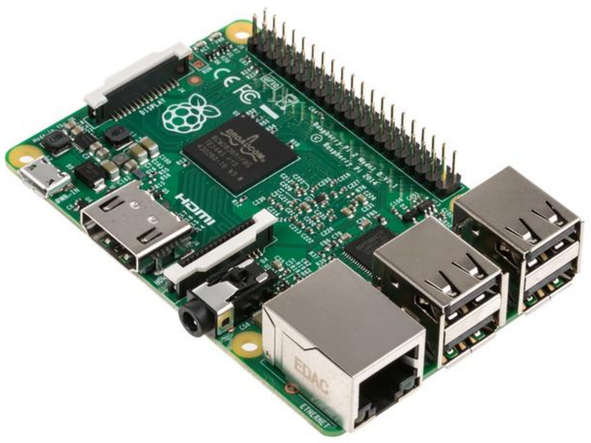


Figure 4: A Raspberry Pi 2 (Element14.com, 2015).

2.2 Peak CAN interface

Directly connected to the Raspberry Pi 2 is the peak CAN interface shown in Figure 5. The main advantage lay in the fact that its timestamp resolution is approximately 42 µs (Peak Systems, 2015). In comparison to Raspberry's CAN Shield, this minimised latency and was chosen to connect the Pi to the CAN network.

Figure 5: Peak CAN Interface (Peak-System.com, 2015).

2.3 Arduino Mega 2560

Prior development was made using an Arduino microcontroller and it had its advantages over the RPi. As our approach was to minimise the number of components being used, it was initially considered to eliminate the Arduino microcontroller. However, the mega 2560 microcontroller (shown in Figure 6) was capable of reading analogue signals which was important for localisation using the potentiometer in the linear actuator. It shall be described in the section: Steering (Potentiometer). It was considered to use an AD Converter (Analogue to Digital Converter) (Alley, 2011) to connect the potentiometer but the performance was significantly inferior to that of the Arduino. Thus, the Arduino was used as an interface between the RPi and the linear actuators.

Another advantage in using an Arduino was the availability of the option of the user connecting to the Arduino through a Teletype Port (Telnet) (Postel, et al., 1983) to make use of arduino's features. Since Arduino's IDE is incompatible with the Raspbian OS, it was necessary to install the 'Ino toolkit' to make changes the Arduino byte-code. Apart from that, the user had the option of connecting to the Arduino directly using a terminal emulator called Picocom.

Figure 6: Arduino Mega2560 Board (Arduino.cc, 2015).

Attached to the Arduino was a CAN Shield and a motor shield connected to the CAN network and the linear actuators respectively. Shields are boards that can be plugged on top of the Arduino extending its capabilities. Different shields usually follow the same philosophy: they are easy to mount, and cheap to produce. Figure 7 shows a prototype of the Arduino CAN Shield and Motor Shield.

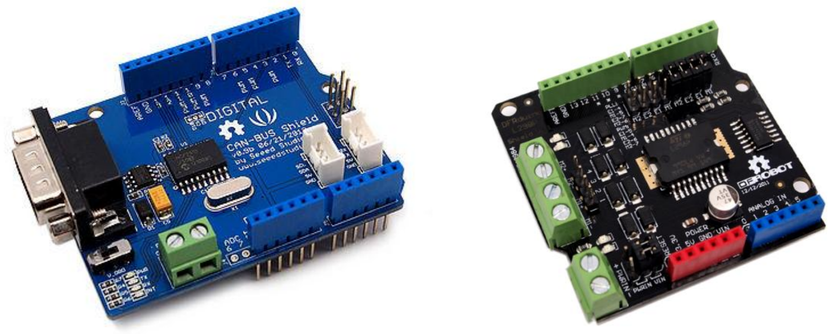


Figure 7: Arduino CAN Shield (L) and Motor Shield (R) (Seeedstudio.com, 2015).

2.4 Linear Actuators

The important factors in selecting linear actuators were speed and load. Although the linear actuators used in the prior development fulfilled the criteria, they did not have any feedback mechanism. A basic requirement for any navigation system is to ensure that the devices are self-aware: they know their own state. It was thus decided, to purchase linear actuators from the same manufacturers that contained some sort of feedback mechanism. The Transmotec DLA series Actuator (shown in Figure 8) was purchased and it replaced the previous actuator because it had a potentiometer which was capable of providing an output voltage proportional to the absolute position of the actuator. Although the feedback was not as accurate as the expectation, time-averaging the signal helped in obtaining a steady signal. Further is discussed in the Steering (Potentiometer) section.

Figure 8: A Transmotec DLA series Actuator with potentiometer feedback.

The significant properties considered in the purchase of the linear actuator were the speed and the thrust which are inversely proportional (Transmotec Product Catalogue, 2015). The trade-off is due to the fact that greater speeds improve the turning speed and is also useful during Heading Correction. Greater thrust improves the turning during low speeds when the kinetic friction is higher.

2.5 Motor

It was imperative to continue with the same motor: the Compact Power Motor (Power Pack CPM90-45-4000-L), shown in Figure 9. This was done because the purchase of an alternate requires a great lead time along with time spent on installation, safety tests etc. Since the motor was CAN based, it was another node in the CAN network that needed an initialisation on every cold start. The ability to easily change direction of motion on receive a CAN message enabled the usage of a reverse gear.

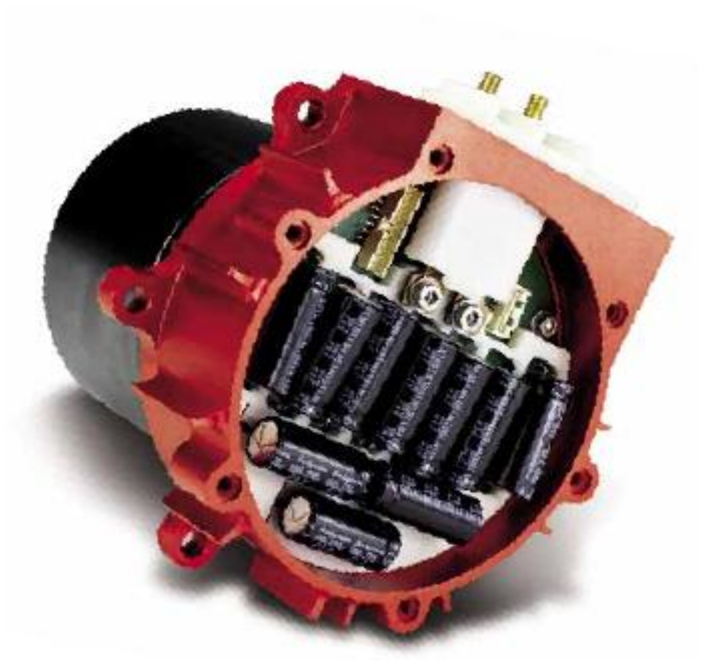


Figure 9: Compact Power Motor (Power Pack CPM90-45-4000-L) (Adigo Drives, 2015).

2.6 GPS System

It was recommended by Volvo to use Racelogic's VBox 3i System as the GPS system. The biggest advantage lies in the fact that these systems are capable of providing an accuracy of up to 2 cm using a supplementary Differential GPS.

Figure 10: RaceLogic Vbox 3 (Racelogic VBox Manual, 2015).

To obtain live dynamic and kinematic data of a vehicle, the primary factor as well as problem is the accuracy. Much of the master thesis has been dedicated to minimise the real-time errors that occur either due to latency or due to system limitations.

A differential GPS is an enhancement to the standard GPS system operating by a stationary ground network or by fixed ground local stations. By knowing the exact position of the stationary receiver, it triangulates errors from the actual GPS satellites and delivers real-time GPS corrections to the vehicle. A base station provides local correction and was sufficient to meet the positional accuracy needs of the system.

Racelogic’s VBox 3i system was used as the GPS in the soft car since it was capable of providing positional information since it was being used in a lot of Volvo Trucks and compatibility would be easy. Capable of transmitting data at a 100 Hz, combined with a Differential GPS (DGPS) it is capable of producing positional information which is accurate to two centimetres (using the proprietary RTK format).

2.6.1 Communication with System

The VBox3i is equipped with 2 CAN Bus interfaces and has 2 RS232 serial ports, one of which is used for communication between the Vbox and the Raspberry Pi. The second RS232 port may be used to connect to a telemetry radio module to receive DGPS information.

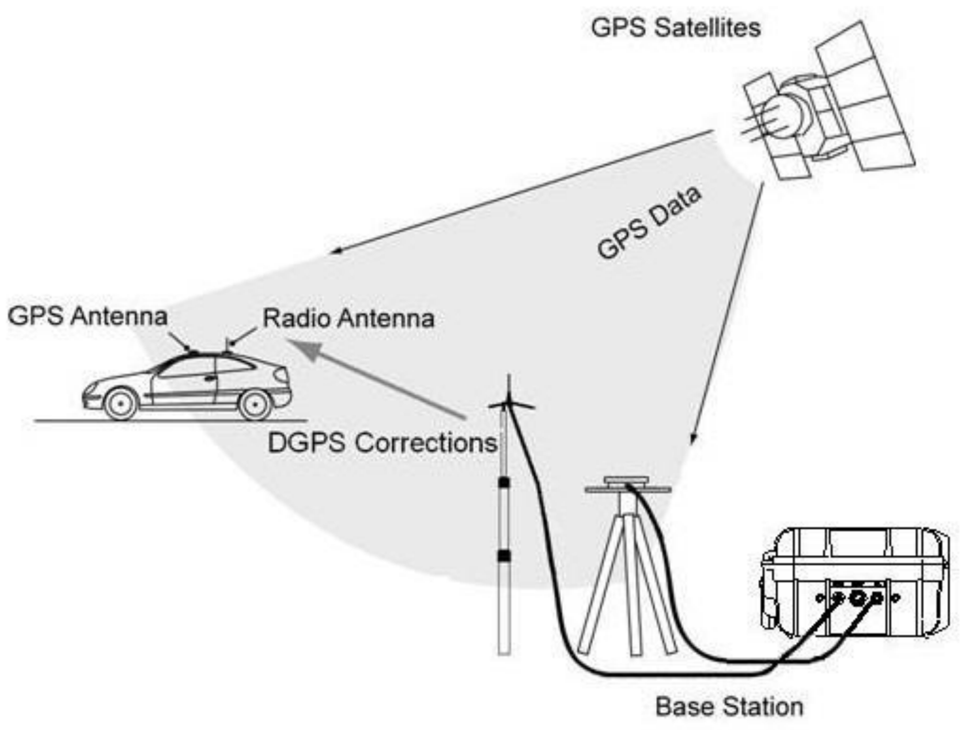


Figure 11: GPS DGPS connection (Racelogic VBox Manual, 2015).

As shown in Figure 11, the car (soft car) possesses a GPS Antenna and a Radio Antenna. The VBox is placed inside the car and is directly connected to the GPS Antennas. Radio antennas are used to interact between the GPS and the DGPS base station which is placed in a stationary position on track. It is important to place the DGPS antenna in the exact same position in order to get correct GPS information. On changing the location, the DGPS has to be reset.

3 CAN Bus

Since there is extensive use of a Controller Area Network (CAN) in the system, an entire chapter is dedicated to understand the theory and internal workings of it. Inside the soft car, the GPS Data Logger, the Motor, the Arduino board (through the CAN Shield) and the Raspberry Pi 2 (through the Peak Can interface) is connected to one another forming a CAN Network.

A controller area network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer (Lawrenz, 2013). It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles, but is also used in many other contexts.

Modern automobiles possess as many as 70 ECUs (Electronic Control Units). Apart from the biggest processor (the engine control unit), transmission, airbags, antilock braking, cruise control, electric power steering, audio systems, power windows, doors, mirror adjustment, battery and recharging systems for hybrid/electric cars are some of the ECUs that are part of the CAN network. The CAN standard was devised to feed information and receive feedback from sensors.

3.1 Architecture Fundamentals

Two or more CAN nodes are required to be part of the network to communicate (Bosch, 1991). These nodes may be as complex as an embedded computer (RPi) and as simple as a simple Input/output device (like the motor). The nodes are connected to each other through a nominal 120ohm twisted pair bus. Shown below (Figure 12) is an example of a high speed CAN network.

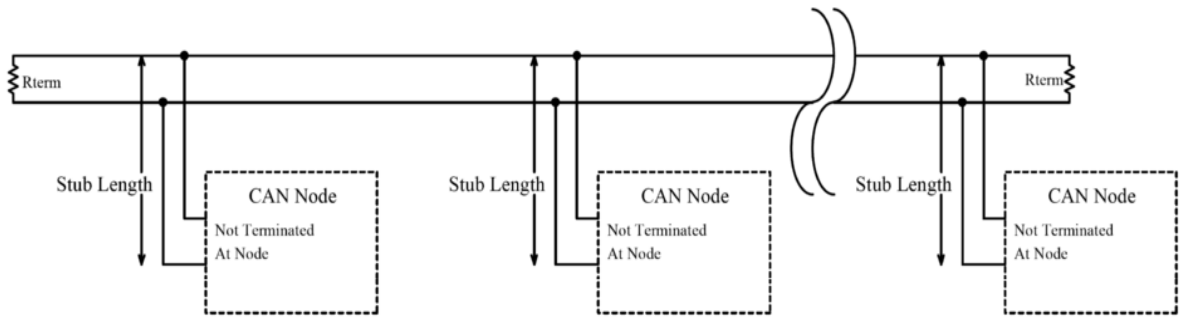


Figure 12: CAN Network example.

Each node in a CAN network consists of a CPU (microprocessor), a CAN Controller and a Transceiver. The nodes are able to either send or receive messages at any given time instant. Following are a couple of lines about the node components

- Processing Unit: The host processor has the right to decide and the messages the node wants to transmit and is supposed to evaluate what the received messages mean. For example, the engine's control device is connected to the processing unit.
- CAN Controller: The controller stores received serial bits from the CAN bus up to the point the entire message is available. After receiving the entire message, it feeds the host processor with it.

- Transceiver: Its job is to convert the data stream from the CAN Bus level (i.e. electrical voltage) to a usable form for the CAN controller and vice-versa. Usually, it has a protective circuitry to protect the controller. Transmitting and Receiving cannot be done simultaneously.

3.1.1 CAN Abstraction Layers

The International Organization for Standardization (ISO) developed the Open System Interconnect (OSI) model in 1984 as a model of computer communication architecture (Wheat, et al., 2001). There are seven layers to the OSI model: Physical, Data Link, Network, Transport, Session, Presentation, and Application. The seven layers are represented below in Figure 13. The intent is that protocols be developed to perform the functions of each layer as needed.

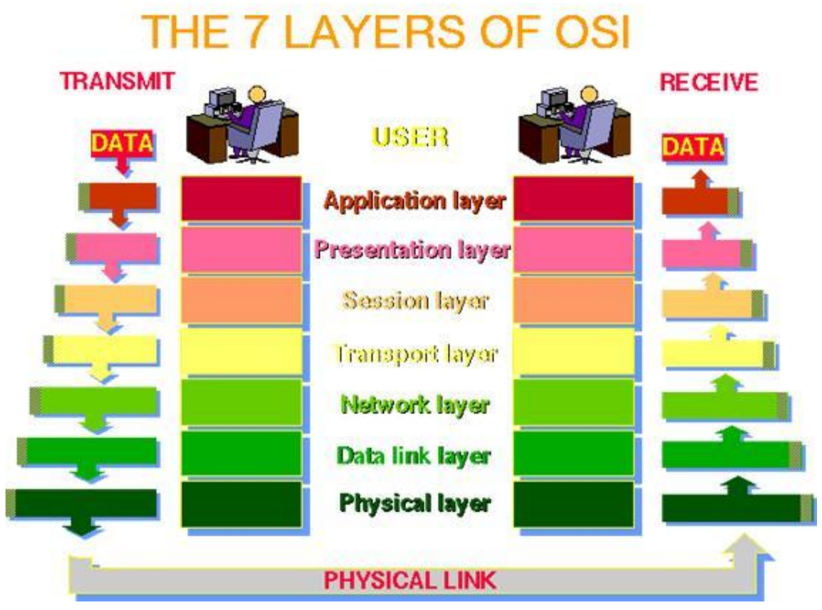


Figure 13: The Seven Layers of the Open Systems Interconnect (Networks Mania, 2008).

The CAN messaging protocol may be decomposed into several such layers containing its own properties and the following describes tasks assigned to each layer:

- Physical Layer: Is responsible for the network interconnection. One CAN Node transceiver is responsible for transmitting a CAN message at a time instant on the physical layer and is received by every other node. Each node has a custom connector and the most common such connector is a 9pin D-sub type male connector with pin 2 assigned as CAN-Low, pin 7 as CAN High, pin 3 as Ground and pin 9 for power. Details about CAN Low and CAN High are provided later in the chapter.
- Data Link Layer: Is responsible for transferring the raw CAN signals from the physical layer and transmitting the messages to the object layer in the fashion that is set by the CAN standard (Bosch,1991). The CAN standard for this layer include specifics about Fault Confinement, Error Detection, Message Validation, Acknowledgement, Arbitration, Message Framing, Transfer Rate and Timing, Information Routing.
- Object Layer: Is responsible for message filtering and status handling.

- Application Layer: Is responsible for processing CAN messages and using it in desired applications and conversely, converting the application output data to a CAN message before its transmission.

3.1.2 CAN Physical Layer

Before dealing with the details of a CAN Frame (which is binary, consisting of zeros and ones) it is important to understand the physical layer of the CAN protocol and examine the relation between actual voltages in the CAN Bus and how it relates to the binary digits.

The CAN standard specifies two logical states: recessive and dominant. Unlike many other standards, the 0 bit is considered to be the dominant state and the 1 state is said to be the recessive state. The differential between the CAN High voltage and the CAN low voltage represent this present dominant or recessive state. Figure 14 below describes the state of the CAN bit in relation to the differential voltage.

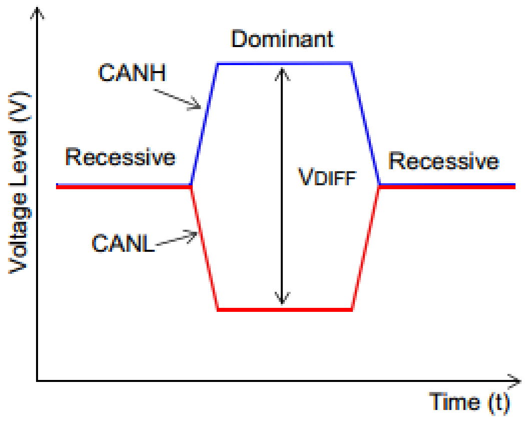
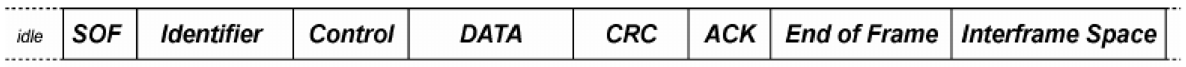


Figure 14: CAN Bus Voltage vs. Time.

It is seen that a large difference represents the Dominant state of the can bit whereas a negligible difference stands for the recessive state. On the MCP2551, a common CAN chip, the threshold for a dominant state is a voltage delta of greater than 1.5 Volt and that for the recessive state is lower than 0.5 Volt.

3.2 CAN Messages

CAN messages are fundamental entities being transmitted and received by CAN nodes in the network. The basic CAN message (also known as a CAN Data Frame) is shown in Figure 15 after which each part is described.



idle	*SOF*	*Identifier*	*Control*	*DATA*	*CRC*	*ACK*	*End of Frame*	*Interframe Space*

Figure 15: CAN Message Format

3.2.1 Idle

Each CAN message is preceded by an ‘idle’ period during which the bus is in recessive. The minimum number of bits during which the idle state must be held is 11. The CAN standard essentially states that after the ‘idle’ period has passed, a “ready for message” condition is applied.

3.2.2 SOF (Start of Frame)

A dominant bit is transmitted at the start of a CAN message signifying its start.

3.2.3 Identifier

Every CAN Data Frame contains an identifier (an address) that is of sizes

- 11bits: For the CAN 2.0A standard
- 29bits: For the CAN 2.0B standard

(In the soft car, all CAN messages transmitted and received are of the CAN 2.0a (11bit) standard)
The identifier competes for priority against other CAN messages sent at the exact time instant. There is a battle of arbitration that priorities lower valued CAN identifiers.

3.2.4 DLC (Data Length Code)

This describes the size of the data that is to be transferred. Four bits are reserved for the DLC despite the fact that only data sizes between zero and eight are used.

3.2.5 Data

The part containing the actual data to be transferred succeeds the DLC. CAN nodes may transmit data in little or big endian bytes orders. It is important to maintain consistency between the byte orders so that interpretation of the data is not difficult.

3.2.6 CRC (Cyclic Redundancy Check)

A CAN Data Frame includes a CRC to increase the reliability of transferring messages. Based on the data, the transmitting node calculates a checksum and posts it in the CRC field which is validated by the receiving nodes CRC calculation. On a match, it may be assumed that there was not any loss of data during the transmission. CAN Frames usually use a 15 bit CRC.

3.2.7 Acknowledgement

The acknowledgement part of the CAN Frame is used by all network nodes to verify whether the CAN transfer is correct or incorrect. The acknowledgement process is a two-step process organised as below:

- All network nodes agreeing that the transfer is correct indicate the condition during the acknowledgement bit by sending a dominant bit
- Any network members, after the first step, that disagree with the transfer indicate with an error flag. Thus each node has the power to veto

3.2.8 EOF (End of Frame)

The last 7 bits (recessive) mark the end of the frame. It also serves as the time that may be used by the nodes that disagree with the CAN message

3.3 Soft Car CAN Network

The CAN Network in the soft car consists of:

- Raspberry Pi: Connected via Peak CAN interface
- Arduino Board: Connected via a CAN Shield
- Motor
- GPS

A 4-way CAN Splitter consisting of 9pin D-sub type ports was used to connect the nodes. Shown below is a table of important CAN messages and their descriptions.

Table 1: CAN Data Frames

CAN Id (HEX)	Transmitting Node	Intended Recipient	Contained Information
0x100	R Pi	Motor	Reference Velocity, Motor Activation
0x118	R Pi	Arduino	Desired Actuator Position
0x119	Arduino	R Pi	Actuator position
0x200	Motor	R Pi, Arduino	Motor Status
0x301	GPS	Arduino, R Pi	GPS: Time, Latitude, Nr of Satellites
0x302	GPS	Arduino, R Pi	GPS: Longitude, Velocity, Heading
0x304	GPS	Arduino, R Pi	GPS: Distance
0x306	GPS	Arduino, R Pi	GPS: True Heading
0x620	R Pi	Motor	Reference Direction: Forward/Reverse
0x630	Motor	R Pi	Direction Status: Forward/Reverse
0x710	Motor	R Pi	Phase Current

Each of the variables are used either in real time or only while the soft car is static. They form the crux of the control system input. In order to ensure that the CAN network is not jammed due to excessive traffic, the CAN messages are synced with the live time transmitted by the GPS whose frequency is 100Hz. Thus, the Raspberry Pi and the Arduino are tuned to transmit the CAN messages once every 10 milliseconds.

3.4 SocketCAN

SocketCAN is an implementation of CAN protocols for Linux. It uses the Berkeley socket API, the Linux network stack and implements the CAN device drives as network interfaces (Kleine-Budde, 2012).

The CAN socket API has been designed as similar as possible to the TCP/IP protocols to allow programmers, familiar with network programming, to easily learn how to use CAN sockets.

The Raspbian Linux distribution includes SocketCAN drivers and it makes CAN data easily available to be programmed in most standard languages (C here) and receiving and transmitting CAN data may be achieved with minimal latency.

The Linux networking subsystem, being highly flexible contains several protocols including IPv6, IPv4, ATM, ISDN etc. To support the CAN system, a new protocol family **PF_CAN** is used that includes a **CAN_RAW** protocol (Schwebel, et al., 2009).

It provides a kernel internal infrastructure for CAN frame sending/receiving and filtration to implement more complex protocols inside the kernel. It also contains drivers for various CAN based devices.

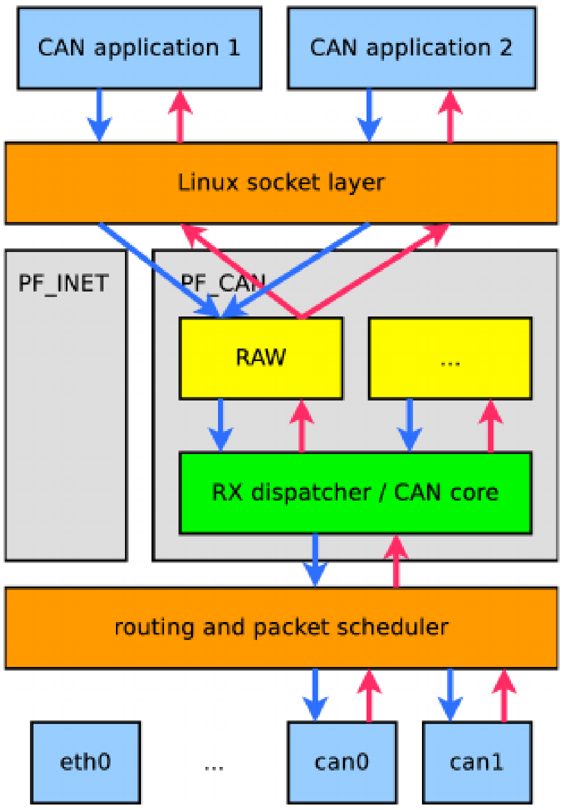


Figure 16: SocketCAN Implementation in a Linux environment (Kleine-Budde, 2012).

With reference to the soft car system, Figure 16 shows the CAN application (soft car control system) interacting with the Linux socket layer. In the source code an initial call is made to open the CAN port and there is continuous transmission and reception of CAN messages thereafter. Details will be provided in the next subsections: Usage and Read/Write CAN Frame.

The primary advantage of using such a protocol family is that it allows the possibility to use a CAN Bus system without the need for using device specific drivers. Thus, platform independence is achieved and for the soft car system, SocketCAN enables easy transfer of technology while upgrading to better hardware.

As compared to character device based solutions, SocketCAN:

- Makes use of established POSIX socket APIs to assist in application development
- Uses the standardised networking driver model as Ethernet drives
- Enables easy implementation of communication protocols and complex filtering inside the kernel
- Supports multi-user and multi-application access

The last feature, multi-application support is useful in creating an autonomous navigation system because real-time debugging is a key requirement in development. In this work, a CAN based tool known as *Candump* has been used in order to read and log CAN data while simultaneously receiving and transmitting data from the main application.

It is possible to open multiple sockets simultaneously and they can send/receive on the same or different can IDs. An application wanting to communicate with a specific transport protocol selects it while opening the socket and the read from/writing to the socket is just like reading any network data byte stream.

3.4.1 Usage

This subsection describes the usage of SocketCAN with reference to its application in the soft car system. Like a standard socket in any TCP/IP, the first requirement is to open it for communication over a CAN network.

The protocol family PF_CAN and the protocol CAN_RAW need to be passed as parameters along with the socket type. The default socket type is SOCK_RAW.

s = socket(PF_CAN, SOCK_RAW, CAN_RAW);

After the creation of the socket, a basic CAN Frame structure and sockaddr structure (Socket Address structure) may be used to send/receive CAN Frames. The structures may be found in the header file:

/linux/can.h

The CAN Frame structure looks like

```
struct can_frame {
canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
__u8  can_dlc; /* frame payload length in byte (0 .. 8) */
__u8 data[8] __attribute__((aligned(8)));
};
```

It may store all the information from the CAN Frame in a systematic manner. An important point to be mentioned here is that there is no specific byte order on the CAN bus and both little endian and big endian may be used. The sockaddr_can structure has an interface index that binds to a specific interface

```
struct sockaddr_can {
sa_family_t can_family;
int  can_ifindex;
union {
      /* transport protocol class address info (e.g. ISOTP) */
      struct { canid_t rx_id, tx_id; } tp; };
      /* reserved for future CAN protocols address information */
      } can_addr;
};
```

To find the interface index, an appropriate ioctl (input/output control) has to be used as follows

```
int s;
struct sockaddr_can addr;
struct ifreq ifr;
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
strcpy(ifr.ifr_name, "can0" );
ioctl(s, SIOCGIFINDEX, &ifr);
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
bind(s, (struct sockaddr *)&addr, sizeof(addr));
```

This is sufficient to read and write from the data byte stream and is all that has been used in the application.

3.4.2 Read/Write CAN Frame

On opening the socket, the CAN Data can be read using the **read()** function and similarly a CAN message may be sent using the **write()** function. While reading the CAN Bus network with:

```
recvbytes = read(soc, &frame_rd, sizeof(struct can_frame));
```

the entire can message is stored in the CAN Frame (here **frame_rd**) and may be utilised to extract information. Likewise, while writing to the CAN network, the structure (here **frame_rd**) must be declared before transmitting with

```
ret = write(soc, &frame, sizeof(frame));
```

4 User Control

To be able to understand the user control system in a thorough manner, it is important to focus on the software development including debugging procedures carried out after testing. In order to build a navigation system, the back-end of the software is the obvious priority whereas the front-end may be left simplistic. Since the front-end of the software is not vital during the development stage, the basic requirements while testing may be listed as:

- Easy access to control features
- Possibility to tune various device and kinematic parameters
- Getting real-time data from the system so as to enable convenient debugging (Verbose)

Using the aforementioned points as guide, it was important to create a user-interface that was intuitive as well as configurable. A command-line interface was sufficient for the needs of development. Using SSH to connect to the Raspbian OS, it is a one-step procedure to open the program that accesses the CAN Bus and offers the user to ability to control the car.

For most parts of the testing, a mobile phone was used to connect to the system and the entire procedure of connecting to the system and obtaining a command prompt took no longer than 15 seconds.

The section describes user-interface is described along with features that are oriented towards navigation and safety.

4.1 Parallel Processing

Parallel processing is the simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads. Ideally, parallel processing makes programs run faster because there are more engines (CPUs or cores) running it (Hwang, et al., 1984).

For the user-interface of any run-time environment (much less a soft car with crashing risks), it is of primary importance to have the prompt available to the user at all times. For instance, navigating through a path may involve calling the set of navigation functions repeatedly until the path has been completed (or there is a timeout).

During this time, if there is a failure or something unexpected, it is of utmost importance to control the system so that emergency calls can be made. To enable this, RPi’s Quad Core processor has been utilised properly using the OpenMP (Open Multi-Processing) API that supports multi-platform shared memory multiprocessing programming.

OpenMP has also been utilised to create a virtual fence around the track so that the soft car automatically stops on reaching this wall and is able to return to a particular base if required. Raspberry Pi 2 uses a 900MHz quad-core ARM Cortex-A7 CPU that allows OpenMP to execute sets of functions in parallel.

OpenMP is an implementation of multithreading, a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them (Chandra, 2001) as seen Figure 17: Representation of

Multithreading (en.wikipedia.org/wiki/OpenMP, 2015). The threads then run concurrently, with the runtime environment allocating threads to different processors.

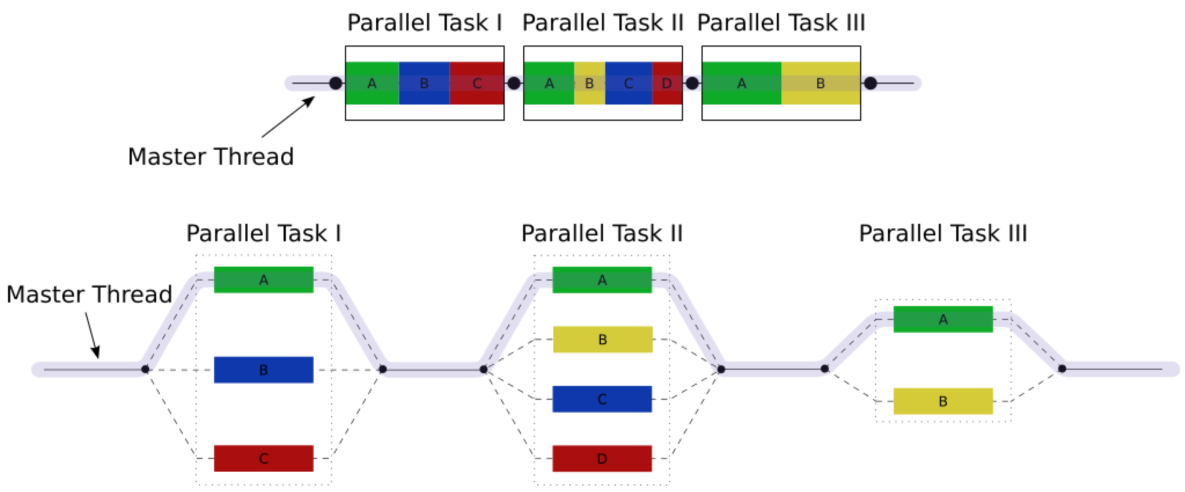


Figure 17: Representation of Multithreading (en.wikipedia.org/wiki/OpenMP, 2015).

A pre-processor directive marks the part of the code that is to be run in parallel. The threads all of id's attached to them that can be obtained using a function: **omp_get_thread_num()**

The thread id may be between 0 and n-1 where n is the number of available cores. After the execution of the parallelised code, the threads join into the master thread that continues until another pre-processor directive calls for a parallelised section. The runtime environment allocates threads to processors. The factors for allocation include processor usage, machine load amongst others. The runtime environment can assign the number of threads based on environment variables, or the code can do so using functions. The OpenMP functions are included in a header file labelled omp.h in C/C++.

For the 4 Core Raspberry Processor, the tasks allocated to separate cores are as follows:

- User Prompt
- GPS Control and Correction
- Sending Data
- Receiving Data

Maintaining an independent user prompt is of primal important due to the fact that the user should have the final control of the soft car at all times. This is especially critical in cases of emergencies when the user should have the power to override expected functions. All the navigation functions are processed by a core. Another Core is used to sending CAN Data Frames. It is important to use a separate core to be able to apply safety features (for example: electrical fence or speed limit) that should override functions results from other cores. The last core is used to receive data. It is vital to get live GPS data as frequently as possible to increase the overall accuracy of the soft car. Descriptions of each of the Processes are part of the next section.

4.1.1 Interaction between Cores

In Figure 18 below, it may be seen how the four cores interact with one-another. Cores 3 and 4 are continuously sending and receiving CAN messages respectively. They run infinite loops until specifically asked by the user (Core 1) to break out of it.

The user may interact directly with Core 2 (Functions) or Core 4 (Sending Data). All the navigational requests are made to Core 2 (Functions). If Core 2 (Function) accepts the request, it communicates with Core 4. Its job is to set the CAN messages to appropriate requests for Core 4 to deliver to the CAN Network. Core 2 also continuously receives live GPS data from Core3 (Receive CAN Data). The user may request some or all of the live data from Core 1 to be displayed and/or logged. The user (Core 1) is permitted to directly set the CAN message that is to be sent out by Core 4. This applies only in the case of an emergency.

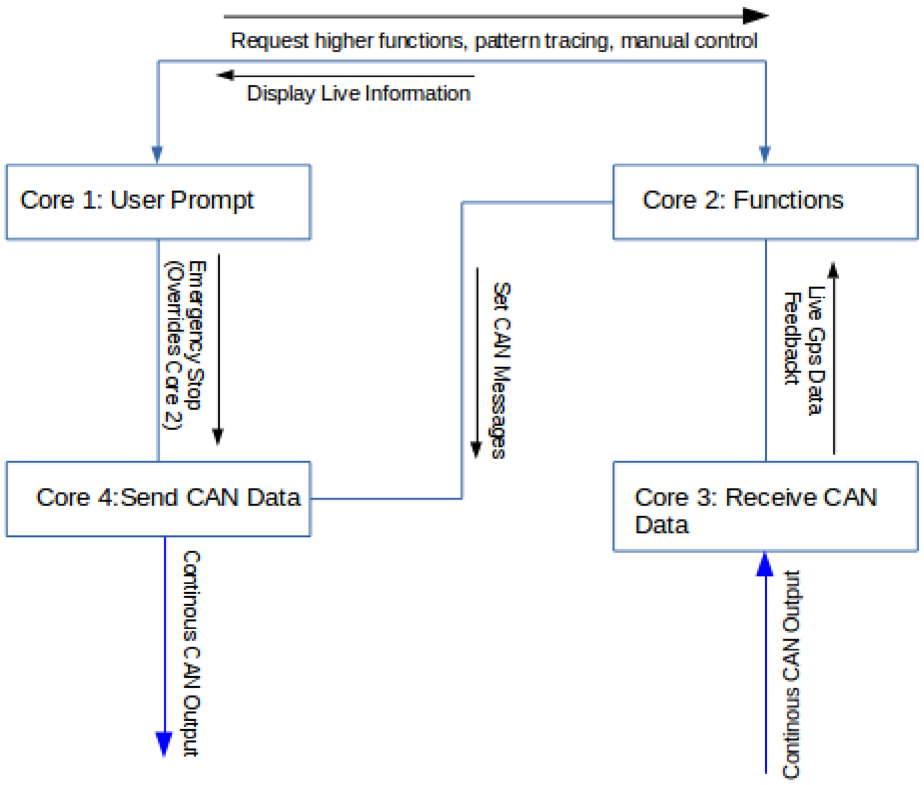


Figure 18: Interaction between Raspberry's 4 Cores.

4.2 User Interface

In this section, the command user interface is described. The user-interface during the development stage primarily focuses on ease of control, ease of tuning parameters and safety. The safety requirements involve emergency stopping of an out-of-control vehicle and emergency change of direction in case of unexpected incoming traffic. The features available to the user through the interface may broadly be classified into the following categories:

- Manual Control (Lower Functions)
- Isolated GPS Correction Features
- Higher Functions
- Path Following

For development purposes and during full testing, it is mandatory to have Manual Control of the soft car system. The requirement is due to the fact that it may be used to setup the car in a particular way, check if the mechanical components of the car are working properly or not, transport the car from one place to another when it's too difficult or time-consuming to build a path for doing so amongst others.

The manual control also consists of other features of the soft car including enabling/disabling logging, displaying the menu for fine tuning driving parameters and displaying live GPS and vehicle data. The manual control functions are regarded as 'lower functions' because the user is directly in control of the motor and turning actuator. These functions do not consist of any feedback from the GPS and are thus, not 'smart'. There are a set of GPS Correction features like velocity correction and heading correction that may be used in isolation or in combination manually so in order to check that it is calibrated properly.

Calibration may be required with change of testing conditions: factors of which may include surface type, weather condition, tyre pressure etc. Higher Functions make repeated use of lower functions with the help of GPS Data in order to form a feedback system. For instance, turning to the right until the heading increases by 60 degrees is a higher function in this context.

Path Following consists of repeated calls of higher functions so that a particular path is covered as expected. The Path Following call also consists of setting up points and lines to use as reference to make turns, accelerations etc. The following lists each function of the each of the four user-interface components.

4.2.1 Manual Control (Lower Functions)

Table 2 is a list of lower functions.

Table 2: Lower Functions

Nr.	Function	Remark
1	Turning Left, Right, Centre	Based on Actuator Potentiometer
2	Set Velocity	Based on Motor Reference Rpm
3	Speed Factor[1]	For higher speeds
4	Initialise Motor	To Restart Motor
5	Set Motor Gear[2]	Forward/Reverse
6	Log Data	Toggle
7	Display Data	Shows live GPS Data
8	Set Base Location	Set as starting point
9	Emergency Stop[3]	Overrides other functions

[1] Since a standard keyboard has keys ranging from 0 to 9, a speed factor may be used to achieve higher speeds. For instance a speed factor of 3 may be set which enables the speed of 9 which the '3'key is pressed and 12 when '4 'is pressed.

[2] Available only when velocity is 0

[3] As dicussed in section: Parallel Processing

4.2.2 GPS Correction Features

Table 3 is a list of GPS Correction Features.

Table 3: Correction Features

Nr.	Function	Remark
1	Heading Correction	Maintains a particular Heading
2	Set Heading	Sets the current heading as references
3	Velocity Correction	Uses GPS Data to correct reference velocity
4	Change Heading Mode	True Heading/ Velocity obtained heading
5	Return to Base	Auto-returns to Base Position

4.2.3 Higher Functions

Table 4 is a list of higher functions.

Table 4: Higher Functions

Nr.	Function	Remark
1	Turning by Delta[4]	Turns soft car until desired difference of heading
2	Turning by Absolute Heading	Turns soft car until desired heading
3	Acceleration/Deceleration	GPS feedback based
4	Lane Change (Distance Based)	Make Lane Change based on distance from initial line
5	Lane Change (Time Based)	Make Lane Change for desired time span

4.2.4 Path Following

Path following may be achieved by using a set of higher and lower functions to follow a particular path repeatedly.

[4] The soft car may be asked to turn by a particular difference in heading. This difference is referred to in the control system and source code as 'delta'. For example, the soft car while initially at 45 degrees may to asked to decrease the heading by 15 degrees (delta=15 degrees)

5 Control System

A bottom-up approach has been used to describe the Control System of the Soft car. It is important to know about the functions and variables that are closer to the motor and turning actuators. Using them, upper level functions have been created which eventually may be controlled manually by the user or by the automated sequence of calls designed for a particular path.

To review, the CAN Bus connects the GPS, motor, Arduino and RPi together forming a CAN Network. The Arduino is connected to the actuators via a motor shield. Since the RPi is the master, it directs the Arduino to steer the soft car through directive CAN Messages. The Motor responds to CAN Messages sent from the RPi.

In this section, first the Arduino board's task to use CAN messages to steering the section car is described. Subsequently, there significant variables in the control system are introduced and its uses are explained. Thereafter, the correction functions are described. Correction functions use GPS information and accordingly steer the soft car in a direction or velocity to minimise the error in position and/or velocity.

5.1 Steering (Potentiometer)

The soft car is steering using an actuator that is connected to the axle. The actuator is controlled by the Arduino board through a motor shield which is capable of passing 12 Volt PWM signals to the actuator. For localisation, the actuator contains a potentiometer that is used to track the absolute position of the actuator. This potentiometer may be traced back by the Arduino board which is capable of reading analogue signals. Figure 19 illustrates the various absolute positions of the actuator with respect to the potentiometer value.

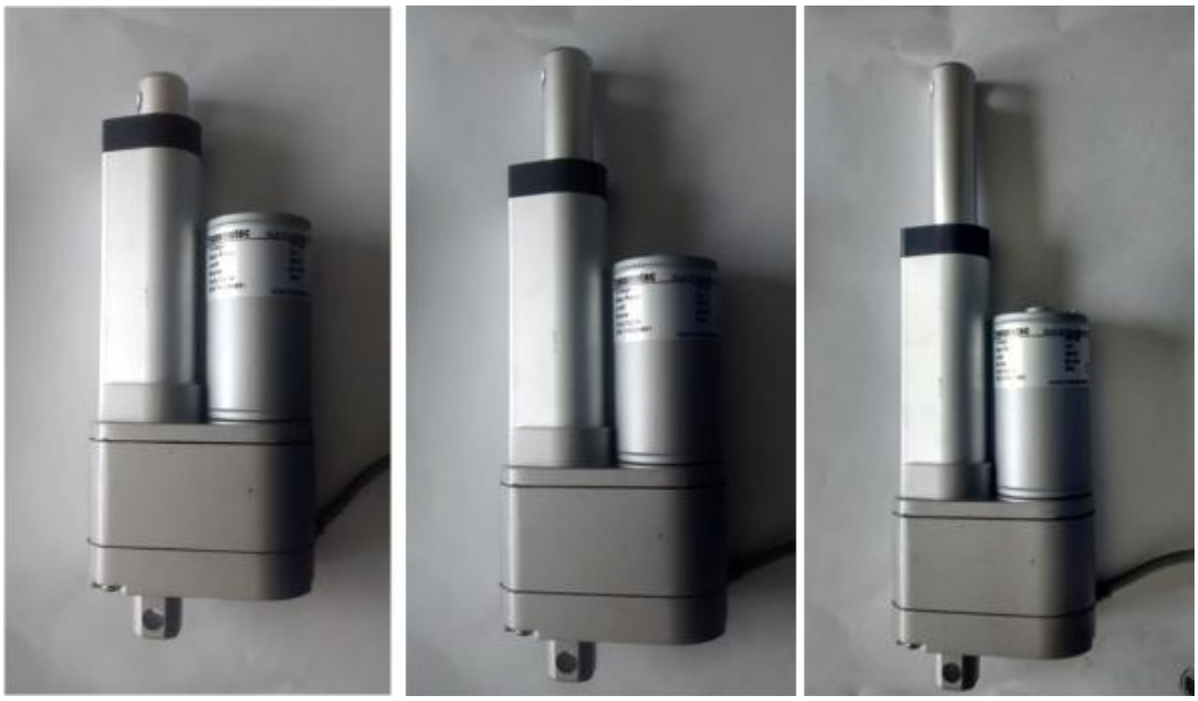


Figure 19: Potentiometer Values: 50 (L), 512 (M), 980 (R) representing different absolute positions of the actuator.

An Arduino board reads analogue signals on a unitary scale of 0 to 1023 that scales from 0 Volts to 5 Volts. The potentiometer in the actuator is connected to a 5 Volt input and thus the inner and outermost positions of the actuator should correspond to the unitary scale of the Arduino analogue reading scale. However, the mechanical locks in the actuators prevent the actuator to reach the end-points of this scale as shown in Figure 19, the innermost position is valued around 50, the outmost at 980 and the centre-most position is exactly 512 which is the midpoint of the analogue reading scale.

The stroke length of the actuator is under-utilised because of the limited requirement of turning radius. In addition, the soft car was mechanically constrained from using the full stroke length of the actuator.

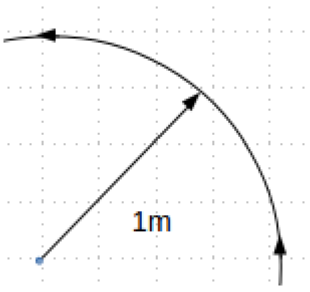


Figure 20: Potentiometer Val: 512

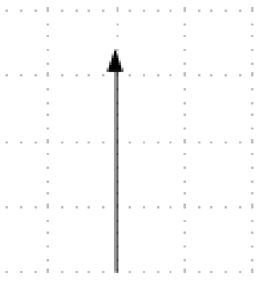


Figure 21: Potentiometer Val: 512

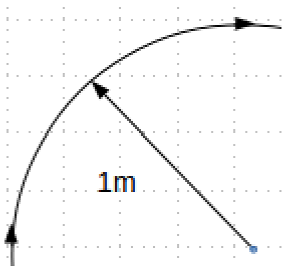


Figure 22: Potentiometer Val: 437

On calibration, a range of 150 points on the analogue reading scale was enough to make a left or right turn of a 1 metre radius of curvature. As shown on the figures above, if the straight line is achieved at a value of 512, a 1 metre radius left turn was achievable at the value of around 587 (512+75) and a 1 metre radius right turn could be made at the value of 437. The significant potentiometer values are shown in Figure 23 below. Table 5 lists the important potentiometer values and their short forms.

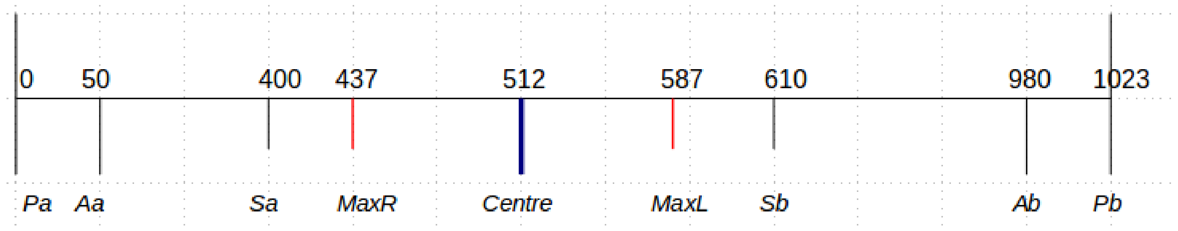
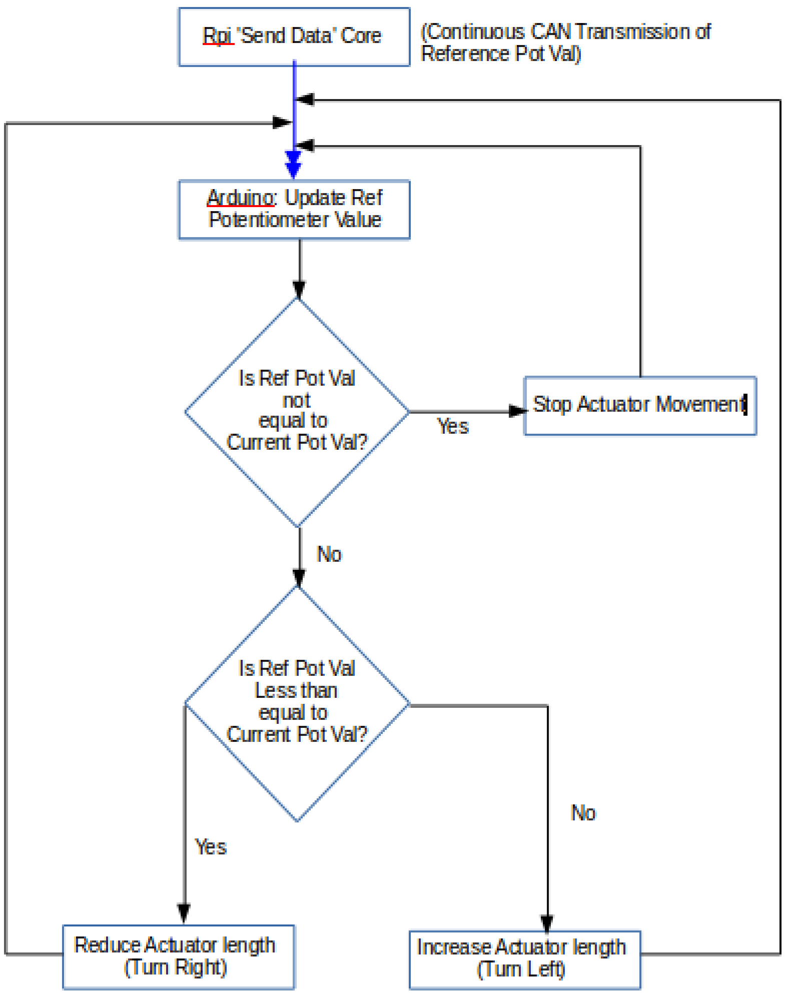


Figure 23: Important Potentiometer Values on the unit scale

Table 5: List of Important potentiometer Values and Short Forms

Nr.	Short	Definition
1	Aa	Actuator Lower Limit
2	Ab	Actuator Upper Limit
3	Pa	Minimum Potentiometer Value
4	Pb	Maximum Potentiometer Value
5	Sa	Soft Car Mechanical Constraint: Right
6	Sb	Soft Car Mechanical Constraint: Left
7	MaxL	1 metre radius left turn
8	MaxR	1 metre radius right turn
9	Centre	Straight Line

Figure 24 describes the workings of the Arduino board. It may be seen that the RPi is transmitting CAN messages (faster than 100Hz) to the Arduino. The CAN message contains the reference value for the potentiometer which is compared to the current potentiometer value transmitted back to the Arduino board. If case of a difference, the motor shield passes a voltage to increase or decrease the length of the actuator till the reference value has been attained.



Rpi 'Send Data' Core
(Continuous CAN Transmission of Reference Pot Val)
Arduino: Update Ref Potentiometer Value
Is Ref Pot Val not equal to Current Pot Val?
Yes
Stop Actuator Movement
No
Is Ref Pot Val Less than equal to Current Pot Val?
Yes
No
Reduce Actuator length (Turn Right)
Increase Actuator length (Turn Left)

Figure 24: Flowchart representation of Arduino code.

Having an idea about the potentiometer values with respect to the turns is enough to understand the next part of the section involving RPi's tasks.

5.1.1 Limitation

One significant constraint in creating the navigation system was the performance of the Arduino. The actuator potentiometer was supplied voltage from the Raspberry Pi. A 5 Volt input into the potentiometer was used by it to supply the Arduino with an input voltage that was proportional to the potentiometer. However, it also was directly affected by the input voltage that fluctuated. The fluctuation was even greater when the Arduino was used to supply the input voltage to the potentiometer.

Thus, for a static actuator, the potentiometer varied. It was necessary for the Arduino board to time-average the potentiometer value over a few milliseconds. This was done in order to get a uniform response.

Secondly, when the Arduino was asked to steer to a specific potentiometer value, it was not able to perform with absolute accuracy that was expected. Thus a tolerance of two to three units in the unitary scale mentioned in the previous section was used. This resulted in a compromise in performance. It was observed that the actuator moved back-and-forth one or twice around the desired potentiometer value with an amplitude of two or three.

5.2 Significant Variables

Since a bottom-up approach has been chosen, it is necessary to understand the important variables that are used in a large number of functions. Despite the popular stance of global variables being evil (Oman, et al., 1990), it was used due to necessity. The follow is a grouped selection of global variables that have been extensively used

- GPS Data Variables
- CAN Data Variables
- Coordinate System Variables
- Potentiometer Value Variables
- Tuning Parameters
- Motor Variables
- Miscellaneous Variables

5.2.1 GPS Data Variables

The GPS Data Variables store CAN message Data and are used by almost all higher functions. All the variables are one of 32bit or 16bit integers. The variables represent the following

- Number of Satellites Connected
- Live Latitude
- Live Longitude
- Live Time
- Live Velocity
- Live Heading (Velocity Based)[5]
- Live Heading (True[6])

Due to mechanical limitations, the live heading (true) was not always available to be used for GPS correction. Thus it was also necessary to use the Velocity based live Heading

5.2.2 CAN Data Variables

Before transmitting the CAN messages to the network, data have to be stored in CAN Data frames. These CAN data (sets) have been listed in Table 6

[5] Live heading is calculated automatically by the VBox by measuring the Doppler shift in signals received from GPS satellites

[6] True Heading refers to the actually heading that is measured by the dual antennas of the GPS

Table 6: CAN Messages continuously saved in the RPi Memory before transmission/ after reception

CAN Id (HEX)	Transmitting Node	Intended Recipient	Contained Information
0x100	R Pi	Motor	Reference Velocity, Motor Activation
0x118	R Pi	Arduino	Desired Actuator Position
0x620	R Pi	Motor	Reference Direction: Forward/Reverse
0x630	Motor	R Pi	Direction Status: Forward/Reverse

This is the shortened list from the CAN Data Frame table in section: CAN Bus. The table only consists of the CAN Data Frames transmitted by the RPi to the Bus. Before transmission, a number of functions may change the information contained within these CAN Data variables. The first two, **0x100** to the motor and **0x118** to the Arduino are used most often.

5.2.3 Coordinate System Variables

Only two of the Coordinate System Variables that are global have been used repeatedly. Called **xNow** and **yNow** in the code, they represent the live position of the soft car on a Cartesian coordinate system with respect to a base that is selected by the user. Since the testing track covers a small area (of less than 100m X 100m), the surface is assumed to be flat and it has been assumed that the distance from any reference point is linearly proportional to the latitudinal and longitudinal difference from the reference point.

5.2.4 Potentiometer Value Variables

It is important to store the potentiometer values that represent

- Moving forward (Straight)
- Max Left
- Max Right

These variables are frequently passed to the CAN Network either by manual control or by higher functions in isolation or part of a pattern. These values are stored as variables and not as constants due to the fact that at times it is necessary to marginally alter their values (especially moving straight forward) because of any change in testing conditions and/or alterations in the soft car mechanics.

5.2.5 Tuning Parameters

During the development stage, it is often necessary to change vehicle parameters to obtain accurate results. These variables are global so that they can be called by the functions and also by the user in order to manipulate them. The tuning parameters include

- Heading Tolerance: When asked to pursuit a particular heading, the heading tolerance is the minimum error margin that the soft car is able to navigate smoothly without diverging in a zigzag manner. For standard test conditions, this value was most often 1.5 degrees
- Coefficient of Potentiometer Value: A value relating the heading difference and the sway from the centre potentiometer value to obtain the desired correction. Described further in section: Convergence Scheme
- Electrical Fence Limit: The electrical fence size can be varied according to track size

5.2.6 Motor and Potentiometer Reference Variables

The reference variables for the motor and the actuators are used for navigation. These lower variables are repeatedly used and are very significant global variables.

5.3 Correction Functions

Live GPS data is used by the system to follow a path accurately and at the desired speed. One way of achieving this is to implement PID Controllers for the system to be able to navigate properly. This section describes the control mechanism applied to obtain desired speed and heading.

5.3.1 GPS Corrected Velocity

Correction of velocity by the GPS was necessary because the motor was not able to steadily maintain the desired velocity. The motor was supposed to have been capable of maintaining a desired speed at up to 95% accuracy. Further tests concluded that the motor was not able to meet the 95% accuracy it claimed. As recommended by Volvo that the navigational system be built with the aim of reaching the highest possible accuracy, it was necessary to use GPS Data to correct the velocity. Figure 25 is a plot of velocity against time when GPS velocity correction is not enabled

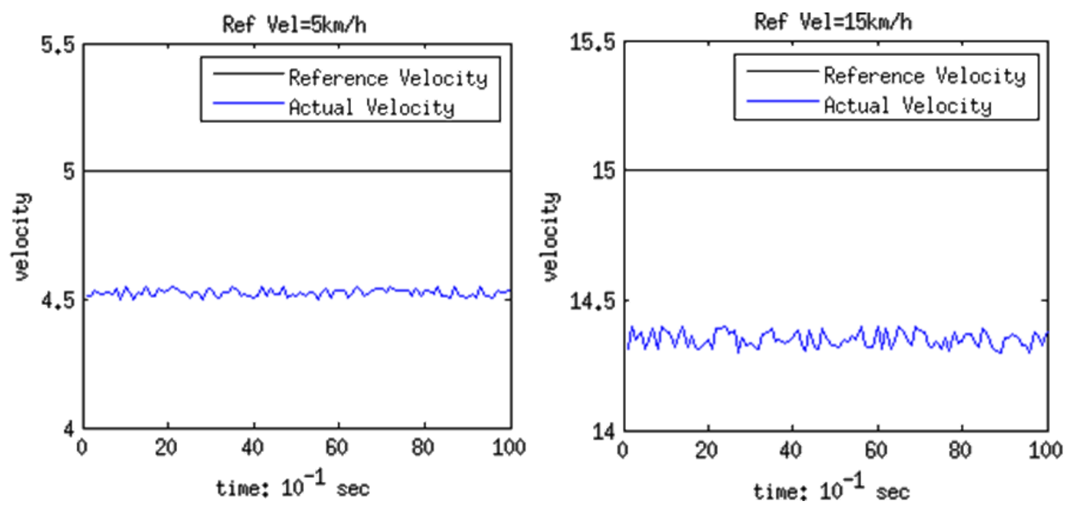


Figure 25: Velocity vs. Time (No GPS Correction) @reference velocity 5 km/h (Left) & 15 km/h (Right).

Reference Velocity 5 km/h and 15km/h shown in Figure 25 above have in common that:

- The Motor is unable to meet the reference velocity. This is probably due to the fact that its controller is setup for lesser torque than present due to the weight of the soft car
- The velocity is inaccurate due to the offset but the precision is high

Considering the GPS Velocity to be the measurement reference, the imprecision may be caused due to track condition or other mechanical factors. An attempt to reduce the offset was made by considering the implementation of a PID Controller to achieve better results.

A PID controller continuously calculates an "error value" as the difference between a measured process variable and a desired set-point. The controller attempts to minimize the error over time by adjustment of a control variable to a new value determined by a weighted sum:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt}$$

where *e* is the error at any given time instant. K_p, K_i and K_d are the proportionality, integral and differential gains respectively which are used as weights for the error, the error integral and error differential to decrease the error in the process variable. In the case of the system, where the velocity is to be corrected, the gains are weights to the following:

- K_p : Error in velocity
- K_i : Error in distance (time integral of velocity)
- K_d: Error in acceleration

As will be seen in the results below, the K_d (differential gain) was not used in the controller because the PI controller was sufficient to meet the accuracy needs of the soft car system. Thus the PI equation for the velocity correction is

$$u(t) = K_p e_v(t) + K_i(D - V_{ref}t)$$

where

- u(t) is the correction applied to the reference velocity at the time instant t
- K_p is Proportionality Gain
- K_i is Integral Gain
- $E_v(t)$ is Error in velocity at time instant t
- D is the total distance covered
- V_{ref} is the desired velocity
- t is the total time

It is important to state that the motor possessed an internal PID controller. Thus any corrective data fed to the motor was being used by the internal controller. Thus, the velocity graphs shown below may not be similar to standard PID control curved graphs. During the initial stage of testing the PI Controller implementation, the Integral gain was ignored and the proportional gain was used. As shown in Figure 26 below, for a reference velocity of 5km/h, the actual (GPS) velocity seemed to be significantly more accurate compared to the velocity without using GPS Corrections. However, actual velocity also became less precise.

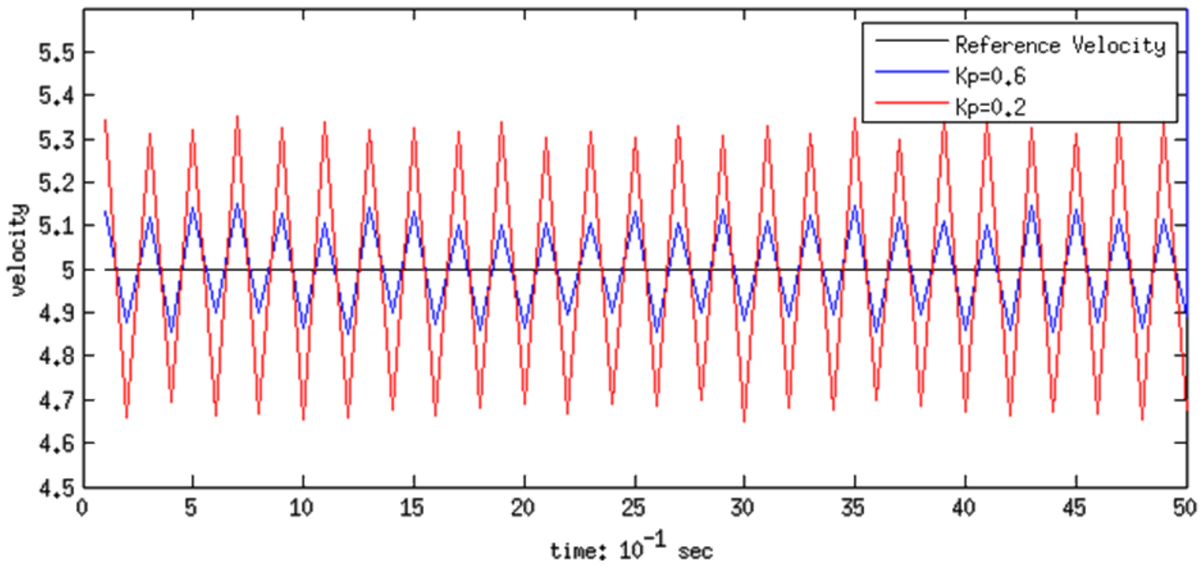


Figure 26: Velocity correction at 5 km/h.

On testing the Proportional Gain with an increased speed (of 15km/h) as shown in Figure 27 below, it may be seen that similar results have been obtained. A Kp (Proportional Gain) value of 0.5 or 0.6 was giving the best results.

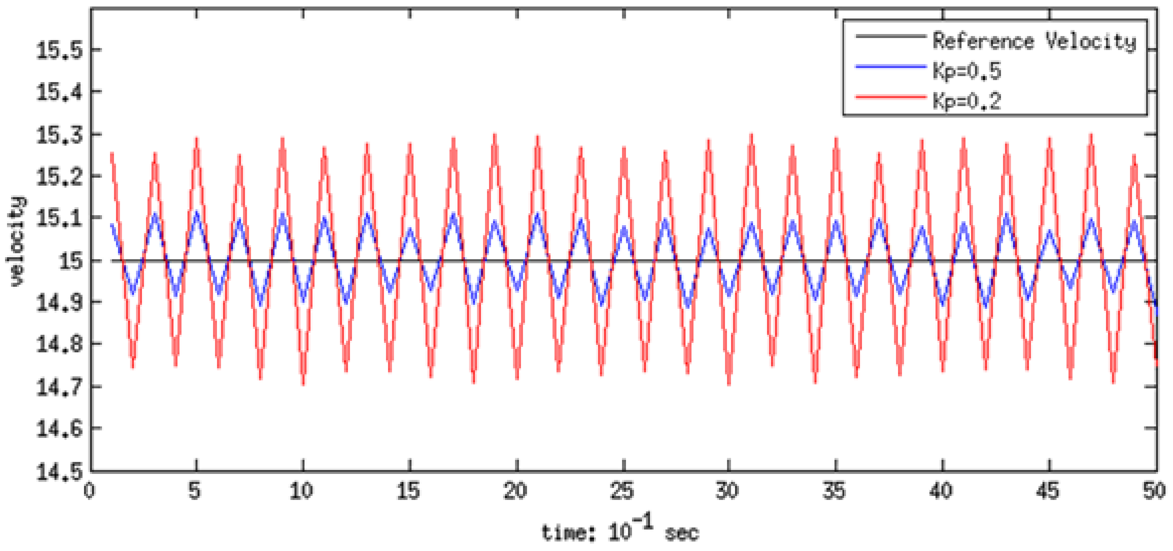


Figure 27: Velocity correction at 15 km/h.

GPS velocity accuracy usually decreases at low velocities (Hegarty, 2005). Thus between the two figures above, it may be seen that the accuracy at 5km/h is marginally lower than at 15 km/h. The most probably cause of this is a lack of GPS accuracy

In order to improve the precision of the actual velocity, the integral gain was also added to correction implementation. As visible in Figure 28 below, the precision improves compared to only applying a proportional gain.

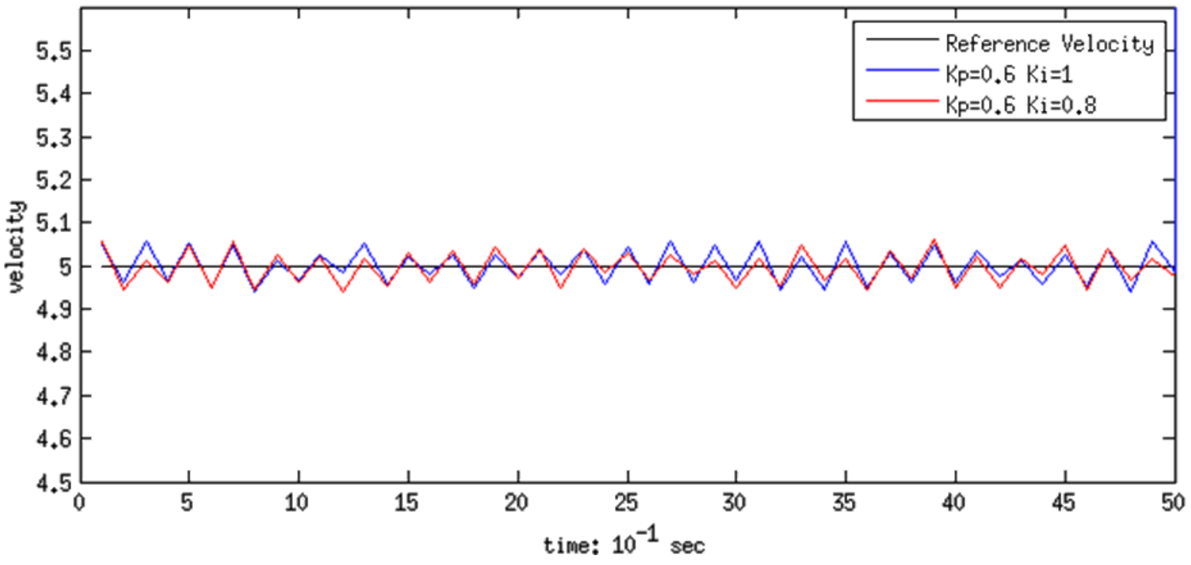


Figure 28: Velocity Correction at 5 km/h.

The 'shape' of the velocity wave remains the same due to the motor processing changes in requested Velocity (via CAN messages) only at a limited frequency. Seen in the figure above, the velocity is accurate to up to 0.1 km/h at a low velocity of 5 km/h. This level of accuracy was deemed suited to the needs of the soft car. Seen in Figure 29 below is the difference of precision obtained in applying an integral gain to the proportional gain for a reference velocity of 15 km/h. It may be seen that there is an increase in accuracy once the integral gain is applied (in the 5th second).

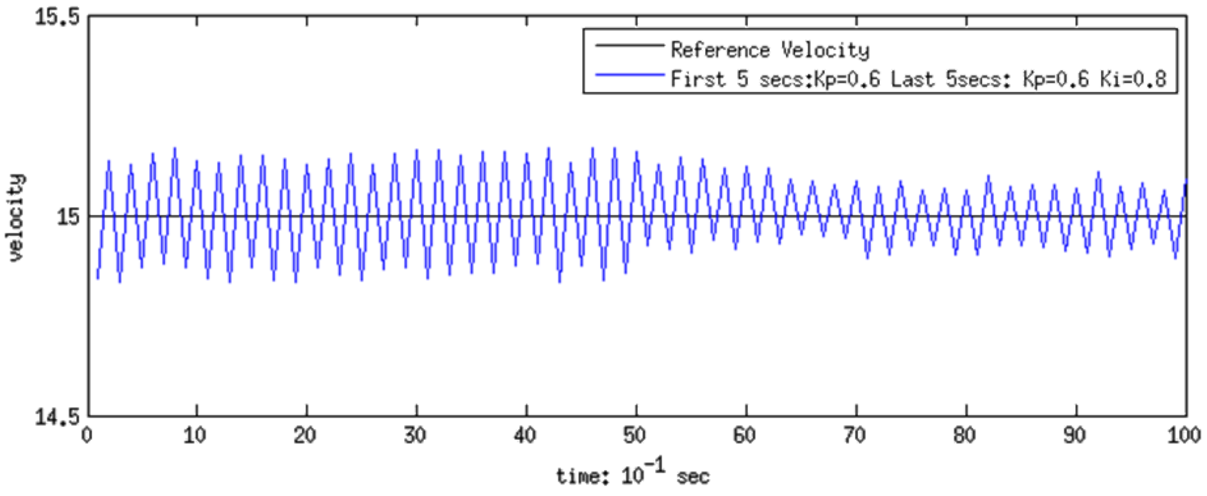


Figure 29: Effect of Integral Gain for Velocity Correction.

During acceleration (between the 4th and 6th second), the precision marginally decreases, as visible in Figure 30 below.

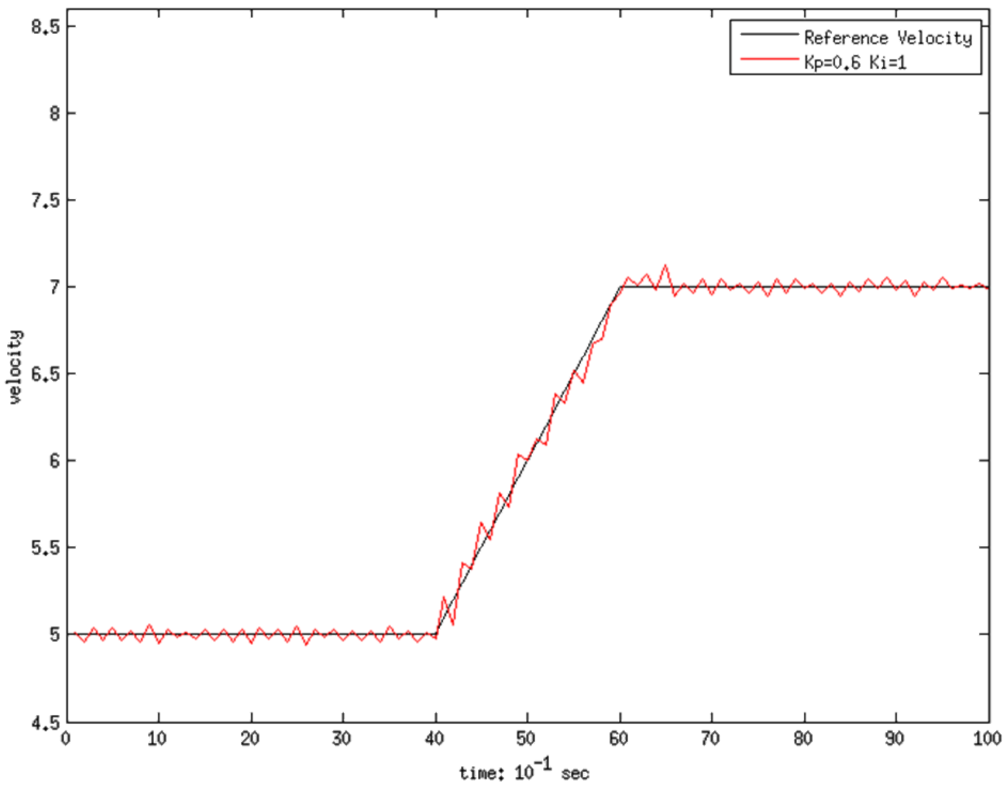


Figure 30: Velocity Correction during Acceleration.

5.3.2 Heading Correction

Unlike the Motor which was directly connected to the CAN network, the turning actuator is controlled by the Arduino. It is, thus, more difficult to maintain accuracy. Most paths or patterns to be used to such tests involve straight lines. It is important to be able to maintain a straight line which is more difficult than intuitively assumed.

Mechanical limitations, such as limited accuracy of the actuator connected to the wheels, debris, minor bumps (as illustrated in Figure 31) or depressions on the track are some of the reasons that cause a deviation when the soft car pursues a straight line without any correction.

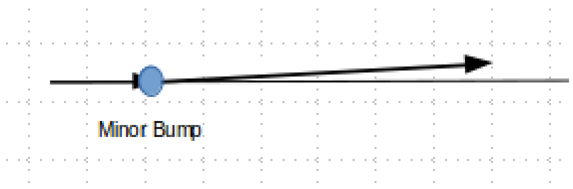


Figure 31: Illustration of a Deflection caused by minor bump.

5.3.2.1 Limitations

One assumption made for correcting the heading via GPS is that the potentiometer value for the centre (Section: Steering (Potentiometer)) is correct and has not been altered by prior testing. It is advisable to recalibrate this potentiometer value after a few tests. The problem arises due to the mechanical limitations of the actuator.

Another problem as mentioned in the section: Limitation arises due to the fact that the Arduino is incapable of maintaining a potentiometer value. During the development stage, a threshold of two or three was used to stop continual fluctuation. The lack of precision occurs due to the limitations of the Arduino Motor Shield.

5.3.2.2 Approach

In order to follow a desired heading, GPS corrections were utilised in the following way:

- A heading threshold or heading tolerance was decided.
- The soft car (midpoint of the line joining the GPS antennas) was allowed to sway to the extent of the heading tolerance (known as θ) and not beyond it.
- The aim was to minimise heading tolerance.

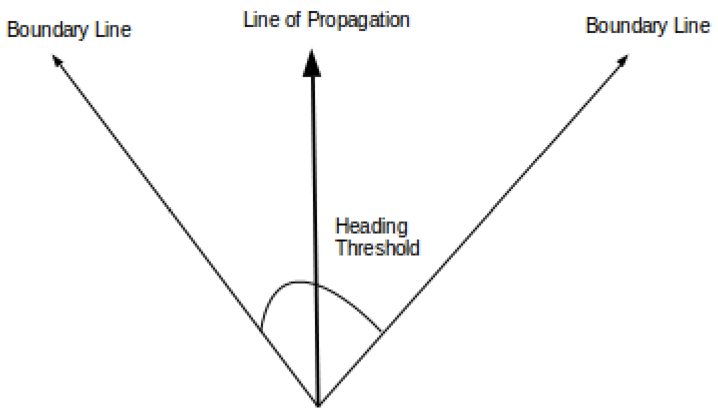


Figure 32: Angle of heading Tolerance.

Figure 32 above illustrates the approach. If the goal is to follow the 'line of propagation', the soft car was allowed to go straight (at centre potentiometer value) until it 'hit' the boundary line. On such an occurrence a correction was applied. This correction is described in the section: Convergence Scheme.

The target was to achieve the lowest heading tolerance possible. The limiting factor was that too much reduction of heading tolerance would cause a divergence, thereby destabilising the straight-line motion.

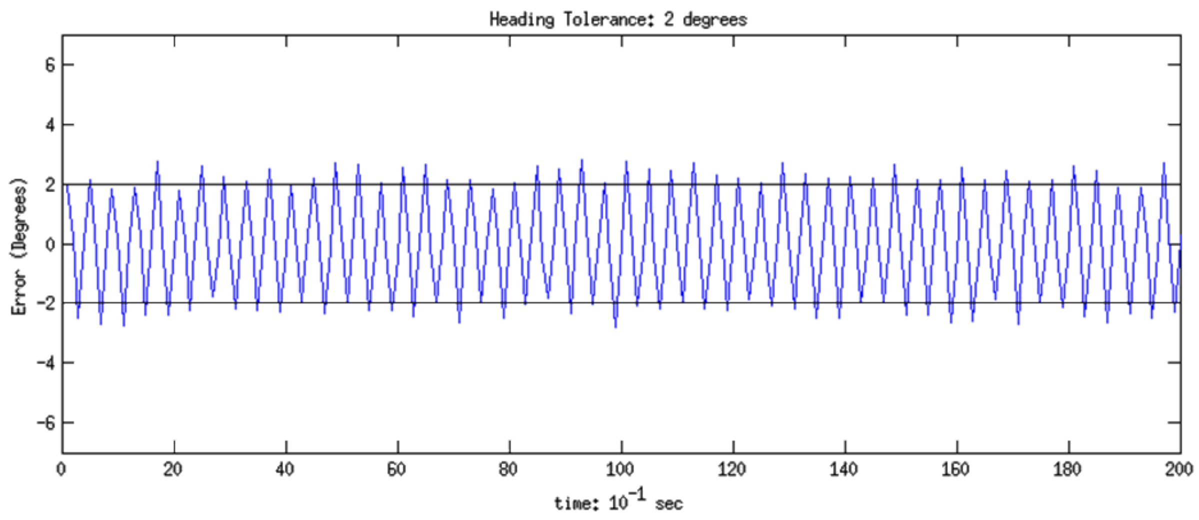


Figure 33: Navigation at Heading Tolerance: 2 degrees.

Figure 33 examines the behaviour of the actuator when it is given a heading tolerance of 2 degrees. As can be seen in the figure, the soft car is sticking within the boundaries conceived by the tolerance angle. The reason why the values are marginally outside the boundaries is because of the inertia of the actuator which propels the actuator slightly farther than the boundary before it reverses direction.

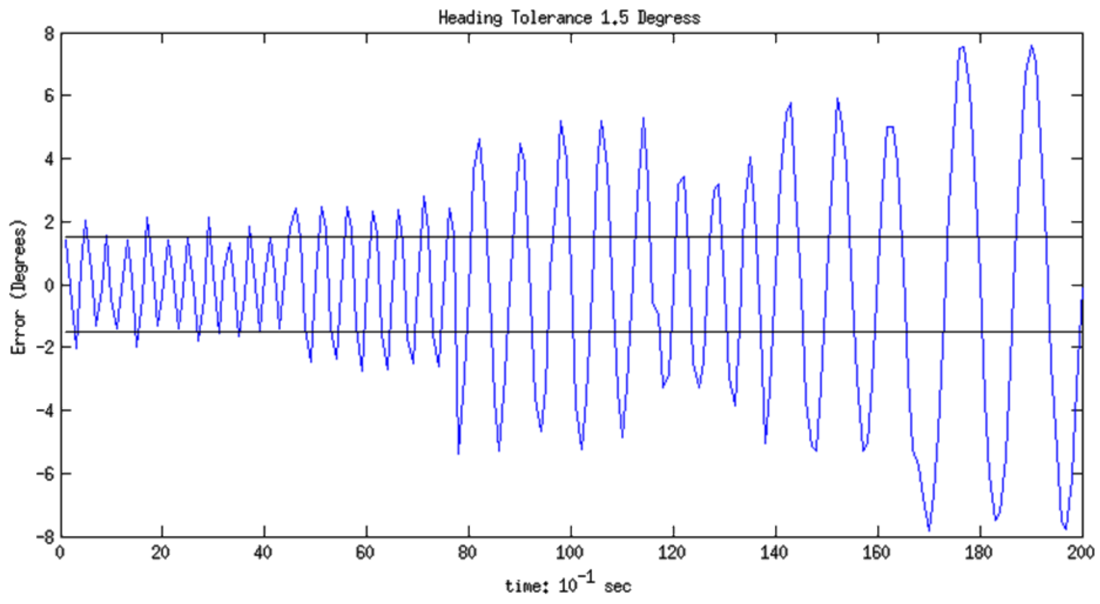


Figure 34: Navigation at Heading Tolerance: 1.5 degrees.

Unlike using a heading tolerance of 2 degrees, in Figure 34 where the tolerance has been lowered to 1.5 degrees does not behave as desired. It may be seen that the soft car performs properly for the first 5 seconds but starts to sway further for the next two or three seconds. The cause of the deviation may be due to minor bumps on the track or it may be because of the mechanical flaws of the actuator.

It is seen that the soft car's performance worsens after 8 seconds before which the corrective function attempts to decrease the angle of oscillation. However, it is visible that the correction function is not good enough to prevent the eventual divergence.

5.3.2.3 Convergence Scheme

In order to develop a robust heading correction system, it is not just necessary to be capable of maintaining a heading but it is also necessary to be able to correct small deviations when required. In this section, an important variable termed as the 'coefficient of potentiometer value' (C_{pv}) is introduced.

In Figure 35 below, it can be seen that there are multiple 'zones' creates based on the heading tolerance (θ). The lines create zones in the heading space which are entitled to specific correction weights.

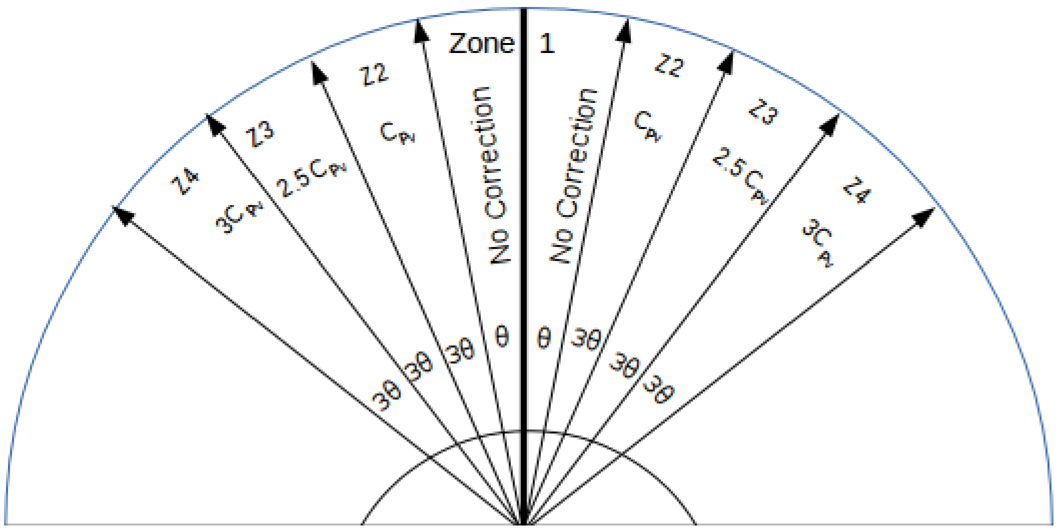


Figure 35: Correction Zones.

The 'coefficient of potentiometer value' (C_{pv}) is the term used to represent the potentiometer value that is added or subtracted from the initial potentiometer value when the soft car 'hits' the Zone 1 boundary. If the soft car tries to pass to the right of Zone 1, a request of a leftward steer of magnitude C_{PV} is sent to the Arduino. Likewise, on trying to go to the left of Zone 1, a request of a rightward steer is made also worth C_{PV}.

In zones 2, 3 and 4 the request made to turn left or right is weighted C_{PV}, $2.5C_{PV}$ and $3C_{PV}$ respectively. The following (Figure 36) is a flowchart that shows how potentiometer calls are made.

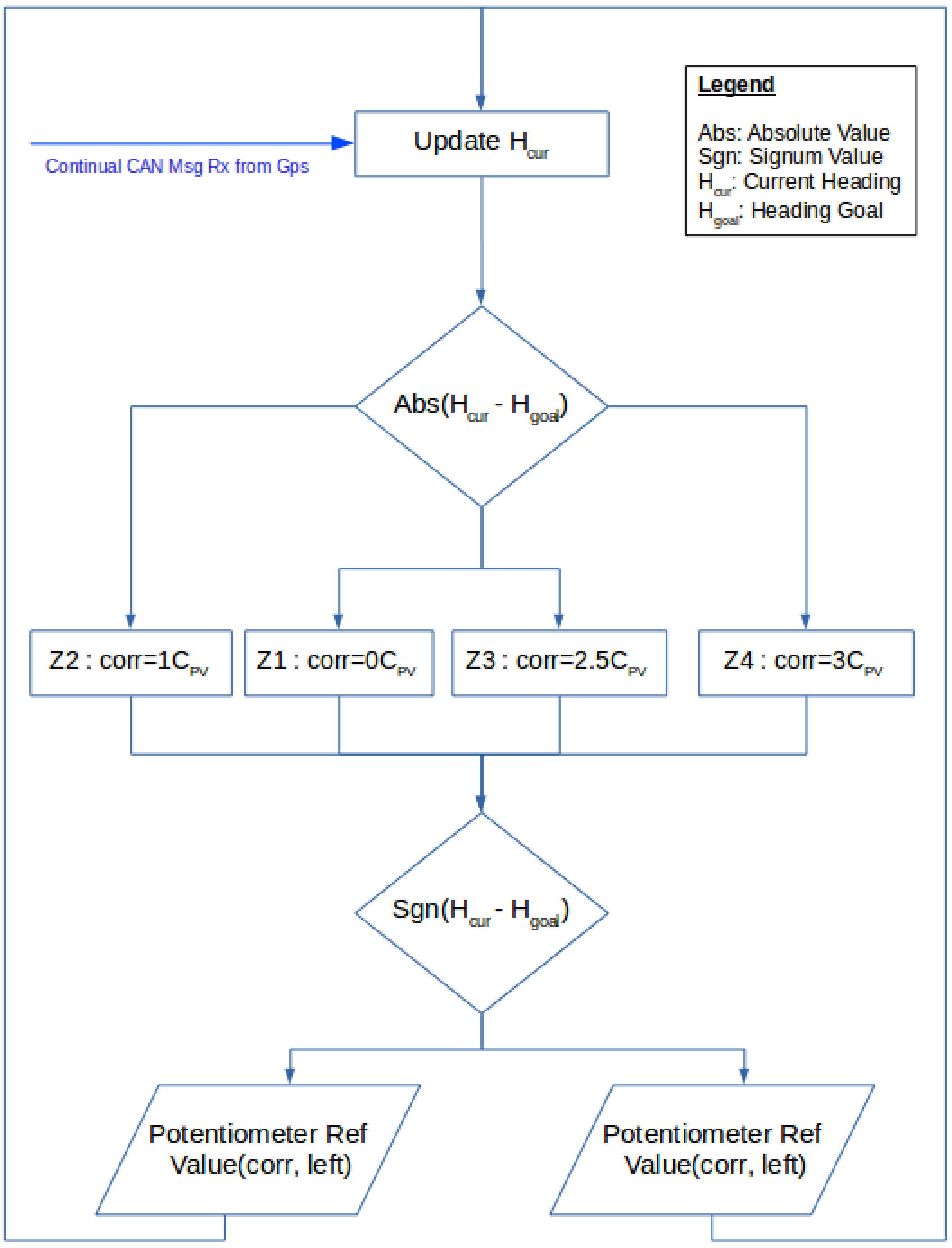


Figure 36: Flowchart representation of Heading Correction.

There is a continuous update of the Heading value. It is compared with the heading goal. Depending on the magnitude of the difference, a factor is multiplied to C_{PV}. Depending on signum value of the difference, a potentiometer value call is made to either the left or to the right

During the development, the following were required to be calibrated

- Coefficient of potentiometer value: C_{pv}
- Angular size of Zone 2, 3 and 4
- Proportional constant multiple to C_{pv} in Zone 2, 3 and 4

The deciding factors for calibration included

- Performance in tests
- Guidelines used for other GPS navigation vehicles (comparable systems)

The calibration was taken places with the following points accounted for:

- The aim is to minimise the time taken to obtain stability (Zone 0)
- On each occasion when the soft car switches between zones, there is a Potentiometer call made to the Arduino through to the actuator. The amount of time needed to 'complete' the potentiometer call varies. If there is a second call made to the Arduino before the completion of the first, the result may be a divergence.
- The smaller the size of the zone, the better the performance
- The difference between the values of CPV in adjacent zones may increase the minimum size of the zones.
- It may be possible but not advised to reduce the tolerance angle to less than 2 but with recalibration[7].

The eventual performance of the heading correction feature is described in Figure 37 below. As can be seen, the soft car aligns with the heading tolerance. It converges within 8 seconds when the initial heading is greater than 15 degrees greater than the desired.

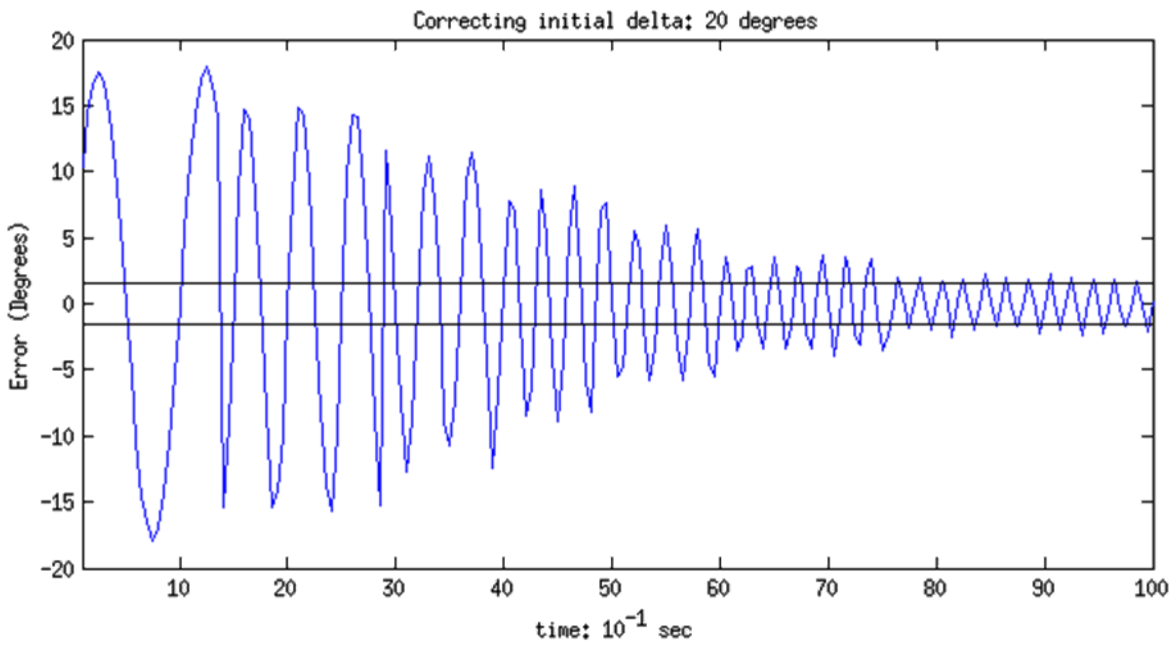


Figure 37: Performance for Heading Correction at higher initial deviation.

5.3.2.4 *Justification of Convergence Scheme over Simple PID Control*

Unlike the velocity correction, a PID control system is not used for heading correction because of weaker performance. Using a proportional gain, where the corrective measure is linearly proportional to the error, the correction was either "sufficient and quick for small angles and Slow

[7] Due to time constraints it was not possible to test it

for greater angles" or "sufficient and quick for greater angles but too fast for greater angles" depending on the value of the proportional Gain as visible in Figure 38.

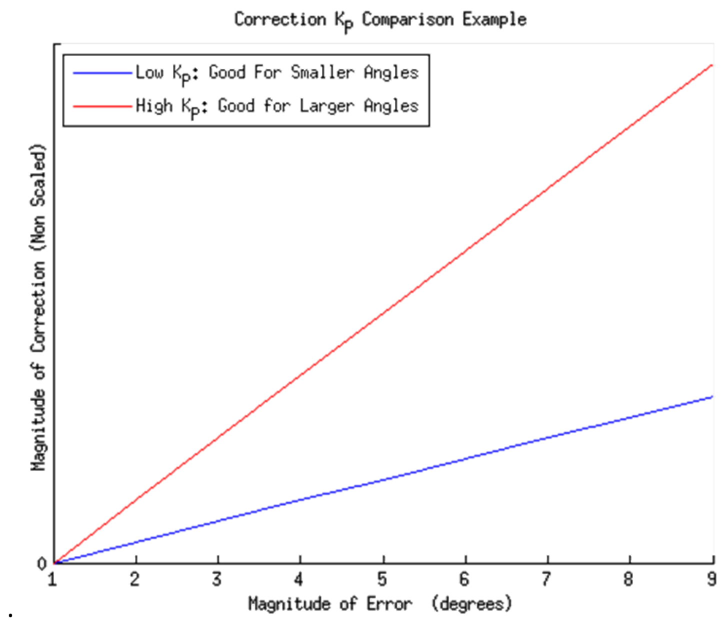


Figure 38: Comparison of multiple Kp Values

The factors causing the anomaly could not be discovered. However, time lags in sending proper correction signals may be a possible cause for this kind of behaviour. The addition of an Integral controller did not help improve that correction system either.

It was then realised that in order to facilitate better correction, it was necessary to use non-linear corrections. The usage of discontinuous functions was considered. The following points were accounted for while attempting the find a function for correction

- Velocity based GPS Heading Data may be slightly fluctuating and absolutely small errors (less than 1 degree should be ignored)
- Due to the limitation of the Arduino board as mentioned in Section: Limitation it may be possible that correction signals do not reach the actuator with a significant lag enough to weaken the performance of the correction scheme
- Sharper Correction (per heading degree) was needed for greater angles than smaller angles (as realised during testing with PI control).

With respect to the points mentioned above, the most intuitive function that could be used was the step (staircase) function. This meant that all the 'zones' in the convergence scheme were radially equally sized. Further calibration was required to derive the size of the zones and the values of the C_{PV} multiplicands for each zone.

One possible justification for the improved performance while using the 'zone' approach is that due to the fact that the Arduino was not requested a change of potentiometer value unless the soft car was changing zones. As mentioned in the section: Convergence Scheme, this meant that there was more time for the Arduino to stabilise.

5.4 Higher Functions

The higher level functions used to make turns and lane changes are most often called directly in the pattern. They may also be called from the user interface. The higher functions in the soft car navigation system are listed below.

- Turning by Delta
- Turning by Absolute Heading
- Lane Change (Time-based)
- Lane Change (Distance-based)

Due to the similarities between the Turning functions, it is explained in one subsection. The time based lane change will be described in this section. However, usage of both, time-based and distance-based lane change will be shown in the pattern.

5.4.1 Turning by Delta/Absolute Heading

Turning calls are made frequently in the patterns used for any testing. At times it may be necessary to turn by a particular angle (delta) and at other times, calls are made to align the soft car to a particular heading. In either case, the difference between the current heading and the goal heading is calculated and based on the magnitude of it, the soft car is either steered towards it before fine-tuning it using heading correction or it is merely corrected using the heading correction function.

5.4.1.1 Illustrative Example

In Figure 39 , the initial heading is 45 degrees and the objective is to reach 37.5 degrees and 300 degrees in terms of absolute heading. The call may also be made using the 'delta' form. The call to turn 7.5 degrees to the left corresponds to obtaining a final heading of 37.5 degrees. Similarly, a request to turn 105 degrees to the left is the same as wanting to achieve a final heading of 300 degrees.

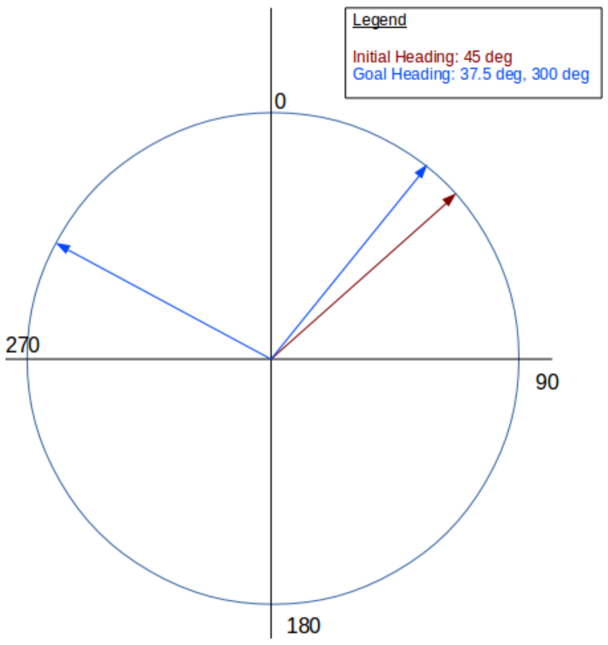


Figure 39: Representation of Initial and goal heading.

Once the ‘delta’ angle has been calculated, the car is steered till it reaches a particular angle (alpha) after which fine tuning is applied. The fine tuning is simply the heading correction applied to achieve the final heading value. Figure 40 shows a relation between the Heading Value and time.

Assuming alpha to be 8 degrees for the above example,

- For a delta of 7.5 degrees and fine tuning is applied directly. The heading would change from 45 degrees to 37.5 degrees in a manner described in section Heading Correction.
- For a delta of 105 degrees the soft car would be steered (to the left) by passing direct potentiometer value data to the Arduino till it reaches the alpha angle (here, 308 degrees: 8 degrees to the right of the goal heading). Reaching 308 degrees, heading correction is applied till the goal heading (of 300 degrees) is obtained.

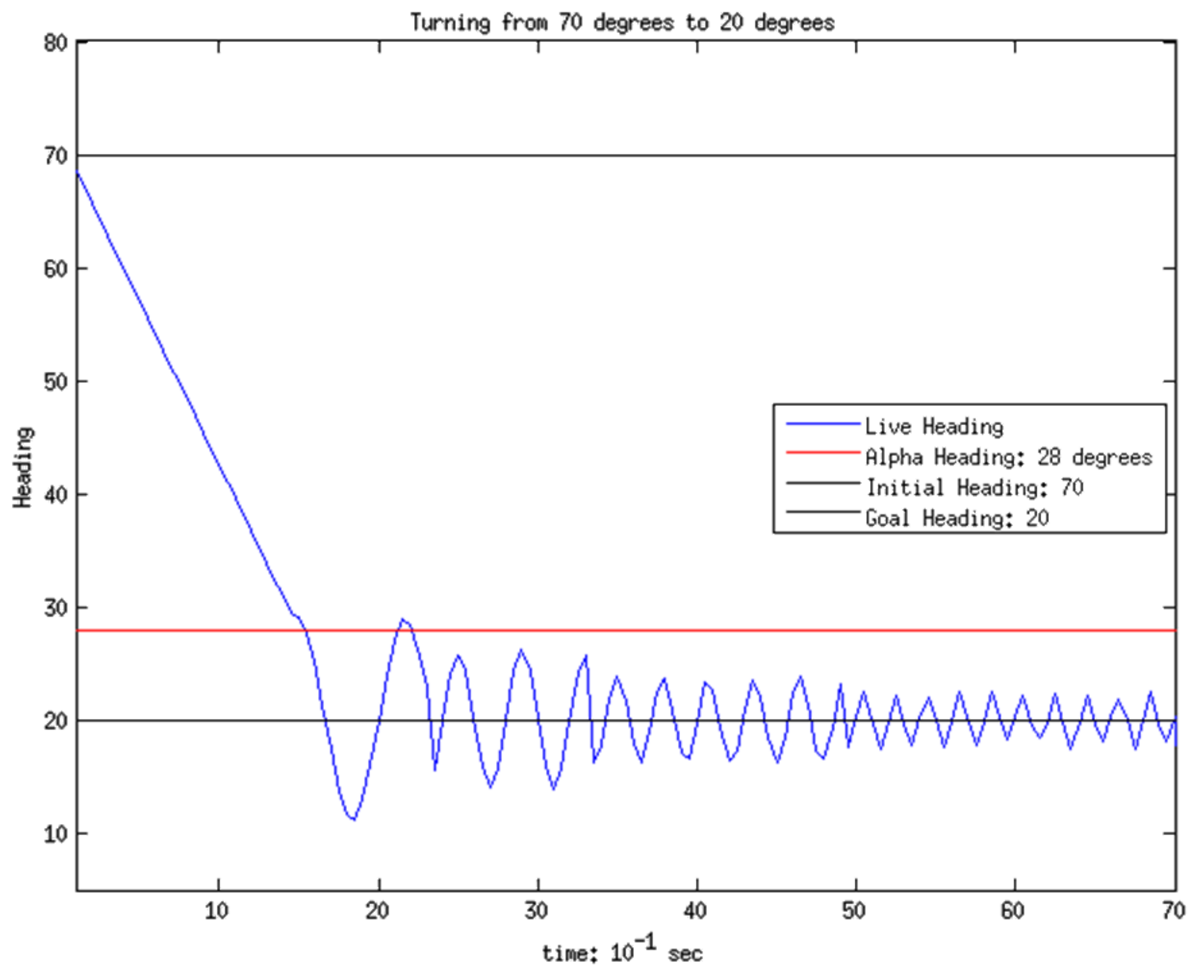


Figure 40: Heading vs. Time (Turning from 70 degrees to 20 degrees).

5.4.1.2 *Calibration of Alpha Angle*

While making any turn, there is continuous calculation of the difference between the soft car’s heading and the goal heading. When this difference (known as the ‘Heading Delta’) is greater than a particular angle, the soft car is steered to the goal heading by passing direct potentiometer value data to the Arduino. When the difference becomes smaller than the angle, fine tuning is applied. This particular angle is known as the alpha angle.

Choosing an alpha angle to make the turn in the fastest way possible can be difficult. The objective of the usage of this angle is to straighten the wheels (centre potentiometer value). Thus, a very small alpha angle may cause the soft car to exceed the goal heading and may cause correction to last quite long. On the other hand, the larger the alpha angle is, the longer it takes to reach the desired goal heading.

Due to lack of facilities to test at higher speeds, it cannot be stated how the alpha angle relates to velocity. For lower speeds (up to 10 km/h), angles of 5 to 8 degrees were found to be most effective.

5.5 Lane Change

The crash tests for which the soft car is intended to be built and developed, often involve lane changes. These lane changes may be based on time or distance. For instance, it may be requested to move to the left (at a given velocity) for 3 seconds and becoming straight again. It may also be asked to turn right till the distance between the initial and the final lane is 80 centimetres. For lane changes, it is necessary to use points and lines in the coordinate system. Figure 41 will illustrate the lane-change function.

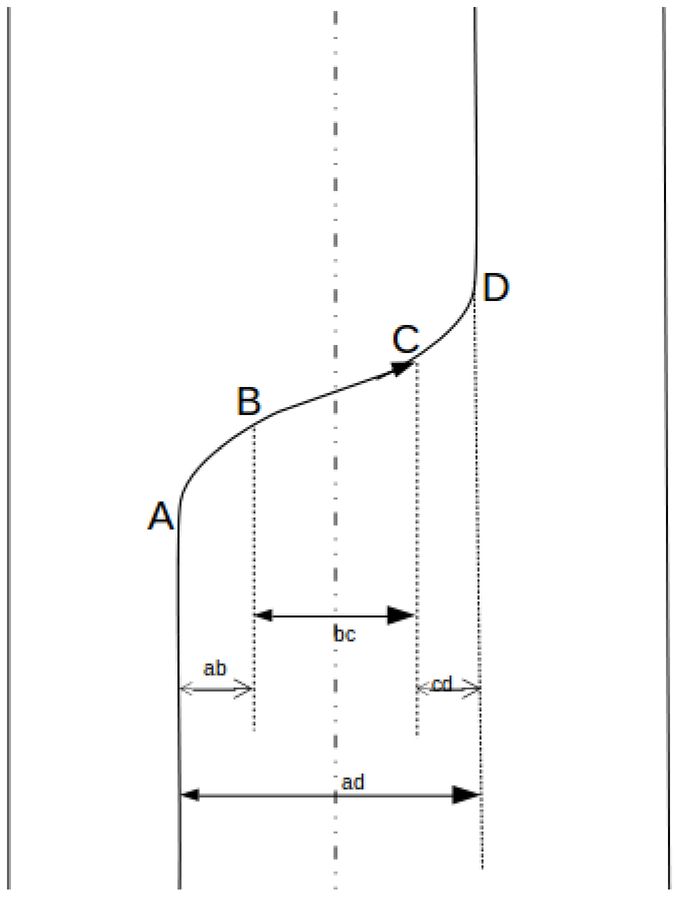


Figure 41: Lane Change Process: Car Direction A→B→C→D.

The figure above shows the lane-change procedure of the soft car. It can be seen that, in order to make a lane change to the right, the following take place,

- A to B: The soft car is asked to steer to the right for a predefined amount of time
- B to C: After the time-interval, the soft car follows a straight line for another time-interval which is may either be the chosen of the user while calling it manually or during a call while tracing the pattern
- C to D: The soft- car steers to the left for the same predefined time as in A to B
- After reaching point D, it is advised to follow the heading using GPS heading correction. The goal heading is the same as that of the lanes i.e. the heading before reaching point A

In this lane change mechanism, the soft-car approaches point B at a constant (but not accurate heading). Since the turn from A to B is the mirror image of that from C to D, the final heading obtained is nearly the same as the initial. Switching on heading correction corrects the heading completely.

It is stated that a time-interval dictates the length between B and C. The vertical distance between the two points may also serve as a variable affecting the lane change.

It may be worth mentioning that because the reference lines (ab, bc, cd) are just used to initiate or stop steering. The path shown in Figure 41 is an illustration of a possible path to be taken by the soft car. The soft car is not passed steering values at every instant to control the exact path during a lane change. There is an ideal 'turning path' (based on speed) which may be calculated during future developments so that an exact path may be measured and steering can be controlled at every instant.

5.5.1 Mechanism

As seen in Figure 42, the lateral distance between A and B (AB) is assumed to be the same as CD. While there is a certain difference in lengths between the two, the assumption has been made because the exact same function is being used on both occasions in opposite directions.

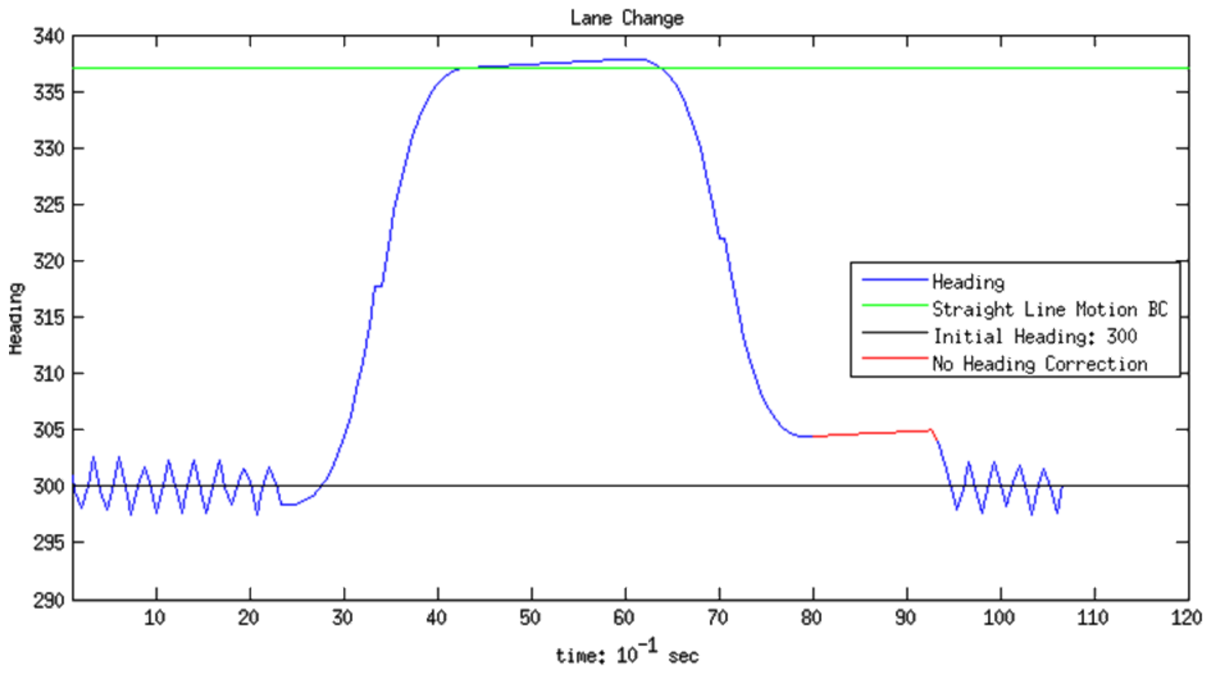


Figure 42: Heading of soft car vs. time during Lane-Change.

While covering AB the soft car steers to the right for the same interval as it steers to the left while covering CD. The distance between B and C may be due to a straight motion of a specific amount of time or by using BC as a variable for the lane-change.

In Figure 42, it is seen that the soft-car is initially travelling with heading correction. At about 2.5 seconds it is asked to make a lane change towards the right. Turning right until about 3.2 seconds the soft car is asked to follow straight once more without any correction. It further takes some time before nearly following a straight line. When the soft car follows a straight line again, it is assumed that point B has been reached.

On ideal testing conditions and with no mechanical limitations, the soft car would have followed a particular heading at its heading in the figure would have coincided with the line marked in green. The length of the straight line, B to C, is dependent on the choice of the user: time or distance ad. After covering the distance, the soft car mirrors the motion from A and B in covering between points C and D. The end of the straight line is marked as point C. The soft car first turns to the left and subsequently straight again with the attempt of regaining the initial heading. However, as seen above, without heading correction, not only has the soft car failed to return to the original heading, it is not even following a straight line.

Minor bumps and depressions on the track might have caused the latter. However, the former, the inability to regain the initial heading is due to the shortcomings of the control system. It is advised to switch on heading correction immediately on reaching point D to follow the initial heading.

In the first case, where the length of BC is time-based, the performance of the lane change may be measured by the final heading of the soft car i.e. the heading of the soft car when it reaches D. A good lane change may characterise of minimal difference in heading between points A and D.

In the second case where the lane change distance is the parameter, not only does the soft car have to ensure that the final heading is close to the initial, it also has to ensure that the distance ad is covered accurately. This is a more complex problem than the time-based lane change.

In order to solve the problem, the following procedure was used

- Vertical distance AB was covered based on a specific interval. Thus distance AB was directly proportional to the speed of the soft car
- The distance AB was used to calculate distance BC on the assumption that AB is the same as CD
- After reaching point D. Heading correction was used most often to compensate for the resulting errors in heading

The performance of the distance based lane change is shown in Figure 43. During a distance-based lane change, even though the soft car attempts to shift laterally by the distance specified by the calling function, there is always an offset due to the poor performance of the control system and/or mechanical limitations.

The offsets shown below are for various lane-change distances for various speeds and directions. It may be seen that soft car velocities at 5, 10 and 15 km/h, making either a left-to-right lane change or a right-to-left lane change, produce varying offsets for various distances.

The y axis in Figure 43 represents the expected lateral shift after the lane change. For instance it is seen that when the soft car is travelling at a velocity of 10 km/h moving from right to left and is expected to make a lane change of 1.2 metres, there is a resultant offset of about 0.4 meters which is an extremely poor performance.

Although the soft car produces better results in most of the other cases, the results shown do not indicate any specific linkage to velocity or lane change distance. However, it appears that offsets produced while changing from a left lane to a right lane are generally positive and the offsets in moving from right to left are negative. It means that in a substantial proportion of the cases, the lane change is exceeding the expected distance.

This data may be utilised to improve the distance based lane change function during future development. A probable cause of offset may be the soft car getting into higher zones (Section: Convergence Scheme) enabling strong oscillations and increasing the possibility of errors.

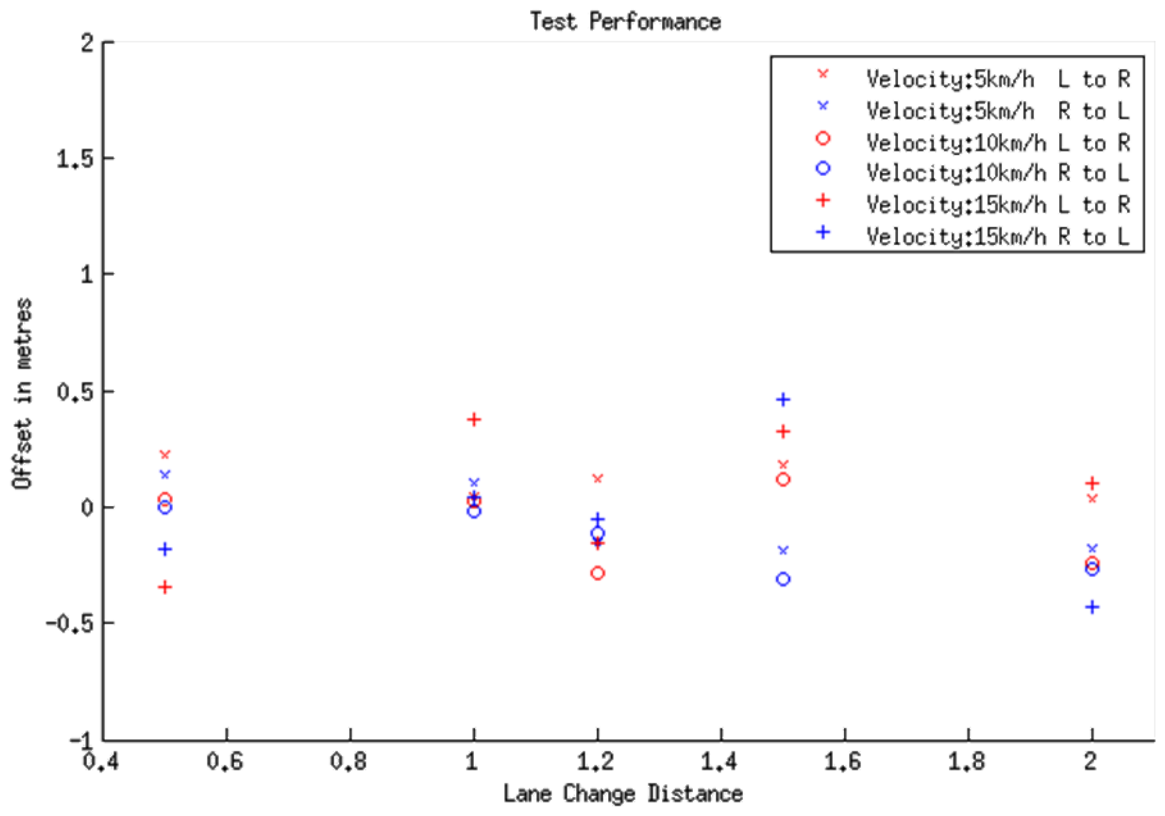


Figure 43: Offset during distance-based Lane Change.

5.5.1.1 *Limitation*

There was limited usage of position feedback due and offsets appeared frequently during a pattern run. Although position feedback via GPS is readily available, it is challenging to implement the 'end' of a lane-change in such a way that the soft-car exactly aligns properly with the expected path-line.

Note that this is much different from aligning in parallel but with an offset (present case) and then slowly correcting offset.

The eventual goal of the soft car is to obtain a perfect lane-change with a proper final alignment. Thus slow-correction of offset was not attempted.

5.6 Pattern

The primary purpose of the soft car is to follow particular pattern repeatedly so that tests may be conducted. The pattern decomposed is a set of time based or distance based function calls. To establish distance based calls, it is necessary to set up reference points and lines which may dictate the calls. The following section describes the establishment of reference lines and points.

5.6.1 Construction of Pattern

On any given test track, a base point in usually required. Most often it marks the starting point of the test or pattern. Points may be placed on the track by real GPS Longitude and Latitude or relative to the base point or other points. Any two points may be used to construct a line. An illustrative example is presented to show the construction of points

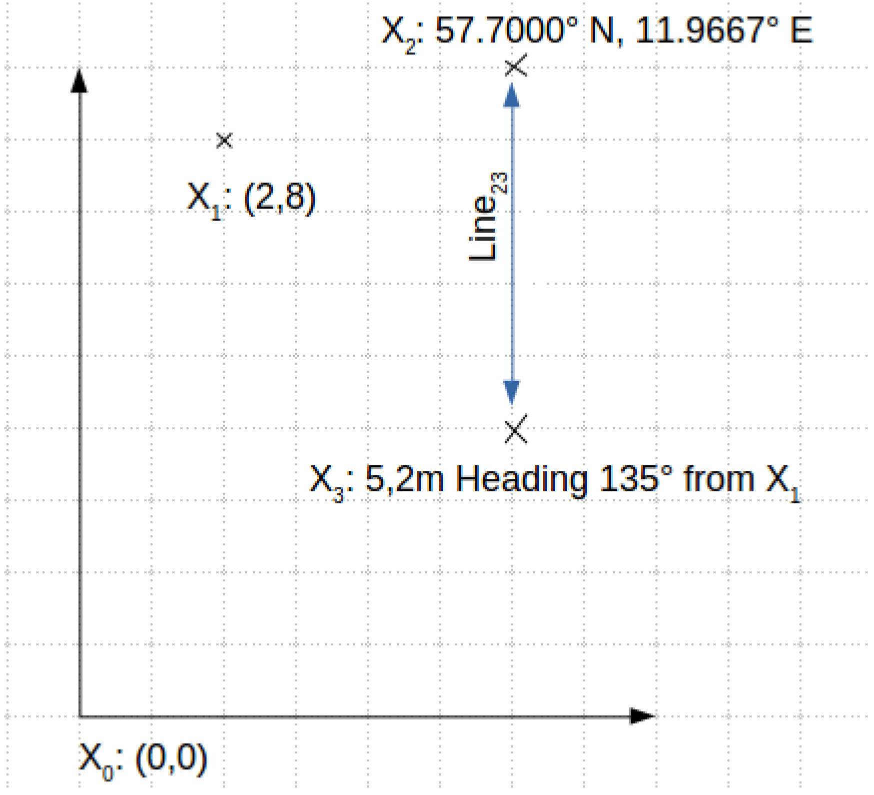

X_2: 57.7000° N, 11.9667° E
X_1: (2,8)
$Line_{23}$
X_3: 5,2m Heading 135° from X_1
X_0: (0,0)

Figure 44: Pattern Construction Using points and lines.

Figure 44 shows three points and one line created and to be used to construct a pattern. It is seen that:

- X_1 is created as a point in the Cartesian coordinate system with X0 as the base point (origin)
- X_2 is created directly using latitude and longitude values
- X_3 is created as a point in the Polar coordinate system with X1 as the base point
- $Line_{23}$ is created using the two points X_2 and X_3

5.6.2 Pattern for Demonstration

The final pattern for demonstration of the workings of the soft car is shown Figure 45 below. It was used to explain the features (especially higher functions) of the soft car and measure the accuracy.

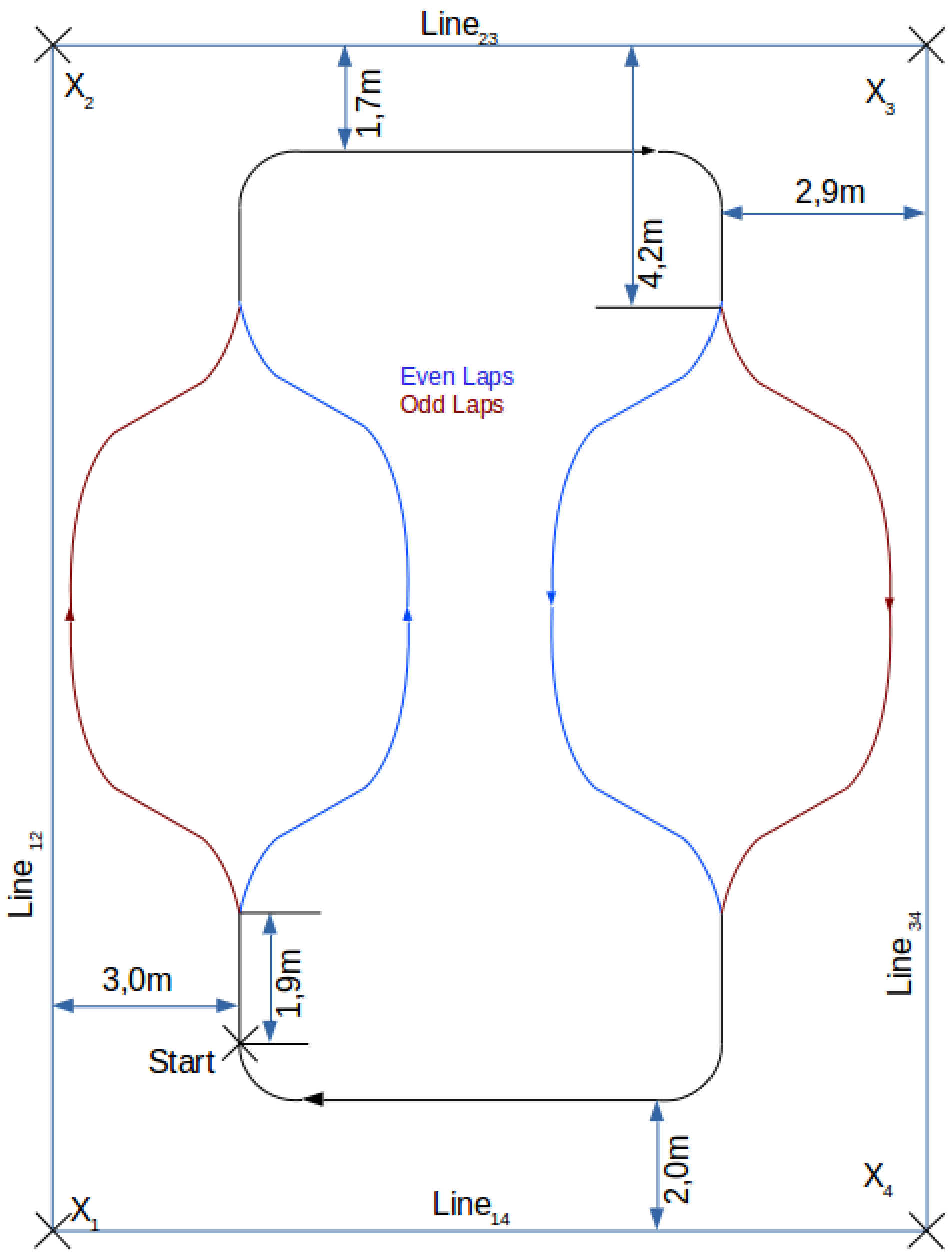


Figure 45: Demonstrative Pattern.

The pattern shown above is constructed using four lines which are generated using four points. The four points X_1, X_2, X_3 and X_4 together form a rectangle whose lines are called as $Line_{12}$, $Line_{23}$, $Line_{34}$ and Line $_{14}$. Another point is used as the starting point of the lap and called X_{start}.

It is visible in the figure above that after starting the lap, the soft car is supposed to make a lane change to the left or right depending on lap being odd or even. It then, comes back to the original lane using the lane change function to come back, following which the soft car turns 90 degrees clockwise two times. After making the turns, the soft car makes the same lane changes as in the previous part of the lap before it turns 90 degrees twice, coming back to the original point. In order to follow this pattern, an algorithm may be written as shown in Figure 46.

```
1. Start Velocity 5 km/h (example), GPS Velocity
   Correction=ON, Heading Correction=ON
2. When distance from Xstart is 1,9m
      a. If Odd Lap: LaneChange (Left,3 seconds)
      b. If Even Lap: LaneChange (Right, 3 seconds)
3. When LaneChange Over
      a. If Odd Lap: LaneChange (Right,3 seconds)
      b. If Even Lap: LaneChange (Left, 3 seconds)
4. When distance from Line23 is 2,7m TurnDelta(90 degrees,
   right)
5. When distance from Line34 is 3,9m TurnDelta(90 degrees,
   right)
6. When distance from Line13 is 4,2m
      a. If Odd Lap: LaneChange (Left,3 seconds)
      b. If Even Lap: LaneChange (Right, 3 seconds)
7. When LaneChange Over
      a. If Odd Lap: LaneChange (Right,3 seconds)
      b. If Even Lap: LaneChange (Left, 3 seconds)
8. When distance from Line14 is 3,0m TurnDelta(90 degrees,
   right)
9. When distance from Line12 is 4,0m TurnDelta(90 degrees,
   right)
```

Figure 46: Algorithm to follow a pattern.

In the above algorithm, LaneChange and TurnDelta are higher functions described in section: Higher Functions.

6 Results

In order to verify the performance of the soft car, a course of two laps of the pattern shown in section: Pattern for Demonstration, were driven by the soft car and the offset to the starting point (X_{start}) was measured. This test would measure the aggregate performance of the soft car and along with aggregate losses of accuracy.

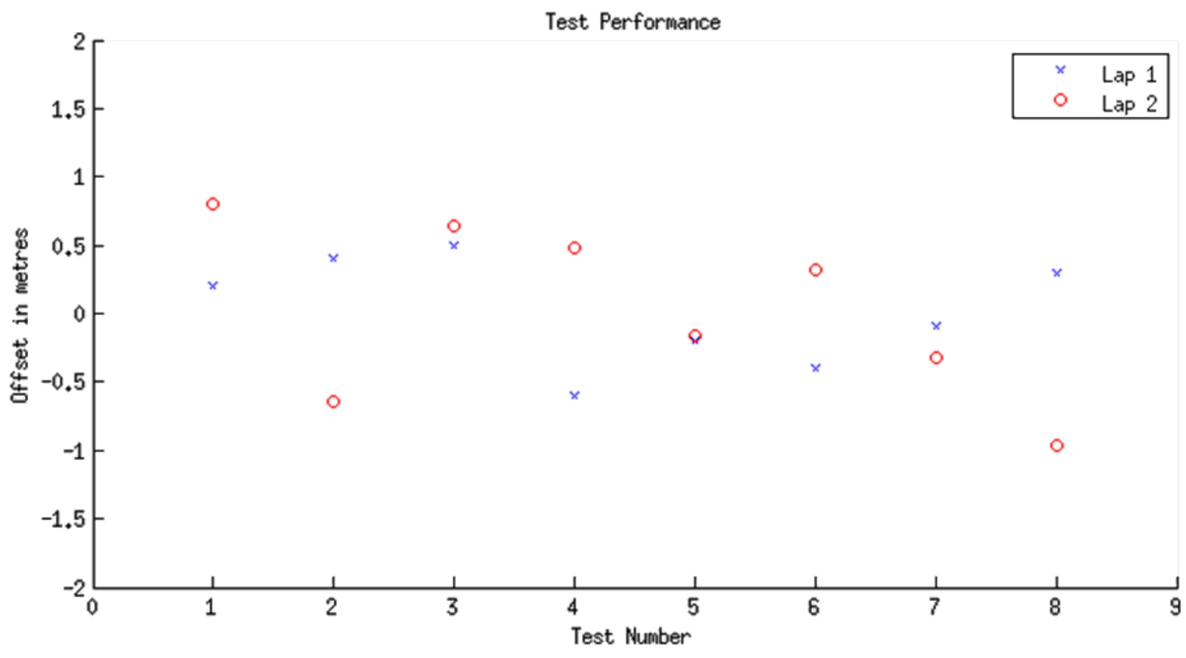


Figure 47: Offset from starting point.

Seen in Figure 47 above, there is considerable offset when the soft car completes a lap or two of the circuit. In eight rounds of testing, the offset was as much as 1 metre which was deemed unsuitable for Volvo's testing purposes without further development. It is also visible that the offset increases in the second lap which seems intuitive as the inaccuracies accrue.

Various mechanical constraints may be responsible for the lack of accuracies. One of the biggest deficiencies of the soft car was its inability to handle slip. Mechanically, the soft car underwent a significant amount of slip during lane changes and also during regular turns. Since the GPS reading of the slip was not always available, it was not used in the source code.

Figure 48 shows how the heading varies over a single lap. The lap is an 'even' lap as the soft car is first performing a left-to-right lane change and then attempting to come back to the initial lane. It is seen that right at the beginning the heading correction is switched on and the soft car is going straight for the first three seconds. It then steers to the right marking the beginning of the first lane change. As described in section Lane Change, the procedure consists of steering right and then going straight for a short span of time before turning left again.

After the soft car completes the lane change it is visible that the soft car is in zone 2. This is a decent lane-change and as shown in the figure, the soft car does not take long in aligning itself to the required angle. Immediately after (at about 13 seconds), the soft car performs another lane change, this time heading back to the previous lane. It appears in a mirror image of the first lane change since the direction has been reversed. This lane-change is also a good performance since the soft car stabilises quickly.

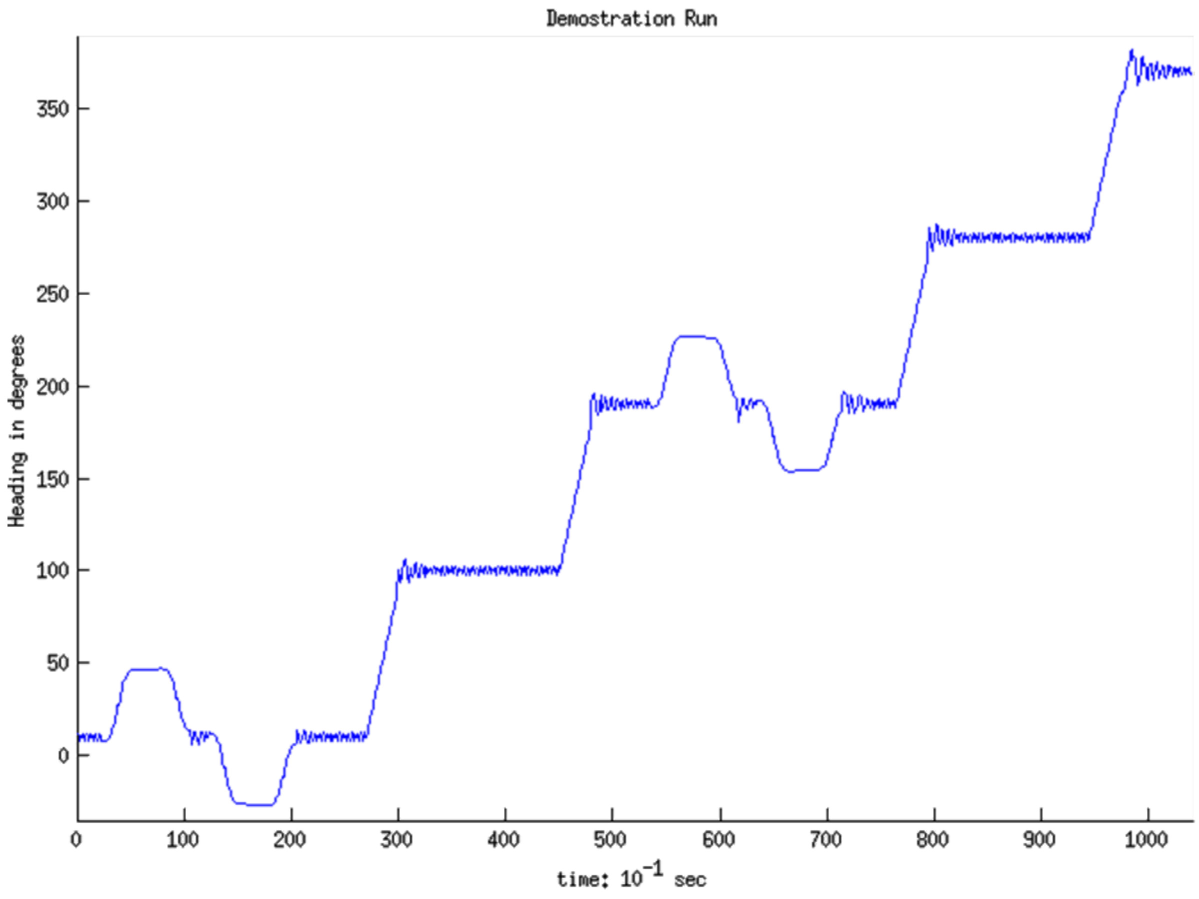


Figure 48: Heading vs. Time (Whole Demonstration Pattern).

At about 27 seconds the soft car turns to the right by 90 degrees. The heading shoots up and it takes about 3 seconds for the heading to increase to the alpha angle (Section: Turning by Delta/Absolute Heading). The soft car has managed to reach zone 3 and it appears that the performance can improve. After moving straight for about 14 seconds, the soft car makes another right turn (at 45 seconds) and this time, the soft car ends up in zone 2: a better performance than the previous turn. The two lane changes are performed subsequently. Between the two lane changes and after the second lane change, the soft car manages to enter zone 2 stabilising quickly.

Finally, the soft car successfully makes the third lane change (starting at about 77 seconds) but the last lane change is a relatively poor performance. It takes a longer time for the soft car to align itself after the final turn.

The consistency of the soft car may be measured by checking which zone the soft car enters after it makes a turn (here, 90 degree turn). As seen in Figure 49, for the four turns, the soft car reaches zone 2 about 36% of the times, zone 3 about 30% of the times, zone 1 about 24% of the times and zone 4 about 10% of the times.

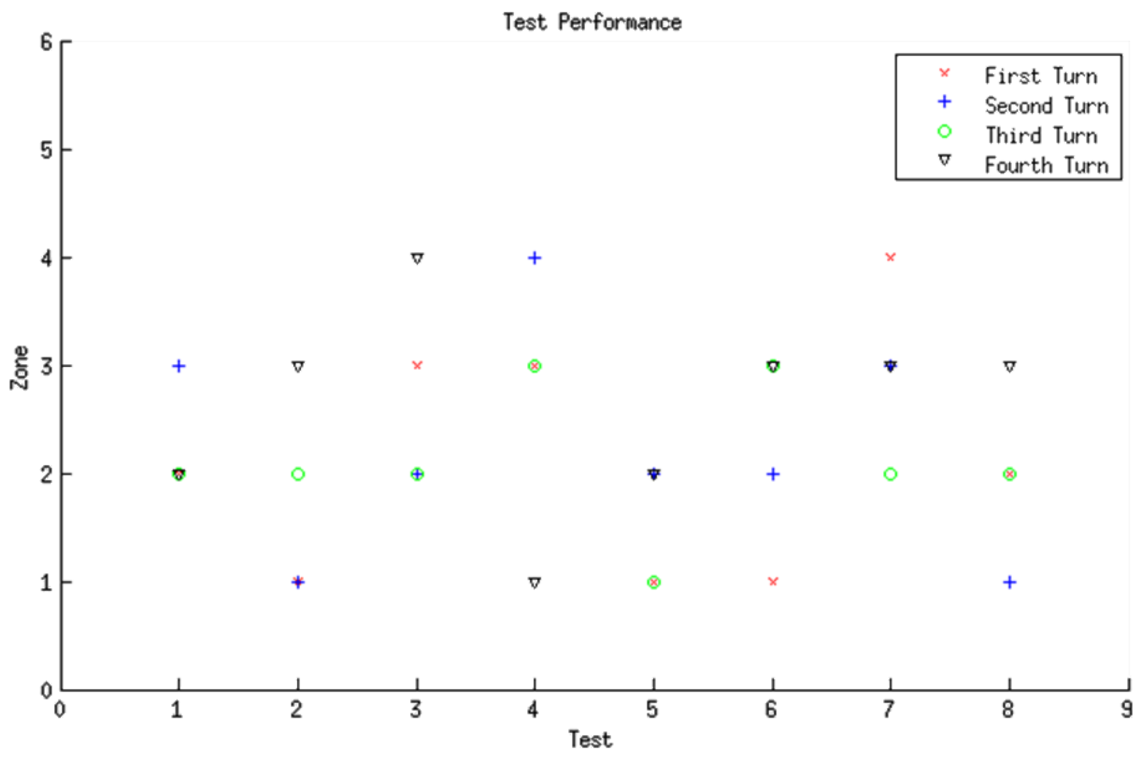


Figure 49: Zone of heading correction after 90 degree turn during demonstration pattern.

This is a clear indicator that zone 4 is reached during exceptional cases. Probable reasons may be bad surface condition or temporary disconnection with GPS satellites. Zone 2 and zone 3 are reached more often than any other and it is also reflected in Figure 48. If the velocity is logged over time for a single lap, it can be seen (as in Figure 50) that the performance is very good. During the start of a lane change process or a turning, there is a slight error in velocity which is immediately corrected. The high performance may probably be attributed to the fact that the PI Control for the velocity does not have much to correct since there is already an internal PID controller in the motor. Secondly, throughout the demonstration pattern, the expected velocity was constant.

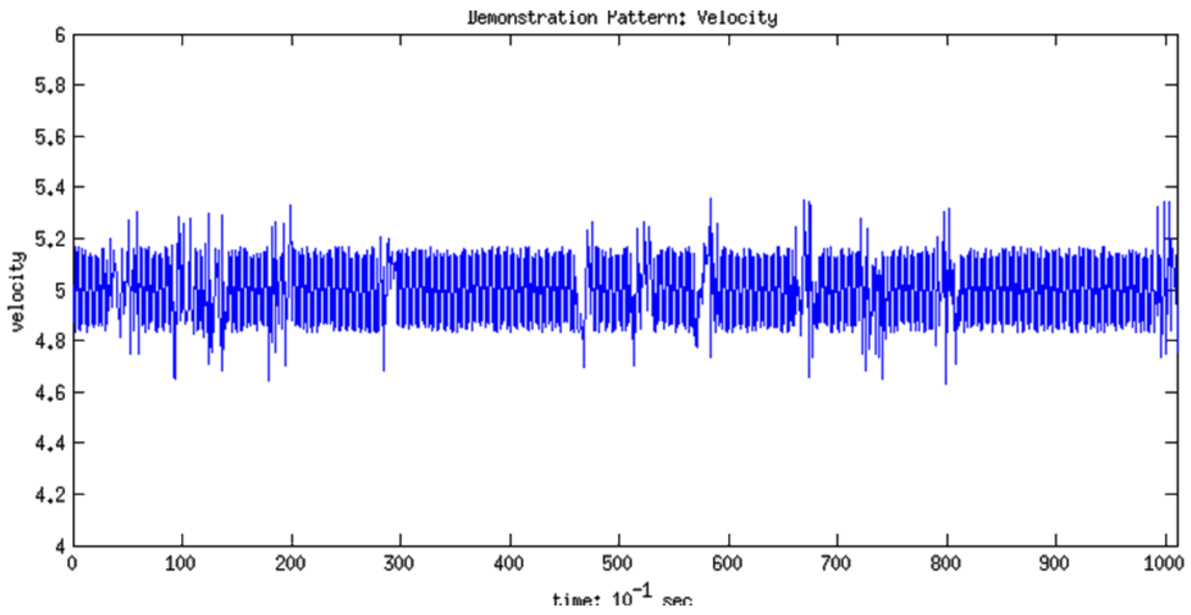


Figure 50: Velocity vs. vs. Time (Whole Demonstration Pattern).

Measuring the offset for the whole course of any lap is difficult because there is no exact path expected to be followed during a lane change. The lines used to construct this pattern (Section: Construction of Pattern) may be used as the reference to measure the position error of the soft car during the pattern demonstration.

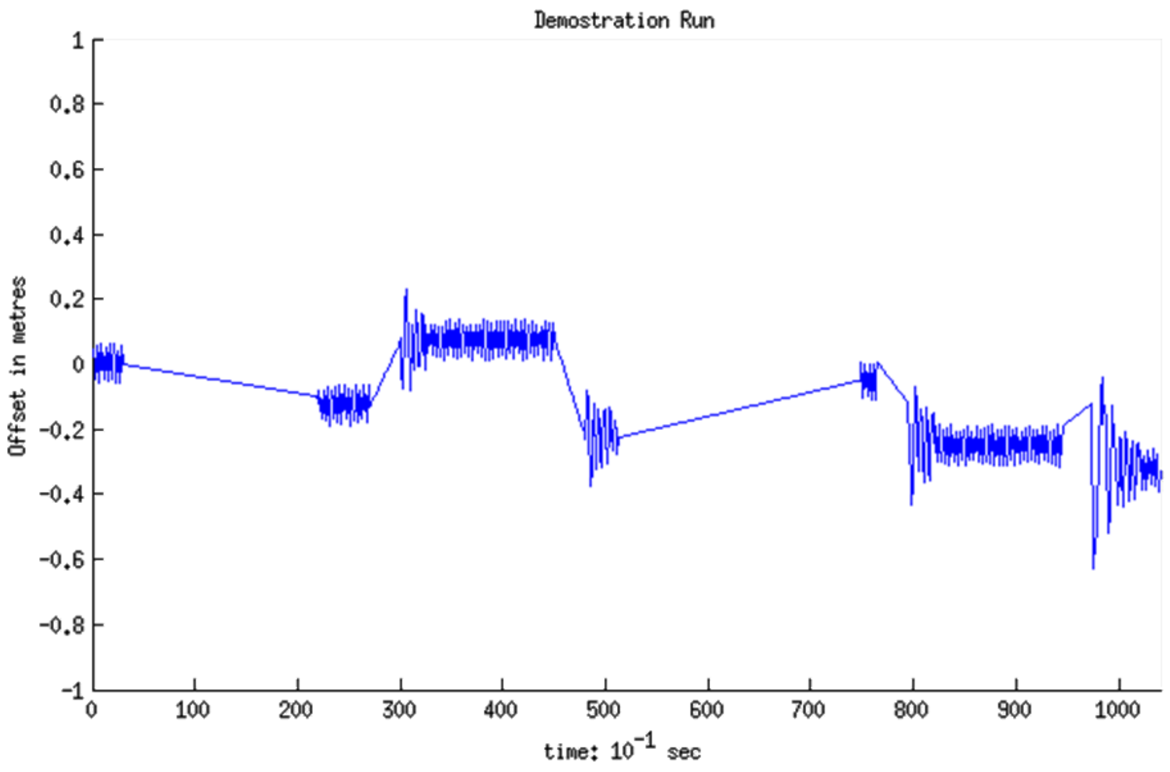


Figure 51: Offset vs. time (Whole demonstration pattern except during Lane-Changes).

Figure 51 shows that position errors have occurred throughout the lap. At the beginning of the lap there is minimal offset and the soft car is oscillating about its expected path. This is probably the best performance possible by the Arduino control. Since the position error cannot be measure during a lane change, the position error is represented in the figure before and after the lane change but not during. In the figure it is seen that after the lane change, the soft car oscillates about an offset of -0.1 metres. Subsequently, it makes a 90 degree turn to the right. It may be intuitive to assume that the turn made the soft car enter zone 3 and the position error has changed. The oscillation now takes place about 0.1 metres. Immediately on completing the 90 degree turn the soft car oscillates with greater amplitude eventually stabilising, as shown in the figure. The offset changes in a similar manner after the second right turn and thereafter, the oscillation takes place about -0.2 metres.

After the second pair of lane changes, the soft car incidentally oscillates about its expected path. The occurring of such an event is merely errors crossing themselves out. Similar shifts take place after the third and fourth turns after which the soft car oscillates about -0.2 and -0.3 metres respectively.

After the fourth turn, it is seen that the soft car has reached zone 4. Thus there is a very large oscillation. One way to measure the performance of the soft car over multiple laps is to track the position offset at any particular point. In Figure 52, the position error after the first pair of lane changes is measured and compared. These points represent the average position offset between the 24th and the 29th second in Figure 51.

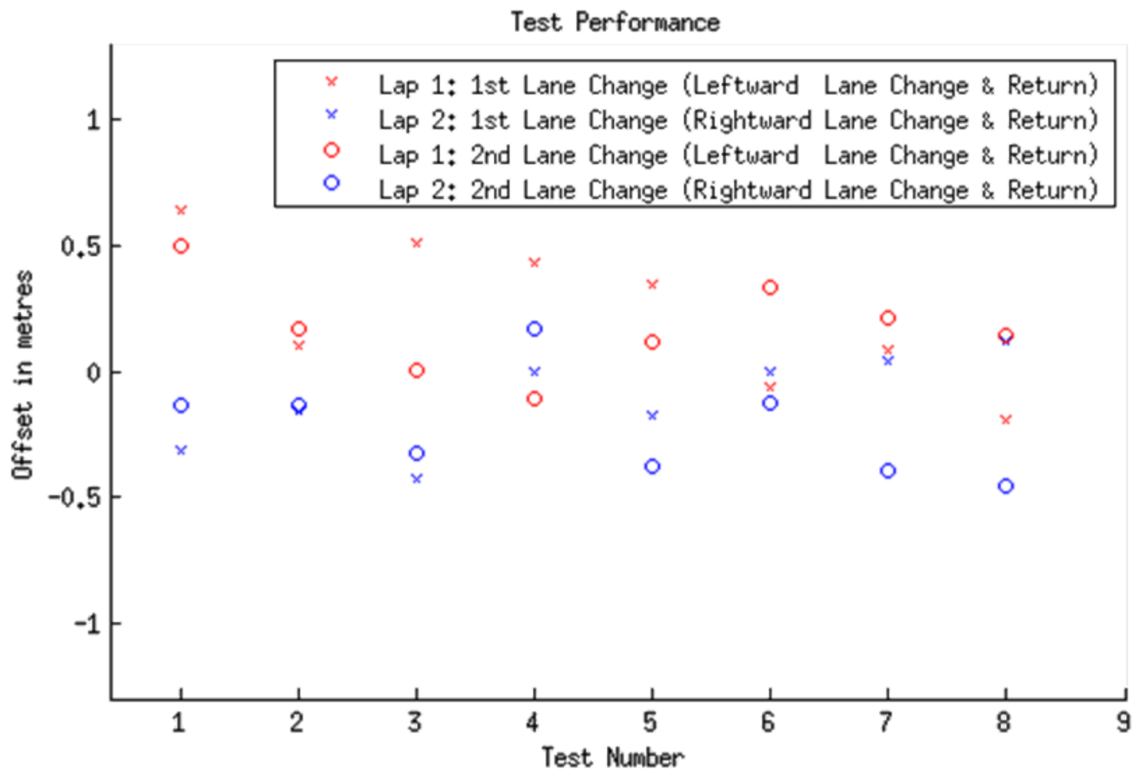


Figure 52: Comparison of post lane-change offset for various speeds, direction and lane-change distance.

It may be seen that there is a relation between the post double lane-change offset and whether it is an even or odd lap. In even laps the soft car first changes to a right lane and subsequently move back to its initial lane. As seen in the figure, this usually causes a negative offset. Likewise, an off lap lane change pair causes a positive offset. It cannot be stated with certainty for either case how much of the error is being added by the first lane change and the proportion of it being caused by the second. The data may be used to improve the control mechanism during future development. The position errors are similar for the first pair of lane changes and the second.

7 Conclusion

This work may be regarded as an initial block for future development of the soft car. Developing a soft car control system usually takes a longer time period. Due to time lost in the initial part of the work, a number of control parameters could not be successfully tested and thus were not implemented.

For instance, there was limited usage of position feedback and offsets appeared frequently during a pattern run as mentioned in Section: Limitation. Future development may require better calibration of the current mechanism but also making usage of additional variables (e.g. velocity) to improve the lane-change.

In the lane-change mechanism, it is also important to keep track of the path of the lane-change. At present, only the final position offset and the heading error are being measured. Creation of a specific reference path is important not just in order to measure the positional offset of the soft car at every time instant but also for enabling other kinds of pattern construction methods (e.g. 'DriveFiles' based on successive X and Y locations (longitude and latitude) with respect to time).

Tyre forces (and its non-linear characteristics) were not accounted for during the testing period and it may be a reason for the failure of PID controller usage for heading correction. PID controllers may provide better heading corrections if applied but it is necessary to alter the gain values based on errors.

The following sub-section summarises the limitations of the work and few recommendations for further development are presented.

7.1 Limitations and Further Development

During the development stage of the navigation system, a number of limitations of the soft car had to be accounted for. The various limitations are explored in this section and recommendations for further development of the soft car are presented.

Due to problems with dual-antenna systems, slip angle could not be measured accounting for loss of accuracy. Before proceeding with further development, there should be a thorough inspection of the Dual-antennas so that there may be usage of slip angle in correction functions.

Due to a lack of braking power, drive tests could not be made at high speeds. The performance of the navigational control system has not been tested at higher speeds that are used in crash tests. The battery management system of the soft car should be changed so that more reverse current is allowed to be sent back to the battery for improved braking

A weak framework meant that the steering wheels could not hold position after a few tests. It was necessary to recalibrate the value for Centre-Potentiometer Value. The soft car should be inspected for mechanical flaws.

It was necessary for the Arduino board to time-average the potentiometer value over a few milliseconds. This was due to voltage fluctuations. Purchase of newer Actuators would improve the performance.

Valuable information was obtained after post-processing data from tests. However, there was not much previous data from similar vehicles to be used for development. Usage of test results from present study would be beneficial to further development.

8 References

Abbot, Doug. 2006. *Linux for Embedded and Real-time Applications.* Burlington : Newnes, 2006.

Alley, Peter. 2011. *Introductory Microcontroller Programming.* Worcester : s.n., 2011.

Autonomous Driven Car. **Arora, Meeshika, Kochar, Priya and Gandhi, Tanvi. 2013.** 2013.

1991. Bosch.de. *Bosch.* [Online] 1991.

Chandra, Rohit. 2001. *Parallel Programming in OpenMP.* s.l. : Mogran Kaufmann, 2001.

Cleevely, David. 2015. Raspberry Pi. *Raspberry Pi 2 Model B.* [Online] 2015.

2013. CPM90 E-Mobility Product Documentation . Unterföhring : s.n., 2013.

2015. Element14.com. *http://www.element14.com/community/servlet/JiveServlet/showImage/38-19321-219119/pi2.jpg.* [Online] Element14, 2015.

Hegarty, Christopher. 2005. *Understanding GPS: Principles and Applications.* s.l. : Artech House, 2005.

2015. http://www.adigodrives.se/produkter/elmotorer/dc-motorer/. *Adigodrives.se.* [Online] Adigo Drives, 2015.

Hwang, K and Faye, A. 1984. *Computer architecture and parallel processing.* New York : McGraw-Hill, 1984.

Kleine-Budde, Marc. 2012. SocketCAN - The official CAN API of the Linux kernel. 2012.

Larsson, Carl and Molin, Johanna. 2013. *Soft Car Control.* Halmstad : s.n., 2013.

Lawrenz, Wolfhard. 2013. *CAN System Engineering, From Theory to Practical Application.* 2013.

2008. Networks Mania. *https://networksmania.wordpress.com/topics/osi-model/.* [Online] 2008.

Oman, Paul W and Cook, Curtis R. 1990. A taxonomy for programming style. 1990.

2015. Peak Systems. *Peak CAN interface.* [Online] 2015. *https://www.peak-system.com/produktcd/Pdf/English/PCAN-USB_UserMan_eng.pdf*

Postel, J and Reynolds, J. 1983. *Telnet Protocol Specification.* 1983.

2015. RaceLogic VBox Manual. 2015.

Schwebel, Robert, et al. 2009. Kernel.org. *https://www.kernel.org/doc/Documentation/networking/can.txt.* [Online] 2009.

Snider, Jarrod. 2009. *Automatic Steering Methods for Autonomous.* Pittsburgh : s.n., 2009.

2015. Transmotech Actuator Catalogue. 2015.

Wheat, Jefferey, Hiser, Randy and Tucker, Jackie. 2001. *Designing A Wireless Network.* 2001.

2015. **SeeedStudio.**
http://www.seeedstudio.com/depot/images/product/2A%20Motor%20Shield%20.jpg