DESIGN AND DEVELOPMENT OF LINE TRACKING AND PATH RECOGNITION SYSTEM FOR VISION GUIDED AUTONOMOUS VEHICLE

YAP CHOON KIAT

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Mechatronics Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2012

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : Yap Choon Kiat

ID No. : 08UEB04837

Date : 11th May 2012

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"DESIGN AND DEVELOPMENT OF LINE TRACKING AND PATH RECOGNITION SYSTEM FOR VISION GUIDED AUTONOMOUS VEHICLE"** was prepared by **YAP CHOON KIAT** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Mechatronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Dr. Tan Yong Chai

Date : _____

DESIGN AND DEVELOPMENT OF LINE TRACKING AND PATH RECOGNITION SYSTEM FOR VISION GUIDED AUTONOMOUS VEHICLE

ABSTRACT

This report presents the design and development of a line tracking and path recognition system for vision guided autonomous vehicle in two competitions.

The first part of the project aim was to build a smart car to participate in the Freescale Cup 2011 competition. The smart car was built to navigate and travel according to the black line on race track with the guidance of the CMUcam3 image sensor module. Infrared sensors were used to enhance the line tracking capability of the smart car. A PD-controller was designed and tuned to control the servo motor which was used to control the steering of the smart car. The smart car was equipped with a Bluetooth serial port plug to establish wireless communication with the computer and the data was transmitted and received via the Hyper Terminal Software. The constructed smart car was able to travel swiftly on the race track with excellent implementation of the line detection and the steering control modules.

As for the second part, the goal was to design an Autonomous Demand Responsive Transit (ADRT) system for the Innovate Malaysia 2012 competition. The path recognition system consisted of road tracking and route identification functions as the core component of this ADRT system. The road tracking function was achieved by using the CMUcam3 and the route identification was accomplished by using a RFID reader to detect and read the Unique ID of the RFID tags that were placed on every junction on the road. With the utilization of National Instruments (NI) Single Board RIO (sbRIO) and LabVIEW, the autonomous vehicle was partially completed and recommendations were given to improve the system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

K_p	Proportional Constant
K_d	Derivative Constant
ASCII	American Standard Code for Information Interchange
bps	bits per second
CCD	Charged-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
DC	Direct Current
DRT	Demand Responsive Transit
FPGA	Field Programmable Gate Array
FRT	Fixed-Route Transit
GPIO	General Purpose Input Output
GUI	Graphical User Interface
IC	Integrated Circuit
LED	Light Emitting Diode
LRT	Light Railway Transit
PID	Proportional Integral Derivative
PLL	Phase-Locked Loop
PWM	Pulse Width Modulation
RFID	Radio-Frequency Identification
RGB	Red Green Blue
SCI	Serial Communication Interface
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receivers Transmitter
VI	Virtual Instrument
VISA	Virtual Instrument Software Architecture

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

1.1.1 The Freescale Cup 2011

The Freescale Cup (previously known as Smart Car Competition) is a high speed, autonomous, race of RC scale cars competition. The race is part of Freescale's University Programs to promote engineering in the classroom. It is open to all local and private universities in Malaysia. Each team has to come up with a maximum of 3 students plus 1 lecturer as advisor.

In 2008, Freescale Semiconductor Malaysia had the fist Smart Car Competition which was successfully held on 12 and 13 December at SIRIM Hall, Shah Alam, Selangor. Due to economic crisis in 2009, 2nd Smart Car Competition was postponed to 2010. The competition was organized in collaboration with IEEE Malaysia Section, Universiti Teknologi MARA (UiTM) and Universiti Kebangsaan Malaysia (UKM). The event was successfully held on 4 and 5 December 2010 at DECTAR Hall, UKM Bangi, Selangor.

In 2011, Smart Car Competition was renamed as The Freescale Cup. The competition has caught the attention of MOHE hence was invited to be part of PECIPTA 2011. The 2011 competition was held in collaboration with MOHE, UiTM and IEEE on 14 and 15 September 2011 at KL Convention Centre (KLCC).

1.1.2 Innovate Malaysia 2012

Innovate Malaysia Design Competition is a multi-discipline engineering design competition open to all undergraduate engineering or computer science students in Malaysia. The goal of the contest is to promote innovation culture and mindsets among university graduates, to enhance knowledge and skill set in practical engineering and promote greater interest in engineering design, and to promote more industry and university collaboration.

Agilent, Altera, Intel, National Instruments, and Silterra as the leaders in the electronic instrument, programmable logic, microprocessor, and IC fabrication respectively, creates an environment of learning through innovation and positive competition by holding a multi-discipline engineering design contest.

1.2 Problem Statements

1.2.1 Line Following Smart Car

The Freescale Cup competition requires students to produce an intelligent racing car that can recognize the track automatically to run on the designed race track. The race track consists of a 3cm wide black line on the 60cm wide raised white background. To elevate the challenge, combinations of hills, tunnels, and sharp turn are included. The race track is unknown to the contestants until the competition day.

A model car chassis with motors is provided by the organizer; limited numbers of sensors are to be installed with freedom to develop additional electronics with the given microcontroller unit. The smart car has to be fully autonomous and the use of remote control is prohibited in the competition.

Speed of the smart car or the time taken to complete the race track is an essential judging criterion of the competition. For this reason, the conventional method adopted to build a line-following robot which picks up the reflected light by using light sensor such as infrared sensor or colour sensor is no longer capable. This is because in order to obtain a reliable line detection result, the sensors have to be placed perpendicularly to the surface of the race track. As a consequence, the path ahead of the smart car is unknown thus the speed is restricted else the smart car might run out of track which will lead to disqualification.

An image sensor is compulsory for the smart car so that the path in front can be seen. The speed limit of the smart car can be boosted to a higher range with the path ahead known because it can react beforehand; this is especially crucial when come to a sharp turn.

1.2.2 Autonomous Demand Responsive Transit (ADRT) System

Public transport remains inadequate was listed as one of the main challenges in the Greater KL report. Despite various Fixed-Route Transit (FRT) services such as light rail transit system, train commuter system and monorail system were provided to the public, the usage between public and private transportation is still low at 16% - 84%. The government aims to improve the situation by setting the target mode split to 25% - 75% in 2012. The problem recognized in the current transport system is due to the poor connectivity in the Demand Responsive Transit (DRT) or Paratransit system.

FRT systems provide predetermined route and schedule, high passenger capacity and consolidation of many passenger trips into a single vehicle. The general public considers them to be inconvenient because of their lack of flexibility. In contrast with the FRT systems, the DRT system provides much of the desired flexibility with door-to-door type of service without following a fixed route or schedules (Quadrifoglio & Li, 2009). DRT system is user-oriented with generally higher cost. For instance, taxi is one of the DRT used to reach a desired destination

where a regular bus service or LRT may not be as viable. The dependency on the DRT system has been increasing over the last few decades especially within residential communities due the steady increase in their population density as a consequence of urban sprawl.

The demand on taxi services is increasing significantly in the urban area. In Malaysia, the requirements to be a taxi driver are as follow:

1. A driving license and many good years of experience in driving.
2. Possess a Public Service Vehicle (PSV) license.
3. Must be in possession of good character and not black listed with the police.
4. Must have the ability to own a vehicle.
5. Knowledge of local and popular place in towns and cities.
6. Able to provide a fair calculation of distances travelled and usage of meters.

These stringent requirements may cause the supply fails to meet the demand in the future. Autonomous DRT system can be used as an alternative for taxi services and to cope with the continuously increasing demand on the DRT system.

1.3 Aims and Objectives

1.3.1 Line Tracking System

The first aim of this project is to build a vision-guided smart car to compete in The Freescale Cup 2011. An image sensor is to be interfaced with the microcontroller unit and served as the “eye” of the smart car to ensure that the path can be followed smoothly at high speed.

The main objective is to enable the smart car to perform line following by using an image sensor. Apart from line detection, to design a precise steering control system for the smart car is an equally important objective as the steering of the smart car plays an important role in following the path smoothly. Other than that, the third objective is to build a wireless monitoring system for the smart car so that its important parameters like speed and sensors data of the smart car can be obtained when it is running on the race track for testing and tuning purpose.

1.3.2 Path Recognition System for ADRT

The second aim of this project is to design and develop an Autonomous Demand Responsive Transit (ADRT) System for Innovate Malaysia 2012 design competition. A small scale prototype will be built to demonstrate and evaluate the ADRT system.

The main objective is to build a path recognition system for the ADRT system. The proposed path recognition system consists of two main functions which are road tracking and route identification. These two functions are to be embedded to an autonomous vehicle which will be used as the agent of ADRT.

Besides, a wireless communication has to be established in between a computer and the autonomous vehicle. This is to provide a linkage for the computer-hosted Graphical User Interface (GUI) to communicate with the vehicle.

1.4 Report Outline

Literature review on existing line-following methods and various types of sensors will be presented in Chapter 2. Apart from that, the different technologies of serial communication and several control method that are widely adopted in line following control will be part of the review as well.

The methodologies in term of building a line following system for the smart car to compete in The Freescale Cup 2011 and developing a path recognition system which is part of the proposed ADRT system for the Innovate Malaysia 2012 design competition are discussed in the third and fourth chapter respectively.

As for Chapter 5 and Chapter 6, it is the results and discussions of both the line following system for smart car and path recognition system for ADRT system. Various diagram and graph will be shown to illustrate the result. At the same time, the result will be discussed.

Chapter 7 which is the conclusion wraps up the outcome of this project for both systems. The result of the project will be compared to the objectives. Other than that, the limitations of this project will be discussed and some recommendations will be given for future improvements.

CHAPTER 2

LITERATURE REVIEW

2.1 Application of Line Following Robot

Line following system has been incorporated in mobile robots so that the robots can travel according to the line autonomously. Ben, Zoran, Tim, Saeid and Philip (2011) introduced the OzTug mobile robot developed to autonomously manoeuvre large loads within a manufacturing environment. The robot is configured to follow a predefined trajectory. Figure 2.1 shows the OzTug mobile robots during initial experimentation.

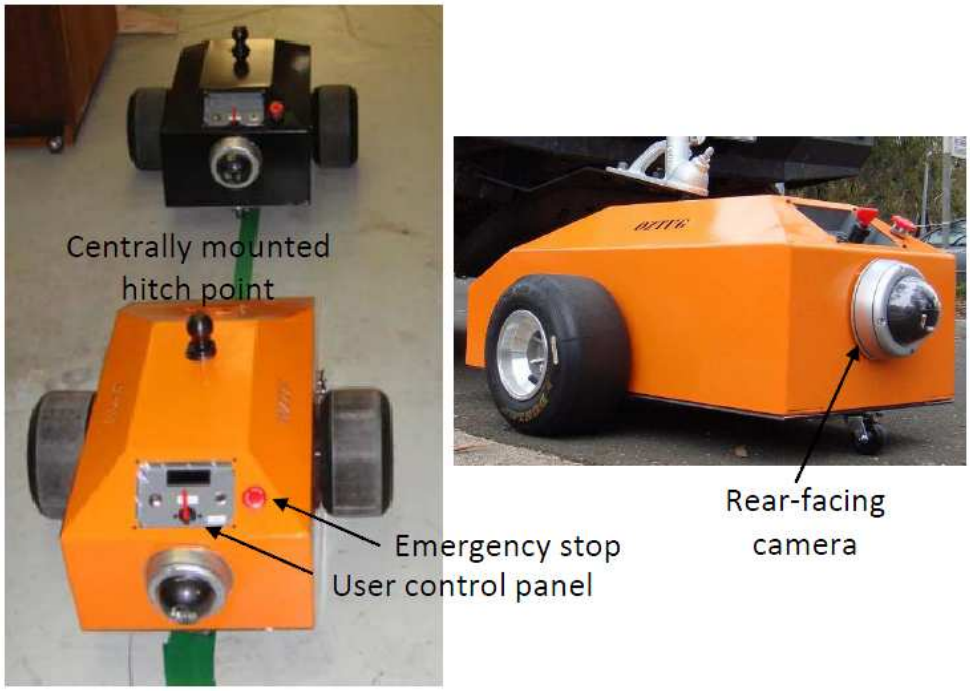


Figure 2.1: OzTug Mobile Robots

Besides manufacturing environment, line following mobile robots also can be deployed as a commercial product. Illnur and Deniz (2009) proposed the design of a line following robot which is commonly used to carry children through shopping mall entertainment place. Figure 2.2 shows the prototype of the line following robot.

Figure 2.2: Prototype of the Line Following Robot

Other than that, Thirumurugan, Vinoth, Kartheeswaran and Vishwanathan (2010) demonstrated the application of line following robot for library inventory management system. In their design, a line following robot is designed using sensor operated motor to keep track the line path predetermined for library book shelf arrangements.

Figure 2.3: Book Searching and Arranging Operation of Line Following Robot

2.2 Existing Line Following Systems

Line following capability can be achieved by a few methods; the most basic method is using the light sensor such as infrared sensor or colour sensor. Figure 2.4 illustrates the application and placement of infrared sensors in the design proposed by Illnur and Deniz (2009).

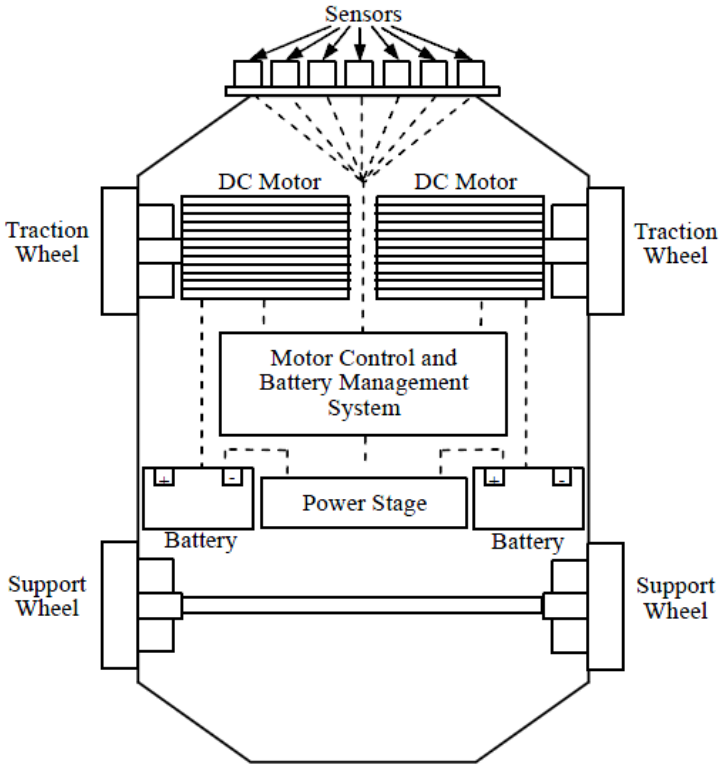


Figure 2.4: Placement of Infrared Sensors

Jean and Marc (2006) presented a webcam-based line following mobile robot equipped with a miniature Linux-based single-board computer as shown in Figure 2.5.

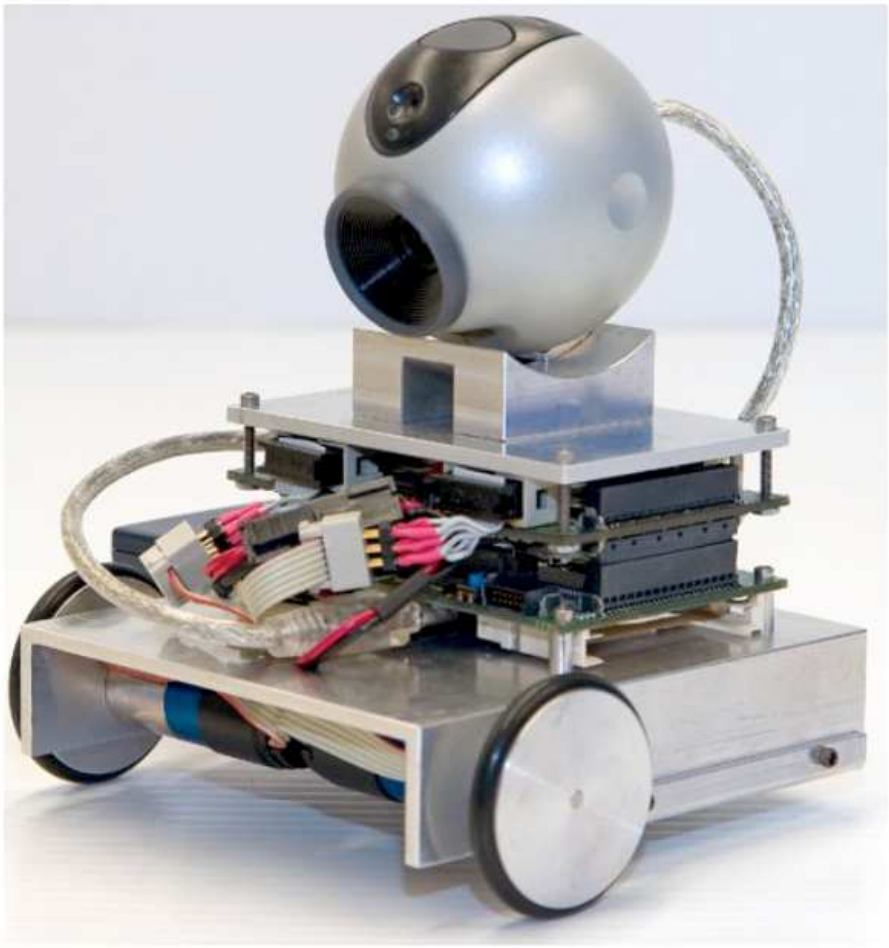


Figure 2.5: Webcam-based Mobile Robot

2.3 Sensors

2.3.1 Infrared Sensor

The infrared sensor consists of an emitter and a detector which must be worked in pair. The emitter will emit an invisible infrared light and the detector will pick up the reflected infrared light. The infrared detector basically acts as a phototransistor with the base voltage determined by the amount of light hitting the transistor.

Greater amount of infrared light will cause more current to flow through the phototransistor to cause a certain amount of voltage drop. By setting the correct

threshold, black or white surfaces can be determined by the output voltage. White surfaces generally reflect well, while black surfaces reflect poorly. The common connection of the infrared sensor is shown in Figure 2.6.

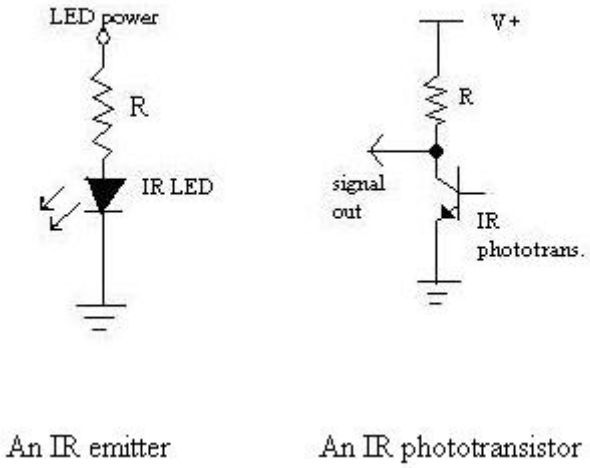


Figure 2.6: Connection of Infrared Sensor

2.3.2 Image Sensor

An image sensor, as known as a camera sensor can be used for line following purpose as well. It captures the image ahead of a line following robot and extract the line information hence line following can be done with advance algorithms.

Charged-coupled device (CCD) and complementary metal oxide semiconductor (CMOS) image sensors are two different technologies for capturing images digitally. Each has unique strengths and weaknesses giving advantages in different applications.

Beside these two types of image sensor, there are more sophisticated image sensor modules specially developed for robotics application. A very famous example of this mind of image sensor is CMUcam3.

2.3.2.1 Charge-Coupled Device (CCD) Image Sensor

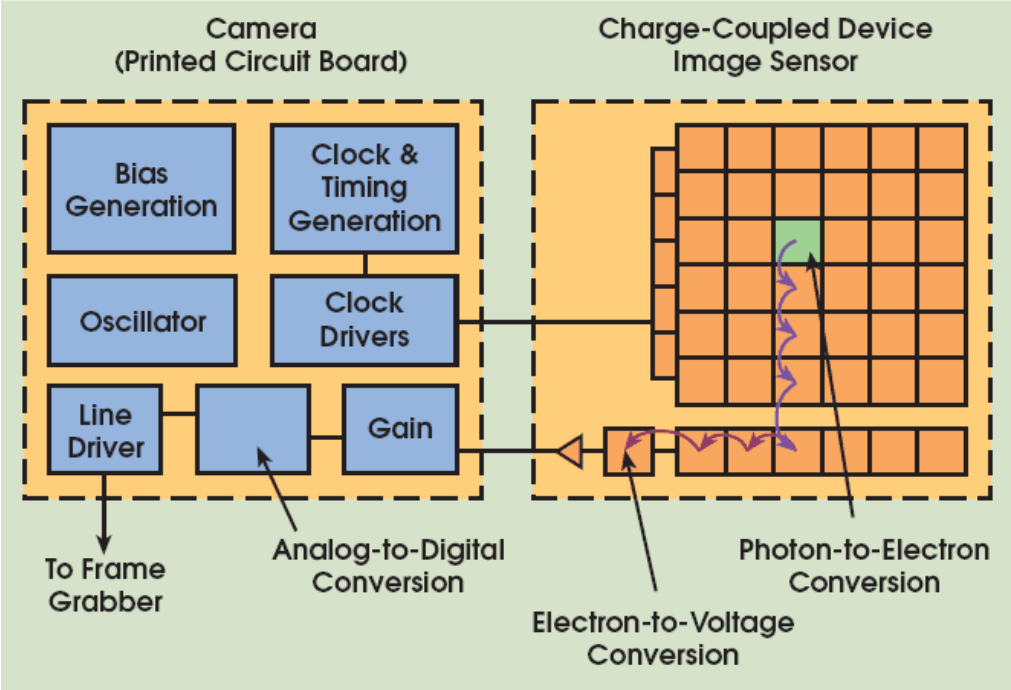


Figure 2.7: Block Diagram of CCD Image Sensor

(Source: Litwiller, 2001)

A CCD is an analogue device. When light strikes the chip it is held as a small electrical charge in each photo sensor. The charges are converted to voltage one pixel at a time as they are read from the chip. Additional circuitry in the camera converts the voltage into digital information. The block diagram of CCD image sensor is illustrated in Figure 2.7.

On a CCD image sensor, most functions take place on the camera's printed circuit board. If the application's demands change, a designer can change the electronics without redesigning the image sensor.

In a CCD sensor, every pixel's charge is transferred through a very limited number of output nodes (often just one) to be converted to voltage, buffered, and sent off-chip as an analogue signal. All of the pixel can be devoted to light capture, and the output's uniformity which is a key factor in image quality is high.

2.3.2.2 Complementary Metal-Oxide-Semiconductor (CMOS) Image Sensor

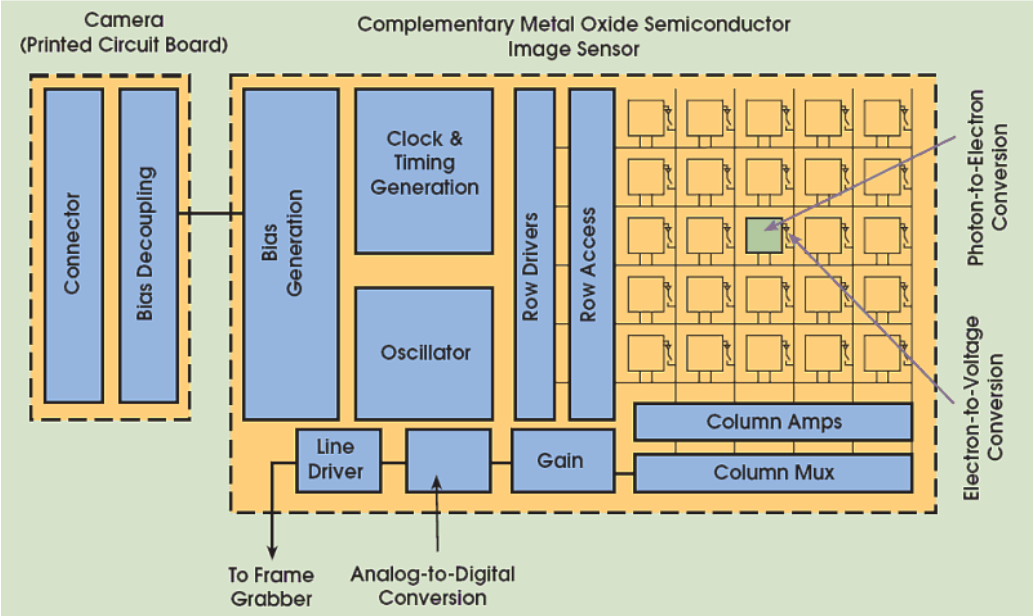


Figure 2.8: Block Diagram of CMOS Image Sensor

(Source: Litwiller, 2001)

A CMOS chip is a type of active pixel sensor made using the CMOS semiconductor process. Extra circuitry next to each photo sensor converts the light energy to a voltage. Additional circuitry on the chip may be included to convert the voltage to digital data. The block diagram of a CMOS image sensor is shown in Figure 2.8.

A CMOS image sensor converts charge to voltage at the pixel, and most functions are integrated into the chip. This makes the image sensor functions less flexible but, for applications in rugged environments, a CMOS camera can be more reliable.

2.3.2.3 CMUcam3

Figure 2.9: CMUcam3

(Source: CMUcam3 Datasheet)

The CMUcam3 which shown in the Figure 2.9 is an ARM7TDMI based fully programmable embedded computer vision sensor. The main processor is the Philips LPC2106 connected to an Omnivision CMOS camera sensor module. The block diagram of the CMUcam3 is shown in Figure 2.10.

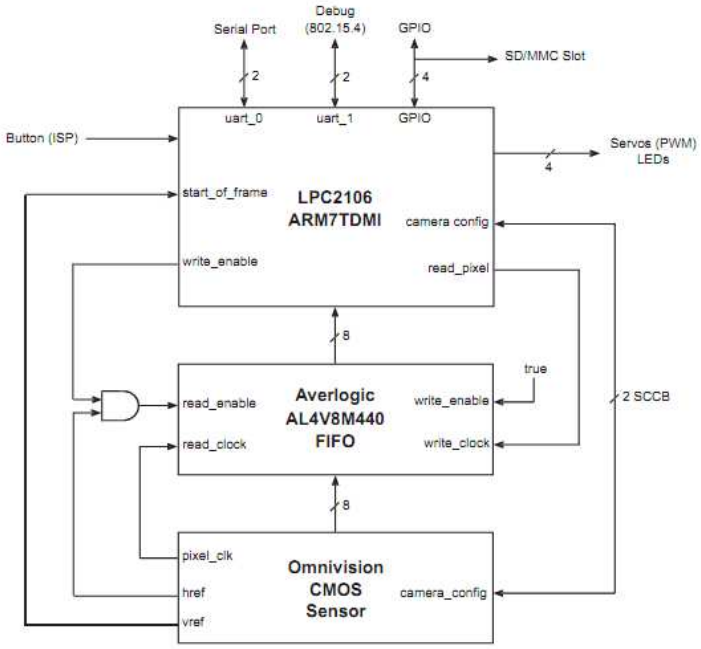


Figure 2.10: CMUcam3 Hardware Block Diagram

(Source: CMUcam3 Datasheet)

Custom C code can be developed for the CMUcam3 using a set of open source libraries and example programs, Executables can be flashed onto the board using the serial port with no external downloading hardware required.

The CMUcam3 is a hardware platform couples with an open source development environment. It is targeted toward users that are already familiar with basic image processing and who are comfortable with microcontroller programming.

The CMUcam2 emulation firmware is available for users who want basic image processing accessible through a simple serial interface and do not wish to implement their own algorithms.

The CMUcam2 provides a simple human readable ASCII communication protocol allowing for interactive control of the camera form a serial terminal program or a microcontroller. For line following application, the colour tracking function can be used to detect the black colour line.

2.4 Serial Communication

Serial communication is a device communication protocol that is standard on almost every computer. The serial port sends and receives bytes of data one bit at a time. Although this is slower than parallel communication, which allows the transmission of an entire byte at once, it is simpler and you can use it over longer distances.

Typically, serial communication is used to transmit ASCII data. They complete communication by using three transmission lines: transmit, receive, and ground. Because serial is asynchronous, the port can transmit data on one line while receiving data on another. Other lines are available for handshaking but are not required. The important serial characteristics are baud rate, data bits, stop bits, and parity. For two ports to communicate, these parameters must match. The typical serial communication connection from a microcontroller to a personal computer is shown in Figure 2.11.

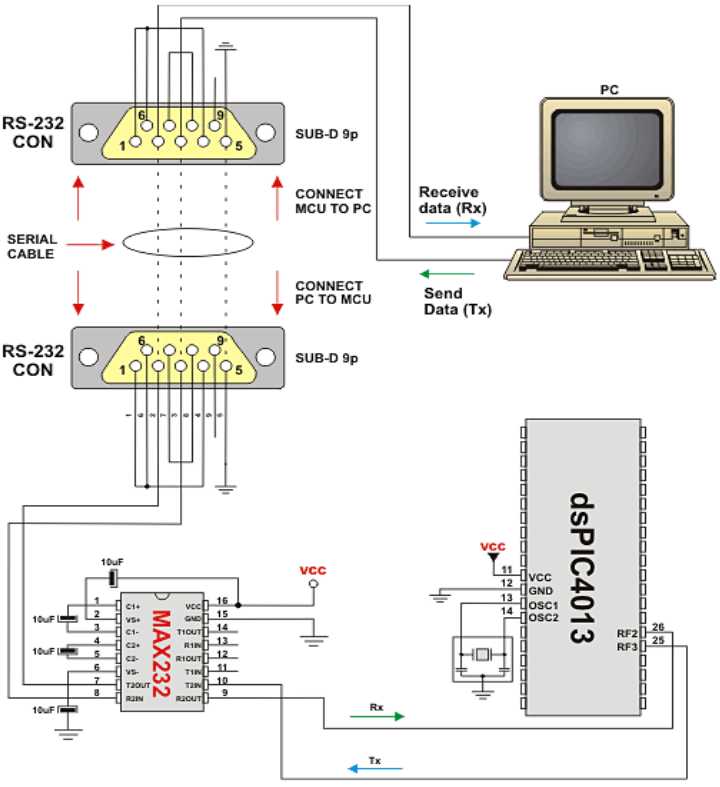


Figure 2.11: Serial Communication with PC

(Source: http://www.mikroe.com)

2.4.1 Universal Asynchronous Receiver Transmitter (UART)

The Universal Asynchronous Receivers Transmitter (UART) module is the basic serial I/O module available in most of the microcontrollers. The UART is a full-duplex asynchronous system that can communicate with peripheral devices, such as personal computer, RS-232, and RS-485 interfaces.

Figure 2.12 shows a simplified block diagram of the UART module. The UART module consists of the three key hardware elements: Baud-rate generator, asynchronous transmitter, and asynchronous receiver.

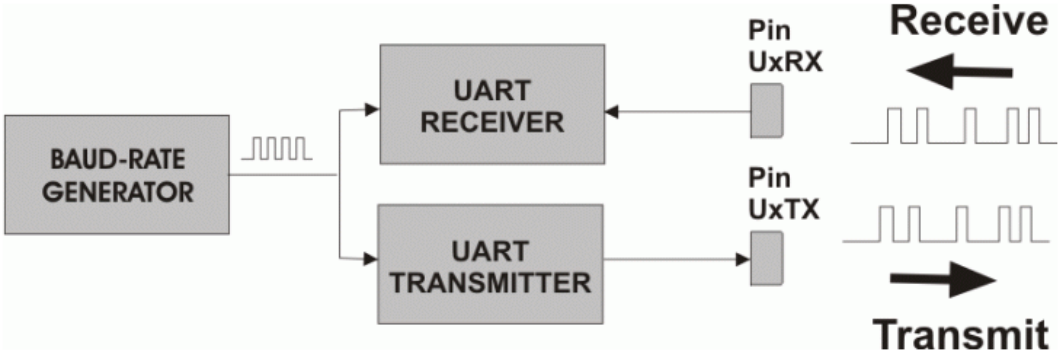


Figure 2.12: UART Module Block Diagram

(Source: http://www.mikroe.com)

A UART is usually an individual or part of an integrated circuit used for serial communications over a computer or peripheral device serial port. UART is now commonly included in microcontrollers. The UART takes byte of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires.

2.4.2 Serial Peripheral Interface (SPI)

The SPI bus is a synchronous serial data link standard, named by Motorola, which operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (SS) lines. Sometimes SPI is called a 4-wire serial bus.

The SPI bus can operate with a single master device and with one or more salve devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. With multiple slave devices, an independent SS signal is required form the master for each slave devices.

Figure 2.13 shows the connection diagram of SPI interface with single slave device while Figure 2.14 demonstrates the connection of a master device with three independent slave devices.

Figure 2.13: Single Slave Device

(Source: http://www.byteparadigm.com)

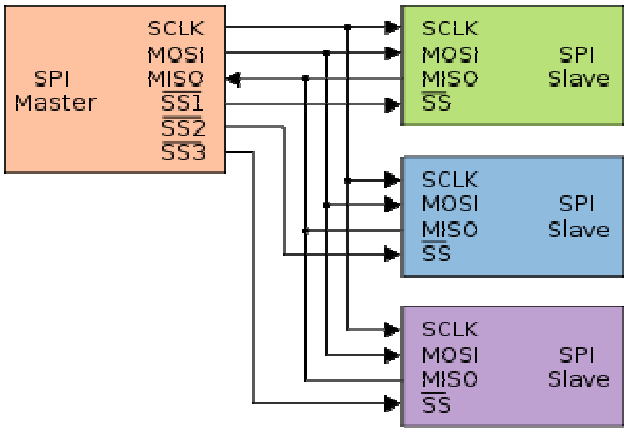


Figure 2.14: Multiple Slave Devices

(Source: http://www.byteparadigm.com)

2.5 Bluetooth Serial Port Plug

Figure 2.15: Bluetooth Serial Port Plug

The Free2Move Bluetooth Serial Port Plug F2M01C1 shown in Figure 2.15 offers a replacement of the serial cable by a wireless link based on the Bluetooth wireless technology.

The F2M01C1 Serial Port Plug is a Class 1 Bluetooth device with a very dense packing. The unit gives a nominal range of approximately 100m. No external drivers are needed, A user-friendly Windows application is included that can be used to configure the plug to suite the application requirements.

The F2M01C1 can be used together with other Bluetooth units that support the Serial Port Profile e.g. laptops and mobile phones. Examples of possible applications include embedded systems, stand alone sensors, computer peripherals, and domestic and industrial appliances.

2.6 Radio-Frequency Identification (RFID) Module

Being developed based on NXP's transponder IC, HF RFID Module SL015M-3 shown in Figure 2.16 is a 13.56MHz reader/writer which complies with ISO15693 and supports I.CODE SLI and Tag_it HFI. It does auto real-time detecting tag which moves into or out of detective range and reports through one output pin's logic level.

Figure 2.16: RFID Module

In addition, it integrates all necessary components and antenna into one printed circuit board. The external microcontroller can work with SL015M-3 to read/write ISO15693 cards and labels by simple serial communication commands.

2.7 LED Driver

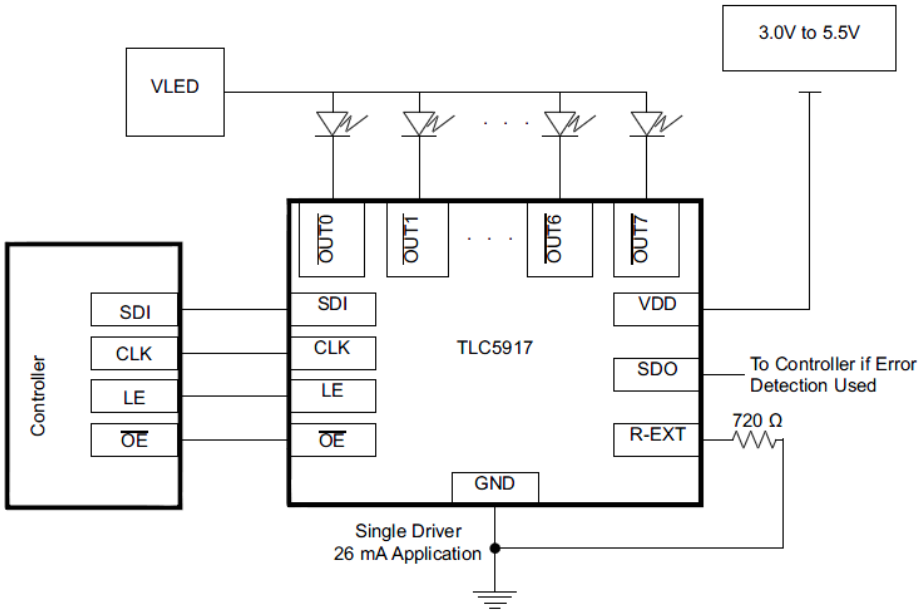


Figure 2.17: LED Driver Block Diagram

(Source: TLD5916 Datasheet)

The TLD5916 is designed for LED displays and LED lighting applications with constant-current control and open-load, short-load, and over temperature detection. The TLC5916 contains an 8-bit shift register and data latches, which convert serial input data into parallel output format. The block diagram is shown in Figure 2.17.

At the output stage, eight regulated current ports are designed to provide uniform and constant current for driving LEDs within a wide range of LED forward voltage variations. Used in system design for LED display applications, e.g. LED panels, it provides great flexibility and device performance.

Users can adjust the output current from 3 mA to 120 mA per channel through an external resistor, which give flexibility in controlling the light intensity of LEDs. The devices are designed for up to 20 V at the output port. The higher clock frequency, 30 MHz, also satisfies the system requirement of high-volume data transmission.

2.8 Proportional Integral Derivative (PID) Control

A PID controller is the most commonly used feedback controller, it is widely used in industrial control systems. A PID controller calculates an “error” value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs.

The PID controller algorithm involves three separate constant parameters: the proportional, the integral, and the derivative values, denoted P, I, and D. P depends on the present error, I on the accumulations of past errors, and D is a prediction of future error, based on current rate of change. The block diagram is shown in Figure 2.18. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, or the power supplied to a heating element.

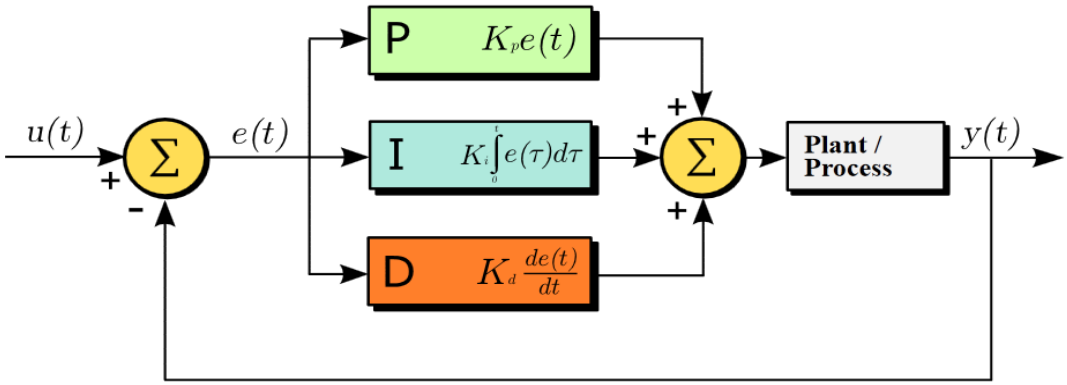


Figure 2.18: PID Controller Block Diagram

(Source: Araki)

Some applications may require using only one or two actions to provide the appropriate system control. This is achieved by setting the other parameters to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions. PI controllers are fairly common, since derivative action is sensitive to measurement noise, whereas the absence of an integral term may prevent the system from reaching its target value due to the control action.

CHAPTER 3

LINE TRACKING SYSTEM FOR THE FREESCALE CUP 2011

3.1 Overall System Architecture

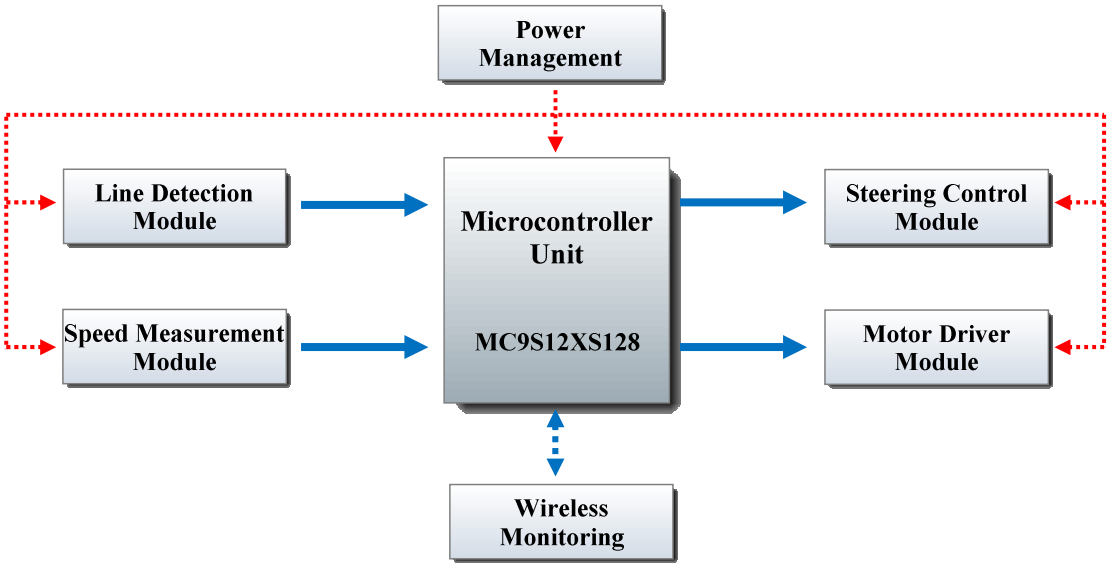


Figure 3.1: Overall System

The design of the overall system of the smart car model is shown in Figure 3.1. The Freescale MC9S12XS128 microcontroller was used as the processor of the system. The line detection and speed measurement modules provide input to the system and the steering control and motor driver modules are the output.

An image sensor and an array of infrared sensors were used in the line detection module to acquire the racing track information. As for speed measurement

module, two pairs of infrared sensor were attached to both rear wheels respectively to quantify the speed of each wheel independently.

Steering control module was used to determine the direction of travelling by controlling the servo based on the racing track information. Motor driver module was used to control the two Direct Current (DC) driving motors.

The power management module was designed to provide different and appropriate voltage level for all the devices in the system and the wireless monitoring module was used as an aid for debugging process.

3.1.1 Interface of Sub-Systems

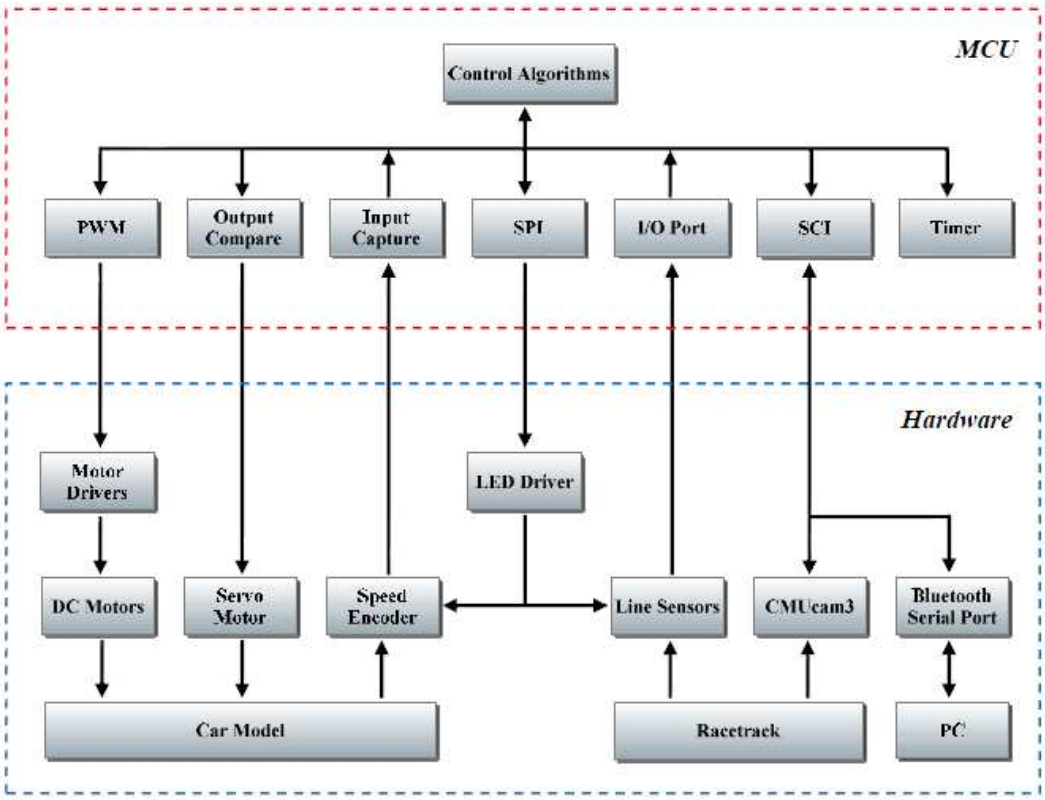


Figure 3.2: Interfacing of MCU and Hardware

The usage of the microcontroller peripherals to interface with the hardware is shown in Figure 3.2. The peripherals used were Pulse-Width Modulation (PWM), Output Compare, Input Capture, Serial Peripheral Interface (SPI), General Purpose Input Output (GPIO) port, Serial Communication Interface (SCI) and Timer.

Besides, the Phase-Locked Loop (PLL) was used to boost the bus frequency to the maximum which is 40 MHz to increase the processing speed; in another word, to increase the number of instructions per second. Refer to Appendix A for the main function that was run in the microcontroller.

3.2 Line Detection Module

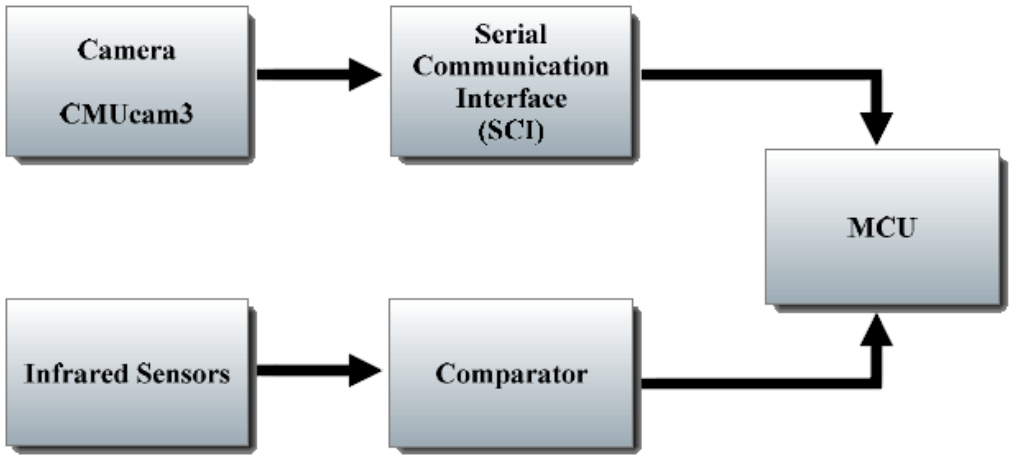


Figure 3.3: Line Detection Module

The line detection module as shown in Figure 3.3 was consisted of an image sensor and an array of infrared sensors. The camera used was CMUcam3 that communicates with the MCU through the SCI. The infrared sensor array consisted of 7 pairs of sensors and the arrangement of the sensors was in a row to detect the current position of the smart car.

3.2.1 CMUcam3

The CMUcam3 was controlled by serial commands and the data was transmitted to MCU at the baud rate of 115200 bits per second (bps), 8 data bits, 1 stop bit, no parity bit and no flow control. All the commands used to control the CMUcam3 were summarized in the Table 3.1.

Table 3.1: CMUcam3 Commands and Description

Command	Description
GV	Get the version of the firmware, to detect the presence of the CMUcam3
PM	Change the CMUcam3 to poll mode
ST	Set the tracking parameter for RGB channel
VW	Resize the frame being captured and processed
OM	Mask out the undesired output
TC	Track colour in the defined range

The thresholds for each Red, Green, and Blue channel must be set correctly in order to track a specified colour. Table 3.2 listed the thresholds set for each channel to track the black colour line.

Table 3.2: Tracking Parameters

Channel	Minimum	Maximum
Red	16	30
Green	16	30
Blue	16	30

Any colour detected by the CMUcam3 fall within the range of preset RGB threshold values will be tracked and the information of that tracked region will be processed. A T-packet data will be output from the CMUcam3 to the microcontroller as a result of colour tracking. The illustration of T-packet data is shown in Table 3.3.

Table 3.3: T-packet Data

T mx my x1 y1 x2 y2 pixels confidence	
mx	The middle of mass x value
my	The middle of mass y value
x1	The left most corner's x value
y1	The left most corner's y value
x2	The right most corner's x value
y2	The right most corner's y value
pixels	Number of pixels in the tracked region
confidence	The confidence level of the data

For line following application, the mx data was sufficient as it was the centroid location of the black line, the black line which was curved to the left or right will cause the mx data output shifted to smaller or bigger value. For example, the mx returned 44 when the line is at the centre and it will return smaller or bigger data when the line is detected at left side or right side. At the end, the output data was masked but left the mx data. Refer to Appendix B for the functions developed to communicate with the CMUcam3.

Figure 3.4 shows the initialization sequence of the CMUcam3. These commands were used to configure the CMUcam3 ready for the line following task. An "ACK" will be replied by the CMUcam3 as the acknowledge flag to indicate the command was executed successfully. If "ACK" was not received after a command was sent, then the command will be sent again.

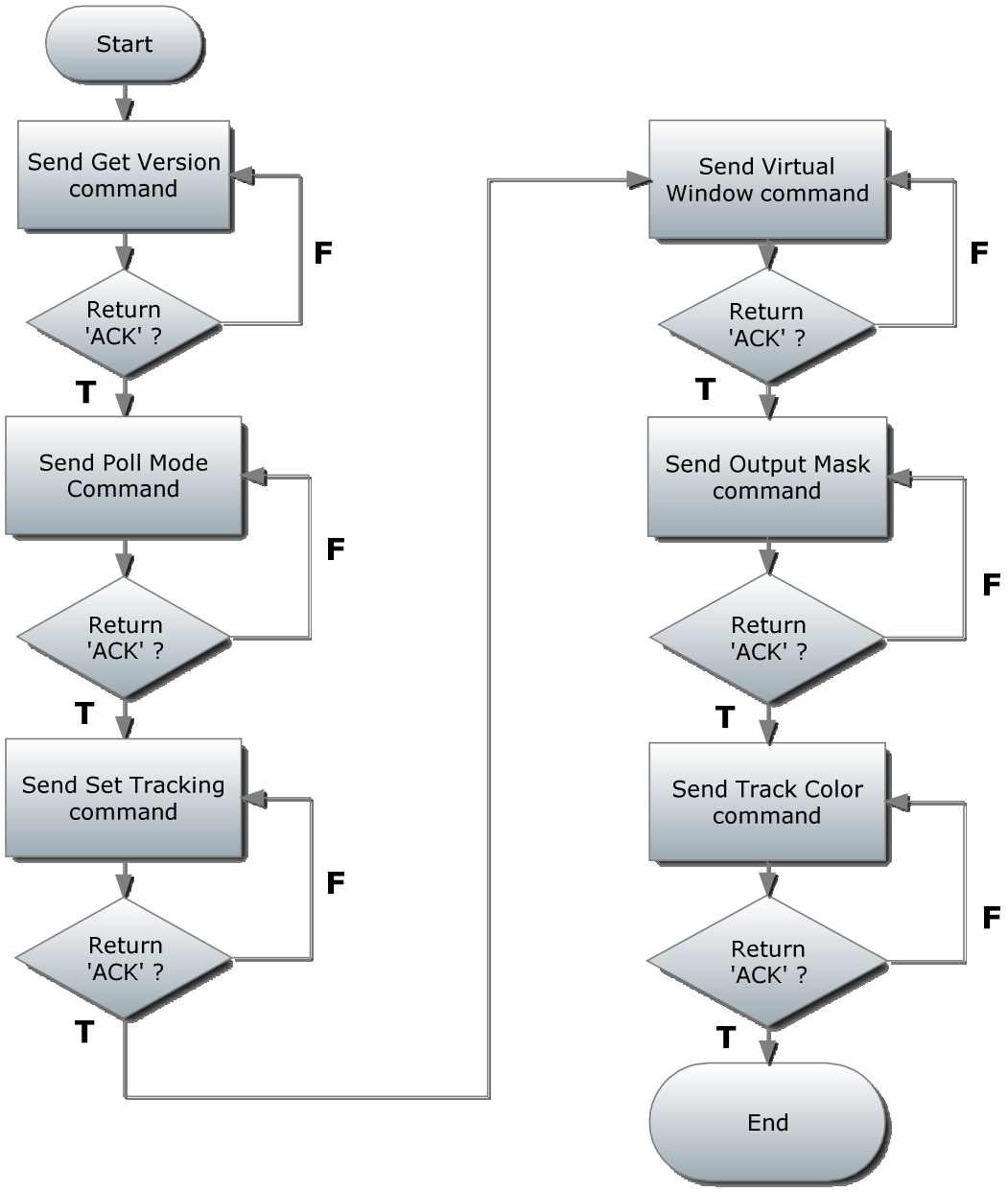


Figure 3.4: CMUcam3 Initialization

3.2.2 Infrared Sensors Array

The infrared sensors were arranged in a straight line to detect the location of the black line with respect to the smart car position. The output of the receiver was fed into comparator ICs LM339 to pre-process the signal before it was fed into the microcontroller.

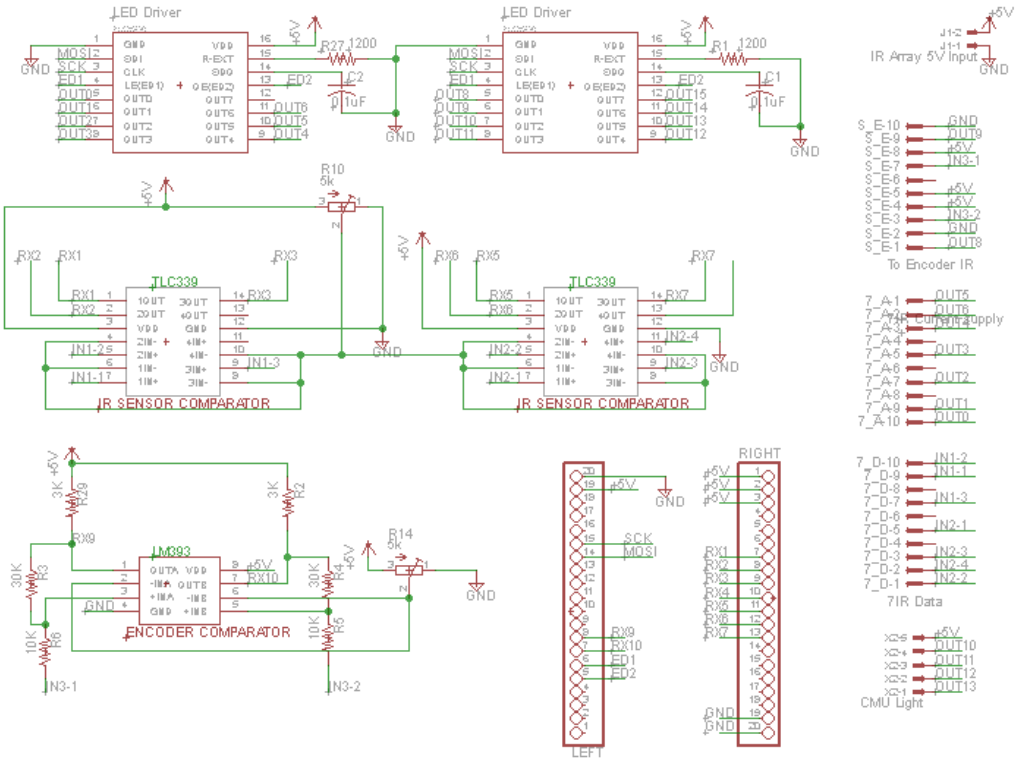


Figure 3.5: Schematic of Infrared Sensor Circuit

For the sake of stable performance, the infrared emitters were powered up using a LED driver IC TLC5916 to ensure that the current supplied was always consistent regardless of the voltage level of the battery. The LED driver IC was controlled by the microcontroller through the SPI. Figure 3.5 shows the schematic diagram of the infrared sensor circuit.

3.3 Steering Control Module

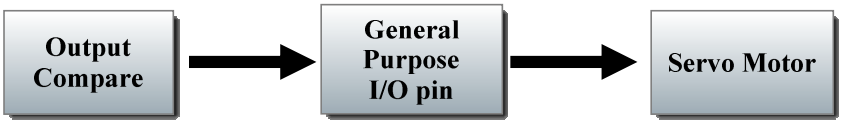


Figure 3.6: Steering Control Module

The steering control of the smart car was achieved by controlling the servo motor which was the Futaba S3010. Servomotor reacts according to the duty cycle of the 50 Hz signal which was fed as the signal.

However, in our system, all the PWM channels were used up for DC motors control so the alternative solution was to use the Output Compare feature to generate a desired signal for the servo motor as shown in Figure 3.6. Refer to Appendix D for the servo control program.

3.3.1 PD Controller

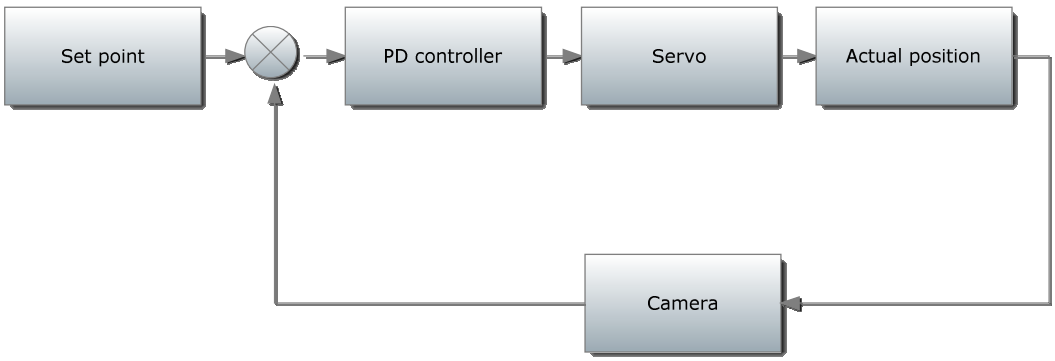


Figure 3.7: PD Control System Block Diagram

The position control of the smart car was achieved by using Proportional-Derivative (PD) controller. The output from the camera will be used to calculate the error in position as the input to the PD controller. The servo was controlled accordingly to react toward the desired set point. The PD controller block diagram was shown in Figure 3.7 and the controller equation was shown in equation 3.1.

$$S_O = S_T \pm K_p \times e \pm K_d \times \frac{e - e_l}{2} \quad (3.1)$$

where

S_O = Servo angle output

S_T = Set point

K_p = Proportional constant

K_d = Derivative constant

e = error

e_l = last error

Larger value of K_p will provide greater response in turning but excessive K_p will cause the system to be unstable and oscillate. The constant K_d helps in respond to the rate of change of error and it can reduce the oscillation but it is sensitive to noise so excessive K_d will cause the system to be unstable. The optimum K_p and K_d value were found out through trial and error method and different values were applied for different path condition.

3.4 Wireless Monitoring Module

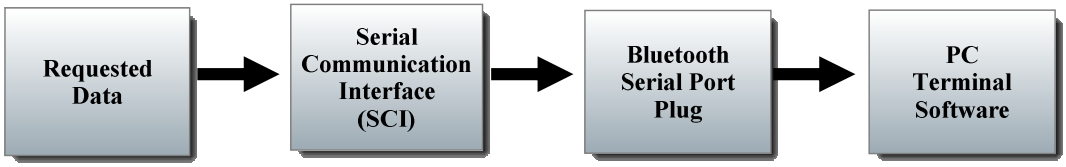


Figure 3.8: Wireless Monitoring Module

By using a Bluetooth serial port plug, any desired data can be requested from the microcontroller and transmit to the computer through a terminal software like HyperTerminal wirelessly. For example, the useful information like the speed and camera line tracking output can be transmitted to the computer for monitoring or

debugging purpose. Figure 3.8 shows the block diagram of the wireless monitoring module. Refer to Appendix C for the functions developed for wireless monitoring by using the Bluetooth serial port plug.

3.5 Mechanical Design

3.5.1 Camera Mounting

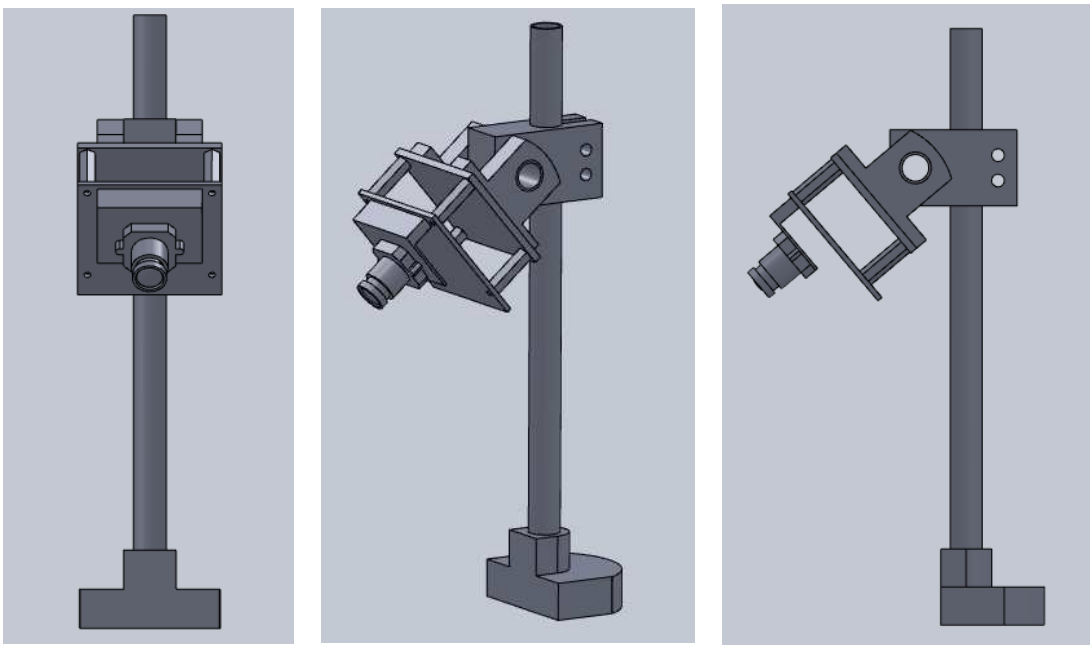


Figure 3.9: Design of Image Sensor Mounting in SolidWorks

The camera stand was designed for the height limit as stated in the rules and regulations which was 25cm. The materials used were aluminium and polyethylene (PE) which are very light weight. The structure was designed and models by using SolidWorks 2010 as shown in Figure 3.9.

This design provides 3 degrees of freedom which are pitch, yaw and height for greater flexibility. These 3 degrees of freedom was needed for the adjustment of

the image sensor's position and angle during testing and tuning. The weight of this mounting stand with the image sensor was about 0.16 kg.

The location of the mounting stand was at the front part of the chassis which was right behind the servo motor to enable the camera to have adequate front view distance.

3.5.2 Wheel Adjustment

The three major front wheels alignment parameters on a car are camber, caster, and toe. The purpose of these adjustments is to reduce tire wear, to ease the turning of the steering wheels, and to ensure that car travels in straight line.

In the design of smart car, positive camber, positive caster and toe-in configuration were chosen.

3.5.2.1 Camber

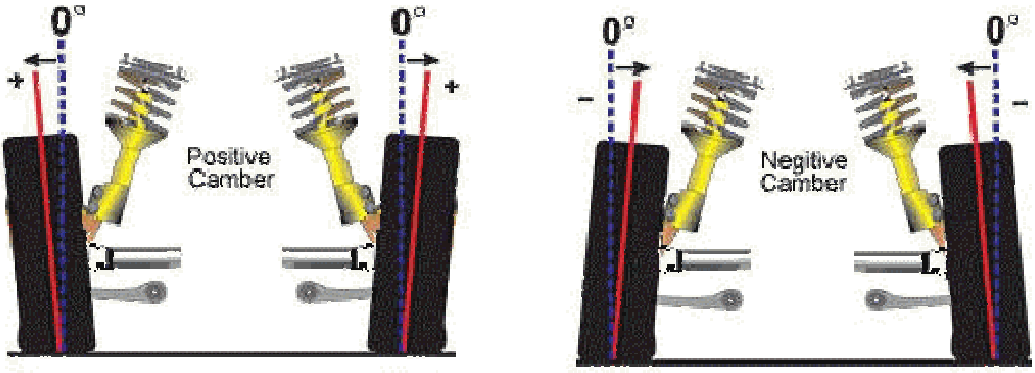


Figure 3.10: Positive Camber and Negative Camber

Camber is the angle of the wheel as shown in the Figure 3.10. If the top of the wheel is tilted out then the camber is positive; if it's tilted in, then the camber is negative. Positive camber can be used to offset vehicle loading to ease the turning of front wheels, whereas negative camber can be used to increase stability

By default, the camber angle was zero degree. Positive camber was selected due to the need to increase the response of the front wheels. However, this configuration will reduce the straight line stability, but it can be compensated with toe-in configuration.

3.5.2.2 Caster

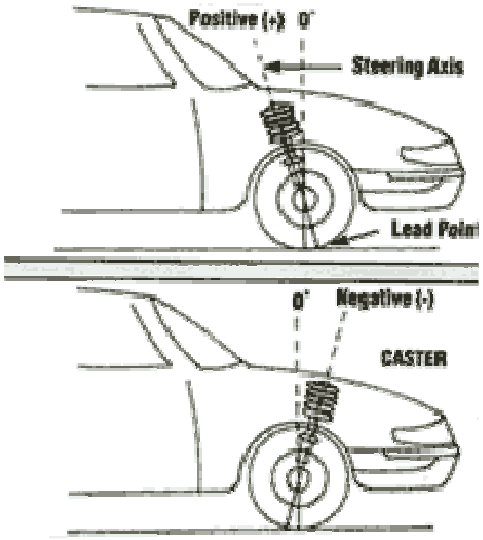


Figure 3.11: Positive Caster and Negative Caster

Caster is the tilt of the steering axis as shown in Figure 3.11. When the wheel is in front of the load then the caster is positive; else it is negative. Positive caster improves straight line stability because the steering axis intersects the ground ahead of the contact patch of the tire. As the car moves forward, the steering axis pulls the wheel along, and since the wheel drag along the ground, it fall directly in line behind the steering axis.

By default, the caster angle was zero degree. Positive caster was chosen to enhance straight line stability and to have a reaction force that help to steer back the front wheels after cornering.

3.5.2.3 Toe

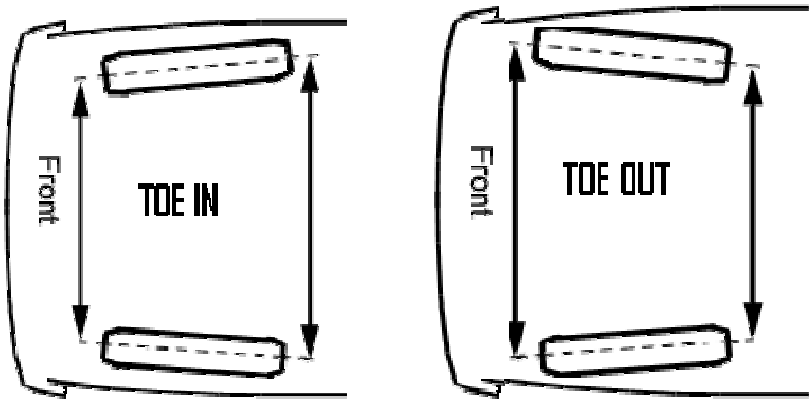


Figure 3.12: Toe In and Toe Out

Toe is the tilt of the wheels toward or away from one another when viewed from the front as shown in Figure 3.12. Tires that “toe-in” point toward each other whereas tires that “toe-out” point away from each other.

By default, the front wheels were neither toe-in nor toe-out. Positive camber configuration which will reduce the straight line stability so toe-in was applied in the design to compensate the effect of positive camber. Besides, increased toe-in will reduce over-steer, steady the car and enhance high-speed stability.

CHAPTER 4

PATH RECOGNITION SYSTEM FOR AUTOMATED DEMAND RESPONSIVE TRANSIT SYSTEM

4.1 Overall System Architecture

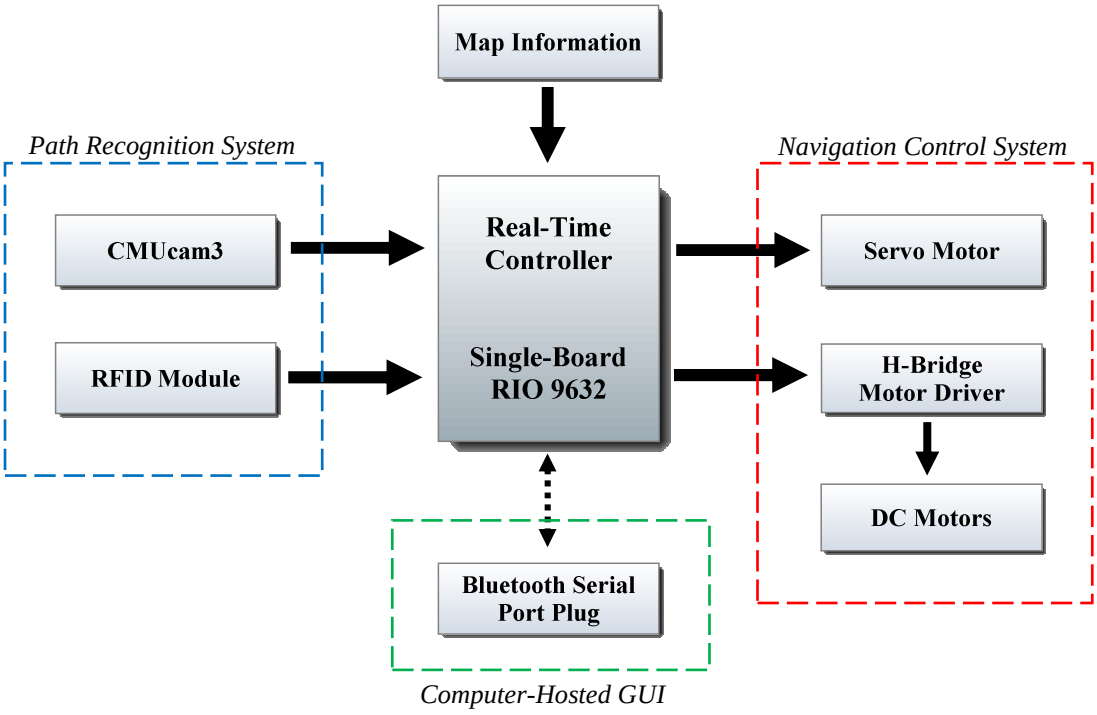


Figure 4.1: Overall System Architecture for ADRT

The overall system architecture for the ADRT was shown in Figure 4.1. The path recognition system for ADRT system consisted of a CMUcam3 and a RFID module which were used for road tracking and route identification respectively. Both of the

CMUcam3 and RFID module used serial communication to interface with the Real-Time controller Single-Board RIO (sb-RIO) 9632 from National Instruments.

The sb-RIO 9632 contains a 400 MHz Real-Time processor and a 2-million gates Field Programmable Gate Array (FPGA). LabVIEW which is a graphical programming language from National Instrument was used to program the sb-RIO. Besides, the additional modules such as LabVIEW Real-Time module and LabVIEW FPGA module were utilized to make use of the Real-Time processor and the FPGA on the sb-RIO. Refer to Appendix E for the project explorer in LabVIEW which listed all the Virtual Instrument (VI) generated by both Real-Time and FPGA modules

A computer-hosted Graphical User Interface (GUI) was developed to acquire the destination input from the passenger. The computer was communicated with the sb-RIO wirelessly via a Bluetooth serial port plug. The pre-known map information was stored in the non-volatile memory of the sb-RIO. With both the destination and map were known, path finding function was executed to find the shortest or the most cost-effective path to the destination.

The movement of the autonomous vehicle was controlled by the navigation control system which was consisted of a servo motor and a pairs of DC motors driven by H-bridge motor driver.

4.2 Path Recognition System

4.2.1 UART Communication

First and foremost, the CMUcam3 and RFID module used for path recognition were required to communicate serially with the sb-RIO. A level-shifted RS-232 serial interface was available on the sb-RIO but it was not applicable for both CMUcam3 and RFID module.

The establishment of the serial communication for CMUcam3 and RFID module were achieved by emulating the UART function in the FPGA contained in the sb-RIO. This required the usage of FIFO memory blocks and any digital IO pins to serve as transmit and receive pins.

Figure 4.2 shows the main virtual instruments (VI) block diagram to implement UART in LabVIEW. The VI requires many other sub-VIs such as read and write function to run simultaneously so that the UART function can be emulated. Refer Appendix F for the others UART functions such as initialize, write and close functions.

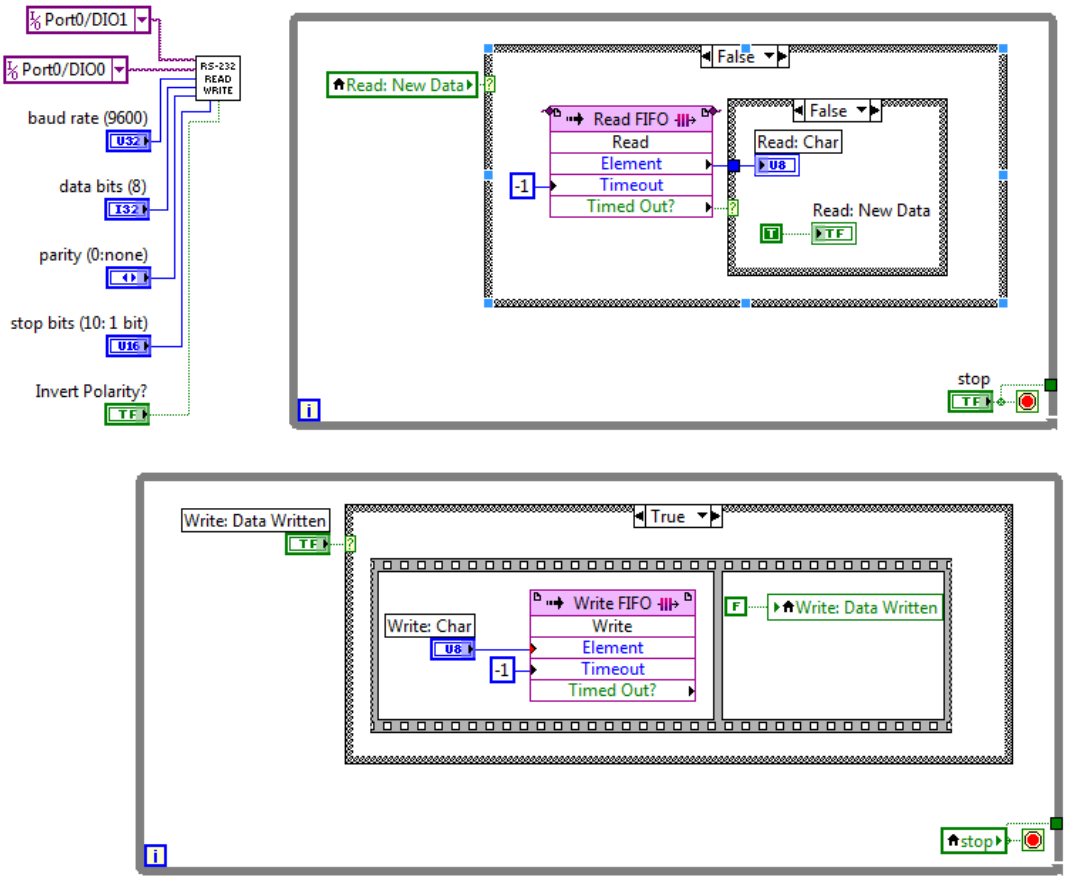


Figure 4.2: UART Emulation in LabVIEW

4.2.2 Road Tracking

The CMUcam3 was used to track the black road to ensure that the vehicle can travel accordingly on the black road. Therefore, the identical commands in previous chapter such as Track Colour were used and the T-Packet data was examined.

The object to be tracked is a 60 cm wide black colour road, as compared to the 3 cm wide black line in previous chapter. As the road is wide, the area exposed to light was increased too, so a different set of tracking parameters shown in Table 4.1 were used. These higher tolerance parameters can help the CMUcam3 to track the road with minimal error.

Table 4.1: Tracking Parameters

Channel	Minimum	Maximum
Red	16	50
Green	16	50
Blue	16	50

With the emulated UART in FPGA, the CMUcam3 can be controlled by the sb-RIO. Figure 4.3 illustrates the setting of tracking parameters for CMUcam3 done in LabVIEW while Figure 4.4 shows the reading of T-packet data as a result of tracking.

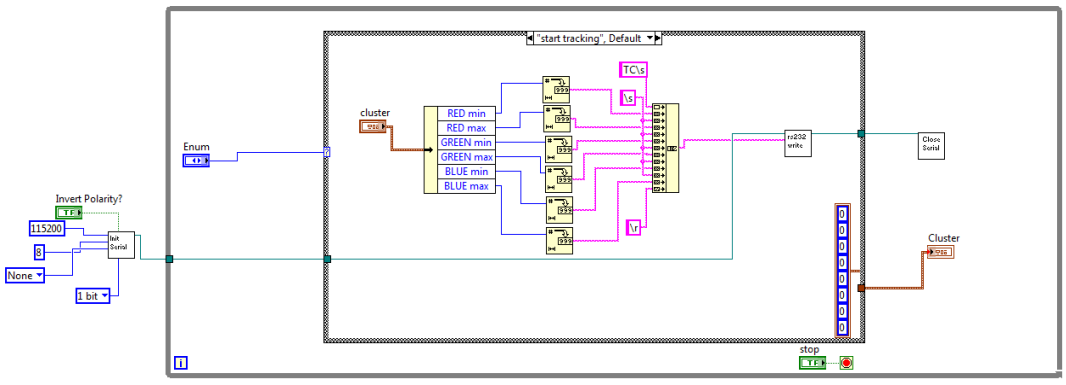


Figure 4.3: Set Tracking Parameters in LabVIEW

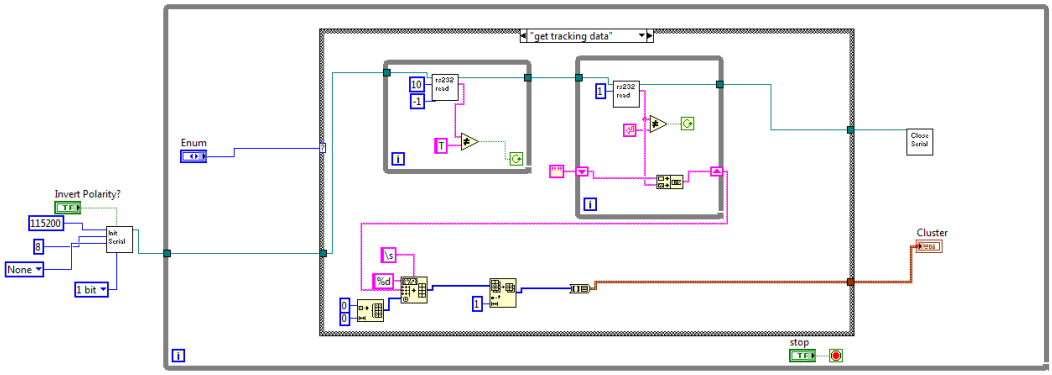


Figure 4.4: Get Tracking Data in LabVIEW

4.2.3 Route Identification

The RFID module was attached at the front of the autonomous vehicle to search for the desired route. RFID tags were placed at every junction so that the vehicle can know once it reached the junction where it needed to turn in. Figure 4.5 illustrates the placement of RFID Tags at the junctions for the autonomous vehicle to identify the route.

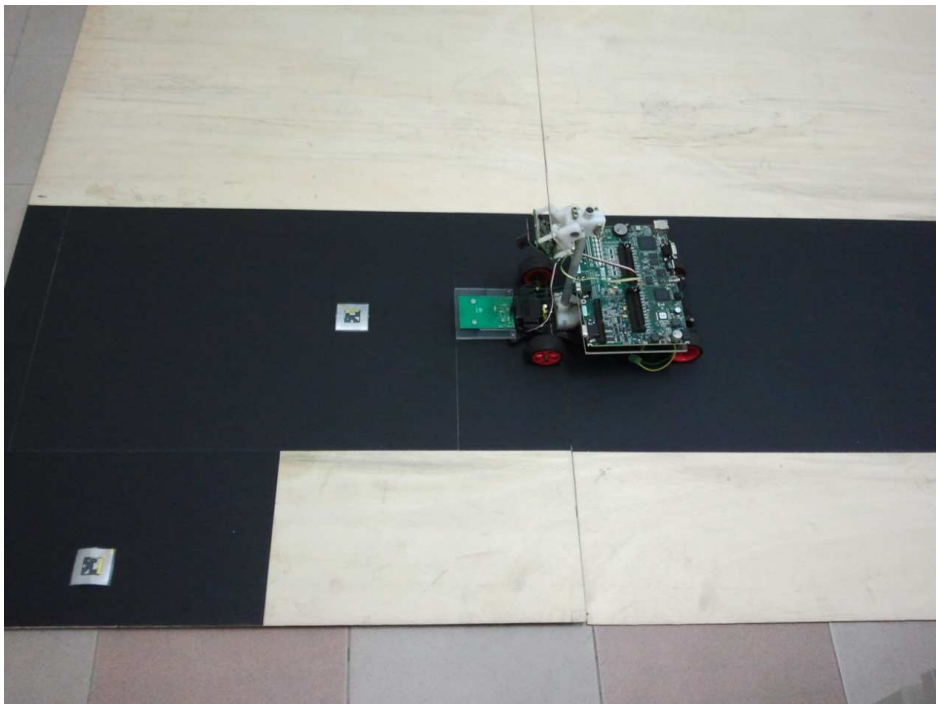


Figure 4.5: Route Identification with RFID Tags

The RFID module was communicated serially with the sb-RIO with a baud rate of 115200 bits per second, 8 data bits, 1 stop bit, no parity bit, and no flow control.

The communication in between the RFID module and the sb-RIO follows the formats stated in Table 4.2 and Table 4.3. The status and command overview were shown in Table 4.4 and Table 4.5.

Table 4.2: Communication Format from Host to Reader

<Preamble><Len><Command><Data><Checksum>	
Preamble	1 byte equal to 0xBA
Len	1 byte indicating the number of bytes from Command to Checksum
Command	1 byte Command code
Data	Variable length depends on the command type
Checksum	1 byte XOR of all the bytes from Preamble to Data

Table 4.3: Communication Format from Reader to Host

<Preamble><Len><Command><Status><Data><Checksum>	
Preamble	1 byte equal to 0xBD
Len	1 byte indicating the number of bytes from Command to Checksum
Command	1 byte Command code
Status	1 byte Command status
Data	Variable length depends on the command type
Checksum	1 byte XOR of all the bytes from Preamble to Data

Table 4.4: Status overview

Status	Description
0x00	Operation success
0x01	No Tag
0x04	Read Fail
0x05	Write Fail
0x06	Unable to read after write
0x07	Read after write error
0xF0	Checksum error
0xF1	Command code error

Table 4.5: Command Overview

Command	Description
0x31	Get tag information
0x32	Get block security status
0x33	Read blocks
0x34	Write a data block
0x35	Write AFI
0x36	Write DSFID
0x37	Lock Block
0x38	Lock AFI
0x39	LOCK DSFID
0x40	Control Red Led
0xFF	Reset

4.3 Servo Motor Control

The servo motor control signal was generated by using the FPGA digital I/O pin; any pins can be used to produce the 50 Hz pulse with the duty cycle varied from 0.5 ms to 2.5 ms.

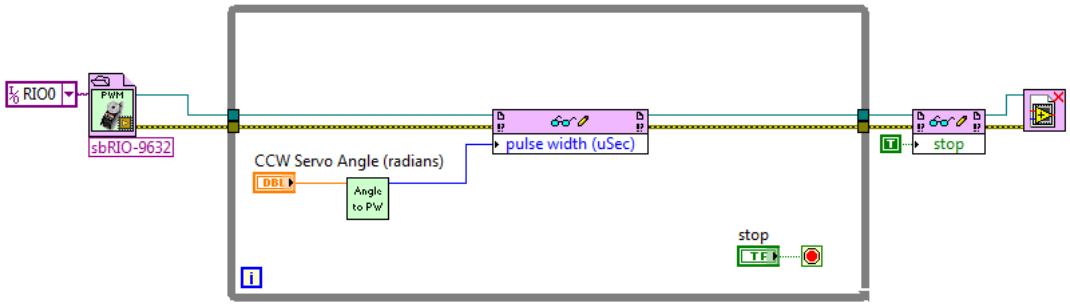


Figure 4.6: Servo Control Main Function

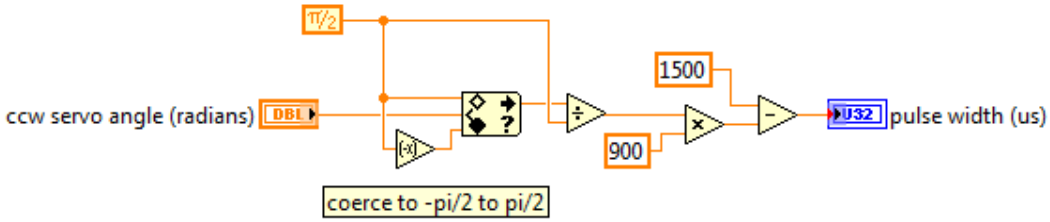


Figure 4.7: Angle to Pulse Width Conversion Function

Figure 4.6 shows the LabVIEW block diagram used to generate the pulse with the desired pulse width. Figure 4.7 shows the function that limits the range of input from $-\pi/2$ to $\pi/2$ and then converts the input from angle to pulse width.

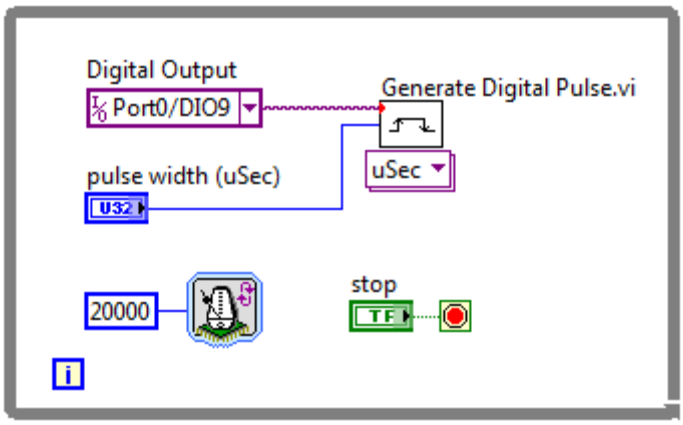


Figure 4.8: Digital Output Function in FPGA

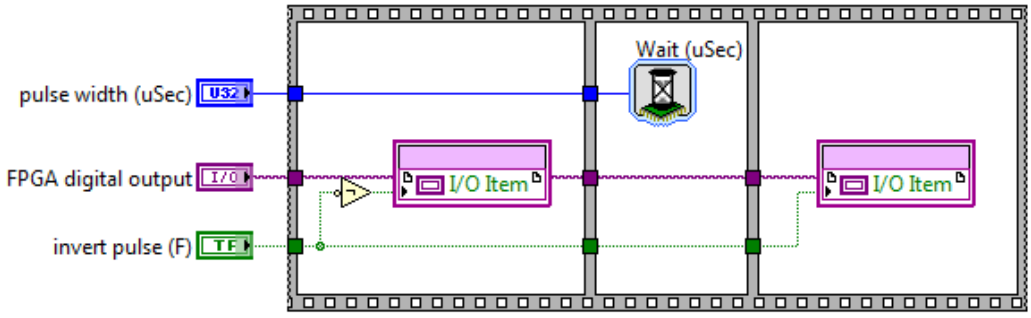


Figure 4.9: Digital Pulse Generation in FPGA

The digital IO pin of the FPGA was selected to generate the pulse, the block diagram is shown in Figure 4.8 and it was running in the FPGA together with the sub-function shown in Figure 4.9 to generate the digital pulse.

4.4 Wireless Communication

The wireless communication in between the computer-hosted GUI and the sb-RIO was established by using the Bluetooth serial port plug. This required the level-shifted RS-232 serial interface so the serial connector on sb-RIO can be used for this purpose.

A communication protocol driver called Virtual Instrument Software Architecture (VISA) was provided in LabVIEW. It allowed the direct access to the serial device attached to the sb-RIO and the read or write from the serial device. Figure 4.10 shows the application of VISA functions to initialize the Bluetooth serial device, read data from the device, and then close the communication.

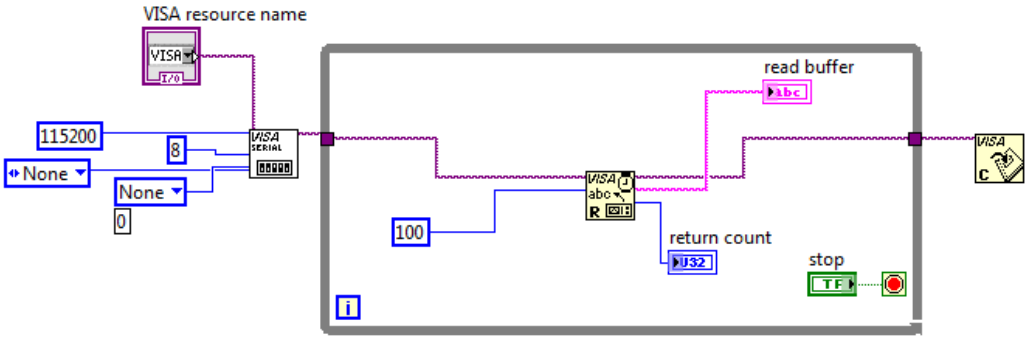


Figure 4.10: VISA Functions in LabVIEW

4.5 Map Design

The prototype of the residential area was designed to have four rows of terrace houses and a row of shop lots which were serves as the destinations of the passenger.

The dimension of this small scale residential area is about 4.8 meters long and 2.4 meters wide. This size of residential area was sufficient for the autonomous vehicle to demonstrate and verify the concept of the ADRT system.

The passenger will be picked up by the autonomous vehicle at the Light Railway Transit (LRT) station. Figure 4.11 shows the top view of the map and Figure 4.12 demonstrates the isometric view.

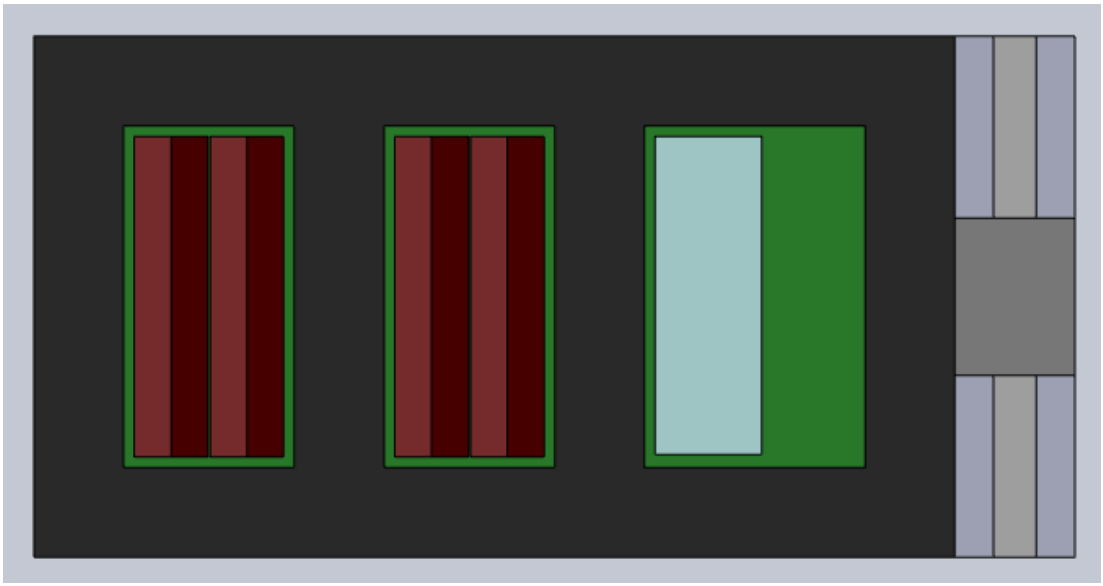


Figure 4.11: Map (Top View)

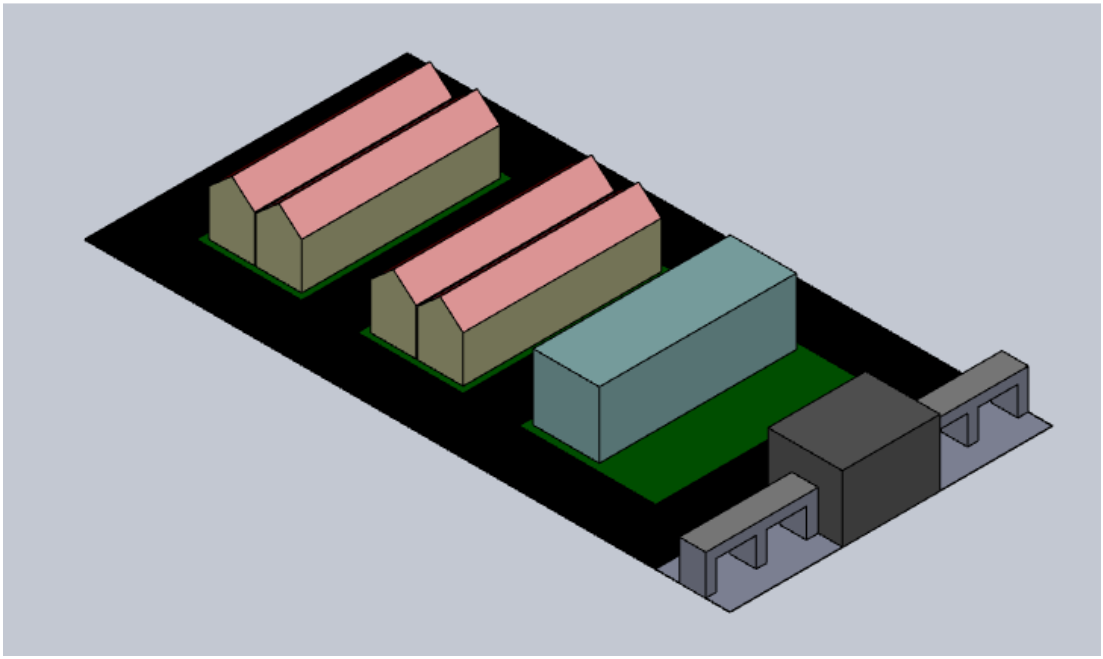


Figure 4.12: Map (Isometric View)

CHAPTER 5

RESULTS AND DISCUSSIONS FOR LINE TRACKING SYSTEM

5.1 Line Tracking

The CMUcam3 was successfully implemented to perform the task of black line tracking with the thresholds set correctly. The maximum front view distance attained with optimal performance is 50 cm measured from the front wheel, which is illustrated in Figure 5.1.

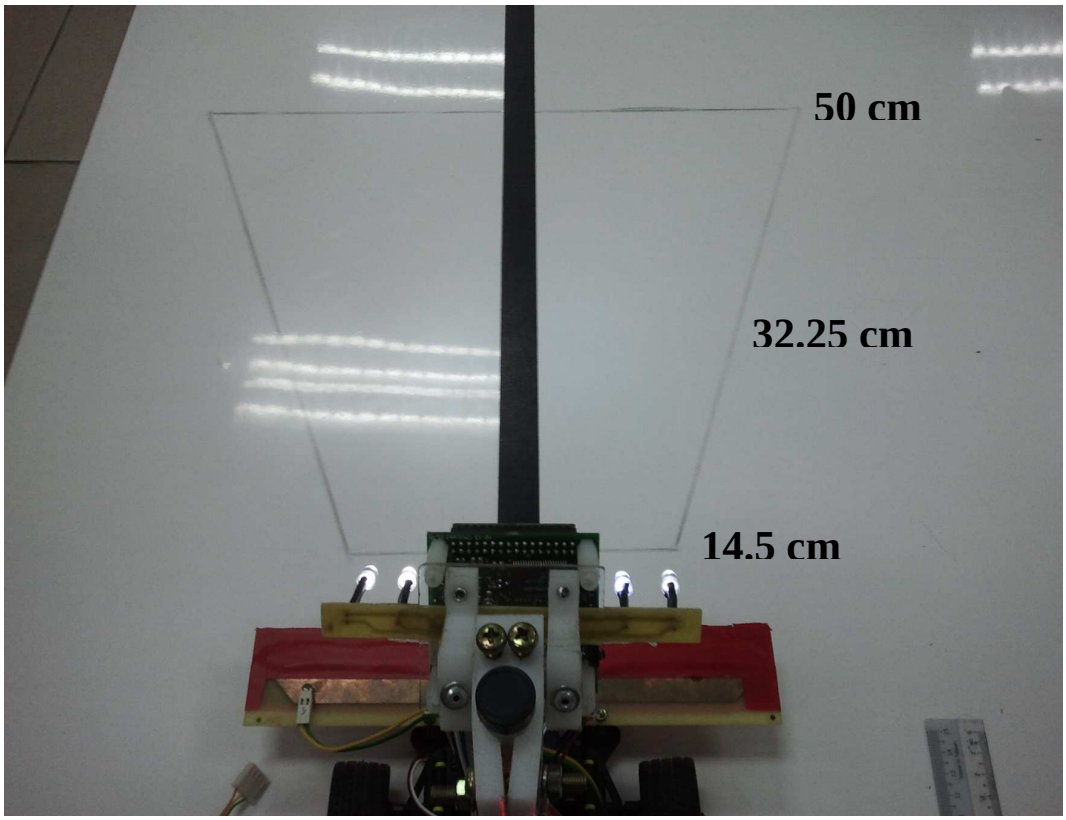


Figure 5.1: Front View Distance of CMUcam3

By using the CMUcam Frame Grabber software, the images seen by the CMUcam3 can be displayed on the computer screen. This is essentially important to adjust the camera lens focal length so that a sharp image can be obtained.

As shown in Figure 5.2, the tracking output data T-packet was displayed as well. The result was handy as the mx was 44 when the black line is right at the centre. Whereas in Figure 5.3, the black line was curved to the left side so the mx is 27.

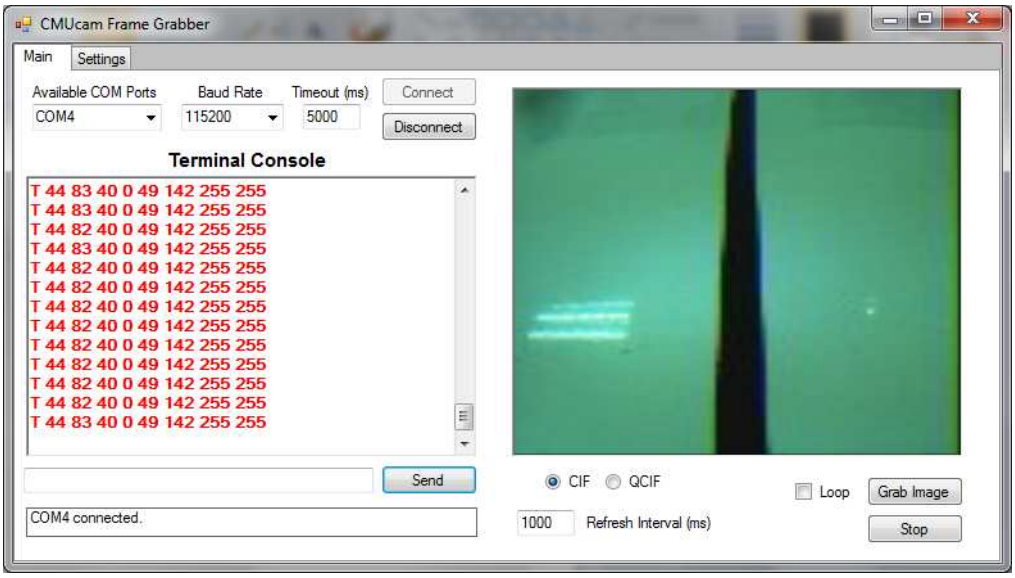


Figure 5.2: Line Tracking Result of Straight Line

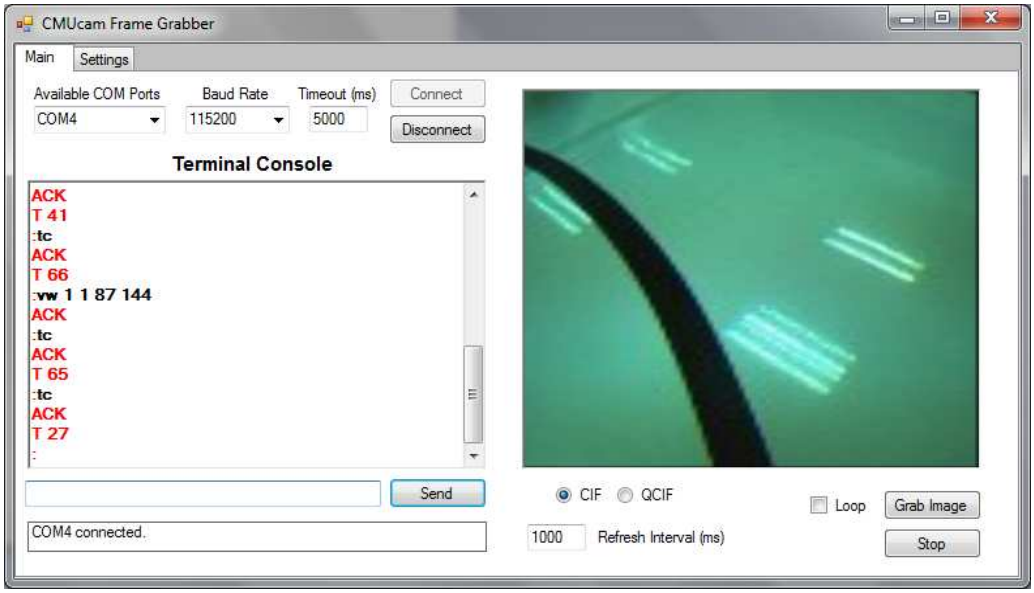


Figure 5.3: Line Tracking Result of Curved Line

One of the problems with the CMUcam3 was the response time taken to output the T-packet was too long. It was above 60 ms for each tracking data to be received and this is unacceptable for a high-speed smart car.

This problem had caused the smart car to run out of the racing track as a consequence of incapable to acquire the racing track information and react accordingly when it was travelling with high speed; especially when the smart car encountered a sharp turn.

The proposed solution to this problem was to reduce the size of the image being taken to be processed. This is different from reducing the resolution so the sharpness of the images would be remained. The segmented images were taken from pixel 68 to pixel 88 which was 20 pixels vertically in contrast with 144 previously; whereas its horizontal pixel number was remained as 88 pixels because it is a crucial parameter to detect the position of vertical black line, so higher resolution was allocated.

With the smaller image size, the response time had greatly reduced to about 20 ms per cycle. This had enabled the speed of smart car to be increased and the tracking performance was improved drastically.

Figure 5.4 shows the segmented image of a curved line which was same with Figure 5.3. Apparently the tracking performance was not affected as the mx value was still 27 which was same with the mx value shown in Figure 5.3.

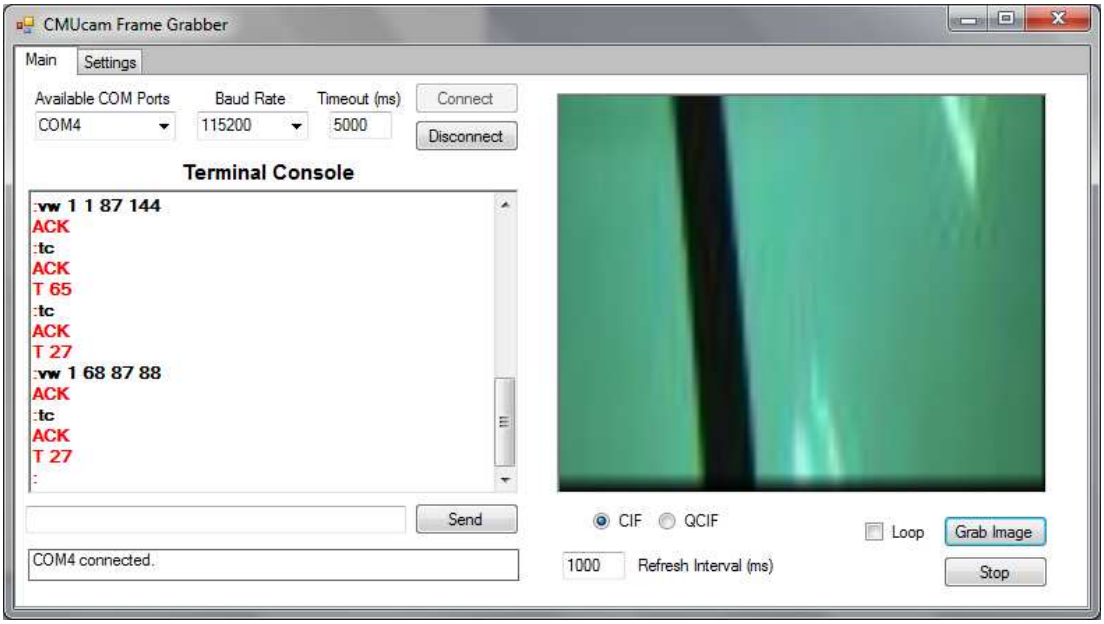


Figure 5.4: Line Tracking Result of Curved Line with Segmented Image

5.2 Steering Control

With the Output Compare feature of the microcontroller, the control signal required for a servo motor was successfully generated. The signal was fixed at 50 Hz which was necessary by the servo motor, and the duty cycle can be varied according to the desired value.

However, the steering angle limits had been set to avoid structural damage due to overturn so the duty cycle can only range from 1.18 ms to 1.84 ms with the 1.5 ms as the neutral position.

Figure 5.5 shows the control signal pulse of the servo motor when a straight line was detected. On the other hand, Figure 5.6 and Figure 5.7 demonstrate the signal sent to servo motor when a rightmost or leftmost line was found respectively.

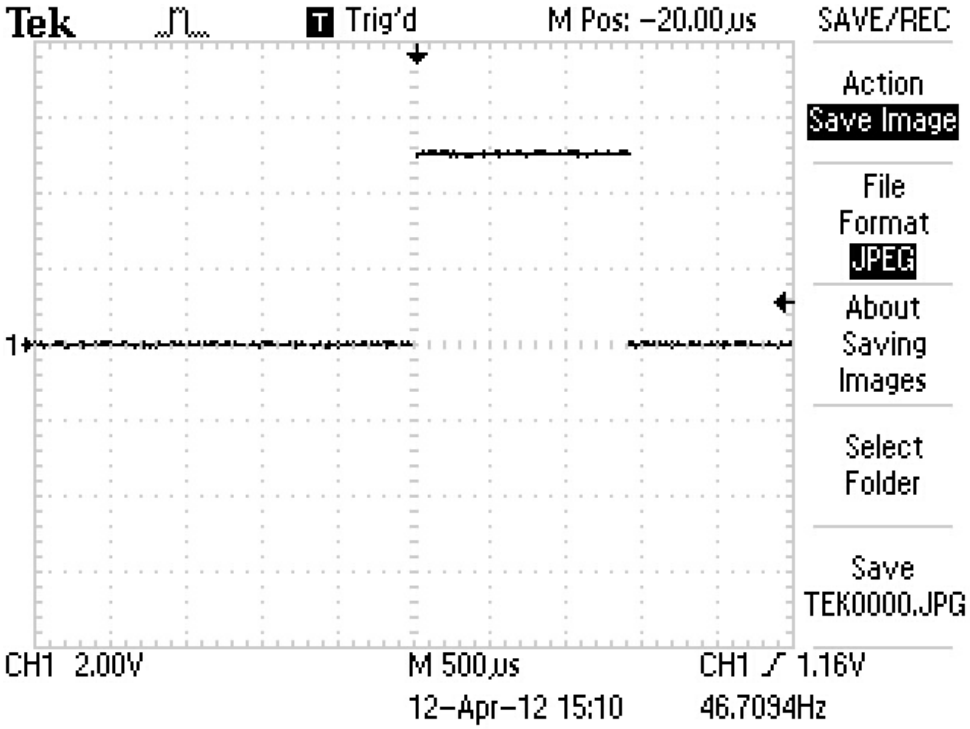


Figure 5.5: Control Signal When Straight Line Detected

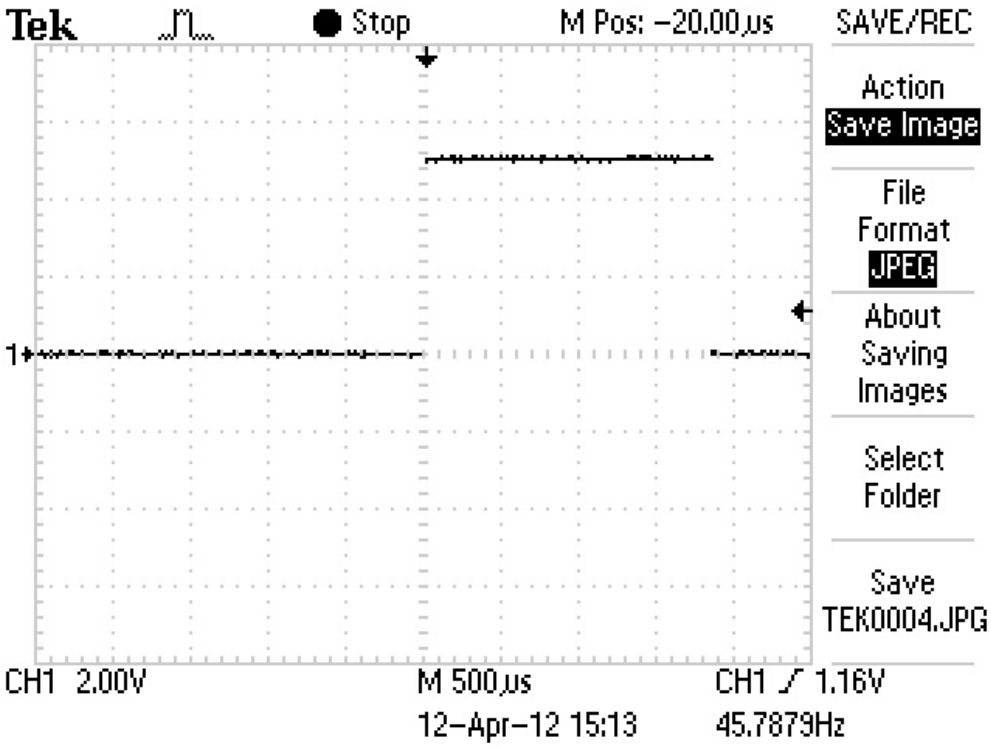


Figure 5.6: Control Signal When Rightmost Line Detected

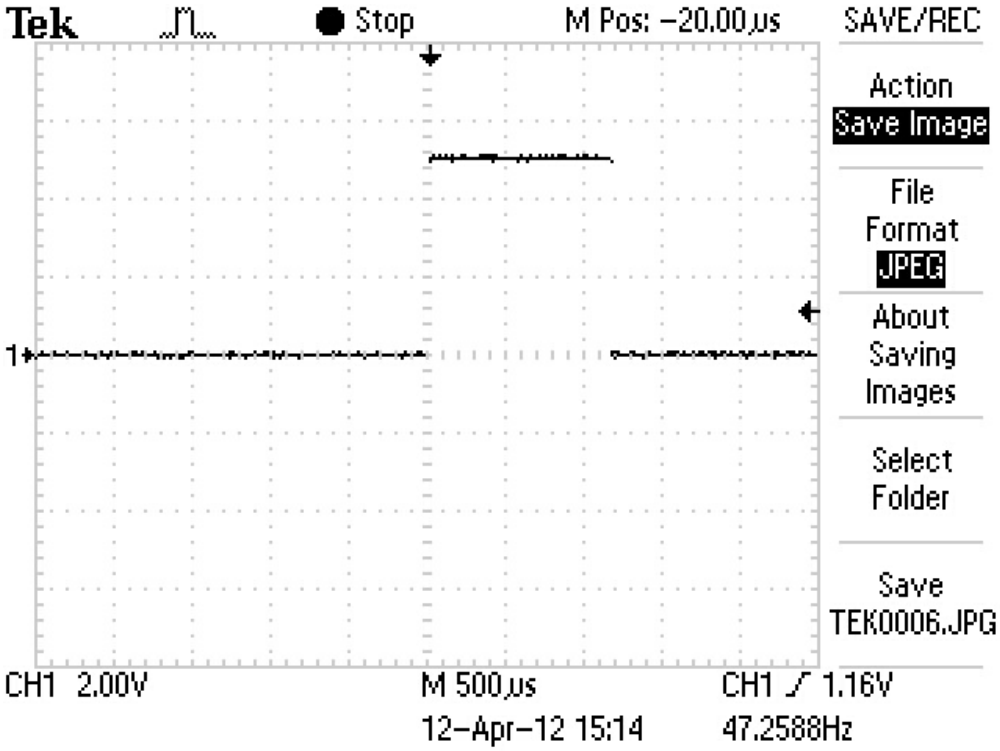


Figure 5.7: Control Signal When Leftmost Line Detected

Previously the servo motor was powered up by 5 V and eventually the response was found out to be slow. After the power supply had changed to 6 V, the response of the servo motor and the torque generated were obviously improved.

5.3 Wireless Communication

The wireless communication was established successfully by using the Bluetooth Serial Port Plug. It was essentially useful during the test run of the smart car so that the smart car can be stopped immediately once it ran out of racing track to prevent any collision and cause any damages to the smart car.

5.4 Outcome in The Freescale Cup 2011

The Freescale Cup 2011 competition was held on 14th and 15th of September 2011 successfully. Figure 5.8 shows the smart car in the parking area during the competition.

The smart car had completed qualifying round and proceeded to the final round successfully. Subsequently the smart was tuned to higher speed to compete in the final round.

Unfortunately, the smart car ran out of the racing track before it can complete the full race. This was due to some dark areas were existed outside of the racing track and the camera had misinterpreted the image and caused the smart car to run out of the track.

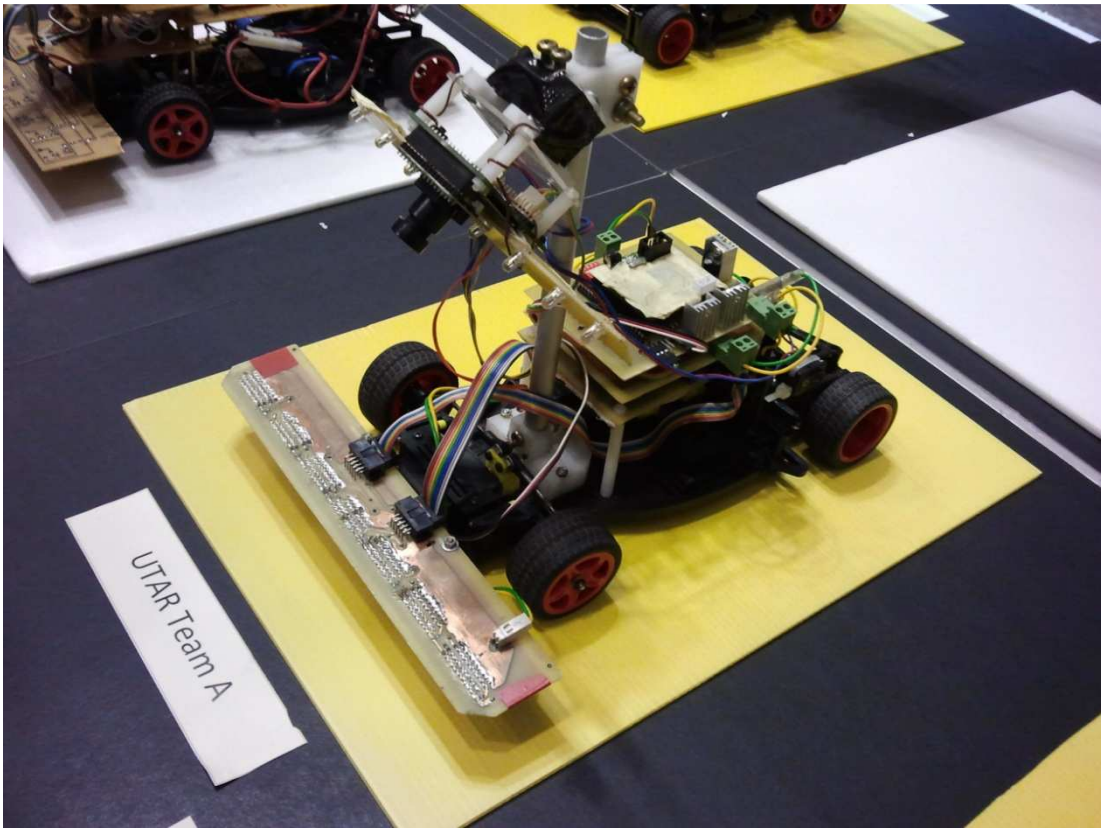


Figure 5.8: Smart Car in The Freescale Cup 2011

CHAPTER 6

RESULTS AND DISCUSSION FOR PATH RECOGNITION SYSTEM

6.1 Road Tracking

The CMUcam3 was successfully setup to track the black colour road and travel accordingly until it reached the junction.

Figure 6.1 shows that the vehicle was in the middle of the road and the tracking result output a mx value of 46. Whereas Figure 6.2 and Figure 6.3 show a mx value of 34 and 55 respectively when the vehicle was slanted to the right side and to the left side of the road.

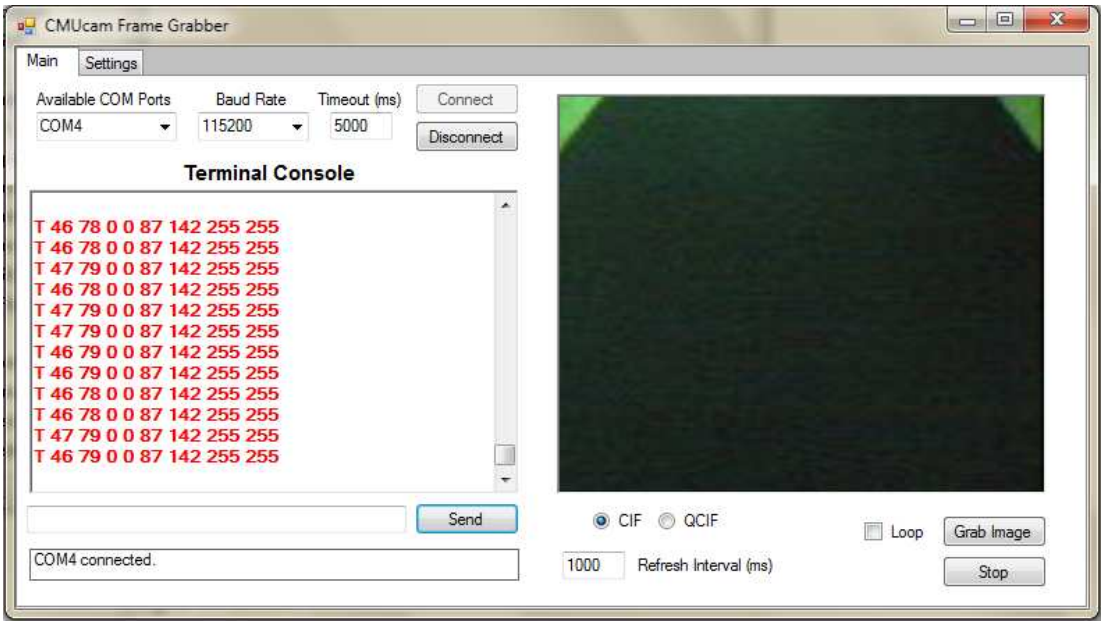


Figure 6.1: Road Tracking Result (Middle)

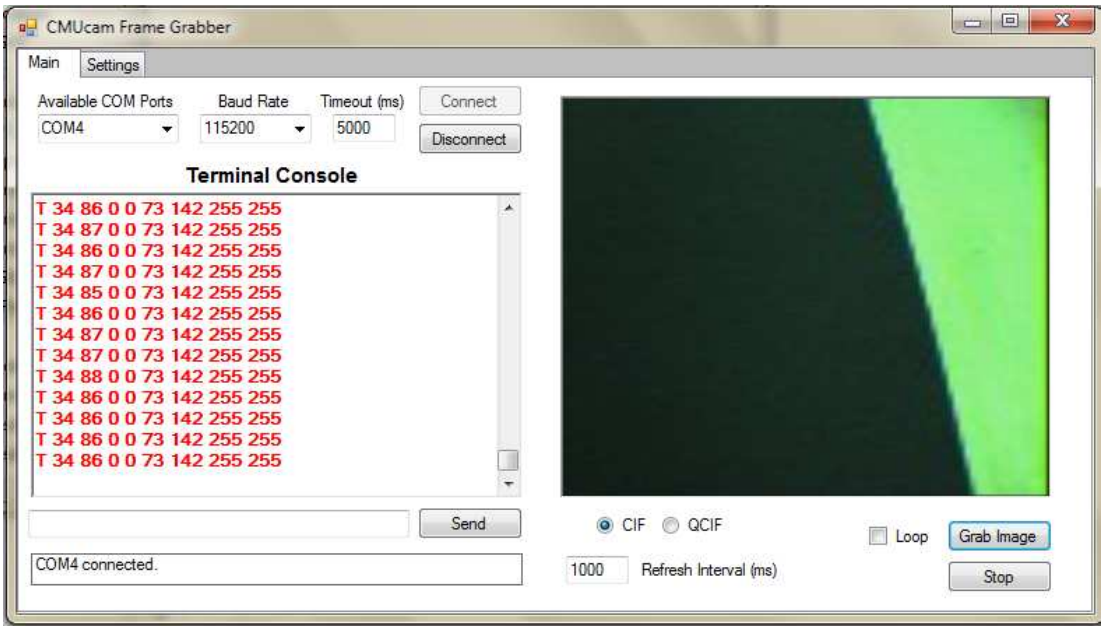


Figure 6.2: Road Tracking Result (Right)

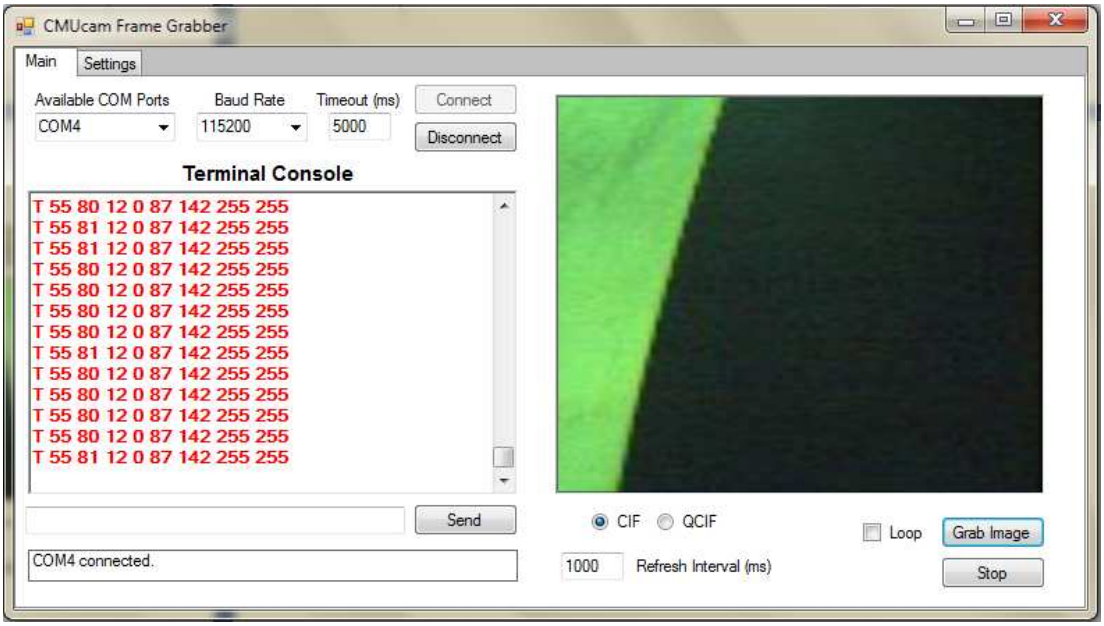


Figure 6.3: Road Tracking Result (Left)

The mx value from the CMUcam3 was useful to represent the orientation of the vehicle with respect to the road. Therefore, the mx value was essential to keep the vehicle travels in the middle of the road through controlling the servo motor.

6.2 RFID Tags Detection

Figure 6.4 illustrates the usage of a terminal software called Docklight to test the RFID module's reading function. The RFID tags were detected by the RFID module when the tags were in the vicinity of the detection range which was up to 80 mm. The status pin of the RFID module can be read by the sb-RIO to determine when a tag was detected.

The GET INFO command was sent and various responds of different conditions such as no tag detected, read failed, and read success with the RFID Tag unique ID following at the back were shown in the Figure 6.4.

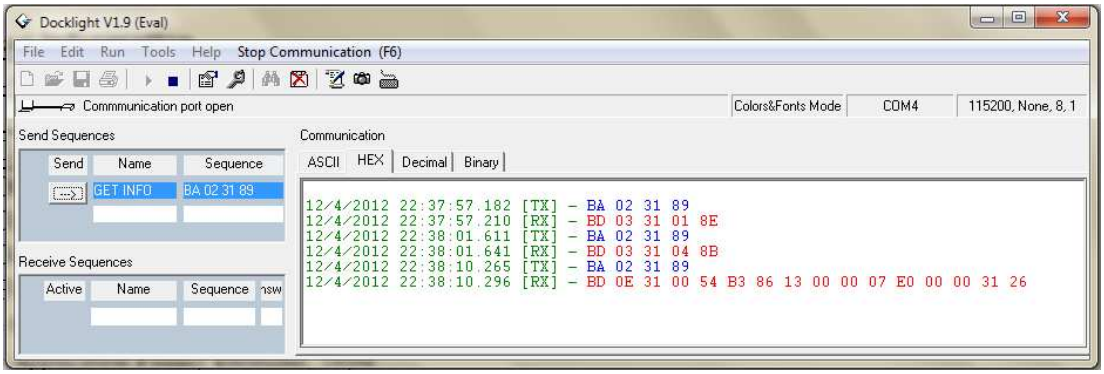


Figure 6.4: Reading of RFID Tag

6.3 Servo Motor Control

The servo motor control signal generated with the FPGA digital IO pin had a amplitude of 3.3 V, which was lesser than the requirement of 5 V. However, 3.3 V was detected as “high” as well and the servo motor was tested to be able to control by the signal generated by the FPGA.

Figure 6.5 shows the signal with 1.5 ms pulse to maintain the servo motor at the middle position. On the other hand, Figure 6.6 and Figure 6.7 illustrate the signal with 0.5 ms and 2.5 ms which were used to steer the servo motor to the leftmost and rightmost position respectively.

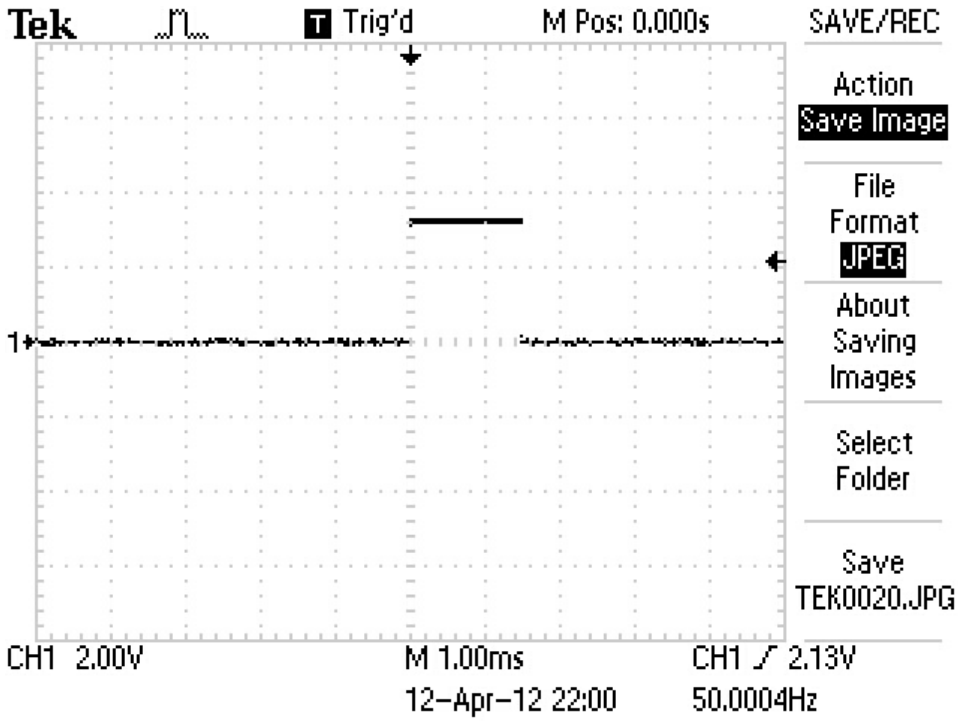


Figure 6.5: Control Signal (1.5 ms Pulse)

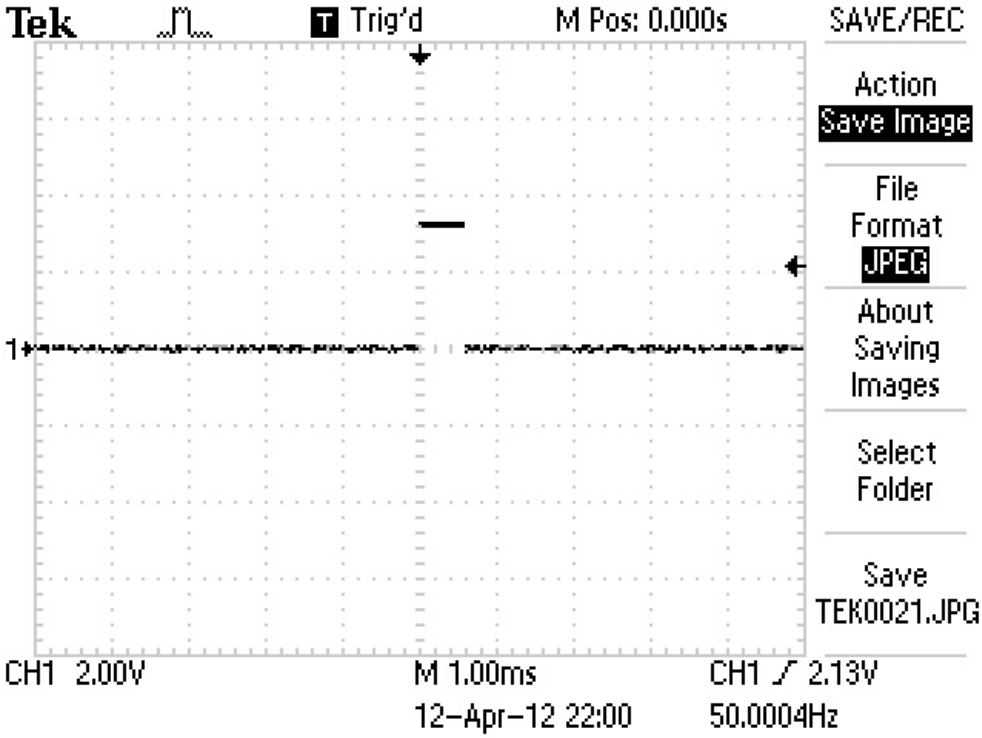


Figure 6.6: Control Signal (0.5 ms Pulse)

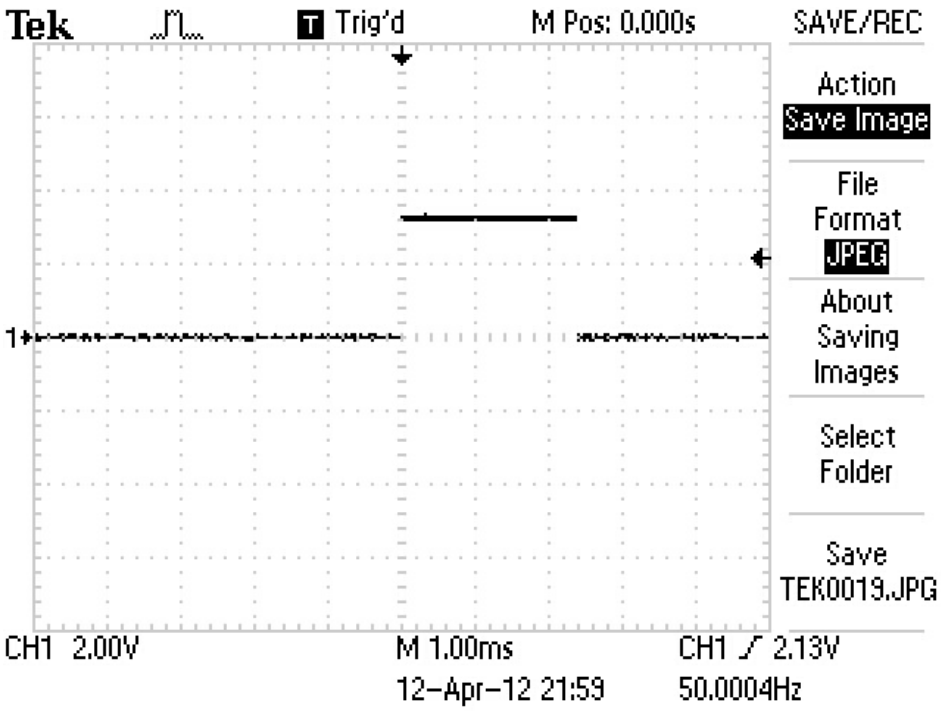


Figure 6.7: Control Signal (2.5 ms Pulse)

6.4 UART Emulation in FPGA

The UART function was successfully emulated in the FPGA, a Graphical User Interface shown in Figure 6.8 was created to set the tracking parameters of the CMUcam3 and to show the tracking result.

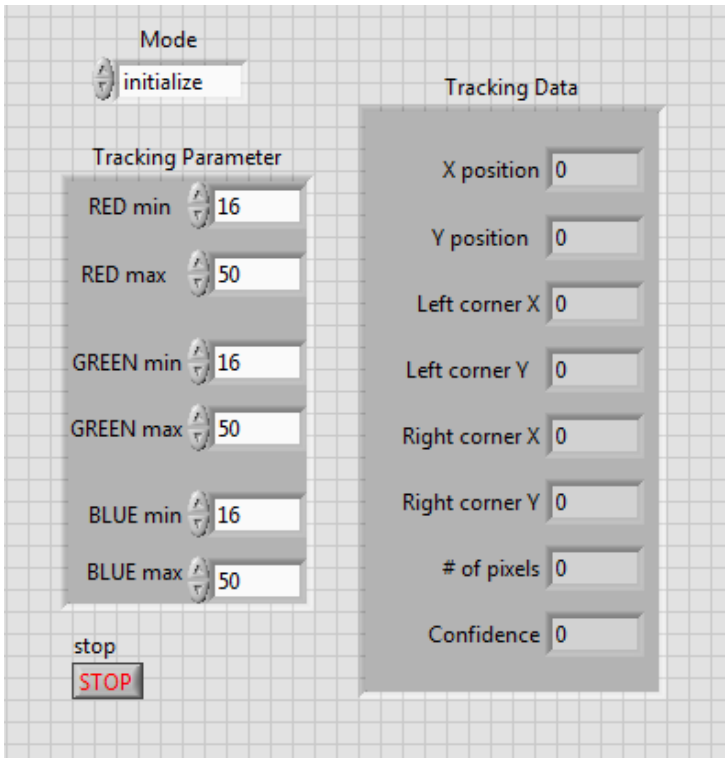


Figure 6.8: Reading of RFID Tag

6.4.1 Conflict of Multiple UART

Two UART functions were required in order to have the CMUcam3 and RFID Module communicated. Although single UART was successfully implemented but when the UART functions were duplicated to create a second UART interface, an error was shown due to some internal conflicts.

The problem was unsolved hence the communication of RFID module with the sb-RIO was not established.

6.5 Outcome

Figure 6.9 demonstrate the experimental setup of the autonomous vehicle for the ADRT system. The NI sb-RIO 9632 was mounted on the chassis and two Lithium-Polymer (Li-Po) batteries were placed underneath to power up the whole system.

The CMUcam3 was installed on the car body together with the mounting stand. The height and angle of the CMUcam3 were adjusted to obtain an optimal view of the road to be tracked. The CMUcam3 was connected to the sb-RIO to have serial communication.

Lastly, the RFID module was attached at the frontmost position of the vehicle. This was proven to aid the autonomous vehicle to detect the juntion earlier and identify the route effectively hence the autonomous vehicle can prepare to turn in advance.

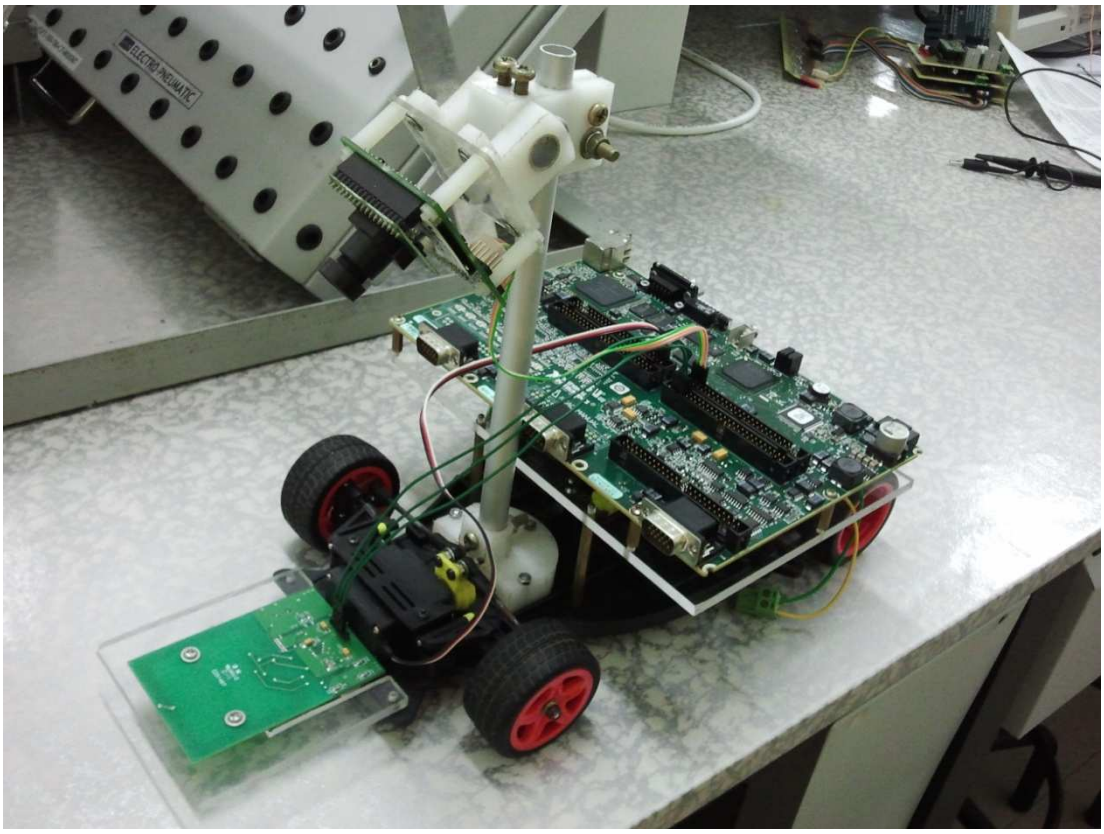


Figure 6.9: Experimental Setup of Autonomous Vehicle

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

This project involves different aspects, ranging from mechanical design, electronic embedded system design, programming of microcontroller and FPGA, motor control, computer to microcontroller interface, wireless communication, and sensors and instrumentations. Every aspects were well studied and understood so that all things can be put together to form a system.

In a nutshell, the objectives of this project have been achieved and it is a very rewarding learning process.

7.1 Line Tracking System

7.1.1 Conclusion

The main objective which is to build a vision guided line following smart car was accomplished by using the CMUcam3to detect the black line on the racing track. The CMUcam3 must be controlled and configured correctly with appropriate parameters in order to perform the line tracking task.

Subsequently, a sophisticated control system was implemented to control the steering of the smart car. The control system was designed based on the PD controller and the parameters were determined through trial and error.

Lastly, a wireless communication was established in between a computer and the smart car. This had enabled the smart car to be started or stopped remotely during the testing stage, which can help to avoid from collision when the smart car was running out of the track.

7.1.2 Recommendation

There are a lot of improvements that can be done in line tracking system. Firstly, the CMUcam3 is an open-source programmable vision sensor module, so a custom source code can be developed to perform a specific task. This provides higher degree of flexibility from developer's point of view in order to improve the tracking performance.

Other than that, computer-based software can be built to establish wireless communication with the smart car instead of using the text-based terminal software. This offers a user-friendly interface and more functions such as Real-Time monitoring can be included.

Last but not least, the control system for steering control module can be more robust and more intelligent by using Fuzzy Logic controller or combine it to form a Fuzzy-PID controller to have superior control.

7.2 Path Recognition System

7.2.1 Conclusion

The road tracking function of the ADRT was achieved by using the CMUcam3 colour tracking function. By setting the correct threshold, the black colour road can be tracked successfully.

With the tracking output of the CMUcam3, the servo motor can be controlled so that the vehicle can be steered autonomously to maintain its position in the middle of the road. The servo motor requires 5 V signal but the 3.3 V signal generated by the FPGA was tested compatible too.

Other than that, the route identification function was accomplished with the RFID module mounted at the front of the autonomous vehicle to detect the RFID tags placed on each junction. As every RFID tag carries an unique ID, therefore each junction can be distinguished. This feature eliminates the reliance on the vision system to identify the junctions.

However, a conflict arose when the UART was duplicated to have the second serial communication interface for the RFID module. This problem had caused the communication in between the RFID module and sb-RIO cannot be established.

7.2.2 Recommendation

Multiple UART were required in this ADRT system but only one UART can be emulated in the FPGA. The proposed solution to this problem is to use the NI 9870 module which is shown in Figure 7.1. This NI 9870 is a serial module that can be attached to the sb-RIO directly to add four serial ports to the system. The four serial ports are directly accessible from the FPGA to offer flexibility in communicating serial devices.

Figure 7.1: NI 9870

REFERENCES

Araki, M., PID Control. Control Systems, Robotics, and Automation – Vol II

Ben, H., Zoran, N., Tim, B., Saeid, N., Philip, C. (2011). OzTug Mobile Robot for Manufacturing Transportation. *IEEE*. 978-1-4577-0653-0

Chapter10: UART Module. (n. d.). Retrieved March 14, 2012. from http://www.mikroe.com/eng/chapters/view/58/chapter-10-uart-module/

Colak, I., & Yildirim, D. (2009). Evolving a Line Following Robot to Use in Shopping Centers for Entertainment. *IEEE*. 978-1-4244-4649-0

Introduction to I2C and SPI Protocols. (2010, July 30). Retrieved March 15, 2012. from http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html

Ismail, A. H., Ramli, H. R., Ahmad, M. H., Marhaban, M. H. (2009). Vision-based System for Line Following Mobile Robot. *IEEE*. 978-1-4244-4683-4.

Jean, D., & Marc, P. (2006). Evolving a Vision-Based Line-Following Robot Controller. *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*

Litwiller, D. (2001, January). CCD vs CMOS: Facts and Fiction. *Issue of Photonics Spectra*. Laurin Publishing Co. Inc.

Quadrifoglio, L., & Li, X. (2009). A Methodology to Derive the Critical Demand Density for Designing and Operating Feeder Transit Services. *Transportation Research Part B (43)*, 922-935.

Thirumurugan, J., Vinoth, M., Kartheeswaran, G., Vishwanathan, M., (2010). Line Following Robot for Library Inventory Management System. IEEE. 978-1-4244-9005-9

APPENDICES

APPENDIX A: Main Program

```
/* header files */
#include <hidef.h>      /* common defines and macros */
#include <MC9S12XS128.h>/* derivative-specific definitions */
#include "math.h"
#include "stdlib.h"
#include "System.h"
#include "PWM.h"
#include "timer_2.h"
#include "timer_3.h"
#include "CMU.h"
#include "SPI.h"
#include "test.h"
#include "parameter.h"

#pragma LINK_INFO DERIVATIVE "MC9S12XS128"
#pragma CODE_SEG DEFAULT

/* global variables */
unsigned char startgo=0;
int ViewError, View;
int SpeedRef = 0;
long ServoOut = 15000;
int CurrView = 44;
int Curr__View = 44;
long LastView = 44;
long LastViewError = 0;
long LastLastViewError = 0;
int SetView = 44;
int path = 0;
int Brake = 0;
int Brake1= 5;
int lastpath = 0;
int ViewStore_Num;
int Store_View[5] = {44};
int StoreView_Accum = 0;
int PassStore_Num;
char Store_Pass[7] = {STRAIGHT};
int ViewPass[5] = {44};
int Kp_Servo = 0;
int Kd_Servo = 0;
unsigned int MainCount=0;
```

```
int PathError=0;
int LastPathError=0;
int PathErrorRecord=0;
int ViewPass_Num;
int ViewPassTotal=0;
int Kp_IR=0;
int Kp_IR1= 80;
int CounterL=0, CounterR=0;
unsigned int count=0;

/* external variables */
extern int PosError;
extern int Position;
extern long angle;
extern int Sp_Factor;
extern unsigned char mx;
extern unsigned char cTPacket[8];
extern CMU_Complete;


void main(void)
{
/* Initialization */
      MCU_init();
      EnableInterrupts;
      CRGFLG = 0xE6;
      _DISABLE_COP();   //disable watchdog timer
      encoder_init();
      SPI_Init();
      SPI_Init();
      TIMER_START;
      servo_init();
      PWM_init();
      Timer3_init();
      BT_init();

      while(SW1==1&&startgo==0);  //wait for the start button
      CMU_init();
      Start_CMU();
      Timer2_init();

/* main control loop */
      while(1)
      {
            CMU_Complete = 0;
            Start_CMU(); // call line tracking function
            CurrView = (int)(mx);

            if(CurrView == 0)
            {
                  CurrView = LastView;
                  MainCount++;
            }
            else MainCount = 0;

/* error calculation */
            LastView = CurrView;
            LastPathError=PathError;
            PathError=CurrView-SetView;
```

```
		if(PathError>LastPathError)
			PathErrorRecord+=1;
		else if(PathError<LastPathError)
			PathErrorRecord-=1;

		if(PathErrorRecord>10)
			PathErrorRecord = 10;
		else if(PathErrorRecord<-10)
			PathErrorRecord = -10;

		ViewPassTotal=0;

		for(ViewPass_Num=0;ViewPass_Num<4;ViewPass_Num++)
		{
			ViewPass[ViewPass_Num]=ViewPass[ViewPass_Num+1];
			ViewPassTotal += ViewPass[ViewPass_Num];
		}

		ViewPass[4] = CurrView;
		ViewPassTotal += ViewPass[4];
		ViewPassTotal /= 5;

		if(ViewPassTotal >= 70 && PathErrorRecord >= 7)
			path = BIG_R_TURN;
		else if(ViewPassTotal <= 18 && PathErrorRecord <= -7)
			path = BIG_L_TURN;
		else if(ViewPassTotal >= 60 && PathErrorRecord < 7 &&
		PathErrorRecord >=4)
			path = SMALL_R_TURN;
		else if(ViewPassTotal <= 28 && PathErrorRecord > -7 &&
		PathErrorRecord <=-4)
			path = SMALL_L_TURN;
		else
			path = STRAIGHT;

		lastpath = path;

/* path classification */
		switch(path)
		{
			case STRAIGHT:
			{
				LED1 = LED2 = LED3 = LED4 = 0;
				switch(lastpath)
				{
					case STRAIGHT:
					{
						SetView = CENTER_SET;
						SpeedRef = TOP_SPEED;
						Kp_Servo = 2;
						Kd_Servo = 0;
						Kp_IR= 30;
						Brake = 0;
						break;
					}

					case SMALL_L_TURN:
					{
						SetView = CENTER_SET;
						SpeedRef = NORMAL_SPEED;
						Kp_Servo = 2;
```

```
                        Kd_Servo = 0;
                        Kp_IR= 30;
                        Brake = 0;
                        break;
                }

                case SMALL_R_TURN:
                {
                        SetView = CENTER_SET;
                        SpeedRef = NORMAL_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 0;
                        Kp_IR= 30;
                        Brake = 0;
                        break;
                }

                case BIG_L_TURN:
                {
                        SetView = CENTER_SET;
                        SpeedRef = SLOW_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = -80;
                        Kp_IR= 30;
                        Brake = 5;
                        break;
                }

                case BIG_R_TURN:
                {
                        SetView = CENTER_SET;
                        SpeedRef = SLOW_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = -80;
                        Kp_IR= 30;
                        Brake = 5;
                        break;
                }
        }
        break;
}

case SMALL_L_TURN:
{
        LED2_LIGHTUP_ONLY;
        switch(lastpath)
        {
                case STRAIGHT:
                {
                        SetView = RIGHT_LESS_SET;
                        SpeedRef = SLOW_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 5;
                        break;
                }

                case SMALL_L_TURN:
                {
                        SetView = RIGHT_LESS_SET;
```

```
                        SpeedRef = FAST_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 0;
                        break;
                }

                case BIG_L_TURN:
                {
                        SetView = RIGHT_LESS_SET;
                        SpeedRef = NORMAL_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 0;
                        Kp_IR= 40;
                        Brake = 0;
                        break;
                }

                case SMALL_R_TURN:
                {
                        SetView = CENTER_SET;
                        SpeedRef = NORMAL_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 7;
                        break;
                }

                case BIG_R_TURN:
                {
                        SetView = CENTER_SET;
                        SpeedRef = SLOW_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 7;
                        break;
                }
        }
        break;
}

case SMALL_R_TURN:
{
        LED3_LIGHTUP_ONLY;
        switch(lastpath)
        {
                case STRAIGHT:
                {
                        SetView = LEFT_LESS_SET;
                        SpeedRef = SLOW_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 5;
                        break;
                }

                case SMALL_R_TURN:
```

```
                {
                        SetView = LEFT_LESS_SET;
                        SpeedRef = FAST_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 0;
                        break;
                }

                case BIG_R_TURN:
                {
                        SetView = LEFT_LESS_SET;
                        SpeedRef = NORMAL_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 0;
                        Kp_IR= 40;
                        Brake = 0;
                        break;
                }

                case SMALL_L_TURN:
                {
                        SetView = CENTER_SET;
                        SpeedRef = NORMAL_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 7;
                        break;
                }

                case BIG_L_TURN:
                {
                        SetView = CENTER_SET;
                        SpeedRef = SLOW_SPEED;
                        Kp_Servo = 2;
                        Kd_Servo = 80;
                        Kp_IR= 40;
                        Brake = 7;
                        break;
                }
        }
        break;
}

case BIG_L_TURN:
{
        LED1_LIGHTUP_ONLY;
        switch(lastpath)
        {
                case STRAIGHT:
                {
                        SetView = RIGHT_MORE_SET;
                        SpeedRef = SLOW_SPEED;
                        Kp_Servo = 3;
                        Kd_Servo = 160;
                        Kp_IR= 40;
                        Brake = 7;
                        break;
                }
```

```
                case SMALL_L_TURN:
                {
                      SetView = RIGHT_MORE_SET;
                      SpeedRef = NORMAL_SPEED;
                      Kp_Servo = 2;
                      Kd_Servo = 160;
                      Kp_IR= 40;
                      Brake = 6;
                      break;
                }

                case BIG_L_TURN:
                {
                      SetView = RIGHT_MORE_SET;
                      SpeedRef = FAST_SPEED;
                      Kp_Servo = 2;
                      Kd_Servo = 160;
                      Kp_IR= 40;
                      Brake = 5;
                      break;
                }

                case SMALL_R_TURN:
                {
                      SetView = RIGHT_LESS_SET;
                      SpeedRef = SLOW_SPEED;
                      Kp_Servo = 3;
                      Kd_Servo = 160;
                      Kp_IR= 40;
                      Brake = 7;
                      break;
                }

                case BIG_R_TURN:
                {
                      SetView = RIGHT_LESS_SET;
                      SpeedRef = SLOW_SPEED;
                      Kp_Servo = 2;
                      Kd_Servo = 160;
                      Kp_IR= 40;
                      Brake = 7;
                      break;
                }
          }
          break;
    }

    case BIG_R_TURN:
    {
          LED4_LIGHTUP_ONLY;
          switch(lastpath)
          {
                case STRAIGHT:
                {
                      SetView = LEFT_MORE_SET;
                      SpeedRef = SLOW_SPEED;
                      Kp_Servo = 3;
                      Kd_Servo = 160;
                      Kp_IR= 40;
                      Brake = 7;
```

```
                                    break;
                                }

                                case SMALL_R_TURN:
                                {
                                    SetView = LEFT_MORE_SET;
                                    SpeedRef = NORMAL_SPEED;
                                    Kp_Servo = 2;
                                    Kd_Servo = 160;
                                    Kp_IR= 40;
                                    Brake = 6;
                                    break;
                                }

                                case BIG_R_TURN:
                                {
                                    SetView = LEFT_MORE_SET;
                                    SpeedRef = FAST_SPEED;
                                    Kp_Servo = 2;
                                    Kd_Servo = 160;
                                    Kp_IR= 40;
                                    Brake = 5;
                                    break;
                                }

                                case SMALL_L_TURN:
                                {
                                    SetView = LEFT_LESS_SET;
                                    SpeedRef = SLOW_SPEED;
                                    Kp_Servo = 3;
                                    Kd_Servo = 160;
                                    Kp_IR= 40;
                                    Brake = 7;
                                    break;
                                }

                                case BIG_L_TURN:
                                {
                                    SetView = LEFT_LESS_SET;
                                    SpeedRef = SLOW_SPEED;
                                    Kp_Servo = 2;
                                    Kd_Servo = 160;
                                    Kp_IR= 40;
                                    Brake = 7;
                                    break;
                                }
                            }
                            break;
                        }
                    }

/* servo control */
                    LastLastViewError = LastViewError;
                    LastViewError = ViewError;
                    ViewError = CurrView-SetView;

                    if(ViewError > 0)
                        ServoOut = ServoPos_2 + (Kp_Servo *  (ViewError *
                        ViewError))+ Kd_Servo * (ViewError-
                        LastLastViewError)/2;
                    else if(ViewError < 0)
```

```
                  ServoOut = ServoPos_2 - (Kp_Servo *  (ViewError *
                  ViewError))+ Kd_Servo * (ViewError-
                  LastLastViewError)/2;
            else
                  ServoOut = ServoPos_2;

            if(ServoOut > HighServoLimit)
                  ServoOut = HighServoLimit;
            else if(ServoOut < LowServoLimit)
                  ServoOut = LowServoLimit;

            if(PosError>=5)
            {
                  if(CounterL<400)
                        CounterL++;
                  ServoOut -= (Kp_IR1*CounterL);
            }
            else if(PosError<=-5)
            {
                  if(CounterR<400)CounterR++;
                        ServoOut += (Kp_IR1*CounterR);
            }
            else
            {
                  CounterL=CounterR=0;
            }

            if(ServoOut > HighServoLimit)
                  ServoOut = HighServoLimit;
            else if(ServoOut < LowServoLimit)
                  ServoOut = LowServoLimit;

            angle = ServoOut;
      }
}
```

APPENDIX B: CMUcam3 Program

```
#include "CMU.h"
unsigned char cState;
unsigned char cMode =1;
unsigned char cRead[30];
unsigned char cSend[30];
unsigned char *cTxPtr;
unsigned char *cRxPtr;
unsigned char cTPacket[8];
unsigned char mx;
unsigned char CMU_Complete = 0;
unsigned char cmdGV[] = "GV\r";
unsigned char cmdPM[] = "PM 0\r";
unsigned char cmdST[] = "ST 16 30 16 30 16 30\r";
unsigned char cmdTC[] = "TC 16 30 16 30 16 30\r";
unsigned char cmdLM[] = "LM 0 2\r";
unsigned char cmdOM[] = "OM 0 1\r";    // show mx only
unsigned char cmdVW[] = "VW 1 40 87 60\r";
unsigned char cmdDS[] = "DS 1 3\r";
unsigned char cCheck=0;

//Functions
void CMU_init (void)
{
      SCI1BDL = 0x16; // baud rate = 115200
      SCI1BDH = 0x00;
      SCI1CR1 = 0x00;
      SCI1CR2 = 0x2C;
      cRxPtr = &cRead[0];
      cState = cIdle;
}

unsigned char CMU_check (void)
{
      if (cState == cRxed)
      {
            cState = cIdle;
            cRxPtr = &cRead[0];
            switch(cRead[0])
            {
                  case 'A':
                    if(cRead[4] == 'T')
                    {
                        cDecodeT();
                        mx = cTPacket[0];
                        CMU_Complete = 1;
                        return cTP;
                    }
                    else
                        return cACK;
```

```
                  case 'N':
                        return cNCK;

                  case 'T':
                        cDecodeT();
                        mx = cTPacket[0];
                        CMU_Complete = 1;
                        return cTP;

                  default:
                        return 'N';
            }
      }
      else
            return 0;
}

void cDecodeT()
{
      unsigned char dS = 6;
      unsigned char dE = 6;
      unsigned char mul;
      unsigned char i = 0;
      unsigned char j = 0;

      while(i < 8)
      {
            mul = 1;
            cTPacket[i] = 0;
            while(cRead[dE] > ' ' )
                  dE++;
            for(j = dE-1; j>= dS; j--)
            {
                  cTPacket[i] += (cRead[j] & 0x0F) * mul;
                  mul *= 10;
            }
            dE++;;
            dS = dE;
            i++;
      }
}

void cGV (void)
{
      cTxPtr = &cmdGV[0];
      cState = cBusy;
      SCI1DRL = *cTxPtr;
      SCI1CR2_TIE = 1;
}

void cST (void)
{
      cTxPtr = &cmdST[0];
      cState = cBusy;
      SCI1DRL = *cTxPtr;
      SCI1CR2_TIE = 1;
}

void cTC (void)
{
      cTxPtr = &cmdTC[0];
```

```
	cState = cBusy;
	SCI1DRL = *cTxPtr;
	SCI1CR2_TIE = 1;
}

void cPM (unsigned char mode)
{
	cTxPtr = &cmdPM[0];
	if(mode)
		cmdPM[3] = '1';
	else
		cmdPM[3] = '0';
	cState = cBusy;
	SCI1DRL = *cTxPtr;
	SCI1CR2_TIE = 1;
}

void cLM (void)
{
	cTxPtr = &cmdLM[0];
	cState = cBusy;
	SCI1DRL = *cTxPtr;
	SCI1CR2_TIE = 1;
}

void cOM (void)
{
	cTxPtr = &cmdOM[0];
	cState = cBusy;
	SCI1DRL = *cTxPtr;
	SCI1CR2_TIE = 1;
}

void cVW (void)
{
	cTxPtr = &cmdVW[0];
	cState = cBusy;
	SCI1DRL = *cTxPtr;
	SCI1CR2_TIE = 1;
}

void cDS (void)
{
	cTxPtr = &cmdDS[0];
	cState = cBusy;
	SCI1DRL = *cTxPtr;
	SCI1CR2_TIE = 1;
}

void CMU_write (void)
{
	unsigned char temp;
	temp = SCI1SR1;
	SCI1SR1_TDRE = 1;
	temp = SCI1SR1;
	SCI1SR1_TC = 1;
	if(cState == cBusy)
	{
		cTxPtr++;
		if(*cTxPtr)
			SCI1DRL=*cTxPtr;
```

```
            else
            {
                  cState = cIdle;
                  SCI1CR2_TIE = 0;
            }
      }
}

void CMU_read (void)
{
      unsigned char data;
      SCI1SR1_RDRF=1;
      data = SCI1DRL;
      if(cMode)
      {
            if (data != ':')
            {
                  *cRxPtr = data;
                  if(*cRxPtr == 13 && cMode != 1)
                        cMode = 0;
                  if(cMode == 1)
                  {
                        if(*cRxPtr == 13)
                              cMode = 2;
                  }
                  else
                        cRxPtr++;
                  }
                  else
                        cState = cRxed;
            }
            else if (data != ':' )
            {
                  *cRxPtr = data;
                  cRxPtr++;
            }
            else
                  cState = cRxed;
}

void Start_CMU()
{
      cCheck = 0;
      cGV();
      while(cCheck==0)
      {
            cCheck = CMU_check();
            if(cCheck == cNCK)
            {
                  cGV();
                  cCheck = 0;
            }
      }

      cCheck=0;
      cPM(1);
      while(cCheck == 0)
      {
            cCheck = CMU_check();
            if(cCheck == cNCK)
            {
```

```
                        cPM(1);
                        cCheck = 0;
                }

        }
        cST();
        while(cCheck==0)
        {
                cCheck = CMU_check();
                if(cCheck == cNCK)
                {
                        cST();
                        cCheck = 0;
                }
        }
        cCheck=0;

        cVW();
        while(cCheck==0)
        {
                cCheck = CMU_check();
                if(cCheck == cNCK)
                {
                        cVW();
                        cCheck = 0;
                }
        }
        cCheck=0;

        cOM();
        while(cCheck==0)
        {
                cCheck = CMU_check();
                if(cCheck == cNCK)
                {
                        cOM();
                        cCheck = 0;
                }
        }
        cCheck=0;
        cTC();
        while(cCheck==0)
        {
                cCheck = CMU_check();
                if (cCheck == cNCK)
                {
                        cTC();
                        cCheck = 0;
                }
        }
        cCheck=0;
}

void End_CMU()
{
        cTC();
        while(cCheck==0)
        {
                cCheck = CMU_check();
                if (cCheck == cNCK)
                {
```

```
                cTC();
                cCheck = 0;
            }
    }
    cCheck=0;
}

/* SCI1 interrupt */
#pragma CODE_SEG __SHORT_SEG NON_BANKED
#pragma TRAP_PROC
void interrupt 21 UART_ISR(void)
{
      {__asm SEI;} //disable interrupts

      if(SCI1SR1&0X80)
            CMU_write();

      if(SCI1SR1_RDRF==1)
            CMU_read();

      {__asm CLI;} //enable interrupts
}
#pragma CODE_SEG DEFAULT
```

APPENDIX C: Bluetooth Serial Port Program

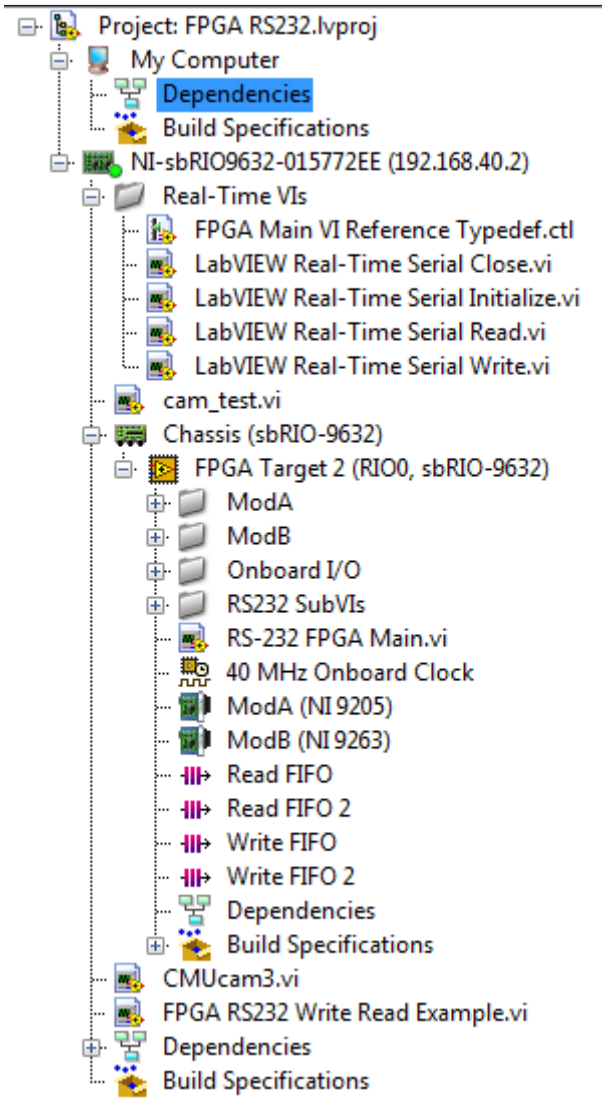
```
#include "BT.h"
#include "System.h"
#include "PWM.h"
#include "timer_3.h"

unsigned char bRead[10];
unsigned char bSend[10];
unsigned char *bTxPtr;
unsigned char *bRxPtr;
extern unsigned char mx,startgo;
extern int RightMPS;
extern long speedrecordL[10],speedrecordR[10];
char temp123[50];
char tempmx[]= "00\r";
long tempspeed,temp;
unsigned char testString[] = "Hello!\r";

void BT_init (void)
{
      SCI0BDL = 0x16; // baud rate = 115200
      SCI0BDH = 0x00;
      SCI0CR1 = 0x00;
      SCI0CR2 = 0x2C;
      bRxPtr = &bRead[0];
}

void test (void)
{
      bTxPtr = &testString[0];
      SCI0DRL = *bTxPtr;
      SCI0CR2_TIE = 1;
}

void send_mx (void)
{
      temp=(int)mx;
      tempmx[0] = temp/10;
      tempmx[1] = temp-10*tempmx[0];
      bTxPtr = &tempmx[0];
      SCI0DRL = *bTxPtr;
      SCI0CR2_TIE = 1;
}

void BT_write (void)
{
      bTxPtr++;
      if(*bTxPtr)
            SCI0DRL=*bTxPtr;
```

```
	else
		SCI0CR2_TIE = 0;
}

void BT_read (void)
{
	int i;
	unsigned char data;
	SCI0SR1_RDRF=1;
	data = SCI0DRL;
	switch (data)
	{
		case 'a': // Start running
		startgo=1;
		break;
		case 'c': // camera output
			send_mx();
		break;
		case 'p': // IR position
			IR_value();
		break;
		case 's': // emergency stop
			LEFTF=0;
			RIGHTF=0;
			while(1);
		break;
		case 'v': // speed measured
			tempspeed=0;
			tempspeed=RightMPS;
			temp123[0] = tempspeed/1000 + 48;
			tempspeed-=((temp123[0]-48)*1000);
			temp123[1] = tempspeed/100 + 48;
			tempspeed-=((temp123[1]-48)*100);
			temp123[2] = tempspeed/10 + 48;
			tempspeed-=((temp123[2]-48)*10);
			temp123[3] = tempspeed + 48;
			bTxPtr = &temp123[0];
			SCI0DRL = *bTxPtr;
			SCI0CR2_TIE = 1;
		break;
		default:
		break;
	}
}

/* SCI0 interrupt */
#pragma CODE_SEG __SHORT_SEG NON_BANKED
#pragma TRAP_PROC
void interrupt 20 UART0_ISR(void)
{
	{__asm SEI;} //DisableInterrupts;
	if(SCI0SR1&0X80)
		BT_write();
	if(SCI0SR1_RDRF==1)
		BT_read();
	{__asm CLI;}  //EnableInterrupts
}
#pragma CODE_SEG DEFAULT
```

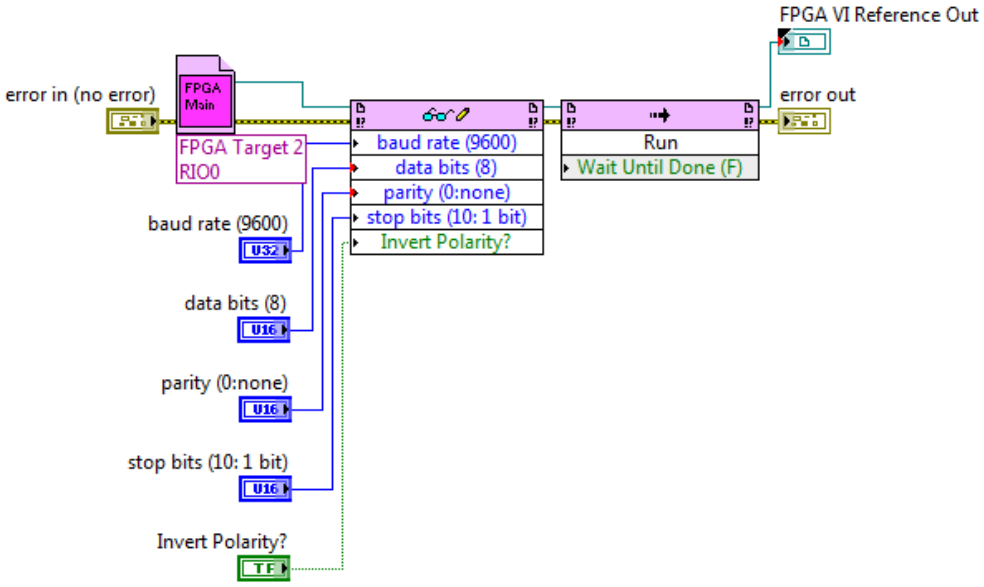
APPENDIX D: Servo Control Program

```
#include <hidef.h>      // common defines and macros
#include <MC9S12XS128.h>// derivative-specific definitions
#include "servo.h"
#include "System.h"

#pragma LINK_INFO DERIVATIVE "MC9S12XS128"
#pragma CODE_SEG DEFAULT

int t = 0;
long angle;

void servo_init(void)
{
      angle = 15000;   //initial angle become center
      SERVO = 0;
      TIOS_IOS0 = 1;    //Enable the 1 to OC mode
      TFLG1 |= 0x01;   //reset interrupt flags of CH1
      TIE |= 0x01;     //enable interrupt CH1
}

#pragma CODE_SEG NON_BANKED
interrupt 8 void timer0(void)
{
      CLEAR_T0_INTFLAG;
      if(SERVO == 1)
      {
            SERVO = 0;
            TC0 = TCNT + 50000;
      }
      else
      {
            SERVO = 1;
            TC0 = TCNT + angle/4;
      }
}
```

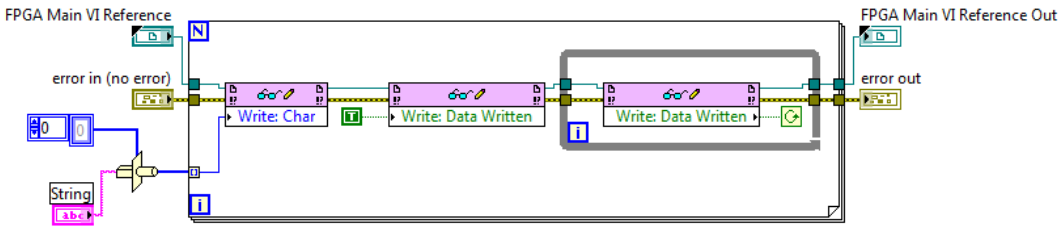
APPENDIX E: LabVIEW Project Explorer



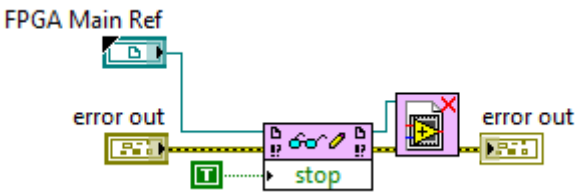
APPENDIX F: LabVIEW UART Functions



UART Initialization Function



UART Write Function



UART Close Function