INTELLIGENT AGV WITH NAVIGATION, OBJECT DETECTION AND AVOIDANCE IN AN UNKNOWN ENVIRONMENT

ELLENOR PETRONELLA BOJE

Dissertation submitted in fulfilment of the requirements for the

MASTER TECHNOLOGIAE: ENGINEERING: ELECTRICAL

in the

School of Electrical and Computer Systems Engineering

of the

Faculty of Engineering, Information and Communication Technology

at the

Central University of Technology, Free State

Promoter: Dr. H.J. Vermaak

Bloemfontein
January 2007

DECLARATION

I, ELLENOR PETRONELLA BOJE, identity number [redacted], and student number 9809740, do hereby declare that this research project which has been submitted to the Central University of Technology for the Degree MASTER TECHNOLOGIAE: ENGINEERING: ELECTRICAL, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology; and has not been submitted before by any person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification.

....   ……………………

SIGNATURE OF STUDENT   DATE

ACKNOWLEDGEMENTS

Firstly, I want to thank the Lord for giving me the ability to complete this study and for providing me with the family and friends that I have. There was never a moment during my studies where I felt that I had a lack of support from my family, friends and colleagues. I would like to thank the following individuals and institutions in particular:

- Herman Vermaak, for his supervision, guidance and advice. I know I was not the perfect student.
- Pieter Veldtsman, for all the help, guidance and inspiration during this project. Without your input I would still be on page one.
- Ben Kotze, for the guidance, advice and motivation.
- Thomas Hoffman and Martin Leonard (two German students) thank you for all your help and inputs with the practical work while doing your internship at CUT. You made my life much easier.
- The Central University of Technology, Free State for the provision of research facilities and financial support.
- I would like to thank my parents for giving me opportunities in life and always believing in me. Thank you for your unconditional love and support.
- To all colleagues and individuals not mentioned here, that helped or supported in any way, thank you.

Finally, I want to thank my husband Carl, for your undying love, extreme support, encouragement and understanding during this study, you light up my life. My little girl, Liebe, you are the apple of my eye, my time is yours again.

SUMMARY

The latest technological trend worldwide, is automation. Reducing human labour and introducing robots to do the work is a pure business decision. The reason for automating a plant can be some, or all, of the following:

- Improve productivity
- Reduce labour and equipment costs
- Reduce product damage
- System reliability can be monitored
- Improves plant safety

When the automation process is started, Automatic Guided Vehicles (AGVs) will be one of the first commodities that can be used. The reason for this is that they are so versatile. They can be programmed to follow specific paths when moving material from one point to another and the biggest advantage of all is that they can operate for twenty four hours a day.

Automatic Guided Vehicles are developed for many different applications and therefore many different types of AGVs are available. All AGVs are equipped with sensors so that they are able to "see" what is happening around them. Since the AGV must be able to function without any human help or control, it must be able to navigate through the work environment. In this study a remote control car was converted to an AGV and thorough research was done on the different types of sensors that can be used to make the AGV more intelligent when it comes to navigating in an unknown environment.

OPSOMMING

Outomatisering is die nuutste neiging wêreldwyd. Deur hande arbeid te verminder en eerder robotte te gebruik om die werk te doen, is definitief ‘n besigheidsbesluit. Redes vir die automatisering van ’n aanleg kan van die volgende of selfs almal insluit:

- Verbeter produktiwiteit
- Verlaag arbeid en toerustingkostes
- Verminder produkbeskadiging
- Stelselbetroubaarheid kan gemonitor word
- Verbeter aanlegveiligheid

Wanneer daar met die outomatiseringsproses begin word, is Outomaties Geleide Voertuie (OGV) een van die eerste hulpmiddels wat gebruik kan word. Die rede hiervoor is dat OGVs so veelsydig is. Die Outomaties Geleide Voertuie kan geprogrammeer word om ‘n spesifieke roete te volg terwyl hulle materiaal van een punt na ‘n ander vervoer. Die grootste voordeel van OGVs is dat hulle vir die volle vier-en-twintig uur van ‘n dag kan werk.

Outomaties Geleide Voertuie word vir verskillende toepassings ontwikkel en daarom is daar baie verskillende tipes OGVs beskikbaar. Alle OGVs word toegerus met sensors om te kan “sien” wat rondom hulle gebeur. Aangesien ‘n Outomaties Geleide Voertuig sonder enige menslike hulp moet kan funksioneer, is dit nodig dat hierdie OGV op sy eie deur die werk omgewing moet kan navigeer. In hierdie studie word ‘n afstandbeheerde motor omgeskakel na ‘n Outomaties Geleide Voertuig en ‘n deeglike navorsingstudie is gedoen op die verskillende tipes sensors wat gebruik kan word om die OGV meer intelligent te maak as dit by navigering in ‘n onbekende omgewing kom.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbreviation	**Description**
2D	Two-dimensional
3D	Three-dimensional
4WD	4-Wheel Drive
AC	Alternating Current
ADC	Analogue to Digital Converter
AGV	Automatic Guided Vehicle
ASCII	American Standard Code for Information Interchange
BDA	Bluetooth Device Address
DC	Direct Current
EMF	Electromagnetic Field
EMI	Electromagnetic Interference
FET	Field-Effect Transistor
GPS	Global Positioning System
I^2C	Inter Integrated Circuit
IR	Infrared
LED	Light Emitting Diode
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
PC	Personal Computer
PCB	Printed Circuit Board
PFM	Pulse-Frequency Modulation
PIC	Programmable Integrated Circuit
PSD	Position Sensing Device
PWM	Pulse-Width Modulation
RC	Remote Control
RF	Radio Frequency
RPM	Revolutions Per Minute
SCL	Synchronous Serial Clock
SDA	I^2C data Input/Output

USART	Universal Serial Asynchronous Receiver Transmitter
USB	Universal Serial Bus
WAN	Wide Area Network
WGAN	Wireless Global Area Network
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WPAN	Wireless Personal Area Network
WWAN	Wireless Wide Area Network

Chapter 1

Introduction to Automatic Guided Vehicles

1.1 Introduction

It is a fact that robots are here to stay. They have become a part of our daily lives, making the execution of tasks effortless. When it comes to automation, robots are frequently used to do the tasks that humans were required to perform a few years back. A mobile robot is often alternately referred to as an Automatic Guided Vehicle (AGV).

AGVs can be used in every step of a production process, from the handling of raw materials up to where the finished product is loaded onto the ship, airplane or truck for transportation to the wholesaler.

AGVs are used in many different industries. Typical industries are [1]:

- Aerospace and defence, see Figure 1.1 [2]
- Automotive, see Figure 1.2 and 1.3 [2]
- Food and beverage
- Paper handling, printing and publishing, see Figure 1.4 [2]
- Chemical processing
- Plastics
- Primary metal industries
- Recycling

- Warehousing

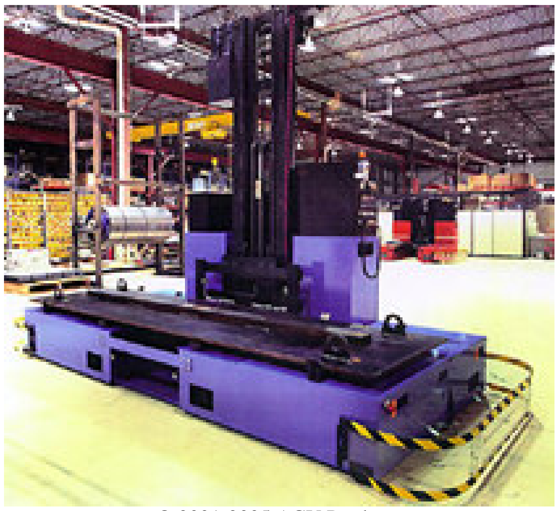

© 2001-2005 AGV Products

Figure 1.1: AGV used in an airplane-manufacturing plant

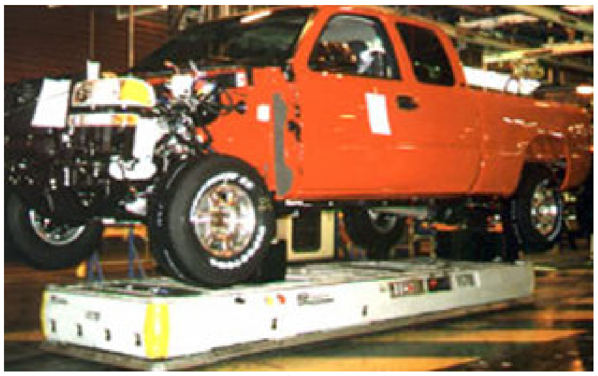
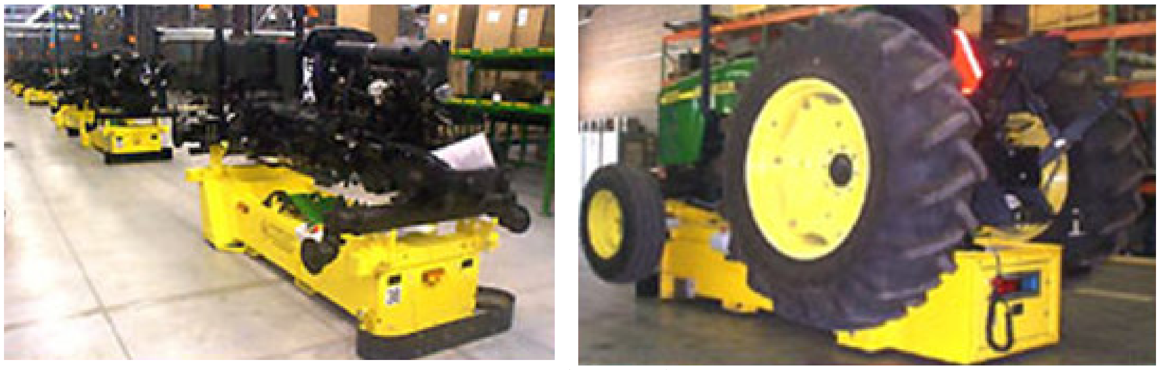
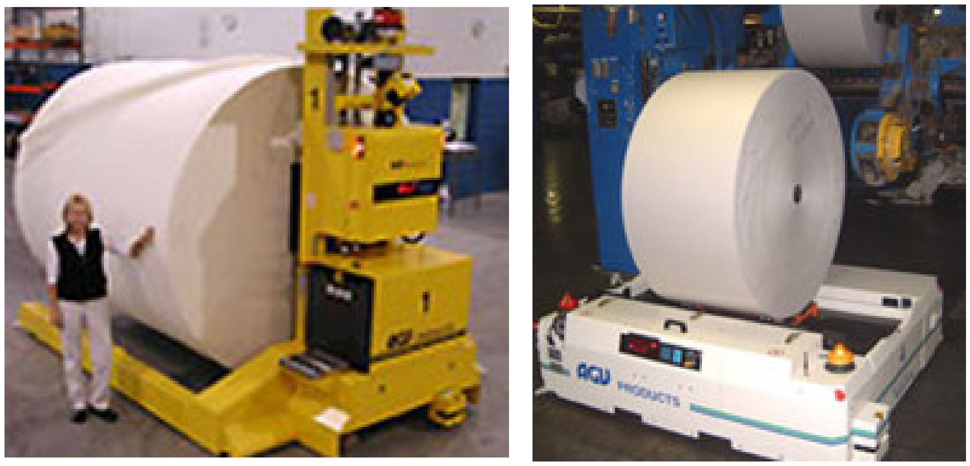

© 2001-2005 AGV Products

Figure 1.2: AGV in the automotive industry



© 2001-2005 AGV Products

Figure 1.3: AGVs used in a tractor-manufacturing plant



© 2001-2005 AGV Products

Figure 1.4: AGVs in the paper-handling industry

Using AGVs offers a lot of advantages, such as [1]:

- Less handling damage
- Can be interfaced with other systems
- AGV provides predictable and reliable operation
- Can move material over long distances
- Improved productivity (can work twenty four hours a day, no sick leave, etc.)

Recently, a lot of research has been published in the field of Automatic Guided Vehicles. In this project the possibility to develop an AGV base making use of a remote control car will be investigated. This newly developed AGV base also needs to be equipped with the correct sensors to be able to function as an AGV in any environment [3].

1.2 Development of the AGV

The first Automatic Guided Vehicles were introduced in the 1950s. The definition of an AGV can be simplified by saying that it is a “driverless” vehicle powered by an electric motor and batteries [4][5], or it can be said that an AGV is an unmanned vehicle that is controlled by a computer [6, p.3]. An ordinary human driven forklift can be seen in Figure 1.5 [7] and the AGV equivalent of a forklift can be seen in Figure 1.6 [1].

Copyright © Toyota Canada Inc.

Figure 1.5: Forklift that needs a human driver

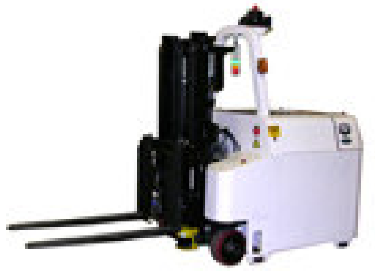


Figure 1.6: Example of an AGV that can do the work of a fork lift

Initially wire guidance was used for navigational purposes. AGVs today, however, in most cases make use of laser guidance. The object detection of AGVs was done by using mechanical bumpers instead of using sensors, which seems to be the current trend.

According to Dematic Global [8], over three thousand AGV systems have been installed during the past 50 years. These systems range from a small number of AGVs, one to three AGVs working together, up to more than 100 AGVs working in one system [8].

The AGV industry is unquestionably growing and more developments in this field are definitely required.

1.3 Problem Statement

What are the characteristics of an AGV and how can an AGV be developed using a remote control car as a starting point? The project objectives are discussed in more detail in Section 1.4.

1.4 Project Objectives

A remote control car is to be taken and converted to form the base of an AGV. This AGV will need a number of sensors to be able to function automatically in any environment. These sensors must address the following problems that an AGV could experience whilst moving in any given environment:

- Automatically controlling the speed and direction of travel
- Observing and avoiding objects that are in the way of the AGV
- Navigate through the environment
- Be monitored using a personal computer (PC) and communicating with this PC to obtain essential data from the AGV that can be stored and used for analysis of the AGV performance

1.5 Hypothesis

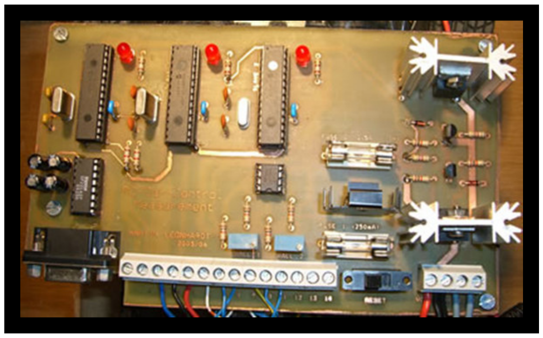
The AGV that will be developed within this study must be able to do the following without any human help:

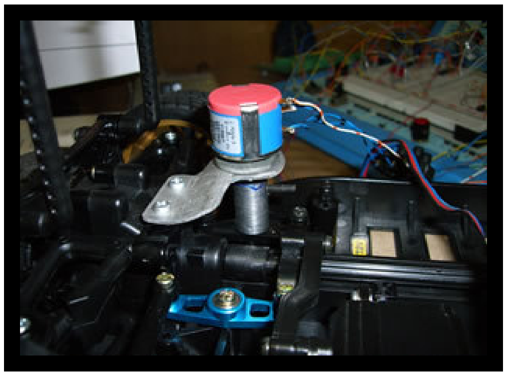
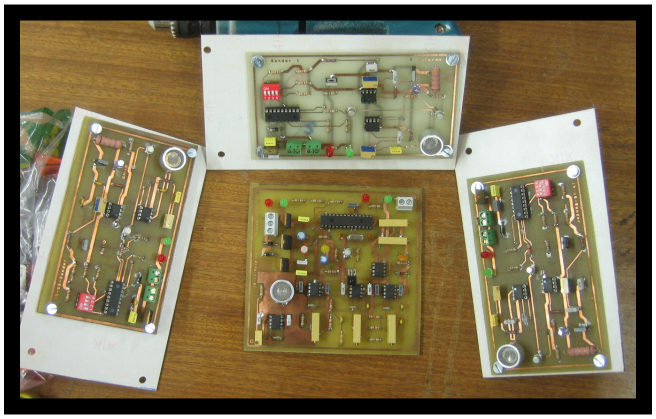
- Control and monitor direction and speed of the AGV
- Navigate through an unknown environment to get from point A to point B
- Detect and avoid any obstacles in its way

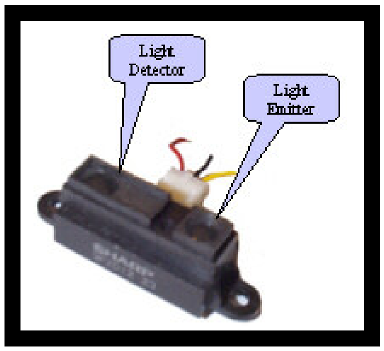
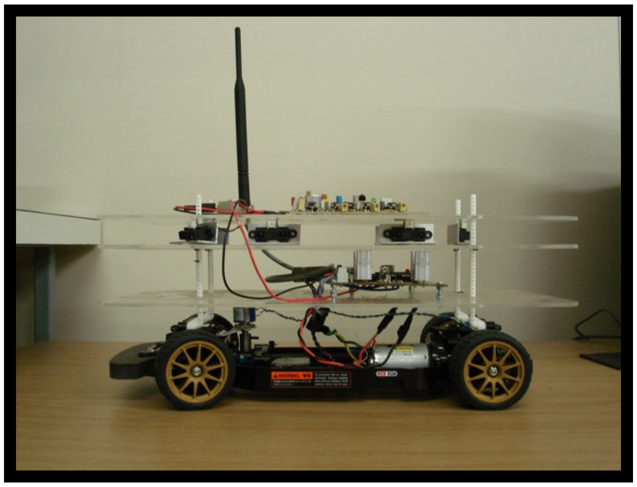
1.6 Research Method

The development cycle of this AGV is shown in Table 1.1:

Table 1.1: Development cycle of AGV for the purpose of this research

1	Obtaining a suitable RC car to use as an AGV base	
2	Controlling the motors using microcontrollers	
3	Establish a communication method between PC and AGV	

4	Adding sensors to monitor speed, direction and distance travelled	
5	Find a suitable system to use for navigation	

6	Find a suitable way to avoid objects	
7	Combining all modules to construct an AGV	

The remote control car that was dismantled and used is a TAMIYA 4WD, shaft-driven with TT-01 chassis. The newly formed AGV base was equipped with sensors to be able to control and monitor the AGV. These sensors included the following:

- Hall effect sensors for determining the speed, distance and acceleration
- Potentiometer for direction sensing, and determining the angle of the wheels

- Ultrasound and IR sensors for beaconing
- IR sensors for object detection

1.7 Outline of the Dissertation

The structure of this dissertation is as follows:

- Chapter 2

  The available technologies within the AGV environment are discussed.

- Chapter 3

  The researcher will discuss the AGV base and the sensors placed on the base for controlling and monitoring the speed, direction and distance travelled. The communication method used between PC and AGV will also be discussed.

- Chapter 4

  A clear explanation of the beaconing system used for navigation will be given.

- Chapter 5

  The operation of the GP2D12 infra red sensors for object detection is explained.

- Chapter 6

  The conclusion of the dissertation, as well as possible future work that can come from this dissertation will be discussed.

1.8 References

1. TRANSBOTICS CORPORATION, Industry education, Learn about AGVs, www.transbotics.com/IndustryEducation/IndustryEducation.html, 2006.
2. AGV Products, Industries, www.agvp.com, 2005.
3. BOJE EP, VERMAAK HJ, Converting a Remote Control Car into an Automatic Guided Vehicle, Interim Central University of Technology, 2005, pp. 13-23.
4. AMERDEN Inc, Automatic Guided Vehicle Systems, www.amerden.com, 2005.
5. MROSZCZYK JW, Safety practices for Automated Guided Vehicles (AGVs), Danvers, www.asse.org/prac_spec_tech2.htm, 2002.
6. LUNDGREN M, Path Tracking and Obstacle Avoidance for a Miniature Robot, Master's Thesis, Umea University, 2003.
7. TOYOTA, Industrial Equipment, www.forklift.toyota.ca, 2006.
8. DEMATIC, Automatic Guided Vehicles, FAQs, www.dematic.us, 2006.

Chapter 2

Available Automatic Guided Vehicle Technologies

2.1 Introduction

Automatic guided vehicles (AGVs) are an exiting and developing technology when it comes to material-handling in industrial environments. The use of these vehicles will inevitably increase productivity and efficiency. A fleet of AGVs can be programmed to operate for twenty-four hours a day and these AGVs are remotely monitored by using sensors to obtain data pertaining to the physical state of the AGV as well as the ambient environment. This chapter will focus on the discussion of the AGV's basic characteristics.

In the following sections we will look at the available AGV technologies.

2.2 AGV Technologies

Automatic Guided Vehicles were first introduced in the 1950s. An AGV is an unmanned vehicle that is controlled by an onboard computer [1, p.6] and powered by batteries. The first AGV was a modified towing truck - it was used to pull a trailer and the navigation was done by following an overhead guide wire [2]. During the 1970's, a new AGV known as the unit load was developed, as can be seen in Figure 2.1 [2].

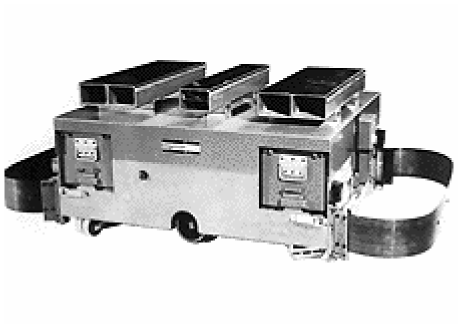


Figure 2.1: Unit Load AGV

AGVs were used as tuggers (trailer AGVs) as well as for carrying small loads. This has changed dramatically, as today, AGVs can be equipped with tools like robot arms and grippers to form forklifts such as the model in Figure 2.2 [3], unit load carriers, side loading and high-lifting AGVs, tuggers and transport carts, just to name a few [4][5].

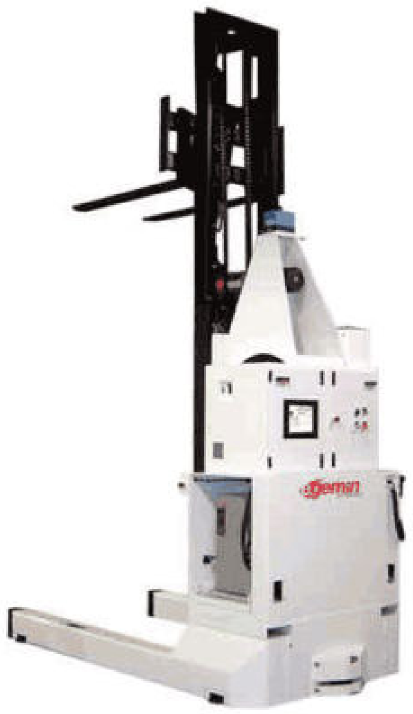


Figure 2.2: Example of a forklift AGV

AGVs can be different sizes and can carry different loads, from no load up to a few tons. The environment in which AGVs function can differ from air-conditioned offices with carpet floors to hospital hallways, factories with concrete floors and even outdoors, such as harbour docksides or even military environments.

The modern AGV is controlled by a microprocessor and some sensors. These sensors are responsible for being the “eyes” of the AGV. These sensors must help with the following tasks:

- Monitoring the physical condition of the AGV – speed, battery power, distance travelled, etc
- Object detection and avoidance
- Navigation
- And many more

AGVs are normally powered by batteries. To get optimum working time from the AGV, battery power must be used effectively.

The benefits of AGVs include the following [5][6, p.3]:

- Reduces labour costs
- Increases productivity and dependability
- Less damage to products due to handling
- Increased safety
- AGVs can be reprogrammed to follow different routes, thus this method of handling material is much more flexible than, for example, conveyor belts

The following recommendations for AGV safety should be followed [2]:

- The travel paths of the AGVs should be clearly marked so that people know which areas to avoid - this includes the turning areas of the AGVs.
- Workers must be trained to be on the lookout for AGVs and to keep clear of an AGV path if a vehicle is approaching.
- When working close to or on an AGV path, cones (obstacles) should be placed around the area concerned to protect the workers.

2.3 Motors and Speed control

One of the biggest challenges in building an AGV is controlling its motors. There are different types of motors available with different ratings. These include DC motors, Brushless DC motors, AC motors and Servomotors.

Preferably, microcontrollers must not be used to drive the DC motors directly. It is made even more complicated if the motor needs to be bidirectional. This section explains the basic principles of selecting and controlling the motors.

The choice of a motor for an AGV helps to define the capabilities of the rest of the system [7, p.3].

- The torque determines the maximum weight of the AGV, as well as the ability of the AGV to go uphill easily. Torque is the force applied to an object on an axle that causes the object to rotate around the axle [7, p.4].

- The motor’s maximum loaded RPM, along with the torque, size, weight and wheel size of the AGV. The wheel size determines the maximum speed that the AGV will be able to travel.
- The motor’s voltage determines the battery voltage required.
- The motor’s minimum, average and maximum currents will determine the size of the battery needed.

There are four basic ways to control the speed of a motor:

- By using mechanical gears to achieve the desired speed.
- Changing the motor’s voltage with a series resistor. By using this method, the torque is reduced and the motor might even stall due to the fact that the motor will draw more current. More current will result in an increase in voltage across the series resistor. This will result in less voltage across the motor. This method can thus be labelled as inefficient.
- Pulse width modulation. The amplitude stays constant and the width of the pulses is varied. PWM is a powerful technique for using microcontrollers to control analogue circuits. By controlling analogue circuits digitally, cost and power consumption can be reduced [8].
- Pulse frequency modulation (PFM). Amplitude is kept constant and the turn on time of the transistors are varied.

One of the biggest problems in delivering power to a load (in this case the motor) is the amount of power that is lost and the heat generated. AGVs are powered by batteries.

Therefore, to get optimum working time from the battery, the power must be used effectively. Using PWM to control the motors will use less power. When using PWM instead of constantly applying power, a pulse train with a fixed amplitude and frequency is applied. Only the width of the pulses is varied. This will cause the average output voltage to be the same as the input voltage, but the amount of power used is reduced. Figure 2.3 is an example of a signal with different duty cycles. The duty cycle is the ratio of the time-high relative to the period expressed as a percentage value.

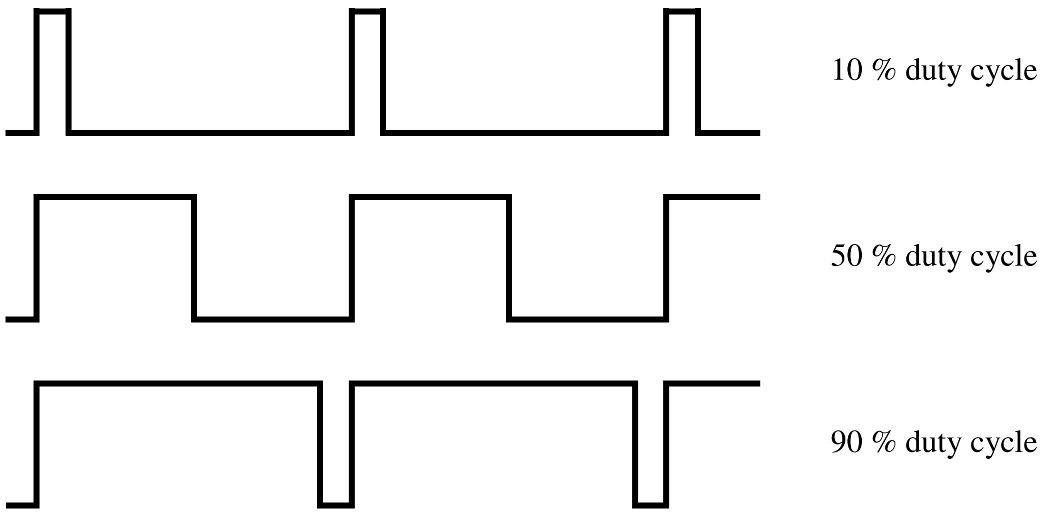


Figure 2.3: PWM signals of varying duty cycles

One of the advantages of PWM, as can be seen in Figure 2.3, is that the signal is digital - it is either on or off, high or low. By keeping the signal digital, noise effects are minimized [8].

An H-bridge circuit makes it possible to run the dc motor in both directions and control the servomotor. The best solution to control the dc motor is with PWM and an H-bridge circuit, like the one that can be seen in Figure 2.4 [9].

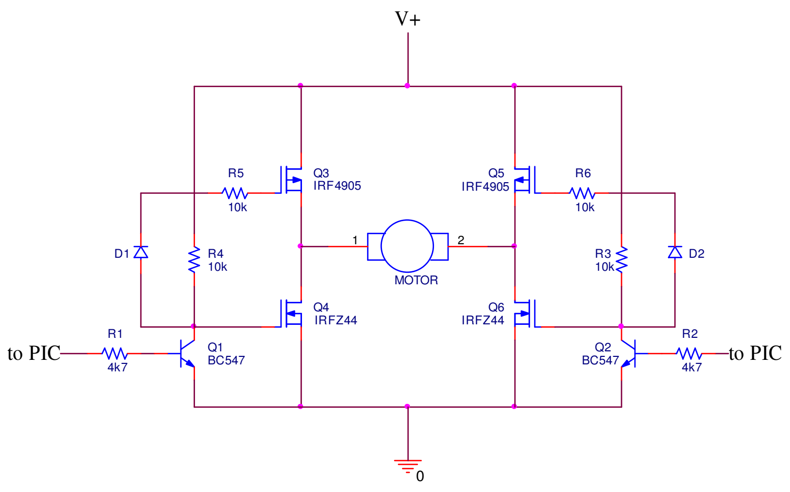


Figure 2.4: H-bridge circuit

Only two of the Metal Oxide Semiconductor Field Effect Transistors (MOSFETs) are on at a time. If the motor is to turn forward the one transistor is pulsed and the other transistor is kept at 5 V and vice versa. Short pulses on the base of the appropriate transistor means that the motor is running slowly and longer pulses will make the motor run faster. Pulse width modulation is used to control the amount of current flowing through the motor [8]. The P-channel MOSFETs on top can source power and the N-channel MOSFETs at the bottom can sink power [10].

An H-bridge circuit can be constructed with relays, transistors or field-effect transistors (FETs). When using transistors or FETs, the motor can be better controlled by using PWM. Higher currents can be reliably switched using FETs and transistors rather than relays. When using transistors, diodes should be connected across each transistor (collector-emitter) to protect the transistors from the back EMF that is generated by the motor's coil when the power is switched on and off. These diodes are called flyback diodes. MOSFETs are much more efficient than transistors, as they provide much more current without dissipating too much heat. MOSFETs usually have the flyback diodes built in [11].

By using capacitors, the electrical noise caused by motors, due to back EMF, can almost be eliminated. The back EMF of the motor is directly proportional to the angular velocity of the motor. The capacitors must be connected between the motor terminals and the case of the motor. The capacitors must be mounted directly onto the motor - if it is placed on the circuit board the leads from the motor to the capacitors become antennas that will enhance electromagnetic interference (EMI) [7, p.9]. The value of the capacitors must be in the nF range, up to 100 nF and breakdown voltages of 300 V. The capacitors will absorb the energy stored in the inductive windings of the motor.

A servomotor is used to direct the AGV. The basic construction and operation principles of a servomotor are basically the same as that of a conventional induction motor. The only difference is that servomotors have been redesigned to meet high precision, high speed, high frequency positioning and speed control of the mechanical parts [12]. There are basically three types of servomotors; AC and DC servomotors and stepper motors. The AC servomotors can be divided into two groups; synchronous (or DC Brushless) and induction-type servomotors [12].

Some important characteristics of servomotors are [12]:

- Motor speed is proportional to supply voltage
- Motor torque is proportional to supply current

- Motor output power is proportional to the speed multiplied by the torque and the input power is proportional to the supply voltage times the supply current
- Temperature has an influence on the operation of the motor

The shaft of the servomotor can be positioned to a specific angular position by sending a signal to the servomotor. The servomotor is smaller than the DC motor and it has built-in control circuitry.

2.4 Sensors

Sensors are needed to provide information about the AGV's surrounding environment and the state of the vehicle itself. In this study, sensors must be used for navigation and object avoidance as well as for monitoring the state of the AGV, more specifically the speed and the steering of the AGV. To determine the distance from obstacles, remote sensing is needed. Remote sensing means that observations or measurements of the obstacle can be made from a distance. Sensors can be grouped in two categories, i.e. active and passive sensors.

Active sensors determine the state of the AGV's environment by transmitting and receiving a signal. Some examples of active sensors are:

- Laser Range Finders
- Radar
- Ultrasonic sensors
- Infrared sensors

Passive sensors, on the other hand, do not emit signals but collect data through observation. Passive sensors can further be divided into two more groups, internal or external. Internal sensors are used to collect data concerning the vehicle's state, such as motor velocity or the steering angle. External sensors are used to determine the state of the vehicle's environment, such as range or proximity. Some examples of passive sensors are:

- Potentiometers
- Tachometers
- Encoders
- Tactile sensors
- Cameras

2.5 Navigation of AGVs

Several methods of guidance and navigation are implemented. The general problems concerning AGV navigation can be summarized as three questions [13, p.10]:

- Where am I?
- Where am I going?
- How will I get there?

Since the 1970s, until about ten years ago, most AGVs used wire guidance, which is electromagnetic wires buried in the floor, for navigation purposes. The first thing that comes to mind when thinking of navigation is the Global Positioning System (GPS). The AGV in this project will only be used indoors, thus the GPS is not an option. GPS systems do not function very accurately indoors. Through the years different methods have been developed and researched. The future might even be an indoor global positioning system (GPS) [5].

AGVs that are used in manufacturing plants mostly use tracks to keep them on course. These tracks can be a groove in the floor, a wire buried in the floor or even a strip of tape attached to the floor. It is difficult to implement the groove and wire method in an existing building, as revamping must be done and this may delay production, or the plant may have to be closed for a while. Using the strip method is much easier, but then the maintenance is much higher since the strip must be replaced after a certain period of time due to corrosion, tearing etc. Since it was not always possible to change the environment and install guide wires, other methods had to be utilized for guidance and navigation, and these includes [7, p.2]:

- Laser scanners
- Microwave transponders
- Inertia gyros
- Ultrasonic sensors
- Embedded magnets
- Camera vision

2.5.1 Wire Guidance

Wire guidance is one of the most popular navigation methods for industrial robots [14]. It is based on passing current through a conductor or wire, which then creates an electromagnetic field around the conductor. The closer the AGV gets to the conductor, the stronger the magnetic field. When an electromagnetic field is passed through a coil, it induces a voltage across the ends of the coil, which is proportional to the field strength. In this case, the AGV is equipped with guide and cross-antennas. The guide antenna must be centered over the guide wire. The cross antenna is used to pick up the intersections of the cross-wires. The guide-antenna has two coils, one coil on each side of the guide-wire. The difference in voltage between the two coils is amplified and this creates the steering signal. When the antenna is centered over the guide-wire, the voltages in the two coils will be equal thus, the AGV is on the right track and no steering adjustments will be made. If the AGV deviates to the left or right, the voltage in the one coil will increase and decrease in the other. This will result in a steering adjustment. To monitor the position of the AGV, cross-wires are installed in the floor. The cross antenna of the AGV picks up these intersections and, in this way, it can determine the position of the AGV [4].

2.5.2. Painted Line Guidance

This method is very similar to wire guidance. Lines are painted on the floor using visible or invisible fluorescent dye and sensors are used to detect these lines and follow them. The advantage of this method is that the paths can be fixed and are easy to alter. The disadvantages are [14]:

- Network of routes should be kept simple.
- The lines must be repainted from time to time due to general deterioration.
- Lines can be obscured by objects thus disabling the AGV guidance. This is why object avoidance is so crucial.

2.5.3. Laser Guidance

When using laser guidance the AGV is equipped with a laser scanner that gives out X and Y coordinates. The laser scanner must be mounted as high as possible on the AGV, so that it is highly visible to the targets. The targets are made of reflective tape. A minimum of three targets must be detected before the system can produce the coordinates. The targets can be up to 30 metres from the AGV. The laser scanner will measure the distance and the angle to each target, and its output will be an X and Y coordinates [2, p.2]. AGVs used in the Ohio State University Medical Center (OSUMC) are used as transporters. The AGVs are used to move materials around the hospital, including carrying the patients' meals, linen, supplies and waste between the patient wings and service floors, with interfaces to the kitchen, laundry, supply and trash areas. These AGVs make use of laser guidance for navigation around the hospital. A rotating infrared light on top of the transporters hits multiple reflectors attached to the walls. The transporters catches the reflections and uses the information to calculate its position. The transporters has a map stored in its memory to plot its exact position [15].

Laser guidance is currently one of the most popular methods used in the AGV industry.

2.5.4. Inertial Guidance

This method uses magnets/gyroscopes, and sometimes accelerometers, to measure the rate of rotation and acceleration. A magnet position sensor is mounted onto the AGV and used to detect small magnets installed in the floor. It also has gyroscope technology that helps the AGV go in the right direction. Usually there is a pair of magnets placed every five to ten metres. The gyro sensor gives the direction of the vehicle and provides an output voltage that is proportional to the rate of turn. The gyro sensor uses the Coriolis Effect to detect the angular rate [4]. A problem with this method is the high cost, as highly accurate gyros are very expensive [16, p.21]. Another type of gyroscope, known as the gyrocompass, is also available. It can align itself with the earth's rotational axis, but it also tends to be a large and costly instrument [17, p.2].

2.5.5. Path-Tracking

The Path-Tracking method was one of the possible solutions to the navigation problem due to the fact that it is said to be very accurate and, therefore, it is discussed in much more detail than the rest of the possibilities. Path-Tracking is another option for AGV navigation, and this process is concerned with determining the speed and steering settings at each instant of time. This is needed for the AGV to follow a certain path. A path consists of a set of points representing specific coordinates in a particular route. Path-Tracking consists of two parts. The first part is the Recoding Phase, where a human operator manually takes the AGV along a predefined path and the AGV records it. The second part is the Path-Tracking Mode - in this mode the AGV takes control and follows the recorded route as closely as possible. If the AGV comes across static obstacles it will take a detour around it and then return to the original route [1, pp.6, 13]. There are different types of Path-Tracking algorithms available. Three of these methods are:

- Follow-the-Carrot method

  This method is quite simple, all one has to do is obtain a goal point then aim the AGV towards that point. A line is drawn from the center of the AGV's coordinate system at right angles to the path. From this the orientation error can be determined, thus the AGV knows after a few calculations how many degrees it needs to turn in which direction to reach the goal point or the carrot point. An orientation error of 0° indicates that the AGV is pointing directly towards the goal point.

See Figure 2.5 for an illustration of the Follow-the-Carrot Path-Tracking method.

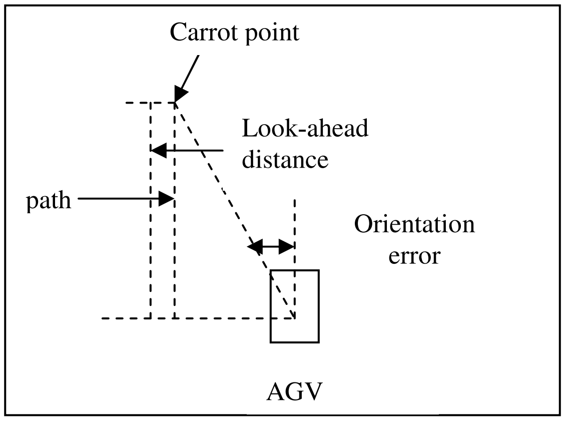


Figure 2.5: Follow-the-Carrot Path Tracking method

Although this method is easy to understand and simple to implement, it has a few major drawbacks [1, p.14]:

- The AGVs using this method tends to cut corners - this happens because the AGV will try to immediately turn towards each new carrot point.
- The vehicle could oscillate about the path, particularly if the look-ahead distance is small or if the vehicle is travelling at higher speeds.

- Pure Pursuit method

The basic concept of this method is to calculate the curving that will take the AGV from its current position to a goal point. The goal point is determined in the same manner as for the Follow-the-Carrot method. A circle is then defined so that it is possible to pass through both the AGV's current position and the goal point. Finally an algorithm must be developed to choose a steering angle for the AGV, as illustrated in Figure 2.6 [1, p.15].

In Figure 2.6:

- D is the distance between the current vehicle position and the goal point
- Δx is the x-offset of the goal point from the origin
- $1/\gamma_r$ is the radius of the circle that goes through both the center of the AGV and the goal point.

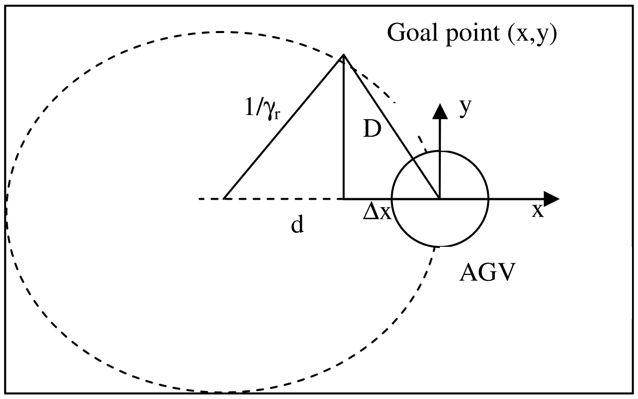


Figure 2.6: Pure Pursuit Path-Tracking method

According to tests done by Lundgren [1] the Pure Pursuit method shows better results than the Follow-the-Carrot method. There was less oscillation and the AGV had more accuracy at the curves [1, p.16].

- Vector Pursuit or Screw-Tracking method

Vector Pursuit is a more recent Path-Tracking method that uses the theory of screws. The Screw Theory can be used to represent the motion of any rigid body in relation to a given coordinate system. Any instantaneous motion can be described as a rotation about a line in space with a related pitch. This method was developed in order to get the AGV to not only arrive at the goal point, but to also have the right orientation and curvature. A screw consists of a centerline and a pitch. A line can be represented using only two points (r_1 and r_2), evident in Figure 2.7. The same line can also be defined as a unit vector (S) [1, p.17].

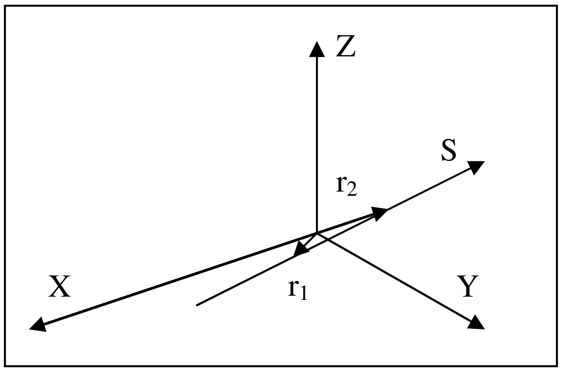


Figure 2.7: A line represented by two vectors.

An instantaneous motion of a rigid body can be illustrated as a rotation about a line in space with an associated pitch. Line S becomes the screw, with a certain pitch (h) and a certain velocity [1, p.17].

All three methods use a look-ahead point - this is a point on the path that is a certain distance, (L), away from the orthogonal projection of the AGV's current position. Changing the look-ahead point can have great effect on the performance of the system. If the distance increases the number of oscillations will reduce, thus ensuring the smooth tracking of the path. The downside to this is that the vehicle will now start to cut corners [1, p.19].

2.5.6. Dead Reckoning

Dead Reckoning is a method of estimating the position of the AGV based on speed, direction and the time that has elapsed since the last known position. The easiest way to implement Dead Reckoning is by using a method called Odometry. Odometry is one of the most widely used navigation methods for robot navigation. Odometry provides information about vehicle displacement based on the rotation of the AGV's wheels. This rotation can be measured in different ways, for instance wheel or shaft encoders, Hall Effect sensors etc [1, p.10]. The advantage of using this method is that it is very cheap compared to other methods such as GPS (Global Positioning Systems) or even ground beaconing systems, it is self-contained and it is always capable of providing the vehicle with an estimation of its position [16, p.20]. Odometry can be used with high sampling rates and it does not require complex mathematical calculations. The disadvantage of odometry, however, is that it tends to accumulate errors over time. A rule of thumb is that a small or medium sized AGV accumulates at least 10 cm of error for every 10 m of travel on a smooth surface [14]. There are mainly two types of errors [1, p.13][14]:

- Systematic errors
    - Unequal wheel diameters
    - Average of both wheel diameters differ from nominal diameter
    - Misalignment of wheels
    - Uncertainty about the effective wheelbase
    - Limited encoder resolution
    - Limited encoder sampling rate
- Non-systematic errors

- Travel over uneven floors
- Travel over unexpected objects on the floor
- Wheel-slippage due to
    - Slippery floors
    - Over acceleration
    - Fast turning (skidding)
    - External forces (interaction with external bodies)
    - Internal forces (e.g. castor wheels)
    - Non-point wheel contact on the floor

Systematic errors are worse than non-systematic errors due to the fact that they build up constantly.

The main problem with this method is wheel-slippage [18, p.2].

A magnetic compass can also be used to provide heading information. Compassing is one of the only methods that can provide absolute heading information without external references for calibration [17].

2.5.7. Beaconing

One system that navigates by means of beaconing is Cricket, which was developed and tested by Priyantha, Chakraborty, and Balakrihnan [19]. A beacon is a small device that is attached to some location within the space of operation. Each mobile and static node needs to get information from the beacons and, in order to do this, each of them needs a receiver or, as Priyantha *et al.* [19] calls it, a listener. A listener is a small device that listens to the messages from the beacons and uses this information to determine its current position [19]. In this particular study they first tried to use a purely RF-based system, but they did not get the wanted results. It was therefore decided to use a combination of RF and ultrasound to enable the listener to determine the distance to the beacons. The RF and ultrasonic signals are sent simultaneously from the beacon. The listener first receives the RF signal, it uses the first few bits as training information and then the listener turns on its ultrasound receiver. Only then does the listener start listening for the ultrasonic pulse, which arrives a short time later since the speed of sound in air is much slower than the speed of light in air. The receiver then uses the difference in time between the receipt of the RF signal and ultrasonic pulse to determine the distance to the beacon [19]. The use of the signal's Time-of-Flight is not a new concept for measuring distance – bats, for instance, also use the same concept to navigate. Another common example is to use the time elapsed between observing the lightning and hearing the thunder to estimate the distance to the lightning.

To avoid RF and ultrasound pulses from different beacons being correlated incorrectly, [18] it was decided not to use a fixed transmission schedule, but rather choose transmission times randomly with a uniform distribution within the interval. This means that each beacon's broadcasting is independent of the other beacons [19].

When the Beaconing System is used, the robot can compute its absolute position by measuring the distance and direction to three or more beacons [16, p.21].

2.5.8. Proximity Detection

Proximity sensors can be used when [18, p.2]:

- The object naturally transmits a signal.
- The object has its own transmitter.
- Reflection – a signal is transmitted to the object and the same signal is once again received by the transmitter.

For the first two methods, passive sensors are needed, and for the latter, active sensors are used.

Proximity detectors have been proven to not be able to provide a solution for AGV navigation on their own. They can, however, be used in conjunction with other methods to improve the AGVs navigation skills [18, p.2].

2.5.9. Landmark Navigation

In this navigation method, distinctive artificial landmarks are placed at known locations within the environment. The advantage of using artificial landmarks is that they can be designed for optimal detectability. As with beaconing, at least three landmarks should be visible to allow the AGV to determine its position [16, p.21].

2.5.10. Tactile Detection

This method involves some interaction between the AGV and the environment. This implies physical contact and requires [18, p.2]:

- Contact that does not harm the AGV or environment
- Identifying the contact
- Developing a strategy for scanning the environment

This method is very hazardous as it relies on physical contact [18, p.2].

2.6 Determining Distance and Speed of Other Objects

Due to the fact that the AGV must navigate through an unknown environment, the object detection must be done from a distance. The AGV must have enough time to stop, turn or accelerate to avoid an obstacle. It is better to totally avoid the collision than to only be able to detect it once it has happened.

Up to about ten years ago the obstacle detection systems of AGVs consisted mostly of mechanical bumpers, which were emergency stops that stopped the AGV when it came in contact with anything [1, p.1]. Since then, other methods have been explored to detect and avoid obstacles. Of these, the most common methods are:

- Stereovision
- Ultrasonic sensors
- Infrared sensors
- Laser sensors

Stereovision was not considered in this study due to the fact that it is considerably more expensive than the other methods. It is also a specialised field and could prove to be a project all on its own.

2.6.1 Ultrasonic Sensors

Ultrasonic sensors use the velocity of sound in air to measure distance from a beacon, or object, to the receiver. The velocity of sound depends on environmental factors such as the ambient temperature and humidity. Within a building these properties can exhibit both temporal and spatial variations. Temporal variations are variations that occur during different times of the day or seasons of the year. Spatial variations are variations of temperature and humidity due to effects such as direct sunlight falling in different sections of a room, the presence of heaters and air conditioners within a room, etc [16, p.5].

Ultrasonic sensors usually work on a frequency of about 40 kHz. Distance is determined by using the Time-of-Flight method. The distance l to a reflected object is calculated by

$$l = \frac{ct}{2} \qquad [2.1]$$

where c is the speed of sound and t is the round-trip time-of-flight, as shown in Figure 2.8 [20].

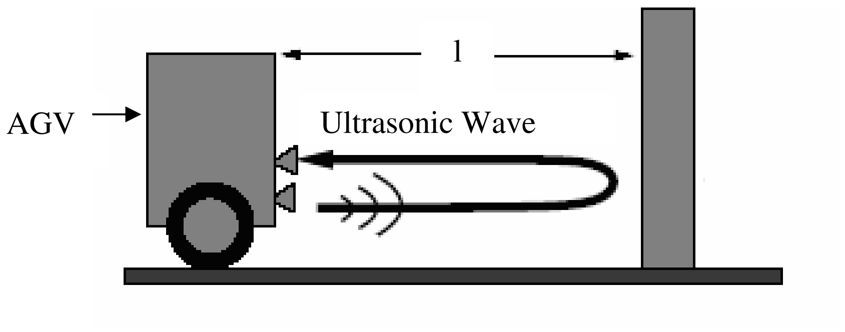


Figure 2.8: AGV using ultrasound for object detection

Figure 2.9 shows a model of the waves that would be reflected if there were two objects in front of the AGV.

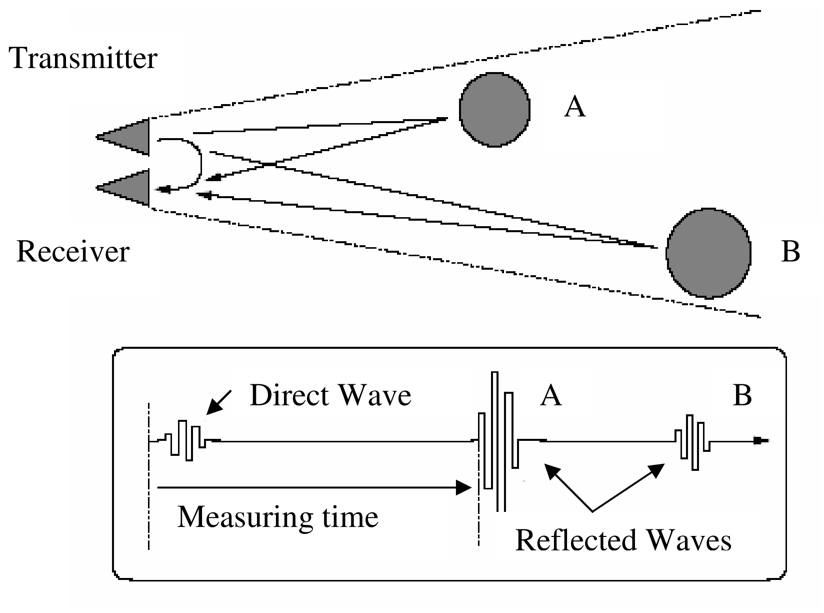


Figure 2.9: A model of reflected waves

Since one uses a receiver and a transmitter sensor next to each other, the receiver picks up a direct wave from the transmitter, as is clearly shown in Figure 2.9. This direct wave must be neglected in order for the AGV to obtain correct information [20].

The advantages of this system are that it is cost-effective and very simple to implement. However, there are some limitations as well. Common to all sonar-ranging systems is the problem of sonar reflection - when the object is at an angle, the range computed will be the closest point to the object. In the case of ramps or dips in the floor, the sensor will detect the ground as an obstacle.

2.6.2 Infrared Sensors

Infrared sensors can be used to measure:

- Ambient light

- Light reflected from obstacles

The values returned by these proximity measurements depends on factors such as the colour, shape and intensity of the obstacle detected, as well as the distance to the obstacle [1, p.8]. According to Lundgren [1], white objects are preferred - the darker the objects became the worse the level of accuracy.

The principle being explored in this study is based on Time-of-Flight methods, similar to ultrasound. An IR signal is transmitted and the receiver waits for a response. The time it took from when the signal was transmitted up until it was received is used to determine the distance from the object. Infrared sensors from the Sharp range, such as the GP2D12 or GP2D02, can be used for this application.

2.6.3 Laser Range-Finding

Several methods of laser range-finding have been developed, including:

- Time-of-flight measurement
- Phase shift measurement
- Triangulation
- Absolute interferometry

Laser range-finders provide fast, high-resolution readings over a long measurement range. Laser range-finders also have the ability to produce 2D and 3D contours of surrounding terrain. The biggest limitation experienced with regard to the laser range-finder is that it is one of the most expensive sensor systems available.

2.7 Communication

A communication method is needed between different AGVs, between the AGV and the user as well as between the AGV and destinations. Different possibilities are available:

- Infrared communication
- Laser communication
- Radio frequencies
- Bluetooth, etc.

In this project the chosen method for communication between the different entities is radio signals, or, more specifically Bluetooth. The reason for this will become evident in the next few pages.

The ideal radio signal will have high speeds, use as little energy as possible and travel far distances. This ideal wave will make the process of data transfer fast (millisecond range) at great distances and will use very little battery power [21]. In reality, this is impossible, due to the fact that the faster the data travels and the greater the distance, the more energy is needed. It is impossible to achieve all three:

- Fast data transmission
- Greater distances
- Very little energy

This is why different methods were developed to transfer data, each method with its own advantages. The characteristics of different Wireless Area Networks (WAN) can be seen in Table 2.1, and typical examples in Table 2.2 [22].

Table 2.1: Characteristics of different Wireless Area Networks

Wide Area Network (WAN)	Range	Power Drain	Transmit speed
Wireless Personal Area Network (WPAN)	10 m	Low	800 Kbps
Wireless Local Area Network (WLAN)	100 m	Medium	11 Mbps
Wireless Wide Area Network (WWAN)	2 – 3 km	High	14.4 – 56 Kbps
Wireless Metropolitan Area Network (WMAN)	30 km	Very High	1.5 Mbps
Wireless Global Area Network (WGAN)	500 – 1500 km	High	64 Kbps

Since AGVs are powered by batteries, one of the most important issues to look at was power consumption. In Table 2.1, it is clear that the WPANs use the least amount of power, thereby making Bluetooth the obvious choice.

Table 2.2: Examples of different Wireless Area Networks

WAN	Example
WPAN	Bluetooth
WLAN	Wi-Fi
WWAN	GSM, CDMA, GPRS, CDPD, TDMA
WMAN	Sprint fixed wireless
WGAN	Satellite

Bluetooth is a wireless technology intended for short range wireless radio links. The primary features of Bluetooth can be summarized as follow [9]:

- Voice and data capabilities

- Robustness
- Low complexity
- Low power consumption
- Low cost

Bluetooth operates in the range of 2.4 to 2.48 GHz. The advantage of this is that these frequencies will be the same worldwide and any Bluetooth device can connect to any other Bluetooth device within range. Each Bluetooth device can communicate with up to seven other devices. A Bluetooth network is called a piconet. One Bluetooth device can be part of many piconets [22]. The disadvantage of using Bluetooth is its limited range.

To establish a connection between two Bluetooth devices, the one device must request a connection and the other must accept. Security is very important. The AGV should receive its commands from one point or person only. If the security level was low, any Bluetooth device will be able to communicate with the AGV and this could be a safety risk. Bluetooth has the following methods for security.

- Authentication is used to verify the identity of the other device - it requires a passkey from the remote device.
- Authorisation is a Yes-or-No security – it is limited to "yes you may connect" or "no you may not connect"
- Every Bluetooth device has a unique identity or device address - it is assigned to the device during manufacturing and the Bluetooth Device Address (BDA) is usually displayed in hexadecimal format.

- Link key is a unique, internally generated, access code. Link Keys are generated automatically when devices connect.
- Connected devices share a unique Link Key, which they exchange each time they connect.

The fact that Bluetooth has low power consumption is the only factor that is limiting the range. The only way to have low power consumption is to transmit weak signals, typically in the range of 1 mW. Although this signal is very weak, it can still transmit through walls [9].

2.8 Batteries

Different batteries have different voltage and power ratings and may also not be able to deliver infinite currents. The power rating is usually indicated in Ampere-Hour, if the battery was rated 1 Ah it means the battery can supply one ampere for one hour. When more than one battery is connected in series it, will increase the voltage, and when it is connected in parallel, it will increase the current [23].

There are many different battery types, such as:

- Nicle cadmium (NiCd)
- Lead acid
- Lithium Ion
- Alkaline

Different types of batteries with the same voltage usually have different current capabilities. The most common battery type is Alkaline batteries and they also happen to be the cheapest.

The problem with alkaline batteries is that they have a low power capacity and they are not rechargeable. Lead acid batteries are rechargeable, inexpensive, work well with solar powered applications and are frequently used for large, low performance type robots. Lead acid batteries have the highest power-to-size ratio, but have the disadvantage of being big and heavy [2]. Nickel Cadmium (NiCad) batteries have the highest current output and are not too expensive, they can also be charged much faster than, for instance, Lead acid. A faster charging time means that the AGV will be working more. By increasing the charging current, the battery can be charged in as little as ten minutes [2]. Nickel Metal Hydride batteries have a good current output and have a high energy capacity. Lithium (Li-ion) batteries are the latest technology and have the same capabilities as NiMHs and NiCads, but the advantage is that it weighs up to 35% less. The disadvantage, however is that it is very expensive.

Since the remote control car that will be transformed into an AGV has a NiCad battery already, the obvious choice for a battery will be the NiCad. The battery of an AGV must be monitored continuously. Battery charging stations can be placed throughout the plant, at loading points, turning points, docking stations etc.

2.9 References

1. LUNDGREN M, Path Tracking and Obstacle Avoidance for a Miniature Robot, Master's Thesis, Umea University, www.cs.umu.se/education/examina/Rapporter/465.pdf, 2003.
2. MATERIAL HANDLING INDUSTRY, eLessons: Understanding AGVs, http://www.mhia.org/psc/PSC_Products_GuidedVehicle_elessons.cfm, 2006.

3. WIKIPEDIA, Automatic Guided Vehicle, http://en.wikipedia.org/wiki/Automated_Guided_Vehicle, 2006.
4. AMERDEN Inc., Automatic Guided Vehicle Systems, www.amerden.com, 2005.
5. MROSZCZYK JW, Safety practices for Automated Guided Vehicles (AGVs), Danvers, www.asse.org/prac_spec_tech2.htm, 2004.
6. COHN JH, AGVs: An unbeatable choice for warehouse operations, Distributor's Edge, October/November edition, Virchow, Crause & Company LLP, 2004, pp. 1-7.
7. SILICON VALLEY, HBRobotics Club Builders Book, HomeBrew Robotics Club, http://www.wildrice.com/HBRobotics/HBRCBuildersBook.html, 2004.
8. BARR M, Introduction to Pulse Width Modulation, Embedded Systems Programming, Volume 14, Number 10, http://www.embedded.com/2001/0109, 2001.
9. BOJE EP, VERMAAK HJ, Converting a Remote Control Car into an Automatic Guided Vehicle, Interim Central University of Technology, 2005, pp. 13-23.
10. MCMANIS C, H-Bridges: Theory and Practice, Robotics Notebook, http://www.mcmanis.com/chuck/robotics/tutorial/h-bridge/index.html, 2003.
11. JORDAN B, Brief H-Bridge Theory of Operation, http://www.dprg.org/tutorials/1998-04a/, 2002.
12. MIKRO KONTROL, Servo basics, http://www.mikrokontrol.co.yu/edukacija/literature/servo_basic.pdf, 1999.
13. BORENSTEIN J, EVERETT HR, FENG L, Where am I? Sensors and Methods for Mobile Robot Positioning, 1st edition, USA, University of Michigan, 1996.
14. BORENSTEIN J, The OmniMate: A Guidewire- and Beacon-free AGV for Highly Reconfigurable Applications, International Journal of Production Research, Vol 38, No 9, June 2000, pp. 1993-2010.

15. GEIER J, OSUMC's Wireless Robots, http://www.wireless-nets.com/papers/wireless_wireless_robots.htm, 2005.

16. ALHAJ ALI SM, Technologies for Navigation in Unstructured Outdoor Environments, Dissertation for PhD submitted to University of Cincinnati, http://www.eng.uc.edu/~elhall/papers.html, 2003.

17. MILLER J, Mini Rover 7 – Electronic Compassing for Mobile Robotics, Circuit Cellar Issue 165, April 2004, pp. 14-23.

18. AL-SUDANI M Dr, WANG WEI L, Uber-Navigation Robot (UR), Exploring Innovation in Education and Research (iCEER-2005), http://www.iaalab.ncku.edu.tw/iceer2005/Form/PaperFile/99-0010.pdf, Taiwan, 2005

19. PRIYANTHA NB, CHAKRABORTY A, BALAKRIHNAN H, The Cricket Location-Support System, Proceedings of 6th annual international conference MobiCom'00, ACM Press, Boston, 2000, pp 32-43

20. MARSHALL B, An Introduction to Robot Sonar, www.robotbuilder.co.uk/resources/articles/138.aspx, 2005.

21. MCCLAIN JT, WIMPEY BJ, BARNHARD DH, POTTER WD, Distributed Robotic Target Acquisition using Bluetooth Communication, Proceedings of the 42nd annual South-east regional conference, 2004, pp. 291-296.

22. LXE Inc, Bluetooth Basics, EMS Technologies Company, www.lxe.com, 2001.

23. BUCHMANN, Battery University, www.batteryuniversity.com, 2003.

Chapter 3

Automatic Guided Vehicle Principles

3.1 Remote Control Car

The car being used for this project is a TAMIYA 4WD shaft-driven TT-01 chassis. Figure 3.1 shows the remote control (RC) car before any changes were made and Figure 3.2 illustrates the AGV base [1].

Figure 3.1: Remote Control Car

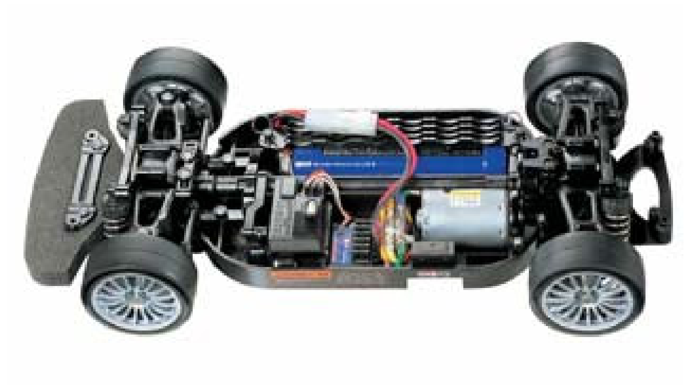


Figure 3.2: The AGV base before any changes were made

The overall length of the car is 441 mm and it is 185 mm wide. The car is powered by a 7.2 V 1500mAh NiCd battery. It has one DC motor for moving forward and backwards and a servomotor for steering. It has a remote control that works at 27 MHz. The receiver unit is the control unit of the car and it transfers the received commands to the motor and servo [1].

3.2 Hardware of AGV

The hardware of the AGV will consist of the following

- DC motor and H-bridge circuit for moving forward and backward
- Servo motor for steering
- Control unit with a microcontroller
- Bluetooth modules for communication
- Sensors for navigation and object avoidance
- Sensors for determining speed and direction
- Batteries to supply power

It is very important to have a base that is very light; a heavy base will consume a lot of power and cause a reduced battery life [2].

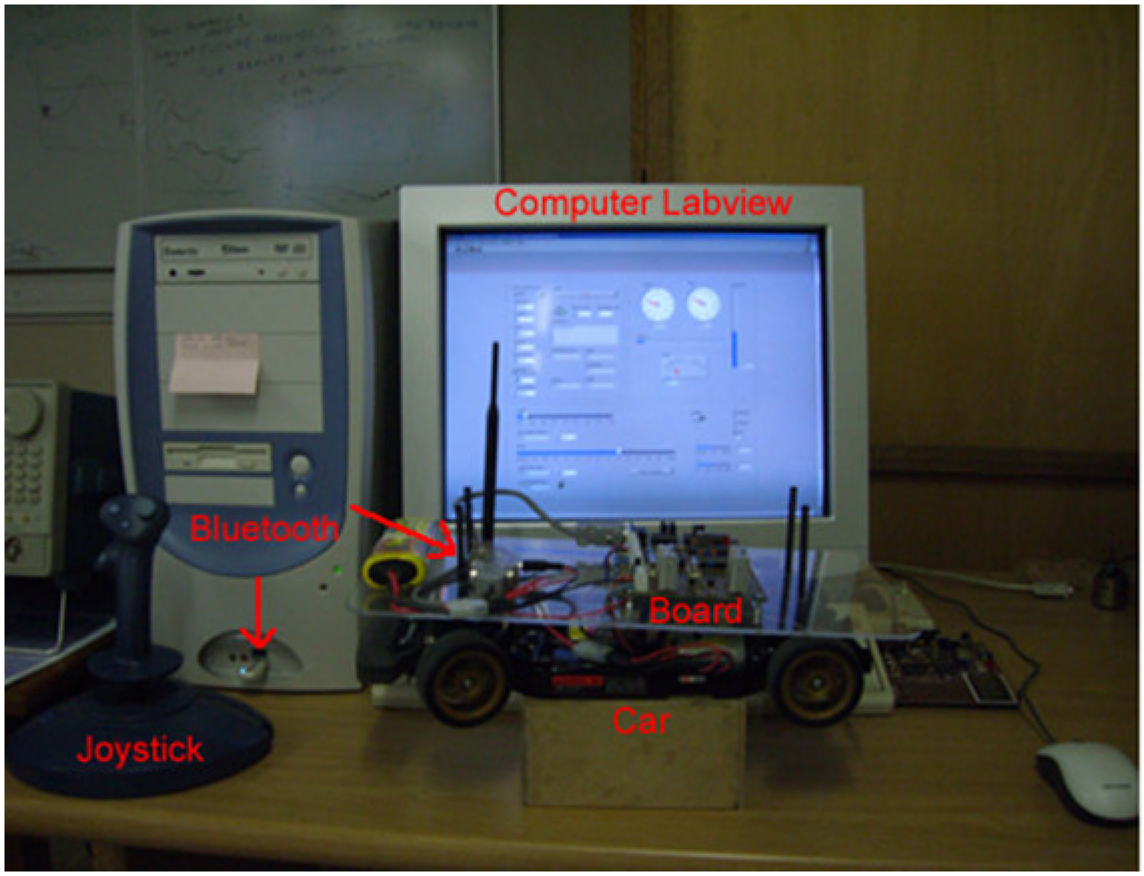


Figure 3.3: Complete system

The complete system can be seen in Figure 3.3. In front is the AGV base with the controller board, Bluetooth module and all the other parts, such as the batteries and motors. All sensors are placed directly onto the AGV. A PIC18F242 microcontroller is used to collect the data. The data is sent to a PC via Bluetooth and the data is used in LabView. Data is also sent from LabView to the AGV. The LabView interface displays the measured values and has controls to manually control the vehicle. On the left is a simple gaming joystick that can also be used to manually control the AGV. The PCB (Board) that was developed can be seen in Figure 3.4.

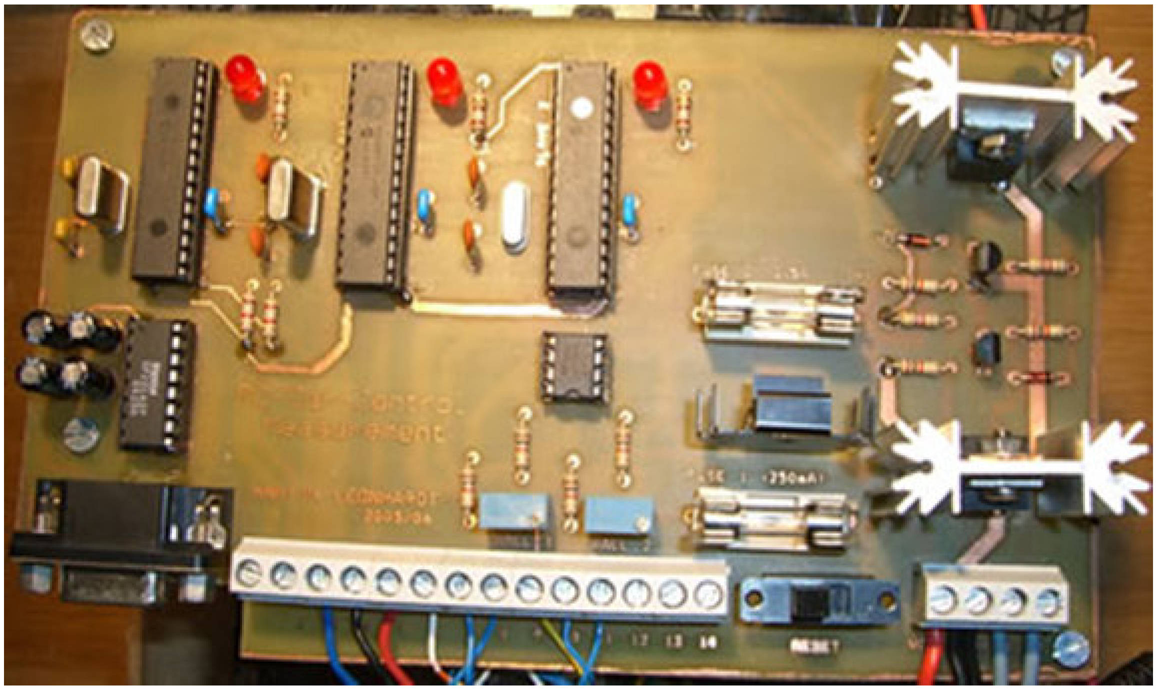


Figure 3.4: Complete PCB

3.3. Motors and Their Control

The first step in changing the RC car into an AGV is to investigate how the motors are going to be controlled. A DC motor is used for forward and backward movement and a Futaba Servo motor is used for steering. See Figure 3.5 for an example of a DC motor.

Figure 3.5: DC motor in RC car

The DC motor inside the RC car is kept as it is, but the speed control of the RC car is replaced by an H-bridge circuit, as shown in Figure 3.6 [2].

The H-bridge circuit makes it possible to run the DC motor in both directions with only one power supply. The output voltage is controlled by pulse width modulation (PWM) - this technique is widely used and very well known [3]. The longer the pulse, the faster the AGV's motor will be turning. The PWM signals are generated by the microcontroller.

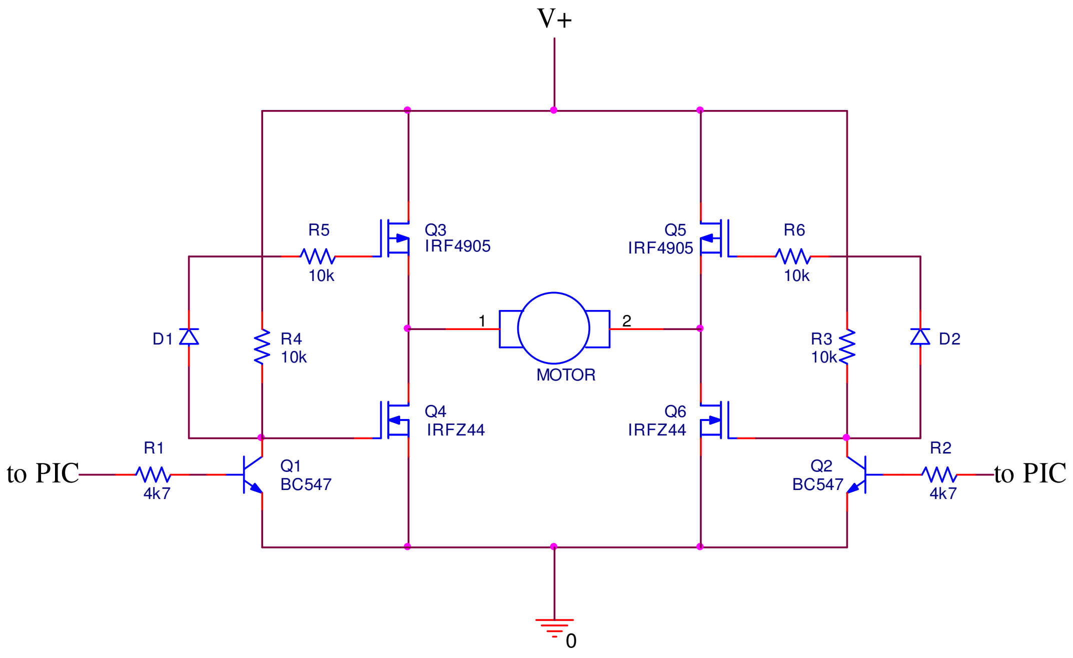


Figure 3.6: H-bridge circuit.

A microcontroller (PIC18F242) from Microchip will be used to control the motors and steering of the AGV. This microcontroller was chosen due to the fact that it has 4 Timers, USART, external interrupts, analogue inputs, PWM and it is rewritable. The external interrupts are used for the inputs coming from the Hall-Effect sensors. One of the analogue inputs is used for the input from the potentiometer at the steering. Timer0 is used to communicate with the PC at fixed intervals. Timer1 is used to set the cycle for the PWM. The PWM feature is used for controlling the servomotor (steering). The USART is needed for RS232 communication between the PIC and the Bluetooth module (HPS-120).

The initial high current drawn by the motor causes the PIC to reset [2]. The maximum current value measured by switching on the AGV, in other words the motor, was 4,2 A. To compensate for this amount of current, the tracks on the PCB was made much wider. Due to this high current and the problem of the PIC resetting, it was necessary to use two batteries. One battery is used to power the motor and H-bridge circuit and the other to power the control unit and the servomotor. A Futaba S-148 servomotor is used and is shown in Figure 3.7. [4].

Figure 3.7: Futaba S-148 Servomotor

A servomotor is used to direct the AGV. The shaft of the servo can be positioned to a specific angular position by sending a coded signal to the servomotor [5, p178]. The servomotor is smaller than the DC motor and it has built-in control circuitry, as well as a gear reduction system to produce a large amount of torque [5, p.179]. The servomotor draws power proportional to the mechanical load. The speed at which the servo is turning depends on how far it is from the desired position. The servo will spin as fast as possible until it gets close to the desired position, then it will slow down to avoid overshooting [6]. The inside of a servomotor is shown in Figure 3.8. [4].

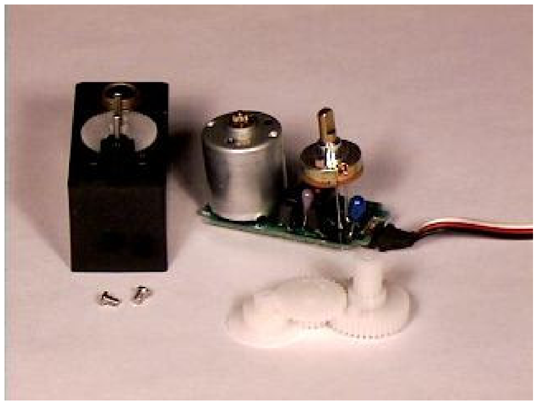


Figure 3.8: The inside of a servo motor

As seen in Figure 3.8., the inside of the motor consist of the control circuitry, the motor, a set of gears and the case. The potentiometer is used to sense the position of the shaft. If the position is not correct the internal control circuitry is used to turn the motor in the correct direction to obtain the desired angle [5, p.179]. There are three wires going to the servo, one is for power (+5 V), ground and a control wire. The amount of power applied to the motor by the control circuitry is proportional to the distance that it needs to travel. This means that if the shaft needs to turn a large distance, the motor will run at full speed and when it needs to turn only a little bit, the motor will run at a slower speed. This is called proportional control [4].

The control wire is used to give the commands to the servo. The angle is determined by the duration of the pulse applied to the control wire. The length of the pulse will determine how far the motor turns. A 1.6 ms pulse will make the motor turn to the 90° position. If the pulse is shorter than 1.6 ms the motor will turn the shaft closer to 0° and when the pulse is longer than 1.6 ms the shaft turns closer to 180°, as illustrated in Figure 3.9. In the RC car, the mechanical layout is designed so that when the shaft turns to the 0° position, the wheels will turn right, and when the shaft turns to the 180° position, the wheels will turn left.

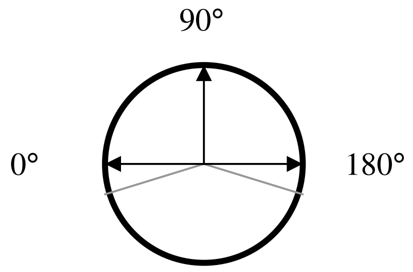


Figure 3.9: Shaft positions.

3.4 Speed and Steering

Hall-effect sensors are used to measure speed, acceleration and the distance travelled by the AGV. Hall-effect sensors work with magnetism. As a magnet comes closer to the sensor, the voltage of the sensor will increase with a few hundred millivolts. This sensor was chosen since it is immune to most environmental noises such as vibration, moisture, ambient lighting, dust, etc. [7]. Ten small magnets are mounted on the inside rim of the wheel to be able to determine the speed of the wheel, as shown in Figure 3.10. The amount of wheel rotations can also be determined as well as the distance travelled. One rotation of the wheel is 20 cm, therefore the hall-effect sensor will be able to give data for every 2 cm that is travelled.

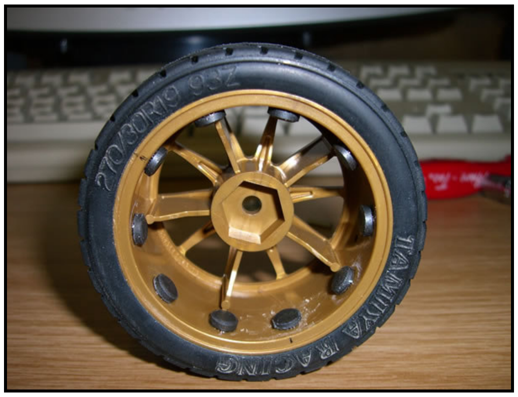


Figure 3.10: Wheel of AGV with magnets on

A bracket was made to mount the hall-effect sensor onto the side-shaft hub, as shown below in Figure 3.11. Another sensor was mounted on the gearbox of the AGV and four magnets on the gears next to the motor, but they were not used during this study (see Figure 3.12).

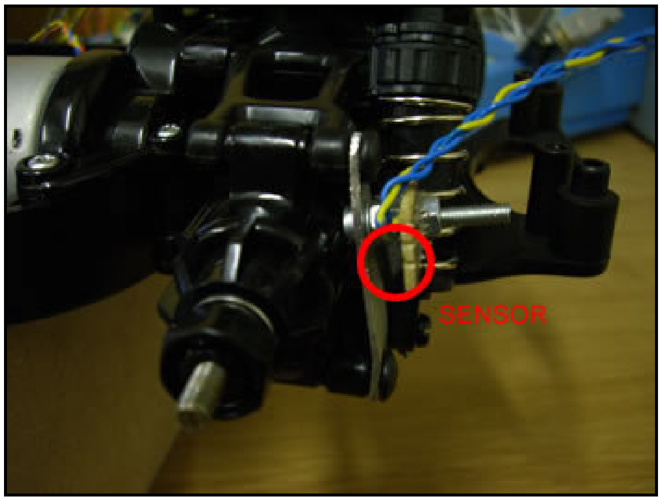


Figure 3.11: Hall-effect sensor mounted onto side-shaft hub

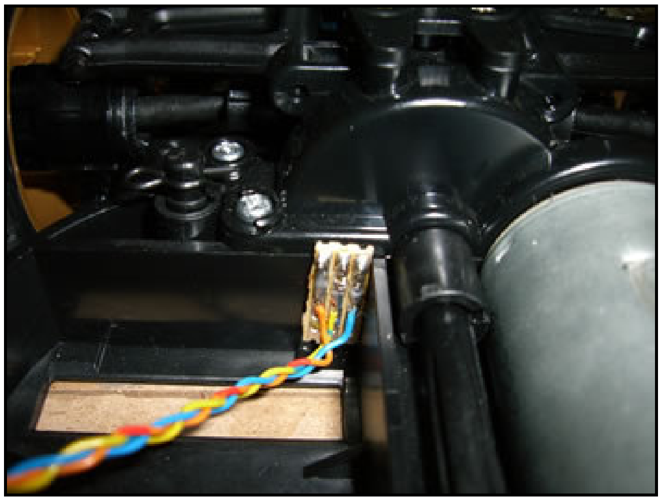


Figure 3.12: Hall-effect sensor mounted onto gearbox

If a magnet is directly in front of the sensor, the output voltage of the hall-effect sensor increases from 2.5 V to approximately 2.8 V. The sensitivity of the hall-effect sensor (UGN3503) used in this study is approximately 1.3 mV/G provided that the supply voltage is 5 V [7]. To be able to determine the speed of the AGV, the number of rotations of the wheel in one second should be calculated. Every ten 2.8 V readings equals one rotation. A comparator circuit, shown in Figure 3.12., was used to provide a digital input, between 3 V and 5 V, to port B of the microcontroller (PIC18F242).

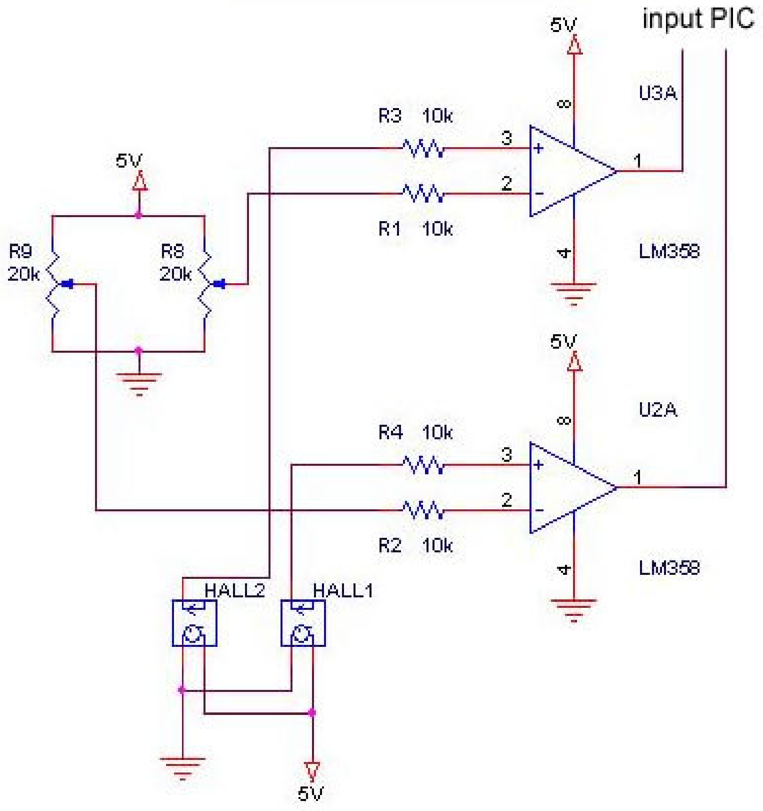


Figure 3.13: Comparator circuit

The preset resistors R_8 and $R_{9,}$ in Figure 3.13, are used to calibrate the comparator. For one rotation of the wheel, ten digital inputs must be counted for one rotation, as illustrated Figure 3.14. The measurements were done using LabView.

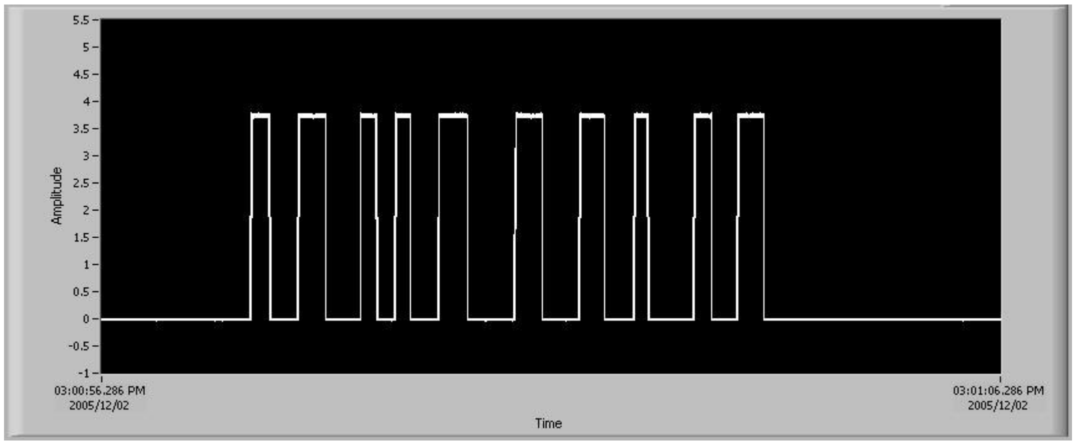


Figure 3.14: Input from hall-effect sensor to microcontroller

To measure the angle of the steering, a precision potentiometer is used, as shown in Figure 3.15. Due to the steering brackets of the AGV, only half a turn of the potentiometer could be used. The potentiometer has an analogue output voltage between 0.4 and 0.8 V. If the potentiometer is connected to a stable input voltage, the output voltage will vary at any radial movement. The output voltage values are then used to determine the angle of the wheels. The signal from the potentiometer is connected to one of the analogue inputs of the microcontroller (PIC18F242).

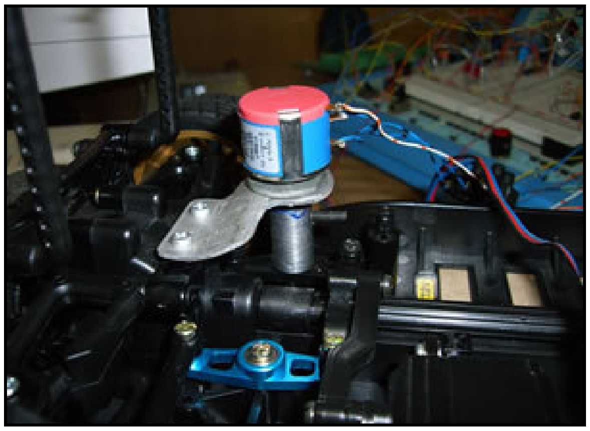


Figure 3.15: Precision potentiometer for steering control

3.5 Bluetooth Communication

In this project, the following Bluetooth modules were used, as shown in Figure 3.16:

- HPS-120 Handywave Version 2
- MSI 6970 Bluetooth dongle

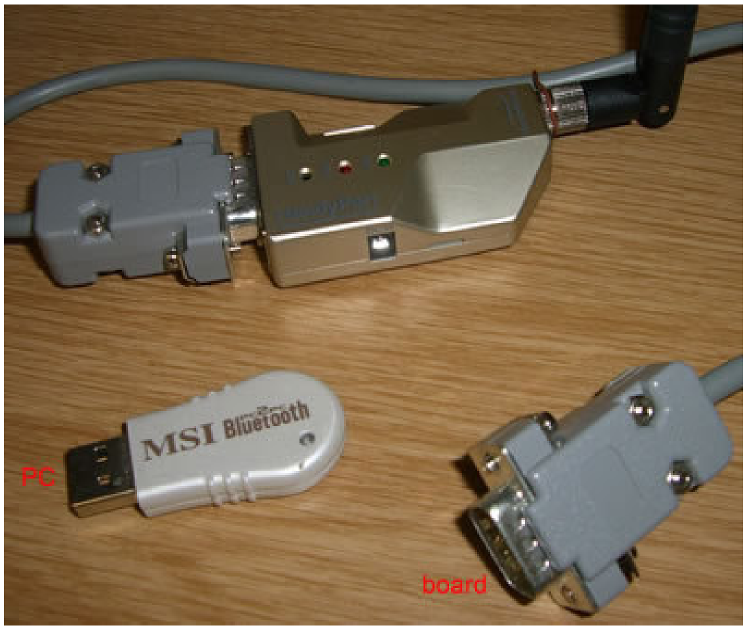


Figure 3.16: Bluetooth modules

Table 3.1 shows some specifications of the MSI 6970 Bluetooth dongle [8, pp. 2-3].

Table 3.1: MSI 6970 Bluetooth dongle specifications

Baud rate	Up to 723 Kbps
Coverage	Up to 30 m Line of sight
Connection	Point-to-point and point-to-multipoint
Hardware interface	USB
Standard	Bluetooth specification version 1.1
Frequency	2.4 to 2.48 GHz
Modulation	GFSK, BT=0.5
Tx power	6 dBm
Rx sensitivity	-82 dBm
Antenna interface	SMA female
Power supply	+5 V from USB interface
Operation temperature	0 to 75 °C

Table 3.2 shows the specifications of the HPS-120 [9, p. 4].

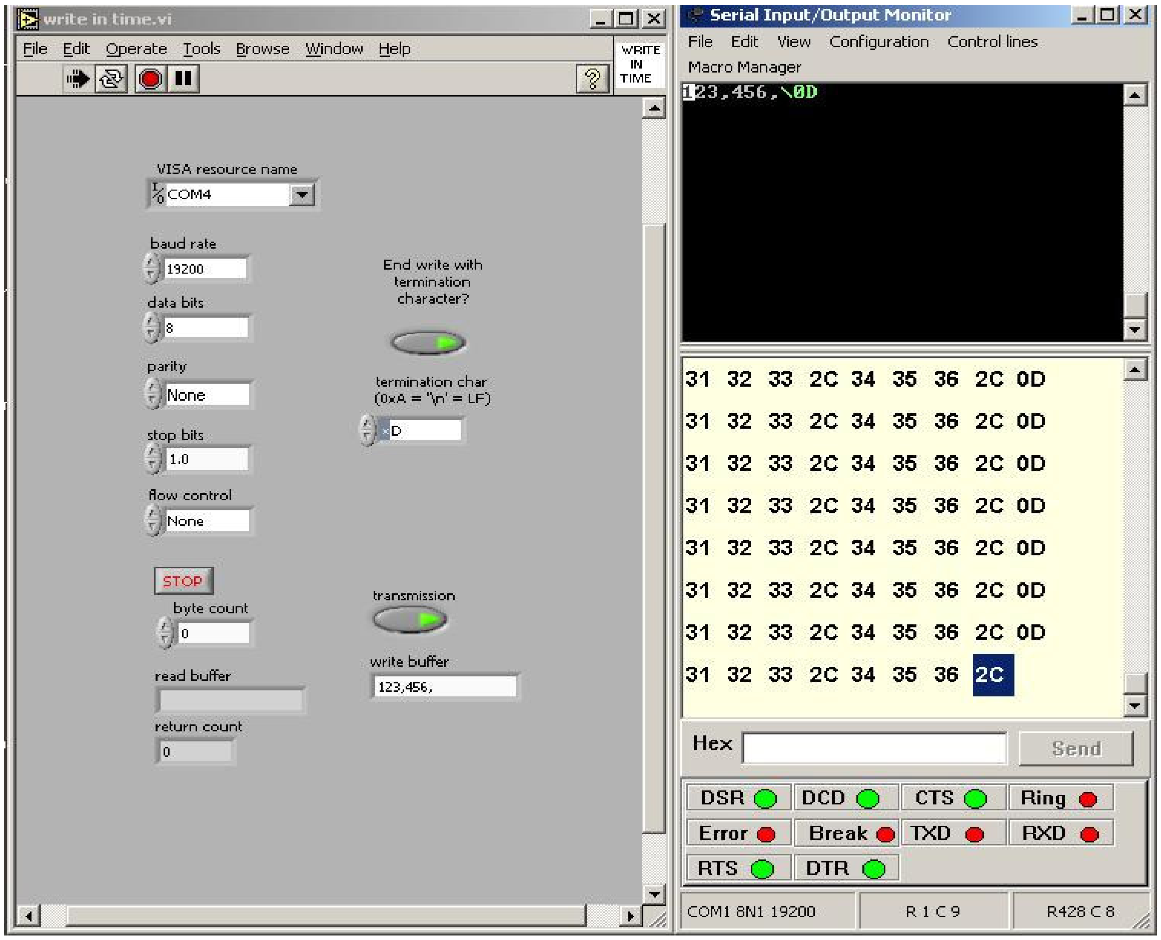
Table 3.2: Specification of HPS-120

Baud rate	Up to 115 Kbps
Coverage	Up to 100 m Line of sight
Connection	Point-to-point
Signal	DCD, TxD, RxD, GND, CTS/DSR, DTR, RTS
RS232 interface	D-Sub 9 pin female
Standard	Bluetooth specification version 1.1
Frequency	2.4 to 2.48 GHz
Hopping	1.6/sec 1 MHz channel space
Modulation	GFSK, 1 Mbps, 0.5BT Gaussian
Tx power	Max 20dBm / typical 16 dBm (class 1)
Rx sensitivity	-84 dBm
Antenna interface	SMA female
Antenna gain	Max 2 dBi
Power supply	+5 V ~ 12 Vdc
Current consumption	Max 110 mA
Operation temperature	-20 to 75 °C
Size	35 mm (W) x 65 mm (D) x 16 mm (H)

To establish a connection between two Bluetooth devices, the one device must request a connection and the other must accept. Security is very important and the AGV should receive its commands from one point or person only.

After the system was setup and tested, it reached a distance of 30 m through walls. The reason for the increased distance would be the fact that the HPS-120 is equipped with an antenna. The HPS-120 was fixed to the AGV base and the dongle was placed in the USB port of the PC.

To establish communication between the Bluetooth device on the AGV and the controller board of the AGV, RS-232 was used. The microcontroller has an USART interface that can be used via a MAX232 IC to obtain RS-232 communication between the microcontroller and the Bluetooth module (HPS-120) or directly to the PC's serial port. This RS-232 communication takes place every 149.5 ms. ASCII codes are sent between the microcontroller and LabView, as shown in Figure 3.17. After every string, a stop bit, in this case "0D", must be sent.


write in time.vi
File Edit Operate Tools Browse Window Help
WRITE IN TIME
Serial Input/Output Monitor
File Edit View Configuration Control lines
Macro Manager
Hex
Send

VISA resource name: COM4
baud rate: 19200
data bits: 8
parity: None
stop bits: 1.0
flow control: None
STOP
byte count: 0
read buffer:
return count: 0
End write with termination character?
termination char (0xA = '\n' = LF): D
transmission
write buffer: 123,456,

```
123,456,\0D
```

```
31  32  33  2C  34  35  36  2C  0D
31  32  33  2C  34  35  36  2C  0D
31  32  33  2C  34  35  36  2C  0D
31  32  33  2C  34  35  36  2C  0D
31  32  33  2C  34  35  36  2C  0D
31  32  33  2C  34  35  36  2C  0D
31  32  33  2C  34  35  36  2C  0D
31  32  33  2C  34  35  36  2C
```

DSR ● DCD ● CTS ● Ring ●
Error ● Break ● TXD ● RXD ●
RTS ● DTR ●

COM1 8N1 19200 | R 1 C 9 | R428 C 8

Figure 3.17: ASCII codes sent between PC and microcontroller

When sending data to the PC, the string is 16 bytes long and contains the information on the speed, acceleration, travelled distance and the angle of the AGV's front wheels. Every value is separated by a comma.

[000,000,0000,00/0d] = [speed, acceleration, distance, angle / stop bit]

The data is written from the PIC by using "printf" and the data is received in LabView by using the VISA commands.

When data is sent to the microcontroller from the PC, the string is 9 bytes long and contains the information for controlling the motors.

[000,0000/0d] = [PWM value steering, PWM value motor / stop bit]

3.6 Microcontrollers

Initially, a PIC16F876 was used for the project because it was known to the researcher and had all the necessary features. In the beginning stages of the project only the control of the motor was done and the Bluetooth communication was used which means that any PIC with USART would be able to work. The PWM for the control of the motors was done by using the timers. The commands to the AGV were specific keys on the keyboard of a computer. In the end the 18F242 suited the needs of the project best, as it has 4 Timers, USART, external interrupts, analogue inputs, 2 PWM onboard and it is rewritable. Since the PIC18F242 only has 2 PWM onboard, it was decided to use more than one microcontroller for controlling the motors. The first PIC was called the master and the other two PICs were the slaves. The different PICs must be able to communicate with each other, therefore I^2C communication was used.

The master PIC is used to do all the communicating. It is used to handle all communication between the PC and the AGV via Bluetooth. It also communicates with the two slave PICs to gather the data from the sensors and to give the commands as they come from the PC. All calculations are done on the master PIC. The reason for this is to be able to send the finished data package to the computer.

The slave 1 microcontroller controls the servomotor of the AGV. Indirectly that means that it is used to control the steering of the AGV. The input from the potentiometer is connected to this PIC. The slave 2 microcontroller controls the motor of the AGV, namely the speed and distance travelled as well as direction in which it has travelled. The hall-effect sensors are connected to this PIC.

To get the I^2C bus working, two pull-up resistors must be used, as shown in Figure 3.18. One master can accommodate up to 256 slaves.

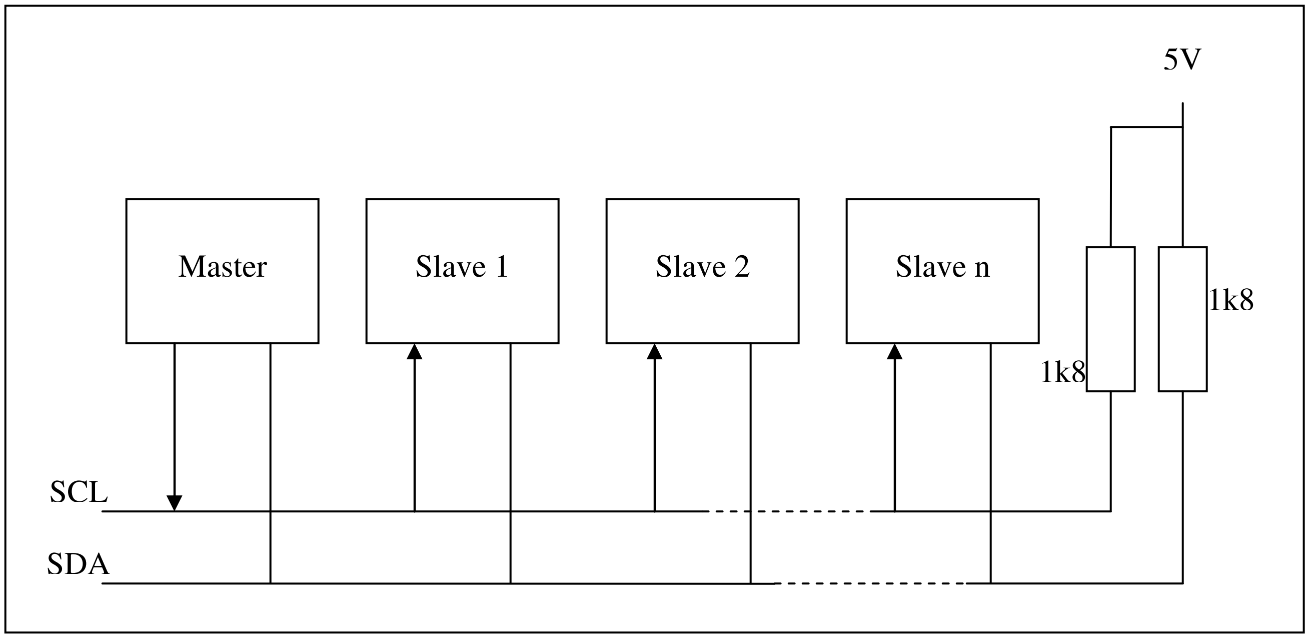


Figure 3.18: I^2C network

When using I^2C communication, two lines on the PIC microcontroller are used, SCL (Synchronous serial clock input) and SDA (I^2C data input/output). I^2C will be discussed in more detail later in this chapter.

All the PIC microcontrollers use a crystal oscillator, the master PIC is using a 20 MHz oscillator and the other two are using 3.278 MHz crystals. Lower crystals must be used in order for the PWM to operate as required, and since the master PIC has to do a lot of calculations and accept as well as transmit all data, a faster crystal was used.

3.7 Software

The programming of the PIC was done by using a CCS C Compiler. The user interface for monitoring and manual control was done in National Instrument's LabView 7.1.

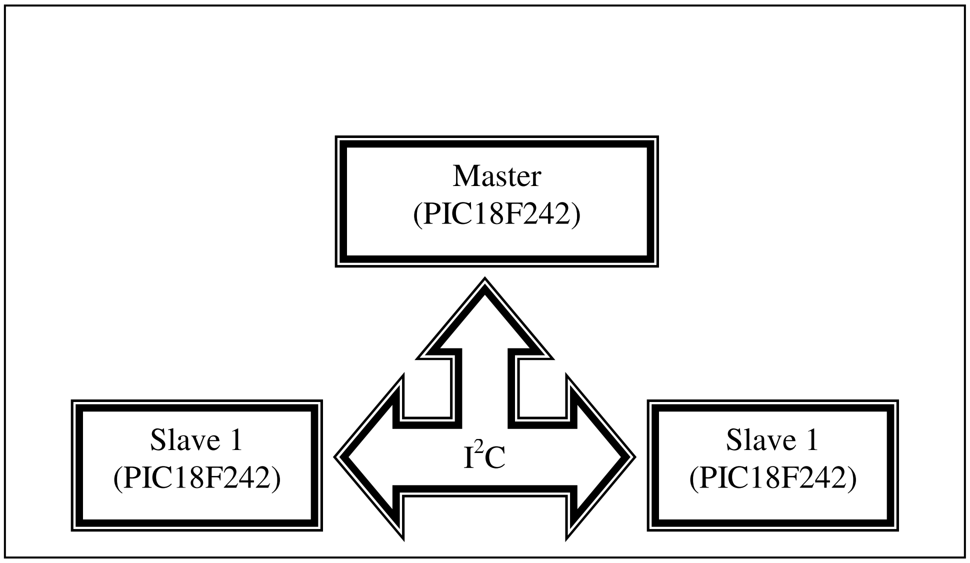


Figure 3.19: Master and slave setup

The I^2C-bus (Inter Integrated Circuit - bus) is a synchronous, serial, two wire connection between one master and several slaves. A total of 127 I^2C devices can be connected to one master. The one wire (SCL) transmits the clock pulse and the other (SDA) transmits the data. The PIC makes provision for two speed modes, standard mode (100 kHz) and fast-mode (400 kHz) [10]. In this project, fast-mode was used. All devices are connected in parallel to the bus and, therefore, each device needs an address. The PIC18F242 has a 7 bit address system that was used for this application. The master PIC initialises all actions. It does not matter if it is receiving data from slaves or if it is sending information to slaves. To start the initialisation the master makes the SDA line low and all the devices connected to the master will see this as the start bit. Now the master sends the 7 bit address to clearly indicate which device should “listen”. After the address is sent, the master will send another bit to indicate whether it wants to write to (0) or read from (1) the slave. The slave must answer the master by sending an acknowledge signal (make the SDA low). After the acknowledge signal, the master knows that it can start its operation. After every 8 bits the slave will send an acknowledge signal to indicate that the master can read the next 8 bits. After the required operation is completed, the master will switch the SDA line from low to high. After a few problems with the C Compiler the I^2C was sorted out by programming the subroutines manually (not using the routines provided by the C Compiler). The master obtains data from the slaves every 149.5 ms.

All the software discussed in this chapter can be seen in Appendix D.

3.8 Interface with Joystick

The joystick is connected to the game port of the sound card in the PC. The joystick has two output signals, one for the x-axis and one for the y-axis. Theses values range between 0 and 65000. The graphical user interface was done using LabView 7.1. In LabView, a Sub-Virtual Instrument (Sub-Vi) was used to assign different values to different commands [11].

The output signal of the x-axis has a very wide range and is very accurate. To keep the servomotor in the middle while not using the joystick, or get it turning while moving the stick to the right or left, a specific range needs to be specified as follows.

0-position	=>	$25000 < x < 30000$
Left turn	=>	$x < 25000$
Right turn	=>	$x > 30000$

The output signal of the y-axis also has a very wide range and it, too; is very accurate. To keep the main motor holding still while not moving the joystick, or get it turning while moving the stick to the front or back, a specific range needs to be specified as follows.

0-position	=>	$21000 < y < 25000$
Move forward	=>	$y < 21000$
Move backward	=>	$y > 25000$

3.9 Calculations

All calculations are done on the master PIC. The formula for calculating speed is the distance travelled divided by the time it took to travel the distance.

$$\text{speed} = \frac{\text{distance}}{\text{time}} \qquad [3.1]$$

Since the AGV moves approximately 20 cm on one rotation of the wheel it is possible to determine the distance by counting the pulses from the hall-effect sensor. Ten pulses are equal to a single rotation. The time was set to be 149.5 ms. This is a predetermined value based solely on the fact that a large amount of data needs to be transmitted and if the time set is lengthened somewhat, it makes the margin for error less. It was also successfully tested with the time set to 104 ms.

After every second speed reading, the acceleration can be calculated. This is done by getting the difference between these two speed readings and then dividing it by the time.

$$\text{acceleration} = \frac{\text{speed2 - speed1}}{\text{time interval}} \qquad [3.2]$$

3.10 How Did the Project Develop?

The initial control of the AGV is shown in Figure 3.20. The remote control part of the car was replaced by Bluetooth. Commands could be given from the keyboard of the PC to the car, such as direction (left, right, forward, backward) and speed (by using arrows on keyboard). The system was then further developed to work from the keypad on the keyboard.

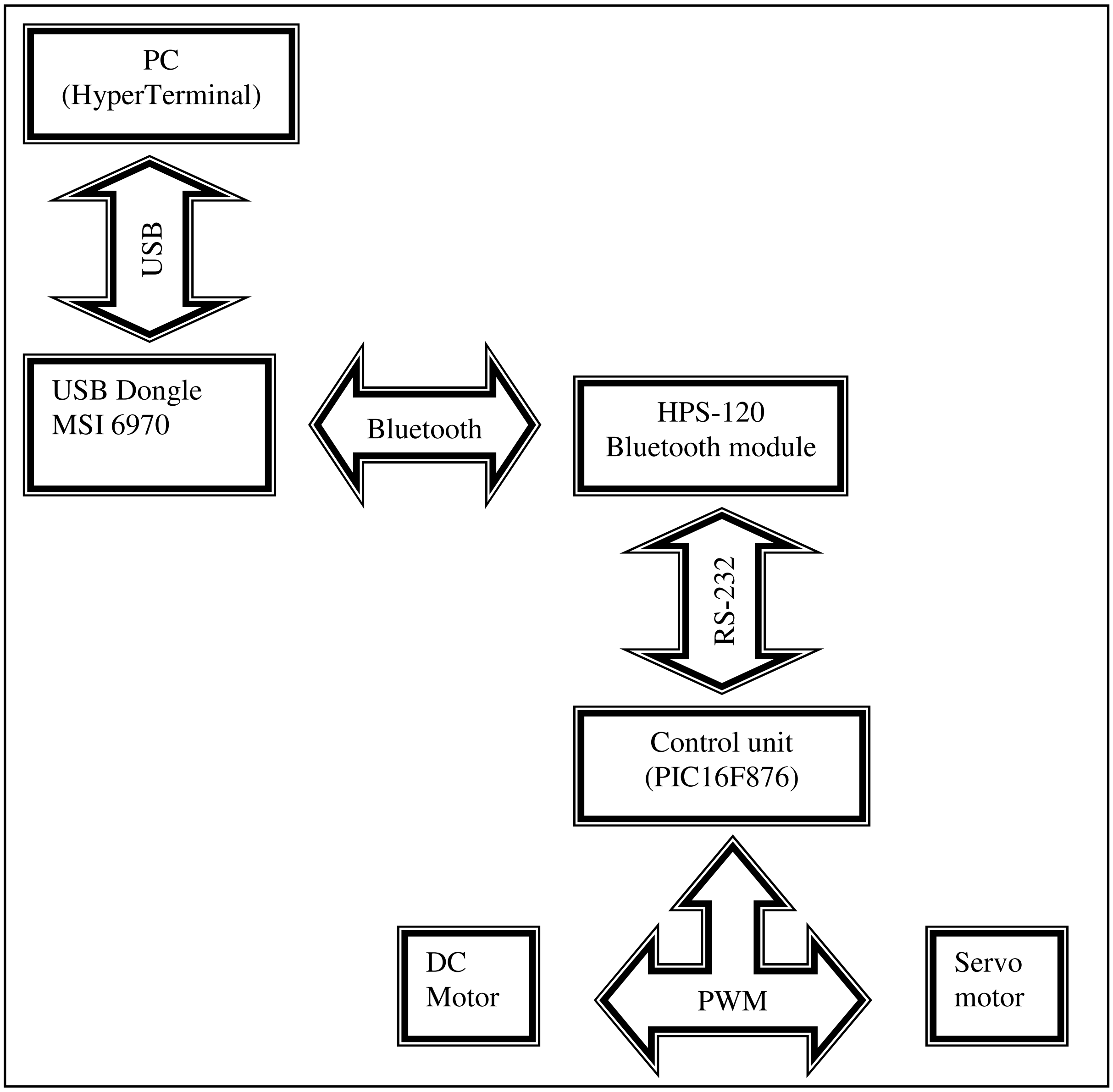


Figure 3.20: Block diagram of initial operation

The PWM was done by using the timers of the PIC. With this system, the AGV could be controlled manually with the keyboard or it could be programmed to follow a predetermined route.

Since a better user interface was necessary, it was decided to start using LabView and the block diagram was changed, as shown in Figure 3.21.

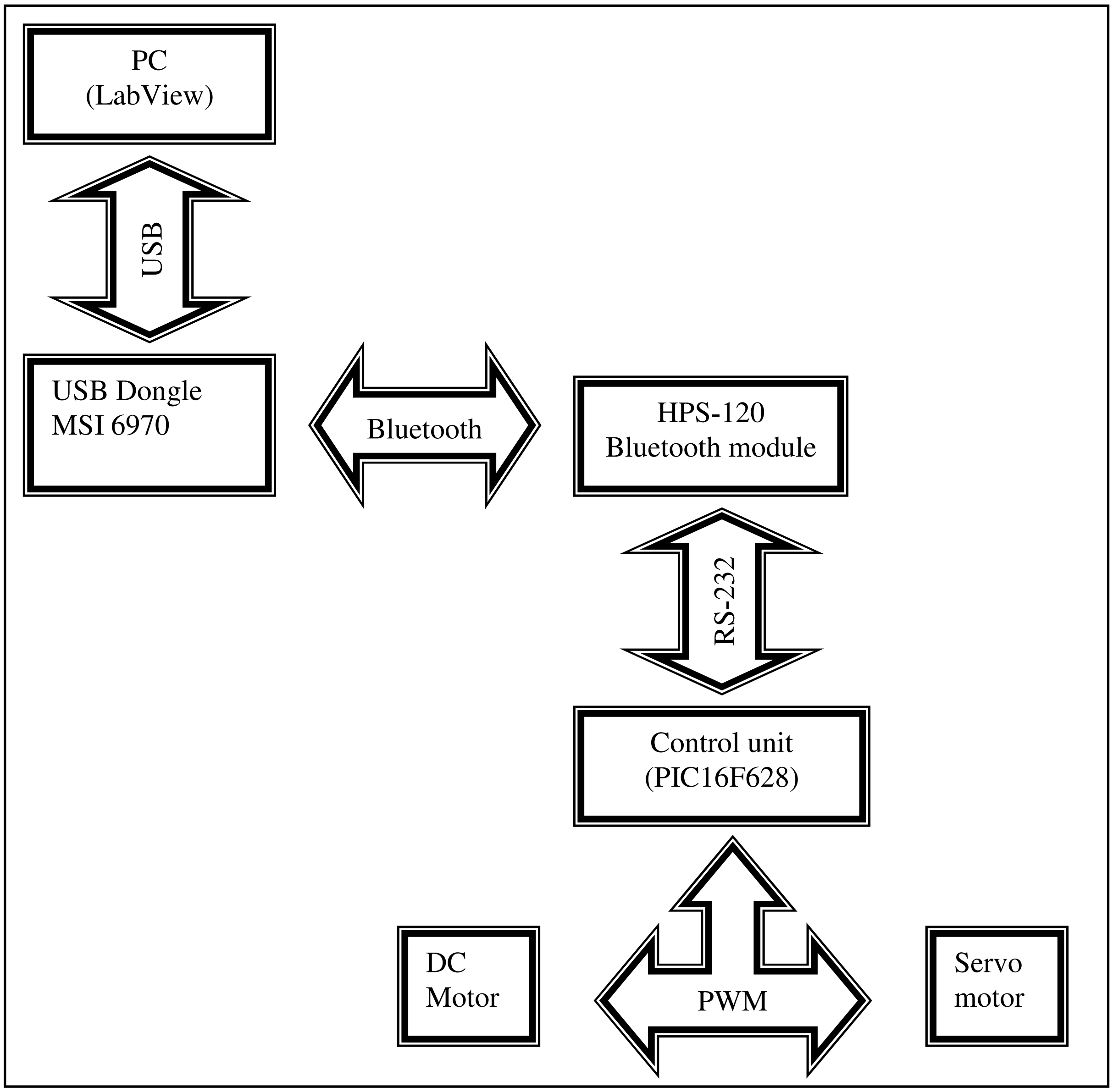


Figure 3.21: New improved circuit

To make the programming simpler, the microcontroller was changed to PIC18F242 due to the fact that it has onboard PWMs. Since there are two motors to control (DC motor and servomotor) and there are a lot of calculations to be done, the system was expanded to three microcontrollers. The problem, however, was to get the different microcontrollers to interact with each other as well as with LabView via Bluetooth. The first solution was RS-232, but since the Bluetooth module was already using that interface another solution had to be found. In the end, I^2C was used to fulfil the communication needs between the different microcontrollers. The final system can be seen in Figure 3.22.

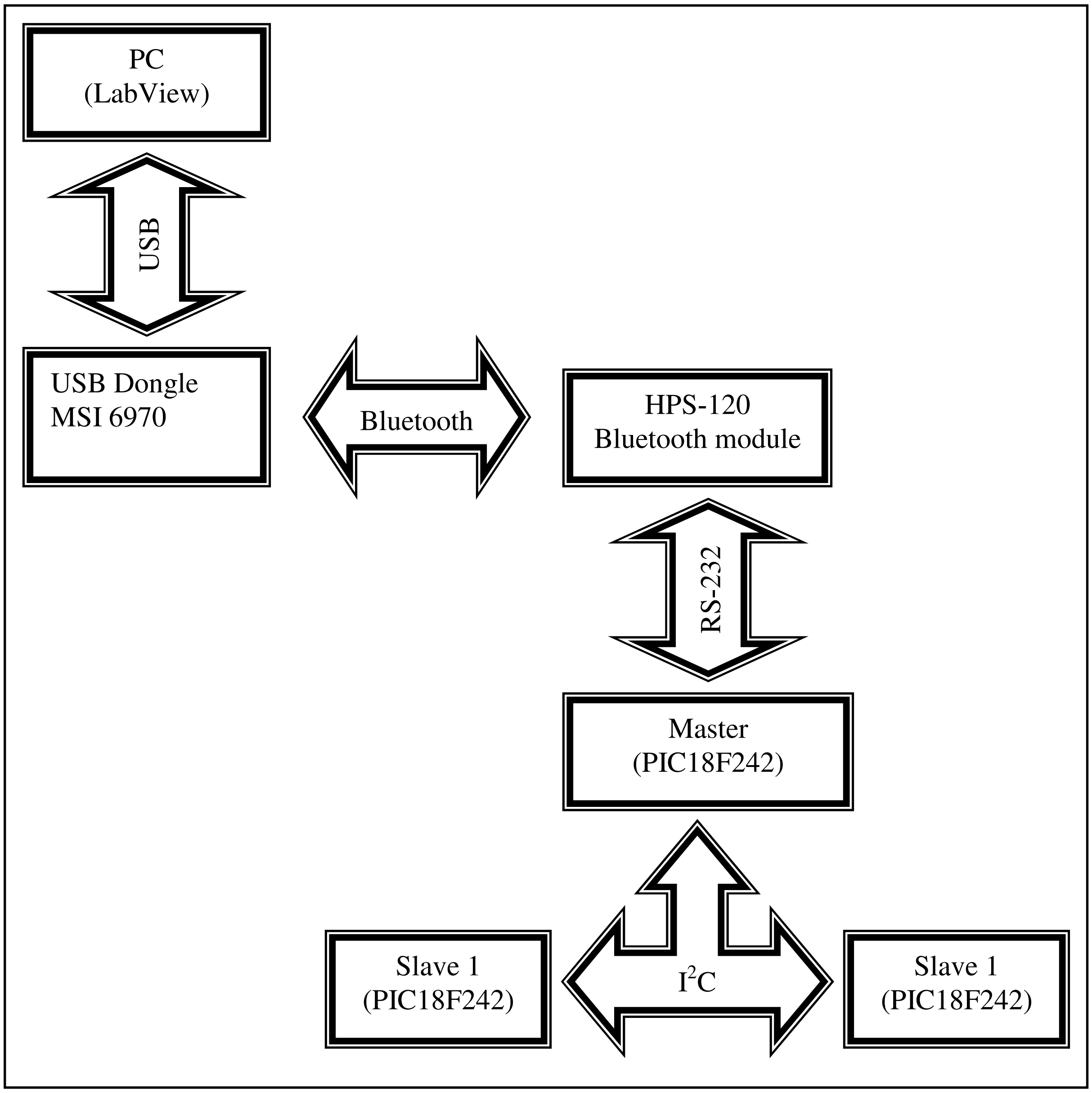


Figure 3.22: Final working system

The schematic diagrams for this part of the project can be found in Appendix A and the connection layout of this particular part of the project can be found in Appendix B. A step by step guide for getting the AGV to operate can be found in Appendix C.

3.11 References

1. TAMIYA, 1/10 scale TT-01 chassis, www.tamiya.com, 2003
2. BOJE E, VERMAAK HJ, Converting a Remote Control Car into an Automatic Guided Vehicle, Central University of Technology, Free State INTERIM, Year 4, Number 2, 2005, pp. 13-23.
3. KAMALASADAN S, HANDE A, A PID Controller for Real-Time DC Motor Speed Control using the C505C Microcontroller, 17th International Conference on Computer Applications in Industry and Engineering (CAINE), Orlando, FL, November 2004, pp. 34-39.
4. Seattle Robotics Society, What is a servo?, www.seattlerobotics.org/guide/servos.html, 2004.
5. HAMBLEN JO, FURMAN MD, Rapid Prototyping of Digital Systems, 2nd edition, Kluwer Academic Publishers, 2001.
6. BLANCH J, TOSUNOGLU S, Servo and Sensor Control on Small Mobile Platforms, ASME Southeastern Region XI Technical Journal, Volume 2, Number 1, April 2003; also presented at ASME South-eastern Region XI Technical Conference, Miami, Florida, April 2003.
7. GILBERT J, DEWEY R, Linear Hall-Effect sensors, Application Note 27702A, http://www.allegromicro.com, 1996.
8. Micro-star International, BToes Bluetooth USB Dongle, 2003.
9. Handywave Co, HPS-120 User Manual, Republic of Korea, 2001.
10. Axel Wolf, I2C (Inter-Integrated Circuit) Bus Technical Overview and Frequently Asked Questions (FAQ), www.esacademy.com/faq/i2c/, 2000.

Chapter 4

Automatic Guided Vehicle Navigation

4.1 Introduction

Since the ideal solution for navigation, Global Positioning System (GPS), does not work in indoor environments, the next best solution had to be used. The available technologies that were considered have been explained in detail in Chapter 2. Some of these are:

- Wire guidance
- Painted line guidance
- Laser guidance
- Inertial guidance
- Path tracking
- Dead reckoning
- Beaconing
- Landmark detection
- Proximity detection
- Tactile detection

Laser guidance is the most popular way of navigation, but it is an expensive solution. Therefore, it was decided to use beaconing for the navigation of the AGV. A beacon is a small device placed in the room where the AGV is suppose to navigate through. Usually it is placed on the ceilings or on the walls of the room so that it is not in the way [1]. When determining position, there are three major techniques that can be used [2]:

- Triangulation

  Triangulation can be used in two ways:

  a. Lateration, using the distance measurements to determine the position as shown in Figure 4.1.[2]

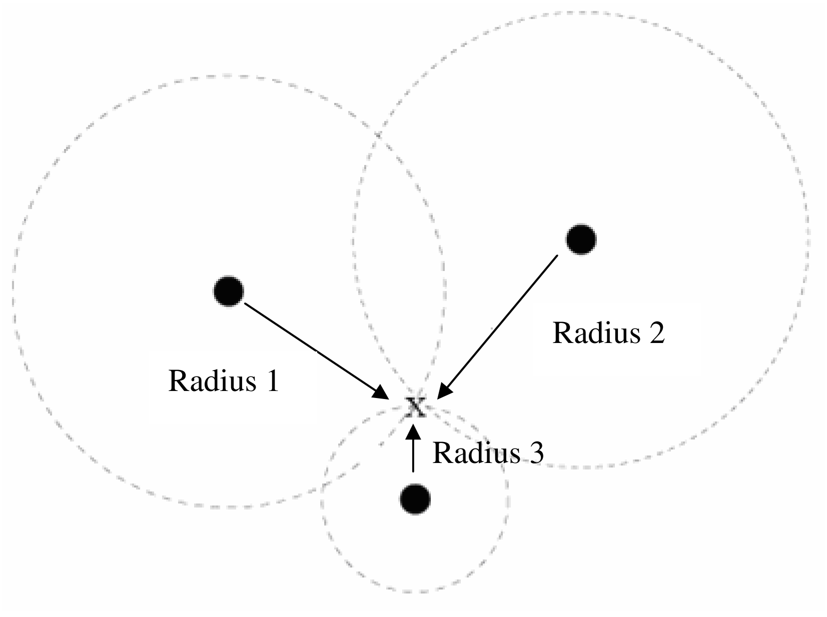


Figure 4.1: Determining position using lateration

  b. Angulation, using the angle measurements to determine the position as shown in Figure 4.2.[2].

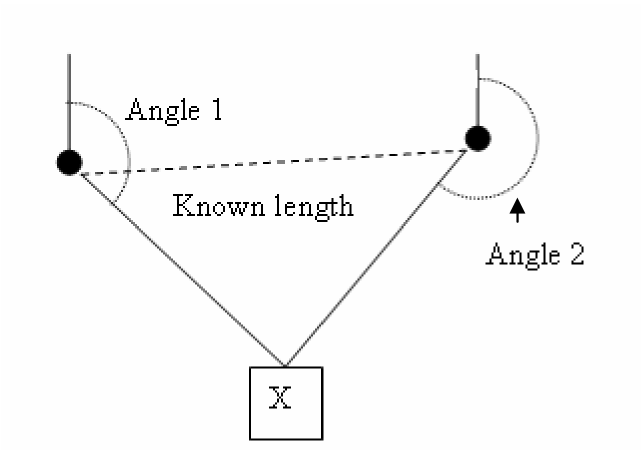


Figure 4.2: Determining position using angulation

- Proximity

  When using proximity to determine the position the AGV must measure the distance to a known location in the room and, according to that, it should be able to determine its position [2].

- Scene analysis

  In this method, features or specific characteristics in an area are used to determine position relative to the features. A camera is needed to use this method [2].

For this application, the navigation system must be able to work indoors, in rooms with a height of not more than three metres. The lab that was used for testing has an area of 60 m^2. The receiver was mounted on to the AGV and the transmitters to the ceiling. Therefore, the receiver would receive signals from the different transmitters and use these signals to determine the position of the AGV.

4.2 The Sensors

The transmitters were fixed at specific positions to the ceiling of the room in question. Each transmitter covered a circular area on the floor. The receiver had to be able to detect three transmitters in order for it to display accurate coordinates. The position of the AGV is where the three circles intersect, as illustrated in Figure 4.3 [3]. Ultrasonic and infrared sensors were used for the communication between the transmitters and receiver. Choosing the correct sensors is an essential part of the study. The reason for using the ultrasonic sensors was based on the work done by Priyantha *et. al.*[1] and the researchers in reference 3 [3]. The reason for using IR instead of RF was based on the work done in reference 3 by Jeunes [3] and Hallaway *et. al.* [4]. Initially, some problems were experienced with the range of transmission of the two types of sensors, since the infrared transmitter were very directional. This meant that at certain positions, the receiver picked up the ultrasound signal but not the infrared. The problem was solved by changing the IR transmitter and receiver to SFH485P and TSOP4838.

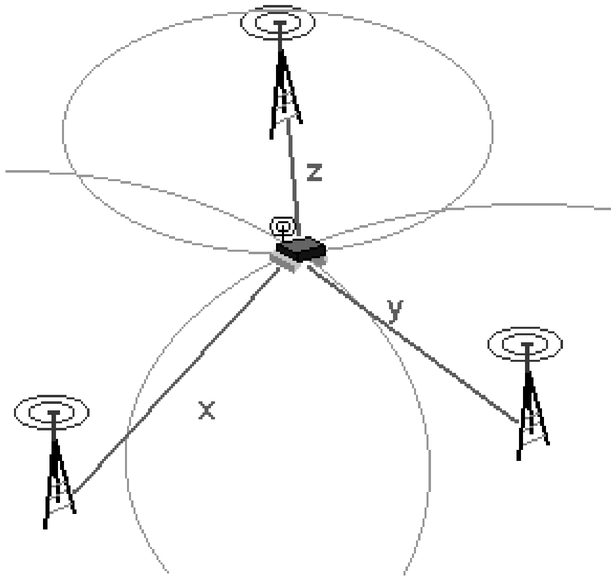


Figure 4.3: The position of the AGV is where the circles intersect[3]

The two signals are transmitted at the same time and, since the speed of light is faster than the speed of sound, the difference between the time that the infrared (red signal) is received and the time that the ultrasound (blue signal) is received is measured, as shown in Figure 4.4. This time is used to calculate the AGV's distance from the beacon [3]. The *Cricket location* system [1] uses the same principle, but instead of IR it uses RF signals.

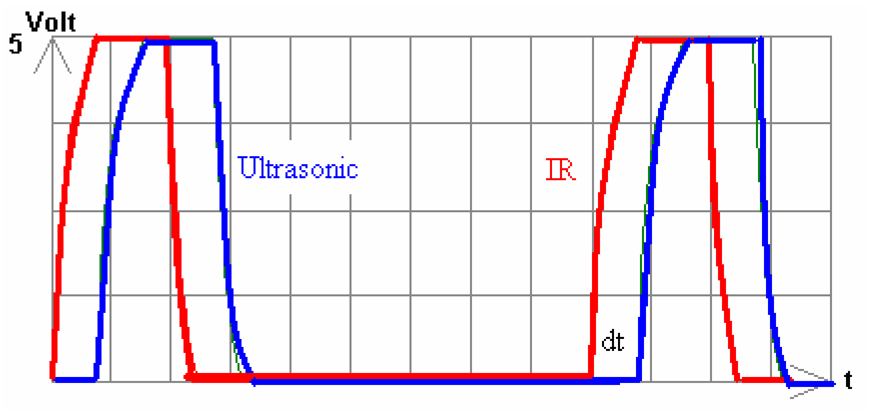


Figure 4.4: The difference between the received signals

Environmental effects do have an influence on the operation of the system, especially when it comes to the ultrasound. If there is any interference on the receiving side of the system, it will lead to incorrect results. These effects can change the velocity of sound as well as the absorption of sound in air. These effects will now be discussed in more detail.

The speed of sound is dependent on the type of medium in which it travels as well as ambient temperature, as shown in table 4.1 and 4.2. The speed of sound can be calculated with the following formula [4]:

$$v = 331 \text{ m/s} + (0.6 \text{ m/s/C})*T \qquad [4.1]$$

The humidity found in air has an insignificant effect on the speed of sound. Air pressure, however, does not have any significant effect on the speed of sound, since the air pressure and the air density are proportional to each other at the same temperature [5].

The approximate speed of sound in different materials and the influence of temperature on the speed of sound can be seen in Table 4.1 and Table 4.2.

Table 4.1: Approximate speed of sound in different materials [6].

Medium	**Speed of sound (m/s)**
Air, dry (20°C)	343
Water	1500
Concrete	3100
Steel	5800
Lead	2160
Glass	5500

Table 4.2: Influence of different temperatures on the speed of sound [6]

Temperature of air in °C	**Speed of sound in m/s**
-5	325.5
0	331.5
5	334.5
10	337.5
15	340.5
20	343.4
25	346.3
30	349.2

4.3 Transmitters

The starting point of this part of the project was a circuit that was found in work done by Jeunes [3], as shown in Figure 4.5. The 555 timer circuit produces a triangular wave at a frequency of 40 kHz. One of the drawbacks of this particular circuit is that it cannot transmit a 40 kHz signal continuously, the frequency decreases to about 35 kHz due to the heat generated within the circuit. The other problem with this circuit is that the receiver cannot distinguish between the different transmitters.

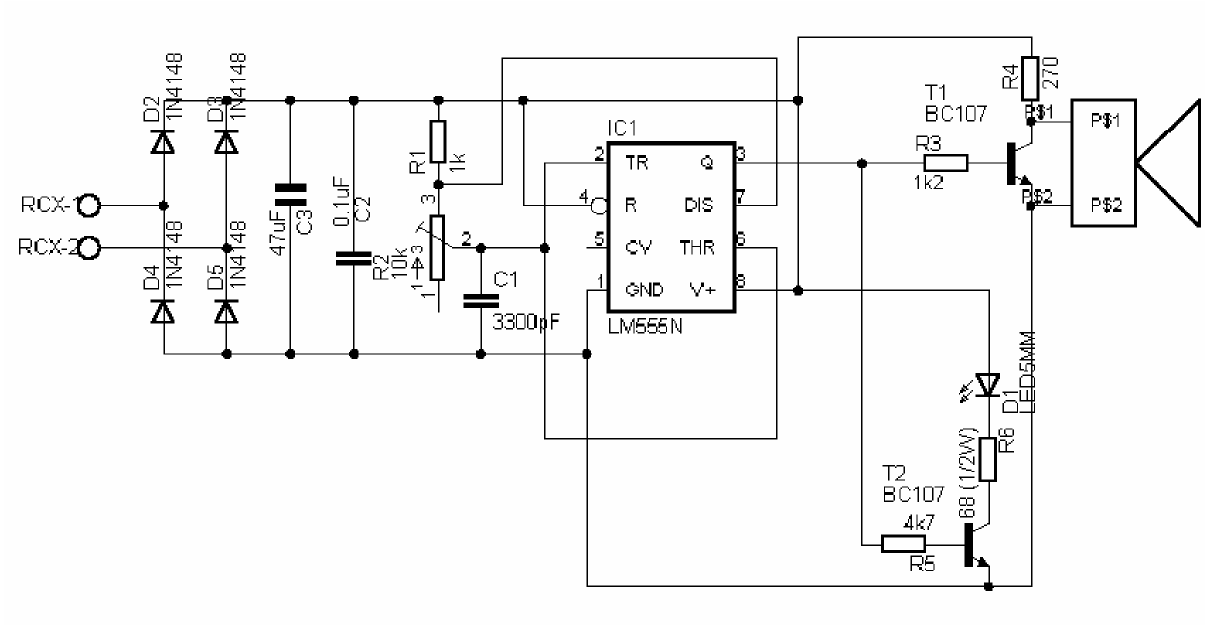


Figure 4.5: Transmitter circuit using a 555 timer[3]

Some improvements had to be made to the circuit. Different burst lengths and fixed timing was needed. The different burst lengths are for the receiver to distinguish between the different transmitters. A PIC microcontroller was used to realize this function. Dip switches were added to the circuit so that the different transmitters could use the same circuit and PIC programme. Each transmitter was given a unique identity by making use of dip switches.

Two 555 timers were still used, one to generate the 40 kHz for the ultrasonic sensor and one for the infrared transmitters, since they were running at 38 kHz (only available model, the ideal situation would be having 40 kHz IR transmitters). The other problem that occurred in the initial circuit was, that the IR transmitters used could either send a cone of light with a small angle and high amplitude or a big angle with small amplitude. The small angle would have worked if there were only one transmitter and one receiver. A different IR transmitter was used which was supplied with much more current than the initial IR transmitter LED. The final circuit used can be seen in Figure 4.6.

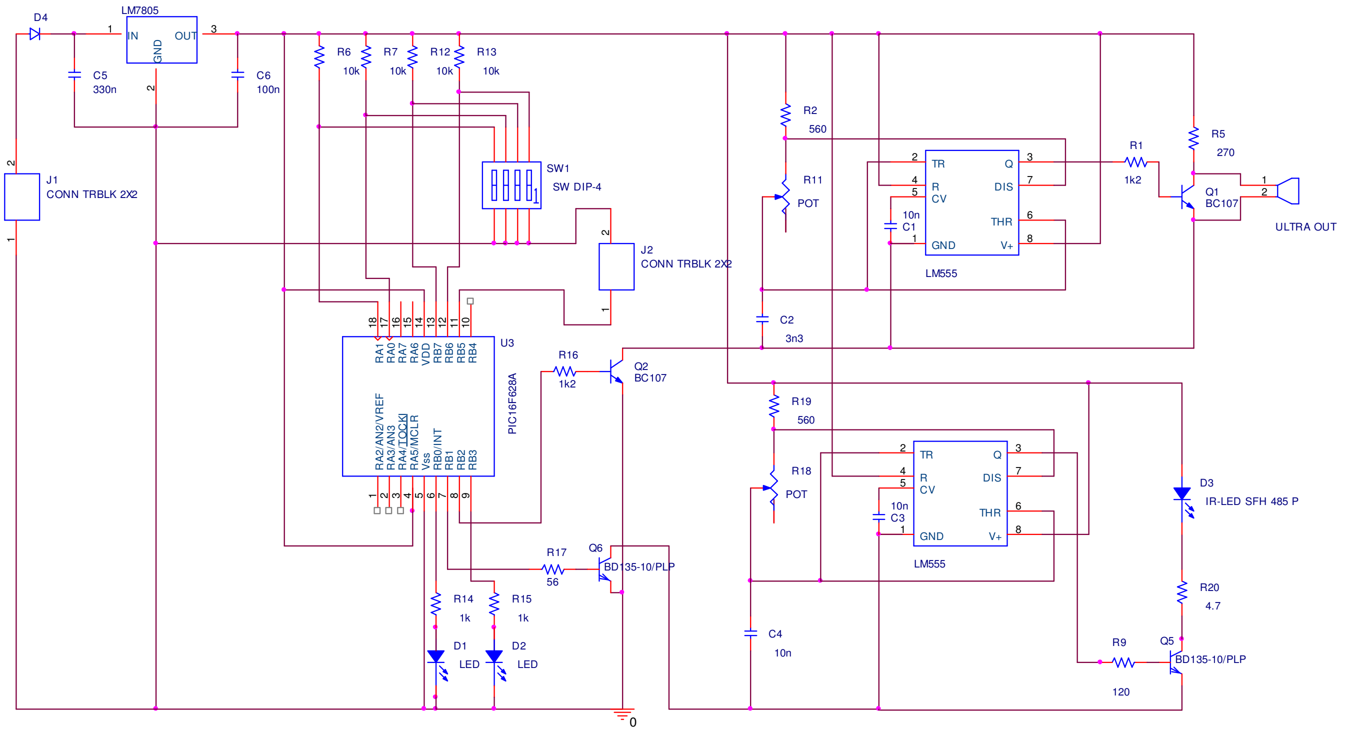


Figure 4.6: Final transmitter circuit

The software for this circuit is written in such a way that the three transmitters are asynchronous to each other and, therefore, there is no interference from the different transmitters on each other. Each transmitter has a different address which is set by the dip switches; the transmitter, named 00 has a special signal. Should the receiver loose the connection to the transmitter at any time, it will wait for the signal from 00 and then start the calculations of its position again. The three transmitters are connected together by one wire and this connection is used to indicate to the transmitter when it should start its transmission.

4.4 Receiver

The first version of the receiver, a circuit from Jeunes [3], was used as a starting point, as illustrated in Figure 4.7. The problems that were experienced with this circuit was that the accuracy differed with different distances and angles. The filter had to be changed to a 40 kHz band pass filter. The signal-to-noise ratio also had to be improved and this proved to be somewhat of a challenge due to the rising time of the ultrasonic sensors.

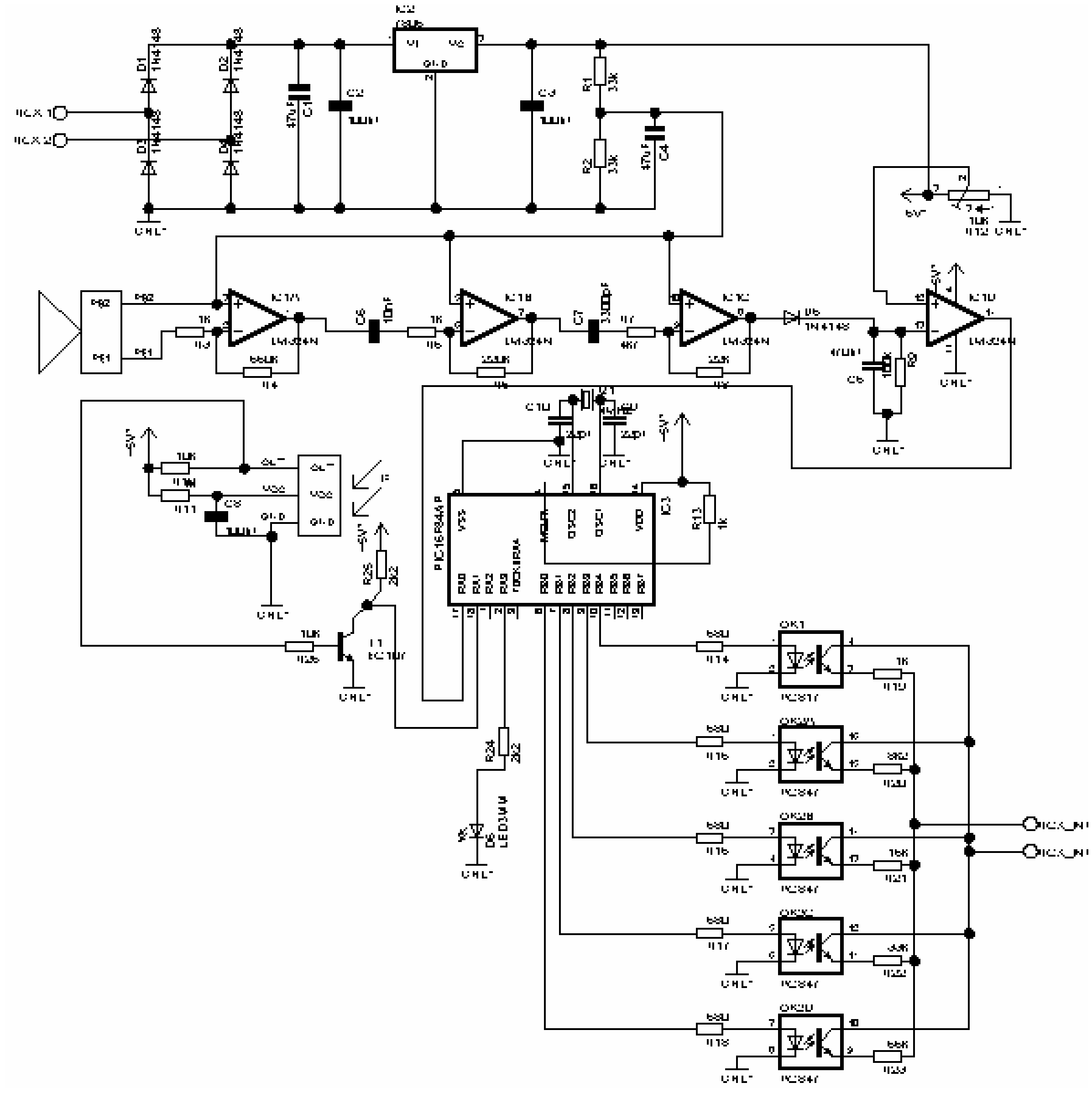


Figure 4.7: First version of receiver circuit[3]

To improve the accuracy of the system, the receiver circuit was changed, as shown in Figure 4.8. Different operational amplifiers were used with a higher bandwidth and input impedance. The gain of the first stage was about 500 and the second 5.

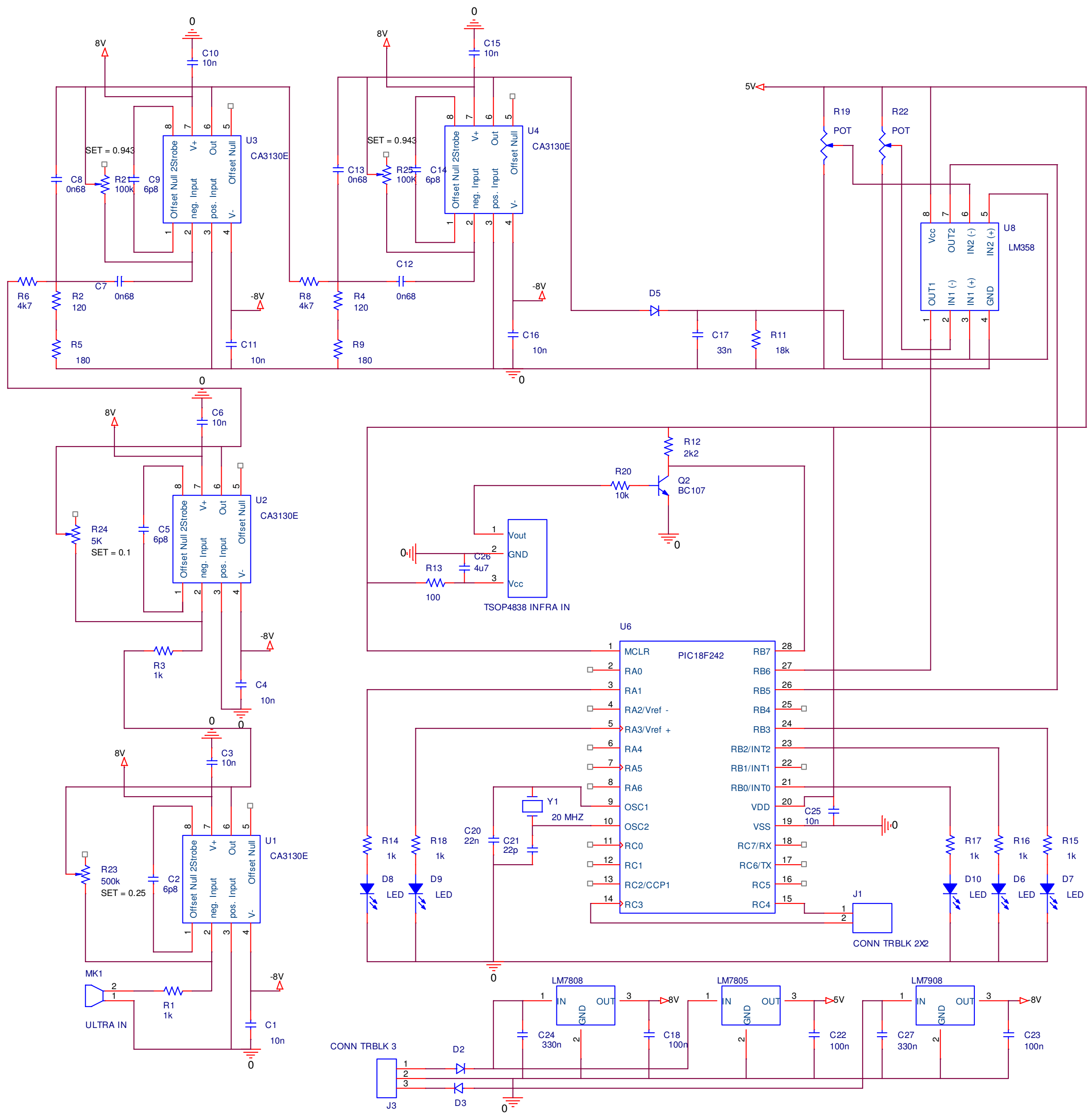


Figure 4.8: Improved receiver circuit

To lower the noise, a fourth order band pass filter was used and an additional gain of 10 was added. This caused the rising edge of the ultrasonic signal to be very steep, which improved the accuracy. To make this slope linear, a diode (D5), capacitor (C17) and a resistor (R11) were placed in the circuit. The capacitor is charged through the diode and discharges slowly through the resistor to produce an almost linear slope which rises at every positive edge of the 40 kHz signal.

The infrared input circuit is much easier. The receiving unit is connected via a capacitor (C26) and a resistor (R13) to the base of the BC107 transistor (Q2). If the infrared signal is received, it will produce 5 V on the input pin of the PIC microcontroller (PIC18F242).

To avoid the possibility of an invalid signal coming from other sound or light sources, some error avoidance had to be done in the software. Firstly, the infrared signal must appear first and then the lower threshold ultrasonic signal must follow and, lastly the ultrasonic signal with the higher threshold. If the signals are not in the correct order they will be ignored. Secondly, a timer was implemented in the software to have an indication whether the expected signal was on time or not. The accurate time for the ultrasonic signal can not be predicted, but since the range of the system is known it is possible to have a time frame to work according to. If it appears outside this time frame, the signal will be ignored. The last method for detecting errors is to compare the last calculated position with the new position. By using some odometry, it will be possible to determine if the coordinates are correct.

4.5 Synchronisation

It is very important for the receiver to know which transmitter is transmitting at a specific time. This synchronisation is done when the AGV is started and when an error occurs. It is done by implementing a special code in the infrared signal of beacon 00. When the programme is started, the receiver will look for this unique code before it starts operating. If the code is found, the calculations start and the receiver can determine its coordinates.

4.6 Determining the Position

When calculating the coordinates of the receiver, the distance to each beacon is needed and this is done in the following way. The method used is triangulation and this principle is shown in Figure 4.9 [7].

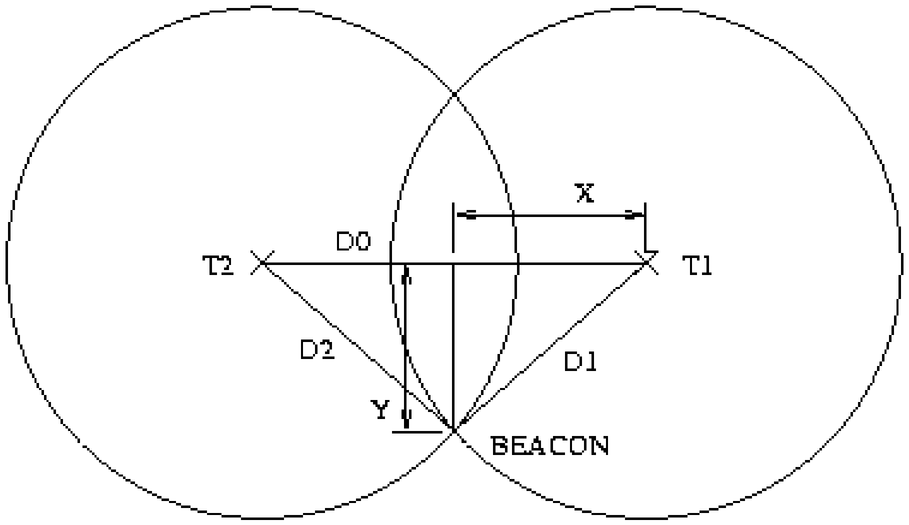


Figure 4.9: Triangulation principle

If the values for D0, D1 and D2 are known it is possible to calculate the distances that are represented by X and Y.

$$X = \frac{D1 \cdot \left(D0^2 + D1^2 - D2^2\right)}{2 \cdot D0 \cdot D1} \qquad [4.2]$$

$$Y = \sqrt{D1 \ - X^2} \qquad [4.3]$$

For every infrared signal received, an ultrasonic signal is to follow. The receiver measures the time it takes between the reception of these two signals in the order mentioned. The closer the receiver gets to the beacon, the steeper the slope of the ultrasonic signal becomes – this is due to the fact that, as the distance increases the slope becomes flatter. The time lost can be calculated by using a linear equation, as illustrated in Figure 4.10 [3].

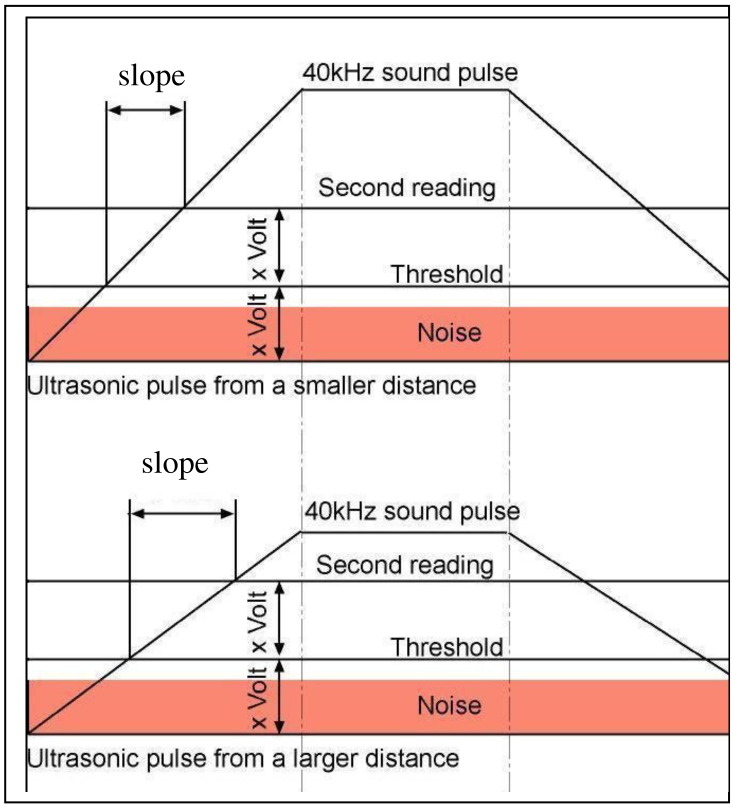


Figure 4.10: Ultrasonic pulse from different distances

The calculations used to calculate the coordinates, as shown in Figure 4.11., are [3]:

$$(\mathrm{x}_1 - x_R)^2 + (\mathrm{y}_1 - y_R)^2 = d_1^2 \qquad [4.3]$$

$$(\mathrm{x}_2 - x_R)^2 + (\mathrm{y}_2 - y_R)^2 = d_2^2 \qquad [4.4]$$

$$(x_3 - x_R)^2 + (y_3 - y_R)^2 = d_3^2 \quad [4.5]$$

suppose $y_1 \neq y_2$

$$A = \frac{d_1^2 - d_2^2 + x_2^2 - x_1^2}{2(y_2 - y_1)} + \frac{y_2 + y_1}{2} \quad [4.6]$$

$$B = \frac{x_2 - x_1}{y_2 - y_1} \quad [4.7]$$

$$y_R = A - Bx_R \quad [4.8]$$

$$ax_R + bx_R + c = 0 \quad [4.9]$$

$$a = 1 + B^2 \quad [4.10]$$

$$b = 2y_1B - 2x_1 - 2AB \quad [4.11]$$

$$c = -d_1^2 + x_1^2 + y_1^2 + A^2 - 2y_1A \quad [4.12]$$

$$x_R = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad [4.13]$$

Provided that b^2-4ac > 0

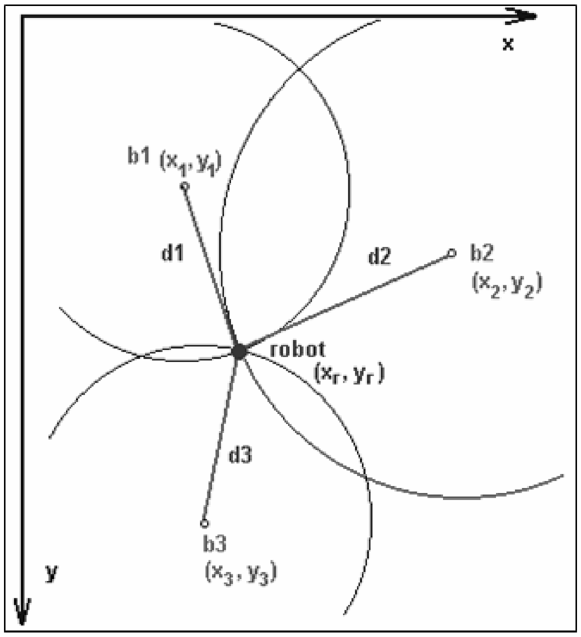


Figure 4 11: Circle diagram for showing the position of the AGV.

4.7 Results

For final testing, the beacons (transmitters) were mounted on the ceiling pointing straight towards the ground. The height was fixed at 2,7 m since that is the height of the ceiling in the room that was used for testing. The system that was installed is shown in Figure 4.12, the receiver has been placed in the middle.

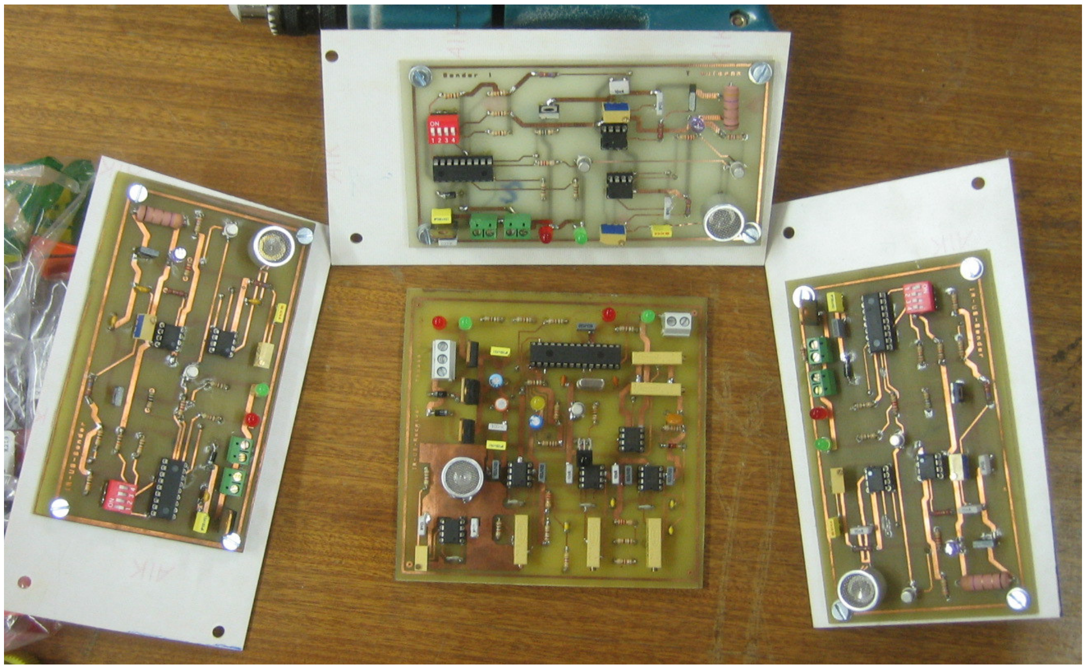


Figure 4.12: Transmitters and receiver of beaconing system

A grid was drawn on the floor of the room for testing purposes, as well as to determine the accuracy of the system. The layout of the grid can be seen in Figure 4.13.

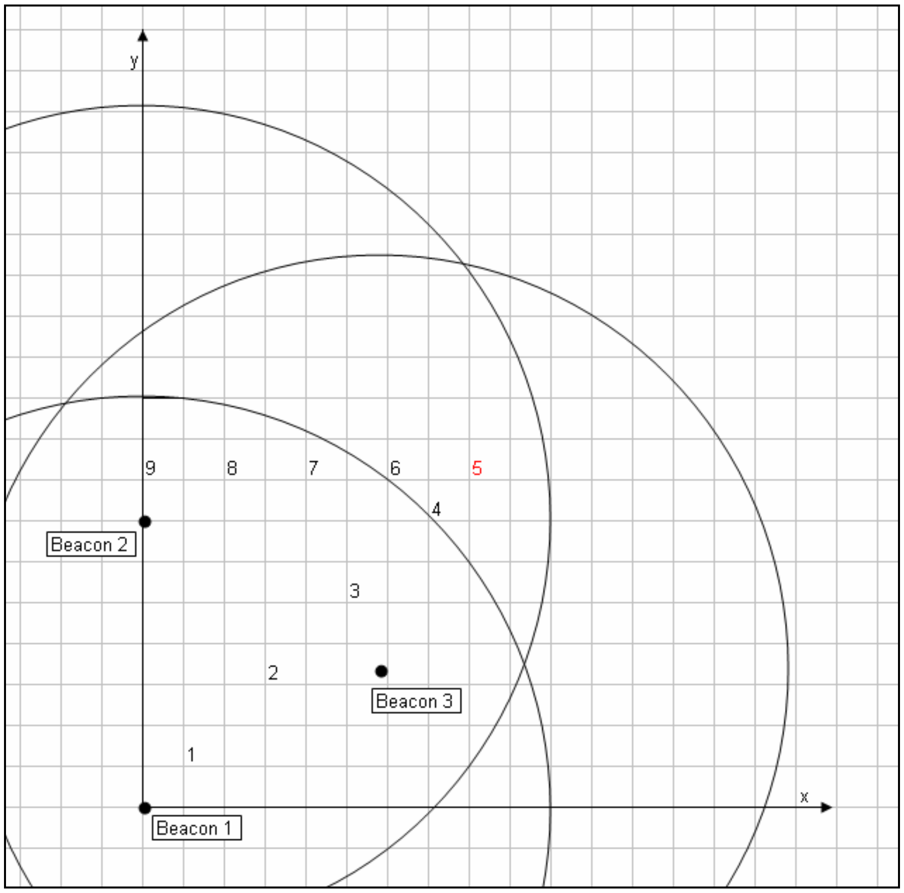


Figure 4.13: Grid layout used for testing purposes

The circles in Figure 4.13 show the area covered by each transmitter. The numbers one up to nine indicates the positions that the tests were conducted in and the results can be seen in Table 4.3. When the angle between transmitter and receiver goes beyond 50°, it becomes difficult to see the difference between the noise and the signal. The radius of the circle covered by one transmitter is 3 m. With a radius of 3 m, the angle between transmitter and receiver is approximately 47°, as shown in Figure 4.14.

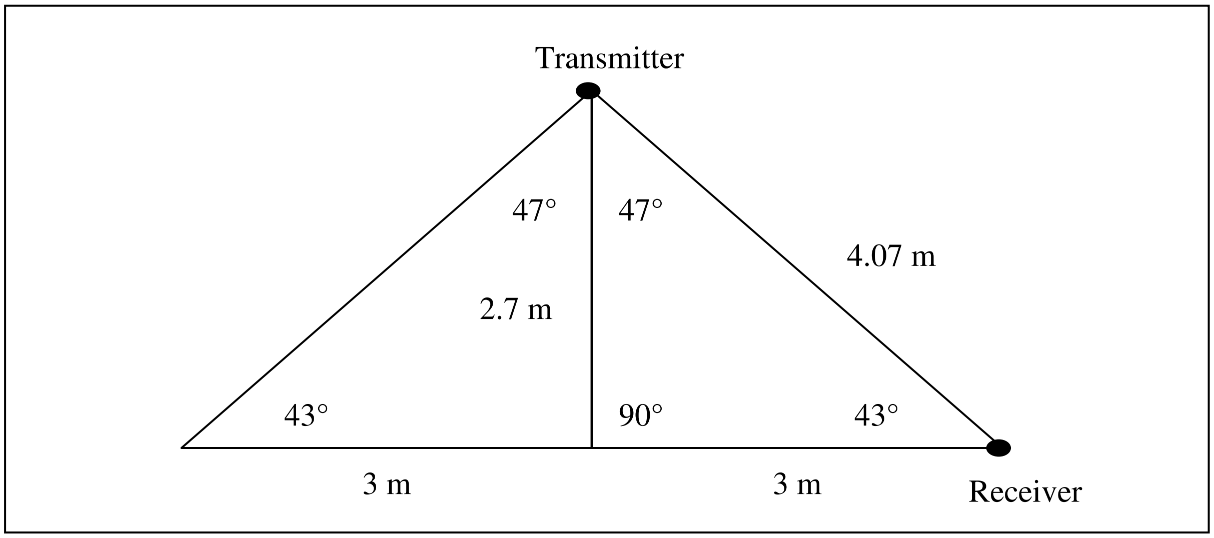


Figure 4.14: Triangle from transmitter to receiver

The final results can be seen in Table 4.3 and all values are in centimeters, the highest being 290 cm.

Table 4.3: Final test results

Position	Physical coordinates		Beaconing system coordinates		Error in cm	
	x	y	x	y	x	y
1	42	29	40	25	2	4
2	102	89	101	86	1	3
3	162	149	164	148	2	1
4	222	209	217	217	5	8
5	252	239	202	290	50	51
6	192	239	195	237	3	2
7	132	239	134	240	2	1
8	72	239	72	241	0	2
9	9	239	10	240	1	1

The distance to each beacon is calculated using the method explained earlier in the chapter and the coordinates are calculated from there. From these results, it is clear that when the receiver is on the edge, or just beyond the edge, of the circle of beacon 1, the results obtained are still in range and can be used. At position 5, the receiver falls outside the operating range. In other words, only two of the three beacons are detected accurately and this explains the large margin of error. Beacon 1 is still detected but the angle becomes too big. At position 5 the angle between transmitter and receiver is approximately 53° and, as expressed earlier, when the angle becomes bigger than 50° it becomes difficult to distinguish between the signal and noise.

If an error occurs, the receiver calculates its coordinates every 450 ms. If the receiver is kept on the same spot, and position calculations are done, then the differences between these positions will be between 3 and 4 cm. To lower this effect, the receiver software can be modified to calculate an average position every second, or longer, depending on the application.

When reviewing this it is evident that the beaconing system works sufficiently if three beacons are detected. There are a few elements that still produce a number of problems, such as the influence that temperature has on the speed of sound and the environmental noises. The system can, however, be improved by mounting the transmitters at a 45° angle, as suggested by Priyantha *et. al.* [1]. Other improvements to the system will be discussed in Chapter 6.

4.8 References

1. PRIYANTHA NB, CHAKRABORTY A, BALAKRIHNAN H, The Cricket Location-Support System, Proceedings of 6th annual international conference MobiCom'00, ACM Press, Boston, 2000, pp 32-43

2. HIGHTOWER J, BORRIELLO G, Location Systems for Ubiquitous Computing, IEEE Computer, August 2001, pp. 57-66.

3. CONVICT EPISCOPAL DE LUXEMBOURG, Infrared and ultrasonic beaconing, www.restena.lu/convict/Jeunes/beacon.htm, 2001

4. HALAWAY D, HŐLLERER T, FEINER S, Coarse, Inexpensive, Infrared Tracking for Wearable Computing, Proceedings of the 7th IEEE International Symposium on Wearable Computers, White Planes, New York, 2003, pp. 69-78.

5. ELERT G, The Physics Factbook, http://hypertextbook.com/facts/2000/CheukWong.shtml, 2000

6. SENGPIEL E, http://www.sengpielaudio.com/calculator-speedsound.htm, 2006

7. ALA-PAAVOLA J, Triangulation Location Beacon, http://vodka.tky.hut.fi/~jap/Eurobot/doc/beacon.html, Helsinki University of Technology, 2005.

Chapter 5

Object Avoidance

5.1 Introduction

For the AGV to be a viable option for operation in industry, it is essential that it is able to avoid obstacles in the environment. In short, the AGV needs "eyes" to see what is happing around it. There are many solutions to this problem. However, the method that was used in this study is infrared (IR) sensors, due to the fact that it was the most cost effective method for object detection and it was easy to implement.

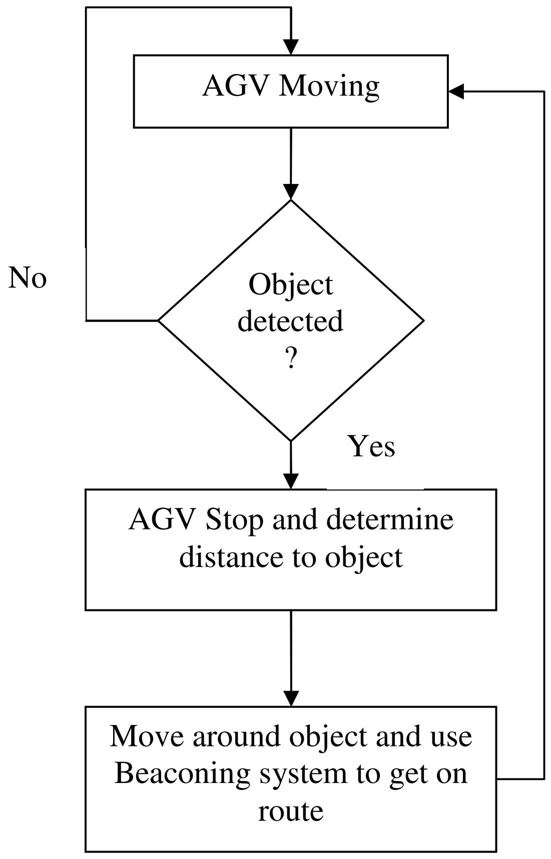

AGV Moving
No
Object detected ?
Yes
AGV Stop and determine distance to object
Move around object and use Beaconing system to get on route

Figure 5.1: Proposed flowchart for object avoidance

5.2 Infrared Sensors

Infrared (IR) sensors (Sharp GP2D12) are used for determining the distance from an object. The sensor is shown in Figure 5.2 [1] and Figure 5.3. This sensor can be connected to a microprocessor using the Analogue-to-Digital Converter (ADC).

Figure 5.2: GP2D12 Infrared sensor

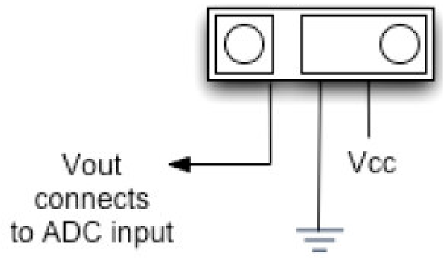


Figure 5.3: GP2D12 Infrared sensor connection

The sensor must be supplied with power and ground, and the third pin is the output voltage (V_{OUT}) which is proportional to the distance measured [1]. A capacitor must be placed on the sensor itself between ground and the supply to limit the noise. There must be no capacitor between ground and V_{OUT} or the supply voltage and V_{OUT}, as this would cause the sensor to give out the wrong values.

The fact that the GP2D12 infrared sensor is low in cost, compact and lightweight makes it one of the most commonly used range finders in robotic applications. The GP2D12 infrared sensor operation is based on the triangulation principle. This principle provides greater reliability and accuracy than the IR sensors, which make use of Time-of-Flight techniques [2]. It uses an IR LED for transmitting a burst of pulses. If there is no object in front of the sensor then the light will never return to the receiver, because there is nothing for it to reflect against. Hence, after 40 ms, it will show that there is no object for 80 cm in front of the sensor. If there is an object in front of the sensor, it will reflect back from the object to the receiver, in this case a Position Sensing Device (PSD), and this will form the shape of a triangle. This is illustrated in Figure 5.4 [3]. It can measure distances between 10 and 80 cm, and the update frequency is 25 Hz or every 40 ms [4].

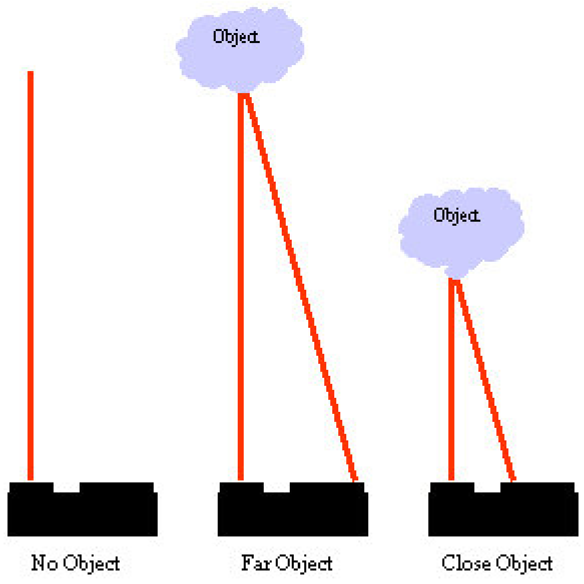


Figure 5.4: Operation of GP2D12 Infrared sensor for different distances

Using this triangulation principle, as shown in Figure 5.5, it is clear that there will be different angles for different distances [5]. These sensors are also less affected by the ambient lighting conditions, and the colour of different objects does not have that big an influence on the output [2].

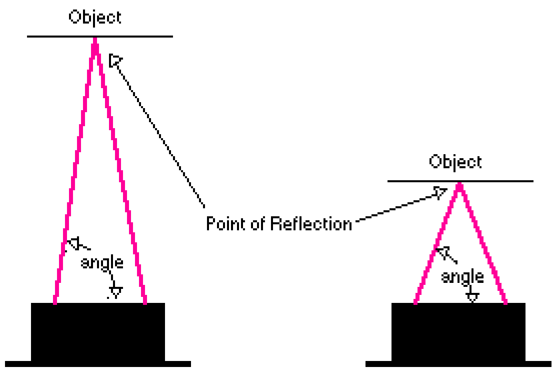


Figure 5.5: Different angles for different distances

5.3 Output Characteristics of Sensor

When the sensor is supplied with 5 V, the measured output voltage for the different sensors for a distance of 10 cm, was between 2,2 V and 2,5 V. When the object was approximately 80 cm away, the readings on the different sensors varied between 0,41 V and 0,44 V. Two sets of the test data, obtained in the lab using one GP2D12 infrared sensor, are shown in the graph in Figure 5.6.

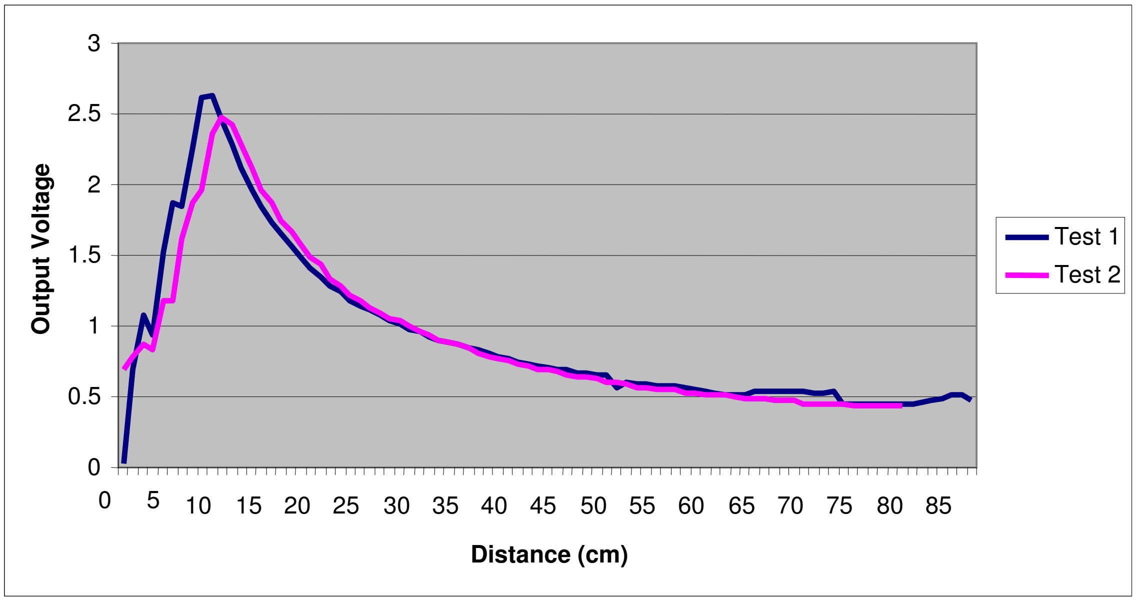


Figure 5.6: Test data plotted on a graph

This graph can be compared to the output characteristics given on the datasheet of the GP2D12 Infrared sensor, as shown in Figure 5.7 [1].

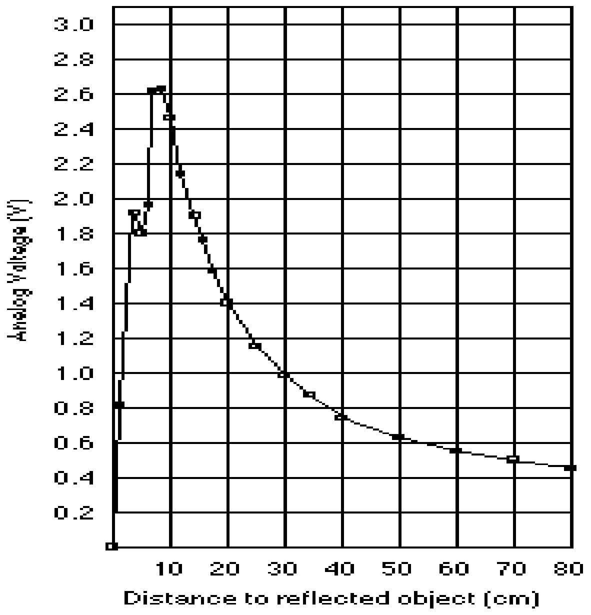


Figure 5.7: Sharp GP2D12Infrared sensor output characteristics[4]

It is clear that the output voltage of the sensor between 10 and 80 cm is logarithmic [4]. Since it would be much easier to use the data if it were a linear function, a formula that can be used to turn the data into a linear function was developed [4]. The formula for this is:

$$V_{OUT} = \frac{1}{\text{distance} + 0.4} \qquad [5.1]$$

Now instead of plotting distance vs. V_{OUT}, one can plot Equation 5.1 vs. V_{OUT}. This will give an output characteristic similar to a straight line.

Since a voltage is measured and the distance is the desired outcome, this formula, together with the straight line formula (Equation 5.2), can be combined to calculate the distance.

$$y = mx + b \qquad [5.2]$$

$$\text{Distance} = \frac{1}{m \times V_{OUT} + b} - 0.4 \qquad [5.3]$$

The m and b values can be determined when the sensor is calibrated. To calibrate the sensor, voltage readings must be taken every 10 cm, starting at 10 cm and ending at 80 cm [1]. By plotting these voltage readings versus Equation 5.1, one can obtain the trend line and, from this line, it is possible to determine the b and m values, as shown in Figure 5.8.

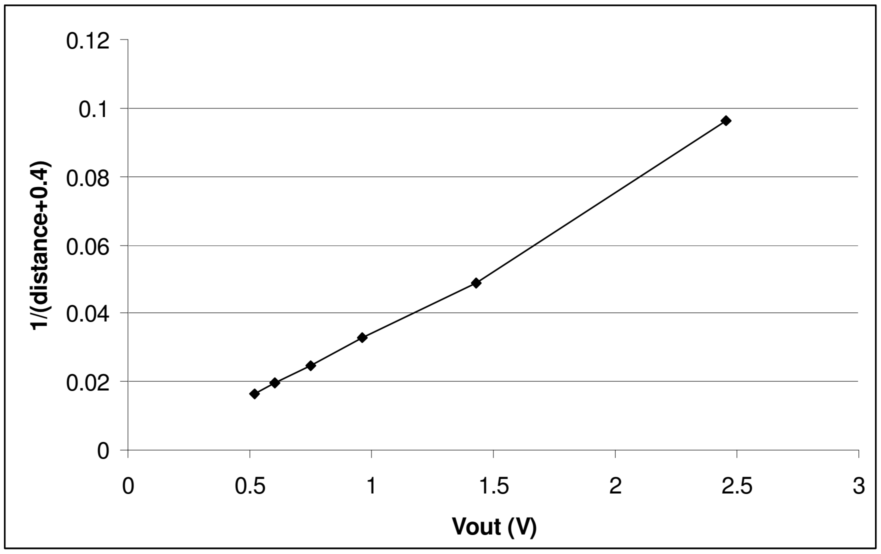


Figure 5.8: Trend line analysis

5.4 Interfacing the Infrared Sensor to the Microcontroller

Eight GP2D12 Infrared sensors are connected to a microcontroller (PIC16F877A) using the analogue inputs. For best resolution 10 bits are used for the incoming data. This will result in a resolution of better than 1 cm across the data range used. Using 10 bits will also make it possible to get readings in millimeters. At a range of 600 mm, the precision of this sensor is almost 20 mm and it becomes more accurate for closer distances (2 mm for 200 mm). One lookup table was used for all the sensors, but since all sensors are not exactly the same this decreased the precision a little.

For this study the range for the sensors was limited to between 10 and 60 cm, and on the lookup table used it would be values of between 103 and 468. Anything smaller than 103 is less than 10 cm and thus not used, whereas anything more than 468 is greater than 60 cm and would also display that the object is far away or not yet in the way. The input data is averaged to be able to improve the precision.

The sensor takes approximately 40 ms for one reading and therefore the program was written to take continuous readings at 40 ms intervals [5]. That means that it is almost 25 readings per second [2].

5.5 Placement of the Infrared Sensors

Based on the test data obtained, a lookup table was implemented in the microcontroller to be able to change the analogue voltage into a distance value. The range of the sensors used for this application was limited in software to between 10 cm and 60 cm. The reason for this can be solely based on the fact that, after 60 cm, the difference in output voltage of the sensor is almost nothing. From the graphs in Figure 5.6 and 5.7, it is evident as to why the operating range is specified from 10 cm onward and not before 10 cm. The sensors output will seem like it is further away than it really is which can cause the AGV to collide with the object. To avoid this scenario, the sensors were placed at specific positions on the AGV, as shown in Figure 5.9.

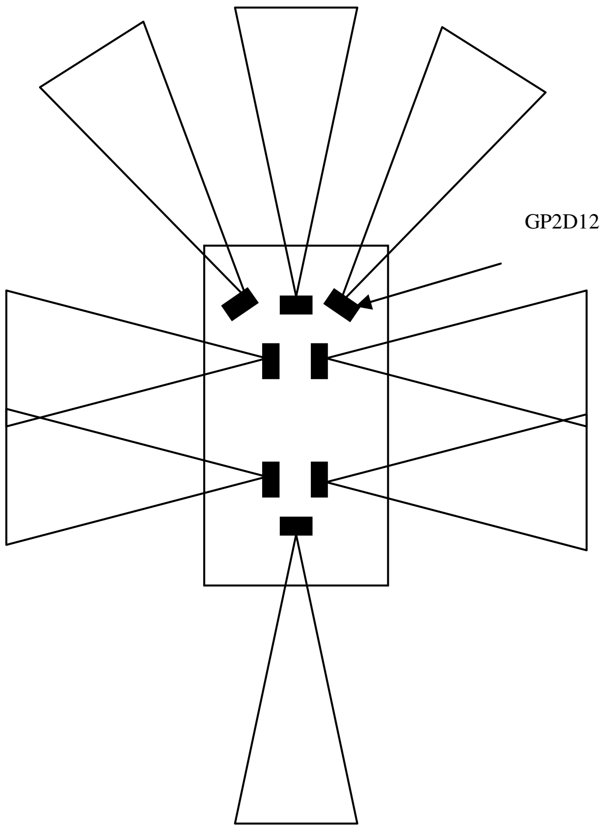


Figure 5.9: Positioning of Infrared sensors on AGV.

5.6 Problems Experienced

A number of problems were experienced when more than one sensor was connected to the same microcontroller. It seemed as though the different sensors were interfering with one another. Adding capacitors directly to the power input, as mentioned earlier, improved the performance of the sensors. After investigation, it was found that this problem could actually be overcome by changing the sensor to a GP2D02 infrared sensor, which also needs a clock pulse and, for this reason, it will use less power than the GP2D12 infrared sensor. The output of the GP2D02 infrared sensor is an 8 bit digital value.

Another problematic element was light. When the sensors were exposed to direct sunlight or a very bright light source, the measured distances was not exact. This problem was solved by placing the sensors in such a way that they will not come into direct contact with these light sources.

The ambient temperature also had a small influence on the accuracy of the sensor, as shown in Table 5.1 [4].

Table 5.1: Influence of temperature on the sensor

	Analogue output voltage at different temperatures			
Distance	0°C	20°C	40°C	60°C
10 cm	2.3 V	2.33 V	2.35 V	2.41 V
20 cm	1.29 V	1.31 V	1.34 V	1.38 V
30 cm	0.9 V	0.91 V	0.95V	0.99 V
40 cm	0.7 V	0.72 V	0.74 V	0.77 V
50 cm	0.59V	0.6 V	0.62 V	0.66 V
60 cm	0.5 V	0.52 V	0.56 V	0.58 V
70 cm	0.42 V	0.48 V	0.5 V	0.51 V
80 cm	0.28 V	0.38 V	0.4 V	0.42 V

Essentially, the object avoidance method was satisfactory, although this part of the study leaves room for a lot of improvement. These suggested improvements will be discussed in more detail in Chapter 6.

5.7 References

1. MICROMEGA CORPORATION, Measuring Distance with the Sharp GP2D12 and GP2D120 Distance Sensors, uM-FPU Application Note 4, http://www.micromegacorp.com, 2005.

2. BLANCH J, TOSUNOGLU S, Servo and Sensor Control on Small Mobile Platforms, ASME Southeastern Region XI Technical Journal, Volume 2, Number 1, April 2003; also presented at ASME South-eastern Region XI Technical Conference, Miami, Florida, April 2003.
3. NAFIS C, Automatically Measuring Snowfall / Snow Depth with an Inexpensive Device, http://www.howmuchsnow.com/snow/#background#background, 2004.
4. SHARP, General Application Note: Distance Measuring sensors, ELECOM group Sharp Corporation, http://www.junun.org/MarkIII/datasheets/sharp-app-note.pdf, 2003.
5. ACRONAME, Demystifying the Sharp IR Rangers, www.acroname.com/robotics/info/articles/sharp/sharp.html, 1994.

Chapter 6

The remote control car was successfully transformed into an automatic guided vehicle. The principles of what was done in this study can now be used in the further development of automatic guided vehicles or mobile robots.

6.1 Summary

- Chapter 1

  In Chapter 1, an introduction to the study was given and the steps to be followed were laid out.

- Chapter 2

  The current technologies and trends concerning AGVs were discussed.

- Chapter 3

  The AGV base was equipped with hall-effect sensors and precision potentiometers in order to control and monitor the AGV speed, direction and distance travelled. A joystick was also added and a LabView interface was used to do the monitoring and controlling. The communication method used between the PC and the AGV was Bluetooth.

- Chapter 4

  A beaconing system was developed for the navigation of the AGV. This system used both infrared and ultrasonic sensors. The AGV needs to "see" at least 3 of the transmitters that were mounted on the ceiling in order for it to be able to determine its position.

- Chapter 5

  The most cost-effective method was using infrared sensors to do object avoidance. The sensor used was from Sharp GP2D12. It is an analogue sensor working within the range of 10 to 80 cm. In this application, the sensors were placed at specific positions on the AGV to make provision for the first 10 cm and the maximum distance was limited to 60 cm.

A complete architectural layout of the project can be seen in figure 6.1.

6.2 Results of Project

The AGV can be controlled manually using a LabView interface, a joystick or it can be pre-programmed to follow a specific route using beaconing and object avoidance. Since both a manual and automatic interface are used, the need for some method of communication was evident and Bluetooth was the one best suited to this study. The AGV can be up to 30 m away from the PC and still receive the correct instructions. The AGV is also transmitting a certain amount of information to the PC so that it is possible to know the state of the AGV and whether it is necessary to change from automatic to manual mode.

The problem of navigation was solved by using a beaconing system. This system used both ultrasonic and infrared sensors. The transmitter units were mounted perpendicular to the ceiling and, by doing this, a 3 m radius was covered. This radius can be increased by mounting the transmitters at an angle. Three transmitters were used and, for absolute accuracy, it was necessary to “see” all three transmitters in order to determine a position. Since the distance travelled is also measured it would be possible to use odometry. If this odometry were combined with the beaconing system, it would be much more accurate since it would solve the problem of wheel-slippage and other environmental effects. The drawback of the beaconing system is that it is confined to indoor use only and that the transmitter needs to be mounted before it can work.

The question of object avoidance was addressed using infrared sensors although many other possibilities are available. This seemed to be the most cost effective solution. GP2D12 Sharp IR sensors were placed onto the AGV and, by using a microcontroller interface, the information regarding objects in range was obtained.

When all of the different parts of the project were integrated some problems were experienced with noise or interference between the different parts. The motor had quite an influence on the ultrasonic sensor that was mounted on the receiver of the beaconing system. This problem was partially solved by moving the receiver further away from the motor.

Essentially, all the parts of this AGV are fully operational, but there is always room for improvement.

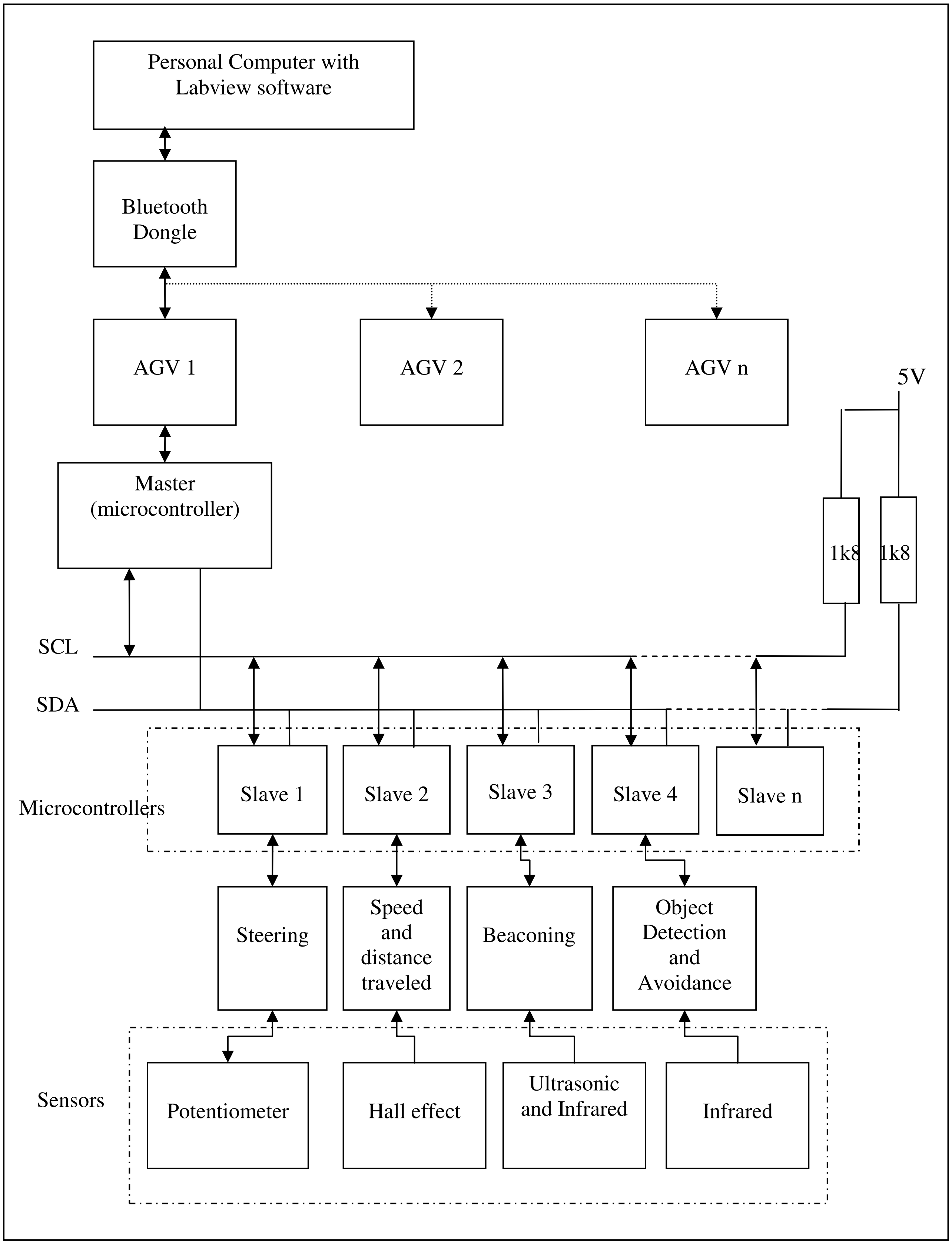


Figure 6.1: System Architecture

6.3 Recommendations for Future Research

Most projects have a certain amount of room for improvement and, this project is no exception. The main aim of this project was to develop an automatic guided vehicle. In industry an AGV is much more versatile than a RC car and, therefore, it leaves a lot of room for improvement and expansion of the system.

6.3.1 AGV Base and Steering

The AGV base can be changed to a bigger AGV, and built from material used in industry, such as metal. The AGV can be modified to be able to carry more weight and larger objects. With the current project, the space for objects on top of the AGV are limited due to the size of the AGV.

The steering control in this project was limited by the potentiometer used and, the increments were 7° apart. This could be improved by using another potentiometer or even another type of sensor.

6.3.2 Beaconing System

To upgrade the beaconing system, a temperature sensor can be added to update the speed of sound in the calculations, since the temperature has an influence on the speed of sound. More transmitters can be built to increase the operating area of the system. The angles at which the beacons are mounted can be adjusted to increase the area even more. The receiver can be made to turn so that it is able to track the transmitters. To do this, one can include a stepper motor so that the receiver is able to rotate in order to pick up the different signals. Other methods can be explored.

6.3.3 Object Avoidance

This topic leaves room for a lot of improvement. Firstly, if one decides to continue using the Sharp sensors then the system can be improved by replacing the GP2D12 sensors with GP2D02 sensors. The GP2D02 sensors cause less interference on each other since the output of this sensor is an eight bit digital value. The main difference between analogue and digital is that a digital signal is a discrete signal. It has two possible values, on (1) or off (0). This makes it less susceptible to noise. Digital signals are easier to transmit and the chances of error are much less [1]. Secondly, if it is not possible to change to the GP2D02, the GP2D12 can be attached to a servomotor in order for it to take more than one reading of the area [2]. By using this method, the number of sensors needed, can also be reduced. The system can also be more accurate since more than one sensor can be used to determine the distance to an object [2].

A gyroscope or compass sensor can be added to the system so that it is possible to move around the obstacle and be on exactly the same track again.

It is also possible to improve the object avoidance by implementing more than one type of sensor. For instance, if the IR sensors are combined with ultrasonic sensors it could improve the ability of the AGV to avoid objects.

6.3.4 Integrated Vision Capability

The ultimate solution for eyes for the AGV will be a camera. By making use of image processing methods it would be possible for the AGV to recognise the different obstacles in the work place. The camera must be able to turn from left to right and right to left to be able to explore the whole room.

6.3.5 Communication

Bluetooth's limitation is its range and, therefore, this topic also has some room for improvement. If the AGV is used in a confined area and will travel in a radius of approximately 30 m, Bluetooth will work perfectly. The moment the AGV needs to travel greater distances it would be necessary to look at other means of communication.

6.3.6 Recording Information

All the information from the AGV is fed to LabView via the Bluetooth communication. This information can be logged, and used to provide detailed information about the route that the AGV has travelled. The system can even be elaborated to provide a complete graphical interface of the AGV and the condition of it, as well as the information recorded about the environment travelled in.

6.3.7 Other Improvements

The AGV can be equipped with load cells to determine the weight of the load that is placed onto it.

A docking station can be built for automatically charging the batteries of the AGV, thereby eliminating the need for any human interface. To be able to do this, battery charging contacts, see Figure 6.2 [3], must be installed on the AGV and in the docking station. When the AGV detects that the battery power is getting low, it should move towards to closest docking station where it must remain until the batteries are fully charged. Only then should the AGV return to its place of work.

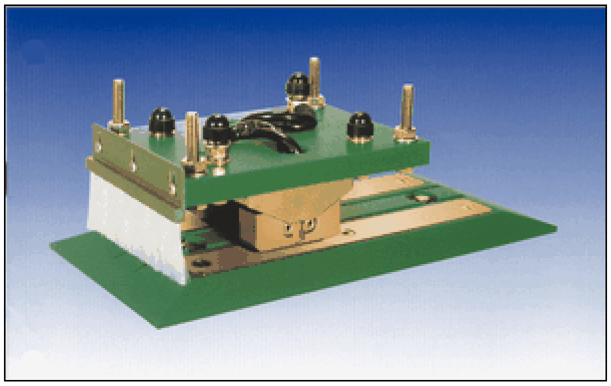


Figure 6.2: Battery charging contacts

6.4 Original Contributions of this Study

The accomplishments of this project can be summarised as:

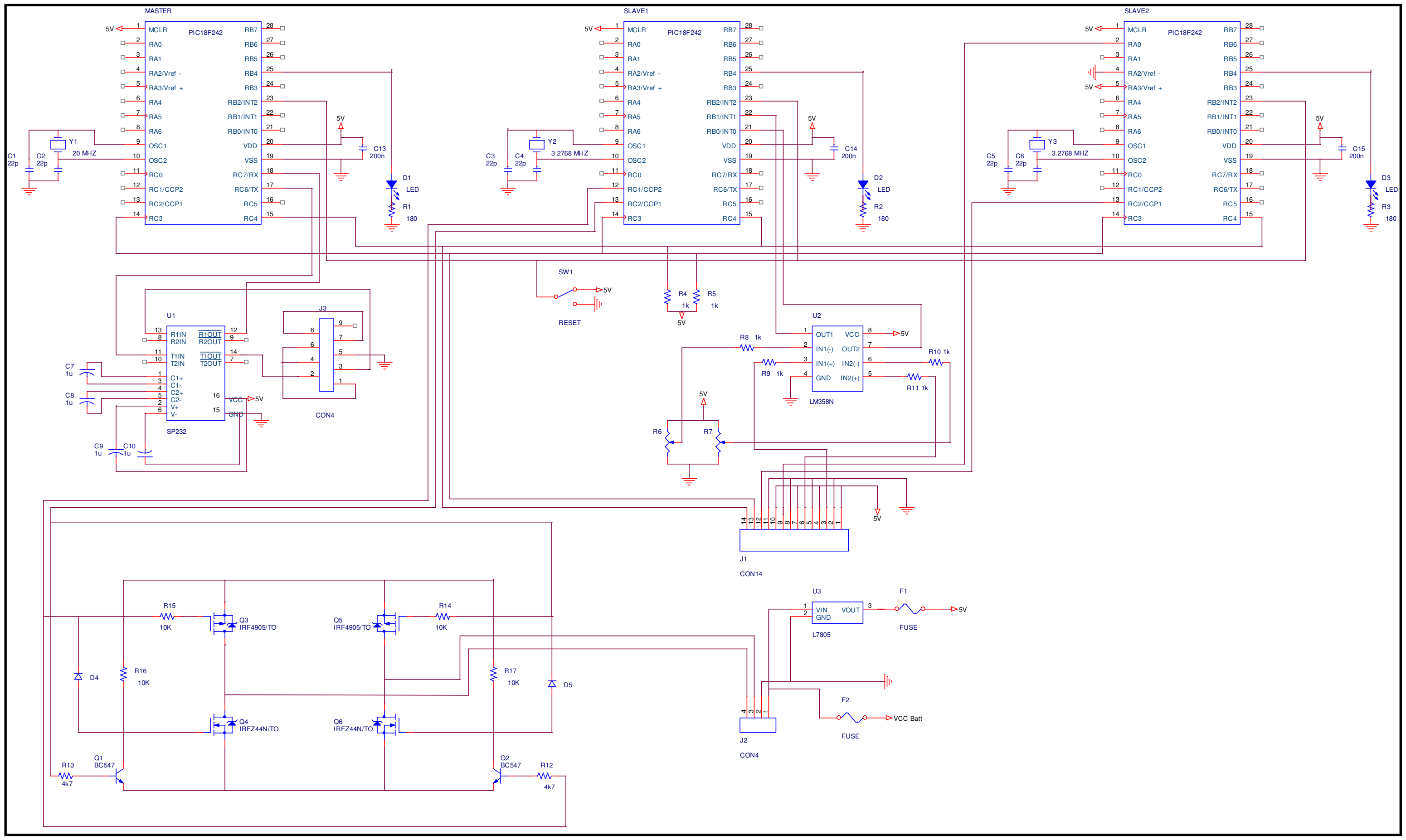
- The remote control of the RC car had to be replaced by a communication system between the PC and the AGV base. Bluetooth was used.

- The receiver of the RC car had to be removed and the car, now an AGV, needed to be controlled using the PC keyboard. The AGV motors needed some control and for this a h-bridge circuit and PIC microcontroller are used.
- Sensors were mounted on the wheel and servomotor to determine the speed, direction and distance travelled by the AGV. A LabView interface was developed to make the user interface easier.
- A master slave setup, with PIC18F242, was developed using I^2C communication.
- A special module on LabView was developed to make it possible to control the AGV with a joystick.
- A beaconing system, for navigation purposes, was developed.
- Object detection was done by using Sharp GP2D12 infrared sensors.

6.5 References

1. RODRIGUEZ E, Digital vs. Analog, www.skullbox.net/dva.php, 2004.
2. BLANCH J, TOSUNOGLU S, Servo and Sensor Control on Small Mobile Platforms, ASME South-eastern Region XI Technical Journal, Volume 2, Number 1, April 2003; also presented at ASME South-eastern Region XI Technical Conference, Miami, Florida, April 2003.
3. VAHLE Electrification Systems, Battery charging contacts, www.globalspec.com, 2006.

Appendix A: Schematic - motor control



Appendix B: Table of connections

<u>PIN – connector J4</u>

1. I²C external data
2. I²C external clock
3. Pulse servo motor
4. GND servo motor
5. VCC servo motor
6. Signal potentiometer
7. GND potentiometer
8. VCC potentiometer
9. Input hall-sensor1
10. GND hall-sensor1
11. VCC hall-sensor 1
12. Input hall-sensor 2
13. GND hall-sensor 2
14. VCC hall-sensor 2

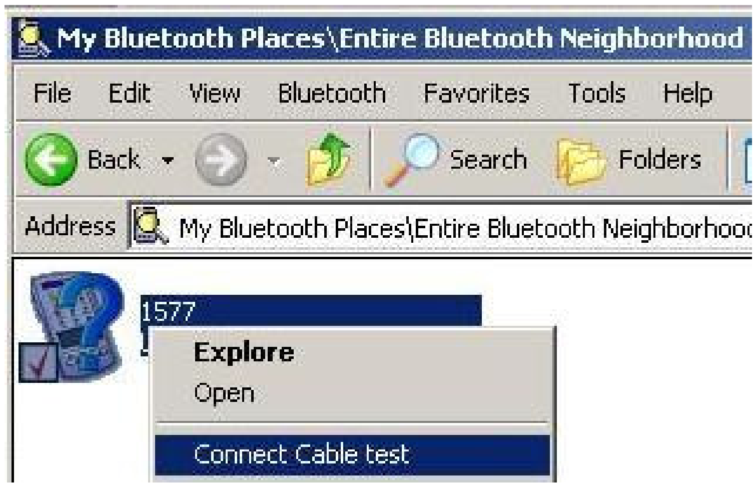
<u>PIN – connector J6</u>

1. VCC battery
2. GND
3. PWM signal main motor (right turn)
4. PWM signal main motor (left turn)

Appendix C: Getting the AGV up and running

This is a step-by-step instruction manual, to get the car running.

1. The batteries must be connected to the PCB and the Bluetooth device. After a while the LED next to the master PIC should start flashing.
2. By use of the Bluetooth device the car needs to be connected to the computer by use of the MSI Bluetooth tool. By selecting the 1577 device and left click on “Connect Cable test”, the connection should be established. The green LED on the Bluetooth device indicates a successful connection.


My Bluetooth Places\Entire Bluetooth Neighborhood
File
Edit
View
Bluetooth
Favorites
Tools
Help
Back
Search
Folders
Address
My Bluetooth Places\Entire Bluetooth Neighborhoo
1577
Explore
Open
Connect Cable test

3. Now LABVIEW can be run. Load the RCcar.vi file. A picture of the front panel can be found in the appendix E.
4. Start of the program by use of the “run” button.
5. The 16 bit input string from the car should now be indicated in the read string, digital display.
6. To make sure that the car starts out of the reset state the switch on the board should be activated to RESET the car. While the reset-button is on all LED’s on the board glow. After releasing the button the LED next to the master should start blinking again. All values of the read string should be zero.
7. Now the car is ready to move.
8. The car can either be controlled by use of the pointer slides or by use of the joystick connected to the game port. To control the car with the joystick, the “Joystick” button on the front panel must be activated.

Note: The car can be controlled much easier with the joystick. It might be necessary that the Joystick needs to be calibrated to keep the car stable while the Joystick is in its zero position. Use the slide buttons at the under-surface of the stick to do so.

9. To switch off the car, disconnect it from the batteries and stop the LabView program. The communication will break down.
10. In case of any malfunction the car should be put into the RESET state.

Appendix D: C Program code

D.1. Program code for Master

```
/*Program developed by: Martin Leonard and Ellenor Boje*/

#include <18F242.h>
#device adc=8
#use delay(clock=20000000)
#fuses NOWDT,WDT128,hs, NOPROTECT, NOOSCSEN, BROWNOUT, BORV20, NOPUT, STVREN,
NODEBUG, LVP, NOWRT, NOWRTD, NOWRTB, NOWRTC, NOCPD, NOCPB, NOEBTR, NOEBTRB
#use rs232(baud=19200,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
#use i2c(Master,fast,sda=PIN_C4,scl=PIN_C3,FORCE_HW)
#include <stdlib.h>

void write_slave1(byte slave_address);
void write_slave2(byte slave_address);
void read_slave1(byte slave_address);
void read_slave2(byte slave_address);
void receive_RDA();
void calculation_speed();
void calculation_acceleration();
void calculation_distance();
void initialize();

#define slave1_address 0x02
#define slave2_address 0x04
#define BUFFER_SIZE 16
BYTE buffer[BUFFER_SIZE];
BYTE next_in = 0;
char  inputstring[BUFFER_SIZE];
char  inputchar;

signed int8 angle_in;
int8  hallwheel;
int16 hallwheelcount;
float speed_out;
float buffer_speed[2];
float acceleration_out;
float distance_out;
int8  RTCCi;

/*Timer interrupt - I2C and RS-232 transmission - calculation*/
#int_RTCC
RTCC_isr()
{
  if(RTCCi == 22)
/*Timer0 overflow every 6.5ms - value 22 sets transmission interval
 present setup = 149,5 ms*/
  {
    output_high(PIN_B4);
    read_slave1(slave1_address);
    read_slave2(slave2_address);
    calculation_speed();
    calculation_acceleration();
    calculation_distance();
                printf("%2.1f,%2.1f,%3.1f,%d\n",speed_out,acceleration_out,
                    distance_out,angle_in);
    output_low(PIN_B4);
    RTCCi = 0;
  }
  else
  {
    RTCCi++;
  }
}

/*This interrupt handles the incoming datastring from the computer via RS-232*/
#int_RDA
RDA_isr()
{
  inputchar = getc();
  if(inputchar == 0x0d)
    {
      receive_RDA();
    }
  else
    {
      buffer[next_in] = inputchar;
      if (next_in < BUFFER_SIZE)
        next_in = (next_in + 1);
    }
}

/*RESET PIC*/
#int_ext2
EXT2_isr()
{
  while(Input(PIN_B2) == 1)
  {
    output_high(PIN_B4);
  }
    initialize();
    printf("%2.1f,%2.1f,%3.1f,%d\n",speed_out,acceleration_out,distance_out,angle_in);
    reset_cpu();
}

/*I2C writes from MASTER to SLAVE1*/
void write_slave1(byte slave_address)
{
  i2c_start();
  delay_ms(1);
  i2c_write(slave_address);
  delay_ms(1);
  i2c_write(inputstring[4]);
  delay_ms(1);
  i2c_write(inputstring[5]);
  delay_ms(1);
  i2c_write(inputstring[6]);
  delay_ms(1);
  i2c_write(inputstring[7]);
  delay_ms(1);
  i2c_stop();
}
```

```
/*I2C writes from MASTER to SLAVE2*/
void write_slave2(byte slave_address)
{
  i2c_start();
  delay_ms(1);
  i2c_write(slave_address);
  delay_ms(1);
  i2c_write(inputstring[1]);
  delay_ms(1);
  i2c_write(inputstring[2]);
  delay_ms(1);
  i2c_write(inputstring[3]);
  delay_ms(1);
  i2c_stop();
}

/*I2C writes from SLAVE1 to MASTER*/
void read_slave1(byte slave_address)
{
  i2c_start();
  delay_ms(1);
  i2c_write(slave_address + 1);
  hallwheel = i2c_read(0);
  i2c_stop();
}
/*I2C writes from SLAVE2 to MASTER*/
void read_slave2(byte slave_address)
{
  i2c_start();
  delay_ms(1);
  i2c_write(slave_address + 1);
  angle_in = i2c_read(0);
  i2c_stop();
}
/*arranges incoming data char and passes information on to the SLAVEs*/
void receive_RDA()
{
  disable_interrupts(INT_RDA);
  strncpy (inputstring, buffer, 16);
  next_in = 0;
  enable_interrupts(INT_RDA);
  write_slave1(slave1_address);
  write_slave2(slave2_address);
}

/*calculation of the speed value*/
void calculation_speed()
{
  hallwheel;
  hallwheelcount = hallwheelcount + hallwheel;
  speed_out = (hallwheel * 0.0207) / 0.1495;
  buffer_speed[1] = buffer_speed[0];
  buffer_speed[0] = speed_out;
}

/*calculation of the acceleration value*/
void calculation_acceleration()
{
  acceleration_out = (buffer_speed[0] - buffer_speed[1])/0.1495;
}

/*calculation of the traveled distance*/
void calculation_distance()
{
  distance_out = hallwheelcount * 0.0207;
}

/*POWER-UP*/
void initialize()
{
  int x = 0;
  int y = 0;

  set_tris_b(0b00000111);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_128|RTCC_8_bit);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_timer_3(T3_DISABLED|T3_DIV_BY_1);

  enable_interrupts(INT_RTCC);
  enable_interrupts(INT_RDA);
  enable_interrupts(INT_EXT2);
  enable_interrupts(GLOBAL);

  for(x = 0; x<BUFFER_SIZE; x++)
  {
    inputstring[x] = 0;
  }
  for(y = 0; y<=1; y++)
  {
    buffer_speed[y] = 0;
  }
  RTCCi = 0;
  inputchar;
  angle_in = 0;
  speed_out = 0;
  acceleration_out = 0;
  distance_out = 0;
  hallwheelcount = 0;
  hallwheel = 0;
}

void main()
{
  output_high(PIN_B4);
  delay_ms(2000);
  output_low(PIN_B4);

  initialize();

  while(1)
```

```
  {
  }
}
```

D.2. Program code for Slave1

```
/*Program developed by: Martin Leonard and Ellenor Boje*/
#include <18F242.h>
#device adc=8
#use delay(clock=3276800)
#fuses NOWDT,WDT128,XT, NOPROTECT, NOOSCSEN, BROWNOUT, BORV20, PUT, STVREN,
NODEBUG, LVP, NOWRT, NOWRTD, NOWRTB, NOWRTC, NOCPD, NOCPB, NOEBTR, NOEBTRB
#use rs232(baud=19200,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
#include <stdlib.h>

unsigned char read_i2c(void);
void write_i2c(unsigned char transmit_byte);
void i2c_interrupt_handler(void);
void initialize(void);
void i2c_error(void);
void write_i2c(unsigned char transmit_byte);
void hand();
void steering_control();
void calculation_angle();
/*Byte definition for I2C registers*/
#define PIC_SSPSTAT_BIT_SMP     0x80
#define PIC_SSPSTAT_BIT_CKE     0x40
#define PIC_SSPSTAT_BIT_DA      0x20
#define PIC_SSPSTAT_BIT_P       0x10
#define PIC_SSPSTAT_BIT_S       0x08
#define PIC_SSPSTAT_BIT_RW      0x04
#define PIC_SSPSTAT_BIT_UA      0x02
#define PIC_SSPSTAT_BIT_BF      0x01
#define PIC_SSPCON1_BIT_WCOL    0x80
#define PIC_SSPCON1_BIT_SSPOV   0x40
#define PIC_SSPCON1_BIT_SSPEN   0x20
```

```
#define PIC_SSPCON1_BIT_CKP     0x10
#define PIC_SSPCON1_BIT_SSPM3   0x08
#define PIC_SSPCON1_BIT_SSPM2   0x04
#define PIC_SSPCON1_BIT_SSPM1   0x02
#define PIC_SSPCON1_BIT_SSPM0   0x01

#define PIC_SSPCON2_BIT_GCEN    0x80
#define PIC_SSPCON2_BIT_ACKSTAT 0x40
#define PIC_SSPCON2_BIT_ACKDT   0x20
#define PIC_SSPCON2_BIT_ACKEN   0x10
#define PIC_SSPCON2_BIT_RCEN    0x08
#define PIC_SSPCON2_BIT_PEN     0x04
#define PIC_SSPCON2_BIT_RSEN    0x02
#define PIC_SSPCON2_BIT_SEN     0x01

#byte PIC_SSPBUF=0xFC9
#byte PIC_SSPADD=0xFC8
#byte PIC_SSPSTAT=0xFC7
#byte PIC_SSPCON1=0xFC6
#byte PIC_SSPCON2=0xFC5

#define RX_BUF_LEN  4
#define NODE_ADDR   0x04
unsigned char slave_buffer[RX_BUF_LEN];
unsigned char steering_string[RX_BUF_LEN];
int buffer_index;
int8  hallwheel = 0;
int8  hallwheelbuffer = 0;
int8  hallgear = 0;
int8  hallgearbuffer = 0;
signed int8  angle_out = 0;
int8  angle_car = 0;
int8   anglein = 0;
int16  anglein1 = 0;
int16  anglein2 = 0;
int16  anglein3 = 0;
int1 calc_angle = 0;
int i = 0;
```

```
/*interrupt occurs every time I2C-bus is busy*/
#INT_SSP
void ssp_interupt ()
{
  hand();
}

/*timer for angle sensor calculation*/
#int_RTCC
RTCC_isr()
{
  if(i == 19)
  {
    calc_angle = 1;
  }
  else
  {
    i++;
  }
}

/*RESET*/
#INT_EXT2
EXT2_irs()
{
  while(Input(PIN_B2) == 1)
  {
    output_high(PIN_B4);
    set_pwm1_duty(82);
  }
  delay_ms(200);
  output_low(PIN_B4);
}

/*this subprogram handles the I2C-bus*/
void hand()
{
```

```
/* [0010 1101] unnecessary bits will be masked out*/
    unsigned char i2c_mask = 0x2D;
    byte temp_sspstat;
    unsigned char this_byte;
    unsigned char tx_byte;
    int x;

/*jointed register information*/
   temp_sspstat = PIC_SSPSTAT & i2c_mask;

   switch(temp_sspstat)
   {

/*write operation, last byte was an address, buffer is full*/
      case 0x09:
/*clear receive buffer*/
         for (x=0; x<RX_BUF_LEN; x++)
         {
           slave_buffer[x]=0x00;
         }
/*clear the buffer index*/
         buffer_index=0;
/*dummy read*/
         this_byte = read_i2c();
         break;

/*write operation, last byte was data, buffer is full*/
      case 0x29:
/*get the byte from the SSP*/
         this_byte = read_i2c();
/*put it into the buffer*/
         slave_buffer[buffer_index] = this_byte;
/*after 3 bytes enable steering control*/
         if (buffer_index == 2)
         {
           steering_control();
         }
         buffer_index++;

/*If index has exceeded the buffer length, buffer needs to be cleared*/
         if (buffer_index >= RX_BUF_LEN)
         {
            buffer_index = 0;
         }
         break;

/*read operation, last byte was an address, buffer is empty*/
      case 0x0c:
/*write the byte to PIC_SSPBUF*/
         write_i2c(angle_out);
         break;

/*read operation, last byte was data, buffer is empty
  not used because only one byte needs to be send!!!*/
      case 0x2c:
         break;
   }
}

/*puts value for PWM from input buffer together*/
void steering_control()
{
   strncpy (steering_string, slave_buffer, 4);

   anglein3 = ((steering_string[0] - 0x30)*100);
   anglein2 = ((steering_string[1] - 0x30)*10);
   anglein1 = (steering_string[2] - 0x30);
   anglein =  anglein3 + anglein2 + anglein1;

   set_pwm1_duty(anglein);
}

/*calculation of the output angle
  numbers in if statements indicate the digital value of the angle-sensor
  this values are greated by the analog to digital input of the 18F242
  because of the inexactly sensor output angles can only be measured within a 7 degree range
  by use of an other sensor this subroutine needs to be changed!!!*/

void calculation_angle()
{
   angle_car = read_adc();

   if(angle_car >= 52 && angle_car <= 54)
   {
      angle_out = -7;
   }
   if(angle_car >= 55 && angle_car <= 57)
   {
      angle_out = -14;
   }
   if(angle_car >= 58)
   {
      angle_out = -21;
   }
   if(angle_car <= 42)
   {
      angle_out = 21;
   }
   if(angle_car >= 43 && angle_car <= 45)
   {
      angle_out = 14;
   }
   if(angle_car >= 46 && angle_car <= 48)
   {
      angle_out = 7;
   }
   if(angle_car >= 49 && angle_car <= 51)
   {
      angle_out = 0;
   }
}

/*POWER-UP*/
```

```
void initialize(void)
{
/*Setup for the input ports*/
   set_tris_a(0b00000001);
   set_tris_b(0b00000111);
   set_tris_c(0b00011000);

   PIC_SSPCON1 = 0x36;
   PIC_SSPADD = NODE_ADDR;
/*Clear the SSPSTAT register*/
   PIC_SSPSTAT = 0x00;

   setup_adc_ports(AN0_VREF_VREF);
   setup_adc(ADC_CLOCK_INTERNAL);
   setup_ccp1(CCP_PWM);
   setup_port_a( ALL_ANALOG );
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32|RTCC_8_bit);
   setup_timer_2(T2_DIV_BY_16,255,1);
   set_adc_channel( 0 );
   set_pwm1_duty(82);

   enable_interrupts(INT_SSP);
   enable_interrupts(INT_RTCC);
   enable_interrupts(INT_EXT2);
}

/*this function returns the byte in SSPBUF*/
unsigned char read_i2c(void)
{
   return PIC_SSPBUF;
}

void write_i2c(unsigned char transmit_byte)
{
   unsigned char write_collision = 1;

/*Is BF bit set in PIC_SSPSTAT?*/
   while (PIC_SSPSTAT & PIC_SSPSTAT_BIT_BF)
```

```
  {
/*if yes? keep waiting*/
  }

/*if not? do write*/
  while (write_collision)
  {
/*clear the WCOL flag*/
    PIC_SSPCON1 &= ~PIC_SSPCON1_BIT_WCOL;
    PIC_SSPBUF = transmit_byte;
/*write collision?*/
    if(PIC_SSPCON1 & PIC_SSPCON1_BIT_WCOL)
    {
      write_collision = 1;
    }
    else
    {
      write_collision = 0;
    }
  }
/*release the clock*/
  PIC_SSPCON1 |= PIC_SSPCON1_BIT_CKP;
}

void main()
{
  output_high(PIN_B4);
  delay_ms(200);
  output_low(PIN_B4);

  initialize();
  enable_interrupts(GLOBAL);

   while (1)
   {
    if(calc_angle == 1)
    {
      calculation_angle();
    calc_angle == 0;
   }
  }
}
```

D.3. Program code for Slave2

```
/*Program developed by: Martin Leonard and Ellenor Boje*/
#include <18F242.h>
#device adc=8
#use delay(clock=3276800)
#fuses NOWDT,WDT128,XT, NOPROTECT, NOOSCSEN, BROWNOUT, BORV20, PUT, STVREN,
NODEBUG, LVP, NOWRT, NOWRTD, NOWRTB, NOWRTC, NOCPD, NOCPB, NOEBTR, NOEBTRB
#use rs232(baud=19200,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
#include <stdlib.h>

unsigned char read_i2c(void);
void write_i2c(unsigned char transmit_byte);
void i2c_interrupt_handler(void);
void initialize(void);
void i2c_error(void);
void write_i2c(unsigned char transmit_byte);
void hand();
void speed_control();
void calculation_speed();
void calculation_acceleration();
void calculation_distance();

/*Byte definition for I2C registers*/
#define PIC_SSPSTAT_BIT_SMP     0x80
#define PIC_SSPSTAT_BIT_CKE     0x40
#define PIC_SSPSTAT_BIT_DA      0x20
#define PIC_SSPSTAT_BIT_P       0x10
#define PIC_SSPSTAT_BIT_S       0x08
#define PIC_SSPSTAT_BIT_RW      0x04
#define PIC_SSPSTAT_BIT_UA      0x02
```

```
#define PIC_SSPSTAT_BIT_BF      0x01

#define PIC_SSPCON1_BIT_WCOL    0x80
#define PIC_SSPCON1_BIT_SSPOV   0x40
#define PIC_SSPCON1_BIT_SSPEN   0x20
#define PIC_SSPCON1_BIT_CKP     0x10
#define PIC_SSPCON1_BIT_SSPM3   0x08
#define PIC_SSPCON1_BIT_SSPM2   0x04
#define PIC_SSPCON1_BIT_SSPM1   0x02
#define PIC_SSPCON1_BIT_SSPM0   0x01

#define PIC_SSPCON2_BIT_GCEN    0x80
#define PIC_SSPCON2_BIT_ACKSTAT 0x40
#define PIC_SSPCON2_BIT_ACKDT   0x20
#define PIC_SSPCON2_BIT_ACKEN   0x10
#define PIC_SSPCON2_BIT_RCEN    0x08
#define PIC_SSPCON2_BIT_PEN     0x04
#define PIC_SSPCON2_BIT_RSEN    0x02
#define PIC_SSPCON2_BIT_SEN     0x01

/* Byte Reg defines*/
#byte PIC_SSPBUF=0xFC9
#byte PIC_SSPADD=0xFC8
#byte PIC_SSPSTAT=0xFC7
#byte PIC_SSPCON1=0xFC6
#byte PIC_SSPCON2=0xFC5

#define RX_BUF_LEN  5
#define NODE_ADDR   0x02

unsigned char slave_buffer[RX_BUF_LEN];
unsigned char speed_string[RX_BUF_LEN];
float buffer_speed[2];
int buffer_index;

int8  hallwheel = 0;
int8  hallwheelbuffer = 0;
int8  hallgear = 0;
```

```
int8  hallgearbuffer = 0;
int16 hallwheelcount = 0;

int8   speedin = 0;
int16  speedin1 = 0;
int16  speedin2 = 0;
int16  speedin3 = 0;

int8 i = 0;
int8 ii = 0;

/*interrupt occurs every time I2C-bus is busy*/
#INT_SSP
void ssp_interupt ()
{
  hand();
}

/*input hall-sensor 1*/
#int_ext
EXT_isr()
{
      if(Input(PIN_B0) == 1)
      {
        hallwheel++;
      }
}

/*input hall-sensor2, not used right now!!!*/
/*#int_ext1
EXT1_isr()
{
      if(Input(PIN_B1) == 1)
      {
        hallgear++;
      }
}*/
```

```
/*RESET*/
#INT_EXT2
EXT2_irs()
{
  while(Input(PIN_B2) == 1)
  {
    output_high(PIN_B4);
    set_pwm1_duty(0);
    set_pwm2_duty(0);
    hallwheel = 0;
  }
  delay_ms(200);
  output_low(PIN_B4);
}

/*this subprogram handles the I2C-bus*/
void hand()
{
/* [0010 1101] unnecessary bits will be masked out*/
   unsigned char i2c_mask = 0x2D;
   byte temp_sspstat;
   unsigned char this_byte;
   unsigned char tx_byte;
   int x;

/*jointed register information*/
   temp_sspstat = PIC_SSPSTAT & i2c_mask;

   switch(temp_sspstat)
   {

/*write operation, last byte was an address, buffer is full*/
     case 0x09:
/*clear receive buffer*/
        for (x=0; x<RX_BUF_LEN; x++)
        {
          slave_buffer[x]=0x00;
        }
```

```
/*clear the buffer index*/
        buffer_index=0;
/*dummy read*/
        this_byte = read_i2c();
        break;

/*write operation, last byte was data, buffer is full*/
     case 0x29:
/*get the byte from the SSP*/
        this_byte = read_i2c();
/*put it into the buffer*/
        slave_buffer[buffer_index] = this_byte;
/*after 3 bytes enable speed control*/
        if (buffer_index == 3)
        {
          speed_control();
        }

        buffer_index++;

/*If index has exceeded the buffer length, buffer needs to be cleared*/
        if (buffer_index >= RX_BUF_LEN)
        {
          buffer_index = 0;
        }
        break;

/*read operation, last byte was an address, buffer is empty*/
     case 0x0c:
/*write the byte to PIC_SSPBUF*/
        write_i2c(hallwheel);
        hallwheel = 0;
        break;

/*read operation, last byte was data, buffer is empty
  not used because only one byte needs to be send!!!*/
     case 0x2c:
        break;
  }
}

/*subroutine for the control of the speed controler*/
void speed_control()
{
  ii++;

  strncpy (speed_string, slave_buffer, 8);

  if(ii <= 5)
  {
  }
  else
  {
    ii=10;
    if(speed_string[0] == '0')
    {
      speedin3 = ((speed_string[1] - 0x30) * 100);
      speedin2 = ((speed_string[2] - 0x30) * 10);
      speedin1 = (speed_string[3] - 0x30);
      speedin  = speedin3 + speedin2 + speedin1;
      set_pwm1_duty(speedin);
      set_pwm2_duty(0);
    }

    if(speed_string[0] == '1')
    {
      speedin3 = ((speed_string[1] - 0x30) * 100);
      speedin2 = ((speed_string[2] - 0x30) * 10);
      speedin1 = (speed_string[3] - 0x30);
      speedin  = speedin3 + speedin2 + speedin1;
      set_pwm1_duty(0);
      set_pwm2_duty(speedin);
    }
  }
}
```

```
void initialize(void)
{
  int i = 0;
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32|RTCC_8_bit);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DIV_BY_16,255,1);
  setup_ccp1(CCP_PWM);
  setup_ccp2(CCP_PWM);
  set_pwm1_duty(0);
  set_pwm2_duty(0);
  set_tris_a(0b00000000);
  set_tris_b(0b00000111);
  set_tris_c(0b00011000);
  PIC_SSPCON1 = 0x36;
  PIC_SSPADD = NODE_ADDR;
  PIC_SSPSTAT = 0x00;
  enable_interrupts(INT_SSP);
  enable_interrupts(INT_EXT);
  enable_interrupts(INT_EXT1);
  enable_interrupts(INT_EXT2);
  enable_interrupts(INT_RTCC);
  ext_int_edge(L_TO_H);
  ext_int_edge(1, L_TO_H);
  for (i=0; i<RX_BUF_LEN; i++)
  {
    speed_string[i]=0x00;
  }

  buffer_speed[1]=0;
  buffer_speed[0]=0;
  output_high(PIN_B4);
  delay_ms(200);
  output_low(PIN_B4);
}

/*this function returns the byte in SSPBUF*/
unsigned char read_i2c(void)
{
  return PIC_SSPBUF;
}

void write_i2c(unsigned char transmit_byte)
{
  unsigned char write_collision = 1;

/*If BF bit set in PIC_SSPSTAT?*/
  while (PIC_SSPSTAT & PIC_SSPSTAT_BIT_BF)
  {
/*If yes? keep waiting*/
  }
/*If not? do write*/
  while (write_collision)
  {
/*clear the WCOL flag*/
    PIC_SSPCON1 &= ~PIC_SSPCON1_BIT_WCOL;
    PIC_SSPBUF = transmit_byte;
/*write collision?*/
    if(PIC_SSPCON1 & PIC_SSPCON1_BIT_WCOL)
    {
      write_collision = 1;
    }
    else
    {
      write_collision = 0;
    }
  }
/*release the clock*/
  PIC_SSPCON1 |= PIC_SSPCON1_BIT_CKP;
}

void main()
{
  initialize();
  enable_interrupts(GLOBAL);

  while (1)
  {
  }
}
```

D.4. Transmitter software

```
/* Authors:  Thomas Hofmann, Ellenor Boje
   Date:    2006-03-06
   Title:   C-File for each sender
*/

#include <16F628A.h>
#use delay(clock=4000000)
#fuses NOWDT,INTRC, PUT, NOPROTECT

#Define N_OF_SYNCS  3     // Number of Infra Sync Signals
#Define SYNC_PERIOD 2000  // in us. Time for one infra high and low at sync ONLY
#Define LOW_TIME    140   // in ms. Time low after every signal out.
#Define HIGH_TIME   10    // in ms. Time Infra and Ultra is transmitting

int8 i, sender = 0, count_3=0, count_2=0, count_1=0;

void send()
{
  output_high(PIN_B0);       // PIN6 = LED
  output_high(PIN_B1);       // PIN7 = INFRA
  output_high(PIN_B2);       // PIN8 = ULTRA
  delay_ms(HIGH_TIME);
  output_low(PIN_B0);
  output_low(PIN_B1);
  output_low(PIN_B2);
  delay_ms(LOW_TIME);
  set_tris_b(0b11000000);
  disable_interrupts(INT_RB);
  output_high(PIN_B5);
  delay_us(100);
  output_low(PIN_B5);
  set_tris_b(0b11100000);
  enable_interrupts(INT_RB);
}

void sync_send()
{
  output_high(PIN_B0);            // PIN6 = LED
  output_high(PIN_B2);            // PIN8 = ULTRA
  for(i = 1; i<=N_OF_SYNCS; i++)
    {
      output_high(PIN_B1);       // PIN7 = INFRA
      delay_us(SYNC_PERIOD/2);
      output_low(PIN_B1);        // PIN7 = INFRA
      delay_us(SYNC_PERIOD/2);
    }
  output_low(PIN_B2);             // PIN8 = ULTRA
  output_low(PIN_B0);             // PIN6 = LED
  delay_ms(LOW_TIME);
  set_tris_b(0b11000000);
  disable_interrupts(INT_RB);
  output_high(PIN_B5);
  delay_us(100);
  output_low(PIN_B5);
  set_tris_b(0b11100000);
  enable_interrupts(INT_RB);
}

void init()
{
  output_low(PIN_B0);
  output_low(PIN_B2);
  output_low(PIN_B3);
  output_low(PIN_B5);


  set_tris_a(0b00000011);
  set_tris_b(0b11100000);

  enable_interrupts(INT_RB);
  enable_interrupts(GLOBAL);

  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);

  If(input(PIN_A0) == 0 && input(PIN_A1) == 0 && input(PIN_B7) == 0
    && input(PIN_B6) == 0)
  {
    sender = 1;
  }

  else If(input(PIN_A0) == 0 && input(PIN_A1) == 0 && input(PIN_B7) == 0
       && input(PIN_B6) == 1)
  {
    sender = 2;
  }

  else If(input(PIN_A0) == 0 && input(PIN_A1) == 0 && input(PIN_B7) == 1
       && input(PIN_B6) == 1)
  {
    sender = 3;
  }

  else
    output_high(PIN_B3);
}

#int_RB
RB_isr()
{
  if(input(PIN_B5) == 1)
  {
    if(sender == 1)
    {
      count_1++;
      if(count_1 == 2)
      {
        sync_send();
        count_1 = 0;
      }
    }
    if(sender == 2)
    {
      count_2++;
      if(count_2 == 1)
        send();
      if(count_2 == 2)
        count_2 = 0;
    }
    if(sender == 3)
    {
      count_3++;
      if(count_3 == 2)
      {
        send();
        count_3 = 0;
      }

    }
  }

}

void main()
{
  init();

  if(sender == 1)
    sync_send();

  while(1)
  {}

}
```

D.5. Receiver software

```
/* Authors:  Thomas Hofmann, Ellenor Boje
  Date:    2006-02-28
  Title:   Receiver Software with LCD output (final version)
*/

#include <18F242.h>

#use delay(clock=20000000)
```

```
#fuses NOWDT,WDT128,HS,NOPROTECT,NOOSCSEN,NOBROWNOUT,BORV20,PUT, OSTVREN,
    NODEBUG,NOLVP,NOWRT,NOWRTD,NOWRTB,NOWRTC,NOCPD,NOCPB,NOEBTR,NOEBTRB

#define LED_I     PIN_A1
#define LED_U1    PIN_B2
#define LED_U2    PIN_B3
#define LED_Init  PIN_B0
#define LED_Err   PIN_A3

#byte PORTB = 0xF81
#byte PORTC = 0xF82

#Define CLK_TIME     3.2      // * E-6
#Define CLK_TIME1    0.8      // * E-6
#Define SONIC_SPEED  347.5    // At about 26C, differs strong!!!
#Define HEIGHT_1     275      // in cm
#Define HEIGHT_2     275      // in cm
#Define HEIGHT_3     268

#Define Transm1_X    3        // in cm  Transm_1 & Transm_2 should be the ones
#Define Transm1_Y    0        // in cm  which have the biggest difference
#Define Transm2_X    0        // in cm  of their Y-Coordinates because of my
#Define Transm2_Y    208      // in cm  positioning algorithm
#Define Transm3_X    176      // in cm
#Define Transm3_Y    100      // in cm

#Define Offset       0        // in us
#Define THRESH_US1   351      // in mV for 2 threshold compensation
#Define THRESH_US2   733      // in mV for 2 threshold compensation

#define TOLERANCE    250000   /*this is the SQARE of the max. distance (in cm)
                     of the two-circle intersection solution point to
                     the third circle*/

#Define DIST_P       150000   // in us. Time Period for one Distance Calc
                     // (1 IR, 2 US Signals)
#Define T_IR         5000     // in us. Time Interval IR Signal can differ from
                     // where it is supposed to be
#Define T_US         1000     // in us. Time Interval US Signal can differ from
                     // where it is supposed to be
#Define MAXSPEED_CAR 1        // in m/s
#Define FAREST_DIST  1000     // in cm. Farest Dist from a beacon to the car

#Define SYNC_TIME    6000     // in us. Time for infra sync (4 infra highs)
#Define SYNC_PERIOD  2000     // in us. Time for one infra high and low at sync
#Define N_OF_SYNCS   3        // Number of Infra Sync Signals

#define set_tris_lcd(x) set_tris_c(x)
#define lcd_type 2            // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40     // LCD RAM address for the second line


#Bit     INFRA = 0xF81.7
#Bit     ULTRA = 0xF81.6
#Bit     ULTRA2 = 0xF81.1
```

```
/*-----------------------------------------------------------------------------
Variable definitions
-----------------------------------------------------------------------------*/

int1  pos_cal;
int8  pos_period=0, test=0;
int16 us_diff_time=0, d=0, d1, d2, d3;
int32 time=0, time2;

int16 d_straight = 1, d3_straight, d2_straight, d1_straight;

int8  initial=0, sync_count=0;
int32 ir_control_time, last_sync=0, present_sync = 0;
int16 us_max, us_min, dist_old_new;

struct position
{
   int32 x;
   int32 y;
}p, p_old;

struct position pos;
float A, B, xr1, xr2, yr1, yr2, sq_d, delta, temp1, temp2, t1, t2, diff,u, v, w;

/*-----------------------------------------------------------------------------
LCD Functions
-----------------------------------------------------------------------------*/

BYTE const LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 0xf82};
                          // These bytes need to be sent to the LCD
                          // to start it up.
                          // The following are used for setting
                          // the I/O port direction register.
struct lcd_pin_map
{                 // This structure is overlayed
          BOOLEAN rs;           // on to an I/O port to gain
          BOOLEAN rw;               // access to the LCD pins.
          BOOLEAN unused;               // The bits are allocated from
          BOOLEAN enable;           // low order up.  rs will
          int     data : 4;         // be pin B0.
} lcd;

struct lcd_pin_map const LCD_WRITE = {0,0,0,0,0}; // For write mode all pins are out
struct lcd_pin_map const LCD_READ = {0,0,0,0,0x0f}; // For read mode

#byte lcd = 0xF82                  // on to port C

int16 hundreds,tens,ones,hundreds1,tens1,ones1,value,value1,
      hundreds2,tens2,ones2,value2;

int16 hundreds11,tens11,ones11, hundreds22,tens22,ones22,
      hundreds0,tens0,ones0, value11, value22, value0;

int16 hundreds_x,tens_x,ones_x, hundreds_y,tens_y,ones_y, coord_x, coord_y;

BYTE lcd_read_byte()
```

```
{
     BYTE low,high;
     set_tris_lcd(LCD_READ);
     lcd.rw = 1;
     delay_cycles(1);
     lcd.enable = 1;
     delay_cycles(1);
     high = lcd.data;
     lcd.enable = 0;
     delay_cycles(1);
     lcd.enable = 1;
     delay_us(1);
     low = lcd.data;
     lcd.enable = 0;
     set_tris_lcd(LCD_WRITE);
     return( (high<<4) | low);
}

void lcd_send_nibble( BYTE n )
{
     lcd.data = n;
     delay_cycles(1);
     lcd.enable = 1;
     delay_us(2);
     lcd.enable = 0;
}

void lcd_send_byte( BYTE address, BYTE n )
{
     lcd.rs = 0;
     while ( bit_test(lcd_read_byte(),7) ) ;
     lcd.rs = address;
     delay_cycles(1);
     lcd.rw = 0;
     delay_cycles(1);
     lcd.enable = 0;
     lcd_send_nibble(n >> 4);
     lcd_send_nibble(n & 0x0f);
}

void lcd_init()
{
   BYTE i;
   set_tris_lcd(LCD_WRITE);
   lcd.rs = 0;
   lcd.rw = 0;
   lcd.enable = 0;
   delay_ms(15);
   for(i=1;i<=3;++i) {
      lcd_send_nibble(3);
      delay_ms(5);
   }
   lcd_send_nibble(2);
   for(i=0;i<=3;++i)
      lcd_send_byte(0,LCD_INIT_STRING[i]);
}
```

```
void lcd_gotoxy( BYTE x, BYTE y)
{
   BYTE address;

   if(y!=1)
     address=lcd_line_two;
   else
     address=0;
   address+=x-1;
   lcd_send_byte(0,0x80|address);
}

void lcd_putc( char c)
{
   switch (c)
   {
     case '\f'   : lcd_send_byte(0,1);
                   delay_ms(2);
                                          break;
     case '\n'   : lcd_gotoxy(1,2);        break;
     case '\b'   : lcd_send_byte(0,0x10);  break;
     default     : lcd_send_byte(1,c);     break;
   }
}

char lcd_getc( BYTE x, BYTE y)
{
   char value;

    lcd_gotoxy(x,y);
    while ( bit_test(lcd_read_byte(),7) ); // wait until busy flag is low
    lcd.rs=1;
    value = lcd_read_byte();
    lcd.rs=0;
    return(value);
}

void Convert_LCD()
{
 if (value != 0)
 {
  hundreds = value/100;
  tens = ((value-(hundreds*100))/10);
  ones = (value-(hundreds*100)-(tens*10));
  if (ones>9)
   ones = 9;
  ones = ones + 0x30;
  tens = tens + 0x30;
  hundreds = hundreds + 0x30;
 }
 if (value1 != 0)
 {
  hundreds1 = value1/100;
  tens1 = ((value1-(hundreds1*100))/10);
  ones1 = (value1-(hundreds1*100)-(tens1*10));
  if (ones1>9)
   ones1 = 9;
  ones1 = ones1 + 0x30;
  tens1 = tens1 + 0x30;
  hundreds1 = hundreds1 + 0x30;
 }
  if (value2 != 0)
 {
  hundreds2 = value2/100;
  tens2 = ((value2-(hundreds2*100))/10);
  ones2 = (value2-(hundreds2*100)-(tens2*10));
  if (ones2>9)
   ones2 = 9;
  ones2 = ones2 + 0x30;
  tens2 = tens2 + 0x30;
  hundreds2 = hundreds2 + 0x30;
 }
  if (value0 != 0)
 {
  hundreds0 = value0/100;
  tens0 = ((value0-(hundreds0*100))/10);
  ones0 = (value0-(hundreds0*100)-(tens0*10));
  if (ones0>9)
   ones0 = 9;
  ones0 = ones0 + 0x30;
  tens0 = tens0 + 0x30;
  hundreds0 = hundreds0 + 0x30;
 }
 if (value11 != 0)
 {
  hundreds11 = value11/100;
  tens11 = ((value11-(hundreds11*100))/10);
  ones11 = (value11-(hundreds11*100)-(tens11*10));
  if (ones11>9)
   ones11 = 9;
  ones11 = ones11 + 0x30;
  tens11 = tens11 + 0x30;
  hundreds11 = hundreds11 + 0x30;
 }
  if (value22 != 0)
 {
  hundreds22 = value22/100;
  tens22 = ((value22-(hundreds22*100))/10);
  ones22 = (value22-(hundreds22*100)-(tens22*10));
  if (ones22>9)
   ones22 = 9;
  ones22 = ones22 + 0x30;
  tens22 = tens22 + 0x30;
  hundreds22 = hundreds22 + 0x30;
 }
  if (coord_x != 0)
 {
  hundreds_x = coord_x/100;
  tens_x = ((coord_x-(hundreds_x*100))/10);
  ones_x = (coord_x-(hundreds_x*100)-(tens_x*10));
  if (ones_x>9)
```

```
   ones_x = 9;
  ones_x = ones_x + 0x30;
  tens_x = tens_x + 0x30;
  hundreds_x = hundreds_x + 0x30;
 }
  if (coord_y != 0)
 {
  hundreds_y = coord_y/100;
  tens_y = ((coord_y-(hundreds_y*100))/10);
  ones_y = (coord_y-(hundreds_y*100)-(tens_y*10));
  if (ones_y>9)
   ones_y = 9;
  ones_y = ones_y + 0x30;
  tens_y = tens_y + 0x30;
  hundreds_y = hundreds_y + 0x30;
 }
}
/*---------------------------------------------------------------------------
 Functions
---------------------------------------------------------------------------*/

void init()
{
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   setup_wdt(WDT_OFF);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_16);  //For a range of 18m DIV_4 used 62
   setup_timer_1(T1_INTERNAL|T1_DIV_BY_4);
   setup_timer_2(T2_DISABLED,0,1);
   setup_timer_3(T3_DISABLED|T3_DIV_BY_8);

   set_tris_a(0);              // Port A is Output
   set_tris_b(0xE2);            // Pin 1, 6 and 7 from Port B are Input
   set_tris_c(0);

   ext_int_edge( 1, L_TO_H);
 //enable_interrupts(INT_EXT1);  //2 Threshold compensation currently not in use
   enable_interrupts(INT_RB);
   enable_interrupts(GLOBAL);

   output_low(LED_Err);
   output_low(LED_U1);
   output_low(LED_U2);
   output_low(LED_I);
   output_low(LED_Init);

   us_max = (FAREST_DIST*10000)/SONIC_SPEED + T_US;
   us_min = (HEIGHT_1*10000)/SONIC_SPEED - T_US;
   dist_old_new = MAXSPEED_CAR * DIST_P * 3 / 1000;      //Max dist change
                                             //after one pos_calc
}

float sqr(float x)
{
   return x*x;
}
```

```
float sqrt(float x)                    //square-root-function
{
             int idx;
   float q,y;
   q=0;
   y=2;
   for(idx=0;idx<14;idx++)    //enough precision and prevent
                                                                    //program from getting lost in loop
   {
       q=x/y;
       y=(y+q)/2;
   }
   return y;
}


struct position pos_calc( int16 d1,  int16 d2,  int16 d3)
{
   diff=(Transm2_Y - Transm1_Y);             //Cutting points of 2 circles
   B =(Transm2_X - Transm1_X) / (diff);
   u=1+(sqr(B));
   A=(sqr(d1)-sqr(d2)+sqr(Transm2_X)-sqr(Transm1_X))/(diff*2)+
     (Transm2_Y+Transm1_Y)/2;
   v=2*(Transm1_Y*B-Transm1_X-A*B);
   w=(-sqr(d1)+sqr(Transm1_X)+sqr(Transm1_Y)+sqr(A)-2*Transm1_Y*A);
   delta=(-4*u*w+sqr(v));
   if(delta<0)                            //Error if no cutting point
   {
     output_high(LED_Err);
     delay_ms(5);
     output_low(LED_Err);
   }
   else
   {
                       sq_d=sqrt(delta);
                       xr1=(sq_d-v)/u/2;
                       xr2=-(v+sq_d)/u/2;
                       yr1=(A-B*xr1);
                       yr2=(A-B*xr2);
                       t1=abs(sqr(Transm3_X-xr1)+sqr(Transm3_Y-yr1)-sqr(d3));   //test 3. circle
     t2=abs(sqr(Transm3_X-xr2)+sqr(Transm3_Y-yr2)-sqr(d3));

                       if (t2<t1)
                       {          //exchange the solution, xr1/yr1 will point the true solution
       temp2=xr1;
                                 xr1=xr2;
                                 xr2=temp2;
                                 temp2=yr1;
                                 yr1=yr2;
                                 yr2=temp2;
       temp2=t2;
                                 t2=t1;
                                 t1=temp2;

                       }
```

```
                    if (t1>TOLERANCE)        //Checking if the cutting point of 3 circles
                      //is in the tolerance
      {
        output_high(LED_Err);
        delay_ms(5);
        output_low(LED_Err);
      }

            }
    pos.x = xr1;
    pos.y = yr1;
    return pos;
}

int16 dist_calc(int32 time, int16 us_diff_time)
{
    int32 time_c;
    int16 d_c;

    d_straight = (time * SONIC_SPEED)/10000;
    /*time_c= us_diff_time * (THRESH_US1 + time * (THRESH_US1 - THRESH_US2)
     / us_diff_time) / (THRESH_US1 - THRESH_US2);
    2 Threshold compensation currently not in use*/

    switch (pos_period)
    {
      case 0:  d_c=sqrt(sqr((time*SONIC_SPEED)/1000) - sqr (HEIGHT_1 * 10))/10;
             break;
      case 1:  d_c=sqrt(sqr((time*SONIC_SPEED)/1000) - sqr (HEIGHT_2 * 10))/10;
             break;
      case 2:  d_c=sqrt(sqr((time*SONIC_SPEED)/1000) - sqr (HEIGHT_3 * 10))/10;
             break;
     default:  output_high(LED_Err); break;
    }
    return d_c;    // dist in cm
}

void dist_pos()
{

    us_diff_time = time2 - time;
    d = dist_calc(time, us_diff_time);

    switch (pos_period)
    {
      case 0:  d1=d;
             d1_straight = d_straight; break;
      case 1:  d2=d;
             d2_straight = d_straight; break;
      case 2:  d3=d;
             d3_straight = d_straight; break;
     default: output_high(LED_Err); break;
    }
    pos_period++;
```

```
  if(pos_period == 3)
  {
    p_old = p;
    p = pos_calc(d1, d2, d3);
    pos_period = 0;
    /*if( abs(p_old.x-p.x) > (dist_old_new + dist_old_new/5) ||
    abs(p_old.y-p.y) > (dist_old_new + dist_old_new/5))

    // If (pos change in reality > Biggest possible pos change + 20%) => Error
    {
      p = p_old;
      output_high(LED_Err);
      delay_ms(8);
      output_low(LED_Err);
    }*/

  }
  test = 0;

  value = d1;               // for LCD output
  value1 = d2;
  value2 = d3;
  value0 = d1_straight;
  value11 = d2_straight;
  value22 = d3_straight;
  coord_x = p.x;
  coord_y = p.y;
}

void i_ir()
{

  set_timer0(0);
  test = 1;

              // Check for 1 because of no low_to_high detection
              // The infra routine has to be faster then 7.3ms
              // because thats the time the sound needs for 2.5 metres
}              // which is the shortest distance


void u_ir()
{
  time =get_timer0() * CLK_TIME;   // time_in_sec = time * E-6

  dist_pos();
  output_high(LED_U1);
  delay_ms(10);
  output_low(LED_U1);
}

void sync()    //Synchronisation to Sender 1 at start and in case of error
{
  if(sync_count == 0)
    {
      sync_count = 1;
      set_timer0(0);
    }
    if(sync_count > 0)
    {
      present_sync = get_timer0() * CLK_TIME;
      if(present_sync < SYNC_TIME)
      {
        if(present_sync - last_sync > SYNC_PERIOD-SYNC_PERIOD/5)
        {
          sync_count++;
          last_sync = get_timer0() * CLK_TIME;
          if(sync_count >= N_OF_SYNCS)
          {
            test = 1;
            initial = 1;
            output_high(LED_Init);
            delay_ms(3);
            output_low(LED_Init);

          }
        }
      }
      else
        sync_count = 0;
    }
}
/*-----------------------------------------------------------------------------
Interrupt SR
-----------------------------------------------------------------------------*/

#int_RB
portb_int()
{
  if(initial == 1)
    ir_control_time = get_timer0() * CLK_TIME;  // time_in_sec = ir_time * E-6

  if(INFRA == 1)
  {

    if(initial == 1 && test == 0 &&
      (ir_control_time + T_IR > DIST_P && ir_control_time - T_IR < DIST_P))
                                    // Conditions for proper IR
      i_ir();

    if(initial == 0)
    {
      sync();
    }

  }
  if(initial == 1 && test == 1 && ULTRA == 1 &&
    (ir_control_time < us_max && ir_control_time > us_min))
    u_ir();
```

```
 #asm               //That PIC just gets interrupt when rising or falling edge
 movf 0xf81,0        // at port B
 #endasm
}

// NOT IN USE YET (2 THRESHOLD COMPENSATION)

#int_EXT1
EXT1_isr()
{
   ir_control_time = get_timer0() * CLK_TIME;
   if(test == 2 && (ir_control_time < us_max && ir_control_time > us_min))
   {
                              output_high(LED_U2);
                              delay_us(8);
                              output_low(LED_U2);
      time2 =(get_timer0() * CLK_TIME);   // time_in_sec = time * E-6
      set_timer1(0);
      test = 3;
      dist_pos();
      output_high(LED_I);
      delay_ms(10);
      output_low(LED_I);
   }

}

void main()
{
   init();

   lcd_init();

   while(1)
   {

   if((get_timer0() * CLK_TIME) > (DIST_P+T_IR)) //Error Condition for new sync
      initial = 0;

   Convert_Lcd();

   lcd_gotoxy(1,1);       //LCD Output routines
   lcd_putc(hundreds);
   lcd_putc(tens);
   lcd_putc(ones);

   lcd_gotoxy(5,1);
   lcd_putc(hundreds1);
   lcd_putc(tens1);
   lcd_putc(ones1);

   lcd_gotoxy(9,1);
   lcd_putc(hundreds2);
   lcd_putc(tens2);
   lcd_putc(ones2);
```

```
  lcd_gotoxy(1,2);
  lcd_putc(hundreds0);
  lcd_putc(tens0);
  lcd_putc(ones0);

  lcd_gotoxy(5,2);
  lcd_putc(hundreds11);
  lcd_putc(tens11);
  lcd_putc(ones11);

  lcd_gotoxy(9,2);
  lcd_putc(hundreds22);
  lcd_putc(tens22);
  lcd_putc(ones22);

  lcd_gotoxy(13,1);
  lcd_putc(hundreds_x);
  lcd_putc(tens_x);
  lcd_putc(ones_x);

  lcd_gotoxy(13,2);
  lcd_putc(hundreds_y);
  lcd_putc(tens_y);
  lcd_putc(ones_y);
  };
}
```

D.6. IR Object detection front and back

```
/***********************************************************************/
/* Written by Ellenor Boje                                   */
/* Date:    2006-06-15                                      */
/* Title:   C-File for each IR front, frontleft, frontright and back    */
/*                                                          */
/***********************************************************************/

#include <16F877A.H>
#DEVICE ADC=10
#device ICD=TRUE

#FUSES  XT,NOWDT,NOPROTECT,PUT,NOLVP,BROWNOUT
#ID     0x00

#use Delay(Clock=4000000)
/***********************************************************************/
/*                            Declarations                                          */
/***********************************************************************/
#byte   PORTA  =  5
#byte   PORTB  =  6
#byte   PORTD  =  8
byte       X_pos = 1;

#define ALL_OUT 0
#define ALL_IN  0xff
#define set_tris_lcd(x) set_tris_b(x)
```

```
#define lcd_type 2           // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40    // LCD RAM address for the second line
BYTE const LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
                         // These bytes need to be sent to the LCD
                         // to start it up.
                         // The following are used for setting
                         // the I/O port direction register.
struct lcd_pin_map
{                // This structure is overlayed
         BOOLEAN rs;           // on to an I/O port to gain
         BOOLEAN rw;               // access to the LCD pins.
         BOOLEAN unused;               // The bits are allocated from
         BOOLEAN enable;          // low order up.  rs will
         int     data : 4;         // be pin B0.
} lcd;

struct lcd_pin_map const LCD_WRITE = {0,0,0,0,0}; // For write mode all pins are out
struct lcd_pin_map const LCD_READ = {0,0,0,0,0x0f}; // For read mode

#byte lcd = 6                  // on to port B (at address 6)
byte ene, tiene, afstandF, afstandB;

BYTE lcd_read_byte()
{
     BYTE low,high;
     set_tris_lcd(LCD_READ);
     lcd.rw = 1;
     delay_cycles(1);
     lcd.enable = 1;
     delay_cycles(1);
     high = lcd.data;
     lcd.enable = 0;
     delay_cycles(1);
     lcd.enable = 1;
     delay_us(1);
     low = lcd.data;
     lcd.enable = 0;
     set_tris_lcd(LCD_WRITE);
     return( (high<<4) | low);
}


void lcd_send_nibble( BYTE n )
{
     lcd.data = n;
     delay_cycles(1);
     lcd.enable = 1;
     delay_us(2);
     lcd.enable = 0;
}


void lcd_send_byte( BYTE address, BYTE n )
{
     lcd.rs = 0;
     while ( bit_test(lcd_read_byte(),7) ) ;
```

```
      lcd.rs = address;
      delay_cycles(1);
      lcd.rw = 0;
      delay_cycles(1);
      lcd.enable = 0;
      lcd_send_nibble(n >> 4);
      lcd_send_nibble(n & 0x0f);
}

void lcd_init()
{
    BYTE i;
    set_tris_lcd(LCD_WRITE);
    lcd.rs = 0;
    lcd.rw = 0;
    lcd.enable = 0;
    delay_ms(15);
    for(i=1;i<=3;++i) {
       lcd_send_nibble(3);
       delay_ms(5);
    }
    lcd_send_nibble(2);
    for(i=0;i<=3;++i)
       lcd_send_byte(0,LCD_INIT_STRING[i]);
}

void lcd_gotoxy( BYTE x, BYTE y)
{
   BYTE address;

   if(y!=1)
     address=lcd_line_two;
   else
     address=0;
   address+=x-1;
   lcd_send_byte(0,0x80|address);
}

void lcd_putc( char c)
{
   switch (c)
   {
     case '\f'   : lcd_send_byte(0,1);
                   delay_ms(2);
                                      break;
     case '\n'   : lcd_gotoxy(1,2);        break;
     case '\b'   : lcd_send_byte(0,0x10);  break;
     default     : lcd_send_byte(1,c);     break;
   }
}

char lcd_getc( BYTE x, BYTE y)
{
   char value;

    lcd_gotoxy(x,y);
    while ( bit_test(lcd_read_byte(),7) ); // wait until busy flag is low
    lcd.rs=1;
    value = lcd_read_byte();
    lcd.rs=0;
    return(value);
}

void Convert_LCD(byte temp)         //convert analog input to data for display
{

 if (temp == 0)
 {
  tiene = 'E';
  ene = 'r';
 }
 else if (temp == 255)
 {
  tiene = 'E';
  ene = 'R';
 }
 else
 {
  tiene = temp/10;
  ene = (temp-(tiene*10));
  if (ene>9)
   ene = 9;
  ene = ene + 0x30;
  tiene = tiene + 0x30;
 }

}

byte Convert_Distance(long result)    //10 bit value are set equal to a distance
{
if(result>467) return(        0           );
if(result<          468       && result>        431       ) return( 11         );
if(result<          432       && result>        402       ) return( 12         );
if(result<          403       && result>        376       ) return( 13         );
if(result<          377       && result>        354       ) return( 14         );
if(result<          355       && result>        337       ) return( 15         );
if(result<          338       && result>        318       ) return( 16         );
if(result<          319       && result>        303       ) return( 17         );
if(result<          304       && result>        287       ) return( 18         );
if(result<          288       && result>        276       ) return( 19         );
if(result<          277       && result>        261       ) return( 20         );
if(result<          262       && result>        253       ) return( 21         );
if(result<          254       && result>        241       ) return( 22         );
if(result<          242       && result>        234       ) return( 23         );
if(result<          235       && result>        227       ) return( 24         );
if(result<          228       && result>        219       ) return( 25         );
if(result<          220       && result>        215       ) return( 26         );
if(result<          216       && result>        205       ) return( 27         );
if(result<          206       && result>        198       ) return( 28         );
if(result<          199       && result>        185       ) return( 29         );
if(result<          186       && result>        188       ) return( 30         );
if(result<          189       && result>        184       ) return( 31         );
if(result<          185       && result>         180       ) return( 32          );
if(result<          181       && result>         176       ) return( 33          );
if(result<          177       && result>         172       ) return( 34          );
if(result<          173       && result>         169       ) return( 35          );
if(result<          170       && result>         164       ) return( 36          );
if(result<          165       && result>         160       ) return( 37          );
if(result<          161       && result>         156       ) return( 38          );
if(result<          157       && result>         152       ) return( 39          );
if(result<          153       && result>         150       ) return( 40          );
if(result<          151       && result>         147       ) return( 41          );
if(result<          148       && result>         144       ) return( 42          );
if(result<          145       && result>         141       ) return( 43          );
if(result<          142       && result>         140       ) return( 44          );
if(result<          141       && result>         136       ) return( 45          );
if(result<          137       && result>         135       ) return( 46          );
if(result<          136       && result>         134       ) return( 47          );
if(result<          135       && result>         133       ) return( 48          );
if(result<          134       && result>         114       ) return( 49          );
if(result<          115       && result>         122       ) return( 50          );
if(result<          123       && result>         121       ) return( 51          );
if(result<          122       && result>         120       ) return( 52          );
if(result<          121       && result>         118       ) return( 53          );
if(result<          119       && result>         117       ) return( 54          );
if(result<          118       && result>         116       ) return( 55          );
if(result<          117       && result>         114       ) return( 56          );
if(result<          115       && result>         113       ) return( 57          );
if(result<          114       && result>         109       ) return( 58          );
if(result<          110       && result>         106       ) return( 59          );
if(result<          107       && result>         104       ) return( 60          );
if(result<          103) return(255);

}

void Stop_Go(long result)     //if closer than 20cm AGV stop
{
   if(result>261)
      output_high(PIN_B2);
   else
      output_low(PIN_B2);
}

void main()
{
 long front,frontleft,frontright,back;
 setup_adc_ports(ALL_ANALOG);

// set_tris_d(0xF0);
 lcd_init();
 while (1)
 {

  set_adc_channel(0);
  delay_ms(40);
  front = read_adc();
  Convert_LCD(Convert_Distance(front));
```

```
lcd_gotoxy(1,1);
lcd_putc('F');
lcd_putc(':');
lcd_putc(' ');
lcd_putc(tiene);
lcd_putc(ene);
lcd_putc('c');
lcd_putc('m');

Stop_Go(front);
delay_ms(500);

set_adc_channel(1);
delay_ms(40);
frontleft = read_adc();
Convert_LCD(Convert_Distance(frontleft));

lcd_gotoxy(1,2);
lcd_putc('F');
lcd_putc('L');
lcd_putc(':');
lcd_putc(' ');
lcd_putc(tiene);
lcd_putc(ene);
lcd_putc('c');
lcd_putc('m');

Stop_Go(frontleft);
delay_ms(500);

set_adc_channel(2);
delay_ms(40);
frontright = read_adc();
Convert_LCD(Convert_Distance(frontright));

lcd_gotoxy(9,1);
lcd_putc('F');
lcd_putc('R');
lcd_putc(':');
lcd_putc(' ');
lcd_putc(tiene);
lcd_putc(ene);
lcd_putc('c');
lcd_putc('m');

Stop_Go(frontright);
delay_ms(500);

set_adc_channel(3);
delay_ms(40);
back = read_adc();
Convert_LCD(Convert_Distance(back));

lcd_gotoxy(9,2);
lcd_putc('B');
lcd_putc(':');
```

```
 lcd_putc(' ');
 lcd_putc(tiene);
 lcd_putc(ene);
 lcd_putc('c');
 lcd_putc('m');

 Stop_Go(back);
 delay_ms(500);

 }
}
```

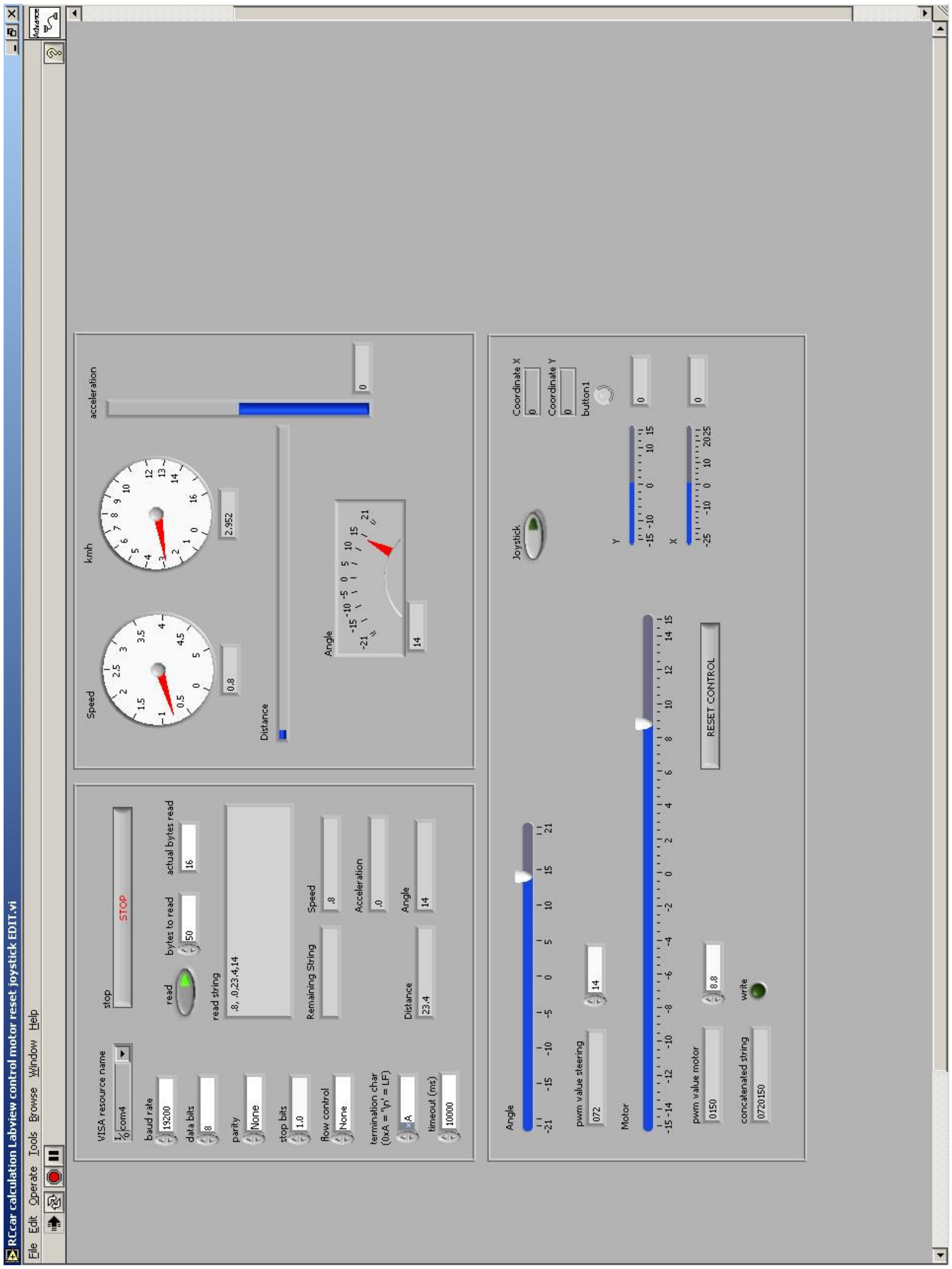
D.7. IR Object detection left and right

```
/**********************************************************************/
/* Written by Ellenor Boje                                  */
/* Date:    2006-07-26                                      */
/* Title:   C-File for each IR left1, left2, right1 and right2          */
/*                                                          */
/**********************************************************************/

#include <16F877A.H>
#DEVICE ADC=10
#device ICD=TRUE

#FUSES  XT,NOWDT,NOPROTECT,PUT,NOLVP,BROWNOUT
#ID     0x00

#use Delay(Clock=4000000)
/**********************************************************************/
/*                              Declarations                          */
/**********************************************************************/
#byte   PORTA  =  5
#byte   PORTB  =  6
#byte   PORTD  =  8
byte       X_pos = 1;

#define ALL_OUT 0
#define ALL_IN  0xff
#define set_tris_lcd(x) set_tris_b(x)
#define lcd_type 2           // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40    // LCD RAM address for the second line
BYTE const LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
                        // These bytes need to be sent to the LCD
                        // to start it up.
                        // The following are used for setting
                        // the I/O port direction register.
struct lcd_pin_map
{               // This structure is overlayed
        BOOLEAN rs;           // on to an I/O port to gain
        BOOLEAN rw;              // access to the LCD pins.
        BOOLEAN unused;              // The bits are allocated from
        BOOLEAN enable;          // low order up.  rs will
        int     data : 4;        // be pin B0.
} lcd;
```

```
struct lcd_pin_map const LCD_WRITE = {0,0,0,0,0}; // For write mode all pins are out
struct lcd_pin_map const LCD_READ = {0,0,0,0,0x0f}; // For read mode

#byte lcd = 6                  // on to port B (at address 6)
byte ene, tiene, afstandF, afstandB;

BYTE lcd_read_byte()
{
     BYTE low,high;
     set_tris_lcd(LCD_READ);
     lcd.rw = 1;
     delay_cycles(1);
     lcd.enable = 1;
     delay_cycles(1);
     high = lcd.data;
     lcd.enable = 0;
     delay_cycles(1);
     lcd.enable = 1;
     delay_us(1);
     low = lcd.data;
     lcd.enable = 0;
     set_tris_lcd(LCD_WRITE);
     return( (high<<4) | low);
}


void lcd_send_nibble( BYTE n )
{
     lcd.data = n;
     delay_cycles(1);
     lcd.enable = 1;
     delay_us(2);
     lcd.enable = 0;
}


void lcd_send_byte( BYTE address, BYTE n )
{
     lcd.rs = 0;
     while ( bit_test(lcd_read_byte(),7) ) ;
     lcd.rs = address;
     delay_cycles(1);
     lcd.rw = 0;
     delay_cycles(1);
     lcd.enable = 0;
     lcd_send_nibble(n >> 4);
     lcd_send_nibble(n & 0x0f);
}

void lcd_init()
{
   BYTE i;
   set_tris_lcd(LCD_WRITE);
   lcd.rs = 0;
   lcd.rw = 0;
   lcd.enable = 0;
```

```
    delay_ms(15);
    for(i=1;i<=3;++i) {
       lcd_send_nibble(3);
       delay_ms(5);
    }
    lcd_send_nibble(2);
    for(i=0;i<=3;++i)
       lcd_send_byte(0,LCD_INIT_STRING[i]);
}

void lcd_gotoxy( BYTE x, BYTE y)
{
   BYTE address;

   if(y!=1)
     address=lcd_line_two;
   else
     address=0;
   address+=x-1;
   lcd_send_byte(0,0x80|address);
}

void lcd_putc( char c)
{
   switch (c)
   {
     case '\f'   : lcd_send_byte(0,1);
                  delay_ms(2);
                                    break;
     case '\n'   : lcd_gotoxy(1,2);        break;
     case '\b'   : lcd_send_byte(0,0x10);  break;
     default     : lcd_send_byte(1,c);     break;
   }
}

char lcd_getc( BYTE x, BYTE y)
{
   char value;

    lcd_gotoxy(x,y);
    while ( bit_test(lcd_read_byte(),7) ); // wait until busy flag is low
    lcd.rs=1;
    value = lcd_read_byte();
    lcd.rs=0;
    return(value);
}

void Convert_LCD(byte temp)         //convert analog input to data for display
{

 if (temp == 0)
 {
  tiene = 'E';
  ene = 'r';
 }
 else if (temp == 255)
 {
  tiene = 'E';
  ene = 'R';
 }
 else
 {
  tiene = temp/10;
  ene = (temp-(tiene*10));
  if (ene>9)
   ene = 9;
  ene = ene + 0x30;
  tiene = tiene + 0x30;
 }
}

byte Convert_Distance(long result)    //10 bit value are set equal to a distance
{
if(result>467) return(        0           );
if(result<          468       && result>        431       ) return( 11        );
if(result<          432       && result>        402       ) return( 12        );
if(result<          403       && result>        376       ) return( 13        );
if(result<          377       && result>        354       ) return( 14        );
if(result<          355       && result>        337       ) return( 15        );
if(result<          338       && result>        318       ) return( 16        );
if(result<          319       && result>        303       ) return( 17        );
if(result<          304       && result>        287       ) return( 18        );
if(result<          288       && result>        276       ) return( 19        );
if(result<          277       && result>        261       ) return( 20        );
if(result<          262       && result>        253       ) return( 21        );
if(result<          254       && result>        241       ) return( 22        );
if(result<          242       && result>        234       ) return( 23        );
if(result<          235       && result>        227       ) return( 24        );
if(result<          228       && result>        219       ) return( 25        );
if(result<          220       && result>        215       ) return( 26        );
if(result<          216       && result>        205       ) return( 27        );
if(result<          206       && result>        198       ) return( 28        );
if(result<          199       && result>        185       ) return( 29        );
if(result<          186       && result>        188       ) return( 30        );
if(result<          189       && result>        184       ) return( 31        );
if(result<          185       && result>        180       ) return( 32        );
if(result<          181       && result>        176       ) return( 33        );
if(result<          177       && result>        172       ) return( 34        );
if(result<          173       && result>        169       ) return( 35        );
if(result<          170       && result>        164       ) return( 36        );
if(result<          165       && result>        160       ) return( 37        );
if(result<          161       && result>        156       ) return( 38        );
if(result<          157       && result>        152       ) return( 39        );
if(result<          153       && result>        150       ) return( 40        );
if(result<          151       && result>        147       ) return( 41        );
if(result<          148       && result>        144       ) return( 42        );
if(result<          145       && result>        141       ) return( 43        );
if(result<          142       && result>        140       ) return( 44        );
if(result<          141       && result>        136       ) return( 45        );
if(result<          137       && result>        135       ) return( 46        );
if(result<          136       && result>        134       ) return( 47        );
if(result<          135       && result>        133       ) return( 48        );
if(result<          134       && result>        114       ) return( 49          );
if(result<          115       && result>        122       ) return( 50          );
if(result<          123       && result>        121       ) return( 51          );
if(result<          122       && result>        120       ) return( 52          );
if(result<          121       && result>        118       ) return( 53          );
if(result<          119       && result>        117       ) return( 54          );
if(result<          118       && result>        116       ) return( 55          );
if(result<          117       && result>        114       ) return( 56          );
if(result<          115       && result>        113       ) return( 57          );
if(result<          114       && result>        109       ) return( 58          );
if(result<          110       && result>        106       ) return( 59          );
if(result<          107       && result>        104       ) return( 60          );
if(result<          103) return(255);
}

void Stop_Go(long result)     //if closer than 13cm from sensor AGV go
{
   if(result>376)
      output_high(PIN_B2);
   else
      output_low(PIN_B2);
}

void main()
{
 long left1,left2,right1,right2;
 setup_adc_ports(ALL_ANALOG);

// set_tris_d(0xF0);
 lcd_init();

 while (1)
 {
  set_adc_channel(0);
  delay_ms(40);
  left1 = read_adc();
  Convert_LCD(Convert_Distance(left2));

  lcd_gotoxy(1,1);
  lcd_putc('L');
  lcd_putc('1');
  lcd_putc(':');
  lcd_putc(' ');
  lcd_putc(tiene);
  lcd_putc(ene);
  lcd_putc('c');
  lcd_putc('m');

  Stop_Go(left1);
  delay_ms(500);

  set_adc_channel(1);
  delay_ms(40);
  left2 = read_adc();
  Convert_LCD(Convert_Distance(left2));
  lcd_gotoxy(1,2);
  lcd_putc('L');
  lcd_putc('2');
  lcd_putc(':');
  lcd_putc(' ');
  lcd_putc(tiene);
  lcd_putc(ene);
  lcd_putc('c');
  lcd_putc('m');

  Stop_Go(left2);
  delay_ms(500);

  set_adc_channel(2);
  delay_ms(40);
  right1 = read_adc();
  Convert_LCD(Convert_Distance(right1));

  lcd_gotoxy(9,1);
  lcd_putc('R');
  lcd_putc('1');
  lcd_putc(':');
  lcd_putc(' ');
  lcd_putc(tiene);
  lcd_putc(ene);
  lcd_putc('c');
  lcd_putc('m');

  Stop_Go(right1);
  delay_ms(500);

  set_adc_channel(3);
  delay_ms(40);
  right2 = read_adc();
  Convert_LCD(Convert_Distance(right2));

  lcd_gotoxy(9,2);
  lcd_putc('R');
  lcd_putc('2');
  lcd_putc(':');
  lcd_putc(' ');
  lcd_putc(tiene);
  lcd_putc(ene);
  lcd_putc('c');
  lcd_putc('m');

  Stop_Go(right2);
  delay_ms(500);

 }
}
```

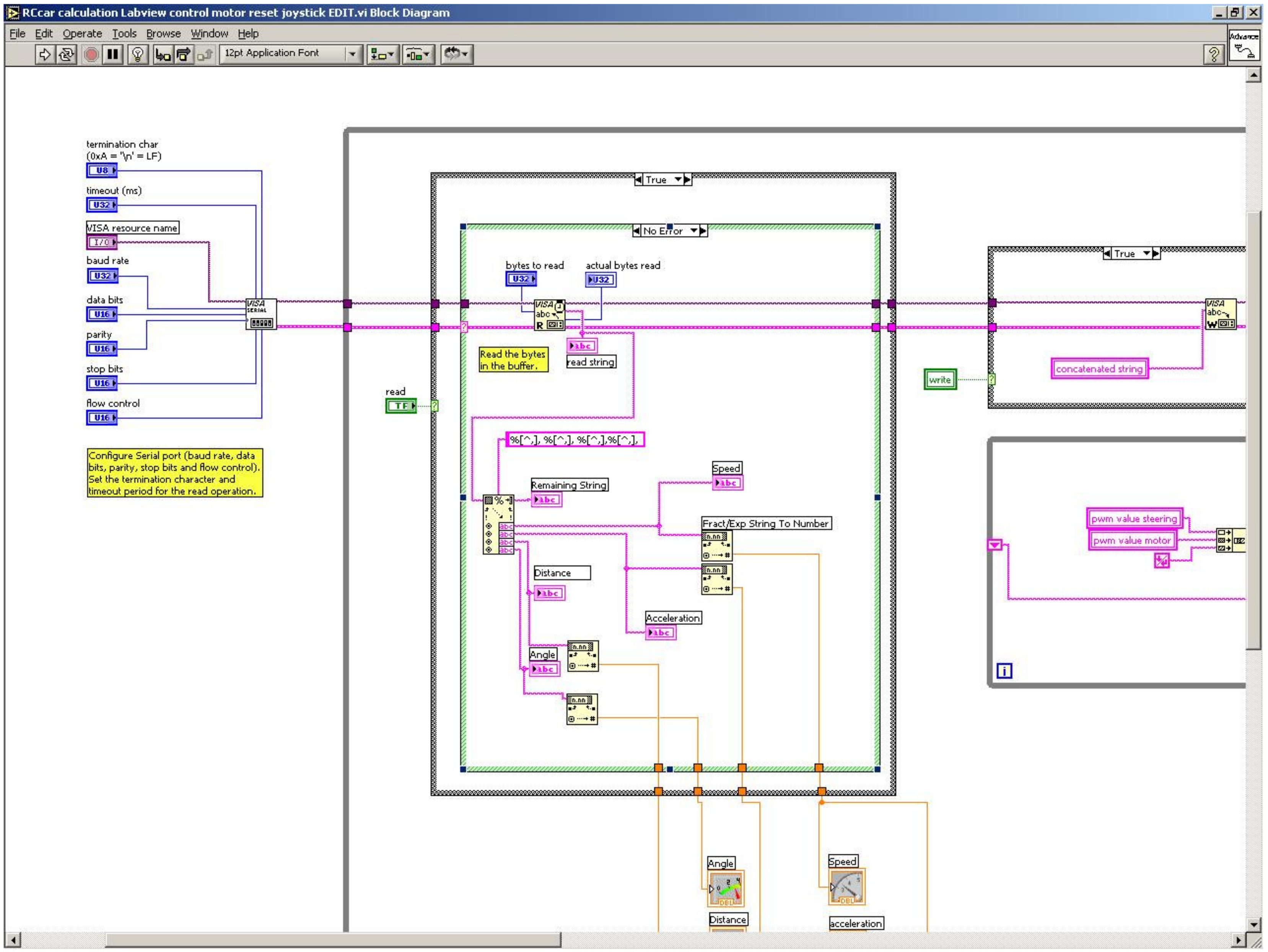
Appendix E: LABVIEW


RCcar calculation Labview control motor reset joystick EDIT.vi
File Edit Operate Tools Browse Window Help
stop
STOP
read
write
RESET CONTROL
Joystick
button1

VISA resource name: com4
baud rate: 19200
data bits: 8
parity: None
stop bits: 1.0
flow control: None
termination char (0xA = '\n' = LF): A
timeout (ms): 10000
bytes to read: 50
actual bytes read: 16
read string: .8, .0,23.4,14
Remaining String:
Speed: .8
Acceleration: .0
Angle: 14
Distance: 23.4

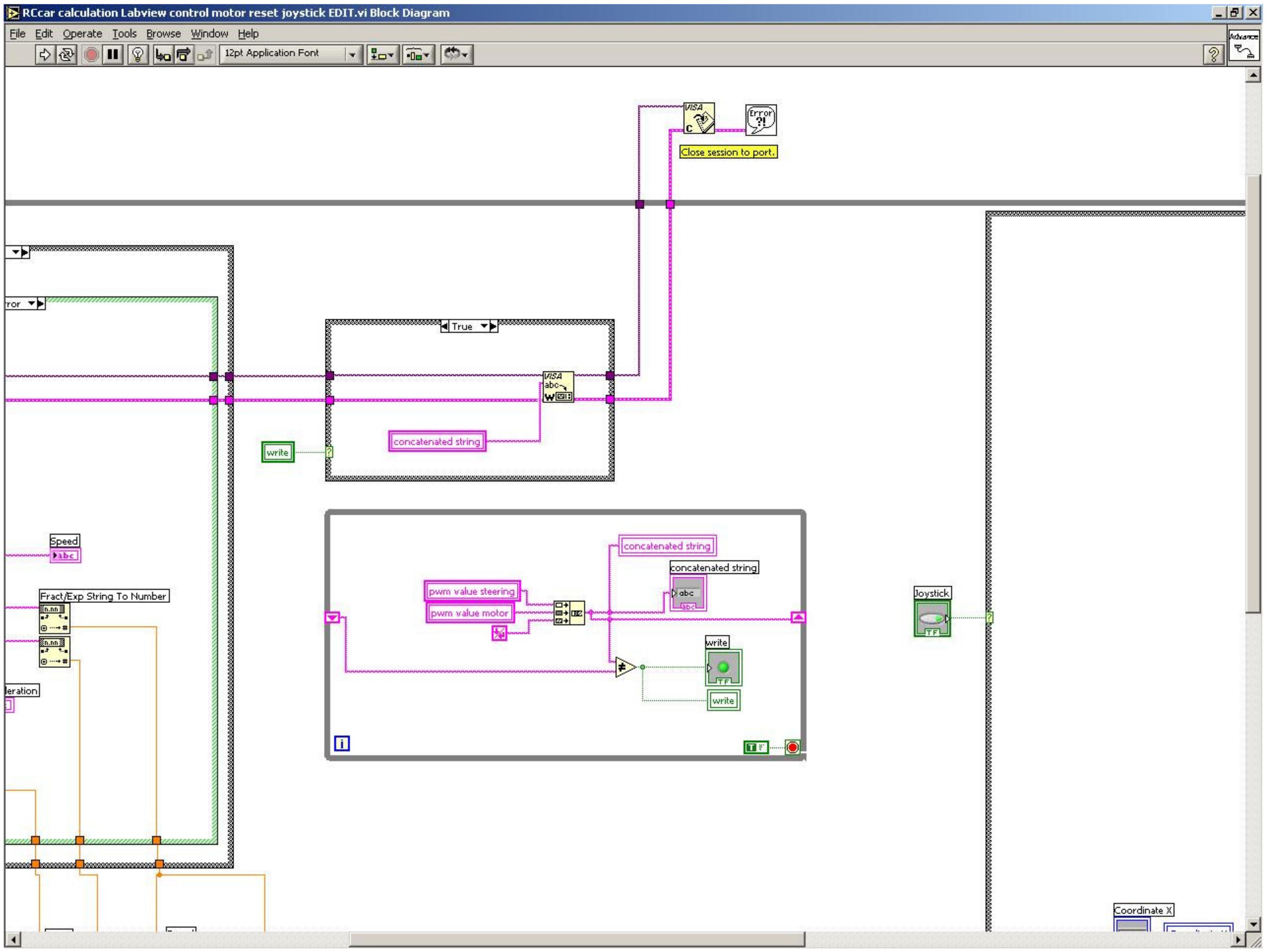
Speed: 0.8
kmh: 2.952
acceleration: 0
Angle: 14
Distance

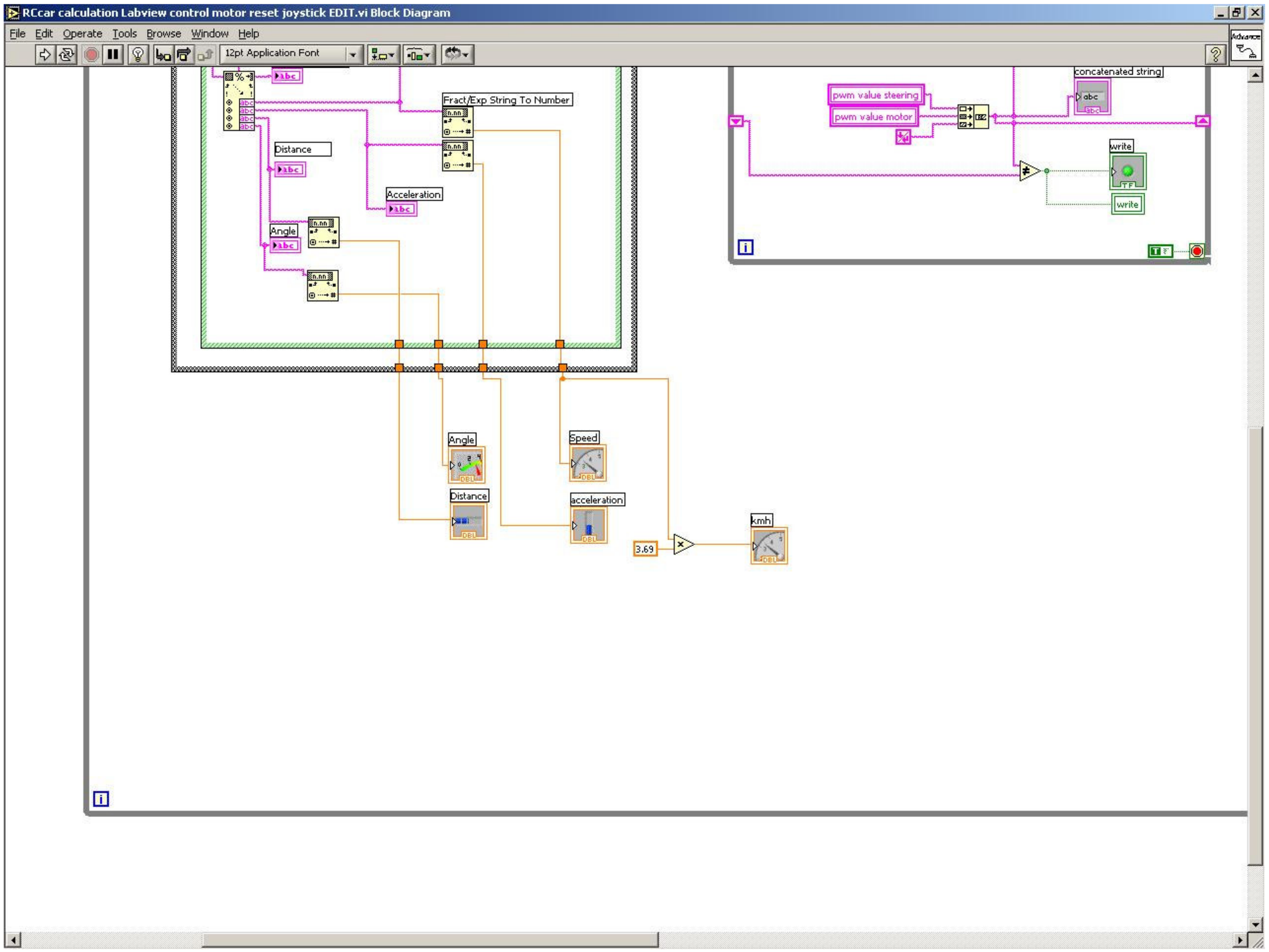
Angle
pwm value steering: 072
14
Motor
pwm value motor: 0150
8.8
concatenated string: 0720150
Coordinate X: 0
Coordinate Y: 0
Y: 0
X: 0

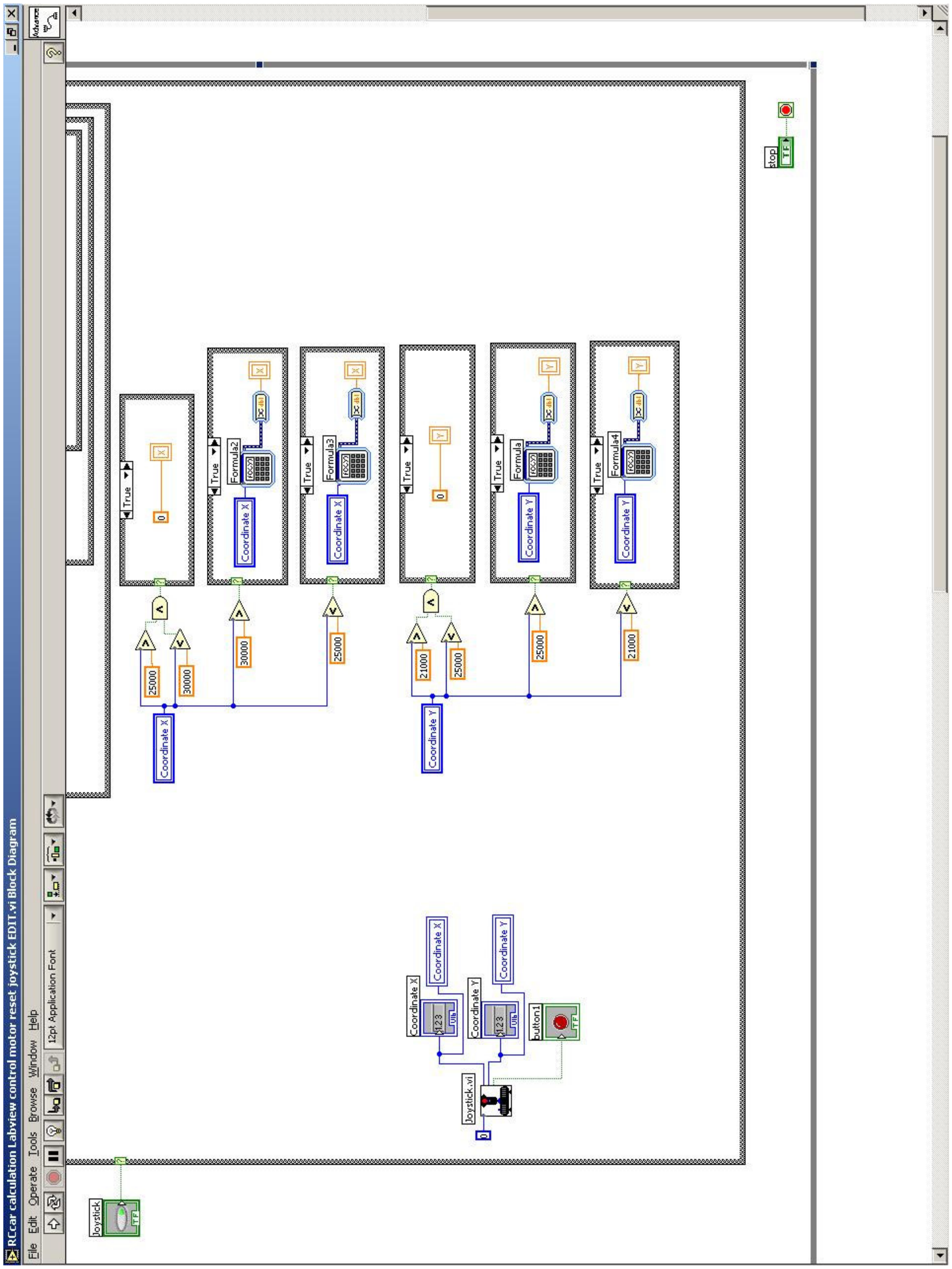


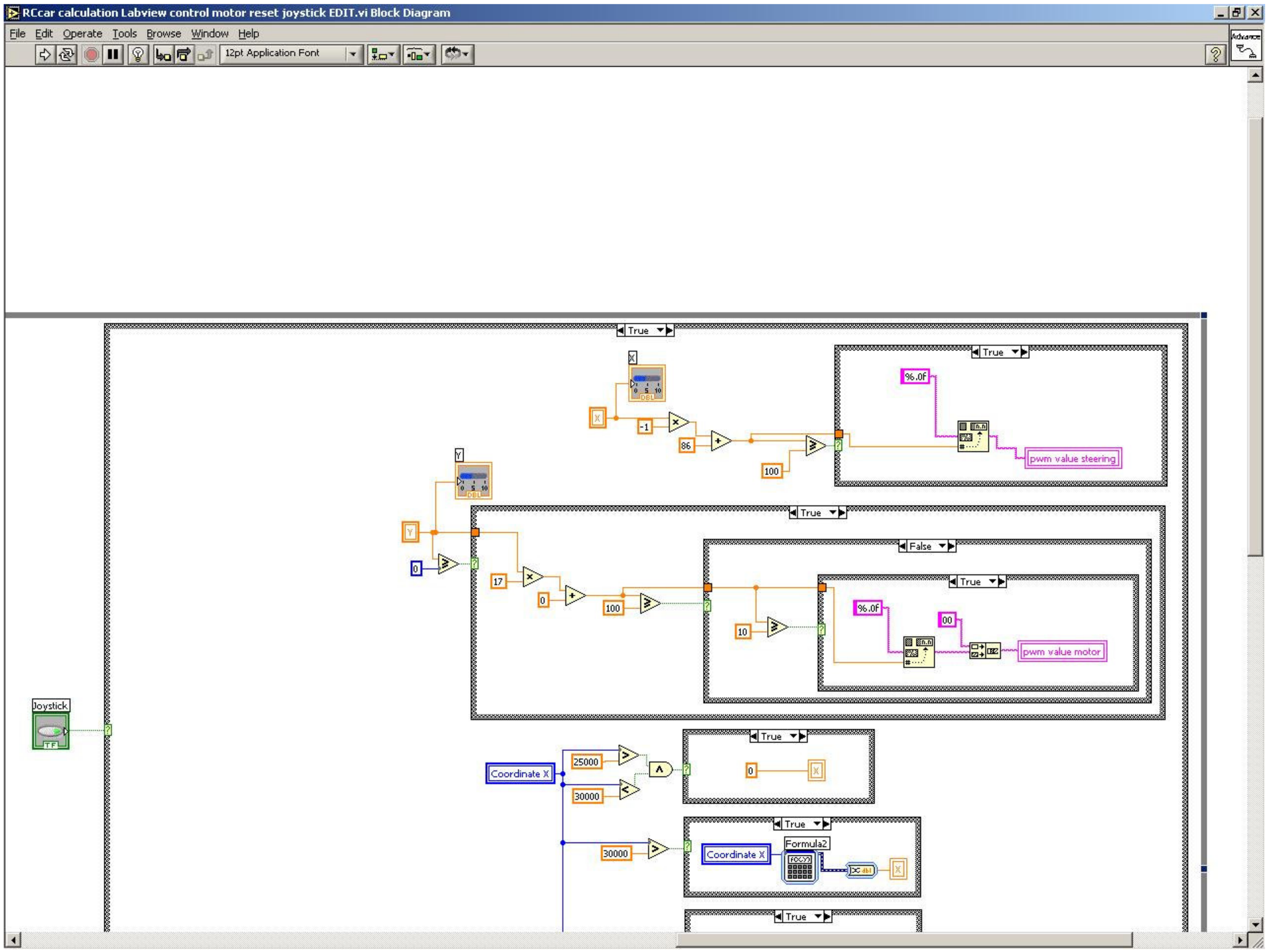
Configure Serial port (baud rate, data bits, parity, stop bits and flow control). Set the termination character and timeout period for the read operation.

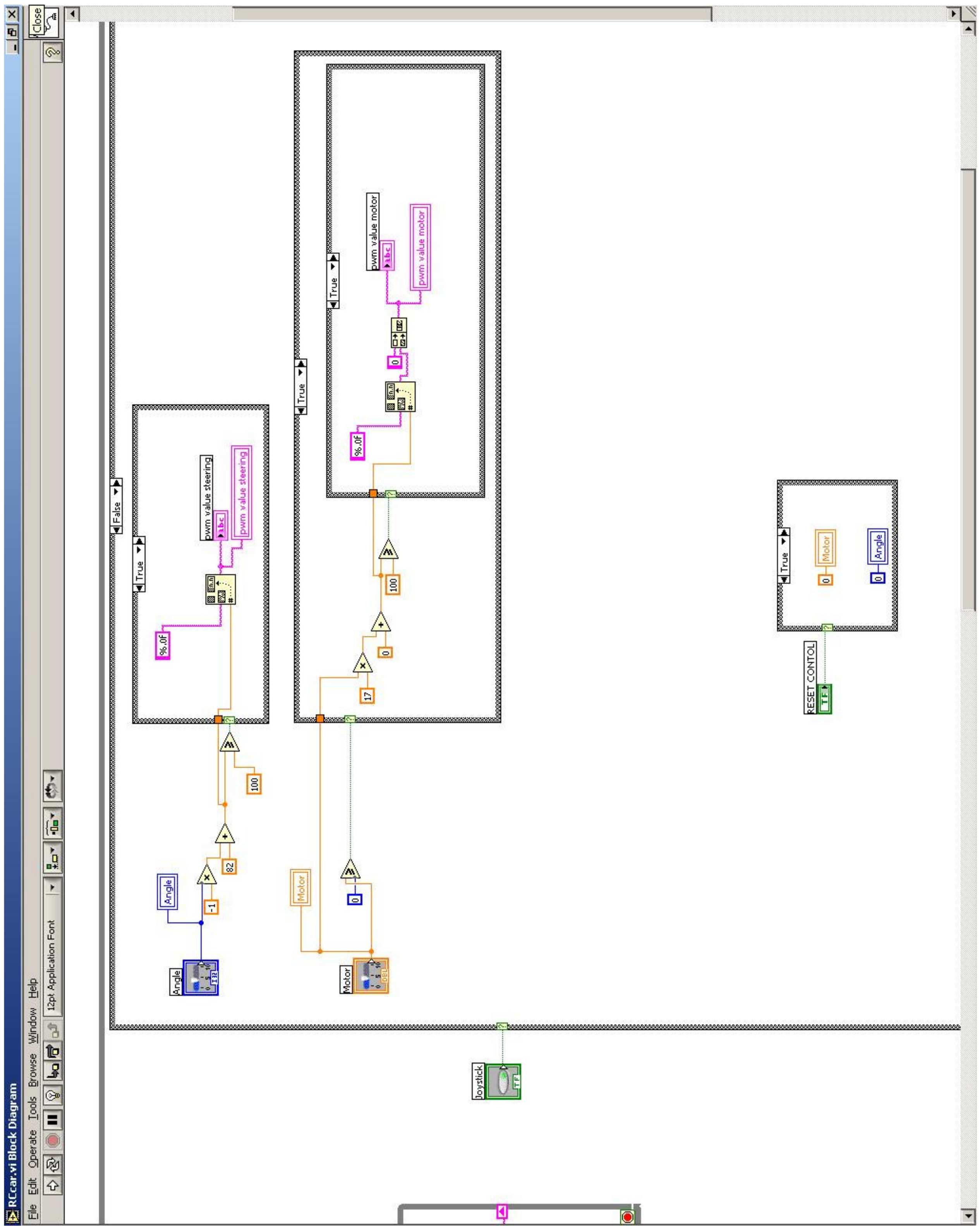
Read the bytes in the buffer.

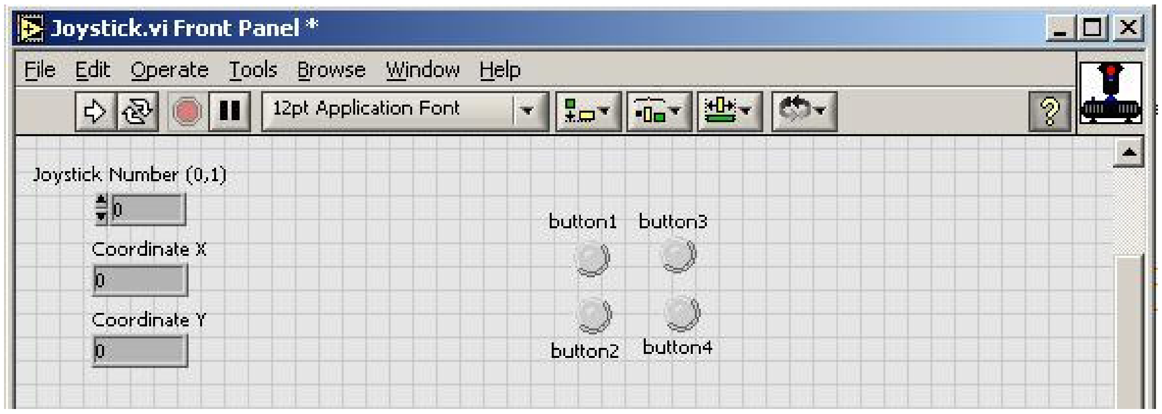

RCcar calculation Labview control motor reset joystick EDIT.vi Block Diagram
File Edit Operate Tools Browse Window Help
12pt Application Font
Close session to port.
True
write
concatenated string
Speed
Fract/Exp String To Number
pwm value steering
pwm value motor
concatenated string
concatenated string
write
write
Joystick
Coordinate X

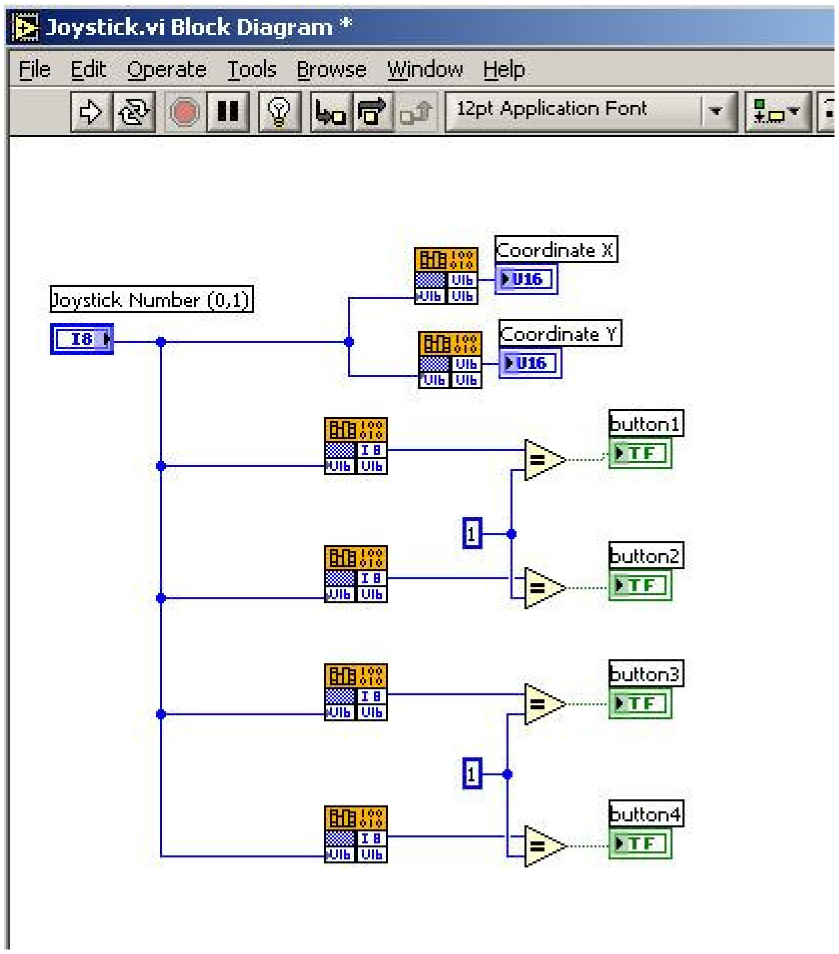

RCcar calculation Labview control motor reset joystick EDIT.vi Block Diagram
File Edit Operate Tools Browse Window Help
12pt Application Font
Fract/Exp String To Number
Distance
Acceleration
Angle
Angle
Speed
Distance
acceleration
3.69
kmh
pwm value steering
pwm value motor
concatenated string
write
write


RCcar calculation Labview control motor reset joystick EDIT.vi Block Diagram
File Edit Operate Tools Browse Window Help
12pt Application Font
Joystick
Joystick.vi
Coordinate X
Coordinate Y
button1
stop
True
Formula
Formula2
Formula3
Formula4
25000
30000
21000
0
X
Y


RCcar calculation Labview control motor reset joystick EDIT.vi Block Diagram
File Edit Operate Tools Browse Window Help
12pt Application Font
True
X
-1
86
100
%.0F
pwm value steering
Y
0
17
False
10
00
pwm value motor
Joystick
Coordinate X
25000
30000
Formula2


RCcar.vi Block Diagram
File Edit Operate Tools Browse Window Help
12pt Application Font
Close
False
True
Angle
Motor
Joystick
pwm value steering
pwm value motor
%.0F
-1
82
100
0
17
RESET CONTOL


Joystick.vi Front Panel *
File Edit Operate Tools Browse Window Help
12pt Application Font
Joystick Number (0,1)
0
Coordinate X
0
Coordinate Y
0
button1
button3
button2
button4


Joystick.vi Block Diagram *
File Edit Operate Tools Browse Window Help
12pt Application Font
Joystick Number (0,1)
I8
Coordinate X
U16
Coordinate Y
U16
button1
TF
button2
TF
button3
TF
button4
TF
1
1