PYTHON CONTROLLED GENERAL PURPOSE VEHICLE

CS04-608 B.Tech Mini Project 2011

Done By

AKHIL K–ETAIECS004

JITHU SUNNY–ETAIECS020

RAHUL C P–ETAIECS035

SALIM ALI U T–ETAIECS039

Guided By

JAYASREE M M

Lecturer

Dept of Computer Science And Engineering
Government Engineering College
Thrissur-680009

ABSTRACT

Python is an agile programming language. Rapid development, powerful libraries, portability & flexibility are counted as the major advantages of Python. The **AVR** is a modified Harvard architecture 8–bit RISC single chip microcontroller which was developed by Atmel. AVRs have been used in various automotive applications such as security, safety, power-train and entertainment systems. Some current usages are in BMW, Daimler–Chrysler and TRW. The motivation of this project is from the realization of the amazing possibilities we have if we combine the best of both worlds, Python & AVR.

This project created a **Python controlled general purpose vehicle**. The user can control the vehicle using the GUI running in a remote machine. The GUI is coded in PyQt4 & the serial port programming snippets embedded in it is coded in Python. Codes corresponding to the user controls is sent to the remote vehicle in serial mode through the USB – UART bridge connected to the USB port of the system & then through an 433 Mhz RF Transmitter – Receiver module. The AVR Atmega8 microcontroller in the remote vehicle receives this code & works accordingly. The code embedded in the microcontroller is written in C programming language & is compiled using avr-gcc compiler.

ACKNOWLEDGEMENT

We would like to thank Prof. K.V. Manoj Kumar, HOD, Dept. of Computer Science & Engineering, for providing the facilities required to complete this project.

We extend our sincere gratitude to Mrs. Jayasree M M for her consistent support & guidance given to us.

We thank Prof. K.S. Valsaraj for his prompt guidance that helped us to complete the project in time & also to adhere strictly to the software engineering process.

We express our deep gratitude to Mr. Pramode C.E for helping us to tackle the various stumbling blocks on our path.

Had not been there such a vast knowledge base, ready to help developers & user groups we wouldnt have completed this project.

We also thank the authors & people behind the following Books & Online tutorials:

1. Byte of Python – Swaroop C H [1]
2. Extreme electronics – Avinash Gupta [2]
3. Rapid GUI Programming with Python and Qt – Mark Summerfield [4]
4. Line Follower Robot–Mayur Agarwal,Prashant Agrawal,Krishna Nand Gupta And Hitesh Meghani[3]
5. Wikipedia–*http://www.wikipedia.org*

Last but not the least, the we would like to thank our senior Appu Sajeev for his invaluable practical tech-support. And we express our gratitude to all our friends & teachers who supported our humble project work.

Contents

List of Figures

Nomenclature

ADC	Analog to Digital Converter
AVR	This term stands for Alf (Egil Bogen) and Vegard (Wollan)(developers), RISC processor
CAN	Controller Area Network
DC	Direct Current
DFD	Data Flow Diagram
EEPROM	Electrically Erasable Programmable Read-Only Memory
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
LED	Light Emitting Diode
OS	Operating System
PWM	Pulse Width Modulation
RAM	Random Access Memory
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
SPI	Serial Peripheral Interface
TWI	Two Wire Interface
UART	Universal Asynchronous Receiver Transmitter
UML	Unified Modelling Language
USART	Universal Synchronous/Asynchronous Receiver Transmitter
USB	Universal Serial Bus

Chapter 1

Introduction

1.1 About AVR microcontrollers and Python programming language

The **AVR** is an 8–bit RISC single–chip microcontroller with Harvard architecture that comes with some standard features such as on–chip program (code) ROM, RAM, data EEPROM, timers and I/O ports. Most AVRs have some additional features like ADC, PWM, and different kinds of serial interface such as USART, SPI, I2C (TWI), CAN, USB, and so on. To be specific the microcontroller we are using is Atmega8.

Python is an interpreted, general–purpose high–level programming language whose design philosophy emphasizes code readability. Python aims to combine "remarkable power with very clear syntax", and its standard library is large and comprehensive.

1.2 Python controlled general purpose vehicle

The user can control the vehicle using the GUI running in a remote machine. The GUI is coded in Pyqt4 & the serial port programming snippets embedded in it is coded in Python. Codes corresponding to the user controls is sent to the remote vehicle in serial mode through the USB–UART bridge connected to the USB port of the system & then through an 433 Mhz RF Transmitter–Receiver module. The AVR Atmega8 microcontroller in the remote vehicle receives this code & works accordingly. The code embedded in the microcontroller is written in C programming language & compiled using avr-gcc compiler.

Chapter 2

Requirement Analysis

2.1 Hardware and Software Requirements

2.1.1 Hardware

Any hardware which supports common operating systems like GNU/Linux or Windows can be used. A USB port is necessary. The associated transmitter modules consists of a CP2102 USB–UART bridge & a 433 Mhz transmitter.

In the case of the remote embedded system part, an AVR Atmega8 is the major hardware required. The other hardware units required includes 433 Mhz Receiver - Transmitter, Motor driver IC LM293D, DC motors, chassis, etc.

2.1.2 Software

Python Programming Language

The programming language used for the development of this application is Python 2.6. It was developed by Guido van Rossum, and is one of those rare programming languages which is both very easy and very powerful.

PyQt4

PyQt4 is the latest version of PyQt which is a python binding of a cross platform GUI tool kit Qt. PyQt is also a free software. Its not part of a standard python package, but has to be separately installed.

2.2 Detailed Functionalities

The functionality provided by the project as seen by the user is triggered by the GUI at the machine. The various controls like forward, reverse, left, right, halt, etc as received from the user through mouse clicks/keystrokes is captured & appropriate signal is sent to the remote vehicle. The action corresponding to the various signals is taken by the machine. The functionality as of now is the basic movements of the vehicle but can be extended to a variety of functions.

2.3 Scope of the project

The GUI provided is demonstrating only the a bit of the scope of the project. The user controls received at the GUI can be transmitted to the remote vehicle & necessary action can be performed there. This is a good framework that can be followed or extended to do any real task like those in industries, military applications, etc.

Chapter 3

Design And Implementation

3.1 System Design

The UML Use Case Diagram is used to give the product view from user perspective. Data Flow Diagrams are used to represent the flow of data through different modules of the system. The overall system is represented using an integrated Data Flow Diagram.

3.1.1 UML Use Case Diagram

The user perspective can be depicted using a UML Use Case Diagram as follows:

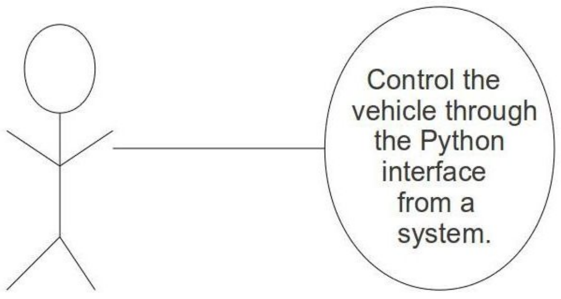
Control the vehicle through the Python interface from a system.

Figure 3.1: UML use–case diagram

The functions of each modules involved are:

1. **Input**: The user can input the various controls through the python interface in the computer.

2. **Processing** & **Communication**: The input is processed and is communicated to the remote RF module in the vehicle through serial transmission. The microcontroller decodes the signals.

3. **Output**: The microcontroller acts according to the signal understood.(Eg:-Fast forward, Blink head light, reverse, honk, etc.)

3.1.2 Data flow through modules–Data Flow Diagram

3.1.2.1 Level Zero DFD

The Level zero DFD or Context level DFD is given below that represents the system in the most outer layer

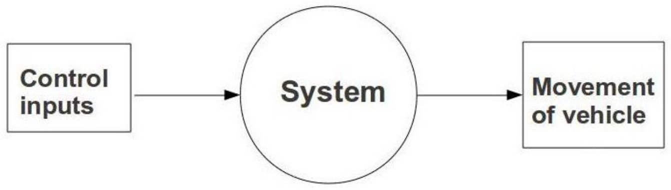

Figure 3.2: Context Level(Level 0) DFD

3.1.2.2 Level 1 Data Flow Diagrams

The Level 1(High level) Data Flow Diagram of the individual modules are given below:-

- **Input**

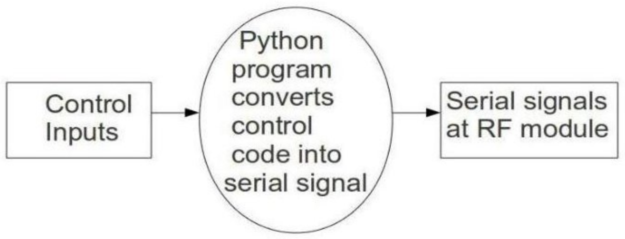
Control Inputs
Python program converts control code into serial signal
Serial signals at RF module

Figure 3.3: Level 1 DFD of input

Input: The user can input the various controls through the python interface in the computer.

- **Processing**

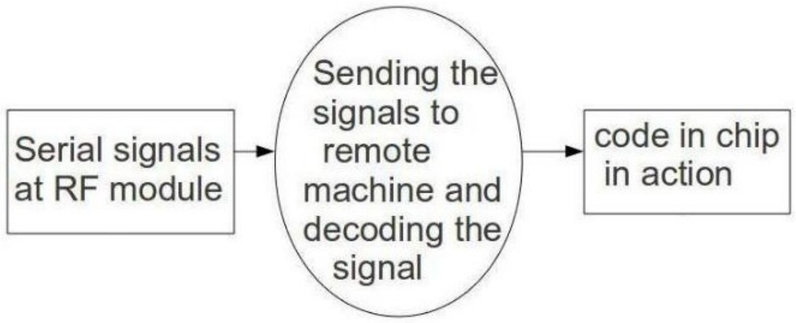
Serial signals at RF module
Sending the signals to remote machine and decoding the signal
code in chip in action

Figure 3.4: Level 1 DFD of processing

Processing & Communication: The input is processed and is communicated to the remote RF module in the vehicle through serial transmission. The RF module in the vehicle receives it & passes it to the microcontroller which then decodes the signals.

- **Output**

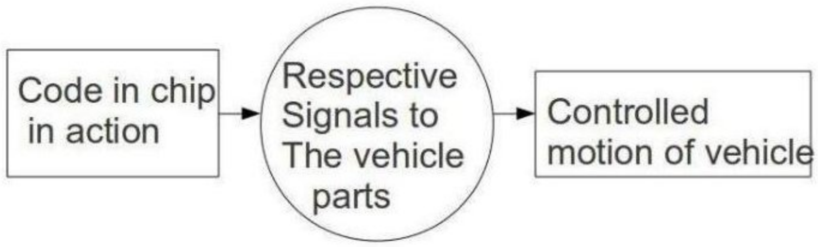
Code in chip in action
Respective Signals to The vehicle parts
Controlled motion of vehicle

Figure 3.5: Level 1 DFD of output part

Output: The microcontroller acts according to the signal understood.(Eg:-Fast forward, Blink head light, reverse, honk, etc.)

3.1.2.3 Level 2 Data Flow Diagrams

The Level 2(inner) Data flow diagrams of the modules input, processing & communication & output is given below:-

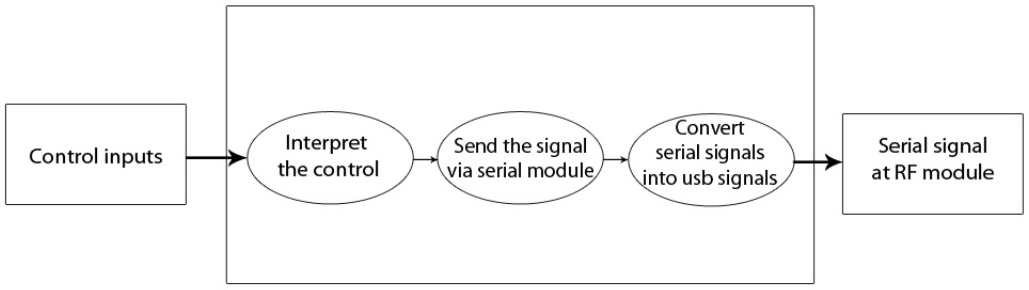
Control inputs
Interpret the control
Send the signal via serial module
Convert serial signals into usb signals
Serial signal at RF module

Figure 3.6: Level 2 DFD of input module

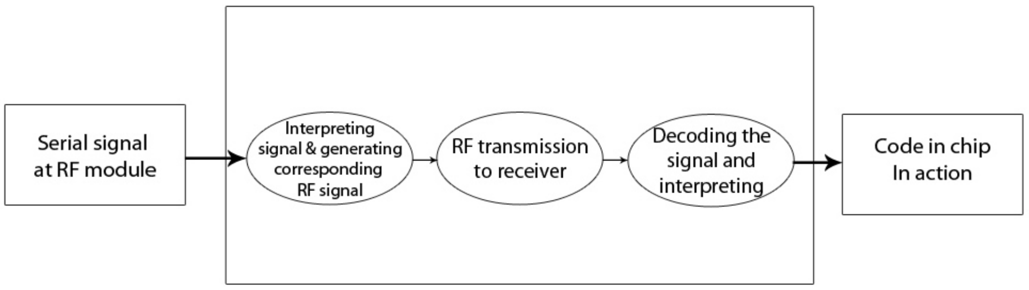
Serial signal at RF module
Interpreting signal & generating corresponding RF signal
RF transmission to receiver
Decoding the signal and interpreting
Code in chip In action

Figure 3.7: Level 2 DFD of processing module

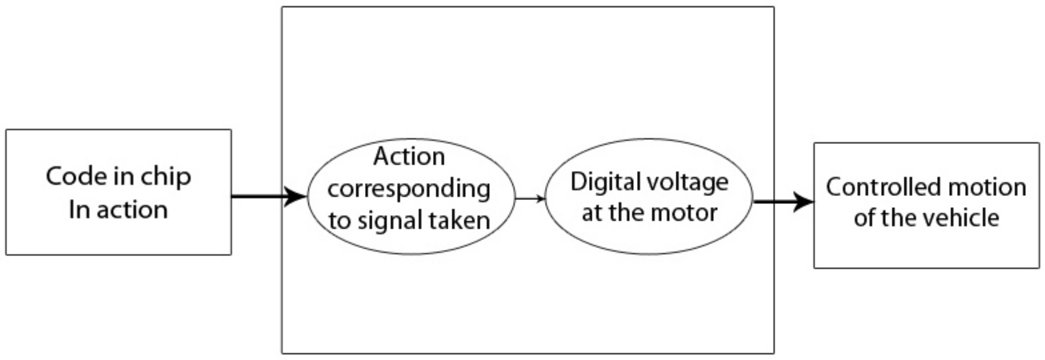
Code in chip
In action
Action corresponding to signal taken
Digital voltage at the motor
Controlled motion of the vehicle

Figure 3.8: Level 2 DFD of output module

3.1.3 Data Flow Diagram of the Integrated System

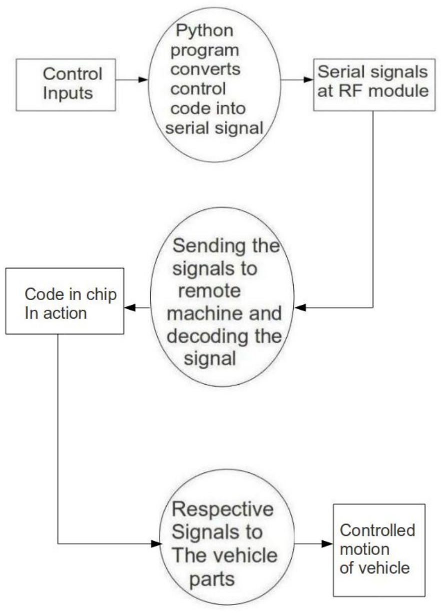
Control Inputs
Python program converts control code into serial signal
Serial signals at RF module
Sending the signals to remote machine and decoding the signal
Code in chip In action
Respective Signals to The vehicle parts
Controlled motion of vehicle

Figure 3.9: Integrated Data Flow Diagram

3.2 GUI/User Interface/Circuit Design

3.2.1 External Libraries & Modules used

The GUI is simple & consists of a half dozen of buttons with labels Accelerate, Reverse, Halt, left, right. The module 'pyserial' is used for writing to the serial port emulated. The binding of Qt GUI programming language for Python programming language, PyQt4 is used for coding the GUI. The modules QtCore, QtGui are used.

3.2.2 Screen shots

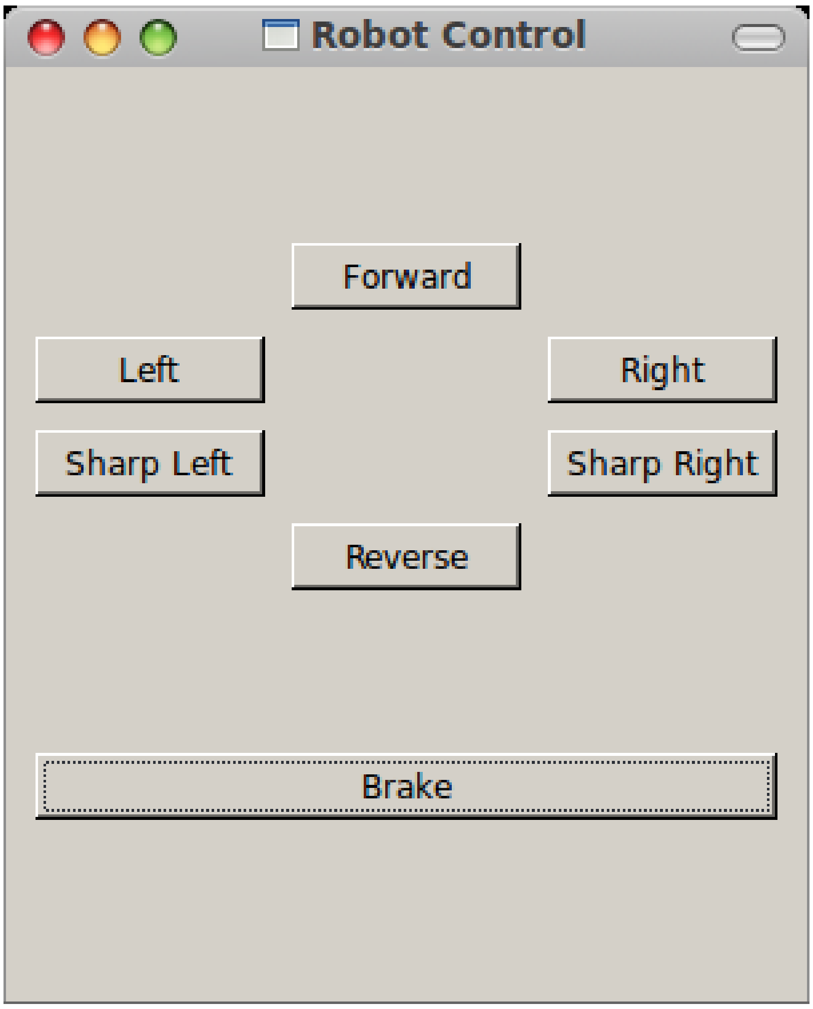

Figure 3.10: screen shot of GUI

3.2.3 Circuit Designs

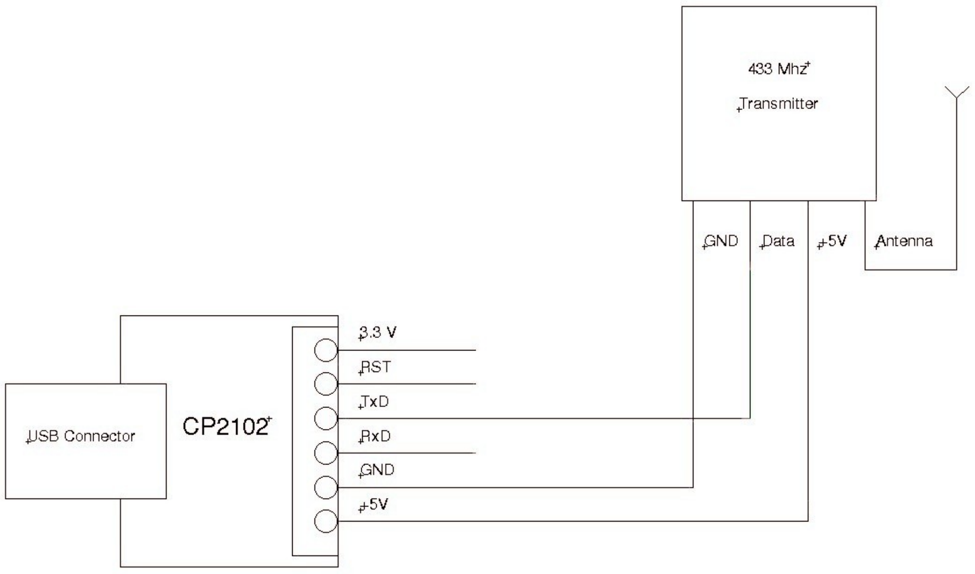

Figure 3.11: Circuit Design–PC Interface Part

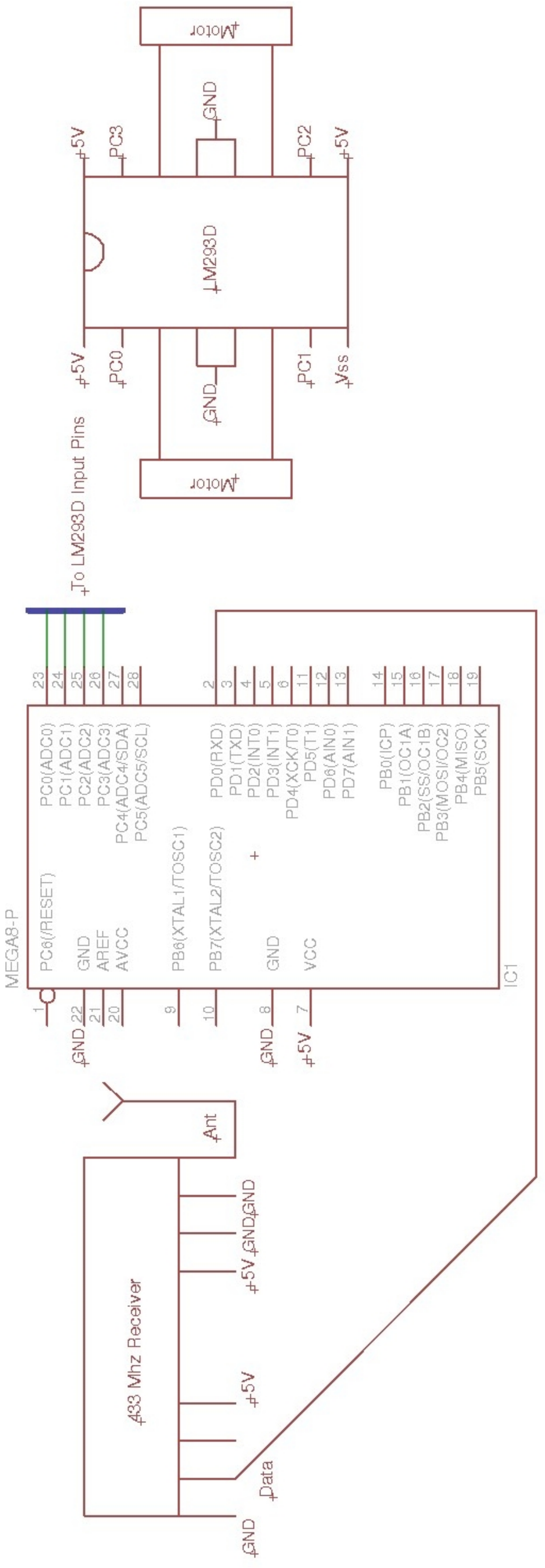

Figure 3.12: Circuit Design–Remote Vehicle

Chapter 4

Coding

4.1 Python code for GUI & Serial Data Communication

The GUI code using the library PyQt4 uses the modules QtGui & QtCore. These are the bare minimum modules required to create a set of buttons working on signals & slots mechanism. The pyserial module provides some high level functions for writing into & reading from the serial port. So in our case we just write into the serial port when we want to send a signal to the vehicle. The code is given below:-

```
#!/usr/bin/env python

import sys, serial
from PyQt4.QtGui import *
from PyQt4.QtCore import *

s = serial.Serial('/tty/USB0', 1200)

class Window(QWidget):
    def button_clicked(self, code):
        s.write(code)
        pass

    def key_pressed(self, code):
        s.write(code)
        pass

    def keyPressEvent(self, event):
        #Keypress is also detected
        if event.key() == Qt.Key_W:
            self.key_pressed('.f')
        elif event.key() == Qt.Key_Z:
            self.key_pressed('.b')
        elif event.key() == Qt.Key_A:
            self.key_pressed('.l')
        elif event.key() == Qt.Key_Q:
            self.key_pressed('.L')
        elif event.key() == Qt.Key_S:
            self.key_pressed('.r')
        elif event.key() == Qt.Key_E:
            self.key_pressed('.R')
        elif event.key() == Qt.Key_X:
            self.key_pressed('.h')

    def __init__(self):
        QWidget.__init__(self, parent = None)

        #Button creation
        forward_button = QPushButton("Forward")
        reverse_button = QPushButton("Reverse")
        left_button = QPushButton("Left")
        s_left_button = QPushButton("Sharp Left")
        right_button = QPushButton("Right")
        s_right_button = QPushButton("Sharp Right")
        brake_button = QPushButton("Brake")

        #Connecting the buttons with the function button_clicked
        forward_button.clicked.connect(lambda: self.button_clicked('.f'))
        reverse_button.clicked.connect(lambda: self.button_clicked('.b'))
        left_button.clicked.connect(lambda: self.button_clicked('.l'))
        s_left_button.clicked.connect(lambda: self.button_clicked('.L'))
        right_button.clicked.connect(lambda: self.button_clicked('.r'))
        s_right_button.clicked.connect(lambda: self.button_clicked('.R'))
        brake_button.clicked.connect(lambda: self.button_clicked('.h'))


        #Setting the layout
        grid = QGridLayout()
        grid.setSpacing(10)

        grid.addWidget(forward_button, 1, 1)
        grid.addWidget(reverse_button, 4, 1)
        grid.addWidget(left_button, 2, 0)
        grid.addWidget(s_left_button, 3, 0)
        grid.addWidget(right_button, 2, 2)
        grid.addWidget(s_right_button, 3, 2)

        vbox = QVBoxLayout()
        vbox.addLayout(grid)
        vbox.addWidget(brake_button)
        self.setLayout(vbox)

        #Centering the window
        screen = QDesktopWidget().screenGeometry()
        mysize = self.geometry()
        hpos = ( screen.width() - mysize.width() ) / 2
        vpos = ( screen.height() - mysize.height() ) / 2
        self.move(hpos, vpos)
        self.setWindowTitle('Robot Control')
        self.resize(300, 350)

if __name__=="__main__" :
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec_())
```

How to run this GUI : save this code as main.py and to run the program run the following

```
 python main.py
```

4.2 C code for the microcontroller

We program the AVR Atmega8 we have using C programming language & a cross compiler called avr-gcc. The code is given below:

```
#include <avr/io.h>
#include <util/delay.h>
#include <inttypes.h>

void USARTInit(uint16_t ubrr_value)
{

   //Set Baud rate
   UBRRL = ubrr_value;
   UBRRH = (ubrr_value>>8);

   /*Set Frame Format


   >> Asynchronous mode
   >> No Parity
```

```
    >> 1 StopBit
    >> char size 8

    */

    UCSRC=(1<<URSEL)|(3<<UCSZ0);


    //Enable The receiver and transmitter
    UCSRB=(1<<RXEN)|(1<<TXEN);


}

char USARTReadChar()
{
//Wait untill a data is available
while(!(UCSRA & (1<<RXC)))
{
//Do nothing
}

//Now USART has got data from host
//and is available is buffer

return UDR;
}

void USARTWriteChar(char data)
{
//Wait untill the transmitter is ready
while(!(UCSRA & (1<<UDRE)))
{
//Do nothing
}

//Now write the data to USART buffer

UDR=data;
}

void main()
{
char data;

DDRC = 0x0f;
```

```
USARTInit(51);


while(1)
{
data = USARTReadChar();

if(data == '.')
{
data = USARTReadChar();

switch (data)
{
case 'f':
PORTC = 0b00000110;
break;
case 'b':
PORTC = 0b00001001;
break;
case 'l':
PORTC = 0b00000010;
break;
    case 'L':
        PORTC = 0b00001010;
        break;
case 'r':
PORTC = 0b00000100;
break;
    case 'R':
        PORTC = 0b00000101;
        break;
case 'h':
PORTC = 0b00001111;
break;
default:
break;
}

}
}
}
```

Chapter 5

Testing and Implementation

5.1 Testing

The whole system contains two major parts: The GUI & the vehicle. So a robust testing process was done to check both these units.

5.1.1 GUI/Serial Data Communication Testing

The GUI we made was tested by setting up an LED circuit at the receiver end & testing from the machine. When user clicks on each button the corresponding LED combination will be lit. Thus the serial data communication part was also checked.

5.1.2 Complete system Testing

The GUI & serial port programming works perfectly, so we set up the final system with the actual motor driver IC LM293D & DC motors replacing the mock LED circuit. It was tested successfully.

5.2 Advantages and Limitations

5.2.1 Advantages

- The user don't have to learn the difficult serial port programming to send simple signals over serial port.
- The framework can be adapted to real world applications different from this.
- The PyQt4, Python coding allows the program to be portable across various platforms like Gnu/Linux, Windows, Mac OS, etc.
- The versatility of AVR atmega8 can be used for doing a variety of applications at the remote vehicle side.

5.2.2 Limitations

- At present only one way communication is implemented. This restricts the extensibility of the system into even more useful application. This can be rectified by using Transceivers(Transmitter + Receiver) at both sides.

5.3 Future Extensions

- Extend the GUI so as to include a option that allows user to specify a movement sequence.
  Eg:- Forward-Right-Forward-Halt-Right-Left-Forward-Reverse.
- Extend this to an automatic taxi. That is, in the GUI, the user can specify the stations in the preferred map. The system finds out the shortest path between the stations & takes the passenger to the destination.
- Can be used to create Industrial robots.

Chapter 6

Conclusion

The project has been completed successfully as specified by the requirements. On our way working on this interesting project, we learned many things. We got an invaluable experience by working on this project & learned various technologies on a need based manner. The system we have developed is just a minimal framework upon which anybody can work on to come up with interesting applications. We intend to extend the project to the above mentioned possibilities. We also expect that this project motivates others to build on this and create more wonderful, useful utilities.

Annexure 1

Pinout Diagrams of Chips Used

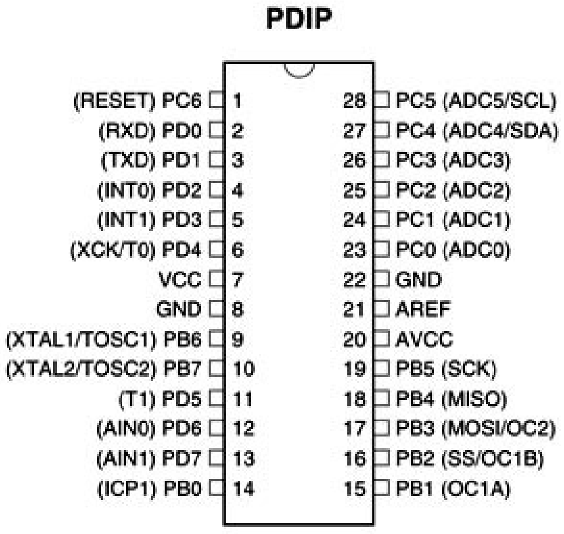

Figure 6.1: Pinout diagram of ATMEGA8

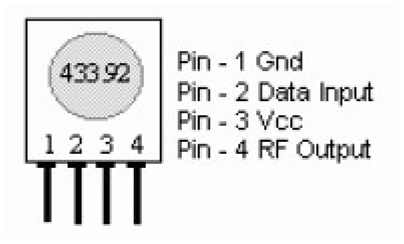

Figure 6.2: Pinout diagram of 433MHz Transmitter

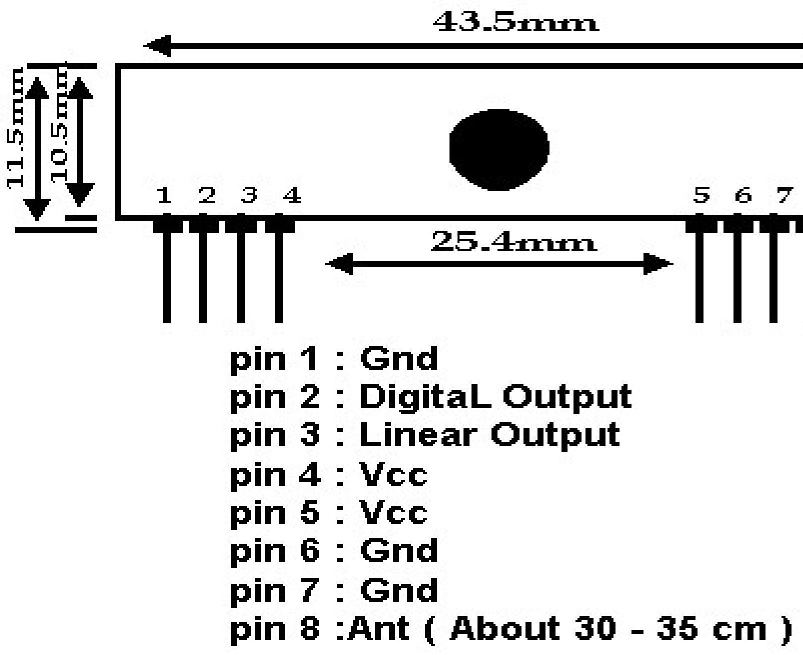

Figure 6.3: Pinout diagram of 433MHz Receiver

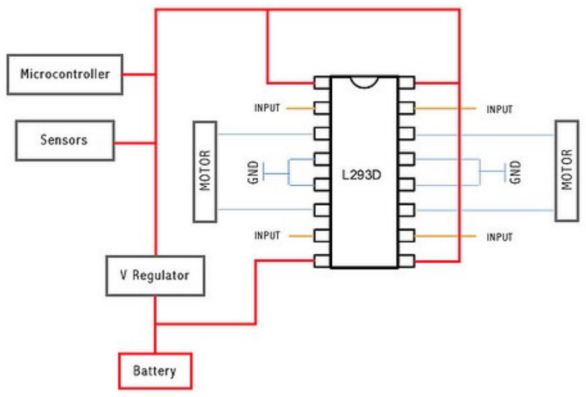

Figure 6.4: Pinout diagram of LM293D

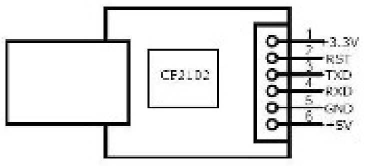

Figure 6.5: Pinout diagram of CP2102

Bibliography

[1] Swaroop CH, *A byte of python*, 2005.

[2] Avinash Gupta, *The avr programming tutorials*.

[3] Krishna Nand Gupta Mayur Agarwal, Prashant Agrawal and Hitesh Meghani, *Line follower robot*, Tech. report, Robotics Workshop EEE Department NIT Trichy, 2008.

[4] Mark Summerfield, *Rapid gui programming with python and qt*.