A Prototype of Web-Based, Modular and Wireless Monitoring Robot

Iman Firmansyah
0606001336

Department of Physics

University of Indonesia

Depok
2008

A Prototype of Web-Based, Modular and Wireless Monitoring Robot

a thesis

submitted in partial fulfillment
of the requirements for the degree

Master of Science

Iman Firmansyah
0606001336

Department of Physics

University of Indonesia

Depok
2008

Letter of Approval

Thesis	:	A Prototype of Web-Based, Modular and Wireless Monitoring Robot
Name	:	Iman Firmansyah
Number	:	0606001336

This thesis has been approved.

Depok, July 2008

Approving

1st Advisor	2nd Advisor
Dr. Santoso Soekirno	Dr. L.T.Handoko

Letter of Approval

Thesis	:	A Prototype of Web-Based, Modular and Wireless Monitoring Robot
Name	:	Iman Firmansyah
Number	:	0606001336

This thesis has been approved.

Depok, July 2008

Approving

1st Examiner	2nd Examiner	3rd Examiner
Dr. Martarizal	Dr. Prawito	Dr. Sastra Kusuma Wijaya

Preface

Over the last decade, the Internet has grown very rapidly from daily application such as E-mail, instant messaging, IP telephony, music and video on-demand, online banking and gaming to the serious scientific and industrial project such as creating web-based measurement and automation. The trend shows that the Internet will eventually be available to everyone, anytime and anywhere they are. However, behind the success of the internet, there is an ever-increasing need for people to design, apply, distribute or even commercialize these technologies. In my opinion, it is very interesting and useful to learn and apply the internet technology from every side of its part deeply.

Due to lack of knowledge about internet technology, in this thesis I am going to try to implement a little part of the internet programming into my project as best as I can, hoping that it's the first step for me to design the next better project which is related to internet technology. This project not only consists of internet programming but also microcontroller programming, computer programming and electronics interfacing.

Finally, I would like to thanks all engineers who have spent their efforts in making the internet so useful that we all now can take the benefit of it. We can't imagine what this world would be without the internet.....

Depok, May 2008

Iman Firmansyah

Acknowledgement

In this great opportunity, first of all, I would like to thanks Allah SWT for giving us everything which is very priceless. I also extend our thanks Dr.Santoso Sukirno who have guide and teach me while I was studying and completing this text. We are indebted to Dr.L.T.Handoko , the group leader, for his great support and contribution in helping, guiding and giving me his knowledges and experiences which are very useful. Thanks also to Bambang Hermanto and Zaenal Akbar as group members for nice cooperation and hard effort, who helped me in completing this thesis as good as it is. Special thanks to my friends at Magister of Physics in University of Indonesia for spending and sharing our time and knowledge together while we were studying.

Grateful acknowledgement is made to Dr.Ing.Priyo Sardjono and Ir.Tommy Budi Waluyo,M.Sc, as the Director and the leader of Research Center for Physics - Indonesian Institutes of Sciences. In addition , The authors would like to take the opportunity to thank all those individuals who have contributed or helped in some way in the preparation of this text.

Finally, I also owe a debt of gratitude to my family for their true and real supports.

Depok, May 2008

Iman Firmansyah

Abstract

A concept of a modular networked robot is introduced. A simple prototype of this kind of robot is developed to be accessible over web through wireless internet. The modularity is realized by dividing the robot into three modules: main, data acquisition and data processing modules. The main module consists of robot's body and main processor not only to handle the data processing, but also to store the whole software-based solutions. The data acquisition module is attached with the remaining hardwares like sensors, microcontrollers and signal conditioner circuits. The data processing module represents the software-based system to filter, store, analyze and display the data or its final results. Finally, all modules are integrated through a web-based interface to realize a mobile control and monitoring system. In this thesis, the detail discussion is given only for the microcontroller-based solutions required in the system.

Keywords : Modular; Wireless; Robot; Microcontroller

Abstrak

Diperkenalkan suatu konsep robot jaringan modular. Sebuah prototipe sederhana dari tipe robot ini dikembangkan sehingga dapat diakses melalui web dengan internet nirkabel. Sifat modular direalisasikan dengan membagi robot menjadi tiga modul: modul utama, akuisisi data dan pemrosesan data. Modul utama terdiri dari keseluruhan robot beserta prosesor utama yang tidak hanya bertugas untuk melakukan pemrosesan data tetapi juga untuk menyimpan seluruh solusi berbasis perangkat lunak. Modul data akuisisi dilengkapi dengan aneka perangkat keras lainnya seperti sensor, mikrokontroler dan rangkaian pengkondisi sinyal. Modul pemrosesan data mewakili sistem berbasis perangkat lunak untuk menyaring, menyimpan, menganalisa dan menampilkan data atau hasil proses akhir. Seluruh modul diintegrasikan sebagai sebuah perangkat muka web untuk merealisasikan sebuah sistem kendali dan pengamatan bergerak. Pada tesis ini diskusi detail diberikan hanya untuk solusi berbasis mikrokontroler yang diperlukan pada sistem ini.

Kata kunci : Modular; Nirkabel; Robot; Mikrokontroler

Contents

List of Figures

Chapter 1

Introduction

The growth and the success of internet technology lead to the great change of human being especially in data communication. Through the TCP/IP as a global standard protocol for communication, internet improves drastically the way of receiving and sending the data among applications such as web browsing, chatting, e-mail transmissions, file transfer, networking and so forth. Furthermore, it also enables people to perform any measurements or even controlling some devices using this protocol over internet.

The main advantage of using TCP/IP for data measurement and control are the interoperability and compatibility access many kind of devices anywhere they are as long as they are connected to the internet. However due to some reasons, sometimes it is possible for us to measure or acquire the data directly; this situation can be seen in a place where the surroundings environment does not allow us to measure some observables directly like inside nuclear reactors, volcanoes and so on.

To overcome these problems, we have developed a simple mobile control and monitoring system that can be put in any desired locations [17]. The robot system carries some sensors to acquire the data. On top of that, the robot is fully accessible to be controlled and monitored over web wirelessly. In this work, the focus is put on building the whole architecture rather than dealing with the mechatronical aspects of the robot itself. For the robot main body, we have deployed the simplest one which consists of two independents wheels and a single free wheel to realize a moveable robotic mechanism into four directions (moving forward, backward, turning left and right).

The system is divided into three modules to make it as modular as possible [17]. The concept of modularity is very useful to improve the flexibility, in particular for future extensions as a generic monitoring system. The modularity enables us to replace the sensors or actuators just to match particular needs. Therefore, this system is divided into three modules: main, data acquisition and data processing modules.

The main module involves the robot's body and a main processor which is responsible not only for being the central controlling unit but also for storing, manipulating and processing the data. The reasons why we deploy a PC main board rather than embedded system is due to flexibility and capability to be programmed using any available software and even we can use all the software needed freely such as Linux operating system including applications. However, using a PC main board requires more power and spaces.

The data-processing module consists of software responsible for acquiring, manipulating and processing the data from different sensors. As an example, the software to process the temperature data is different from other sensors such as humidity sensors and so on. As mentioned above, we can freely using any programming languages such as C, Java, Python, Perl, PHP and many more to develop those software. The last data acquisition modules or hardware modules has multiple hardwares related to data acquisition, measurement and control system.

The thesis is organized as follows. In chapter 2, the basic theory of each part in the system including programming language is described. Chapter 3 discusses the application and realization to build the whole system, followed by the results of our design in chapter 4. Finally, the summary is presented in the last chapter.

Chapter 2

Theoretical Background

In this chapter, all of the basic theories are presented in more detail, from the hardwares to softwares, required in designing and building the whole system.

2.1 Parallel Port

Concerning the data acquisition and control system, most people tend to choose a GPIB or data acquisition card due to its reliability and modularity especially for those users who design a complex system. This is the right choice to simplify the hardware and to reduce the time in designing the system as well. The most important thing we can achieve using this devices is the user can easily programme and interface it with some ready-to-use software such as LabView, Visual Basic and many more, so that the user can create the attractive interface on the screen easily with less effort.

However, there are other alternatives. We could also use I/O port for data acquisition and control systems. One of the cheapest and easiest solutions goes to parallel port or LPT. Parallel ports are available on all PCs and have high compatibility. We don't have to design or even buy an expensive data acquisition card, just plug the device and programme it, and then the I/O port is ready not only to control the devices but also to measure some physical observables using appropriate sensors through the port. Unfortunately, nowadays parallel port has been replaced by USB port due to its simplicity and speed of data transferring, and it has disappeared in modern laptops.

Parallel port is also known as a printer port or Centronics port, and is found mostly on the back of PC as a D-Type 25 Pin female connector [2] . It allows an input of up to 9 bits or output of 12 bits at a single period of time. Hence it requires less external circuit to implement many simple tasks. The parallel port has three commonly used base addresses which is composed of 4 control lines, 5 status lines and 8 data lines as shown in Tab. 2.1.

Parallel port is also simple and easy to programme, it can be programmed under both Windows and Linux [2]. The codes below are some examples of how to programme the parallel port that resides in an I/O address 378h using many different programming languages:

Assembler

```
MOV DX,0378H
MOV AL,N
OUT DX,AL
```

BASIC

```
Out &H378, N
```

Pascal

```
Port[$378]:= N
```

C

```
Outp(0x378,N);  or
Outportb(0x378,N);  or
Outb(N,0x378);
```

Where N is the data to be generated from the parallel port.

2.2 Sensors

A sensor is a device that converts a physical variable into an electrical signal [14]. There are various sensors from analog sensors through digital sensors which

Table 2.1: Pin Assignments of the D-Type 25 pin Parallel Port Connector

Pin No (D-Type 25)	SPP Signal	Direction In/out	Register	Hardware Inverted
1	nStrobe	In/Out	Control	Yes
2	Data 0	Out	Data	
3	Data 1	Out	Data	
4	Data 2	Out	Data	
5	Data 3	Out	Data	
6	Data 4	Out	Data	
7	Data 5	Out	Data	
8	Data 6	Out	Data	
9	Data 7	Out	Data	
10	nAck	In	Status	
11	Busy	In	Status	Yes
12	Paper-Out/Paper-End	In	Status	
13	Select	In	Status	
14	nAuto-Linefeed	In/Out	Control	Yes
15	nError/nFault	In	Status	
16	nInitialize	In/Out	Control	
17	nSelect-Printer/nSelect-In	In/Out	Control	Yes
18-25	Ground	Gnd		

are used by robot for sensing and measuring physical variables. Sensors give the robot ability to understand its environment, such as where the robots position is, how the robots environment temperature is, or even what the physical condition surrounding of the robot is, and so on. Before the robot can understand the information from the sensors, firstly it needs some signal conditioning as can be seen below

A signal can be defined as any physical quantity that varies with time, space, or any other independent variable [14]. Based on Fig. 2.1, the signal conditioning consists of multiplexer, amplifier, filter, A/D converter and microcontrollers. Microcontrollers are the most important devices of almost all types of modern digital instruments systems. They play an important role in data acquisition, data processing and control, and instrument communication. In this section, however, we will select and discuss only a number of sensors which will be used in this project.

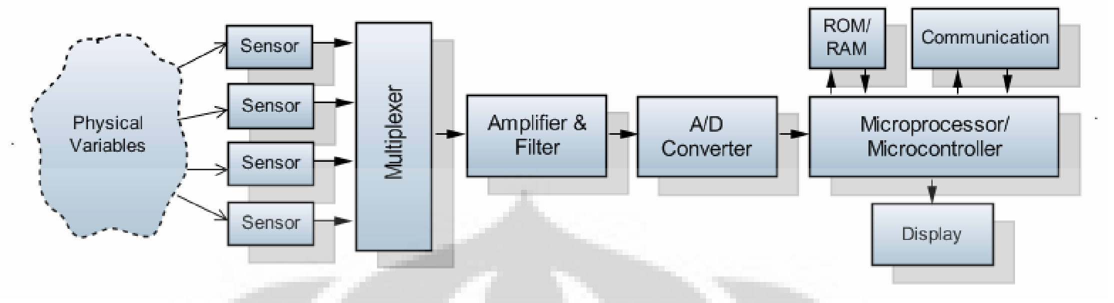


Figure 2.1: Signal Conditioning

2.2.1 Electronics Compass

Electronics compass is crucial tools for navigation especially for mobile robot in order to the robot knows its position [5]. Nowadays, replacing an old analog magnetic compass by an electronic solution offers many advantages like having a solid-state component without moving parts and the ease of interfacing with other electronic systems such as microcontroller. This electronics compass provides control lines for reset, calibration, mode selection and communication channel even though not all of which have to be used for all applications.

The magnetic field of the earth depicted in Fig. 2.2 is the physical quantity to be evaluated by a compass, its lines point from the earth's South Pole to its north pole [4]. The field lines are perpendicular to the earth surface at the poles and parallel at the equator. An important fact is that the magnetic poles do not coincide with the geographical poles, which are defined by the earth's axis of rotation. The angle between the magnetic and the rotation axis is about 11.5. Therefore, the magnetic field lines do not exactly point to geographic or "true" north.

For the magnetic field sensors within electronics compass system, the magneto resistive (MR) technology is the preferable solution [4]. The MR technology offers a much more cost effective solution and higher sensitivity, as it requires no coils to be wound and can be fabricated in an IC-like process. Magneto resistive (MR) sensors make use of the magneto resistive effect, the property of a current carrying magnetic material to change its resistivity in the presence of an external magnetic

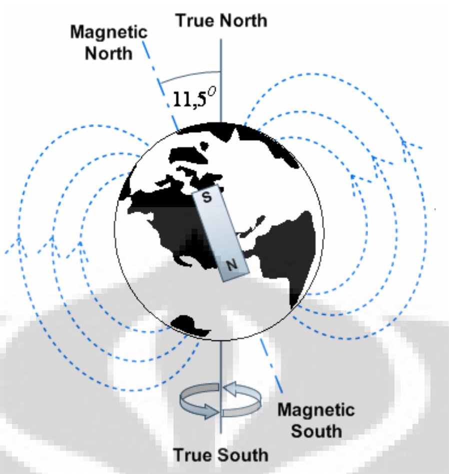


Figure 2.2: Earth's Magnetic Field

field. Fig. 2.3 shows a strip of ferromagnetic material, called permalloy.

During deposition of the permalloy strip, a strong external magnetic field is applied parallel to the strip axis. By doing this, a preferred magnetization direction is defined within the strip. In absence of any external magnetic field, the magnetization always points into this direction. Based on Fig. 2.3, this is assumed to be the x-direction, which is also the direction of current flow. An MR sensor now relies on two basic effects, the strip resistance R depends on the angle α between the direction of the current and the direction of the magnetization and the direction of magnetization and therefore α can be influenced by an external magnetic field Hy, where Hy is parallel to the strip plane and perpendicular to the preferred direction.

When no external magnetic field is present, the permalloy has an internal magnetization vector parallel to the preferred direction, i.e. $\alpha = 0$. In this case, the strip resistance R has its maximum value Rmax. If now an external magnetic

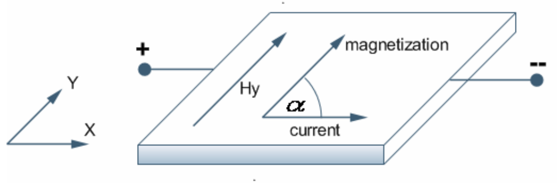


Figure 2.3: Magnetoresistive effect

field Hy is applied, the internal magnetization vector of the permalloy will rotate around an angle α. At high field strengths, the magnetization tends to align itself parallel to Hy and the rotation angle α approaches 90^o. In this case, the resistance reaches its minimum value Rmin. The equation $R = R_0 + \Delta R.cos^2\alpha$ for $\alpha = 0^o \Rightarrow Rmax$ and $\alpha = 90^o \Rightarrow Rmin$ gives the functional dependence between R and α, where Ro = Rmin and $\Delta R = (Rmax - Rmin)$. Finally, the function of R versus Hy is as follows:

$$R = R_0 + \Delta R.(1 - (\frac{H_y}{H_0})^2) \tag{2.1}$$

2.2.2 Binary sensors

Binary sensors or digital sensors as shown in Fig. 2.4 is easy to implement, this type of sensors only returns either 0 or 1 which is mean produces voltage 0 or 5 volts [7]. There are two types of binary sensors, the first is mechanical base sensor such as tactile sensor and the second is binary sensors which is made based on optical switch. A tactile sensor, for example using a micro switch will generate a high signal or 5 volts when this sensor is attached to pull-up resistor, otherwise this sensor return 0 volt or low signal. Unfortunately, the switch sensor produces an error or bouncing effect. The error with this configuration due to the mechanical nature of any switches when a switch is operated like a push button or limit switches. Spikes of low and high voltages always occurs across the switch resulting a series of High and Low signals as if the switch is pressed many times.

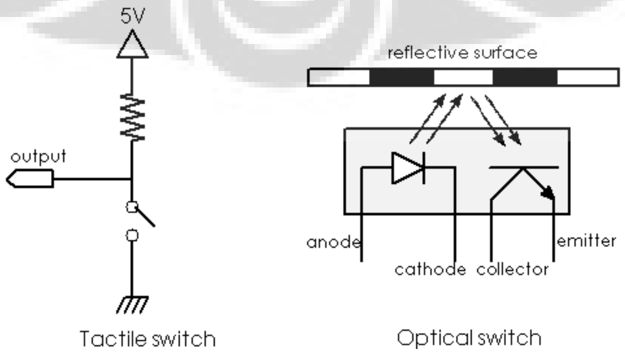
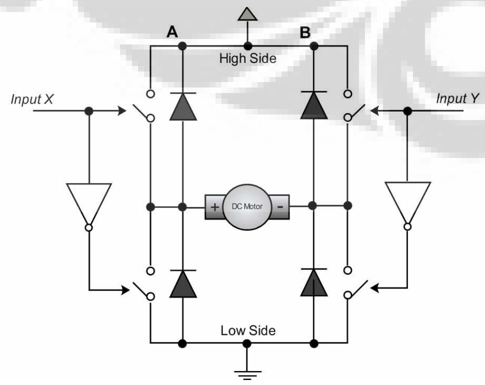


Figure 2.4: Binary Sensor

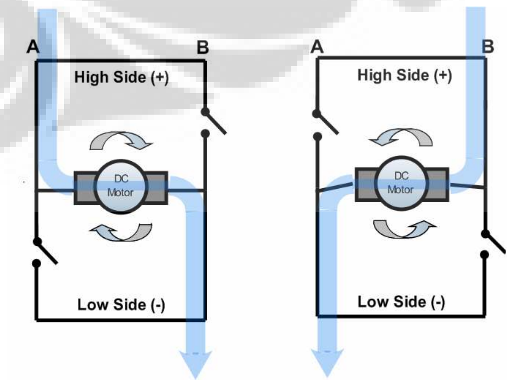
The second configuration is using optical methods, this method using a pair of infrared source and receiver [6]. The sensor consists of an infrared LED and phototransistor mounted side by side in the same plastic package. The package material allows infrared light to pass, but filters ambient visible light. When a reflective surface is placed at a suitable distance in front of the sensor, some of the emitted light is reflected back to the phototransistor, which then conducts. Optical methods are very useful in sensing objects and surfaces, the presence of an object can be sensed if it breaks a light beam or in another if it reflects the beam. One of the real applications of this sensor is for calculating the rotation of the robots wheels.

2.3 Actuators

Now, let us concern about the actuator in the robot to make it moveable in several directions. Generally, the DC and stepper motors are widely used in robotic system due to its simplicity and low cost of implementation. Moreover, both types of motor are widely used to generate rotating moves for robot mechanics. The stepper motor is suitable for rotating system which requires precise angular displacement, but it consumes more electricity power. On the other hand, the DC motor so consumes less that it powerful for robotic systems with small power source like LMNR [19].



(a) The Principle of H-Bridge



(b) H-Bridge operation

Figure 2.5: H-Bridge Circuit

H-bridge is a to enable the robot moving in several directions such as moving forward and backward, turning left and right or even stop [10]. In principal, the H-bridge does a task to control the motor to run forward or backward by changing its switches. Further, combining two DC motors and deploying the H-bridge circuits is enough to realize the simplest mechatronical architecture as mentioned above. The typical H-Bridge circuit is depicted in Fig. 2.5(a).

The H-bridge circuits basically consist of four switches that are changed by a specific configuration to control the DC current direction to the motors . Switching input X to low and input Y to high states yields the upper right and lower left circuits are closed, while the other two remain open [6]. This makes the current flows to the DC motor from the high side of Pair B to the low side of pair A. Therefore the motor starts rotating as shown in Fig. 2.5(b). If both of these inputs are reversed, then all switches change the state, and the current flows with the opposite direction which makes the motor rotates reversely. In order to protect the circuit from the reversible current, it must be connected as shown in the figure.

2.4 Navigation & Turning Geometries

Navigation is the most important part for a mobile system to catch up its current position [14]. This is also important to determine how to reach the next destinations based on the actual position as shown in Fig. 2.6. There are several methods to determine the robots positions. One of them is using the dead reckoning algorithm. Dead reckoning is a process of calculating the robots position relying on the previously known position.

The mathematical equation used for calculation of dead reckoning depends on the architecture of robot's actuators [16]. For instance, a car-like steering wheels which have a constant curve and known radius, the differential wheels that can make very narrow turns or even spin in its position, etc. To implement a dead reckoning method, first of all we need to know the robots position and orientation or direction. The starting position and the distance can thereafter be calculated from wheel encoder attached to the wheel. Hence, it gives a fixed position in

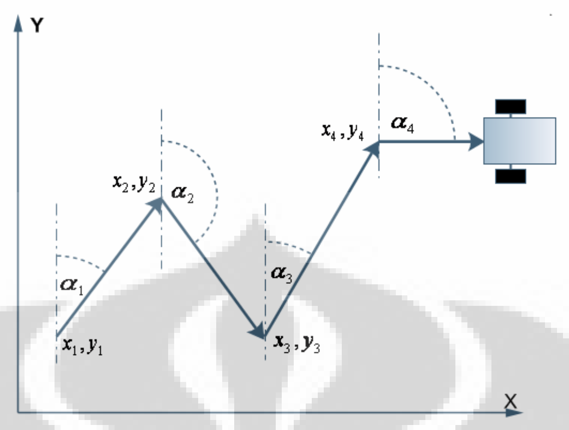


Figure 2.6: Dead Reckoning.

(x,y) space. The actual orientation can be measured by using embedded compass, present compass has built in digital interface so that it will simplify us in interfacing and designing the circuits especially when we interface it using microcontroller. However, there are some drawbacks when especially applied for longer period. All inaccuracies might be due to sensor error or wheel slippages and less accurate orientation [16]. The dead-reckoning method is depicted at Fig. 2.7.

Measuring the distance of robot on a straight line is much easier than on a curved line when the robot turns left or right [10]. For a straight line, the distance is calculated by counting the wheel rotations. However for a curved line, we should deploy some mathematical tools to obtain a relevant equation. A general movement of robots traveling on a curved line is shown in Fig. 2.7. Each wheel is moving with different speed which makes the robot turns left or right. Let us consider if the left wheel goes as far as d_L, while the right wheel takes d_R. The result is the robot would turn through the arc of a circle, centered at point O with radius R. It turns with an angle of θ.

As d_L and d_R are the controllable variables, it is useful to relate all variables in term of them. As the angle subtended by arcs d_L, while d_R is the same,

$$\frac{d_L}{(R + \frac{A}{2})} = \frac{d_R}{(R - \frac{A}{2})} = \theta \; , \tag{2.2}$$

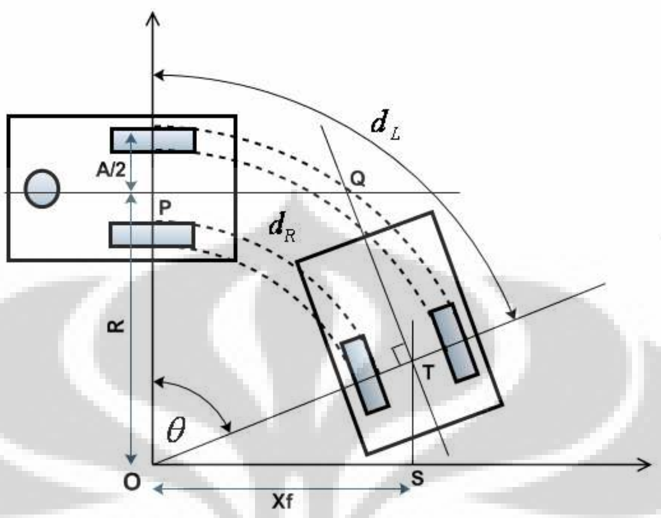


Figure 2.7: Turning Geometris.

which leads to,

$$d_L.(R-\frac{A}{2}) = d_R.(R+\frac{A}{2}) \ . \tag{2.3}$$

It finally yileds,

$$R = \frac{A}{2}.\frac{(d_L+d_R)}{(d_L-d_R)} \ . \tag{2.4}$$

Knowing that

$$\theta = \frac{d_L}{(R+\frac{A}{2})} \ , \tag{2.5}$$

and substituting R one obtains,

$$\theta = \frac{(d_L-d_R)}{A} \ . \tag{2.6}$$

2.5 Wireless Networking

For many years, people used to connect computers or any devices (printers, laboratory equipment etc) each other through networks [11]. However, an evolution has been taking place, since the wireless network is getting to replace the wired one. The wireless network enables people to extend their workplaces and to use computer applications anywhere. As a result, it provides users the freedom of

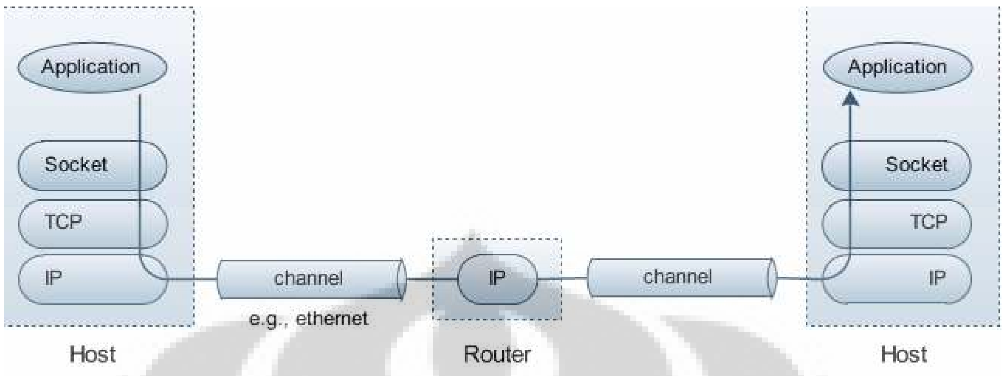


Figure 2.8: TCP/IP.

movement especially when they want to connect into networks from separated locations.

A computer network consists of multi machines interconnected by communication channels. These machines are called as hosts and routers. Implementing a useful network requires some advanced technologies. To keep things manageable and modular, different protocols are designed to solve different sets of problems. A protocol is an agreement about the packets exchanged by communicating programs and what they mean. A protocol tells how packets are structured for example, where the destination information is located in the packet and how big it is as well as how the information is to be interpreted.

TCP/IP is the most popular protocol among many network architectures [13]. Sometimes it is called as a protocol suite or protocol family as shown in Fig. 2.8. It turns out to be useful to organize protocols in a family into layers, Fig. 2.8 shows relationships among those protocols, applications, and the sockets API in the hosts and routers, as well as the flow of data from one application to another. The boxes labeled with TCP and IP represent implementations of those protocols. Such implementations typically reside in the operating system of a host. Applications access the services provided by UDP and TCP through the sockets API. The arrow depicts the data flow from the application, through the TCP and IP implementations, to the network, and vice versa.

TCP/IP solves the problem by translating the sequence of channels and routers between any two hosts to be a single host-to-host channel. The Internet Protocol provides a datagram service: each packet is handled and delivered by the network independently, like telegrams or parcels sent via the postal system. To make this work, each IP packet has to contain the address of its destination, just as every package we mail is addressed to somebody's address. The layer above the IP is called the transport layer. It offers a choice between two protocols: TCP and UDP. Each builds on the service provided by IP, but they do it in different ways to provide different kinds of channels used by application protocols with different needs.

Depending on the size of the physical area that can be covered by wireless networks, wireless networks can be divided into Wireless Personal-Area Network (PAN), Wireless Local-Area Network (LAN), Wireless Metropolitan-Area Network (MAN) and Wireless Wide-Area Network (WAN) [12]. In the following, we are focusing on wireless local area network or LAN in building the system as shown in Fig. 2.9. Wireless LANs is similar to traditional Ethernet-wired LANs both in hardware architecture and in protocols. The only difference between them is that wireless LAN does not needs the wires as a data transmission media. Wireless LAN consists of radio NICs, wireless access points, routers and antenna.

The most important part of a wireless LAN is a radio NIC that provides wireless connectivity. An access point contains a radio card that communicates with individual user devices on the wireless LAN, as well as a wired NIC that interfaces to a distribution system, such as Ethernet. System software within the access point bridges together the wireless LAN and distribution sides of the access point.

By definition, a router transfers packets between networks [13]. The router chooses the next best link to send packets on through the closest available paths. Routers use Internet Protocol (IP) packet headers and routing tables, as well as internal protocols, to determine the best path for each packet. A wireless LAN router adds a built-in access point function to a multiport Ethernet router. This combines multiple Ethernet networks with wireless connections so that it gives wireless users the same ability as wired users to send and receive packets over

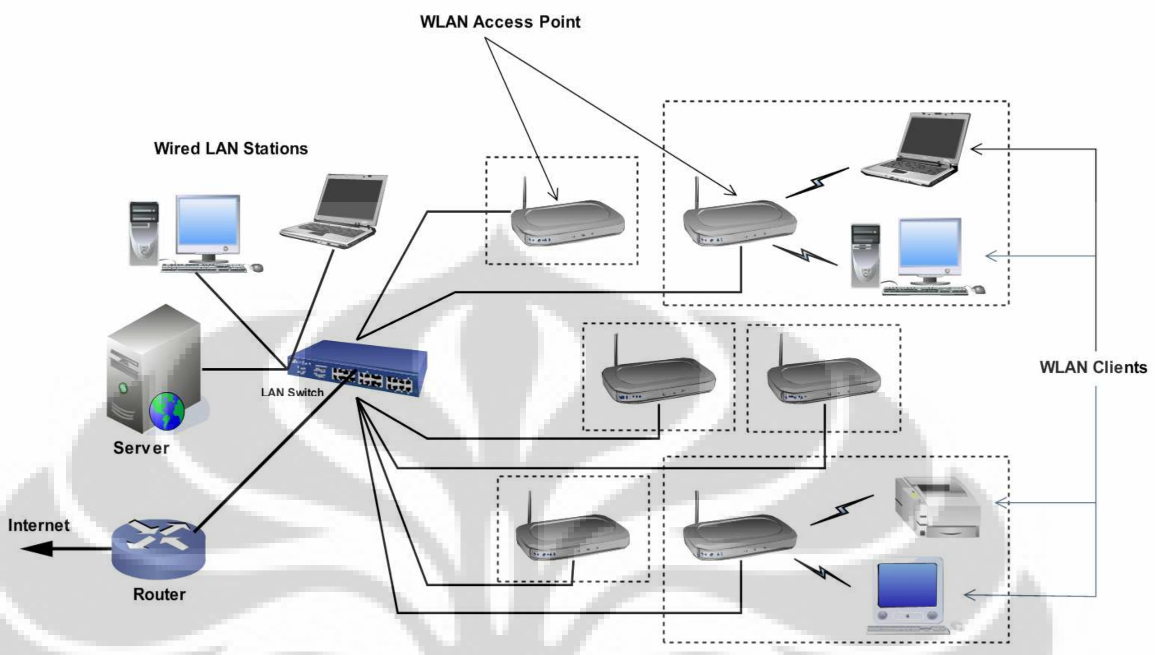


Figure 2.9: Wireless LAN.

multiple networks. Routers also implement Dynamic Host Configuration Protocol (DHCP) services for all devices. DHCP assigns private IP addresses to devices.

2.6 Common Gateway Interface (CGI)

CGI stands for ”Common Gateway Interface”. It is a program interface on the server which accepts requests sent by the client [8]. CGI is the part of web server modules to communicate with another programme running on the server. Using CGI, then the Web server is able to receive several requests including some specific data or input data from the client. Finally, CGI program processes those data and the server passes the response back to the Web browser such that the result can be viewed by the client through web browsers. All processes are diagrammatically shown in Fig. 2.10.

Generally, CGI is used to build dynamic HTML such as online dictionary, accessing multiple database engines and graphic manipulation libraries or even executing a source code to either read or write the data to the hardwares on a

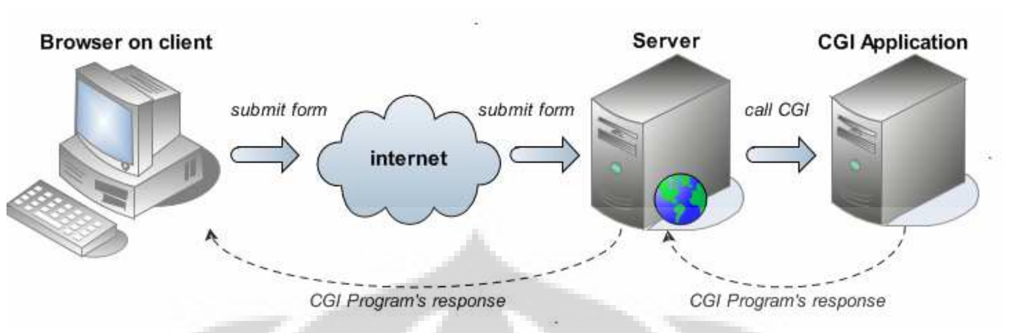


Figure 2.10: Common Gateway Interface (CGI).

server. However, the most interesting part of CGI is its ability to be programmed using different programming languages: AppleScript, C/C++, C Shell, Perl, Tcl, and Visual Basic and so on.

Chapter 3

System Design

The concept of modular wireless robot is shown in Fig. 3.1. It is consisting of three modules: 1) main unit, 2) data-acquisition and 3) data processing modules [17]. The goal of this concept is to improve the flexibilities of robot to meet various purposes of monitoring tools by less modification. This characteristic would enable users to easily replace, for instance the censors and relevant data processing modules with any available packages later on with the same main unit. Or, inversely replacing the main unit with the same data acquisition and processing modules.

According to the concept, the system has some characteristics as below:

1. All aspects are fully controllable wirelessly over web through TCP/IP protocol.

2. Acquired data are stored and processed at the robot's local system independent from external apparatus.

3. The processed data can be retrieved and analyzed by users also over web in such a way that no need for additional software installation at the user's terminal.

4. The hardware driven parts are replaced as much as possible with the software driven systems, even in the main unit.

5. High compatibility due to limited proprietary hardwares, and then embedded software, used in the system. Moreover all software-based parts are developed

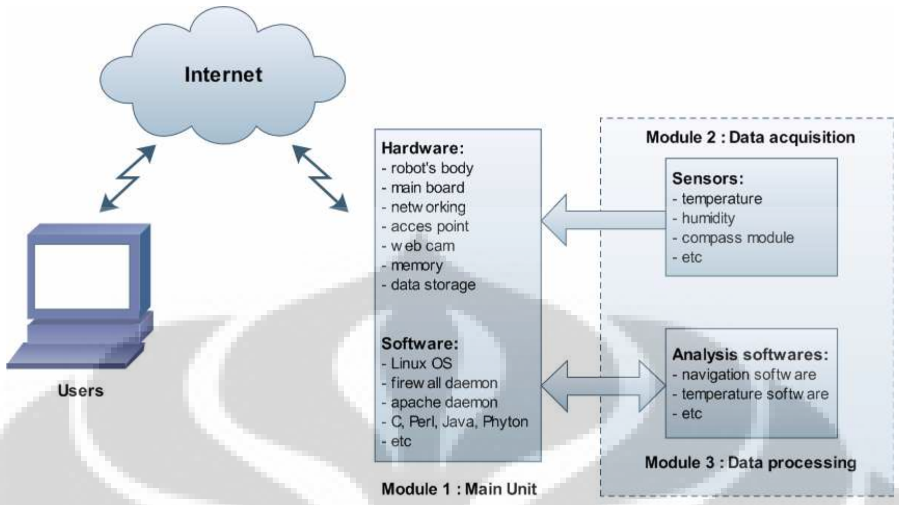


Figure 3.1: The concept of modularity.

using freely available open-source softwares. In our case we use Ubuntu Linux for the operating system, Apache for the web-server and some GNU Public License development languages.

Data acquisition module contains some hardware interfaces for both controlling the actuators and retrieving the measured data from sensors. As can be seen from Fig. 3.2, two microcontrollers are used in this design since there are two main tasks related to hardware in the system for controlling and data retrieval. Both microcontrollers are connected to the parallel ports. This port is used as communication channel between them not only for sending the command but also for receiving the data from data acquisition module.

In principle both devices can be integrated in a single microcontroller. However, due to the requirement of modularity, especially high degree of freedoms not only in controlling actuator but also in taking the data from multiple sensors. The second reason to deploy two microcontrollers is to reduce the latency time due to the limitation of microcontroller itself.

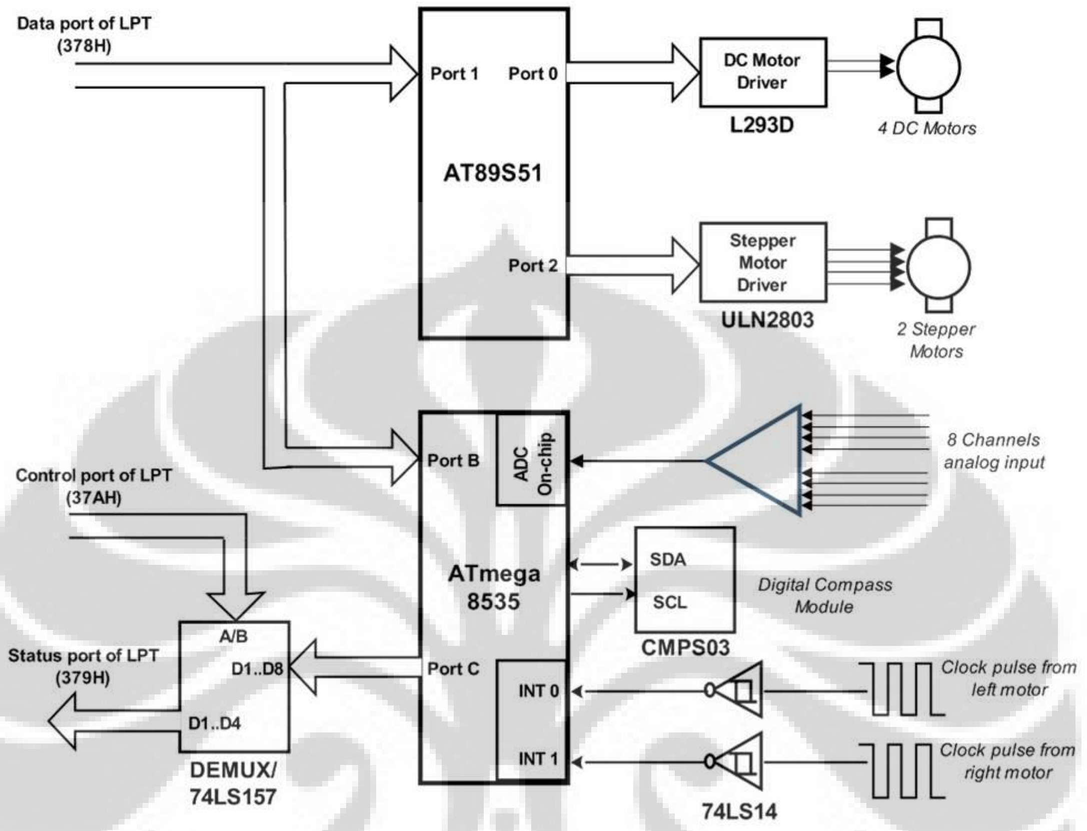


Figure 3.2: Data Acquisition Module.

3.1 Parallel Port Programming

Writing and reading the data using parallel port is easier than using serial port or USB port due to the data format of this port, i.e. no need for users to add some specific IC converter for interfacing the parallel port. For instance, IC max232 from Maxim should be used to interface the devices through either serial port or USB port. Basically, these ICs work as a converter of the serial data format and make it readable by the devices. Since the data receives from or sends to LPT port in parallel format, in some simple applications one can use the data directly without any additional complex circuits through this port for both data reading and sending, such as driving LEDs or controlling the relays.

In this design, we use the parallel port for both reading and writing the data into microcontroller triggered by particular commands from main system. As the

Table 3.1: Parallel port addresses.

Port	LPT1	LPT2
Data	0x378	0x278
Status	0x379	0x279
Control	0x37A	0x27A

result, the microcontroller translates each different command into different task subsequently, such as driving the motors or reading the data from sensor attached on the system.

3.1.1 Parallel port programming under Linux

Parallel or LPT port can be programmed in any operating systems, not only Windows but also Linux. However, there are clearly some differences for each platform. In Windows, especially in Win95/98, we can use *inportb*() and *outportb*() or $_i np$() or $_o utp$() functions. On the other hand, we need to write a kind of device driver or dynamic link library to access parallel port in the newest versions like Windows NT, Windows 2000 and Windows XP.

In this section, we discuss programming the parallel port under Linux. We use Linux since its free license and it runs on the most of PC main boards. Programming parallel port under Linux is almost the same as in Windows, even though slight differences between both of them are there. Before programming the parallel port, first of all we need to know about the port address. As shown in Tab. 3.1.1, each parallel port address consists of data, status and control port addresses.

In Linux, all hardware device-specific functions are encapsulated in the open, close, read, write modules [3]. This is similar to the process of reading and writing the data into a file. In order to do so, one should invoke the system root privileges to directly access the hardware by calling the *ioperm()* function declared in "unistd.h". As a result, it enables us to access the hardware devices. The syntax is *ioperm(port, num, turn_on)*, where *port* is the first port number, and *num* is the number of consecutive ports to give the access. After calling *ioperm()* to enable the ports being used, then one can access the parallel port. Use *inb(port)* to read a byte of data from a port, and *outb(value, port)* function to write a byte of data. Below is a sample code called *writelpt.c* for writing the data to a port address that resides at 0x378 :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <asm/io.h>

#define base    0x378   /* printer port base address */
#define status  base+1  /* status port address */
#define control base+2  /* control port address */

main(int argc, char **argv)
    {
    int value;

    if (argc!=2) {
        fprintf(stderr,"Error input parameter.....\n"),
        exit(1);}
    if (sscanf(argv[1],"%i",&value)!=1) {
        fprintf(stderr,"Error input parameter.....\n"),
        exit(1);}
    if ((value<0) || (value>255)) {
        fprintf(stderr,"Parameter must be between 0 and 255\n"),
        exit(1);}
    if (ioperm(base,1,1)) {
        fprintf(stderr, "Access denied to port at %x\n", base);
        exit(1);}

       outb((unsigned char)value, base); //send value to data port
    }
```

Save the above source code as namely `writelpt.c` and compile it using the command :

gcc -O writelpt.c -o writelpt

In order to make the script executable by all users, one should change the ownership by :

chmod +s writelpt

After these procedures, everyone can run the script. For example running ***./writelpt 255*** will send 255 or 0xFF value to the data port address.

3.1.2 Parallel port setting to input in nibble mode

Parallel port consists of three port addresses, each with its own specific function. For a parallel port resides at *0x378* address, it has status port address in *base+1* or *0x379* address, and control port address in *base+2* or *0x37A* address. In some newer mainboards, data or base address can be used for both reading or writing the data that would simplify the design of control and monitoring system over parallel port.

In the current work, we use base address as output port even though we could also use this port as an input port. However, using this port address in two conditions at the same time requires more additional circuits such as multiplexer and latch register, and also it might cause conflict among incoming and outgoing data.

To avoid this problem, we set the port in nibble mode by using another port address instead of using base address. One of the ports that can be used to input the data is status port address. Status port address as mentioned before resides in *base+1* or *0x379* address which has five input pins. Nibble mode uses a Quad 2 line to 1 line multiplexer (74LS157) to read a nibble of data at a time. The 74LS157 acts as if four switches, when the A/B input is low, the A inputs are selected and vice versa. The Y outputs from 74LS157 are connected to status port, in such a manner that it represents the nibble of the status register. Thereafter, we use a software based solution to construct those two nibbles into a byte.

In the circuit shown in Fig. 3.3, first of all one must decide whether the multiplexer reads either inputs A or B [2]. If one tries to read the low nibble first, we must place A/B low by setting low Bit 0 of the control port on Pin 1.

```
outb (inb(0x37A) | 0x01, 0x37A);
```

After the low nibble is selected, we can read and store the low nibble on variable *input* from the status port, and then we AND the result with 0xF0 and shifts the results right 4 bits.

```
input = (inb(0x379) & 0xF0);
input = input >> 4;
```

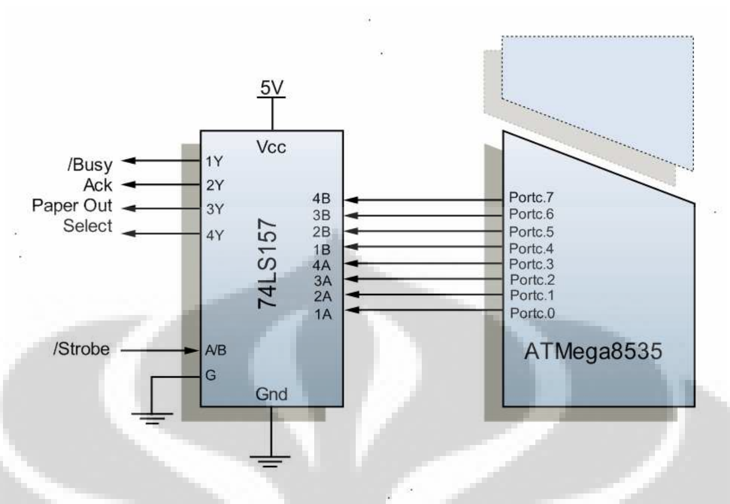


Figure 3.3: 74LS157 Multiplexer.

To read the high nibble, switches the multiplexer to select inputs B. Then one can read the high nibble and put the two nibbles together to make a byte,

```
outb (inb(0x37A) | 0xFE, 0x37A);
input = input | (inb(0x379) & 0xF0);
byte = byte ^ 0x88;
```

The last command is toggling the two inverted bits which were read on the Busy line.

3.2 Microcontroller Interfacing

Microcontroller is the most important part of this design, it translates or decodes the command that is sent by main modules into appropriate action both driving an actuator and reading the data from sensors. To avoid the design from error collision especially when the microcontroller drives an actuator or reads the data from sensors at the same time, we separate the microcontroller into two categories. The first one is microcontroller that has the responsibility for driving an actuator and the second one is responsible for reading the data. Tab. 3.2 is the microcontroller commands sent by main module:

Table 3.2: Command Function

Command (in decimal)	Action	Command (in decimal)	Action
1	read sensor at channel 0	9	read compass module
2	read sensor at channel 1	10	read counter 0
3	read sensor at channel 2	11	read counter 1
4	read sensor at channel 3	20	forward
5	read sensor at channel 4	21	backward
6	read sensor at channel 5	22	left
7	read sensor at channel 6	23	right
8	read sensor at channel 7	24	stop

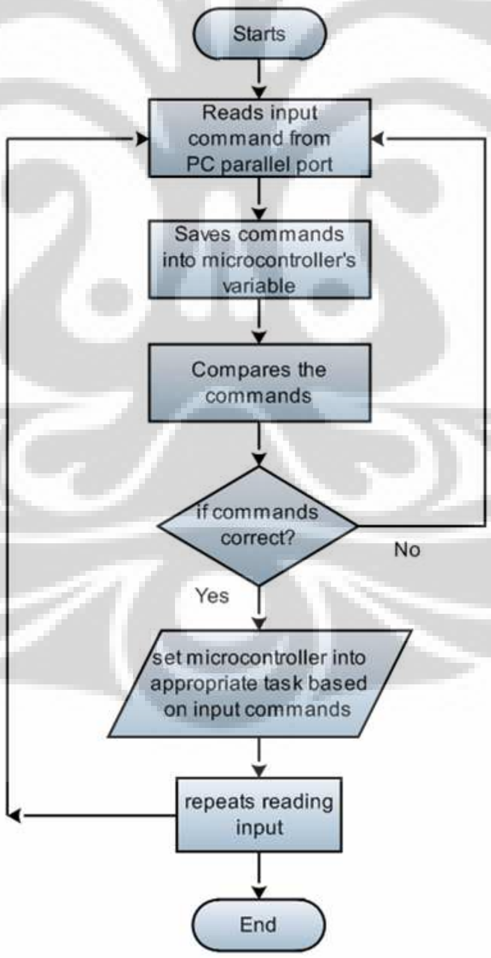

Starts
Reads input command from PC parallel port
Saves commands into microcontroller's variable
Compares the commands
if commands correct?
No
Yes
set microcontroller into appropriate task based on input commands
repeats reading input
End

Figure 3.4: A flowchart representing the microcontroller program.

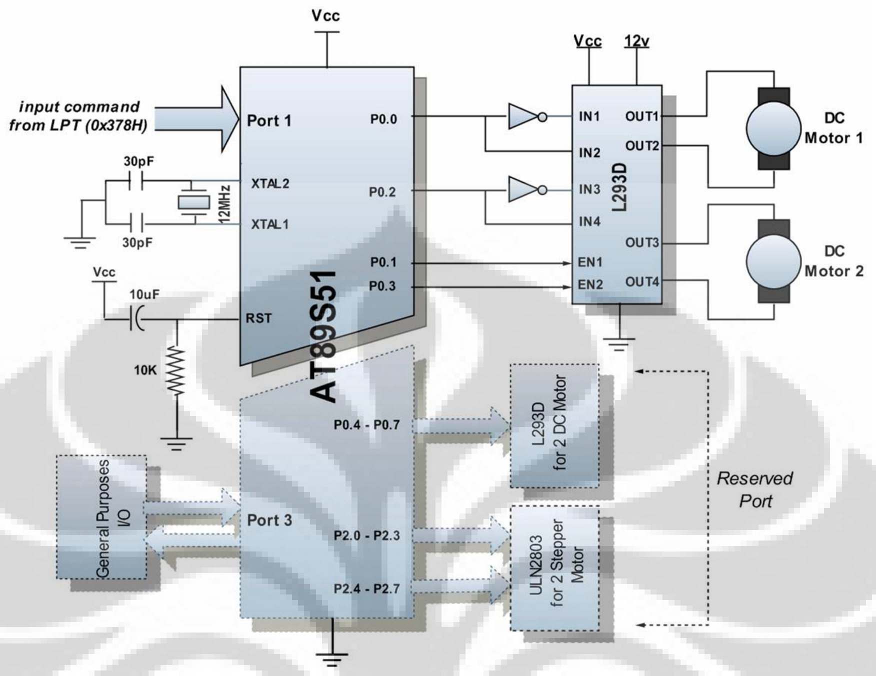


Figure 3.5: Motor Driver using AT89S51

Table 3.3: Truth Table for Controlling DC motor

IN1	IN2	Description
0	0	Motor stops or breaks
0	1	Motor runs anti-clockwise
1	0	Motor runs clockwise
1	1	Motor stops or breaks

3.2.1 DC Motor Controlling

The block diagram for controlling the actuator is given in Fig. 3.5. As shown in Fig. 3.5 for AT89S51 microcontroller. It is responsible not only for receiving all incoming commands sent from main module, but also for controlling all actuators as DC and stepper motors.

As the robot is purposed to be as modular as possible, more I/Os are required to satisfy the requirements. In the above circuits, the interface is designed for totally 4 DC motors, 2 stepper motors and one byte additional general purposes I/O port. However in the real prototype, only two DC motors are used for two independent wheels. The rest I/Os can be used for future expansion like controlling the web-cam using stepper motor and so on.

The LMNR [19] has the ability to move on a flat space in any directions (moving forward and moving backward, turning left and right). This is realized by deploying H-bridge circuits to drive each DC motor in two directions. Therefore, combining two independent DC motors would enable free moving easily. As shown in Fig. 3.5, the H-bridge is built using IC L293D. L293D consist of dual H-Bridge motor driver and the diodes inside to protect the circuit from EMF.

Basically, controlling each DC motor needs three inputs of L293D governed by AT89S51 microcontroller. However, using inverter logic IC such as 74LS04 can reduce a number of pins controller. So, only pins 0 and 1 of Port0 are used to control one DC motor. Tab. 3.2.1 shows how to control DC motor using L293D as dual H-Bridge motor driver. The condition in the table is realized by setting the pin (EN1 or EN2) to be (1).

3.2.2 Sensor Compass Reading

Navigation is another important aspect for mobile system like robot to catch up the current position and direction. One of the basic devices in navigation is compass. In LMNR a digital compass module called as CMPS03 (Fig. 3.6) is used. The compass module has been specifically designed for robotic system by generating a unique number to represent the direction against the North Pole.

The compass module is embedded with the Philips KMZ51 magnetic field sensor which is sensitive enough to detect the Earths magnetic field. Two magnetic sensors are mounted at both facing angles to obtain the direction of the horizontal component of the Earths magnetic field. There are two ways of getting the bearing from the module [5]. A PWM signal is available on pin 4, or an I2C interface is provided on pins 2 and 3. To get the data using I2C interface, an I2C protocol is needed. I2C bus is an abbreviation for Inter Integrated Circuit bus. It is also known as IIC. I2C is a serial and synchronous bus protocol. In a simple I2C system, there can only be one master with multiple slaves. The difference

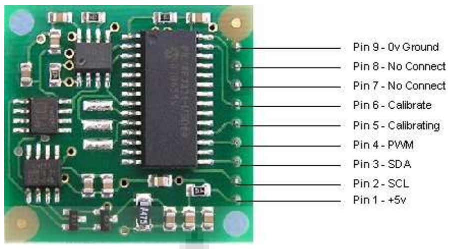


Figure 3.6: CMPS03 Compass Module

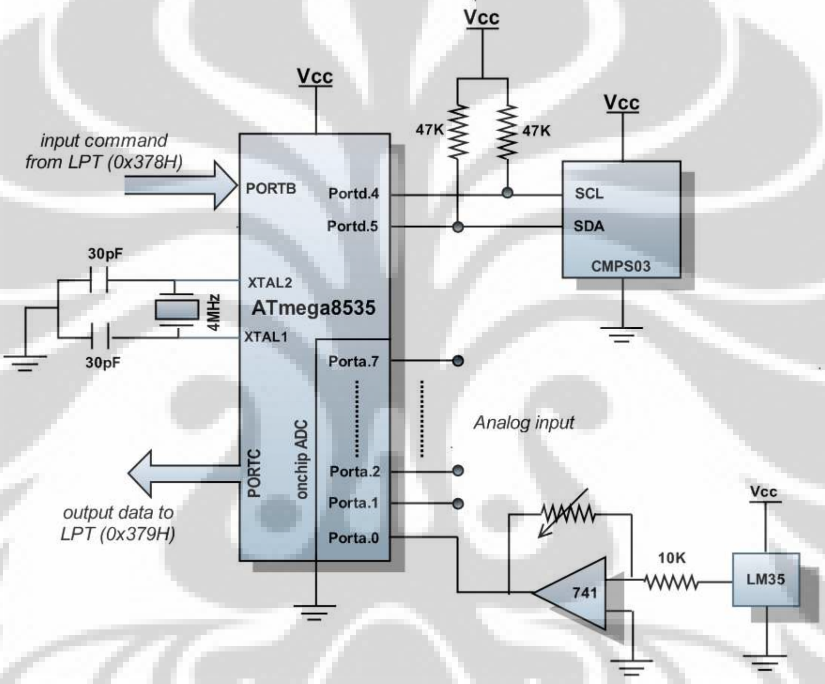
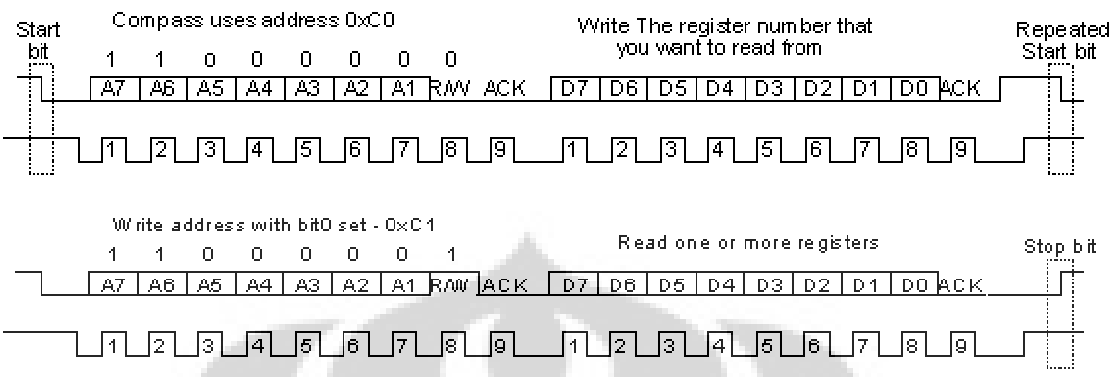


Figure 3.7: Interfacing ATmega8535 with CMPS03

between master and slave is the master generates the clock pulse and defines when the communication should occur. A typical hardware configuration for interfacing CMPS03 compass module using I2C interface is shown in Fig. 3.7. To read angles using I2C communication protocol with a compass module, first send a start bit and the module address (0XC0) with the read/write bit low, and followed with a register number being read. This is again followed by a repeated start and the module address with the read/write bit high (0XC1). Finally, one or two bytes for 8bit or 16bit registers can be read respectively. This communication timing is depicted in Fig. 3.8. The CMPS03 compass module has a 16 byte array of



Start bit
Compass uses address 0xC0
1 1 0 0 0 0 0 0
A7 A6 A5 A4 A3 A2 A1 R/W ACK
Write The register number that you want to read from
D7 D6 D5 D4 D3 D2 D1 D0 ACK
Repeated Start bit
1 2 3 4 5 6 7 8 9
Write address with bit0 set - 0xC1
1 1 0 0 0 0 0 1
Read one or more registers
Stop bit

Figure 3.8: I2C communication Protocol

Table 3.4: CMPS03 Array Register

Register	Function
0	Software Revision Number
1	Compass Bearing as a byte, i.e. 0-255 for a full circle
2, 3	Compass Bearing as a word, i.e. 0-3599 for a full circle
4, 5	Internal Test - Sensor1 difference signal - 16 bit signed word
6, 7	Internal Test - Sensor2 difference signal - 16 bit signed word
8, 9	Internal Test - Calibration value 1 - 16 bit signed word
10, 11	Internal Test - Calibration value 2 - 16 bit signed word
12	Unused - Read as Zero
13	Unused - Read as Zero
14	Unused - Read as Undefined
15	Calibrate Command - Write 255 to perform calibration step.

registers, some of them double up as 16 bit registers shown in Tab. 3.2.2.

3.2.3 Wheel Rotation Counter

The distance or wheel rotation can be measured by using pulse counter. There are several ways to count the pulse or clock, such as using counter IC CD4040, Timer/Counter internal register and External interrupt pin of microcontroller. In this design, the external interrupt INT0 and INT1 on PORTD bit 2 and bit 3 can serve as an external interrupt source to the ATmega8535 microcontroller [9], as can be seen in Fig. 3.9. The input pulse can be from any number of sources, and the most common and easiest to implement are mechanical and optical switch. However due to bouncing effect of a mechanical switch, the optical switch

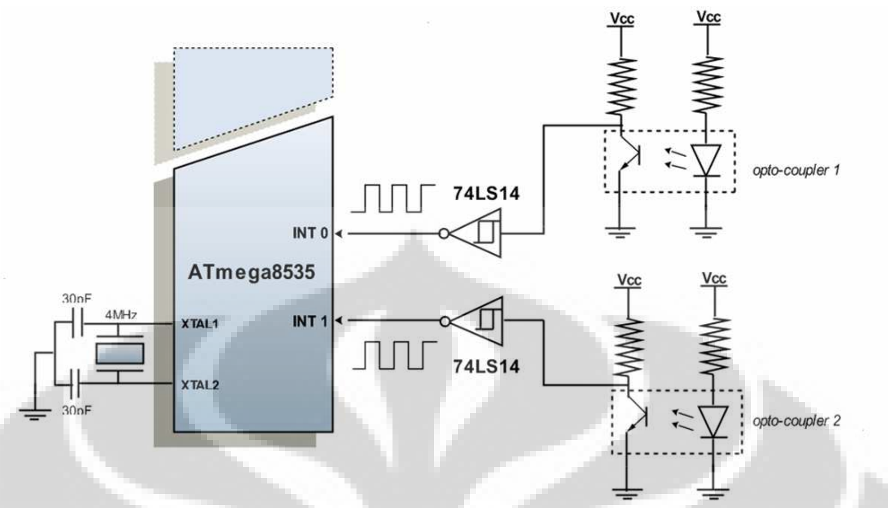


Figure 3.9: Wheel Rotation Counter

is preferable for this design.

The pulse or clock produced by the wheel rotation is proportional to the distance. Before the microcontroller is ready to count the distance, the pulse produced by opt coupler is feed to 74LS14 Schmitt trigger IC. Finally, this outputs pulse can be directly counted by microcontroller through external interrupt INT0 and INT1 pin. In software side, the interrupt method is used rather than polling methods. The pseudo-code in BASIC to count the pulse produced by the wheel rotation can be seen below:

```
Dim Cntr0 as Integer
Dim Cntr1 as Integer

On Int0 Counter0
On Int1 Counter1

Enable Interrupts
Enable Int0
Enable Int1

    Do
      ...
      ...
    Loop
End

Counter0:               'external interrupt 0 routine
    Waitms 100
    Incr Cntr0
Return

Counter1:               'external interrupt 1 routine
    Waitms 100
    Incr Cntr1
Return
```

3.3 CGI Programming

Modular networked robot is able to be accessed over the web interface. This enables the users to access the whole system wherever they are as long as they can connect to the network. The main advantage of this approach is it requires only a web browser in the user side, while all softwares including web server and others programming languages as PHP, C and Java completely embedded inside the main robot. So, the users needs only a web browser to both monitor and control everything, and no need for software installation at the client's side.

To realize full access to the whole system through web, especially to control and acquire the data from hardwares attached to the robot, there should be an interface between main unit and data acquisition module. The solution is using CGI as depicted in Fig. 3.10. CGI or Common Gateway Interface is a programme interface over HTTP server to receive requests from client browsers, perform specific processes and return the information back to the user through Hypertext Markup Language (HTML) pages [8].

Generally, people use CGI programs to perform any type of processing either directly or indirectly by accessing servers. In this section, the CGI program is used to access hardwares over web. Under Linux environment it requires: Linux as operating system, web server software (Apache is the most popular one), web interface and parallel port controlling CGI-BIN script as already written in the preceding section.

Basically, we can use any programming languages to build CGI application, the most popular programming languages which is used in CGI are *AppleScript (Macintosh Only), C/C++ (UNIX, Windows, Macintosh), C Shell (UNIX Only),*

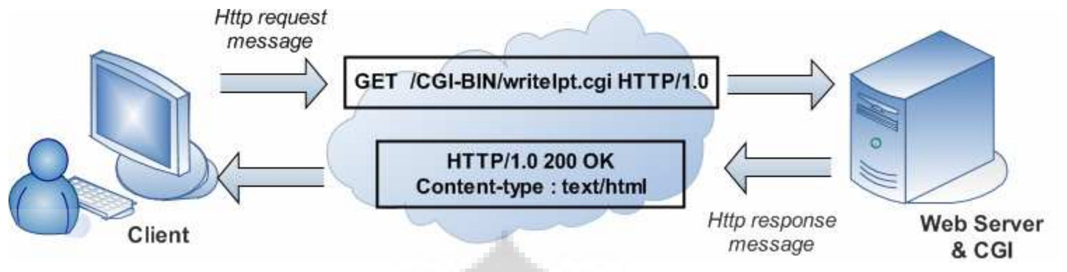


Figure 3.10: CGI Application

Perl (UNIX, Windows, Macintosh), *Tcl (UNIX Only)* and *Visual Basic (Windows Only)*. In this time, we are going to use *C shell* due to its simplicity and capability for calling other programs. Below is an example of CGI application to send the data to the parallel port over web. Firstly is a CGI script called *writelpt.cgi* which is responsible for calling *writelpt.c* source code to send data *0xff* data to the parallel port [3].

```
#!/bin/sh
# CGI script for sending the data
# to the parallel port
#
# tells the browser by sending HTTP headers
echo "Content-type: text/html;charset=ISO-8859"
echo
....
echo
# execute the command
/usr/sbin/writelpt 0xff
# this page will be viewed on browser
echo "<html><head></head><body>"
echo "Parallel port controlling over web<br>"
# we may add other html syntax here as we wish
echo
echo
....
....
echo
echo "<a href=\"/index.html\">Go back to main page</a>"
echo "</body></html>"
```

Secondly, the html source code is responsible for displaying the data to the users over a browser. As can be seen from the source code, the browser will send request "method=get" to the server to execute CGI script called *writelpt.cgi* when the user hits submit button [3].

```
<html>
<head>
       <title>web-based control</title>
</head>
<body>
  <center>
      <h1>Parallel port controlling test over web</h1>
      <h2>Sending 0xFF data to the parallel port</h3>
      <p>
         <form action="/cgi-bin/writelpt.cgi" method="get" name="on">
         <input type=submit value="Send...">
         </form>
      <p>
  </center>
</body>
</html>
```

Chapter 4

The Result

In this chapter, the whole results including the robot prototype, web-based user interface and algorithms for programming the microcontroller are presented.

4.1 The Robot Prototype

In LMNR [19], several devices and software are needed to build the whole prototype. Those components, either hardwares or softwares, are well integrated into a system. To build the overall systems, first of all the global architecture is made as shown in Fig. 4.1. This system contains main devices such as mainboard , wireless access point, web camera, microcontroller, and other electronic components including digital compass module and wheel encoder.

Each device in Fig. 4.1 has a specific function. The wireless access point is responsible for transmitting and receiving the data especially when a client sends a request to the server and vice versa. The motherboard is used as a processor unit responsible not only for web server but also for storing all embedded software. This module acts as a master controller to control and monitor the robot's actuators and sensors.

Beside that, the second microcontroller is attached for different tasks. The first one is AT89S51 that is responsible for controlling the DC motor as a wheel controller, and the second one is ATmega8535 for measuring or acquiring the data from the sensors. For the wheel counter, opt coupler is used to calculate how far the robot has traveled. To measure the direction the digital compass module

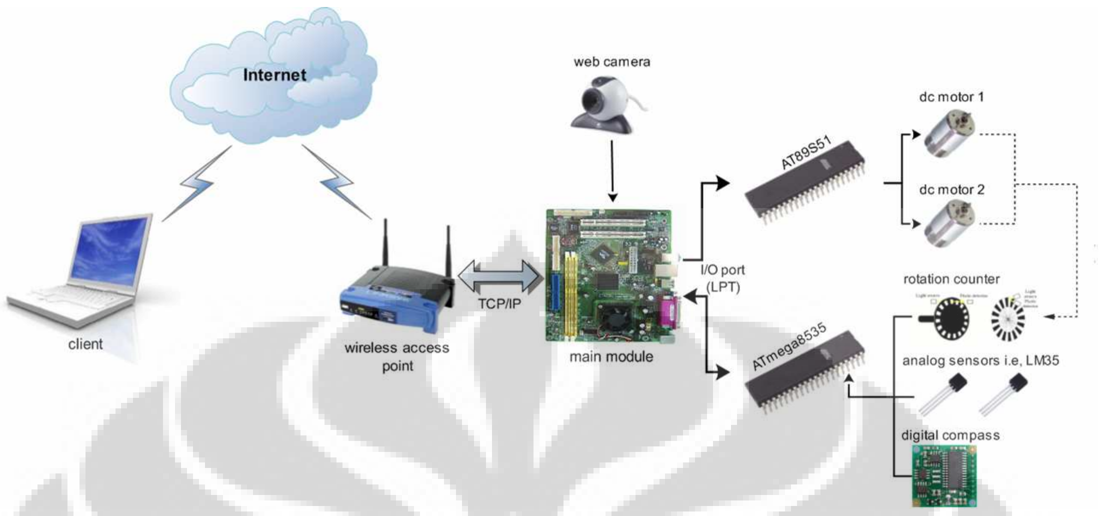


Figure 4.1: Diagram Schematic.

CMPS03 is deployed. This module produces the linear data based on the actual robot's angle position to the North Pole. The prototype can be seen in Fig. 4.2.

For the software, there are some softwares required to build the prototype. It is ranging from the software for main module until another one for programming the microcontrollers. In the current prototype, the following softwares are required:

- Linux Operating system (Ubuntu).
- Apache web server software.
- Camserv server software for video streaming over web.
- Web browser for the user interface web pages.
- CGI-BIN script.
- C/C++ compiler for accessing the hardware port (GCC).
- Some additional softwares (JAVA, PHP, C, etc).
- Microcontroller compiler (BASCOM-8051, BASCOM-AVR).

4.2 Web-based User Interface

Data retrieval from the sensors and its resulted analysis are displayed over web browser. Fig. 4.3 shows the web-based user interface for controlling the robot and

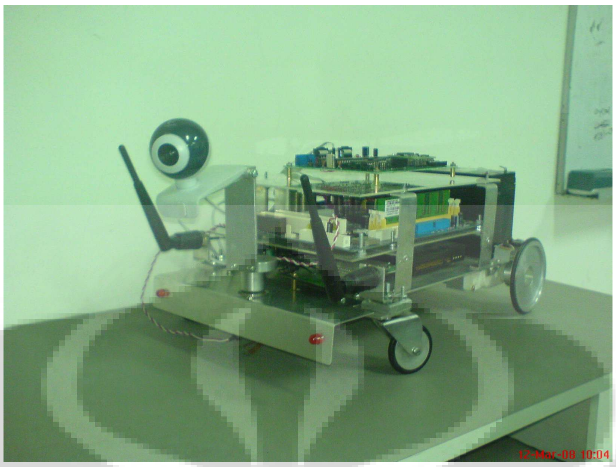


Figure 4.2: Robot Prototype.

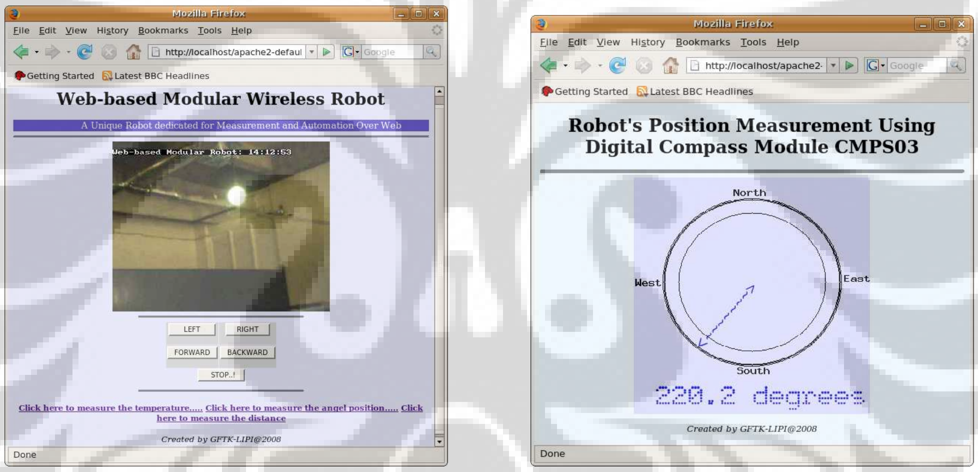
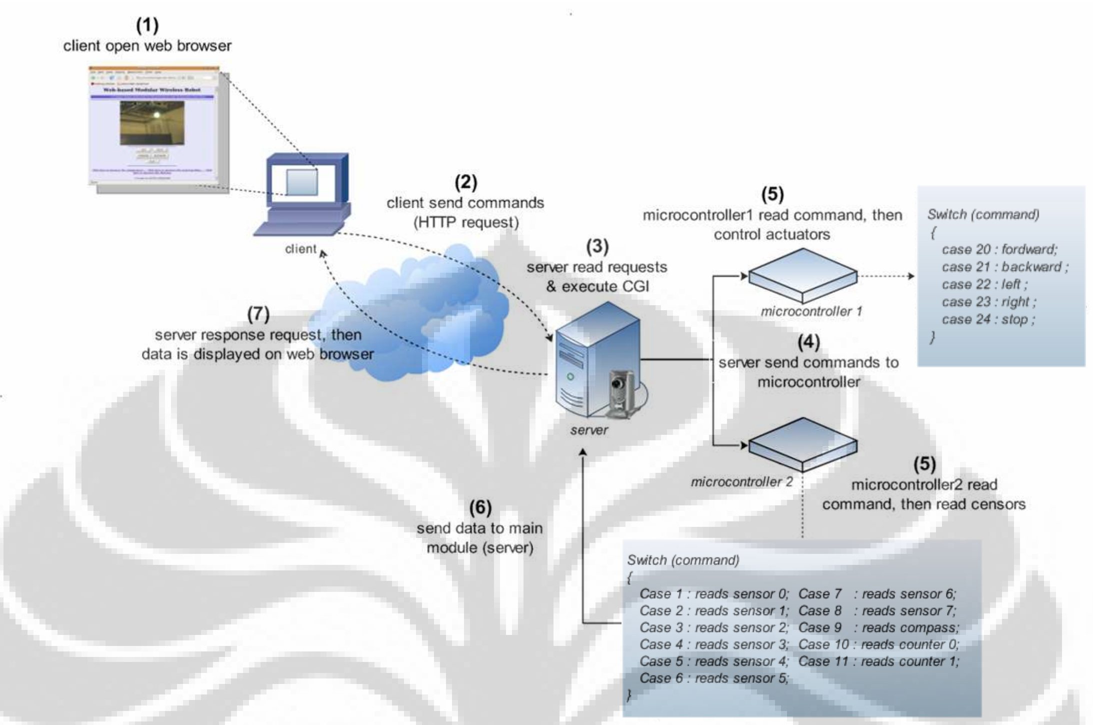

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/apache2-defaul
Getting Started Latest BBC Headlines
Web-based Modular Wireless Robot
A Unique Robot dedicated for Measurement and Automation Over Web
Web-based Modular Robot: 14:12:53
LEFT
RIGHT
FORWARD
BACKWARD
STOP..!
Click here to measure the temperature..... Click here to measure the angel position..... Click here to measure the distance
Created by GFTK-LIPI@2008
Done
Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/apache2-
Getting Started Latest BBC Headlines
Robot's Position Measurement Using Digital Compass Module CMPS03
North
West
East
South
220.2 degrees
Created by GFTK-LIPI@2008
Done

Figure 4.3: Web-based control and Compass measurement screen shoot

measuring the angle from digital compass module.

4.3 The Flow of Messages

The data flow between client and main module or server is in the following way. First, the client downloads a main web pages from the web server on the main module (the robot). Then the client's web browser displays web page containing all relevant information. Finally, the users send particular requests through the web interface which will be sent over network back to the main unit of robot. These procedures triggers the commands to be executed on the robot over web.


(1)
client open web browser
client
(2)
client send commands (HTTP request)
(3)
server read requests & execute CGI
(4)
server send commands to microcontroller
(5)
microcontroller1 read command, then control actuators
microcontroller 1
server
microcontroller 2
(5)
microcontroller2 read command, then read censors
(6)
send data to main module (server)
(7)
server response request, then data is displayed on web browser

```
Switch (command)
{
  case 20 : fordward;
  case 21 : backward ;
  case 22 : left ;
  case 23 : right ;
  case 24 : stop ;
}
```

```
Switch (command)
{
  Case 1 : reads sensor 0;  Case 7  : reads sensor 6;
  Case 2 : reads sensor 1;  Case 8  : reads sensor 7;
  Case 3 : reads sensor 2;  Case 9  : reads compass;
  Case 4 : reads sensor 3;  Case 10 : reads counter 0;
  Case 5 : reads sensor 4;  Case 11 : reads counter 1;
  Case 6 : reads sensor 5;
}
```

Figure 4.4: Flow connection between client and main module

At this point, the server is ready to pass the commands to microcontroller. Finally, The microcontroller then translate each command into appropriate task such as turning right or left, moving forward or backward or even read the data from the sensors. All feedbacks are sent back to the server and displayed on the client's web browser. The process flow is depicted in Fig. 4.4.

4.4 Measuring the Distance

The distance taken by robot movement can be counted using opt coupler attached on each wheel. Counting the wheel in straight direction is easier than in curve direction. First, knowing the wheel diameter and the degree of each transition level are important. The transition level is the change in binary level from 1 to 0 and vice versa which is used by external interrupt pins (INT0 and INT1) of ATmega8535 as a sign to count the wheel rotation.

Based on Fig. 4.5 in LMNR the wheels are divided into eight transition levels.

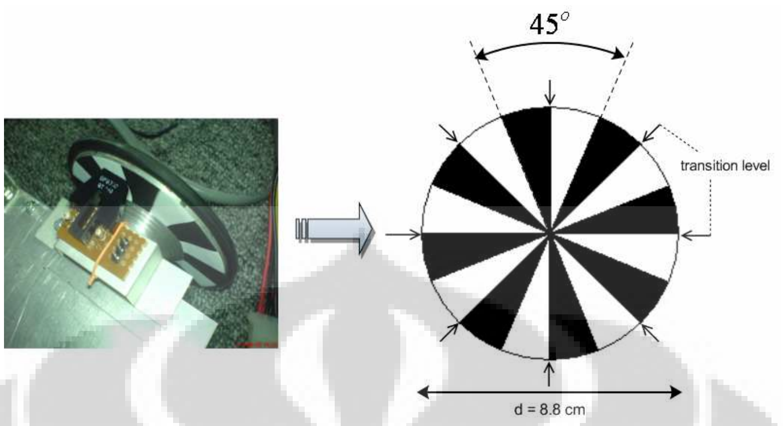


Figure 4.5: Wheel Encoder

This means one rotations is equal to the counted eight transition level. Mathematically, it satisfies the following equation:

$$d_L = \frac{CountingValue}{8}.\pi.d \tag{4.1}$$

In the case of movement on a curve or circle line, one can consider two special cases. One is when only left wheel is moving, while the other one is stopping. In this case the robot turn right/left depending on which wheels is stopping. The distance can be calculated if one knows the passed angle in a single movement. As mentioned before, this can be done by using digital compass module CMPS03. Then the distance is expressed as below [10]:

$$\theta = \frac{(d_L - d_R)}{A} \tag{4.2}$$

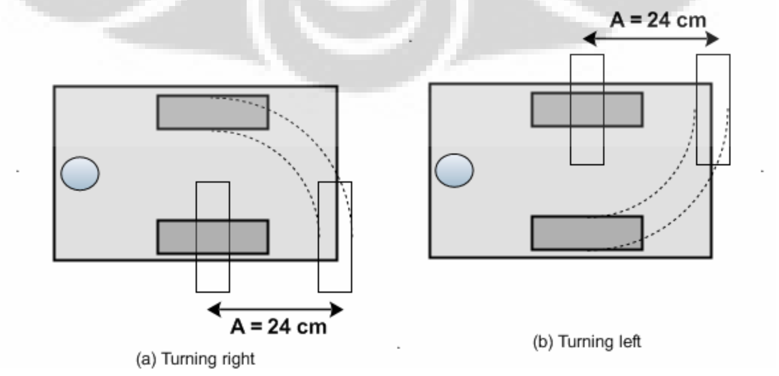


Figure 4.6: Turning Position

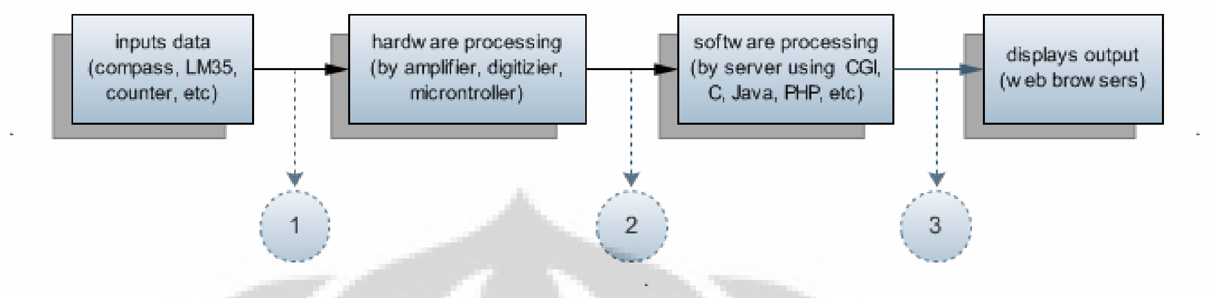

inputs data
(compass, LM35, counter, etc)
hardware processing
(by amplifier, digitizier, microntroller)
software processing
(by server using CGI, C, Java, PHP, etc)
displays output
(web browsers)
1
2
3

Figure 4.7: Flow of data conversion

4.5 Measurement Results

Tabs. 4.5 and 4.4 show the measured data from the censors (compass CMPS03 and temperature censor LM35 as an example) in the binary data which is processed by microcontroller and its internal A/D converter. Tab. 4.4 shows microcontroller's pin condition when the command is received and executed by microcontroller with the actual result is shown for each command. Again, those data are captured by main module (server) and displayed on the web browser.

Table 4.1: Compass data conversion

(1) Angle (to north pole)	(2) Data (in decimal)	(3) Output value on server
0^o	0	0^o
40^o	29	40^o
90^o	64	90^o
130^o	92	129^o
180^o	128	180^o
220^o	156	220^o
270^o	191	269^o
310^o	220	310^o
360^o	255	360^o

Table 4.2: Temperature data conversion

(1)	(2)			(3)
Temperature from $LM35$	V_{in} ($volts$)	Amplified Voltage ($gain = 5X$)	Data (in decimal)	Output value on server
29^oC	0.29	1.45	74	29^oC
31^oC	0.31	1.55	79	30^oC
35^oC	0.35	1.75	89	34^oC
38^oC	0.38	1.9	97	38^oC
42^oC	0.42	2.1	107	41^oC
45^oC	0.45	2.25	115	45^oC
47^oC	0.47	2.35	120	47^oC
50^oC	0.5	2.5	128	50^oC
53^oC	0.53	2.65	135	52^oC

Table 4.3: Microcontroller's Pin Condition

Input Commands in decimal	AT89S51 pin condition	ATMega8535 pin condition	Results Robot Condition
20	P0.0 = P0.2 = 1	-	moving fordward
	P0.1 = P0.3 = 1	-	
21	P0.0 = P0.2 = 1	-	moving backward
	P0.1 = P0.3 = 0	-	
22	P0.0 = P0.1 = 1	-	turning left
	P0.2 = P0.3 = 0	-	
23	P0.2 = P0.3 = 1	-	turning right
	P0.0 = P0.1 = 0	-	
24	P0.0 = P0.2 = 0	-	stop
1	-	Porta.0 = ADC	reads ADC channel 0
2	-	Porta.1 = ADC	reads ADC channel 1
3	-	Porta.2 = ADC	reads ADC channel 2
4	-	Porta.3 = ADC	reads ADC channel 3
5	-	Porta.4 = ADC	reads ADC channel 0
6	-	Porta.5 = ADC	reads ADC channel 1
7	-	Porta.6 = ADC	reads ADC channel 2
8	-	Porta.7 = ADC	reads ADC channel 3
9	-	Pord.4 = Scl	reads compass
	-	Portd.5 = Sck	CMPS03
10	-	Portc = Cntr0	reads counter 0
11	-	Portc = Cntr1	reads counter 1

Chapter 5

Discussion

A prototype of mobile, modular and networked monitoring system which is controllable over web has been developed. The system is divided into three modules to realize the modularity and to improve its flexibilities. The modularity both in hardware and software enables partial expansions or replacements according to the users needs.

Three modules are the main module, data acquisition module and data processing module. In this architecture, the main module consists of main board, wireless access point and the robot's body itself. The data acquisition module consists of several sensors which is responsible for acquiring some physical observables from the neighboring environment as temperature, humidity and so on. The last module is for data processing module, and it consists of software-based solution to process and analyze the data. For instance, a package of environment monitoring system would process the acquired surrounding temperature and humidity and analyze it for a particular purpose. On the other hand, it is also responsible for processing the signal from the sensors and also providing the current position and direction by performing dead-reckoning-based calculation. In this sense, the module is also replacing the hardware-based components with some software-based peripherals.

However, the prototype is intended mostly for a battle field to implement the whole concept of a modular networked robot. More efforts are still required to realize more sophisticated modular monitoring system. Some of them are: using a real embedded PC instead of conventional mainboard to reduce the size and power consumption. Also, more comprehensive data processing module, especially the improved digital signal processing.

In the hardware communication, one could also use another I/O port for communication channels between main module and data acquisition module such as USB and serial port. These might be better alternatives instead of parallel port. Moreover, we could also use the interrupt methods in both microcontroller and computer programming. In this method, the microcontroller polls the data that would slow the whole speed. As the result, there is only one executed task by microcontroller each time. For instance, when the user is controlling the robot direction, they can not measure the data from the sensor and vice versa. However, this would still be quick enough to respond the incoming commands from the users.

Bibliography

[1] Dhananjay V. Gadre, *Programming the Parallel Port: Interfacing the PC for Data Acquisition and Process Control*, R&D Books (1998)

[2] Craig Peacock, *Interfacing the Standard Parallel Port*, (1998)

[3] Tomi Engdahl, *Parallel port interfacing made easy: Simple circuits and programs to show how to use PC parallel port output capabilities*, (2006)

[4] Thomas Stork, *Electronic Compass Design using KMZ51 and KMZ52*, Philips Semiconductor (2000)

[5] Gerald Coe, *CMPS03 - Robot Compass Module*, (2007)

[6] Thomas Bräunl, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*, Springer (2006)

[7] Gordon McComb, Myke Predko, *Robot Builders Bonanza*, McGraw-Hill (2006)

[8] Shishir Gundavaram, *CGI Programming on the World Wide Web*, O'reilly (1996)

[9] Claus Kuhnel, *BASCOM Programming of Microcontrollers with Ease*, Universal Publishers (2001)

[10] Tim Wilmshurst, *Designing Embedded Systems with PIC Microcontrollers*, Newnes (2007)

[11] Jim Geier, *Wireless Networks first-step*, Cisco Press (2004)

[12] Michael F. Finneran, *Voice Over WLANs*, Newnes (2007)

[13] Michael J.Donahoo, Kenneth L.Calvert, *TCP/IP Sockets in C: Practical Guide for Programmers*, Morgan Kaufmann Publishers (2001)

[14] Halit Eren, *Wireless Sensors and Instruments*, CRC Press (2006)

[15] G.W. Lucas, *A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators*, (2001)

[16] Ibrahim Kamal, *WFR, a Dead Reckoning Robot*, (2007)

[17] I. Firmansyah, B. Hermanto and L.T. Handoko, *Control and Monitoring System for Modular Wireless Robot*, Proc. of the Industrial Electronics Seminar 2007, Surabaya, Indonesia, 2007.

[18] LIPI Wireless Robot, *http://robot.teori.fisika.lipi.go.id.*

[19] LIPI Networked Robot, *http://opennr.teori.fisika.lipi.go.id.*

Appendix A

Program Listing

```
'-----------------------------------------------------------------------------------
' Source Code for ATMEGA8535 microcontroller
' By Iman Firmansyah (2008)
' Created with BASCOM-AVR: 1.11.8.1
' BASCOM-AVR is Copyrights by MCS Electronics
'-----------------------------------------------------------------------------------

$regfile = "m8535.dat"
$crystal = 4000000

Config Portb = Input                             ' input command from PC
Config Portc = Output                            ' output data to PC
Config Scl = Portd.4                             ' scl CMPS03
Config Sda = Portd.5                             ' sda CMPS03
Config Int0 = Rising                             ' external interrupt 0
Config Int1 = Rising                             ' external interrupt 1
Config Debounce = 50
Config Adc = Single , Prescaler = Auto , Reference = Internal

Declare Sub Convert_adc(channel As Byte)
Declare Function Read_cmps03() As Byte
Declare Function Rn_cmps_firmware() As Byte

Dim Data_in As Byte
Dim Data_out As Byte
Dim Analog_in As Integer
Dim Degree As Byte
Dim Channel As Byte
Dim Cntr0 As Integer
Dim Cntr1 As Integer
Const Debouncetime = 50

Enable Interrupts
Enable Int0
Enable Int1
```

```
On Int0 Button0
On Int1 Button1

   Waitms 10
   I2cinit
   Do
         Data_in = Pinb
         Select Case Data_in
            Case 1 :                           ' read sensor at channel 0
               Channel = 0
               Call Convert_adc(channel)
            Case 2 :                           ' read sensor at channel 1
               Channel = 1
               Call Convert_adc(channel)
            Case 3 :                           ' read sensor at channel 2
                Channel = 2
               Call Convert_adc(channel)
            Case 4 :                           ' read sensor at channel 3
               Channel = 3
               Call Convert_adc(channel)
            Case 5 :                           ' read sensor at channel 4
               Channel = 4
               Call Convert_adc(channel)
            Case 6 :                           ' read sensor at channel 5
               Channel = 5
               Call Convert_adc(channel)
            Case 7 :                           ' read sensor at channel 6
                Channel = 6
               Call Convert_adc(channel)
            Case 8 :                           ' read sensor at channel 7
               Channel = 7
               Call Convert_adc(channel)
            Case 9 :                           ' read compas module/cmps03
               Degree = Read_cmps03()
               Portc = Degree
            Case 10 :                          ' read counter 0
                Portc = Cntr0
            Case 11 :                          ' read counter 1
                Portc = Cntr1
        End Select
   Loop

   Button0:                                    ' external interrupt (INT 0)
      Incr Cntr0
      Waitms Debouncetime
      Gifr = 64
   Return

   Button1:                                    ' external interrupt (INT 1)
      Incr Cntr1
      Waitms Debouncetime
      Gifr = 64
   Return
End


Sub Convert_adc(channel As Byte)
    Start Adc
    Analog_in = Getadc(channel)
    Shift Analog_in , Right , 2
    Portd = Analog_in
    Stop Adc
End Sub

Function Read_cmps03() As Byte
   Local Lob As Byte
   Local Hib As Byte
   Local Cmps_slaveid As Byte
   Local Cmps_slaveid_read As Byte
   Cmps_slaveid = &HC0
   Cmps_slaveid_read = Cmps_slaveid + 1

   I2cstart
   I2cwbyte Cmps_slaveid
   I2cwbyte 1
   I2cstop

   I2cstart
   I2cwbyte Cmps_slaveid_read
   I2crbyte Lob , Nack
   I2cstop
   Read_cmps03 = Lob
End Function

Function Rn_cmps_firmware() As Byte
   Local Firmware As Byte
   Local Cmps_slaveid As Byte
   Local Cmps_slaveid_read As Byte
   Cmps_slaveid = &HC0
   Cmps_slaveid_read = Cmps_slaveid + 1

   I2cstart
   I2cwbyte Cmps_slaveid
   I2cwbyte 0
   I2cstop

   I2cstart
   I2cwbyte Cmps_slaveid_read
   I2crbyte Firmware , Nack
   I2cstop
   Rn_cmps_firmware = Firmware
End Function
```

```
'-----------------------------------------------------------------
' Source Code for AT89S51 microcontroller
' By Iman Firmansyah (2008)
' Created with BASCOM-8051: 2.0.11.0
' BASCOM-8051 is Copyrights by MCS Electronics
'-----------------------------------------------------------------
' P1    --->  Input command from PC
' P0    --->  Output control to DC motor
' P2    --->  Output control to Stepper motor
' P0.0  --->  Enable DC motor 1
' P0.1  --->  Output pin to DC motor 1
' P0.2  --->  Enable DC motor 2
' P0.3  --->  Output pin to DC motor 2
' P0.4  --->  Enable DC motor 3
' P0.5  --->  Output pin to DC motor 3
' P0.6  --->  Enable DC motor 4
' P0.7  --->  Output pin to DC motor 4
'-----------------------------------------------------------------

$regfile = "Reg51.dat"
$crystal = 12000000

Enable_dc1 Alias P0.0
Enable_dc2 Alias P0.2
Dc1 Alias P0.1
Dc2 Alias P0.3

Dim Input_cmd As Byte
Dim A As Byte

A = 3
Do
   Input_cmd = P1
    Select Case Input_cmd
      Case 20 :               'forward
         Enable_dc1 = 1
         Enable_dc2 = 1
         Dc1 = 1
         Dc2 = 1
         Wait A
         Enable_dc1 = 0
         Enable_dc2 = 0
      Case 21 :               'backward
         Enable_dc1 = 1
         Enable_dc2 = 1
         Dc1 = 0
         Dc2 = 0
         Wait A
         Enable_dc1 = 0
         Enable_dc2 = 0
      Case 22 :               'left
```

```
         Enable_dc1 = 1
         Enable_dc2 = 0
         Dc1 = 1
         Dc2 = 0
         Wait A
         Enable_dc1 = 0
         Enable_dc2 = 0
      Case 23 :              'right
         Enable_dc1 = 0
         Enable_dc2 = 1
         Dc1 = 0
         Dc2 = 1
         Wait A
         Enable_dc1 = 0
         Enable_dc2 = 0
      Case 24 :              'stop
         Enable_dc1 = 0
         Enable_dc2 = 0
    End Select
Loop
End
```

```
'-----------------------------------------------------------------------
' Source Code for reading the data through parallel port
' By Iman Firmansyah (2008)
' Created with GCC
'-----------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/io.h>

#define base 0x378           /* printer port base address */
#define status base+1        /* status port address */
#define control base+2       /* control port address */
#define value 255            /* numeric value to send to printer port */

main(int argc, char **argv)
{
   int input=0;
   if (ioperm(base,1,1))
   fprintf(stderr, "Couldn't get the port at %x\n", base), exit(1);

   if (ioperm(status,1,1))
   fprintf(stderr, "Couldn't get the port at %x\n", base), exit(1);

   if (ioperm(control,1,1))
   fprintf(stderr, "Couldn't get the port at %x\n", base), exit(1);

   // send 0x09 to read compass
   // this data can be changed based on command list table
   outb(0x09, base);
   usleep(5000);

   outb(inb(control)|0x01,control);
   input=(inb(status)&0xF0);
   usleep(100);
   input=input>>4;

   outb(inb(control) & 0xFE,control);
   input=input|(inb(status)&0xF0);
   usleep(100);
   input=(input^0x88);
   usleep(100);
   printf("%d",input);
}
```

```
'----------------------------------------------------------------------
' Source Code for displaying angle from CMPS03 over web browser
' By Iman Firmansyah (2008)
' Created with PHP 5 and Apache Web Server
'----------------------------------------------------------------------

<?php
header("Content-type: image/jpeg");
$im = @imagecreate(320, 340)
   or die("no GD!");

$output = shell_exec('/usr/sbin/inputlpt');
$a=0;
$degrees = ($output/255)*360;
$b=bcadd($a, $degrees, 1);
$c=($b+90);
$d=$c-(2*$b);

$background_color = imagecolorallocate($im, 220, 220, $output);
$text_color = imagecolorallocate($im, 0, 0, 255);
$line_color = imagecolorallocate($im, 0, 0, 0);

imagestring($im, 7, 135, 13, "North", $line_color);
imagestring($im, 7, 140, 270, "South", $line_color);
imagestring($im, 7, 285, 137, "East", $line_color);
imagestring($im, 7, 2, 143, "West", $line_color);
imageTTFtext($im, 19, $d, 160, 150,$text_color,
"/usr/share/fonts/truetype/arphic/Achilles.ttf","<~~~~~~~>");

imageTTFtext($im, 23, 0, 30, 325, $text_color,
"/usr/share/fonts/truetype/arphic/Atomic Clock Radio.ttf","$b degrees");

imageellipse($im, 160, 150, 200, 200, $line_color);
imageellipse($im, 160, 150, 238, 238, $line_color);
imageellipse($im, 160, 150, 240, 240, $line_color);
imageellipse($im, 160, 150, 243, 243, $line_color);

imagejpeg($im);
imagedestroy($im);
?>
```

```
'-------------------------------------------------------------------------
' CGI script for controlling the robot
' By Iman Firmansyah (2008)
'-------------------------------------------------------------------------

#!/bin/sh
# Parallel port CGI script
#
# Send HTTP headers
echo Content-type: text/html;charset=ISO-8859
echo
# Do the controlling
# Changes the value sent to LPT based on desired command
/usr/sbin/lptout1 2
# Output web page data
echo "<html><body bgcolor="#E6E6FA"><center>"
echo "<h1>Web-based Modular Wireless Robot</h1>"
echo "<FONT color="#ffffff" size="+1">"
echo "<marquee bgcolor="#6A5ACD" font-color="white">
A Unique Robot dedicated for Measurement and Automation Over Web
</marquee></FONT>"
echo "<hr noshade="noshade" size="3" width="100%">"
echo "<img src="http://10.100.100.188:8003" width="320" height="250">
</center>"
echo "<center><hr noshade="noshade" size="3" width="40%">"
echo "<table cellpadding="1" cellspacing="1" width="25%">"
echo "<tbody><tr><td bgcolor="#D3D3D3"><center>"
echo "<form action="/cgi-bin/kiri2.cgi" method="get" name="kiri">"
echo "<input value="     LEFT     " type="submit"></form></center></td>"
echo "<td bgcolor="#D3D3D3"><center>"
echo "<form action="/cgi-bin/kanan2.cgi" method="get" name="kanan">"
echo "<input value="   RIGHT   " type="submit"></form></center></td></tr>"
echo "<tr><td bgcolor="#D3D3D3"><center>"
echo "<form action="/cgi-bin/maju2.cgi" method="post" name="maju">"
echo "<input value="FORWARD" type="submit"></form></center></td>"
echo "<td bgcolor="#D3D3D3"><center>"
echo "<form action="/cgi-bin/mundur2.cgi" method="get" name="mundur">"
echo "<input value="BACKWARD" type="submit"></form></center></td></tr>
</tbody></table>"
echo "<center><form action="/cgi-bin/stop2.cgi" method="get" name="stop">"
echo "<input value="    STOP..!  " type="submit"></form></center>"
echo "<hr noshade="noshade" size="3" width="40%"><h4>"
echo "<a href="url" target="_blank">
Click here to measure the temperature.....</a>"
echo "<a href="url" target="_blank">
Click here to measure the angel position.....</a>"
echo "<a href="url" target="_blank">
Click here to measure the distance</a></h4>"
echo "<i> Created by GFTK-LIPI@2008</i></center></body></html>"
#
```

```
'------------------------------------------------------------------------
' Camserv's configuration file (Stream live video out onto the web)
' Camserv is a free program to do streaming video through the web.
' Camserv takes a video-for-linux video stream, generally from a camera,
' and streams it out live to requesting clients. Works with Mozilla,
' Netscape Navigator, and (under protest) Microsoft Internet Explorer.
' Includes camserv-specific dynamic library modules libjpg_filter,
' librand_filter, libtext_filter, libvideo_basic, libvideo_v4l.
'------------------------------------------------------------------------

# video_basic: The 'basic' color-changing video module.
    [video_basic]
    path /usr/lib/camserv/libvideo_basic.so.0

# video_v4l_bttv:  Example of a common BTTV module for video4linux.
# port 0 == TV, port 1 = Composite 1, port 2 = Composite 2
# frequency == is the channel frequency for the TV
# autobright == 0 turns off autobrightness adjusting, otherwise it adjusts
#               the brightness of the picture every 'autobright' frames.
# brightmean == The mean pixel value that is the 'goal' of the autobright.
#               (0->255)
# brightx1->y2 == (x1,y1) top left coords, and (x2,y2) bottom right coords
#                 of a rectangle of which to take the average pixel value.
#                 this is then used in calculating the mean to adjust the
#                 brightness of the image.
# mode == the video norm to use:  0 == PAL, 1 == NTSC, 2 == SECAM, 3 == AUTO
# color,hue,contrast,brightness,whiteness = 0->60000, representing
# the value of each component.

    [video_v4l_bttv]
    path /usr/lib/camserv/libvideo_v4l.so.0
    device_path /dev/video0
    port 0
    mode 3

#frequency 74.43
    color 30000
    hue 30000
    contrast 30000
    brightness 30000
    whiteness 30000
    autobright 1
    brightmean      128
    brightx1 0
    brighty1 320
    brightx2 0
    brighty2 240

# FreeBSD BTTV driver:
# port 0  = Video
#      1  = Tuner
# Channel Sets:
# nabscst   1
# cableirc  2
# cablehrc  3
# weurope   4
# jpnbcst   5
# jpncable  6
# xussr     7
# australia 8

    [video_fbsd_bttv]
    path /usr/lib/camserv/libvideo_fbsd_bttv.so.0
    port 1
    width 320
    height 240
    autobright 100

#brightness 0
#chroma 180
#contrast 1000
    channelset      2
    channel         60

    [video_v4l_qcam]
    path /usr/lib/camserv/libvideo_v4l.so.0
    device_path /dev/video1
    port 0
    color 30000
    hue 30000
    contrast 30000
    brightness 30000
    whiteness 30000
    autobright 0

    [jpg_filter]
    path            /usr/lib/camserv/libjpg_filter.so.0
    quality 30

# text_filters:  Text filters are used to provide text on your webcam
# bg,fg          == #RRGGBB if RGB camera, #CC if B&W camera,else 'transparent'
# x,y            == Coordinates for the text
# mangle         == 0 turns off mangling of the 'text', 1 turns it on
# text           == Text to display.  If mangling == 1, special metachars
#                   such as '%X' will be expanded -- see date(1) for lots of
#                   flags
# fontname       == 6x11 or 8x8 for the fontsize.

    [hello_world_banner]
    path            /usr/lib/camserv/libtext_filter.so.0
    text Hello World
    bg #000000
    fg #ffffff
    x 0
    y 0
    mangle 0
    fontname 6x11

    [time_stamp]
    path            /usr/lib/camserv/libtext_filter.so.0
    text Web-based Robot Time: %X
    bg #000000
    fg #ffffff
    x 0
    y 11
    mangle 1
    fontname 8x8

    [static_filter]
    path /usr/lib/camserv/librand_filter.so.0
    num_perline 20
    coloredpix 0
#
# You can add the imlib2_filter to your filters list to display pictures
# over your own, or small regions, or whatever your heart desires.
    [imlib2_filter]
    path /usr/lib/camserv/libimlib2_filter.so.0
    file /tmp/my_nasty_picture.png
    x 0
    y 0

##########################################
#  Begin Fixed Sections                  #
##########################################

# socket parameters:
# listen_port = port the camserv program listens on
#       max_frames  = Maximum # of frames to send to the client before
#                     closing the connection (0 disables)
#       max_bytes   = Maximum # of bytes to send to a client before
#                     closing the connection (0 disables)
#       max_seconds = Maximum # of seconds a client can be connected before
#                     being closed (0 disables)

    [socket]
    listen_port 9192
    max_frames 0
    max_bytes 0
    max_seconds 0

    [filters]
    num_filters           2
    filter0_section time_stamp
    filter1_section jpg_filter

# [video] - This section is devoted to all things dealing with the pictures
#           taken by the input video module.  These are general things which
#           should be used by all video modules.
#
#           IMPORTANT:  If you are seeing cycling colours instead of the
#                       video for your camera, you need to change video_basic
#                       to video_v4l_qcam or video_v4l_bttv

    [video]
    video_section video_v4l_bttv
    width 320
    height 240
    maxfps                  0
    memhack 1

    [main]
# To do a single time invocation of the output from the camserv,
# use output_snapfile which designates the output location, and
# output_presnaps to take a number of pictures before finally outputting
# the final image.
#output_snapfile foo.jpg
#output_presnaps 100
```