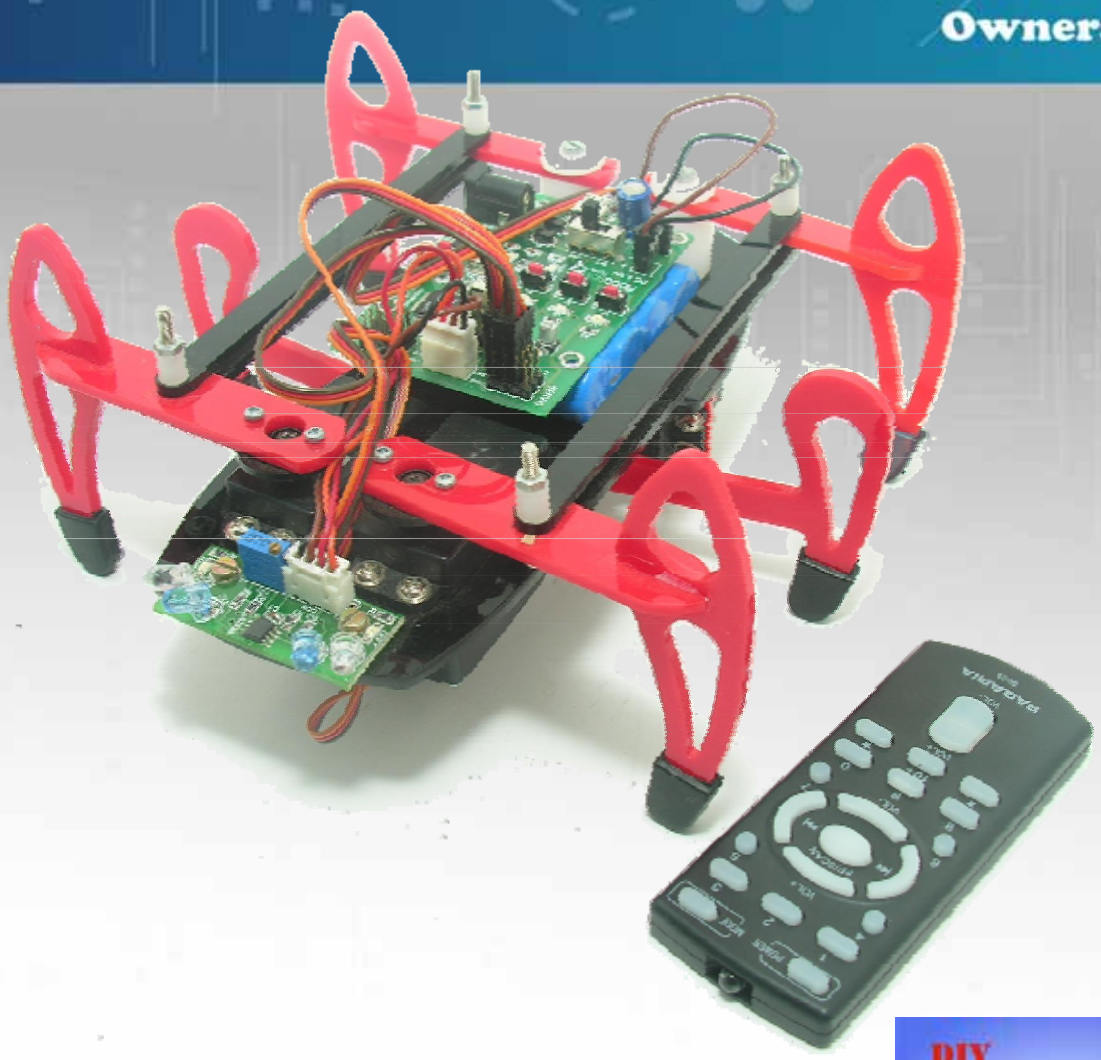
Robosoft Systems

HexaBOT

Owners Manual



Robosoft Systems

Skill Level:
Expert

Soldering
Not Required

4 hrs

Important note

This manual refers to the product HexaBOT. It can also be used in RoboCRAB (a workshop conducted by Robosoft Systems®). All the shown components will not be available if the product is bought in parts. Nevertheless, you can refer to the specific section as that of the parts that you have bought to know the assembly details. The components shown in this manual may not be the same as those in the packaging and may change without notice.

Please note that the manual is not comprehensive and cannot be considered as a complete reference for any theories presented here. Please refer to the compact disc that has accompanied the product for a more detailed overview of the various technologies and programming language that has been put to use in this product.

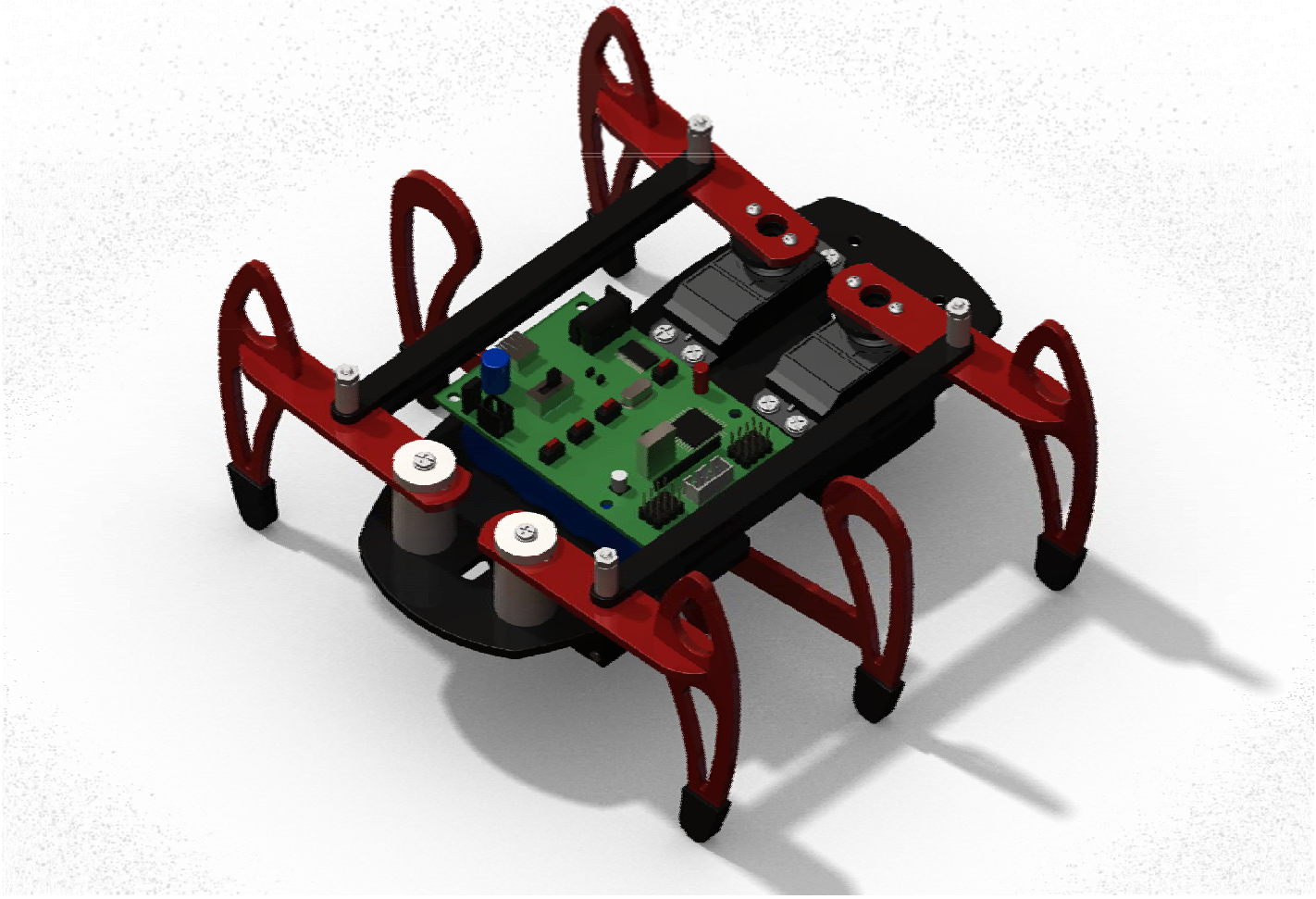
For further details and supported accessories, please visit our website http://www.robosoftsystems.co.in/roboshop.

HexaBOT features

- **Philips P89V51RD2 SMD µc on board.**
- **Autonomous robot.**
- **Completely mobile robot.**
- **ISP (In System Programming) support.**
- **Easy USB interface.**
- **On board provision for battery charging.**
- **Accessories: battery, adaptor, software cd.**
- **Compact size.**
- **Separate detachable TSOP based obstacle avoiding module.**
- **Three servo motors for accurate movement.**
- **6V DC battery for continuous operation.**
- **IR remote for wirelessly controlled motion.**

HexaBOT

Owner's Manual



HexaBOT is not a toy.

Robosoft Systems® employs high performance motors and electronic equipments.

Robosoft Systems® cannot be held liable for damages or injuries caused by the improper or un recommended use of HexaBOT.

All advanced modifications or deviations from the directions contained herein are considered to be at the risk of the HexaBOT

This is the HexaBOT Owners Manual v1.00 for the robot HexaBOT.

For more information go to http://www.robosoftsystems.co.in

or contact us at support@robosoftsystems.co.in

Contents

Safety instructions

Be careful while using the screw driver, place the machine on a platform and then tighten the screws. Don't tighten the screws keeping the assembly in your hands, as the screw driver may slip and you may injure your hands.

Don't work on your machine with wet hands. The machine is having electronics parts & battery, hence there are chances that you may get an electric shock(It may cause severe injuries or loss of life).

The components on the electronic boards are having soldering joints at the bottom end which are vulnerable to short circuit, to avoid such a situation don't place the circuits on a conducting surface when the supply voltage is there, otherwise there can be a short circuit between the soldering joints causing permanent damage.

The battery provided to you is having two wires, please don't short the wires together, if you do so that will discharge the battery quickly. This can damage the battery in the long run reducing its life.

The microcontroller board is very fragile. Please take appropriate care when handling the microcontroller board. It may break if you apply more pressure than you ought to.

About Robosoft Systems ®

Robosoft Systems ® is a young & dynamic organisation still in its nascent stage. It all started with a vision back in 2008 when a group of young engineer's bitten by the entrepreneurship bug came together for transforming their ideas into reality, & it had to be the dynamic field of Robotics just like the young minds themselves.

Just one year into its inception & the company is creating waves in the field of Educational & Industrial Robotics with a wide range of intriguing products. We at Robosoft Systems ® believe in passing on the good knowledge. With 70% of the countries population being young generation, the future lies in the schools & colleges, hence we have been conducting WORKSHOPS at schools & colleges all across INDIA many of them in association with the most respected media organisation TIMES OF INDIA(NIE) and one of the most prestigious Technical Institutes in the world IIT Bombay .The WORKSHOPS have become an instant hit since it provides the students the ultimate stage for understanding science to its core concepts & going beyond the limits of textbooks.

On the technical front the company has had path breaking success lead by a dedicated team of R&D engineers. The company has been launching products defying cutting edge technology. This pursuit of excellence has won the company many accolades. Recently the company entered into a joint venture with SOLARBOTICS® CANADA, a company which deals in high end SOLAR BOTS.

Managing Partner

Robosoft Systems®

Introduction

This manual has been written keeping in mind robotic enthusiasts from all age groups. It will take you right through the building steps of the HexaBOT to controlling it.

The HexaBOT is a basic level robot from Robosoft Systems®. It helps you to get a feel of what robotics is all about. The HexaBOT has been developed by bringing together the laws & concepts of Physics, Electronics & Mechanics hence it helps one to inherit this knowledge as one is implementing it.

The HexaBOT is basically a robot which uses legged locomotion. It is quite evident from the name that it will be six legs for locomotion. The robot can work as manual controlled or it can be autonomous also. Speciality of HexaBOT is ,It is using only 3 servo motors to control 6 legs. MCS51 (P89V51RD2) based controller based board is used for controlling servo motors hence to control HexaBot. The TSOP sensor based obstacle avoiding module helps machine roam around room and avoiding obstacles. HexaBOT can also be controlled using IR remote. Your microcontroller board can be programmed using flash tool. It is easy to install and easy to use. Simple USB interface allows board to connect with PC or Laptop. Prior knowledge to MCS51 systems will allow user to use the controller board for embedded systems development.

Statutory Warning:

This is a highly addictive robotics kit; please do your school homework before starting.

C—How to program

Need for computing

Why do we use computers ? The most orthodoxically common answer we get to hear is that "The computer makes our life easier". But that is easy said than done. I have been coming across this line since my early college days, but genuinely speaking I never understood that, neither did I bothered to. The only think I knew to do on the computer was Gaming. So let us try to understand the job of this revolutionary gadget.

Example: Previously the use of typewriters was widespread to write letters & other official documents. But the problem with it was that it wasn't that flexible, one typing mistake even in the last line of the letter could compel you to type it all over again. This task was simplified with the introduction of MS-Word. This is one of the simplest e.g. I could imagine, now you can think of other complex industry application where the computer has brought about a world of difference.

Thus in this way the computer along with some software helped us to find a solution to our problems. But the computer doesn't do all this on its own. We first need to tell the computer what to do, to solve our problem and for this we need to communicate with the computer and its not Hindi, English or Marathi that the computer understands. The only language that the computer understands is the Binary language which has only two distinct symbols i.e. "Logic 0" and "Logic 1". But than it becomes even more difficult for us to program the computer, just imagine the size of programs if we were to write programs using 1's and 0's and thereafter forget about the job of debugging the program if some mistakes were committed. So there has to be some other way out. To overcome this problem a stream of programming languages were developed over the past 4 decades. Some got very popular and some couldn't make it. We will go with one of the most popular one's and that's the C programming language.

Although we assume that you people must be aware of the different numbering systems and their significance we have included this topic just to give an overview. It is important to have a sound knowledge of numbering systems as they play an important role in the representation and manipulation of data. Some of the most popular systems are the Decimal system used by human being, Binary system used by Computers and Hexadecimal system used for convenient representation of binary values.

Decimal Number System:

The Decimal System also known as Positional System is the most widely used numbering system in the world today. The distinctive feature of numbering systems is the Base of the numbering system. Each system has its unique Base. Decimal has a Base of 10. This system has 10 distinct symbols from 0 to 9, hence the name Base of 10.

Ten Thousands	Thousands	Hundreds	Tens	Ones
10^4	10^3	10^2	10^1	10^0

The Decimal System is also called as the positional system because in this system weight of each digit in a number depends on its position.

Example: In the decimal number 93152 the weightage of number 2 is 10^0, similarly the weight of 5 is 10^0. Therefore we write the equation

$$(9x10^4) + (3x10^3) + (1x10^2) + (5x10^1) + (2x10^0) = 93152$$

Binary Number System:

In the binary number system there are only two distinct symbols i.e. 0 and 1. Binary number system has been used in computers because the is a close proximity between the two. Just like the binary system even the computer has two distinct possibilities i.e. "ON" or "OFF", which can be easily represented by Logic 1 and Logic 0.

10^2	10^1	10^0
2^2	2^1	2^0

The binary system works exactly on the same principles as the decimal system, it has a base of 2 rather than base 10. Even a binary number carries some weight as shown in the above fig.

Example: In the binary number 11110 the weight of 0 is 2^0, and the weight of the next 1 is 2^1. Hence we can find the decimal equivalent as follows.

$$(1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0) = (31)_{10}$$

Hexadecimal Number System:

The Hexadecimal Numbering System is a Base 16 system i.e. it has 16 unique symbols that can be used. Out of the 16 symbols the initial 10 symbols are same as in decimal numbering system from 0 to 9, after this we use the notations from A to F to represent values from 10 to 15 respectively. There is an important relationship between the binary & hexadecimal numbering systems. The hexadecimal numbering system is the preferred choice for representing binary data in any application. The reason being that conversion between binary & hexadecimal is the easiest and fastest. Even if you try to convert hexadecimal number into decimal number than it would require some calculation and a finite amount of time, whereas on the other hand you can just memorise a table and easily convert a binary number into an hexadecimal number.

Example: A binary number 111100000010, can be easily converted into a hexadecimal number by grouping the digits into groups of 4's.

$$1111\ 0000\ 0010 = F02$$

C is a *general purpose programming language* which was developed at the AT&T Bells Laboratory in the USA in 1972. It was originally designed for implemented on the ***UNIX OS*** on a machine called as the DEC PDP-11 by a man called Mr. Dennis Ritchie. C is not a very high level language nor a big one and it is not specialized for any particular area of application . Many parts of the popular operating systems like Windows, Linux and UNIX are still written in C. This is because when it comes to performance i.e. the speed of execution nothing beats C.

How to use C

To do anything in a programming language we would require some quantity or entity in which the user can first give the input, which will be processed and than the program will produce the desirable output. It works just like an open loop system as we all have studied.

This quantity or entity which we are talking about can be realized in the form of a VARIABLE in C language. Let it be a task as simple as printing your NAME or a complex one such as interfacing temperature sensor with your μc and measure the temperature. Everything begins with the use of a VARIABLE, hence it is the basic building block in learning C.

Constants and Variables:

A *constant* is a quantity that doesn't changes e.g. 20. This quantity can be stored in a location in the memory of the computer. The addresses of the memory are specified in Hexadecimal format which starts from 0000h and upper limit is determined by the size of the memory. Now suppose while writing a program for temp measurement you saved the reading from the sensor, which was stored in memory location FF0Ah, now after some time you want to access that value, then it is really inconvenient for you to remember and use the address FF0Ah to fetch the reading. You could imagine the enormity of the difficulty if you were to develop an application with thousands of readings which is a common practice at the industry level.

To overcome this difficulty most of the programming languages use the concept of variables. A quantity which may vary during program execution is called variable. A variable can be considered as a name given to the location in the memory of the computer where the constant is stored. So this solves our problem of remembering the addresses of the locations, now we can choose a name which is easy to remember and relevant to its use.
Example: To store our age we can use the variable name "age", similarly we can use "temp" for storing the temperature.

Data Types in C:

When we are creating a variable we are actually allocating space for that variable in the memory. What should be the size of that variable 8, 16 or 32 bits. The size is user depend, we can decide upon the size of a variable depending upon our requirement. The same is achieved in C with the help of data type, the following table explains it.

Type	Size (byte)	Min	Max	Format specifiers
signed char	1	-128	+127	%c
unsigned char	1	0	255	%c
signed int	2	-32768	+32767	%d
unsigned int	2	0	65535	%u
long int	4	-2147483648	+2147483647	%ld
float	4	-3.4e38	+3.4e38	%f

Note: "char" is by default taken as "signed char". Similarly "int" is by default taken as "signed int" of size 2 bytes.

The above data types also falls in a categorisation called as "keywords". C has a set of 32 keywords. With the combination of a keyword and a variable name we can create a C instruction.
Example: int height; or char gender;

In the above e.g. height is a variable name stored in the memory and it type is "signed int" hence its size is 2 bytes. Similarly gender is a variable name having type "signed char" and size is 1 byte. In both the e.g. you must have observed the semicolon (;) sign. This a statement terminator in C. All the statements should be terminated by a semicolon.

C is also known as a Structured Programming Language, because it provides a structured approach towards solving problems. The programs in C have a pre-defined format as shown below.

- Header files (Pre-Processor directives)
- Prototype declaration
- Global declaration/definition
- Main function
- User defined functions

There has to be a main function in every C program. It is mandatory, the CPU always starts its operation from the main function.

Syntax of a Function:

```
return_type  function_name (parameters)
{
        Statement(s);
}
```

function_name is the identity of the function. The user can give a relevant name to the function for instance if we are writing a function to add two numbers we can call the function as add.

return_type specifies the type of value that a function sends back after executing, for e.g. in the add function after adding two integers the result will be an integer, hence the return type is int.

parameters when we are calling the function add to add two numbers, we should also pass the two integers which are to be added, these are nothing but parameters.

Function Body:

The function body consists of different kinds of instructions like.

- Type declaration instructions (int age, char name)
- Arithmetic instructions (will consist of operators like +,-,/,* etc.)
- I/P and O/P instructions (printf , scanf)
- Control flow instructions (if, else, while)

Sample program

```
#include<stdio.h>        //Header File

void main(void)
{
     printf("Hello World");
}

Output :  Hello World
```

Note : The above e.g. explains one of the simplest programs possible in C language. The structure of this program is not at full potential of C, it doesn't even covers the "VARIABLE" aspect of C. The program starts with the header file ***#include<stdio.h>*** where we are telling the compiler to include the file stdio.h which consists of various library functions which we will be using quite often e.g. the ***printf*** function which we are calling to print "*Hello World*" is a library function in stdio.h than there are user defined functions which we have to create in the program if it is not provided in the library.

Types of variables

Variables can be classified based on various criteria's, but we are discussing variables with respect to the scope in which they are declared/defined. Based on this classification the types of variables are.

- Local variables
- Global variables

Local Variables :

The variables which are declared or defined inside a function viz. main or user defined functions are called as local variables or **automatic variables.** They are always stored in the **Stack Segment** of the memory and they exist as long as that function is being executed. Once the function is exited the variables are deleted and the will be created again if that function is called again for execution. These variables can be accessed only inside the function they defined. If you try to print its value outside it function it will give an error.

Example—Local variables

```
#include<stdio.h>

void main(void)
{
     int a;                    // Local Variable (uninitialized)
     int b = 100;              //Local Variable (initialized)
     printf("a = %d",a);
     printf("b = %d",b);
}

Output : a = 4379 (some garbage value)
          b = 100
```

Global Variable :

Global variables are the ones which are declared or defined outside a function. They are never declared/defined inside any function like main or any other user defined function. These variables are always stored in the **Data Segment** of the memory and they exist as long as that program is being executed. They are called as global because unlike local variables they can be accessed from any part of the program.

Example—Global variables

```
#include<stdio.h>
int a;                    //Global variable (uninitialized)
int b =100;               //Global variable (initialized)
void main(void)
{
     Printf("a = %d  \n  b = %d",a,b);
}
          Output : a = 0    b = 100
```

C provides a rich set of operators spanning from arithmetic, logical, bitwise to combinational operators. If more than one operator appears in the same equation than we have a priority and precedence table to refer.

Priority	Operators
1	( ) [ ] -> .
2	! ~ - * & (sizeof) cast ++ --
3	/ %
4	+ -
5	< <= >= >
6	== !=
7	&
8	^
9	&&
10	||
11	?:
12	= += -=
13	,

The priority of the operator is given in the above table and the precedence of the operators are from left to right for all rows except for row number 3, 4, 11, 12 where the precedence is from left to right.

Control instructions

While loop

The ***while*** loop is called a control instruction because it changes the flow of execution of program depending upon certain conditions. The syntax of this loop is:

```
while (expression)
{
        statement(s);
}
```

If the expression inside the parentheses of the while instruction evaluates out to be **true** than the body of the while loop is executed otherwise the program execution directly jumps to the instruction after the while loop.

In C an expression is said to be true if it is a value greater than 0. Therefore

while (expression) *if expression = 0 then it is false and while is not executed.*
if expression >= 1 then it is true and while is executed.

For loop

It is one of the most widely used control instruction, it has a structure slightly complicated than the while loop. The syntax of this loop is:

```
for (initialize; test; increment)
{
        statement(s);
}
```

The structure is executed as follows:

- The program control first initializes a variable.
- Then it goes to the next step i.e. test, where it tests the expression. If that expression evaluates out to be true, the body of for loop is executed; otherwise the program control leaves the for loop and executes the next instruction.

- If the expression is true then after executing the for body the program control goes to the increment section where it increments a variable.
- After this it again comes to the test section and again checks the same expression and executes the for loop only if result is true and it keeps on doing the same unless the test evaluates to be false.

Do While loop

The ***Do While*** loop is a variant of the While loop. It performs nearly the same function as the while loop. It differs from the while only in the aspect that it does not check the condition before it executes the loop for the first time. The syntax of this loop is:

```
do
{
        statement(s);
}
while(expression);
```

The loop is executed at least *once* even if the expression evaluates to be false. After the loop has executed once, then the expression is checked and if it is found to be false, then the loop terminates and the control terminates to the next instruction after while. If the expression is found to be true then the control again jumps to the instruction within the loop.

If– else statement

This is a very versatile feature of C which enables us to take decision depending on a situation.

```
if (expression)
{
        statement(s);
}
else
{
        statement(s);
}
```

Only if a particular condition is met the body of "if" will be executed otherwise the body of the "else" will be executed.

Switch statement

The ***switch*** statement is useful in eliminating the multiple if else statements. It comes handy in situations where we need to make a decision from an array of options available. The syntax of this statement is:

```
switch (expression)
{
        case 1:
                statement(s);
                break;
        case 2:
                statement(s);
                break;
        :
        :
        :
        default:
                statement(s);
                break;
}
```

The switch statement provides a way for multi branched conditions in a very neat manner. Imagine you have ten conditions to check and having to write ten if-else blocks. It is generally a good practice to write a default block which provides for default execution of statements when nothing else matches.

Embedded C

There is not much of a difference between C and Embedded C programming. Embedded C is designed to be more efficient than C when is comes utilisation of memory. Hence it is the language of choice for embedded system designer working on μc's and μp's, as these chips have limitations of on chip memory the code space for 8051 is limited to 64KB. The programs written in C are given to special compilers which produces the hex file and the goal of an embedded programmer is to create smaller hex files, hence a good understanding of C data types is necessary.

Data types

The various data types in Embedded C are:

1. unsigned char
2. signed char
3. unsigned int
4. signed int
5. float
6. Sbit(single bit)
7. bit and SFR(special function register)

While programming it will be a good practice to replace unsigned int data type with unsigned char where ever possible, this significantly reduces memory consumption, normally this replacements can be done in places where we are using loop counters (for, while etc). We can use the sbit data type wherever we require a flag (semaphore) instead of using a char or int. signed char and int should be used only in places where we are dealing with values which may be negative or positive like temperature.

Sbit:

The sbit keyword is widely used specifically designed to access single bits of SFR's. some of the SFR's in our controller are designed to be bit addressable. Some of these SFR's are P0 through P3, hence we can use sbit to use single bits of the ports P0– P3.

Bit:

The bit data type allows access to single bits of bit addressable memory space between 20 to 2Fh.

SFR:

The 8051 has many SFR's which are located just above the internal RAM starting from 80h-FFh.

Bitwise operators

The bitwise operator's are of great importance in embedded C programming, they help a great deal in reducing the size of the code. There are six of these operators.

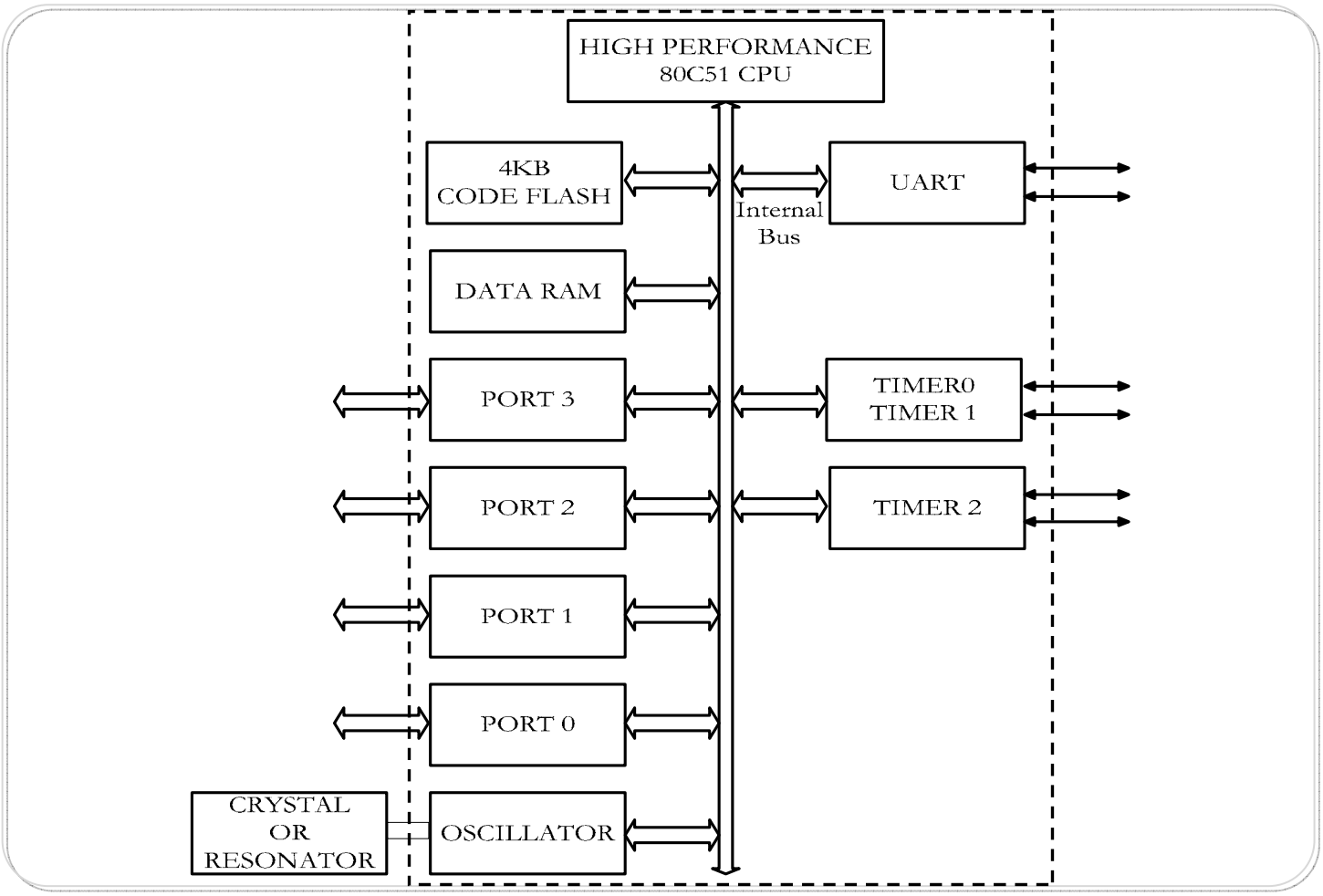
1. & AND
2. | OR
3. ^ XOR
4. ~ NOT (Complement)
5. >> RIGHT SHIFT
6. << LEFT SHIFT

Example program

```
#include <reg51.h>
void main(void)
{
     P0 = 0x35 & 0x0F;        //ANDing
     P1 = 0x04 | 0x68;        //ORing
     P2 = 0x54 ^ 0x78;        //XORing
     P0 = ~0x55;              //Complementing
     P1 = 0x9A >> 3;          //right shift (divide by 2)
     P2 = 0x77 << 4;          //left shift (multiply by 2)
}
```

Note : We are using the header file #include <reg51.h> because it contains the prototype declarations of the 8051 μc resources, so we are using ports P0,P1,P2 directly in the code.

µController architecture



Inside the 8051µC

The above block gives an overview of the 8051, although there are other resources that the 8051 provides for the user's disposal most importantly the register banks. The array of register are.

1. Accumulator (A)
2. B register
3. 4 banks of 8 registers each (R0 to R7)
4. Data Pointer (DPTR)

The accumulator is the most important register it is 8 bits wide and used in many internal operations of the ALU (Arithmetic Logic Unit) as well as in external communication of the 8051.

Ports

The 8051 has 4 I/O ports(P0-P3). As the very name suggests these ports are used to bring and send the data IN and OUT of the 8051.

Port 0

P0 occupies the pins 32 through 39 on the 8051 chip. It can be configured as an I/P or O/P port through proper programming. In order to do so we need to connect an external pull up resistor at each port pin. This is due to the fact that P0 uses open drain technology unlike the other ports.

Dual role of P0 :

When we are interfacing external memory with the 8051 for data storage and then fetching it, we also need to take care of the addresses in which the data is stored i.e. the user also needs to specify the address, P0 is responsible to bring that data/address IN and OUT of the controller hence it is also designated as AD0-AD7. This saves our pins.

Port 1

P1 occupies the pins 1 to 8 in the chip. It can be configured as an I/P or O/P port with the help of programming. This port doesn't requires any pull up resistors as it has built in pull ups for all the 8 pins. Although for practical reasons you will still find external pull ups on an 8051 development board. P1 employs N-MOS technology.

Port 2

P2 occupies pins 21 to 28 in the 8051 chip, it can be used as I/P or O/P port. Just like P1, P2 doesn't require any pull up resistors as it has internal pull ups.

Dual role of P2 :

Since the 8051 can access 64KB of external memory, it needs a path for the 16 bits of the address. While P0 provides the lower 8 bits via A0-A7 it is the job of P2 to provide bits A8-A15. in other words when8051 is connected to external memory, P2 cannot be used for I/O.

Port 3

P3 occupies pins 10 to 17 in the 8051 chip, it is a bidirectional port and it doesn't requires external pull ups. Port 3 has the additional function of providing some extremely important signals such as interrupts. The table below provides these alternate functions.

P3 bits	Function
P3.0	RxD
P3.1	TxD
P3.2	INTO
P3.3	INT1
P3.4	T0
P3.5	T1
P3.6	WR
P3.7	RD

Each of the ports discussed above can be programmed as I/P or O/P port. A port can be configured as I/P port by writing a 1 to all its pins, similarly a port can be configured as an O/P port by writing a 0 to all its 8 pins.

Example program

```
#include <reg51.h>
void main(void)
{
     P0 = 0xFF;     // Input port
     P1 = 0x00;     // Output port
     P1 =  P0;      //Taking value from I/P port and writing it to O/P
```

Note : On reset all the ports of 8051 are automatically configured as Input ports by default. But still in programming we need to configure them as I/P or O/P port because there is a possibility that the status of the port pins may change due to some externally connected devices.

Timers

The 8051 has 2 timers : Timer 0 and Timer 1. They can be used either as timers or as event counters. These are basically hard wired registers which are 16 bits wide. Now since the 8051 has an 8 bit architecture, each 16 bit timer is accessed as two separate registers of lower byte and higher byte. In the timer mode the registers are used to generate some time delays and in counter mode it is used to count external events happening outside the 8051.

Timer 0 registers

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

The lower byte register is called TL0 (timer 0 lower byte) and higher byte register is known as TH0. These can be accessed as any other register such as A,B, R0 etc.

Timer 1 registers

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

These 16 bit register is split into 2 bytes, referred to as TL1(Timer 1 lower byte) and TH1(timer 1 higher byte).

Other important registers

The registers shown above are mere registers which can do some counting of numbers, but there is a great deal of programming which goes behind running the timers according to our requirement, for this programming to be implemented we use some registers namely TMOD, TCON.

TMOD register

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

Both timers 0 and 1 use the same register, called TMOD. To set the various timer operation modes. TMOD is an 8 bit register in which the lower 4 bits are set aside for timer 0 and the upper 4 bits for timer 1. In each case the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation.

TCON register

TF1	TR1	TF0	TRO	IE1	IT1	IE0	IT0

As of now the most important bit in the TCON register is the TR0 or TR1 bits, that's because these are the timer run control bits which initiates the timer count.

Interrupts

A micro controller can be thought of as a master which gets its work done from other devices connected as peripherals to it. For this purpose the µc needs to manage it communication with these peripherals efficiently. One of the methods of communicating with the peripherals is POLLING, wherein the µc continuously checks whether its peripherals wants its help and whenever there is a request for communication the µc serves that device, but this method is not efficient at all, due to the wastage of the µc's time and also it is not possible to assign priorities to devices in case there are some important one's. The other more efiicient option is to go for the interrupt method.

In the interrupt method the µc instead of wasting its time in polling the devices it can busy do some other work. The peripheral devices need to tell the µc if they need its help, this is done in the form of an interrupt. Whenever the µc receives an interrupt it first takes a backup of its current work and than goes to help the interrupting device by executing its ISR.

Interrupt Service Routine

For every interrupt, there must be an ISR, or interrupt handler. For every interrupt there is a fixed location in memory that holds the address of its ISR. This area of the memory is called the Interrupt Vector Table (IVT). The 8051 has 6 different interrupts.

Interrupts	ROM Location
Reset	0000
External hardware interrupt 0 (INT0)	0003
Timer 0 interrupt (TF0)	000B
External hardware interrupt 1 (INT1)	0013
Timer 1 interrupt (TF1)	001B
Serial COM interrupt	0023

Enabling or disabling interrupts

Upon reset, all interrupts are disabled, meaning that the μc will not respond to them. The interrupts must be enabled by software for the μc to respond. The is a register called IE (interrupt Enable) that is used to enable/disable the interrupts.

EA	--	ET2	ES	ET1	EX1	ET0	EX0

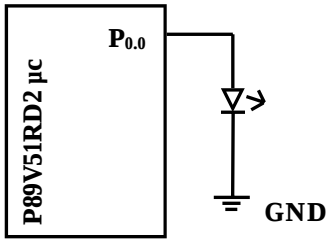
Interrupt priority

What happens if two or more interrupts are activated at the same time? Which interrupt will be serviced first. When the 8051 is powered up, the default priorities are assigned to the interrupts.

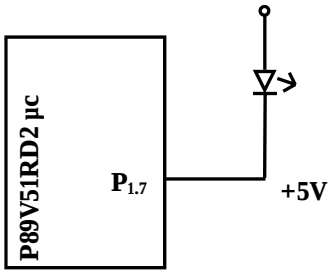
Priority Level	Interrupt
1	External Interrupt (INT0)
2	Timer Interrupt 0 (TF0)
3	External Interrupt 1 (INT1)
4	Timer Interrupt1 (TF1)
5	Serial Communication (RI, TI)

Now suppose if external hardware interrupt 0 and 1 are activated at the same time, than external interrupt 0 (INT0) is serviced first because it is having higher default priority. Only after INT0 has been serviced INT1 is serviced.

To glow the LED's we first need to understand our hardware connection between the μc and the LED, and also study other ways to do it.



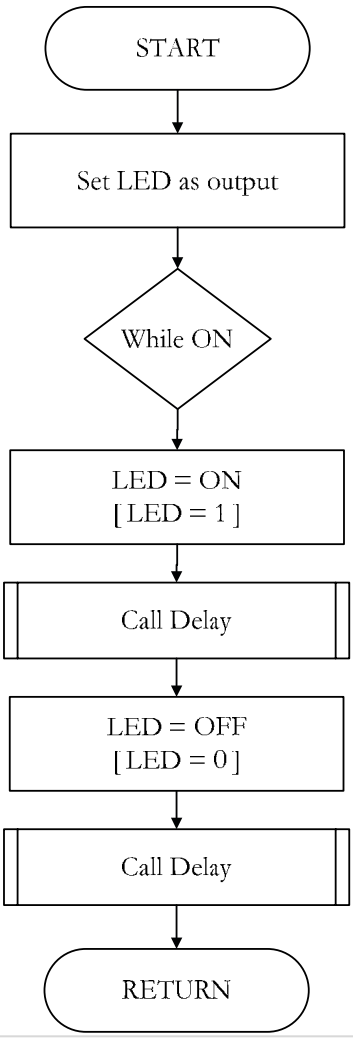
Active high control



Active low control

The above fig. shows the two configuration in which an LED can be corrected, the first fig. of active high control shows the anode of LED is connected to the port pin and cathode is grounded, hence we need to pull the pin high to glow the LED. In the second fig. of active low control the anode of the LED is connected to +5V and cathode is connected to the port pin.

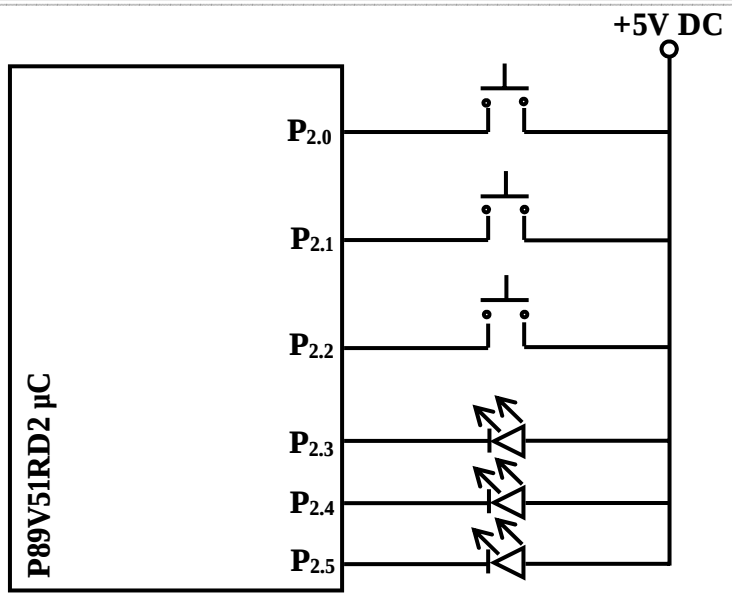
Hence we need to pull the pin low so as to glow the LED. Pulling the pin low will provide ground to the LED. As we all know that an LED is basically a PN junction diode, which emits light when it is forward biased, so we are meeting this condition in both the above cases by the different hardware connections.

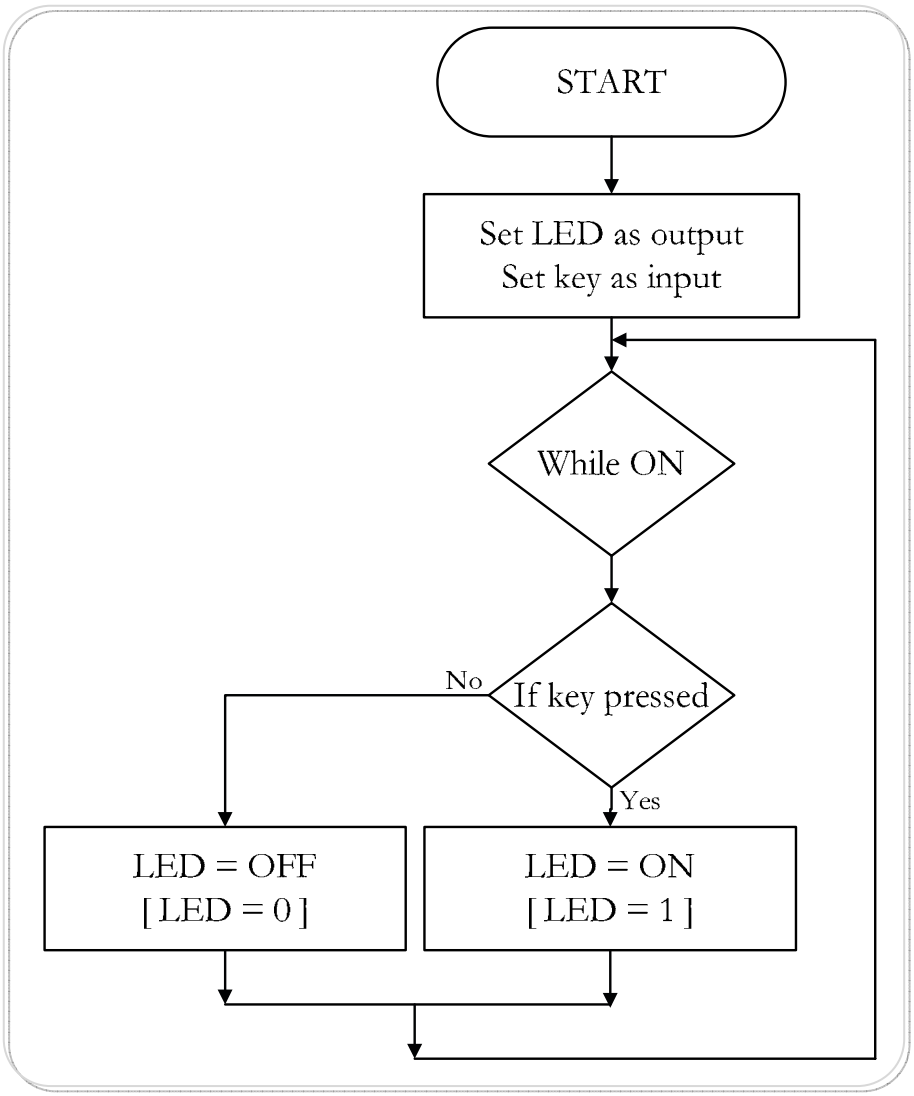


Example program– Glow LED's

```
#include <reg51.h>
sbit LED1 = P2^3;   //LED 1 is connected at port pin 2.3
sbit LED2 = P2^4;   //LED 1 is connected at port pin 2.4
sbit LED3 = P2^5;   //LED 1 is connected at port pin 2.5
void main(void)
{
     LED1 = 0;      // Glow LED1
     LED2 = 0;      // Glow LED2
     LED3 = 0;      // Glow LED2
     delay(100);    // wait for some time between LED ON and LED OFF
     LED1 = 1;      //LED1 OFF
     LED2 = 1;      //LED2 OFF
     LED3 = 1;      //LED3 OFF
     delay(100);
}
```

Example circuitry—Interfacing keys





Your board is having a total of 5 push button switches out of which two are RESET and PSEN, which we will leave aside for the time being and study the remaining 3 switches which are connected to port pins P2.0 to P2.2. We can use these switches for various needs in our programming.

Example program– Interfacing keys

```
#include <reg51.h>
sbit SW1 = P2^0;
sbit SW2 = P2^1;
sbit SW3 = P2^2;
sbit LED1 = P2^3;
sbit LED2 = P2^4;
sbit LED3 = P2^5;
void main(void)
{
     LED1 = LED2 = LED3 = 1;
     SW1 = SW2 = SW3 = 1;
     while(1)
     {
          if(SW1 == 0)
               LED1 = 0;
          else
               LED1 = 1;
          if(SW2 == 0)
               LED2 = 0;
          else
               LED2 = 1;
          if(SW3 == 0)
               LED3 = 0;
          else
               LED3 = 1;
     }
}
```

Servo motors

A "servo" is a generic term used for an automatic control system. It comes from the Latin word "servus" - slave. In practical terms, that means a mechanism that you can set and forget, and which adjusts itself during continued operation through feedback. Disk drives, for example, contain a servo system ensuring that they spin at a desired constant speed by measuring their current rotation, and speeding up or slowing down as necessary to keep that speed.

A Servo motor consists of the following components:

- A DC motor
- Gears with an output shaft
- Position sensing mechanism
- Control circuitry

The control circuitry indicates to the servo the position that the output shaft *should have*. The position-sensing mechanism tells the servo what position the shaft *currently has*. The control circuitry notes the difference between the desired position and the current position, and uses the motor to "make it so". If the difference in position is large, the motor moves rapidly to the correct position; if the difference is small, the adjustment is more subtle.

Fig 1

Fig 2

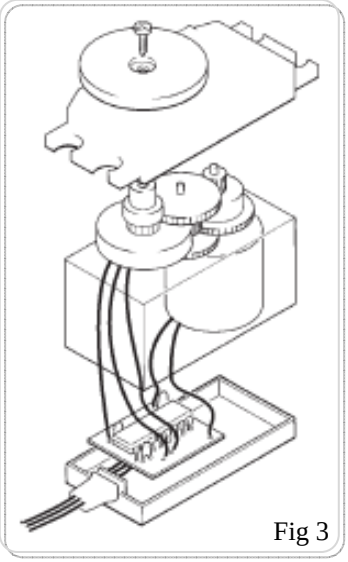


Fig 3

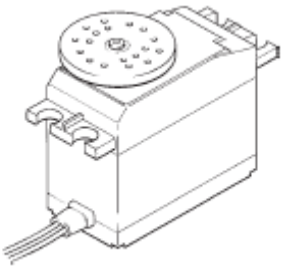


Fig 4

The figures the line diagram schematic and the exploded view of a servo motor. You can see the *control circuitry* and *position sensing mechanism* inside the servo motor. The position sensing is nothing but a potentiometer which gives the relative resistance to the origin at any point of time. The gear train inside the servo motor is also evident in the diagram. Finally we have three wires emanating out of the motor; VCC, Ground and Control. The motor shown can be driven with a maximum voltage of 5V DC.

Controlling the servo

Controlling the servo is quite easy. But you cannot drive it directly just by plugging it into 5V DC. You need a train of pulses along the control line. The width of the pulses that you provide determines the angle of rotation of the shaft of the servo motor. This method is called ***Pulse Width Modulation***.

To drive the servo, the pulses that you need to provide should have a minimum delay of 20ms. That is the off time between the pulses should have a minimum duration of 20ms. This off time may or may not be accurate. You cannot offset it by too much, but. You can exceed the 20ms mark somewhat but you cannot go below that. Now how do you think you drive the servo motor to various angles? Do you know that you cannot drive a stepper motor continuously in circles.* You can only make it rotate to several predefined angles like 0°, 45°, 90°, 135° and 180°. You would require to pass pulses of different widths, though! Below, we have listed the pulse width required to move the servo to several angles:

- 0° ———— 0.5 ms
- 45°———— 1.0 ms
- 90°———— 1.5 ms
- 135°———— 2.0 ms
- 180°———— 2.5 ms

Please refer overleaf for a graphical overview of the pulse width.

If you notice, in each case, the off time of the signals remain the same: 20ms. This 20ms off time ensures that the pulses are repeated 50 times in one second. After you ensure this, you may vary the on time of the pulses from 0.5ms to 2.5ms to vary the angle of rotation of the servo motor shaft. Beware that the on time of the pulses should be maintained accurately, otherwise the servo won't rotate at all. You have to keep on providing the same pulse width until you want a change in rotation. After that you have to provide the new pulse width to maintain the position of the servo.

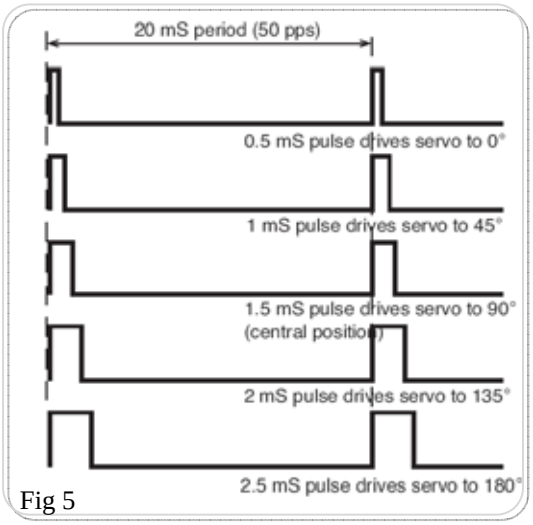


Fig 5

You have to understand that there is no easy way to make the servo rotate continuously. A servo is used only in those cases where you have fixed scope of rotations and you require accurate and exact positioning. Suppose that you require that your motor rotate only 45° and then stop over there. For this to happen, you have to provide a continuous pulse train with an *off time* of ***20ms*** and an *on time* of ***1ms***. This situation can also be attained with a stepper motor, but in a different way. But the difference will be evident when you actually turn off the system and subject the motor to nay free rotations. In a stepper motor you do not have any reference as to where your motor shaft is exactly positioned. But a servo motor always knows where its shaft is positioned. So if you again turn the system on and start giving the same pulse width of 1ms, the motor will get back to its original 45° position.

There are servo motors available which can actually rotate 360° continuously. But these motors are very costly and are practically not feasible for small time projects. There is one another way in which you can make a normal servo motor rotate continuously. If you open a servo motor, you will find a hardware lock which actually prevents the motor rotate more than 180°. You have to break that lock which will allow the motor to move to any angle you want. If you want the motor to come back to its reset position (0°) you have to resume pulses of 0.5ms width.

The advantage of this motor is that it can be operated even without a microcontroller. To achieve you can use two 555 timer IC to generate the desired pulses. Why two? This is because one 555 timer IC will generate a pulse with 20ms on time and the other one to invert it and also to introduce a specific on time in those pulses. The other method is to generate PWM using a microcontroller.

Interfacing the servo motor

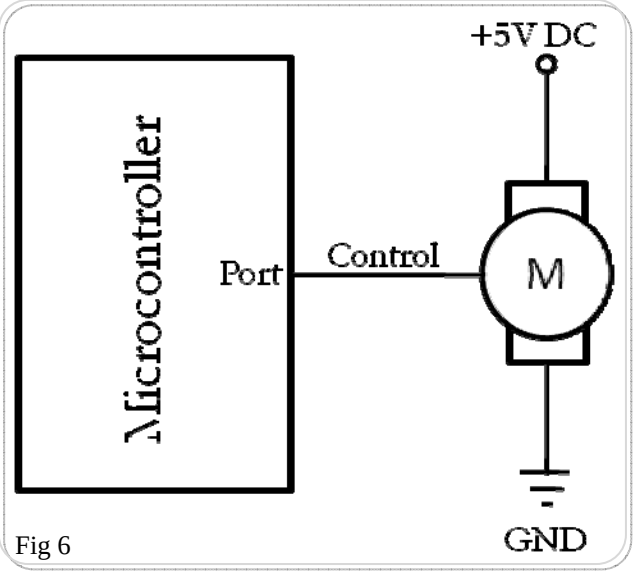


Fig 6

Beside we have shown the functional diagram of how to connect a servo motor to a microcontroller. A servo motor has three leads, VCC, GND and Control. We have connected VCC to +5V DC, GND to Ground and Control to any port pin of the microcontroller which may be free. The servo motor draws the required electricity from an external source other than the microcontroller. So we do not require to connect a motor driver circuit or IC to the motor. In the next section we will look at the way about how to generate PWM through the microcontroller.

Generating PWM

A microcontroller has four I/O ports. Each I/O port has eight pins. That means there are thirty two pins available for I/O. These I/O ports can be independently set to either input or output. They are called bit-addressable. This means that if one pin of a specific port has been marked for output then the other pins of the same port can be used for input. Now to connect the servo motor to a pin of the microcontroller we need to configure that pin in output mode. We have an inbuilt timer circuit inside the microcontroller. A timer is a circuit which can be used to count up or down or to generate time delay. So we use an inbuilt timer to generate a signal which has an on time of 20ms. We keep the output port pin low during that time. Then we reset that timer and again use it to generate another signal which has an on time of 0.5ms. We keep the output port pin high during that time. Thus if we analyze the output port pin, we will have a continuous signal which has an on time of 0.5ms and an off time of 20ms. When this signal reaches the servo motor on the control wire, it makes the servo to remain steady at 0° (reset) position. Even if you try to move the shaft of the motor with your hand during this time, the motor will oppose the force that you apply. Now if you want to move the shaft of the motor to the 90° position, you have to make the timer generate the delays so that the output port pin stays low for 20ms and high for 1.5ms. The use of timers and start and stop of them are further explained in detail in the disc that accompanies this product. Please refer to them.

Programming the servo motor

We have seen how the motor can be interfaced to the microcontroller. We have also shown you how to generate the PWM (Pulse Width Modulation) signals. Now we have to program the microcontroller in such a way that it starts and stops the timer and generates time delays, and also makes the output port pin go high or low. Now the entire program has been provided in the disc. We assume that you know the "C" language. Over here we will only concentrate on the functional flowcharts so as to drive a servo motor and eventually the whole robot.

Beside we have shown a flowchart to generically control a servo motor. To control the servo motor, we need a timer to produce delay. Any servo require pulses which has an off time of 20ms and an on time which may be varying from 0.5ms to 2.5ms. So you may replace X with any values ranging from 0.5ms to 2.5ms. First of all we reset the output pin and make the controller wait for 20ms. After that we set the pin and make the controller wait again for a specific *on time* which depends on your application. Over here we use the same timer to set and reset the output pin. If you try to analyse this flowchart, you will find out that we have actually produced a PWM output on the output port pin which has an off time of 20ms and an on time according to your need. The signal diagram that it produces has already been explained before. Refer to Fig. 5.

In this manual we have only presented the conceptual overview of the functions. The programs has not been discussed in detail over here. For details on how to use a timer and setting and resetting of the output pins, please refer to the disc.

This is the way you generally program a servo. Hereafter we won't go into such operational details. For simplicity, we will just mention forward, backward, right and left.

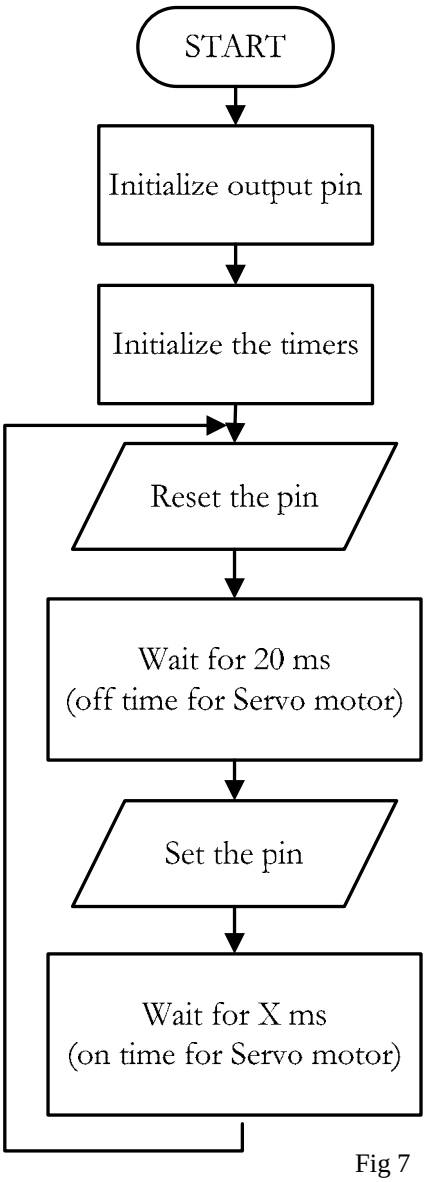


Fig 7

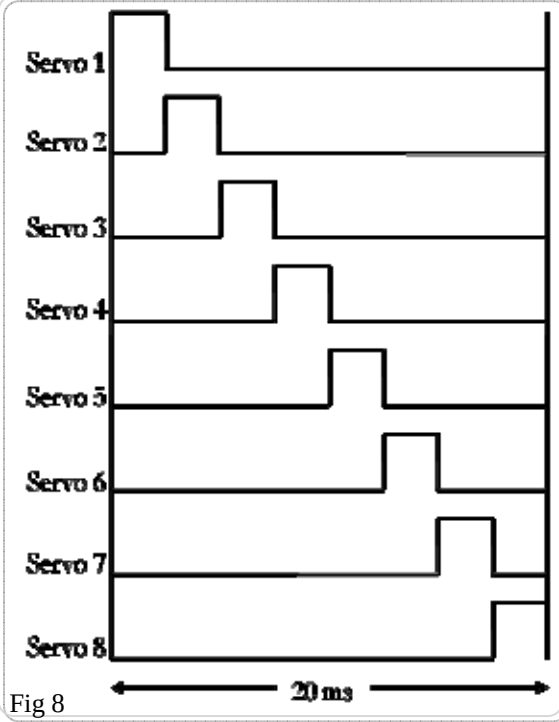


Fig 8

The figure beside shows how we can actually control eight servo motors using a single timer. We consider that the requirement is to turn each and every motor by 180°. Now to make a servo motor rotate by 180°, we need to provide pulses which has an on time of 2.5ms. If we divide 20ms by 2.5ms we get 8. So eight servo motors can be driven at once using a single timer. Now obviously all the motors cannot be run at once. Each motor will follow the previous motor. After the motors have reached the 180° position, all the motors will stop and maintain their positions. Though logically you cannot say that the motors are synchronous, but we may say that the motors act in parallel. The activation high pulse for each motor come only after the previous one has finished.

Programming the motions of HexaBOT

The flowchart beside shows the steps that has to be followed to make the HexaBOT go forward. The robot has three motors: right, left and center. The flowchart is self-explanatory. But still we will explain the forward motion. For the other movement, you are required to figure out the details for yourself by looking at the flowchart. First of all the right motor is moved backward, then the left motor is moved forward. Lastly the center motor makes the right leg move up. After this the controller calls a delay, which is necessary. Without delay the servos will not respond. In the next step the right motor is moved forward, then the left motor is moved backward. Lastly the center motor makes the left leg go up. The controller calls a delay again and the entire process is terminated and the control returns to the calling program. If the calling program decides to call the forward procedure once again, then all the steps that has just been explained are repeated once more before the control returns to the calling program once again.

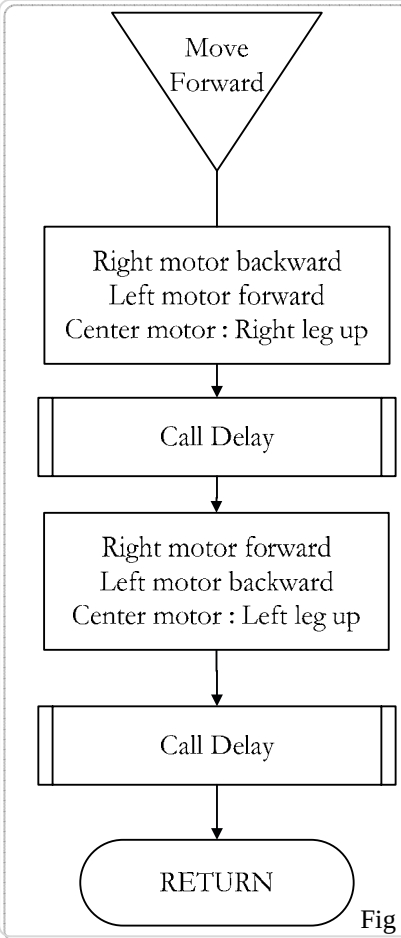


Fig 9

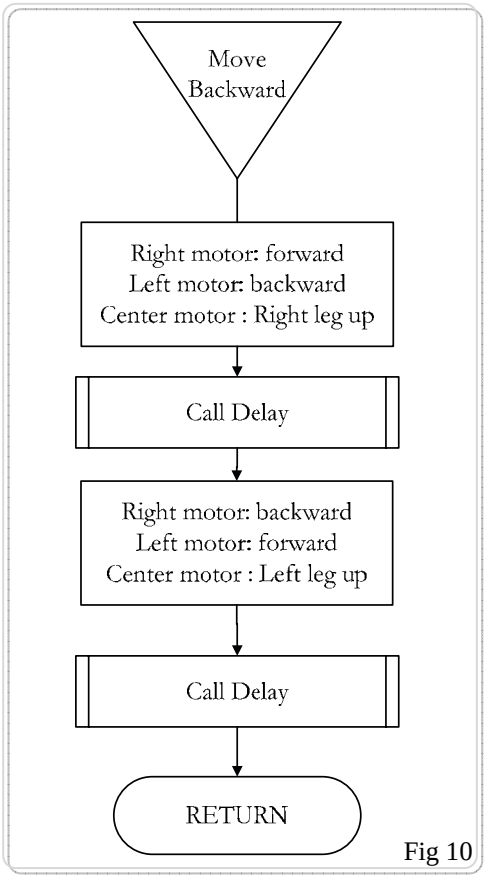
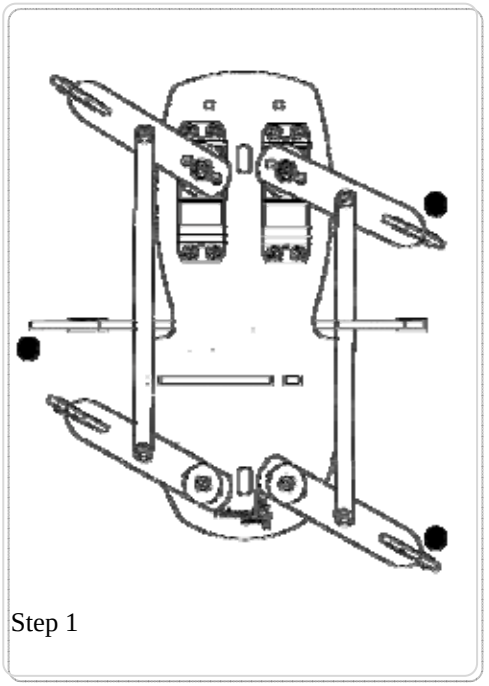
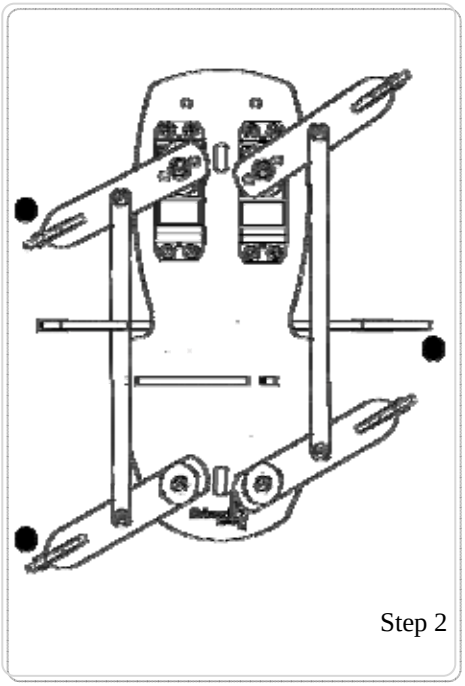


Fig 10

Moving forward


Step 1


Step 2

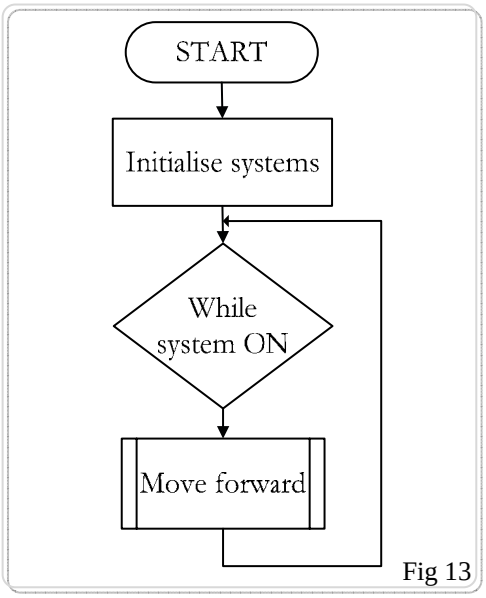
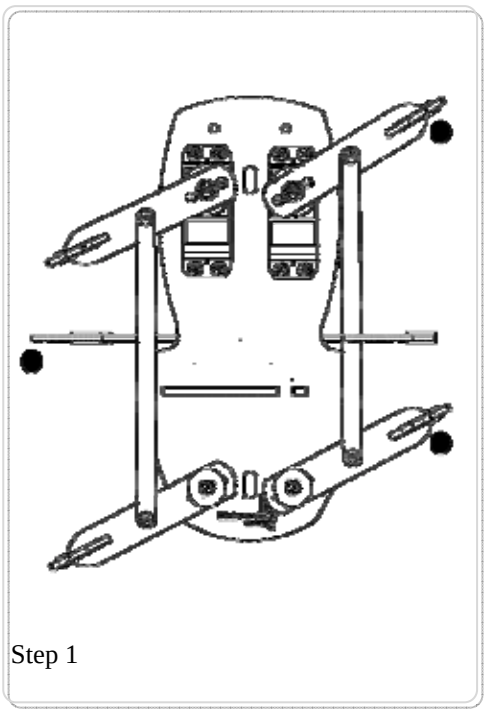
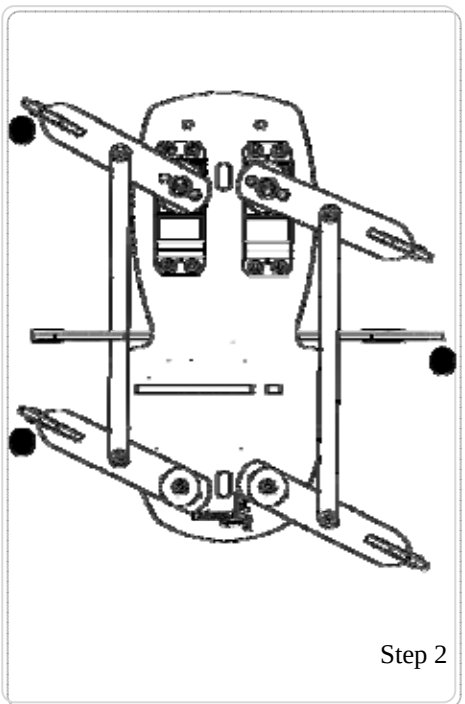


Fig 13

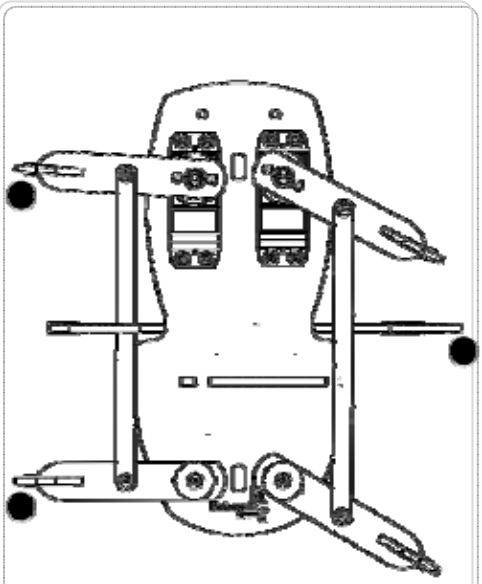
The above flowchart shows how you can call the Move Forward procedure from the main program.

Moving backward

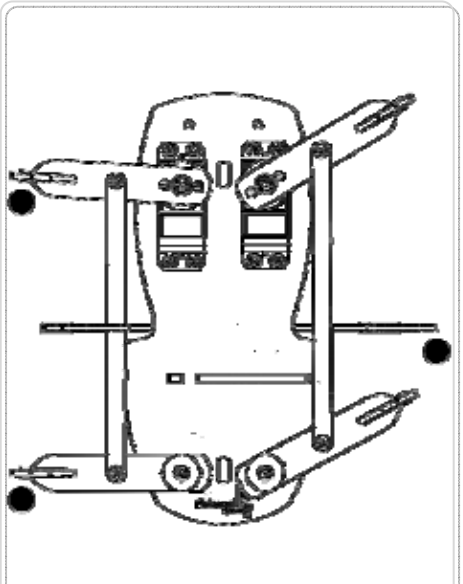

Step 1


Step 2

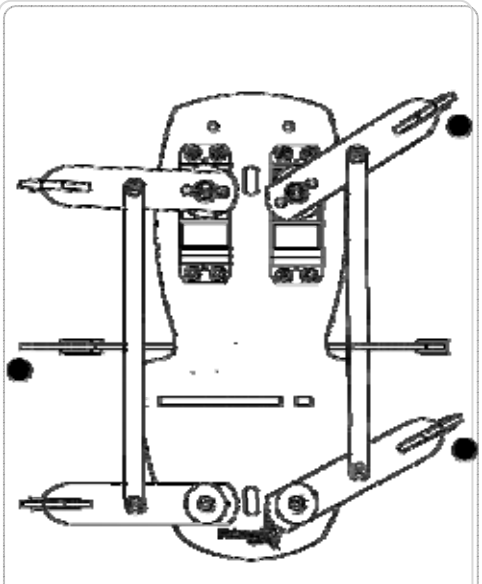
Turning left



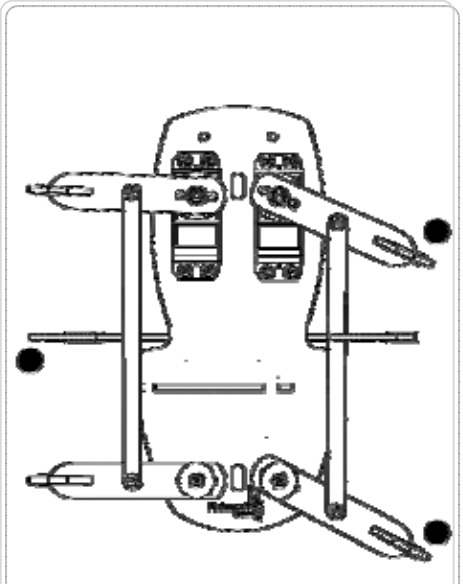
Step 1



Step 2



Step 3



Step 4

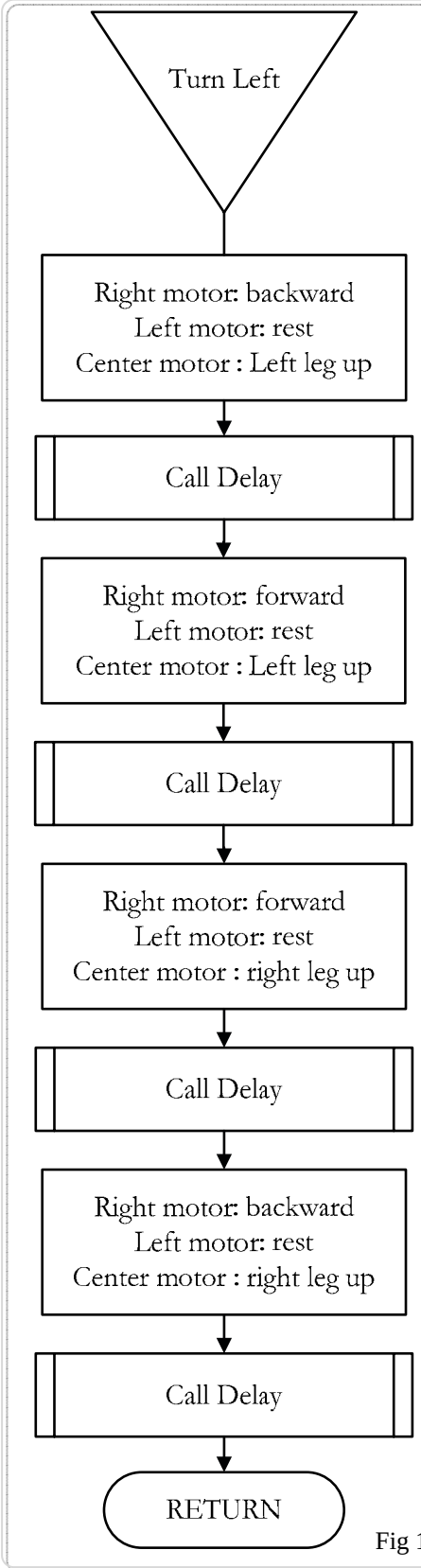


Fig 11

Turning right

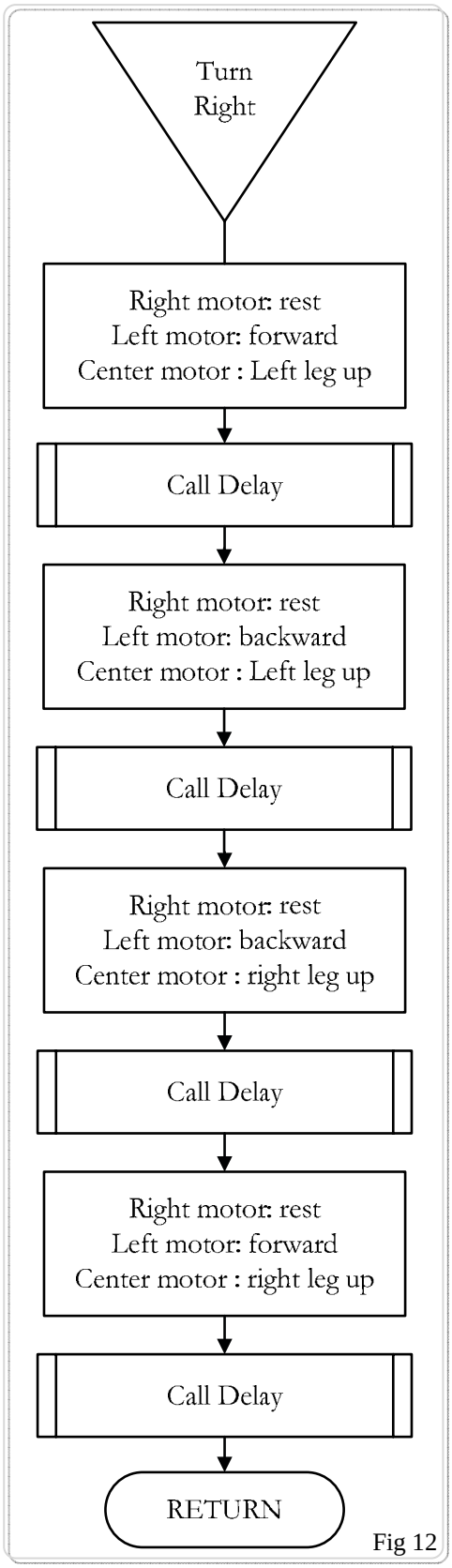
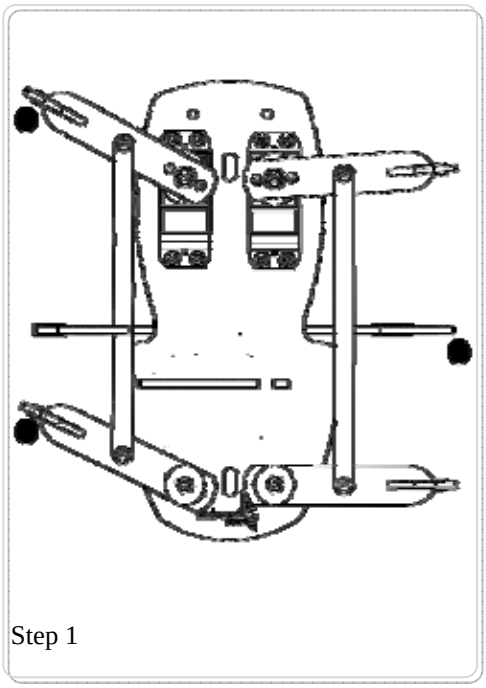
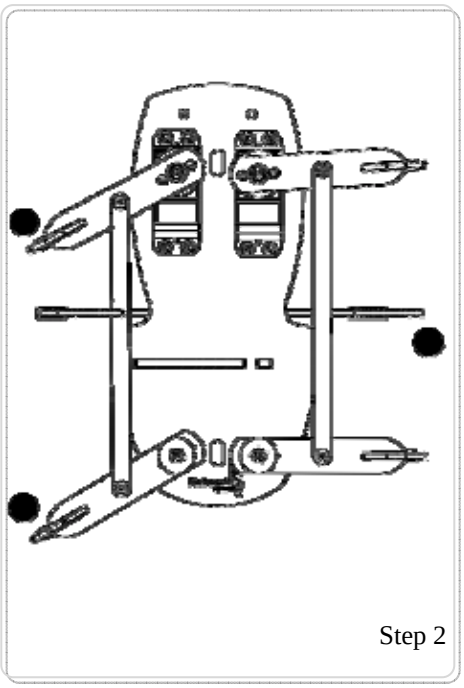


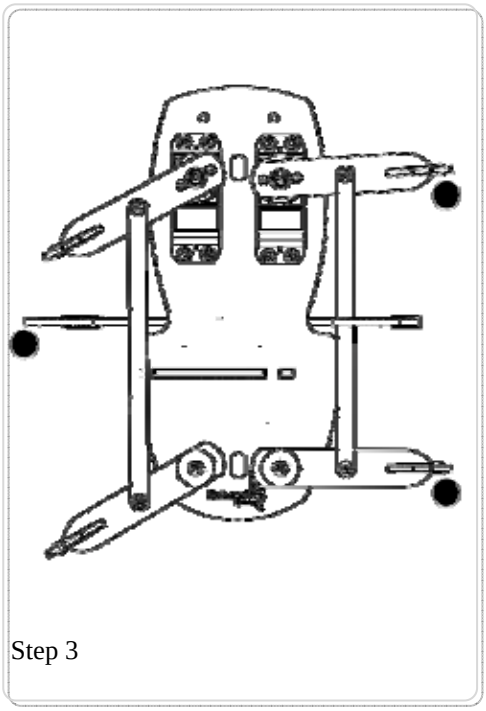
Fig 12



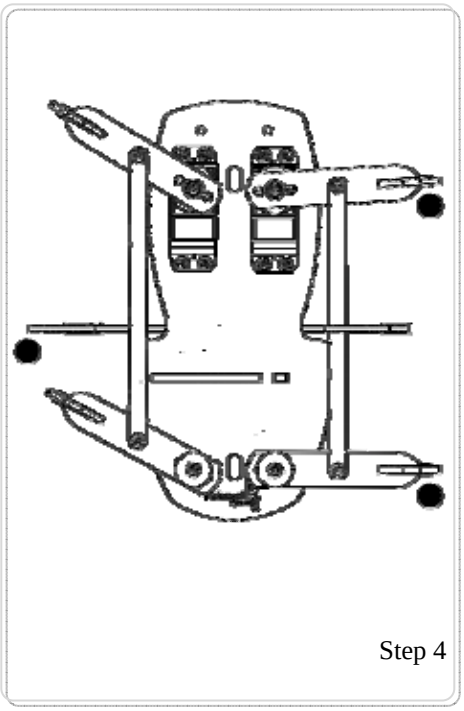
Step 1



Step 2



Step 3



Step 4

Infra Red fundamentals

Infrared radiation is electromagnetic radiation whose wavelength is longer than that of visible light (400-700nm), but shorter than that of terahertz radiation (100μm - 1mm) and microwaves (~30,000μm). Infrared radiation spans roughly three orders of magnitude (750nm and 100μm). "Near infrared" light is closest in wavelength to visible light and "far infrared" is closer to the microwave region of the electromagnetic spectrum.

Infrared produces analog outputs. It is used in short and medium range communications and control. Some systems operate in *line-of-sight mode*; this means that there must be a visually unobstructed straight line through space between the transmitter (source) and receiver (destination). Other systems operate in *diffuse mode*, also called *scatter mode*. This type of system can function when the source and destination are not directly visible to each other. An example is a television remote-control box. The box does not have to be pointed directly at the set, although the box must be in the same room as the set, or just outside the room with the door open.

Unlike radio-frequency (RF) wireless links, IR wireless cannot pass through walls. Therefore, IR communications or control is generally not possible between different rooms in a house, or between different houses in a neighbourhood (unless they have facing windows). This might seem like a disadvantage, but IR wireless is more private than RF wireless. Some IR wireless schemes offer a level of security comparable to that of hard-wired systems. It is difficult, for example, to eavesdrop on a well-engineered, line-of-sight, IR laser communications link.

In our robot we are using IR sensors for two purposes.

- Navigation
- Obstacle avoidance

The IR transmitter is nothing but IR LED (Light Emitting Diode). But there is a difference in the IR receiver that has been used. They are not simple IR receivers, but they are high-performance TSOP's (Thin Small Outline Packages).

Thin Small Outline Package

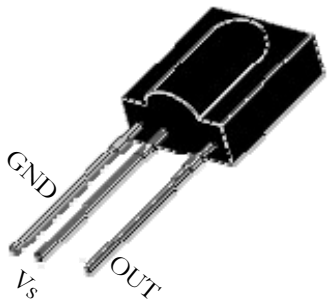


Fig 13

The TSOP series are miniaturized receivers for infrared remote control systems. TSOP is the standard IR remote control receiver series, supporting all major transmission codes. The TSOP has internal filter for PCM frequency. The output of TSOP is high if there is no IR light falling on the TSOP. If there are IR rays falling on the TSOP the output goes active low. As the outputs are always high or low, the output from the TSOP can be directly read by the microcontroller. The one that we are using is TSOP 1738. The operating frequency range of this sensor is 32KHz to 38KHz.

Features of TSOP 1738

- Photo detector and preamplifier in one package
- Internal filter for PCM frequency
- TTL and CMOS compatibility
- Output active low
- High immunity against ambient light
- Continuous data transmission possible (≤ 2400 bps).

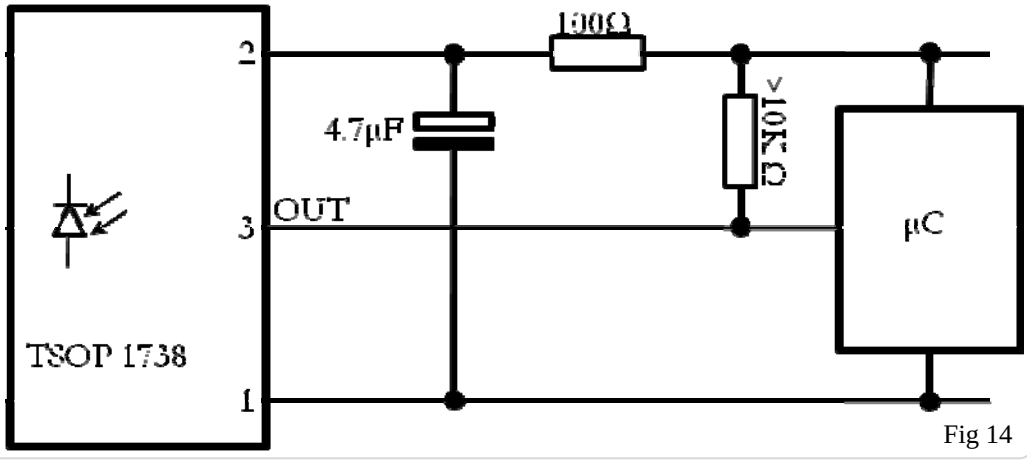


Fig 14

The diagram beside shows how to interface a TSOP 1738 receiver with a microcontroller. The 10K resistance is optional though.

Fig 15

Our robot can also be controlled using IR remote. This option has been provided so that you can control the robot wirelessly according to your wishes. For this to happen you have to program the robot to respond to the buttons pressed on the remote. As we had done previously, we will illustrate the flowchart for navigation using the remote control. The detailed program can be found on the disc.

The remote control that has been provided in this kit has been shown beside. According to the remote, we have defined certain variables in the program. These variables and their set conditions are mentioned below.

FWD : is set when forward key is pressed

BWD : is set when Backward key is pressed

RIGHT : is set when Right key is pressed

LEFT : is set when Left key is pressed

STOP : is set when center stop key is pressed

NUM0 : is set when Num '0' key is pressed

NUM1 : is set when Num '1' key is pressed

NUM2 : is set when Num '2' key is pressed

NUM3 : is set when Num '3' key is pressed

NUM4 : is set when Num '4' key is pressed

NUM5 : is set when Num '5' key is pressed

NUM6 : is set when Num '6' key is pressed

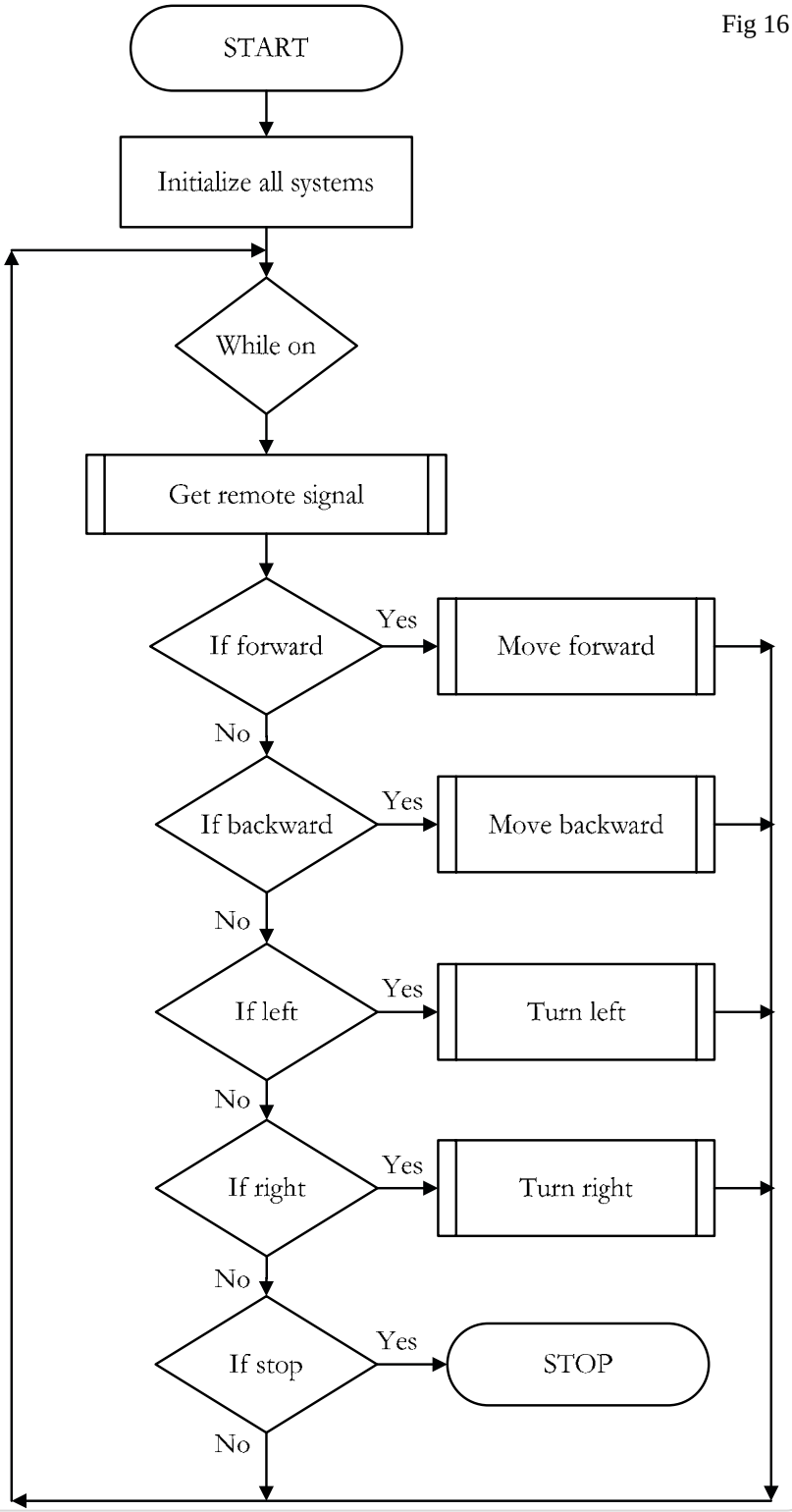
NUM7 : is set when Num '7' key is pressed

NUM8 : is set when Num '8' key is pressed

NUM9 : is set when Num '9' key is pressed

These variables should be used when you try to move the machine using the remote control. You can even define your own variables if you do not want to use the library.

Fig 16


START
Initialize all systems
While on
Get remote signal
If forward
Yes
Move forward
No
If backward
Yes
Move backward
No
If left
Yes
Turn left
No
If right
Yes
Turn right
No
If stop
Yes
STOP
No


POWER
MODE
1
2
3
4
VOL+
5
PP/SCAN
6
7
VOL-
8
9
0
*
10+
*
VOL+
VOL-
PAGARIA
6883

Fig 17

Assembling the HexaBOT

Below we list all the steps for assembling the machine. If any explanation is required, then we will give them at appropriate places.

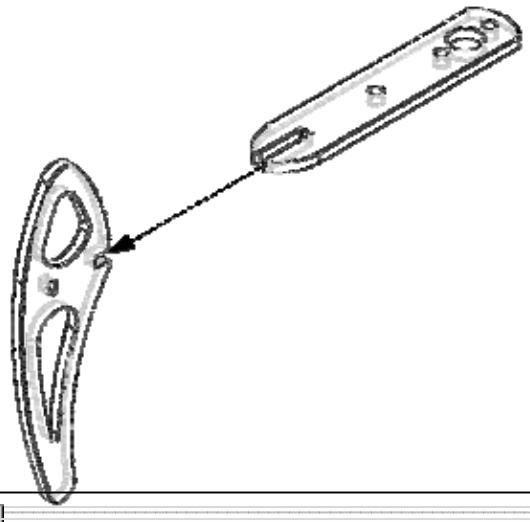


Fig 18

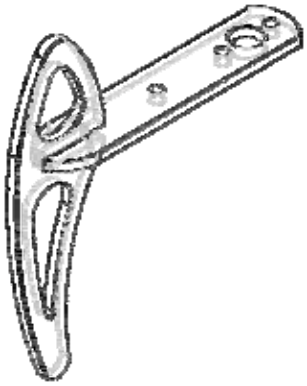


Fig 19

Fig 20

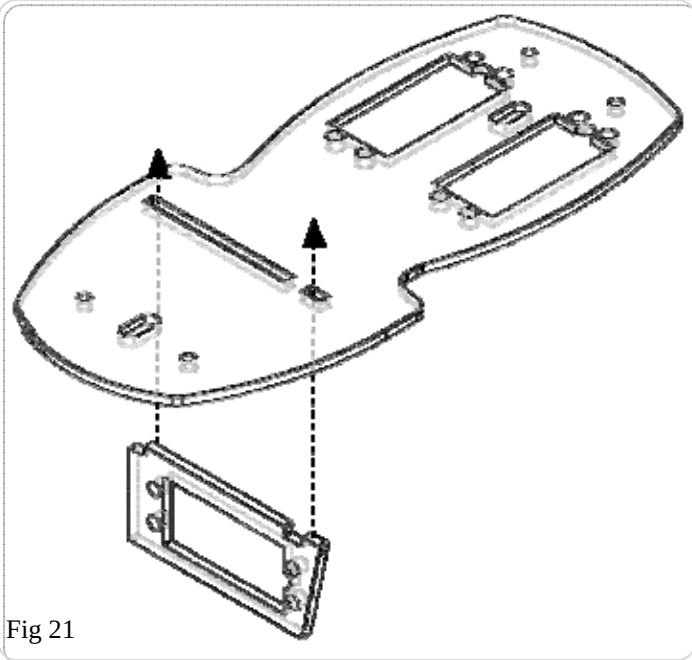


Fig 21

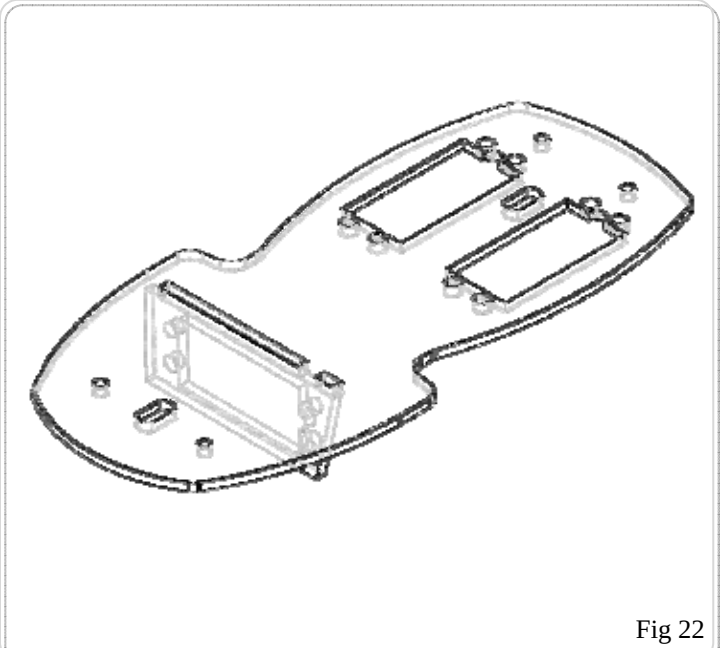


Fig 22

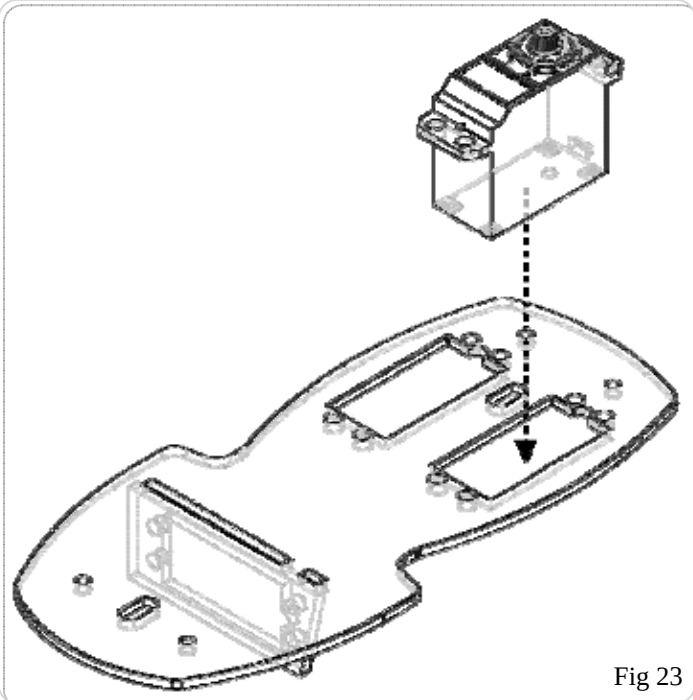


Fig 23

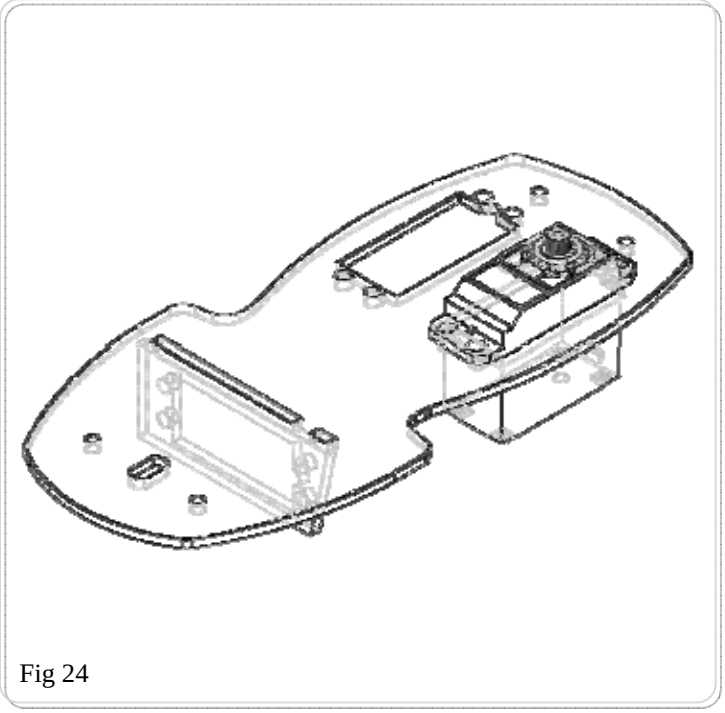


Fig 24

In the steps that we have shown, you must have noticed that we have not used any screws to fix the legs of the HexaBOT. To fix these legs, you have to use any industrial strength adhesive like FeviKwik®. But please take care to fix the legs in exactly 90°. If you fail to do so, then your robot will not follow a straight line while moving forward or backward. Also take care not to glue your fingers to the acrylic parts.

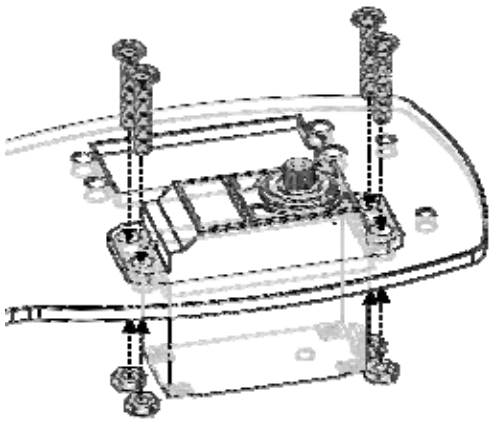


Fig 25

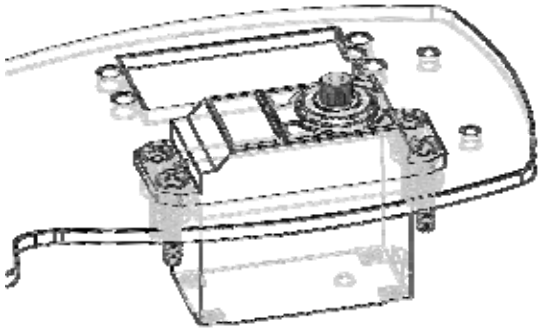


Fig 26

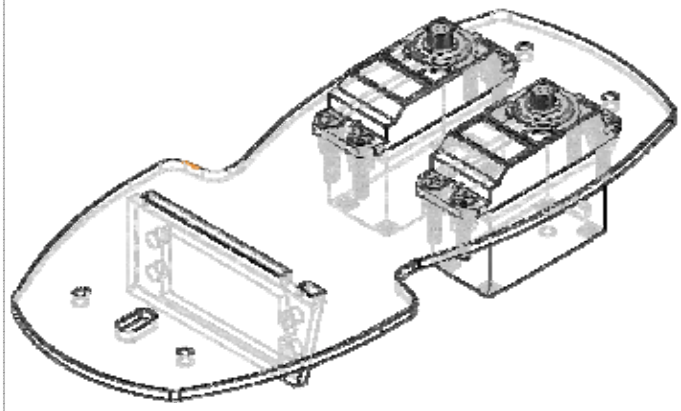


Fig 27

To fix these motors, you have to keep in mind the direction that you are putting them in. You must have noticed that there is only one side of the motor which has the wires coming out of it. The set of three wires originate through a rubber stub. On the board, we have provided notches on one side of the motor holes. The rubber stubs will go through those notches. Do not exert more pressure than required.

The easiest way to glue the acrylic components together is to first attach the components with your hands in exactly the way you want and then apply FeviKwik® on the corners where the components meet each other.

Be extra careful when you glue the red acrylic rectangular frame to the black acrylic body. Sometimes the rectangular part refuses to fit snugly into the black body. Give very light pressure when you try to fit that component in. Once when you have put that in, just apply FeviKwik® from the top layer. Also you have to keep in mind that the shaft of the motors will come on the opposite side of the Robosoft Systems® logo.

Fig 28

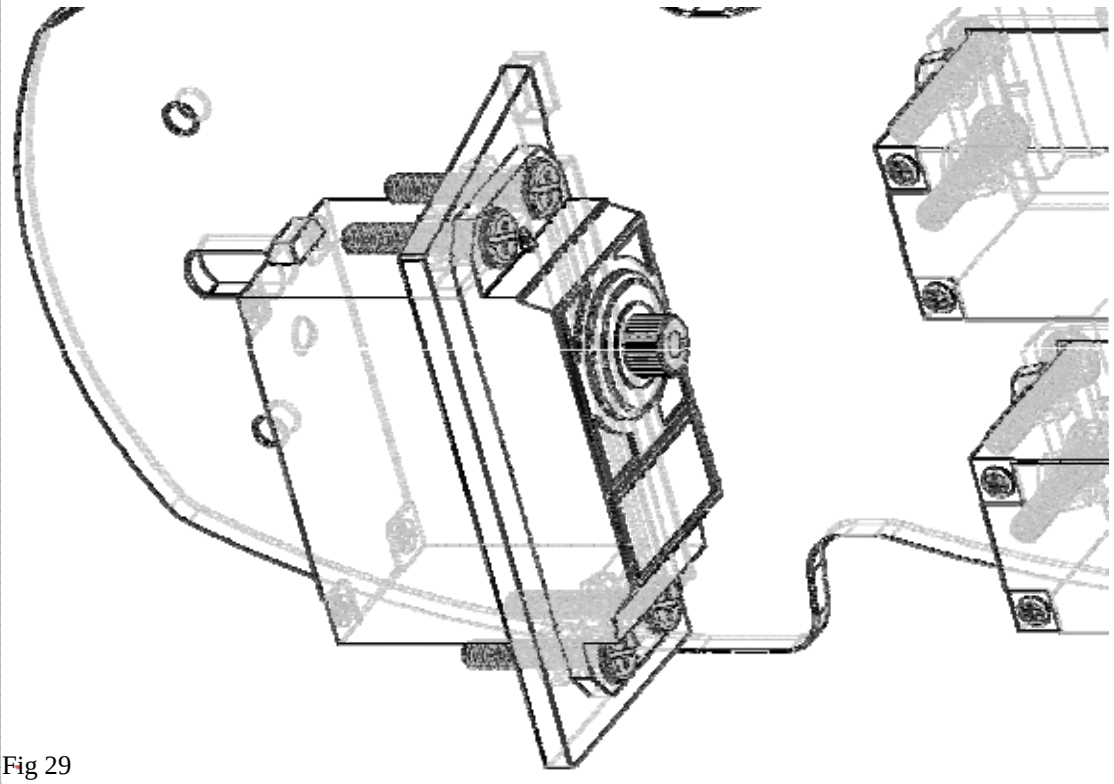


Fig 29

Besides we have shown two figures which gives you an exploded view from the bottom of the robot. It shows you how to mount the motor from the bottom of the robot into the red rectangular acrylic frame. You have to keep in mind that the shaft of the motor will face the other two motors. Similar to the other two motors, we have provided a notch for the wire of the motor to go in.

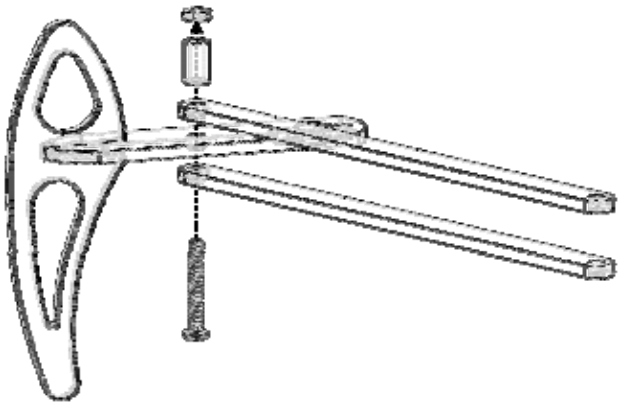


Fig 30

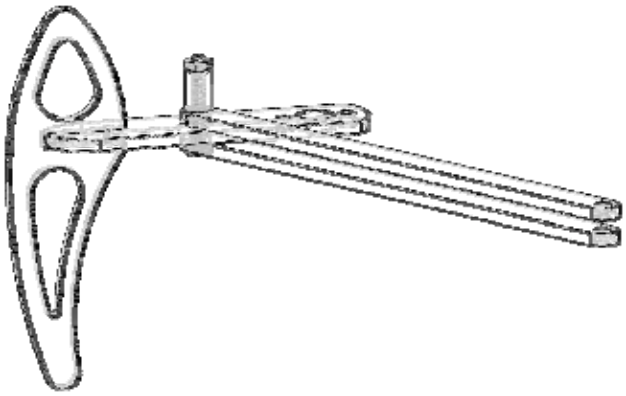


Fig 31

Here we have listed put the steps to mount the black acrylic strips on the legs. For this you will need the smaller white plastic spacers. The screws that you will require, are, 3X15mm and 3mm nuts. The screws will come from the bottom and the spacers from the top. After that you will have to tighten the nuts from above. But while tightening the nuts, do not over tighten them; otherwise the robot will lose flexibility.

Fig 32

Fig 33

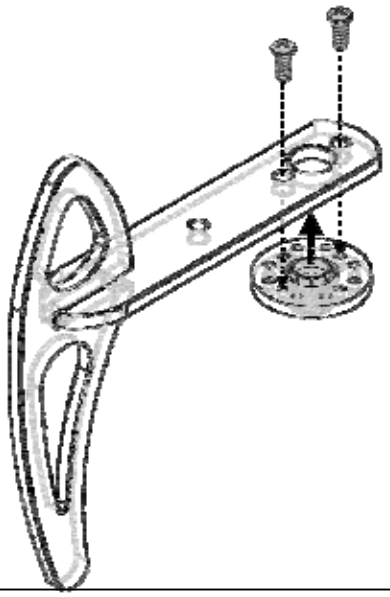


Fig 34

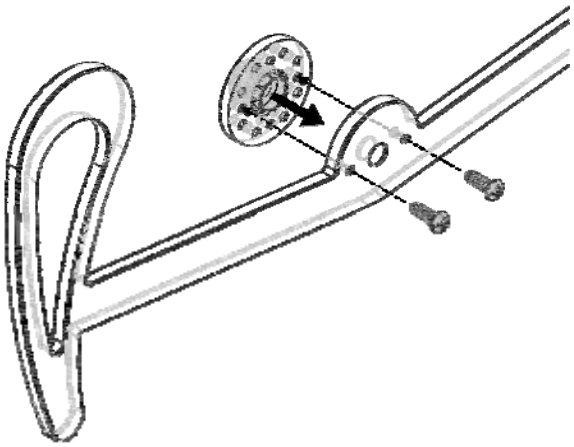


Fig 35

Fig 36

Fig 37

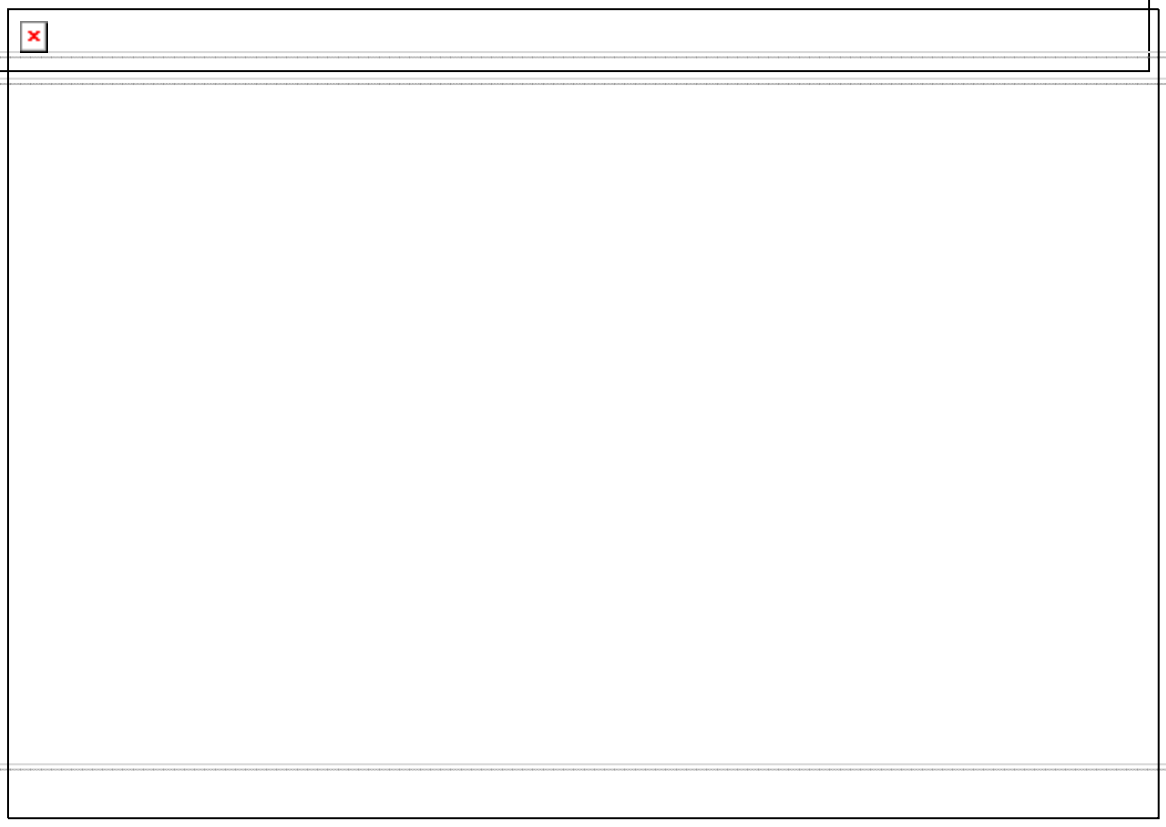


Fig 38

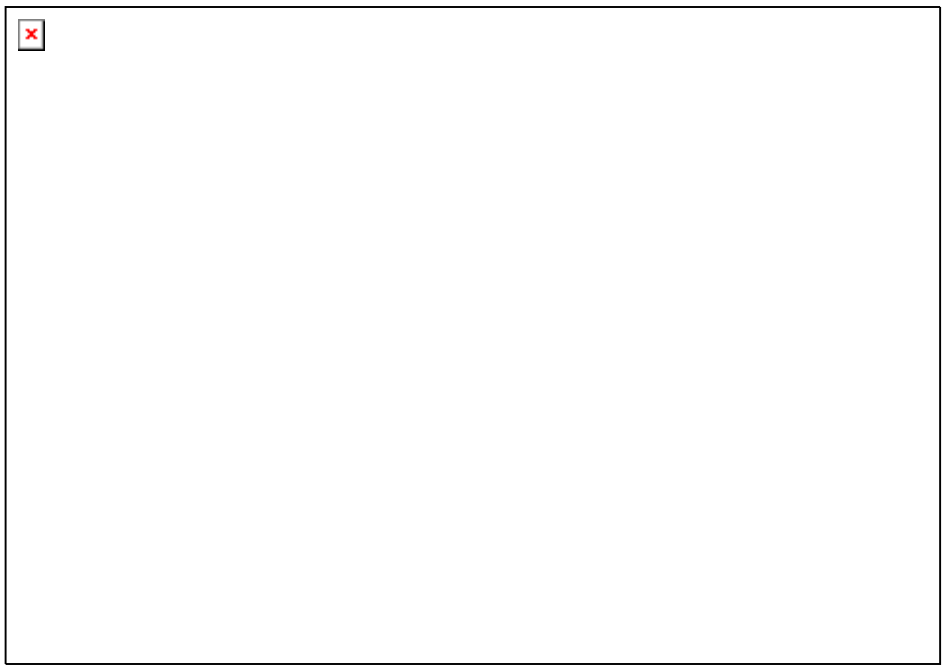


Fig 39

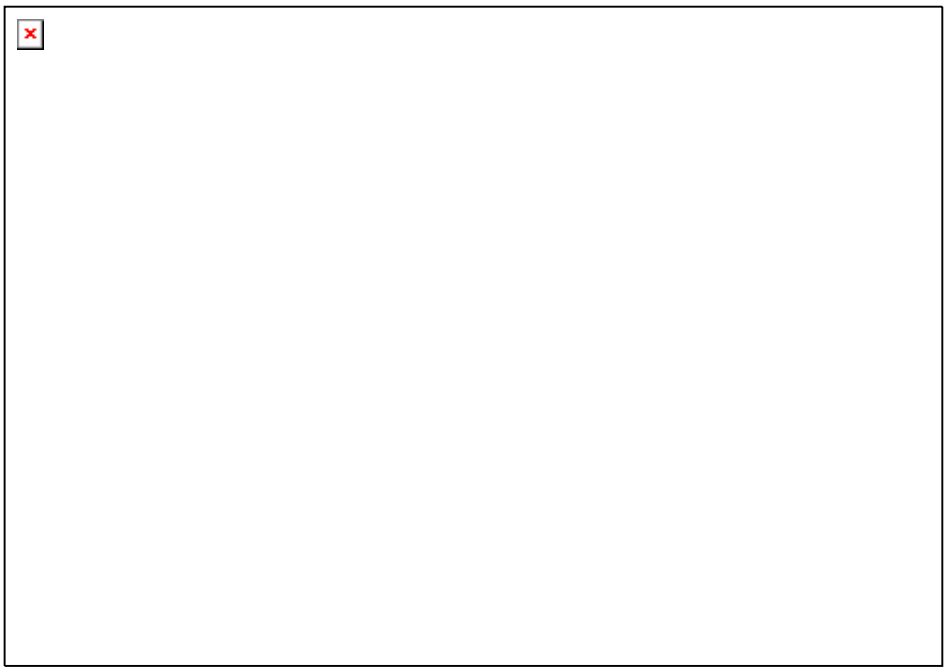


Fig 40

The fig. 39 and 40 shows how you can attach the bottom leg to the bottom servo motor.

The fig. 41 and 42 show the final steps in assembling the machine. After everything else is complete, you will have to put the black rubber shoes onto the legs of the HexaBOT. Fig. 42 shows the final assembled view of the machine as everything else is complete. Only the thing that is remaining is to mount the electronic board and the battery to the machine and your machine will be ready. You will just need to program the machine to perform various kinds of motions.

Fig 41

Fig 42

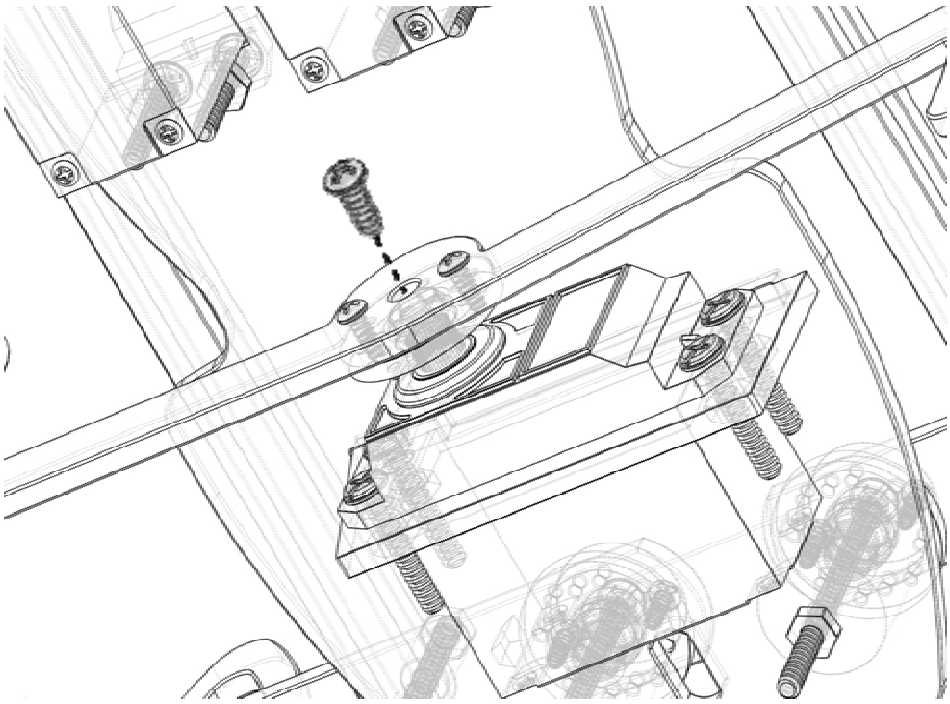


Fig 39

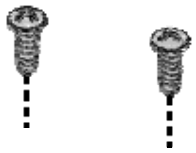
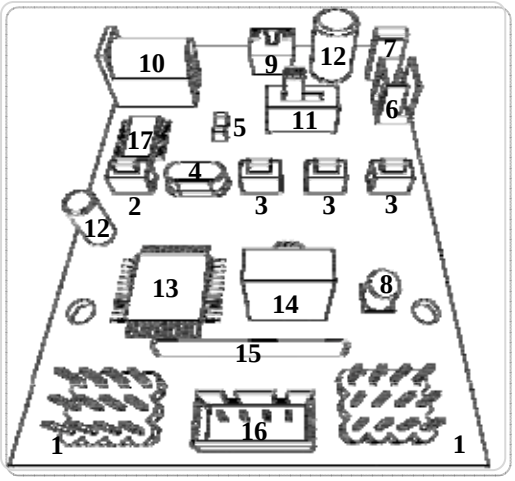


Fig 40

Know your board

The microcontroller board



Beside, we have shown a schematic of the actual electronic board that has been provided with the HexaBOT. You can say that this is the brain of the robot. It contains all the controller IC's and other circuitry so that you can write your own programs and make the HexaBOT walk according your wish. You have to mount the board on the HexaBOT using a double-sided tape, so that the board does not fall of from the robot when it walks. Below we have listed some of the important components that are worth listing.

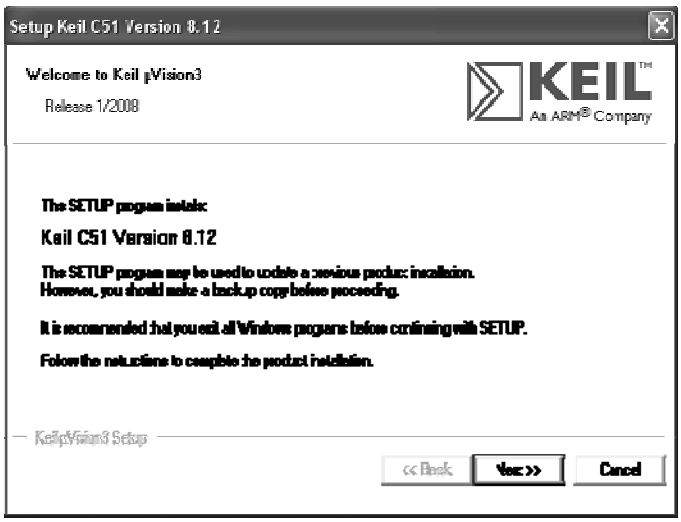
Here are the components along with their uses:

1. Servo motor connector	You can connect the servo motors to these
2. Reset switch	You can reset the microcontroller using this switch
3. Other switches	These switches perform various inputs
4. Crystal	This provide the clock signals to the microcontroller
5. Diodes	This gives protection from reverse current
6. Battery connector	You can connect the battery to this socket
7. Jumper	This provides Vcc to the servo motors
8. SMD capacitor	It provides noise filtering for the TSOP sensor
9. USB connector	You can connect the board to your computer through this
10. Battery charger	You can connect the charger to charge your battery
11. Power switch	This switches the board between USB or Battery supply
12. Capacitors	These are used to store the charges
13. Microcontroller	It is used to store the programs and make your HexaBOT run
14. TSOP IR receiver	It is used to receive the IR remote control signals
15. Pull up resistor	They pull up the current of Port 1
16. Sensor connector	You can connect the IR sensor module to this

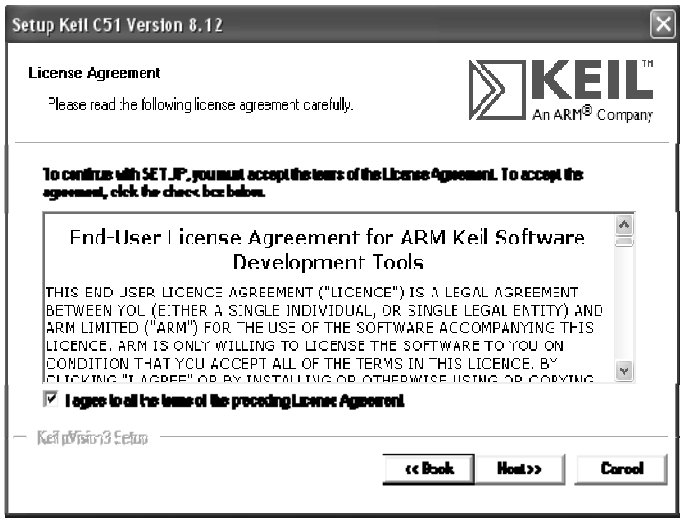
Installing the softwares

Installing KEIL™

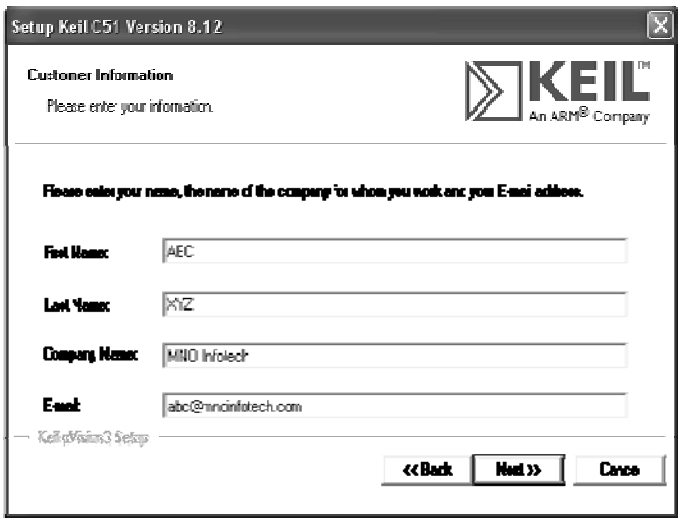
Below we have graphically listed the steps required to install KEIL™ onto your computer. Change the parameters as necessary. You will have to run the file "c51v812.exe" in the folder "\Keil-uVision-v8.12-C51" which you will find in the "Software" folder in your disc.



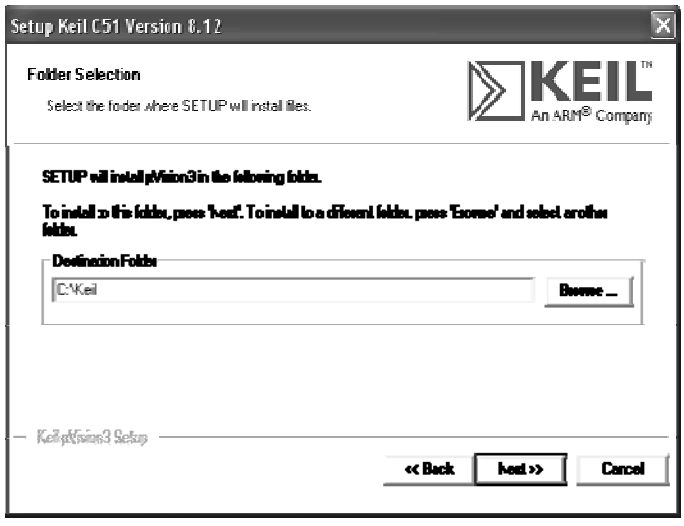
Step 1



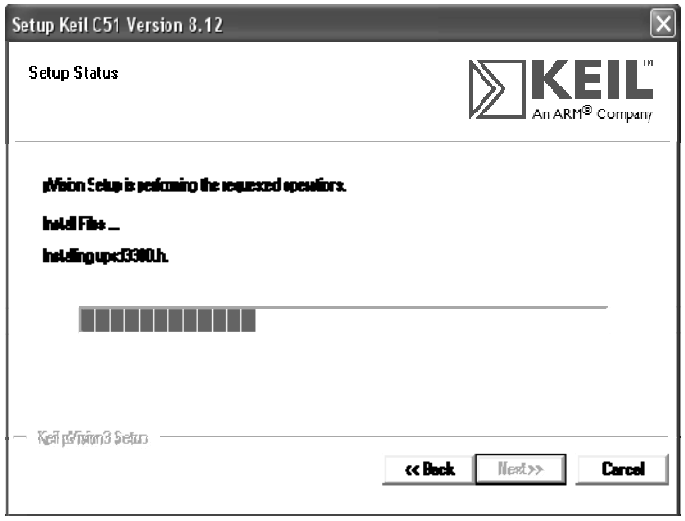
Step 2



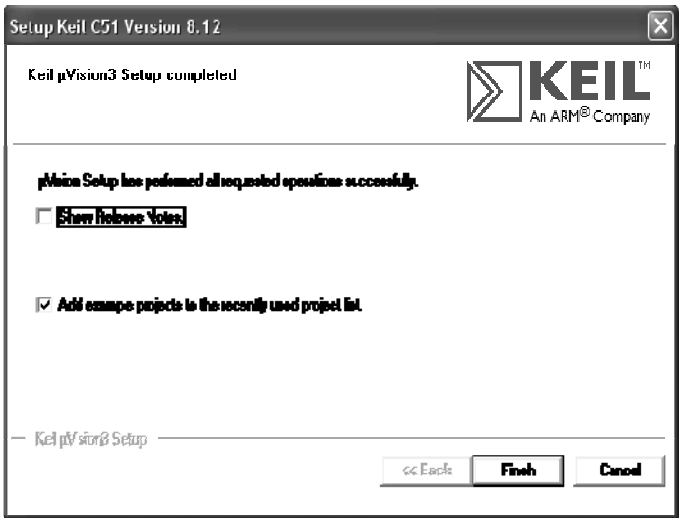
Step 3



Step 4



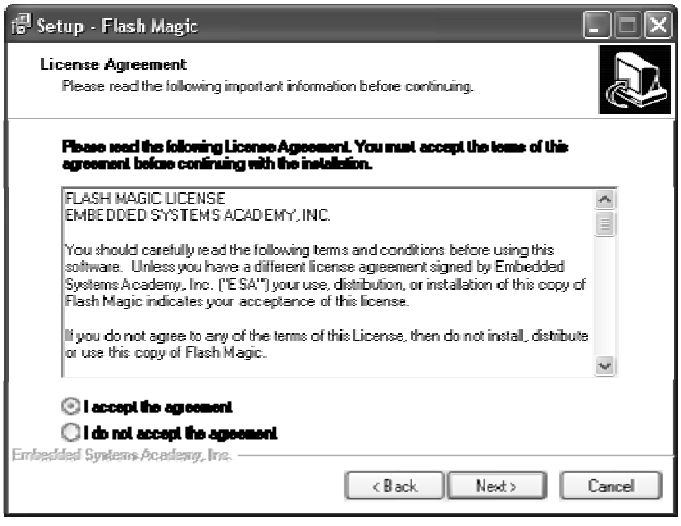
Step 5



Step 6

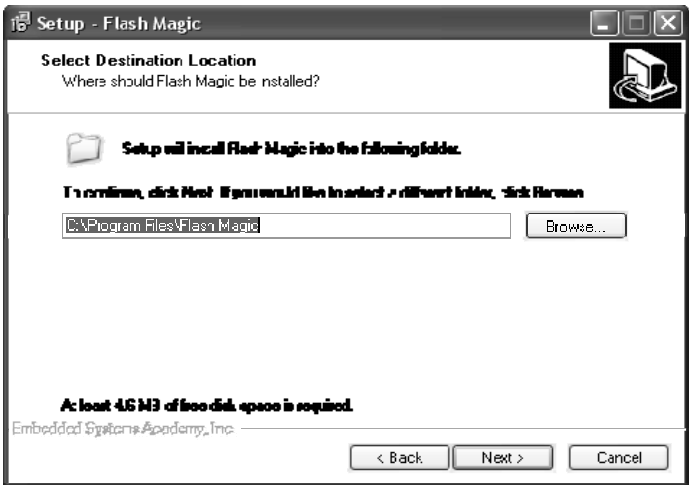
Installing Flash Magic

Step 1

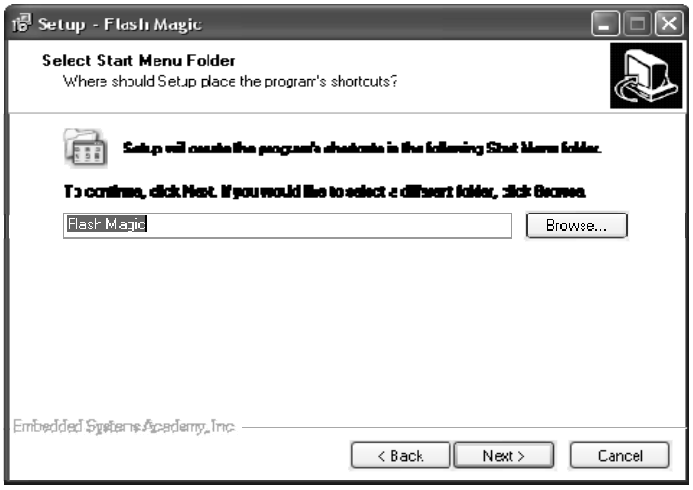


Step 2

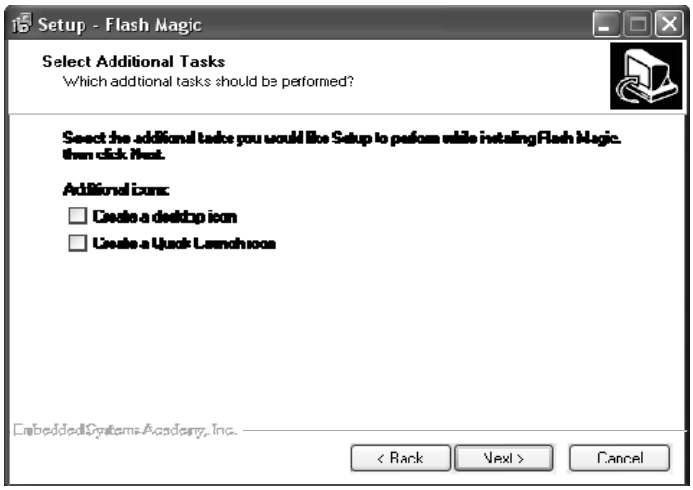
The figures that we have provided here are only for demonstrational purposes. You will have to fill in the exact parameters in the boxes as per your requirement. For details on how to use the softwares, please refer to the documentation provided in the accompanied disc.



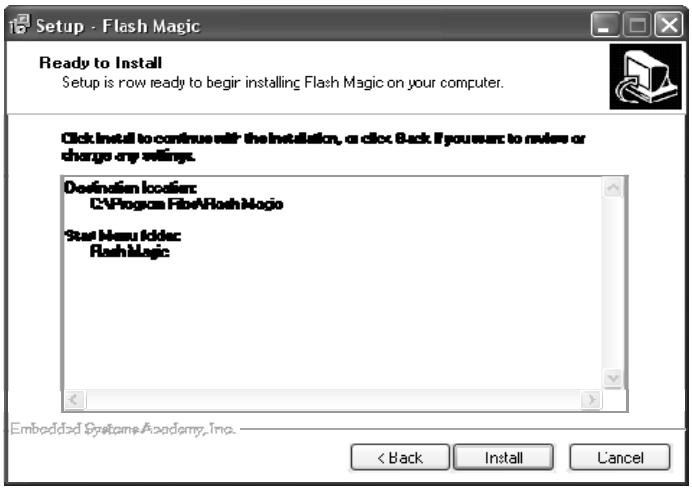
Step 3



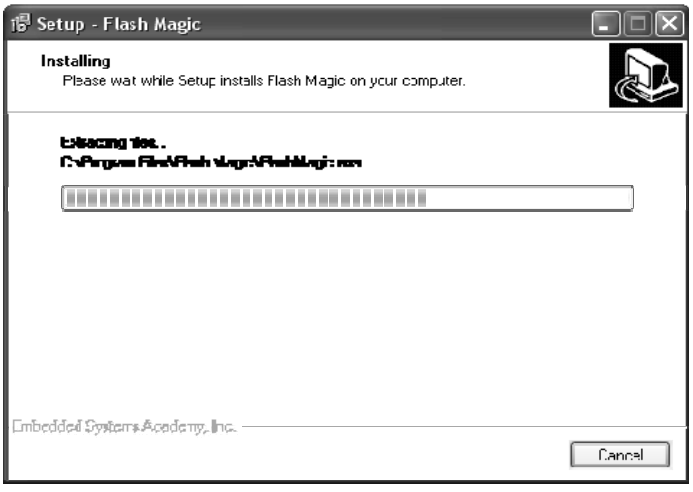
Step 4



Step 5



Step 6



Step 7

Step 8

Interfacing the board

Fig. 1

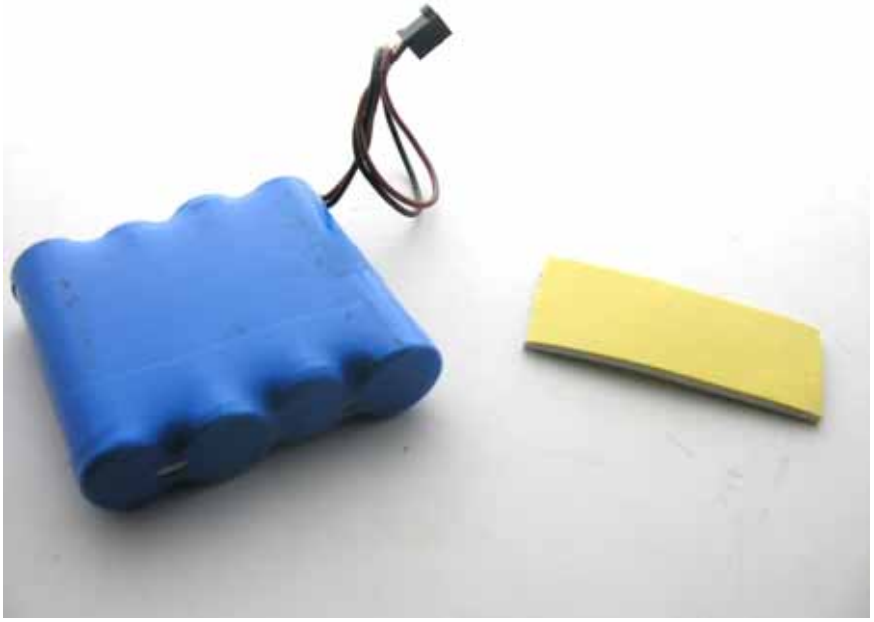


Fig. 2

Fig. 3

Fig. 4

Fig. 5

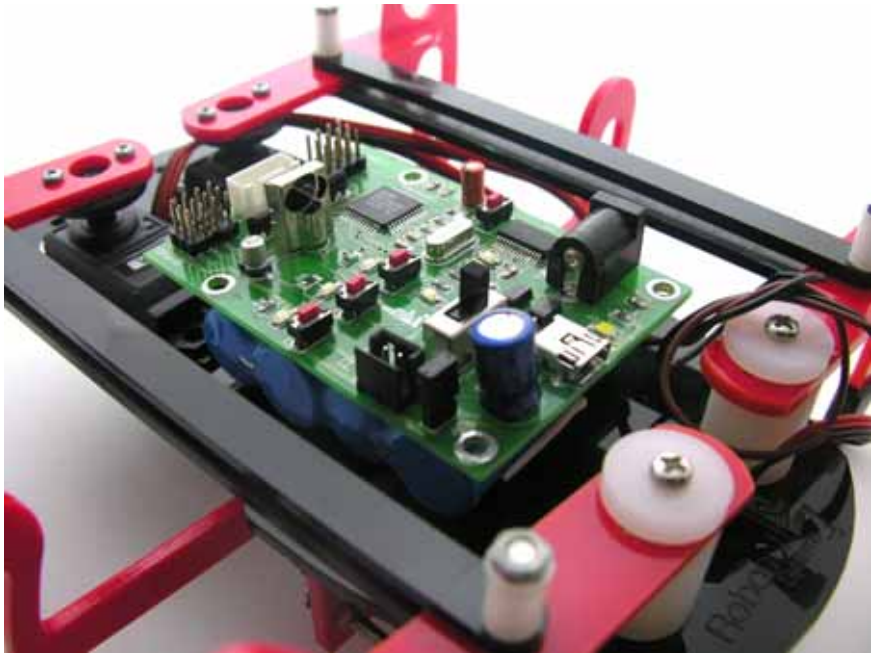


Fig. 6

Fig. 7

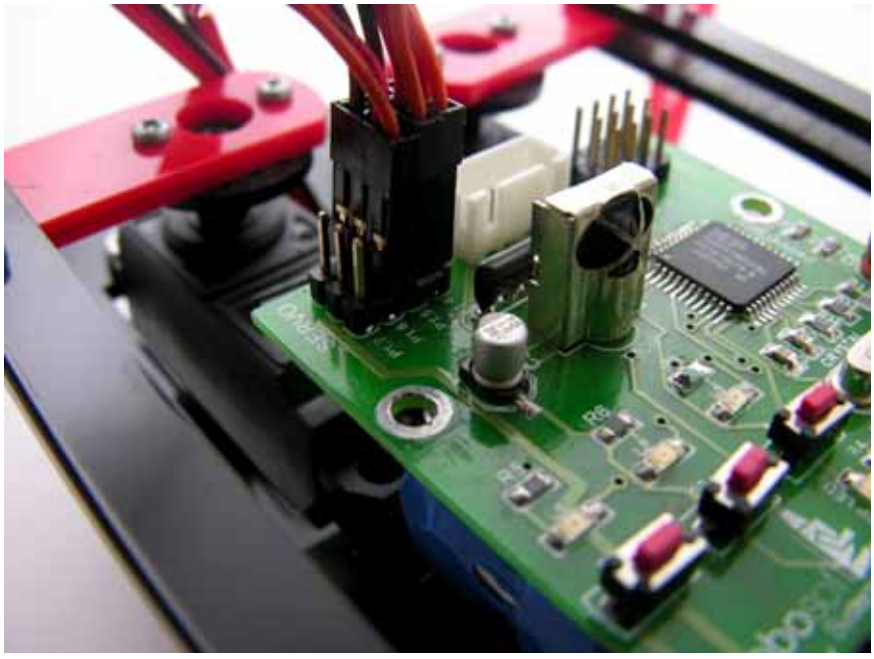


Fig. 8

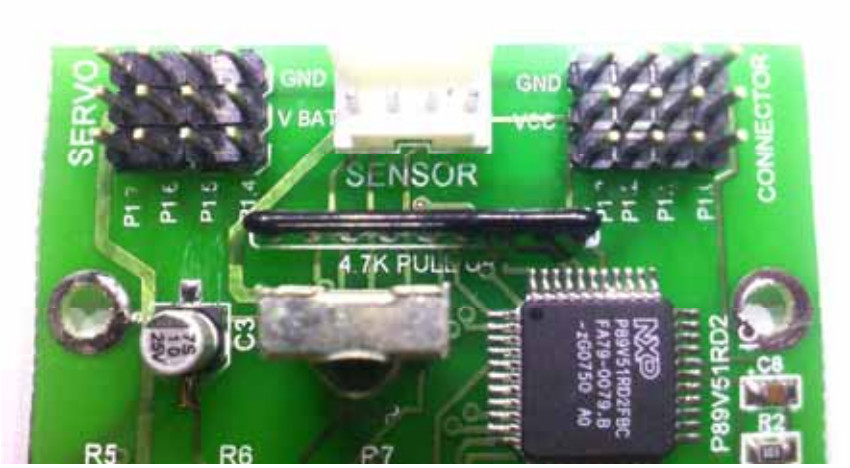


Fig. 9

Servo Connection	
P1.7	EM
P1.6	LM
P1.5	CM
P1.4	RM

Abbreviation	
EM	Extra Motor
LM	Left Motor
RM	Right Motor
CM	Center Motor

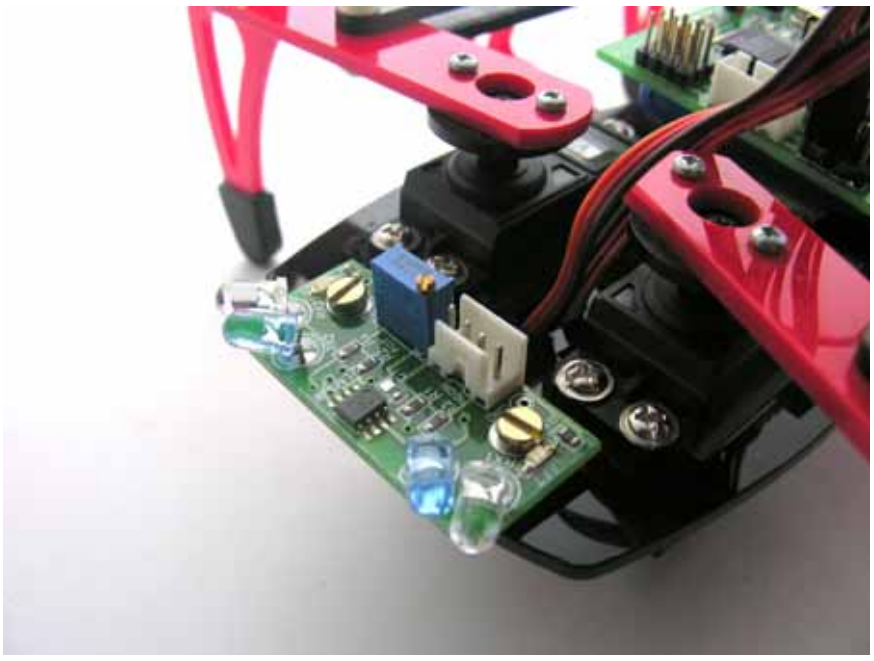


Fig. 10

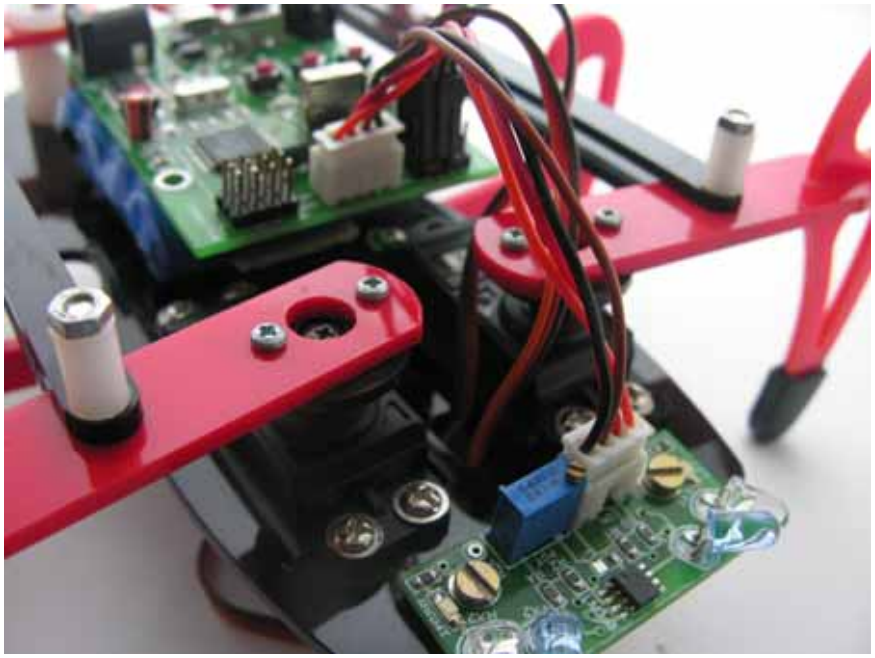


Fig. 11

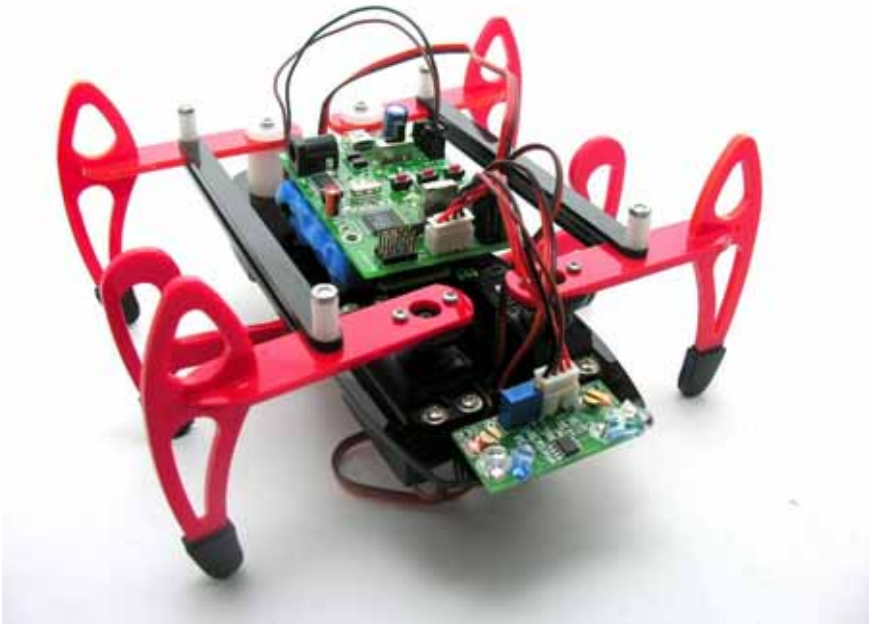
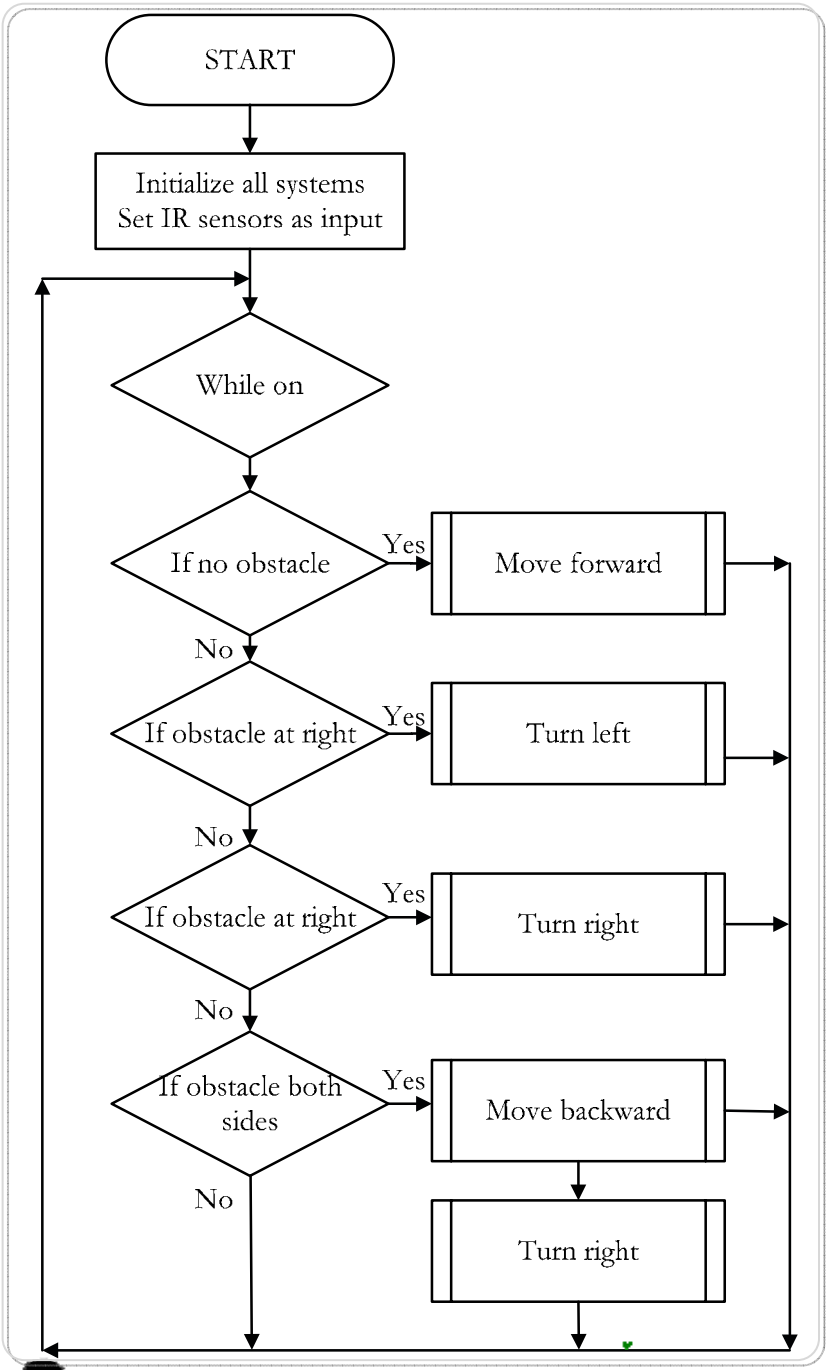
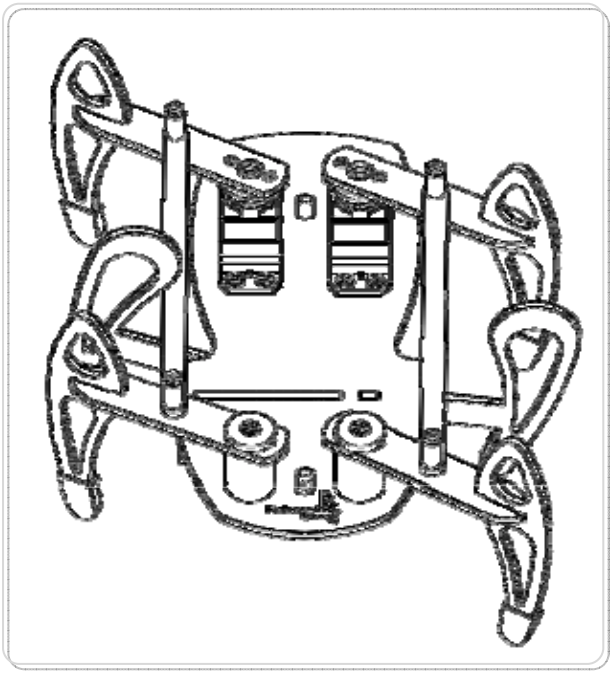


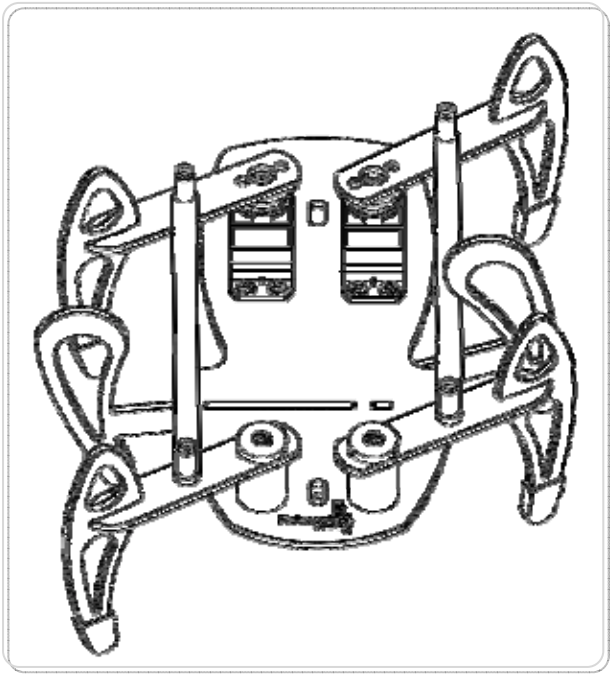
Fig. 12

HexaBOT PRO

Want to go pro now? Let's try to make our HexaBOT work autonomously, shall we? That means we will make use of the obstacle detector module and make the HexaBOT work by itself and avoid obstacles by itself. For detailed program, please







A-61, 1 st Floor, Raj Industrial Complex,
Military Road, Marol,
Andheri (East),
Mumbai – 400059
India

Contact Information
Office: (91-22) 66751507
Office: (91-22) 29207086
Mobile: (91) 9930776661

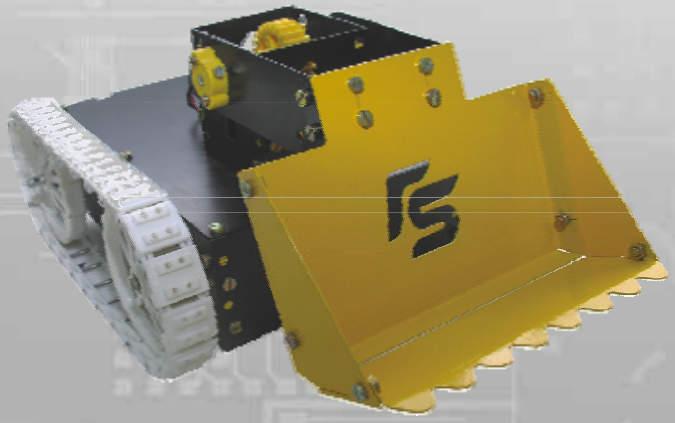
Email: info@robosoftsystems.co.in

Visit us at
www.robosoftsystems.co.in

Liked our HexaBot? Want more?

Robo DUMSTER

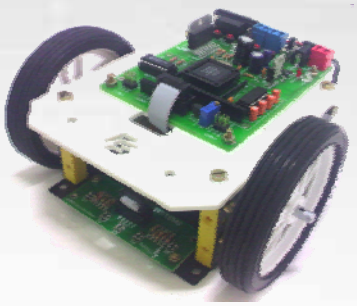
Features :
Artistic laser cut machine
Easy to build
Attractive LED Joystick



Beginner level

Programming not required

Robo BRAINS

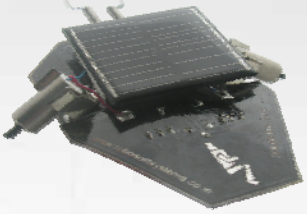


Expert level

Programming required

Features :
Attractive mechanical assembly
µC board for autonomous operation
Line following sensor
Fire and proximity sensor

Solar Bug



Intermediate level

Programming not required

Features :
Autonomous robot.
Light seeking capability.
Obstacle avoidance.
Solar powered
Easy to build

Robosoft Systems