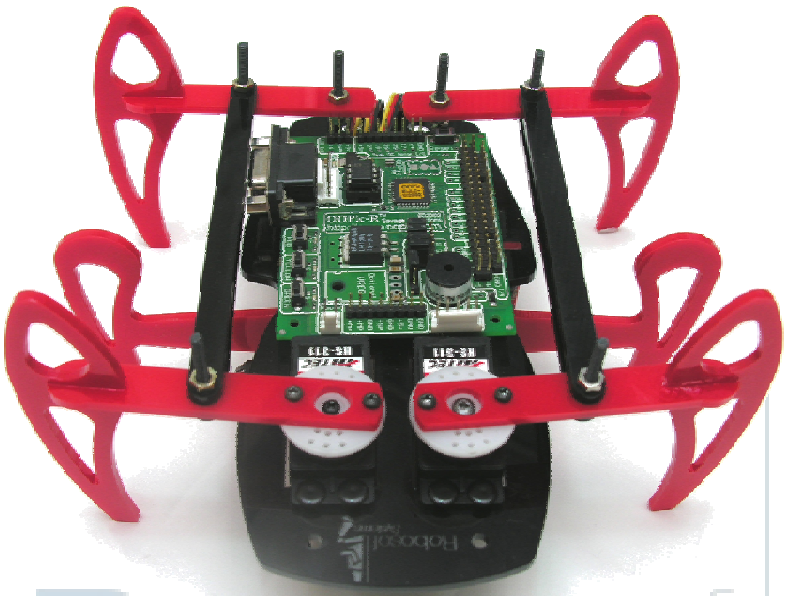
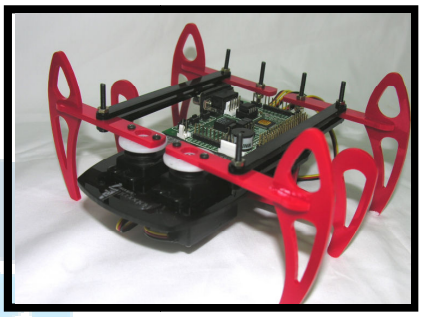
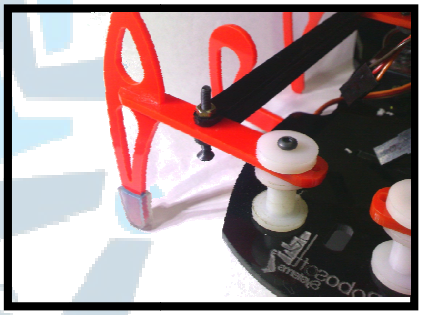
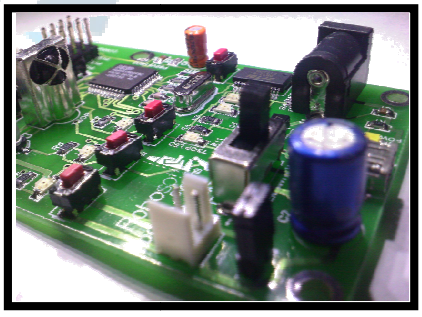
Robosoft Systems

ROBOCRAB The Hexabot









ROBOCRAB the Hexabot is robot developed for advance robotics workshop designed & manufactured by Robosoft Systems ® for school children, This robotics workshop will be learning developing legged and autonomous robot by applying basic law of physics, math's and of course basic concept of robotics.

Users Manual

Robosoft Systems

Numeral System

A numeral system (or system of numeration) is a writing system for expressing numerals.

- Represent a useful set of numbers (e.g. all whole numbers, integers, or real numbers)
- Give every number represented a unique representation (or at least a standard representation)
- Reflect the algebraic and arithmetic structure of the numbers.

Numeral systems are sometimes called *number systems*, but that name is misleading, as it could refer to different systems of numbers, such as the system of real numbers, the system of complex numbers.

Do you know which numeral systems we use?

It is “Decimal” numeral system

The **decimal** (**base ten** or occasionally **denary**) numeral system has ten as its base. It is the most widely used numeral system.

	Ten Thousands	Thousand	Hundreds	Tens	Ones
Starts From >>	0	0	0	0	0
Ends To >>	9	9	9	9	9

Binary Number (**base 2**) Numeral System

To understand binary numbers, begin by recalling elementary school math. When we first learned about numbers, we were taught that, in the decimal system, things are organized into columns:

H	T	O
1	9	3

Such that "H" is the hundreds column, "T" is the tens column, and "O" is the ones column. So the number "193" is 1-hundreds plus 9-tens plus 3-ones.

Years later, we learned that the ones column meant 10^0, the tens column meant 10^1, the hundreds column 10^2 and so on, such that

10^2	10^1	10^0
1	9	3

The number 193 is really { $(1*10^2) + (9*10^1) + (3*10^0)$ }.

As you know, the decimal system uses the digits 0-9 to represent numbers. If we wanted to put a larger number in column 10^n (e.g., 10), we would have to multiply $10*10^n$, which would give $10^{(n+1)}$, and be carried a column to the left. For example, putting ten in the 10^0 column is impossible, so we put a 1 in the 10^1 column, and a 0 in the 10^0 column, thus using two columns.

The binary system works under the exact same principles as the decimal system, only it operates in base 2 rather than base 10. In other words, instead of columns being

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

Instead of using the digits 0-9, we only use 0-1 (again, if we used anything larger it would be like multiplying $2*2^n$ and getting $2^{(n+1)}$, which would not fit in the 2^n column. Therefore, it would shift you one column to the left. For example, "3" in binary cannot be put into one column. The first column we fill is the right-most column, which is 2^0, or 1. Since 3 > 1, we need to use an extra column to the left, and indicate it as "11" in binary $(1*2^1) + (1*2^0)$.

Number Systems

Decimal	Binary	Ternary	Octal	Hexadecimal
0,1,2…,8,9	0,1	0,1,2	0,1,…6,7	0,1,..,E,F
0	0	0	0	0
1	1	1	1	1
2	10	2	2	2
3	11	10	3	3
4	100	11	4	4
5	101	12	5	5
6	110	20	6	6
7	111	21	7	7
8	1000	22	10	8
9	1001	100	11	9
10				A
11				B
12				C
13				D
14				E
15				F
16				10
17				11
18				
19				
20				
21				

Exercise 1

1. Try converting these numbers from binary to decimal:

- 10
- 111
- 1010
- 1100
- 1111

Remember:

2^3	2^2	2^1	2^0	decimal
		1	0	
	1	1	1	
1	0	1	0	
1	1	0	0	
1	1	1	1	

2. Try converting these numbers from decimal to binary:

- 12
- 4
- 9
- 11

Remember:

decimal	2^3	2^2	2^1	2^0
12				
4				
9				
11				

Note: This Page intentionally kept blank

Digital Logic System

A logic gate performs a logical operation on one or more logic inputs and produces a single logic output. The logic normally performed is Boolean logic and is most commonly found in digital. Logic gates are primarily implemented electronically using diodes or transistors, but can also be constructed using electromagnetic relays, optics, molecules, or fluidics, even mechanical elements.

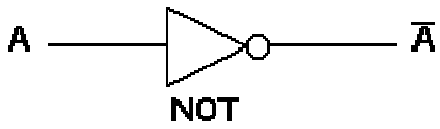
Boolean functions may be practically implemented by using electronic gates. The following points are important to understand.

- Electronic gates require a power supply.
- Gate INPUTS are driven by voltages having two nominal values, e.g. 0V and 5V representing logic 0 and logic 1 respectively.
- The OUTPUT of a gate provides two nominal values of voltage only, e.g. 0V and 5V representing logic 0 and logic 1 respectively. In general, there is only one output to a logic gate except in some special cases.
- There is always a time delay between an input being applied and the output responding.

NOT Gate:

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs.

NOT Gate Symbol



OR Gate:

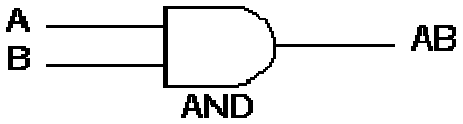
The OR gate gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false."

OR Gate Symbol

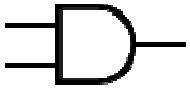
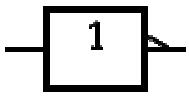
AND Gate:

The AND gate is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false."

AND Gate Symbol



LOGIC Gates Details:

<table>
<tr><th>Type</th><th>Distinctive shape</th><th>Rectangular shape</th><th>Boolean</th><th>C Syntax</th><th>Truth table</th></tr>
<tr><td>AND</td><td>  </td><td>  </td><td>$A \cdot B$</td><td>A && B</td><td>
<table>
<tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr>
<tr><td>A</td><td>B</td><td>A AND B</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>OR</td><td>  </td><td>  </td><td>A + B</td><td>A || B</td><td>
<table>
<tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr>
<tr><td>A</td><td>B</td><td>A OR B</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>NOT</td><td>  </td><td>  </td><td>$\overline{A}$</td><td>!A
~A</td><td>
<table>
<tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr>
<tr><td colspan="2">A</td><td>NOT A</td></tr>
<tr><td colspan="2">0</td><td>1</td></tr>
<tr><td colspan="2">1</td><td>0</td></tr>
</table>
</td></tr>
</table>

Exercise 2

1. Calculate LOGIC value for following sets

- ~10 = ( )
- ~110 = ( )
- ~1010 = ( )
- ~1100 = ( )
- ~1111 = ( )

- 10 && 01 = ( )
- 110 && 101 = ( )
- 1010 && 0101 = ( )
- 1100 && 1010 = ( )
- 1111 && 0010 = ( )

- 10 || 01 = ( )
- 110 || 101 = ( )
- 1010 || 0101 = ( )
- 1100 || 1010 = ( )
- 1111 || 0010 = ( )

- ((~1100) && (~1010)) || (0110) = ( )
- ((1101) || (1010)) && (~0110) = ( )

Exercise 3

Truth Table for NOR

C= ~(B || A);

let D = B || A

There for C=~ D

Truth Table for NOR

Input A	Input B	Output D	Output C
0	0		
0	1		
1	0		
1	1		

Truth Table for NAND

C= ~(B && A);

let D = B && A

There for C=~ D

Truth Table for NAND

Input A	Input B	Output D	Output C
0	0		
0	1		
1	0		
1	1		

Basic C Concept

C program basically has the following form:

- Preprocessor Commands
- Type definitions
- Function prototypes -- declare function types and variables passed to function.
- Variables
- Functions

We must have a ***main( )*** function.

A function has the form:

type function_name (parameters)
{
local variables

C Statements

}

If the type definition is omitted C assumes that function returns an integer type.

NOTE: This can be a source of problems in a program.

/* Sample program 1*/

```
#include<stdio.h>

void main (void)
 {
 printf( “Hello World \n” );
 }
```

NOTE:

- C requires a semicolon at the end of every statement.
- printf is a *standard* C function -- called from main.
- \n signifies newline. Formatted output -- more later.

The output of this would be:

Hello World

/* Sample program 2 */

```
#include<stdio.h>

void main (void)
 {
 printf( “\n*1\n**2\n***3\n****4”);
 }
```

The output of this would be:

***1**

****2**

*****3**

******4**

Variables

C has the following simple data types:

C type	Size (bytes)	Lower bound	Upper bound
char	1	—	—
unsigned char	1	0	255
short int	2	−32768	+32767
unsigned short int	2	0	65536
(long) int	4	-2^{31}	$+2^{31}-1$
float	4	$-3.2 \times 10^{\pm 38}$	$+3.2 \times 10^{\pm 38}$
double	8	$-1.7 \times 10^{\pm 302}$	$+1.7 \times 10^{\pm 302}$

e.g.
```
int  i,j,k;
float x,y,z;
char ch;
```

Defining Global Variables

Global variables are defined above main() in the following way:-

```
 int k,j;

main ()
{

}
```

It is also possible to pre-initialize global variables using the operator **" = "** for assignment.

For example:-

```
int sum=0;

 main()
{

 }
```

This is the same as:-

```
int sum;

 main()
{
 sum=0;
}
```

...but is more efficient.

C also allows multiple assignment statements using =, for example:

```
a=b=c=d=3;
```

...which is the same as, but more efficient than:

```
a=3;
b=3;
c=3;
d=3;
```

This kind of assignment is only possible if all the variable types in the statement are the same.

Defining Local Variables

In computer science, a **local variable** is a variable that is given local scope. Such a variable is accessible only from the function or block in which it is declared. Local variables are contrasted with global variables.

For example:-

```
int sum=0;                  //Global Variable

 main()
{
Int a;                      //Local Variable

}
```

Printing Out and Inputting Variables

C uses formatted output.

printf :

The printf function has a special formatting character (%) -- a character following this defines a certain format for a variable:

%c -- characters
%d -- integers
%f -- floats

e.g. printf("%c %d %f ",ch,i,x);

/* Sample Program 3*/

```
#include <stdio.h>

unsigned char ch='X';
int rad = 5;
float  pi = 3.14;

void  main (void)
{

printf(" %c  #  %d  # %f ",ch,rad,pi);

}
```

The output of this would be:

```
X  #  5  #  3.14
```

note: Format statement enclosed in "...", variables follow after. Make sure order of format and variable data types match up.

scanf()

scanf() is the function for inputting values to a data structure: Its format is similar to printf:

i.e. scanf(``%c %d %f'',&ch,&rad,&pi);

NOTE: & before variables. Please accept this for now and remember to include it. It is to do with pointers which we will meet later (Section 17.4.1).

Arithmetic Operations

As well as the standard arithmetic operators (+, -, * ,/) found in most languages, C provides some more operators. There are some notable differences with other languages, such as Pascal.

Assignment is = *i.e.*

i = 4;

ch = `y';

Increment ++,

Decrement – – ;

which are more efficient than their long hand equivalents,

for example:

x++ is faster than x=x+1.

The ++ and -- operators can be either in post-fixed or pre-fixed. With pre-fixed the value is computed before the expression is evaluated whereas with post-fixed the value is computed after the expression is evaluated.

/* Sample Program 4*/

```
#include<stdio.h>
int x = 0;

 main()
 {
 printf( “\n %d ”, x);      //x=0;
 x++:
 printf( “\n %d ”, x);      //x=1;
 x++:
 printf( “\n %d ”, x);      //x=2;
 x++:
 printf( “\n %d ”, x);      //x=3;
 x--:
printf( “\n %d ”, x);       //x=2;
 x--:
printf( “\n %d ”, x);       //x=1;
}
```

The output of this would be:

0

1

2

3

2

1

Division " / " is for both integer and float division. So be careful.

The answer to: x = 3 / 2 is 1 even if x is declared a float!!

RULE: If both arguments of / are integer then do integer division.

So make sure you do this. The correct (for division) answer to the above is x = 3.0 / 2 or x= 3 / 2.0 or (better) x = 3.0 / 2.0.

There is also a convenient shorthand way to express computations in C.

It is very common to have expressions like: i = i + 3 or x = x*(y + 2)

This can written in C (generally) in a *shorthand* form like this:

$$expr_1\ op\ =\ expr_2$$

which is equivalent to (but more efficient than):

$$expr_1\ =\ expr_1\ op\ expr_2$$

So we can rewrite i = i + 3 as i += 3

and x = x*(y + 2) as x *= y + 2.

NOTE: that x *= y + 2 means x = x*(y + 2) and NOT x = x*y + 2.

Comparison Operators

To test for equality is **“ == “**

A warning: Beware of using “=’” instead of “==”, such as writing accidentally

```
if ( i = j ) .....
```

This is a perfectly LEGAL C statement (syntactically speaking) which copies the value in "j" into "i", and delivers this value, which will then be interpreted as TRUE if j is non-zero. This is called assignment by value -- a key feature of C.

Not equals is: **“ != “**

Other operators

< (less than) ,

> (grater than),

<= (less than or equals),

>= (greater than or equals)

are as usual.

Logical Operators

Logical operators are usually used with conditional statements.

The two basic logical operators are:

&& for logical AND,

|| for logical OR.

Beware & and | have a different meaning for bitwise AND and OR

Order of Precedence

It is necessary to be careful of the meaning of such expressions as

a + b * c

We may want the effect as either

(a + b) * c

or

a + (b * c)

All operators have a priority, and high priority operators are evaluated before lower priority ones. Operators of the same priority are evaluated from left to right, so that

a - b - c

is evaluated as

(a - b) - c

as you would expect.

From high priority to low priority the order for all C operators (we have not met all of them yet) are:

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

Thus

a < 10 && 2 * b < c

is interpreted as

(a < 10) && ((2 * b) < c)

Conditional Statement

The While Loop

The C programming language has several structures for looping and conditional branching. We will cover them all in this chapter and we will begin with the while loop.

The while loop continues to loop while some condition is true. When the condition becomes false, the looping is discontinued. It therefore does just what it says it does, the name of the loop being very descriptive.

/* Sample Program 5 */

```
#include<stdio.h>

int i=0;

void main(void)

{

printf("\t START");

while(i<5)

        {

        printf("\t %d",i);

        i++;

        }

printf("\t END");

}
```

The output of this would be:

START 0 1 2 3 4 END

Whenever result of compare or logical operation is true in while statement, the code will be executed in loop. The true is considered as non zero element. That means the result which is not false (0).So if we want to execute any statement infinitely we can write statement as

```
while (1)

{

}
```

That is condition is always true

/* Sample Program 6 */

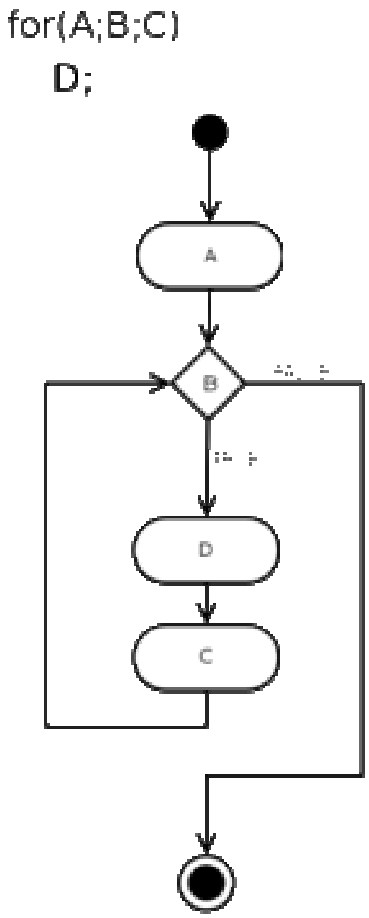
```
#include<stdio.h>

void main(void)

{

printf("\t START");

while(1)

        {

        printf("+");

        }

printf("\t END");

}
```

The output of this would be:

START++ and so on

I think pages of this manual are also not enough to complete the output of this code. The string END will be never printed in this code.

The for Loop



For statement, is characterized by a three-parameter loop control expression; consisting of an initialize, a loop-test, and a counting expression.

The three control expressions, separated by semicolons here, are from left to right the initialize expression, the loop test expression, and the counting expression. The initialize is evaluated exactly once right at the beginning. The loop test expression is evaluated at the beginning of each iteration through the loop, and determines when the loop should exit. Finally, the counting expression is evaluated at the end of each loop iteration, and is usually responsible for altering the loop variable.

/* Sample Program 7 */

```
#include<stdio.h>

int  i;

void main(void)

{

printf("\t START");

for(i=0;i<10;i++)

        {

        printf("\t %d",i);

        }

printf("\t END");

}
```

The output of this would be:

START 0 1 2 3 4 5 END

We may use for loop for generating software delay.

Writing code something like

Statment1

for(i=0;i<10000;i++);

Stement2

will perform nothing for 10000 times so generating delay between current and next statement /instruction.

The if and else statement

The if-else statements enable your program to selectively execute other statements, based on some criteria. The simplest version, the **if** statement, is shown below. The block governed by if (delimited with '{' and '}') is executed if the expression is true, otherwise the execution continues after the last '}'.

```
if (expressions)
{
  statement(s)
}
```

If you want to execute other statements when the expression is false, you use the else statement.

```
if (expression)
{
  statement(s) executed if expression is true
}
else
{
  statement(s) executed if expression is false
}
```

Lets have an example..

```
int i=7;
if(I >5)

{
printf("\n i is greater than 5");
}
else
{
printf(""\n i is smaller than 5");
}
```

Here output will be **i is greater than 5**

Another statement, **elseif**, executes statements if an earlier if expression was false and its own expression is true; elseif is used to form chains of conditions. If expression 1 is true, the if block executes and then the program jumps to the last '}' of the else block. Expression 2 will never be evaluated. If, however, expression 1 is false, expression 2 will be evaluated and the **elseif**-**else** will work as a regular if-else as shown above.

```
if (expr 1)
{
  statement(s) executed if expr 1 is true
}
elseif (expr 2)
{
  statement(s) executed if expr 1 is false and expr 2 is true
}
else
{
  statement(s) executed if expr 1 is false and expr 2 is false
}
```

The Break Statement

The break statements enable your program to exit from current loop. Break statement may called conditionally or unconditionally. That is if you have written any for loop and you want to break from the loop on certain condition then you can write break statement to exit form current loop.

Let's have an example

```
int i=0;

for(i=0;i<10;i++)
{
        printf("\n%d",i);
        if(i==5)
        {
        Printf("\n Loop breaks");
        }
}
```

Output will look like this

```
1
2
3
4
5
Loop breaks
```

The Switch Statement

The switch statement is used instead of using multiple if else statement, where value of 1 variable is compared with different conditional variable and whichever conditional variable is equal to actual variable the routine or program for that variable is executed. Each conditional variable is written next to case.

The general form of a switch statement is:

```
switch (variable)
 {
case expression1:
                    do something 1;
                    break;

case expression2:
                    do something 2;
                    break;

....


default:
                    do default processing;
                    break;
}
```

After completion of every **case** there should be **break** statement written so that it will exit from loop. If you are not writing the break statement after completion of case then next case statement will be also executed.

If not a single case is satisfied then **default** statement will be executed.

Let's take an example for switch case

```
int i=0;

for (i=0;i<6;i++)
{
   switch (i)
   {
         case 0 :    printf("\n ZERO");
                     break;

         case 2 :    printf("\n EVEN2");
                     break;

         case 4 :    printf("\n EVEN4");
                     break;

         default :   printf("\n ODD");
                     break;

   }
}
```

Output for this code will be

```
ZERO
ODD
EVEN2
ODD
EVEN4
ODD
```

The goto Statement

The **goto** statement is used to jump from 1 line of code to any other line from same code file. It is obvious that you have to defile address first in your file where you have to jump. That address is known as “label”. Now you can jump to that particular label from any where in code file.

To define label you have to type label name and then put colon “**:**” after that. In following example START is label defined

NOTE: you cannot jump from one function to other function

Let’s take an example for goto statement

```
printf(“code starts here”)
START:
printf(“\n Sunday”);
printf(“\n Monday”);
printf(“\n Tuesday”);
goto START;
printf(“\n Wednesday”);
printf(“\n Thursday”);
```

Output for this code will be

```
Code starts here
Sunday
Monday
Tuesday
Sunday
Monday
Tuesday
```

And so on…….

The line Wednesday and Thursday will be never print since it is never reached in code execution.

Note: This Page intentionally kept blank

Flow chart

A **flowchart** is common type of chart that represents an algorithm or process showing the steps as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Flow chart building blocks

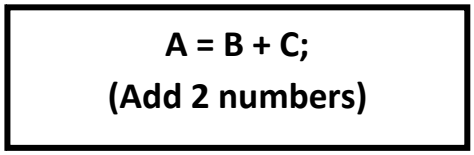
A typical flowchart from older Computer Science textbooks may have the following kinds of symbols:

Start and end symbols

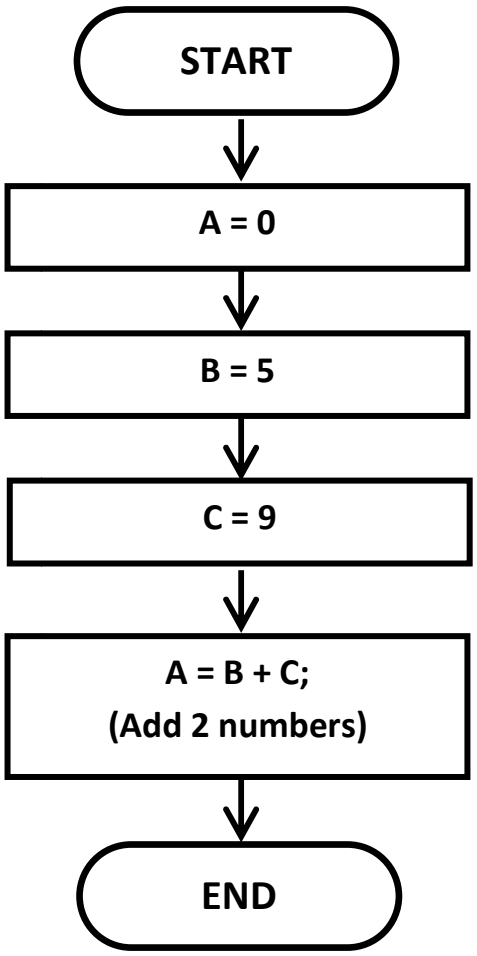
Start and End represented as lozenges, ovals or rounded rectangles, usually containing the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit enquiry" or "receive product".

Process

Process represented as rectangles. Examples: "Add 1 to X"; "replace identified part"; "save changes" or similar.

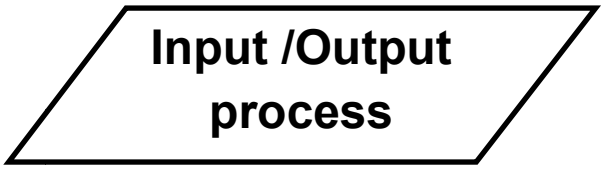


Example



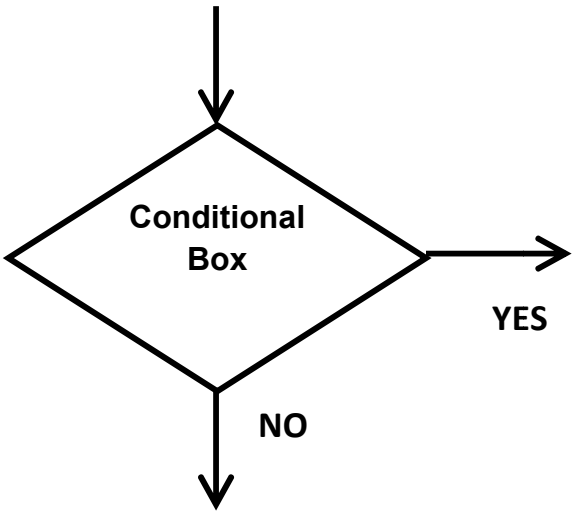
Input/ Output

Input Output process Represented as a parallelogram. Examples: Get X from the user; display X.

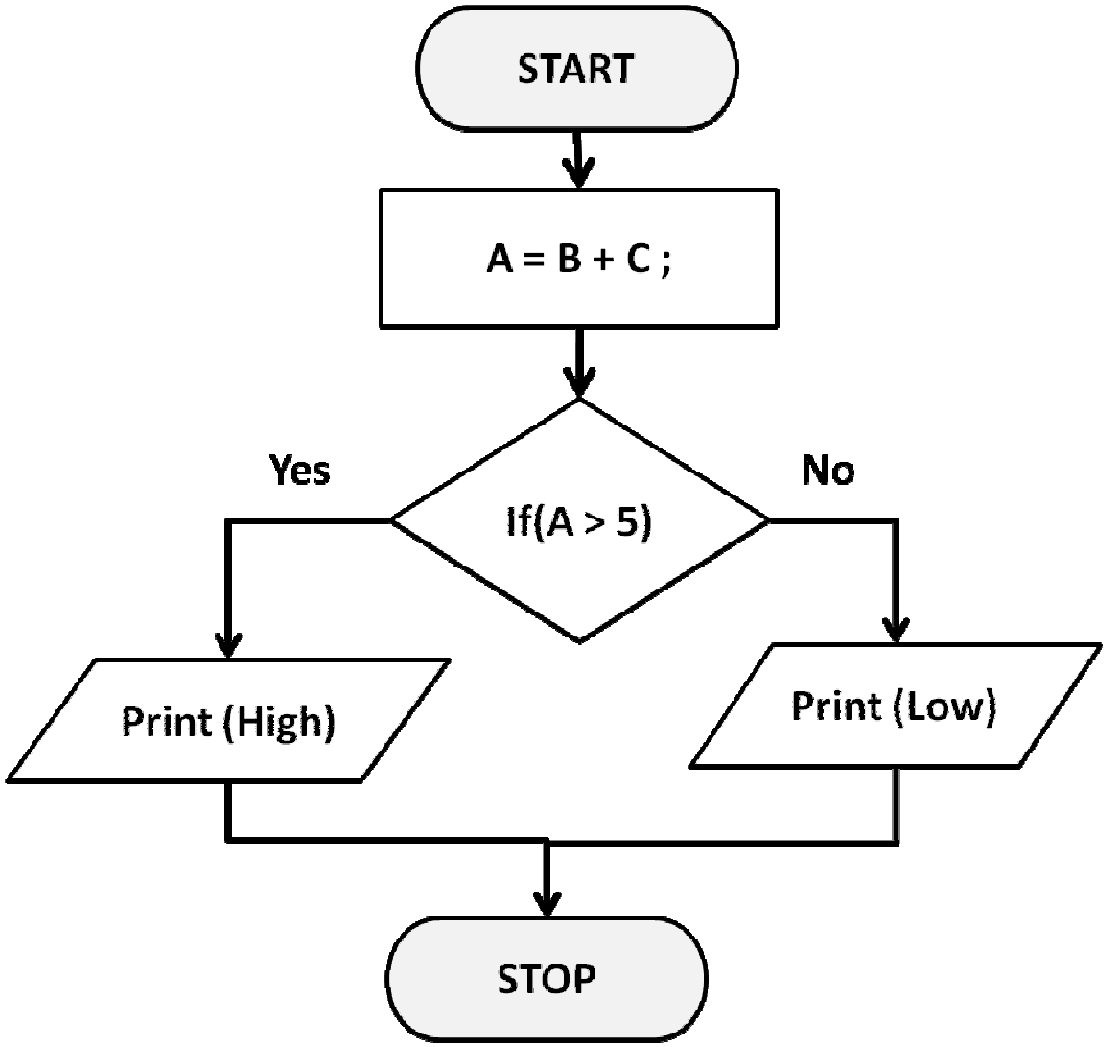


Conditional or decision

Conditional or decision Represented as a diamond (rhombus). These typically contain a Yes/No question or True/False test. This symbol is unique in that it has two arrows coming out of it, usually from the bottom point and right point, one corresponding to Yes or True, and one corresponding to No or False. The arrows should always be labeled. More than two arrows can be used, but this is normally a clear indicator that a complex decision is being taken, in which case it may need to be broken-down further, or replaced with the "pre-defined process" symbol.



Example



Note: This Page intentionally kept blank

How to Use Robosoft Library

#include "robosoft.h"

This is the library file which we will be using for our Advance Legged Robotic Workshop. This file includes all the basic predefined function for our legged robot.

robosoft( );

This is the function you will be calling in in your main( ) function initially to initialize your controller board.

```
void main(void)
{
robosoft();
      while(1)
      {
      //body of program
      }
}
```

Servo Control Variables

R_M : is servo Variable for Right Servo

L_M : is servo Variable for Left Servo

C_M : is servo Variable for Center Servo

E_M : is servo Variable for Extra Servo

Wireless Remote Variables

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

NUM_0 : is set when Num '0' key is pressed

NUM_1 : is set when Num '1' key is pressed

NUM_2 : is set when Num '2' key is pressed

NUM_3 : is set when Num '3' key is pressed

NUM_4 : is set when Num '4' key is pressed

NUM_5 : is set when Num '5' key is pressed

NUM_6 : is set when Num '6' key is pressed

NUM_7 : is set when Num '7' key is pressed

NUM_8 : is set when Num '8' key is pressed

NUM_9 : is set when Num '9' key is pressed

MD : is set when MODE key is pressed

PW : is set when POWER key is pressed

How to use keys on board

Keys are connected on 3 port pins, those pins are
Port 3.2, Port 3.3 and Port 3.4,
These pins are defined in program as

```
sbit  KEY1 = P3^2;
sbit  KEY2 = P3^3;
sbit  KEY3 = P3^4;
```

These are active low input port pins, so when you press ant key respective port pin will go LOW. i.e. 0V

How to use Led on board

LED s are connected on 3 port pins, those pins are
Port 3.5, Port 3.6 and Port 3.7,
These pins are defined in program as

```
sbit  LED1 = P3^5;
sbit  LED2 = P3^6;
sbit  LED3 = P3^7;
```

These are active low output port pins, so when you write 1 to pin LED if off and it is on for LOW value. i.e. 0v

How to use Obstacle avoiding Sensors

Sensors are connected on 2 port pins, those pins are
Port 2.0, Port 2.1,
These pins are defined in program as

```
sbit  OBS_RIGHT = P2^0;
sbit  OBS_LEFT = P2^1;
```

These are active low input port pins, so when pin value is LOW that means obstacle is detected.

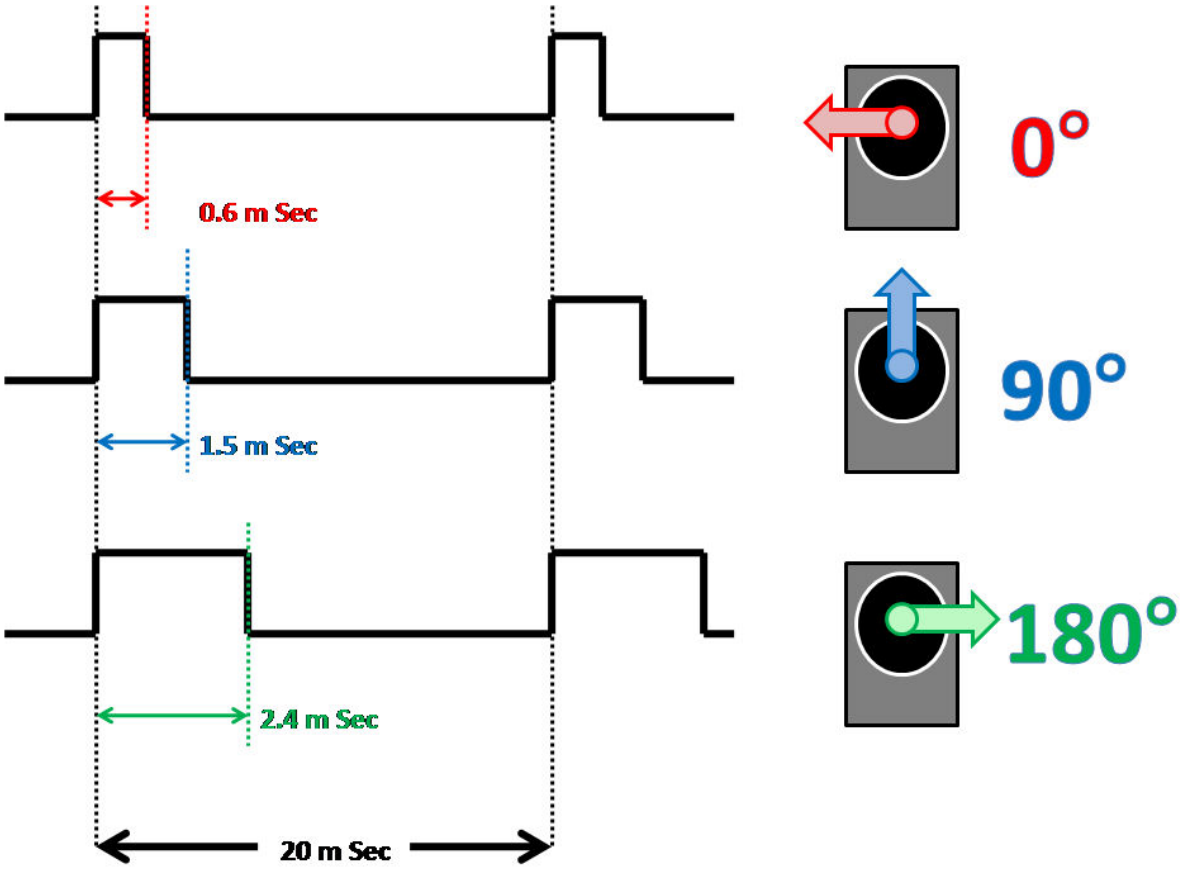
Note: This Page intentionally kept blank

How Servo Motor is controlled?

So, how does a servo work? The servo motor has some control circuits and a potentiometer (a variable resistor, aka pot) that is connected to the output shaft. In the picture above, the pot can be seen on the right side of the circuit board. This pot allows the control circuitry to monitor the current angle of the servo motor. If the shaft is at the correct angle, then the motor shuts off. If the circuit finds that the angle is not correct, it will turn the motor the correct direction until the angle is correct. The output shaft of the servo is capable of travelling somewhere around 180 degrees. Usually, its somewhere in the 210 degree range, but it varies by manufacturer. A normal servo is used to control an angular motion of between 0 and 180 degrees. A normal servo is mechanically not capable of turning any farther due to a mechanical stop built on to the main output gear.

The amount of power applied to the motor is proportional to the distance it needs to travel. So, if the shaft needs to turn a large distance, the motor will run at full speed. If it needs to turn only a small amount, the motor will run at a slower speed. This is called proportional control.

How do you communicate the angle at which the servo should turn? The control wire is used to communicate the angle. The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Coded Modulation. The servo expects to see a pulse every 20 milliseconds (.02 seconds). The length of the pulse will determine how far the motor turns. A 1.5 millisecond pulse, for example, will make the motor turn to the 90 degree position (often called the neutral position). If the pulse is shorter than 1.5 ms, then the motor will turn the shaft to closer to 0 degress. If the pulse is longer than 1.5ms, the shaft turns closer to 180 degress.



As you can see in the picture, the duration of the pulse dictates the angle of the output shaft (shown as the green circle with the arrow). Note that the times here are illustrative and the actual timings depend on the motor manufacturer. The principle, however, is the same.

How to control Servo in Robosoft Library

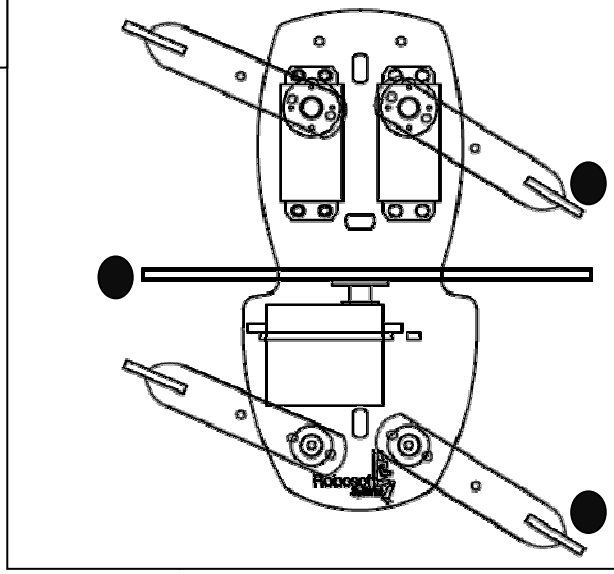
R_M=238;

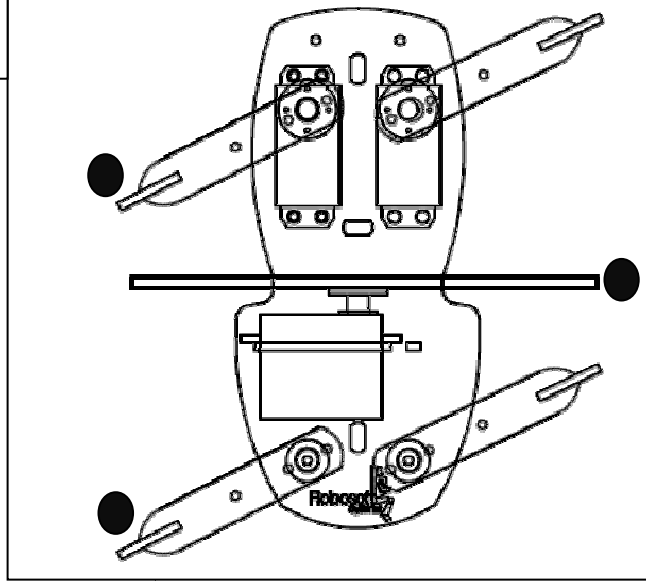
// assign value to servo so it will go to respective position

//for library 238 considered as 90 degree

How to control Legged Machine Motion?

Moving Machine Forward

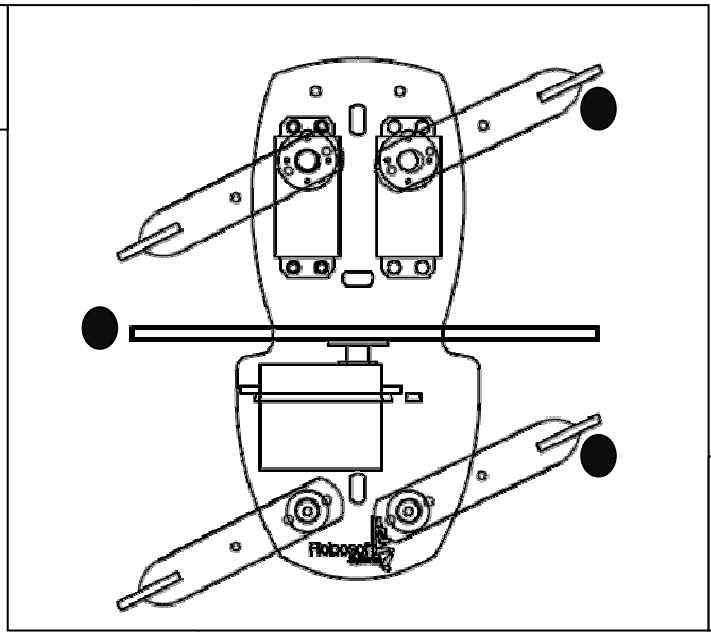
STEP No.		
1		Middle Leg RIGHT UP

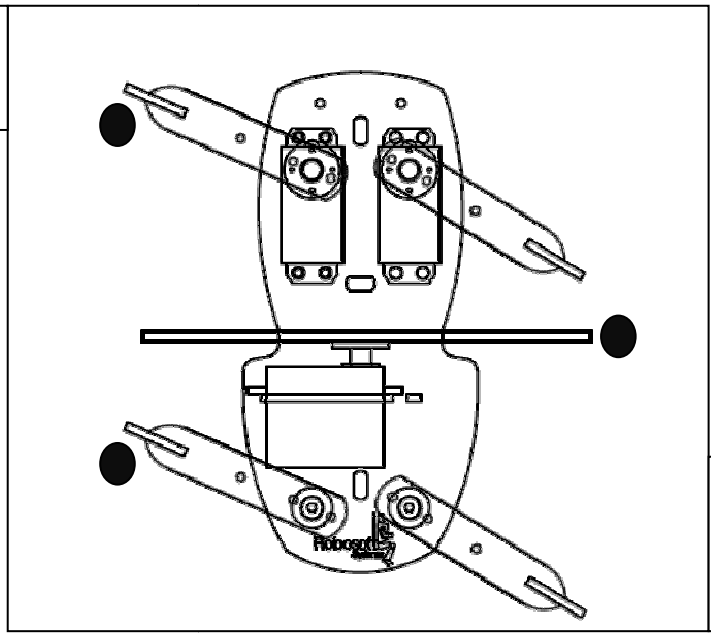
STEP No.		
2		Middle Leg LEFT UP

Note:

If you want to continue with the same motion then keep on repeating step 1 to 2, Black Dot is Ground Point Contact.

Moving Machine Backward

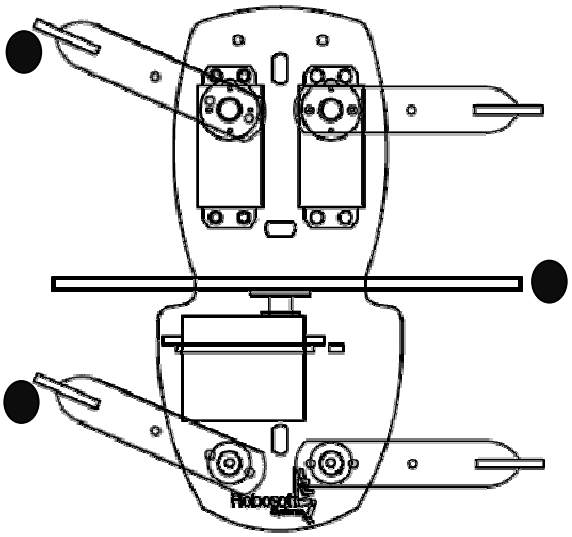
STEP No.		
1		Middle Leg RIGHT UP

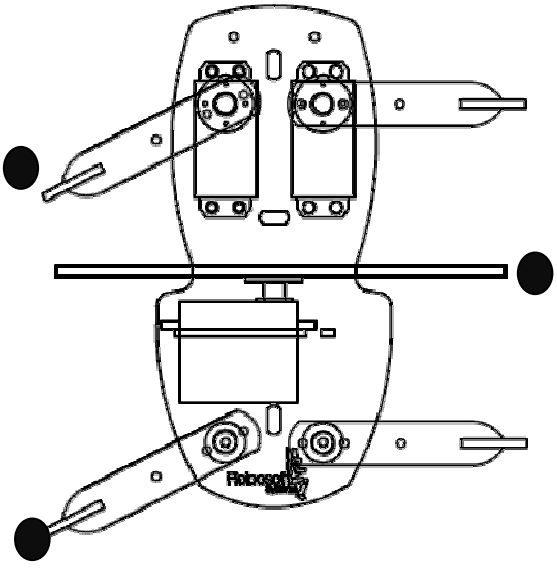
STEP No.		
2		Middle Leg LEFT UP

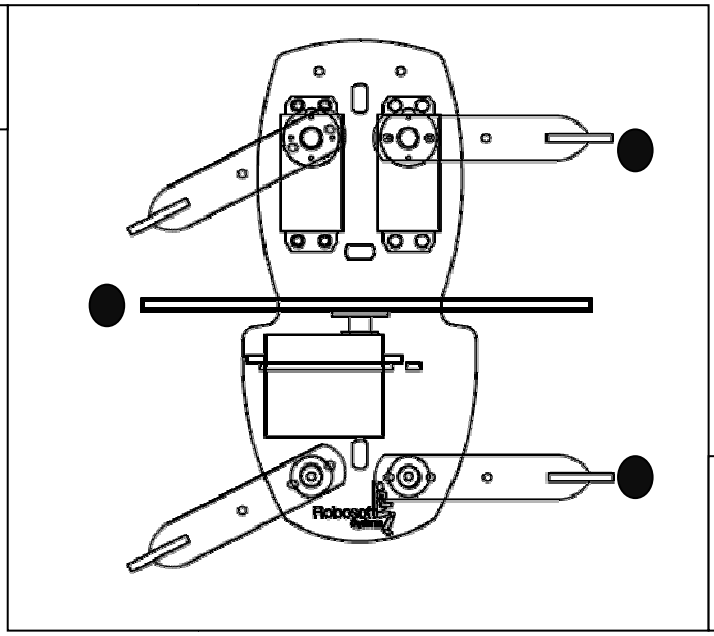
Note:

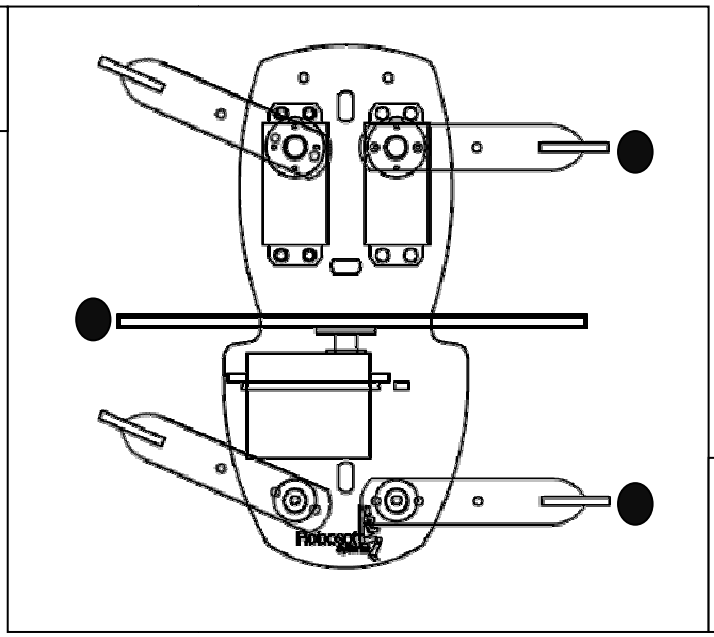
if you want to continue with the same motion then keep on repeating step 1 to 2, ,Black Dot is Ground Point Contact.

Turning Machine Right

STEP No.		
1		Middle Leg LEFT UP

STEP No.		
2		Middle Leg LEFT UP

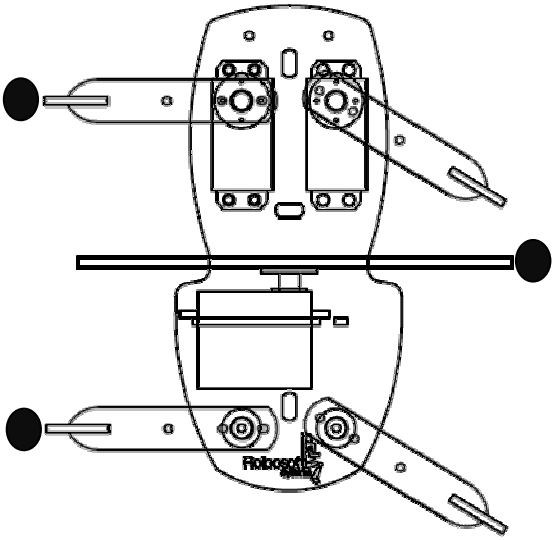
STEP No. 3		Middle Leg RIGHT UP

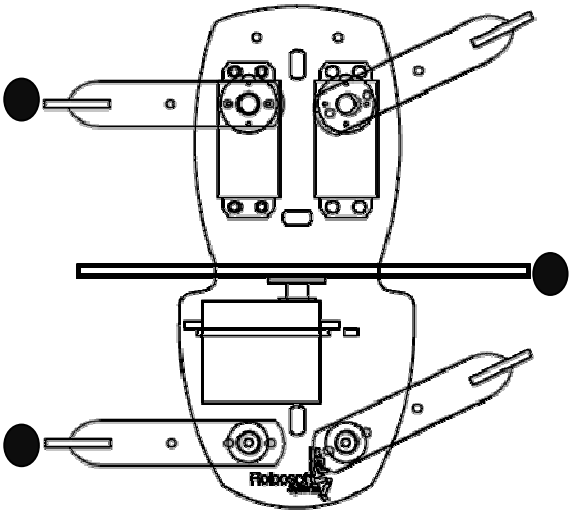
STEP No. 4		Middle Leg RIGHT UP

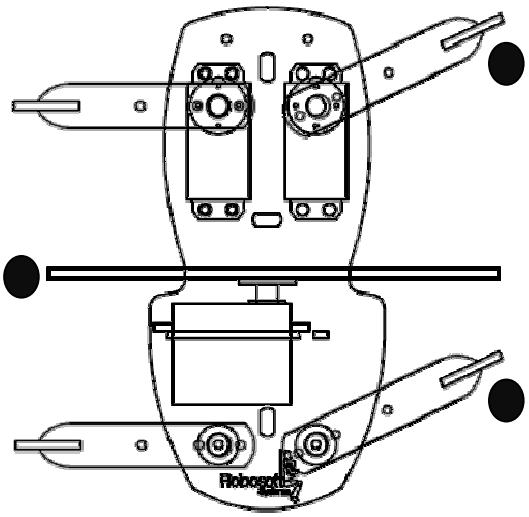
Note:

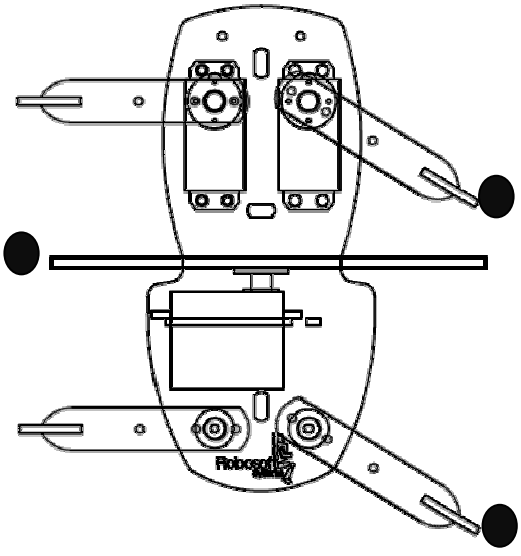
If you want to continue with the same motion then keep on repeating step 1 to 4, Black Dot is Ground Point Contact.

Turning Machine Left

STEP No.		
1		Middle Leg LEFT UP

STEP No.		
2		Middle Leg LEFT UP

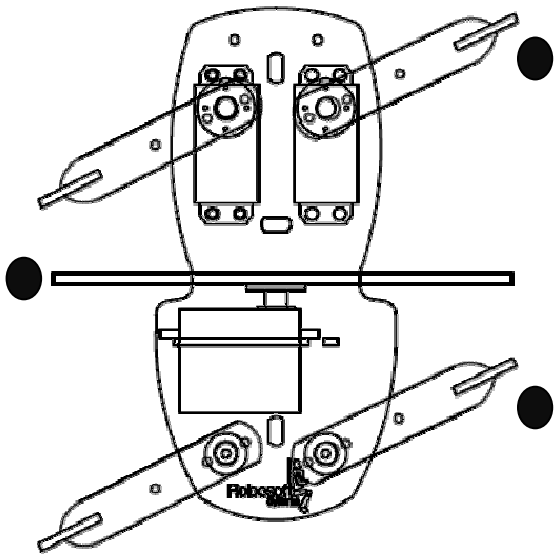
STEP No.		
3		Middle Leg RIGHT UP

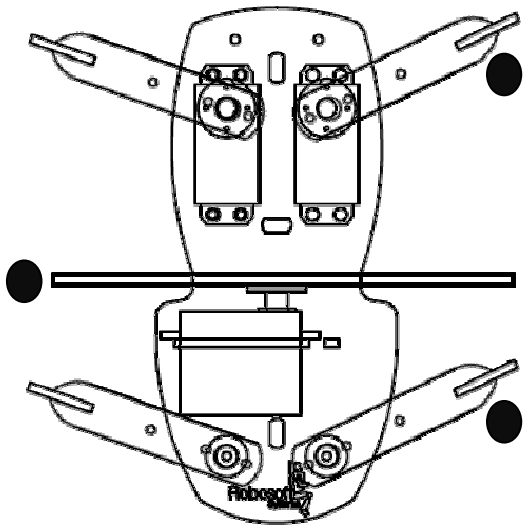
STEP No.		
4		Middle Leg RIGHT UP

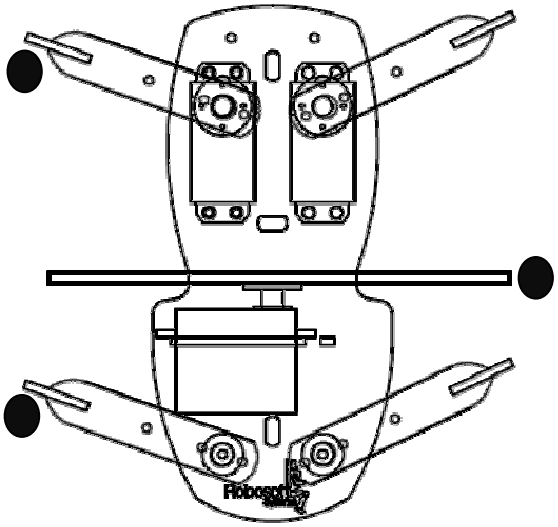
Note:

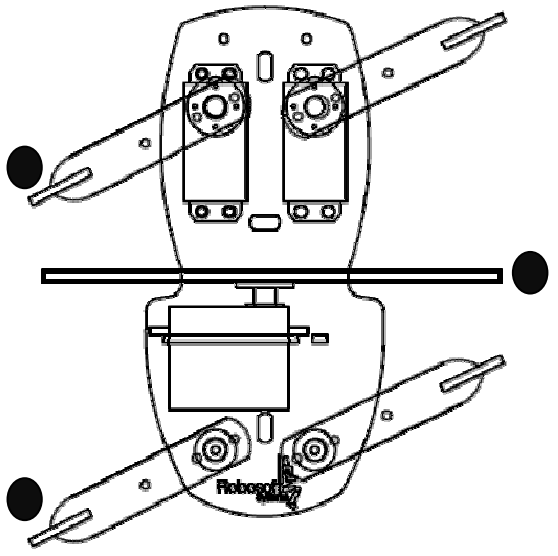
If you want to continue with the same motion then keep on repeating step 1 to 4, Black Dot is Ground Point Contact.

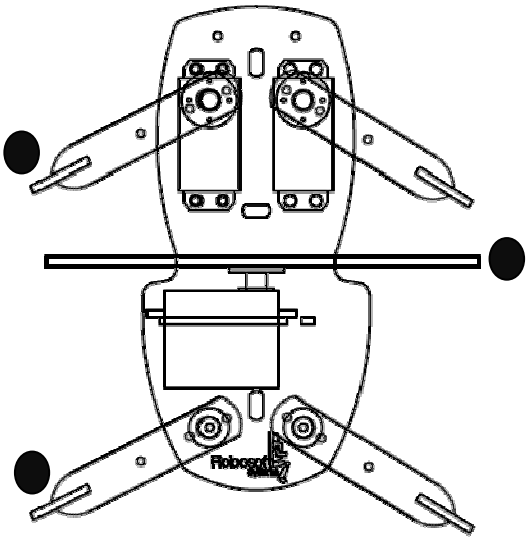
Turning Machine Right off center

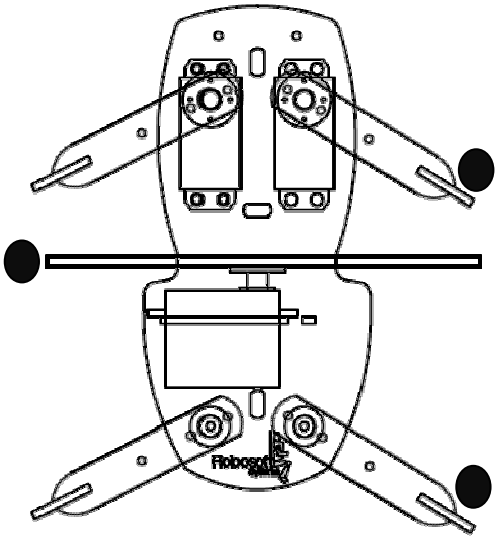
STEP No.		
1		Middle Leg RIGHT UP

STEP No.		
2		Middle Leg RIGHT UP

STEP No.		
3		Middle Leg LEFT UP

STEP No.		
4		Middle Leg LEFT UP

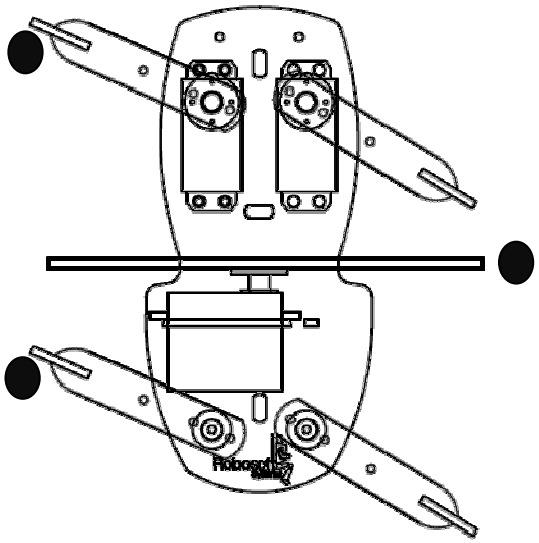
STEP No.		
5		Middle Leg LEFT UP

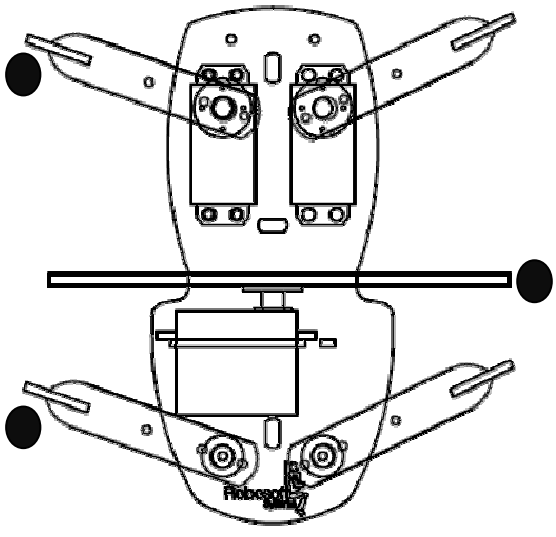
STEP No.		
6		Middle Leg RIGHT UP

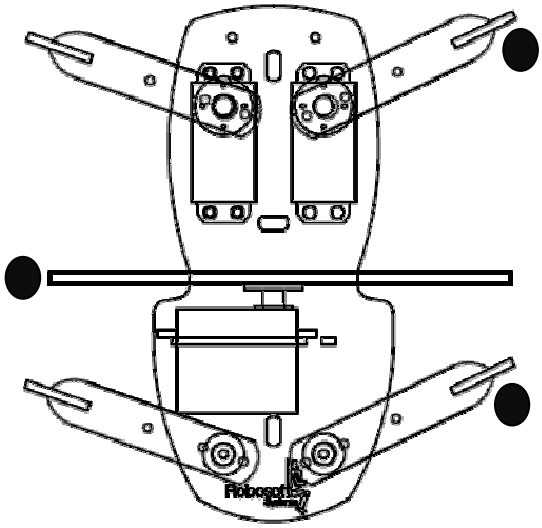
Note:

If you want to continue with the same motion then keep on repeating step 1 to 6 Black Dot is Ground Point Contact.

Turning Machine Left off center

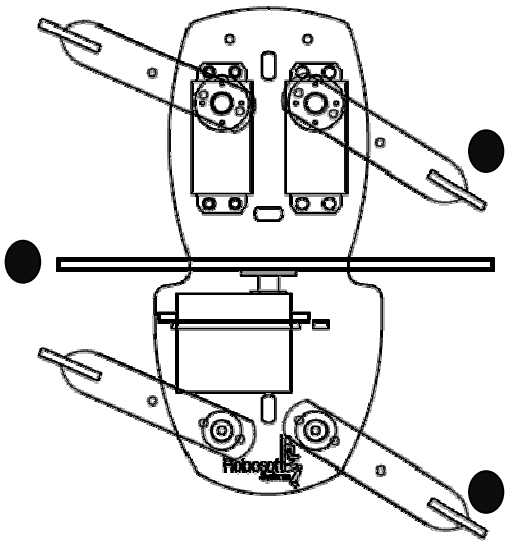
STEP No.		
1		Middle Leg LEFT UP

STEP No.		
2		Middle Leg LEFT UP

STEP No.	
3	

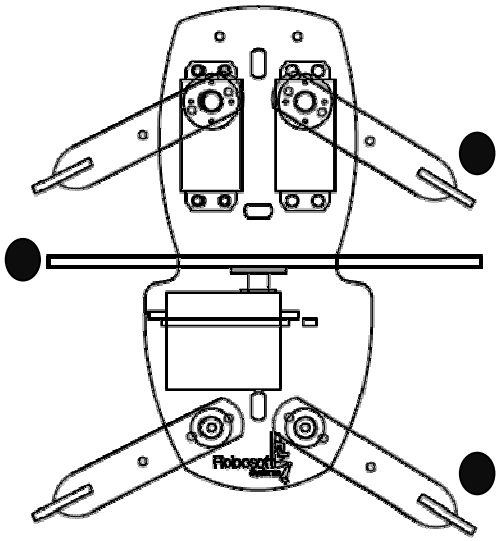
Middle Leg

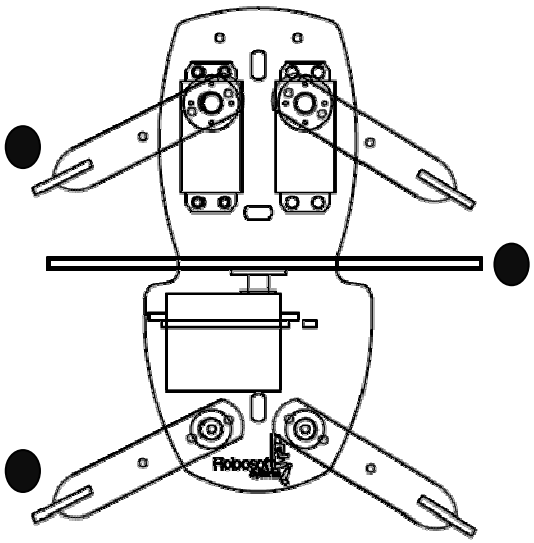
RIGHT UP

STEP No.	
4	

Middle Leg

RIGHT UP

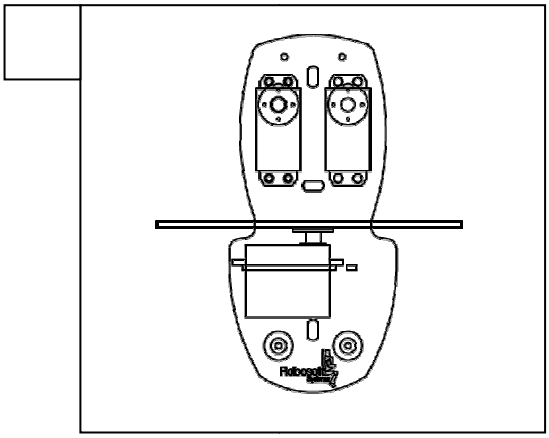
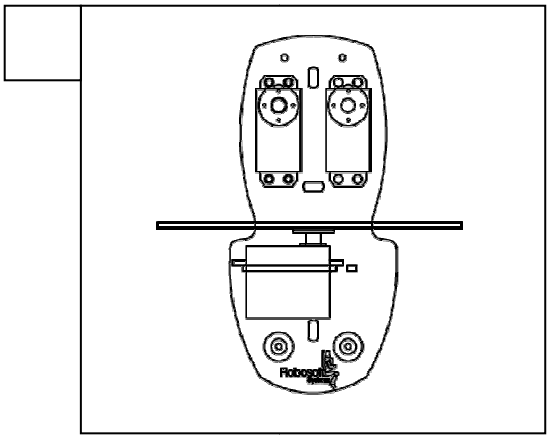
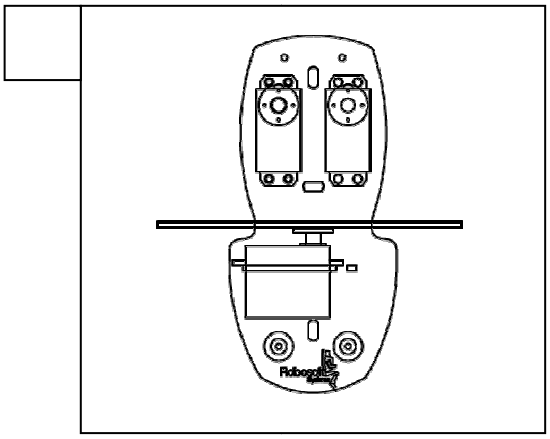
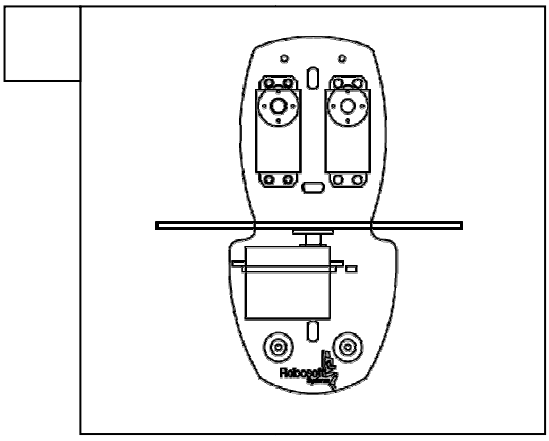
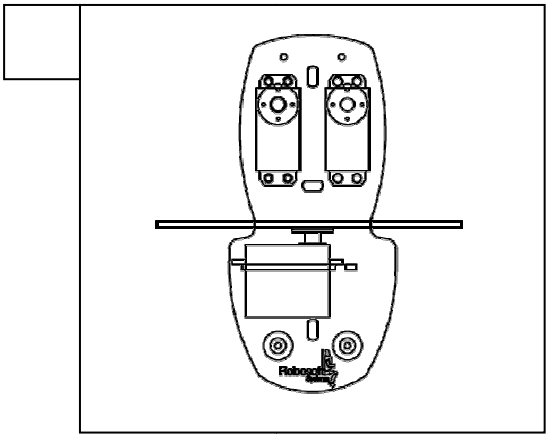
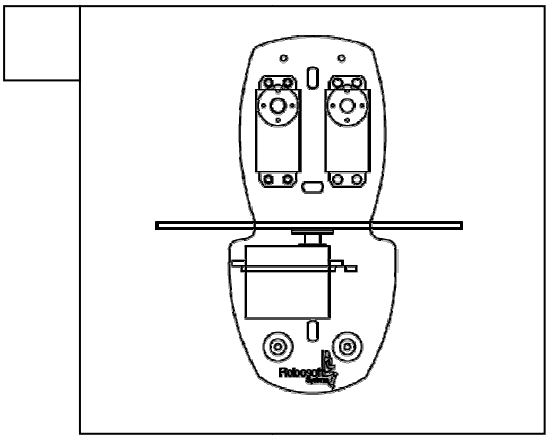
STEP No.		
5		Middle Leg RIGHT UP

STEP No.		
6		Middle Leg LEFT UP

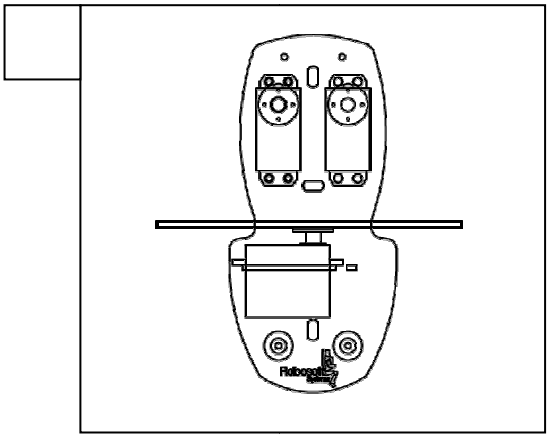
Note:

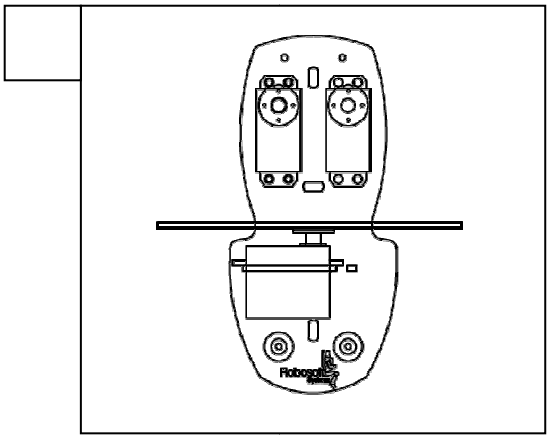
If you want to continue with the same motion then keep on repeating step 1 to 6, Black Dot is Ground Point Contact.

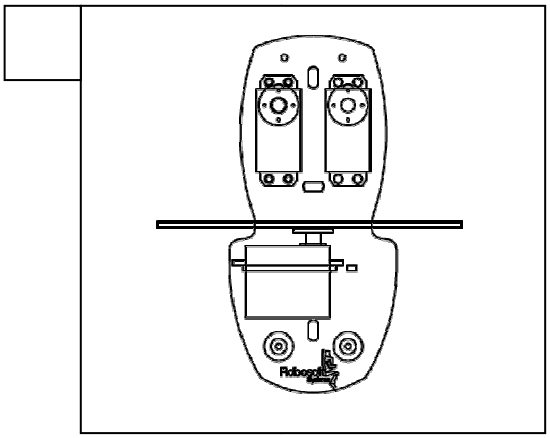
You can design your own manoeuver (Page 1)

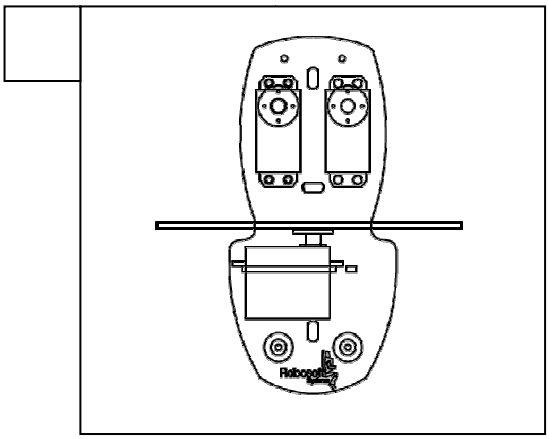







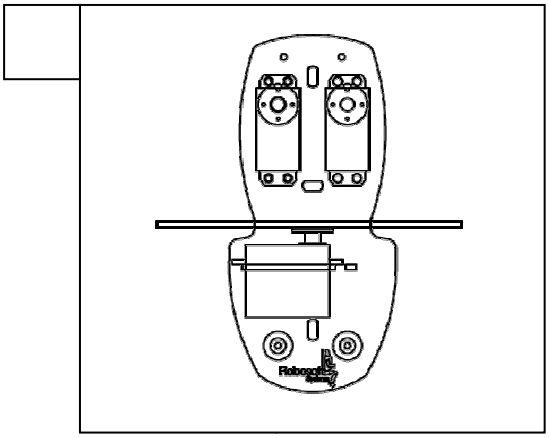
You can design your own manoeuver (Page 2)

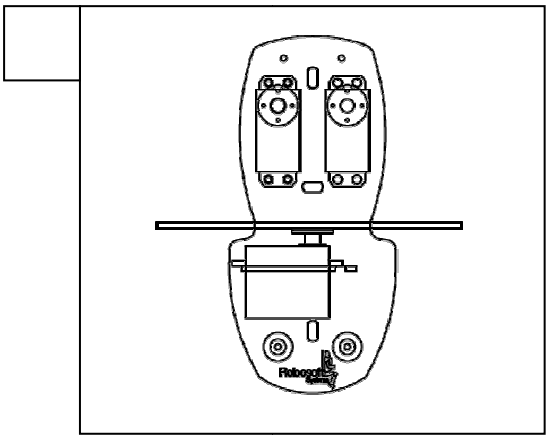




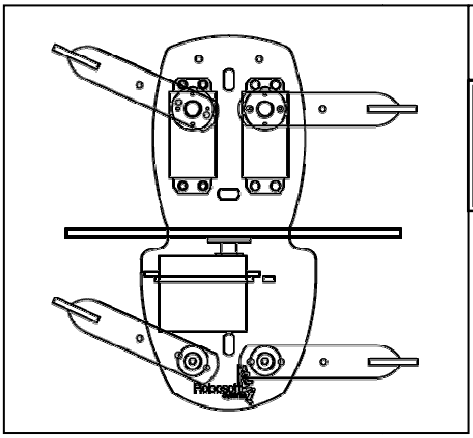




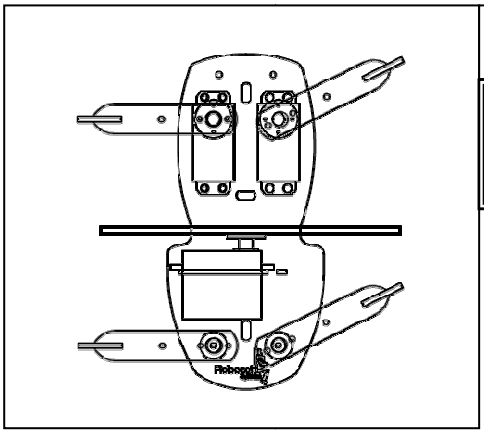




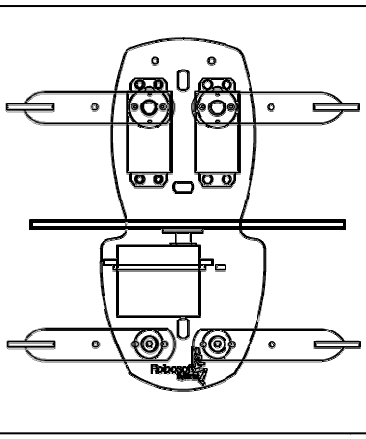
Before we start Lets find out all respective values



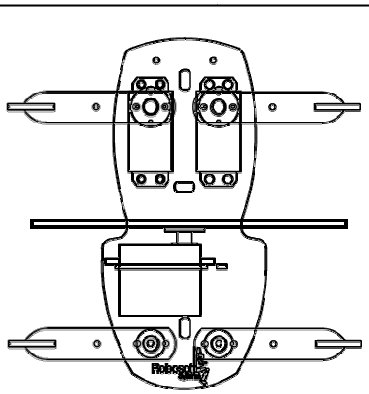
Value



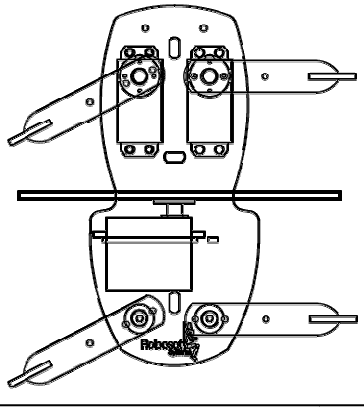
Value



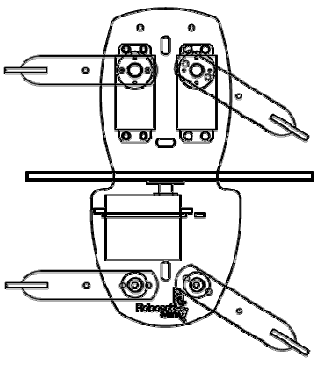
Value



Value



Value



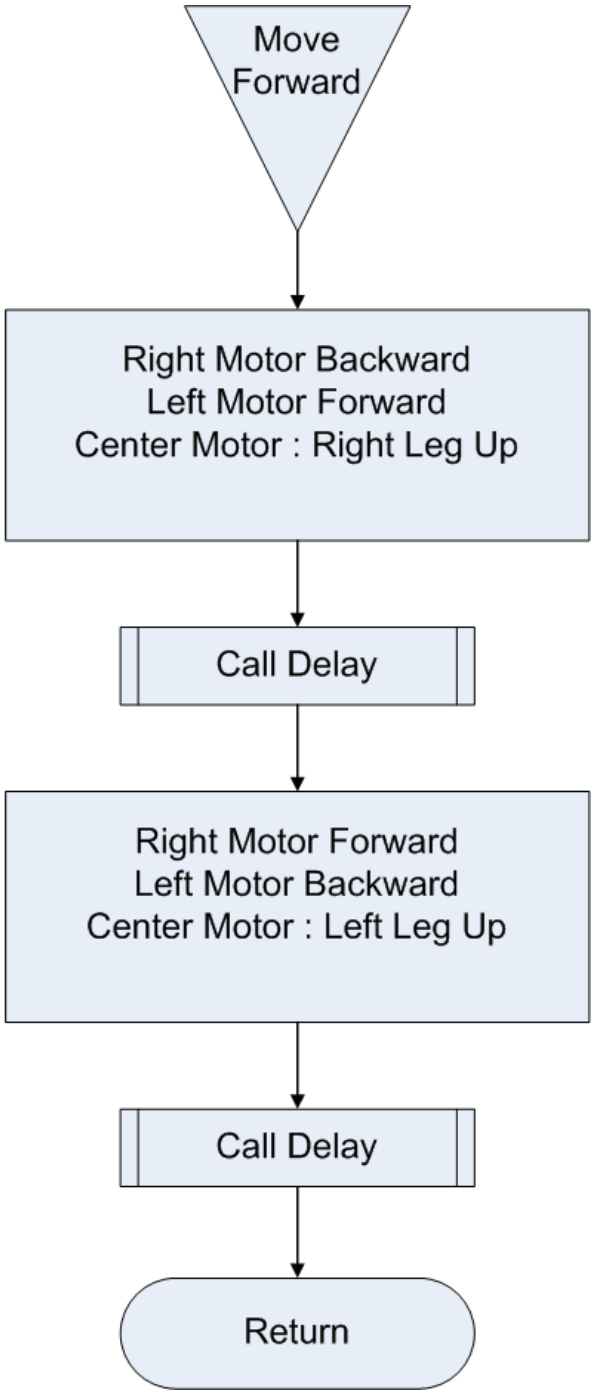
Value

Middle Leg LEFT UP		Middle Leg REST		Middle Leg RIGHT UP	

Note: This Page intentionally kept blank

Flow Chart

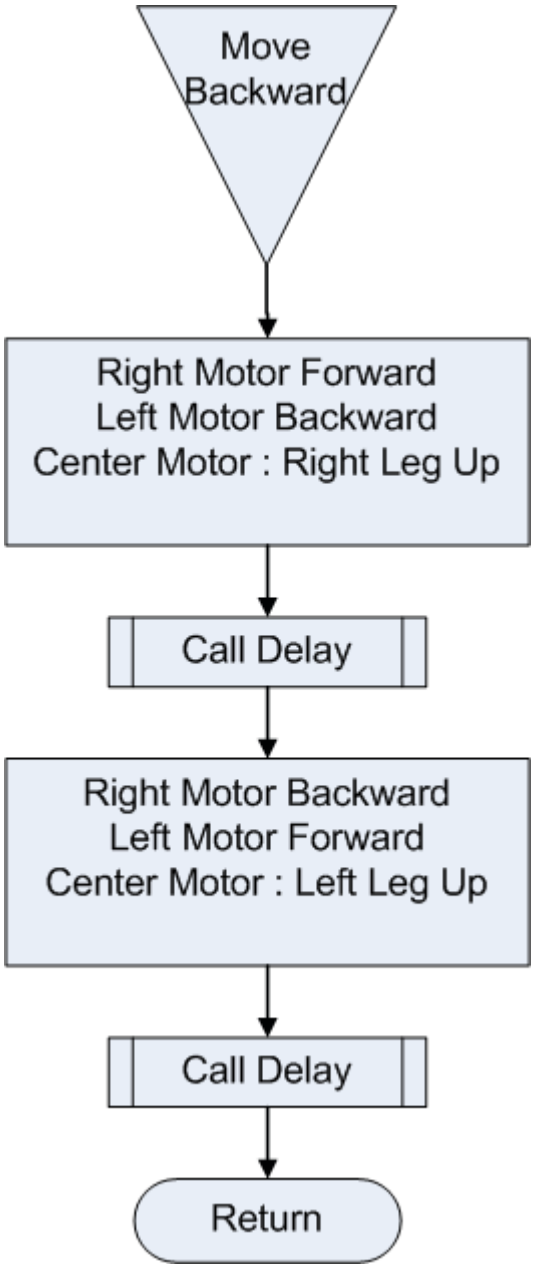
1. Flowchart for go forward



```
#include"robosoft.h"

#define RFWD 234     //Count For Right Motor Forward
#define RNEU 238      //Count For Right Motor Neutral
#define RBAK 242     //Count For Right Motor Backward

#define LFWD 242     //Count For Left Motor Forward
#define LNEU 238      //Count For Left Motor Neutral
#define LBAK 234     //Count For Left Motor Backward

#define CLFT 236     //Count for Center Motor Left Up
#define CNEU 238      //Count for Center Motor Neutral
#define CRIT 242     //Count for Center Motor Right Up

void goforward(void)
{
R_M=RBAK;
L_M=LFWD;
C_M=CRIT;
wait1sec();

R_M=RFWD;
L_M=LBAK;
C_M=CLFT;
wait1sec();
}
```

Flow Chart

2. Flowchart for go backward



```
void gobackward(void)
{
R_M=RFWD;
L_M=LBAK;
C_M=CRIT;
wait1sec();

R_M=RBAK;
L_M=LFWD;
C_M=CLFT;
wait1sec();
}


void rest (void)
{
R_M=RNEU;
L_M=LNEU;
C_M=CNEU;
wait1sec();
}
```

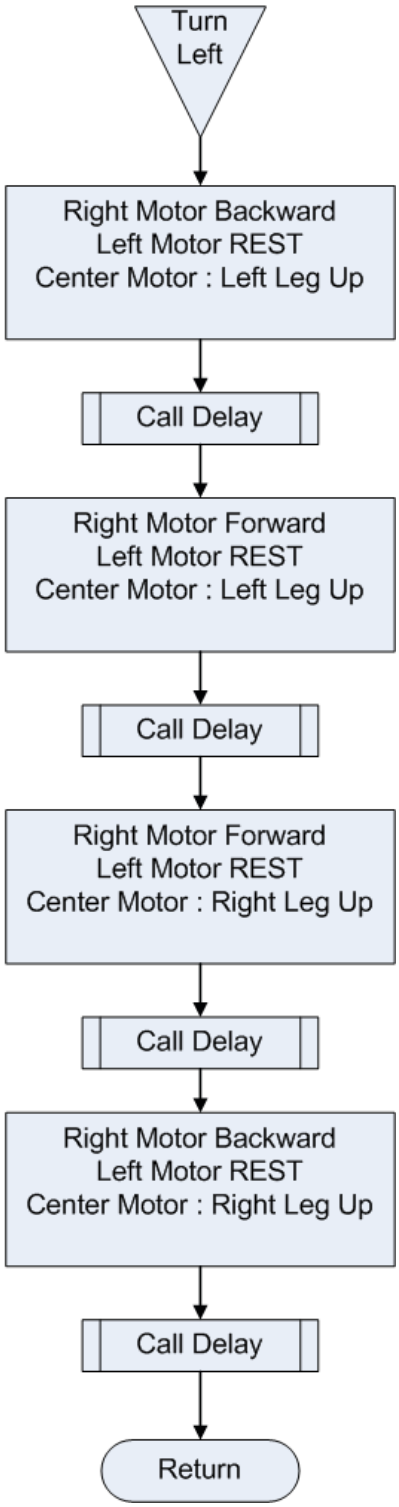
Flow Chart

3. Flowchart for turn Right

```
void turnright(void)
{
R_M=RNEU;
L_M=LFWD;
C_M=CLFT;
wait1sec();

R_M=RNEU;
L_M=LBAK;
C_M=CLFT;
wait1sec();

R_M=RNEU;
L_M=LBAK;
C_M=CRIT;
wait1sec();

R_M=RNEU;
L_M=LFWD;
C_M=CRIT;
wait1sec();
}
```

Flow Chart

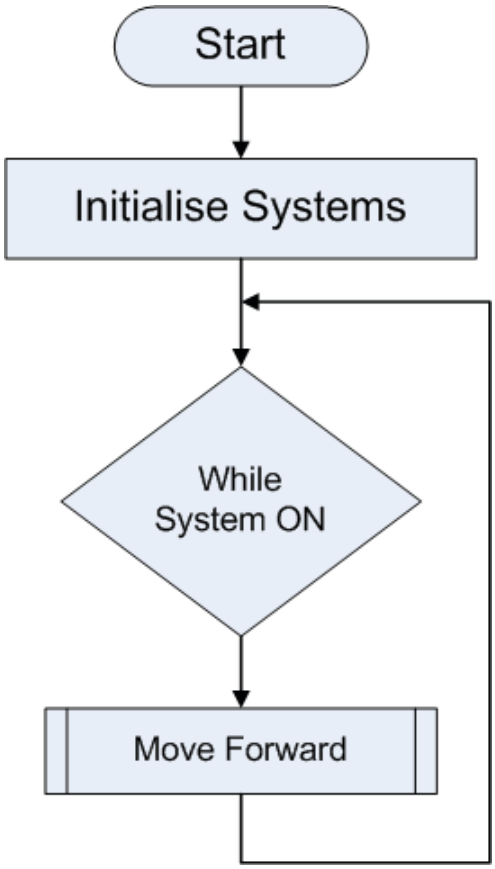
4. Flowchart for turn Left



```
void turnleft(void)
{
R_M=RBAK;
L_M=LNEU;
C_M=CLFT;
wait1sec();

R_M=RFWD;
L_M=LNEU;
C_M=CLFT;
wait1sec();

R_M=RFWD;
L_M=LNEU;
C_M=CRIT;
wait1sec();

R_M=RBAK;
L_M=LNEU;
C_M=CRIT;
wait1sec();
}
```

Flow Chart

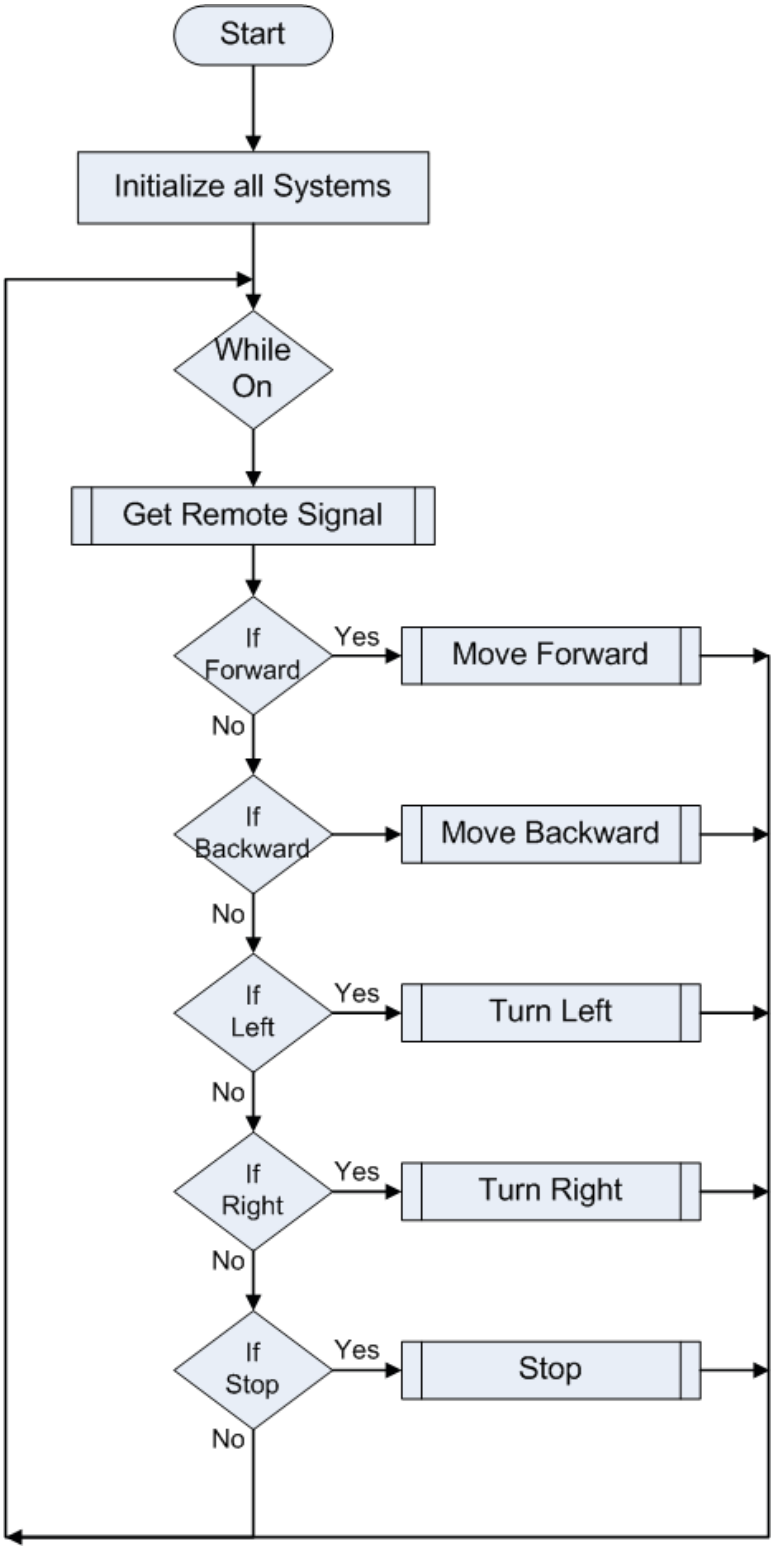
5. Flowchart for Just Run Forward




```
void main(void)
{
robosoft();
rest();
while(1)
   {
    goforward();
    }
}
```

Flow Chart

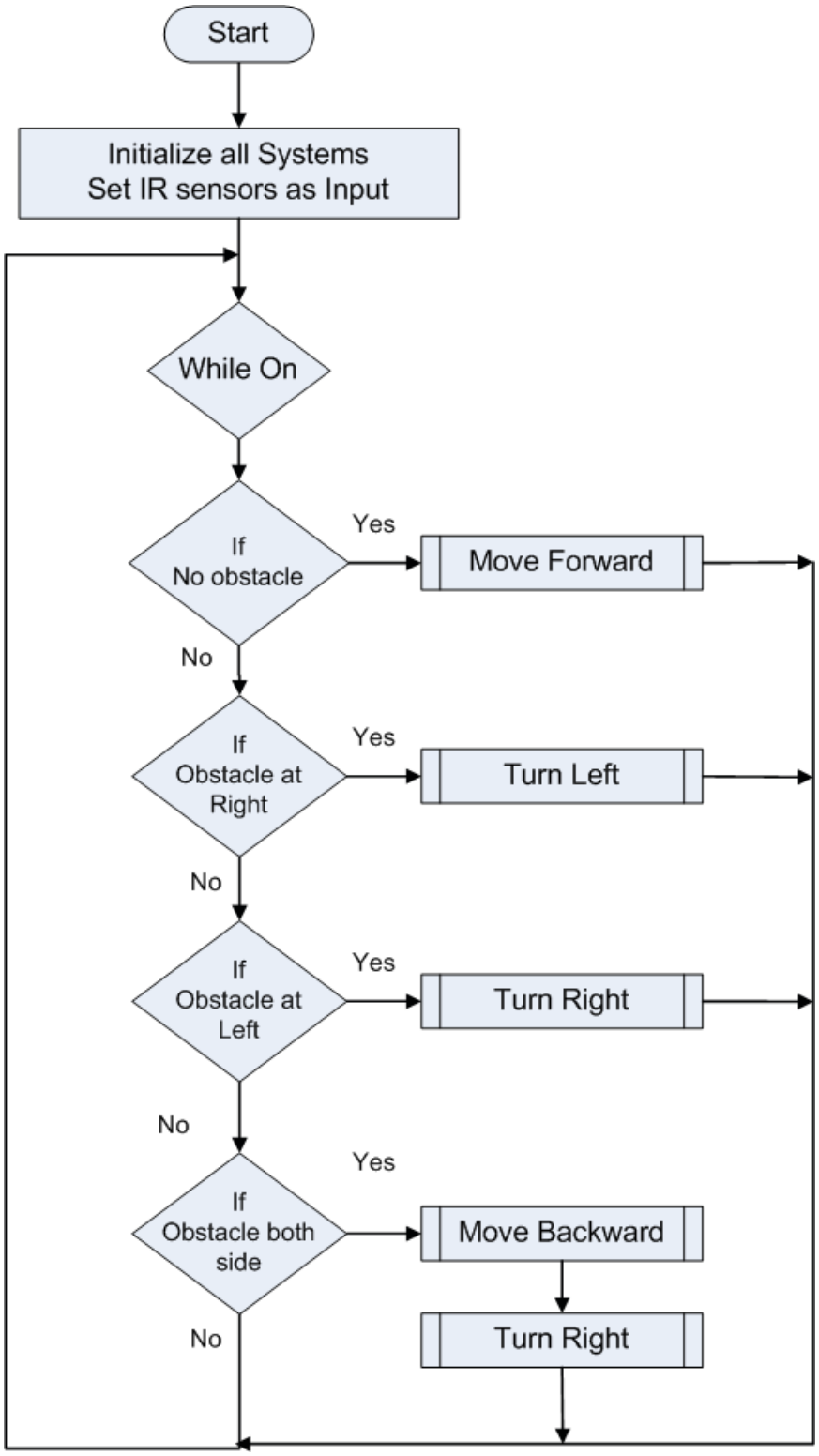
6. Flowchart for Navigation



```
main()
{
robosoft();
rest();
  while(1)
     {
      IR_REMOTE();

      if(FWD)              //if forward flag set go forward
          {
          goforward();
          }

      if(BWD)              //if Backward flag set go Backward
          {
          gobackward();
          }

      if(LEFT)             //if Left flag set go Left
          {
          turnleft();
          }

      if(RIGHT)            //if Right flag set go Right
          {
          turnright();
          }

      if(STOP)             //If stop flag set Stop
          {
         rest();
          }

     }

}
```

Flow Chart

6. Flowchart for Obstacle Avoiding

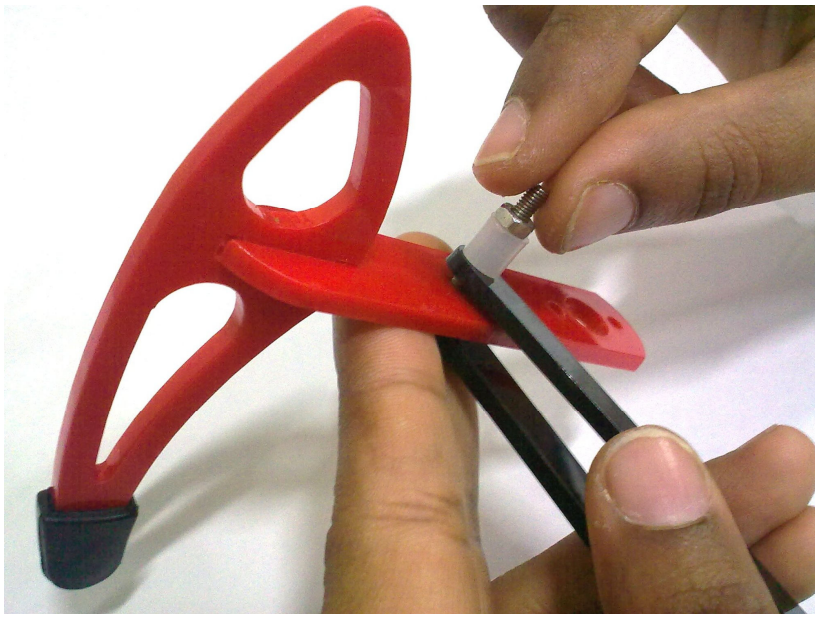


```
main()
{
robosoft();
Lobs=1;                 //setting sensors as input pin
Robs=1;                  //setting sensors as input pin
rest ();
while(1)
        {
        if(Lobs==0 && Robs==0)
                {
                goforward();
                }

        if(Lobs==1 && Robs==0)
                {
                turnright();
                }

        if(Lobs==0 && Robs==1)
                {
                turnleft();
                }

        if(Lobs==1 && Robs==1)
                {
                gobackward();
                gobackward();
                gobackward();
                gobackward();
                turnright();
                turnright();
                turnright();
                }
        }
}
```

Assembly Manual:

Step 1: Fix the leg as shown in image and then apply fevikwik to stick it permanently Repeat same procedure for all 4 legs	
Step 2: Similarly fix the bottom plate also	

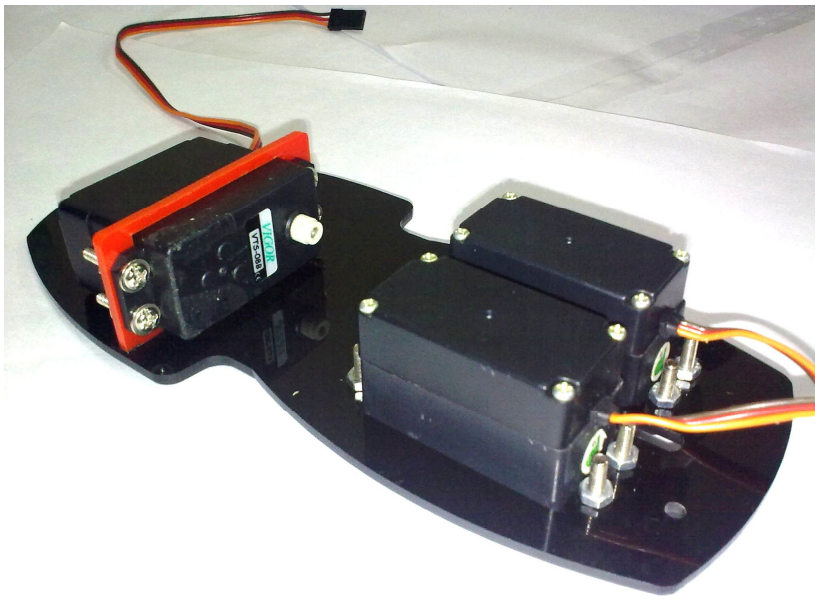
Step 3: Mount the rubber cap to all six legs	

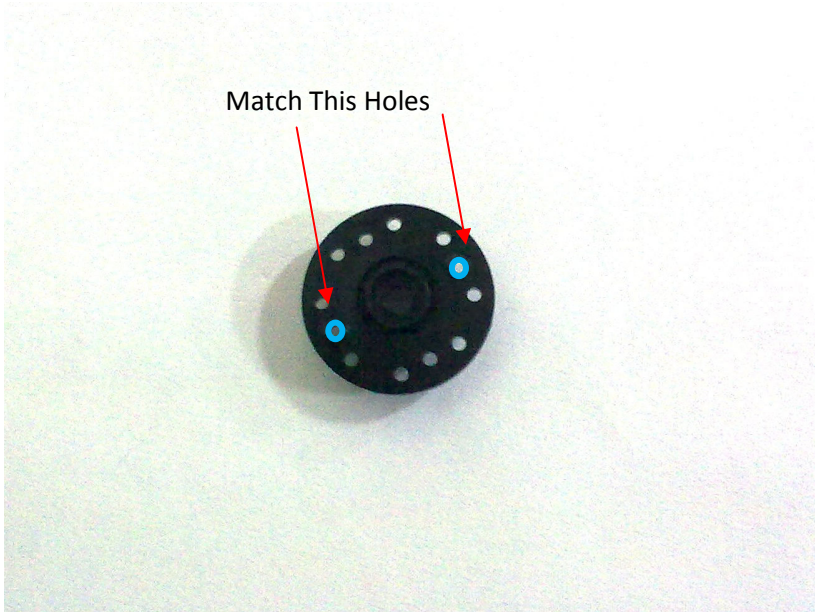
Step 4: Fix the two long linkages between two legs Put the screw from bottom, sandwich the leg between two linkages, Put the white spacer on and then fix it with nut as shown in figure	

Step 5: Your pair of leg should look like this	

Step 6: Put the servo inside a slot on the top. Match the notch of slot with cable of motor as shown in figure	

Step 7: Now fix the servo motors using screws and nuts	

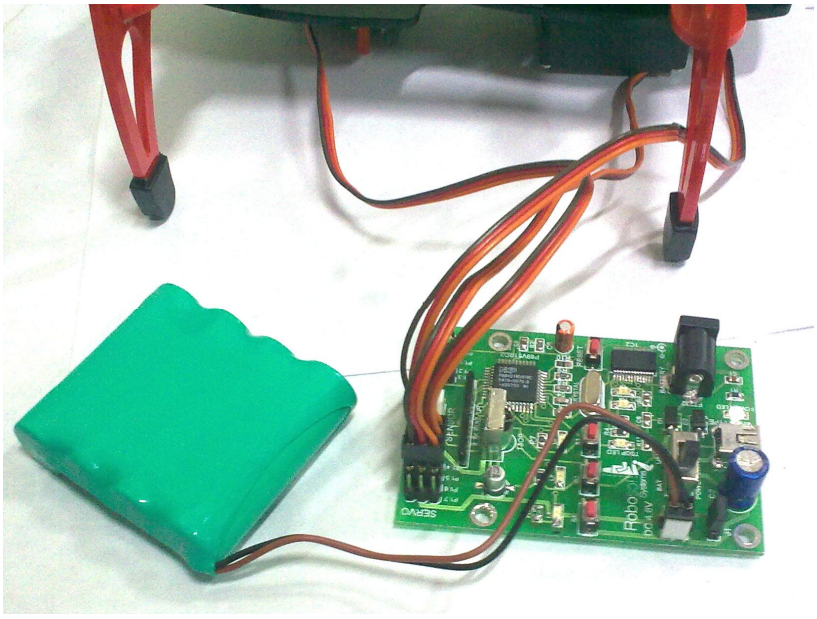
Step 8: Similarly you have to fix servo motor on bottom plate and fix it with screw and nuts	

Step 9: This is a servo ring plate you will be mounting this plates ion machine legs But remember u have to match holes in images while mounting legs.	

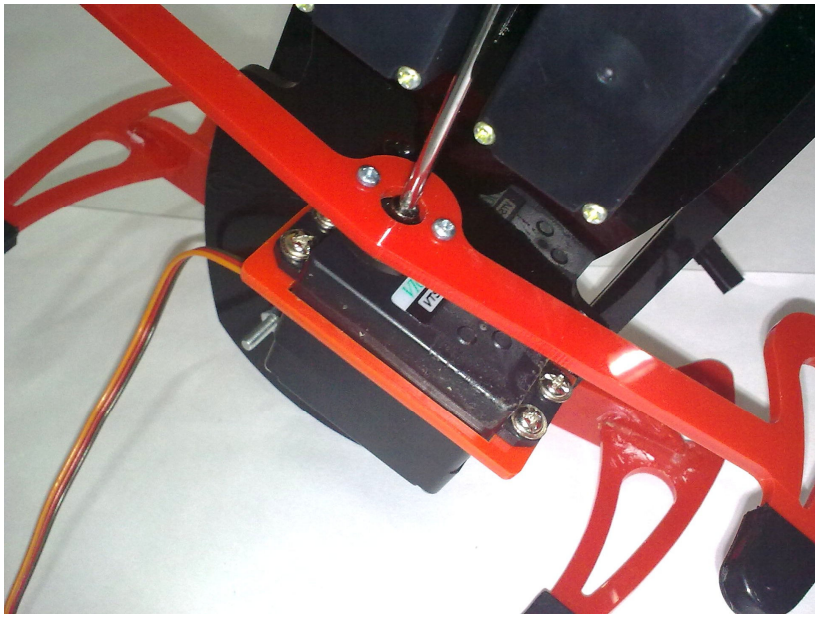
Step 10: Fix this circular plate on machine leg with small self threading screws and match two holes from previous steps	

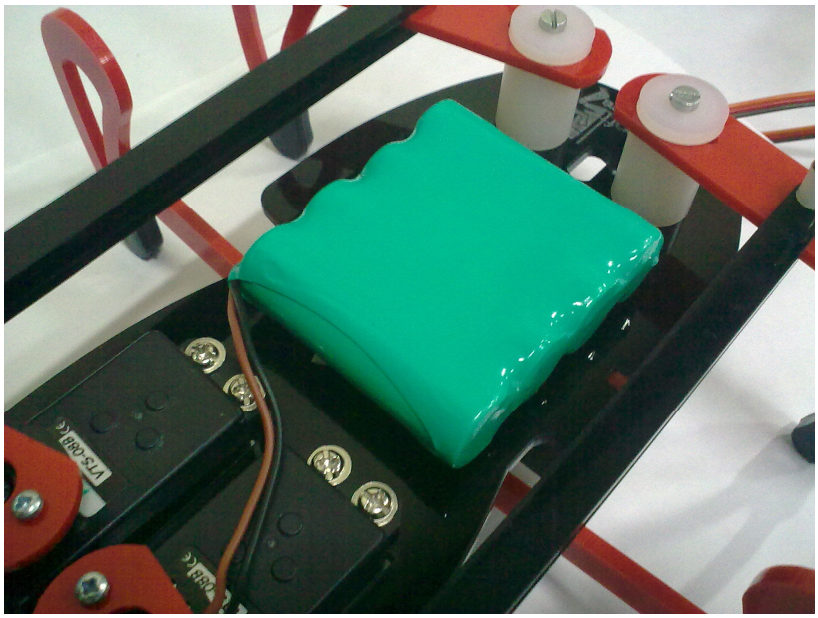
Step 11: Similarly fix the plate on center leg also as shown in fig.	

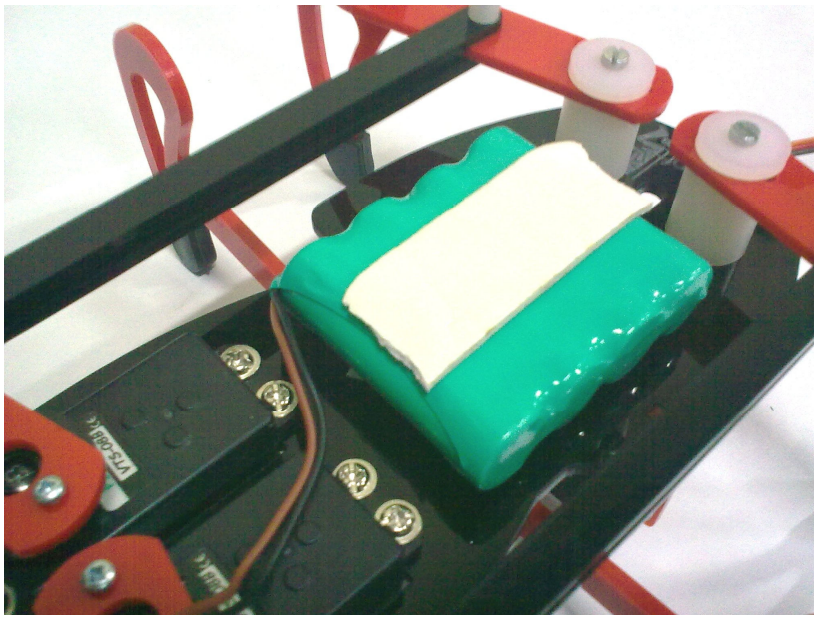
Step 12: Now fix both sets of legs using white big spacers And now your machine should look like this	

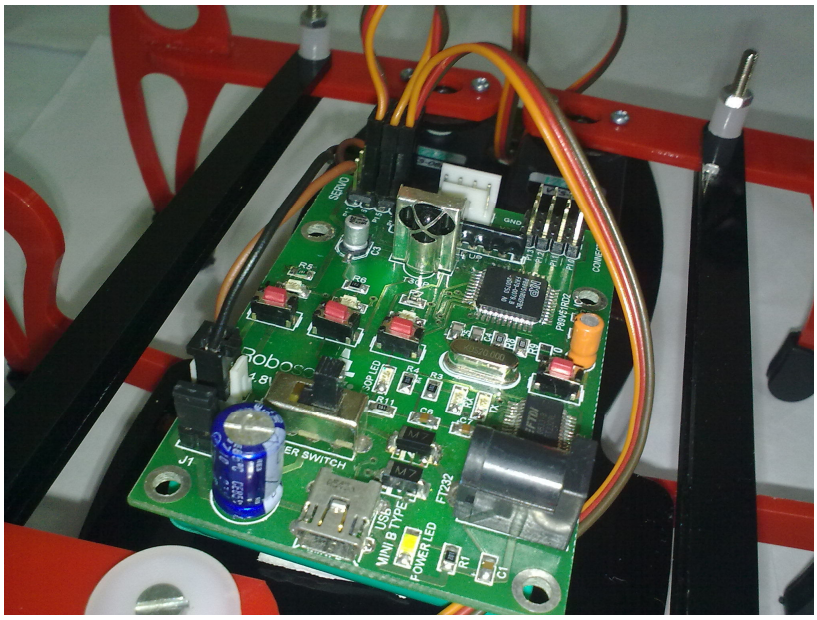
Step 13: Now connect your machine to controller board Load default position code on your board and set resting position for all motors	

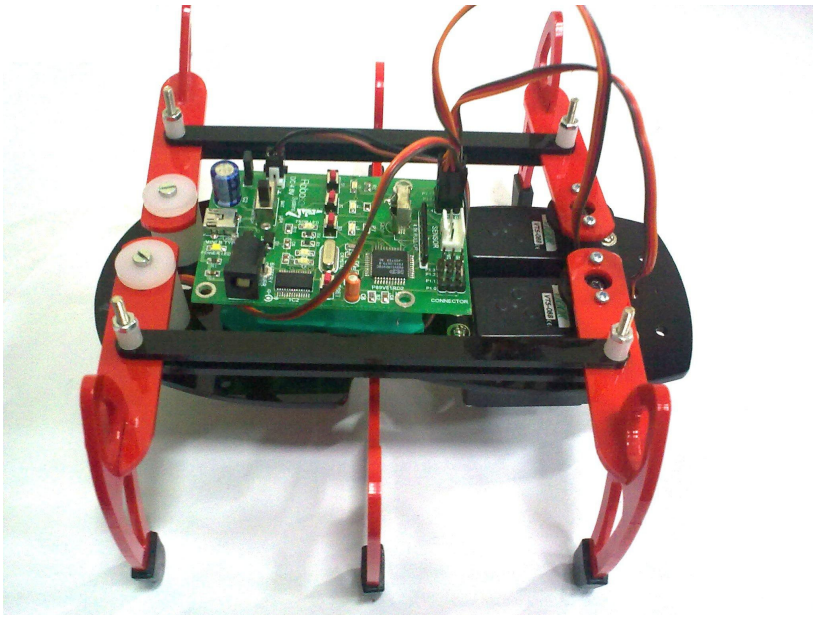
Step 14: Once you fix the position of servo you can mount your legs of machine on servo motors It should be exactly perpendicular to main machine body	

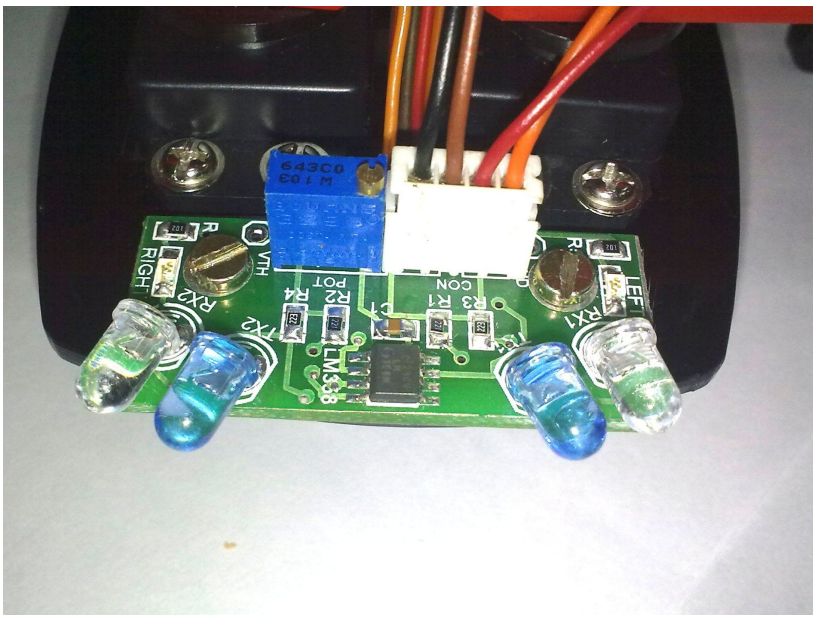
Step 15: Similarly mount center leg and you	

Step 16: Now fix battery on machine top using double sided tape	

Step 17: Again put double side tape on top of battery	

Step 18: And now fix your board on battery And do the servo and battery Connection	

Step 17: Now your machine look like this	

Step 18: Now mount the sensor PCB using 3x12 mm screws and nuts And your machine is ready to use	

Robosoft Systems

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) 66751507
Mobile: (91) 9930776661