ROBOTIC VISION USING MATLAB

EFY SANI THEO TESTED

■ ARINDAM CHAKRABORTY, ANANDAMOYEE SENGUPTA, SUNANDA CHATTERJEE, NILASISH ACHARYA

Building a robot is not only a passion but also a dream for most budding engineers. Competitions and R&D in robotics are already being encouraged in various institutions in our country. This article describes a visual sensor system used in the field of robotics for identification and tracing of the object. The program is designed to capture a red ball through a PC-based webcam using Matlab software.

The article does not aim at designing of electronic circuits and mechanical parts of the robot. It describes image capturing and processing techniques, followed by an introduction to actual robotic application to trace the red ball using the serial COM port of the PC.

Robotic vision and control

The whole system of making a robot to follow a red ball can be divided into four blocks: image acquisition, processing of image, decision-making and motion control.

Image acquisition can be achieved by using a PC-based webcam or a digital video camera. This device will capture the image and send it to the camera processor for further processing in the computer. Its main function is to convert the light energy received into electrical signals.

Image processing involves conversion of RGB colour images into grayscale images, setting of threshold levels, saturation of the features into binary images and setting of cut-off values to remove noise in the binary image.

Decision-making is done through the software program and motion control through either software or constant monitoring by the operator from the keyboard.

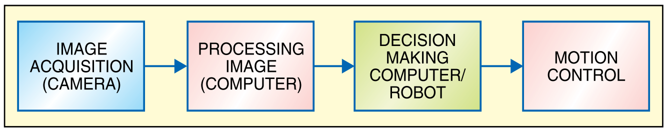
IMAGE ACQUISITION (CAMERA)
PROCESSING IMAGE (COMPUTER)
DECISION MAKING COMPUTER/ ROBOT
MOTION CONTROL

Fig. 1: Block diagram of the robotic vision and control system

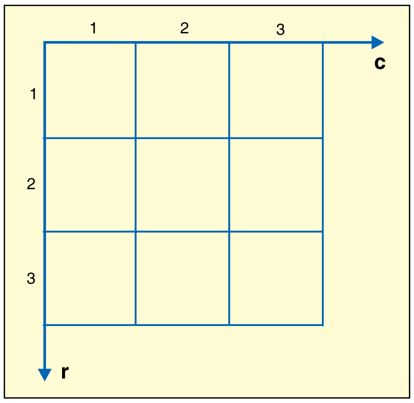

Fig. 2: Pixel coordinates

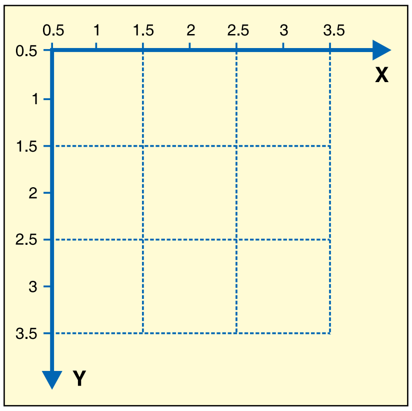

Fig. 3: Spatial coordinates

Before going to the software program in detail, let us see the coordinate systems to understand the image acquisition process.

Pixel coordinates. Generally, the most convenient method for expressing locations in an image is to use pixel coordinates. Here, the image is treated as a grid of discrete elements, ordered from top to bottom and left to right. That is, the first component 'r' (the row) increases downward, while the second component 'c' (the column) increases to the right (see Fig. 2). For example, the data for the pixel in the third row, second column is stored in the matrix element (3, 2). When the syntax for a function uses 'r' and 'c,' it refers to the pixel coordinate system. You can use Matlab matrix subscripting to access the values of individual pixels.

Spatial coordinates. Consider a pixel as a square patch. From this perspective, a location such as (3.3, 2.2) is a spatial coordinate. Here, locations in an image on a plane are described in terms of 'x' and 'y.' When the syntax uses 'x' and 'y,' it refers to the spatial coordinate system. Fig. 3 shows the coordinate convention of the spatial coordinate system. Notice that 'y' increases downward.

An image may be defined as a two-dimensional f(x, y**)** function**,** where 'x' and 'y' are spatial coordinates, and the amplitude of 'f' at any pair of coordinates (x, y) is called the intensity or gray level of the image at the point. The finite discrete values of the coordinates (x, y) and amplitude of 'f' are the digital images**.**

An intensity image is a data matrix whose values have been scaled to represent intensities. When the elements of an intensity image are of class 'uint8' or class 'uint16,' they have integer values in the range [0, 255] and [0, 65535], respectively. The class of the elements used in this program is 'uint8.'

Software program

After installing the Matlab software, you will see its icon on your compu-

TABLE I
Instructions for Image Acquisition

Instruction	Descriptions
preview(obj)	Immediately activates a live image preview window for the video input object 'obj'.
closepreview(obj)	Closes the image preview window associated with image acquisition object 'obj'.
celldisp(c) or celldisp (dev_info. SupportedFormats)	Recursively displays the contents of a cell array
start(obj)	The control starts capturing the frames. The frames captured are stored in the memory. The control also obtains exclusive use of the image acquisition device associated with the video input object 'obj' and locks the device configuration. Starting an object is a necessary first step to acquire image data.
triggerconfig(obj, 'manual')	Sets the trigger configuration. Here, 'manual' gets the image only when the video is triggered.
set(obj, 'FramesPerTrigger', 1)	Sets the number of frames captured per trigger. Here only one frame is captured per trigger.
set(obj, 'TriggerRepeat', Inf)	Sets the number of times for which the trigger can be repeated.
flushdata(obj)	Removes all data from the data acquisition engine and resets the SamplesAvailable property to zero.
stop(obj)	Stops video capturing.
delete(obj)	Deletes the object 'obj'
clear obj	Clears the memory buffer

Note: Inf- indicates that there can be any number of triggers we want. If we specify any number then we will have to start the video capture again after we have finished triggering the specified number of times.

ter's desktop. Run the software and the screen shows the main Matlab application window containing the following sub-windows:

1. Command window. In this window, the user types Matlab commands and expressions at '>>' prompt and the outputs of these commands are displayed.

2. Workspace browser. This window shows variables defined in the current session and also some information about them.

3. Current directory. This window displays contents of the current directory and its path.

4. Command history window. This window contains a record of the commands (including both current and previous sessions) that the user has entered in 'Command' window.

5. Others. There are one or more windows which are shown only when the user activates graphics.

Matlab has built-in adaptors to access the camera. At the Matlab command prompt, write the 'imaqhwinfo' instruction and press 'Enter' key. This instruction returns information about all the adaptors available in the system:

```
ans =
InstalledAdaptors: {'winvideo'}
MATLABVersion: '7.0.1 (R14SP1)'
ToolboxName: 'Image Acquisition
Toolbox'
ToolboxVersion: '1.7 (R14SP1)'
```

The 'info=imaqhwinfo' instruction returns information about a specific device accessible through a particular adaptor.

The output of the 'info = imaqhwinfo('winvideo')' instruction is:

```
info =
AdaptorDllName:G:\MATLAB701\toolbox\
imaq\imaqadaptors\win32\
mwwinvideoimaq.dll'
AdaptorDllVersion: '1.7 (R14SP1)'
AdaptorName: 'winvideo'
DeviceIDs: {1x0 cell}
DeviceInfo: [1x0 struct]
```

The output of the 'dev_info = imaqhwinfo('winvideo', 1)' instruction is:

```
dev_info =
DefaultFormat: 'RGB24_640x480'
DeviceFileSupported: 0
DeviceName: 'WebCam Vista #2'
DeviceID: 1
ObjectConstructor:
'videoinput('winvideo', 1)'
SupportedFormats: {1x11 cell}
```

Again, the 'obj = videoinput('adaptorname', deviceID,' format')' instruction constructs a video input object where the 'adaptorname' string specifies the name of the device adaptor that the object 'obj' is associated with. 'deviceID' is the identifier of the device in numerical number. If 'deviceID' is not specified, the first available device ID is used. The 'format' string specifies the video format for the object. If the format is not specified, the device default format is used. For example, the output of the 'obj=videoinput ('winvideo', 1, 'RGB24_640×480')' instruction will be:

```
Summary of Video Input Object Using
'WebCam Vista'.
Acquisition Source(s): input1 is
available.
Acquisition Parameters: 'input1' is
the current selected source.
10 frames per trigger using the
selected source.
'RGB24_640×480' video data to be
logged upon START.
Grabbing first of every 1 frame(s).
Log data to 'memory' on trigger.
Trigger Parameters: 1 'immediate'
trigger(s) on START.
Status: Waiting for START.
0 frames acquired since starting.
0 frames available for GETDATA.
```

Binary images (B&W). A binary image is a logical array of 0's and 1's. Numeric-array images consisting of 0's and 1's are converted into binary, with '1' indicating white colour (maximum intensity) and '0' indicating black colour (minimum intensity):

```
>>imBw=[1 0 1;0 0 1;1 1 0]
```

RGB images. A red, green and blue (RGB) image is an MxNx3 array of colour pixels, where each colour pixel is a triplet corresponding to the red, green and blue components of an RGB image at a specific spatial location.

An RGB image may be viewed as a stack of three gray-scale images that, when fed into the RGB colour monitors, produce a colour image. Eight bits are used to represent the pixel values of each component of the image. Thus an RGB image corresponds to 24 bits.

Detection of the red ball

1. Capture a frame and store it in a variable, say, 'rgb_image'

2. Extract the red, green and blue components of the images and store them in variables fR, fG and fB:

```
fR= rgb_image ( : , : , 1);%extracts
the red component.
fG= rgb_image ( : , : , 2);%extracts
the green component.
fB= rgb_image (: , : , 3);%extracts
the blue component.
```

Here, fR, fG and fB are image matrices. In Matlab, comments are written after '%' sign.

3. Next, find the red object in the image. (R_THRESHOLD=)140, (G_THRESHOLD=)105 and (B_THRESHOLD=)100 are specific numbers called 'threshold.' The technique for finding these numbers is described later on.

The following statement creates a B&W image array 'I':

```
I= ((fR≥140) & (fG≤105) & (fB≤100));
```

That is, the result of logically 'ANDed' image matrices fR, fG and fB is stored in 'I.'

If the following three conditions are satisfied, the pixel value of the image is set to '1':

(i) fR≥140 if the value of the red component of the pixel is greater than 140.

(ii) fG≤105 if the value of the green component of the pixel is less than 105

(iii) fB≤100 if the value of the blue component of the pixel is less than 100

4. Once you make the B&W image, you will find that apart from the region of your red ball there are also some unwanted white regions in the image. These unwanted white regions are called 'noise.'

Before you plot the centre of the image, filter the noisy parts of the image as follows:

```
Se = strel ('disk', 20); % creates a
flat, disk-shaped structuring element
with radius 20
B= imopen (I, se);%morphological
opening
Final= imclose (B, se);%morphological
closing
```

Morphological opening removes those regions of an object which cannot contain the structuring element, smoothes object contours, breaks thin connections and removes thin protrusions. Morphological closing also tends to smooth the contours of objects besides joining narrow breaks and filling long, thin gulfs and holes smaller than the structuring element:

5. Once you obtain the desired part, find the centre of the ball.

The following statement computes all the connected components in a binary image:

```
[L, n]= bwlabel(Final),
```

Here 'n' is the total number of connected components and 'L' is the label matrix. (Each connected component is given a unique number.)

The following statement:

```
[r, c]= find (L= = K) % K= 1, 2 …n
```

returns the row and column indices for all pixels belonging to the Kth object:

rbar= mean (n);

cbar= mean(c);

Variables 'rbar' and 'cbar' are the coordinates of the centre of mass.

As you have already filtered the image, the final image contains only one white region. But in case there is a computational fault due to excessive noise, you might have two connected components. So form a loop from '1' to 'n' using 'for' statement, thus calculating the centre of mass for all objects. The syntax is:

```
for K=1:n
```

If there are no components in a frame, the control doesn't enter the loop and 'rbar' and 'cbar' remain initialised to zero.

For checking the output on the computer, use the following instructions:

```
imshow (rgb_image);
hold on
plot (cbar, rbar, 'marker', ' * ',
```

TABLE II
Instructions for Image Processing

Instruction	Description
im = imread('redball.jpg')	Reads the JPEG image into an image array 'im'
size(im)	Returns the row and column dimensions of an image; for example, [m, n] = size (im);
imshow(im)	Displays the image on the Matlab desktop.
figure, imshow (im2)	Displays image 'im2' in another window; used when needed to display two images in two different windows
imview(im)	Displays the image on the Matlab Desktop. It also shows the pixel value at the corresponding mouse pointer
imwrite(A,filename,fmt)	Writes the image 'A' to the file specified by filename in the format specified by 'fmt'
tf = islogical(A)	Returns logic 1 (true) if 'A' is a logical array and logic '0' (false) otherwise
I = rgb2gray(RGB)	Converts the true-colour RGB image into the grayscale intensity image 'I'
BW = im2bw(I)	Converts the intensity image 'I' into B&W

Note: 1. Syntax of the functions specified can be obtained from the Matlab Help.
2. If the image file is not in the current directory, the whole path of the file should be specified.
3. After every statement there should be a semicolon. Semicolon is given to suppress the output at the command window.

TABLE III
Decision Table

cbar ≥ x1	cbar ≤ x2	rbar ≥ y1	rbar ≤ y2	Binary	Decimal(e=)	Move
0	0	0	0	0	0	X
0	0	0	1	01	1	X
0	0	1	0	10	2	X
0	0	1	1	11	3	X
0	1	0	0	100	4	X
0	1	0	1	101	5	Left
0	1	1	0	110	6	Left
0	1	1	1	111	7	Left
1	0	0	0	1000	8	X
1	0	0	1	1001	9	Right
1	0	1	0	1010	10	Right
1	0	1	1	1011	11	Right
1	1	0	0	1100	12	X
1	1	0	1	1101	13	Forward
1	1	1	0	1110	14	Backward
1	1	1	1	1111	15	Stop

```
'MarkerEdgeColor', B );
```

These statements pop-up a window where a 'blue' mark is plotted on the detected centre of mass of the red ball. They have been commented out in the main program for testing.

Plotting the position of the object

The position of the red ball is plotted as described below:

1. Divide the frame captured by the camera (refer Fig. 4) into five parts by means of points 'x1' and 'x2' on X-axis and points 'y1' and 'y2' on Y-axis.

2. Calculate x1, x2, y1 and y2 by the following method:

x1= x/2-numx; x2=x/2+numx
y1=y/2-numy; y2=y/2+numy

'numx' and 'numy' are arbitrary numbers which you have to find out. These depend on the size of your ball. We have taken numx=120 and numy=30 in our program.

Calculate the coordinates of the centre of the frame, which is nothing but (x/2, y/2). 'x' is the maximum dimension of X-axis (in the program it is 640) and 'y' is the maximum dimension of Y-axis (in the program it is 480).

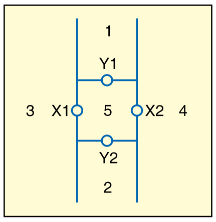

Fig. 4: Captured frame

3. Various conditions for detecting the position of the ball are:

(i) If the ball is in region 5, it is at the centre of the frame. The robot should stop moving.

(ii) If the ball is in region 3, it is at the left of the frame. The robot should move left.

(iii) If the ball is in region 4, it is at the right of the frame. The robot should move right.

(iv) If the ball is in region 1, it is at the upper part of the frame. The robot should move forward.

(v) If the ball is in region 2, it is at the lower part of the frame. The robot should move backward.

Define the above conditions in the code for decision making as given below:

(i) cbar≥x1 (output either '0' or '1')

(ii) cbar≤x2 (output either '0' or '1')

(iii) rbar≤y2 (output either '0' or '1')

(iv) rbar≥y1 (output either '0' or '1')

4. If you consider these four conditions to be four bits, you can easily convert the information into a number for easy computation. This can be done by the following code:

```
e=((cbar>= x1)*2*2*2 % bit number 3
+(cbar<= x2)*2*2     % bit number 2
+(rbar>= y1)*2       % bit number 1
+(rbar<= y2))        % bit number 0
```

Thereafter, you can simply generate a simple switch-case code for outputting the appropriate data through serial port, to control the external device such as a robot.

The decision table derived from the above conditions is shown in Table III. 'X' denotes a "don't care" situation. Some of the cases are imaginary in the practical world. The robot will stop in such situations.

For calculating the dimensions of the frame, use the sizeof( ) function. This function returns the size of the frame.

Setting the RGB threshold values

The red ball is of particular interest. The steps for setting the RGB threshold values of the red ball follow:

1. Take at least ten snaps of the red ball at various angles (refer Fig. 5)

2. Read each image

3. Display the image by using 'imview' function

4. Note down the pixel values of the red ball by placing the mouse cursor at various portions of the red ball. The threshold for red component (R_THRESHOLD) should be the least value of the red component found in the red ball. The threshold for green component (G_THRESHOLD) should be the maximum value of the green component found in the red ball. The threshold for blue component (B_THRESHOLD) should be the maximum value of the blue component found in the red ball.

In the screenshot shown in Fig. 5, the least value of the red component is '144.' So R_THRESHOLD can be taken as '144.' The maximum value of the green component is '115.' So G_THRESHOLD can be taken as '115.'

The maximum value of the blue component is '100.' So B_THRESHOLD can be taken as '100.'

Syntax of the functions specified can be had from the Matlab Help. If the image file is not in the current directory, specify the whole path of the file. A semicolon is given after every statement to suppress the output at the command window.

Motion control of the robot

The computer processes the image of the red ball and sends out five different data through its serial COM port. The data depends on the location of the red ball, viz, upper, lower, left, right and centre of the frame, as captured by the camera.

Place the PC-based (USB-based) camera in front of the robot, so that it acts as a visual sensor.

You have to design a robot that connects to the serial port of your PC.

Program the serial port as follows:

1. obj = serial('port'); creates a serial-port object associated with the serial port specified by the port. If the port does not exist or if it is in use, you will not be able to connect the serial-port object to the device. The syntax is:

```
>>ser=serial('COM1');
```

2. obj = serial('port','Property Name',PropertyValue,...); creates a serial-port object with the specified property names and property values. If an invalid property name or property value is specified, an error is returned and the serial-port object is not created. The syntax is:

```
>> ser= serial('COM1','BaudRate',
9600,'DataBits',8);
```

3. fopen(obj); before you can perform a read or write operation, 'obj' must be connected to the device with the fopen function. The syntax is:

```
>>fopen(ser);
```

4. fprintf(ser,'data'); sends data to the serial port. The syntax is:

```
>>fprintf(ser,'F');
```

5. fclose(obj); disconnects 'obj' from the device. The syntax is:

```
>>fclose(ser);
```

Here the robot will be controlled by seding the following codes to the serial port:

1. Key 'F' for forward movement
2. Key 'L' for left movement
3. Key 'R' for right movement
4. Key 'B' for backward movement

Any other code will make the robot stop. The code for implementing motion control of the robot is given at the end of this article.

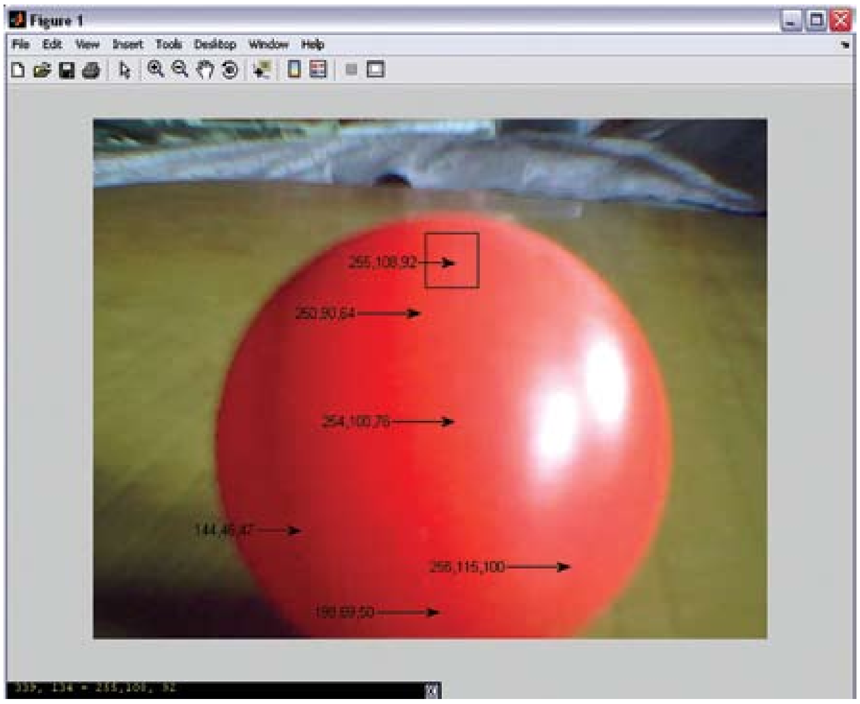

Fig. 5: Screenshot of the red ball

Testing steps

1. Install Matlab (minimum version: Matlab 7.0.1(R14))

2. Install the camera along with related software

3. Restart your computer

4. Run your camera software and check the image captured by it

5. Take snaps of the red ball from various angles and set the threshold values

6. When all is right, open Matlab software, go to 'm.file' from 'file' menu bar, type the code 'Robotvision.m' (given at the end of the article) and save it. Alternatively, copy the Robotvision.m file from this month's EFY-CD

7. Click 'run' button in 'Debug' option

8. Hold the red ball in your hand near the camera lens

9. Move the ball left, right, forward and backward. This information will be displayed on your screen. For example, if you move the ball left, you can see 'move left' command on the screen

10. Now, place your camera on top of your robot. The robot will trace the red ball by itself

To check the serial-port data communication, you can use a serial-port monitor, which can be freely downloaded from 'www.download.com/Free-Serial-Port-Monitor/3000-2212_4-70394.html'

Caution. If you want to end execution of your program, go to the Matlab command prompt and press 'Ctrl+C' key. Thereafter, write 'fclose(ser)' against the prompt or else you will not be able to access your serial port further.

EFY note. The source code of this article has been included in this month's EFY-CD.

ROBOTVISION.M

```
%*************************************
% Red Ball Tracker - Robot Vision
% Software used Matlab 7.0.1(R14)
% Algorithim used RGB Colour Detection
%*************************************
% SOME LINES HERE ARE COMMENTED OUT FOR
FASTER PROCESSING.THE READER SHOULD UN-
COMMENT THEM WHEN NEEDED

clear;
clc  % Clearing Matlab desktop
vid=videoinput('winvideo',1,'RGB24_
640X480');  %  Defining  the  video  input
object
set(vid,'FramesPerTrigger',1); % Setting
frames per trigger
%  preview(vid);  %/////   Showing  the
video of the moving Ball(TO BE USED %
% WHILE TESTING)

pause(10);% Waiting  for  a  certain  time
for the system to get initialised
rgb_image = getsnapshot(vid); % Storing
Image in an array variable
[a b c]= size(rgb_image); % Determining
the size of the captured frame.
y=a;
x=b;
% Defining Boundaries
x1=x/2-120;
x2=x/2+120;
y1=y/2-30;
y2=y/2+30;
ser=serial('COM1'); % Defining the speci-
fied COM Port to be used
fopen(ser);  %  starting  serial
Communication,opening serial port
while(1)
    rgb_image  =  getsnapshot(vid);  %
storing image in an array variable
    flushdata(vid); %Flushing the buffer
    rbar=0;
    cbar=0;
    e=0;
        fR=rgb_image(:,:,1);fG=rgb_
image(:,:,2);fB=rgb_image(:,:,3);% Stor-
ing  RGB  components  of  the  image  in
seperate arrays
        I=((fR>=140)  &  (fG<=105)  &
(fB<=100)); % Converting the RGB Image
into binary image///Detecting only the
red component
        % Following are the steps For De-
tecting the red ball
    se=strel('disk',20);
    B=imopen(I,se);
    final=imclose(B,se);
    [L,n]=bwlabel(final);
    %imshow(rgb_image); %////THIS IS TO
BE USED ONLY WHILE TESTING
    %hold on   % ////THIS IS TO BE USED
ONLY WHILE TESTING
    for k=1:n
        [r,c]=find(L==k);
        rbar=mean(r);
        cbar=mean(c);
         %plot(cbar,rbar,'Marker','*','
MarkerEdgeColor','B','MarkerSize',20)
%////THIS IS TO BE USED ONLY WHILE TEST-
ING
                e=(((cbar>=x1)*2*2*2)  +
((cbar<=x2)*2*2)  +  ((rbar>=y1)*2)  +
(rbar<=y2))  %  Converting  to  decimal
number
    end
    % Decision Making Conditions
    switch (e)
        case 5
                            disp('Move
left'),fprintf(ser,'L');
        case 6
                            disp('Move
left'),fprintf(ser,'L');
        case 7
                            disp('Move
left'),fprintf(ser,'L');
        case 9
            disp('Move right'),fprintf
(ser,'R');
        case 10
            disp('Move right'),fprintf
(ser,'R');
        case 11
            disp('Move right'),fprintf
(ser,'R');
        case 13
            disp('Move forward'),fprin
tf(ser,'F');
        case 14
                            disp('Move
back'),fprintf(ser,'B');
        otherwise
            disp('Stop Moving'),fprint
f(ser,'S');
    end
end
fclose(ser); % closing serial port
```

●

Do it Yourself Electronic Kits and learn in the process

Choose from a wide range of kits to support your creative ventures. Kits 'n' Spares, a venture of EFY Group, brings to you the kits, complete with components, circuits, PCBs, hard-to-find ICs and easy-to-use instruction manuals

NEW KITS

→ Chip-Talk Hobby Projects	Rs 1990/-
→ Cell Phone Operated Land Over	Rs 2500/-
→ Micro-Ultrasonic Distance Meter	Rs 1150/-
→ Secured Room Access System	Rs 1000/-
→ Microcontroller-based Bi-directional Visitor Counter	Rs 900/-
→ Microcontroller Based Code Lock	Rs 800/-
→ Microcontroller Based Heart Rate Meter	Rs 650/-
→ Microcontroller Based Tachometer	Rs 500/-
→ Automatic Low Power Emergency Light	Rs 450/-
→ PC-based Automatic Time Manager	Rs 250/-
→ FM Booster	Rs 60/-

** Please add 4% tax for local sale in delhi or 4% for order outside Delhi. Forwarding charges Rs 40/-per kit.*

For the 'engineer in you

Kits'n'Spares

Please send your remittance by DD/MO* to :

Kits'n'Spares
D-88/5, Okhla Industrial Area, Phase-1,
New Delhi 110020; Phone: 26371661, 26371662
E-Mail: kits@efyindia.com, kitsmktg@efyindia.com
Website: www.kitsnspares.com

Also available at Kits'n'Spares Outlet:

EFY Tech Center
303, Dohil Chambers, 46, Nehru Place,
New Delhi-110019; Phone: 46597697

To view our entire range of products and to purchase them, visit our website: www.kitsnspares.com

OUR DEALER NETWORK

■ AHEMDABAD—RELIEF ROAD, Ph: 09825200451, 079-22139841; NARANPURA—Ph:9898121289 ■ AIZWAL—Ph: 0389-2310719, 2311909, 09837055376 ■ AMBALA—Ph: 09315387182, 09416113930 ■ ASSAM-(GUWAHATI)—Ph: 09957193093 ■ BANGLORE—Ph: 080-41464573, 09341268861 ■ BOPAL—Ph: 0755-4281576, 09926554365 ■ BHUBANESHWAR—Ph: 09437279441 ■ BHATINDA—Ph: 09417723151 ■ BHILAI NAGAR—Ph: 09827192818 ■ BARAILLY—Ph: 0581-2556675, 09837055376 ■ CALCUTTA—Ph: 033-4947123, 32544558, 09433012210 ■ CHANDIGARH—Ph: 09815140504 ■ CHENNAI—Ph: 044-42046343, 09994545727 ■ CUDDALORE—Ph: 09344412298 ■ COIMBATORE—Ph: 0422-2394113, 2381083 ■ DELHI-(JANAKPURI)—Ph: 09899795696, 22501716 ■ DEHRADUN—Ph: 09412369744 ■ GHAZIABAD—BOOK WORLD, Ph: 09810636816; ORBITECH—Ph: 9891661368 ■ HISAR—Ph: 09466662123, 09812260340 ■ HYDERABAD—Ph: 040-24756823, 09849070411 ■ JALANDHAR—Ph: 09988299442, 0181-2602314 ■ JAIPUR—Ph: 09828123982 ■ JODHPUR—Ph: 09828128568 ■ KOLKATA—Ph: 033-32544558, 24947123 ■ LUDHIANA—Ph: 09815742497 ■ LUCKNOW—Ph: 09454109057 ■ MADURAI —Ph: 09842103477, 09842119512 ■ MANGLORE—Ph: 0824-2429533 ■ MUMBAI—Ph: 09967478289 ■ NAGARCOIL—Ph: 04652-222505 ■ NAGALAND—Ph: 03862-229799, 09436006160 ■ NOIDA—Ph: 9212331453 ■ OOTY—Ph: 09443117715 ■ PUNE—Ph: 020-24305057, 09422555754 ■ SOLAN—Ph: 09816076344 ■ SECUNDRABAD—Ph: 09396669217 ■ SHOLAPUR—Ph: 0217-2720099, 09823383081 ■ THRISSUR—Ph: 0487-2427567, 2427629 ■ TRICHY—Ph: 0431-2791294 ■ VARANSI—Ph: 0542-2277357