Introduction to Video analysis using MATLAB

Deepak Malani, REC Calicut
Anant Malewar, IIT Bombay

Foreword:

The wide area of vision based autonomous systems is broadly referred as machine vision. <u>We have dealt with a beginner level module in this field, which we refer as vision-controlled motion (VCM)</u>.

Keywords:

Machine Vision, Image acquisition, MATLAB, Vision based robot control

Table of Contents
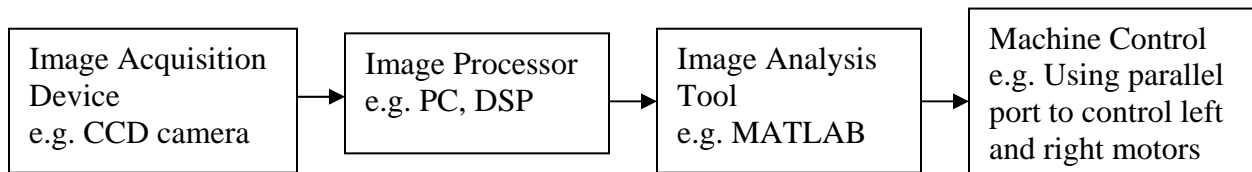
1   Introduction

In this section, an overview of vision controlled motion (VCM) as a system is explained.

1.1   *System Description:*

The system for VCM consists of:

1. Image acquisition setup: It consists of a video camera, web camera, or an analogue camera with suitable interface for connecting it to processor.

2. Processor: It consists of either a personal computer or a dedicated image processing unit.

3. Image analysis: Certain tools are used to analyze the content in the image captured and derive conclusions e.g. locating position of an object.

4. Machine control: After making the conclusion, mechanical action is to be taken e.g. Using serial or parallel port of a PC to control left and right motors of a robot to direct it towards the ball

Pictorially, the system can be represented as:

Image Acquisition Device e.g. CCD camera	→	Image Processor e.g. PC, DSP	→	Image Analysis Tool e.g. MATLAB	→	Machine Control e.g. Using parallel port to control left and right motors

1. Image capturing can be done using video camera available in various resolutions e.g. 640 x 480 pixels. There are two types of cameras generally available: Digital cameras (CCD – charge coupled device and CMOS sensor based) and analogue cameras. Digital cameras generally have a direct interface with computer (USB port), but analogue cameras require suitable grabbing card or TV tuner card for interfacing with PC.

Power requirements: CCD cameras give high quality, low-noise images. It generates an analog signal and uses analog to digital converter (ADC) and thus consumes high power.

CMOS cameras have lesser sensitivity resulting in poor image quality but consume lesser power resulting in more battery life.

2. Image analysis consists of extracting useful information from the captured images. We first decide the characteristics of the object to look for, in the image. This characteristic of the object must be as robust as possible. Generally, for the purpose of tracking or identifying the object we utilize:

i. Color

ii. Intensity

iii. Texture or pattern

iv. Edges – circular, straight, vertical stripes

v. Structure – Arrangement of objects in a specific manner

Quantitative/ Statistical analysis of image:

i. Center of gravity - point where the desired pixels can be balanced

ii. Pixel count – a high pixel count indicates presence of object

iii. Blob – an area of connected pixels

3. Machine control consists of controlling a robot based on the conclusion derived from image analysis.

To achieve this using PC, its parallel port or serial port can be used for driving the robot. For e.g. H-bridge, PWM control can be extensively used for right and left motor movement of a robot. Serial-port can be used for transferring data, which necessitates a micro-controller on the robot to interpret the data.

1.2 Tools for Image Processing

Once the image acquisition setup is ready, the captured images need to be stored in suitable format to process them. Generally, a raw image is stored as a matrix of color intensities.

MATLAB provides a very easy platform for image acquisition and processing. Even serial and parallel ports can be directly accessed using MATLAB. It serves as a handy tool for beginner level VCM. It provides a powerful built-in library of many useful functions for image processing.

An Open source equivalent of MATLAB is SCILAB (can be downloaded from http://www.scilab.org for free) and Scilab Image Processing (SIP) toolbox (http://siptoolbox.sourceforge.net). However, support available in SIP is limited as there are a few developers who contribute to this toolbox.

2 Getting started with MATLAB

MATLAB stands for MATrix LABoratory, a software developed by Mathworks Inc (www.mathworks.com). MATLAB provides extensive library support for various domains of scientific and engineering computations and simulations.

2.1 MATLAB Interface

When you click the MATLAB icon (from your desktop or Start>All Programs), you typically see three windows: Command Window, Workspace and Command History. Snapshots of these windows are shown below.
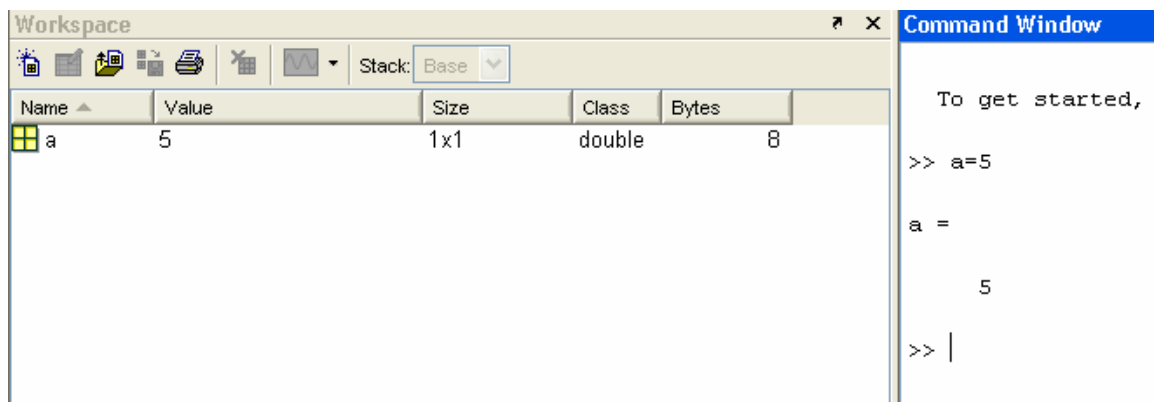
Command Window:
This is the window where you write the commands and see the output.
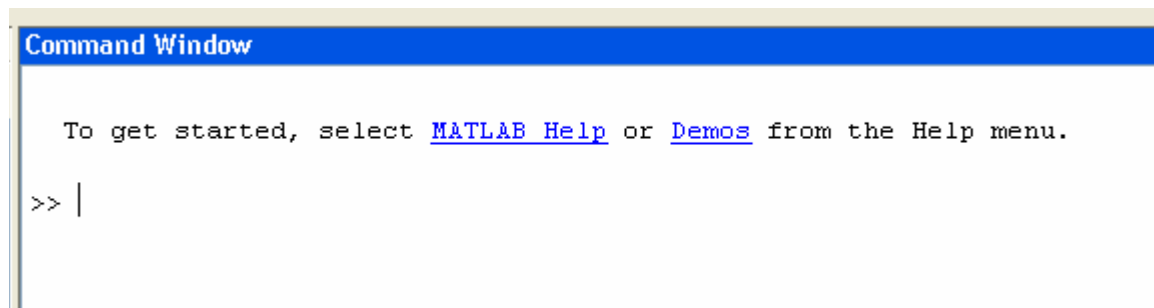
Workspace
This window shows the variables defined by you in current session on MATLAB.

Command History stores the list of recently used commands for quick reference.

Writing basic instructions:

In MATLAB, variables are stored as matrices (singular: matrix), which could be either an integer, real numbers or even complex numbers. These matrices bear some resemblance to array data structures (used in computer programming).

Let us start with writing simple instructions on MATLAB command window. To define an integer, type a=4 and hit enter.

>> a=5

As soon as you hit enter, the value of a is shown again on the command window. Also this variable is stored as a 1x1 matrix, as can be seen in the workspace.
To avoid seeing the variable, add a semicolon after the instruction

>> a=5;

Similarly to define a 2x2 matrix, the instruction in MATLAB is written as

>> b=[1 2; 3 4];

If you are familiar with operations on matrix, you can find the determinant or the inverse of the matrix.

>> determin= det(b)
>> d=inv(b)

2.2 Working with Images

Let us begin with the basic instructions to read and display an image in MATLAB.

2.2.1 Reading an image

Once you have started MATLAB, type the following in the Command Window

>> im=imread('cameraman.tif');

This command stores the image *cameraman.tif* in a variable with name *im.*

```
Command Window

  To get started, select MATLAB Help or Demos from the Help menu.

>> im=imread('cameraman.tif');
>>
```

The image cameraman.tif is stored in my computer at the following location (C:\Program Files\MATLAB71\toolbox\images\imdemos)
You may alternatively store an image in the following location and read it with the above command
C:\Program Files\MATLAB71\work

2.2.2   Displaying an image

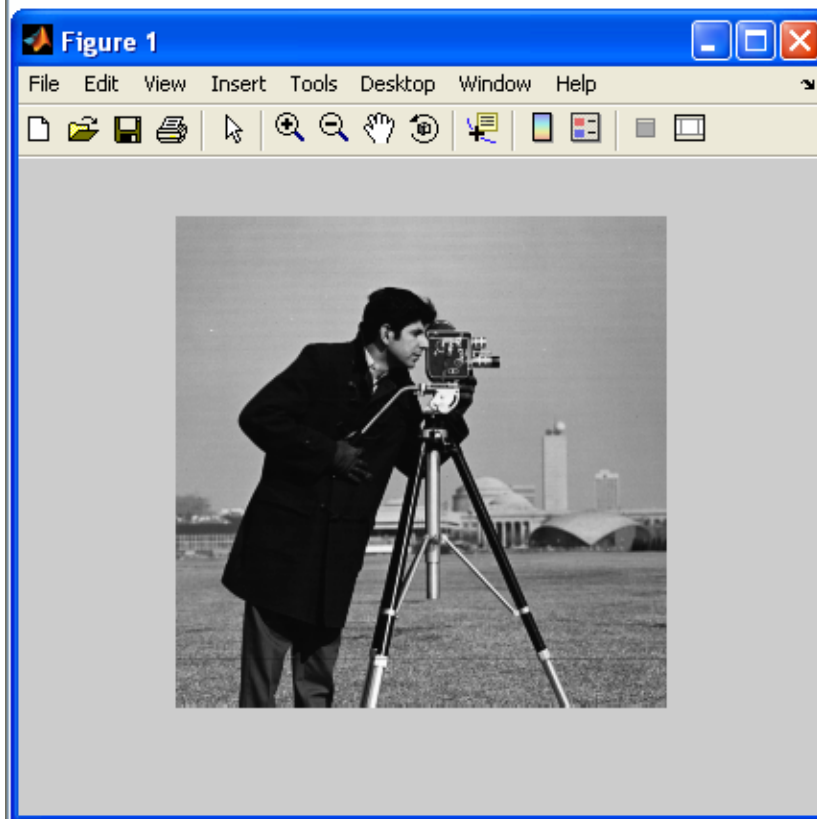You can display the image in another window by using *imshow* command

>>figure,imshow(im);

This pops up another window (called as figure window), and displays the image *im*.

Command Window

```
  To get started, select MATLAB Help or Demos from the Help menu.

>> im=imread('cameraman.tif');
>> figure,imshow(im);
>>
```


To know the breadth and height of the image, use the *size* function,
>>s=size(im);
>>s

Observe and Understand
In MATLAB, an image is stored as a two-dimensional matrix, where the matrix contains the values of intensity of light of the pixels. A pixel is the smallest possible image that can be depicted on your screen. Resolution of an image is equal to the number of pixels that can be fitted per square inch on your monitor.
To actually see the values stored in this matrix, type
>> im(1:10,1:10)
>> figure,imshow(im(50:150,50:150));

2.2.3   Generating an image

1. Binary image
An image which contains only black and white colors is called binary image.
>>imbin=[0 1;1 0];
>>figure,imshow(imbin,'InitialMagnification','fit')
Black indicates zero intensity and white indicates maximum intensity.

2. Grayscale image
A grayscale image contains intensities values in a range from zero to maximum including black and white. Typically this range is between 0 to 255. Thus values near to zero are dark shades of gray, while those near 255 indicate light gray.

>> imgray=[0.0 0.2 0.4 0.8 1.0];
>> figure,imshow(imgray,'InitialMagnification','fit')
or
>>imgray=[0 32 64 128 196 255];
>> figure,imshow(uint8(imgray),'InitialMagnification','fit')

<u>Note:</u>
By default, the range is 0 to 1 (type: double). However you can convert the image values from *double* type to *unsigned integer*(8 bits) *type* by using

>>imgray=uint8(imgray)

3. Colored images
A colored image can be considered composed of three basic colors: Red, Green and Blue. Hence it is also known as RGB image. A colored image is represented as 3D array with 3rd dimension used for indexing the color componets.

>> imRGB=imread('peppers.png');
>> figure,imshow(imRGB);

Here the matrix imrgb is a three dimensional matrix. To understand this, type
>> imRGB(1:5,1:5,:)

To understand how these 3 colors form the image, we will now try to display each color component separately:
>> size(imRGB)

ans =

  384   512   3

>> imR = zeros(384,512,3);
>> imR(:,:,1) = imRGB(:,:,1);
>> imG = zeros(384,512,3);

```
>> imG(:,:,2) = imRGB(:,:,2);
>> imB = zeros(384,512,3);
>> imB(:,:,3) = imRGB(:,:,3);
>> imtemp=[imR imG imB imRGB];
>> figure,imshow(imtemp);
```

Usually, each color component in a color image can be accessed separately as a 2D matrix as follows,
```
>> R = imRGB(:,:,1);
>> size(R)
```

ans =

  384   512

```
>> figure, imshow(R);
```

4. Data cursor
To see the values of the colors in the figure window, goto Tools>Data Cursor, and click over any point in the image. You can see the RGB values of the pixel at location (X,Y).

5. Converting RGB image to grayscale image
Colored image can be converted to a grayscale image by using rgb2gray function (a function available in image processing toolbox)
```
>>imGRAY=rgb2gray(imRGB);
```

6. Thresholding
A grayscale image (also referred as *intensity* image) can be converted to binary image by a process called thresholding. Image Processing toolbox provides a function *graythresh* to compute an optimal threshold value for converting grayscale image to a binary image.

```
>> level=graythresh(imGRAY);
>> imBW = im2bw(imGRAY,level);
>> figure, imshow(imBW);
```

This function converts pixel intensities between 0 to *level* to zero intensity (black) and between *level+1* to 255 to maximum (white)

2.3 *M-files*

MATLAB provides a facility to execute multiple command statements with a single command. This is done by writing a .m file. Goto File > New > M-file
For example, the graythresh function can be manually written as a m-file as:

```
im=imread('cameraman.tif');
figure,imshow(im);
```

```
s=size(im);

temp=im;
thresh=128;
for i=1:s(1,1)
   for j=1:s(1,2)
     if temp(i,j)<thresh
        temp(i,j)=0;
     else
        temp(i,j)=255;
     end
   end
end
figure,imshow(temp);
```
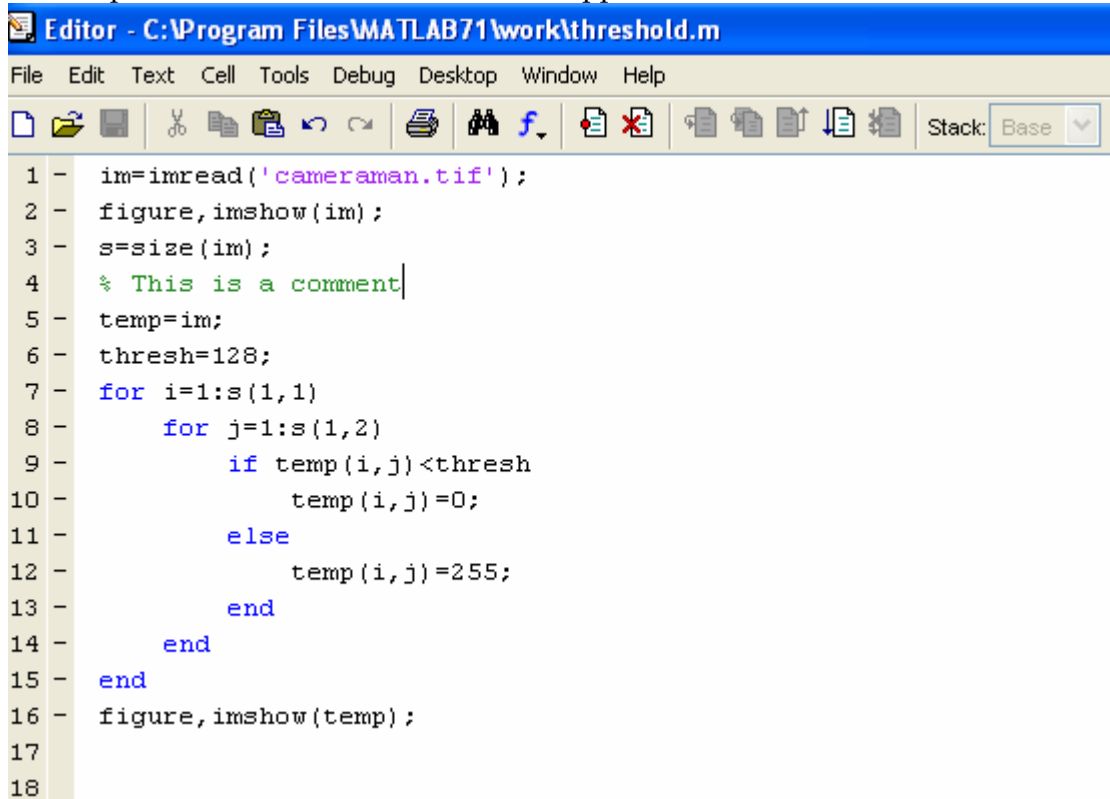
Save this file as "threshold.m" in the *work* directory and type

>>threshold

in the MATLAB command window.

The snapshot below shows how the .m file appears in the matlab editor.

Comments:

Observe that, comments (in green) can be written after the symbol '%'. A commented statement is not considered for execution.

M-files become a very handy utility for writing lengthy programs and can be saved and edited, as and when required. We shall now see, how to define your own *functions* in MATLAB.

2.4 Functions

Functions help in writing organized code with minimum repetition of logic. While writing m-files, it often happens that some set of common instructions are repetitively required. Instead of rewriting the instruction set every time, you can define a function, where you need to pass the data to be operated upon and chose the values you need the function to return.

The following function calculates the center of gravity of the object in the image. This file is to be saved with the filename same as the name of the function (e.g. center.m).

```
function [cent]= center(im)
s=size(im);
cx=0;cy=0;
n=1;
   for i=1:s(1,1)
     for j=1:s(1,2)
        if im(i,j)==255 % 1 or 255
           n=n+1;
           cx=cx+i;
           cy=cy+j;
        end
     end
   end
cxavg=cx/n;
cyavg=cy/n;
[cent]=[cyavg,cxavg];
```

This function can be called as follows:

>>cog=center(imscan);

This instruction will pass the matrix *imscan* to the function, and stores the value returned from the function to the variable *cog.*

3   Image Acquisition

Before proceeding further, let us look at the support provided by image acquisition toolbox in MATLAB.

Now-a-days most of the cameras are available with USB interface. Once you install the driver for the camera, the computer detects the device whenever you connect it. Alternatively, if you have a Digital Video camcorders or CCD camera connected with grabber card and interfaced with computer, Windows OS automatically detects the device.

In order t execute out further instructions, you will need a functional USB webcam connected to your PC.

In MATLAB, you can check if the support is available for your camera. MATLAB has built-in *adaptors* for accessing these devices. An adaptor is a software that MATLAB uses to communicate with an image acquisition device.

>> imaqhwinfo
>> cam=imaqhwinfo;
>> cam.InstalledAdaptors

To get more information about the device, type

>>dev_info = imaqhwinfo('winvideo',1)

Note: Instead of 'winvideo', if *imaqhwinfo* shows another adaptor, then type that adaptor name instead of 'winvideo'.

Your output should be similar to one shown in the snapshot below.

```
Command Window
>> dev_info = imaqhwinfo('winvideo',1)

dev_info =

            DefaultFormat: 'RGB24_320x240'
    DeviceFileSupported: 0
              DeviceName: 'USB PC Camera'
                DeviceID: 1
      ObjectConstructor: 'videoinput('winvideo', 1)'
       SupportedFormats: {1x10 cell}


>>
```

3.1   Previewing video

You can preview the video captured by the image by defining an object and associate it with the device.

>> vid=videoinput('winvideo',1, 'RGB24_320x240')

You will see the details of the acquisition parameters, as shown in the following snapshot.

```
Command Window
>> vid = videoinput('winvideo',1,'RGB24_320x240')

Summary of Video Input Object Using 'USB PC Camera'.

    Acquisition Source(s):  input1 is available.

  Acquisition Parameters:  'input1' is the current selected source.
                           10 frames per trigger using the selected source.
                           'RGB24_320x240' 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.
```

Now to see the video being captured by the camera , use *preview* command
>> preview(vid)

You should see a window pop-up, that displays what your camera is capturing.

The camera may support multiple video formats. To see for yourself all the supported formats, type

>>dev_info = imaqhwinfo('winvideo',1);
>>celldisp(dev_info.SupportedFormats); %displays list of supported formats

Check out for yourself the display of other formats, by replacing `RGB24_320x240` with other formats, in the definition of the object *vid*.

3.2   Capturing and storing images

1. Capturing an image
Now to capture an image from the video, define the object *vid* as described before and use *getdata* to capture a frame from the video.

>>start(vid);   % This command initiates capturing of frames and stores
                % the frames in memory
>>im=getdata(vid,1);
>>figure,imshow(im);

2. Storing the image
You can store the captured image as a *.jpg* or *.gif* file using *imwrite* function.

>>imwrite(im,'testimage.gif');

The image will be stored in 'MATLAB71\work' folder. Check this out yourself.

3.3 Getting ahead with Image Acquisition

Every time you want to capture an instantaneous image, you have to stop the video, start it again and use the *getdata* or *peekdata* function. To avoid this repetitive actions, the Image Acquisition toolbox provides an option for triggering the video object when required and capture an instantaneous frame. Create an m-file with following sequence of commands:

```
vid=videoinput('winvideo',1);
triggerconfig(vid,'manual');
set(vid,'FramesPerTrigger',1 );
set(vid,'TriggerRepeat', Inf);
start(vid);
for i=1:5
   trigger(vid);
   im= getdata(vid,1);
   figure,imshow(im);
end
stop(vid),delete(vid),clear vid;
```

In the above code, object *im* gets overwritten while execution of each of the *for* loop. To be able to see all the five images, replace *im* with *im(:,:,:,i)*.

In this code, the instructions 2-5 set the properties of object *vid. triggerconfig* sets the object to manual triggering, since its default triggering is of type *immediate.* In immediate triggering, the video is captured as soon as you *start* the object 'vid'. The captured frames are stored in memory. Getdata function can be used to access these frames. But in manual triggering, you get the image only when you '*trigger'* the video.

'FramesPerTrigger' decides the number of frames you want to capture each time '*trigger'* is executed.

TriggerRepeat has to be either equal to the number of frames you want to process in your program or it can be set to Inf. If set to Inf, you can use '*trigger'* as many times as you want. If set to any positive integer, you will have to '*start'* the video capture again after trigger is used for those many number of times.

Also, once you are done with acquiring of frames and have stored the images, you can stop the video capture and clear the stored frames from the memory buffer, using following commands:

```
>>stop(vid);
>>delete(vid);
>>clear vid;
```

Note: *getsnapshot* returns one image frame and is independent of FramesPerTrigger property.

So, if you want continuous grabbing of images, you can write an m-file as:

```
vid=videoinput('winvideo',1)
triggerconfig(vid,'manual');
set(vid,'FramesPerTrigger',1);
set(vid,'TriggerRepeat', Inf);
start(vid);
while(1)
{
    trigger(vid);
    im= getdata(vid,1);
    % write your image processing algorithm here
    %
    % you may break this infinite while loop if a certain condition is met
}
```

3.4 Interfacing with PC ports

MATLAB provides support to access serial port (also called as COM port) and parallel port (also called as printer port or LPT port) of a PC.

Note: If you are using a desktop PC or an old laptop, you will most probably have both, parallel and serial ports. However in newer laptops parallel port may not be available.
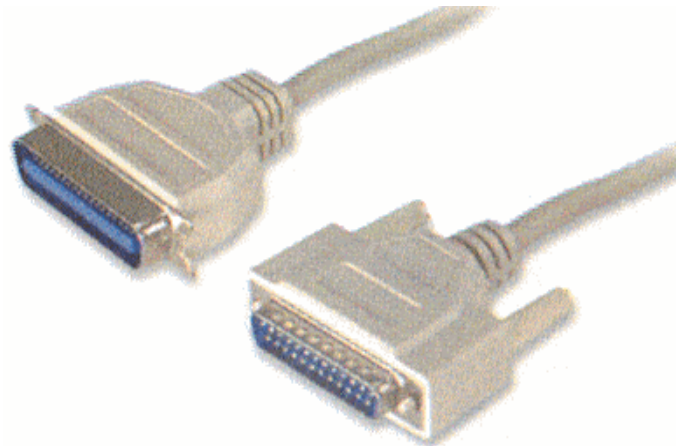
3.4.1 Parallel Port

Parallel port has 25 pins as shown in figure below. Parallel port cables are locally available (commonly referred as printer port cables). These cables are handy to connect port pins with your circuit. Pins 2-9 are bi-directional data pins (pin 9 gives the most significant bit (MSB)), pins 10-13 and 15 are output pins (status pins), pins 1,14,16,17 are input pins (control pins), while pins 18-25 are Ground pins.
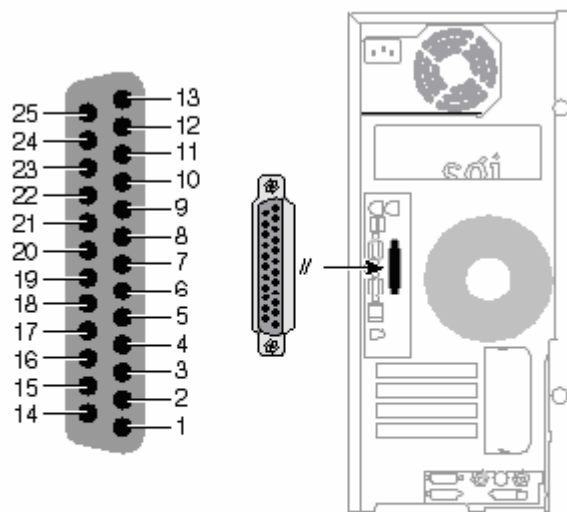
Parallel Port

courtesy: techpubs.sgi.com

Parallel port cables

courtesy: www.cknow.com

MATLAB has an *adaptor* to access the parallel port (similar to adaptor for image acquisition). To access the parallel port in MATLAB, define an object

>> parport= digitalio('parallel','LPT1');

You may obtain the port address using,

>> get(parport,'PortAddress')
>> daqhwinfo('parallel'); % To get data acquisition hardware information

You have to define the pins 2-9 as output pins, by using *addline* function

>> addline(parport, 0:7, 'out')

Now put the data which you want to output to the parallel port into a matrix; e.g.

>> dataout = logical([1 0 1 0 1 0 1 1]);

Now to output these values, use the *putvalue* function

>> putvalue(parport,dataout);

Alternatively, you can write the decimal equivalent of the binary data and output it.

>> data = 23;
>> putvalue(parport,data);

You can connect the pins of the parallel port to the driver IC for the left and right motors of your robot, and control the left, right, forward and backward motion of the vehicle. You will need a H-bridge for driving the motor in both clockwise and anti-clockwise directions.

3.4.2 Serial Port

If you have to transmit one byte of data, the serial port will transmit 8 bits as one bit at a time. The advantage is that a serial port needs only one wire to transmit the 8 bits (while a parallel port needs 8).
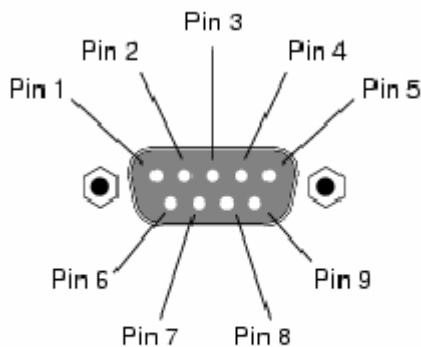
courtesy: techpubs.sgi.com
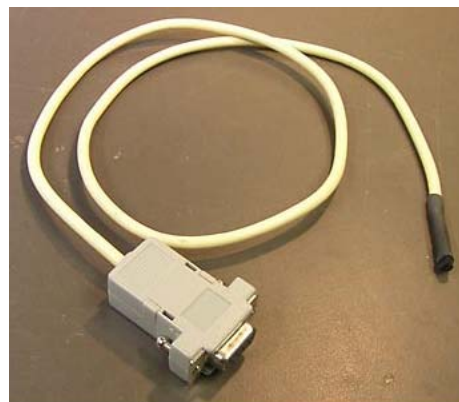
courtesy: martybugs.net

Pin 3 is the Transmit (TX) pin, pin 2 is the Receive (RX) pin and pin 5 is Ground pin. Other pins are used for controlling data communication in case of a modem. For the purpose of data transmission, only the pins 3 and 5 are required.

At the receiver side, you need a voltage level converter called as RS232 IC which is a standard for serial communication. And to interpret the serial data, a microcontroller with UART (Universal asynchronous receiver transmitter) is required aboard the robot. Most of the microcontrollers like AVR ATMEGA 8, Atmel/Philips 8051 or PIC microcontrollers have a UART. The UART needs to be initialized to receive the serial data from PC.

In this case, the microcontroller is connected to the motor driver ICs which control the right and left motors. After processing the image, and deciding the motion of the robot, transmit codeword for left, right, forward and backward to the microcontroller through the serial port (say 1-Left, 2-Right, 3-Forward, 4-Backward).

MATLAB code for accessing the serial port is as follows:

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

To send data through the serial port, the available commands

```
>> fwrite (ser,1); % for left motion
>> fwrite (ser,2); % for right motion
```

You can close the port in case there are other applications using this port using the *fclose* command.

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

Microcontroller has an output port whose pins can be used to control the driver IC. Thus, microcontroller interprets the serial data from the PC and suitably controls the motors through the output pins and the motor driver or H-bridge.

4 Appendix

4.1 Color Spaces

For storing the information content of an image, various formats or color spaces can be used. RGB (Red, Green, Blue) is one of the color spaces. In fact in TFT LCD screens, the color filter substrate contains these three primary color filters.

Other color spaces are YCbCr and HSV (Hue, saturation and color)

YCbCr format

In this format, luminance (light intensity) information is stored as a single component (Y), and chrominance information is stored as two color-difference components (Cb and Cr). Cb represents the difference between the blue component and a reference value. Cr represents the difference between the red component and a reference value. (YUV, another color space widely used for digital video, is very similar to YCbCr but not identical.)
You can convert the image from RGB format to YCbCr format.

>>imrgb=imread('peppers.png');
>>imycbcr=rgb2ycbcr(imrgb);
>> figure,imshow(imrgb),figure,imshow(imycbcr)

HSV format

The following figure illustrates the HSV color space.
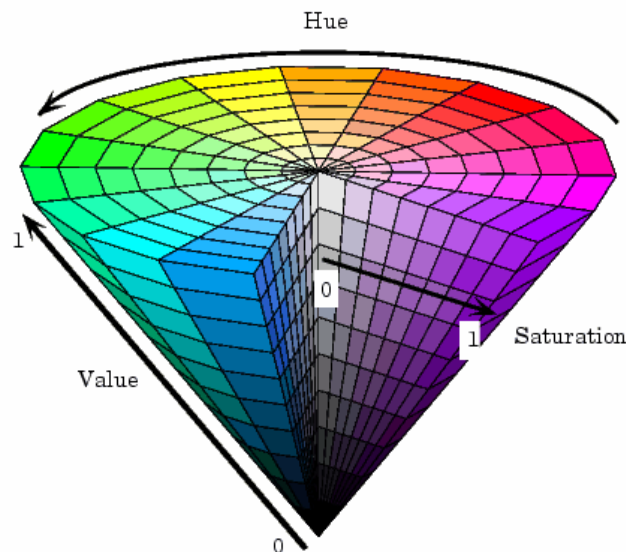
As hue varies from 0 to 1.0, the corresponding colors vary from red through yellow, green, cyan, blue, magenta, and back to red, so that there are actually red values both at 0 and 1.0. As saturation varies from 0 to 1.0, the corresponding colors (hues) vary from unsaturated (shades of gray) to fully saturated (no white component). As value, or brightness, varies from 0 to 1.0, the corresponding colors become increasingly brighter.

Again in this case, an image can be converted between the RGB and HSV formats.

```
>> imrgb = imread('peppers.png');
>> imhsv= rgb2hsv(imrgb);
>> figure,imshow(imrgb),figure,imshow(imhsv);
```

5 References

Rich documentation of MATLAB is available in the MATLAB Full Product Family Help. In this documentation, some of the illustrations and explanations have been borrowed directly from the MATLAB Help section.

Wherever images have been borrowed from websites, the source of the image has been acknowledged.

Resource Material

For a good coverage of the concepts in image processing, following books have been used by the writers of this document.

Books
1. Digital Image Processing Using MATLAB by Gonzalez and Woods

Websites
Some of the websites which were helpful in preparing this document are listed below:

1. www.roborealm.com/machine_vision.php
2. www.roborealm.com/tutorial/line_following/slide010.php
3. www.roborealm.com/links/vision_software.php
4. www.roborealm.com/tutorial/color_object_tracking_2/slide010.php
5. www.societyofrobots.com/programming_computer_vision_tutorial.shtml

Lecture Notes
As an area of research, Machine Vision involves lot of mathematics involving various areas such as co-ordinate geometry, vector algebra, calculus, probability and statistics etc. This field also encompasses pattern recognition, biometrics, The following website links to lecture notes on Machine Vision available for download through OpenCourseware (OCW) at MIT's website.

http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-801Fall-2004/CourseHome/

Video Lectures
Video lectures of a course on Image Processing are available for free download from Purdue University and University of California, Berkeley.

1. www.ecn.purdue.edu/VISE/visevideo/ee637S05/
2. webcast.berkeley.edu/course_details.php?seriesid=1906978282