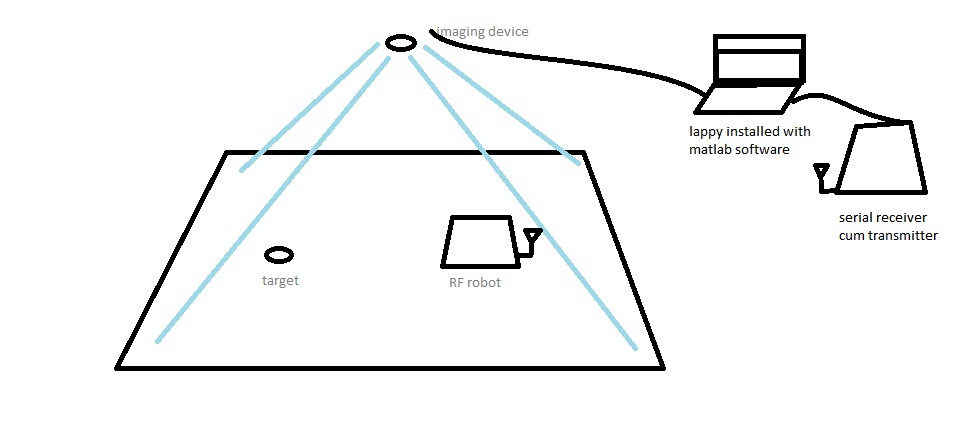
**Ball tracking RF robot using Matlab**

**Abstract:**

Artificial intelligence expects the machine to do something pre-defined without any human intervention. This project aims to implement the same using image processing combined with robotics. The bulk processing like image capture, image processing, move calculation, etc. is performed at the remote machine installed with Matlab. Only the essential commands of move are transmitted to the robot using serial port. The serial port itself is attached to RF transmitter unit. The RF transmitter unit converts the RS-232 signal to 5v level, processes it and then finally transmits it over air through HT12E encoder attached to RF antenna. The received signal is processed by the microcontroller to scan for vital commands (i.e. left, right, forward and backward). Correspondingly the vehicle moves in those directions to get closer to the target. A threshold distance is specified. Upon reaching that distance the robot automatically stops. This simple setup shows the power of robotics to take decisions automatically and respond accordingly.



**Schematic showing the complete setup**

The implementations of the project are huge. If processed correctly the software can be extended to identify any possible object and perform some pre-defined tasks based on the routines which might eliminate need of human operator. Thus making it a self- sustained and smart robotics systems and it can be used for search and rescue, mining, removing hazardous chemicals and it is capable of working in radiation zones. The possibilities are limitless.

**IMAGE PROCESSING BASICS**

**What is Image Processing?**

**Image processing** is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually **Image Processing** system includes treating images as two dimensional signals while applying already set signal processing methods to them.

It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

*Image processing basically includes the following three steps*.

1. Importing the image with optical scanner or by digital photography.
2. Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
3. Output is the last stage in which result can be altered image or report that is based on image analysis.

**Purpose of Image processing**

The purpose of image processing is divided into 5 groups. They are:

1.      Visualization - Observe the objects that are not visible.

2.      Image sharpening and restoration - To create a better image.

3.      Image retrieval - Seek for the image of interest.

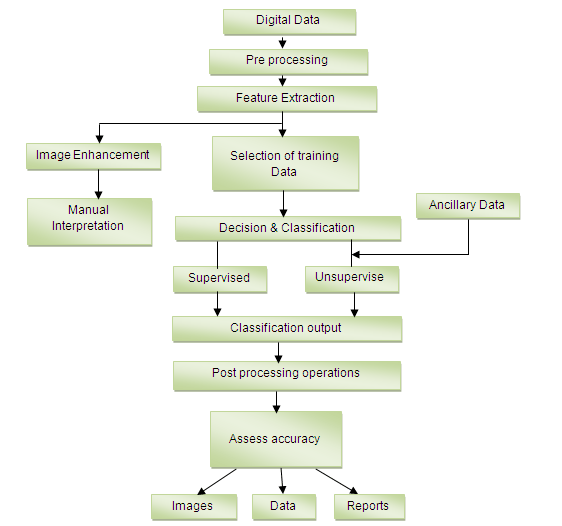
4.      Measurement of pattern – Measures various objects in an image.

5.      Image Recognition – Distinguish the objects in an image.

**Types**

The two types of **methods used for Image Processing** are **Analog and Digital** Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing.

Digital Processing techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.



**Image processing in Matlab**

Matlab has extensive image processing features which encouraged us to use it in our project. Representation of images is done in the form of matrices. In Matlab matrix calculations can be easily carried out. Almost all image processing techniques requires manipulation of image matrix either it may be segmentation or compression. Now I am going to provide an overview of Matlab functions used in this project.

The functions presented are in the order they are implemented in the project.

**videoinput**

Create video input object

**Syntax**

obj = videoinput(*adaptorname*)  
obj = videoinput(*adaptorname*,deviceID)  
obj = videoinput(*adaptorname*,deviceID,*format*)  
obj = videoinput(*adaptorname*,deviceID,*format*,P1,V1,...)

**Description**

obj = videoinput(*adaptorname*) constructs the video input object obj. A video input object represents the connection between MATLAB and a particular image acquisition device. *adaptorname* is a text string that specifies the name of the adaptor used to communicate with the device. Use the imaqhwinfo function to determine the adaptors available on your system.

obj = videoinput(*adaptorname*,deviceID) constructs a video input object obj, where deviceID is a numeric scalar value that identifies a particular device available through the specified adaptor, *adaptorname*. Use the imaqhwinfo(*adaptorname*) syntax to determine the devices available through the specified adaptor. If deviceID is not specified, the first available device ID is used. As a convenience, a device's name can be used in place of the deviceID. If multiple devices have the same name, the first available device is used.

obj = videoinput(*adaptorname*,deviceID,*format*) constructs a video input object, where *format* is a text string that specifies a particular video format supported by the device or the full path of a device configuration file (also known as a camera file).

**Examples**

Construct a video input object.

obj = videoinput('matrox', 1);

Select the source to use for acquisition.

set(obj, 'SelectedSourceName', 'input1')

View the properties for the selected video source object.

src\_obj = getselectedsource(obj);

get(src\_obj)

Preview a stream of image frames.

preview(obj);

Acquire and display a single image frame.

frame = getsnapshot(obj);

image(frame);

Remove video input object from memory.

delete(obj);

**getselectedsource**

Return currently selected video source object

**Syntax**

src = getselectedsource(obj)

**Description**

src = getselectedsource(obj) searches all the video source objects associated with the video input object obj and returns the video source object, src, that has the Selected property value set to 'on'.

To select a source for acquisition, use the SelectedSourceName property of the video input object.

obj must be a 1-by-1 video input object.

**imaqmem**

Limit memory or display memory usage for Image Acquisition Toolbox software

**Syntax**

mem = imaqmem  
imaqmem(*field*)  
imaqmem(limit)

**Description**

mem = imaqmem returns a structure containing the following fields:

| **Field** | **Description** |
| --- | --- |
| MemoryLoad | Number between 0 and 100 that gives a general idea of current memory utilization |
| TotalPhys | Total number of bytes of physical memory |
| AvailPhys | Number of bytes of physical memory currently available |
| TotalPageFile | Total number of bytes that can be stored in the paging file |
| AvailPageFile | Number of bytes available in the paging file |
| TotalVirtual | Total number of bytes that can be addressed in the user mode portion of the virtual address space |
| AvailVirtual | Number of bytes of unreserved and uncommitted memory in the user mode portion of the virtual address space |
| FrameMemoryLimit | Total number of bytes image acquisition frames can occupy in memory  By default, the toolbox sets this limit to equal all available physical memory at the time the toolbox is first used or queried. |
| FrameMemoryUsed | Number of bytes currently allocated by the Image Acquisition Toolbox software |

imaqmem(*field*) returns information for the field specified by the text string *field*.

imaqmem(limit) configures the frame memory limit, in bytes, for the Image Acquisition Toolbox software. limit is used to determine the maximum amount of memory the toolbox can use for logging image frames.

|  |
| --- |
| **Note**   Configuring the frame memory limit does not remove any logged frames from the image acquisition memory buffer. To remove frames from the buffer, you can bring them into the MATLAB workspace, using the getdata function, or remove them from memory, using the flushdata function. |

**Examples**

Use imaqmem to get information about system memory.

imaqmem

ans =

MemoryLoad: 85

TotalPhys: 263766016

AvailPhys: 37306368

TotalPageFile: 643878912

AvailPageFile: 391446528

TotalVirtual: 2.1474e+009

AvailVirtual: 1.6307e+009

FrameMemoryLimit: 38313984

FrameMemoryUsed: 0

Retrieve information about a specific field returned by imaqmem.

memlimit = imaqmem('FrameMemoryLimit')

memlimit =

38313984

Specify the amount of memory available for the toolbox to log image frames (FrameMemoryLimit).

imaqmem(30000000)

ans =

MemoryLoad: 85

TotalPhys: 263766016

AvailPhys: 37634048

TotalPageFile: 643878912

AvailPageFile: 391479296

TotalVirtual: 2.1474e+009

AvailVirtual: 1.6307e+009

FrameMemoryLimit: 30000000

FrameMemoryUsed: 0

# triggerconfig

Configure video input object trigger properties

## Syntax

triggerconfig(obj,*type*)  
triggerconfig(obj,*type*,*condition*)  
triggerconfig(obj,*type*,*condition*,*source*)  
config = triggerconfig(obj)  
triggerconfig(obj,config)

## Description

triggerconfig(obj,*type*) configures the value of the TriggerType property of the video input object obj to the value specified by the text string *type*. For a list of valid TriggerType values, use triggerinfo(obj). *type* must specify a unique trigger configuration.

obj can be either a single video input object or an array of video input objects. If an error occurs, any video input objects in the array that have already been configured are returned to their original configurations.

triggerconfig(obj,*type*,*condition*) configures the values of the TriggerType and TriggerCondition properties of the video input object obj to the values specified by the text strings *type* and *condition*. For a list of valid TriggerType and TriggerCondition values, use triggerinfo(obj). *type* and *condition* must specify a unique trigger configuration.

triggerconfig(obj,*type*,*condition*,*source*) configures the values of the TriggerType, TriggerCondition, and TriggerSource properties of the video input object obj to the values specified by the text strings *type*, *condition*, and *source*, respectively. For a list of valid TriggerType, TriggerCondition, and TriggerSource values, use triggerinfo(obj).

config = triggerconfig(obj) returns a MATLAB structure config containing the object's current trigger configuration. obj must be a 1-by-1 video input object. The field names of config are TriggerType, TriggerCondition, and TriggerSource. Each field contains the current value of the object's property.

triggerconfig(obj,config) configures the TriggerType, TriggerCondition, and TriggerSource property values for video input object obj using config, a MATLAB structure with field names TriggerType, TriggerCondition, and TriggerSource, each containing the desired property value.

## Examples

### Example 1

Construct a video input object.

vid = videoinput('winvideo', 1);

Configure trigger properties for the object.

triggerconfig(vid, 'manual')

Trigger the acquisition.

start(obj)

trigger(obj)

Remove video input object from memory.

delete(vid);

### Example 2

This example uses a structure returned from triggerinfo to configure trigger parameters.

Create a video input object.

vid = videoinput('winvideo', 1);

Use triggerinfo to get all valid configurations for the trigger properties for the object.

config = triggerinfo(vid);

Pass one of the configurations to the triggerconfig function.

triggerconfig(vid,config(2));

Remove video input object from memory.

delete(vid);

# preview

Preview of live video data

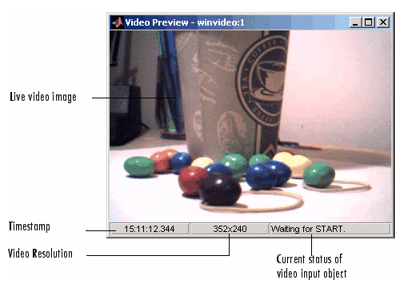
## Syntax

preview(obj)  
preview(obj,himage)  
himage = preview(...)

## Description

preview(obj) creates a Video Preview window that displays live video data for video input object obj. The window also displays the timestamp and video resolution of each frame, and the current status of obj. The Video Preview window displays the video data at 100% magnification (one screen pixel represents one image pixel). The size of the preview image is determined by the value of the video input object ROIPosition property.

**Components of a Video Preview Window**



The Video Preview window remains active until it is either stopped using stoppreview or closed using closepreview. If you delete the object, by calling delete(obj), the Video Preview window stops previewing and closes automatically.

preview(obj,himage) displays live video data for video input object obj in the image object specified by the handle himage. preview scales the image data to fill the entire area of the image object but does not modify the values of any image object properties. Use this syntax to preview video data in a custom GUI of your own design (see Examples).

himage = preview(...) returns himage, a handle to the image object containing the previewed data. To obtain a handle to the figure window containing the image object, use the ancestor function. For more information about using image objects, see image. See the Custom Update Function section for more information about the image object returned.

**start**

Obtain exclusive use of image acquisition device

**Syntax**

start(obj)

**Description**

start(obj) obtains exclusive use of the image acquisition device associated with the video input object obj and locks the device's configuration. Starting an object is a necessary first step to acquire image data, but it does not control when data is logged.

obj can either be a 1-by-1 video input object or an array of video input objects.

Data logging is controlled with the TriggerType property.

| **Trigger Type** | **Logging Behavior** |
| --- | --- |
| 'hardware' | Data logging occurs when the condition specified in the object's TriggerCondition property is met via the TriggerSource. |
| 'immediate' | Data logging occurs immediately. |
| 'manual' | Data logging occurs when the trigger function is called. |

Use the triggerconfig function to configure the object's trigger settings.

When an acquisition is started, obj performs the following operations:

1. Transfers the object's configuration to the associated hardware.
2. Executes the object's StartFcn callback.
3. Sets the object's Running property to 'On'.

If the object's StartFcn errors, the hardware is never started and the object's Running property remains 'Off'.

The start event is recorded in the object's EventLog property.

An image acquisition object stops running when one of the following conditions is met:

* The stop function is issued.
* The requested number of frames is acquired. This occurs when

FramesAcquired = FramesPerTrigger \* (TriggerRepeat + 1)

where FramesAcquired, FramesPerTrigger, and TriggerRepeat are properties of the video input object.

* A run-time error occurs.
* The object's Timeout value is reached.

**Examples**

The start function can be called by a video input object's event callback.

obj.StopFcn = {'start'};

**trigger**

Manually execute trigger for analog input or output object

**Syntax**

trigger(obj)

**Arguments**

|  |  |
| --- | --- |
| obj | An analog input or analog output object or an array of these device objects. |

**Description**

trigger(obj) manually executes a trigger.

**Remarks**

After trigger is issued,

* The absolute time of the trigger event is recorded by the InitialTriggerTime property.
* The Logging property or Sending property is set to On.
* The callback function specified by TriggerFcn is executed.
* The trigger event is recorded in the EventLog property.

You can issue trigger only if TriggerType is set to Manual, Running is On, and Logging is Off.

# getdata

Extract analog input data, time, and event information from data acquisition engine

## Syntax

data = getdata(obj)

data = getdata(obj,samples)

data = getdata(obj,samples,'*type*')

[data,time] = getdata(...)

[data,time,abstime] = getdata(...)

[data,time,abstime,events] = getdata(...)

[data,...] = getdata(obj, 'P1', V1, 'P2', V2,...)

## Arguments

|  |  |
| --- | --- |
| obj | An analog input object. |
| samples | The number of samples to extract. If samples is not specified, the number of samples extracted is given by the SamplesPerTrigger property. |
| '*type*' | Specifies the format of the extracted data as double (the default) or as native. |
| data | An m-by-n array, where m is the number of samples extracted and n is the number of channels contained by obj. |
| time | An m-by-1 array of relative time values in seconds, where m is the number of samples extracted. time = 0 is defined as the point at which data logging begins, i.e., when the Logging property of obj is set to On. Measurement of time, with respect to 0, continues until the acquisition is stopped, i.e., when the Logging property of obj is set to Off. |
| abstime | The absolute time of the first trigger returned as a clock vector. This value is identical to the value stored by the InitialTriggerTime property. |
| events | A structure containing a list of events that occurred during the time period the samples were extracted. |

## Description

data = getdata(obj) extracts the number of samples specified by the [SamplesPerTrigger](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/daq/help.jar%21/samplespertrigger.html) property for each channel contained by obj. data is an m-by-n array, where m is the number of samples extracted and n is the number of channels.

data = getdata(obj,samples) extracts the number of samples specified by samples for each channel contained by obj.

data = getdata(obj,samples,'*type*') extracts the number of samples specified by samples in the format specified by *type* for each channel contained by obj.

[data,time] = getdata(...) returns data as sample-time pairs. time is an m-by-1 array of relative time values, where m is the number of samples returned in data. Each element of time indicates the relative time, in seconds, of the corresponding sample in data, measured with respect to the first sample logged by the engine.

[data,time,abstime] = getdata(...) extracts data as sample-time pairs and returns the absolute time of the trigger. The absolute time is returned as a clock vector and is identical to the value stored by the [InitialTriggerTime](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/daq/help.jar%21/initialtriggertime.html) property.

[data,time,abstime,events] = getdata(...) extracts data as sample-time pairs, returns the absolute time of the trigger, and returns a structure containing a list of events that occurred during the time period the samples were extracted. The events returned are a sub set of those stored by the [EventLog](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/daq/help.jar%21/eventlog.html) property.

[data,...] = getdata(obj, 'P1', V1, 'P2', V2,...) specifies the number of samples to be returned, the format of the data matrix, and whether to return a tscollection object.

The following table shows a summary of properties.

| **Property** | **Description** |
| --- | --- |
| Samples | Specify the number of samples to return. |
| DataFormat | Specify the data format as double (default) or native. |
| OutputFormat | Specify the output format as matrix (default) or tscollection. |

|  |
| --- |
| **Note**   When the [ClockSource](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/daq/help.jar%21/clocksource.html) property for this function is set to one of the External options, the timing will be controlled externally and the values returned in the time variable will not accurately reflect the actual relative time of each sample. It is however an approximation based on the [SampleRate](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/daq/help.jar%21/samplerate.html) you have configured. |

**imshow**

Display image

**Syntax**

imshow(I)  
imshow(I,[low high])  
imshow(RGB)  
imshow(BW)  
imshow(X,map)  
imshow(filename)  
himage = imshow(...)  
imshow(..., param1, val1, param2, val2,...)

**Description**

imshow(I) displays the grayscale image I.

imshow(I,[low high]) displays the grayscale image I, specifying the display range for I in [low high]. The value low (and any value less than low) displays as black; the value high (and any value greater than high) displays as white. Values in between are displayed as intermediate shades of gray, using the default number of gray levels. If you use an empty matrix ([]) for [low high], imshow uses [min(I(:)) max(I(:))]; that is, the minimum value in I is displayed as black, and the maximum value is displayed as white.

imshow(RGB) displays the truecolor image RGB.

imshow(BW) displays the binary image BW. imshow displays pixels with the value 0 (zero) as black and pixels with the value 1 as white.

imshow(X,map) displays the indexed image X with the colormap map. A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0.

imshow(filename) displays the image stored in the graphics file filename. The file must contain an image that can be read by imread or dicomread. imshow calls imread or dicomread to read the image from the file, but does not store the image data in the MATLAB workspace. If the file contains multiple images, imshow displays the first image in the file. The file must be in the current directory or on the MATLAB path.

himage = imshow(...) returns the handle to the image object created by imshow.

imshow(..., param1, val1, param2, val2,...) displays the image, specifying parameters and corresponding values that control various aspects of the image display. The following table lists all imshow parameters in alphabetical order. Parameter names can be abbreviated, and case does not matter.

**Examples**

Display an image from a file.

imshow('board.tif')

Display an indexed image.

[X,map] = imread('trees.tif');

imshow(X,map)

Display a grayscale image.

I = imread('cameraman.tif');

imshow(I)

Display the same grayscale image, adjusting the display range.

h = imshow(I,[0 80]);

# subplot

## Syntax

h = subplot(m,n,p) or subplot(mnp)  
subplot(m,n,p,'replace')  
subplot(m,n,P)  
subplot(h)   
subplot('Position',[left bottom width height])  
subplot(..., prop1, value1, prop2, value2, ...)  
h = subplot(...)

## Description

subplot divides the current figure into rectangular panes that are numbered rowwise. Each pane contains an axes object which you can manipulate using [Axes Properties](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/axes_props.html). Subsequent plots are output to the current pane.

h = subplot(m,n,p) or subplot(mnp) breaks the figure window into an m-by-n matrix of small axes, selects the pth axes object for the current plot, and returns the axes handle. The axes are counted along the top row of the figure window, then the second row, etc. For example,

subplot(2,1,1), plot(income)

subplot(2,1,2), plot(outgo)

plots income on the top half of the window and outgo on the bottom half. If the CurrentAxes is nested in a uipanel, the panel is used as the parent for the subplot instead of the current figure. The new axes object becomes the current axes.

**rgb2gray**

Convert RGB image or colormap to grayscale

**Syntax**

I = rgb2gray(RGB)  
newmap = rgb2gray(map)

**Description**

I = rgb2gray(RGB) converts the truecolor image RGB to the grayscale intensity image I. rgb2gray converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

newmap = rgb2gray(map) returns a grayscale colormap equivalent to map.

**Examples**

Convert an RGB image to a grayscale image.

I = imread('board.tif');

J = rgb2gray(I);

figure, imshow(I), figure, imshow(J);

Convert the colormap to a grayscale colormap.

[X,map] = imread('trees.tif');

gmap = rgb2gray(map);

figure, imshow(X,map), figure, imshow(X,gmap);

**Algorithm**

rgb2gray converts RGB values to grayscale values by forming a weighted sum of the *R*, *G*, and *B* components:

0.2989 \* R + 0.5870 \* G + 0.1140 \* B

Note that these are the same weights used by the rgb2ntsc function to compute the *Y* component.

**im2bw**

Convert image to binary image, based on threshold

**Syntax**

BW = im2bw(I, level)  
BW = im2bw(X, map, level)  
BW = im2bw(RGB, level)

**Description**

BW = im2bw(I, level) converts the grayscale image I to a binary image. The output image BW replaces all pixels in the input image with luminance greater than level with the value 1 (white) and replaces all other pixels with the value 0 (black). Specify level in the range [0,1]. This range is relative to the signal levels possible for the image's class. Therefore, a level value of 0.5 is midway between black and white, regardless of class. To compute the level argument, you can use the function [graythresh](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/graythresh.html). If you do not specify level, im2bw uses the value 0.5.

BW = im2bw(X, map, level) converts the indexed image X with colormap map to a binary image.

BW = im2bw(RGB, level) converts the truecolor image RGB to a binary image.

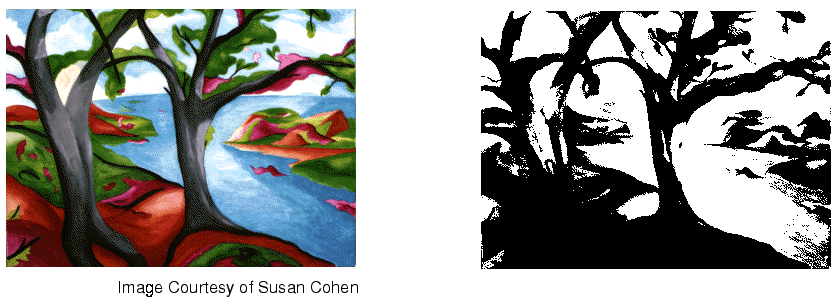
If the input image is not a grayscale image, im2bw converts the input image to grayscale, and then converts this grayscale image to binary by thresholding.

**Examples**

load trees

BW = im2bw(X,map,0.4);

imshow(X,map), figure, imshow(BW)



# bwconncomp

Find connected components in binary image

## Syntax

CC = bwconncomp(BW)  
CC = bwconncomp(BW,conn)

## Description

CC = bwconncomp(BW) returns the connected components CC found in BW. The binary image BW can have any dimension. CC is a structure with four fields.

| **Field** | **Description** |
| --- | --- |
| Connectivity | Connectivity of the connected components (objects) |
| ImageSize | Size of BW |
| NumObjects | Number of connected components (objects) in BW |
| PixelIdxList | 1-by-NumObjects cell array where the kth element in the cell array is a vector containing the linear indices of the pixels in the kth object. |

bwconncomp uses a default connectivity of 8 for two dimensions, 26 for three dimensions, and conndef(ndims(BW),'maximal') for higher dimensions.

CC = bwconncomp(BW,conn) specifies the desired connectivity for the connected components. conn can have the following scalar values.

| **Value** | **Meaning** |
| --- | --- |
| **Two-dimensional connectivities** | |
| 4 | 4-connected neighborhood |
| 8 | 8-connected neighborhood |
| **Three-dimensional connectivities** | |
| 6 | 6-connected neighborhood |
| 18 | 18-connected neighborhood |
| 26 | 26-connected neighborhood |

Connectivity can be defined in a more general way for any dimension using a 3-by-3-by- ... -by-3 matrix of 0s and 1s. conn must be symmetric about its center element. The 1-valued elements define neighborhood locations relative to conn.

## Examples

### Example 1

Calculate the centroids of the 3-D objects.

BW = cat(3, [1 1 0; 0 0 0; 1 0 0],...

[0 1 0; 0 0 0; 0 1 0],...

[0 1 1; 0 0 0; 0 0 1]);

CC = bwconncomp(BW);

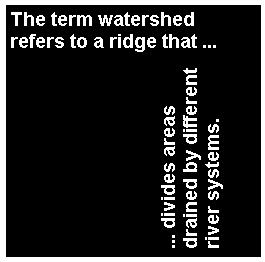
S = regionprops(CC,'Centroid');

### Example 2

Erase the largest letter from the image.

BW = imread('text.png');

imshow(BW);



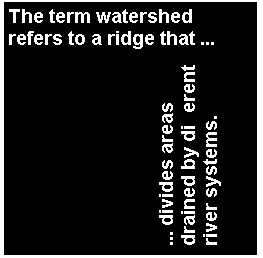
CC = bwconncomp(BW);

numPixels = cellfun(@numel,CC.PixelIdxList);

[biggest,idx] = max(numPixels);

BW(CC.PixelIdxList{idx}) = 0;

figure, imshow(BW);



# regionprops

Measure properties of image regions

## Syntax

STATS = regionprops(BW, *properties*)  
STATS = regionprops(CC, *properties*)  
STATS = regionprops(L, *properties*)  
STATS = regionprops(..., I, *properties*)

## Description

STATS = regionprops(BW, *properties*) measures a set of properties for each connected component (object) in the binary image, BW. The image BW is a logical array; it can have any dimension.

STATS = regionprops(CC, *properties*) measures a set of properties for each connected component (object) in CC, which is a structure returned by bwconncomp.

STATS = regionprops(L, *properties*) measures a set of properties for each labeled region in the label matrix L. Positive integer elements of L correspond to different regions. For example, the set of elements of L equal to 1 corresponds to region 1; the set of elements of L equal to 2 corresponds to region 2; and so on.

STATS = regionprops(..., I, *properties*) measures a set of properties for each labeled region in the image I. The first input to regionprops—either BW, CC, or L—identifies the regions in I. The sizes must match: size(I) must equal size(BW), CC.ImageSize, or size(L).

STATS is a structure array with length equal to the number of objects in BW, CC.NumObjects, or max(L(:)). The fields of the structure array denote different properties for each region, as specified by *properties*.

## Properties

*properties* can be a comma-separated list of strings, a cell array containing strings, the single string 'all', or the string 'basic'. If *properties* is the string 'all', regionprops computes all the shape measurements, listed in [Shape Measurements](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brxe6mx-1). If called with a grayscale image, regionprops also returns the pixel value measurements, listed in [Pixel Value Measurements](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brxe6mx-2). If *properties* is not specified or if it is the string 'basic', regionprops computes only the 'Area', 'Centroid', and 'BoundingBox' measurements. You can calculate the following properties on N-D inputs: 'Area', 'BoundingBox', 'Centroid', 'FilledArea', 'FilledImage', 'Image', 'PixelIdxList', 'PixelList', and 'SubarrayIdx'.

**Shape Measurements**

|  |  |  |
| --- | --- | --- |
| ['Area'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8hc) | ['EulerNumber'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8ik) | ['Orientation'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8ji) |
| ['BoundingBox'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8hf) | ['Extent'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8io) | ['Perimeter'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8jn) |
| ['Centroid'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8ln) | ['Extrema'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8iq) | ['PixelIdxList'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8ju)' |
| ['ConvexArea'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8h_) | ['FilledArea'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8i1) | ['PixelList'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8jy) |
| ['ConvexHull'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8h3) | ['FilledImage'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8i5) | ['Solidity'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8j2) |
| ['ConvexImage'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8h7) | ['Image'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8i_) | ['SubarrayIdx'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brkib5s-1) |
| ['Eccentricity'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8id) | ['MajorAxisLength'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8jb) |  |
| ['EquivDiameter'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8ih) | ['MinorAxisLength'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#bqkf8jf) |  |

**Pixel Value Measurements**

|  |  |  |
| --- | --- | --- |
| ['MaxIntensity'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brjjqtx-1) | ['MinIntensity'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brjjqup-1) | ['WeightedCentroid'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brjjqwg-1) |
| ['MeanIntensity'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brjjquo-1) | ['PixelValues'](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/regionprops.html#brjjqvz-1) |  |

## Definitions

'Area' — Scalar; the actual number of pixels in the region. (This value might differ slightly from the value returned by bwarea, which weights different patterns of pixels differently.)

'BoundingBox' — The smallest rectangle containing the region, a 1-by-Q \*2 vector, where Q is the number of image dimensions: ndims(L), ndims(BW), or numel(CC.ImageSize). BoundingBox is [ul\_corner width], where:

|  |  |
| --- | --- |
| ul\_corner | is in the form [x y z ...] and specifies the upper-left corner of the bounding box |
| width | is in the form [x\_width y\_width ...] and specifies the width of the bounding box along each dimension |

'Centroid' – 1-by-Q vector that specifies the center of mass of the region. Note that the first element of Centroid is the horizontal coordinate (or x-coordinate) of the center of mass, and the second element is the vertical coordinate (or y-coordinate). All other elements of Centroid are in order of dimension.

This figure illustrates the centroid and bounding box. The region consists of the white pixels; the green box is the bounding box, and the red dot is the centroid.



'ConvexHull' — p-by-2 matrix that specifies the smallest convex polygon that can contain the region. Each row of the matrix contains the x- and y-coordinates of one vertex of the polygon. This property is supported only for 2-D input label matrices.

'ConvexImage' — Binary image (logical) that specifies the convex hull, with all pixels within the hull filled in (i.e., set to on). (For pixels that the boundary of the hull passes through, regionprops uses the same logic as roipoly to determine whether the pixel is inside or outside the hull.) The image is the size of the bounding box of the region. This property is supported only for 2-D input label matrices.

'ConvexArea' — Scalar that specifies the number of pixels in 'ConvexImage'. This property is supported only for 2-D input label matrices.

'Eccentricity' — Scalar that specifies the eccentricity of the ellipse that has the same second-moments as the region. The eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length. The value is between 0 and 1. (0 and 1 are degenerate cases; an ellipse whose eccentricity is 0 is actually a circle, while an ellipse whose eccentricity is 1 is a line segment.) This property is supported only for 2-D input label matrices.

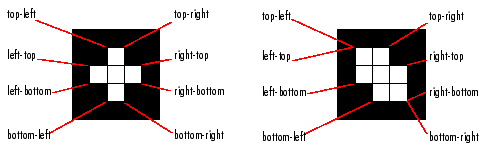
'EquivDiameter' — Scalar that specifies the diameter of a circle with the same area as the region. Computed as sqrt(4\*Area/pi). This property is supported only for 2-D input label matrices.

'EulerNumber' — Scalar that specifies the number of objects in the region minus the number of holes in those objects. This property is supported only for 2-D input label matrices. regionprops uses 8-connectivity to compute the EulerNumber measurement. To learn more about connectivity, see [Pixel Connectivity](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/toolbox/images/help.jar%21/f18-16264.html#f18-12600).

'Extent' — Scalar that specifies the ratio of pixels in the region to pixels in the total bounding box. Computed as the Area divided by the area of the bounding box. This property is supported only for 2-D input label matrices.

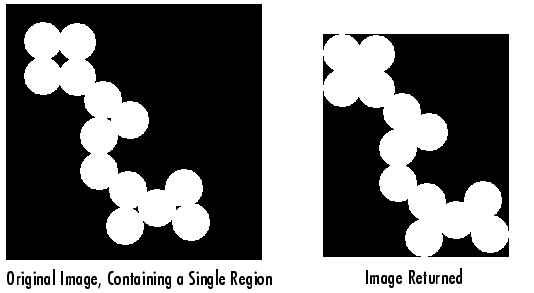
'Extrema' — 8-by-2 matrix that specifies the extrema points in the region. Each row of the matrix contains the x- and y-coordinates of one of the points. The format of the vector is [top-left top-right right-top right-bottom bottom-right bottom-left left-bottom left-top]. This property is supported only for 2-D input label matrices.

This figure illustrates the extrema of two different regions. In the region on the left, each extrema point is distinct. In the region on the right, certain extrema points (e.g., top-left and left-top) are identical.



'FilledArea' — Scalar specifying the number of on pixels in FilledImage.

'FilledImage' — Binary image (logical) of the same size as the bounding box of the region. The on pixels correspond to the region, with all holes filled in.



'Image' — Binary image (logical) of the same size as the bounding box of the region; the on pixels correspond to the region, and all other pixels are off.

'MajorAxisLength' — Scalar specifying the length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region. This property is supported only for 2-D input label matrices.

'MaxIntensity' — Scalar specifying the value of the pixel with the greatest intensity in the region.

'MeanIntensity' — Scalar specifying the mean of all the intensity values in the region.

'MinIntensity' — Scalar specifying the value of the pixel with the lowest intensity in the region.

'MinorAxisLength' — Scalar; the length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region. This property is supported only for 2-D input label matrices.

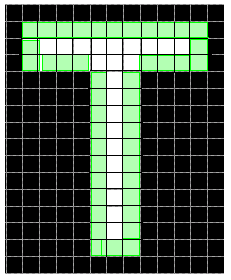
'Orientation' — Scalar; the angle (in degrees ranging from -90 to 90 degrees) between the x-axis and the major axis of the ellipse that has the same second-moments as the region. This property is supported only for 2-D input label matrices.

This figure illustrates the axes and orientation of the ellipse. The left side of the figure shows an image region and its corresponding ellipse. The right side shows the same ellipse, with features indicated graphically:

* The solid blue lines are the axes.
* The red dots are the foci.
* The orientation is the angle between the horizontal dotted line and the major axis.



'Perimeter' — Scalar; the distance around the boundary of the region. regionprops computes the perimeter by calculating the distance between each adjoining pair of pixels around the border of the region. If the image contains discontiguous regions, regionprops returns unexpected results. The following figure shows the pixels included in the perimeter calculation for this object.



'PixelIdxList' — p-element vector containing the linear indices of the pixels in the region.

'PixelList' — p-by-Q matrix specifying the locations of pixels in the region. Each row of the matrix has the form [x y z ...] and specifies the coordinates of one pixel in the region.

'PixelValues' — *p*-by-1 vector, where *p* is the number of pixels in the region. Each element in the vector contains the value of a pixel in the region.

'Solidity' — Scalar specifying the proportion of the pixels in the convex hull that are also in the region. Computed as Area/ConvexArea. This property is supported only for 2-D input label matrices.

'SubarrayIdx' — Cell-array containing indices such that L(idx{:}) extracts the elements of L inside the object bounding box.

'WeightedCentroid' — *p*-by-Q vector of coordinates specifying the center of the region based on location and intensity value. The first element of WeightedCentroid is the horizontal coordinate (or x-coordinate) of the weighted centroid. The second element is the vertical coordinate (or y-coordinate). All other elements of WeightedCentroid are in order of dimension.

## Example

Label the connected pixel components in the text.png image, compute their centroids, and superimpose the centroid locations on the image:

BW = imread('text.png');

s = regionprops(BW, 'centroid');

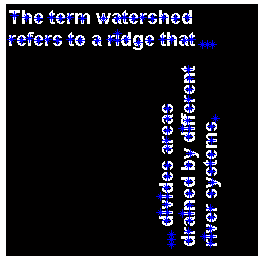
centroids = cat(1, s.Centroid);

imshow(BW)

hold on

plot(centroids(:,1), centroids(:,2), 'b\*')

hold off



**round**

Round to nearest integer

**Syntax**

Y = round(X)

**Description**

Y = round(X) rounds the elements of X to the nearest integers. For complex X, the imaginary and real parts are rounded independently.

**Examples**

a = [-1.9, -0.2, 3.4, 5.6, 7.0, 2.4+3.6i]

a =

Columns 1 through 4

-1.9000 -0.2000 3.4000 5.6000

Columns 5 through 6

7.0000 2.4000 + 3.6000i

round(a)

ans =

Columns 1 through 4

-2.0000 0 3.0000 6.0000

Columns 5 through 6

7.0000 2.0000 + 4.0000i

**atan**

Inverse tangent; result in radians

**Syntax**

Y = atan(X)

**Description**

Y = atan(X) returns the inverse tangent (arctangent) for each element of X. For real elements of X, atan(X) is in the range [–*π*/2, *π*/2].

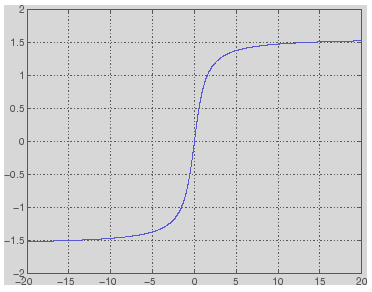
The atan function operates element-wise on arrays. The function's domains and ranges include complex values. All angles are in radians.

**Examples**

Graph the inverse tangent function over the domain –20 ≤ *x* ≤ 20.

x = -20:0.01:20;

plot(x,atan(x)), grid on



**stoppreview**

Stop previewing video data

**Syntax**

stoppreview(obj)

**Description**

stoppreview(obj) stops the previewing of video data from image acquisition object obj.

To restart previewing, call preview again.

**Examples**

Create a video input object and open a Video Preview window.

vid = videoinput('winvideo',1);

preview(vid)

Stop previewing video data.

stoppreview(vid);

Restart previewing.

preview(vid)

**stop**

Stop video input object

**Syntax**

stop(obj)

**Description**

stop(obj) halts an acquisition associated with the video input object obj. obj can be either a single video input object or an array of video input objects.

The stop function

* Sets the object's Running property to 'Off'
* Sets the object's Logging property to 'Off', if needed
* Executes the object's StopFcn callback

An image acquisition object can also stop running under one of the following conditions:

* The requested number of frames is acquired. This occurs when
* FramesAcquired = FramesPerTrigger \* (TriggerRepeat + 1)

where FramesAcquired, FramesPerTrigger, and TriggerRepeat are properties of the video input object.

* A run-time error occurs.
* The object's Timeout value is reached.

The stop event is recorded in the object's EventLog property.

**Examples**

The stop function can be called by a video input object's event callback.

obj.TimerFcn = {'stop'};

# clear

Remove items from workspace, freeing up system memory

## Syntax

clear  
clear name  
clear name1 name2 name3 ...  
clear global name  
clear -regexp expr1 expr2 ...  
clear global -regexp expr1 expr2 ...  
clear *keyword*  
clear('name1','name2','name3',...)

## Description

clear removes all variables from the workspace, releasing them from system memory.

clear name removes just the code file or MEX-file function or variable name from your base workspace. If called from a function, clear name removes name from both the function workspace and in your base workspace. You can use wildcards (\*) to remove items selectively. For example, clear my\* removes any variables whose names begin with the string my. Clearing removes debugging breakpoints in code files and reinitializes persistent variables. If name is global, clear removes it from the current workspace, but it remains accessible to any functions declaring it global. If name has been locked by mlock, it remains in memory.

# INTRODUCTION TO SERIAL COMMUNICATIONS

A parallel port sends and receives data eight bits at a time over 8 separate wires. This allows data to be transferred very quickly; however, the cable required is more bulky because of the number of individual wires it must contain. Parallel ports are typically used to connect a PC to a printer and are rarely used for much else. A serial port sends and receives data one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. In fact, two-way (full duplex) communications is possible with only three separate wires - one to send, one to receive, and a common signal ground wire.

* [Bi-directional Communications](http://www.taltech.com/support/entry/serial_intro#bidir)
* [Communicating by Bits](http://www.taltech.com/support/entry/serial_intro#commbits)
* [The Parity Bit](http://www.taltech.com/support/entry/serial_intro#parity)
* [RS-232C](http://www.taltech.com/support/entry/serial_intro#232c)
* [DCE and DTE Devices](http://www.taltech.com/support/entry/serial_intro#dce)
* [9 to 25 Pin Adapters](http://www.taltech.com/support/entry/serial_intro#9to5)
* [Baud vs. Bits per Second](http://www.taltech.com/support/entry/serial_intro#baud)
* [Cables, Null Modems, and Gender Changers](http://www.taltech.com/support/entry/serial_intro#cablenulls)
* [Cables Lengths](http://www.taltech.com/support/entry/serial_intro#length)
* [Gender Changers](http://www.taltech.com/support/entry/serial_intro#gender)
* [Null Modem Cables and Null Modem Adaptors](http://www.taltech.com/support/entry/serial_intro#null)
* [Synchronous and Asynchronous Communications](http://www.taltech.com/support/entry/serial_intro#synch)

##### **Bi-Directional Communications**

The serial port on your PC is a full-duplex device meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data. Some types of serial devices support only one-way communications and therefore use only two wires in the cable - the transmit line and the signal ground.

##### **Communicating by Bits**

Once the start bit has been sent, the transmitter sends the actual data bits. There may either be 5, 6, 7, or 8 data bits, depending on the number you have selected. Both receiver and the transmitter must agree on the number of data bits, as well as the baud rate. Almost all devices transmit data using either 7 or 8 databits.

Notice that when only 7 data bits are employed, you cannot send ASCII values greater than 127. Likewise, using 5 bits limits the highest possible value to 31. After the data has been transmitted, a stop bit is sent. A stop bit has a value of 1 - or a mark state - and it can be detected correctly even if the previous data bit also had a value of 1. This is accomplished by the stop bit's duration. Stop bits can be 1, 1.5, or 2 bit periods in length.

##### **The Parity Bit**

Besides the synchronization provided by the use of start and stop bits, an additional bit called a parity bit may optionally be transmitted along with the data. A parity bit affords a small amount of error checking, to help detect data corruption that might occur during transmission. You can choose either even parity, odd parity, mark parity, space parity or none at all. When even or odd parity is being used, the number of marks (logical 1 bits) in each data byte are counted, and a single bit is transmitted following the data bits to indicate whether the number of 1 bits just sent is even or odd.

For example, when even parity is chosen, the parity bit is transmitted with a value of 0 if the number of preceding marks is an even number. For the binary value of 0110 0011 the parity bit would be 0. If even parity were in effect and the binary number 1101 0110 were sent, then the parity bit would be 1. Odd parity is just the opposite, and the parity bit is 0 when the number of mark bits in the preceding word is an odd number. Parity error checking is very rudimentary. While it will tell you if there is a single bit error in the character, it doesn't show which bit was received in error. Also, if an even number of bits are in error then the parity bit would not reflect any error at all.

Mark parity means that the parity bit is always set to the mark signal condition and likewise space parity always sends the parity bit in the space signal condition. Since these two parity options serve no useful purpose whatsoever, they are almost never used.

##### **RS-232C**

RS-232 stands for Recommend Standard number 232 and C is the latest revision of the standard. The serial ports on most computers use a subset of the RS-232C standard. The full RS-232C standard specifies a 25-pin "D" connector of which 22 pins are used. Most of these pins are not needed for normal PC communications, and indeed, most new PCs are equipped with male D type connectors having only 9 pins.

##### **DCE and DTE Devices**

Two terms you should be familiar with are DTE and DCE. DTE stands for Data Terminal Equipment, and DCE stands for Data Communications Equipment. These terms are used to indicate the pin-out for the connectors on a device and the direction of the signals on the pins. Your computer is a DTE device, while most other devices are usually DCE devices.

If you have trouble keeping the two straight then replace the term "DTE device" with "your PC" and the term "DCE device" with "remote device" in the following discussion.

The RS-232 standard states that DTE devices use a 25-pin male connector, and DCE devices use a 25-pin female connector. You can therefore connect a DTE device to a DCE using a straight pin-for-pin connection. However, to connect two like devices, you must instead use a null modem cable. Null modem cables cross the transmit and receive lines in the cable, and are discussed later in this chapter. The listing below shows the connections and signal directions for both 25 and 9-pin connectors.

|  |  |
| --- | --- |
| 25 Pin Connector on a DTE device (PC connection) | |
| Male RS232 DB25 | 25 to 9 PIN |
| **Pin Number** | **Direction of signal:** |
| 1 | Protective Ground |
| 2 | Transmitted Data (TD) Outgoing Data (from a DTE to a DCE) |
| 3 | Received Data (RD) Incoming Data (from a DCE to a DTE) |
| 4 | Request To Send (RTS) Outgoing flow control signal controlled by DTE |
| 5 | Clear To Send (CTS) Incoming flow control signal controlled by DCE |
| 6 | Data Set Ready (DSR) Incoming handshaking signal controlled by DCE |
| 7 | Signal Ground Common reference voltage |
| 8 | Carrier Detect (CD) Incoming signal from a modem |
| 20 | Data Terminal Ready (DTR) Outgoing handshaking signal controlled by DTE |
| 22 | Ring Indicator (RI) Incoming signal from a modem |

|  |  |
| --- | --- |
| 9 Pin Connector on a DTE device (PC connection) | |
| Male RS232 DB9 | 9 PIN |
| **Pin Number** | **Direction of signal:** |
| 1 | Carrier Detect (CD) (from DCE) Incoming signal from a modem |
| 2 | Received Data (RD) Incoming Data from a DCE |
| 3 | Transmitted Data (TD) Outgoing Data to a DCE |
| 4 | Data Terminal Ready (DTR) Outgoing handshaking signal |
| 5 | Signal Ground Common reference voltage |
| 6 | Data Set Ready (DSR) Incoming handshaking signal |
| 7 | Request To Send (RTS) Outgoing flow control signal |
| 8 | Clear To Send (CTS) Incoming flow control signal |
| 9 | Ring Indicator (RI) (from DCE) Incoming signal from a modem |
|  |  |

The TD (transmit data) wire is the one through which data from a DTE device is transmitted to a DCE device. This name can be deceiving, because this wire is used by a DCE device to receive its data. The TD line is kept in a mark condition by the DTE device when it is idle. The RD (receive data) wire is the one on which data is received by a DTE device, and the DCE device keeps this line in a mark condition when idle.

**RTS** stands for **Request To Send**. This line and the CTS line are used when "hardware flow control" is enabled in both the DTE and DCE devices. The DTE device puts this line in a mark condition to tell the remote device that it is ready and able to receive data. If the DTE device is not able to receive data (typically because its receive buffer is almost full), it will put this line in the space condition as a signal to the DCE to stop sending data. When the DTE device is ready to receive more data (i.e. after data has been removed from its receive buffer), it will place this line back in the mark condition. The complement of the RTS wire is CTS, which stands for Clear To Send. The DCE device puts this line in a mark condition to tell the DTE device that it is ready to receive the data. Likewise, if the DCE device is unable to receive data, it will place this line in the space condition. Together, these two lines make up what is called RTS/CTS or "hardware" flow control. The Software Wedge supports this type of flow control, as well as Xon/XOff or "software" flow control. Software flow control uses special control characters transmitted from one device to another to tell the other device to stop or start sending data. With software flow control the RTS and CTS lines are not used.

**DTR** stands for **Data Terminal Ready**. Its intended function is very similar to the RTS line. DSR (Data Set Ready) is the companion to DTR in the same way that CTS is to RTS. Some serial devices use DTR and DSR as signals to simply confirm that a device is connected and is turned on. The Software Wedge sets DTR to the mark state when the serial port is opened and leaves it in that state until the port is closed. The DTR and DSR lines were originally designed to provide an alternate method of hardware handshaking. It would be pointless to use both RTS/CTS and DTR/DSR for flow control signals at the same time. Because of this, DTR and DSR are rarely used for flow control.

**CD** stands for **Carrier Detect**. Carrier Detect is used by a modem to signal that it has a made a connection with another modem, or has detected a carrier tone.

The last remaining line is **RI** or **Ring Indicator**. A modem toggles the state of this line when an incoming call rings your phone.

The Carrier Detect (CD) and the Ring Indicator (RI) lines are only available in connections to a modem. Because most modems transmit status information to a PC when either a carrier signal is detected (i.e. when a connection is made to another modem) or when the line is ringing, these two lines are rarely used.

##### 9 to 25 Pin Adapters

The following table shows the connections inside a standard 9 pin to 25 pin adapter.

|  |  |
| --- | --- |
| 9 Pin Connector | 25 Pin Connector |
| Pin 1 DCD | Pin 8 DCD |
| Pin 2 RD | Pin 3 RD |
| Pin 3 TD | Pin 2 TD |
| Pin 4 DTR | Pin 20 DTR |
| Pin 5 GND | Pin 7 GND |
| Pin 6 DSR | Pin 6 DSR |
| Pin 7 RTS | Pin 4 RTS |
| Pin 8 CTS | Pin 5 CTS |
| Pin 9 RI | Pin 22 RI |

##### 

##### **Baud vs. Bits per Second**

The baud unit is named after Jean Maurice Emile Baudot, who was an officer in the French Telegraph Service. He is credited with devising the first uniform-length 5-bit code for characters of the alphabet in the late 19th century. What baud really refers to is modulation rate or the number of times per second that a line changes state. This is not always the same as bits per second (BPS). If you connect two serial devices together using direct cables then baud and BPS are in fact the same. Thus, if you are running at 19200 BPS, then the line is also changing states 19200 times per second. But when considering modems, this isn't the case.

Because modems transfer signals over a telephone line, the baud rate is actually limited to a maximum of 2400 baud. This is a physical restriction of the lines provided by the phone company. The increased data throughput achieved with 9600 or higher baud modems is accomplished by using sophisticated phase modulation, and data compression techniques.

##### **Cables, Null Modems, and Gender Changers**

In a perfect world, all serial ports on every computer would be DTE devices with 25-pin male "D" connectors. All other devices to would be DCE devices with 25-pin female connectors. This would allow you to use a cable in which each pin on one end of the cable is connected to the same pin on the other end. Unfortunately, we don't live in a perfect world. Serial ports use both 9 and 25 pins, many devices can be configured as either DTE or DCE, and - as in the case of many data collection devices - may use completely non standard or proprietary pin-outs. Because of this lack of standardization, special cables called null modem cables, gender changers and custom made cables are often required.

##### **Cables Lengths**

The RS-232C standard imposes a cable length limit of 50 feet. You can usually ignore this "standard", since a cable can be as long as 10000 feet at baud rates up to 19200 if you use a high quality, well shielded cable. The external environment has a large effect on lengths for unshielded cables. In electrically noisy environments, even very short cables can pick up stray signals. The following chart offers some reasonable guidelines for 24 gauge wire under typical conditions. You can greatly extend the cable length by using additional devices like optical isolators and signal boosters. Optical isolators use LEDs and Photo Diodes to isolate each line in a serial cable including the signal ground. Any electrical noise affects all lines in the optically isolated cable equally - including the signal ground line. This causes the voltages on the signal lines relative to the signal ground line to reflect the true voltage of the signal and thus canceling out the effect of any noise signals.

|  |  |  |
| --- | --- | --- |
| Baud Rate | Shielded Cable Length | Unshielded Cable Length |
| 110 | 5000 | 1000 |
| 300 | 4000 | 1000 |
| 1200 | 3000 | 500 |
| 2400 | 2000 | 500 |
| 4800 | 500 | 250 |
| 9600 | 250 | 100 |

##### 

##### **Gender Changers**

A problem you may encounter is having two connectors of the same gender that must be connected.

***Note:*** The parallel port on a PC uses a 25 pin female connector which sometimes causes confusion because it looks just like a serial port except that it has the wrong gender. Both 9 and 25 pin serial ports on a PC will always have a male connector.

##### Null Modem Cables and Null Modem Adaptors

If you connect two DTE devices (or two DCE devices) using a straight RS232 cable, then the transmit line on each device will be connected to the transmit line on the other device and the receive lines will likewise be connected to each other. A Null Modem cable or Null Modem adapter simply crosses the receive and transmit lines so that transmit on one end is connected to receive on the other end and vice versa. In addition to transmit and receive, DTR & DSR, as well as RTS & CTS are also crossed in a Null modem connection.

Null modem adapter are available at most computer and office supply stores for under $5.

##### **Synchronous and Asynchronous Communications**

There are two basic types of serial communications, synchronous and asynchronous. With Synchronous communications, the two devices initially synchronize themselves to each other, and then continually send characters to stay in sync. Even when data is not really being sent, a constant flow of bits allows each device to know where the other is at any given time. That is, each character that is sent is either actual data or an idle character. Synchronous communications allows faster data transfer rates than asynchronous methods, because additional bits to mark the beginning and end of each data byte are not required. The serial ports on IBM-style PCs are asynchronous devices and therefore only support asynchronous serial communications.

Asynchronous means "no synchronization", and thus does not require sending and receiving idle characters. However, the beginning and end of each byte of data must be identified by start and stop bits. The start bit indicate when the data byte is about to begin and the stop bit signals when it ends. The requirement to send these additional two bits cause asynchronous communications to be slightly slower than synchronous however it has the advantage that the processor does not have to deal with the additional idle characters.

An asynchronous line that is idle is identified with a value of 1, (also called a mark state). By using this value to indicate that no data is currently being sent, the devices are able to distinguish between an idle state and a disconnected line. When a character is about to be transmitted, a start bit is sent. A start bit has a value of 0, (also called a space state). Thus, when the line switches from a value of 1 to a value of 0, the receiver is alerted that a data character is about to come down the line.

**SERIAL COMMUNICATION IN MATLAB**

# serial

Create serial port object

## Syntax

obj = serial('*port*')  
obj = serial('*port*','*PropertyName*',PropertyValue,...)

## Description

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

*Port* object name will depend upon the platform that the serial port is on. instrhwinfo ('serial') provides a list of available serial ports. This list is an example of serial constructors on different platforms:

| **Platform** | **Serial Port Constructor** |
| --- | --- |
| Linux and Linux 64 | serial('/dev/ttyS0'); |
| Mac OS X and Mac OS X 64 | serial('/dev/tty.KeySerial1'); |
| Solaris 64 | serial('/dev/term/a'); |
| Windows 32 and Windows 64 | serial('com1'); |

obj = serial('*port*','*PropertyName*',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.

## Remarks

When you create a serial port object, these property values are automatically configured:

* The [Type](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/type.html) property is given by serial.
* The [Name](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/name.html) property is given by concatenating Serial with the port specified in the serial function.
* The [Port](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/port.html) property is given by the port specified in the serial function.

You can specify the property names and property values using any format supported by the [set](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.set.html) function. For example, you can use property name/property value cell array pairs. Additionally, you can specify property names without regard to case, and you can make use of property name completion. For example, the following commands are all valid on a Windows platform.

s = serial('COM1','BaudRate',4800);

s = serial('COM1','baudrate',4800);

s = serial('COM1','BAUD',4800);

Refer to [Configuring Property Values](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/f92576.html#f61191) for a list of serial port object properties that you can use with serial.

Before you can communicate with the device, it must be connected to obj with the [fopen](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.fopen.html) function. A connected serial port object has a [Status](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/status.html) property value of open. An error is returned if you attempt a read or write operation while the object is not connected to the device. You can connect only one serial port object to a given serial port.

## Example

This example creates the serial port object s1 associated with the serial port COM1 on a Windows platform.

s1 = serial('COM1');

The Type, Name, and Port properties are automatically configured.

get(s1,{'Type','Name','Port'})

ans =

'serial' 'Serial-COM1' 'COM1'

To specify properties during object creation

s2 = serial('COM2','BaudRate',1200,'DataBits',7);

**fopen (serial)**

Connect serial port object to device

**Syntax**

fopen(obj)

**Description**

fopen(obj) connects the serial port object, obj to the device.

**Remarks**

Before you can perform a read or write operation, obj must be connected to the device with the fopen function. When obj is connected to the device:

* Data remaining in the input buffer or the output buffer is flushed.
* The [Status](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/status.html) property is set to open.
* The [BytesAvailable](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/bytesavailable.html), [ValuesReceived](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/valuesreceived.html), [ValuesSent](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/valuessent.html), and [BytesToOutput](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/bytestooutput.html) properties are set to 0.

An error is returned if you attempt to perform a read or write operation while obj is not connected to the device. You can connect only one serial port object to a given device.

Some properties are read-only while the serial port object is open (connected), and must be configured before using fopen. Examples include InputBufferSize and OutputBufferSize. Refer to the property reference pages to determine which properties have this constraint.

The values for some properties are verified only after obj is connected to the device. If any of these properties are incorrectly configured, then an error is returned when fopen is issued and obj is not connected to the device. Properties of this type include [BaudRate](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/baudrate.html), and are associated with device settings.

If you use the [help](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/help.html) command to display help for fopen, then you need to supply the pathname shown below.

help serial/fopen

**Example**

This example creates the serial port object s, connects s to the device using fopen, writes and reads text data, and then disconnects s from the device. This example works on a Windows platform.

s = serial('COM1');

fopen(s)

fprintf(s,'\*IDN?')

idn = fscanf(s);

fclose(s)

# fprintf (serial)

Write text to device

## Syntax

fprintf(obj,'cmd')  
fprintf(obj,'*format*','cmd')  
fprintf(obj,'cmd','*mode*')  
fprintf(obj,'*format*','cmd','*mode*')

## Description

fprintf(obj,'cmd') writes the string cmd to the device connected to the serial port object, obj. The default format is %s\n. The write operation is synchronous and blocks the command-line until execution completes.

fprintf(obj,'*format*','cmd') writes the string using the format specified by *format*.

fprintf(obj,'cmd','*mode*') writes the string with command line access specified by *mode*. *mode* specifies if cmd is written synchronously or asynchronously.

fprintf(obj,'*format*','cmd','*mode*') writes the string using the specified format. *format* is a C language conversion specification.

You need an open connection from the serial port object, obj, to the device before performing read or write operations.

Use the [fopen](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.fopen.html) function to open a connection to the device. When obj has an open connection to the device it has a [Status](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/status.html) property value of open. Refer to [Troubleshooting Common Errors](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/f62852.html#br8zt6b-1) for fprintf errors.

To understand the use of fprintf refer to [Completing a Write Operation with fprintf](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/f62852.html#f94797) and [Rules for Writing the Terminator](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/f62852.html#br8pqg6-1).

## Input Arguments

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *format* | ANSI C conversion specification includes these conversion characters.   | **Specifier** | **Description** | | --- | --- | | %c | Single character | | %d or %i | Decimal notation (signed) | | %e | Exponential notation (using lowercase e as in 3.1415e+00) | | %E | Exponential notation (using uppercase E as in 3.1415E+00) | | %f | Fixed-point notation | | %g | The more compact of %e or %f, as defined above. Insignificant zeros do not print. | | %G | Same as %g, but using uppercase E | | %o | Octal notation (unsigned) | | %s | String of characters | | %u | Decimal notation (unsigned) | | %x | Hexadecimal notation (using lowercase letters a–f) | | %X | Hexadecimal notation (using uppercase letters A–F) | |
| *mode* | Specifies whether the string cmd is written synchronously or asynchronously:   * sync: cmd is written synchronously and the command line is blocked. * async: cmd is written asynchronously and the command line is not blocked.   If *mode* is not specified, the write operation is synchronous.  If you specify asynchronous *mode*, when the write operation occurs:   * The [BytesToOutput](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/bytestooutput.html) property value continuously updates to reflect the number of bytes in the output buffer. * The MATLAB file callback function specified for the [OutputEmptyFcn](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/outputemptyfcn.html) property is executed when the output buffer is empty.   Use the [TransferStatus](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/transferstatus.html) property to determine whether an asynchronous write operation is in progress.  For more information on synchronous and asynchronous write operations, see [Controlling Access to the MATLAB Command Line](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/f62852.html#f62883). |

## Examples

Create a serial port object s and connect it to a Tektronix TDS 210 oscilloscope. Write the RS232? command with [fprintf](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.fprintf.html). RS232? instructs the scope to return serial port communications settings. This example works on a Windows platform.

s = serial('COM1');

fopen(s)

fprintf(s,'RS232?')

Specify a format for the data that does not include the terminator, or configure the terminator to empty.

s = serial('COM1');

fopen(s)

fprintf(s,'%s','RS232?')

The default format for fprintf is %s\n. Therefore, the terminator specified by the [Terminator](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/terminator.html) property is automatically written. However, in some cases you might want to suppress writing the terminator.

Specify an array of formats and commands:

s = serial('COM1');

fopen(s)

fprintf(s,['ch:%d scale:%d'],[1 20e-3],'sync');

**fclose (serial)**

Disconnect serial port object from device

**Syntax**

fclose(obj)

**Description**

fclose(obj) disconnects obj from the device, where obj is a serial port object or an array of serial port objects.

**Remarks**

If obj was successfully disconnected, then the [Status](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/status.html) property is configured to closed and the [RecordStatus](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/matlab_external/recordstatus.html) property is configured to off. You can reconnect obj to the device using the [fopen](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.fopen.html) function.

An error is returned if you issue [fclose](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.fclose.html) while data is being written asynchronously. In this case, you should abort the write operation with the [stopasync](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/stopasync.html) function, or wait for the write operation to complete.

If you use the [help](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/help.html) command to display help for fclose, then you need to supply the pathname shown below.

help serial/fclose

**Example**

This example creates the serial port object s on a Windows platform, connects s to the device, writes and reads text data, and then disconnects s from the device using fclose.

s = serial('COM1');

fopen(s)

fprintf(s, '\*IDN?')

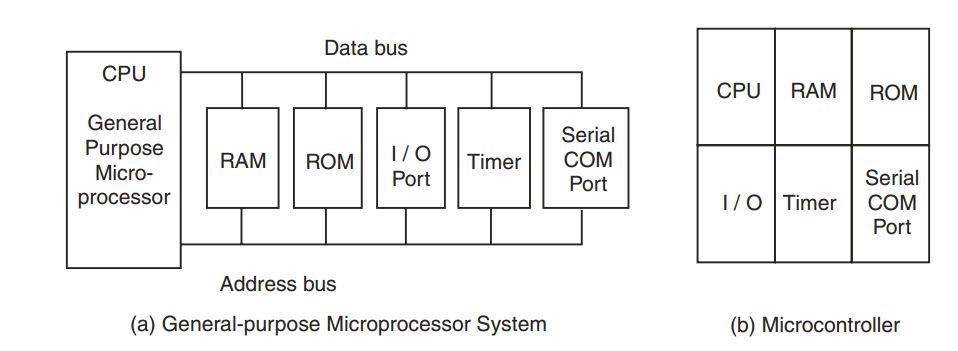
idn = fscanf(s);

fclose(s)

At this point, the device is available to be connected to a serial port object. If you no longer need s, you should remove from memory with the [delete](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.delete.html) function, and remove it from the workspace with the [clear](jar:file:///C:/Program%20Files/MATLAB/R2010a/help/techdoc/help.jar%21/ref/serial.clear.html) command.

**MICROCONTROLLERS (MCU)**

Figure shows the block diagram of a typical microcontroller, which is a true computer on a chip. The design incorporates all of the features found in micro-processor CPU, ALU, PC, SP, and registers. It also added the other features needed to make a complete computer: ROM, RAM, I/O,timer & counters,and clock circuit.



**Structure of microprocessor and microcontroller**

**DIFFERENCE BETWEEN MICROCONTROLLER & MICROPROCESSOR**

It is very clear from figure that in microprocessor we have to interface additional circuitry for providing the function of memory and ports, for example we have to interface external RAM for data storage, ROM for program storage, programmable peripheral interface (PPI) 8255 for the Input Output ports, 8253 for timers, USART for serial port. While in the microcontroller RAM, ROM, I/O ports, timers and serial communication ports are in built. Because of this it is called as “system on chip”. So in micro-controller there is no necessity of additional circuitry which is interfaced in the microprocessor because memory and input output ports are inbuilt in the microcontroller. Microcontroller gives the satisfactory performance for small applications. But for large applications the memory requirement is limited because only 64 KB memory is available for program storage. So for large applications we prefer microprocessor than microcontroller due to its high processing speed.

**8051 MICROCONTROLLER**

**Description:-**

The AT89S52 is a low-power, high-performance CMOS 8-bit microcontroller with 8Kbytes of in-system programmable Flash memory. The device is manufactured using Atmel’s high-density non-volatile memory technology and is compatible with the industry-standard 80C51 instruction set and pin out. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional non-volatile memory programmer. By combining a versatile 8-bit CPU with in-system programmable Flash on a monolithic chip, the Atmel AT89S52 is a powerful microcontroller which provides a highly-flexible and cost-effective solution to many embedded control applications. The AT89S52 provides the following standard features: 8K bytes of Flash, 256 bytes of RAM, 32 I/O lines, Watchdog timer, two data pointers, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89S52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next interrupt

or hardware reset.

**Features:-**

• 8K Bytes of In-System Programmable (ISP) Flash Memory

– Endurance: 1000 Write/Erase Cycles

• 4.0V to 5.5V Operating Range

• Fully Static Operation: 0 Hz to 33 MHz

• Three-level Program Memory Lock

• 256 x 8-bit Internal RAM

• 32 Programmable I/O Lines

• Three 16-bit Timer/Counters

• Eight Interrupt Sources

• Full Duplex UART Serial Channel

• Low-power Idle and Power-down Modes

• Interrupt Recovery from Power-down Mode

• Watchdog Timer

• Dual Data Pointer

• Power-off Flag

• Fast Programming Time

• Flexible ISP Programming (Byte and Page Mode)

**PIN CONFIGURATION OF 8051 MICROCONTROLLER**

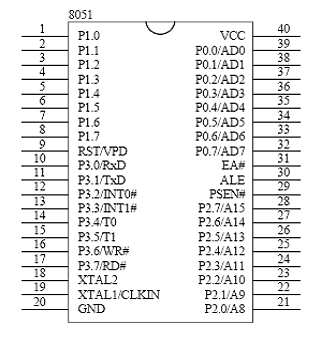
Although 8051 family members come in different packages such DIP(dual in line package),QFP(Quad flat package), and LLC(leadless chi0p carrier),they all have **40** pins that are dedicated to various functions such as I/O,RD,WR, address, data and interrupts.

**VCC**:

Pin 40 provides supply voltage to the chip. The voltage source is +5 Volts.

**GND:**

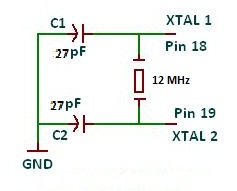
Pin 20 is the ground.



**FIG 28: PIN DIAGRAM OF THE P89C51**

**XTAL1 and XTAL2:**

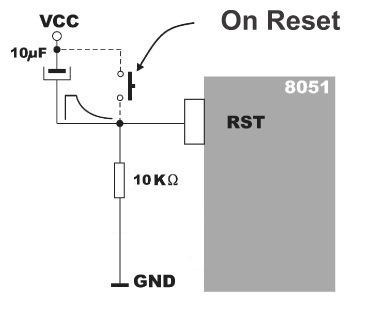
The 8051 has an on chip oscillator but requires an external clock to run it. Most often a quartz crystal oscillator is connected to inputs XTAL1 (pin 19) and XTAL2 (pin 18). The quartz crystal oscillator connected to XTAL1 and XTAL2 also needs two capacitors of 27 pf value. One side of each capacitor is connected to the ground. Speed refers to the maximum oscillator frequency connected to XTAL .When the 8051 is connected to a crystal oscillator is powered up we can observe the frequency on the XTAL2 pin using the oscilloscope.



**RST:**

Pin 9 is the RESET pin. It is an input and is active high. Upon applying a high pulse to this pin the microcontroller well reset and terminate all activities. This is often referred to as a power on reset .Activating a power on reset will cause all values the registers to be lost. It will set program counter to all 0s.

In order for the RESET input to be effective it must have a minimum duration of two machine cycles. In other words the high pulse must be high for a minimum of two machine cycles before it is allowed to go low.



**EA:**

The 8051 family members such as the 8751/52, 89C51/52 or DS89C4\*0 all come with on chip ROM to store programs. In such cases the EA pin is connected to Vcc. For family members such as the 8031 and 8032 in which there is no on chip ROM, code is stored on an external ROM and is fetched by 8031/32. Therefore for the 8031 the EA pin must be connected to GND to indicate that the code is stored externally. EA which stands for “external access” is pin number 31 in the DIP packages. It is an input pin and must be connected to either Vcc or GND. In other words it can not be unconnected.

**PSEN:**

This is an output pin. PSEN stands for “program store enable”. In an 8031 based system in which an external ROM holds the program code, this pin is connected to the OE pin of the ROM.

**ALE:** ALE stands for “address latch enable. It is an output pin and is active high. When connecting an 8031 to external memory, port 0 provides both address and data. In other words the 8031 multiplexes address and data through port 0 to save pins. The ALE pin is used for de-multiplexing the address and data by connecting to G pin of the 74LS373 chip.

**PORTS 0,1,2,3:**

All the ports upon RESET are configured as input, since P0-P3 have value FFH on them. The following is a summary of features of P0-P3.

**PORT 0:**

Port 0 is also designated as AD0-AD7 allowing it to be used for both address and data. When connecting an 8051/31 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save pins. ALE indicates if p0 has address A0-A7.in the 8051 based systems where there is no external memory connection the pins of P0 must be connected externally to 10k-ohm pull-up resistor. This is due to the fact that P0 is an open drain, unlike P1, P2 and P3. Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips. In many systems using the 8751, 89c51 or DS89c4\*0 chips we normally connect P0 to pull up resistors.

**PORT 1, PORT 2:**

In 8051 based systems with no external memory connection both P1 and P2 are used as simple I/O. however in 8031/51 based systems with external memory connections P2 must be used along with P0 to provide the 16-bit address for the external memory. P2 is also designated as A8-A15 indicating its dual function. Since an 8031/51 is capable of accessing 64k bytes of external memory it needs a path for the 16 bits of address. While P0 provides the lower 8 bits via A0-a7 it is the job P2 to provide bits A8-A15 of the address. In other words when the 8031/51 is connected to external memory P2 is used for the upper 8 bits of the 16 bit address and it cannot be used for I/O.

**PORT 3:**

Port 3 occupies a total of 8 pins 10 through 17. It can be used as input or output. P3 does not need any pull-up resistors the same as P1 and P2 did not. Although port 3 is configured as input port upon reset this is not the way it is most commonly used. Port 3 has the additional function of providing some extremely important signals such as interrupts.

**Port 3 Alternate functions:**

|  |
| --- |
| **P3 Bit Function Pin** |
| P3.0 RxD 10 |
| P3.1 TxD 11 |
| P3.2 INT0 12 |
| P3.3 INT1 13 |
| P3.4 T0 14 |
| P3.5 T1 15 |
| P3.6 WR 16 |
| P3.7 RD 17 |

**Table 1: Alternate functions of Port 3**

**Difference between RAM and ROM**

**•** RAM is used for data storage while ROM is used for program storage.

• Data of RAM can be changed during processing while data of ROM can’t

be changed during processing.

• We can take an example of calculator. If we want to perform addition of

two numbers then we type the two numbers in calculator, this is saved in

the RAM, but the Algorithms by which the calculation is performed is saved

in the ROM. Data which is given by us to calculator can be changed but the

algorithm or program by which calculation is performed can’t be changed.

**PROGRAMMING MODEL**

In programming model of 8051 we have different types of registers are available and te4hse registers are used to store temporarily data is then the information could be a byte of data to be processed or an address pointing to the data to be fetched the majority of registers is 8051 are 8-bikt registers.

**a) ACCUMULATOR (REGISTER A):**

Accumulator is a mathematical register where all the arithmetic and logical operations are done is this register and after execution of instructions the outpour data is stored in the register is bit addressable near. We can access any of the single bit of this register.

**b) B REGISTER:**

B register is same as that of accumulator of. It is also an 8 bit register and every bit of this is accessible. This is also a mathematical register B which is used mostly for multiplication and division.

**c) PSW (PROGRAM STATUS WORD) Register:**

Program status word register is an 8 bit register. It is also referred to as the flag register. Although the PSW register is 8 bits wide, only 6 bits of it are used by the 8051. The unused bits are user-definable flags. Four of the flags are called conditional flags, meaning that they indicate some conditions that result after an instruction is executed. These four are CY (carry), AC (auxiliary carry), P (parity) and OV (overflow).

|  |  |  |
| --- | --- | --- |
| CY | PSW.7 | Carry Flag |
| AC | PSW.6 | Auxiliary Carry Flag |
| F0 | PSW.5 | Available to the user for General Purpose |
| RS1 | PSW.4 | Register Bank Selector Bit 1 |
| RS0 | PSW.3 | Register Bank Selector Bit 0 |
| OV | PSW.2 | Overflow Flag |
| -- | PSW.1 | User Definable Bit |
| P | PSW.0 | Parity Flag. |

**c) SP (STACK POINTER, ADDRESS 81H):**

This is the stack pointer of the microcontroller. This SFR indicates where the next value to be taken from the stack will be read from in Internal RAM. If you push a value onto the stack, the value will be written to the address of SP + 1. That is to say, if SP holds the value 07h, a PUSH instruction will push the value onto the stack at address 08h. This SFR is modified by all instructions, which modify

The stack, such as PUSH, POP, LCALL, RET, RETI, and whenever interrupts are provoked by the microcontroller.

**d) DPL/DPH (DATA POINTER LOW/HIGH, ADDRESSES 82H/83H):**

The SFRs DPL and DPH work together to represent a 16-bit value called the Data Pointer. The data pointer is used in operations regarding external RAM and some instructions involving code memory. Since it is an unsigned two-byte integer value, it can represent values from 0000h to FFFFh (0 through 65,535 decimal).

Two instructions which are used to start and terminate program.

**• ORG →** this instruction indicate the origin of program ORG 3000H

→ means program starts from 3000H loc

→ this instruction hasn’t take any memory space. It is used to show the

starting address of program.

**• END →** this instruction show the END of program or it is used to terminate

the program.

**MICROCONTROLLER CODE**

; RObot

; Input P1.0-P1.4

; Motor P2.0 to P2.3

org 00h

mov p2,#0ffh

back:

jnb p1.0,aa0

jnb p1.1,aa1

jnb p1.2,aa2

jnb p1.3,aa3

setb p2.0

setb p2.1

setb p2.2

setb p2.3

jmp back

aa0:

clr p2.0

setb p2.1

clr p2.2

setb p2.3

jmp back

aa1:

setb p2.0

clr p2.1

setb p2.2

clr p2.3

jmp back

aa2:

clr p2.0

setb p2.1

clr p2.2

clr p2.3

jmp back

aa3:

clr p2.0

clr p2.1

clr p2.2

setb p2.3

jmp back

delay1: acall delay

acall delay

ret

delay:

mov r7,#0ffh

aa:mov r6,#0ffh

bb: djnz r6,bb

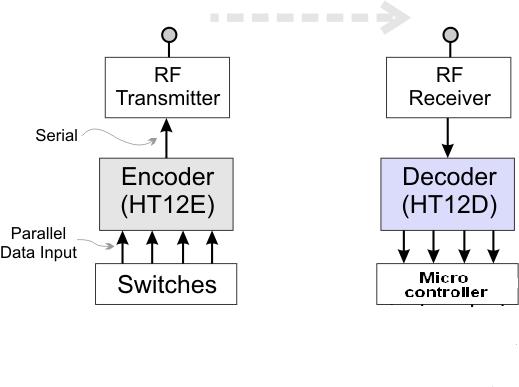
djnz r7,aa

ret

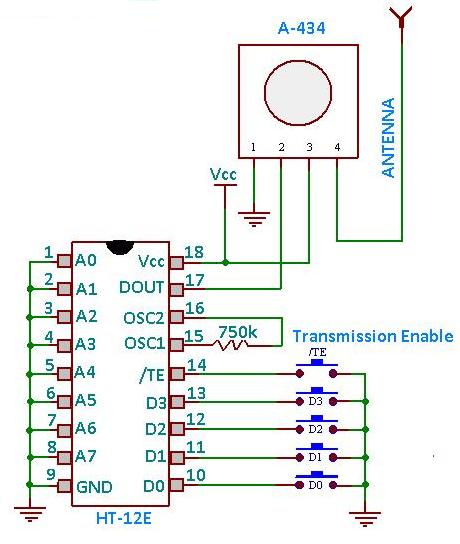
end

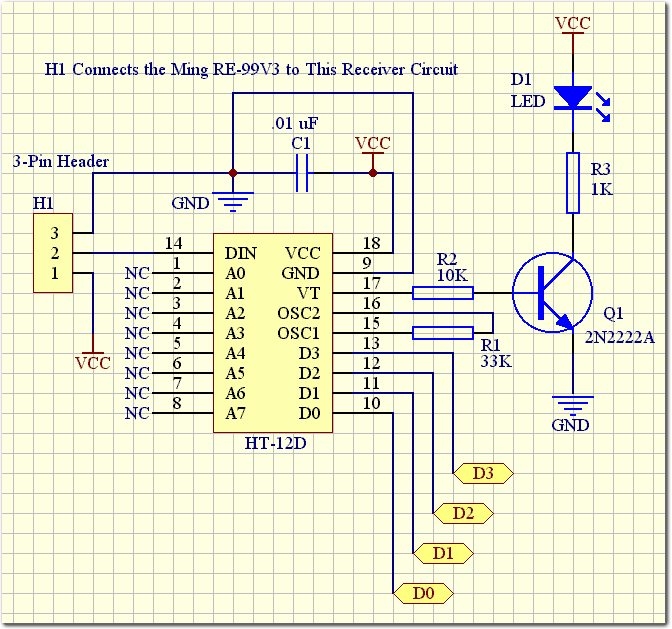
**RF MODULE**

RF module comes in the transmitter and receiver pair. Varity of RF modules are available in the market. These RF modules operate at some particular frequency like 433 MHz and 315 MHz. HT 12 D and HT 12 E ICs are used with the RF modules to encode and decode the data and channels. RF modules send data serially by air as shown below:-



HT 12 E is an encoder IC which is used to encode the channel and data. It’s a 18 pin IC. 4 pins are used for the data and 8 pins are used for addressing. 17th pin is the output pin, which is further connected to the RF module of some particular frequency. Pin diagram and circuit is shown below : -



HT 12 D is an decoder IC which is used to decode the channel and data. It’s a 18 pin IC. 4 pins are used for the data and 8 pins are used for addressing. 14th pin is the input pin, which is connected to the RF receiver. Pin diagram and circuit is shown below : -

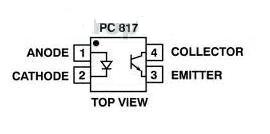
**ISOLATORS**

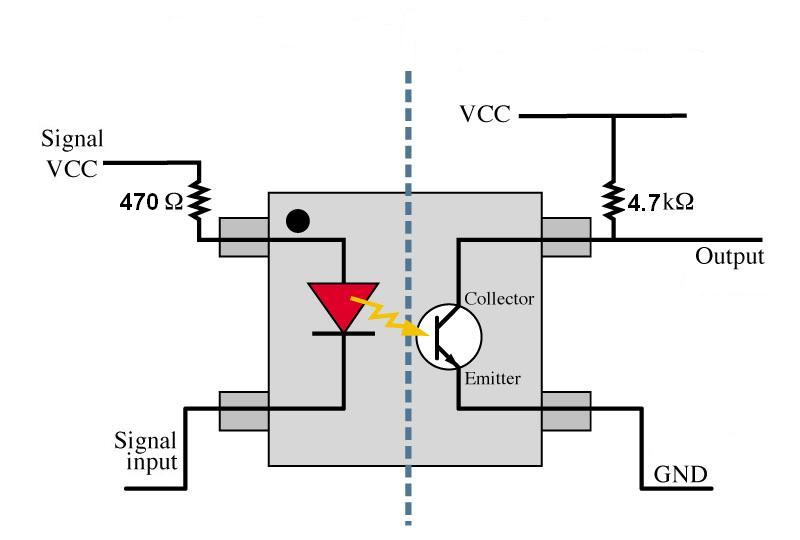
**Opto coupler (PC-817)**

An isolator device to electrically insulate and isolate a separate component in a circuit board arrangement to allow for relatively fast and convenient diagnostic inspection of a circuit to locate failed components

In [electronics](http://en.wikipedia.org/wiki/Electronics), an opto-isolator, also called an optocoupler, photocoupler, or optical isolator, is "an electronic device designed to transfer electrical signals by utilizing light waves to provide coupling with electrical isolation between its input and output.The main purpose of an opto-isolator is "to prevent [high voltages](http://en.wikipedia.org/wiki/High_voltage) or rapidly changing voltages on one side of the circuit from damaging components or distorting transmissions on the other side

An opto-isolator contains a source (emitter) of light, almost always a [near infrared](http://en.wikipedia.org/wiki/Infrared) [light-emitting diode](http://en.wikipedia.org/wiki/Light-emitting_diode) (LED), that converts electrical input signal into light, a closed optical channel (also called dielectrical channel[[5]](http://en.wikipedia.org/wiki/Opto-isolator#cite_note-M100-5)), and a [photosensor](http://en.wikipedia.org/wiki/Photodetector), which detects incoming light and either generates electric [energy](http://en.wikipedia.org/wiki/Energy) directly, or [modulates](http://en.wikipedia.org/wiki/Modulation) [electric current](http://en.wikipedia.org/wiki/Electric_current) flowing from an external power supply. The sensor can be a [photoresistor](http://en.wikipedia.org/wiki/Photoresistor), a [photodiode](http://en.wikipedia.org/wiki/Photodiode), a [phototransistor](http://en.wikipedia.org/wiki/Phototransistor). Pin diagram of PC 817 is shown below.





PC 817 is a 4 pin opto coupler as shown above. A series resistance of 470 ohm is used to limit the voltage across the diode. +5V power supply is connected to the first pin of IC, which is the anode pin diode. 2nd pin is connected to the port of microcontroller. When the second pin is low then we get low output, when the input to 2nd pin high we get high voltage the output. Thus we isolate the voltage having the same logic level.

**MATLAB CODE**

%% set-up serial port

serial\_port = serial('COM4'); %com port 4

fopen(serial\_port); %open port

fprintf(serial\_port,'%c','s');

tangle = 30; %threshhold angle

direction = 0; %initial direction

distance\_old = 70; %reference distance

%% setup wecam

vid = videoinput('winvideo', 1, 'YUY2\_320x240');

src = getselectedsource(vid);

vid.FramesPerTrigger = 1;

vid.ReturnedColorspace = 'rgb';

imaqmem(1000000000);

vid.TriggerRepeat = Inf;

triggerconfig(vid, 'manual');

preview(vid);

start(vid);

%% process images

flag=true;

while(flag==true)

trigger(vid);

m = getdata(vid);

imshow(m(:,:,:,1));

% image segmentation

a = m(:,:,:,1);

subplot(2,3,1),imshow(a);

b = rgb2gray(a);

subplot(2,3,2),imshow(b);

c = im2bw(a,graythresh(b));

subplot(2,3,3),imshow(c);

d = ~ c;

subplot(2,3,4),imshow(d);

% object detection

cc = bwconncomp(d, 4);

s = regionprops(cc, 'Centroid','Orientation');

subplot(2,3,5),imshow(c);

hold on

plot(s(1).Centroid(1), s(1).Centroid(2), '--rs','LineWidth',2,...

'MarkerEdgeColor','k',...

'MarkerFaceColor','g',...

'MarkerSize',10);

plot(s(2).Centroid(1), s(2).Centroid(2), '--rs','LineWidth',2,...

'MarkerEdgeColor','k',...

'MarkerFaceColor','g',...

'MarkerSize',10);

hold off

% angle finding

angle\_box = round(s(1).Orientation); %angle of rect

angle\_pic = round(atan((s(2).Centroid(2) - s(1).Centroid(2))/(s(2).Centroid(1) - s(1).Centroid(1)))\*100);

distance = round(sqrt((s(2).Centroid(2) - s(1).Centroid(2))^2 + (s(2).Centroid(2) - s(1).Centroid(1))^2));

% move calculation

if(distance < 20)

flag=false;

break;

end

if(distance > distance\_old)

direction =~ direction;

end

if ((angle\_box <= angle\_pic + tangle) && (angle\_box >= angle\_pic - tangle))

fprintf(serial\_port,'%c',get\_direction\_char(direction));

elseif (angle\_box > angle\_pic + tangle)

fprintf(serial\_port,'%c','l'); %move right -> 'l'

else

fprintf(serial\_port,'%c','r'); %move left -> 'r'

end

pause(1);

fprintf(serial\_port,'%c','s');

pause(2);

distance\_old = distance;

end

%% clear webcam object

stoppreview(vid);

stop(vid);

delete(vid);

clear vid;

% close com port

fclose(serial\_port);

clear serial\_port;

Another function get\_direction\_char is used to get the traversal direction:

function [direction]=get\_direction\_char(x)

if(x==0)

direction = 'f';

else

direction = 'b';

end

end