CHAPTER-1

1.1 INTRODUCTION

1.1.1 Motivation:

In India, out of the 121 Cr populations,2.68 Cr persons are physically challenged, where by 20 % are impaired in movement. Using technology, the voice commands are given to robots to do simple tasks which they cannot perform through their physical body. And also, this robotics can be used in the industries to carry out work like lifting high weighs and some mix up operations that cannot be performed by the humans.

1.1.2 Problem statement:

The traditional wired buttons-controlled robot becomes very bulgy and it also limits the distance the robot goes. The Wireless Hand controlled Robot will function by a hand gesture from which the movements of the hand can be used as the input for the movement of the robot. The basic idea of our project is to develop a system (Robot) which can recognize the Human Interaction with it to accomplish the certain tasks assigned to it.

In our project we will design a robot which will contain the sensors mounted on it to capture the movement of the hand and convert the raw mechanical data into electrical form. This data will be further processed and converted into an understandable format .

Zigbee will act as a transmitter of the data for wireless communication purpose. Once the transmitted data is received by the receiver module which will be connected to the Aurdino, it will be processed and further sent to the aurdino. Aurdino will deduce the commands and accordingly it will actuate the motor drivers to control the Motors for various tasks on the robot.

1.1.3 Objective:

The aim of the project is to develop a human machine interface used for control robot. Our objective is to make this device simple as well as cheap so it can be produced and used for number of purposes. The objective of this project is to build a robot that can be controlled by gesture wirelessly. In this project user is also able to control motions of the robot by hand gestures and performing predefined gestures. This can be also used in many potential applications such as wireless controller car racing etc.

2.1 Literature Review

2.1.1 Computer vision and Digital Image Processing

The sense of sight is arguably the most important of man's five senses. It provides a huge amount of information about the world that is rich in detail and delivered at the speed of light. However, human vision is not without its limitations, both physical and psychological. Through digital imaging technology and computers, man has transcending many visual limitations. He can see into far galaxies, the microscopic world, the sub-atomic world, and even “observe” infra-red, x-ray, ultraviolet and other spectra for medical diagnosis, meteorology, surveillance, and military uses, all with great success.

While computers have been central to this success, for the most part man is the sole interpreter of all the digital data. For a long time, the central question has been whether computers can be designed to analyze and acquire information from images autonomously in the same natural way humans can. According to Gonzales and Woods , this is the province of computer vision, which is that branch of artificial intelligence that ultimately aims to “use computers to emulate human vision, including learning and being able to make inferences and taking actions based on visual inputs.”

The main difficulty for computer vision as a relatively young discipline is the current lack of a final scientific paradigm or model for human intelligence and human vision itself on which to build a infrastructure for computer or machine learning . The use of images has an obvious drawback. Humans perceive the world in 3D, but current visual sensors like cameras capture the world in 2D images. The result is the natural loss of a good deal of information in the captured images. Without a proper paradigm to explain the mystery of human vision and perception, the recovery of lost information (reconstruction of the world) from 2D images represents a difficult hurdle for machine vision .

However, despite this limitation, computer vision has progressed, riding mainly on the remarkable advancement of decades-old digital image processing techniques, using the science and methods contributed by other disciplines such as optics, neurobiology, psychology, physics, mathematics, electronics, computer science, artificial intelligence and others.

Computer vision techniques and digital image processing methods both draw the proverbial water from the same pool, which is the digital image, and therefore necessarily overlap. Image processing takes a digital image and subjects it to processes, such as noise reduction, detail enhancement, or filtering, for the purpose of producing another desired image as the result. For example, the blurred image of a car registration plate might be enhanced by imaging techniques to produce a clear photo of the same so the police might identify the owner of the car.

On the other hand, computer vision takes a digital image and subjects it to the same digital imaging techniques but for the purpose of analyzing and understanding what the image depicts. For example, the image of a building can be fed to a computer and thereafter be identified by the computer as a residential house, a stadium, high-rise office tower, shopping mall, or a farm barn.

Russell and Norvig identified three broad approaches used in computer vision to distill useful information from the raw data provided by images. The first is the feature extraction approach, which focuses on simple computations applied directly to digital images to measure some useable characteristic, such as size.

This relies on generally known image processing algorithms for noise reduction, filtering, object detection, edge detection, texture analysis, computation of optical flow, and segmentation, which techniques are commonly used to pre-process images for subsequent image analysis. This is also considered an “uninformed” approach. The second is the recognition approach, where the focus is on distinguishing and labelling objects based on knowledge of characteristics that sets of similar objects have in common, such as shape or appearance or patterns of elements, sufficient to form classes. Here computer vision uses the techniques of artificial intelligence in knowledge representation to enable a “classifier” to match classes to objects based on the pattern of their features or structural descriptions.

A classifier has to “learn” the patterns by being fed a training set of objects and their classes and achieving the goal of minimizing mistakes and maximizing successes through a step-by-step process of improvement. There are many techniques in artificial intelligence that can be used for object or pattern recognition, including statistical pattern recognition, neural nets, genetic algorithms and fuzzy systems.

The third is the reconstruction approach, where the focus is on building a geometric model of the world suggested by the image or images and which is used as a basis for action. This corresponds to the stage of image understanding, which represents the highest and most complex level of computer vision processing.

Here the emphasis is on enabling the computer vision system to construct internal models based on the data supplied by the images and to discard or update these internal models as they are verified against the real world or some other criteria.

CHAPTER-3

3.1 OpenCV

OpenCV is a widely used tool in computer vision. It is a computer vision library for real-time applications, written in C and C++, which works with the Windows, Linux and Mac platforms. OpenCV was started by Gary Brad sky at Intel in 1999 to encourage computer vision research and commercial applications and, side-by-side with these, promote the use of ever faster processors from Intel . OpenCV contains optimised code for a basic computer vision infrastructure so developers do not have to re-invent the proverbial wheel.

A digital image is generally understood as a discrete number of light intensities captured by a device such as a camera and organized into a two-dimensional matrix of picture elements or pixels, each of which may be represented by number and all of which may be stored in a particular file format (such as jpg or gif) . OpenCV goes beyond representing an image as an array of pixels. It represents an image as a data structure called an IplImage that makes immediately accessible useful image data or fields, such as:

- width – an integer showing the width of the image in pixels
- height – an integer showing the height of the image in pixels
- image Data – a pointer to an array of pixel values
- n channels – an integer showing the number of colors per pixel
- depth – an integer showing the number of bits per pixel
- width Step – an integer showing the number of bytes per image row
- image Size – an integer showing the size of in bytes.

OpenCV has a module containing basic image processing and computer vision algorithms.

These include:

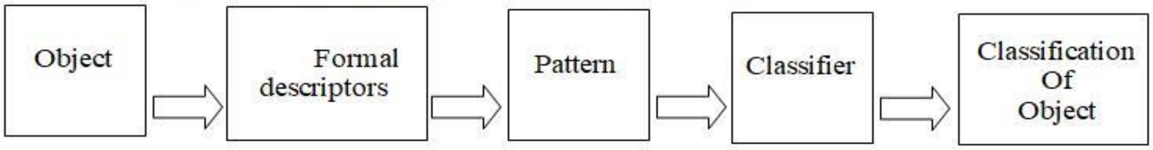
- smoothing (blurring) functions to reduce noise
- dilation and erosion functions for isolation of individual elements
- flood fill functions to isolate certain portions of the image for further processing
- filter functions, including Sobel, Laplace and Canny for edge detection
- Huff transform functions for finding lines and circles
- Affine transform functions to stretch, shrink, warp and rotate images
- Integral image function for summing sub regions (computing Haar wavelets)
- Histogram equalization function for uniform distribution of intensity values
- Contour functions to connect edges into curves
- Bounding boxes, circles and ellipses
- Moments functions to compute Hu's moment invariants
- Optical flow functions (Lucas-Kanade method)
- Motion tracking functions (Kalman filters) and
- Face detection/ Haar classifier.

3.2 Pattern Recognition and Classifiers

In computer vision a physical object maps to a particular segmented region in the image from which object descriptors or features may be derived. A feature is any characteristic of an image, or any region within it, that can be measured. Objects with common features may be grouped into classes, where the combination of features may be considered a pattern.

Object recognition may be understood to be the assignment of classes to objects based on their respective patterns. The program that does this assignment is called a classifier.

Figure 1. General pattern recognition steps.



The most important step is the design of the formal descriptors because choices have to be made on which characteristics, quantitative or qualitative, would best suit the target object and in turn determines the success of the classifier.

In statistical pattern recognition, quantitative descriptions called features are used. The set of features constitutes the pattern vector or feature vector, and the set of all possible patterns for the object form the pattern space X (also known as feature space).

Quantitatively, similar objects in each class will be located near each other in the feature space forming clusters, which may ideally be separated from dissimilar objects by lines or curves called discrimination functions. Determining the most suitable discrimination function or discriminant to use is part of classifier design.

A statistical classifier accepts *n* features as inputs and gives 1 output, which is the classification or decision about the class of the object. The relationship between the inputs and the output is a decision rule, which is a function that puts in one space or subset those feature vectors that are associated with a particular output. The decision rule is based on the particular discrimination function used for separating the subsets from each other.

The ability of a classifier to classify objects based on its decision rule may be understood as classifier learning, and the set of the feature vectors (objects) inputs and corresponding outputs of classifications (both positive and negative results) is called the training set. It is expected that a well-designed classifier should get 100% correct answers on its training set.

A large training set is generally desirable to optimize the training of the classifier, so that it may be tested on objects it has not encountered before, which constitutes its test set. If the classifier does not perform well on the test set, modifications to the design of the recognition system may be needed.

3.3 Moment Invariants

As mentioned previously, feature extraction is one approach used in computer vision. According to A.L.C. Barczak, feature extraction refers to the process of distilling a limited number of features that would be sufficient to describe a large set of data, such as the pixels in a digital image. The idea is to use the features as a unique representation of the image.

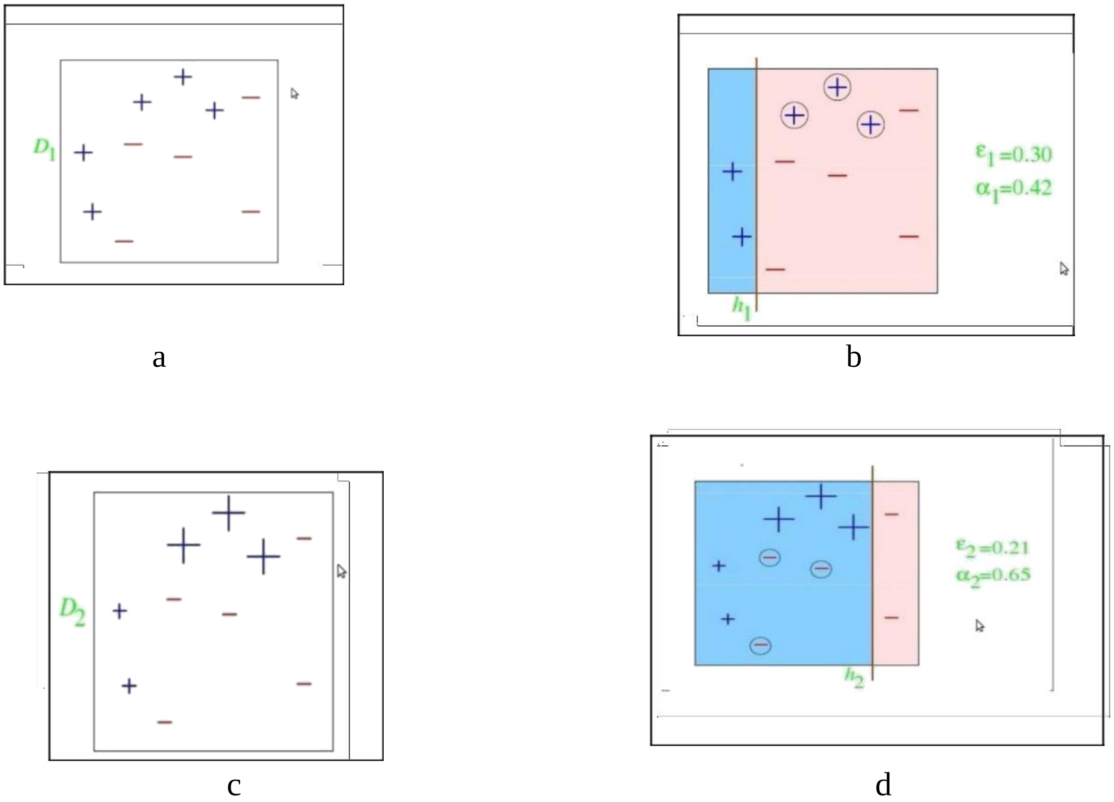
Since a digital image is a two-dimensional matrix of pixels values, region-based object descriptions are affected by geometric transformations, such as scaling, translation, and rotation. For example, the numerical features describing the shape of a 2D object would change if the shape of the same object changes as seen from a different angle or perspective. However, to be useful in computer vision applications, object descriptions must be able to identify the same object irrespective of its position, orientation, or distortion.

One of the most popular quantitative object descriptors are moments. The concept of statistical characteristics or moments that would be indifferent to geometric transformations was first formulated by Hu in 1962. Moments are polynomials of increasing order that describe the shape of a statistical distribution. The order of a moment is indicated by its exponent. The geometric moments of different orders represent different spatial characteristics of the image intensity distribution.

3.4 AdaBoost Algorithm

AdaBoost (Adaptive Boosting) is a machine learning algorithm to build a classifier based on boosting, a technique which was introduced by Freund and Schapire in 1999. The idea of boosting is to construct a single strong classifier from a linear combination of weak classifiers. A weak classifier uses a decision rule that has a performance value of only slightly above 50%, just over chance. A weak classifier might use a simple threshold function. In the method of boosting, at each round of training, the training features are given new weights based on whether they were correctly classified or not in the previous round by the weak classifiers. Mis-classified features are given bigger weights and correctly classified features are given lower weights for the next training round.

Figure 2 shows an illustration of boosting.

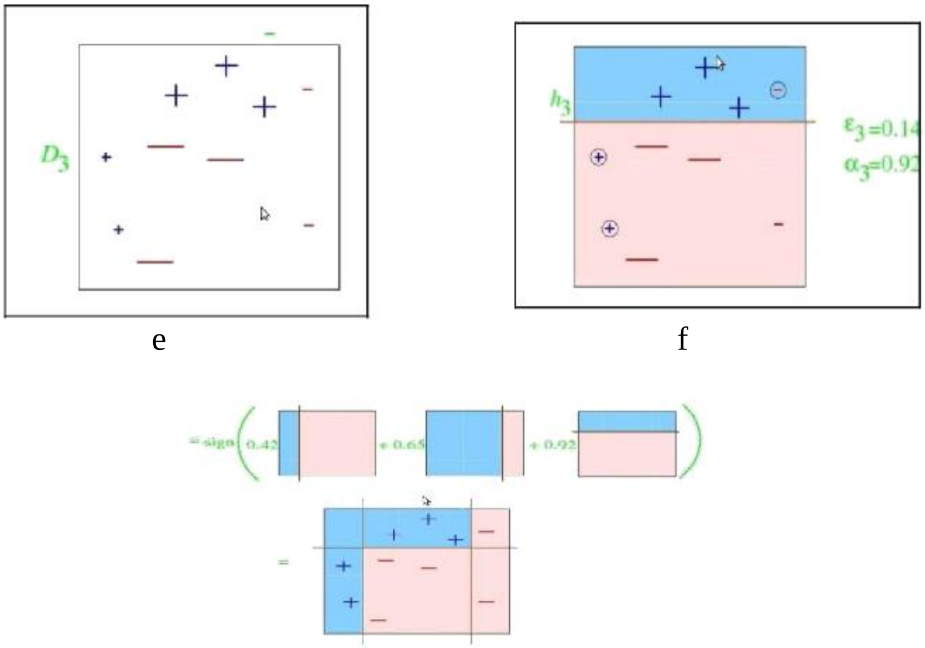


a

b

c

d



e

f

g

Figure 2. Illustration of stages in boosting. (a) Elements are given equal weights in $D_{1.}$ *(b)* The classifier h_1 incorrectly identifies 3 elements. (c) In D_2 the 3 mis-identified elements are given greater weight, the correct ones are minimized. (d) The classifier h_2 incorrectly identifies 3 elements. (e) In D_3 the 3 misidentified elements are given greater weight, the correct ones are minimized. (f) The classifier h_3 incorrectly identifies 1 elements. (g) The final classifier combines the acceptable performances of classifiers h_1, h_2, and h_3.

The algorithm for a discrete version of AdaBoost is shown below :

- Input the training examples with labels {(x_1, y_1), (x_2,y_2),... (x_i, y_i), where x_i is a feature vector in the feature space X and y_i is a label from Y = {-1 or +1} for negative or positive results.
- Initialize all the weights $D_1(i)$ = 1/(n), where n is the number of elements in the feature vector.
- For $t = 1,2,$ … T (T is the number of rounds):

CHAPTER-4

4.1 Description of the Robot

The Massey robot (known as Omni-Robot) has an omni-directional camera that gives it a 360 degree field of vision so that it can see in all directions. It is also capable of omni-directional movement (at any angle) with the use of its 3 Omni-Wheel sets. It can run at a maximum acceleration of 1.8 m/sec^2. The robot has an on-board PC that runs on Windows and can communicate with it directly by serial connection. A remote network connection is also possible with the server program running on the robot PC. It has direct power connection as well as a battery pack. It has a native manual dialog control program for robot movement in serial mode with buttons for the following movements: Forward, Left, Halt, Right, Backward, Rotate Clock-wise, and Rotate Counter Clock-wise.

A computer vision application can be made to capture images containing hand gestures from the robot's omni-directional camera, detect and recognize the hand gestures, and output the applicable commands to navigate the robot.

4.1.1 Methodology

This project availed of several algorithms commonly used in computer vision. These include those used in colour segmentation, labelling, blob detection, feature extraction, and gesture recognition.

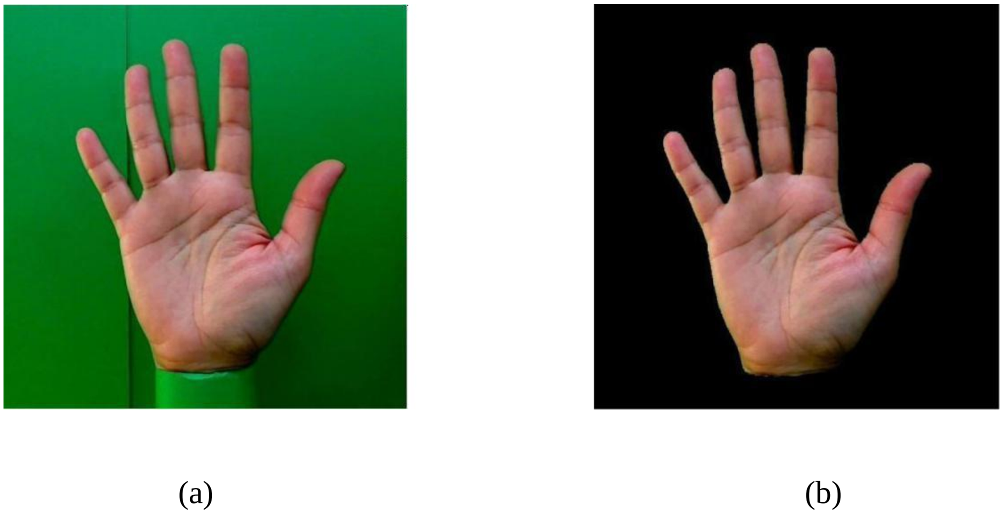
4.1.1.1 Algorithm for Color Segmentation Using Thresholding

Segmentation is the process of identifying regions within an image. Color can be used to help in segmentation. In this project, the hand on the image was the region of interest. To isolate the image pixels of the hand from the background, the range of the HSV values for skin colour was determined for use as the threshold values. Segmentation could then proceed after the conversion of all pixels falling within those

threshold values to white and those without to black.
The algorithm used for the colour segmentation using thresholding is shown below:

- Capture an image of the gesture from the camera.
- Determine the range of HSV values for skin colour for use as threshold values.
- Convert the image from RGB colour space to HSV colour space.
- Convert all the pixels falling within the threshold values to white.
- Convert all other pixels to black.
- Save the segmented image in an image file.

Figure 3 shows a sample raw image and the resulting image after colour segmentation by thresholding.



(a) (b)

Figure 3: Sample images. (a) Original image. (b) Image after colour segmentation.

4.1.1.2 Algorithm for Labeling and Blob Detection

The gesture in the colour segmented image should be recognized as one object before it can be interpreted. This can be done through the process of labeling and blob detection.

Labeling is the process of giving each region a unique integer number or label for the purpose of regional identification. In effect, while no two neighboring regions should have the same label, the pixels within one region should have the same label or description so that the region could be interpreted as one object or blob.

For the purpose of determining whether pixels might belong to the same region, their adjacency relationships can be examined. The two most common adjacency relationships are:

- 4-adjacency and
- 8-adjacency

In 4-adjacency, a pixel is considered connected to its neighboring pixels if they occupy the left- most, top-most, right-most, and bottom positions with respect to the pixel. Using the (x,y) coordinate descriptions, a 4-adjacency relationship for pixel (x, y) is shown in Figure 4.

	(x, y-1)	
(x-1, y)	(x, y)	(x+1, y)
	(x, y+1)	

Figure 4. 4-Adjacency connectivity model.

In 8-adjacency, the neighboring pixels also include the top-left-most, top-right-most, bottom-left- most, and the bottom-right-most positions. Using the (x,y) coordinate descriptions, a 8-adjacency relationship for pixel (x, y) is shown in Figure 5.

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	(x, y)	(x+1, y)
(x-1, y+1)	(x, y+1)	(x+1, y+1)

Figure 5. 8-Adjacency connectivity model.

In labelling algorithms, pixels are examined one by one, row by row, and moving from left to right. Therefore, for the practical implementation of the algorithm, only the pixels that may be considered as existing at each point in time with respect to the pixel under scrutiny are considered. For the 4- adjacency model, these pixels would be the top-most and the left-most, as shown in Figure 6.

	(x, y-1)
(x-1, y)	(x, y)

Figure 6. Significant pixels in 4-adjacency model.

In the 8-adjacency model, these pixels would be the top-left-most, the top-most, the top-right-most, and the left-most, as shown in Figure 7.

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	(x,y)	

Figure 7. Significant pixel in 8-adjacency model.

Usually, images that undergo labelling are given preliminary processing to become binary images or grayscale images. This will make it easier to identify the pixels of interest because the background pixels would either be zero-valued (colored black) or measured against a threshold value in grayscale.

The labelling procedure looks at each pixel in sequence, checks it against the threshold value, and identifies its neighboring pixels based on the adjacency relationship model being used. If the neighboring pixels belong to a set, the pixel under consideration will be placed in that set. If the pixel has no neighbors, then it will be placed in a new set. Thereafter, all sets with connecting pixels will be merged together into one set or blob and be considered as one object.

The algorithm for labelling and blob detection using an 8-adjacency relationship and a threshold value for a grayscale image is as follows:

1. Select the threshold value for the grayscale image.
2. For each non-zero pixel in the image that is above the threshold value:
    - If the pixel has non-zero labeled pixels in its immediate 8-adjacency neighborhood (namely, the top-right, top, top-left and left pixels), then give it a non-zero label (for example, assign the minimum value of the pixels in this neighborhood to the pixel as its label).
    - If the pixel has no neighboring pixels, then give it a new unused label and include it as part of a new set.
3. After all the pixels have been labeled and placed in sets, merge together the sets with connected pixels into one blob or object. Objects may be given different colours to distinguish them visually.

Figure 8 shows a sample original segmented image of hand with skin colour that is converted to a grayscale image, and the finally into two separate images, one showing a single blob using a lower lower threshold and another showing multiple blob using a higher threshold value.

a

b

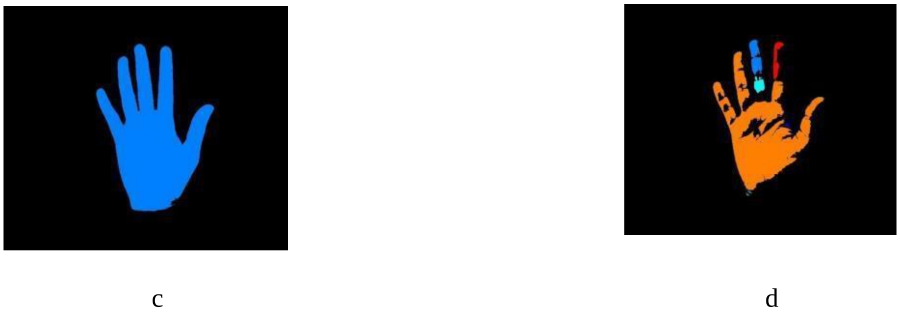


Figure 8: Sample images showing the results of labelling and blob detection. (a) Original segmented image. (b) Image in grayscale. (c) Image with a single blue-colored blob, using of a threshold pixel value = 10. (d) Image showing multiple blobs with individual colours, using a higher threshold pixel value = 120.

4.1.1.3 CONVEX DETECT BASED FINGERTIP DETECTION

In issues involving human hands such as sign language recognition and gesture recognition, fingertip is one of the most popular characteristics because the number of fingertips can be considered to be the number of fingers and the direction of fingertips can effectively express the stretch information of fingers. Contour analysis is a commonly used method for fingertip detection, which achieves the location of fingertip based on geometric features of contour, such as the edge curvature method used for contour detection this kind of algorithms require high accuracy of contour and a large amount of computation, and are very dependent on the quality of gesture segmentation.

1) Contour

The contour of hand is a series of points which are the boundary pixels of the hand area. After obtaining the contour, the gesture and its shape then can be detected and recognized by using contours analysis. Contour is constituted by the connection of edges, common edge recognition algorithms contain Sobel, Canny, Prewitt, Roberts, and Fuzzy logic methods .Here we use Canny edge detection method to extract the contour of hand gesture, the red curve in is the contour of hand gesture contained.

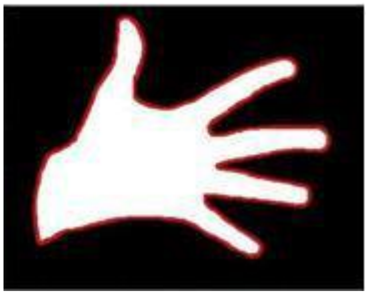


Figure 9. Detected contour of hand gesture

2) Convex hull

The convex hull of hand gesture contour is the convex polygon surrounded by all the convex vertices in gesture contour, the polygon composed by red curve is the convex hull of hand gesture in the figure, and is the separated convex hull extracted

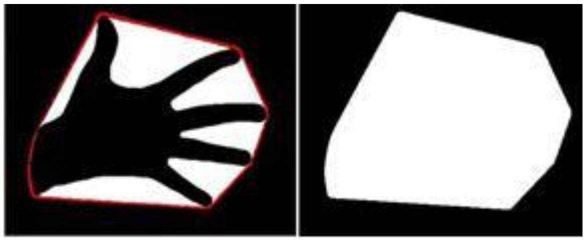


Figure 10. Convex hull of hand gesture

3) Convex defect

The convex defect is defined as the difference between gesture convex hull and contour, they are contained in the convex hull but not hand area. As the white areas 1-6 are all the convex defects. The data structure of each of the convex defects contains three components: start contour point, end contour point and concave contour point.

For example, for convex defect 2, P1 is its start contour point which is the starting point of the defect, P2 is its end contour point which is the termination point of the defect, and P3 is the concave point which is the furthest point away from the convex hull, and the furthest distance is the depth of convex defect.

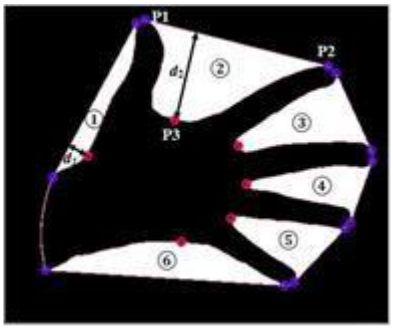


Figure 11. Convex defects of hand gesture

We can get from that the fingertip is closely related to the convex defect, which is close to the start and end contour points of convex defect.

Therefore, it is possible to detect the fingertips by using hand gesture contour and convex defects.

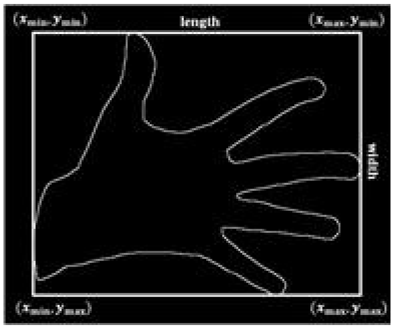


Figure 12. Definition of length and width of hand gesture contour

The count and position of fingertips can be determined as following:

1) Conduct noise elimination on the obtained convex defects. The normalized depth of convex defect cannot be too small or too large, that the depth of convex defects between fingers is more obvious than other convex defects, so the depth of convex defect can be used to distinguish whether a convex defect is a convex defect between fingers or not.

Based on the physiological structure of human hand and large numbers of experiments, it is appropriate to define the depth minimum and maximum threshold as big as the 1/5 and 1/2 of the height of hand gesture contour respectively, and the height of hand gesture contour is defined as the length of the quadrilateral formed by points composed of the minimum x-coordinate value, maximum x-coordinate value, minimum y-coordinate value and maximum y-coordinate value, namely, points of (xmin, ymin), (xmin, ymax), (xmax, ymin), (xmax, ymax) .Thus, for the convex depth in d2 is the depth meeting the condition of gesture convex defects while d1 is not.

2) Scan filtered convex defects clockwise, take the start contour point of the first convex defect and the end contour point of the last convex defect as the first and last fingertip respectively.

3)Take use of the average position of the end contour point of current convex defect and the start contour point of next convex defect as the position of current fingertip.

4.1.1.4 FEATURES EXTRACTION

For the binary image with convex defects, we can extract the convexity of gesture and relative position of fingertips as features used for hand gesture recognition. Through the observation and analysis of gesture contour and convex hull, we can get that with different gesture the tightness to its convex hull is also different, as shown in the convex hull of the fist gesture in the shown figure almost contains the whole gesture contour, but the gesture contour in has big difference with its convex hull, with several depression existing between. The tightness of hand gesture contour to its convex hull is defined as the gesture convexity, which is denoted by δ, its value is the area ratio of gesture contour and convex hull.

δ =(contourArea) / (hullArea)

where, hull Area is the area of convex hull, contour Area is the area of gesture contour, we can get according to the image that hull Area > contour Area, so $\delta \varepsilon (0,1)$, and the bigger the value, the tighter the gesture contour to convex hull.

In addition, different gestures can be distinguished by the relative position of fingertips, which is composed of two values of α and β, α is the summation of angles with centroid as vertex, lines from centroid to the first fingertip and other fingertips as edges, and β is the value of the angle with centroid as vertex, lines from centroid to the first fingertip and $\beta = \theta N-1$,
where N is the number of fingertips and θ is the angle between the first fingertip and the other fingertip to the gesture centroid been considered as vertex, it has $N = 3$, $\alpha = \theta 1 + \theta 2$, $\beta = \theta 2$.
Different gestures have significantly different relative position of fingertips, so α, β can be used to the further recognition of different hand gestures.

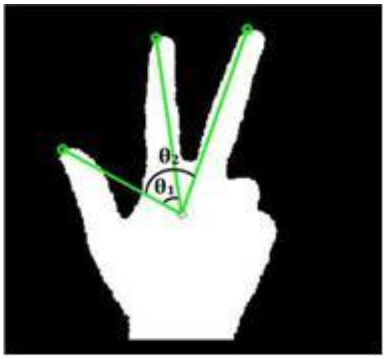


Figure 13. Definition of relative position of fingertips

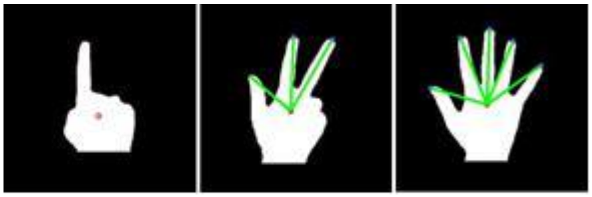


Figure 14. Different fingertip relative position of different hand gestures

CHAPTER-5

5.1 Project Steps and Results

Step 1: Choose Four Gestures for Simple Robot Control

The first step in the project was to choose four gestures that would be used for simple robot control. The gestures should be simple, easy to understand, and easy for ordinary people to use. Also, the gestures should be different enough from each other so it would be easy to formulate some unique features for each of them for purposes of object recognition. Finally, the most common navigation commands were chosen, namely Stop, Move Forward, Move Backward, Move Left, and Move Right.

Step 2: Implement the Algorithms Using C/C++ and OpenCV

The next step was to implement the needed algorithms to process the images and to recognize the gestures, using C/C++ and OpenCV. The resulting program should do the following procedure:

- capture the gestures through a web camera
- set-up an interface to convert from RGB to HSV colour space and determine the HSV values for skin colour
- use the HSV values to reduce the image to black and white
- produce a grayscale image in preparation for labelling and blob detection
- produce a blob of the hand in preparation for feature extraction
- extract features from the blob for gesture recognition
- output the recognized gesture/ command for robot navigation

5.1.1 Capture of Images through a Webcam

If the webcam is properly installed, OpenCV would easily detect it and OpenCV capture functions can be used to get the images from the camera. The CVCaptureFrom CAM() function initializes capturing video from the camera and stores it in the CvCapture structure, which is the video capturing structure. The cvCaptureFromCAM() function may be passed an index number to identify which camera is to be the source of images. In case there is only 1 camera, the parameter (0) is used. Thereafter, the CVGrabFrame() function may grab the frame (image) from the CVCapture structure, and from there the frame may be retrieved through the CVRetrieveFrame function and stored in an IplImage structure [9]. From there, the image can be displayed on the screen by the CVShowImage() function in a window created through the CVNamedWindow() function. For video streaming the process of capture and display may be placed inside a loop.

Below is the sample code fragment for capturing and displaying images through the webcam:

```
IplImage
*imagecaptured;
CvCapture* capture = 0;
capture=cvCaptureFrom
CAM (0);
for ( ; ; )
{
if( cvGrabFrame( capture )) {
image captured = cvRetrieveFrame( capture );
}
cvNamedWindow("capturedimage",0);
cvShowImage( "captured image", image captured );}
```

5.1.2 Interface for Adjustment of HSV Values for Skin color

The range of HSV values for skin colour had to be determined in preparation for the reduction of the image into two colours for eventual segmentation. OpenCV trackbars were used for this purpose. The trackbars or sliders represented the minimum and maximum range of the Hue, Saturation and Value of the HSV colour space. Skin colour was successfully isolated .

However, the combination did not work all of the time. The HSV values for skin colour had to be adjusted each time, depending on the type of camera used and the amount of illumination present. Fortunately, making the periodic adjustments was facilitated by the use of the trackbars.

The cvCreateTrackbar() function used to create the trackbars have the following parameters: the trackbar name, window name, pointer to integer value showing the trackbar position, maximum position of the trackbar, and the pointer to the function to be called when the trackbar changes position.

5.1.2.1 Production of a Black and White Image

After the HSV skin colour range was determined, it was used as the threshold range to convert the image into two colours: black for the background and white for the hand. It was easier to implement the conversion through a separate function. Here the Find markers() function first converts the source RGB image to an HSV image (using cvCvtColor()), and then loops through each pixel in the HSV image to pick out the skin-colored pixels, turns the corresponding pixels in the original source image to white, and finally turns all other pixels in the original source image to black. This converts the original image into black and white.

5.1.2.2 Production of a Grayscale Image

In OpenCV one of the methods to convert images to grayscale is through the use of the cvCvtColor () function. This function is actually a general function that converts an image from one colour space to another. Grayscale is created by changing the pixel intensities of each colour by specific amounts so that it results in a single gray intensity for output in a single-channel image, as follows:

Red * 0.299 + Green * 0.587 + Blue * 0.114 = Gray scale pixel value

The cvCvtColor() function uses the following parameters: a source 3-bit, 16-bit or single-precision floating-point image, a destination image of the same data type, although channels are allowed to differ (in this case only 1-channel), and the colour conversion code, which follows the format CV_ "colour space" 2 "colour space", namely: CV_BGR2GRAY.

The cvCreateImage() function creates an image and allocates the image data accordingly. Its parameters are: image size (width and height), image bit-depth, and number of channels per pixel.

- IPL_DEPTH_8U (unsigned 8-bit integers)
- IPL_DEPTH_8S (signed 8-bit integers)
- IPL_DEPTH_16U (unsigned 16-bit integers)
- IPL_DEPTH_16S (signed 16-bit integers)
- IPL_DEPTH_32S (signed 32-bit integers)
- IPL_DEPTH_32F (single precision floating-point numbers)
- IPL_DEPTH_64F (double precision floating-point numbers)

5.1.2.3 Implementation of Heuristic Rules and Output

Based on the features extracted as discussed above, the following heuristic rules for gesture recognition were easily set out:

- For the Move Left gesture, the longest distance from the center of mass is from the left- most pixel of the hand (the tip of the forefinger) to the hand's center of mass.

- For the Move Right gesture, the longest distance from the center of mass is from the right- most pixel of the hand (the tip of the forefinger) to the hand's center of mass.

However, preliminary testing with the Move Forward and Stop gestures revealed that just getting the longest distance from the center of mass to the top-most pixel (tip of the forefinger) alone was not enough to distinguish between the two gestures, since both gestures have the fingers pointing upwards or in the same direction.

The following additional heuristic rules were constructed to remedy this problem:

- If the longest distance from the hand's center of mass points upward and only one finger is detected, then the gesture is interpreted as Move Forward.

- If the longest distance from the hand's center of mass points upward and more than one finger is detected, then the gesture is interpreted as Stop.

CHAPTER-6

6.1 Introduction to Robotics

A **robot** is a virtual or mechanical artificial agent in practice, it is usually an electro-mechanical machine which is guided by computer or electronic programming, and is thus able to do tasks on its own. Another common characteristic is that by its appearance or movements, a robot often conveys a sense that it has intent of its own.

The Robotic Industries Association defines *robot* as follows: "A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks." Recently, however, the industry's current working definition of a robot has come to be understood as any piece of equipment that has three or more degrees of movement or freedom.

Robotics is an increasingly visible and important component of modern business, especially in certain industries. Robotics-oriented production processes are most obvious in factories and manufacturing facilities; in fact, approximately 90 percent of all robots in operation today can be found in such facilities. These robots, termed "industrial robots," were found almost exclusively in automobile manufacturing plants as little as 15 to 20 years ago. But industrial robots are now being used in laboratories.

Today's robotics systems operate by way of hydraulic, pneumatic, and electrical power. Electric motors have become progressively smaller, with high power-to-weight ratios, enabling them to become the dominant means by which robots are powered.

Robots are programmed either by guiding or by off-line programming. Most industrial robots are programmed by the former method. This involves manually guiding a robot from point to point through the phases of an operation, with each point stored in the robotic control system. With off-line programming, the points of an operation are defined through computer commands. This is referred to as manipulator level off-line programming. An important area of research is the development of off-line programming that makes use of higher-level languages, in which robotic actions are defined by tasks or objectives.

An industrial robot is officially defined by ISO as an automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes. The field of robotics may be more practically defined as the study, design and use of robot systems for manufacturing (a top-level definition relying on the prior definition of robot).

Robots may be programmed to move through a specified continuous path instead of from point to point. Continuous path control is necessary for operations such as spray painting or arc welding a curved joint. Programming also requires that a robot be synchronized with the automated machine tools or other robots with which it is working. Robots are moving out of the realm of science fiction and into real-life applications, with the usage of robots in industry, food service and health care. Robots have long been used in assembling machines, but reliability was a problem as was the need to design products so that robots could assemble them. Now with better controls and sensors, and the use of complex programming, robots are being used in areas dangerous to humans, such as nuclear power plants. While robots have not proved successful in food service several home robots will carry dishes and other small loads from room to room. A friend, recovering from hip surgery, used his cye to carry food from the kitchen to the living room, and the dirty dishes back into the kitchen again. Since he was on crutches, this was a real lifesaver. Future robots could carry water in a storage container, and use this to water plants, or even fill a pets bowl.

The use of industrial robots is becoming more widespread. They are primarily used for the automation of mass production in factories. Industrial robots have the ability to perform the same tasks repeatedly without stopping. An industrial robot is used for applications such as welding, painting, assembly, palletizing, cutting, and material handling. Robot supports a variety of robotic applications such as arc welding, spot welding, machine loading, and palletizing, which utilize robotic grippers, and robotic tooling.

Typical applications of robots include welding, painting, assembly, pick and place, packaging and palletizing, product inspection, and testing, all accomplished with high endurance, speed, and precision.

6.1.1 Selecting a robot:

A large range of robots with different components, techniques, and means of operation have already been designed and manufactured. These are selected according to their utility and financial considerations. A futuristic robot, with modern sensors and appropriate software, can perform tasks efficiently, accurately, and quickly, but will be expensive.

Thus, all the relevant factors must be considered while selecting robots for industrial applications, including the initial expenditure and the benefits to be achieved in using the robot.

6.1.2 Advantages of Robotics:

Robotics is very advantageous in several ways to man kind. For example, humans work in many unsuitable places and conditions like chemical plants, or pharmaceuticals and exposure to some chemicals constantly may not be good for the humans. However, if these responsibilities are automated using robots, then human beings need not face work based injuries and diseases. When it comes to handling hazardous materials robots are better suited. There are similar advantageous applications for a robot in several other industries.

Today, robots are also used to launch satellites and travel to a different planet altogether. Robots are being launched on Mars to explore the planet and are being designed with intelligence at par with humans.

Robotic systems have the capability of impressively meliorating the quality of work. They don't make any mistakes and errors as humans do. This saves a lot of important output and production time. They provide optimum output in regards to quality as well as quantity. In the medical field, they are used to carry out complicated surgeries which are very difficult for doctors and surgeons to perform.

The use of robotic systems in the industrial sector is a necessity nowadays, as more and more products are to be manufactured in a very less time, and that too with high-quality and accuracy. Big industrial manufacturing giants have robotic systems that work 24/7. Such systems can even do the work of approximately 100 or more human workers at a time.

Future robotics systems may come up with benefits that we can't even imagine of. In many films, the robotic hand has been showed; who knows it may become a reality in the near future. The advantages of robotics are certainly predicted to grow in several other fields over time.

6.1.3 Disadvantages of Robotics:

- Robots cannot respond properly at the times of emergency and danger
- They are expensive
- They have limited duties
- They can only do what they have been ordered to do
- High initial cost of robotic systems and robots
- Possible need for extra space, and new technology, to accommodate robotic systems and robots
- Importance of using highly skilled and technical engineers, programmers and others to set up robotic systems and robots to prevent unnecessary future problems and mishaps

6.1.4 Applications:

Robotics has been of interest to mankind for over one hundred years. A robots characteristics change depending on the environment it operates in. Some of these are:

- **OUTER SPACE**

  Manipulative arms that are controlled by a human are used to unload the docking bay of space shuttles to launch satellites or to construct a space station.

- **THE INTELLIGENT HOME**

  Automated systems can now monitor home security, environmental conditions and energy usage. Door and windows can be opened automatically and appliances such as lighting and air conditioning can be pre programmed to activate. This assists occupants irrespective of their state of mobility.

- **EXPLORATION**

  Robots can visit environments that are harmful to humans. An example is monitoring the environment inside a volcano or exploring our deepest oceans. NASA has used robotic probes for planetary exploration since the early sixties.

- **MILITARY ROBOTS**

  Airborne robot drones are used for surveillance in today's modern army. In the future automated aircraft and vehicles could be used to carry fuel and ammunition or clear mine fields.

- **FARMS**

  Automated harvesters can cut and gather crops. Robotic dairies are available allowing operators to feed and milk their cows remotely.

- **THE CAR INDUSTRY**

  Robotic arms that are able to perform multiple tasks are used in the car manufacturing process. They perform tasks such as welding, cutting, lifting, sorting and bending. Similar applications but on a smaller scale are now being planned for the food processing industry in particular the trimming, cutting and processing of various meats such as fish, lamb, beef.

- **HOSPITALS**

  Under development is a robotic suit that will enable nurses to lift patients without damaging their backs. Scientists in Japan have developed a power-assisted suit which will give nurses the extra muscle they need to lift their patients - and avoid back injuries.

  The suit was designed by Keijiro Yamamoto, a professor in the welfare-systems engineering department at Kanagawa Institute of Technology outside Tokyo. It will allow caregivers to easily lift bed-ridden patients on and off beds.

  In its current state the suit has an aluminum exoskeleton and a tangle of wires and compressed-air lines trailing from it. Its advantage lies in the huge impact it could have for nurses. In Japan, the population aged 14 and under has declined 7% over the past five years to 18.3 million this year. Providing care for a growing elderly generation poses a major challenge to the government.

  Robotics may be the solution. Research institutions and companies in Japan have been trying to create robotic nurses to substitute for humans.

Yamamoto has taken another approach and has decided to create a device designed to help human nurses.

In tests, a nurse weighing 64 kilograms was able to lift and carry a patient weighing 70 kilograms. The suit is attached to the wearer's back with straps and belts. Sensors are placed on the wearer's muscles to measure strength. These send the data back to a microcomputer, which calculates how much more power is needed to complete the lift effortlessly.

The computer, in turn, powers a chain of actuators - or inflatable cuffs - that are attached to the suit and worn under the elbows, lower back and knees. As the wearer lifts a patient, compressed air is pushed into the cuffs, applying extra force to the arms, back and legs. The degree of air pressure is automatically adjusted according to how much the muscles are flexed. A distinct advantage of this system is that it assists the wearer‟s knees, being only one of its kinds to do so.

A number of hurdles are still faced by Yamamoto. The suit is unwieldy, the wearer can't climb stairs and turning is awkward. The design weight of the suit should be less than 10 kilograms for comfortable use. The latest prototype weighs 15 kilograms. Making it lighter is technically possible by using smaller and lighter actuators. The prototype has cost less than ¥1 million ($8,400) to develop. But earlier versions developed by Yamamoto over the past 10 years cost upwards of ¥20 million in government development grants.

- **DISASTER AREAS**

Surveillance robots fitted with advanced sensing and imaging equipment can operate in hazardous environments such as urban setting damaged by earthquakes by scanning walls, floor sand ceilings for structural integrity.

- **ENTERTAINMENT**

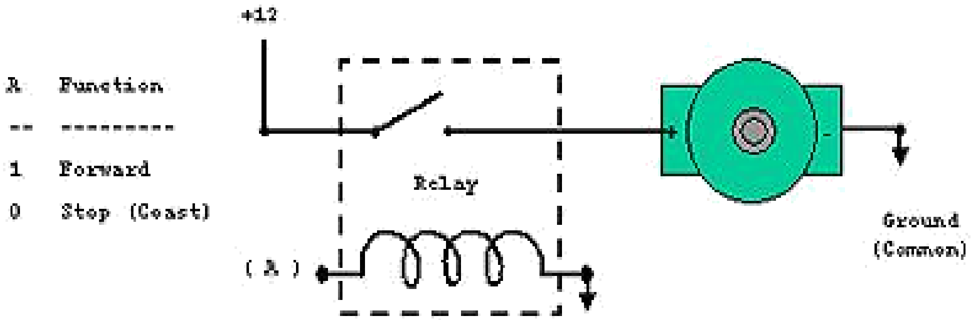
Interactive robots that exhibit behaviors and learning ability. SONY has one such robot which moves freely, plays with a ball and can respond to verbal instructions.

6.1.5 H-Bridge

How do we make a motor turn?

You take a battery; hook the positive side to one side of your DC motor. Then you connect the negative side of the battery to the other motor lead. The motor spins forward. If you swap the battery leads the motor spins in reverse.

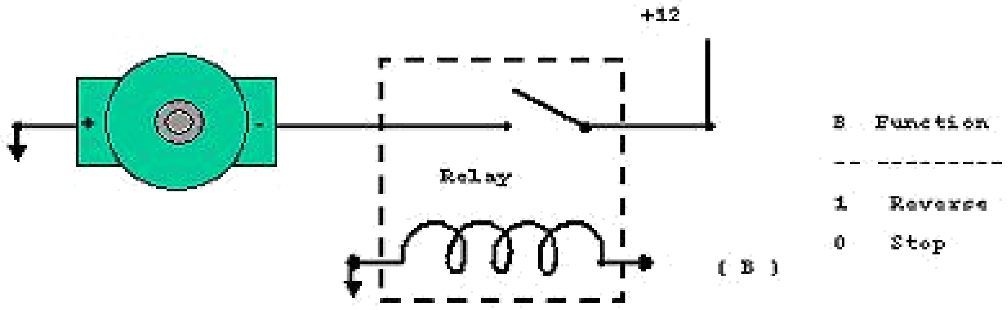
Ok, that's basic. Now lets say you want a Micro Controller Unit (MCU) to control the motor, how would you do it?



A	Function
--	---------
1	Forward
0	Stop (Coast)

Figure 15: Connections for clockwise rotation of motor

If you connect this circuit to a small hobby motor you can control the motor with a processor (MCU, etc.) Applying a logical one, (+12 Volts in our example) to point A causes the motor to turn forward. Applying a logical zero, (ground) causes the motor to stop turning (to coast and stop).



B	Function
--	---------
1	Reverse
0	Stop

Figure 16: Connections for anti clockwise rotation of motor

Hook the motor up in this fashion and the circuit turns the motor in reverse when you apply a logical one (+12Volts) to point B. Apply a logical zero, which is usually a ground, causes the motor to stop spinning.

If you hook up these circuits you can only get the motor to stop or turn in one direction, forward for the first circuit or reverse for the second circuit.

You can also pulse the motor control line, (A or B) on and off. This powers the motor in short burst and gets varying degrees of torque, which usually translates into variable motor speed. But if you want to be able to control the motor in both forward and reverse with a processor, you will need more circuitry. You will need an H-Bridge.

An H-bridge is an electronic circuit which enables a voltage to be applied across a load in either direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards and backwards. H-bridges are available as integrated circuits, or can be built from discrete components.

Let's start with the name, H-bridge. Sometimes called a "full bridge" the H-bridge is so named because it has four switching elements at the "corners" of the H and the motor forms the cross bar. The basic bridge is shown in the figure to the right.

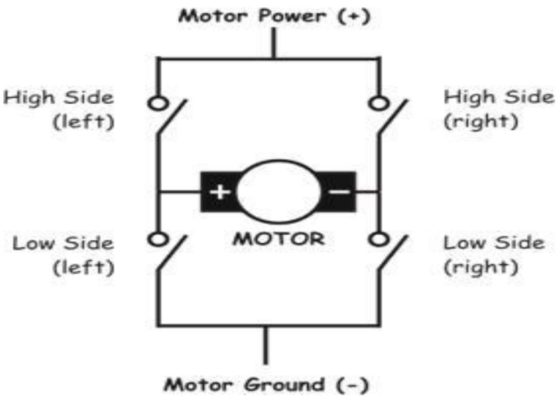
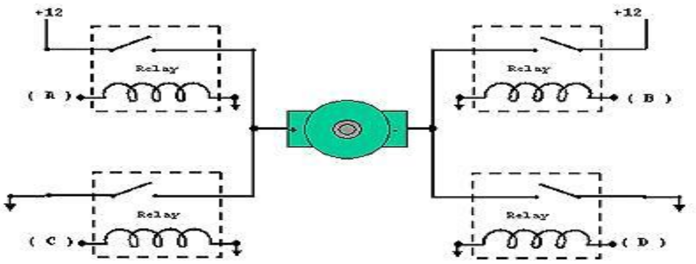


Figure 17: Basic H-bridge

The key fact to note is that there are, in theory, four switching elements within the bridge. These four elements are often called, high side left, high side right, low side right, and low side left (when traversing in clockwise order).

The "high side drivers" are the relays that control the positive voltage to the motor. This is called sourcing current. he "low side drivers" are the relays that control the negative voltage to sink current to the motor. "Sinking current" is the term for connecting the circuit to the negative side of the power supply, which is usually ground.



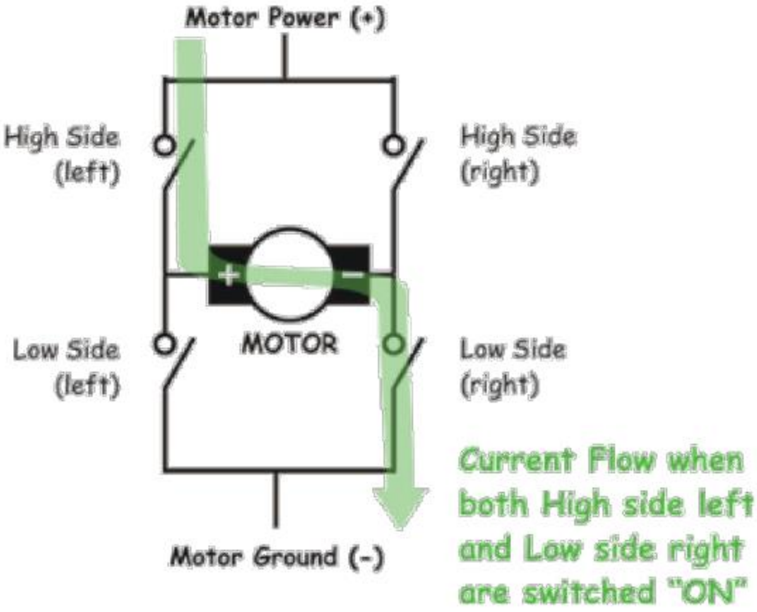
A	B	C	D	Function
1	0	0	1	Forward
0	1	1	0	Reverse
1	1	0	0	Brake
0	0	1	1	Brake
1	0	1	0	Fuse test :-)
0	1	0	1	Fuse test :-)

Don't do the fuse tests

Figure 18: Schematic diagram of H-Bridge using relay

The switches are turned on in pairs, either high left and lower right, or lower left and high right, but never both switches on the same "side" of the bridge. If both switches on one side of a bridge are turned on it creates a short circuit between the battery plus and battery minus terminals. This phenomena is called shoot through in the Switch-Mode Power Supply (SMPS) literature. If the bridge is sufficiently powerful it will absorb that load and your batteries will simply drain quickly .

To power the motor, you turn on two switches that are diagonally opposed. In the picture to the right, imagine that the high side left and low side right switches are turned on.



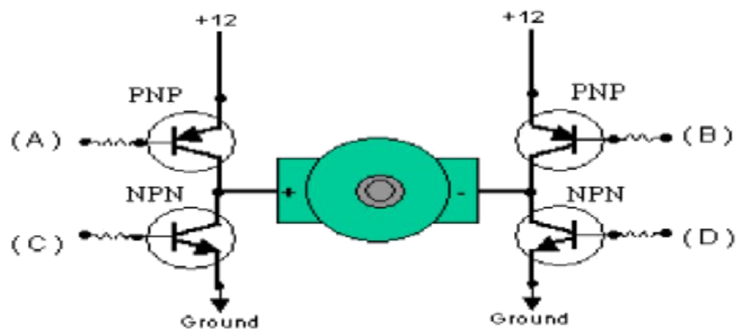
6.1.6 Working of basic bridge

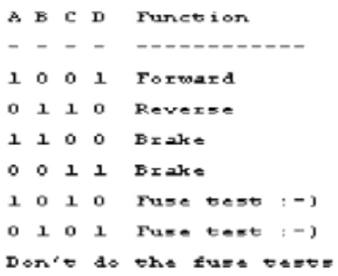
Then for reverse you turn on the upper right and lower left circuits and power flows through the motor in reverse.

6.1.6.1 Semiconductor H-Bridges

We can better control our motor by using transistors or Field Effect Transistors (FETs).Most of what we have discussed about the relays H-Bridge is true of these circuits. You don't need diodes that were across the relay coils now. You should use diodes across your transistors though. See the following diagram showing how they are connected.

These solid state circuits provide power and ground connections to the motor, as did the relay circuits. The high side drivers need to be current "sources" which is what PNP transistors and P-channel FETs are good at. The low side drivers need to be current "sinks" which is what NPN transistors and N-channel FETs are good at.





A	B	C	D	Function
1	0	0	1	Forward
0	1	1	0	Reverse
1	1	0	0	Brake
0	0	1	1	Brake
1	0	1	0	Fuse test :-)
0	1	0	1	Fuse test :-)

Don't do the fuse tests

Figure 19: Semiconductor H-Bridge

If you turn on the two upper circuits, the motor resists turning, so you effectively have a breaking mechanism. The same is true if you turn on both of the lower circuits. This is because the motor is a generator and when it turns it generates a voltage.

If the terminals of the motor are connected (shorted), then the voltage generated counteracts the motors freedom to turn. It is as if you are applying a similar but opposite voltage to the one generated by the motor being turned. Vis-ã-vis, it acts like a brake.

To be nice to your transistors, you should add diodes to catch the back voltage that is generated by the motor's coil when the power is switched on and off. This flyback voltage can be many times higher than the supply voltage! If you don't use diodes, you could burn out your transistors.

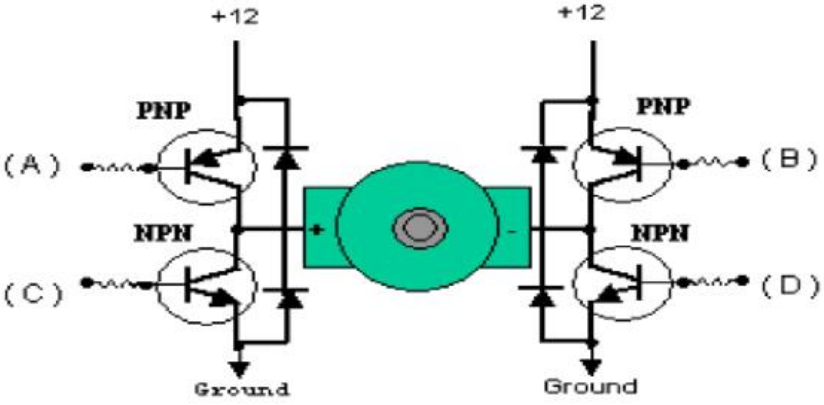


Figure 20: Semiconductor H-Bridge using diodes

Transistors, being a semiconductor device, will have some resistance, which causes them to get hot when conducting much current. This is called not being able to sink or source very much power, i.e.: Not able to provide much current from ground or from plus voltage.

Mosfets are much more efficient, they can provide much more current and not get as hot. They usually have the flyback diodes built in so you don't need the diodes anymore. This helps guard against flyback voltage frying your MCU.

To use Mosfets in an H-Bridge, you need P-Channel Mosfets on top because they can "source" power, and N-Channel Mosfets on the bottom because then can "sink" power. N-Channel Mosfets are much cheaper than P-Channel Mosfets, but N-Channel Mosfets used to source power require about 7 volts more than the supply voltage, to turn on. As a result, some people manage to use N-Channel Mosfets, on top of the H-Bridge, by using cleaver circuits to overcome the breakdown voltage.

It is important that the four quadrants of the H-Bridge circuits be turned on and off properly. When there is a path between the positive and ground side of the H-Bridge, other than through the motor, a condition exists called "shoot through". This is basically a direct short of the power supply and can cause semiconductors to become ballistic, in circuits with large currents flowing.

6.1.7 H-Bridge devices

The L 293 has 2 H-Bridges, can provide about 1 amp to each and occasional peak loads to 2 amps. Motors typically controlled with this controller are near the size of a 35 mm film plastic canister.

Figure 21: H-Bridge device

The L298 has 2 H-bridges on board, can handle 1amp and peak current draws to about 3amps. You often see motors between the size a of 35 mm film plastic canister and a coke can, driven by this type H-Bridge. The LMD18200 has one h-bridge on board, can handle about 2 or 3 amps and can handle a peak of about 6 amps. This H-Bridge chip can usually handle an average motor about the size of a coke. There are several more commercially designed H-Bridge chips as well.

6.1.8 DC Motors

Electric motors are everywhere! In your house, almost every mechanical movement that you see around you is caused by an AC (alternating current) or DC (direct current) electric motor. Let's start by looking at the overall plan of a simple two-pole DC electric motor. A simple motor has six parts, as shown in the diagram below:

- Armature or rotor
- Commutator
- Brushes
- Axle
- Field magnet
- DC power supply of some sort

An electric motor is all about magnets and magnetism: A motor uses **magnets** to create motion. If you have ever played with magnets you know about the fundamental law of all magnets: Opposites attract and likes repel. So if you have two bar magnets with their ends marked "north" and "south," then the north end of one magnet will attract the south end of the other. On the other hand, the north end of one magnet will repel the north end of the other (and similarly, south will repel south). Inside an electric motor, these attracting and repelling forces create **rotational motion**.

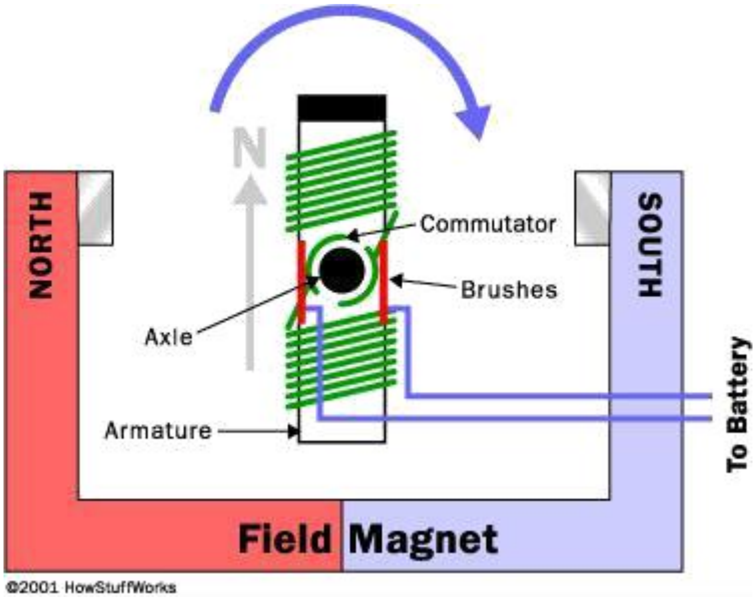


Figure 22: Principle of working of motor

In the above diagram, you can see two magnets in the motor: The armature (or rotor) is an electromagnet, while the field magnet is a permanent magnet (the field magnet could be an electromagnet as well, but in most small motors it isn't in order to save power).To understand how an electric motor works, the key is to understand how the electromagnet works. (See How Electromagnets Work for complete details.) An electromagnet is the basis of an electric motor. You can understand how things work in the motor by imagining the following scenario. Say that you created a simple electromagnet by wrapping 100 loops of wire around a nail and connecting it to a battery. The nail would become a magnet and have a north and south pole while the battery is connected.

Now say that you take your nail electromagnet, run an axle through the middle of it and suspend it in the middle of a horseshoe magnet as shown in the figure below. If you were to attach a battery to the electromagnet so that the north end of the nail appeared as shown, the basic law of magnetism tells you what would happen: The north end of the electromagnet would be repelled from the north end of the horseshoe magnet and attracted to the south end of the horseshoe magnet. The south end of the electromagnet would be repelled in a similar way. The nail would move about half a turn and then stop in the position shown.

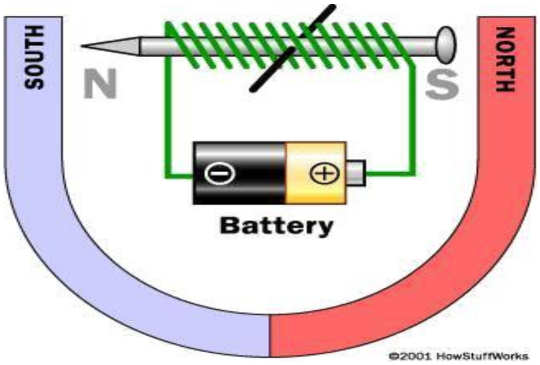


Figure 23: Electromagnet in Horse shoe magnet

You can see that this half-turn of motion is simply due to the way magnets naturally attract and repel one another. The key to an electric motor is to then go one step further so that, at the moment that this half-turn of motion completes, the field of the electromagnet flips.

The flip causes the electromagnet to complete another half-turn of motion. You flip the magnetic field just by changing the direction of the electrons flowing in the wire (you do that by flipping the battery over).

If the field of the electromagnet were flipped at precisely the right moment at the end of each half-turn of motion, the electric motor would spin freely.

6.1.8.1 Armature, Commutator and Brushes

Consider the image shown below. The armature takes the place of the nail in an electric motor. The armature is an electromagnet made by coiling thin wire around two or more poles of a metal core.

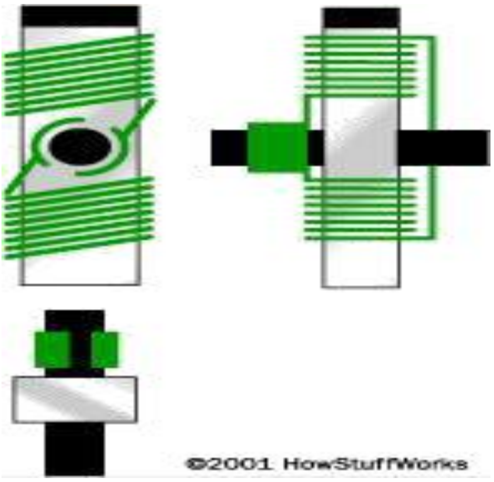


Figure 24: Armature

The armature has an **axle**, and the commutator is attached to the axle. In the diagram to the right, you can see three different views of the same armature: front, side and end-on. In the end-on view, the winding is eliminated to make the commutator more obvious. You can see that the commutator is simply a pair of plates attached to the axle. These plates provide the two connections for the coil of the electromagnet.

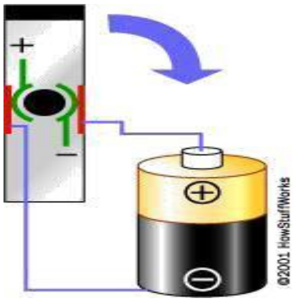


Figure 25: Brushes and commutator

The "flipping the electric field" part of an electric motor is accomplished by two parts: the **commutator** and the **brushes**.

Figure 25 shows how the commutator and brushes work together to let current flow to the electromagnet, and also to flip the direction that the electrons are flowing at just the right moment. The contacts of the commutator are attached to the axle of the electromagnet, so they spin with the magnet. The brushes are just two pieces of springy metal or carbon that make contact with the contacts of the commutator.

6.1.8.2 Parts of motor put together

When you put all of these parts together, what you have is a complete electric motor:

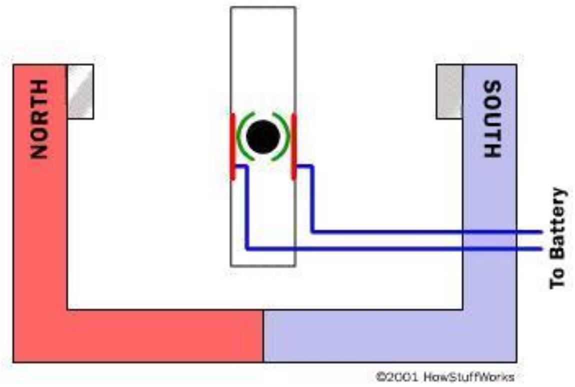


Figure 26: Commutator action

In this figure, the armature winding has been left out so that it is easier to see the commutator in action. The key thing to notice is that as the armature passes through the horizontal position, the poles of the electromagnet flip. Because of the flip, the north pole of the electromagnet is always above the axle so it can repel the field magnet's north pole and attract the field magnet's south pole.

If you ever have the chance to take apart a small electric motor, you will find that it contains the same pieces described above: two small permanent magnets, a commutator, two brushes, and an electromagnet made by winding wire around a piece of metal. Almost always, however, the rotor will have **three poles** rather than the two poles as shown in this article. There are two good reasons for a motor to have three poles:

It causes the motor to have better dynamics. In a two-pole motor, if the electromagnet is at the balance point, perfectly horizontal between the two poles of the field magnet when the motor starts, you can imagine the armature getting "stuck" there. That never happens in a three-pole motor.

Each time the commutator hits the point where it flips the field in a two-pole motor, the commutator shorts out the battery (directly connects the positive and negative terminals) for a moment. This shorting wastes energy and drains the battery needlessly. A three-pole motor solves this problem as well.

It is possible to have any number of poles, depending on the size of the motor and the specific application it is being used in.

The motor being dissected here is a simple electric motor that you would typically find in a toy:

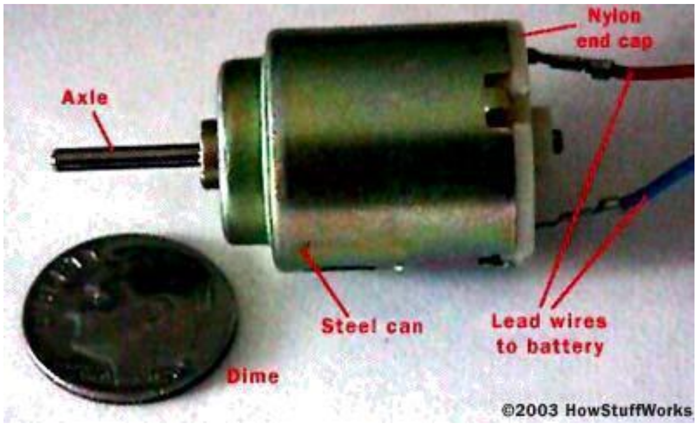


Figure 27: Dissected motor

You can see that this is a small motor, about as big around as a dime. From the outside you can see the steel can that forms the body of the motor, an axle, a nylon end cap and two battery leads. If you hook the battery leads of the motor up to a flashlight battery, the axle will spin. If you reverse the leads, it will spin in the opposite direction. Here are two other views of the same motor. (Note the two slots in the side of the steel can in the second shot -- their purpose will become more evident moment.)

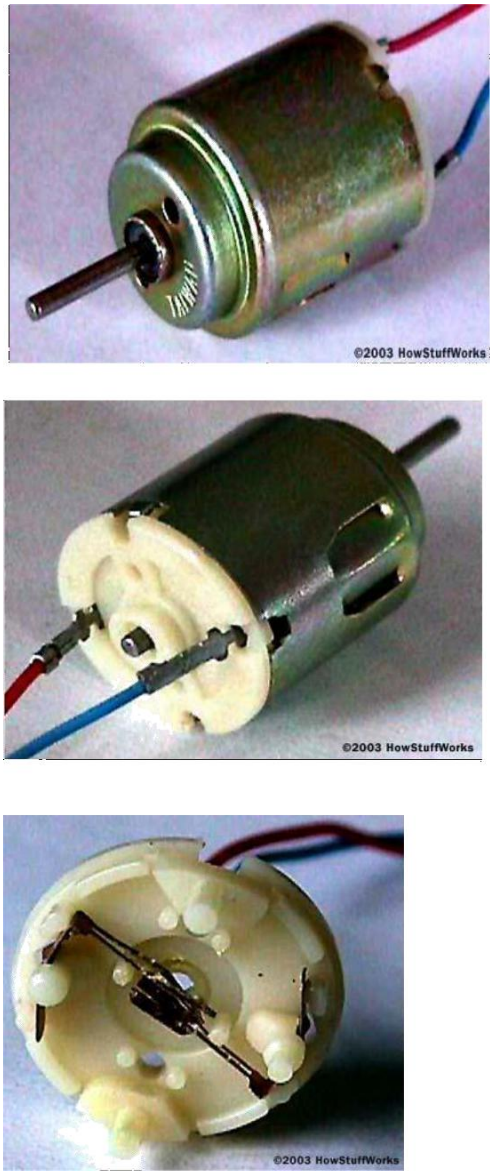


Figure 28: Nylon end caps

The nylon end cap is held in place by two tabs that are part of the steel can. By bending the tabs back, you can free the end cap and remove it. Inside the end cap are the motor's brushes. These brushes transfer power from the battery to the commutator as the motor spins.

6.1.8.3 More Motor Parts

The axle holds the armature and the commutator. The armature is a set of electromagnets, in this case three. The armature in this motor is a set of thin metal plates stacked together, with thin copper wire coiled around each of the three poles of the armature. The two ends of each wire (one wire for each pole) are soldered onto a terminal, and then each of the three terminals is wired to one plate of the commutator. The figures below make it easy to see the armature, terminals and commutator.

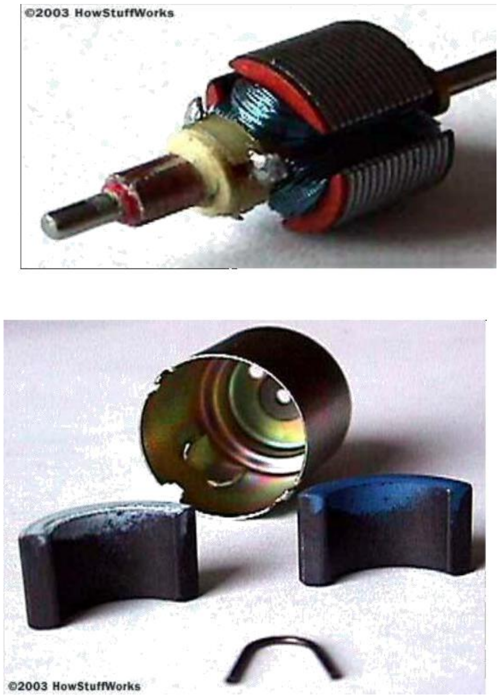


Figure 29: Other motor parts

.

CHAPTER-7

7.1 Discussion and Conclusion(software)

The results also showed that the gesture recognition application was quite robust for static images. However, the video version was enormously affected by the amount of illumination, such that is was necessary to check and adjust the HSV values for skin colour when starting the program to get the proper output. Sometimes the adjustment was difficult to do because of the lighting conditions and the amount of objects in the background.

The application was very susceptible to noise on the video stream. Slight hand movements could affect gesture recognition. Nevertheless, if the hand is steady enough for long enough, the program outputs the correct command.

It was also observed that while the program was executing there were memory leaks. Attempts to remedy the problem were made by using the OpenCV functions to release memory. Despite this, the leaks continued. Perhaps the leaks were due to the implementation of OpenCV functions for the sequences behind the scenes.

For integrating the program with the robot in the future, it would be necessarily to consider other output such as speed or velocity as part of the navigational control commands.

Based on the results, a computer vision application could detect and recognize simple hand gestures for robot navigation using simple heuristic rules. While the use of moment invariants was not considered suitable because the same gestures could be used pointing in opposite directions, other learning algorithms like AdaBoost could be explored to make the program more robust and less affected by extraneous objects and noise.

7. 2 Conclusion (Hardware)

The hand region is detected from the background by the background subtraction method. Then, the palm and fingers are segmented. On the basis of the segmentation, the fingers in the hand image are discovered and recognized. The recognition of hand gestures is accomplished by a simple rule classifier.

The performance of our method is evaluated on a data set of 1300 hand images. The experimental results show that our approach performs well and is fit for the real-time applications. Moreover, the proposed method outperforms the state-of-art FEMD on an image collection of hand gestures.

The performance of the proposed method highly depends on the result of hand detection. If there are moving objects with the color similar to that of the skin, the objects exist in the result of the hand detection and then degrade the performance of the hand gesture recognition.

However, the machine learning algorithms can discriminate the hand from the background. ToF cameras provide the depth information that can improve the performance of hand detection. So, in future works, machine learning methods and ToF cameras may be used to address the complex background problem and improve the robustness of hand detection.

APPENDIX

CODE:

```
#!/usr/bin/env python
Import cv2
Import numpy as np
Import math
Import time
Import serial
data=serial.serial(
„COM3‟,
baudrate=9600,
parity=serial.PARITY_NONE,
stopbits=serial.STOPBITS_ONE,
bytesize=serial.EIGHTBITS,
#(
                    Timeout=1 # must use when using data.readline()
)
cap = cv2.VideoCapture(0)
time.sleep(3)
#print(cap.isopened())
error=0
while(cap.isopened()):
#read image
Ret,img=cap.read()
# get hand data from the rectangle sub window on the
screen Cv2.rectangle(img,(300,300),(100,100), (0,255,0),0)
Crop_img=img[100:300, 100:300] #convert to grayscale
Grey=cv2.cvtcolor(crop_img, cv2.COLOR_BGR2GRAY)
# applying Gaussian blur
Value=( 35, 35)
```

```
  blurred = cv2.GaussianBlur(grey, value, 0)
  # thresholdin: Otsu's Binarization method_, thresh1 =
  cv2.threshold(blurred, 127, 255,
  cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
  # show thresholded image
  # cv2.imshow('Thresholded', thresh1)
  # check OpenCV version to avoid unpacking
  error (version, _, _) = cv2.__version__.split('.')
  ##   if version == '3':
  image, contours, hierarchy = cv2.findContours(thresh1.copy(), \
  cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
  ## elif version == '2': ##
  contours, hierarchy =
  cv2.findContours(thresh1.copy(),cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE
#   find contour with max area
  cnt = max(contours, key = lambda x: cv2.contourArea(x))
  # create bounding rectangle around the contour (can skip below two
  lines) x, y, w, h = cv2.boundingRect(cnt) cv2.rectangle(crop_img,
  (x, y), (x+w, y+h), (0, 0, 255), 0)
#   finding convex hull
  hull = cv2.convexHull(cnt)
  # drawing contours
  drawing = np.zeros(crop_img.shape,np.uint8)
  cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0)
  cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0)
  # finding convex hull
  hull = cv2.convexHull(cnt, returnPoints=False)
  # finding convexity defects
  defects = cv2.convexityDefects(cnt, hull)
  count_defects = 0
  cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)
```

```
    #   applying Cosine Rule to find angle for all defects (between fingers)
    #   with angle > 90 degrees and ignore defects
      for i in range(defects.shape[0]):
      s,e,f,d = defects[i,0]
      start = tuple(cnt[s][0])
      end = tuple(cnt[e][0])
      far = tuple(cnt[f][0])
      # find length of all sides of
      a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
      b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
      c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)
      # apply cosine rule here
      angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57
#     ignore angles > 90 and highlight rest with red dots
      if angle <= 90:
      count_defects += 1
          cv2.circle(crop_img, far, 1, [0,0,255], -1) dist
             = cv2.pointPolygonTest(cnt,far,True) #---
    #   draw a line from start to end i.e. the convex points (finger tips)
    #          (can skip this part)
      cv2.line(crop_img,start, end, [0,255,0], 2)
      cv2.circle(crop_img,far,5,[0,0,255],-1) #---
      error=1
      #   define actions required
      if count_defects == 1:
      data.write('1')
      cv2.putText(img,"1111111111111 ", (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
      2, 2)
      elif count_defects == 2:
      data.write('2')
      str = "2222222222222222222"
```

```
   cv2.putText(img, str, (5, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, 2)
   elif count_defects == 3:
   data.write('3')
   cv2.putText(img,"33333333333333333", (50, 50),
   cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
   elif count_defects == 4:
   data.write('4')
   cv2.putText(img,"444444444444444444", (50, 50),
   cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
   else:
   data.write('0')
   cv2.putText(img,"others", (50, 50),\
   cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
#  show appropriate images in windows
   cv2.imshow('Gesture', img)
   all_img = np.hstack((drawing, crop_img))
#       cv2.imshow('Contours', all_img)
#     k = cv2.waitKey(1)
if cv2.waitKey(1) & 0xFF == ord('q'):
   break
 #      if k == 4:
   #   cv2.destroyAllWindows()
        #   break
    if error==0:
   print('Camera interupted \nPlease exeute the script again')
```

REFERENCES:

1. A. D. Bagdanov, A. Del Bimbo, L. Seidenari, and L. Usai, “Real-time hand status recognition from RGB-D imagery,” in Proceedings of the 21st International Conference on Pattern Recognition (ICPR '12), pp. 2456–2459, November 2

2. M. Elmezain, A. Al-Hamadi, and B. Michaelis, “A robust method for hand gesture segmentation and recognition using forward spotting scheme in conditional random fields,” in Proceedings of the 20th International Conference on Pattern Recognition (ICPR '10), pp. 3850–3853, August 2010.

3. C.-S. Lee, S. Y. Chun, and S. W. Park, “Articulated hand configuration and rotation estimation using extended torus manifold embedding,” in Proceedings of the 21st International Conference on Pattern Recognition (ICPR '12), pp. 441– 444, November 2012.

4. M. R. Malgireddy, J. J. Corso, S. Setlur, V. Govindaraju, and D. Mandalapu, “A framework for hand gesture recognition and spotting using sub-gesture modeling,” in Proceedings of the 20th International Conference on Pattern Recognition (ICPR '10), pp. 3780–3783, August 2010.

5. P. Suryanarayan, A. Subramanian, and D. Mandalapu, “Dynamic hand pose recognition using depth data,” in Proceedings of the 20th International Conference on Pattern Recognition (ICPR '10), pp. 3105–3108, August 2010.