ACKNOWLEDGEMENTS

I take this opportunity to express my deep sense of gratitude and sincere thanks to all who helped me to complete the seminar successfully.

I am deeply indebted to my guide **Mr. Viki Prasad**, Assistant Professor, Department of Electrical and Electronics Engineering for her excellent guidance, positive criticism and valuable comments. I am also greatly thankful to **Mrs. Sindhu D Pillai**, Assistant Professor, Department of Electrical and Electronics Engineering, for her support and guidance to prepare and organise the presentation. I wholeheartedly thank my Senior Advisor **Mrs. M Deepthi**, Associate Professor, Department of Eelctrical and Electronics Engineering and **Dr. Imthias Ahamed T P**, Head of the Department for their support and cooperation.

Finally I thank my parents and friends who directly and indirectly contributed to the successful completion of my seminar.

APARNA R

Reg. No: 124030

ABSTRACT

A powered wheelchair is a mobility-aided device for persons with moderate/severe Physical disabilities or chronic diseases as well as the elderly. In order to take care for different disabilities, various kinds of interface have been developed for powered wheelchair control; such as joystick control, head control and sip-puff control. Many people with disabilities do not have the ability to control powered wheelchair using the above mentioned interfaces. We specifically aim at people suffering from quadriplegia. Quadriplegia is the paralysis caused by illness or injury that results in the partial or total loss of use of all four limbs and torso. Great people like Stephen Hawking and Max Brito have been suffering from this crippling phenomenon. Our project is an attempt to make lives of the people suffering from this phenomenon simple and by simpler we mean self-reliant, which will thereby reinstate their confidence and their happiness. The idea is to create an Eye Monitored System which allows movement of the patient's wheelchair depending on the eye movements. We have created a device where a patient sitting on the Wheel Chair assembly looking directly at the camera, is able to move in a direction just by looking in that direction. The camera signals are monitored by a Open CV script, which will then guide the motors wired to the AtMega1284P Microcontroller over the Serial Interface to move in a particular direction. The system is cost effective and thus can be used by patients spread over a large economy range.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Statistics suggests that there are 11,000 new cases of quadriplegia every year in United States alone. Tetraplegia, also known as quadriplegia, is paralysis caused by illness or injury that results in the partial or total loss of use of all four limbs and torso. Researchers suggest that this leads to a drop in the self-reliance of these people. We realized that this technology can intervene and help reinstate confidence of people suffering from Quadriplegia, by creating a medium, through which they can move at will. The complexity lies in the fact that they may be able to move only their eyes and partially their head.

We precisely aim at targeting the movements of the eye. The idea is to create an eye monitored wheelchair system where a camera constantly stares at the person's eyes and based on the combined movement of eye and head, it decides to move the wheelchair in the direction the person desires to move in.

The camera is wired to the person's laptop which has an OpenCV script running which constantly captures snapshots and processes them. The script based on the processing determines whether the person wants to move in a particular direction, and communicates this information using serial communication to a microcontroller which drives motors of the wheelchair in the desired direction.

The existing market products include wheel chair controlled by tongue, wheel chair controlled by brain signals and wheel chair controlled with joystick. The above mentioned products requires wires or sensors attached to the user or patient on a wheel chair and the one controlled by joystick is useful only for people who can move their fingers. Many other alternatives use complex or invasive methods to control wheel chair. This is not user friendly as well as might reduce the comfort level of the user. Our prototype uses movements of eye to control the wheelchair, using a webcam as sensor, is an advantage compared to other products. Hence it is an ultimate solution.

We aim to bring in a very user friendly method to control the wheelchair for patients suffering from injuries like paralysis. The product will help in bringing a better support to customers i.e. Patients who are suffering from full paralysis or quadriplegia like injury.

2. LITERATURE SURVEY

Studies show that many people like Stephen Hawking, Max Brito, etc. are suffering from various kinds of paralytic disorders. To make them self reliant, many developments have been put forward in the field of wheel chair manufacturing. As a result, various wheel chair control methods have been implemented like tongue control, gesture control, joystick control, voice operated, IR controlled, EMG controlled, EEG controlled, and so on. But all these types have shown many drawbacks such as discomfort to the patient using and some may have limited usage based on the control method used.

Tongue Operated Wheelchair

• Involves the use of magnet pierced or stuck on the tongue.

• The principle used here is to detect the variation in magnetic field produced when the tongue is moved using a sensor.

Voice Operated Wheelchair

• It involves a voice recognition module that contains a microphone.

• The principle used here is speech recognition which is used to control the wheelchair.

IR-Controlled Wheelchair

• It uses an eye-ball sensor to sense the motion of eye-ball.

• Here the movement of eyes is recognized using an infrared module.

Table 2.1 Drawbacks of various methods

METHODS	DRAWBACKS
Tongue operated wheelchair	Discomfort to the person, variation in magnetic field may not be precisely detected
Voice based methods	Less accuracy in: 1. Speed of speech 2. Background noise reduction
Eye – ball motion based, using IR	Affected by light intensity

EEG controlled wheelchair uses EEG signals extracted from brain for controlling action. From the EEG signals extracted, visual signals, called SSVEP signals, are separated and the patient is exposed to a particular colour. The control of wheelchair is done by sensing colour. A particular colour is set for the movement of wheelchair. The wheelchair moves according to the colour sensed. The major drawback of this method off wheelchair control is that the extraction of EEG signals is highly complicated and the sensing of colour varies from person to person.

Hence, we come forward with a new control method for wheelchair, that is, eye control. This method is mainly applicable to patients whose upper torso got paralysed or for those who are able to move their eyeballs only.

3. BLOCK DIAGRAM

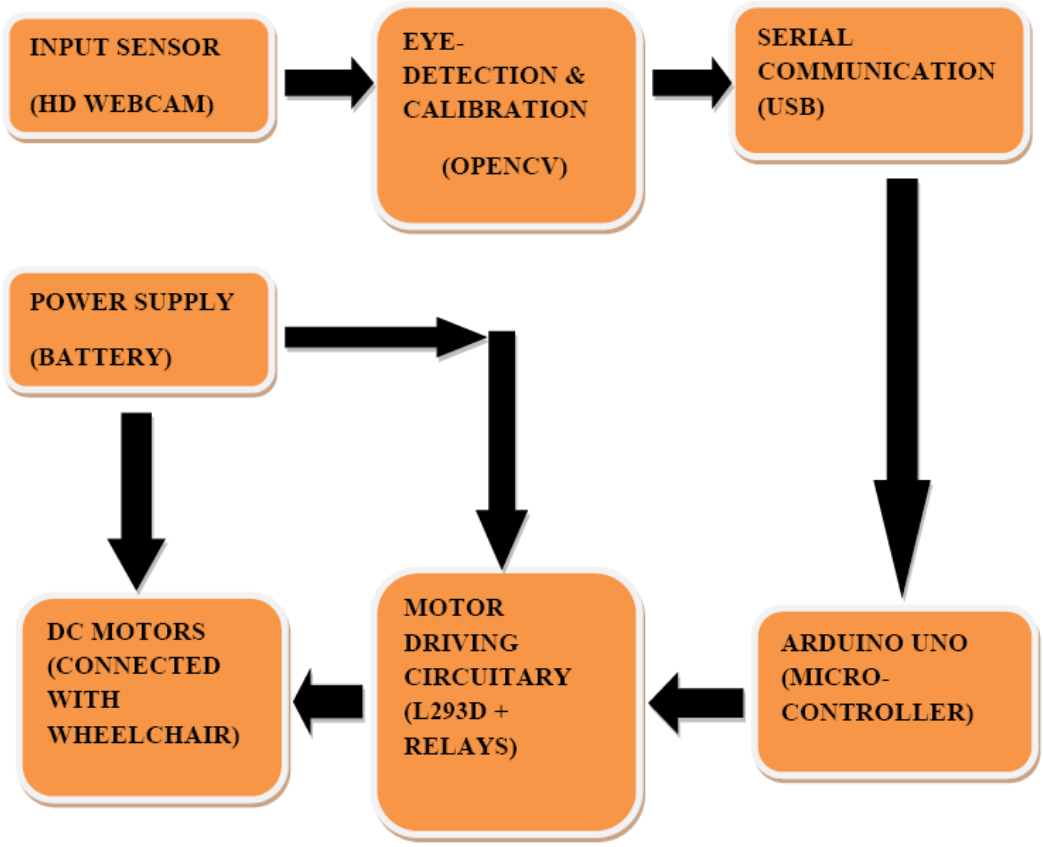

Fig.3.1. Block Diagram

3.1 BLOCK DIAGRAM EXPLANATION

3.1.1 Input Sensors

The input sensor that is used here is a 20 megapixel webcam from iBall (High Definition Camera) with a lighting system (using LED's), which functions depending on the lighting of the surrounding. This sensor is mounted on to a wearable cap.

3.1.2 Eye Detection and Calibration

OpenCV is an image processing software (an alternative for MATLAB). It is very fast, compared to MATLAB and efficient for real-time processing and applications. Here OpenCV is used for detecting the eyes using Hough Transform and the movement of eyes is analysed and a threshold is set for detecting four conditions. ie. Forward, Right, Left and Stop. A code corresponding for each of this movement is generated and is transmitted serially via a USB cable. In some cases MATLAB is not suitable for real-time processing as it requires a large RAM and might often get stuck. Hence OpenCV is a very good option.

3.1.3 Universal Serial Bus

We have used universal serial bus for serial communications from the OpenCV to the micro controller. USB 3.0 provides a fourth transfer mode with a data signalling rate of 5.0 GB/s, in addition to the modes supported by earlier versions. The payload throughput is 4 GB/s and the specification considers it reasonable to achieve around 3.2 GB/s (0.4 GB/s or 400 MB/s). Communication is full-duplex and this architecture is supported by most of the micro-controllers.

3.1.4 Arduino Uno

The Arduino Uno board utilizes an Atmel AVR 8-bit/16-bit/32-bit micro-controller (ATMEGA328P). The micro-controller is used for receiving the serial 4-bit or 8-bit data that is used to control the motors depending on the value of bits. This values are processed and conditions are set to drive the L293D (Motor Driver IC).

3.1.5 Motor Driving Circuitry

The motor driving circuitry consists of an L293D IC connected with 12v Relays which is connected with the power supply for the DC motors. The L293D utilizes an H-bridge circuitry.

3.1.6 DC motors

Here DC wiper motors (17W, 12V) were used to rotate the wheel of the chair. These motors are single directional and could take loads upto 100kg.

3.1.7 Power Supply

The power supply consists of an automotive Lead Acid battery (12v, 35Ah). This provides power supply necessary for the motors as well as the motor driving circuitry.

4. WORKING PRINCIPLE

First we process the captured image and after histogram equalization, we calibrate the subject's eye-position. This is kept as the default eye position and we continuously monitor the co-ordinate values of the eye position in the terminal window. We have created a circle to track the iris by detecting the border of the eyes by contrast difference calculation. We should calibrate the default position of the eyes such that our eyes are kept at the normal way and it actually coincides with the co-ordinates that we have set.

We calculate the position of eye from its center or default co-ordinates and hence detect straight, left, right and stop movements. Also other co-ordinates out of the x & y (North, East, West, South) axes are automatically assigned to pause the OpenCV code, eg. if the position of the eye is such that it is in North-West, the movement is paused for 5 seconds. The Arduino UNO will switch the wheel chairs motor to corresponding directions through the control of L293D.We have used a lot of functions to develop this code such as Hough Transform, Hough Circle, Gaussian Blur, Histogram Equalization, etc. which are the most important ones used in the algorithm.

4.1 Histogram Equalization:

This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

4.2 Hough Transform:

The Hough transform is a feature extraction technique used in image analysis, computer vision and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

4.3 Gaussian Blur:

A Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail in the image.

5. CIRCUIT DIAGRAM

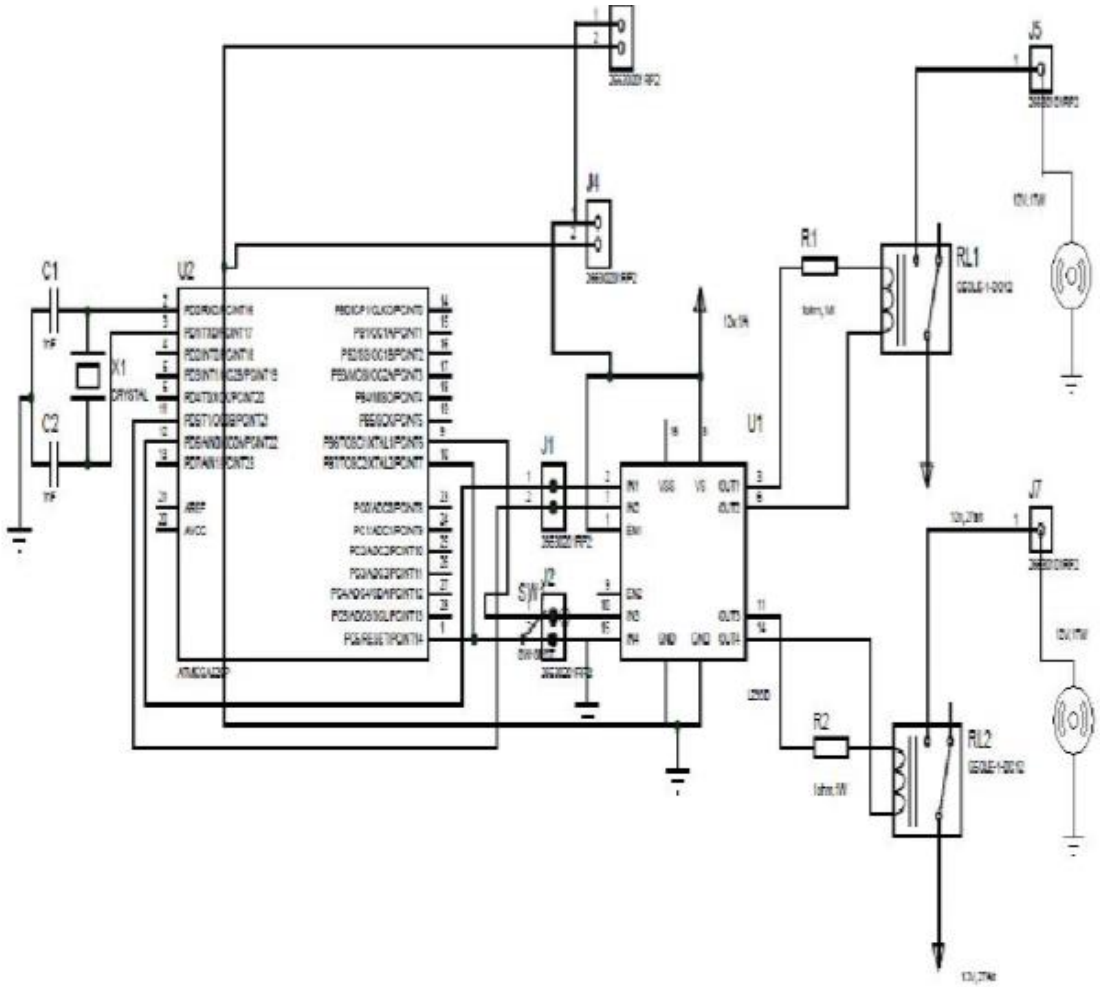

Fig. 5.1 Circuit Diagram

6. CIRCUIT DIAGRAM EXPLANATION

The circuit can be mainly divided into 2 different parts:

1. Controller Section

2. Motor Driver Section

6.1 CONTROLLER SECTION

The controller section consists of the micro-controller (ATMEGA328P) loaded with an hex file to give out digital high or low to the 4 input pins of L293D IC (H-bridge). The controller mainly performs the task depending on the conditions given to it.

Here we have programmed it to receive serial data via USB from OpenCV. The OpenCV gives serial 4-bit binary data to the Arduino UNO [ATMEGA328P] for three different conditions, which are

1. Straight – ASCII number: 49
2. Right – ASCII number: 50
3. Left – ASCII number: 51
4. Stop - ASCII number: 52

6.2 MOTOR DRIVER SECTION

The motor control circuitry is very simple in design, it consists of an L293D interfaced with the Arduino UNO board and 2 relays (12V, 5A). The L293D IC's are capable of tolerating up to 1A max, operating at a voltage of 12V. The L293D switches the power to the motor through the Relays and hence controlling them depending on the input from micro-controller.

Fig. a. Wheelchair used for control

7. HARDWARE IMPLEMENTATION

7.1 THE HEAD-MOUNT SENSOR MODULE

This module consists of a high definition webcam that is fixed on to a hard cap, making it a wearable and comfortable to wear device. The position of the webcam was decided on the basis of many trial and run calibration tests and it was fixed to the most suitable position (For easy detection of eye).

This webcam is approximately placed at20cm from the eye, it might have a slight variation from person to person. The webcam is movable in all angles and hence one can easily adjust it in such a way a clear picture of eye is obtained. The device is designed only to track any one of the eye and not both the eyes at once.

7.2 ARDUINO UNO BOARD

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable.

Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

Fig.7.1. The Arduino UNO (ATMEGA 328P)

The Uno is one of the more popular boards in the Arduino family and a great choice for the beginners. “Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. The Uno differs from all the preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

There are many varieties of Arduino boards that can be used for different purposes. Some boards look a bit different from the one another, but most Arduinos have the majority of these components in common.

7.2.1 POWER

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can

come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

V_{IN}: The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

5V: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board.

V_3: A 3.3 volt supply generated by the on-board regulator. Maximum current drawn is 50 mA.

GND: Ground pins.

IOREF: This pin on the Arduino board provides the voltage reference with which the microcontroller operates.

A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

7.2.2 MEMORY

The ATmega328 has 32 KB (with 0.5 KB used for the boot loader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

7.2.3 INPUT AND OUTPUT

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kΩ. In addition, some pins have specialized functions:

Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

LED: There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labelled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

AREF: Reference voltage for the analog inputs. Used with analogReference().

RESET: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

7.2.4 COMMUNICATION

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A Software Serial library allows for serial communication on any of the Uno's digital pins.The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

7.2.5 PROGRAMMING

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). The ATmega328 on the Arduino Uno comes returned with a boot loader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol.
You can also bypass the boot loader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header. The ATmega16U2 (or 8U2 in the rev1

and rev2 boards) firmware source code is available. The ATmega16U2/8U2 is loaded with a DFU boot loader, which can be activated by:

On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.

On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU boot loader). See this user-contributed tutorial for more information.

7.2.6 AUTOMATIC (SOFTWARE) RESET

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of theATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nano farad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip.

The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the boot loader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the boot loader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time

configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line.

7.2.7 USB OVER CURRENT PROTECTION

The Arduino Uno has a resetable poly fuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

7.2.8 PHYSICAL CHARACTERISTICS

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

7.2.9 SPECIFICATIONS

Table 7.1 Arduino UNO Specifications

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

7.3 MOTOR CONTROL CIRCUITRY

The motor control circuitry consists of relays and L293D motor driver IC. The L293D is a quadruple high-current half-H drivers. It can tolerate bi-directional drive currents of upto 1A at voltages from 4.5V to 36V.It provides bi-directional drive currents of upto 600-mA at voltages from4.5V to 36V.It is used to drive inductive loads such as relays, solenoids, dcand bi-polar stepping motors, as well as other high-current/high-voltage load sin positive-supply applications. All inputs are TTL compatible.

Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2 EN and drivers 3 and 4 enabled by 3,4 EN. When an enable input is

high, the associated drivers are enabled, and their outputs are active and in phase with their inputs.

When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable form or control application.

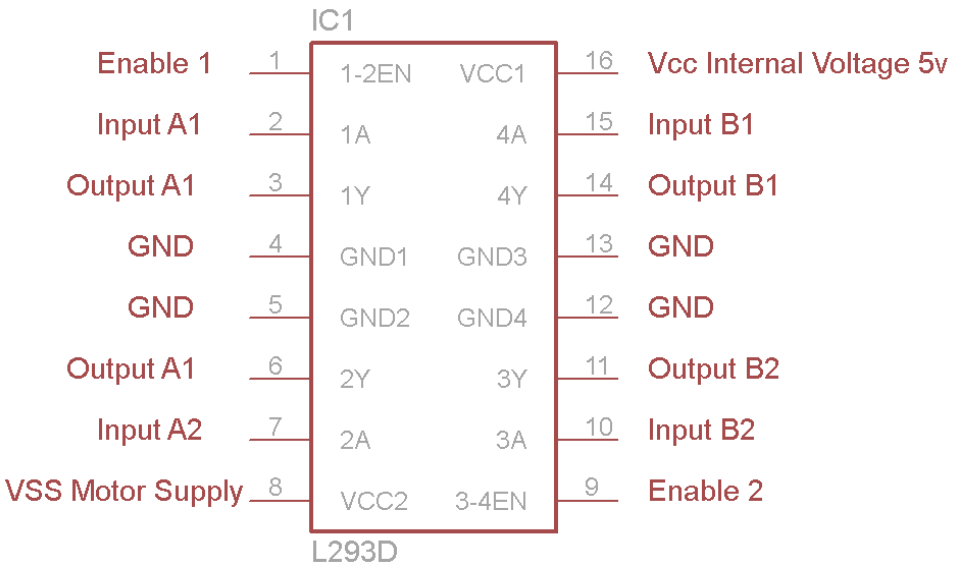

Fig.7.2. Pin out diagram of IC L293D

8. SOFTWARE IMPLEMENTATION

8.1. OPEN SOURCE COMPUTER VISION – OPENCV

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel Russia research center in Nizhny Novgorod, and now supported by Willow Garage and Itseez. It is free for use under the open-source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

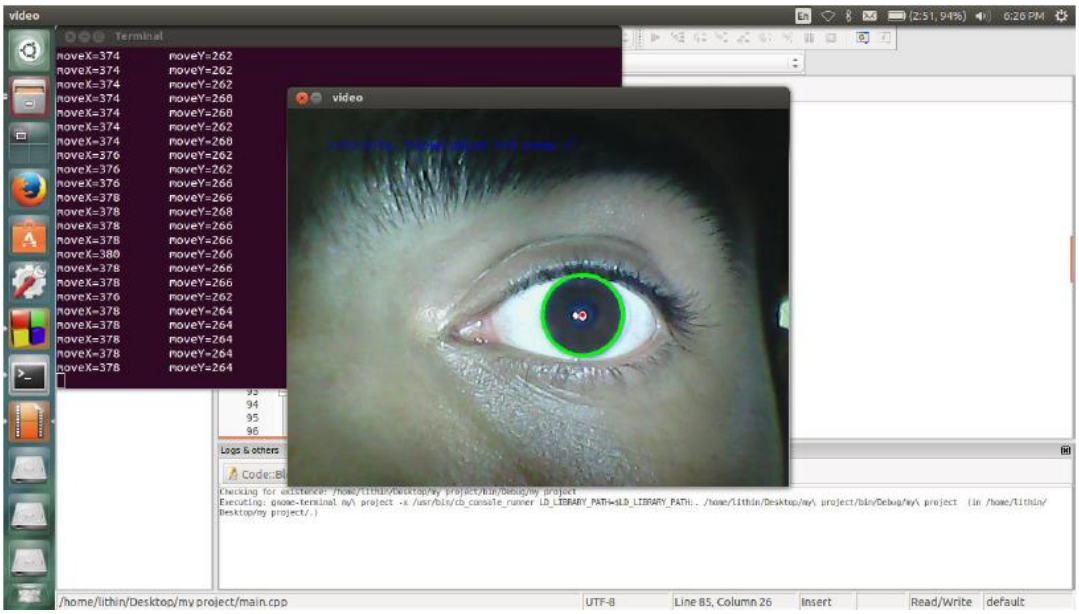

Fig.8.1. Eye Tracking in Open CV

➢ ***General description***

- Open source computer vision library in C/C++.
- Optimized and intended for real-time applications.
- OS/hardware/window-manager independent.
- Generic image/video loading, saving, and acquisition.
- Both low and high level API.

- Provides interface to Intel's Integrated Performance Primitives (IPP) with processor specific optimization (Intel processors).

➢ ***Features:***

- Image data manipulation (allocation, release, copying, setting, conversion).
- Image and video I/O (file and camera based input, image/video file output).
- Matrix and vector manipulation and linear algebra routines (products, solvers, eigen values, SVD).
- Various dynamic data structures (lists, queues, sets, trees, graphs).
- Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, color conversion, morphological operations, histograms, image pyramids).
- Structural analysis (connected components, contour processing, distance transform, various moments, template matching, Hough transform, polygonal approximation, line fitting, ellipse fitting, Delaunay triangulation).
- Camera calibration (finding and tracking calibration patterns, calibration, fundamental matrix estimation, homography estimation, stereo correspondence).
- Motion analysis (optical flow, motion segmentation, tracking).
- Object recognition (eigen-methods, HMM).
- Basic GUI (display image/video, keyboard and mouse handling, scroll-bars).
- Image labeling (line, conic, polygon, text drawing).

➢ ***OpenCV modules:***

- *cv* - Main OpenCV functions.
- *cvaux* - Auxiliary (experimental) OpenCV functions.
- *cxcore* - Data structures and linear algebra support.
- *highgui* - GUI functions.

Resources:

➢ Reference manuals:

- <opencv-root>/docs/index.htm

- Web resources:

- Official webpage: http://www.intel.com/technology/computing/opencv/
- Software download: http://sourceforge.net/projects/opencvlibrary/

- Books:

- Open Source Computer Vision Library by Gary R. Bradski, Vadim Pisarevsky, and Jean-Yves Bouguet, Springer, 1st ed. (June, 2006).

- Sample programs for video processing (in <opencv-root>/samples/c/):

- color tracking: camshiftdemo
- point tracking: lkdemo
- motion segmentation: motempl
- edge detection: laplace

- Sample programs for image processing (in <opencv-root>/samples/c/):

- edge detection: edge
- segmentation: pyramid_segmentation
- morphology: morphology
- histogram: demhist
- distance transform: distrans
- ellipse fitting: fitellipse

8.1.1 OPEN CV NAMING CONVENTIONS

- ***Function naming conventions:***

cvActionTargetMod(...)

Action = the core functionality (e.g. set, create)
Target = the target image area (e.g. contour, polygon)
Mod = optional modifiers (e.g. argument type)

- ***Matrix data types:***

CV_<bit_depth>(S|U|F)C<number_of_channels>

S = Signed integer

U = Unsigned integer

F = Float

- ***Image data types:***

IPL_DEPTH_<bit_depth>(S|U|F)

E.g.: IPL_DEPTH_8U means an 8-bit unsigned image.

IPL_DEPTH_32F means a 32-bit float image.

- ***Header files:***

#include <cv.h>

#include <cvaux.h>

#include <highgui.h>

#include <cxcore.h> // unnecessary - included in cv.h

8.1.2 COMPILATION INSTRUCTIONS

- ***Linux:***

g++ hello-world.cpp -o hello-world \

-I /usr/local/include/opencv -L /usr/local/lib \

-lm -lcv -lhighgui –lcvaux

- ***Windows:***

In the project preferences set the path to the OpenCV header files and the path to the OpenCV library files.

8.1.3 GUI COMMANDS

Window management

- Create and position a window:

```
cvNamedWindow("win1", CV_WINDOW_AUTOSIZE);
cvMoveWindow("win1", 100, 100); // offset from the UL corner of the screen
```

- Load an image:

```
IplImage* img=0;

img=cvLoadImage(fileName);
if(!img) printf("Could not load image file: %s\n",fileName);
```

- Display an image:

```
cvShowImage("win1",img);
```

Can display a color or grayscale byte/float-image. A byte image is assumed to have values in the range 0-255. A float image is assumed to have values in the range 0-1. A color image is assumed to have data in BGR order.

- Close a window:

```
cvDestroyWindow("win1");
```

- Resize a window:

```
cvResizeWindow("win1",100,100); // new width/height in pixels
```

8.1.4 INPUT HANDLING

- Handle mouse events:

  - Define a mouse handler:

```
void mouseHandler(int event, int x, int y, int flags, void* param)
 {
   switch(event)
    {
     case CV_EVENT_LBUTTONDOWN:
           if(flags & CV_EVENT_FLAG_CTRLKEY)
                 printf("Left button down with CTRL pressed\n");
               break;
     case CV_EVENT_LBUTTONUP:
       printf("Left button up\n");
     break;
     }
}
```

x,y: pixel coordinates with respect to the UL corner event:

CV_EVENT_LBUTTONDOWN, CV_EVENT_RBUTTONDOWN, CV_EVENT_MBUTTONDOWN,

CV_EVENT_LBUTTONUP, CV_EVENT_RBUTTONUP, CV_EVENT_MBUTTONUP,
CV_EVENT_LBUTTONDBLCLK,
CV_EVENT_RBUTTONDBLCLK, CV_EVENT_MBUTTONDBLCLK,
CV_EVENT_MOUSEMOVE:

flags: CV_EVENT_FLAG_CTRLKEY, CV_EVENT_FLAG_SHIFTKEY, CV_EVENT_FLAG_ALTKEY,
CV_EVENT_FLAG_LBUTTON, CV_EVENT_FLAG_RBUTTON, CV_EVENT_FLAG_MBUTTON

- Register the handler:

```
mouseParam=5;
cvSetMouseCallback("win1",mouseHandler,&mouseParam);
```

- Handle keyboard events:

- The keyboard does not have an event handler.
- Get keyboard input without blocking:

```
int key;
key=cvWaitKey(10); // wait 10ms for input
```

- Get keyboard input with blocking:

```
int key;
key=cvWaitKey(0); // wait indefinitely for input
```

- The main keyboard event loop:

```
while(1)
  { key=cvWaitKey(10);
      if(key==27) break;
          switch(key)
             {
               case 'h': ... break; case 'i': ... break;
              }
   }
```

- Handle trackbar events:

  - Define a trackbar handler:

```
void trackbarHandler(int pos)
    {
       printf("Trackbar position: %d\n",pos);
     }
```

  - Register the handler:

```
int trackbarVal=25;

int maxVal=100;

cvCreateTrackbar("bar1", "win1", &trackbarVal ,maxVal , trackbarHandler);
```

  - Get the current trackbar position:

```
int pos = cvGetTrackbarPos("bar1","win1");
```

  - Set the trackbar position:

```
cvSetTrackbarPos("bar1", "win1", 25);
```

8.1.5 BASIC OPEN CV DATA STRUCTURES

Image Data Structure

- IPL image:

IplImage

|-- int nChannels; // Number of color channels (1, 2, 3, 4)
|-- int depth; // Pixel depth in bits:

| // IPL_DEPTH_8U, IPL_DEPTH_8S,

| // IPL_DEPTH_16U, IPL_DEPTH_16S,

| // IPL_DEPTH_32S, IPL_DEPTH_32F,

| // IPL_DEPTH_64F

|-- int width; // image width in pixels
|-- int height; // image height in pixels

|-- char* imageData; // pointer to aligned image data

| // Note that color images are stored in BGR order

|-- int dataOrder; // 0 - interleaved color channels,

| // 1 - separate color channels

| // cvCreateImage can only create interleaved images

|-- int origin; // 0 - top-left origin,

| // 1 - bottom-left origin (Windows bitmaps style)

|-- int widthStep; // size of aligned image row in bytes
|-- int imageSize; // image data size in bytes = height*widthStep

|-- struct _IplROI *roi;// image ROI when not NULL specifies image

| // region to be processed.

|-- char *imageDataOrigin; // pointer to the unaligned origin of image data

| // (needed for correct image deallocation)

|-- int align; // Alignment of image rows: 4 or 8 byte alignment

| // OpenCV ignores this and uses widthStep instead

|-- char colorModel[4]; // Color model - ignored by OpenCV

Matrices and Vectors

- Matrices:

CvMat // 2D array
|-- int type; // elements type (uchar,short,int,float,double) and flags
|-- int step; // full row length in bytes
|-- int rows, cols; // dimensions

```
|-- int height, width;              // alternative dimensions reference
|-- union data;
 |-- uchar* ptr;                    // data pointer for an unsigned char matrix
|-- short* s;                       // data pointer for a short matrix
|-- int* i;                         // data pointer for an integer matrix
|-- float* fl;                      // data pointer for a float matrix
 |-- double* db;                    // data pointer for a double matrix
CvMatND                            // N-dimensional array
 |-- int type;                      // elements type (uchar,short,int,float,double) and flags
|-- int dims;                       // number of array dimensions
|-- union data;
 | |-- uchar* ptr;                  // data pointer for an unsigned char matrix
| |-- short* s;                     // data pointer for a short matrix
| |-- int* i;                       // data pointer for an integer matrix
| |-- float* fl;                    // data pointer for a float matrix
| |-- double* db;                   // data pointer for a double matrix
|
 |-- struct dim[];                  // information for each dimension
|-- size;                           // number of elements in a given dimension
 |-- step;                          // distance between elements in a given dimension
CvSparseMat                        // SPARSE N-dimensional array
```

- Generic arrays:

```
CvArr*              // Used only as a function parameter to specify that the
                    // function accepts arrays of more than a single type, such
                    // as: IplImage*, CvMat* or even CvSeq*. The particular array
                    // type is determined at runtime by analyzing the first 4
                    // bytes of the header of the actual array.
```

- Scalars:

```
CvScalar
 |-- double val[4];                 //4D vector
```

Initializer function:

CvScalar s = cvScalar(double val0, double val1=0, double val2=0, double val3=0);

Example:

```
CvScalar s = cvScalar(20.0);
 s.val[0]=10.0;
```

Note that the initializer function has the same name as the data structure only starting with a lower case character. It is not a C++ constructor.

Other Data Structures

- Points:

CvPoint p = cvPoint(int x, int y);

CvPoint2D32f p = cvPoint2D32f(float x, float y);

CvPoint3D32f p = cvPoint3D32f(float x, float y, float z);

E.g.:

```
p.x=5.0;
 p.y=5.0;
```

- Rectangular dimensions:

CvSize r = cvSize(int width, int height);

CvSize2D32f r = cvSize2D32f(float width, float height);

- Rectangular dimensions with offset:

CvRect r = cvRect(int x, int y, int width, int height);

8.1.6 WORKING WITH IMAGES

Allocating and Releasing Images

- Allocate an image:

IplImage* cvCreateImage(CvSize size, int depth, int channels);

size: cvSize(width,height);

depth: pixel depth in bits:

IPL_DEPTH_8U, IPL_DEPTH_8S,

IPL_DEPTH_16U, IPL_DEPTH_16S,

IPL_DEPTH_32S, IPL_DEPTH_32F, IPL_DEPTH_64F

channels: Number of channels per pixel. Can be 1, 2, 3 or 4. The channels are interleaved. The usual data layout of a color image is b0 g0 r0 b1 g1 r1 ...

- Release an image:

IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
cvReleaseImage(&img);

- Clone an image:

```
IplImage* img1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
IplImage* img2;
 img2=cvCloneImage(img1);
```

- Set/get the region of interest:

```
void cvSetImageROI(IplImage* image, CvRect rect);

void cvResetImageROI(IplImage* image);

vRect cvGetImageROI(const IplImage* image);
```

The majority of OpenCV functions support ROI.

Set/get the channel of interest:

```
void cvSetImageCOI(IplImage* image, int coi); // 0=all
int cvGetImageCOI(const IplImage* image);
```

The majority of OpenCV functions do NOT support COI.

Reading and Writing Images

- Reading an image from a file:

```
IplImage*
img=0;

img=cvLoadImage(fileName);

if(!img)
  printf("Could not load image file: %s\n",fileName);
```

Supported image formats: BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF, TIF

By default, the loaded image is forced to be a 3-channel color image. This default can be modified by using:

```
img=cvLoadImage(fileName,flag);
```

flag: >0 the loaded image is forced to be a 3-channel color image =0 the loaded image is forced to be a 1 channel grayscale image <0 the loaded image is loaded as is (with number of channels in the file).

- Writing an image to a file:

```
if(!cvSaveImage(outFileName,img))
  printf("Could not save: %s\n",outFileName);
```

The output file format is determined based on the file name extension.

Image Conversion

- Convert to a grayscale or color byte-image:

```
cvConvertImage(src, dst, flags=0);
src = float/byte grayscale/color image
dst = byte grayscale/color image

flags = CV_CVTIMG_FLIP (flip vertically)
        CV_CVTIMG_SWAP_RB (swap the R and B channels)
```

- Convert a color image to grayscale:

Using the OpenCV conversion:

```
cvCvtColor(cimg,gimg,CV_BGR2GRAY); // cimg -> gimg
```

- Convert between color spaces:

```
cvCvtColor(src,dst,code); // src -> dst
code = CV_<X>2<Y>
```

<X>/<Y> = RGB, BGR, GRAY, HSV, YCrCb, XYZ, Lab, Luv, HLS

Drawing Commands

- Draw a box:

// draw a box with red lines of width 1 between (100,100) and (200,200)
cvRectangle(img, cvPoint(100,100), cvPoint(200,200), cvScalar(255,0,0), 1);

- Draw a circle:

// draw a circle at (100,100) with a radius of 20. Use green lines of width 1
cvCircle(img, cvPoint(100,100), 20, cvScalar(0,255,0), 1);

- Draw a line segment:

// draw a green line of width 1 between (100,100) and (200,200)
cvLine(img, cvPoint(100,100), cvPoint(200,200), cvScalar(0,255,0), 1);

- Draw a set of polylines:

```
CvPoint curve1[]={10,10, 10,100, 100,100, 100,10};
CvPoint curve2[]={30,30, 30,130, 130,130, 130,30, 150,10};
CvPoint* curveArr[2]={curve1, curve2};
int nCurvePts[2]={4,5};
int nCurves=2;
 int isCurveClosed=1;
int lineWidth=1;

cvPolyLine(img,curveArr,nCurvePts,nCurves,isCurveClosed,cvScalar(0,255,255),line
Width);
```

- Draw a set of filled polygons:

```
cvFillPoly(img,curveArr,nCurvePts,nCurves,cvScalar(0,255,255));
```

- Add text:

```
CvFont font;
double hScale=1.0;
double vScale=1.0;
```

```
int lineWidth=1;
cvInitFont(&font,CV_FONT_HERSHEY_SIMPLEX|CV_FONT_ITALIC,
hScale,vScale,0,lineWidth);
cvPutText (img,"My comment",cvPoint(200,400), &font, cvScalar(255,255,0));
```

8.1.7 WORKING WITH MATRICES

Allocating and Releasing Matrices

- General:
  - OpenCV has a C interface to matrix operations. There are many alternatives that have a C++ interface (which is more convenient) and are as efficient as OpenCV.
  - Vectors are obtained in OpenCV as matrices having one of their dimensions as 1.
  - Matrices are stored row by row where each row has a 4 byte alignment.

- Allocate a matrix:

```
CvMat* cvCreateMat(int rows, int cols, int type);
```

type: Type of the matrix elements. Specified in form CV_<bit_depth>(S|U|F)C<number_of_channels>. E.g.: CV_8UC1 means an 8-bit unsigned single-channel matrix, CV_32SC2 means a 32-bit signed matrix with two channels.

Accessing Matrix Elements

- Assume that you need to access the (i,j) cell of a 2D float matrix.

- Indirect matrix element access:

```
cvmSet(M,i,j,2.0); // Set M(i,j)
t = cvmGet(M,i,j); // Get M(i,j)
```

- Direct matrix element access assuming a 4-byte alignment:

```
CvMat* M = cvCreateMat(4,4,CV_32FC1);
 int n = M->cols; float *data = M->data.fl;
data[i*n+j] = 3.0;
```

- Direct matrix element access assuming possible alignment gaps:

```
CvMat* M = cvCreateMat(4,4,CV_32FC1);
int step = M->step/sizeof(float); float *data = M->data.fl;

(data+i*step)[j] = 3.0;
```

- Direct matrix element access of an initialized matrix:

```
double a[16];
CvMat Ma = cvMat(3, 4, CV_64FC1, a);
 a[i*4+j] = 2.0; // Ma(i,j)=2.0;
```

8.1.8 WORKING WITH VIDEO SEQUENCES

Capturing a Frame From a Video Sequence

- OpenCV supports capturing images from a camera or a video file (AVI).
- Initializing capture from a camera:

```
CvCapture* capture = cvCaptureFromCAM(0);          // capture from video device #0
```

- Initializing capture from a file:

```
CvCapture* capture = cvCaptureFromAVI("infile.avi");
```

- Capturing a frame:

```
IplImage* img = 0;
```

```
if(!cvGrabFrame(capture)) {                    // capture a frame
printf("Could not grab a frame\n\7");
 exit(0);
 }

img=cvRetrieveFrame(capture);           // retrieve the captured frame
```

To obtain images from several cameras simultaneously, first grab an image from each camera. Retrieve the captured images after the grabbing is complete.

- Releasing the capture source:

```
cvReleaseCapture(&capture);
```

Note that the image captured by the device is allocated/released by the capture function. There is no need to release it explicitly.

Saving a Video File

- Initializing a video writer:

```
CvVideoWriter *writer = 0;

int isColor = 1;
int fps = 25;                                  // or 30
int frameW = 640;                             // 744 for firewire cameras
int frameH = 480;                              //480 for firewire cameras
writer= cvCreate Video Writer("out.avi",CV_FOURCC('P','I','M','1'),
                                fps,cvSize(frameW, frameH), is Color);
```

- Writing the video file:

```
IplImage* img = 0;
int nFrames = 50;
for(i=0;i<nFrames;i++){
```

```
cvGrabFrame(capture);                    // capture a frame
img=cvRetrieveFrame(capture);            //retrieve the captured frame
cvWriteFrame(writer,img);                // add the frame to the file
}
```

To view the captured frames during capture, add the following in the loop:

```
cvShowImage("mainWin", img);
key=cvWaitKey(20); // wait 20 ms
```

Note that without the 20[msec] delay the captured sequence is not displayed properly.

- Releasing the video writer:

```
cvReleaseVideoWriter(&writer);
```

9. CONCLUSION

Considering the work we have done so far, we have achieved:

- To make this a very user-friendly and comfortable product, we have done this to some extend by using iris detection and tracking as input. The next thing we have done is to make this project a low cost one so that it is available to people over a wide span of economy.

- This product was found to be above averege efficient in tracking eye and the movements were followed quickly as soon as the instructions were recieved, there was no delay. This system was compared with various other medically assistive technologies available in the market and proved to be better than them in many ways.

- The system could have been made much more better by implementing the system on a Digital Signal Processor Kit (Like Beaglebone, TMS320C6000 or Raspberry Pi) which would have avoided the PC/Laptop as processor hence making it more cost effective and conserving space.

- This work could be modified to make new innovating projects which involves the use of eye as a controlling input. Eg. Eye-Controlled Mouse Pointer or Keyboard for quadriplegics, etc..

10. FUTURE SCOPE

- The system could have been made much more better by implementing the system on a Digital Signal Processor Kit (Like Beaglebone, TMS320C6000 or Raspberry Pi) which would have avoided the PC/Laptop as processor hence making it more cost effective and conserving space.
- This work could be modified to make new innovating projects which involves the use of eye as a controlling input. Eg. Eye-Controlled Mouse Pointer or Keyboard for quadriplegics, etc.
- Movements towards the North-East, North-West, South-East, South-West, reverse directions can also be programmed and added along with the current forward, right and left directions.

11.REFERENCES

[1] Ankur Thakkar, Dharshan Shah, "*Eye Monitered Wheelchair for People Suffering from Quadriplegia*", Cornell University, 2014

[2] Danish Chopra. "*Detection of Iris Movement by Estimating Sclera Size and Sync it with the Caster Wheel of a Wheelchair*".International Journal of Electronics Engineering, 2 (2), 2010, pp.283 – 286

[3] Moon, M. Lee, J. Chu, and M. Mun, "*Wearable EMG-based HCI for electric-powered wheelchair users with motor disabilities*," Proc. Of IEEE Int. Conf. on Robotics and Automation, pp. 2649–2654, 2005.

APPENDIX A

Program for Eye Detection and Tracking - OpenCV

```
#include </home/devi/opencv-2.4.9/modules/imgproc/include/opencv2/imgproc/imgproc.hpp>
#include </home/devi/opencv-2.4.9/modules/highgui/include/opencv2/highgui/highgui.hpp>
#include </home/devi/opencv-2.4.9/modules/objdetect/include/opencv2/objdetect/objdetect.hpp>
#include <iostream>
#include <stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>
using namespace std;
using namespace cv;
FILE *file;
#define waittime 5
void serialsend(int s)
// defining serial communication function
{
file=fopen("/dev/ttyACM0","w");
usleep(40);
fprintf(file,"%d",s);
usleep(40);
fclose(file);
}


void hough(Mat& frame, Mat& tpl, Rect& rect, int& x,int& y,int& z)
```

```
{
Mat src_gray;
cvtColor( frame, src_gray, COLOR_BGR2GRAY ); // color image to gray image
equalizeHist(src_gray,src_gray);// histogram equilization
GaussianBlur( src_gray, src_gray, Size(9, 9), 2, 2 );//image smoothning
vector<Vec3f> circles;
HoughCircles(src_gray,circles,3,1,20,100,30,40,70);//draw circle
if(circles.size())
{
Point center(cvRound(circles[0][0]), cvRound(circles[0][1]));
int radius = cvRound(circles[0][2]);
circle( frame, center, 3, Scalar(0,0,255), -1, 8, 0);
circle( frame, center, radius, Scalar(0,255,0), 3, 8, 0 );
x=cvRound(circles[0][0]);
y=cvRound(circles[0][1]);
z=cvRound(circles[0][2]);
if((x>z)&&(x<(frame.rows-z))&&(y>z)&&(y<(frame.cols-z)))
{
rect=Rect((circles[0][0]-radius),(circles[0][1]-radius),(2*radius),(2*radius));
tpl=src_gray(rect);
}
}
}
```

```
int main()
{
int X, Y;
VideoCapture cap(2);//video capture from webcam
namedWindow("video");
if (!cap.isOpened())
return 1;
Mat frame, eye_tpl;
```

```
Rect eye_bb;
int x=0,y=0,z=0;
char ch;
while ((ch=waitKey(15)) != 'c')
{
//printf("CHAr : %c",ch);
calib:
cap>> frame;
//press ‘c’ for calibrating eye position
if (frame.empty())
break;
flip(frame, frame, 0);
Mat gray,eyec;
cvtColor(frame, gray,COLOR_BGR2GRAY);
equalizeHist(gray,gray);
hough(frame,eye_tpl,eye_bb,x,y,z);
cout<<"moveX="<<x<<"\t"<<"moveY="<<y<<endl;//print coordinates of eye on terminal
putText(frame, "Calibrating...Please adjust and press 'C'",cv::Point(50,50),
CV_FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(255),1.5,8,false);
imshow("video", frame);//show video on terminal
X=x;
Y=y;
}

while((ch=waitKey(15))!=' ')
{
cap>> frame;
//press space key to start driving mode
if (frame.empty())
break;
flip(frame, frame, 0);
Mat gray,eyec;
cvtColor(frame, gray,COLOR_BGR2GRAY);
```

```
equalizeHist(gray,gray);
hough(frame,eye_tpl,eye_bb,x,y,z);
cout<<"moveX="<<x-X<<"\t"<<"moveY="<<y-Y<<endl;
putText(frame,  "Calibrated!",cv::Point(50,50),  CV_FONT_HERSHEY_SIMPLEX,
0.5,
cv::Scalar(255),1.5,8,false);
imshow("video", frame);
if((ch=waitKey(15)) == 'r')
goto calib;
}


while((ch=waitKey(15)) != 'q')
{
if((ch=waitKey(15)) == 'r') //press ‘r’ for recalibrating the eye
goto calib;
cap>> frame;
if(frame.empty())
break;
flip(frame, frame, 0);
Mat gray,eyec;
cvtColor(frame, gray,COLOR_BGR2GRAY);
equalizeHist(gray,gray);
hough(frame,eye_tpl,eye_bb,x,y,z);
cout<<"moveX="<<x-X<<"\t"<<"moveY="<<y-Y<<endl;
if(x-X <-20 && y-Y > -15 && y-Y < 12)
{
serialsend(2);
// send a bit serially to controller to rotate motor left
cout<<"Left\n";
putText(frame, "Left",cv::Point(50,50), CV_FONT_HERSHEY_SIMPLEX, 0.5,
cv::Scalar(255),1.5,8,false);
}
else if(x-X >20 && y-Y > -15 && y-Y < 12)
```

```
{
serialsend(3);
// send a bit serially to controller to rotate motor right
putText(frame, "Right",cv::Point(50,50), CV_FONT_HERSHEY_SIMPLEX,
0.5,cv::Scalar(255),1.5,8,false);
cout<<"Right\n";
}
else if(y-Y < -25 && x-X > -15 && x-X < 15)
{
serialsend(1); // send a bit serially to controller to rotate motor straight
putText(frame, "Straight",cv::Point(50,50), CV_FONT_HERSHEY_SIMPLEX,
0.5,cv::Scalar(255),1.5,8,false);
cout<<"Straight\n";
}
else if(y-Y > 15)
{
serialsend(4);
putText(frame, "Stop",cv::Point(50,50), CV_FONT_HERSHEY_SIMPLEX, 0.5,
cv::Scalar(255),1.5,8,false);
cout<<"Stop\n";
}
else if(x-X < -27 && y-Y <-27)
{
time_t t1 = time(0);
while(time(0)-t1 < waittime)
{
serialsend(4);
// pause the program for a predefined interval
cout<<"Entering inactive mode..will reset after"<<(int)waittime<<"seconds\n";
x=X;
y=Y;
}
cout<<"Quitting inactive mode\n";
}
```

```
imshow("video", frame);
}
return 0;
}
```

APPENDIX B

Program for Motor Control - Arduino UNO

```
#define mstraight 49
#define mleft 50
#define mright 51
#define mstop 52
void moveforward()
{
digitalWrite(12,0);
digitalWrite(11,1);
digitalWrite(10,0);
digitalWrite(9,1);
}
void moveleft()
{
digitalWrite(12,0);
digitalWrite(11,1);
digitalWrite(10,0);
digitalWrite(9,0);
}
void moveright()
{
digitalWrite(12,0);
digitalWrite(11,0);
digitalWrite(10,0);
digitalWrite(9,1);
}
void movestop()
{
digitalWrite(12,0);
digitalWrite(11,0);
digitalWrite(10,0);
digitalWrite(9,0);
```

```
}
void setup()
{
Serial.begin(9600);
pinMode(12,OUTPUT);
pinMode(11,OUTPUT);
pinMode(10,OUTPUT);
pinMode(9,OUTPUT);
pinMode(13,OUTPUT);
}
int count = 0;
char a = '\0';
void loop()
{
while(!Serial.available())
{
count++;
if(count > 40000)
{
movestop();
count = 0;
}
}
count = 0;
while(Serial.available())
{
a=Serial.read();
if(a == mleft)
moveleft();
else if(a == mright)
moveright();
else if(a == mstraight)
moveforward();
else if(a == mstop)
```

```
movestop();
}
}
/*
void loop()
{
while(!Serial.available());
while(Serial.available())
Serial.print(Serial.read());
Serial.println(' ');
}
*/
```