INDEX

FIGURE INDEX

ABSTRACT

Gestures play a major role in the daily activities of human life, especially during communication, providing an easy understanding. Gesture recognition refers to recognizing meaningful expressions of motion by a human, involving arms, hands, head/body. Between all the gestures hand gestures helps us to express more in less time. And moreover in today's developed Era the Human-machine interface has developed a lot mainly employing hand gestures. As in present controlling the home appliances using an infrared remote has been common and moreover it's not so different from using a remote to operate the appliances.

Here, in our project we propose an application for hand gesture recognition, of a limited set of hand gestures, for operating low voltage appliances used as a replacement for the actual home appliances. As of now the hand gesture recognition is tough in its own form. We have considered a fixed number of gestures and a reasonable environment in order to achieve the gesture detection and tried to produce a compelling way for gesture detection. Our approach contains steps for recognizing the hand region, contour extraction, locating the edges and counting the number of edges to recognize the gesture and finally implementing the corresponding action on the hardware. When we come to the hardware part which consists of a microcontroller which reads the data given by the hand gesture detection software through a communication module and the microcontroller takes the necessary action on the appliances.

1. INTRODUCTION

Home automation is the use and control of home appliances remotely or automatically. Home automation satisfies the resident's needs and desires by adjustable light, temperature, ambient music, automatic shading, safety & security, even arrangement of wire. Home automation technologies are the latest fascination with housing mechanism. However, with the appearance of new electronic technologies and their combination with older, traditional building technologies, the smart home is at last becoming a reality. The basic idea of home automation is to monitor a dwelling place by using sensors and control systems. Through adjustable various mechanisms, user can enjoy customized heat, ventilation, lighting, and other servers in living condition. The more closely adjust the entire living mechanical system and loop control system, the intelligent home can provide a safer, more comfortable, and more energy economical living condition.

1.1 Motivation Of The Project

The basic problems faced by disabled people in day-to-day life in their own house to turn ON or OFF the equipment's like lights, fans and difficulty in analysing switches are observed many times.[1] And the major issue being faced by the country is loss of power (power shortage).[2] This power shortage can be solved by two ways majorly; one way is from load shedding & second way is that people should be enlightened to switch OFF the appliances when not needed. Often it is observed that the street lights in cities are usually forgotten to be switched OFF during the day and this can be solved by taking initiatives in order to switch OFF the street lights during day time, and to save power. In order to overcome these problems, we design & develop of a smart home automation system

1.2 Aim Of The Project

In order to overcome the problems encountered, we design & develop of a smart home automation system which uses

(i) Hand gestures for disabled people to switch ON or OFF fans & lights or equipment's of the house, Here in this condition we are using camera to which live feed of hand gestures are given and from camera it goes to Central processor (Video processing system), Control signals from processing units are sent to relay, from relay different appliances can be controlled on given gestures.

(ii) Power saving capabilities to switch ON lights only in the presence of people, Here in this condition we are detecting when human is IN & OUT of the room from sensors, the signal from sensors is fed to microcontroller, then the control signal from the microcontroller is given to relay which controls the ON & OFF of the lights of the room respected to IN & OUT of humans.

(iii)Switch ON exterior lights based on light intensity, Here in this condition light intensities like dim, bright, are detected by the sensors & sent to microcontroller for processing, the processed signal is then fed to relay which in turn switch ON or OFF the lights based on intensities.

1.3 Methodology

The following methodology was followed during the project,

- Detailed study of Image Processing, MATLAB and Arduino tools.
- Study and selection of suitable Arduino board for interfacing and its programming.
- Design of suitable hardware required.
- Building up suitable simulation required.
- Adjusting live feedback from camera for simulation.
- Choosing suitable colour model in Image Processing.
- Calculating suitable values of dilation & erosion for images.
- Testing real time inputs like gestures and sensitivity from simulation model.
- Processing of model prepared.
- Real time testing and its validation.

2.BLOCK DIAGRAM :-

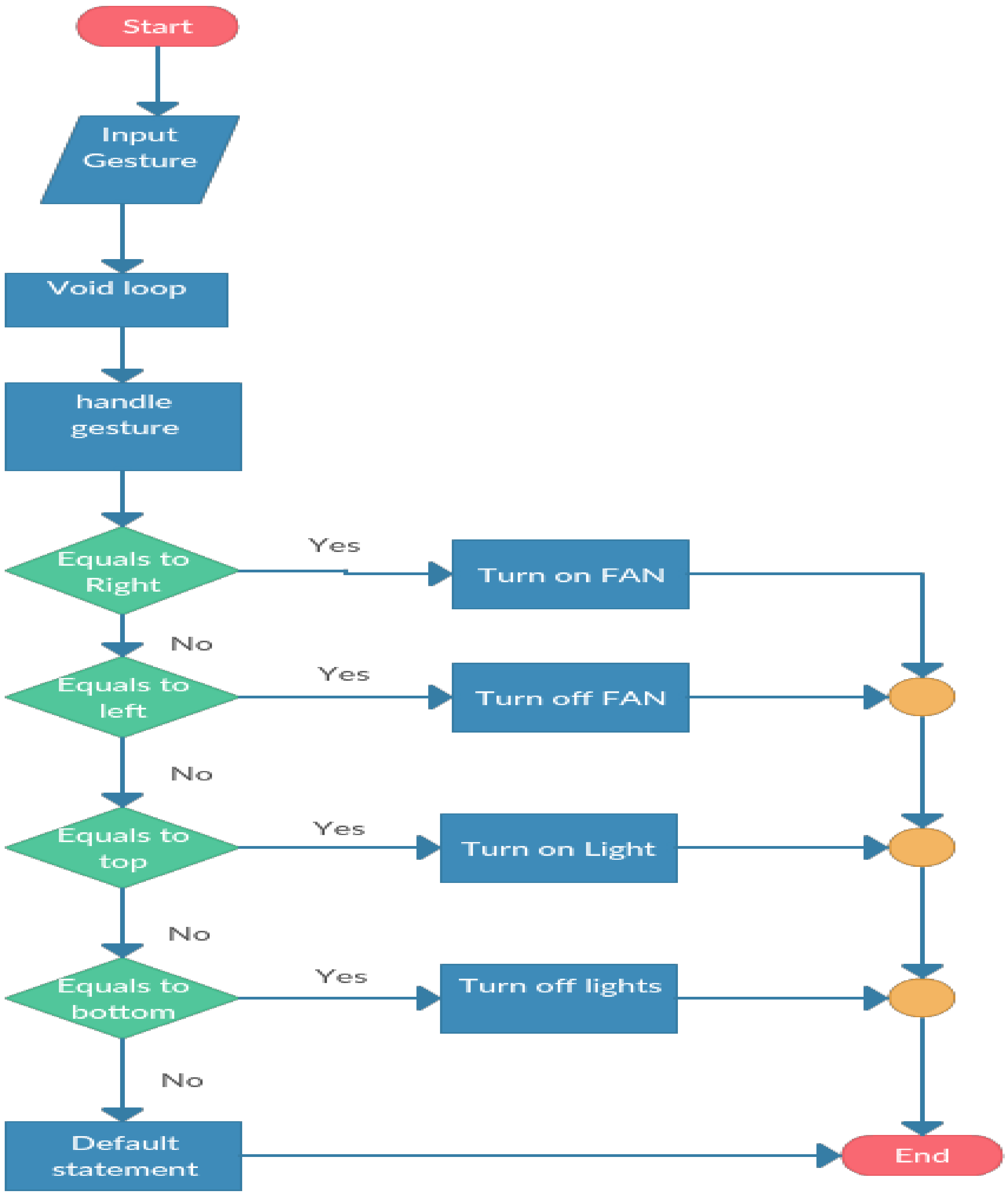


Fig.1 Flow Chart

3.FRONT END :

1. WHAT IS PYTHON ?

Python is a widely used general-purpose, high-language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

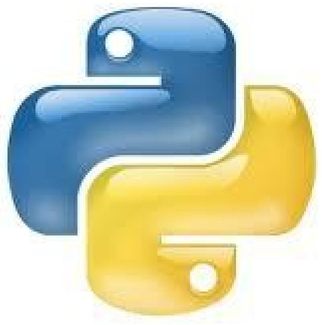


Fig 2 : Python

2. WHAT IS OPEN CV?

As explained in wiki,

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and It sees. It is free for use under the open source BSD license.
The library is cross-platform. It focuses mainly on real-time image processing.

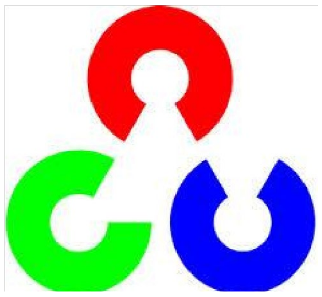


Fig 3 : OpenCv

WHY PYTHON IS USED FOR FRONT-END LANGUAGE ?

Reasons to Use Python for Front-End Web Development. Python is considered one of the best programming languages for web development because it's relatively easy to understand and it has a huge array of tools and functionalities. It's also extremely scalable and can carry out a wide range of outcomes.

FEACTURES OF PYTHON:

Python is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming. In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

Features in Python

There are many features in Python, some of which are discussed below as follows:

1. Easy to code: Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, JavaScript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

2. Free and Open Source: Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

3. Object-Oriented Language: One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

4. GUI Programming Support: Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

5. High-Level Language: Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

6. Extensible feature: Python is a Extensible language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

7. Python is Portable language: Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

8. Python is Integrated language: Python is also an Integrated language because we can easily integrated python with other languages like C, C++, etc.

9. Interpreted Language: Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called bytecode.

Platform: Python 2.7

Libraries : OpenCV (any version), NumPy, math, serial (from pyserial module)

Hardware Requirements : Camera/Webcam, Relays, Loads (like fan, bulb, etc), Resistors, Capacitors, Microcontroller, Clock (for frequency), serial cable, MAX232, Diodes and Connecting wires (male and female).

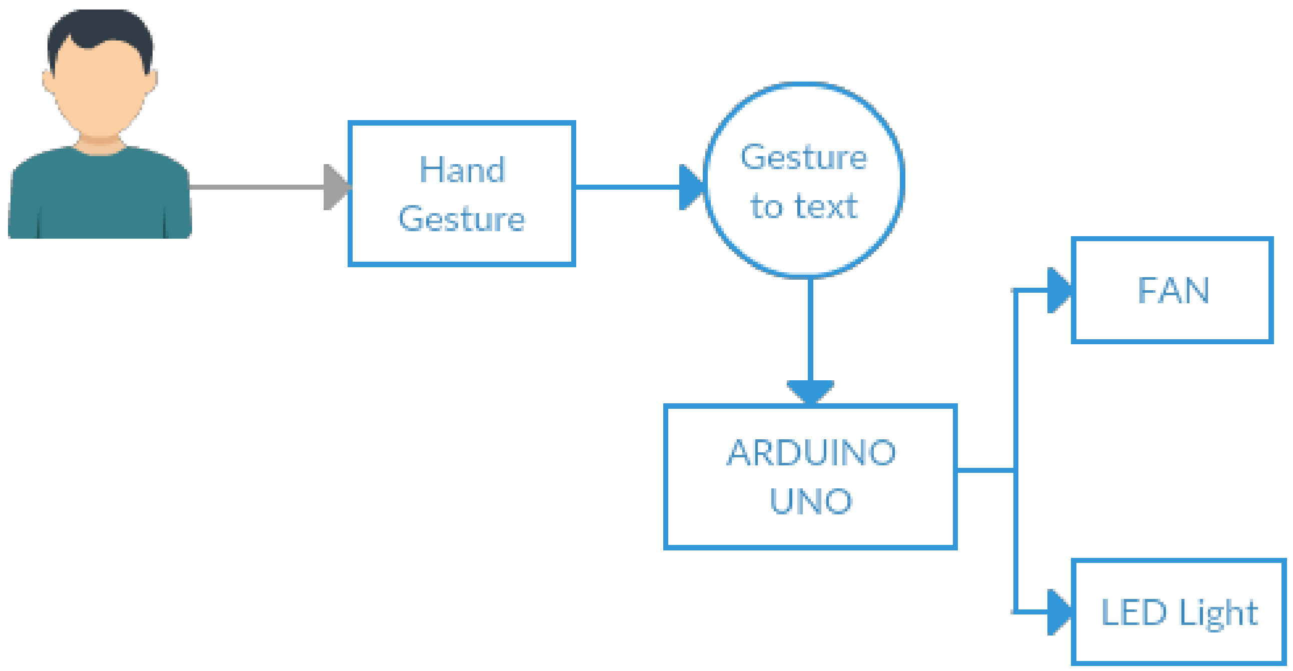


Fig 4 : Block Diagram

4.Installation procedure

1) First install Python 2.7. Leave all settings as default. In that case, Python will be installed in default folder C:\Python27\

2) Download PyCharm ide for running python programs. It is not a compulsory step, but it is one of the most easy way to work with python. This is because you can download modules directly from PyCharm>settings>Project >Project Interpreter>'plus sign'>install required module or package.

2) Now install NumPy module. Again leave everything default. NumPy will find Python directory and will be installed to most appropriate folder. Or else you can directly download it inside PyCharm.

3) Now double-click OpenCV.exe. It will ask for extraction folder. Give it as just C:\. It will extract all files to C:\opencv\ . Wait until everything is extracted.

4) Now copy everything in the folder C:\opencv\build\python\x86\2.7\ (most probably, there will be only one file cv2.pyd) and paste it in the folder C:\Python27\Lib\site-packages\

5) Now open your "Python IDLE" (from Start > All Programmes > Python 2.7 > Python IDLE) and just type following : import cv2

If everything OK, it will import cv2 module, otherwise an error message will be shown.

5.Source Code

```
import cv2

import mediapipe as mp

import time

import controller as cnt



time.sleep(2.0)



mp_draw=mp.solutions.drawing_utils

mp_hand=mp.solutions.hands



tipIds=[4,8,12,16,20]



video=cv2.VideoCapture(0)



with mp_hand.Hands(min_detection_confidence=0.5,

               min_tracking_confidence=0.5) as hands:

    while True:

        ret,image=video.read()

        image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
image.flags.writeable=False

results=hands.process(image)

image.flags.writeable=True

image=cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

lmList=[]

if results.multi_hand_landmarks:

    for hand_landmark in results.multi_hand_landmarks:

        myHands=results.multi_hand_landmarks[0]

        for id, lm in enumerate(myHands.landmark):

            h,w,c=image.shape

            cx,cy= int(lm.x*w), int(lm.y*h)

            lmList.append([id,cx,cy])

        mp_draw.draw_landmarks(image, hand_landmark, mp_hand.HAND_CONNECTIONS)

fingers=[]

if len(lmList)!=0:

    if lmList[tipIds[0]][1] > lmList[tipIds[0]-1][1]:

        fingers.append(1)

    else:

        fingers.append(0)

    for id in range(1,5):
        if lmList[tipIds[id]][2] < lmList[tipIds[id]-2][2]:

            fingers.append(1)

        else:

            fingers.append(0)

    total=fingers.count(1)

    cnt.led(total)

    if total==0:

        #cv2.rectangle(image, (20, 300), (270, 425), (0, 255, 0), cv2.FILLED)

        cv2.putText(image, "0", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,

            2, (255, 0, 0), 5)

        cv2.putText(image, "LED OFF", (100, 375), cv2.FONT_HERSHEY_SIMPLEX,

            2, (255, 0, 0), 2)

    elif total==1:

        #cv2.rectangle(image, (20, 300), (270, 425), (0, 255, 0), cv2.FILLED)

        cv2.putText(image, "1", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,

            2, (255, 0, 0), 5)

        cv2.putText(image, "LED ON", (100, 375), cv2.FONT_HERSHEY_SIMPLEX,

            2, (255, 0, 0), 2)


cv2.imshow("Frame",image)
```

```
        k=cv2.waitKey(1)

        if k==ord('q'):

            break

    video.release()

    cv2.destroyAllWindows()
```

To Transfer Output Through Serial Port To Hardware

CODE :

```
import pyfirmata
comport='COM7'
board=pyfirmata.Arduino(comport)
led_1=board.get_pin('d:13:o')
def led(total):
    if total==0:
        led_1.write(0)
    elif total==1:
        led_1.write(1)
```

IMPORT FOLLOWING CODE INTO ARDUINO UNO

STANDERD FIRMATA

CODE :

```
#include <Servo.h>

#include <Wire.h>

#include <Firmata.h>



#define I2C_WRITE                   B00000000

#define I2C_READ                    B00001000

#define I2C_READ_CONTINUOUSLY       B00010000

#define I2C_STOP_READING            B00011000

#define I2C_READ_WRITE_MODE_MASK    B00011000

#define I2C_10BIT_ADDRESS_MODE_MASK B00100000

#define I2C_END_TX_MASK             B01000000

#define I2C_STOP_TX                 1

#define I2C_RESTART_TX              0

#define I2C_MAX_QUERIES             8

#define I2C_REGISTER_NOT_SPECIFIED  -1
```

```
#define MINIMUM_SAMPLING_INTERVAL   1

#ifdef FIRMATA_SERIAL_FEATURE

SerialFirmata serialFeature;

#endif

int analogInputsToReport = 0; // bitwise array to store pin reporting

byte reportPINs[TOTAL_PORTS];       // 1 = report this port, 0 = silence

byte previousPINs[TOTAL_PORTS];     // previous 8 bits sent

byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else

unsigned long currentMillis;        // store the current value from millis()

unsigned long previousMillis;       // for comparison with currentMillis

unsigned int samplingInterval = 19; // how often to run the main loop (in ms)

struct i2c_device_info {

  byte addr;

  int reg;

  byte bytes;

  byte stopTX;

};
```

```
i2c_device_info query[I2C_MAX_QUERIES];

byte i2cRxData[64];

boolean isI2CEnabled = false;

signed char queryIndex = -1;

unsigned int i2cReadDelayTime = 0;

Servo servos[MAX_SERVOS];

byte servoPinMap[TOTAL_PINS];

byte detachedServos[MAX_SERVOS];

byte detachedServoCount = 0;

byte servoCount = 0;

boolean isResetting = false;

void setPinModeCallback(byte, int);

void reportAnalogCallback(byte analogPin, int value);

void sysexCallback(byte, byte, byte*);

void wireWrite(byte data)

{

#if ARDUINO >= 100
```

```
  Wire.write((byte)data);

#else

  Wire.send(data);

#endif

}

byte wireRead(void)

{

#if ARDUINO >= 100

  return Wire.read();

#else

  return Wire.receive();

#endif

}

void attachServo(byte pin, int minPulse, int maxPulse)

{

  if (servoCount < MAX_SERVOS) {

    if (detachedServoCount > 0) {
      servoPinMap[pin] = detachedServos[detachedServoCount - 1];
      if (detachedServoCount > 0) detachedServoCount--;
    } else {
      servoPinMap[pin] = servoCount;
      servoCount++;
    }
    if (minPulse > 0 && maxPulse > 0) {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
    } else {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin));
    }
  } else {
    Firmata.sendString("Max servos attached");
  }
}
void detachServo(byte pin)
{
  servos[servoPinMap[pin]].detach();
  if (servoPinMap[pin] == servoCount && servoCount > 0) {
    servoCount--;
  } else if (servoCount > 0) {
    detachedServoCount++;
    detachedServos[detachedServoCount - 1] = servoPinMap[pin];
  }
  servoPinMap[pin] = 255;
}
void enableI2CPins()
{
  byte i;
  for (i = 0; i < TOTAL_PINS; i++) {
    if (IS_PIN_I2C(i)) {
      setPinModeCallback(i, PIN_MODE_I2C);
    }
  }
  isI2CEnabled = true;

  Wire.begin();

}

void disableI2CPins() {

  isI2CEnabled = false;

  queryIndex = -1;

}

void readAndReportData(byte address, int theRegister, byte numBytes, byte stopTX) {

  if (theRegister != I2C_REGISTER_NOT_SPECIFIED) {

    Wire.beginTransmission(address);

    wireWrite((byte)theRegister);

    Wire.endTransmission(stopTX); // default = true

    if (i2cReadDelayTime > 0) {

      delayMicroseconds(i2cReadDelayTime);

    }

  } else {

    theRegister = 0;  // fill the register with a dummy value
  }

  Wire.requestFrom(address, numBytes);  // all bytes are returned in requestFrom

  if (numBytes < Wire.available()) {

    Firmata.sendString("I2C: Too many bytes received");

  } else if (numBytes > Wire.available()) {

    Firmata.sendString("I2C: Too few bytes received");

  }

  i2cRxData[0] = address;

  i2cRxData[1] = theRegister;



  for (int i = 0; i < numBytes && Wire.available(); i++) {

    i2cRxData[2 + i] = wireRead();

  }



  // send slave address, register and received bytes

  Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);

}
```

```
void outputPort(byte portNumber, byte portValue, byte forceSend)
{
  // pins not configured as INPUT are cleared to zeros
  portValue = portValue & portConfigInputs[portNumber];
  // only send if the value is different than previously sent
  if (forceSend || previousPINs[portNumber] != portValue) {
    Firmata.sendDigitalPort(portNumber, portValue);
    previousPINs[portNumber] = portValue;
  }
}
* check all the active digital inputs for change of state, then add any events
 * to the Serial output queue using Serial.print() */
void checkDigitalInputs(void)
{
  /* Using non-looping code allows constants to be given to readPort().
   * The compiler will apply substantial optimizations if the inputs
 * to readPort() are compile-time constants. */

if (TOTAL_PORTS > 0 && reportPINs[0]) outputPort(0, readPort(0, portConfigInputs[0]), false);

if (TOTAL_PORTS > 1 && reportPINs[1]) outputPort(1, readPort(1, portConfigInputs[1]), false);

if (TOTAL_PORTS > 2 && reportPINs[2]) outputPort(2, readPort(2, portConfigInputs[2]), false);

if (TOTAL_PORTS > 3 && reportPINs[3]) outputPort(3, readPort(3, portConfigInputs[3]), false);

if (TOTAL_PORTS > 4 && reportPINs[4]) outputPort(4, readPort(4, portConfigInputs[4]), false);

if (TOTAL_PORTS > 5 && reportPINs[5]) outputPort(5, readPort(5, portConfigInputs[5]), false);

if (TOTAL_PORTS > 6 && reportPINs[6]) outputPort(6, readPort(6, portConfigInputs[6]), false);

if (TOTAL_PORTS > 7 && reportPINs[7]) outputPort(7, readPort(7, portConfigInputs[7]), false);

if (TOTAL_PORTS > 8 && reportPINs[8]) outputPort(8, readPort(8, portConfigInputs[8]), false);

if (TOTAL_PORTS > 9 && reportPINs[9]) outputPort(9, readPort(9, portConfigInputs[9]), false);

if (TOTAL_PORTS > 10 && reportPINs[10]) outputPort(10, readPort(10, portConfigInputs[10]), false);

if (TOTAL_PORTS > 11 && reportPINs[11]) outputPort(11, readPort(11, portConfigInputs[11]), false);

if (TOTAL_PORTS > 12 && reportPINs[12]) outputPort(12, readPort(12, portConfigInputs[12]), false);

if (TOTAL_PORTS > 13 && reportPINs[13]) outputPort(13, readPort(13, portConfigInputs[13]), false);

if (TOTAL_PORTS > 14 && reportPINs[14]) outputPort(14, readPort(14, portConfigInputs[14]), false);

if (TOTAL_PORTS > 15 && reportPINs[15]) outputPort(15, readPort(15, portConfigInputs[15]), false);
```

```
   }

void setPinModeCallback(byte pin, int mode)

   {

    if (Firmata.getPinMode(pin) == PIN_MODE_IGNORE)

      return;



    if (Firmata.getPinMode(pin) == PIN_MODE_I2C && isI2CEnabled && mode != PIN_MODE_I2C) {

      // disable i2c so pins can be used for other functions

      // the following if statements should reconfigure the pins properly

      disableI2CPins();

    }

    if (IS_PIN_DIGITAL(pin) && mode != PIN_MODE_SERVO) {

      if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {

        detachServo(pin);

      }

    }

    if (IS_PIN_ANALOG(pin)) {
```

```
    reportAnalogCallback(PIN_TO_ANALOG(pin), mode == PIN_MODE_ANALOG ? 1 : 0); // turn on/off
reporting
  }
  if (IS_PIN_DIGITAL(pin)) {
    if (mode == INPUT || mode == PIN_MODE_PULLUP) {
      portConfigInputs[pin / 8] |= (1 << (pin & 7));
    } else {
      portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
    }
  }
  Firmata.setPinState(pin, 0);
  switch (mode) {
    case PIN_MODE_ANALOG:
      if (IS_PIN_ANALOG(pin)) {
        if (IS_PIN_DIGITAL(pin)) {
          pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
          // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
          digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        }
        Firmata.setPinMode(pin, PIN_MODE_ANALOG);
      }
      break;
    case INPUT:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
        // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
        digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        Firmata.setPinMode(pin, INPUT);
      }
      break;
    case PIN_MODE_PULLUP:
```

```
  if (IS_PIN_DIGITAL(pin)) {
    pinMode(PIN_TO_DIGITAL(pin), INPUT_PULLUP);
    Firmata.setPinMode(pin, PIN_MODE_PULLUP);
    Firmata.setPinState(pin, 1);
  }
  break;
case OUTPUT:
  if (IS_PIN_DIGITAL(pin)) {
    if (Firmata.getPinMode(pin) == PIN_MODE_PWM) {
      // Disable PWM if pin mode was previously set to PWM.
      digitalWrite(PIN_TO_DIGITAL(pin), LOW);
    }
    pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
    Firmata.setPinMode(pin, OUTPUT);
  }
  break;
case PIN_MODE_PWM:
  if (IS_PIN_PWM(pin)) {
    pinMode(PIN_TO_PWM(pin), OUTPUT);
    analogWrite(PIN_TO_PWM(pin), 0);
    Firmata.setPinMode(pin, PIN_MODE_PWM);
  }
  break;
case PIN_MODE_SERVO:
  if (IS_PIN_DIGITAL(pin)) {
    Firmata.setPinMode(pin, PIN_MODE_SERVO);
    if (servoPinMap[pin] == 255 || !servos[servoPinMap[pin]].attached()) {
      // pass -1 for min and max pulse values to use default values set
      // by Servo library
      attachServo(pin, -1, -1);
    }
  }
  break;
case PIN_MODE_I2C:
      if (IS_PIN_I2C(pin)) {
        // mark the pin as i2c
        // the user must call I2C_CONFIG to enable I2C for a device
        Firmata.setPinMode(pin, PIN_MODE_I2C);
      }
      break;
    case PIN_MODE_SERIAL:
#ifdef FIRMATA_SERIAL_FEATURE
      serialFeature.handlePinMode(pin, PIN_MODE_SERIAL);
#endif
      break;
    default:
      Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
  }
  // TODO: save status to EEPROM here, if changed
}
```

```
/*
 * Sets the value of an individual pin. Useful if you want to set a pin value but
 * are not tracking the digital port state.
 * Can only be used on pins configured as OUTPUT.
 * Cannot be used to enable pull-ups on Digital INPUT pins.
 */
void setPinValueCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS && IS_PIN_DIGITAL(pin)) {
    if (Firmata.getPinMode(pin) == OUTPUT) {
      Firmata.setPinState(pin, value);
      digitalWrite(PIN_TO_DIGITAL(pin), value);
    }
  }
}

void analogWriteCallback(byte pin, int value)
```

```
{
 if (pin < TOTAL_PINS) {
  switch (Firmata.getPinMode(pin)) {
   case PIN_MODE_SERVO:
    if (IS_PIN_DIGITAL(pin))
     servos[servoPinMap[pin]].write(value);
    Firmata.setPinState(pin, value);
    break;
   case PIN_MODE_PWM:
    if (IS_PIN_PWM(pin))
     analogWrite(PIN_TO_PWM(pin), value);
    Firmata.setPinState(pin, value);
    break;
  }
 }
}
```

```
void digitalWriteCallback(byte port, int value)

{

  byte pin, lastPin, pinValue, mask = 1, pinWriteMask = 0;



  if (port < TOTAL_PORTS) {

    // create a mask of the pins on this port that are writable.

    lastPin = port * 8 + 8;

    if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;

    for (pin = port * 8; pin < lastPin; pin++) {

      // do not disturb non-digital pins (eg, Rx & Tx)

      if (IS_PIN_DIGITAL(pin)) {

        // do not touch pins in PWM, ANALOG, SERVO or other modes

        if (Firmata.getPinMode(pin) == OUTPUT || Firmata.getPinMode(pin) == INPUT) {

          pinValue = ((byte)value & mask) ? 1 : 0;

          if (Firmata.getPinMode(pin) == OUTPUT) {

            pinWriteMask |= mask;

          } else if (Firmata.getPinMode(pin) == INPUT && pinValue == 1 && Firmata.getPinState(pin) != 1)
{
```

```
            // only handle INPUT here for backwards compatibility
      #if ARDUINO > 100
            pinMode(pin, INPUT_PULLUP);
      #else
            // only write to the INPUT pin to enable pullups if Arduino v1.0.0 or earlier
            pinWriteMask |= mask;
      #endif
           }
           Firmata.setPinState(pin, pinValue);
          }
         }
         mask = mask << 1;
        }
        writePort(port, (byte)value, pinWriteMask);
       }
      }
//void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
```

```
//}

void reportAnalogCallback(byte analogPin, int value)

{

  if (analogPin < TOTAL_ANALOG_PINS) {

    if (value == 0) {

      analogInputsToReport = analogInputsToReport & ~ (1 << analogPin);

    } else {

      analogInputsToReport = analogInputsToReport | (1 << analogPin);

      // prevent during system reset or all analog pin values will be reported

      // which may report noise for unconnected analog pins

      if (!isResetting) {

        // Send pin value immediately. This is helpful when connected via

        // ethernet, wi-fi or bluetooth so pin states can be known upon

        // reconnecting.

        Firmata.sendAnalog(analogPin, analogRead(analogPin));

      }

    }
  }
  // TODO: save status to EEPROM here, if changed
}


void reportDigitalCallback(byte port, int value)
{
  if (port < TOTAL_PORTS) {
    reportPINs[port] = (byte)value;
    // Send port value immediately. This is helpful when connected via
    // ethernet, wi-fi or bluetooth so pin states can be known upon
    // reconnecting.
    if (value) outputPort(port, readPort(port, portConfigInputs[port]), true);
  }
  // do not disable analog reporting on these 8 pins, to allow some
  // pins used for digital, others analog.  Instead, allow both types
  // of reporting to be enabled, but check if the pin is configured
  // as analog when sampling the analog inputs.  Likewise, while
  // scanning digital pins, portConfigInputs will mask off values from any

  // pins configured as analog

}

void sysexCallback(byte command, byte argc, byte *argv)

{

  byte mode;

  byte stopTX;

  byte slaveAddress;

  byte data;

  int slaveRegister;

  unsigned int delayTime;



  switch (command) {

    case I2C_REQUEST:

      mode = argv[1] & I2C_READ_WRITE_MODE_MASK;

      if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {

        Firmata.sendString("10-bit addressing not supported");
        return;

      }

      else {

        slaveAddress = argv[0];

      }



      // need to invert the logic here since 0 will be default for client

      // libraries that have not updated to add support for restart tx

      if (argv[1] & I2C_END_TX_MASK) {

        stopTX = I2C_RESTART_TX;

      }

      else {

        stopTX = I2C_STOP_TX; // default

      }



      switch (mode) {

        case I2C_WRITE:
```

```
  Wire.beginTransmission(slaveAddress);
  for (byte i = 2; i < argc; i += 2) {
    data = argv[i] + (argv[i + 1] << 7);
    wireWrite(data);
  }
  Wire.endTransmission();
  delayMicroseconds(70);
  break;
case I2C_READ:
  if (argc == 6) {
    // a slave register is specified
    slaveRegister = argv[2] + (argv[3] << 7);
    data = argv[4] + (argv[5] << 7);  // bytes to read
  }
  else {
    // a slave register is NOT specified
    slaveRegister = I2C_REGISTER_NOT_SPECIFIED;
```

```
      data = argv[2] + (argv[3] << 7);  // bytes to read

    }

    readAndReportData(slaveAddress, (int)slaveRegister, data, stopTX);

    break;

  case I2C_READ_CONTINUOUSLY:

    if ((queryIndex + 1) >= I2C_MAX_QUERIES) {

      // too many queries, just ignore

      Firmata.sendString("too many queries");

      break;

    }

    if (argc == 6) {

      // a slave register is specified

      slaveRegister = argv[2] + (argv[3] << 7);

      data = argv[4] + (argv[5] << 7);  // bytes to read

    }

    else {

      // a slave register is NOT specified
      slaveRegister = (int)I2C_REGISTER_NOT_SPECIFIED;
      data = argv[2] + (argv[3] << 7);  // bytes to read
    }
    queryIndex++;
    query[queryIndex].addr = slaveAddress;
    query[queryIndex].reg = slaveRegister;
    query[queryIndex].bytes = data;
    query[queryIndex].stopTX = stopTX;
    break;
  case I2C_STOP_READING:
    byte queryIndexToSkip;
    // if read continuous mode is enabled for only 1 i2c device, disable
    // read continuous reporting for that device
    if (queryIndex <= 0) {
      queryIndex = -1;
    } else {
      queryIndexToSkip = 0;
```

```
// if read continuous mode is enabled for multiple devices,
// determine which device to stop reading and remove it's data from
// the array, shifiting other array data to fill the space
for (byte i = 0; i < queryIndex + 1; i++) {
  if (query[i].addr == slaveAddress) {
    queryIndexToSkip = i;
    break;
  }
}

for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
  if (i < I2C_MAX_QUERIES) {
    query[i].addr = query[i + 1].addr;
    query[i].reg = query[i + 1].reg;
    query[i].bytes = query[i + 1].bytes;
    query[i].stopTX = query[i + 1].stopTX;
  }
```

```
        }
        queryIndex--;
      }
      break;
    default:
      break;
  }
  break;
case I2C_CONFIG:
  delayTime = (argv[0] + (argv[1] << 7));

  if (argc > 1 && delayTime > 0) {
    i2cReadDelayTime = delayTime;
  }

  if (!isI2CEnabled) {
    enableI2CPins();
  }


  break;

case SERVO_CONFIG:

  if (argc > 4) {

    // these vars are here for clarity, they'll optimized away by the compiler

    byte pin = argv[0];

    int minPulse = argv[1] + (argv[2] << 7);

    int maxPulse = argv[3] + (argv[4] << 7);


    if (IS_PIN_DIGITAL(pin)) {

      if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {

        detachServo(pin);

      }

      attachServo(pin, minPulse, maxPulse);

      setPinModeCallback(pin, PIN_MODE_SERVO);

    }
  }
  break;
case SAMPLING_INTERVAL:
  if (argc > 1) {
    samplingInterval = argv[0] + (argv[1] << 7);
    if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
      samplingInterval = MINIMUM_SAMPLING_INTERVAL;
    }
  } else {
    //Firmata.sendString("Not enough data");
  }
  break;
case EXTENDED_ANALOG:
  if (argc > 1) {
    int val = argv[1];
    if (argc > 2) val |= (argv[2] << 7);
    if (argc > 3) val |= (argv[3] << 14);
    analogWriteCallback(argv[0], val);
  }
  break;
case CAPABILITY_QUERY:
  Firmata.write(START_SYSEX);
  Firmata.write(CAPABILITY_RESPONSE);
  for (byte pin = 0; pin < TOTAL_PINS; pin++) {
    if (IS_PIN_DIGITAL(pin)) {
      Firmata.write((byte)INPUT);
      Firmata.write(1);
      Firmata.write((byte)PIN_MODE_PULLUP);
      Firmata.write(1);
      Firmata.write((byte)OUTPUT);
      Firmata.write(1);
    }
    if (IS_PIN_ANALOG(pin)) {
      Firmata.write(PIN_MODE_ANALOG);
```

```
          Firmata.write(10); // 10 = 10-bit resolution
        }
        if (IS_PIN_PWM(pin)) {
          Firmata.write(PIN_MODE_PWM);
          Firmata.write(DEFAULT_PWM_RESOLUTION);
        }
        if (IS_PIN_DIGITAL(pin)) {
          Firmata.write(PIN_MODE_SERVO);
          Firmata.write(14);
        }
        if (IS_PIN_I2C(pin)) {
          Firmata.write(PIN_MODE_I2C);
          Firmata.write(1);  // TODO: could assign a number to map to SCL or SDA
        }
#ifdef FIRMATA_SERIAL_FEATURE
        serialFeature.handleCapability(pin);
#endif
```

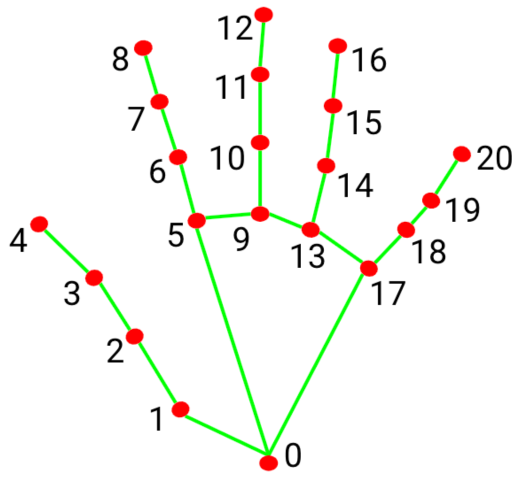
```
        Firmata.write(127);
      }
      Firmata.write(END_SYSEX);
      break;
    case PIN_STATE_QUERY:
      if (argc > 0) {
        byte pin = argv[0];
        Firmata.write(START_SYSEX);
        Firmata.write(PIN_STATE_RESPONSE);
        Firmata.write(pin);
        if (pin < TOTAL_PINS) {
          Firmata.write(Firmata.getPinMode(pin));
          Firmata.write((byte)Firmata.getPinState(pin) & 0x7F);
          if (Firmata.getPinState(pin) & 0xFF80) Firmata.write((byte)(Firmata.getPinState(pin) >> 7) & 0x7F);
          if (Firmata.getPinState(pin) & 0xC000) Firmata.write((byte)(Firmata.getPinState(pin) >> 14) &
0x7F);
        }
        Firmata.write(END_SYSEX);
      }
      break;
    case ANALOG_MAPPING_QUERY:
      Firmata.write(START_SYSEX);
      Firmata.write(ANALOG_MAPPING_RESPONSE);
      for (byte pin = 0; pin < TOTAL_PINS; pin++) {
        Firmata.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
      }
      Firmata.write(END_SYSEX);
      break;

    case SERIAL_MESSAGE:
#ifdef FIRMATA_SERIAL_FEATURE
      serialFeature.handleSysex(command, argc, argv);
#endif
      break;
  }
```

```
}

* SETUP()

void systemResetCallback()

{

  isResetting = true;



  // initialize a defalt state

  // TODO: option to load config from EEPROM instead of default

#ifdef FIRMATA_SERIAL_FEATURE

  serialFeature.reset();

#endif

  if (isI2CEnabled) {

    disableI2CPins();

  }

  for (byte i = 0; i < TOTAL_PORTS; i++) {

    reportPINs[i] = false;    // by default, reporting off

    portConfigInputs[i] = 0;  // until activated
  previousPINs[i] = 0;

}

for (byte i = 0; i < TOTAL_PINS; i++) {

  // pins with analog capability default to analog input

  // otherwise, pins default to digital output

  if (IS_PIN_ANALOG(i)) {

    // turns off pullup, configures everything

    setPinModeCallback(i, PIN_MODE_ANALOG);

  } else if (IS_PIN_DIGITAL(i)) {

    // sets the output to 0, configures portConfigInputs

    setPinModeCallback(i, OUTPUT);

  }

  servoPinMap[i] = 255;

}

// by default, do not report any analog inputs

analogInputsToReport = 0;
```

```
    detachedServoCount = 0;

    servoCount = 0;

TODO: this can never execute, since no pins default to digital input

        but it will be needed when/if we support EEPROM stored config

    for (byte i=0; i < TOTAL_PORTS; i++) {

      outputPort(i, readPort(i, portConfigInputs[i]), true);

    }

    */

    isResetting = false;

  }

  void setup()

  {

    Firmata.setFirmwareVersion(FIRMATA_FIRMWARE_MAJOR_VERSION,
  FIRMATA_FIRMWARE_MINOR_VERSION);



    Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);

    Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);

    Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);

Firmata.attach(SET_PIN_MODE, setPinModeCallback);

Firmata.attach(SET_DIGITAL_PIN_VALUE, setPinValueCallback);

Firmata.attach(START_SYSEX, sysexCallback);

Firmata.attach(SYSTEM_RESET, systemResetCallback);



// to use a port other than Serial, such as Serial1 on an Arduino Leonardo or Mega,

// Call begin(baud) on the alternate serial port and pass it to Firmata to begin like this:

// Serial1.begin(57600);

// Firmata.begin(Serial1);

// However do not do this if you are using SERIAL_MESSAGE



Firmata.begin(57600);

while (!Serial) {

  ; // wait for serial port to connect. Needed for ATmega32u4-based boards and Arduino 101

}
```

```
  systemResetCallback();  // reset to default config

}

void loop()

{

  byte pin, analogPin;

  checkDigitalInputs();

  while (Firmata.available())

    Firmata.processInput();

  // TODO - ensure that Stream buffer doesn't go over 60 bytes

  currentMillis = millis();

  if (currentMillis - previousMillis > samplingInterval) {

    previousMillis += samplingInterval;

    /* ANALOGREAD - do all analogReads() at the configured sampling interval */

    for (pin = 0; pin < TOTAL_PINS; pin++) {

      if (IS_PIN_ANALOG(pin) && Firmata.getPinMode(pin) == PIN_MODE_ANALOG) {

        analogPin = PIN_TO_ANALOG(pin);

        if (analogInputsToReport & (1 << analogPin)) {
          Firmata.sendAnalog(analogPin, analogRead(analogPin));
        }
      }
    }
    if (queryIndex > -1) {
      for (byte i = 0; i < queryIndex + 1; i++) {
        readAndReportData(query[i].addr, query[i].reg, query[i].bytes, query[i].stopTX);
      }
    }
  }
#ifdef FIRMATA_SERIAL_FEATURE
  serialFeature.update();
#endif
}
```

6.Hand Gesture Tracking :



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fig 5 : Hand Tracking

Hand Landmark Model

After the palm detection over the whole image our subsequent hand landmark model performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, we have manually annotated ~30K real-world images with 21 3D coordinates, as shown below (we take Z-value from image depth map, if it exists per corresponding coordinate). To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, we also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.

Gesture	COMMAND
One Finger	Fan ON
Two Finger	Fan OFF
Three Finger	Light ON
Four Finger	Light OFF

Python serial port communication

Please give the **COM number corresponding to your Serial port** or USB to Serial Converter instead of COM24.

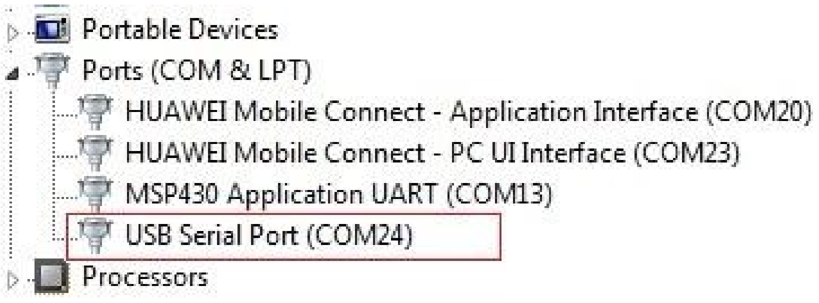

```
Portable Devices
Ports (COM & LPT)
    HUAWEI Mobile Connect - Application Interface (COM20)
    HUAWEI Mobile Connect - PC UI Interface (COM23)
    MSP430 Application UART (COM13)
    USB Serial Port (COM24)
Processors
```

Fig 6 : Port

The last part of the code takes user's input and convert if to an ascii string, and the 'write()' method sends it to the device. Note that carriage return characters '\r' and '\n' are appended to the end of the string such that the device knows when to perform the action.

7.HARDWARE

1. Introduction to Arduino UNO

The Arduino Uno is a type of Arduino board that is provided as an open-source board that uses an ATmega328p microcontroller in the board. The Arduino Uno contains a set of analog and digital pins that are input and output pins which are used to connect the board to other components. There are a total of fourteen I/O pins placed inboard in which six are analog input pins. The board has a USB connection that can be used to a power supply to the board. The board is used for electronics projects and used to design the circuit.

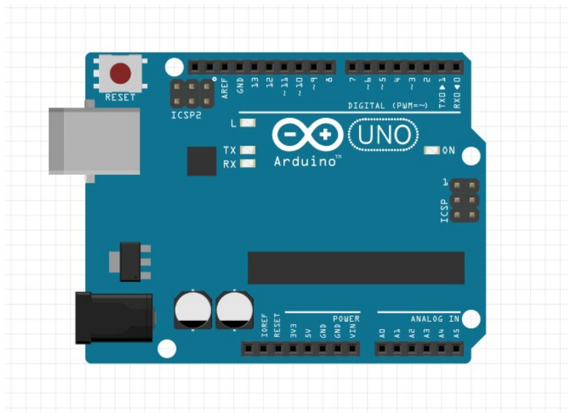


Fig 7: Arduino UNO

What is Arduino UNO?

The Arduino UNO is categorized as a microcontroller that uses the ATmega328 as a controller in it. The Arduino UNO board is used for an electronics project and mostly preferred by the beginners. The Arduino UNO board I type of Arduino board only. The Arduino board is the most used board of all Arduino boards. The board contains 14 digital input/ output pins in which 6 are analog input pin, one power jack, USB connector, one reset button, ICSP header, and other components. All these components are attached in the Arduino UNO board to make it functioning and can be used in the project. The board is charged by USB port or can be directly charged by the DC supply to the board

2. 5V Relay Module

The relay module with a single channel board is used to manage high voltage, current loads like solenoid valves, motor, AC load & lamps. This module is mainly designed to interface through different microcontrollers like PIC, Arduino, etc.

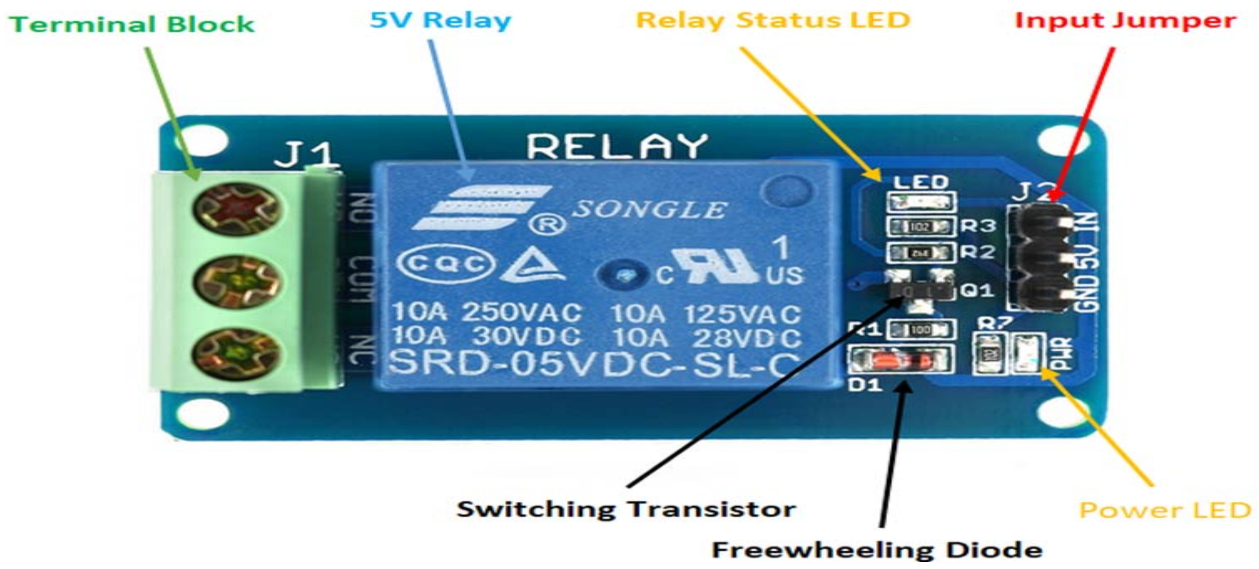


Fig 8 : 5V Relay Module

CONNECTION :

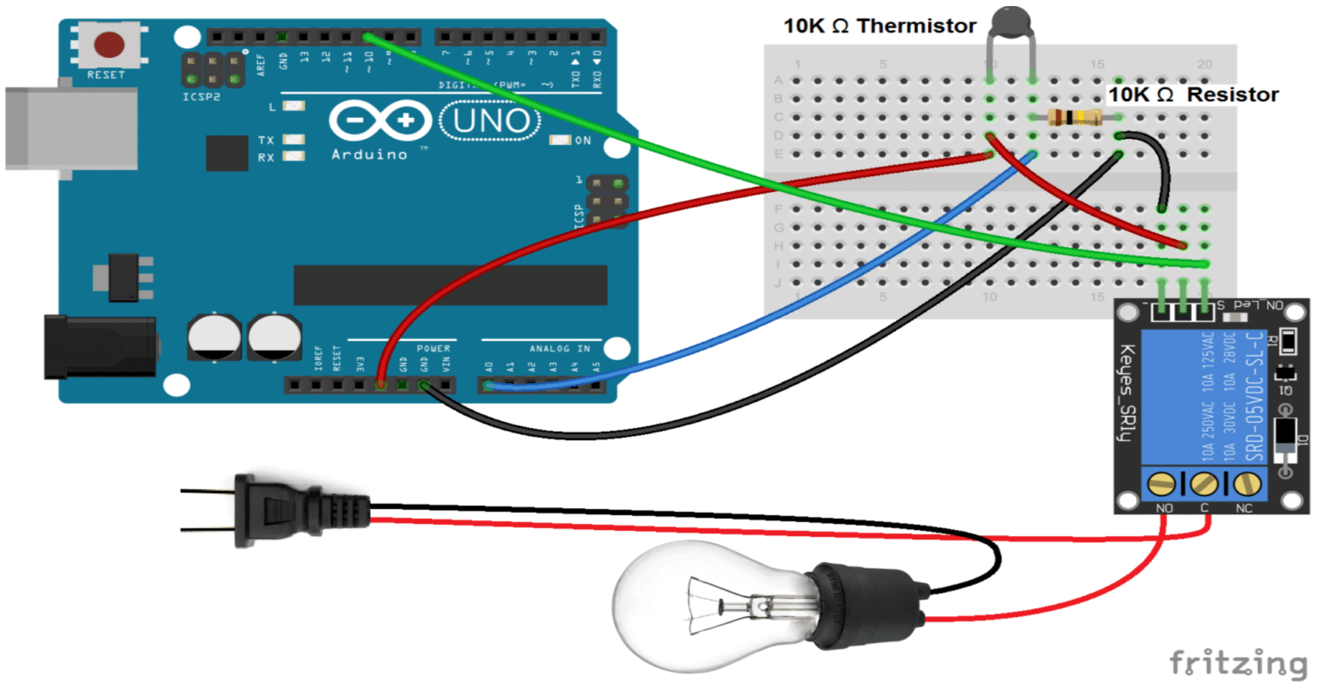
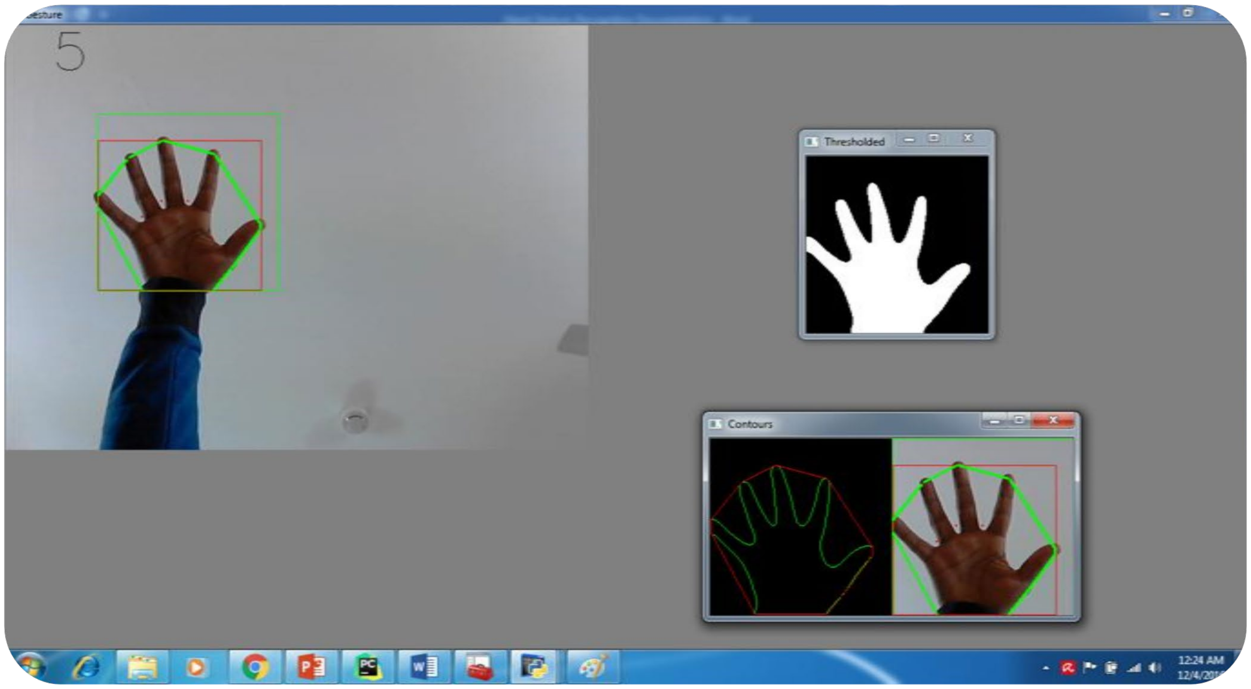
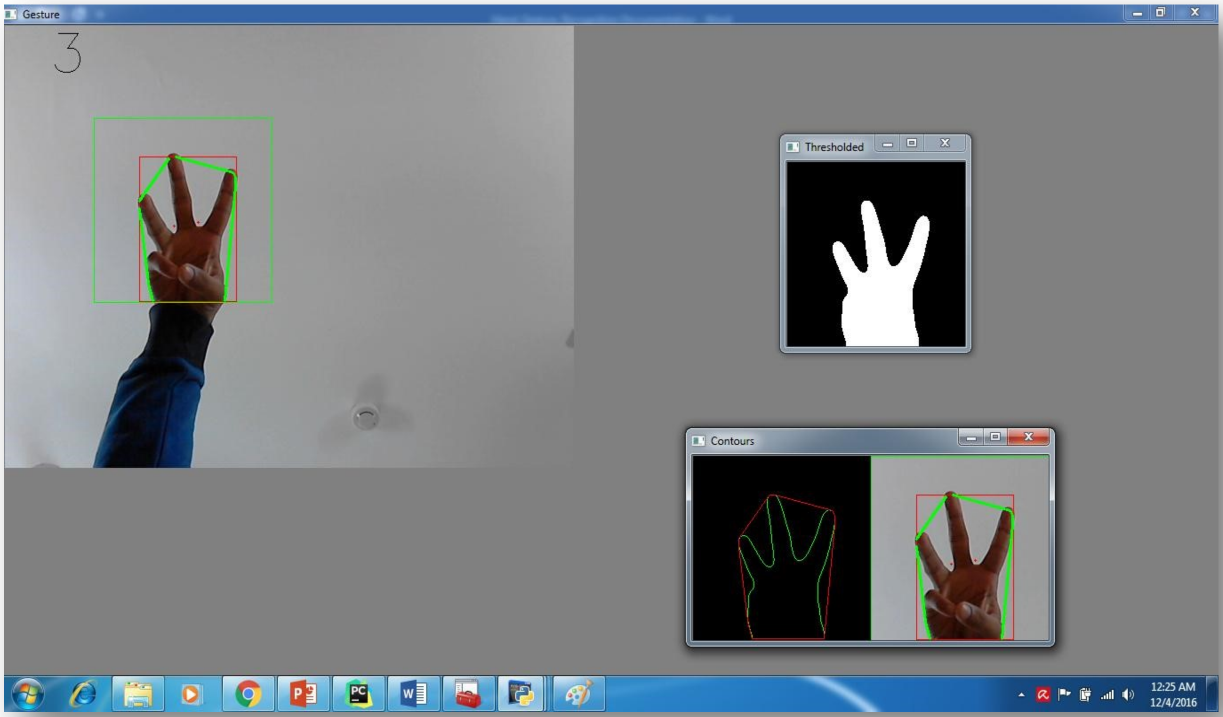


Fig 9 : Connection

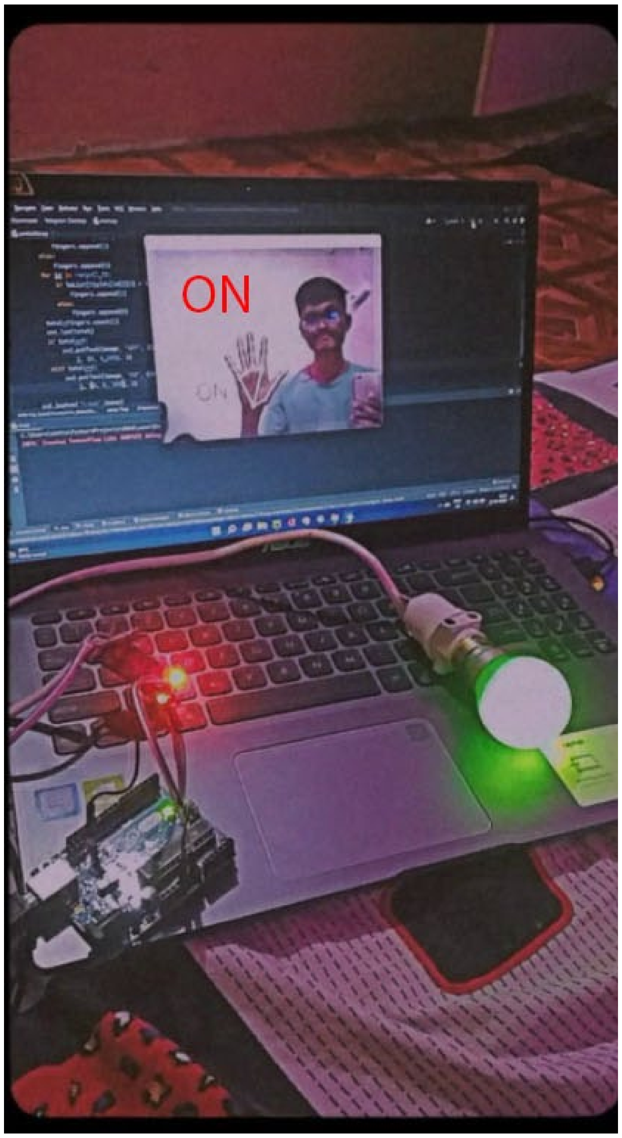
8.OUTPUT :

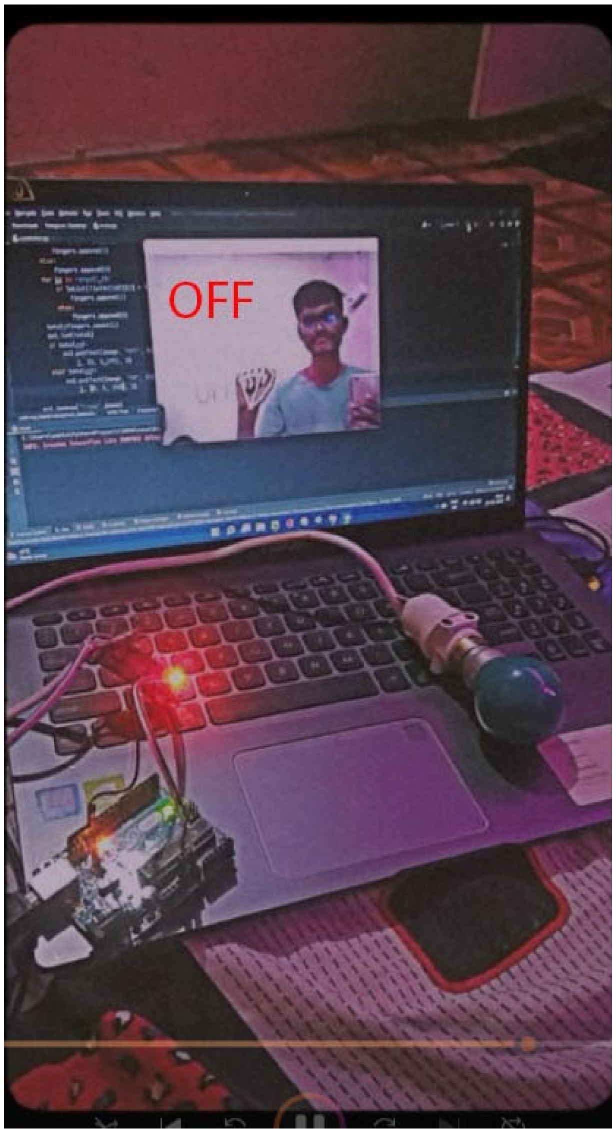
Illustrated in the next page are some of the screenshots of hand recognition and gesture being detected as required.





https://youtube.com/shorts/ETNypV6c6Ps?feature=share





Conclusion

The goal of our project was to design a useful and fully functional real-world product that efficiently translates the movement of hand to electrical signals that can control the home appliances. Our motivation is to help differentially able people to control the electrical appliances more easily. The gesture control automation system uses a glove to recognize the hand positions and outputs onto a display and control the electronic devices like fan, light, music system etc. The system was trained and tested for multiple users successfully. The proposed system has the advantage of low power consumption, simple hardware and hand gestures, easy to operate and user friendly.

References & Tutorials:

1) OpenCV documentation: http://docs.opencv.org/2.4.8/
2) HTML book series, full of basic tutorials on Python. This particular article is very useful to understand how to work with classes in Python: http://learnpythonthehardway.org/book/ex40.html
3) Filtering and smoothening images: http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html
4) Contours in OpenCV: http://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html
5) Pyserial documentation: https://pythonhosted.org/pyserial/
6) PIC16F877A program and burn: https://www.elprocus.com/pic-microcontrollerprogramming-using-c-language/
7) PIC16F877A DESCRIPTION: http://microcontrollerslab.com/pic16f877a-introductionfeatures/
8) MAX232 DESCRIPTION: http://www.ti.com/lit/ds/symlink/max232.pdf
9) RELAY DESCRIPTION: http://www.circuitstoday.com/working-of-relays