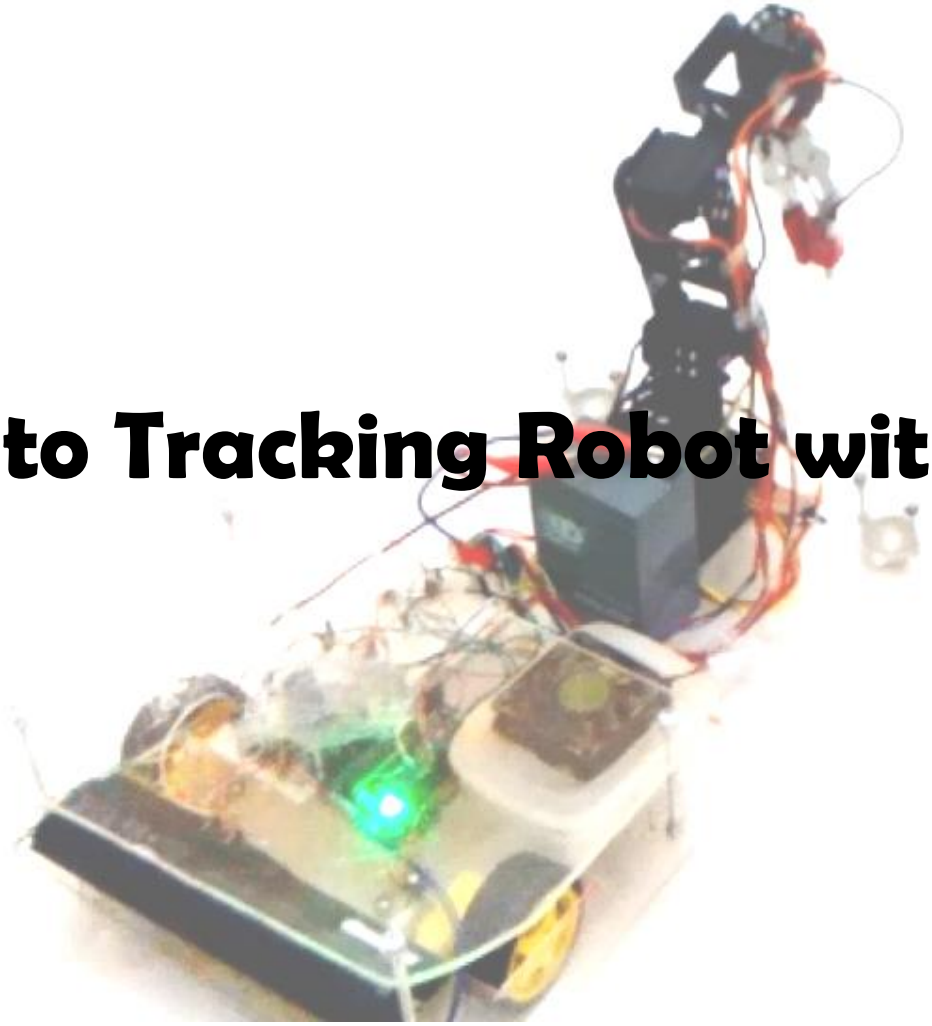
Auto Tracking Robot with Arm



Faculty of Computers and Information

Helwan University

Final Year Project

Team Members :

1- **Mohamed Gomaa zaafan**

2- **Eslam Sobhy Mohamed**

3- **Eslam Ibrahem Ali**

4- **Mohamed Mokhtar Taha**

Acknowledgement

Firstly, all thanks and gratitude to "Allah". We would like to take this opportunity to thank several people to their support which made this project possible through our years, thanks and appreciation for our Supervisors: prof: Aliaa Youssef & Dr: Maged Wafi for patience with us throughout the entire academic year.

Abstract

Our project is a Tracking robot with an arm, with similar functions to a human arm, Servo motor is used for joint rotation. In order for the robot to pick up or move something, you have two choices, first choice to make it under your control, what you need is your mobile or your remote control of TV to perform several actions in a particular order. we also have made a specialized remote to control the robot from far distance (over ٣٠ meters) using RF module (Radio frequency), add to this you can control the robot without any wireless device as you can control it by your hand fingers, Second choice to let it go pick up what you need, you can define specific object to detect and it will come to you with what you want.to perform all this actions (detection and recognition) we used Open Source Computer Vision Library (OpenCV) running on raspberry pi ٢ with its powerful processor .raspberry pi ٢ sends the result of the detection to Arduino, and Arduino in turns control the robot to pick up the detected object.

Contents

Chapter 1
Introduction

1.1. Overview

Robots today are like personal computers 35 years ago—a budding technology that has the potential to revolutionize the way we live our daily lives. If someone takes you 35 years ahead in time, don't be surprised to see robots roaming the streets and working inside buildings, helping and collaborating safely with humans on a lot of daily tasks. Don't be surprised also if you see robots in industries and hospitals, performing the most complex and precision-demanding tasks with ease. And you guessed it right, to do all this they will need highly efficient, intelligent, and robust vision systems.

The term robot comes from the Czech word **robota**, generally translated as "forced labor." This describes the majority of robots fairly well. Most robots in the world are designed for heavy, repetitive manufacturing work. They handle tasks that are difficult, dangerous or boring to human beings. The most common manufacturing robot is the **robot with an arm**.

Controlling the robot is done by rotating individual **servo motors** connected to each joint, as we say before you can control the robot through Bluetooth in your mobile or using remote control, you can also let it to detect objects you need. Since every project tries to do object recognition in its own way, each project has to decide which is the best approach to achieve its final goal. The current state of technology provides different ways to achieve this task. We decided to implement our project with the help of a programming library (roughly, a set of prewritten functions that can execute relevant higher-level tasks) called OpenCV, Raspberry pi used to perform as a computer so we can run OpenCV on it.

Your arm's job is to move your hand from place to place. Similarly, the robotic arm's job is to move an **end effector** from place to place. You can outfit robotic arms with all sorts of end effectors, which are suited to a particular application. One common end effector is a simplified version of the hand, which can grasp and carry different objects. Robotic hands often have built-in **pressure sensors** that tell the computer how hard the robot is gripping a particular object. This keeps the robot from dropping or breaking whatever it's carrying. Other end effectors include blowtorches, drills and spray painters.

1.2. Robot Functions

1- **Exploration robot**

Reaching to unreachable places and taking video streaming from this places then pick up what you want from these places

2- **Detect metal**

Can be taking as one feature in detecting bombs

3- **Integration Of control between several controller**

Bluetooth & Remote Control .....

4- **Control by your hand Fingers**

5- **Control from Far distance using RF module**

By remote we have made...

6- **Following Line using low cost sensor**

Can help in factories to pick up things from place to place in a specific line

7- **Pick up objects without your control by detecting it**

Sending dimensions and distance of objects to Arduino to self-control

1.3. Robot Features:

General Features:

1) Resolution

The resolution of a robot is a feature determined by the design of the control unit and is mainly dependent on the position feedback sensor. It is important to distinguish the programming resolution from the control resolution. When programming resolution is equal to control resolution. In this case both resolutions can be replaced with one term: the system Resolution.

2) Accuracy

Accuracy refers to a robot's ability to position its wrist end at a desired target point within the work volume, and it is defined in terms of spatial resolution. At first accuracy depends on robot technology and how closely the control increments can be defined for each of the joint motions. The term accuracy in robotics is often confused with the terms resolution and repeatability. The final accuracy of a robotic system depends on its mechanical inaccuracies which are caused mainly by backlash in the manipulators joints and bending of the links.

3) Repeatability

Repeatability is a statistical term associated with accuracy; it describes how a point is repeated. Repeatability does not describe the error with respect to absolute coordinates.

4) Size:

The physical size of a robot, which influences its capacity and its capabilities.

5) Servomotor

The arm has will have seven servos which are controlled through the use of only one microcontroller at mega 16.

6) Easy to use

The arm is very user friendly because of the computer interface developed by us, even layman could operate it.

7) It could lift objects up to weight of 250 gm.

8) Efficiency

Enabling the base rotation without the help of any gears or ball bearing, also using only good torque servo motors and three castor wheels for rotating the whole body.

9) Simplicity

Keeping the design of robotic arm gripper simple, as well as implementing the gripping mechanism with using simple gears and with two servo motors as shown in Figure 1.2.

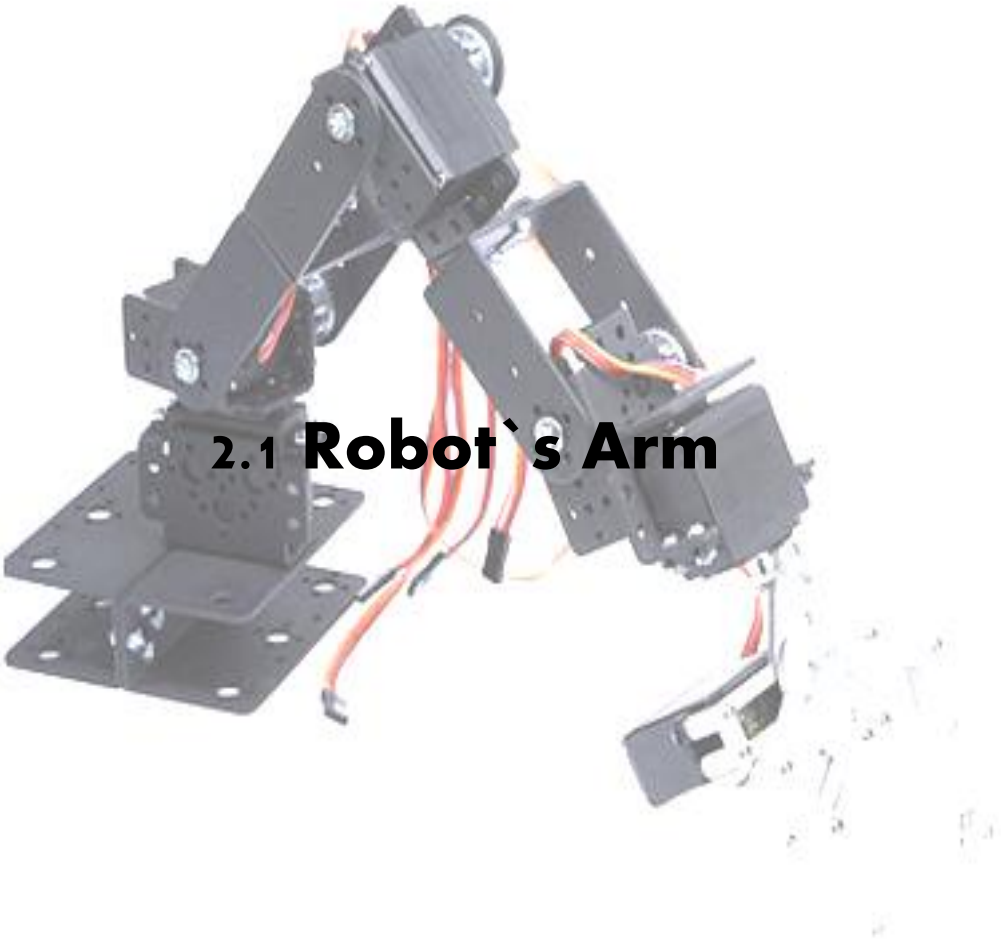
1.4. Robot Advantages:

- Easy to Use
- Low Cost
- Fun!
- Super Compact Body
- Includes Software Command Line Control
- High Quality Construction
- Detect objects and computing distance
- Real-time position
- Robotic arm delivers fast, accurate, and repeatable movement.
- Base rotation, single plane shoulder, elbow, wrist motion, a functional gripper, and optional wrist rotate.

1.5. Main components

1. Dof Arm
2. Servo Motors
3. Raspberry pi
4. Webcam
5. Arduino UNO
6. Dc motors
7. Servo Motors
8. IC 293D (Control DC Motors)
9. Bluetooth sensor
10. Ultrasonic Sensor

Chapter 2
Hardware Environment



2.1 Robot`s Arm

2.1.1 Robot's Arm Description

A robotic arm is a robotic manipulator, programmable, with similar functions to a human arm .Servo motor is used for joint rotation. It has about same number of degree of freedom as in human arm. Humans pick things up without thinking about the steps involved. In order for a robot or a robotic arm to pick up or move something, someone has to tell it to perform several actions in a particular order from moving the arm, to rotating the "wrist" to opening and closing the "hand" .So, we can control each joint through any interface.

Arm Specification:

Weight: approx. 500 g

Material: aluminums alloy

Function: clamping objects

The arm is from the main sections of our project and consists of three parts: the shoulder, elbow and wrist. These are all joints, with the shoulder resting at the base of the arm, typically connected to the controller, and it can move forward, backward or spin. The elbow is in the middle and allows the upper section of the arm to move forward or backward independently of the lower section. Finally, the wrist is at the very end of the upper arm and attaches to the end effector, as shown in figure 1.4.

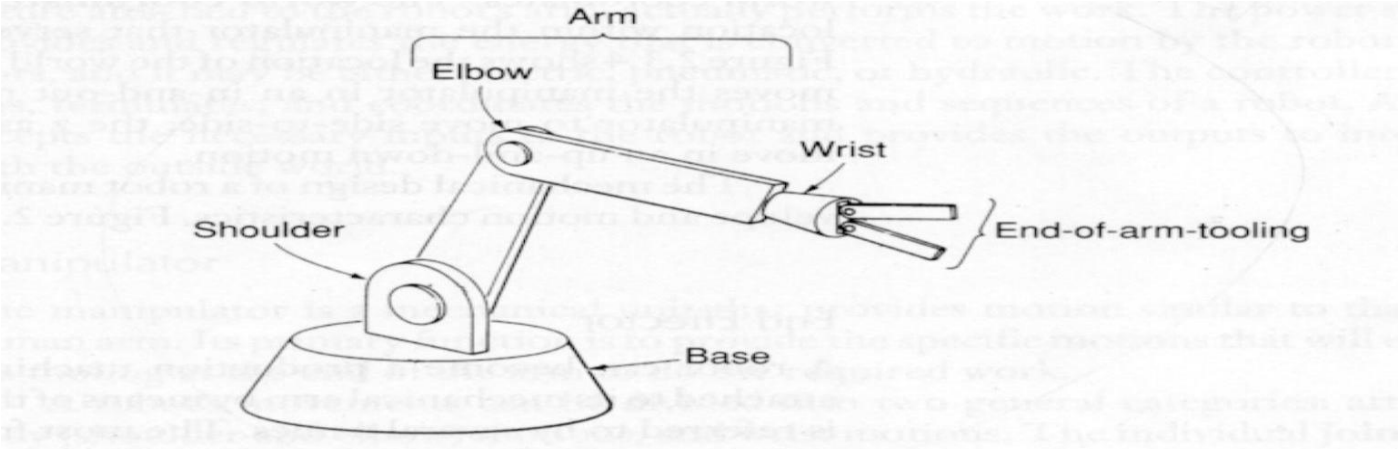


Figure 1.4 Parts of manipulator

2.1.2 Six degrees of freedom

Six degrees of freedom (6DoF) refers to the freedom of movement of a rigid body in three-dimensional space. Specifically, the body is free to move forward/backward, up/down, left/right (translation in three perpendicular axes) combined with rotation about three perpendicular axes, often termed pitch, yaw, and roll.

Robotics

Serial and parallel manipulator systems are generally designed to position an end-effector with six degrees of freedom, consisting of three in translation and three in orientation. This provides a direct relationship between actuator positions and the configuration of the manipulator defined by its forward and inverse kinematics.

Robot arms are described by their degrees of freedom. This number typically refers to the number of single-axis rotational joints in the arm, where higher number indicates an increased flexibility in positioning a tool. This is a practical metric, in contrast to the abstract definition of degrees of freedom which measures the aggregate positioning capability of a system.

In 2007, Dean Kamen, inventor of the Segway, unveiled a prototype robotic arm with 14 degrees of freedom for DARPA. Humanoid robots typically have 30 or more degrees of freedom, with six degrees of freedom per arm, five or six in each leg, and several more in torso and neck

Engineering

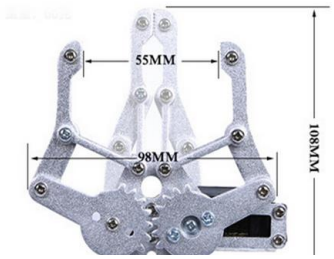
The term is important in mechanical systems, especially biomechanical systems for analyzing and measuring properties of these types of systems that need to account for all six degrees of freedom. Measurement of the six degrees of freedom is accomplished today through both AC and DC magnetic or electromagnetic fields in sensors that transmit positional and angular data to a processing unit. The data are made relevant through software that integrate the data based on the needs and programming of the users.

Ascension Technology Corporation has recently created a 6DoF device small enough to fit in a biopsy needle, allowing physicians to better research at minute levels. The new sensor passively senses pulsed DC magnetic fields generated by either a cubic transmitter or a flat transmitter and is available for integration and manufacturability by medicalOEMs

An example of six degree of freedom movement is the motion of a ship at sea.

2.1.3 Arm Component

Alloy mechanical claw x 1



Multi-functional bracket x 5

Long U shape bracket x 3

L shape bracket x 1

U beam bracket x 2

End Effector

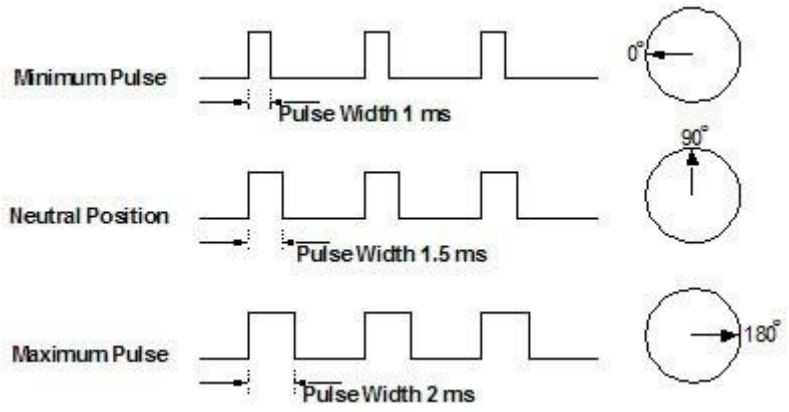
End effector acts as the hand of the robotic arm. It is often composed of two claws, though sometimes three, that can open or close on command. It can also spin on the wrist, making maneuvering material and equipment easy, as shown.

2.1.4 Servo used in the Arm

Servos are a special type of DC motors with built in gearing and feedback control loop circuitry and they don't require motor controllers. These motors are mainly developed for making robots, toys, etc. that are mainly used for education and not for industrial applications. Most servo motors can rotate about 90 to 180 degrees.

Note, if you are running multiple servos simultaneously, as we used, you can just put a few of these program blocks in sequential order. You can run as many servos as you have of digital ports. So how many milliseconds do you keep the port high? It all depends on the servo. You may have to tweak for each individual servo some several microseconds' difference.

The standard **time vs. angle** is represented in this chart:



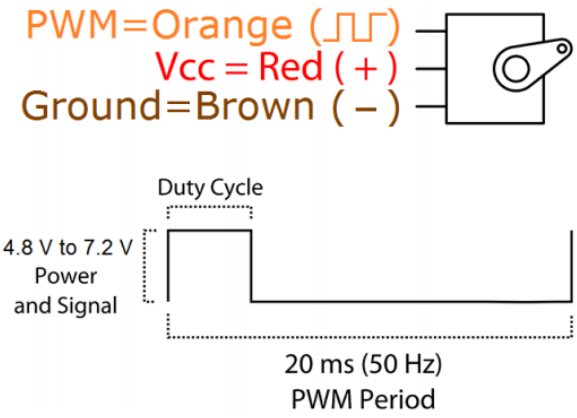
Servo used:

MG996R servo (High Torque Metal Gear Dual Ball Bearing Servo)

This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwidth and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications



- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)
- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA –
- Stall Current 2.5 A (6V)
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 ºC – 4.8 V a 7.2 V – 900 mA (6V) double ball bearing design 55 ºC

2.1.5 Steps to build an arm

A robot manipulator consists of links connected by joints. The links of the manipulator can be considered to form a kinematic chain. The business end of the kinematic chain of the manipulator is called the end effector and it is analogous to the human hand. The end effector can be a gripper or can be designed to perform any desired task such as welding, painting, assembly.

1. Mechanical design

In constructing the arm, we made use of five servo motors (including gripper) since our structure allows movement in all three dimensions. There is a servo motor at the base, which allows for angular movement of the whole structure; other two at the shoulder and elbow to allow the upward and downward movement of the arm; one for the movement of the wrist while the last servo motor at the end effector allows for the gripping of objects.

The serial arm is a six degree of freedom system. Three DOF control the position of the arm in the Cartesian space, one for wrist orientation and one additional servo for actuating gripper. The robot features in the control GUI or teach pendant are base rotation, shoulder, elbow, wrist rotate, and a functional gripper. The base of the robotic arm is made up of Perspex or acleric.

2. Assembly details of arm

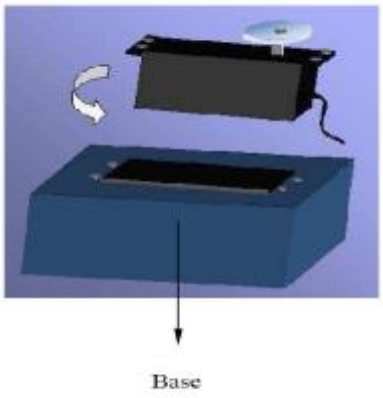


Figure 1.7 First step

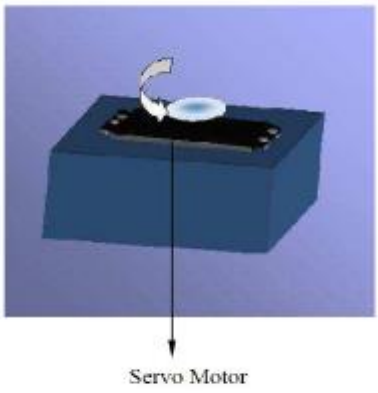


Figure 1.8 second step

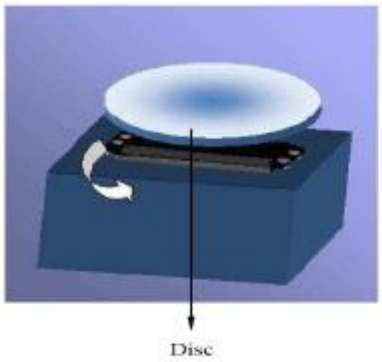


Figure 1.9 Third step

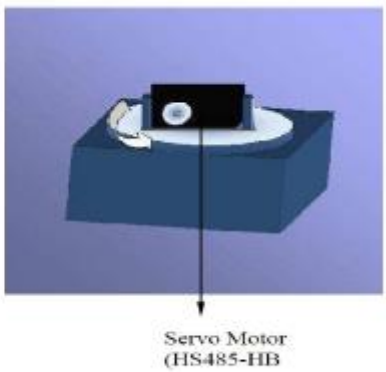


Figure 1.10 Fourth Step

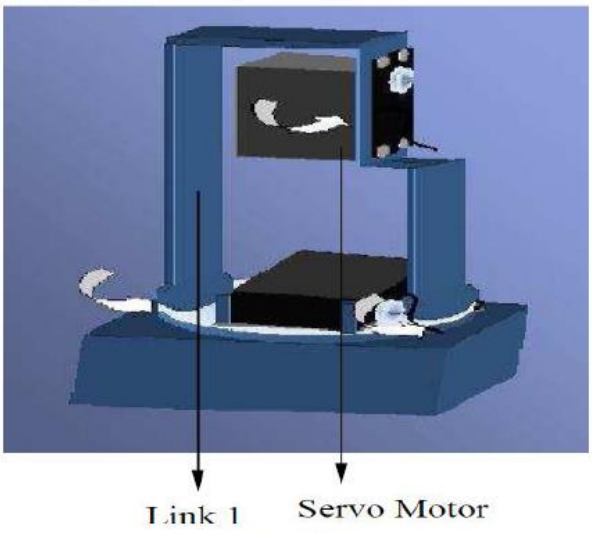


Figure 1.11 Fifth step

NOTE:

Before fixing the links to servo motor's shaft, the motor shaft be brought to center position 90 degree (can be changed depend on what you need) and then the links are to be fixed in upright position.

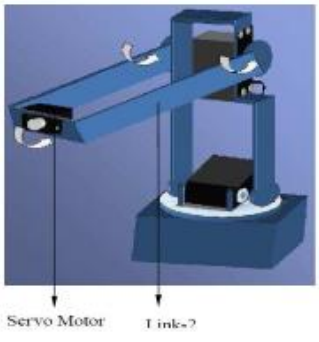


Figure 1.12 sixth step

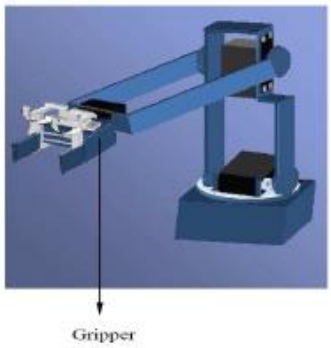
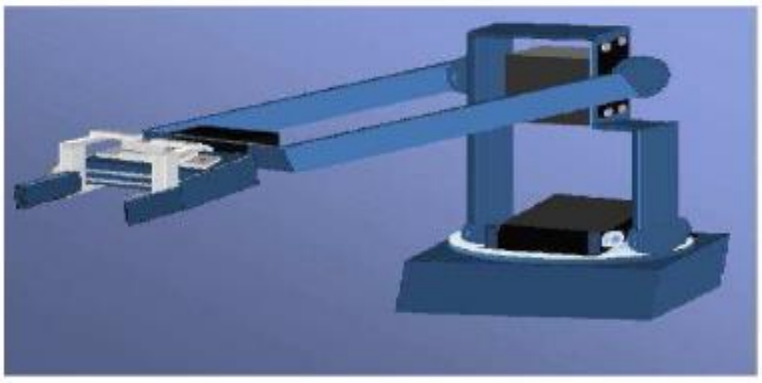


Figure 1.13 seventh step



Fully Assembled Serial Arm

Figure 1.14 Final step

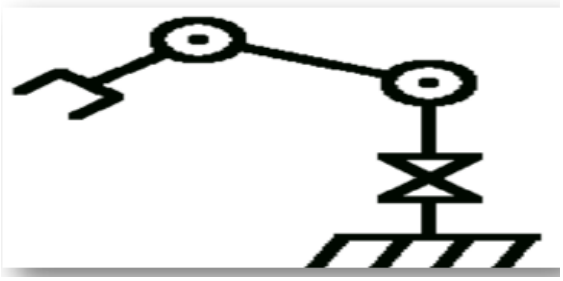
3. Robot workspace degrees

1. Degrees of Freedom (DOF)

In mechanics, the **degree of freedom (DOF)** of a mechanical system is the number of independent parameters that define its configuration. It is the number of parameters that determine the state of a physical system and is important to the analysis of systems of bodies in mechanical engineering, aeronautical engineering, robotics, and structural engineering.

A degree of freedom is a joint on the arm, a place where it can bend or rotate or translate. We can typically identify the number of degrees of freedom by the number of actuators on the robot arm (in case of serial arms).

2. Robot Workspace (Work Volume)

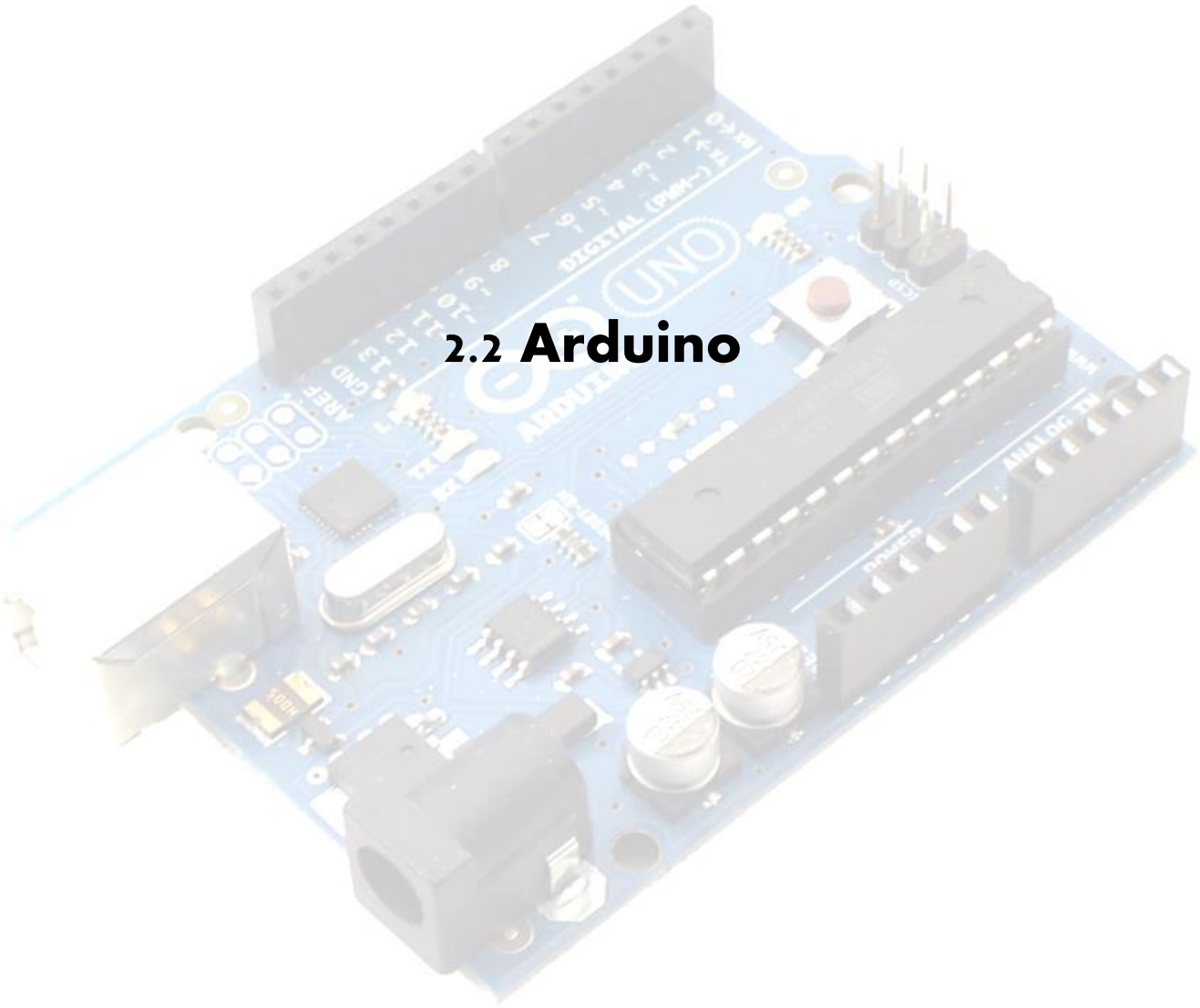


The robot workspace (sometimes known as reachable space) is a collection of points that the end effector (gripper) can reach. The workspace is dependent on the DOF angle/translation limitations, the arm link lengths, the angle at which something must be picked up at. The workspace is highly

Dependent on the robot configuration, as shown in figure1.15. The figure describes the workspace for our serial arm. It should be noted that it does not include the DOF which controls the wrist orientation as the workspace is independent of orientation variable.

The shoulder and elbow joints rotate a maximum of 180 degrees. To determine the workspace, trace all locations that the end effector (gripper) can reach.

2.2 Arduino



2.2.1 What is Arduino

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.

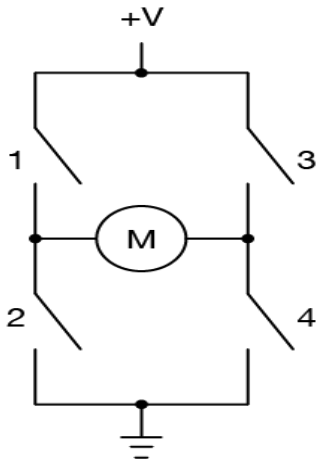
2.2.2 Dc Motor Control

When trying to move things with microcontrollers, three kinds of motors are most commonly used: DC motors, RC servomotors, and stepper motors.

There are two easily controllable parameters of a DC motor, direction and speed. To control the direction, you reverse the direction of the voltage through the motor. To control the speed, you pulse width modulate it.

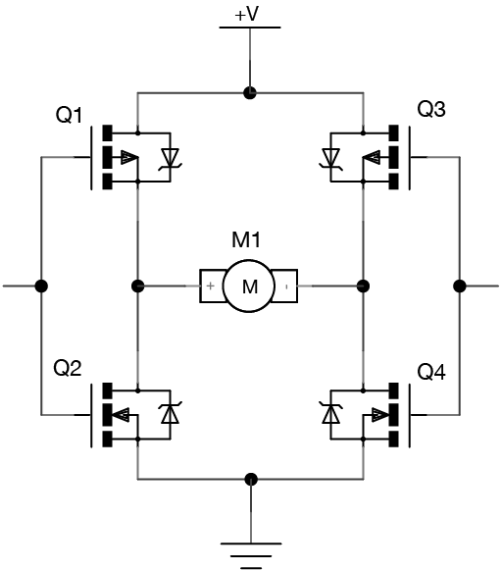
Direction

To control a DC motor from a microcontroller, you use switching arrangement known as an H-bridge, consisting of four switches with the motor in the center. Here's the schematic for a typical H-bridge:



When switches 1 and 4 are closed and 2 and 3 are open, voltage flows from the supply to 1 to the motor to 4 to ground. When 2 and 3 are closed and 1 and 4 are open, polarity is reversed, and voltage flows from the supply to 3 to the motor to 2 to ground.

An H-bridge can be built from transistors, so that a microcontroller can switch the motor, like this:



This schematic uses MOSFETs, which are good for controlling motors. The top two are P-channel, and the bottom two are N-channel, so that the proper two transistors always switch together. When the left control pin is high, Q1 turns off because it's a P-channel and Q2 turns on because it's an N-channel. The same happens with Q3 and Q4. If you were using this circuit, you'd want to make sure that the control pins are always reversed; when one is high, the other is low.

Although you can make your own H-bridges, it's usually easier to use a controller manufactured specifically for the job. A pre-manufactured H-bridge chip will include diodes to protect the transistors from back voltage, sometimes a current sensing pin to sense the current the motor is drawing, and much more. There are many motor drivers available from various electronics suppliers. Look around to find one that suits your needs and price range.

Speed

A DC motor's speed is proportional to the supplied voltage. If the voltage drops too far, the motor won't get enough power to turn, but within a certain range, usually 50% of the rated voltage, the motor will run at varying speeds. The most effective way to adjust the speed is by using pulse width modulation. This means that you pulse the motor on and off at varying rates, to simulate a voltage.

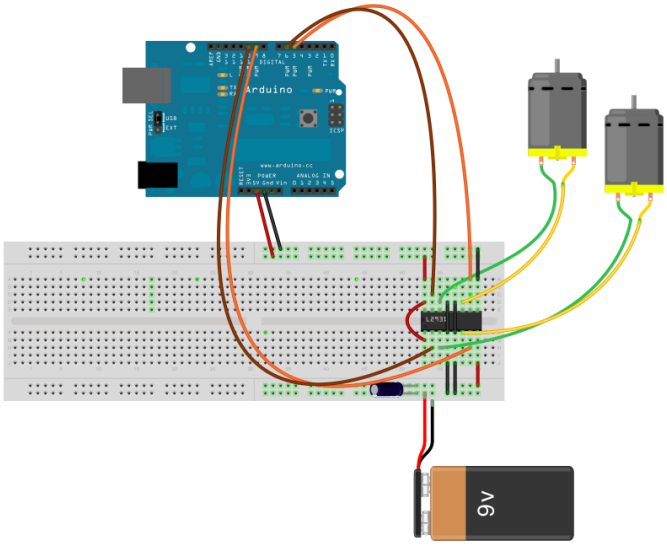


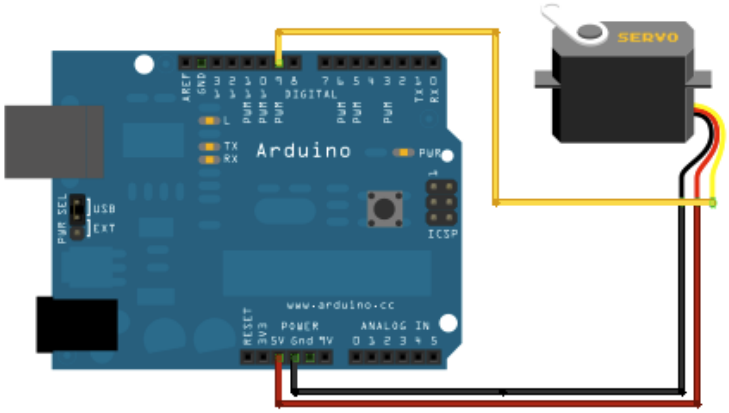
Figure 1.16 shows how to connect DC motors using H-bridge (l239d)

2.2.3 Servo Motor control

Unlike dc motors, with servo motors you can position the motor shaft at a specific position (angle) using control signal. The motor shaft will hold at this position as long as the control signal not changed. This is very useful for controlling robot arms, unmanned airplanes control surface or any object that you want it to move at certain angle and stay at its new position.

Servo motors may be classified according to size or torque that it can withstand into mini, standard and giant servos. Usually mini and standard size servo motors can be powered by Arduino directly with no need to external power supply or driver.

Usually servo motors comes with arms (metals or plastic) that is connected to the object required to move (see figure below to the right).

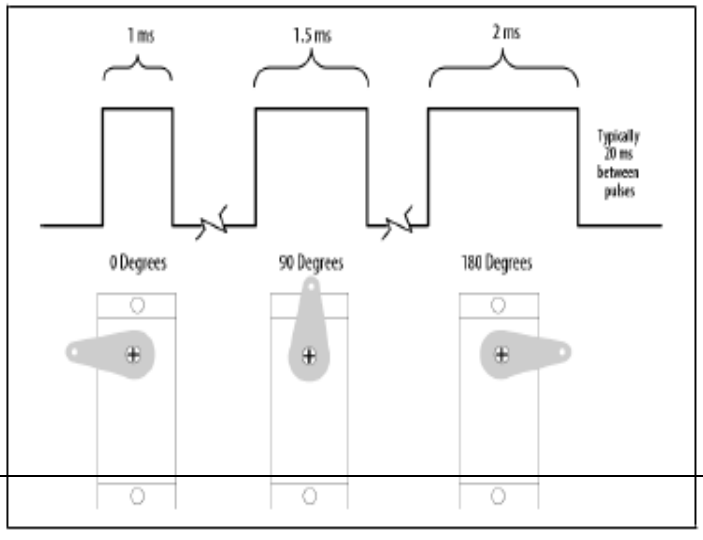


How Does the Servo Motor Works?
Your servo have 3 wires:
Black wire: GND (ground), RED wire: +5v, Colored wire: control signal

1. Connect the black wire from the servo to the Gnd pin on the Arduino
2. Connect the red wire from the servo to the +5V pin on the Arduino
3. Connect the third wire (usually orange or yellow) from the servo to a digital pin on the Arduino

The third pin accept the control signal which is a pulse-width modulation (PWM) signal. It can be easily produced by all micro- controllers and Arduino board. This accepts the signal from your controller that tells it what angle to turn to. The control signal is fairly simple compared to that of a stepper motor. It is just a pulse of varying lengths. The length of the pulse corresponds to the angle the motor turns to.

The pulse width sent to servo ranges as follows:



Minimum: 1 millisecond ---> Corresponds to 0 rotation angle.
Maximum: 2 millisecond ---> Corresponds to 180 rotation angle.

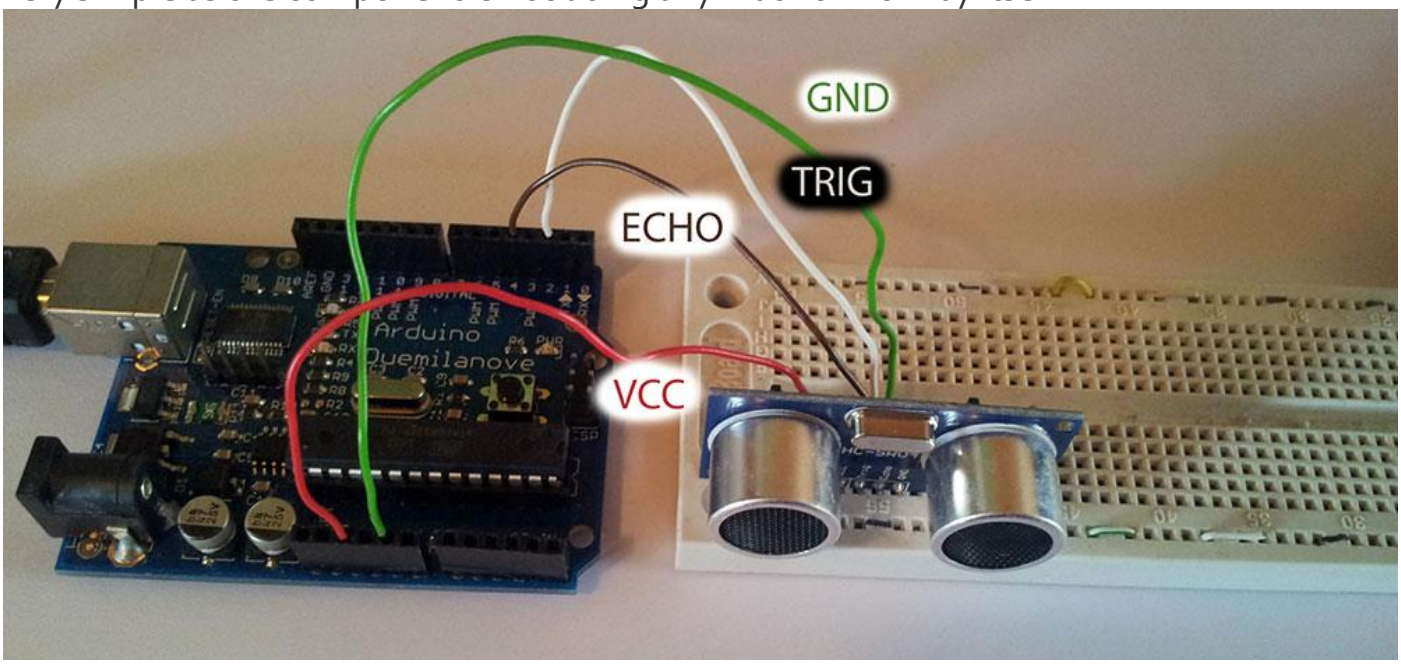
Any length of pulse in between will rotate the servo shaft to its corresponding angle. For example, 1.5 ms pulse corresponds to rotation angle of 90 degree. This is will explained in figure below.

2.2.4 Sensors connected with Arduino

1. Ultrasonic Sensor

This component is very simple, as it only has 4 pins.
First and the last are power pins Vcc and GND. The middle ones are Trigger pin and Echo pin.

To understand what these pins do, you must understand how this thing works. It's actually very simple as the component is not doing any math or work by itself.



So, this module is sending ultrasonic waves. Than these waves reflect from a surface and come back to the module. Arduino is triggering those waves on the trigger pin and than listening for the "echo". Once it receives it, it calculates the distance base on the time spend waiting for the wave to come back. In the code, we are also going to change that value to cm, as it easier to read.

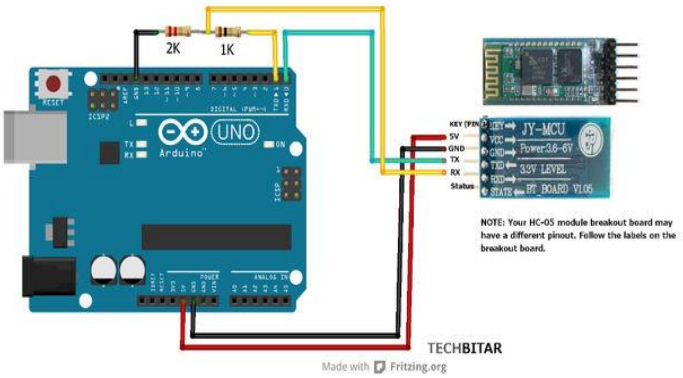
The module will be connected to any I/O pins of the Arduino. The trigger pin will be output and the echo pin will be input.

2. Bluetooth sensor

Probably all Bluetooth modules have the same pin-out.
It has power pins Vcc and GND and communication pins RX and TX.

Now you can probably see, that this module connects to the RX and TX pins of a micro controller such as Arduino or a USB to serial converter.

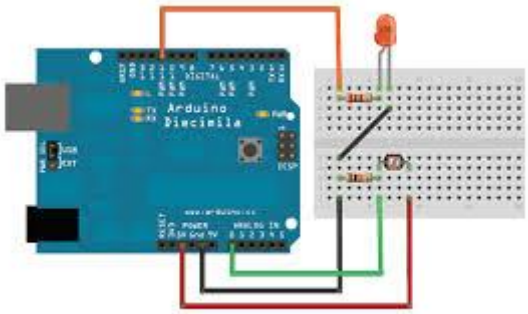
This module works fully with 5V, so there is no problem connecting it to Arduino. All the pins are labeled on the back of the module



3-Light sensor

LEDs are commonly used as lights, but then can also be used as photodiodes to detect light. Connect one pin of LDR to 5v and another pin to Ao input. This is a Light sensor using LDR and Arduino, you can make it either Shadow detector or light detector by simply changing the code.

Connect one pin of 10k resistor to GND and another pin to Ao input.



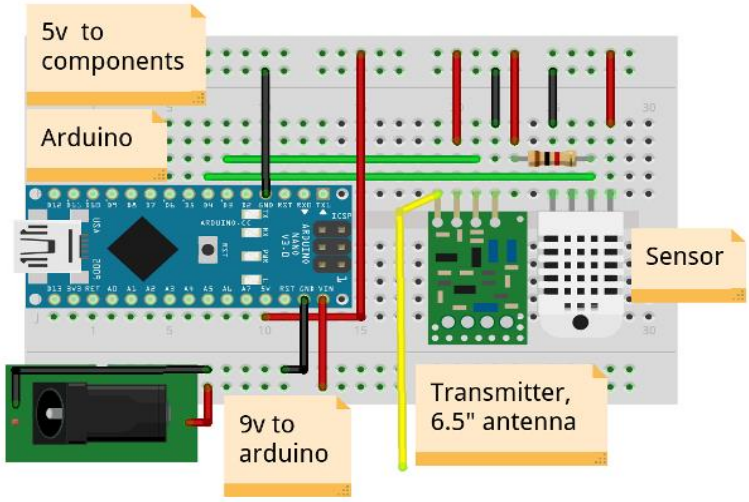
4-Metal detector

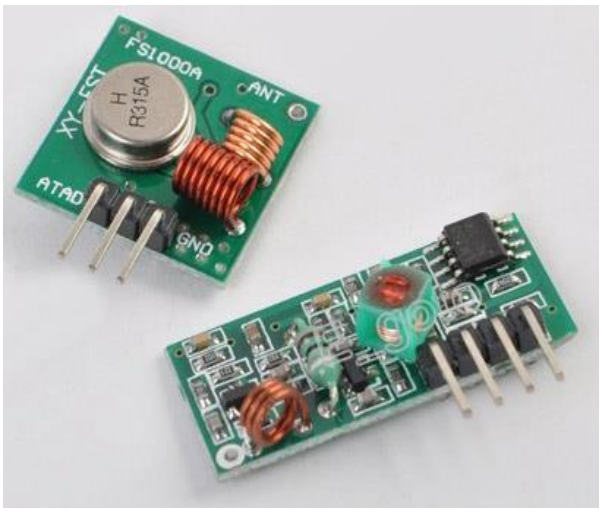
A **metal sensor** is an electronic instrument which detects the presence of metal nearby. Metal detectors are useful for finding metal inclusions hidden within objects, or metal objects buried underground. They often consist of a handheld unit with a sensor probe which can be swept over the ground or other objects. If

the sensor comes near a piece of metal this is indicated by a changing tone in earphones, or a needle moving on an indicator. Usually the device gives some indication of distance; the closer the metal is, the higher the tone in the earphone or the higher the needle goes.

5-RF sensor

It includes two parts, one transmitter units and one receiver unit. The transmitters are made up of an Arduino board, sensor, and RF transmitter. The receiver unit is composed of an Arduino board, an RF receiver.

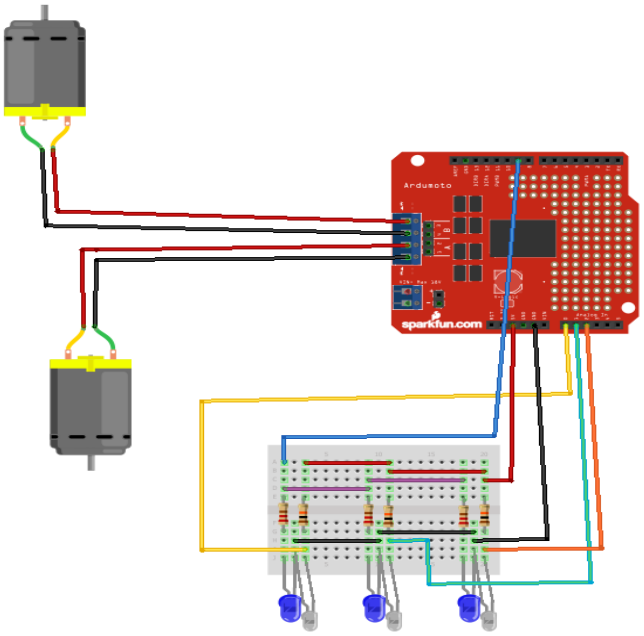


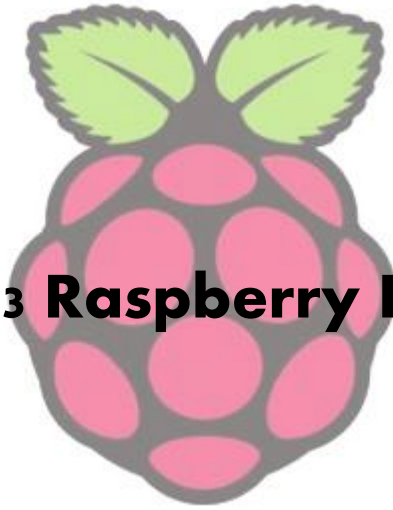


5. **Following line sensor**

There are basically two ways of building a line following robot. First, using a light dependent resistor (LDR) and second, using Infrared (IR) LED. A dark object reflects less light than a bright object, or we can also say, a dark object absorbs more light than a bright object, this is the only fundamental logic behind a line sensing robot. So, in a line sensing robot, we make a sensor that detects the difference between bright object and a dark object, or say, distinguishes a black line from a white surface.

The circuit for making Infrared based line sensing robot:


Arduumoto
sparkfun.com



2.3 Raspberry Pi

2.3.1 Introduction

The Raspberry Pi is a low cost, **credit-card sized computer** that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

What's more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

2.3.2 Purpose of using raspberry pi

Specifications and performance:

As for the specifications, the Raspberry Pi is a credit card-sized computer powered by the Broadcom BCM2835 system-on-a-chip (SoC). This SoC includes a 32-bit ARM1176JZFS processor, clocked at 700MHz, and a Video core IV GPU. It also has 256MB of RAM in a POP package above the SoC. The Raspberry Pi is powered by a 5V micro USB AC charger or at least 4 AA batteries (with a bit of hacking).

While the ARM CPU delivers real-world performance similar to that of a 300MHz Pentium 2, the Broadcom GPU is a very capable graphics core capable of hardware decoding several high definition video formats. However, in order to keep costs of the Raspberry Pi low, the UK charity has only licensed the H.264 codec for hardware decoding (and it is unclear if users will be able to purchase/activate additional codecs). In that regard, the Video core IV GPU is rather potent as it is capable of hardware decoding 1080p30 H.264 with bit-rates up to 40Mb/s.

The Raspberry Pi model available for purchase at the time of writing — the Model B — features HDMI and composite video outputs, two USB 2.0 ports, a 10/100 Ethernet port, SD card slot, GPIO (General Purpose I/O Expansion Board) connector, and analog audio output (3.5mm headphone jack). The less expensive Model A strips out the Ethernet port and one of the USB ports but otherwise has the same hardware. It is this model that is the "$25 PC" that originally made so many headlines.

2.3.3 Raspberry Pi 2 Model B: (Used)

The Raspberry Pi 2 Model B is the second generation Raspberry Pi. It replaced the original Raspberry Pi 1 Model B+ in February 2015. Compared to the Raspberry Pi 1 it has:

- A 900MHz quad-core ARM Cortex-A7 CPU

- 1GB RAM

✓ *Like the (Pi 1) Model B+, it also has :*

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- Video Core IV 3D graphics core

Because it has an ARMv7 processor, it can run the full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, as well as Microsoft Windows 10 (see the blog for more information).

The Raspberry Pi 2 has an identical form factor to the previous (Pi 1) Model B+ and has complete compatibility with Raspberry Pi 1.

We recommend the Raspberry Pi 2 Model B for use in schools: it offers more flexibility for learners than the leaner (Pi 1) Model A+, which is more useful for embedded projects and projects which require very low power.

2.3.4 Why raspberry pi 2 is used!

The Raspberry Pi 2 is undoubtedly better than the Model B+, but do you need it?

The Raspberry Pi has been a huge success in both the homebrew development scene and in educating young minds about how computers work. Its tiny form-factor and incredible price make it irresistible to anyone who's interested in understanding how computers work, or those who want something small, powerful and flexible to tinker with.

	Pi 1 B+	Pi 2 B
CPU	Arm11	Cortex A7
Cores	1	4
Clock	700MHz	900MHz
GPU	Videocore IV	Videocore IV
Memory	512MB	1GB
USB Ports	4	4
Flash	None	None
Storage	microSD	microSD
Network	10/100	10/100
GPIO	40-pin	40-pin
Wifi	No	No
Bluetooth	No	No
RRP	$35	$35

2.3.5 Raspberry pi and Arduino connection

Both the PI and Arduino support two additional types of communication for talking to peripheral devices. There's SPI which is a high speed serial protocol and I2C. Like RS232, SPI needs level shifters, but not exactly so for I2C.

I2C is a 2 wire protocol allowing for 127 devices to be connected to a single bus. One device is the master (The PI in our case) and then the peripherals.

Configuring the PI for I2C

First we need to enable the I2C module on the PI.

Remove I2C from the module blacklist

As root edit /etc/modprobe.d/raspi-blacklist.conf and comment out the line blacklisting i2c-bcm2708

```
$ cat /etc/modprobe.d/raspi-blacklist.conf
# blacklist spi and i2c by default (many users don't need them)
blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

Next add i2c-dev to the /etc/modules file so it's loaded on boot:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
ipv6
i2c-dev
```

Finally install i2c-tools:

```
$ sudo apt-get install i2c-tools
$ sudo adduser pi i2c
```

Now reboot the PI.

Configuring the Arduino

The following sketch implements a simple I2C slave with two commands:

Command 1 will toggle the onboard led on the Arduino.

Command 2 will return the Arduino's temperature in Celsius.

```ino
#include <Wire.h>

#define SLAVE_ADDRESS 0x04
int number = 0;
int state = 0;

double temp;

void setup() {
 pinMode(13, OUTPUT);

 // initialize i2c as slave
 Wire.begin(SLAVE_ADDRESS);

 // define callbacks for i2c communication
 Wire.onReceive(receiveData);
 Wire.onRequest(sendData);
}

void loop() {
 delay(100);
 temp = GetTemp();
}

// callback for received data
void receiveData(int byteCount){

 while(Wire.available()) {
  number = Wire.read();

  if (number == 1){
   if (state == 0){
    digitalWrite(13, HIGH); // set the LED on
    state = 1;
   } else{
    digitalWrite(13, LOW); // set the LED off
    state = 0;
   }
  }

  if(number==2) {
   number = (int)temp;
  }
 }
}

// callback for sending data
void sendData(){
 Wire.write(number);
}

// Get the internal temperature of the arduino
double GetTemp(void)
{
 unsigned int wADC;
 double t;
 ADMUX = (_BV(REFS1) | _BV(REFS0) | _BV(MUX3));
 ADCSRA |= _BV(ADEN); // enable the ADC
 delay(20); // wait for voltages to become stable.
 ADCSRA |= _BV(ADSC); // Start the ADC
 while (bit_is_set(ADCSRA,ADSC));
 wADC = ADCW;
 t = (wADC - 324.31 ) / 1.22;
 return (t);
}
```

The Raspberry PI client

Here's a simple C application which will now talk to the Arduino over I2C:

```c
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>

// The PiWeather board i2c address
#define ADDRESS 0x04

// The I2C bus: This is for V2 pi's. For V1 Model B you need i2c-0
static const char *devName = "/dev/i2c-1";

int main(int argc, char** argv) {

  if (argc == 1) {
    printf("Supply one or more commands to send to the Arduino\n");
    exit(1);
  }

  printf("I2C: Connecting\n");
  int file;

  if ((file = open(devName, O_RDWR)) < 0) {
    fprintf(stderr, "I2C: Failed to access %d\n", devName);
    exit(1);
  }

  printf("I2C: acquiring buss to 0x%x\n", ADDRESS);

  if (ioctl(file, I2C_SLAVE, ADDRESS) < 0) {
    fprintf(stderr, "I2C: Failed to acquire bus access/talk to slave 0x%x\
    exit(1);
  }

  int arg;

  for (arg = 1; arg < argc; arg++) {
    int val;
    unsigned char cmd[16];

    if (0 == sscanf(argv[arg], "%d", &val)) {
      fprintf(stderr, "Invalid parameter %d \"%s\"\n", arg, argv[arg]);
      exit(1);
    }

    printf("Sending %d\n", val);

    cmd[0] = val;
    if (write(file, cmd, 1) == 1) {

      // As we are not talking to direct hardware but a microcontroller we
      // need to wait a short while so that it can respond.
      //
      // 1ms seems to be enough but it depends on what workload it has
      usleep(10000);

      char buf[1];
      if (read(file, buf, 1) == 1) {
    int temp = (int) buf[0];

    printf("Received %d\n", temp);
      }
    }

    // Now wait else you could crash the arduino by sending requests too f
    usleep(10000);
  }

  close(file);
  return (EXIT_SUCCESS);
}
```

Save that as main.c and compile it:

```
1    pi@mimas ~ $ gcc main.c -o main
```

Now you'll notice there's a couple of usleep() waits in this code, once between sending the command and again after reading the response. I've found that this is necessary for two reasons.

1. The Arduino is emulating an I2C device, so it won't respond immediately unlike a dedicated device so you need to wait a short while before reading it otherwise you won't get a response.
2. Without the second delay you can confuse the Arduino by requesting another command too quickly, necessitating the Arduino to be reset before it can be used again.

I found that 10000 (10ms) is enough here.

Wiring the two together:

Now this is simple: First power down both the Arduino and the Raspberry PI – never connect things whilst they are powered up!

Next simply connect the two with 3 wires using the following table:

Raspberry PI		Arduino
GPIO 0 (SDA)	<–>	Pin 4 (SDA)
GPIO 1 (SCL)	<–>	Pin 5 (SCL)
Ground	<–>	Ground

Testing

Power up both the Arduino and Raspberry PI. Once it's up and running log in and run i2cdetect:

```
pi@mimas ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- 04 -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- UU -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

What you are now seeing is a list of all I2C devices connected. The one you are interested in is 04 which happens to be your arduino.

Lets toggle the led:

```
pi@mimas ~ $ ./main 1
I2C: Connecting
I2C: acquiring buss to 0x4
Sending 1
Received 1
```

You should now see the onboard led turn on. Run it again and the led goes out.

How about the temperature?

```
pi@mimas ~ $ ./main 2
I2C: Connecting
I2C: acquiring buss to 0x4
Sending 2
Received 35
```

35 just happens to be the Arduino's internal temperature in Celsius (in this example we've just returned the integer temperature).

That's about it. The next thing I now need to do is to get those additional sensors working with the Arduino and wrap them in an I2C slave.

Oh, one last thing, as I said earlier and in that diagram, you can have many devices on the one I2C bus, so to add another Arduino all you need to do is to connect the three wires to that Arduino as well and make certain it is using a different address.

2.3.6 Running Arduino IDE on raspberry pi

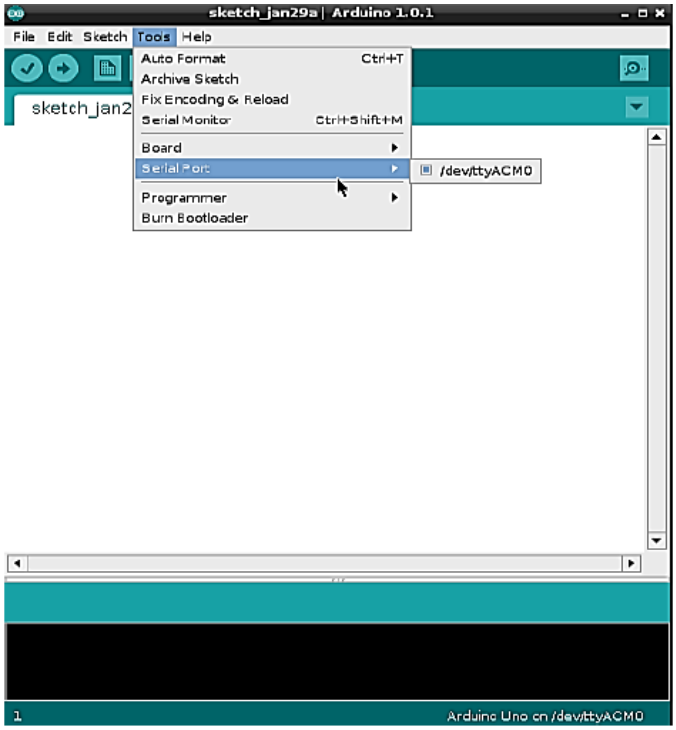
The Arduino IDE is available for the Raspberry Pi. It is a little bit slow, but usable. Use these commands to install

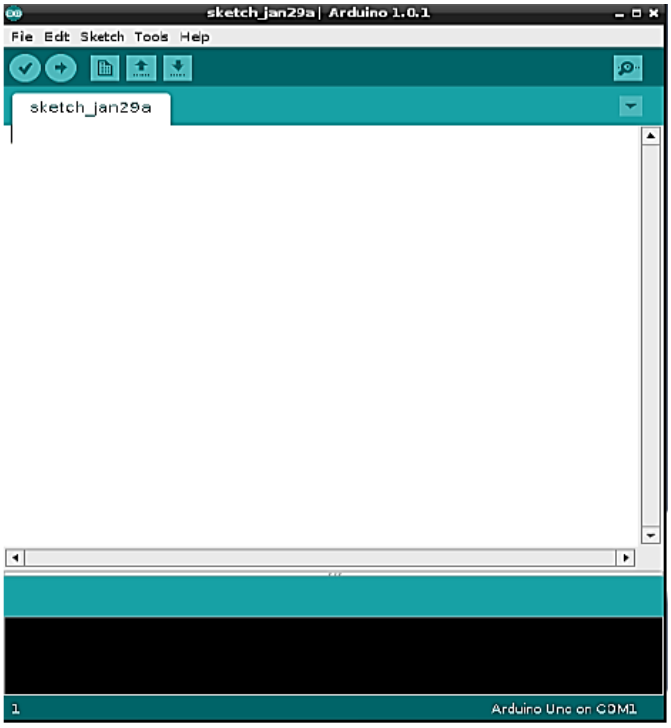
```
sudo apt-get install -y arduino
sudo apt-get install -y python-serial
```

You can see your Arduino on dev folder by name ttACM0 or ttyACM1

```
pi@raspberrypi ~ $ ls /dev/tty*
/dev/tty     /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0    /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1    /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10   /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11   /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12   /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyACM0
/dev/tty13   /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttyAMA0
```

Now...by typing just Arduino on your terminal you will open it





Chapter 3
Software Environment

3.1 Computer vision

Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions.

A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, object pose estimation, learning, indexing, motion estimation, and image restoration.

2. Typical tasks of computer vision

2.1Recognition

The classical problem in computer vision, image processing, and machine vision is that of determining whether or not the image data contains some specific object, feature, or activity.

Different varieties of the recognition problem are described in the literature:

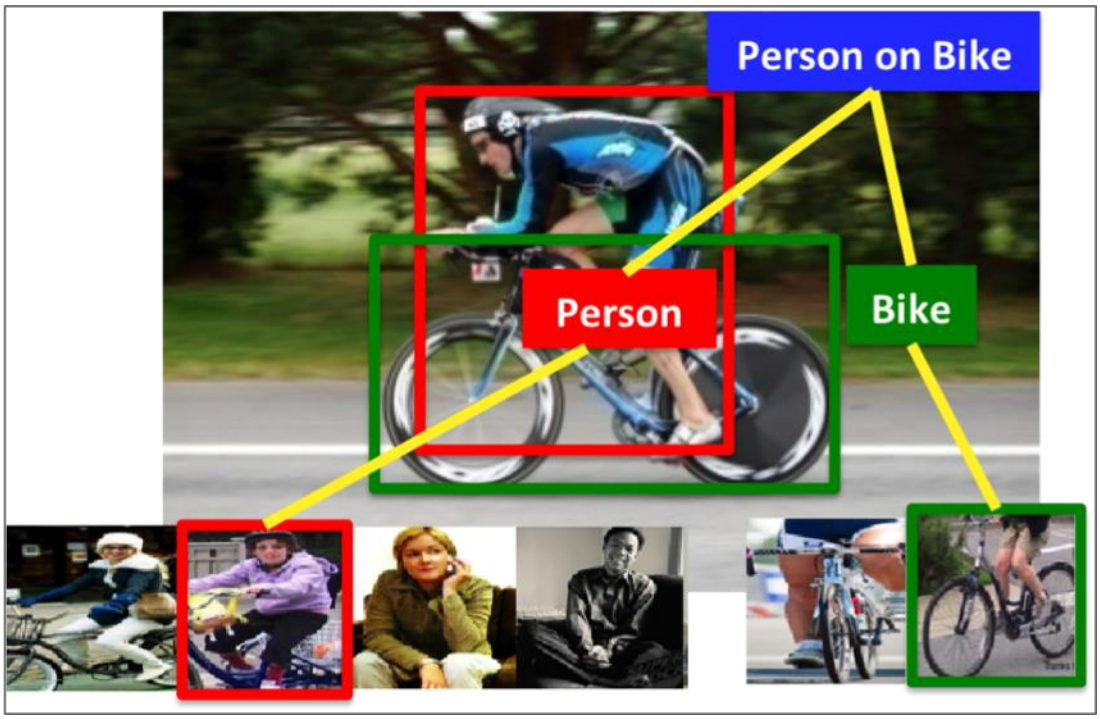
2.1.1 Object recognition (also called object classification)

Task (within computer vision) of finding and identifying objects in an image or video sequence.

Humans recognize a multitude of objects in images with little effort, despite the fact that the image of the objects may vary somewhat in different viewpoints, in many different sizes and scales or even when they are translated or rotated. Objects can even be recognized when they are partially obstructed from view.

This task is still a challenge for computer vision systems. Many approaches to the task have been implemented over multiple decades.

One or several pre-specified or learned objects or object classes can be recognized, usually together with their 2D positions in the image or 3D poses in the scene.



2.2 Identification

An individual instance of an object is recognized.

Examples include identification of a specific person's face or fingerprint, identification of handwritten digits, or identification of a specific vehicle.

2.3 Detection

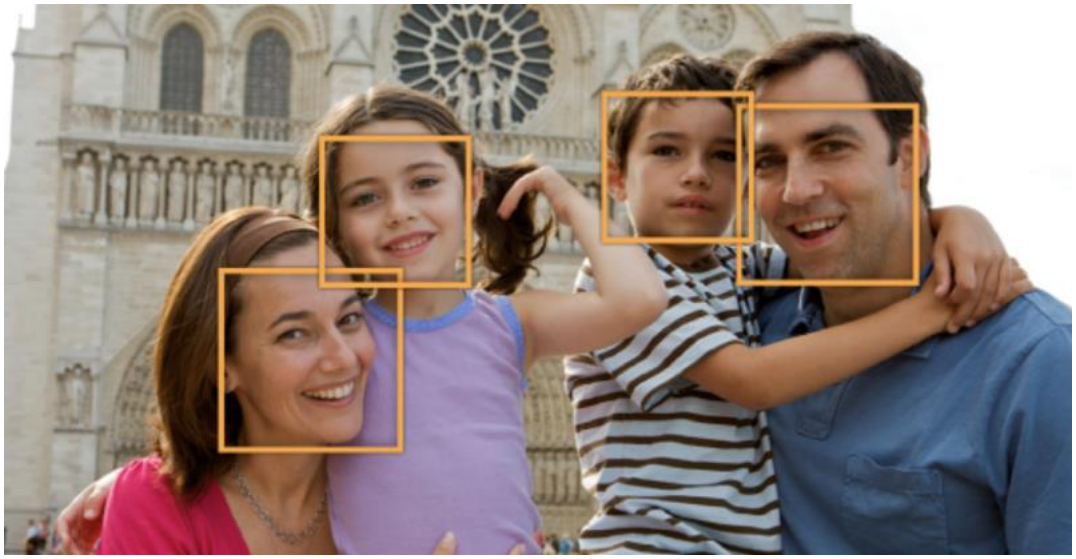
a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.

Well-researched domains of object detection include face detection and pedestrian detection.

Object detection has applications in many areas of computer vision, including image retrieval and video surveillance. The image data are scanned for a specific condition.

Examples include detection of possible abnormal cells or tissues in medical images or detection of a vehicle in an automatic road toll system.

Detection based on relatively simple and fast computations is sometimes used for finding smaller regions of interesting image data which can be further analyzed by more computationally demanding techniques to produce a correct interpretation.



3. Motion analysis

Several tasks relate to motion estimation where an image sequence is processed to produce an estimate of the velocity either at each points in the image or in the 3D scene, or even of the camera that produces the images. Examples of such tasks are:

3.1**Tracking**

Tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing.

Video tracking can be a time consuming process due to the amount of data that is contained in video. Adding further to the complexity is the possible need to use object recognition techniques for tracking, a challenging problem in its own right.

Following the movements of a (usually) smaller set of interest points or objects (*e.g.*, vehicles or humans) in the image sequence.

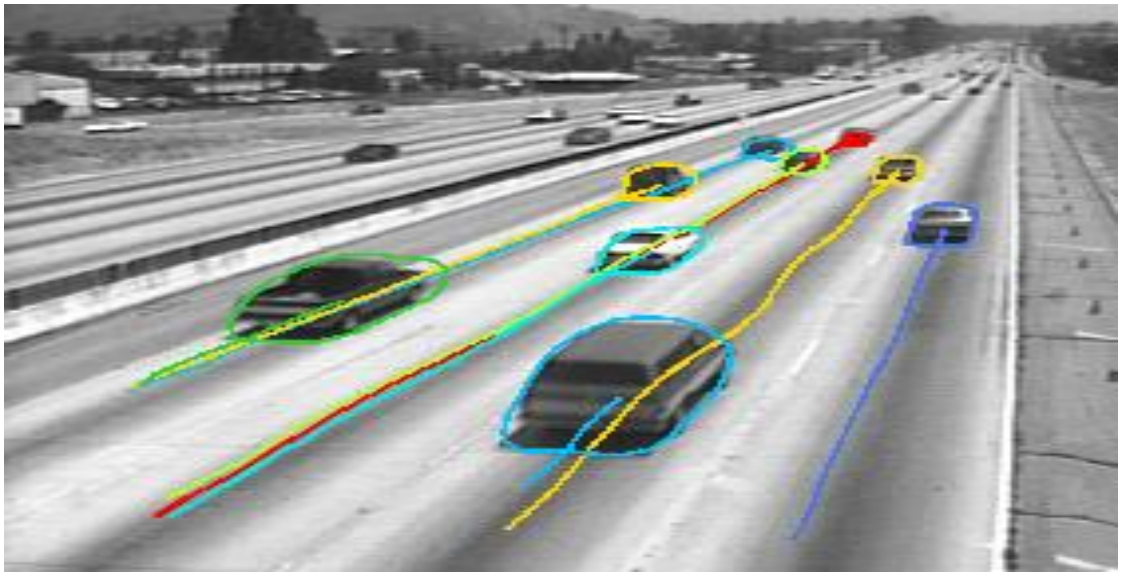
Objective

The objective of video tracking is to associate target objects in consecutive video frames. The association can be especially difficult when the objects are moving fast relative to the frame rate. Another situation that increases the complexity of the problem is when the tracked object changes orientation over time. For these situations video tracking systems usually employ a motion model which describes how the image of the target might change for different possible motions of the object.

Examples of simple motion models are:

When tracking planar objects, the motion model is a 2D transformation (affine transformation or homography) of an image of the object (e.g. the initial frame).

When the target is a rigid 3D object, the motion model defines its aspect depending on its 3D position and orientation.



4. Computer vision system methods

The organization of a computer vision system is highly application dependent. Some systems are stand-alone applications which solve a specific measurement or detection problem, while others constitute a sub-system of a larger design which, for example, also contains sub-systems for control of mechanical actuators, planning, information databases, man-machine interfaces, etc. The specific implementation of a computer vision system also depends on if its functionality is pre-specified or if some part of it can be learned or modified during operation. Many functions are unique to the application. There are, however, typical functions which are found in many computer vision systems.

4.1 Detection/segmentation – At some point in the processing a decision is made about which image points or regions of the image are relevant for further processing. Examples are

Selection of a specific set of interest points

Segmentation of one or multiple image regions which contain a specific object of interest.

4.2 High-level processing – At this step the input is typically a small set of data, for example a set of points or an image region which is assumed to contain a specific object. The remaining processing deals with, for example:

Image recognition – classifying a detected object into different categories.

4.3 Decision making: Making the final decision required for the application, for example:

Match / no-match in recognition applications

3.2 Intro to Opencv

OpenCV is an open source computer vision library. The library is written in C and C++ and runs under Linux, Windows and Mac OS X. There is active development on interfaces for Python, Ruby, Matlab, and other languages.

OpenCV was designed for computational efficiency and with a strong focus on real-time applications. OpenCV is written in optimized C and can take advantage of multicore processors. If you desire further automatic optimization on Intel architectures [Intel], you can buy Intel's Integrated Performance Primitives (IPP) libraries [IPP], which consist of low-level optimized routines in many different algorithmic areas.

OpenCV automatically uses the appropriate IPP library at runtime if that library is installed. One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand-inhand, OpenCV also contains a full, general-purpose Machine Learning Library (MLL).

This sub library is focused on statistical pattern recognition and clustering. The MLL is highly useful for the vision tasks that are at the core of OpenCV's mission, but it is general enough to be used for any machine learning problem.

3.3 Object tracking and calculating distance

Overview

the robot search for objects which take a circle shape then detect it and recognize the object`s color by assistant cases i gave to it , then tracking the object with determining its position (x ,y) in the camera`s capture frame , and determine the distance between the camera and the object according to object`s size.

Implementation

Include C++ and opencv libraries

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <math.h>
#include <deque>
#include "opencv\cv.h"
#include "opencv\highgui.h"
#include "opencv\cxcore.h"
```

Struct CvSeq with parameters (Capture frame, storage element for Hough circles)

```
CvSeq* getCirclesInImage(IplImage*, CvMemStorage*, IplImage*);
```

Declare global constant values

```
const int HISTORY_SIZE = 5;
const int X_THRESH = 15;
const int Y_THRESH = 15;
const int R_THRESH = 20;
const int MATCHES_THRESH = 3;
const int HUE_BINS = 32;
```

- Prepare the camera
- the images you bring out of the camera
- Open the camera

```
CvCapture *capture = 0;
IplImage* frame = 0;


capture = cvCaptureFromCAM(0);
if (!capture) {
    printf("Could not connect to camera\n");
    return 1;
}

frame = cvQueryFrame(capture);
```

- Create output window
- Used as storage element for Hough circles
- Grayscale image

```
cvNamedWindow("processed_video", CV_WINDOW_AUTOSIZE);

CvMemStorage* storage = cvCreateMemStorage(0);

IplImage* grayscaleImg = cvCreateImage(cvSize(640, 480), 8, 1);
```

- Query for the next frame

```
frame = cvQueryFrame(capture);
if (!frame) break;
```

- Iterate through the list of circles found by cvHoughCircles()
  Calculate the distance assistant by size

```
for (int i = 0; i < circles->total; i++) {
    int matches = 0;
    float* p = (float*)cvGetSeqElem(circles, i);
    float x = p[0];
    float y = p[1];
    float r = p[2];
    float size = 2 * (22 / 7)* r;
    float eq = (7499.7 / (2 * (22 / 7)* r));
    cout << "X=" << p[0] <<"  " "Y=" << p[1] <<"  " "Distance = " << pow(eq, (1 / 0.915)) << e

    if (x - r < 0 || y - r < 0 || x + r >= frame->width || y + r >= frame->height) {
        continue;
    }
```

- Start matching the points at ROI (region of insert) and determine the color of this region assistant by some case

```
if (matches > MATCHES_THRESH) {
    cvSetImageROI(frame, cvRect(x - r, y - r, 2 * r, 2 * r));
    IplImage* copy = cvCreateImage(cvSize(2 * r, 2 * r), frame->depth, 3);
    cvCvtColor(frame, copy, CV_BGR2HSV);
    IplImage* hue = cvCreateImage(cvGetSize(copy), copy->depth, 1);
    cvCvtPixToPlane(copy, hue, 0, 0, 0);
    int hsize[] = { HUE_BINS };
    float hrange[] = { 0, 180 };
    float* range[] = { hrange };
    IplImage* hueArray[] = { hue };
    int channel[] = { 0 };
    CvHistogram* hist = cvCreateHist(1, hsize, CV_HIST_ARRAY, range, 1);
    cvCalcHist(hueArray, hist, 0, 0);
    cvNormalizeHist(hist, 1.0);
    int highestBinSeen = -1;
    float maxVal = -1;
    for (int b = 0; b<HUE_BINS; b++) {
        float binVal = cvQueryHistValue_1D(hist, b);
        if (binVal > maxVal) {
            maxVal = binVal;
            highestBinSeen = b;
        }
    }
```

- Cases of colors

```
    cvResetImageROI(frame);
    const char *color;
    switch (highestBinSeen) {
    case 2: case 3: case 4:
        color = "orange";
        break;
    case 5: case 6: case 7: case 8:
        color = "yellow";
        break;
    case 9: case 10: case 11: case 12:
    case 13: case 14: case 15: case 16:
        color = "green";
        break;
    case 17: case 18: case 19: case 20:
    case 21: case 22: case 23:
        color = "blue";
        break;
    case 24: case 25: case 26: case 27:
    case 28:
        color = "purple";
        break;
    default:
        color = "red";
    }
    char label[64];
    sprintf(label, "color: %s", color);
    drawCircleAndLabel(frame, p, label);
}
```

- Houghification
- Convert to a single-channel, grayspace image
- Gaussian filter for less noise
- Detect the circles in the image

```
cvCvtColor(frame, grayscaleImg, CV_BGR2GRAY);

cvSmooth(grayscaleImg, grayscaleImg, CV_GAUSSIAN, 7, 9);

CvSeq* circles = cvHoughCircles(grayscaleImg,
    storage,
    CV_HOUGH_GRADIENT,
    2,
    grayscaleImg->height / 4,
    200,
    100);
return circles;
```

- Draw the circle on the original image

```
void drawCircleAndLabel(IplImage* frame, float* p, const char* label) {

    CvFont font;
    cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX, 1, 1, 0.0, 1, 8);
    cvCircle(frame, cvPoint(cvRound(p[0]), cvRound(p[1])), cvRound(p[2]), CV_RGB(255, 0, 0), 3, 8, 0);
    cvPutText(frame, label, cvPoint(cvRound(p[0]), cvRound(p[1])), &font, CV_RGB(255, 0, 0));
}
```

- Making thresholding for the frame to detect the object with circle shape

```
bool circlesBeHomies(float* c1, float* c2) {
    return (abs(c1[0] - c2[0]) < X_THRESH) && (abs(c1[1] - c2[1]) < Y_THRESH) &&
        (abs(c1[2] - c2[2]) < R_THRESH);
}
```

3.4 Finger number detection

This program is based on phases:

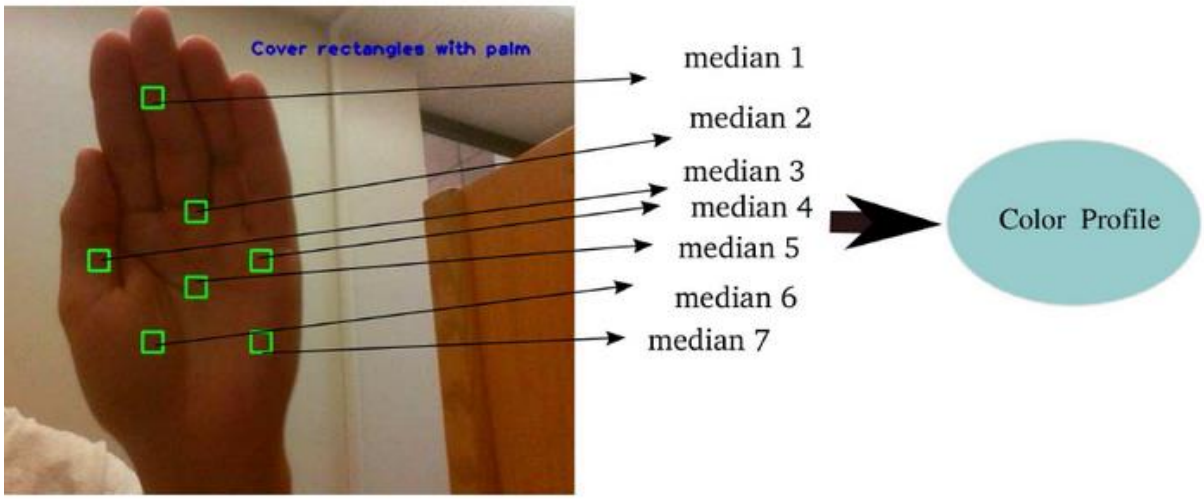
First sampling color to get hand color

Hand tracking in this program is based on color recognition as a result of that the program is initialized by sampling color from hand.

This applied by adding 7 rectangles in different places on the middle of the screen, we used function called WaitForPalmCover () .

Its steps are simple, Set seven small rectangles position in different places in middle of the screen, Loop 50 time until user set his hand during 50 loop .

We draw the seven rectangles.



Each triangle has its own color so we will work on getting the average of all these colors in order to determine the skin color.

Second step we get average of colors:

We are going to use function average ( ) which will accomplish this mission, It has simple steps:

- Change the color of the image to HSL (HUE, SATURATION , LIGHTNESS)
- For each rectangle loop for each pixel inside it
- For each rectangle generate vector of colors from each pixel
- For each rectangle get the middle color number for each channel one for Hue and one for SATURATION, one for LIGHTNESS

This is the output of getting average color.

Third step is to initialize track bar for each, We are going to use function initTrackbars( ).

This function work is easy is to set value for upper and lower value for each one of the three channels HUE and SATURATION, LIGHTNESS ,And this is done for each triangle.

```
c_lower[i][0] = 12;
c_upper[i][0] = 7;
c_lower[i][1] = 30;
c_upper[i][1] = 40;
c_lower[i][2] = 80;
c_upper[i][2] = 80;
```

These values are not authorized yet this will be clearly explained next step.

Forth step is for skinning hand:

Use function called produce Binaries ( ).

First thing done in this function is to normalize color.

Generally this is done by setting lower bound color and upper bound color for each channel HUE and SATURATION, LIGHTNESS for each triangle that was set before but this time it is set based on average color explained before in second step.

Explanation:

If the average color for each triangle mines each lower bound color for each Channel is less than 0 then set that lower bound equal to average color, If the average color for each triangle plus each upper bound color for each Channel is higher than 255 then set that upper bound equal to 255 – average color, This is normalization.

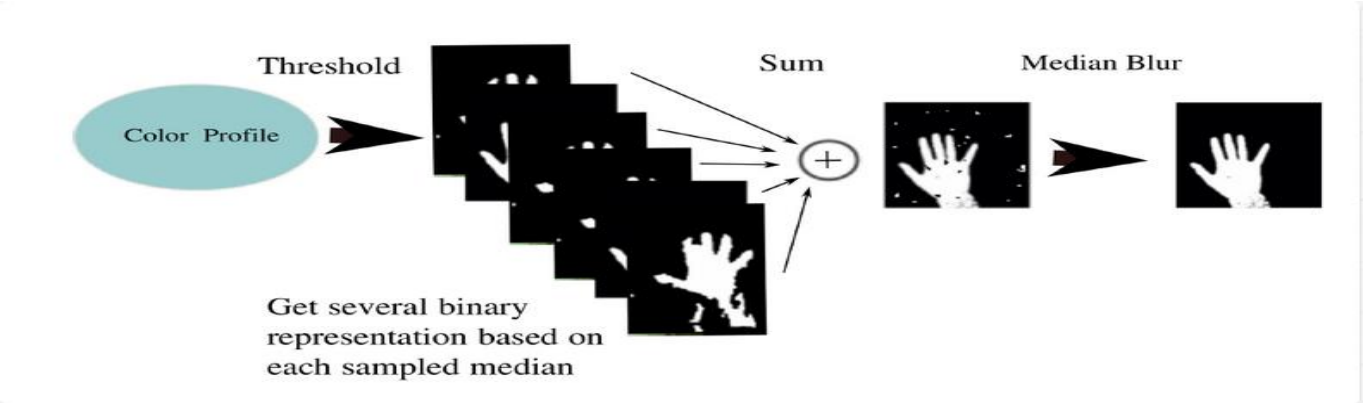
Note I used this way to get binary image as the difference is too small between average and the expected numbers that had been set in initializing track bar.

Second step is to set it in images:

We will make lower bound scalar for contain all lower bound of the three channels HSL, And upper bound scalar also contains all upper bound of the three channels HSL, Then set mat image contain the range of this lower and upper bounds, This step is done for each triangles average color

Third step is to get the sum of the seven produced binary image:

Final step of skinning hand is applying the median blur filter the Median filter is a common technique for smoothing. Median smoothing is widely used in edge detection algorithms because it preserves edges while removing noise.



This image is easily explain everything.

Fifth step CONTOURS

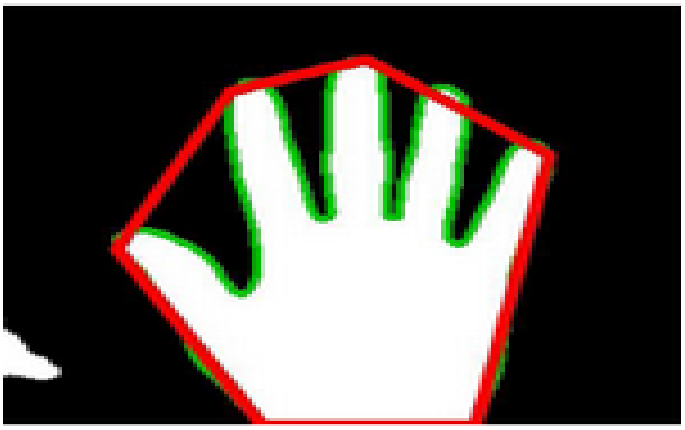
First before getting directly to algorithm I'll explain some functions had been used.

In order to be able to use contours we apply it on binary image as I did in skinning hand step the contours follow the 1 s pixels, In case of our program it is applied on hand.

First function to explain is find contour is used to cover all white spaces in image as shown.

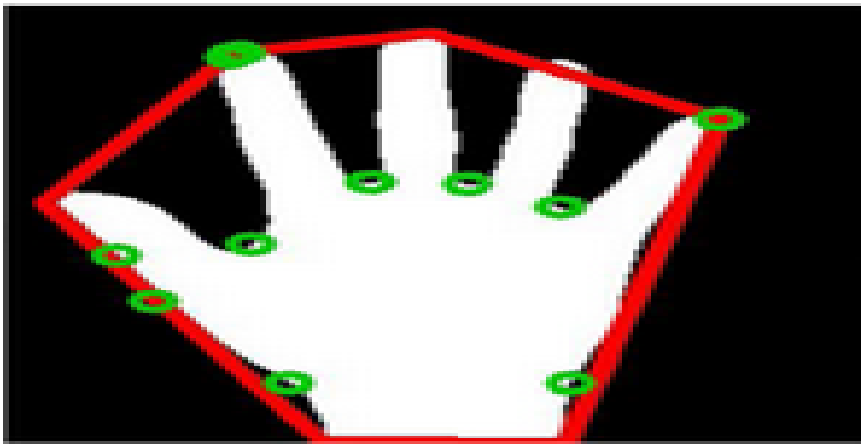
And this is an important function that is going to help getting things like biggest contour which will help finding the one finger condition which will be explained later.

Second function is convex hull: which is the smallest polygon enclose all points, So this function`s job is to search for edges to complete a full polygon contains inside all the 1s pixels and it is done as shown.

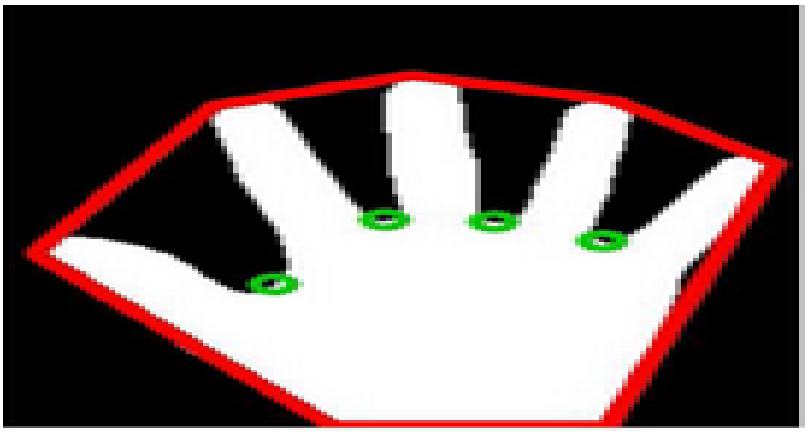


So it give us a sequence of vertices in counter clock wise order.

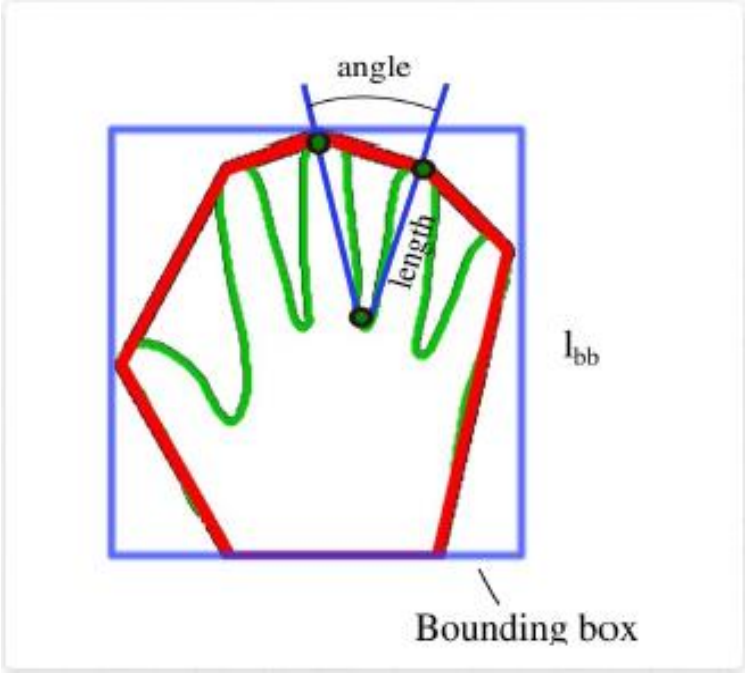
Third function is approxpolyDP: it is nearly similar to the one before put it uses less vertices as shown in image as circles.



Final using convexity defects function it set a vertex between each two fingers based on the angle between two fingers.



There are two condition to prevent convexity defects are Angle > 95 or length of the finger is less than 1/6 of hand height.



We calculate the angle using the following algorithm:

Assume the length of fingers are d1 , d2 and each point as shown have x , y, Say the top left finger's name A, Top right B, Middle point M, First we calculate :

(A.x - M.x)*(B.x - M.x) + (A.y - M.y)*(B.y - M.y) = num1

D1 * D2 = D

Angle = Shift cos(num1/D)

Now let us start in program algorithm

As beginning we call function named Make Contours (), Using this function we search on the index of the biggest contour size, to Using biggest contour we are going to use several function

- First applying boundingRect () to draw flexible rectangle cover hand.
- Second applying convex hull () function.
- Third applying approxpolyDP () function .
- Forth and most important convexity defects ().

After all these functions we are going to use function named eliminate defects (), Used to check defects between each two fingers .

So We are going to check to things first that the finger length is bigger than 1/5 of total hand And the angle between the two fingers is less than 95 degree, Then function removeRedundantEndPoints () is going to be applied. It is used for removing endpoint of convexity defects if they are at the same fingertip, By checking to things if distance between two fingers is less than 1/6 of hand width, If not then check the distance between the end of the first two and the start of the next if it is less than 1/6 of hand width then they are on the same finger so it is set as end finger of first two.

Now we are going to find finger tips in order to know the number of fingers, So we are going to call function getFingerTips ()

- First clear finger tips if it contains any data.
- Loop on each defect and for each one get points of first finger and second and point that represent angle in between.
- Set these two fingers as beginning and each other defect found add one finger.
- If no defect found then it might be only one finger or no zero number in the two cases there in no defect so the solution is done by.
- Calling function named Checkforonefinger ( ), This function will do two things
- Search the contour to find the highest y point in hand.
- Then search again all the contour and for each point if all of these condition came true then there is no finger else there is one.
    1. Case 1 that point. Y of contour is lower than highest y plus the minimum finger length which is 1/6 of hand
    2. Case 2 if that point. Y of contour is not equal to highest point. y
    3. Case 3 if that point. X of contour is not equal to highest point. x
    4. Then draw finger tips and contours and defect positions

Finally making sure of the numbers in image, It's is by checking the number for at least 12 time before accept it.

Chapter 4
Conclusions and Future improvements

4.1 Applications of Robotic arm

a. The robotic arm can be designed to perform any desired task such as welding, gripping, spinning etc., depending on the application.

b. In space: the space shuttle Remote Manipulator System has multi degree of freedom robotic arms that have been used to perform a variety of tasks such inspections the Space.

c. In medical science: "Neuroarm" uses miniaturized tools such as laser scalpels with pinpoint accuracy and it can also perform soft tissue manipulation, needle insertion, suturing, and cauterization.

d. Welding Applications such as:
    - Arc Welding
    - Electron Beam
    - Flux Cored Welding
    - Resistance Welding
    - Spot Welding

e. Material Handling Applications
    - Material Handling
    - Part Transfer
    - Pick and Place

4.2 Importance:

Robot arms play an important role in the following:-

Egypt Aluminum Company

Cement Factory

Spinning Factory

Paper factory

4.3 Future Importance of Robotic Arm

In the future, we seek to use and apply robot arms in the following important applications such as:-

Applying at Pharmaceutical Factory

Development of hospitals by using robot in Surgeries

Replacement of human arm by artificial limbs for handicapped.

Extinguish the fire

Development of factories

Detecting bombs

4.4 References

OpenCV official site tutorial

http://docs.opencv.org/doc/tutorials/tutorials.html

OpenCV Practical

http://link.springer.com/book/10.1007%2F978-1-4302-6080-6

Arduino official site examples and questions

https://www.arduino.cc/en/Tutorial/HomePage

Arduino workshop [john boxall]

http://www.amazon.com/Arduino-Workshop-Hands-On-Introduction-Projects/dp/1593274483

Arduino cookbook [Michael Margolis]

https://books.google.com.eg/books/about/Arduino_Cookbook.html?id=raHyKejOBF4C&hl=ar

Arduino + android projects for the evil genius

http://www.free-engineering-books.com/2013/03/arduino-android-projects-for-evil.html

Raspberry pi official site

https://www.raspberrypi.org/resources/make/

Raspberry pi cookbook [Simon monk]

https://www.adafruit.com/images/product-files/2274/2274sampler.pdf

Simply raspberry pi [Abdallah Eid]

http://simplyarduino.com/?page_id=851

Getting started with raspberry pi [matt Richardson]

http://shop.oreilly.com/product/0636920023371.do

Helwan University

Faculty of computers and information

Auto Tracking Robot with Arm

Graduation Project

Team Work:

Mohamed Gomaa zaafan	Team Leader	ID: 20110
Eslam Sobhy Mohamed		ID: 20110098
Eslam Ibrahem Ali		ID: 20110089
Mohamed Mokhtar Taha		ID: 20110369

Under Supervision Of

Prof: Aliaa Youssef

Dr: Maged wafi