PROJECT NO. : 23

Development of Humanoid Robot for Security Purpose

A Project Report
Submitted in partial fulfilment of
requirement for award of the degree of

Bachelor of Technology
in
Electrical Engineering

Submitted by:

ABHYUDAI NOUNI (10115005)
AVNISH KUMAR (10115029)
SWAPNIL TANEJA (10115107)

Guided by:
Prof. Yogesh V. Hote

Department of Electrical Engineering
Indian Institute of Technology Roorkee
Roorkee – 247 667, INDIA

May 2014

DECLARATION BY THE CANDIDATES

We hereby certify that the work which is presented in this project entitled "Development of Humanoid Robot for Security Purpose" in partial fulfilment for the award of degree of Bachelor of Technology (Electrical Engineering) submitted to the Department of Electrical Engineering, Indian Institute of Technology Roorkee is an authentic record of our own work carried out during the period from August, 2013 to May, 2014 under the guidance of Prof. Y. V. Hote, Department of Electrical Engineering, I.I.T. Roorkee.

The matter embodied in this project report to the best of our knowledge has not submitted for the award of any other degree elsewhere.

May 8, 2014

Abhyudai Nouni Avnish Kumar Swapnil Taneja

ACKNOWLEDGEMENT

It is with a deep sense of gratitude and indebtedness that we express my sincere gratefulness to our project guide **Prof. Yogesh V. Hote**, Assistant Professor, Department of Electrical Engineering, Indian Institute of Technology, Roorkee under whose able guidance, constant supervision and encouragement, this work has been accomplished. We thank him for taking time out of his busy schedule and aiding us with his priceless suggestions, encouragement and cooperation. Without his guidance and mentorship, this work would never have reached its completion.

(Abhyudai Nouni)
(Avnish Kumar)
(Swapnil Taneja)

CONTENTS

PREFACE

This report describes the work that was carried out under the guidance of Dr. Y.V. Hote for the design and construction of a Humanoid robot for security purpose. This Humanoid robot will be later kept in the Control and Robotics Lab and can be adapted to other uses also.

The development of Humanoid robots is the latest trend in Robotics as well as Artificial Intelligence. In today's world, Machine Learning is one of the most researched and discussed topics. The combination of Machine Learning, Computer Vision, Artificial Intelligence and several such inter-disciplinary fields culminate into futuristic projects like Augmented Reality and Humanoid robots.

This report explains the intricacies involved in making a simplified version of a humanoid robot with features like Hands and Shoulder-joints, Neck and Face movements, Face recognition ability, Proximity Sensing ability, Locomotion and automation. These topics are discussed in this report in detail as separate chapters are based on the basics of the components involved in making and designing such a humanoid robot.

Starting with the overview of the system and the initial design stages, the report discusses the idea and the concept behind the robot even before fabrication. In Chapter 2, Principle of Robotics and Robot Dynamics, the report discusses how components like motors and wheels were chosen for this robot based on specific calculations and requirements. Chapter 3 discusses the theory of locomotion which enables the ground movement of the Robot. The fourth Chapter is based on the Human-like Gestures of the robot achieved by use of servo motors. The fifth Chapter discusses how the Microcontroller is often called the heart of the Robot and how it automatically controls all the other components of the robot. Chapter 6 explains how Sensors work and how they are used to detect proximity of objects or humans to the Robot, thereby providing a "sixth sense" to the robot. The last Chapter discusses the theory of Face Recognition and how it can be used to provide "Vision" to the Robot.

The Arduino integrated development environment (IDE) is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the Wiring projects. It includes a code editor and is capable of compiling and uploading programs to the Microcontroller board with a single click

Microsoft Visual C++ (often abbreviated as MSVC or VC++) is a commercial (free version available), integrated development environment (IDE) product from Microsoft for the C, C++, and C++/CLI programming languages. It features tools for developing and debugging C++ code.

Appendix A and B contain Images of the System and the Codes for Arduino and Visual C++ respectively.

List of Figures

List of Tables

Chapter 1

Introduction

System Overview

The Prototype system can be divided into five sub-systems on the basis of their respective functioning. These sub-systems are conceptually very different from each other and use varied offshoots of engineering. These are interfaced together as a system.

Following are the sub-systems:

i) Locomotion

The system is provided with locomotion with the help of 2 DC Motors. For controlling the speed as well as direction of motors, it is interfaced with microcontroller like Arduino. The Arduino controls the DC Motors via the Motor Driver. The Arduino is pre-loaded with functions for controlling the speed as well as the direction of the motors. The Motor Speed is controlled via PWM pins on the Arduino which employ Pulse Width modulation to supply a specific percentage of the Battery Voltage Input to the Motor Driver.

ii) Gestures

The robot uses servos to perform various gestures. These gestures can be divided into four main categories, though there is tremendous scope of increasing the gestures. The gestures were tried to be kept simple and human-like. There are in all 4 servo motors, of which two are located at the shoulder joint and two at the neck for 2-dimensional movement of the neck.

(1) Hand motion while walking
(2) Handshake movement
(3) Neck Rotation (Horizontal)
(4) Neck Flexion (Vertical)

For getting all this gestures, Servos are interfaced with microcontroller like Arduino.

iii) Automatic Control

The system is provided with automatic control, through a microcontroller. The Microcontroller used in this project is Arduino UNO. DC Motors, Servo Motors and sensors all controlled by microcontroller by interfacing with it.

iv) Proximity Sensing

GP2Y0A02YK IR Range Sensor is an Infrared distance measuring sensor unit capable of measuring distance from 20 to 150cm. It is composed of an integrated combination of PSD (position sensitive detector), IRED (infrared emitting diode) and signal processing circuit.

v) Face recognition and Tracking

A Webcam interfaced with Arduino microcontroller via USB provides visual input which is processed and analysed using face detection algorithms. Once a face is detected, the robot tracks the face as long as the person's face lies in the field of view of the webcam.

Block Diagram:

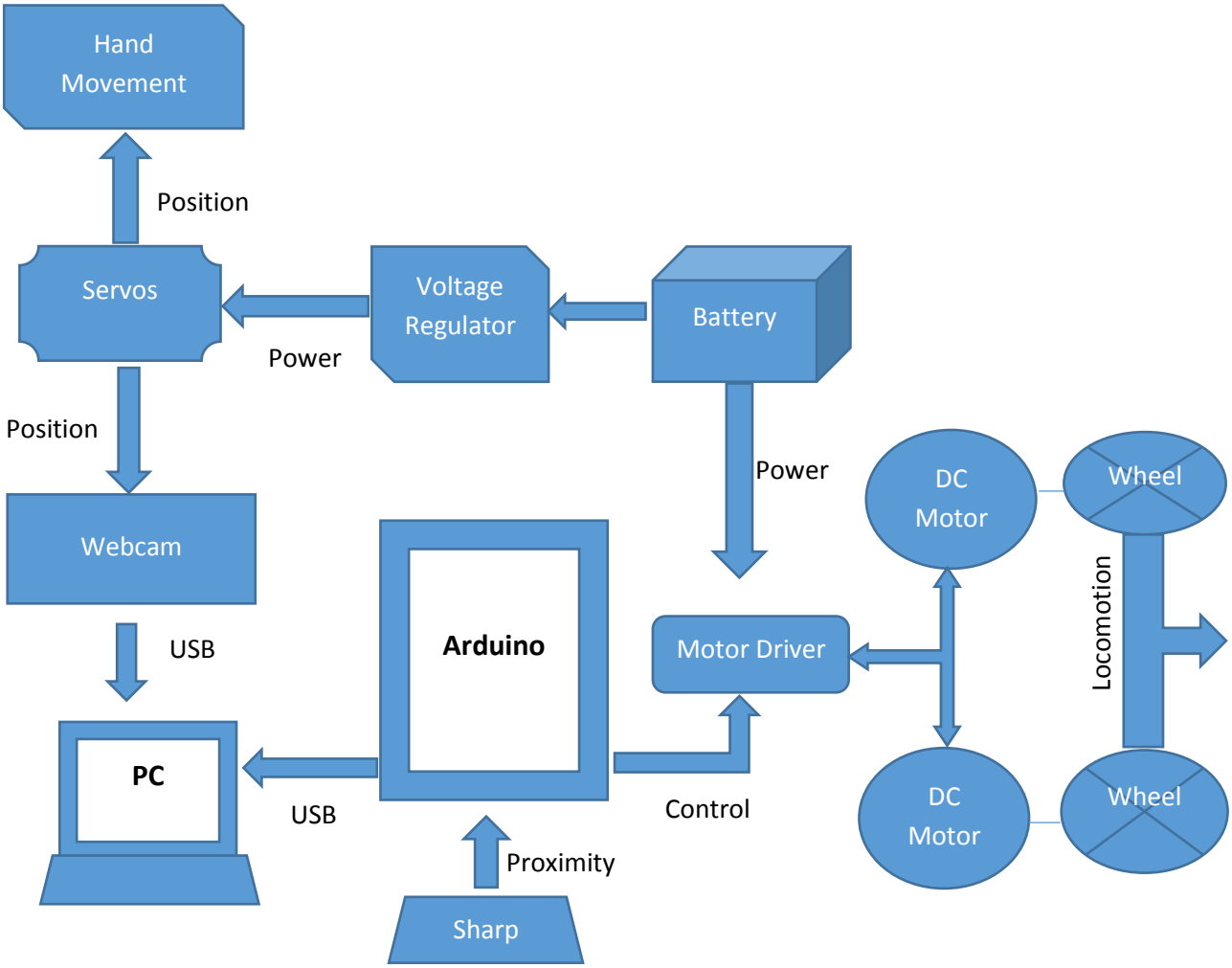


Fig. 1.1: Block Diagram of the System

Initial Design:

This design was made using SolidWorks software which is a very popular software for 3-D modelling, CAD and similar designing purposes. This model resembles the actual humanoid robot that we initially intended to design. The Face shown here is only for display purposes. The body is made of wood.

Fig. 2.2: Initial Design made in SolidWorks Software

Physical features:

1. Height = 75 cm approx.
2. Weight = 10 kg (approx.) including motors and electronics.
3. 2 Hands (Shoulder-joints)
4. Neck (mobile)
5. 2 Legs (fixed)
6. A base for locomotion (supported on 2 wheels and 2 castor wheels)

Chassis:

The main material which was to be used for the body of the robot was Plywood of 12 mm thickness which was machined and shaped accordingly for three parts – the torso, the legs and the base.

All the major electronic components – microcontroller, circuitry, motors, pulleys etc. are inside the torso of the robot.

Additionally there are two hands attached using shoulder joints. Also there is a face attached using neck joint. All three of these joints are movable and will be rotated using motor shafts connected using a pulley assembly.

The base is the component responsible for locomotion of the robot. The locomotion is wheeled differential type.

Problems with this Design:

The fabrication of the bot with the material as wood was started. The base and the torso was made. The calculations were done according to an estimated weight of 10kg. But, the weight of the robot exceeded that limit excluding electronic components like wires, motor driver, microcontroller, servos etc. This meant that the DC Motors which had been selected based on those calculations would not be able to supply enough Power to move the bot. Also, the Servo motors would draw large amounts of current if the hands were also to be made of wood. The design had a major flaw. Partially this was due to a compromise between robustness and light-weightiness while choosing the thickness of the plywood.

New Material for Chassis:

To correct this, a new material had to be bought to replace the wood. After discussions and suggestions from various people, Acrylic was chosen to be the material which would comprise the main chassis or body of the robot. The advantages of Acrylic over Plywood are several. It is more robust, much lighter and more presentable. Further it was machined to a greater accuracy and was not prone to wear and tear or damage from water.

Acrylic is used primarily for decorative or functional applications, such as picture frames or salad bowls. It can also be used in low-stress structural applications, such as robot bodies, as long as limitations for weight and impact shock are observed.

Components:

Base:

The Drive is a differential type with two driven wheels and two castor wheels. Two driving wheels are attached to motor shafts and the other two are ball castor wheels. The reason for 1 extra castor wheel is because it avoids toppling of the whole structure.

A differential drive is the most basic drive, which consists of two sets of wheels that can be driven independently. This is the most commonly used form of locomotion system used in robots as it's the simplest and easiest to implement. It has free moving wheels in the front (and/or in the back) accompanied with a left and right wheel. The two wheels are driven by different motors.

Motors:

Motors are required for locomotion as well as for mobile joints like neck and shoulders. Overall there will be 5 DC motors. The motors are different in specifications based on the purpose for which they are being used.

The robot has 2 DC motors for driving the wheels on the base – High speed (300 rpm), medium torque with 12 V DC supply.

There are 2 servo motors for hands at the shoulder joints with 5V Supply.
In addition, there are 2 servo motors for neck rotation in 2 directions also operating with maximum speed at 5V.

DC MOTOR	**SERVO MOTOR**
	
Specifications • RPM: 300 at 12V • Voltage: 4V to 12V • Stall torque: 12Kg-cm at maximum limited stall current of 4Amp. • Shaft diameter: 6mm • Shaft length: 18mm • Gear assembly: Spur • Brush type: Carbon • Motor weight: 160gms	**Specifications** • Dimension: 40mm x 20mm x38mm • Torque: 4kg-cm at 6V • Stall current: 900mA • Idle current: 5mA • Operating voltage: 4.8V to 6V • Motor weight: 50gms • Operating speed: 0.15sec/60 degree • Temperature range: -20°C to 55°C • 0.6 ms for 0 degree Rotation • 2.2 ms for 180 degree Rotation

Table 1.1: Specification and Diagram of Motors used

Voltage Supply:

The Voltage Supply is a 12V rechargeable Lithium Polonium Battery. The specifications are 3 Cell, 11.1V, 3300mAh, 20C. There is also a charger for the battery and the battery needs to be recharged via a charger connected to normal household AC Mains Supply.

Fig. 1.3: 12V Li-Po Rechargeable Battery

Fig. 1.4: Charger for 12V Li-Po battery

Voltage Regulation:

7805 is a voltage regulator integrated circuit. It is a member of 78xx series of fixed linear voltage regulator ICs. The voltage source in a circuit may have fluctuations and would not give the fixed voltage output. The voltage regulator IC maintains the output voltage at a constant value. The xx in 78xx indicates the fixed output voltage it is designed to provide. 7805 provides +5V regulated power supply. Capacitors of suitable values can be connected at input and output pins depending upon the respective voltage levels. This is used to supply 5V to servos from the 12V battery used for DC Motor Supply.

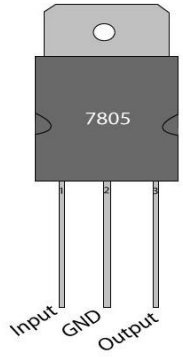


Fig. 1.5: 7805 IC

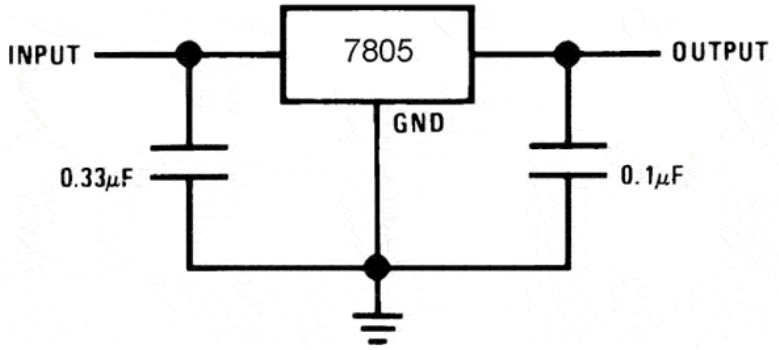


Fig. 1.6: Connection for 7805 Voltage Regulator

Components Summary:

1. DC Motors – 2 dc motors for wheels – High speed , medium torque - (300 rpm, 12V, 12kg-cm
2. Servo Motors – 4 Servo motors for Neck and Hands (4.8V-6V, 4kg-cm at 6V)
3. LiPo rechargeable battery + charger (12 V).
4. Arduino Uno microcontroller
5. Sharp IR Sensor (10 cm to 80 cm), distance sensor
6. Motor Driver (L298D 4A)
7. Voltage regulator 7805 IC for supplying 5V.
8. Webcam

Applications:

This humanoid robot was designed with the objective of security in a location like a closed room wherein there will be many people entering and leaving. The robot will be strategically placed at a location wherein it can clearly view the entry point and hence record the face of the person.

It doubles up to welcome a person into a room as well as records an image of his face which is easily saved into the storage of a computer for recording purpose. It is friendly in the sense that it is human-like in appearance and is harmless for the most part. It also is smart and aware of its surroundings as it can sense presence of objects or obstacles in its path. Also, it can recognize faces of humans and track them which gives an appearance as if the humanoid is following you with his 'eyes'.

The security purpose of the bot can be employed in a lab setting wherein there is expensive and important equipment and it is not feasible to keep a person in charge of security, day and night. This robot serves a wonderful purpose, wherein it can record who enters (or leaves) the lab, at what time.

The face recognition software can be further tweaked to customize the database of faces in the storage of a PC, which would add new faces to the storage and also recognize faces which have previously been recorded.

The security purpose of this Humanoid can be further enhanced by adding a GSM Module which would send an SMS to a given mobile number every time a new or unknown person enter the room.

Chapter 2

Principle of Robotics

Introduction to Robotics

Robotics is a science that combines the technology and knowledge necessary to build robots. Due to the fast development of technology, the term robot, as an automatic machine that replaces humans, is not so clearly defined anymore. A robot is no longer just a humanoid robot, robotic hand at an auto assembly line, autopilot in aircraft, artificial intelligence built of living neurons or simple cleaning robot; it is also computer software that completes tasks meant for humans (for example, compiles reports). It is known that robots are built to replace humans at certain tasks. There are many reasons for this: dangerous working conditions, cheaper production, monotonous work may cause humans to err, new systems are so complex and time-critical, that automatic systems can make better decisions than humans.

Generations of Robotics:

The growth of robots can be grouped into robot generations, based on characteristics breakthroughs in robots capabilities. These generations are overlapping and include futuristic projections.

First Generation

The first generation robots are repeating, non-servo, pick-and-place, or point-to-point kind. The technology for these is fully developed and at present, about 80% robots in use in the industry are of its kind. It is predicted that these will continue to be in use for a long time.

Second Generation

The addition of sensing devices and enabling the robot to alter its movement in response to sensory feedback marked the beginning of second generation. These robots exhibit path control capabilities. This technological breakthrough came around 1980s and is yet not matured.

Third Generation

The third generation is marked with robots having human like intelligence. The growth in computers led to high-speed processing of information and, thus, robots also acquired artificial intelligence self-learning, and conclusion drawing capabilities by past experience. Online computations and control, artificial vision and active force/torque interaction with the environment are the significant characteristics of these robots. The technology is still in infancy and has to go a long way.

Fourth Generation

This is futuristic and may be a reality only during this millennium. Prediction about its features is difficult, if not impossible. It may be a true android or an artificial biological robot or super humanoid capable of producing its own clones.

Three Laws of Robotics

Asimov also proposed his three "Laws of Robotics" and he later added a 'zeroth law'.

Law Zero

A robot may not injure humanity or through inaction, allow humanity to come to harm.

Law One

A robot may not injure a human being or through inaction, allow a human being to come to harm, unless this would violate a higher order law.

Law Two

A robot must obey orders given it by human beings, except where such orders would conflict with a higher order law.

Law Three

A robot must protect its own existence as long as such protection does not conflict with a higher order law.

Robot dynamics

Introduction to Mechanical Engineering Theory, Dynamics

While **statics** is the study of structures at a fixed point in time, dynamics is the study of structures over a period of time. Basically statics studies things that don't move, while dynamics studies things that do. Statics is concerned with moments, forces, stresses, torque, pressure, etc. Dynamics is concerned with displacement, velocity, acceleration, momentum, etc. Forces generated or required for a moving robot can be calculated and/or optimized.

Displacement and Velocity

We all know what velocity is, but how one can design a robot to go at a defined velocity? Of course we can put a really fast motor on robot and it will go fast enough. But if one can calculate it then this can be designed to go required speed without doubt, and the rest of the motor force can be left for torque. So how this can be done? For an example, suppose there is a wheeled robot that we want to run over old people with. We know from experiments that

old people can run at 3 feet per second. So what motor rpm do we need, and what diameter should wheels be?

Conceptually, every time wheel rotates an entire revolution, robot travels the distance equal to the circumference of the wheel. So the circumference is multiplied by the number of rotations per minute, and then the distance, robot travels in a minute can be calculated.

Velocity = circumference * rpm
Velocity = diameter * pi * rpm OR Velocity = 2 * radius * pi * rpm

For example, if motor has a rotation speed (under load) of 100rpm (determined by looking up the motor part number online) and one want to travel at 3 feet per second, then:

3 ft/s = diameter * pi * 100rpm
3 ft/s = diameter * pi * 1.67rps (rotations per second)
diameter = 3 ft/s / (3.14 * 1.67 rps)
diameter = 0.57 ft, or 6.89"

Robot Wheel Diameter vs Torque

One can notice that the larger the diameter of the wheel, or higher the rpm, the faster robot will go. But this isn't entirely true in that there is another factor involved. If robot requires more torque than it can give, it will go slower than calculated. Heavier robots will go slower. Now what we need to do is compare the **motor torque**, robot **acceleration**, and **wheel diameter**. These three attributes will have to be balanced to achieve proper torque.

Motor Torque and Force

High force is required to push other robots around, or to go up hills and rough terrain, or have high acceleration. As calculable with **statics**, just by knowing wheel diameter and motor torque, the force can be determined which robot is capable of.

Torque = Distance * Force
Distance = Wheel Radius
Force = Torque / Wheel Radius

Acceleration

But you also want to be concerned with acceleration. For a typical robot on flat terrain, one probably want acceleration to be about half of max velocity. So if robot velocity is 3 ft/s, we want acceleration to be around 1.5 ft/s^2. This means it would take 2 seconds (3 / 1.5 = 2) to reach maximum speed.

Force = Mass * Acceleration

There is one other factor to consider when choosing acceleration. If robot is going up inclines or through rough terrain, a higher acceleration due to countering gravity will be needed. If say robot was going straight up a wall, an additional 9.81 m/s^2 (32 ft/s^2) acceleration to counteract would be required. A typical 20 degree incline would require 11 ft/s^2..

It can be noted that motor acceleration and torque are not constants, and that motor acceleration will decrease as motor rotational velocity increases.

Robot Motor Factor

The robot motor factor (RMF) is simply a way by which one can do a quick calculation to optimize robot. Basically by combining and simplifying all the equations above into one big equation to help one choose the motor that best suits robot.

Torque * rps > = Mass * Acceleration * Velocity / (2 * pi)

RMF = Torque * rps

1) To use this equation, a set of motors which will work for robot should be taken and the torque and rps (rotations per second) for each should be noted down.
2) Then the two numbers together for each should be multiplied. This will be robot motor factor.
3) Next, the weight of robot should be estimated. For this basically the weight of all the parts should be added up.
4) Lastly, your desired velocity and acceleration should be chosen.
5) Both sides of the equation should be compared.

We have three DC motors:
Motor A: 2 lb. ft., 1rps => RMF = 2 lb. ft. rps
Motor B: 2.5 lb. ft., 2rps => RMF = 5 lb. ft. rps
Motor C: 2 lb. ft., 4rps => RMF = 8 lb. ft. rps

For our robot, we want a velocity of 3 ft./s(roughly 1m/s), an acceleration of 2 ft./s^2, and we estimate robot weight to be 5 lbs.

So RMF >= 5 lbs * 2 ft. /s^2 * 3 ft. /s / (2 * pi)

Therefore RMF >= **4.77** lb. * ft. * rps

So this means a motor with an RMF greater or equal to 4.77 is needed. Looking at the list, Motor B and C both will work. However Motor C is probably overkill, so it's just a waste of money. Therefore we would use Motor B. It should be noted that if none of the motors would work, one would have to either reduce weight, or go slower, or find another motor.

Note: if we convert rps to radians/sec, RMF can be measured in watts.

Calculating Wheel Diameter
So now what robot wheel diameter should be used? Going back to an earlier equation,

Velocity = diameter * pi * rps
OR
diameter = velocity / (pi * rps)

3 ft./s / (pi * 2/s) = wheel diameter = .48 feet = 5.73"

Now this is finished! Motor B, with a wheel diameter of 5.73" can be used.

Although the above equations are intended for robot wheels, they will also work for any other robot part. If we are designing a robot arm, instead of using diameter robot arm length should be used. Then one can calculate how fast the arm will move with a certain weight being carried.

Robot Motor Factor, Efficiency

The RMF calculated is only for a 100% efficient system. But in reality this never happens. Gearing and friction and many other factors cause inefficiency. There are general rules that would get really close to efficiency. If we have external (not inside the motor) gearing, efficiency should be reduced by ~15%. If we are using treads like on a tank robot, it should be reduced by another ~30%. It robot operates on rough high friction terrain, it should be reduced another ~10%.

For example, a tank robot on rough terrain would have an efficiency of (100% - 30%)*(100% - 10%) = 63% or 0.63.

The RMF equation, incorporating efficiency, is

Torque * rps > = Mass * Acceleration * Velocity * (1/efficiency) / (2 * pi)

Where efficiency is a percentage expressed as a decimal number (i.e. 80% = .8).

Calculations

Robot Specifications

- Mass = 10kg (approx...)
- Velocity = 1.5 m/s (required)
- Acceleration required = 1m/s^2
- Efficiency = 0.5 (Worst case assumption)

300 RPM Side Shaft 37mm Diameter Compact DC Gear

Motor Specifications

- RPM: 300 at 12V
- Voltage: 4V to 12V
- Stall torque: 3.3Kg-cm at stall current of 1.3Amp.
- Shaft diameter: 6mm
- Shaft length: 22mm
- Gear assembly: Spur
- Brush type: Carbon

Fig. 2.1: DC Motor

- Motor weight: 130gms

RMF for Robot = Mass * Acceleration * Velocity /2pi * (1/efficiency)

= 10kg* 1 m/s^2 * 1.5 m/s / 2*3.14156 * (1/0.5)

= 0.796 kgm^2/s^3 * (3)

= 4.776 Watts

RMF of motor = Torque * RPM/60 = 3.3kgcm *300/60 *2pi rad/sec

= 3.3 * 0.098 kgm2/s2 * 5 * 2*3.14156 rad/s

= 10.16 Watts

106mm Diameter x 44mm Thick x 6mm Bore Wheel

Specifications

- Wheel diameter: **106mm**
- Wheel thickness: 44mm
- Hole diameter: 6mm
- Metal bush with M3 bolt for firm grip on the motor's shaft.
- Wheel weight: 123gms

Fig. 2.2: Wheel

Diameter = velocity / (pi * rps)

= 1.5 m/s / (3.1415*300/60)

= 100cm /5*3.1415

= 9.54 cm

= **95.4 mm**

From the above calculations, we can see that the motor is suitable for the bot for the required specifications. Also the wheel diameter is sufficient to provide the desired speed.

Chapter 3

Locomotion for Ground Movement

Introduction to Locomotion

Drive Mechanism

Differential drive is a method of controlling a robot with only two motorized wheels. What makes this algorithm important for a robot builder is that it is also the simplest control method for a robot.

For the locomotion using a differential drive, the wheels have to be independently controlled in such a way as to provide full range of 2-D motion on the ground so that the robot is able to move forward and backward, as well as turn and change direction.

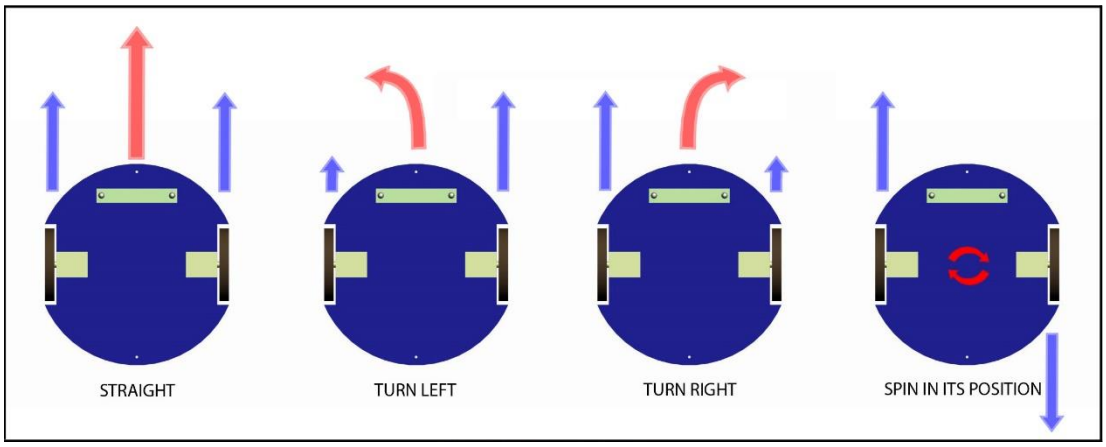


Fig. 3.1: Locomotion using a differential drive

Direction Control

DC motor has two terminals. If we apply positive to one terminal and ground to another motor will rotate in one direction, if we reverse the connection the motor will rotate in opposite direction. If we keep both leads open or both leads ground it will not rotate (but some inertia will be there). We call it Free State. If we apply positive voltage to both leads then braking will occurs. This can be easily done by establishing an H-Bridge as shown in figure.

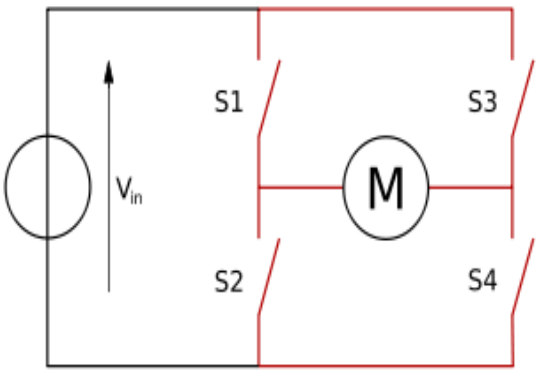


Fig. 3.2: Motor terminal connection circuit for direction control

When S1 and S4 are closed, the motor rotates in one direction. When S2 and S3 are closed, the motor moves in the opposite direction. When both terminals across motors are high, the motor is stalled.

Terminal 1	**Terminal 2**	**State**
0	1	Clockwise
1	0	Anti-clockwise
1	1	Breaking Condition
0	0	Free run

Table 3.1: State of motor according to connection

This can be done by using a simple a DPDT. However, breaking condition isn't achievable here. L293D is an IC containing 2 H-Bridges. This IC can be used to control motion of motor via microcontroller. The Hercules motor driver uses the same principle.

Speed Variation

There are mainly two ways of controlling speed of DC motor:

1. Varying the supply voltage
2. Pulse width modulation (PWM)

First method is by varying the supply voltage, i.e. voltage across the motor. But this method affects the power output by lowering it. And also, this is a somewhat a tough task because we mostly microcontrollers to vary the supply voltage and to generate variable supply voltage we should go for analog circuit, then come analog to digital conversion and other steps which makes the task tedious. Hence it is not a feasible option.

The second method is to do it via PWM. The function of the H-bridge can be enhanced by using PWM to control the speed of the motor. The PWM signal is illustrated in Figure shown on next page.

When the PWM signal is high, the motor is on; when low, the motor is off. A duty cycle is given by percentage ratio of T-on and total time. The greater the on time, greater the duty cycle, the higher the average voltage and greater is the speed. For example, if you apply a PWM of 50% duty cycle to a 100 RPM motor, it would run at 50 RPM (50% of 100 RPM).

PWM can be generated using IC555 or using microcontroller or computer parallel port. We use the PWM pins on the Arduino for this purpose.

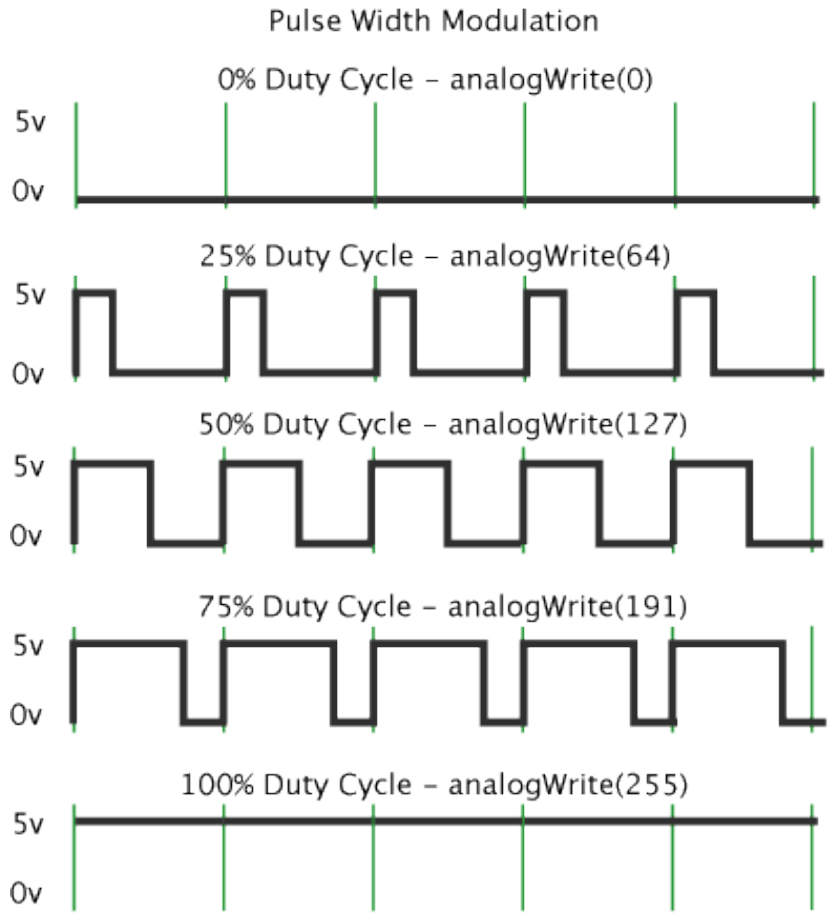


Fig. 3.3: Pulse width modulation clock diagram

DC Motor

A DC motor relies on the fact that like magnet poles repels and unlike magnetic poles attracts each other. A coil of wire with a current running through it generates an electromagnetic field aligned with the center of the coil. By switching the current on or off in a coil its magnet field can be switched on or off or by switching the direction of the current in the coil the direction of the generated magnetic field can be switched 180°.

A simple *DC motor* typically has a stationary set of magnets in the stator and an armature with a series of two or more windings of wire wrapped in insulated stack slots around iron pole pieces (called stack teeth) with the ends of the wires terminating on a commutator. The armature includes the mounting bearings that keep it in the center of the motor and the power shaft of the motor and the commutator connections. The winding in the armature continues to loop all the way around the armature and uses either single or parallel conductors (wires), and can circle several times around the stack teeth. The total amount of current sent to the coil, the coil's size and what it's wrapped around dictate the strength of the electromagnetic field created. The sequence of turning a particular coil on or off dictates what direction the effective electromagnetic fields are pointed. By turning on and off coils in sequence a rotating magnetic field can be created. These rotating magnetic fields interact with the magnetic fields of the magnets (permanent or electromagnets) in the stationary part of the

motor (stator) to create a force on the armature which causes it to rotate. In some DC motor designs the stator fields use electromagnets to create their magnetic fields which allow greater control over the motor. At high power levels, DC motors are almost always cooled using forced air.

Since the series-wound DC motor develops its highest torque at low speed, it is often used in traction applications such as electric locomotives, and trams. The DC motor was the mainstay of electric traction drives on both electric and diesel-electric locomotives, street-cars/trams and diesel electric drilling rigs for many years. The introduction of DC motors and an electrical grid system to run machinery starting in the 1870s started a new second Industrial Revolution. DC motors can operate directly from rechargeable batteries, providing the motive power for the first electric vehicles and today's hybrid cars and electric cars as well as driving a host of cordless tools. Today DC motors are still found in applications as small as toys and disk drives, or in large sizes to operate steel rolling mills and paper machines.

Connections of DC motor:

There are three types of electrical connections between the stator and rotor possible for DC electric motors: series, shunt/parallel and compound (various blends of series and shunt/parallel) and each has unique speed/torque characteristics appropriate for diffent loading torque profiles/signatures.

Series connection

A series DC motor connects the armature and field windings in series with a common D.C. power source. The motor speed varies as a non-linear function of load torque and armature current; current is common to both the stator and rotor yielding current squared (I^2) behaviour. A series motor has very high starting torque and is commonly used for starting high inertia loads, such as trains, elevators or hoists.[2] This speed/torque characteristic is useful in applications such asdragline excavators, where the digging tool moves rapidly when unloaded but slowly when carrying a heavy load.

With no mechanical load on the series motor, the current is low, the counter-EMF produced by the field winding is weak, and so the armature must turn faster to produce sufficient counter-EMF to balance the supply voltage. The motor can be damaged by over speed. This is called a runaway condition.

Shunt connection

A shunt DC motor connects the armature and field windings in parallel or shunt with a common D.C. power source. This type of motor has good speed regulation even as the load varies, but does not have the starting torque of a series DC motor.[3] It is typically used for industrial, adjustable speed applications, such as machine tools, winding/unwinding machines and tensioners.

Compound connection

A compound DC motor connects the armature and fields windings in a shunt and a series combination to give it characteristics of both a shunt and a series DC motor.[4] This motor is used when both a high starting torque and good speed regulation is needed. The motor can be connected in two arrangements: cumulatively or differentially. Cumulative compound motors connect the series field to aid the shunt field, which provides higher starting torque but less speed regulation. Differential compound DC motors have good speed regulation and are typically operated at constant speed.

Motor Ratings

Voltage

DC motors are non-polarized - meaning that one can reverse voltage without any bad things happening. Typical DC motors are rated from about 6V-12V. The larger ones are often 24V or more. But for the purposes of a robot, you probably will stay in the 6V-12V range. So why do motors operate at different voltages? As we all know (or should), voltage is directly related to motor torque. More voltage, higher the torque. But we should not go running motor at 100V cause that's just not nice. A DC motor is rated at the voltage it is most efficient at running. If we apply too few volts, it just won't work. If we apply too much, it will overheat and the coils will melt. So the general rule is, one should try to apply as close to the rated voltage of the motor. Also, although a 24V motor might be stronger, do one really want robot to carry a 24V battery (which is heavier and bigger) around? Recommendation is not surpass 12V motors unless we really need the torque.

Current

As with all circuitry, one must pay attention to current. There are two rated current which should be paid attention. The first is **operating current**. This is the average amount of current the motor is expected to draw under a typical torque. After Multiplying this number by the rated voltage and the average power draw required to run the motor can be calculated. The other current rating is the **stall current**. This is when one power up the motor, and put enough torque on it to force it to stop rotating. This is the maximum amount of current the motor will ever draw, and hence the maximum amount of power too. So we must design all control circuitry capable of handling this stall current. Also, if one plan to constantly run

motor, or run it higher than the rated voltage, it is wise to **heat sink** motor to keep the coils from melting.

Power Rating

How high of a voltage can one over apply to a motor? Well, all motors are (or at least should be) rated at a certain wattage. Wattage is energy. Inefficiency of energy conversion directly relates to heat output. Too much heat, the motor coils melt. So the manufacturers of [higher quality] motors know how much wattage will cause motor failure.

The equation is:

Power (watts) = Voltage * Current

Power Spikes

There is a special case for DC motors that change directions. To reverse the direction of the motor, one must also reverse the voltage. However the motor has a built up **inductance** and **momentum** which resists this voltage change. So for the short period of time it takes for the motor to reverse direction, there is a large power spike. The voltage will spike double the operating voltage. The current will go to around stall current.

Torque

There are two **torque** value ratings which one must pay attention to. The first is **operating torque**. This is the torque the motor was designed to give. Usually it is the listed torque value. The other rated value is **stall torque**. This is the torque required to stop the motor from rotating. If we are designing a wheeled robot, good torque means good acceleration. The rule should be if we have 2 motors on robot, it should be made sure the stall torque on each is enough to lift the weight of entire robot times wheel radius. Always we should favour torque over velocity. Torque ratings can change depending on the voltage applied. So if one need a little more torque, going 20% above the rated motor voltage value is fairly safe. We should remember that this is less efficient, and that one should heat sink motor.

Velocity

Velocity is very complex when it comes to DC motors. The general rule is, motors run the most efficient when run at the highest possible speeds. Obviously however this is not possible. There are times we want our robot to run slowly. So first one want **gearing** - this way the motor can run fast, yet one can still get good torque out of it. Unfortunately gearing automatically reduces efficiency no higher than about 90%. So we should include a 90% speed and torque reduction for every gear meshing when gearing is calculated. For example, if we have 3 spur gears, therefore meshing together twice, we will get a 90% x 90% = 81% efficiency. The voltage and applied torque resistance obviously also affects speed.

Basic DC motor equations

Let

- E_b = induced or counter emf (V)
- I_a = armature current (A)
- k_b = counter emf equation constant
- k_n = speed equation constant
- k_T = torque equation constant
- n = armature speed (rpm)
- R_m = motor resistance (Ω)
- T = motor torque (Nm)
- V_m = motor input voltage (V)
- $Ø$ = machine's total flux (Wb)

Counter emf equation

The DC motor's counter emf is proportional to the product of the machine's total flux strength and armature speed:

$$E_b = k_b * Ø * n$$

Voltage balance equation

The DC motor's input voltage must overcome the counter emf as well as the voltage drop created by the armature current across the motor resistance, that is, the combined resistance across the brushes, armature winding and series field winding, if any:

$$V_m = E_b + R_m * I_a$$

Torque equation

The DC motor's torque is proportional to the product of the armature current and the machine's total flux strength:

$$T = k_b * I_a * Ø / (2\pi)$$
$$T = k_T * I_a * Ø$$

Where

$$k_T = k_b / (2\pi)$$

Speed equation

Since

$$n = E_b / (k_b * Ø) \text{ and } V_m = E_b + R_m * I_a$$

we have

$$n = (V_m - R_m * I_a) / (k_b * Ø)$$

$$n = k_n * (V_m - R_m * I_a) / Ø$$

Where

$$k_n = 1 / k_b$$

Motor Driver:

A motor driver circuit is designed to drive an electromagnetic load, such as a brushed or brushless motor, stepper motor or a solenoid or relay.

Motors typically require voltages and/or currents that exceed what can be provided by the analog or digital signal processing circuitry that controls them. The motor driver provides the interface between the signal processing circuitry and the motor itself. It is essentially the "amplifier" for the motor.

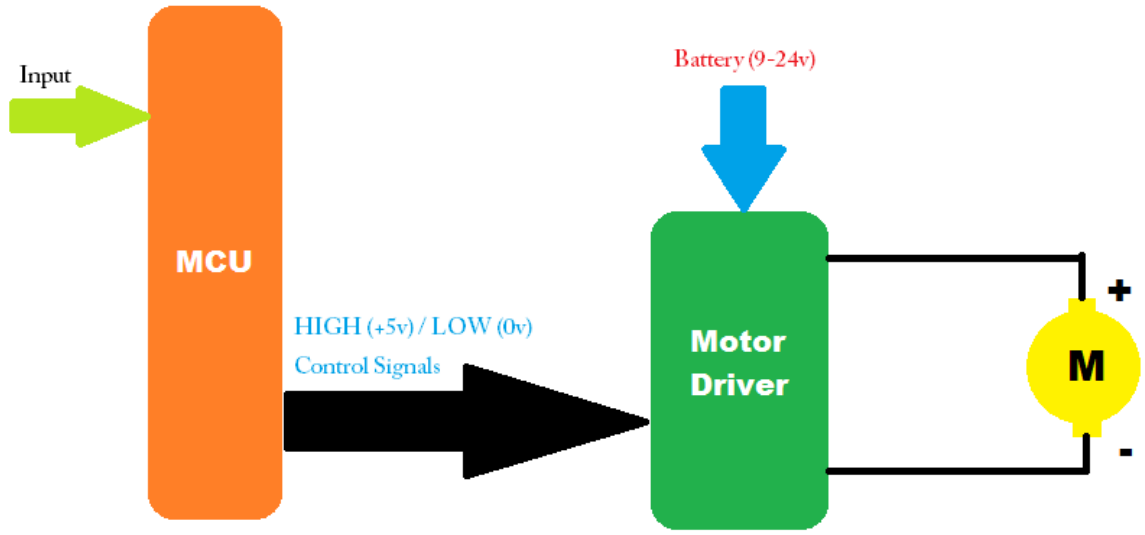


Fig. 3.4: Block diagram of Motor Driver

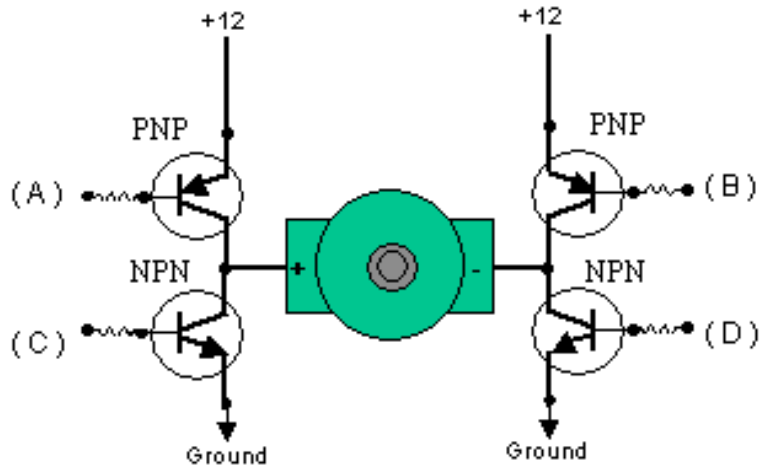
Motor drivers can be constructed from discrete components, completely integrated inside an IC, or may employ both discrete and integrated components. When current and voltage levels allow, integration of the entire motor driver inside a single IC generally provides the highest level of functionality and performance at the smallest physical size.

In addition to providing high-voltage and high-current drive, motor drivers also often integrate control circuitry, such as current regulation or digital state machines to operate the motor.

Motor Driver ICs are primarily used in autonomous robotics only. Also most microprocessors operate at low voltages and require a small amount of current to operate while the motors require a relatively higher voltages and current. Thus current cannot be supplied to the motors from the microprocessor. This is the primary need for the motor driver IC.

H-Bridge

The original concept of the H-Bridge was being able to control the direction a motor was going. Forward or backward. This was achieved by managing current flow through circuit elements called transistors. The formation looks like an H and that's where it gets the name H-Bridge. Here is what it looks like:



A	B	C	D	Function
-	-	-	-	----------
1	0	0	1	Forward
0	1	1	0	Reverse
1	1	0	0	Brake
0	0	1	1	Brake

Fig. 3.5: H-Bridge Circuit

The picture above illustrates the 4 base cases that we can get out of the simple version of an H-Bridge. The two cases that interest us are when A & D are both 1 and when B & C are both 1. When A & D are 1 current from the battery will flow from point A through the motor to D's ground. However for the case when B & C are both 1, current will flow in the opposite direction from B through the motor to C's ground.

The L298HN Motor Driver

The advantage that the HN offers is that all the extra diodes typically necessary with a standard L298 circuit are already internally in the chip. It saves us as designers an extra element for the motor control circuit.

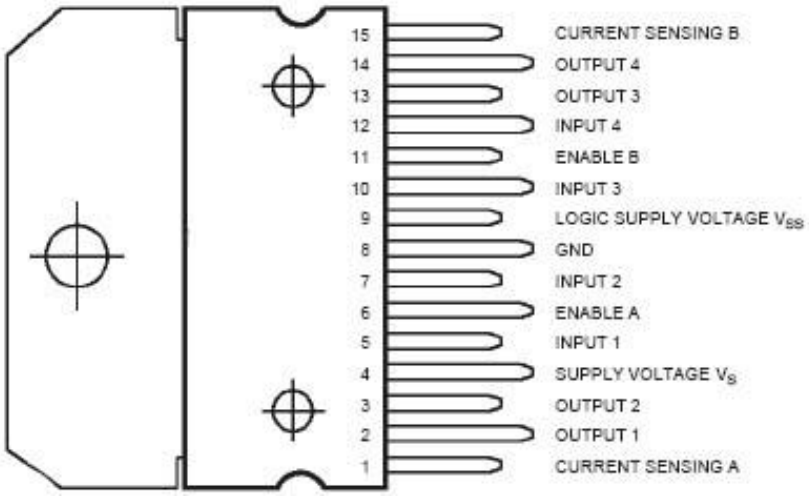


Fig. 3.6: Pin Diagram of L298HN Motor Driver

Varying DC Motor Speed

Pins 5 & 7 in the chip pinout above are inputs 1 & 2 respectively. These inputs take what is called a PWM input. The frequency of the PWM is dependent upon the motor. For our motor we'll use a 1 KHz input frequency. This means the motor speed will be updates 1 thousand times a second. The duty cycle of the PWM will determine the speed & direction of the motor.

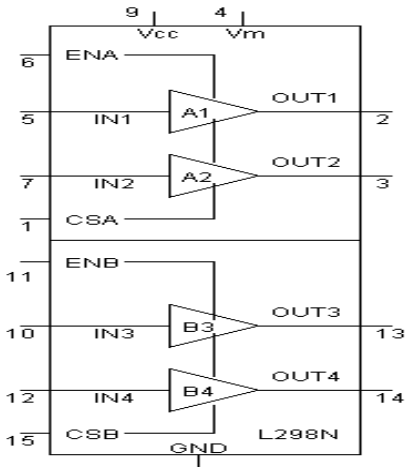


Fig. 3.7: Pin Connection for varying DC Motor Speed

Schematic Diagram

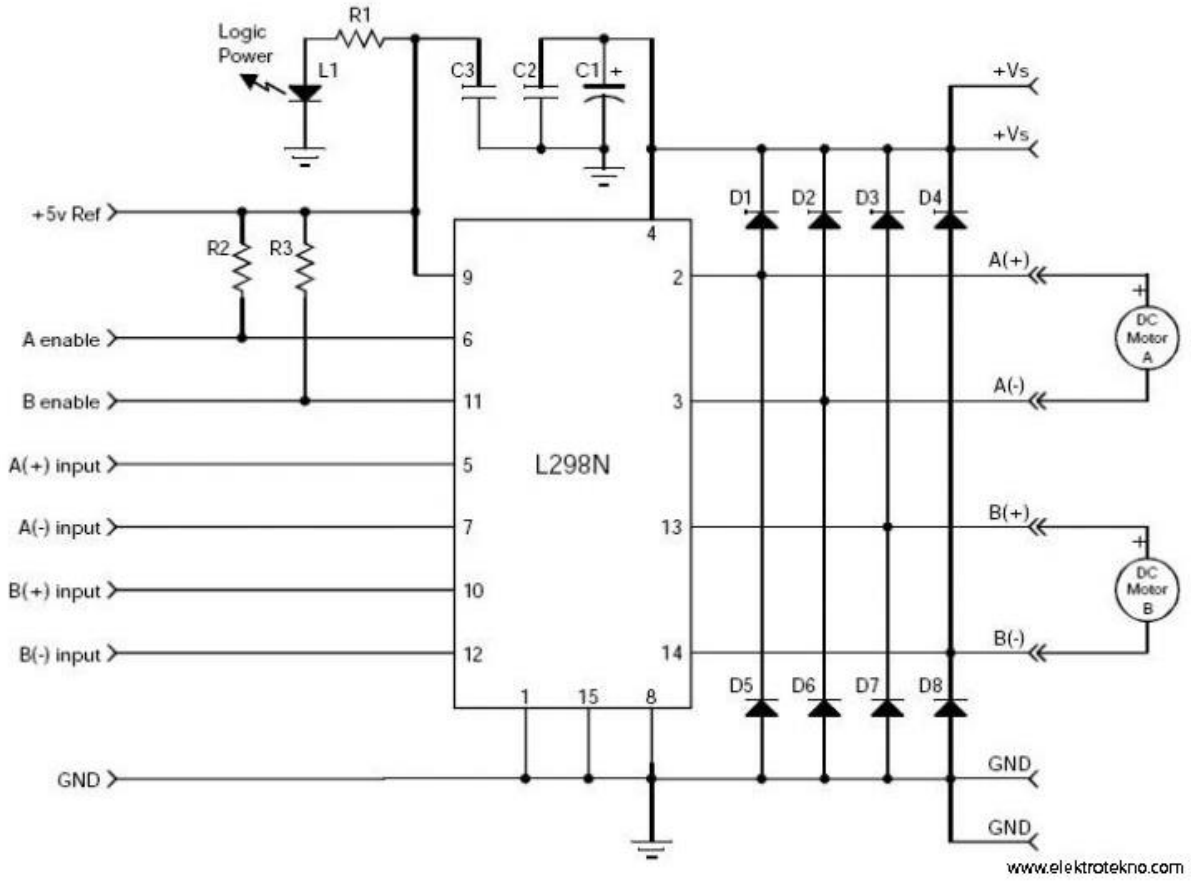


Fig. 3.8: Schematic Diagram of Motor Driver

Interfacing with Arduino

The Arduino controls the DC Motors via the Motor Driver. The Arduino is pre-loaded with functions for controlling the speed as well as the direction of the motors. The Motor Speed is controlled via PWM pins on the Arduino which employ Pulse Width modulation to supply a specific percentage of the Battery Voltage Input to the Motor Driver. This is ensured by the Enable pins. There are two enable pins provided on the motor driver for 2 different DC Motors. The other two pins are Digital Input pins for each motor. These can be either made HIGH or LOW, which will in turn control the direction of the motor accordingly.

The Arduino or the Microcontroller effectively acts as a control mechanism whereas the motor driver acts as a driving mechanism for the DC motors. As we can clearly see from the above block diagram, the Motor Driver channelizes the power input from the battery (9-24V) to the DC motors. The Microcontroller tells the motor driver the polarity in which this power input should be supplied to the motor. In addition the microcontroller also controls the Motor speed by controlling the PWM applied to the power input.

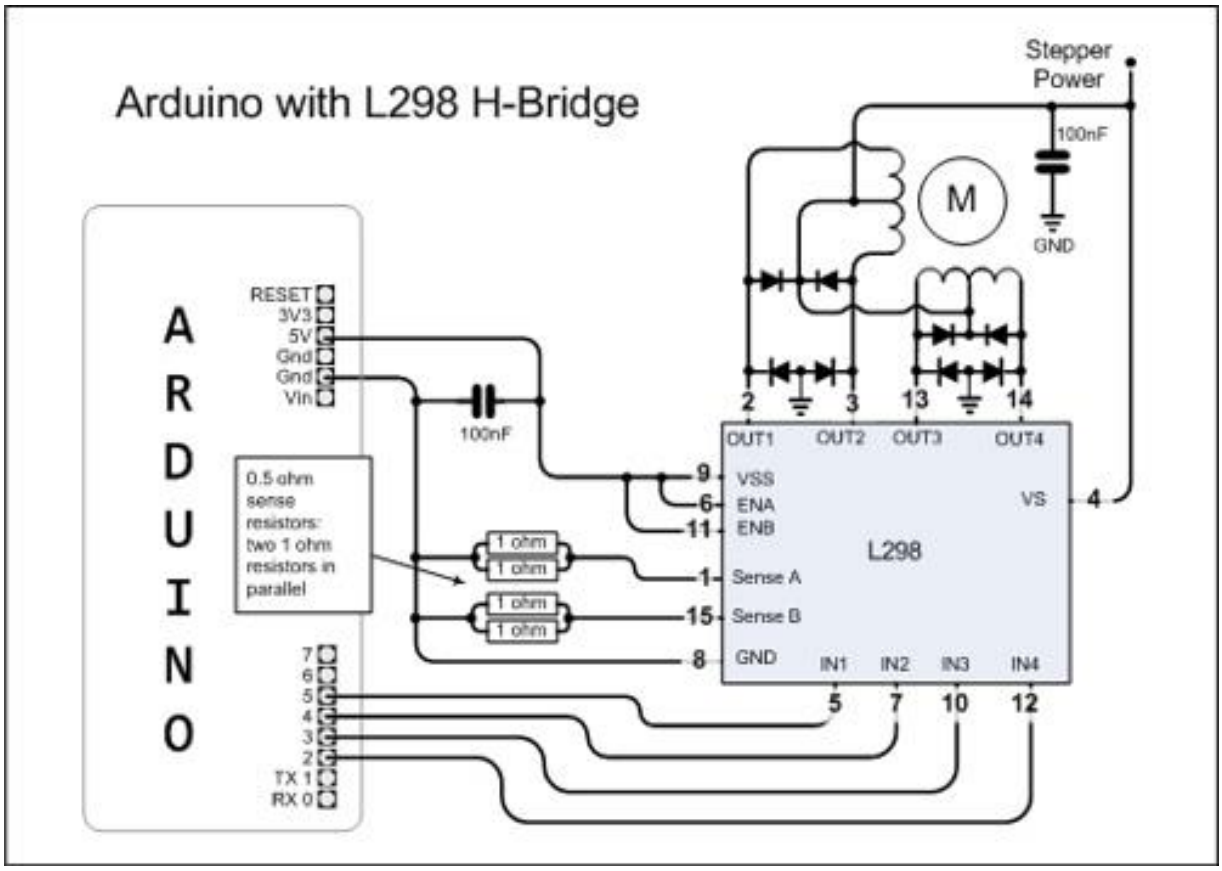


Fig. 3.9: Interfacing Circuit of Motor Driver with Arduino

For example, if the Supply from the battery terminals is 12V, and the PWM from the microcontroller is set as 120; then the output from the motor driver to the motor terminals and effectively the terminal voltage at the motor will be $\frac{120}{255} \times 12\ V = 5.647\ V$.

Programming Functions for Locomotion

There are five basic functions for locomotion. These are listed below.

1. Move_Forward
2. Move_Backward
3. Turn_left
4. Turn_right
5. Stop

1. MOVE_FORWARD

DESCRIPTION: Makes the robot move in the forward direction

INPUT PARAMETERS: Left Motor PWM, RIGHT Motor PWM

OUTPUT: Left_INA, Left_INB, Left_PWM, Right_INA, Right _INB, Right _PWM

EXPLANATION:

A. The Digital Pin INA for left motor is made HIGH.
B. The Digital Pin INB for left motor is made LOW.
C. This ensures that the direction of left motor is forward.
D. The Digital Pin INA for right motor is made HIGH.
E. The Digital Pin INB for right motor is made LOW.
F. This ensures that the direction of right motor is forward.
G. The PWM pin for left motor is set equal to the input parameter. The range for this parameter is 0-255.
H. The PWM pin for right motor is set equal to the input parameter. The range for this parameter is also 0-255. Typically it will be the same as that for the left motor.
I. Ideally, both motors are identical, but they might differ marginally in terms of speed at a given voltage. So sometimes, two different PWM speeds may be required.

SAMPLE CODE

```
void move_forward(int lpwm,int rpwm)
{
  digitalWrite(lINA,HIGH);
  digitalWrite(lINB,LOW);
  analogWrite(lPWM,lpwm);
  digitalWrite(rINA,LOW);
  digitalWrite(rINB,HIGH);
  analogWrite(rPWM,rpwm);
}
```

2. MOVE_BACKWARD

DESCRIPTION: Makes the robot move in the forward direction

INPUT PARAMETERS: Left Motor PWM, RIGHT Motor PWM

OUTPUT: Left_INA, Left_INB, Left_PWM, Right_INA, Right _INB, Right _PWM

EXPLANATION:

A. The Digital Pin INA for left motor is made HIGH.
B. The Digital Pin INB for left motor is made LOW.
C. This ensures that the direction of left motor is forward.
D. The Digital Pin INA for right motor is made HIGH.
E. The Digital Pin INB for right motor is made LOW.
F. This ensures that the direction of right motor is forward.
G. The PWM pin for left motor is set equal to the input parameter. The range for this parameter is 0-255.
H. The PWM pin for right motor is set equal to the input parameter. The range for this parameter is also 0-255. Typically it will be the same as that for the left motor.
I. Ideally, both motors are identical, but they might differ marginally in terms of speed at a given voltage. So sometimes, two different PWM speeds may be required.

SAMPLE CODE

```
void move_backward(int rpwm,int lpwm)
{
  digitalWrite(lINA,LOW);
  digitalWrite(lINB,HIGH);
  analogWrite(lPWM,lpwm);
  digitalWrite(rINA,LOW);
  digitalWrite(rINB,HIGH);
  analogWrite(rPWM,rpwm);
}
```

3. TURN_LEFT

DESCRIPTION: Makes the robot move in the forward direction

INPUT PARAMETERS: Left Motor PWM, RIGHT Motor PWM

OUTPUT: Left_INA, Left_INB, Left_PWM, Right_INA, Right _INB, Right _PWM

EXPLANATION:

A. The Digital Pin INA for left motor is made HIGH.
B. The Digital Pin INB for left motor is made LOW.
C. This ensures that the direction of left motor is forward.
D. The Digital Pin INA for right motor is made HIGH.
E. The Digital Pin INB for right motor is made LOW.
F. This ensures that the direction of right motor is forward.
G. The PWM pin for left motor is set equal to the input parameter. The range for this parameter is 0-255.
H. The PWM pin for right motor is set equal to the input parameter. The range for this parameter is also 0-255. Typically it will be the same as that for the left motor.
I. Ideally, both motors are identical, but they might differ marginally in terms of speed at a given voltage. So sometimes, two different PWM speeds may be required.

SAMPLE CODE

```
void turn_left(int rpwm)
{
  digitalWrite(lINA,HIGH);
  digitalWrite(lINB,HIGH);
  digitalWrite(rINA,HIGH);
  digitalWrite(rINB,LOW);
  analogWrite(rPWM,rpwm);
}
```

4. TURN_RIGHT

DESCRIPTION: Makes the robot move in the forward direction

INPUT PARAMETERS: Left Motor PWM, RIGHT Motor PWM

OUTPUT: Left_INA, Left_INB, Left_PWM, Right_INA, Right _INB, Right _PWM

EXPLANATION:

A. The Digital Pin INA for left motor is made HIGH.
B. The Digital Pin INB for left motor is made LOW.
C. This ensures that the direction of left motor is forward.
D. The Digital Pin INA for right motor is made HIGH.
E. The Digital Pin INB for right motor is made LOW.
F. This ensures that the direction of right motor is forward.
G. The PWM pin for left motor is set equal to the input parameter. The range for this parameter is 0-255.
H. The PWM pin for right motor is set equal to the input parameter. The range for this parameter is also 0-255. Typically it will be the same as that for the left motor.
I. Ideally, both motors are identical, but they might differ marginally in terms of speed at a given voltage. So sometimes, two different PWM speeds may be required.

SAMPLE CODE

```
void turn_right(int lpwm)
{
  digitalWrite(rINA,HIGH);
  digitalWrite(rINB,HIGH);
  digitalWrite(lINA,HIGH);
  digitalWrite(lINB,LOW);
  analogWrite(lPWM,lpwm);
}
```

5. **STOP**

DESCRIPTION: Makes the robot move in the forward direction

INPUT PARAMETERS: Left Motor PWM, RIGHT Motor PWM

OUTPUT: Left_INA, Left_INB, Left_PWM, Right_INA, Right _INB, Right _PWM

EXPLANATION:

A. The Digital Pin INA for left motor is made HIGH.
B. The Digital Pin INB for left motor is made LOW.
C. This ensures that the direction of left motor is forward.
D. The Digital Pin INA for right motor is made HIGH.
E. The Digital Pin INB for right motor is made LOW.
F. This ensures that the direction of right motor is forward.
G. The PWM pin for left motor is set equal to the input parameter. The range for this parameter is 0-255.
H. The PWM pin for right motor is set equal to the input parameter. The range for this parameter is also 0-255. Typically it will be the same as that for the left motor.
I. Ideally, both motors are identical, but they might differ marginally in terms of speed at a given voltage. So sometimes, two different PWM speeds may be required.

SAMPLE CODE

```
void stop()
{
  digitalWrite(lINA,HIGH);
  digitalWrite(lINB,HIGH);
  digitalWrite(rINA,HIGH);
  digitalWrite(rINB,HIGH);
}
```

Chapter 4

Gestures using Servo Motors

Gestures of humanoid

The robot uses servos to perform various gestures. These gestures can be divided into four main categories, though there is tremendous scope of increasing the gestures. The gestures were tried to be kept simple and human-like. There are in all 4 servo motors, of which two are located at the shoulder joint and two at the neck for 2-dimensional movement of the neck.

1. Hand motion while walking
2. Handshake movement
3. Neck Rotation (Horizontal)
4. Neck Flexion (Vertical)

Before we delve deeper into the mechanics of these motions, a good understanding of servo motors is deemed necessary.

Servo Motor

A **servo motor** is a rotary actuator that allows for precise control of angular position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.

Servomotors are not a specific class of motor although the term *servomotor* is often used to refer to a motor suitable for use in a closed-loop control system. Servos are DC motors with built in gearing and feedback control loop circuitry. And no motor drivers required!

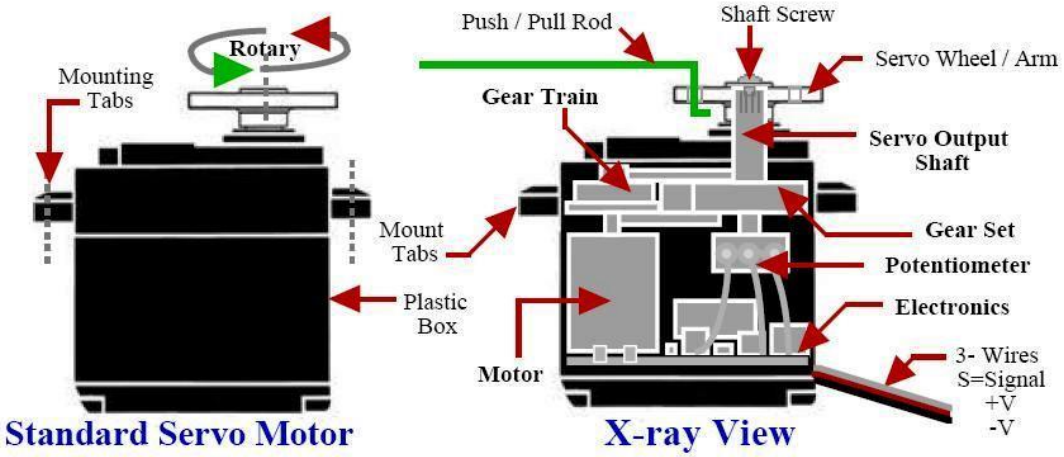


Fig. 4.1: Standard Servo motor and its X-ray View

Servos are extremely popular with robot, RC plane, and RC boat builders. Most servo motors can rotate about 90 to 180 degrees. Some rotate through a full 360 degrees or more. However, servos are unable to continually rotate, meaning they can't be used for driving wheels

(unless modified), but their precision positioning makes them ideal for robot arms and legs, rack and pinion steering, and sensor scanners to name a few. Since servos are fully self-contained, the velocity and angle control loops are very easy to implement, while prices remain very affordable.

Servo Voltage

Servos can operate under a range of voltages. Typical operation is from 4.8V to 6V. There are a few micro sized servos that can operate at less, and now a few Hitec servos that operate at much more. The reason for this standard range is because most microcontrollers and RC receivers operate near this voltage. So what voltage should we operate at? Well, unless we have a battery/voltage/current/power limitation, we should operate at 6V. This is simply because DC motors have higher torque at higher voltages.

Regulating Voltage to a Servo

It is already known that servos have a voltage rating. After going above that voltage and servo overheats and possibly fries. So suppose we have a 7.2V battery and we want to use a 5V regulator to power your servos, it will work, but it's a huge waste of battery power.

If we have 7.2V regulated to 5V and the servos draw a total of 1.5A of current.

Wasted power is:
(7.2V-5V)*1.5A = 3.3W

Percentage wise, its
(7.2V-5V)/7.2V = 30.6%

That's the battery energy percentage wasted to thermal heat - almost 1/3rd which is a significant percentage.

Speaking of heat, voltage regulator probably has thermal shutdown, meaning that if it overheats it will throttle down current to servos - meaning servos will have lower torque and lower speed. If voltage regulator doesn't have thermal shutdown, it will just fry instead (not a good thing). But if there is really a need to regulate for servos, one can use a switching regulator (like ~83% efficiency on average).

Efficiency and Noise

Due to noise and control circuitry requirements, servos are less efficient than DC motors uncontrolled. To begin with, the control circuitry typically drains 5-8mA just on idle. Secondly, noise can more than triple current draw during a holding position (not moving), and almost double current during rotation.

Feedback system of Servo Motor:

The block diagram below shows the interaction between Microcontroller which provides the position command to the servo, and the Servo control circuit, which receives in addition to this, the position feedback from the position pot which gives feedback about the angle turned by the motor. Accordingly the drive current is supplied to the motor after varying the pulse width/duration appropriately.

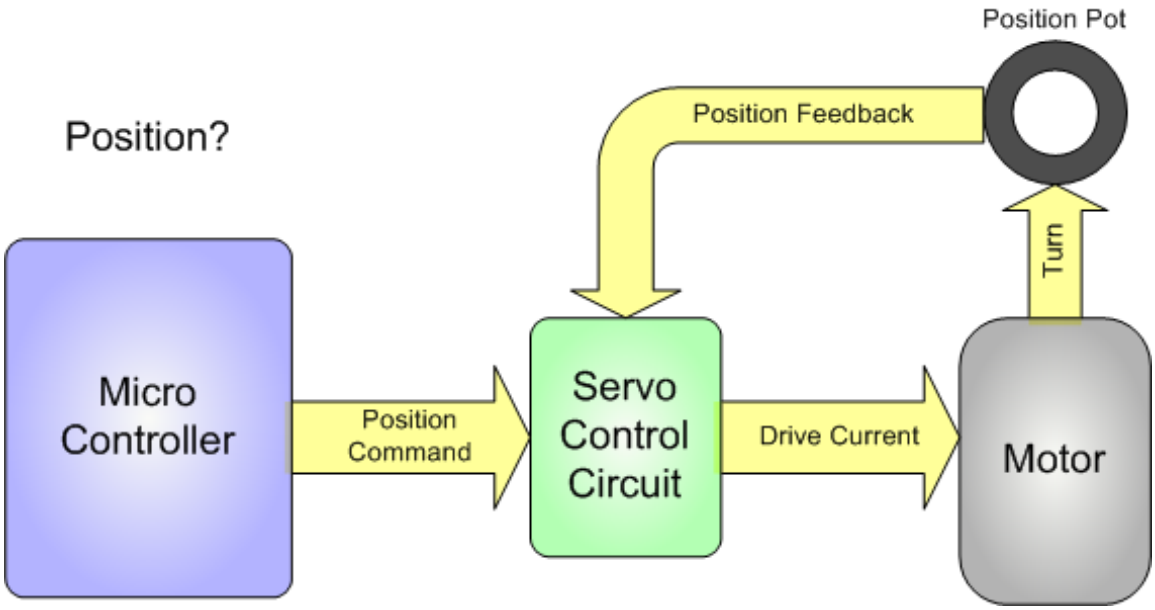


Fig. 4.2: Feedback System of Servo Motor

Gear Types

More expensive servos come with metal gears for higher torque and longer life, followed by Karbonite and then nylon gears for the cheapest.

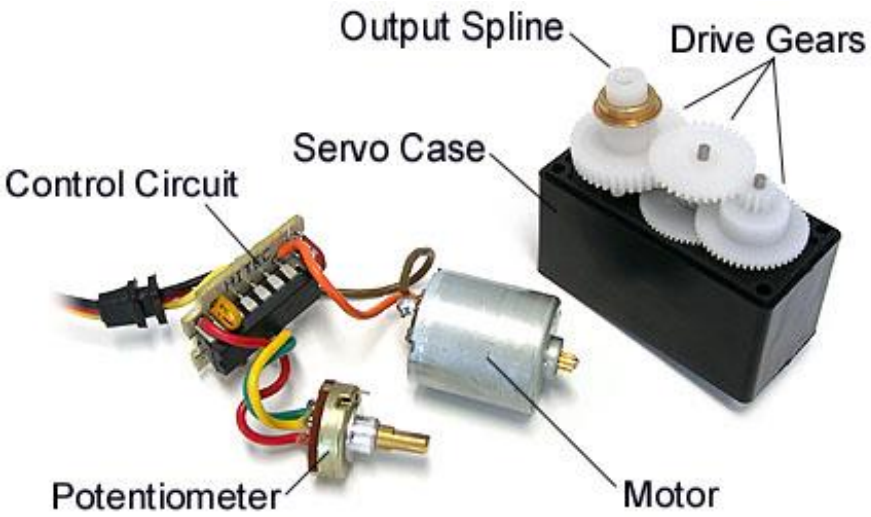


Fig. 4.3: Components of Servo Motor

Nylon Gears - Nylon gears are most common in servos. They are extremely smooth with little or no wear factors. They are also very lightweight, but lack in durability and strength.

Karbonite Gears - Karbonite gears are relatively new to the market. They offer almost 5 times the strength of nylon gears and also better wear resistance. Cycle times of well over 300,000 have been observed with these gears with virtually no wear. Servos with these gears are more expensive but what we get in durability is more than equalled.

Metal Gears - Metal gears have been around for some time now. Although the heaviest and having the highest wear rate of all gear types, they offer unparalleled strength. With a metal output shaft, side-loads can be much greater. In applications that are jarred around, metal gears are best. Unfortunately, due to wear, metal gears will eventually develop slight play in the gear-train. Accuracy will slowly be lost.

Velocity

The servo turn rate, or **transit time**, is used for determining servo rotational velocity. This is the amount of time it takes for the servo to move a set amount, usually 60 degrees. For example, suppose we have a servo with a transit time of 0.17sec/60 degrees at no load. This means it would take nearly half a second to rotate an entire 180 degrees. More if the servo were under a load. This information is very important if high servo response speed is a requirement of robot application. It is also useful for determining the maximum forward velocity of robot if servo is modified for full rotation. The worst case turning time is when the servo is at the minimum rotation angle and is then commanded to go to maximum rotation angle, all while under load. This can take several seconds on a very high torque servo.

Interfacing with Arduino:

To interface a servo with a microcontroller like Arduino, we require to output three pins to the three terminals of the Servo.

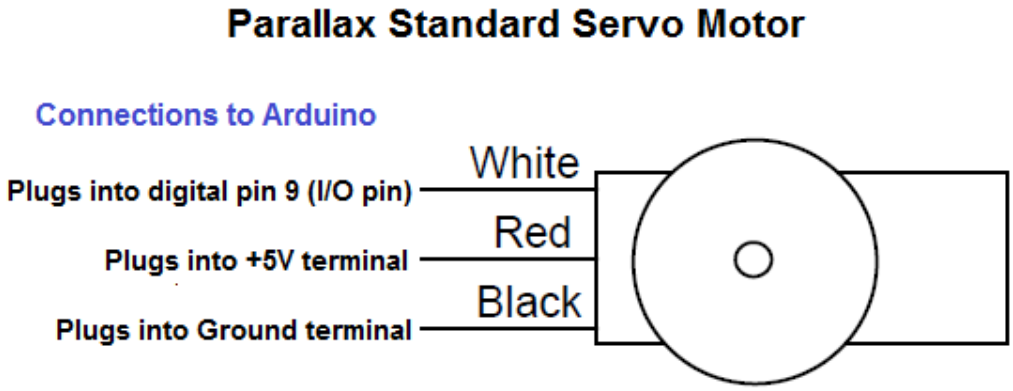


Fig. 4.4: Connection of Servo to Arduino

All servos have three wires. These are listed below

- Black or Brown is for *ground*.
- Red is for *power* (~4.8-6V).
- Yellow, Orange, or White is the *signal* wire (3-5V).

To use a servo, simply we have to connect the black wire to ground, the red to a 4.8-6V source, and the yellow/white wire to a signal generator (such as from your microcontroller). Then Vary the square wave pulse width from 1-2ms and now servo is now position/velocity controlled.

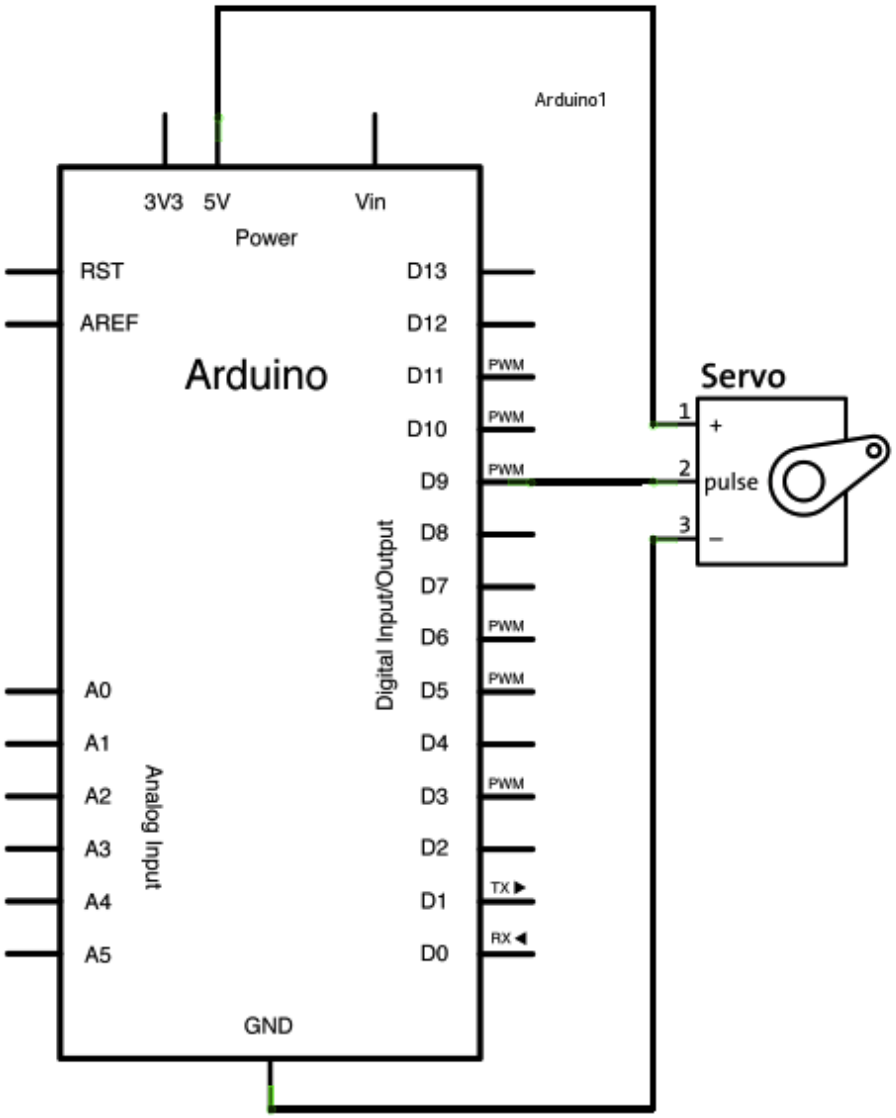


Fig. 4.5: Interfacing of Servo with Arduino

The Arduino supplies the signal to the servo which is implemented by the Signal wire.

Signal Wire

While the black and red wires provide power to the motor, the signal wire is what we use to command the servo. The general concept is to simply send an ordinary logic square wave to servo at a specific wave length, and servo goes to a particular angle (or velocity if servo is modified). *The wavelength directly maps to servo angle*.

If we are running multiple servos simultaneously, we can just put a few of these program blocks in sequential order. We can run as many servos as we have of digital ports.

So the time for which we keep the port high depends on the servo. We may have to tweak for each individual servo some several micro seconds' difference.

Pulse Width control

For Zero rotation the width or duration of the pulse is 1 milliseconds which is minimum. On the contrary, for full rotation (180 degrees), the width or duration of the pulse is 2 milliseconds which is maximum. The value between the above two periods is used to put the servo into a neutral position (90 degrees).

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

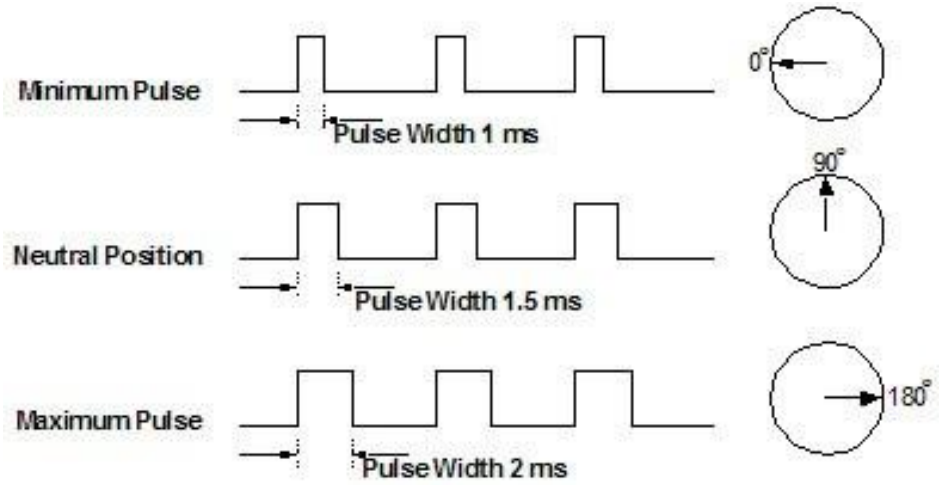


Fig. 4.6: Pulse width control timing diagram

For clockwise rotation the width or duration of the pulse is between 0.7 and 1 milliseconds. On the contrary, for clockwise rotation the width or duration of the pulse is between 1.7 and 2 milliseconds. The value between the upper limits of the above two periods is used to centre the servo rotation.

Functions for Servo

The four gestures discussed earlier are accomplished using different combination of servos. The gestures are listed again for reference.

1. Hand motion while walking

2. Handshake movement

3. Neck Rotation (Horizontal)

4. Neck Flexion (Vertical)

1. Hand motion while walking

DESCRIPTION: Makes the hands of the robot move in opposite fashion

INPUT PARAMETERS: Left Servo, Right Servo

OUTPUT: Left Servo Position, RIGHT Servo Position

EXPLANATION:

A. First the Left and Right Servo are set to initial positions.
B. Then the Left servo is moved forward while the right servo is moved in opposite direction.
C. The movement is reversed after reaching an angle of rotation equal to 90 degrees.
D. The to and fro motion of hands continues to simulate the motion of hands of a human while walking.

SAMPLE CODE:

```
void hands()
{
 for(pos = 0; pos < 90; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                // in steps of 1 degree
    servo_left.write(pos);
    servo_right.write(pos+60);    // tell servo to go to position in variable 'pos'
    delay(8);                      // waits 8ms for the servo to reach the position
  }
  for(pos = 90; pos>=0; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    servo_left.write(pos);
    servo_right.write(pos+60);              // tell servo to go to position in variable 'pos'
    delay(8);                    // waits 8ms for the servo to reach the position
  }
}
```

2. Handshake movement

DESCRIPTION: Initiates and Completes a handshake with right hand

INPUT PARAMETERS: Left Servo, Right Servo

OUTPUT: Left Servo Position, RIGHT Servo Position

EXPLANATION:

A. First the Left Servo is brought to rest in a vertical position.
B. Then the Right Servo is brought to a horizontal elevated position such as to simulate the raising of right hand.
C. The right servo is moved to and fro in small period of oscillation for three repetitions to simulate shaking of hands.
D. The right servo is brought back to rest at the initial vertical position to indicate completion of handshake.

SAMPLE CODE:

```
void handshake()
{
 pos=0;
 int d;
 // Raise Hands
  for(pos=100;pos>=0;pos--)
  { d=(100-pos)/2;
    servo_left.write(d);
    servo_right.write(pos);
    delay(5);
  }

    //Shake hands
  int x=0;
  while(x<3)
  {for(pos = 0; pos < 40; pos += 1)  // goes from 0 degrees to 40 degrees
    {                               // in steps of 1 degree
    servo_right.write(pos);    // tell servo to go to position in variable 'pos'
    delay(10);    // waits 15ms for the servo to reach the position
    }
    for(pos = 40; pos>=0; pos-=1)     // goes from 40 degrees to 0 degrees
    {
      servo_right.write(pos);              // tell servo to go to position in variable 'pos'
    delay(10);                       // waits 15ms for the servo to reach the position
    }
   x++;

  }

  delay(1000);
```

3. Neck Rotation

DESCRIPTION: Makes the neck of the robot rotate in horizontal plane

INPUT PARAMETERS: Horizontal Servo Position, X-coordinate of Face on Webcam output

OUTPUT: Horizontal Servo Position

EXPLANATION:

A. First the Horizontal Servo is brought to rest in centre position.
B. Then the Horizontal Servo position is read to determine current angle or position of the servo in the Horizontal Plane.
C. The Horizontal servo position is updated depending on certain inputs.
D. The Horizontal servo is moved to the new position.

SAMPLE CODE:

```
Servo servox;

 pinMode(servopinx,OUTPUT);


void rotate()
{
   servox.attach(servopinx);

 servox.write(servocenterx);                    // center servo
 delay(200);

   posx = servox.read();                         // Read current positions

  if( posx >= incx ) posx += distancex;          // Update new position/angle
  servox.write(posx);
 }
```

4. Neck Flexion

DESCRIPTION: Makes the neck of the robot rotate in vertical plane

INPUT_PARAMETERS: Vertical Servo Position, Y-coordinate of Face on Webcam output

OUTPUT: Vertical Servo Position

EXPLANATION:

A. First the Vertical Servo is brought to rest in centre position.
B. Then the Vertical Servo position is read to determine current angle or position of the servo in the Vertical Plane.
C. The Vertical servo position is updated depending on certain inputs.
D. The Vertical servo is moved to the new position.

SAMPLE CODE:

```
Servo servox;

 pinMode(servopinx,OUTPUT);


void rotate()
{
   servox.attach(servopinx);

 servox.write(servocenterx);                    // center servo
 delay(200);

   posx = servox.read();                         // Read current positions

  if( posx >= incx ) posx += distancex;          // Update new position/angle
  servox.write(posx);
 }
```

Chapter 5

Automation with Microcontroller

Microcontrollers

A microcontroller is basically a computer which is placed on a single integrated circuit chip. It consists of memory, a processor, as well as input-output interfaces. Microcontrollers are programmed to run a certain task, which means, if there is a need to change or enhance its functionality, one must install a new program on the chip. Features that differentiate microcontrollers from other computers (PC, laptop, server, etc.) are:

- All functions are placed on a single chip in a smaller and more compact scale.
- It is programmed to perform a certain task; in order to change its functionality new software must be installed.
- It consumes less power because all physical characteristics are smaller and less energy demanding than in a PC, laptop or server. Developers of microcontrollers concentrate on low energy demand so that mobile applications that use batteries can work longer.
- Single purpose inputs and outputs. Microcontrollers have so called peripherals, which establish connections between a microcontroller and other microcontrollers or computers (e.g. USB, CAN, UART), help to understand the processes in the real physical world (e.g. switching actions, temperature measuring, etc.) and help control conditions (e.g. control motor, trigger alert, etc.)

Microcontrollers can be found in a variety of everyday items: household appliances (e.g. microwave ovens, TV-sets), toys (Lego NXT, talking dolls), vehicles (cars, hoists), etc. Microcontrollers' wide usage has been possible because they are easy to program and have a wide range of functionalities; hence, it is very easy to add new features and upgrade the level of intelligence of the appliance they are in.

Decision of choosing Microcontroller for this project

The Microcontroller used in this project is Arduino UNO. The reason for choosing an Arduino based board was to ensure ease of programming and because of the user-friendly nature of the board and the software. Other available options are Atmega, PIC etc.

Within the Arduino family, the reason for choosing UNO over other options like Leonardo, Mega2560 etc. is due to the minimum pin requirements which are summarized below.

	Sharp	Servos	DC Motors	PWM	**Total**
Analog	1				**1**
Digital		4	4	2	**10**

Table 5.1: Pin requirements for microcontroller

Arduino UNO

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

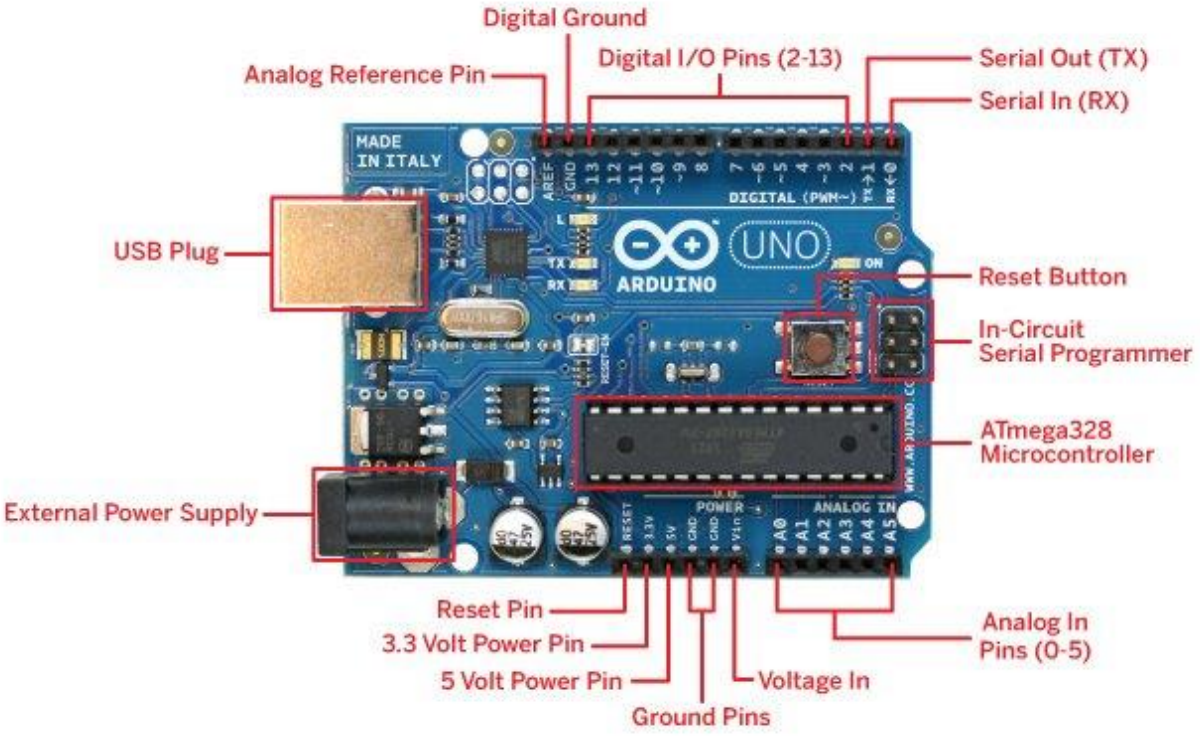


Fig. 5.1: Arduino UNO with description

Physical Characteristics

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

Arduino UNO Circuit Diagram

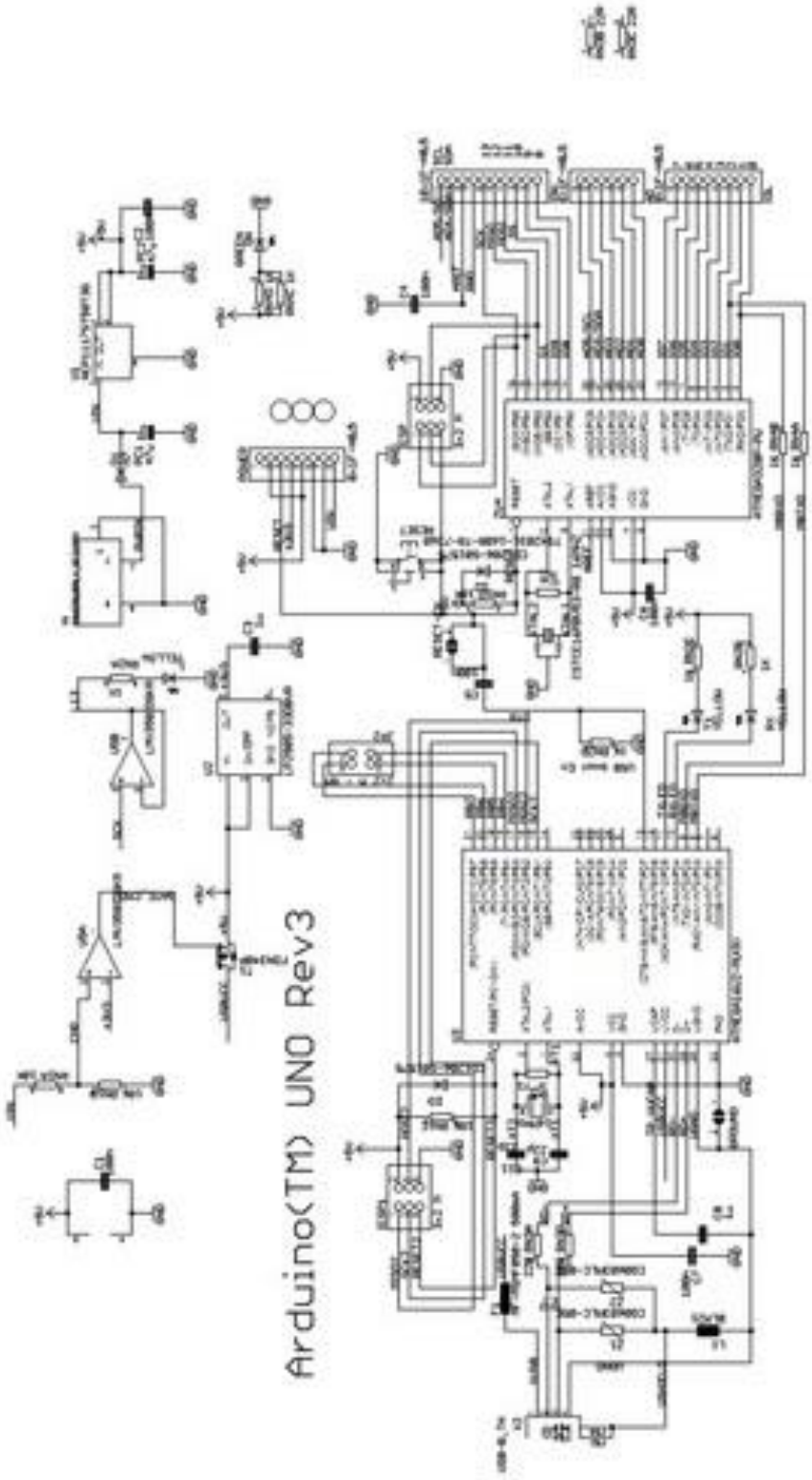


Fig. 5.2: Arduino UNO Circuit Diagram

Features

Power

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

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

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

The power pins are as follows:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V.This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND. Ground pins.
- IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

Memory

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

Input and Output

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

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

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

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

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

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

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software. The user can select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows the user to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

The user can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

The user can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or the user can use the ISP header with an external programmer (overwriting the DFU bootloader).

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of theATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

USB Overcurrent Protection

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

Block Diagram

The Block diagram of Arduino UNO showing all the pins and Power terminals is shown below. The PWM pins are on the Digital side and marked with a tilde (~) sign. The other side contains the 6 analog pins and the power pins which serve a dual, input and output purpose.

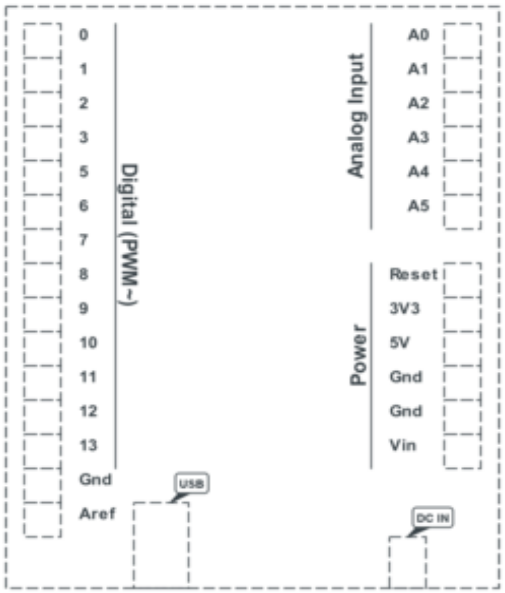


Fig. 5.3: Arduino UNO: Block diagram showing all pins and Power terminals

Chapter 6
Proximity Sensing Using Infrared Sensors

Introduction to Sensors

Sensors are devices converting any kind of physical attributes (temperature, luminance, force, acceleration etc.) to an understandable form for humans or machines. With the help from sensors the microcontroller receives information from the surrounding environment and makes decisions based on it. There are many types of sensors; approximately 195 different types are listed in Wikipedia. Only sensors with electrical output signal can be connected to microcontrollers. Based on electrical output signal, is possible to segment sensors as digital and analogue sensors.

A good sensor obeys the following rules:

- Is sensitive to the measured property only
- Is insensitive to any other property likely to be encountered in its application
- Does not influence the measured property

Ideal sensors are designed to be linear or linear to some simple mathematical function of the measurement, typically logarithmic. The output of such a sensor is an analogue signal and linearly proportional to the value or simple function of the measured property. The sensitivity is then defined as the ratio between output signal and measured property.

In analogue sensor any change in physical attributes changes one of its electrical values, usually voltage, current or resistance. Since microcontrollers are digital devices, the signal have to be converted from analogue to digital before delivering it to controller. For this purpose analogue-digital converters are used witch usually are built-in to the microcontroller.

The basic functions of sensors in robotics can be grouped into:

1) Status Sensors
2) Environment Sensors
3) Quality Control Sensors
4) Safety Sensors
5) Work cell control sensors

The sensors used are:

1) Acoustic Sensors
2) Optic Sensors
3) Pneumatic Sensors
4) Force/ Torque Sensors
5) Optical Encoders

Analogue sensor which already includes digitizer of information, it is called digital sensor. Digital sensors can also standardize information, calibrate sensors and perform a great deal of other functions. There are many ways for transmitting info from digital sensor to

microcontroller: the easiest is with logical signals, more complex way – through some data link interface. The following exercises though, are introducing simpler sensors and sensors known in robotics.

Infrared Sensors

For measuring the distance to an object there are Infrared optic sensors. These are discussed below.

Infrared LED

LED's are special diodes that emit light when connected in a circuit. The light is produced when current passes through in the forward direction. To produce light, the forward voltage must be higher than the diode's internal barrier voltage.

Fig. 6.1: Infrared LED

There are a couple of key differences in the electrical characteristics of infrared LEDs versus visible light LEDs. Infrared LEDs have a lower forward voltage, and a higher rated current compared to visible LEDs. This is due to differences in the material properties of the junction. A typical drive current for an infrared LED can be as high as 50 milliamps. IR (Infrared) sensors are more popular because there's less noise and ambient light than at normal optical wavelengths.

IR Receiver

The IR sensor employed is a photo diode, it is photo detector capable of converting light into either current or voltage, depending upon the mode of operation. Photodiodes are similar to regular semi-conductor diodes except that they may be either exposed (to detect vacuum UV or X-rays) or packaged with a window or optical fibre connection to allow light to reach the

sensitive part of the device. Many diodes designed for use specifically as a photodiode will also use a PIN junction rather than the typical PN junction.

Most photodiodes look similar to a light emitting diode. When a photon of efficient energy strikes the diode, it excites an electron thereby creating a mobile electron and a positively charged electron hole. If the absorption occurs in the junction's depletion region, or one diffusion length away from it, these carriers are swept from the junction by the built-in field of the depletion region. Thus holes move toward the anode, and electron toward the cathode and a photocurrent is produced.

Photovoltaic and photoconductive modes are its two modes of operation.

Critical performance parameters of a photodiode include:

1) Responsivity: The ratio of generated photocurrent to incident light power, typically expressed in A/W when used in photoconductive mode.
2) Dark current: The current through the photodiode in the absence of light, when it is operated in photoconductive mode.
3) Noise-equivalent power (NEP): The minimum input optical power to generate photocurrent, equal to the rms noise current in a 1HZ bandwidth.

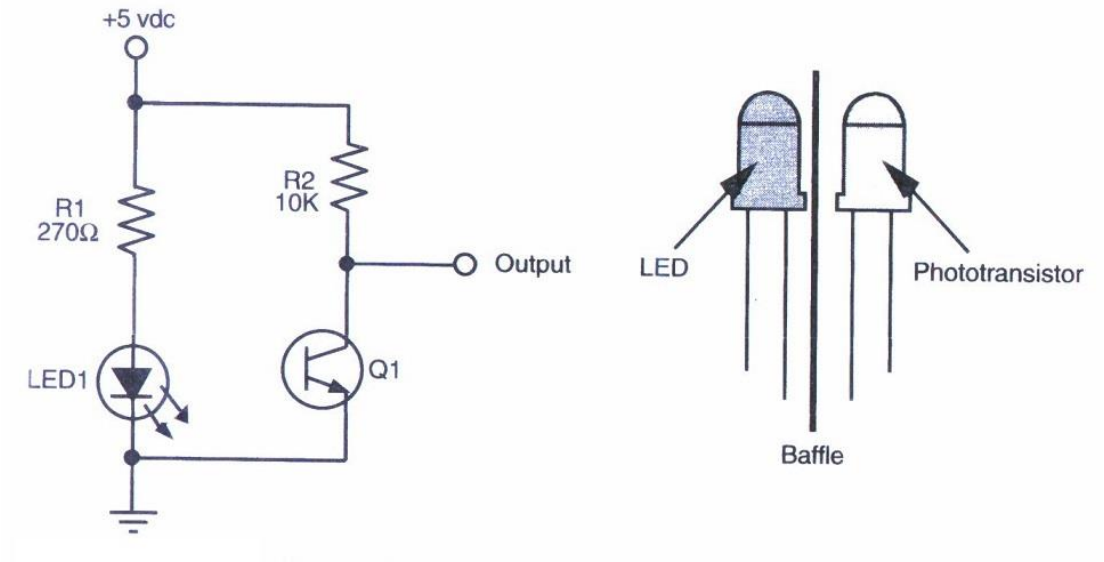


Fig. 6.2: Circuit diagram showing LED and Phototransistor assembly

Infrared Emitter-Detector assembly

The LED and detector have very narrow emission and detection angles, so they are placed 1/8 to 1/4 inch apart maximum, basically parallel and almost contacting.

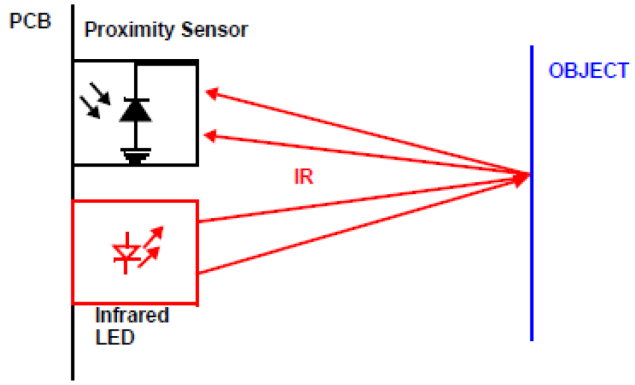


Fig. 6.3: Working of Infrared Sensors

The infrared emitter and detector are assembled together as shown in figure above. The emitter detector basic circuit is as explained below:

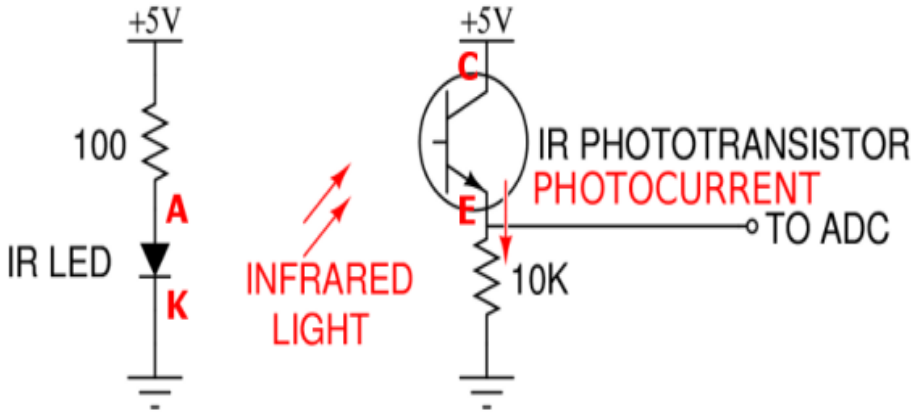


Fig. 6.4: Emitter-Detector Basic Circuit

This SHARP distance sensor bounces IR off objects to determine how far away they are. It returns an analog voltage that can be used to determine how close the nearest object is.

‘Sharp’ Sensors

For measuring the distance to an object there are optical sensors using triangulation measuring method. Company “Sharp” produces most common infra-red (IR) wavelength using distance sensors which have analogue voltage output. The sensors made by “Sharp” have IR LED equipped with lens, which emits narrow light beam. After reflecting from the object, the beam will be directed through the second lens on a position-sensible photo detector (PSD). The conductivity of this PSD depends on the position where the beam falls. The conductivity is converted to voltage and if the voltage is digitalized by using analogue-digital converter, the distance can be calculated. The route of beams reflecting from different distance is presented on the drawing next to the text.

The output of distance sensors by “Sharp” is inversely proportional, this means that when the distance is growing the output is decreasing (decreasing is gradually slowing). Exact graph of the relation between distance and output is usually on the data-sheet of the sensor. All sensors have their specific measuring range where the measured results are creditable and this range depends on the type of the sensor. Maximum distance measured is restricted by two aspects: the amount of reflected light is decreasing and inability of the PSD registering the small changes of the location of the reflected ray. When measuring objects which are too far, the output remains approximately the same as it is when measuring the objects at the maximum distance. Minimum distance is restricted due to peculiarity of Sharp sensors, meaning the output starts to decrease (again) sharply as the distance is at certain point (depending on the model 4-20 cm). This means that to one value of the output corresponds two values of distance. This problem can be avoided by noticing that the object is not too close to the sensor.

GP2Y0A02YK “Sharp” Sensor

GP2Y0A02YK IR Range Sensor is an Infrared distance measuring sensor unit capable of measuring distance from 20 to 150cm. It is composed of an integrated combination of PSD (position sensitive detector), IRED (infrared emitting diode) and signal processing circuit. The variety of the reflectivity of the object, the environmental temperature and the operating duration are not influenced easily to the distance detection because of adopting the triangulation method.

This popular sensor made by Sharp produces an analog output that varies from 2.5V at 20cm to 0.4V at 150cm. Advantages of using this sensor are that an external control circuit is not required and the output can be directly connected to a microcomputer.

Sensor Properties

Sensor Type	Distance (Infrared)
Sensor Output Type	Ratiometric
Measurement Distance Min	200 mm
Measurement Distance Max	1.5 m
Response Time Max	50 ms

Table 6.1: Sensor Properties

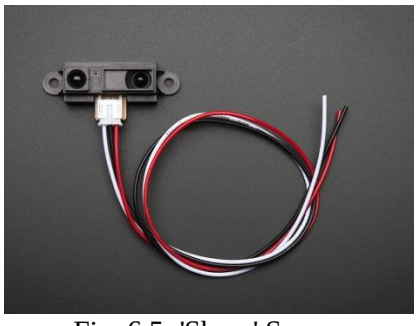


Fig. 6.5: 'Sharp' Sensor

Electrical Properties

Supply Voltage Min	4.5 V DC
Supply Voltage Max	5.5 V DC
Current Consumption Max	50 mA

Table 6.2: Electrical Properties of Sensor

Physical Properties

Weight	6 g
Operating Temperature Min	-10 °C
Operating Temperature Max	60 °C

Table 6.3: Physical Properties of sensor

Principle of Triangulation

The GP2Y0A02YK Infrared Sensor by sharp works on the principle of triangulation. This means that first the transmitter in the sensor emits a collimated beam of infrared radiation which strikes an object present in the line of sight of the sensor.

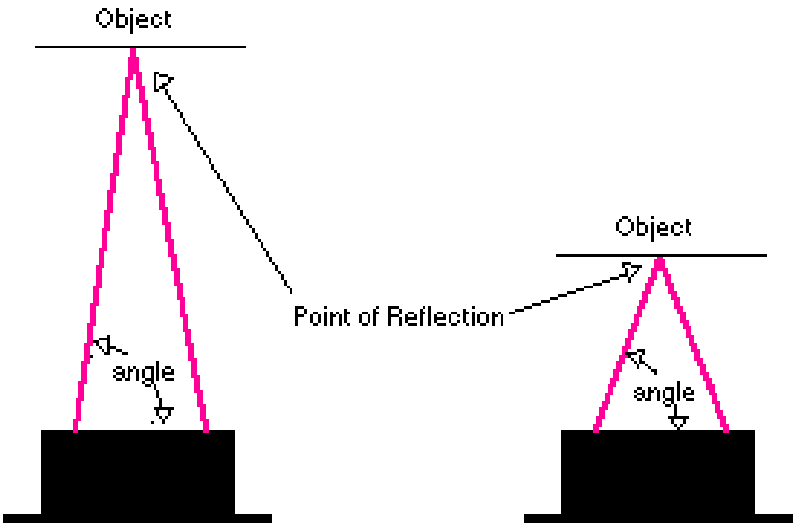


Fig. 6.6: Principle of Triangulation

The beam of Infrared radiation gets reflected back and is received by the detector in the sensor which is in fact a CCD array. This detection method gives us the angle at which the beam hits the array. Since the distance between the transmitter and receiver is fixed as well as known, the signal processing unit in the sensor can calculate the distance of the object from the sensor using simple geometry. It can also make use of a Look-up Table (LUT) for this purpose. This is converted to an analog voltage value and can be measured using the output signal terminal provided on the sensor.

Mathematical Interpretation

On the datasheet of the GP2Y0A21YK is graph of relation between its output voltage and measured distance. This graph is not a linear one, however the graph of inverse values of output voltage and distance almost is, and from that is quite easy to find the formula for converting voltage to distance. To find the formula, the points of the same graph must be inserted to any kind of spreadsheet application and then generate a new graph. In spreadsheet programs is possible to calculate automatically the trend-line. Next, the graph of GP2Y0A02YK corrected output voltage inverse value's relation to the corrected inverse value of measured distance with linear trend-line is presented. To simplify, the output voltage is already converted to 10 bit +5 V values of analogue-digital converter with comparison voltage.

Based on "typical values" from Sharp sensor, the formula to translate Sensor Value into Distance (the formula is only valid for a Sensor Value between 80 and 490) is:

Distance (cm) = 9462/ (Sensor Value - 16.92)

This sensor can find the distance to objects that present a very narrow edge such as a wall at a very sharp angle.

The output of this sensor will vary from unit to unit, and based on the characteristics of the target (reflectance, size, direction of motion, object alignment etc.). This formula is based on the data provided by Sharp. If you find that you are not getting good results with the standard formulas, you may want to derive your own formula to better characterize your situation.

Connection with Microcontroller

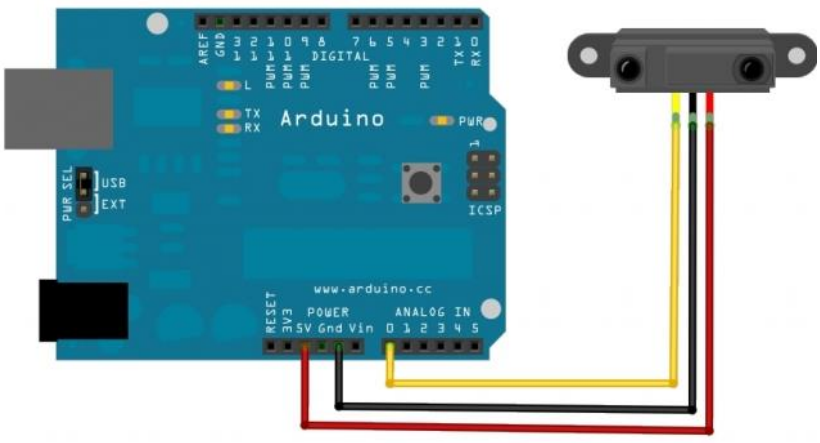
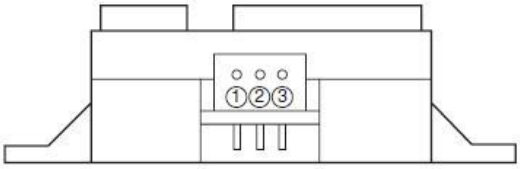


Fig. 6.7: Connection of 'sharp' sensor with Microcontroller

There are three pin terminals leading to three wire connections from the Sharp Sensor to the Microcontroller, here the Arduino Uno. As shown above the 5V terminal from the Arduino goes into the rightmost pin terminal on Sharp(Vcc). The Leftmost pin terminal is the the analog output signal(Vo) to the Arduino which can be connected to any one of the analog inputs A0 to A5. The middle wire(GND) goes to the Gnd terminal on the Arduino.

Functions for Sensors:

The 'Sharp' IR sensor has three pins: V_{CC}, Ground and Signal. The V_{CC} may be between 4.5V and 5.5V. It is normally taken from the 5V pin of the Arduino. The ground terminal is common with the Arduino. Also, the Signal is an analogue input the Arduino which has a value from 0 to 1024.



PIN	SIGNAL NAME
①	V_O
②	GND
③	V_{CC}

Fig. 6.8: Pin Diagram of 'Sharp'Sensor

How the Sensor Works

1. A pulse of IR light is emitted by the emitter.
2. This light travels out in the field of view and hits an object.
3. The reflected light returns to the detector and creates a triangle between the point of reflection, the emitter, and the detector.
4. The angles in this triangle vary based on the distance to the object.
5. The receiver uses a precision lens to transmit the reflected light onto various portions of the enclosed linear CCD array based on the angle of the triangle described above.
6. The CCD array can then determine what angle the reflected light came back at and therefore, it can calculate the distance to the object.

Sample Code

```
int sensorpin = 0;          // analog pin used to connect the sensor
int val = 0;                // to store the values from sensor

void setup()
{
  Serial.begin(9600);           // starts the serial monitor
}

void loop()
{
  val = analogRead(sensorpin);   // reads value of the sensor
  Serial.println(val);          // prints to the serial monitor
  delay(500);                   // time before printing next value
}
```

Modified code for obtaining Distance (in cm)

```
int sensorpin = 0;          // analog pin used to connect the sensor
int distance;
int value;

void setup()
{
  Serial.begin(9600);         // starts the serial monitor
}


void loop()
{ value= analogRead(sensorpin);
  distance = 9462/(value - 16.92);                    // conversion formula
  Serial.print("\nDistance in centimers: ");
  Serial.print(distance);           // print to the serial monitor
  delay(500);                                                          //make it readable
}
```

Modified code for obtaining Voltage (in Volts)

The analog input can also be viewed in terms of voltage. This is made possible by an LUT which comprises of the Voltage values corresponding to each analogue value. An open-source library contains the header file for the same.

```
#include <DistanceGP2Y0A21YK.h>

DistanceGP2Y0A21YK Dist;
int distance;

void setup()
{
  Serial.begin(9600);                                          // starts the serial monitor
  Dist.begin(A0);
}

void loop()
{
  distance = Dist.getDistanceVolt();
  Serial.print("\nADC input mV: ");
  Serial.print(distance);                                      // print to the serial monitor
  delay(500);                                                          //make it readable
}
```

Chapter 7
Face Tracking with Webcam

Face Recognition

Face recognition is an easy task for humans. Even one to three day old babies are able to distinguish between known faces. So how hard could it be for a computer? Are inner features (eyes, nose, mouth) or outer features (head shape, hairline) used for a successful face recognition? How is an image analysed and how does the brain encode it? It was shown by David Hubel and Torsten Wiesel, that the brain has specialized nerve cells responding to specific local features of a scene, such as lines, edges, angles or movement. Since humans don't see the world as scattered pieces, their visual cortex must somehow combine the different sources of information into useful patterns. Automatic face recognition is all about extracting those meaningful features from an image, putting them into a useful representation and performing some kind of classification on them.

Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition. One of the first automated face recognition systems was: marker points (position of eyes, ears, nose ...) were used to build a feature vector (distance between the points, angle between them ...). The recognition was performed by calculating the Euclidean distance between feature vectors of a probe and reference image. Such a method is robust against changes in illumination by its nature, but has a huge drawback: the accurate registration of the marker points is complicated, even with state of the art algorithms. Some of the latest work on geometric face recognition was like a 22-dimensional feature vector was used and experiments on large datasets have shown, that geometrical features alone my not carry enough information for face recognition.

The Eigenfaces method took a holistic approach to face recognition: A facial image is a point from a high-dimensional image space and a lower-dimensional representation is found, where classification becomes easy. The lower-dimensional subspace is found with Principal Component Analysis, which identifies the axes with maximum variance. While this kind of transformation is optimal from a reconstruction standpoint, it doesn't take any class labels into account. Imagine a situation where the variance is generated from external sources, let it be light. The axes with maximum variance do not necessarily contain any discriminative information at all, hence a classification becomes impossible. So a class-specific projection with a Linear Discriminant Analysis was applied to face recognition. The basic idea is to minimize the variance within a class, while maximizing the variance between the classes at the same time.

Recently various methods for a local feature extraction emerged. To avoid the high-dimensionality of the input data only local regions of an image are described, the extracted features are (hopefully) more robust against partial occlusion, illumination and small sample size. Algorithms used for a local feature extraction are Gabor Wavelets, Discrete Cosines Transform and Local Binary Patterns. It's still an open research question what's the best way to preserve spatial information when applying a local feature extraction, because spatial information is potentially useful information.

OpenCV

OpenCV (Open Source Computer Vision) is a popular computer vision library started by Intel in 1999. The cross-platform library sets its focus on real-time image processing and includes patent-free implementations of the latest computer vision algorithms. In 2008 Willow Garage took over support and OpenCV 2.3.1 now comes with a programming interface to C, C++, Python and Android. OpenCV is released under a BSD license so it is used in academic projects and commercial products alike.

OpenCV 2.4 comes with the **FaceRecognizer** class for face recognition.

The algorithms are:

- Eigenfaces
- Fisherfaces
- Local Binary Patterns Histograms

Face Database

We are doing face recognition, so we'll need some face images! Three interesting databases are:

- AT&T Facedatabase: The AT&T Facedatabase, sometimes also referred to as *ORL Database of Faces*, contains ten different images of each of 40 distinct subjects. For some subjects, the images should be taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images should be taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

- Yale Facedatabase A, also known as Yalefaces: The AT&T Facedatabase is good for initial tests, but it's a fairly easy database. The Eigenfaces method already has a 97% recognition rate on it, so any great improvements with other algorithms won't be seen. The Yale Facedatabase A is a more appropriate dataset for initial experiments, because the recognition problem is harder. The database consists of 15 people (14 male, 1 female) each with 11 grayscale images sized 320x243 pixel. There are changes in the light conditions (centre light, left light, right light), facial expressions (happy, normal, sad, sleepy, surprised, wink) and glasses (glasses, no-glasses).

- Extended Yale Facedatabase B : The Extended Yale Facedatabase B contains 2414 images of 38 different people in its cropped version. The focus of this database is set on extracting features that are robust to illumination, the images have almost no variation in emotion/occlusion. The Extended Yale Facedatabase B is the merge of the two databases, which is now known as Extended Yalefacedatabase B.

Local Binary Patterns Histograms

Eigenfaces and Fisherfaces take a somewhat holistic approach to face recognition. The data is treated as a vector somewhere in a high-dimensional image space. We all know high-dimensionality is bad, so a lower-dimensional subspace is identified, where (probably) useful information is preserved. The Eigenfaces approach maximizes the total scatter, which can lead to problems if the variance is generated by an external source, because components with a maximum variance over all classes aren't necessarily useful for classification. So to preserve some discriminative information a Linear Discriminant Analysis should be applied and optimized as described in the Fisherfaces method.

Now real life isn't perfect. None can guarantee perfect light settings in the images or 10 different images of a person. Covariance estimates for the subspace *may* be horribly wrong, so will the recognition. The Eigenfaces method had a 96% recognition rate on the AT&T Facedatabase.

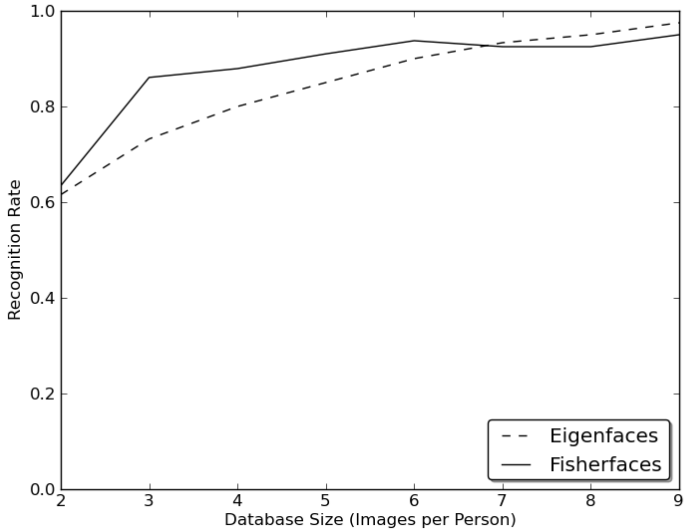


Fig. 7.1: Recognition Rate of Eigenfaces and Fisherfaces with Database Size

This research is concentrated on extracting local features from images. The idea is to not look at the whole image as a high-dimensional vector, but describe only local features of an object. The image representation doesn't only suffer from illumination variations. It also has to be robust against things like scale, translation or rotation in images. Just like **SIFT**, the Local Binary Patterns methodology has its roots in 2D texture analysis. The basic idea of Local Binary Patterns is to summarize the local structure in an image by comparing each pixel with its neighbourhood. Take a pixel as centre and threshold its neighbours against. If the intensity of the centre pixel is greater-equal its neighbour, then denote it with 1 and 0 if not. Binary number for each pixel looks like 11001111. So with 8 surrounding pixels there are 2^8 possible combinations, called *Local Binary Patterns* or sometimes referred to as *LBP codes*. The first LBP operator described in literature actually used a fixed 3 x 3 neighbourhood.

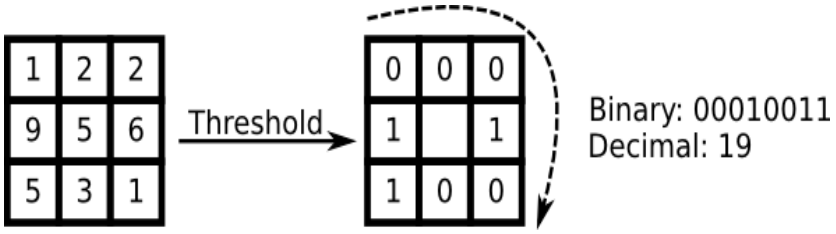


Fig. 7.2: Local Binary Pattern Histograms

Face Detection using Haar Cascades

- Basics of face detection using Haar Feature-based Cascade Classifiers
- Extension of the same for eye detection etc.

Basics

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then one need to extract features from it. For this, haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

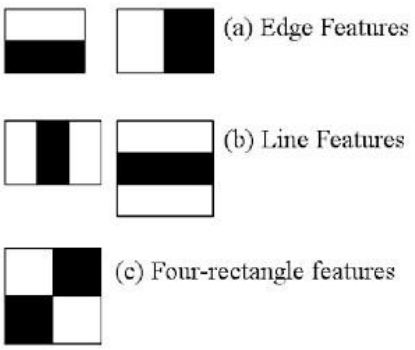


Fig. 7.3: Haar-Cascades Feature types

Now all possible sizes and locations of each kernel is used to calculate plenty of features. (A 24x24 window results over 160000 features). For each feature calculation, sum of pixels under white and black rectangles are needed. To solve this, OpenCV introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels.

But among all these features calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So to select the best features out of 160000+ features, Adaboost is used.

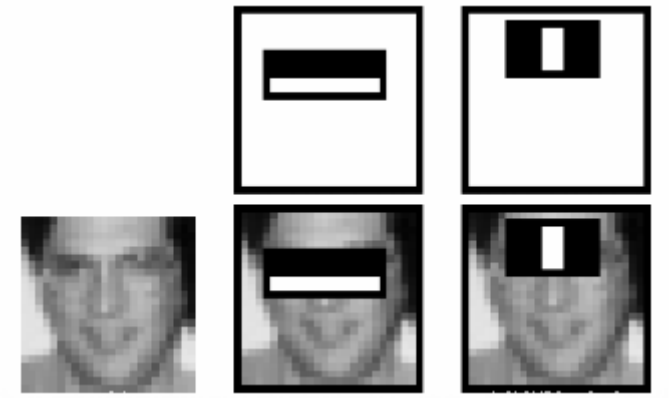


Fig. 7.4: Good features

For this, each and every feature is applied on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. The features with minimum error rate are selected, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

Haar Feature-based Cascade Classifier for Object Detection

The object detector described below has been initially proposed by Paul Viola and improved by Rainer Lienhart.

First, a classifier (namely a *cascade of boosted classifiers working with haar-like features*) is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (*stages*) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers, and are calculated as described below. The current algorithm uses the following Haar-like features:

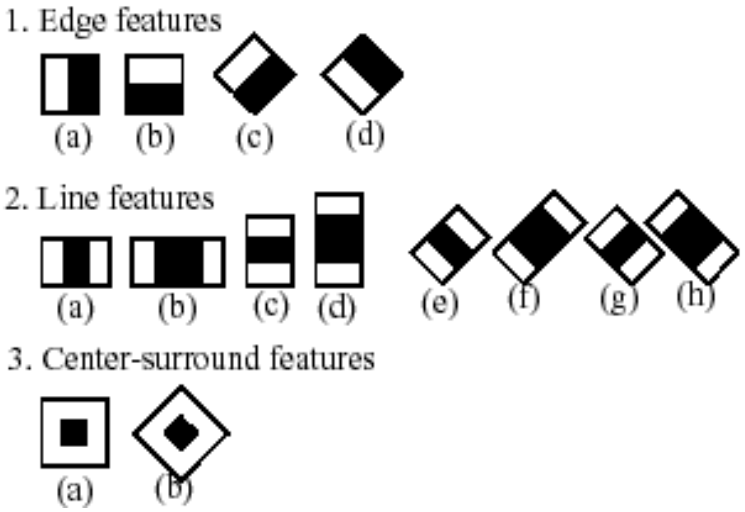


Fig. 7.5: Haar-like features used in algorithm

The feature used in a particular classifier is specified by its shape (1a, 2b etc.), position within the region of interest and the scale (this scale is not the same as the scale used at the detection stage, though these two scales are multiplied). For example, in the case of the third line feature (2c) the response is calculated as the difference between the sum of image pixels under the rectangle covering the whole feature (including the two white stripes and the black stripe in the middle) and the sum of the image pixels under the black stripe multiplied by 3 in order to compensate for the differences in the size of areas. The sums of pixel values over a rectangular regions are calculated rapidly using integral images.

Interfacing circuitry:

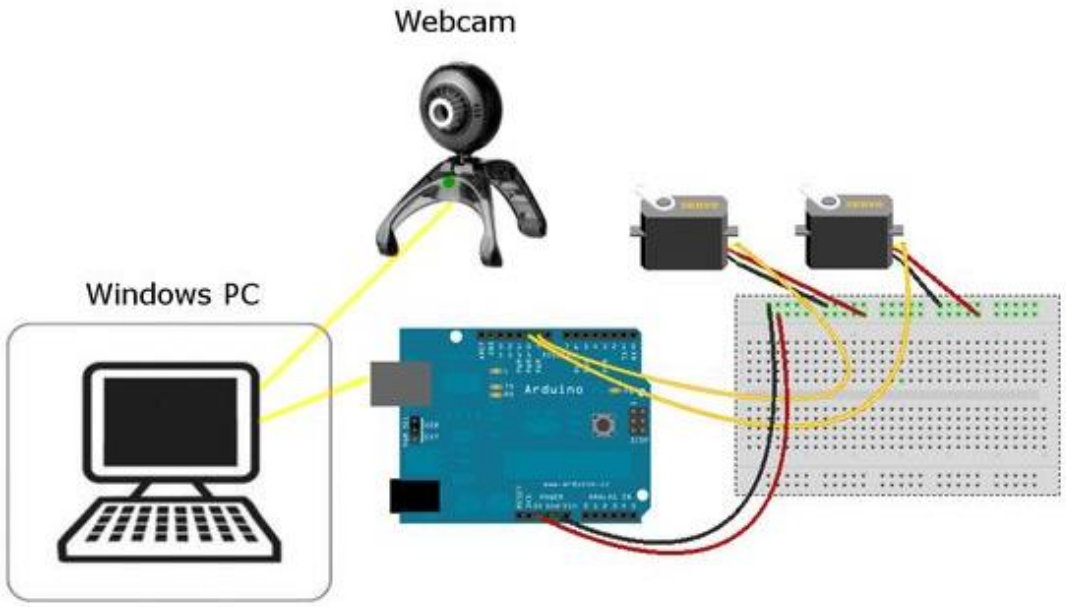


Fig. 7.6: Interfacing webcam and Arduino with PC and using two Servos to move the webcam

Servos

The yellow/signal wire for the pan (X axis) servo goes to digital pin 12.
The yellow/signal wire for the tilt (y axis) servo goes to digital pin 13.
The red/Vcc wires of both servos go to the Arduino's 5V pin.
The black/GND wires of both servos go to Arduino's GND pin.

Webcam

The webcam's USB goes to the PC. The C++ code will identify it via a number representing the USB port it's connected to. .

Arduino

The Arduino Uno is connected to the PC via USB. Take note of the COM port the USB is connected to. One can find the com port from the Arduino Tools/Serial Ports menu. There will be a check mark next to the active USB port. This is the com port that is to be used in the C++ code to communicate with Arduino.

CONCLUSION

Humanoid robots today serve one out of two purposes. Either they make life simpler for humans or they serve a recreational purpose. This humanoid does a mix of both. It doubles up to welcome a person into a room as well as records an image of his face which is easily saved into the storage of a computer for recording purpose. It is friendly in the sense that it is human-like in appearance and is harmless for the most part. It also is smart and aware of its surroundings as it can sense presence of objects or obstacles in its path. Also, it can recognize faces of humans and track them which gives an appearance as if the humanoid is following you with his 'eyes'.

The security purpose of the bot can be employed in a lab setting wherein there is expensive and important equipment and it is not feasible to keep a person in charge of security, day and night. This robot serves a wonderful purpose, wherein it can record who enters (or leaves) the lab, at what time.

There were several challenges and hurdles faced by us during this project. The humanoid robot being a hardware-intensive project, we had to undergo through several hardware failures. The first was discovering the flaw in initial design which was corrected by replacing the body material. Secondly, the L298D motor driver was permanently damaged due to uncontrolled current spikes. The motor Driver had to be replaced with another one with better current rating and heat sinking capacities. Lastly, the battery for Servo motors eventually got drained so much that it could no longer supply adequate voltage to the Servos. An alternate solution was applied wherein Voltage regulation was used to step down the voltage from the battery for DC Motors to supply power to the servos as well.

Despite all challenges, the Humanoid robot is fully functional and the battery supply for the robot is rechargeable and should be recharged every time the terminal voltage falls below the critical voltage level.

FUTURE SCOPE

This project has immense scope for improvement and further work. It is in fact a good feature that this project is adaptable enough to allow for addition of further features into the humanoid.

Features which can be added are discussed below:

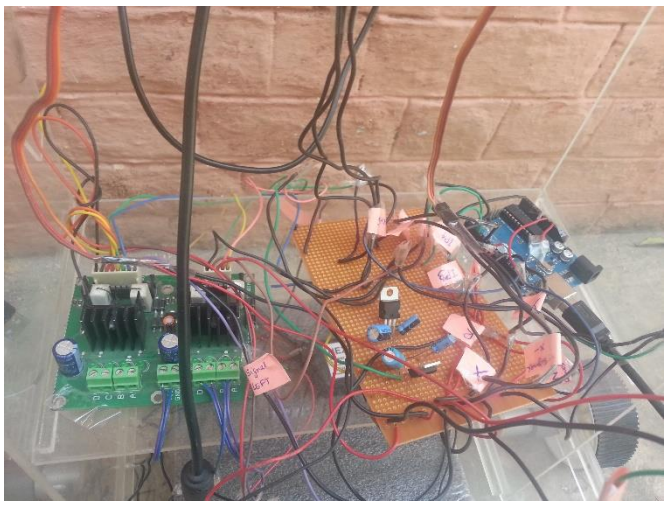
1. SPEAKER
   A Speaker Circuit which would play a short pre-recorded message for welcoming a person or while performing gestures like Hand-shakes. It can also be used to provide instructions for some specific purpose.

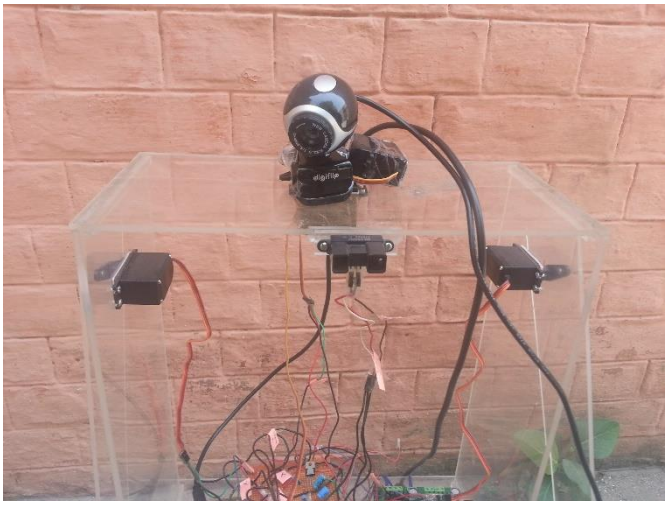
2. HAND GRIPPING MECHANISM
   The Shoulder joints provide arm rotation. This can be extended into wrist or elbow joints which would increase the degree of freedom. Also, gripping mechanisms at hands could enhance the capabilities of the robot to move and pick and place objects.

3. GSM MODULE
   The security purpose of this Humanoid can be further enhanced by adding a GSM Module which would send an SMS to a given mobile number every time a new or unknown person enter the room.

4. FACE RECOGNITION BASED CUSTOMIZATION
   The face recognition software can be further tweaked to customize the database of faces in the storage of a PC, which would add new faces to the storage and also recognize faces which have previously been recorded.

Appendix A: Images of the System





Appendix B: Codes

Arduino Code

```
#include <Servo.h>
#include<Serial.h>

#define  servomaxx   180   // max degree servo horizontal (x) can turn
#define  servomaxy   180   // max degree servo vertical (y) can turn
#define  screenmaxx   320   // max screen horizontal (x)resolution
#define  screenmaxy   240    // max screen vertical (y) resolution
#define  servocenterx   90  // center po#define  of x servo
#define  servocentery   90  // center po#define  of y servo
#define  servopinx   12   // digital pin for servo x
#define  servopiny   13  // digital servo for pin y
#define  baudrate 57600  // com port speed. Must match your C++ setting
//#define distancex 2  // x servo rotation steps
//#define distancey 2  // y servo rotation steps

int distancex = 3;
int distancey = 3;
int valx = 0;       // store x data from serial port
int valy = 0;       // store y data from serial port
int posx = 0;
int posy = 0;
int incx = 10;  // significant increments of horizontal (x) camera movement
int incy = 10;  // significant increments of vertical (y) camera movement

Servo servox;
Servo servoy;

short MSB = 0;  // to build  2 byte integer from serial in byte
short LSB = 0;  // to build  2 byte integer from serial in byte
int   MSBLSB = 0;  //to build  2 byte integer from serial in byte
// transferred from 4May
int EN1 = 6;
int INA1 = 3;
int INB1 = 2;

int EN2 = 11;
int INA2 = 4;
int INB2 = 5;

int serv_l =  8;
int serv_r =  10;

int serv_x =  12;
int serv_y =  13;
```

```
int Sharp = A5;

Servo servo_left;
Servo servo_right;

int pwm  = 140;
int pwm2 = 180;
int pos = 0;

void setup() {

  Serial.begin(baudrate);        // connect to the serial port
  Serial.println("Starting Cam-servo Face tracker");

  pinMode(servopinx,OUTPUT);    // declare the LED's pin as output
  pinMode(servopiny,OUTPUT);    // declare the LED's pin as output

  servoy.attach(servopiny);
  servox.attach(servopinx);

  // center servos

  servox.write(servocenterx);
  delay(200);
  servoy.write(servocentery);
  delay(200);
  //code below setup from 4May
  pinMode(EN1, OUTPUT);
  pinMode(INA1, OUTPUT);
  pinMode(INB1, OUTPUT);

  pinMode(EN2, OUTPUT);
  pinMode(INA2, OUTPUT);
  pinMode(INB2, OUTPUT);

  servo_left.attach(serv_l);
  servo_right.attach(serv_r);

  pinMode(Sharp,INPUT);
}

int p = 0;

void serialEvent(){

  while(1)
  {
  //servox.write(0);
 while(Serial.available() <=0); // wait for incoming serial data
  if (Serial.available() >= 4)  // wait for 4 bytes.
```

```
 {

  // get X axis 2-byte integer from serial
  MSB = Serial.read();
  delay(5);
  LSB = Serial.read();
  MSBLSB=word(MSB, LSB);
  valx = MSBLSB;
  delay(5);

  // get Y axis 2-byte integer from serial
  MSB = Serial.read();
  delay(5);
  LSB = Serial.read();
  MSBLSB=word(MSB, LSB);
  valy = MSBLSB;
  delay(5);

  // read last servos positions
  posx = servox.read();
  posy = servoy.read();
  if(p==0)
  {
    p=1;
    distancex= 3;
    distancey =3;
  }
  else if (p==1)
  {
    p=2;
    distancex= 2;
    distancey =2;
  }
  else if(p==2)
  {
    p=0;
    distancex= 1;
    distancey =1;
  }
 //Find out if the X component of the face is to the left of the middle of the screen.
  if(valx < (screenmaxx/2 - incx)){

    if( posx >= incx ) posx += distancex; //Update the pan position variable to move the servo to the left.
  }
  //Find out if the X component of the face is to the right of the middle of the screen.
  else if(valx > screenmaxx/2 + incx){
    if(posx <= servomaxx-incx) posx -=distancex; //Update the pan position variable to move the servo to
the right.
  }

  //Find out if the Y component of the face is below the middle of the screen.
```

```
    if(valy < (screenmaxy/2 - incy)){
      if(posy >= 5)posy += distancey; //If it is below the middle of the screen, update the tilt position
variable to lower the tilt servo.
    }
    //Find out if the Y component of the face is above the middle of the screen.
    else if(valy > (screenmaxy/2 + incy)){
      if(posy <= 175)posy -= distancey; //Update the tilt position variable to raise the tilt servo.
    }
  // Servos will rotate accordingly
    servox.write(posx);
    servoy.write(posy);

  }
 digitalWrite(INA1, HIGH);
  digitalWrite(INB1, HIGH);
  digitalWrite(INA2, HIGH);
  digitalWrite(INB2, HIGH);

  analogWrite(EN1, 0);
  analogWrite(EN2, 0);
 }
}

int at=0;

void loop()
{
   for(pos = 0; pos < 90; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    servo_left.write(pos);
    servo_right.write(pos+60);    // tell servo to go to position in variable 'pos'
    delay(8);                       // waits 8ms for the servo to reach the position
  }
  for(pos = 90; pos>=0; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    servo_left.write(pos);
    servo_right.write(pos+60);              // tell servo to go to position in variable 'pos'
    delay(8);                       // waits 8ms for the servo to reach the position
  }

   digitalWrite(INA1, HIGH);
  digitalWrite(INB1, LOW);
  digitalWrite(INA2, HIGH);
  digitalWrite(INB2, LOW);

  analogWrite(EN1, pwm);
  analogWrite(EN2, pwm2);

 int a = analogRead(Sharp);
  int D = 9462/(a - 16.92);
}
```

C++ Code

Face Detection and Tracking

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include "Tserial.h"

using namespace std;
using namespace cv;

/** Function Headers */
void detectAndDisplay( Mat frame );

/** Global variables */
String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;
string window_name = "Capture - Face detection ";

// Serial to Arduino global declarations
  int arduino_command;
  Tserial *arduino_com;
  short MSBLSB = 0;
  unsigned char MSB = 0;
  unsigned char LSB = 0;
// Serial to Arduino global declarations

int main( int argc, const char** argv )
{
  CvCapture* capture;
  Mat frame;

  // serial to Arduino setup
  arduino_com = new Tserial();
  if (arduino_com!=0) {
      arduino_com->connect("COM3", 57600, spNONE); }
  // serial to Arduino setup

  //-- 1. Load the cascades
  if( !face_cascade.load( face_cascade_name ) ){ printf("--(!)Error loading\n"); return -1; };
  if( !eyes_cascade.load( eyes_cascade_name ) ){ printf("--(!)Error loading\n"); return -1; };

  //-- 2. Read the video stream
  capture = cvCaptureFromCAM( 1 );
  if( capture )
  {
    while( true )
    {
      frame = cvQueryFrame( capture );
      //-- 3. Apply the classifier to the frame
```

```
    if( !frame.empty() )
     { detectAndDisplay( frame ); }
    else
     { printf(" --(!) No captured frame -- Break!"); break; }

    int c = waitKey(5);
    if( (char)c == 'c' ) { break; }
   }
  }
 // Serial to Arduino - shutdown
   arduino_com->disconnect();
   delete arduino_com;
   arduino_com = 0;
 // Serial to Arduino - shutdown
 return 0;
}

/**
 * @function detectAndDisplay
 */
void detectAndDisplay( Mat frame )
{

  std::vector<Rect> faces;
  Mat frame_gray;

  cvtColor( frame, frame_gray, CV_BGR2GRAY );
  equalizeHist( frame_gray, frame_gray );
  //-- Detect faces
  face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );

  for( int i = 0; i < faces.size(); i++ )
   {
    Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );
    ellipse( frame, center, Size( faces[i].width*0.5, faces[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 255 ), 2, 8, 0 );

    // send X,Y of face center to serial com port
    // send X axis
            // read least significant byte
    LSB = faces[i].x & 0xff;
    // read next significant byte
    MSB = (faces[i].x >> 8) & 0xff;
            arduino_com->sendChar( MSB );
            arduino_com->sendChar( LSB );

   // Send Y axis
           LSB = faces[i].y & 0xff;
           MSB = (faces[i].y >> 8) & 0xff;
           arduino_com->sendChar( MSB );
           arduino_com->sendChar( LSB );
   // serial com port send

    Mat faceROI = frame_gray( faces[i] );
    std::vector<Rect> eyes;
   }

  //-- Show what you got
  imshow( window_name, frame );
}
```

Serial Communication

Tserial.cpp

```
/* ---------------------------------------------------------------------- --
--                                                                        --
--                    PC serial port connection object                    --
--                      for non-event-driven programs                     --
--                                                                        --
--                                                                        --
--                                                                        --
--  Copyright @ 2001         Thierry Schneider                            --
--                         thierry@tetraedre.com                          --
--                                                                        --
--                                                                        --
--                                                                        --
-- ---------------------------------------------------------------------- --
--                                                                        --
--  Filename : tserial.cpp                                                --
--  Author   : Thierry Schneider                                          --
--  Created  : April 4th 2000                                             --
--  Modified : April 8th 2001                                             --
--  Plateform: Windows 95, 98, NT, 2000 (Win32)                           --
-- ---------------------------------------------------------------------- --
--                                                                        --
--  This software is given without any warranty. It can be distributed    --
--  free of charge as long as this header remains, unchanged.             --
--                                                                        --
-- ---------------------------------------------------------------------- */

/* ---------------------------------------------------------------------- */
#define STRICT
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include <conio.h>
#include <windows.h>

#include "Tserial.h"

/* ---------------------------------------------------------------------- */
/* ------------------------    Tserial   ---------------------------- */
/* ---------------------------------------------------------------------- */
Tserial::Tserial()
{
   parityMode      = spNONE;
   port[0]         = 0;
   rate            = 0;
   serial_handle   = INVALID_HANDLE_VALUE;
}
/* ---------------------------------------------------------------------- */
/* ------------------------    ~Tserial    ------------------------ */
/* ---------------------------------------------------------------------- */
Tserial::~Tserial()
{
   if (serial_handle!=INVALID_HANDLE_VALUE)
      CloseHandle(serial_handle);
```

```
   serial_handle = INVALID_HANDLE_VALUE;
}
/* ------------------------------------------------------------------ */
/* -------------------------    disconnect   ------------------------- */
/* ------------------------------------------------------------------ */
void Tserial::disconnect(void)
{
   if (serial_handle!=INVALID_HANDLE_VALUE)
      CloseHandle(serial_handle);
   serial_handle = INVALID_HANDLE_VALUE;
}
/* ------------------------------------------------------------------ */
/* -------------------------    connect     ------------------------- */
/* ------------------------------------------------------------------ */
int  Tserial::connect         (char *port_arg, int rate_arg, serial_parity parity_arg)
{
   int erreur;
   DCB  dcb;
   COMMTIMEOUTS cto = { 0, 0, 0, 0, 0 };

   /* ------------------------------------------- */
   if (serial_handle!=INVALID_HANDLE_VALUE)
      CloseHandle(serial_handle);
   serial_handle = INVALID_HANDLE_VALUE;

   erreur = 0;

   if (port_arg!=0)
   {
      strncpy(port, port_arg, 10);
      rate     = rate_arg;
      parityMode= parity_arg;
      memset(&dcb,0,sizeof(dcb));

      /* ------------------------------------------------------------------ */
      // set DCB to configure the serial port
      dcb.DCBlength      = sizeof(dcb);

      /* ---------- Serial Port Config ------- */
      dcb.BaudRate       = rate;

      switch(parityMode)
      {
         case spNONE:
                  dcb.Parity     = NOPARITY;
                  dcb.fParity    = 0;
                  break;
         case spEVEN:
                  dcb.Parity     = EVENPARITY;
                  dcb.fParity    = 1;
                  break;
         case spODD:
                  dcb.Parity     = ODDPARITY;
                  dcb.fParity    = 1;
                  break;
      }


      dcb.StopBits       = ONESTOPBIT;
```

```
    dcb.ByteSize       = 8;

    dcb.fOutxCtsFlow    = 0;
    dcb.fOutxDsrFlow    = 0;
    dcb.fDtrControl     = DTR_CONTROL_DISABLE;
    dcb.fDsrSensitivity = 0;
    dcb.fRtsControl     = RTS_CONTROL_DISABLE;
    dcb.fOutX           = 0;
    dcb.fInX            = 0;

    /* ----------------- misc parameters ----- */
    dcb.fErrorChar      = 0;
    dcb.fBinary         = 1;
    dcb.fNull           = 0;
    dcb.fAbortOnError   = 0;
    dcb.wReserved       = 0;
    dcb.XonLim          = 2;
    dcb.XoffLim         = 4;
    dcb.XonChar         = 0x13;
    dcb.XoffChar        = 0x19;
    dcb.EvtChar         = 0;

    /* ---------------------------------------------------------------------- */

                        serial_handle    = CreateFile(port, GENERIC_READ | GENERIC_WRITE,
                       0, NULL, OPEN_EXISTING,NULL,NULL);
              // opening serial port


    if (serial_handle    != INVALID_HANDLE_VALUE)
    {
       if(!SetCommMask(serial_handle, 0))
          erreur = 1;

       // set timeouts
       if(!SetCommTimeouts(serial_handle,&cto))
          erreur = 2;

       // set DCB
       if(!SetCommState(serial_handle,&dcb))
          erreur = 4;
    }
    else
       erreur = 8;
  }
  else
    erreur = 16;


  /* ---------------------------------------------- */
  if (erreur!=0)
  {
    CloseHandle(serial_handle);
    serial_handle = INVALID_HANDLE_VALUE;
  }
  return(erreur);}


/* ---------------------------------------------------------------------- */
```

```
/* ------------------------    sendChar    ------------------------ */
/* ------------------------------------------------------------------ */
void Tserial::sendChar(char data)
{
   sendArray(&data, 1);
}

/* ------------------------------------------------------------------ */
/* ------------------------    sendArray   ------------------------ */
/* ------------------------------------------------------------------ */
void Tserial::sendArray(char *buffer, int len)
{
   unsigned long result;

   if (serial_handle!=INVALID_HANDLE_VALUE)
      WriteFile(serial_handle, buffer, len, &result, NULL);
}
/* ------------------------------------------------------------------ */
/* ------------------------    getChar     ------------------------ */
/* ------------------------------------------------------------------ */
char Tserial::getChar(void)
{
   char c;
   getArray(&c, 1);
   return(c);
}

/* ------------------------------------------------------------------ */
/* ------------------------    getArray    ------------------------ */
/* ------------------------------------------------------------------ */
int  Tserial::getArray       (char *buffer, int len)
{
   unsigned long read_nbr;

   read_nbr = 0;
   if (serial_handle!=INVALID_HANDLE_VALUE)
   {
      ReadFile(serial_handle, buffer, len, &read_nbr, NULL);
   }
   return((int) read_nbr);
}
/* ------------------------------------------------------------------ */
/* ------------------------    getNbrOfBytes ------------------------ */
/* ------------------------------------------------------------------ */
int Tserial::getNbrOfBytes    (void)
{
   struct _COMSTAT status;
   int            n;
   unsigned long   etat;

   n = 0;

   if (serial_handle!=INVALID_HANDLE_VALUE)
   {
      ClearCommError(serial_handle, &etat, &status);
      n = status.cbInQue;
   }

   return(n);}
```

REFERENCES

1. http://www.societyofrobots.com/robottheory.shtml
2. http://forums.trossenrobotics.com/tutorials/
3. http://www.learningaboutelectronics.com/Articles/
4. http://electronics.stackexchange.com/questions/94346/battery-power-supply-for-arduino-and-dc-motors-with-l293d
5. http://forum.arduino.cc/index.php/
6. http://communityofrobots.com/tutorial/kawal/how-use-sharp-ir-sensor-arduino
7. http://www.phidgets.com/products.php?product_id=3522
8. http://www.instructables.com/id/Face-detection-and-tracking-with-Arduino-and-OpenC/
9. http://docs.opencv.org/trunk/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html
10. http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html
11. http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#CascadeClassifier