International Conference on Advances in Materials, Computing and Communication Technologies
In Association with International Journal of Scientific Research in Science and Technology
Volume 9 | Issue 1 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)

CNN Algorithm Based Fault Identification in Three Phase Induction Motor Using Artificial Intelligence Techniques

G. Shasikala[1], M. Rahila Thaslim[2], E.Aswini[2], S. Sasi[2], M. Divya[2]

[1]Assistant Professor, Department of Electrical and Electronic Engineering, Er. Perumal Manimekelai college of Engineering, Hosur, Tamil Nadu, India

[2]Department of Electrical and Electronic Engineering, Er. Perumal Manimekelai college of Engineering, Hosur, Tamil Nadu, India

ABSTRACT

The motors are one of the most crucial electrical equipment and are extensively used in industries in a wide range of applications. This project presents a machine learning model for the fault detection and classification of motor faults by using three phase voltages and currents as inputs. The aim of this work is to protect vital electrical components and to prevent abnormal event progression through early detection and diagnosis. This work presents a fast forward convolutional neural network model to detect some of the commonly occurring electrical faults like overvoltage, under voltage, unbalanced voltage, overload, ground fault. A separate model free monitoring system wherein the motor itself acts like a sensor is presented and the only monitored signals are the input given to the motor. Limits for current and voltage values are set for the faulty and healthy conditions, which is done by a classifier. Real time data from a motor is used to train and test the neural network. The model so developed analyses the voltage and current values given at a particular instant and classifies the data into no fault or the specific fault. The model is then interfaced with a real motor to accurately detect and classify the faults so that further necessary action can be taken.

Keywords : Motor, Fault Analysis, Machine Learning, CNN, Predictive Maintenance.

I. INTRODUCTION

Aiming at overcoming the limitation caused by single sensor signal and improving fault classification accuracy and model stability, a multi-signal CNN-based approach using time frequency distribution image is proposed in this paper, and both the vibration and current signals are leveraged. Compared with previous works, the proposed deep architecture is able to learn from multiple input signals simultaneously so that the classification accuracy and the stability of the deep model are improved. Through model training, the proposed deep architecture is able to automatically learn and select suitable features that contribute to accurate classification, which reduces overall human intervention. The main contributions of this project can be summarized as follows, This project proposes a multi-signal model for induction motor fault diagnosis, and the proposed framework is able to analyze multiple sensor signals simultaneously. By leveraging various sensor signals, more useful features are obtained, which finally contributes to accurate prediction. Besides, in order to achieve optimal performance, two different model architectures are designed and investigated. This project builds a deep architecture based on convolutional neural network to establish multi-signal fault diagnosis framework. Convolutional neural network has the capability of automatically learning representations layer by layer and gives condition prediction based on learned features. Time-frequency images of each sensor signal are obtained by wavelet transform and are used as the input samples of the proposed deep architecture.

II. LITERATURE SURVEY

1) **B. Samanta and C. Nataraj, "Use of particle swarm optimization for machinery fault detection", Engineering Applications of Artificial Intelligence**

However, to achieve reliable and robust induction motor fault diagnosis and realize detection of several faults simultaneously, including stator faults, rotor faults and bearing faults, various types of sensor signals should be leveraged together. This is because each type of sensor signal contains specific information related to certain working condition. Thus, combing these signals and analyzing them together can be an effective way to realize multi-fault diagnosis under complex conditions. In fact, feature extraction from sensor signals is a crucial step, and the performance of fault diagnosis method depends on how precise the extracted features can represent the fault signature of the monitored induction motor. Nevertheless, the performance of conventional feature-based fault diagnosis depends on the quality of extracted features and its suitability to a certain task. This requires expert knowledge and human intervention, consequently bringing uncertainty to overall performance.

2) **B.S. Yang, M.S. Oh, A. C. C. Tan et al., "Fault diagnosis of induction motor based on decision trees and adaptive neuro-fuzzy inference".**

Machine learning introduces an effective solution to above problems. Successful applications including Support Vector Machine (SVM), Bayesian Networks (BN), Artificial Neural Networks (ANNs) , fuzzy inference and deep learning (DL), have been seen in various fault diagnosis scenario. Among these techniques, deep learning presents an attractive option for feature learning, benefiting from its powerful learning ability from massive data in a much deeper network structure. Deep learning architectures refer to neural networks that have multiple hidden layers, which are able to learn hierarchical representations from raw data. With the improvement of computing abilities and advanced techniques of network training, deep learning has succeeded in various applications such as speech recognition, natural language processing .

3) **R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring"**

In machine fault diagnosis, deep architecture can automatically learn useful representations from massive sensor data and recognize certain fault patterns . Compared with traditional ways in fault diagnosis where features are manually extracted and selected, which require prior knowledge and great efforts during analysis, deep learning method can learn representations and patterns hierarchically from input data and select those that can best represent machine working condition through gradually adjusting connected weights. This reduces influence caused by human intervention and improves efficiency. By making use of the deep architecture and massive training data, deep learning method has the ability to model complex working conditions and output accurate predictions. There are several typical deep architectures that have been successfully applied to machine fault diagnosis.

4) S. Shao, W. Sun, R. Yan, P. Wang, and R. Gao, "A deep learning approach for fault diagnosis of induction motors in manufacturing"

For induction motor fault diagnosis, our previous studies include both traditional feature-based methods and machine learning-based methods. Multiple class feature selection (MCFS) for induction motor fault diagnosis was proposed in using selected features. Meanwhile, several works focus on introducing deep learning for induction motor fault diagnosis to realize intelligence fault recognition. For example, 1D CNN-based convolutional discriminative feature learning (CDFL) method is used to learn from vibration signals and SVM is used to classify multiple faults. A one-layer sparse auto-encoder-based deep learning approach has been proved to be effective in induction motor fault diagnosis and a DBN based fault diagnosis framework is proposed and the influence of model configuration is investigated .

5) W. Sun, R. Zhao, R. Yan, S. Shao, and X. Chen, "Convolutional discriminative feature learning for induction motor fault diagnosis".

Accordingly, induction motor fault diagnosis plays a significant role in safety production and equipment maintenance, and it can improve machine quality, enhance working safety and reduce maintenance cost. Large amounts of researches have been carried out in developing reliable and efficient induction motor fault diagnosis techniques. For decades, many fault diagnosis methods are based on motor-current analysis which is called Motor Current Signature Analysis (MCSA). After data acquisition, critical features associated with the working state are extracted from motor current data using signal processing techniques. MCSA has been one of the most widely used fault diagnosis techniques for induction motor as it is inexpensive and noninvasive, and it turns out to be successful in detecting stator faults and rotor faults.

III. EXISTING METHOD AND PROPOSED SYSTEM

3.1 EXISTING METHOD

In the present scenario when a fault occurs, detecting fault source is difficult and entire line has to be dug in order to check entire line and fix faults. In existing method, the fault has to be detected only by analyzing the complete circuit which consumes a lot man power as well as money. Further immediate detection of fault is required to repair it and to avoid the consequences. But the existing method failed in that purpose.

3.1.1. DISADVANTAGES

- Time-consuming
- Delay in process
- Consumption of man-power and money
- Inefficiency in locating the faults

3.2 PROPOSED SYSTEM

This project is organized as follows. Theoretical background and related work of time-frequency distribution and convolutional neural network is introduced in Section. The proposed CNN based method for motor fault diagnosis is gives experimental verification based on sensor data acquired on a machine fault simulator and comparison experiments are implemented to compare performances between traditional methods and the proposed method .Experiments are carried out to verify the effectiveness of the proposed approach and both vibration and voltage and current signals of motor under six different working conditions are utilized. In addition, single-signal model based on deep model and some other fault diagnosis methods, including conventional feature-based method and feature learning-based methods are investigated to give a comparison. an AVR microcontroller based motor fault detection system. The basic principle behind the system is Ohms law. When fault occurs in the motor, the voltage varies which is used to calculate the fault distance

IV. SYSTEM FUNCTION

4.1 BLOCK DIAGRAM

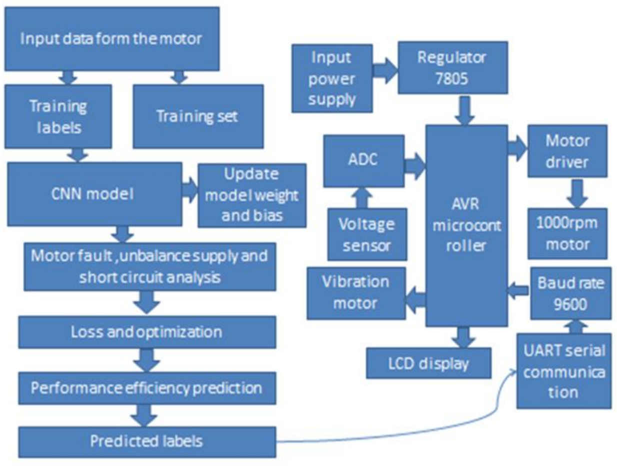
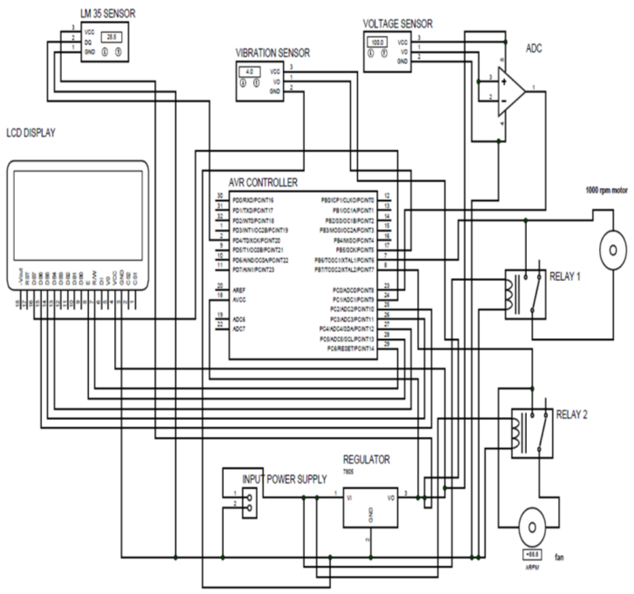


Fig.no: : Block Diagram of the proposed System

4.2 CIRCUIT DIAGRAM



V. HARDWARE REQUIREMENTS

5.1 AVR MICROCONTROLLER

AVR was developed in the year 1996 by Atmel Corporation. The architecture of **AVR** was developed by Alf-Egil Bogen and Vegard Wollan. AVR derives its name from its developers and stands for **Alf-Egil Bogen Vegard Wollan RISC microcontroller**, also known as Advanced Virtual RISC. **AVR micro controllers** are available in three categories:

1. TinyAVR

Less memory, small size, suitable only for simpler applications

2. MegaAVR

These are the most popular ones having good amount of memory (upto 256 KB), higher number of inbuilt peripherals and suitable for moderate to complex applications.

3. XmegaAVR

Used commercially for complex applications, which require large program memory and high speed.

The following table compares the above mentioned AVR series of microcontrollers:

Series Name	Pins	Flash Memory	Special Feature
TinyAVR	6-32	0.5-8 KB	Small in size
MegaAVR	28-100	4-256KB	Extended peripherals
XmegaAVR	44-100	16-384KB	DMA,Event System included

Importance of AVR

What's special about AVR?

They are fast: **AVR microcontroller** executes most of the instructions in single execution cycle. AVRs are about 4 times faster than PICs, they consume less power and can be operated in different power saving modes. Let's do the comparison between the three most commonly used families of microcontrollers.

	8051	PIC	AVR
SPEED	Slow	Moderate	Fast
MEMORY	Small	Large	Large
ARCHITECTURE	CISC	RISC	RISC

ADC	Not Present	Inbuilt	Inbuilt
Timers	Inbuilt	Inbuilt	Inbuilt
PWM Channels	Not Present	Inbuilt	Inbuilt

AVR is an 8-bit microcontroller belonging to the family of Reduced Instruction Set Computer (**RISC**). In RISC architecture the instruction set of the computer are not only fewer in number but also simpler and faster in operation. The other type of categorization is CISC (Complex Instruction Set Computers). Click to find out differences between RISC and CISC. We will explore more on this when we will learn about the architecture of AVR microcontrollers in following section.

8-bit means that the microcontroller is capable of transmitting and receiving 8-bit data. The input/output registers available are of 8-bits. The AVR family controllers have register based architecture which means that both the operands for an operation are stored in a register and the result of the operation is also stored in a register. Following figure shows a simple example performing OR operation between two input registers and storing the value in Output Register.

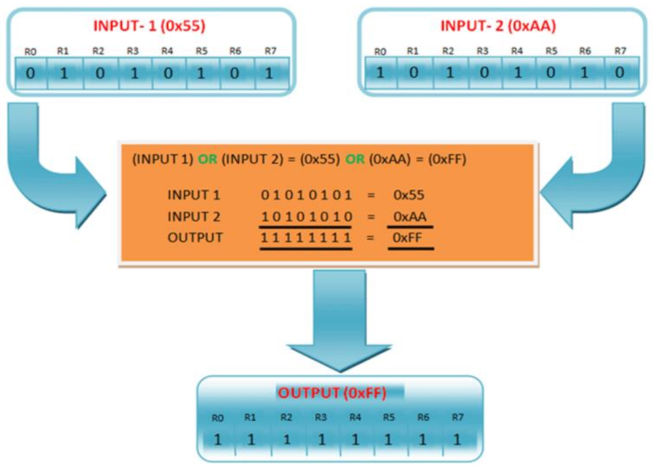


Fig. 2: Block Diagram Showing Simple Example Carrying Out OR Operation Between Two Input Registers And Value Storage In Output Register

The CPU takes values from two input registers INPUT-1 and INPUT-2, performs the logical operation and stores the value into the OUTPUT register. All this happens in 1 execution cycle.

In our journey with the AVR we will be working on Atmega16 microcontroller, which is a 40-pin IC and belongs to the megaAVR category of AVR family. Some of the features of Atmega16 are:

a. 16KB of Flash memory
b. 1KB of SRAM
c. 512 Bytes of EEPROM
d. Available in 40-Pin DIP
e. 8-Channel 10-bit ADC
f. Two 8-bit Timers/Counters
g. One 16-bit Timer/Counter
h. 4 PWM Channels
i. In System Programmer (ISP)
j. Serial USART
k. SPI Interface
l. Digital to Analog Comparator.

Architecture of AVR

The AVR microcontrollers are based on the advanced RISC architecture and consist of 32 x 8-bit general purpose working registers. Within one single clock cycle, AVR can take inputs from two general purpose registers and put them to ALU for carrying out the requested operation, and transfer back the result to an arbitrary register. The ALU can perform arithmetic as well as logical operations over the inputs from the register or between the register and a constant. Single register operations like taking a complement can also be executed in ALU. We can see that AVR does not have any register like accumulator as in 8051 family of microcontrollers; the operations can be performed between any of the registers and can be stored in either of them.

AVR follows Harvard Architecture format in which the processor is equipped with separate memories and buses for Program and the Data information. Here while an instruction is being executed, the next instruction is pre-fetched from the program memory.

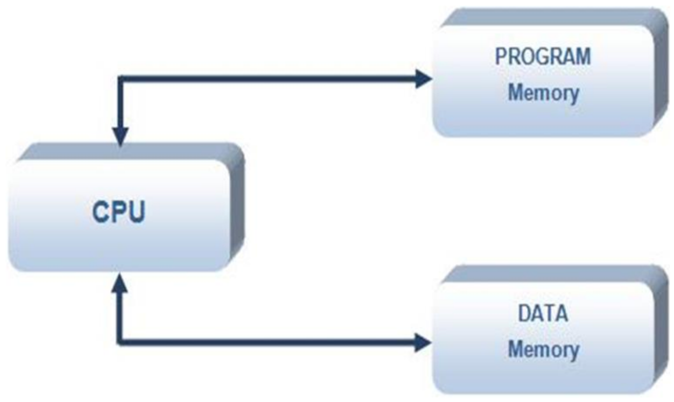


Fig. 3: Block Diagram Of memory architecture In AVR

Since AVR can perform single cycle execution, it means that AVR can execute 1 million instructions per second if cycle frequency is 1MHz. The higher is the operating frequency of the controller, the higher will be its processing speed. We need to optimize the power consumption with processing speed and hence need to select the operating frequency accordingly.

There are two flavors for Atmega16 microcontroller:

1. **Atmega16:-** Operating frequency range is 0 – 16 MHz.
2. **Atmega16L:-** Operating frequency range is 0 – 8 MHz.

If we are using a crystal of 8 MHz = 8 x 10^6 Hertz = 8 Million cycles, then AVR can execute 8 million instructions.

The **AT** refers to Atmel the manufacturer, **Mega** means that the microcontroller belong to Mega AVR category, **16** signifies the memory of the controller, which is 16KB.

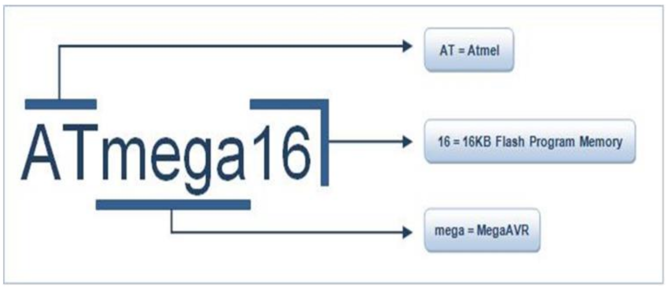


Fig. 4: Naming Convention Of AVR Microcontroller Architecture Diagram: Atmega16

Following points explain the building blocks of **Atmega16 architecture**:

- **I/O Ports:** Atmega16 has four (PORTA, PORTB, PORTC and PORTD) **8-bit** input-output ports.
- **Internal Calibrated Oscillator:** Atmega16 is equipped with an internal oscillator for driving its clock. By default Atmega16 is set to operate at internal calibrated oscillator of 1 MHz. The maximum frequency of internal oscillator is 8Mhz. Alternatively, ATmega16 can be operated using an external crystal oscillator with a maximum frequency of 16MHz. In this case you need to modify the fuse bits. (Fuse Bits will be explained in a separate tutorial).

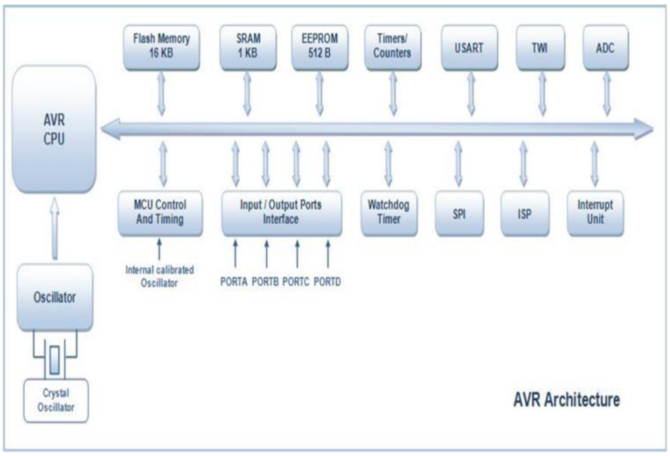


Fig. 5: Block Diagram Explaining AVR Architecture

- **ADC Interface:** Atmega16 is equipped with an 8 channel ADC\ (**Analog to Digital Converter**) with a resolution of 10-bits. ADC reads the analog input for e.g., a sensor input and converts it into digital information which is understandable by the microcontroller.
- **Timers/Counters:** Atmega16 consists of two 8-bit and one 16-bit timer/counter. Timers are useful for generating precision actions for e.g., creating time delays between two operations.
- **Watchdog Timer:** Watchdog timer is present with internal oscillator. Watchdog timer continuously monitors and resets the controller if the code gets stuck at any execution action for more than a defined time interval.

- **Interrupts:** Atmega16 consists of 21 interrupt sources out of which four are external. The remaining are internal interrupts which support the peripherals like USART, ADC, Timers etc.
- **USART: Universal Synchronous and Asynchronous Receiver and Transmitter** interface is available for interfacing with external device capable of communicating serially (data transmission bit by bit).

Architecture Continued

- **General Purpose Registers:** Atmega16 is equipped with 32 general purpose registers which are coupled directly with the Arithmetic Logical Unit (ALU) of CPU.
- **Memory:** Atmega16 consist of three different memory sections:

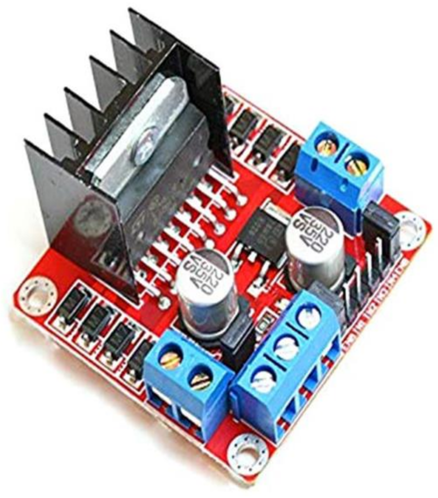
1. **Flash EEPROM**: Flash EEPROM or simple flash memory is used to store the program dumped or burnt by the user on to the microcontroller. It can be easily erased electrically as a single unit. Flash memory is non-volatile i.e., it retains the program even if the power is cut-off. Atmega16 is available with 16KB of in system programmable Flash EEPROM.
2. **Byte Addressable EEPROM**: This is also a nonvolatile memory used to store data like values of certain variables. Atmega16 has 512 bytes of EEPROM, this memory can be useful for storing the lock code if we are designing an application like electronic door lock.
3. **SRAM**: Static Random Access Memory, this is the volatile memory of microcontroller i.e., data is lost as soon as power is turned off. Atmega16 is equipped with 1KB of internal SRAM. A small portion of SRAM is set aside for general purpose registers used by CPU and some for the peripheral subsystems of the microcontroller.

- **ISP:** AVR family of controllers have **In System Programmable** Flash Memory which can be programmed without removing the IC from the circuit, ISP allows to reprogram the controller while it is in the application circuit.
- **SPI: Serial Peripheral Interface**, SPI port is used for serial communication between two devices on a common clock source. The data transmission rate of SPI is more than that of USART.
- **TWI: Two Wire Interface** (TWI) can be used to set up a network of devices, many devices can be connected over TWI interface forming a network, the devices can simultaneously transmit and receive and have their own unique address.
- **DAC:** Atmega16 is also equipped with a **Digital to Analog Converter** (DAC) interface which can be used for reverse action performed by ADC. DAC can be used when there is a need of converting a digital signal to analog signal.

Mega AVR Family

Various microcontroller of Mega AVR series:
ATmega8 and Atmega32 are other members of Mega AVR series controllers. They are quite similar to ATmega16 in architecture. Low power version Mega AVR controllers are also available in markets. The following table shows the comparison between different members of Mega AVR family:

5.2 L298N MOTOR DRIVER MODULE



L298N Motor Driver Module
is a high power motor driver module for driving DC and Stepper Motors. This module consists of an L298 motor driver IC and a 78M05 5V regulator. **L298N Module** can control up to 4 DC motors, or 2 DC motors with directional and speed control.

5.2.1 L298N Module Pin Configuration:

Pin Name	Description
IN1 & IN2	Motor A input pins. Used to control the spinning direction of Motor A
IN3 & IN4	Motor B input pins. Used to control the spinning direction of Motor B
ENA	Enables PWM signal for Motor A
ENB	Enables PWM signal for Motor B
OUT1 & OUT2	Output pins of Motor A
OUT3 & OUT4	Output pins of Motor B
12V	12V input from DC power Source
5V	Supplies power for the switching logic circuitry inside L298N IC
GND	Ground pin

Table 1: Table for driver pin connections

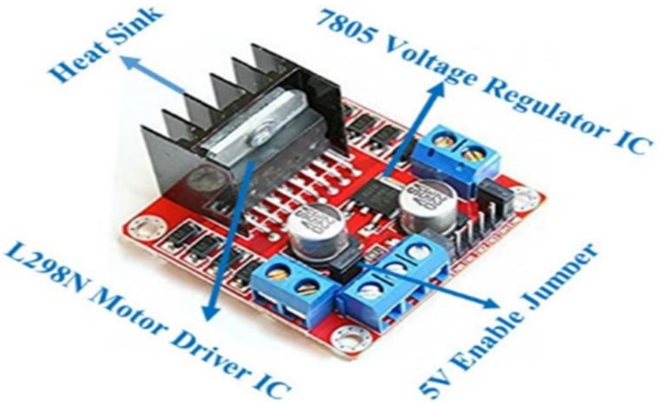
5.2.2 L298 Module Features & Specifications:

- Driver Model: L298N 2A
- Driver Chip: Double H Bridge L298N
- Motor Supply Voltage (Maximum): 46V
- Motor Supply Current (Maximum): 2A
- Logic Voltage: 5V
- Driver Voltage: 5-35V
- Driver Current:2A
- Logical Current:0-36mA
- Maximum Power (W): 25W
- Current Sense for each motor
- Heatsink for better performance
- Power-On LED indicator

Alternate Driver Modules: TMC2209, DRV8825, A4988, L9110S, DRV8711

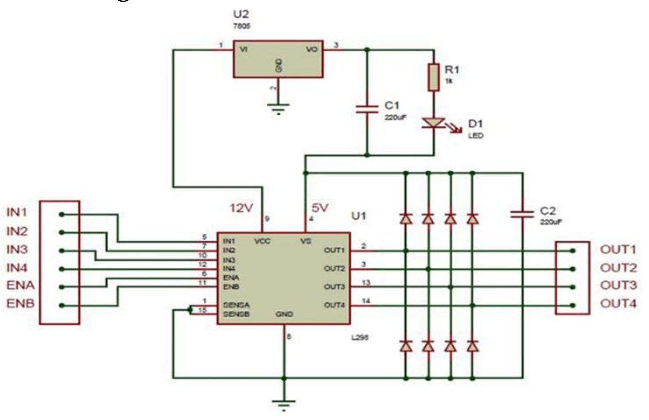
Related Components: LM298 Motor Driver IC, 78M05 Voltage Regulator, Capacitors, Resistors, Heat Sink

Brief about L298N Module
The L298N Motor Driver module consists of an L298 Motor Driver IC, 78M05 Voltage Regulator, resistors, capacitor, Power LED, 5V jumper in an integrated circuit.



78M05 Voltage regulator will be enabled only when the jumper is placed. When the power supply is less than or equal to 12V, then the internal circuitry will be powered by the voltage regulator and the 5V pin can be used as an output pin to power the microcontroller. The jumper should not be placed when the power supply is greater than 12V and separate 5V should be given through 5V terminal to power the internal circuitry.

ENA & ENB pins are speed control pins for Motor A and Motor B while IN1& IN2 and IN3 & IN4 are direction control pins for Motor A and Motor B.
Internal circuit diagram of L298N Motor Driver module is given below:

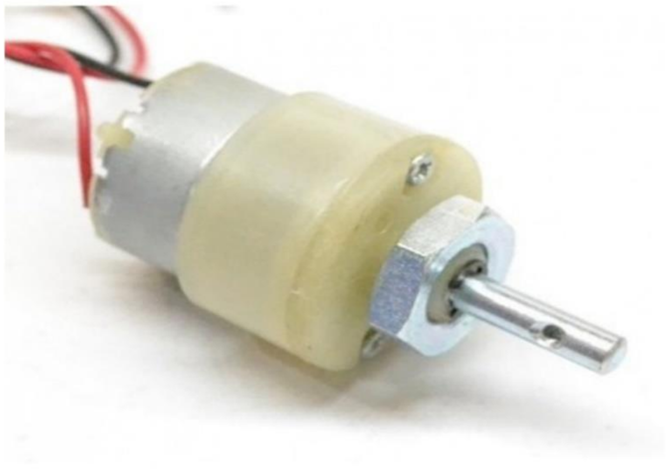


Applications of L298N Module

- Drive DC motors.
- Drive stepping motors
- In Robotics

5.3 1000 RPM MOTOR

DC Motor – 1000RPM – 12Volts geared motors are generally a simple DC motor with a gearbox attached to it. This can be used in all-terrain robots and variety of robotic applications. These motors have a 3 mm threaded drill hole in the middle of the shaft thus making it simple to connect it to the wheels or any other mechanical assembly.



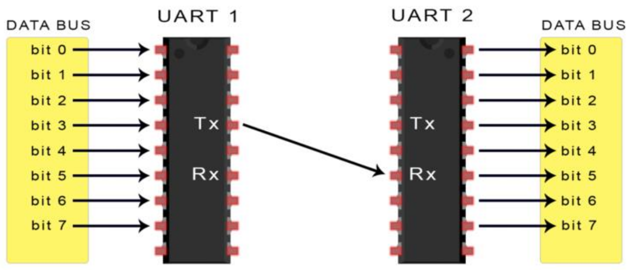
1000 RPM 12V DC geared motors widely use for robotics applications. Very easy to use and available in standard size. Also, you don't have to spend a lot of money to control motors with an Arduino or compatible board. The most popular L298N H-bridge module with onboard voltage regulator motor driver can be used with this motor that has a voltage of between 5 and 35V DC or you can choose the most precise motor diver module from the wide range available in our Motor divers category as per your specific requirements.

Nut and threads on the shaft to easily connect and internally threaded shaft for easily connecting it to the wheel. DC Geared motors with robust metal gearbox for heavy-duty applications, available in the wide RPM range and ideally suited for robotics and industrial applications. Very easy to use and available in standard size. Nut and threads on the shaft to easily connect and internally threaded shaft for easily connecting it to the wheel.

Specifications and Features: -

- RPM: 1000.
- Operating Voltage: 12V DC
- Gearbox: Attached Plastic (spur)Gearbox
- Shaft diameter: 6mm with internal hole
- Torque: 0.5 kg-cm
- No-load current = 60 mA(Max)
- Load current = 300 mA(Max).

5.4 BASICS OF UART COMMUNICATION

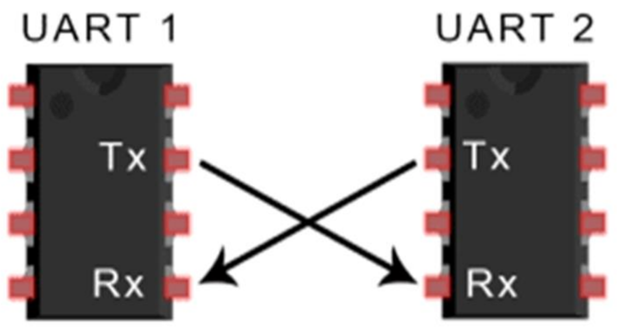


UART stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data.

One of the best things about UART is that it only uses two wires to transmit data between devices. The principles behind UART are easy to understand.

5.4.1 INTRODUCTION TO UART COMMUNICATION

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:



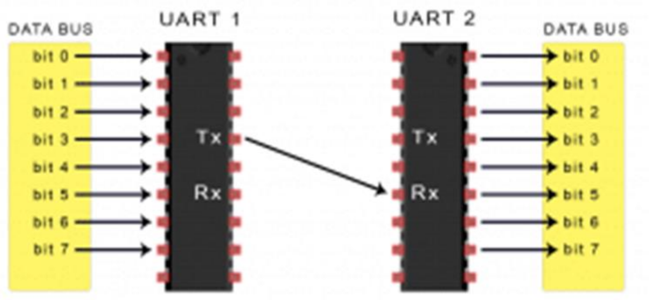
UARTs transmit data *asynchronously*, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate.* Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

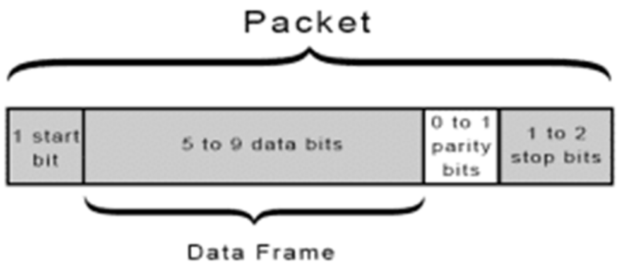
Both UARTs must also must be configured to transmit and receive the same data packet structure.

HOW UART WORKS

The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end:



UART transmitted data is organized into *packets*. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional *parity* bit, and 1 or 2 stop bits:



START BIT

The UART data transmission line is normally held at a high voltage level when it’s not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

DATA FRAME

The data frame contains the actual data being transferred. It can be 5 bits up to 8 bits long if a parity bit is used. If no parity bit is used, the data frame can be 9 bits long. In most cases, the data is sent with the least significant bit first.

PARITY

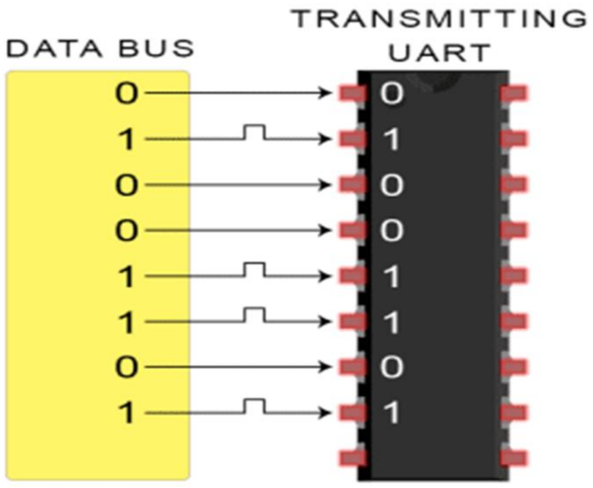
Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long distance data transfers. After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 bits in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bits in the data frame should total to an odd number. When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd; or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

STOP BITS

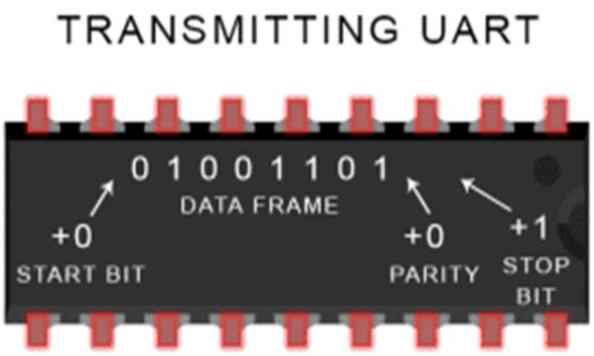
To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two bit durations.

STEPS OF UART TRANSMISSION

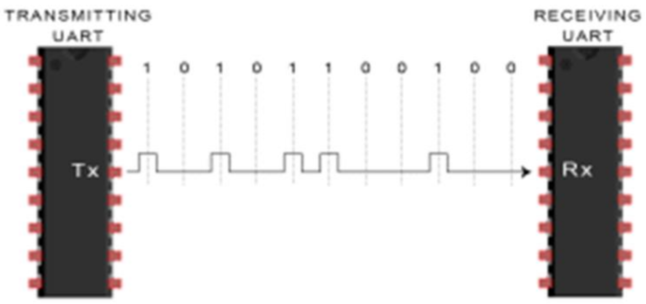
1. The transmitting UART receives data in parallel from the data bus:



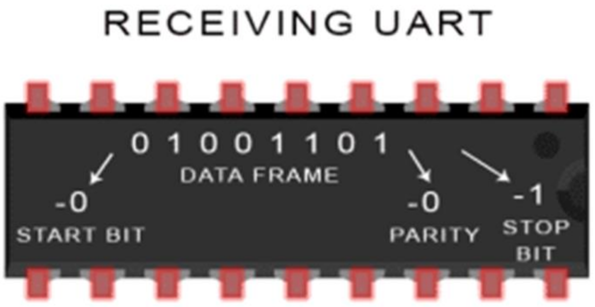
2. The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame:



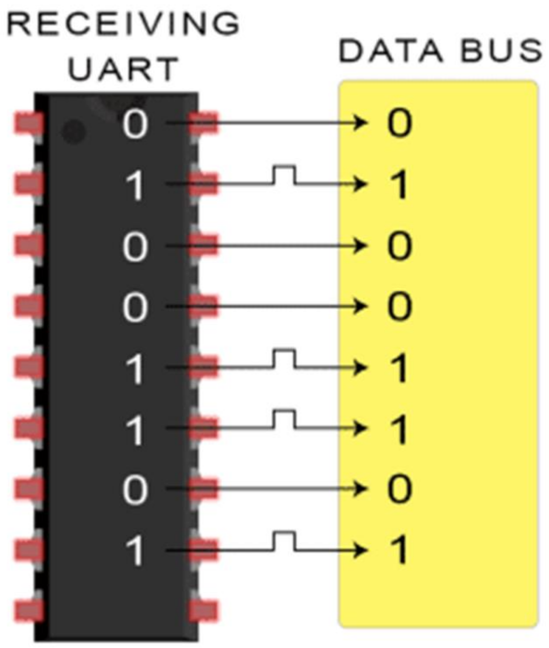
3. The entire packet is sent serially from the transmitting UART to the receiving UART. The receiving UART samples the data line at the pre-configured baud rate:



4. The receiving UART discards the start bit, parity bit, and stop bit from the data frame:



5. The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end:



ADVANTAGES AND DISADVANTAGES OF UARTS

No communication protocol is perfect, but UARTs are pretty good at what they do. Here are some pros and cons to help you decide whether or not they fit the needs of your project:

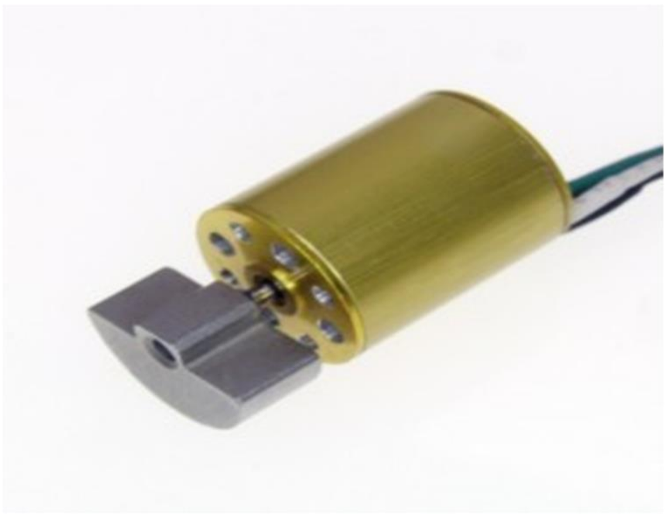
ADVANTAGES

- Only uses two wires
- No clock signal is necessary
- Has a parity bit to allow for error checking
- The structure of the data packet can be changed as long as both sides are set up for it
- Well documented and widely used method

DISADVANTAGES

- The size of the data frame is limited to a maximum of 9 bits
- Doesn't support multiple slave or multiple master systems
- The baud rates of each UART must be within 10% of each other

5.5 VIBRATION MOTORS



Vibration Motor

There are two basic types of vibration motor. An **eccentric rotating mass vibration motor (ERM)** uses a small unbalanced mass on a DC motor when it rotates it creates a force that translates to vibrations. **A linear resonant actuator (LRA)** contains a small internal mass attached to a spring, which creates a force when driven.

An Introduction to Vibration Motors

Small vibration motors have been around since the 1960s. Initially, they were developed for massaging products, but their development took a new turn in the 1990s when consumers required vibra-call on their mobile / cell phones. Today, designers and users alike have learned from two decades of mobile phones, that vibration alerting is an excellent way to alert operators to an event.

These days miniature vibrating motors are used in a wide range of products, such as tools, scanners, medical instruments, GPS trackers, and control sticks. Vibrator motors are also the main actuators for haptic feedback which is an inexpensive way to increase a product's value and differentiate it from the competition.

5.5.1 Different Vibration Motor Form Factors

Whilst the end goal of vibration motors is to produce a force, there are many ways of achieving it. There is a wide range of physical forms, both internal and external, to help you achieve this. To guide you, the articles below help describe the main characteristics of each type. You will notice that some motors may belong to multiple categories, for example, our encapsulated motors are based on ERMs, so you may want to look at several different types for additional information.

The most popular form factor is the ERM or 'pager' motor. This is because there are lots of DC motors available in this cylindrical form, where the eccentric mass helps create an unbalanced force. They are also the most versatile - they can be mounted on PCBs, encapsulated, use a variety of power connections, and even be based on brushless motors. However, coin or 'pancake' motors use the same operating principle - but their eccentric mass is kept in their small circular body (which is where they get their names from). They are restricted in amplitude because of their size but have extremely low profiles (only a few mm!) which make them popular in applications which space is restricted.

Encapsulation is the process of sealing an ERM into a plastic housing. These units are popular for applications where the motor is housed with injection moulding or ones that require waterproof vibration motors. Motors that are mounted to PCBs have several connection types, including SMD vibration motors, through-hole chassis, or spring pad terminals. Some even use leaded power connections and a specialised mounting technique (like adhesives) so that PCB tracks do not need to reach the motor itself.

Linear Resonant Actuators (LRA) are the most unique devices in our collection of vibration motors. Although they sometimes look like coin motors, they do not use an eccentric mass to create force. Instead, they have a magnetic mass attached to a spring and driven by a voice coil - a similar design to loudspeakers. This means they are very efficient, have excellent response, and need different drivers compared to a DC vibration motor. Similarly, brushless vibration motors also have different driving requirements. Without wearing the brushes of the commutator, they last for a very long time but (unless the vibration motor has an internal driver IC) are slightly more complicated to operate.

Common Vibration Motor Applications And Examples

Haptic Feedback and Vibration Alerting are not necessarily applications themselves, but they are different methods of implementing vibrations within applications. Haptic Feedback uses advanced vibration patterns and effects to convey complicated information to users. Vibration Alerting tends to be a simple on/off alert, perhaps with a ramp effect. A good example is a mobile phone, which in the early days would simply vibrate to notify the user of a call/text. Later, they would play SMS in Morse code, now they have a range of effects for games and applications. Over time, they have progressed from Vibration Alerting to Haptic Feedback.

Vibration motors are a popular method for moving materials and products that can become stuck in chutes or hoppers without the need for human intervention. These applications typically employ some of our larger motors, however, emulsifiers or scientific experiments sometimes make use of smaller vibration motors that meet their specific requirements. Conversely, medical applications will often use smaller devices as within handheld equipment.

Light, portable devices that need to alert users are perfect for miniature vibration motors - especially when they are powered by batteries. Cars are using more and more sensors to assist drivers, unfortunately, these are accompanied by annoying beeps and tones. Vibration motors help give silent information to the driver or augment the audio only warnings.

5.6 A/D CONVERTERS (ADC)

Analog to Digital Converters (ADC) translate analog electrical signals for data processing purposes. With products matching performance, power, cost, and size needs, Analog Devices offers the industry's largest A/D converter portfolio. As the world's leading provider, these data converters enable accurate and reliable conversion performance in a range of applications such as communications, energy, healthcare, instrumentation and measurement, motor and power control, industrial automation, and aerospace/defense. A variety of A/D converter resources are provided to assist the engineer in every project phase, from product selection to circuit design.

ADC Stands for "Analog-to-Digital Converter." Since computers only process digital information, they require digital input. Therefore, if an analog input is sent to a computer, an analog-to-digital converter (ADC) is required. This device can take an analog signal, such as an electrical current, and digitize it into a binary format that the computer can understand.

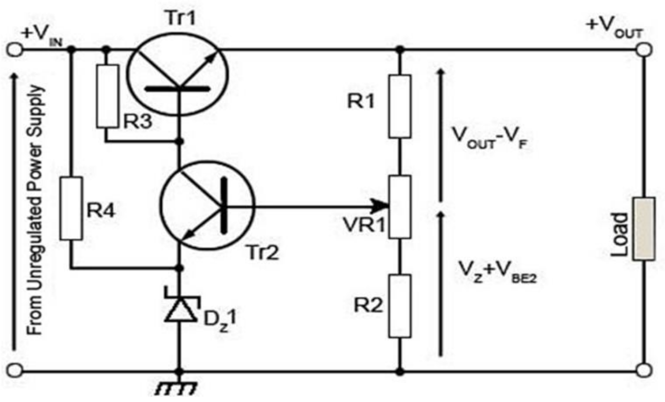
A common use for an ADC is to convert analog video to a digital format. For example, video recorded on 8mm film or a VHS tape is stored in an analog format. In order to transfer the video to a computer, the video must be converted to a digital format. This can be done using an ADC video conversion box, which typically has composite video inputs and a Firewire output. Some digital camcorders that have analog inputs can also be used to convert video from analog to digital.

ADCs may also be used to convert analog audio streams. For example, if you want to record sounds from a microphone, the audio must be converted from the microphone's analog signal into a digital signal that the computer can understand. This is why all sound cards that have an analog audio input also require an ADC that converts the incoming audio signal to a digital format. The accuracy of the audio conversion depends on the sampling rate used in the conversion process. Higher sampling rates provide a better estimation of the analog signal, and therefore produce a higher-quality sound.

While ADCs convert analog inputs into a digital format that computers can recognize, sometimes a computer must output an analog signal. For this type of conversion, a digital-to-analog converter (DAC) is used.

5.7 POWER SUPPLY

The DC power supplies used for electronics and computers are of two main types, switching and linear, both supplied from the 120V AC mains. Switching supplies include a regulator, and are a special study, so they will not be mentioned further here. The linear supply is relatively simple and inexpensive, and in many cases is completely adequate. For some applications, a regulator is not required, either because the load is constant or because small variations in the output voltage do not matter. The basic ideas of linear power supply design will be presented here, and you should make a small supply just for the experience, if you have never done so before. A linear supply consists of transformer, rectifier, surge-limiting resistor, filter capacitor, and bleeder. The transformer should have separate primary and secondary windings so that the output is isolated from the power-line ground. It is very dangerous otherwise, so isolation is important.

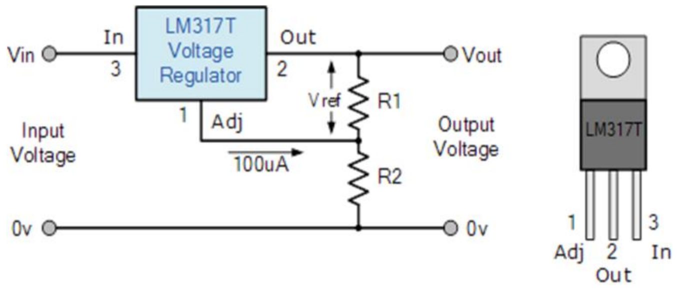


A transformer cannot usually take a DC current through a winding, but if a winding is center-tapped, equal DC currents can flow from the center tap to the ends of the windings without inconvenience. Small currents can, of course, be tolerated. The rectifier is usually a full-wave bridge of silicon diodes, though two diodes at the ends of a center-tapped winding can also be used. The capacitor is a large aluminum electrolytic, up to 10 000 μF, and a voltage rating of 450 V. For higher voltages, capacitors may be used in series (halving the equivalent capacitance but doubling the voltage rating); the leakage in the capacitors will equalize the voltages. A resistor is used in series with the capacitor to limit the surge of current in the first half-cycle of operation, when the capacitor is uncharged. To handle the sharp power pulse, a wire-wound power resistor is required, though after the turn-on surge, it hardly dissipates any power at all. The bleeder resistance is connected across the capacitor to discharge it when the supply is turned off. It should carry up to 10% of the rated output current, and helps to stabilize the operation of the supply, eliminating any rises in voltage at very low currents that may occur. For high voltage (> 50V) supplies, a bleeder resistor is essential to remove the hazard of an unexpected voltage in the filter capacitor when the supply is turned off. The bleeder resistor should be rated to dissipate the necessary power in steady operation. The AC input should be fused, using a slow-blow fuse of about two or three times the normal input current. The purpose of this fuse is to save the power transformer if a filter capacitor fails, usually by becoming a short circuit, or if the rectifier fails, also usually by presenting a short circuit. This can happen even if the output is protected against a short by a voltage regulator. Usually, insufficient current flows in this case to open the protective device of the line, which may be 15 A or more, and is designed to prevent fire. The current may be more than sufficient to burn out the transformer primary, if it is rated for an ampere or less. The transformer is usually the most expensive part of the power supply, and is worth saving. If a regulator is used, the difference in voltages will cause a power dissipation that is greater, the greater the voltage difference. At maximum load, the output voltage need only be enough greater than the output voltage to operate the regulator. The series of 1A diodes, 1N4001 to 1N4007, have peak inverse voltage ratings from 50 V to 1200 V, and easily handle most modest rectifying tasks. Their peak surge current is 25 A, and you should analyze the circuit and choose the surge-limiting resistor accordingly. The capacitor is sized on the basis of the permissible ripple in the output. To estimate the ripple, consider that the capacitor supplies the maximum output current I continuously, and is "topped up" to the output voltage every 1/120 s for a full-wave rectifier, and every 1/60 s for a half-wave. The charge drawn by the load is then I/120 C (full-wave) and this equals C V, where ?V is the amplitude of the ripple. Therefore, C = I/120V.

5.7.1 VOLTAGE REGULATOR

Voltage regulators are widely used in electronics power supply circuits. They provide very high degrees of regulation and low levels of ripple, although their levels of efficiency are much lower than another popular form of regulator called the switch mode regulator. However linear regulators are still used in large quantities because of their relative simplicity and high levels of performance.

It is possible to make voltage regulator circuits from both discrete components as well as being able to use IC regulators. The IC regulators enable very high levels of performance to be achieved, often using comparatively few components, but often for many projects it is possible to use a few available components to make a perfectly adequate voltage regulator circuit.



Basic concept behind voltage regulator circuits

Although there are many different voltage regulator circuits and integrated circuit regulators, the basic concepts for these circuits fall into one of two basic categories:

- Series regulator circuit
- Parallel or shunt regulator circuit.

All voltage regulator circuits fall into one of these categories, although of the two, the most common type seen for full voltage regulator circuits is the series regulator.

VI. SYSTEM SPECIFICATIONS

6.1 Hardware Specification

Processor	:	INTEL i3(7th generation)
RAM	:	4 GB RAM
Hard disk	:	1TB
Monitor	:	20' color monitor

- AVR microcontroller
- L298 Motor driver
- 1000 rpm motor
- Power supply
- UART
- PC
- ADC
- Vibration motor

6.2 Software Specification

Front end	: GUI
Back end	: python
Software	: thonny
Platform	: Windows 8
Software for hardware	: AVR studio

6.3 Convolution neural network

- A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and is used mainly for image processing, classification, segmentation and also for other auto correlated data.
- A convolution is essentially sliding a filter over the input.
- Rather than looking at an entire image at once to find certain features it can be more effective to look at smaller portions of the image.

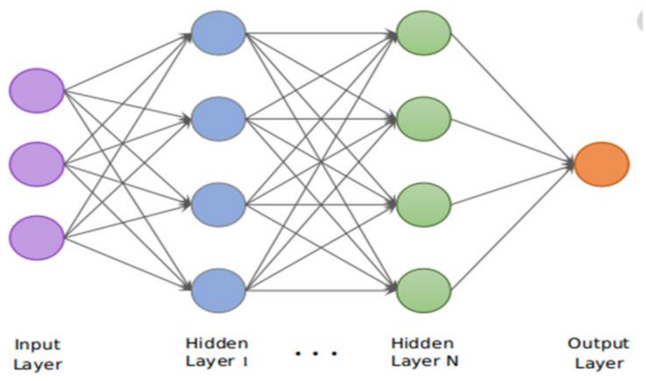


Fig no: convolution layers

1. **Input Layers:** It's the layer in which we give input to our model. The number of neurons in this layer is equal to total number of features in our data (number of pixels incase of an image).
2. **Hidden Layer:** The input from Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layers can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by addition of learnable biases followed by activation function which makes the network nonlinear.
3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or soft max which converts the output of each class into probability score of each class.

- The data is then fed into the model and output from each layer is obtained this step is called feed forward, we then calculate the error using an error function, some common error functions are cross entropy, square loss error etc. After that, we back propagate into the model by calculating the derivatives. This step is called Back propagation which basically is used to minimize the loss.
- Here's the basic python code for a neural network with random inputs and two hidden layers.

6.4 AVR STUDIO

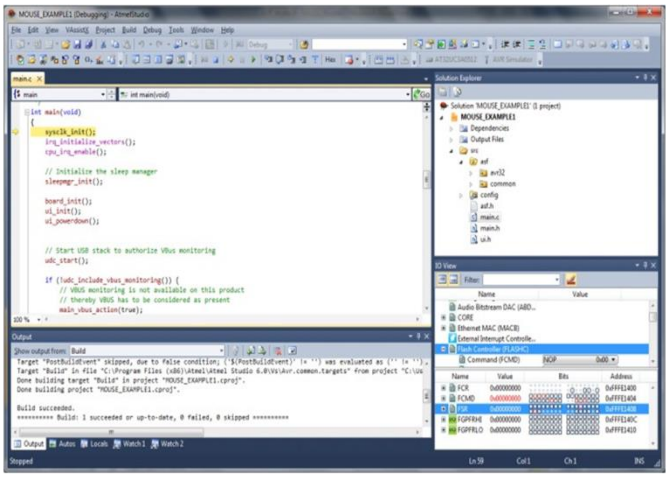
Introduction

AVR Studio is a Development Tool for the AT90S Series of AVR microcontrollers. This manual describes the how to install and use AVR Studio. AVR Studio enables the user to fully control execution of programs on the AT90S In-Circuit Emulator or on the built-in AVR Instruction Set Simulator. AVR Studio supports source level execution of Assembly programs assembled with the Atmel Corporation's AVR Assembler and C programs compiled with IAR Systems' ICCA90 C Compiler for the AVR microcontrollers. AVR Studio runs under Microsoft Windows95 and Microsoft Windows NT

Description

This section gives a brief description of the main features of AVR Studio. AVR Studio enables execution of AVR programs on an AVR In-Circuit Emulator or the built-in AVR Instruction Set Simulator. In order to execute a program using AVR Studio, it must first be compiled with IAR Systems' C

Compiler or assembled with Atmel's AVR Assembler to generate an object file which can be read by AVR Studio.



```
int main(void)
{
	sysclk_init();
	irq_initialize_vectors();
	cpu_irq_enable();

	// Initialize the sleep manager
	sleepmgr_init();

	board_init();
	ui_init();
	ui_powerdown();


	// Start USB stack to authorize VBus monitoring
	udc_start();

	if (!udc_include_vbus_monitoring()) {
		// VBUS monitoring is not available on this product
		// thereby VBUS has to be considered as present
		main_vbus_action(true);
```

```
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Atmel Studio 6.0\Vs\Avr.common.targets" from project "C:\Us
Done building target "Build" in project "MOUSE_EXAMPLE1.cproj".
Done building project "MOUSE_EXAMPLE1.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Name	Value	Address
FCR	0x00000000	0xFFFE1400
FCMD	0x00000000	0xFFFE1404
FSR	0x00000000	0xFFFE1408
FGPFRHI	0x00000000	0xFFFE140C
FGPFRLO	0x00000000	0xFFFE1410

In addition to the Source window, AVR Studio defines a number of other windows which can be used for inspecting the different resources on the microcontroller. The key window in AVR Studio is the Source window. When an object file is opened, the Source window is automatically created. The Source window displays the code currently being executed on the execution target (i.e. the Emulator or the Simulator), and the text marker is always placed on the next statement to be executed. The Status bar indicates whether the execution target is the AVR In-Circuit Emulator or the built-in Instruction Set Simulator. By default, it is assumed that execution is done on source level, so if source information exists, the program will start up in source level mode. In addition to source level execution of both C and Assembly programs, AVR Studio can also view and execute programs on a disassembly level. The user can toggle between source and disassembly mode when execution of the program is stopped. All necessary execution commands are available in AVR Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until that statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code breakpoints, and every breakpoint can be defined as enabled or disabled. The breakpoints are remembered between sessions. The Source window gives information about the control flow of the program. In addition, AVR Studio offers a number of other windows which enables the user to have full control of the status of every element in the execution target. The available windows are: 1. Watch window: Displays the values of defined symbols. In the Watch window, the user can watch the values of for instance variables in a C program. 2. Register window: Displays the contents of the register file. The registers can be modified when the execution is stopped.

6.5 EMBEDDED C

What is an Embedded C Programming

In every embedded system based projects, Embedded C programming plays a key role to make the microcontroller run & perform the preferred actions. At present, we normally utilize several electronic devices like mobile phones, washing machines, security systems, refrigerators, digital cameras, etc. The controlling of these embedded devices can be done with the help of an embedded C program. For example in a digital camera, if we press a camera button to capture a photo then the microcontroller will execute the required function to click the image as well as to store it.

Embedded C Programming

Embedded C programming builds with a set of functions where every function is a set of statements that are utilized to execute some particular tasks. Both the embedded C and C languages are the same and implemented through some fundamental elements like a variable, character set, keywords, data types, declaration of variables, expressions, statements. All these elements play a key role while writing an embedded C program.

The embedded system designers must know about the hardware architecture to write programs. These programs play a prominent role in monitoring and controlling external devices. They also directly operate and use the internal architecture of the microcontroller, such as interrupt handling, timers, serial communication, and other available features.

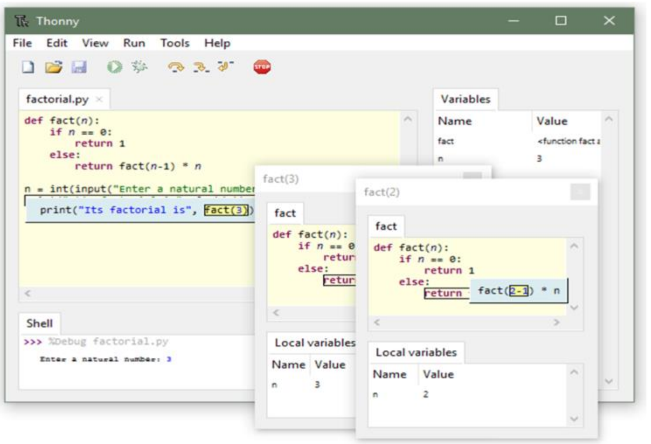
Embedded System Programming

As we discussed earlier, the designing of an embedded system can be done using Hardware & Software. For instance, in a simple embedded system, the processor is the main module that works like the heart of the system. Here a processor is nothing but a microprocessor, DSP, microcontroller, CPLD & FPGA. All these processors are programmable so that it defines the working of the device.

An Embedded system program allows the hardware to check the inputs & control outputs accordingly. In this procedure, the embedded program may have to control the internal architecture of the processor directly like Timers, Interrupt Handling, I/O Ports, serial communications interface, etc.

So embedded system programming is very important to the processor. There are different programming languages are available for embedded systems such as C, C++, assembly language, JAVA, JAVA script, visual basic, etc. So this programming language plays a key role while making an embedded system but choosing the language is very essential.

6.6 THONNY SOFTWARE



```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(n-1) * n

n = int(input("Enter a natural number
    print("Its factorial is", fact(3))
```

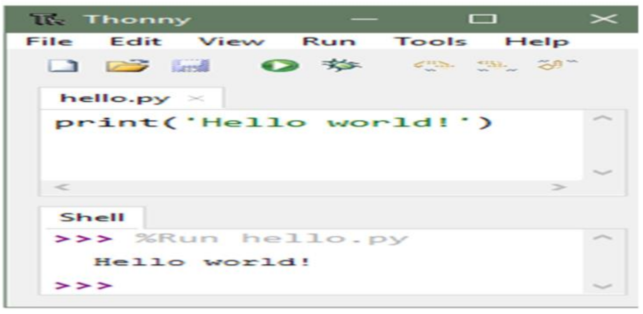
```
>>> %Debug factorial.py
  Enter a natural number: 3
```

Name	Value
fact	<function fact a
n	3

Fig no: Thonny window page

Features

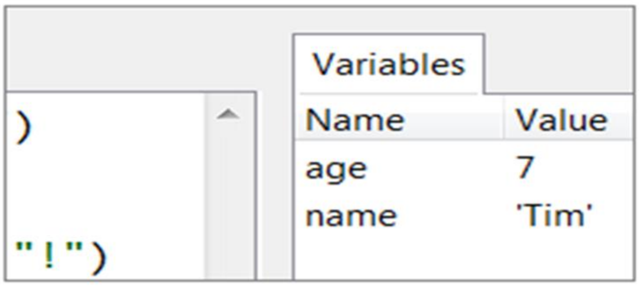
Easy to get started. Thonny comes with Python 3.7 built in, so just one simple installer is needed and you're ready to learn programming. (You can also use a separate Python installation, if necessary.) The initial user interface is stripped of all features that may distract beginners.



```
print('Hello world!')
```

```
>>> %Run hello.py
  Hello world!
>>>
```

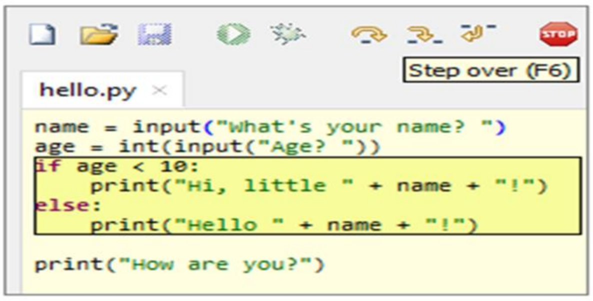
No-hassle variables. Once you're done with hello-worlds, select *View* → *Variables* and see how your programs and shell commands affect Python variables.



Name	Value
age	7
name	'Tim'

Fig no: no-hassle variables

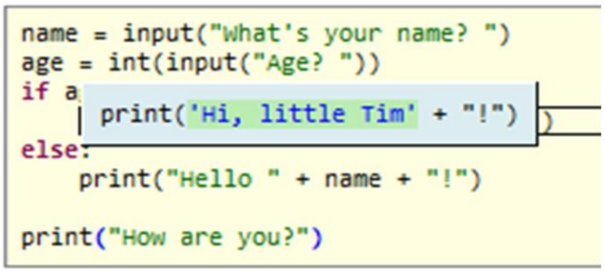
Simple debugger. Just press Ctrl+F5 instead of F5 and you can run your programs step-by-step, no breakpoints needed. Press F6 for a big step and F7 for a small step. Steps follow program structure, not just code lines.



```
name = input("What's your name? ")
age = int(input("Age? "))
if age < 10:
    print("Hi, little " + name + "!")
else:
    print("Hello " + name + "!")

print("How are you?")
```

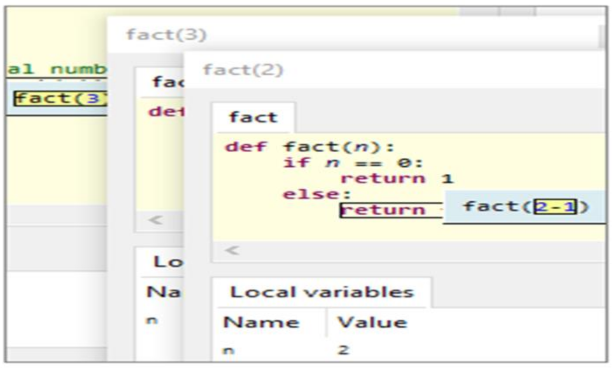
Fig no: simple debugger

Step through expression evaluation. If you use small steps, then you can even see how Python evaluates your expressions. You can think of this light-blue box as a piece of paper where Python replaces sub expressions with their values, piece-by-piece.



```
name = input("What's your name? ")
age = int(input("Age? "))
if a
    print('Hi, little Tim' + "!")  )
else:
    print("Hello " + name + "!")

print("How are you?")
```

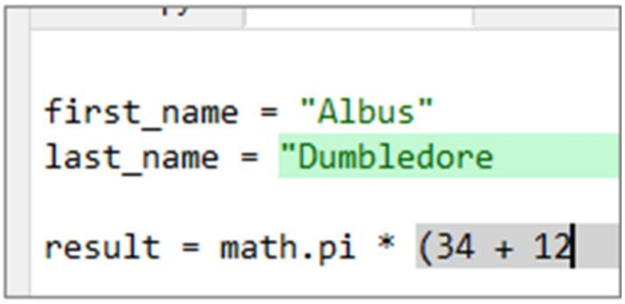
Faithful representation of function calls. Stepping into a function call opens a new window with separate local variables table and code pointer. Good understanding of how function calls work is especially important for understanding recursion.



```
def fact(n):
    if n == 0:
        return 1
    else:
        return   fact(2-1)
```

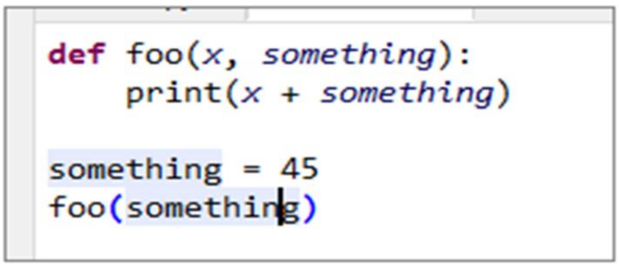
Name	Value
n	2

Highlights syntax errors. Unclosed quotes and parentheses are the most common beginners' syntax errors. Thonny's editor makes these easy to spot



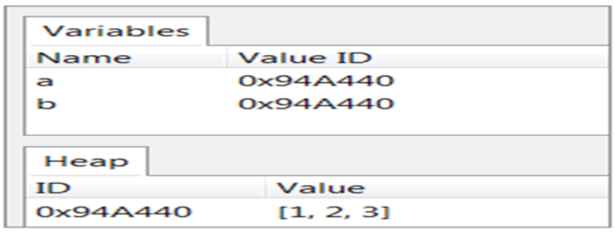
```
first_name = "Albus"
last_name = "Dumbledore

result = math.pi * (34 + 12
```

Explains scopes. Highlighting variable occurrences reminds you that the same name doesn't always mean the same variable and helps spotting typos. Local variables are visually distinguished from globals.



```
def foo(x, something):
    print(x + something)

something = 45
foo(something)
```

Mode for explaining references. Variables are initially presented according to simplified model (name → value) but you can switch to more realistic model (name → address/id → value).



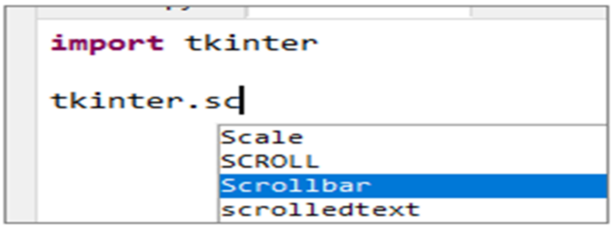
Variables

Name	Value ID
a	0x94A440
b	0x94A440

Heap

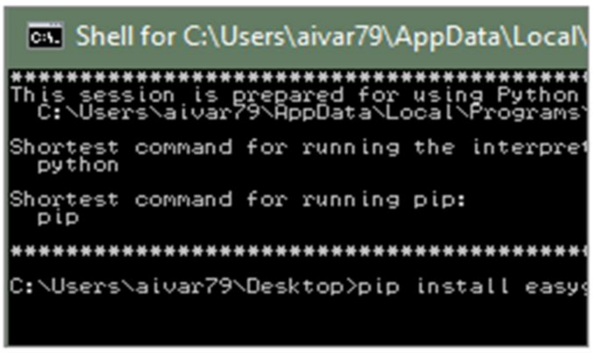
ID	Value
0x94A440	[1, 2, 3]

Code completion. Students can explore APIs with the help of code completion.



```
import tkinter

tkinter.sd
          Scale
          SCROLL
          Scrollbar
          scrolledtext
```

Beginner friendly system shell. Select *Tools → Open system shell* to install extra packages or learn handling Python on command line. PATH and conflicts with other Python interpreters are taken care of by Thonny.



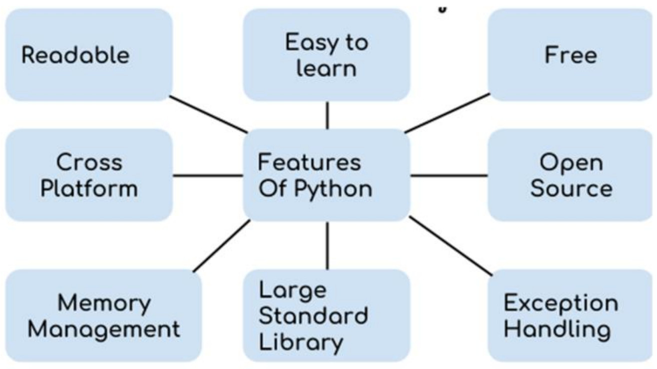
```
Shell for C:\Users\aivar79\AppData\Local\
******************************************
This session is prepared for using Python
  C:\Users\aivar79\AppData\Local\Programs\

Shortest command for running the interpret
  python

Shortest command for running pip:
  pip
******************************************
C:\Users\aivar79\Desktop>pip install easy
```

6.7 PYTHON PROGRAM

- Python is a dynamic, interpreted (byte code-compiled) language. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible, and you lose the compile-time type checking of the source code. Python tracks the types of all values at runtime and flags code that does not make sense as it runs.
- An excellent way to see how Python code works is to run the Python interpreter and type code right into it. If you ever have a question like, "What happens if I add an int to a list?" Just typing it into the Python interpreter is a fast and likely the best way to see what happens.

As you can see above, it's easy to experiment with variables and operators. Also, the interpreter throws, or "raises" in Python parlance, a runtime error if the code tries to read a variable that has not been assigned a value. Like C++ and Java, Python is case sensitive so "a" and "A" are different variables. The end of a line marks the end of a statement, so unlike C++ and Java, Python does not require a semicolon at the end of each statement. Comments begin with a '#' and extend to the end of the line.

6.7.1 Features of Python programming language



1. **Readable:** Python is a very readable language.
2. **Easy to Learn:** Learning python is easy as this is a expressive and high level programming language, which means it is easy to understand the language and thus easy to learn.
3. **Cross platform:** Python is available and can run on various operating systems such as Mac, Windows, Linux, Unix etc. This makes it a cross platform and portable language.
4. **Open Source:** Python is a open source programming language.
5. **Large standard library:** Python comes with a large standard library that has some handy codes and functions which we can use while writing code in Python.
6. **Free:** Python is free to download and use. This means you can download it for free and use it in your application. See: Open Source Python License. Python is an example of a FLOSS (Free/Libre Open Source Software), which means you can freely distribute copies of this software, read its source code and modify it.
7. **Supports exception handling:** If you are new, you may wonder what is an exception? An exception is an event that can occur during program

exception and can disrupt the normal flow of program. Python supports exception handling which means we can write less error prone code and can test various scenarios that can cause an exception later on.

8. **Advanced features:** Supports generators and list comprehensions. We will cover these features later.
9. **Automatic memory management:** Python supports automatic memory management which means the memory is cleared and freed automatically. You do not have to bother clearing the memory.

VII. CONCLUSION

This project proposes a CNN-based multi-signal fault diagnosis framework with its application on motors, where time-frequency distributions of sensor signals are utilized as the input images. CNN is utilized as the building block for multi-signal model, which shows the capability to learn discriminative features automatically from TFD images and produce accurate fault classification. Both vibration and voltage signals are used to verify the performance of the presented framework. The CNN-based multi-signal architectures are designed and their performances are explored through experiments, where merged model has achieved the best performance. Furthermore, the influence of measurement uncertainty is discussed based on statistical analysis. Future study can be conducted aiming at improving performance of deep convolutional neural network by tuning hyper-parameters and network architecture. To overcome the problem of limited training data for a deep architecture, data augmentation can be performed to enlarge the dataset. In addition, existing pre-trained models for fault diagnosis can be further investigated.
“After Completing this project we have attained all PO’s”

VIII. REFERENCES

[1]. E. Elbouchikhi, V. Choqueuse, F. Auger, and M. E. H. Benbouzid, “Motor current signal analysis based on a matched subspace detector”, IEEE Transactions on Instrumentation and Measurement, vol. 66, no. 12, pp. 3260–3270, Dec. 2017.

[2]. T. Yang, H. Pen, Z. Wang, and C. Chang, “Feature knowledge based fault detection of induction motors through the analysis of stator current data”, IEEE Transactions on Instrumentation and Measurement, vol. 65, no. 3, pp. 549–558, Mar. 2016.

[3]. F. B. Batista, P. C. M. Lamim Filho, R. Pederiva, and V. A. D. Silva, “An empirical demodulation for electrical fault detection in induction motors”, IEEE Transactions on Instrumentation and Measurement, vol. 65, no. 3, pp. 559–569, Mar. 2016.

[4]. D. Z. Li, W. Wang, and F. Ismail, “An enhanced bispectrum technique with auxiliary frequency injection for induction motor health condition monitoring”, IEEE Transactions on Instrumentation and Measurement, vol. 64, no. 10, pp. 2679–2687, Oct. 2015.

[5]. R. Yan, X. Chen, and S. C. Mukhopadhyay, Structural Health Monitoring: An Advanced Signal Processing Perspective. Springer, Apr. 2017.

[6]. W. Sun, R. Zhao, R. Yan, S. Shao, and X. Chen, “Convolutional discriminative feature learning for induction motor fault diagnosis”, IEEE Transactions on Industrial Informatics, vol. 13, no. 3, pp. 1350- 1359, Jun. 2017.

[7]. O. E. Hassan, M. Amer, A. K. Abdelsalam, and B. W. Williams, “Induction motor broken rotor bar fault detection techniques based on fault signature analysis–a review”, IET Electric Power Applications, vol. 12, no. 7, pp. 895-907, Aug. 2018.

[8]. Y. Tian, D. Guo, K. Zhang, L. Jia, H. Qiao, and H. Tang, “A review of fault diagnosis for traction induction motor”, in 2018 37th Chinese Control Conference (CCC). IEEE, pp. 5763–5768, Jul. 2018.

[9]. A. M. Da Silva, R. J. Povinelli, and N. A. Demerdash, “Induction machine broken bar and stator short-circuit fault diagnostics based on three-phase stator current envelopes”, IEEE Transactions on Industrial Electronics, vol. 55, no. 3, pp. 1310–1318, Mar. 2018.

[10]. M. A. Hmida and A. Braham, “An on-line condition monitoring system for incipient fault detection in double-cage induction motor”, IEEE Transactions on Instrumentation and Measurement, vol. 67, no. 8, pp. 1850-1858, Aug. 2018.

[11]. A. Sapena-Ba˜n´o, M. Pineda-Sanchez, R. Puche-Panadero, J. MartinezRoman, and D. Mati´c, “Fault diagnosis of rotating electrical machines in transient regime using a single stator currents fft”, IEEE Transactions on Instrumentation and Measurement, vol. 64, no. 11, pp. 3137–3146, Nov. 2016.

[12]. L. Noureddine, A. Hafaifa, and A. Kouzou, “Rotor fault diagnosis of scig-wind turbine using hilbert transform”, in 2017 9th IEEE-GCC Conference and Exhibition (GCCCE). IEEE, pp. 1–9, May. 2017.

[13]. P. A. Delgado-Arredondo, D. Morinigo-Sotelo, R. A. Osornio-Rios, J. G. Avina-Cervantes, H. Rostro-Gonzalez, and R. de Jesus RomeroTroncoso, “Methodology for fault detection in induction motors viasound and vibration signals”, Mechanical Systems and Signal Processing, vol. 83, pp. 568–589, Jan. 2017.

[14]. M. D. Prieto, G. Cirrincione, A. G. Espinosa, J. A. Ortega, and H. Henao, “Bearing fault detection by a novel condition-monitoring scheme based on statistical-time features and neural networks”, IEEE Transactions on Industrial Electronics, vol. 60, no. 8, pp. 3398–3407, Aug. 2018.

[15]. A. Glowacz, “Dc motor fault analysis with the use of acoustic signals, coiflet wavelet transform, and k-nearest neighbor classifier”, Archives of Acoustics, vol. 40, no. 3, pp. 321–327, Jun. 2016.

[16]. A. Glowacz and Z. Glowacz, “Diagnosis of stator faults of the singlephase induction motor using acoustic signals”, Applied Acoustics, vol. 117, pp. 20–27, Feb. 2017.

[17]. B. Samanta and C. Nataraj, “Use of particle swarm optimization for machinery fault detection”, Engineering Applications of Artificial Intelligence, vol. 22, no. 2, pp. 308–316, Mar. 2018.

[18]. B. Samanta, K. R. Al-Balushi, and S. A. Al-Araimi, “Artificial neural networks and genetic algorithm for bearing fault detection”, Soft Computing-A fusion of foundations, methodologies and applications, vol. 10, no. 3, pp. 264–271, Feb. 2019.

[19]. B.S. Yang, M.S. Oh, A. C. C. Tan et al., “Fault diagnosis of induction motor based on decision trees and adaptive neuro-fuzzy inference”, Expert Systems with Applications, vol. 36, no. 2, pp. 1840–1849, Mar. 2018.

[20]. G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks”, Science, vol. 313, no. 5786, pp. 504–507, Jul. 2017.