Human - Machine Communication using Natural Languages  
Tejas Shetty  
Faculty of Computer Science, B. K. Birla College of Arts, Science and Commerce, Kalyan, Maharashtra, India  

Abstract:  
Machines have been making our lives easier since pre-historic times. Our ancestors discovered and used two of the simple machines i.e the wedge (in the form of a hand axe) and an inclined plane (ramp to move heavy objects) and that began the long-term relationship between the human and the machine. As time passed, the humans became smarter and as a result, the machines were made more complex to complete harder tasks which took up a lot of time and work yet became easy to use so that the general public can easily use those machines. To make these machines easily usable, they are provided with a user interface. Command-Line Interfaces are one of the oldest user interfaces used in machines where the humans typed the commands directly into the machine or the computer. As the machines evolved, so did the user interfaces. Some other user interfaces were Graphical User interface - an easy to use interface where you just have to point and click with a pointing device, Menu-Driven interface - an easy to use interface comprised of a series of menus and sub-menus which the user accesses by pressing buttons, Form-Based interface - an interface that uses an electronic form which a user completes in order to enter data into a system and Natural Language Interface - a spoken interface where the user interacts with the computer by talking to it. The main purpose of this research paper is to create a simple Natural Language Interface and make it work with a simple machine, here Arduino based machine to produce the desired output.

Keywords: Machine, User Interface, Natural Language, Arduino, Python, Speech Recognition.

I. INTRODUCTION  
Let us first understand what a machine is. A machine is a mechanical structure that uses power to apply forces and control movement to perform an intended action. Machines can be driven by animals and people, by natural forces such as wind and water, and by chemical, thermal, or electrical power, and include a system of mechanisms that shape the actuator input to achieve a specific application of output forces and movement. They can additionally include computers and sensors that monitor performance and set up movement, typically known as mechanical systems. The first known machine was the hand axe, made by chipping flint to form a wedge, made in the pre-historic times which is not only the oldest known machine but also one of the six simple machines - Lever, Wheel and axle, Pulley, Inclined plane, Wedge and Screw. A simple machine is a mechanical device that changes the direction or magnitude of a force. That is, a simple machine uses a single applied force to do work against a single load force. These were the devices that put a load into motion and calculated the ratio of the output force to the input force, known today as a “mechanical advantage”. Simple machines are generally considered the elementary “building blocks” for more complicated machines (called “compound machines”). For example, wheels, levers and pulleys are used to make a bicycle. As time passed, humans learned new concepts and evolved the old ones in many subjects like physics, chemistry, electronics, etc. This evolution happened in terms of machines as well to not only solve some real-world problems but also to reduce a person’s workload. The machines became more and more complex but the machine makers found ways to make them easily usable so that the general public, without much knowledge regarding the machine, can use it as well. One of the ways was to introduce a User Interface between the human and the machine. A User Interface is nothing but a space where interactions between a human and a machine occur. The goal of this interaction is to allow effective operation and management of the machine from the human end, while the machine at the same time feeds back data that aids the operators' decision-making process.

There are different types of user interfaces namely:

- **Command Line Interface** - The Command-Line Interfaces are one of the oldest interfaces. The computer responds to the commands typed by the operator. One of the most well-known example of a command-line interface is the MS-DOS. This interface, however, has a drawback. To use this interface, the operator has to remember a range of different commands. Due to this reason, this interface may not be ideal for novice users.
- **Graphical User Interface** - The Graphical User Interfaces (GUI) are sometimes also referred to as WIMP because they use Windows, Icons, Menus, and Pointers. Operators use a pointing device (such as a mouse, touch-pad or trackball) to control a pointer on the screen which then interacts with other on-screen elements. It allows users to interact with devices through graphical icons and visual cues.
- **Menu Driven Interface** - A menu-driven interface can be commonly seen in ATMs (also known as automated teller machines), ticket machines and information kiosks (for example in a museum). They provide a simple and easy to use interface comprised of a series of menus and sub-menus that the user accesses by pressing buttons, usually on a touch-screen device.
- **Form-Based Interface** - A form-based interface uses text-boxes, drop-down menus, text areas, check-boxes, radio boxes and buttons to create an electronic form that a user completes.
so as to enter information into a system. This is normally used on websites to gather data from a user, or in call centres to allow operators to quickly enter data gathered over the phone.

- Natural Language Interface - In neuropsychology, linguistics, and the philosophy of language, a natural language or ordinary language is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation. Natural languages can take different forms, such as speech or signing. English, for example, is a natural language. A natural language interface is a spoken interface where the user interacts with the computer by talking to it using a natural language. Also known as a ‘conversational interface’, this interface simulates having a conversation with a computer. Made famous by science fiction films like Star Trek, natural language systems are not yet advanced enough to be in wide-spread use. They are commonly used in Interactive Voice Response Systems (telephone systems) where the user can speak their responses instead of pressing numbered buttons. An example of this interface is Voice Recognition. This is the kind of interface used by the popular iPhone assistant called Siri and Cortana in Windows. Now that we have learned the basics about machines and user interfaces and it’s types, let us proceed with the research. For the demonstration of this research, we will be using a micro-controller (here, an Arduino Nano), a Stepper Motor and an A4988 Stepper Motor Driver paired together as a machine, and a natural language interface to drive that Arduino powered machine. The natural language interface will be built using Python language and some additional modules like SpeechRecognition, PySerial, etc. This interface will basically hear the command given by the human through a microphone and send it to the Arduino where that command is processed which will then drive the machine as per the command. Python is an interpreter, high-level, general-purpose programming language. Created by Guido van Rossum, Python was first released in 1991. Python's design philosophy emphasizes code readability with its notable use of significant white-space. It’s language constructs and object-oriented approach aim to help programmers write clear, logical code for not only small but also large scale projects.

Python is a great choice for not only beginners but also professionals due to its following features:

- It is easy to use.
- It is platform Independent and is an integrable language.
- It is a multi-paradigm programming language.
- It doesn’t need a “bulky” code unlike some other programming languages and is dynamically typed.
- Has a large standard library providing a rich set of modules and functions.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one - and preferably only one - obvious way to do it" design philosophy. Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture.' To build a natural language interface out of Python, we need the following modules:

- SpeechRecognition - SpeechRecognition is a library, created by Anthony Zhang, used to perform speech recognition functions using several engines and APIs. It can work both online and offline. Some engines and APIs supported by the Speech Recognition module are:
  - CMU Sphinx (works offline)
  - Google Speech Recognition
  - Google Cloud Speech API
  - Microsoft Bing Voice Recognition
  - IBM Speech-to-Text

From the above engines, we will be using the Google Speech Recognition as it works online, is known to be highly accurate and has many language packs for different individuals. Here we will be using the Indian English language pack.

- PyAudio - PyAudio is a Python supported cross-platform audio input/output stream library. To make the Speech Recognition module get input from a microphone, we have to get the PyAudio module.
- google-api-Python-client - The Google API Client for Python is a client library for accessing some of the many Google APIs like Moderator, Plus, Speech, etc. Here we have to get this module to connect the SpeechRecognition module to the Google Speech Recognition for recognizing the voice from the microphone.
- PySerial - PySerial is a Python Serial Port Extension module. It is used to communicate with devices connected to the serial ports of the host computer. In our case, the Arduino will be connected to one of the computer’s serial port and the PySerial will act as a middle man between the user and the Arduino to send the commands and receive the outputs to and from the Arduino.
- Time - Time is an inbuilt module provided by Python used to perform various operations regarding time, it’s conversions and representations. Here we will be using a method from the time module named sleep(sec). This method is basically used to halt the execution of the program for a time specified in the argument of the method.

Now that we have collected the required modules to get our research started, let’s talk about the plan to get the demonstration working.

II. SETTING UP

The Arduino powered machine we will be using is just a simple Arduino to a NEMA 17 Stepper connection via an A4988 driver as shown in the below pin-out diagram.

The Stepper motor is connected to an A4988 driver board. A 12v 2A power supply, coupled with a 0.47uF capacitor for decoupling, is given to the A4988 driver to its pins VMOT...
and GND and a 5V power supply from the Arduino is given to
the VDO pin of the A4988. The RST and SLP pins are
connected to each other and the STEP pin is connected to
the Arduino’s D2 pin and the DIR pin is connected to
the Arduino’s D3 pin. Lastly, after the pins are properly
connected, the Arduino is connected to one of the computer’s
serial port, here COM5. After connecting to the computer, we
will be testing whether the above circuit works. We will be
using Arduino IDE for the same. To test the above circuit, we
will be writing the following code:

```cpp
const int step = 2; // Step pin.
const int dir = 3; // Dir pin.
const int rot = 200; // Defines 200 pulses for one rotation of
motor. This value depends on the motor.
void setup() {
  // Sets the two pins as Outputs.
pinMode(step, OUTPUT);
pinMode(dir, OUTPUT);
}
void loop() {
  digitalWrite(dir, HIGH); // Moves the motor in a particular
direction.
  for(int x = 0; x < rot; x++) {
    digitalWrite(stepPin, HIGH);
delayMicroseconds(750);
digitalWrite(stepPin, LOW);
delayMicroseconds(500);
  }
delay(1000); // One second delay
digitalWrite(dirPin, LOW); // Changes the rotations direction
}
```

After uploading this code to the Arduino, the motor will start
rotating in a pattern. The motor will first rotate in one direction
and complete one rotation and then will wait for a second.
Then it will change the direction of the rotation and will
complete a rotation (at a different speed than before) and then
will wait for a second. This pattern will keep on going till the
Arduino is turned off or the code is changed. But this shows
that the above circuit is working and that we can proceed with
the demonstration.

### III. DEMONSTRATION

The demonstration will be divided into two parts, namely the
Initial Stage where we will get the basic modules and
interface-machine communication working and the Final Stage
where the interface-machine communication will be evolved
thoroughly to make it work according to the user’s needs. Both
the stages will work on two parts: the Interface and the
Processor. The Interface part will hear the human for a command according to which the machine must work and
send that command to the Arduino whereas the Processor part
will be the Arduino side (the code and the machine included)
which will get the command from the interface and will
produce an output according to the command spoken by the
human using the Interface.

#### I. INITIAL STAGE:
At the initial stage, we will focus on
achieving two goals.
- Making sure a communication medium is formed
  between the Interface and the Arduino and
- Making sure the Arduino responds to the request sent by
  the user through the interface.

**(i). INTERFACE:** To achieve the above-mentioned goals,
we will need to first make the Arduino and the interface
communicate. For that, we have to load the PySerial module
first and create a Serial interface out of it for the Arduino. The
following code demonstrates an interface using PySerial:

```python
from serial import Serial
ser = Serial(port='COM5',baudrate=115200,timeout=1)
sleep(3) // Wait for Arduino to initialize completely.
print(ser.readline().decode('ascii')) // Prints welcome from
Arduino.
ch = input("Enter a character: ")
ser.write(ch.encode()) // Sends the character to the Arduino.
ardData = ser.readline().decode('ascii') // Reads the response
from the Arduino.
print(ardData) // Prints the response.
```

This will create a basic interface for the Arduino to be
controlled. This code will just ask for a character from the
human and send it to the Arduino and requests output from
the Arduino.

**(ii). PROCESSOR:** Now that the interface is coded to send
a character to the Arduino, the Arduino needs to process that
character and send the output as well to achieve the second
goal. The following code is uploaded on the Arduino to
respond to the request sent by the interface:

```cpp
char ch;
void setup() {
  Serial.begin(115200); // Serial baudrate initialized as same as
  whats mentioned in the interface file.
  Serial.println("Arduino Started!"); // Printing as welcome
  message at the beginning after Arduino initializes.
}
void loop() {
  while(Serial.available()){
    ch = Serial.read(); // Reads serial sent by the interface and
    stores it in variable ch.
    Serial.println("Arduino Reads: ");
    Serial.println(ch); // Sends output to the interface as response.
  }
}
```
Now that both the codes are in place, we will run the interface file. After running the interface file, there will be a delay of 3 seconds where the interface file will let the Arduino initialize properly. If the connection is properly established between the interface and the device, the interface will print the welcome message which was coded in the Arduino. After printing the welcome message it will ask for a character.

After entering the character ‘t’, it sends the character to the Arduino, after which the Arduino processes it and sends a response back to the interface.

The goals for the initial stage are achieved and the code can be evolved more according to the user’s will. Hence moving to the Final Stage.

II. FINAL STAGE: At the final stage, we will focus on achieving the following goals:

- Implementing Speech Recognition.
- Making the interface hears the command at the press of a button.
- The command after being recognized by the user travels from the Interface to the Processor with some added delimiters to make the processing of the commands easy.
- Making sure the Processor is responding to the command serially as well as performing the intended output.

II(i). INTERFACE: In the initial stage, we successfully created an interface which communicated with the Arduino without any problems. In this stage, we are going to improve it to accept commands through voice on a press of a button and send it to the Arduino where the Processor part handles the command and processes an output. The following code demonstrates the same:

```python
import speech_recognition as sr
import serial
import time

# Methods
def Speech():
    r = sr.Recognizer()  # Initialize the Speech Recognition with sr.Microphone() as source:
    print("Speak something ->")
    rec = r.listen(source)  # Makes the computer listen to commands from the microphone and stores the heard data in a variable rec.

# Main Code
ser = Serial(port='COM5', baudrate=115200, timeout=1)
print(ser.readline().decode('ascii'))
ch = input("Enter y to listen or q to exit: ")
while (ch != 'q'):
    if(ch == 'y'):
        command = Speech()
        ser.write(command.encode())
        sleep(1.25)
        ardData = ser.readline().decode('ascii')
        print(ardData)
    else:
        print("Wrong choice")
        ch = input("Enter y to listen or q to exit: ")
```

Here we created a simple, yet highly effective natural language interface which will recognize the commands from the input we give via a microphone i.e recognize the commands it “hears” and will forward that command with some additional delimiters to the Processor part where the command will be processed accordingly.

II(ii). PROCESSOR: Now that the interface is coded to send the command it hears to the Arduino, the Arduino needs to process it. But before it, we have to decide what commands should the Arduino execute and what it should skip. Since we are using the Arduino with a NEMA 17 Stepper motor, the only thing we can do is to make the motor rotate. However, we can make it rotate in different ways:

- We can make it run indefinitely at a specific speed.
- We can rotate the motor at a specific angle.
- We can make the motor “move to” a position at a specific speed. When the position is reached, the motor will stop rotating.
- Stopping the motor entirely.

These, for now, are the three operations which we will be working on. Now that the operations are decided, we have to select commands for the same. That is, there will be a specific command for a specific operation. Since we are using English as the language to communicate with the Processor, we will be using simple English sentences with a special keyword to drive our machine. The keywords will differentiate between the operations to be performed and according to those
keywords, the sentence will be broken down to extract the data required to run the machine. Hence, to:

- Make the motors run indefinitely at a specific speed, we are going to use the statement: “Run at speed x” where x is the speed in RPM.
- Rotate the motor at a specific angle, we are going to use the statement: “Rotate x degrees” where x is the angle of rotation.
- Move the motor at a specific position at a specific speed, we are going to use the statement: “Move to position x with speed y” where x is the number of steps the stepper should move and y is the speed in RPM.
- To stop the motor, we are going to use the statement: “Stop”.

The underlined words in the above statements will act as the keyword. Now that we have the commands and the operations these commands will execute sorted out, we’ll upload the following code to the Arduino:

```cpp
#include <AccelStepper.h> // Importing AccelStepper library created by Mike McCauley to control stepper motors accurately and easily.

// Variable Declarations.
char ch;
String request;
String command;

AccelStepper Stepper(1, 2, 3); // Initializing AccelStepper instance with our Stepper Motor

// The setup code that runs once.
void setup(){
  Serial.begin(115200); // Initializing Serial Port.
  Serial.println("Arduino started listening");
}

// Loop code that keeps running repeatedly.
void loop(){
  while(Serial.available()){  
    ch = Serial.read();
    request += ch;
    if(ch != '.'){ break; }
    // Since Arduino reads one char at a time, this code is used to check that the entire command is read rather than one char at a time.
    // It checks whether the full sentence is read and stored in the variable request.
    else{
      Serial.print("Arduino heard: ");
      request.toLowerCase();
      // If the control identifies the keyword "run", this piece of code will execute.
      if(request.substring(1, 4) == "run"){
        int del0 = request.indexOf('');
        int speed = request.substring(del0+1, del1).toInt();
        Stepper.setMaxSpeed(speed*3.33334);
        Stepper.setSpeed(speed*3.33334);
        Serial.println(request.substring(1, del1+1));
        while(ch != '#'){
          Stepper.runSpeed();
          ch = Serial.read();
        }
      }
      // If the control identifies the keyword "move", this piece of code will execute.
      else if(request.substring(1, 5) == "move"){
        int del0 = request.indexOf(' ');
        int del1 = request.indexOf(',', del0+1);
        int del2 = request.indexOf(',', del1+1);
        int del3 = request.indexOf(',', del2+1);
        int del4 = request.lastIndexOf(' ');
        int del5 = request.indexOf('.');
        int position = request.substring(del2+1, del3).toInt();
        int speed = request.substring(del4+1, del5).toInt();
        Stepper.moveTo(position);
        Stepper.setMaxSpeed(speed*3.33334);
        Stepper.setSpeed(speed*3.33334);
        Serial.println(request.substring(1, del5+1));
        while(ch != '#'){
          Stepper.runSpeedToPosition();
          ch = Serial.read();
        }
      }
      // If the control identifies the keyword "rotate", this piece of code will execute.
      else if(request.substring(1, 7) == "rotate"){
        Stepper.setCurrentPosition(0);
        delay(500);
        int del0 = request.indexOf(' ');
        int del1 = request.indexOf(',', del0+1);
        int deg = request.substring(del0+1, del1).toInt();
        int steps = (deg/1.8);
        Stepper.moveTo(steps);
        Stepper.setMaxSpeed(200.0004);
        Stepper.setSpeed(200.0004);
        Serial.println(request.substring(1, del1+1));
        while(ch != '#'){
          Stepper.runSpeedToPosition();
          ch = Serial.read();
        }
      }
      // If the control identifies the keyword "stop", this piece of code will execute.
      else if(request.substring(1, 4) == "stop"){
        Stepper.setCurrentPosition(0);
        delay(500);
        Stepper.setSpeed(0);
      }
    }
  }
}
```

IJESC, November 2019

http://ijesc.org/
else if(request.substring(1, 5) == "stop") {
    int del0 = request.indexOf('.');
    Stepper.setSpeed(0);
    Serial.println(request.substring(1, del0+1));
    while(ch != '#') {
        ch = Serial.read();
    }
    // If the control doesn't identify any keyword, this piece of code will execute.
    else {
        int del0 = request.indexOf('.');
        Serial.println("Wrong command: " + request.substring(1, del0+1));
        while(ch != '#') {
            ch = Serial.read();
        }
        request = "#";
    }
}

Now that both the codes are in place, we will launch the Final Interface file.

After the 3 second delay at the start, we will get this screen:

Here we have to press the 'y' button so that the speech recognition module can start and hear our commands.

When the Speak Something text appears, we will be speaking one of the statements as decided before with a value.

Here we said in the command “run at speed 200.” As soon as the interface got the command (from the microphone) and recognized it, it forwarded the command to the Arduino where the Processor code extracted the keyword and the data and performed the operation it was meant to do, here, running the motors at the speed of 200 RPM. To make the Arduino listen to another command, we can press the ‘y’ button or ‘q’ button to quit.

Here we pressed the ‘y’ button and gave the Arduino to rotate the stepper 90 degrees. As soon as the Arduino got this command, it stopped the prior operation i.e running at the speed of 200 rpm indefinitely and then rotated the motor 90 degrees from the stopped position. This shows that before the new command is executed, the old command is stopped.

This time after pressing ‘y’ button, an unidentified command was passed which is “hello world”. Since this command was unidentified, the Arduino returned the statement “Wrong Command: hello world.”. You can press ‘y’ as many times as you want to provide a command.

Pressing ‘q’ will just close the Interface.
IV. CONCLUSION

This was a simple demonstration of creating and using a natural language interface which can accept commands spoken in simple English and execute operations according to those commands. For, limited machines as shown above, this demonstration stood tall to its expectations. Also, since adding more components to the Arduino is possible, we can expand the number of commands and improve the machine according to a user’s will. There is also a vast scope of improvement here. The speech recognition can be improved, Interface can be paired with artificial intelligence, the performance can be improved, the machine can be made more advanced etc. are some of the future scopes. The possibilities, in this case, are endless.

V. REFERENCES

[1]. https://en.wikipedia.org/wiki/Machine/


[6]. https://pypi.org/project/SpeechRecognition/

[7]. https://pypi.org/project/pyserial/


[9]. http://www.airspayce.com/mikem/arduino/AccelStepper/