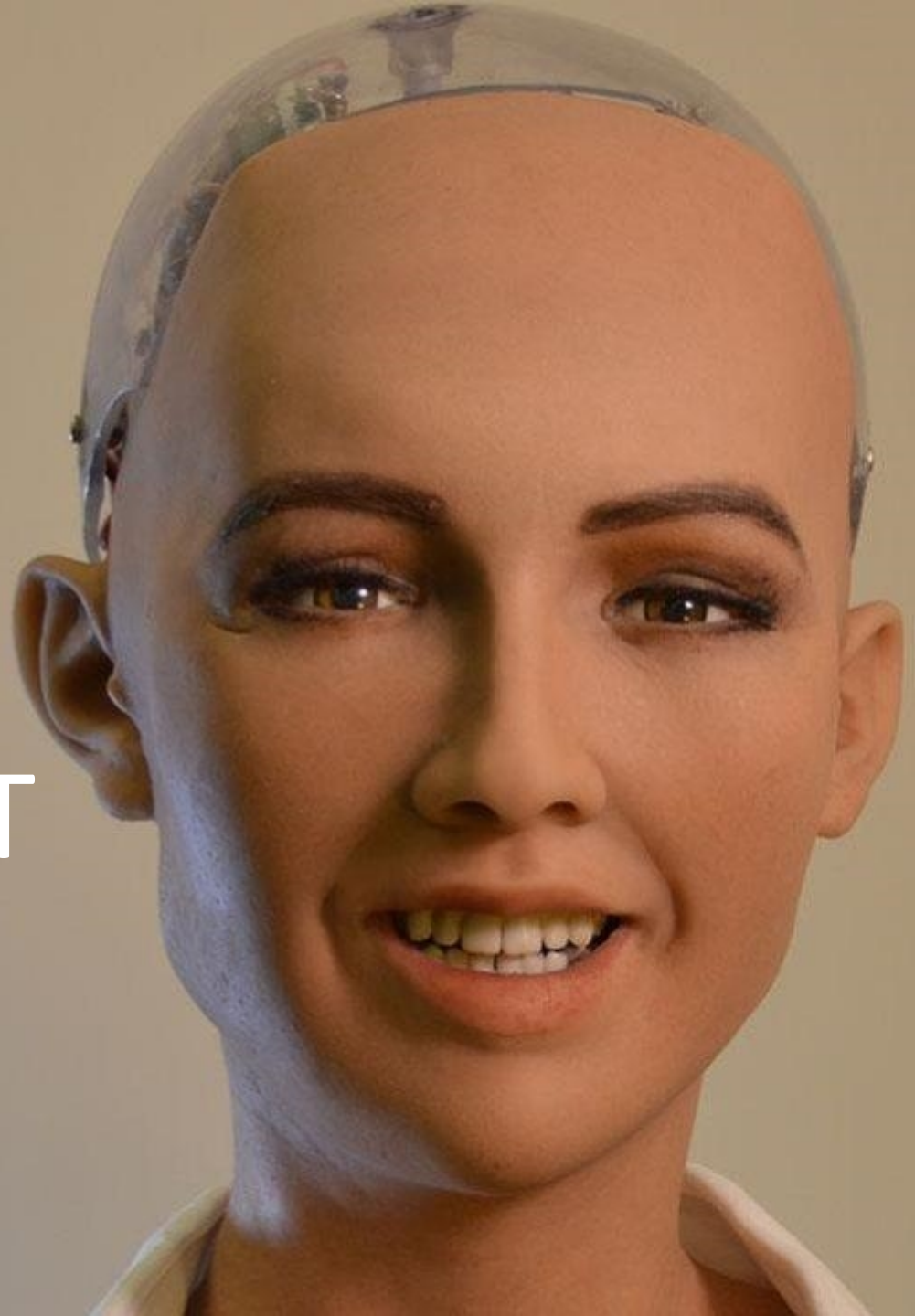
PART II

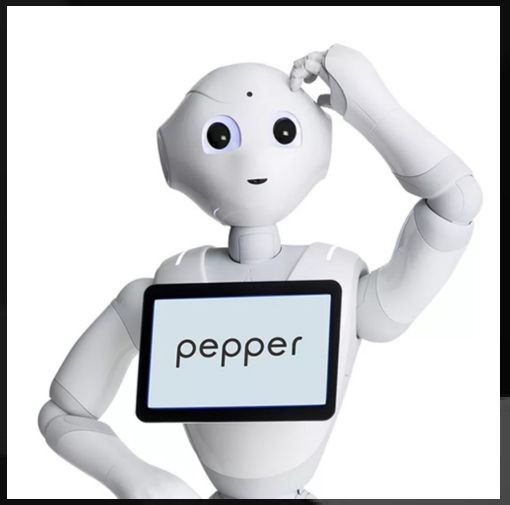
Application for HUMANOID ROBOT

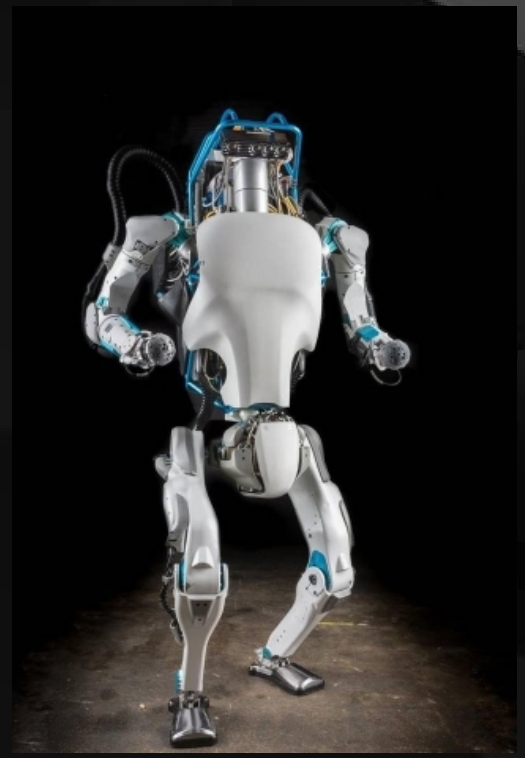


INDEX

INTRODUCTION

A **humanoid robot** is a robot with its body shape built to resemble the human body. The design may be for functional purposes, such as interacting with human tools and environments, for experimental purposes, such as the study of bipedal locomotion, or for other purposes. In general, humanoid robots have a torso, a head, two arms, and two legs, though some forms of humanoid robots may model only part of the body, for example, from the waist up. Some humanoid robots also have heads designed to replicate human facial features such as eyes and mouths





Our Application

Our Application can recognize people's faces using facial recognition technology

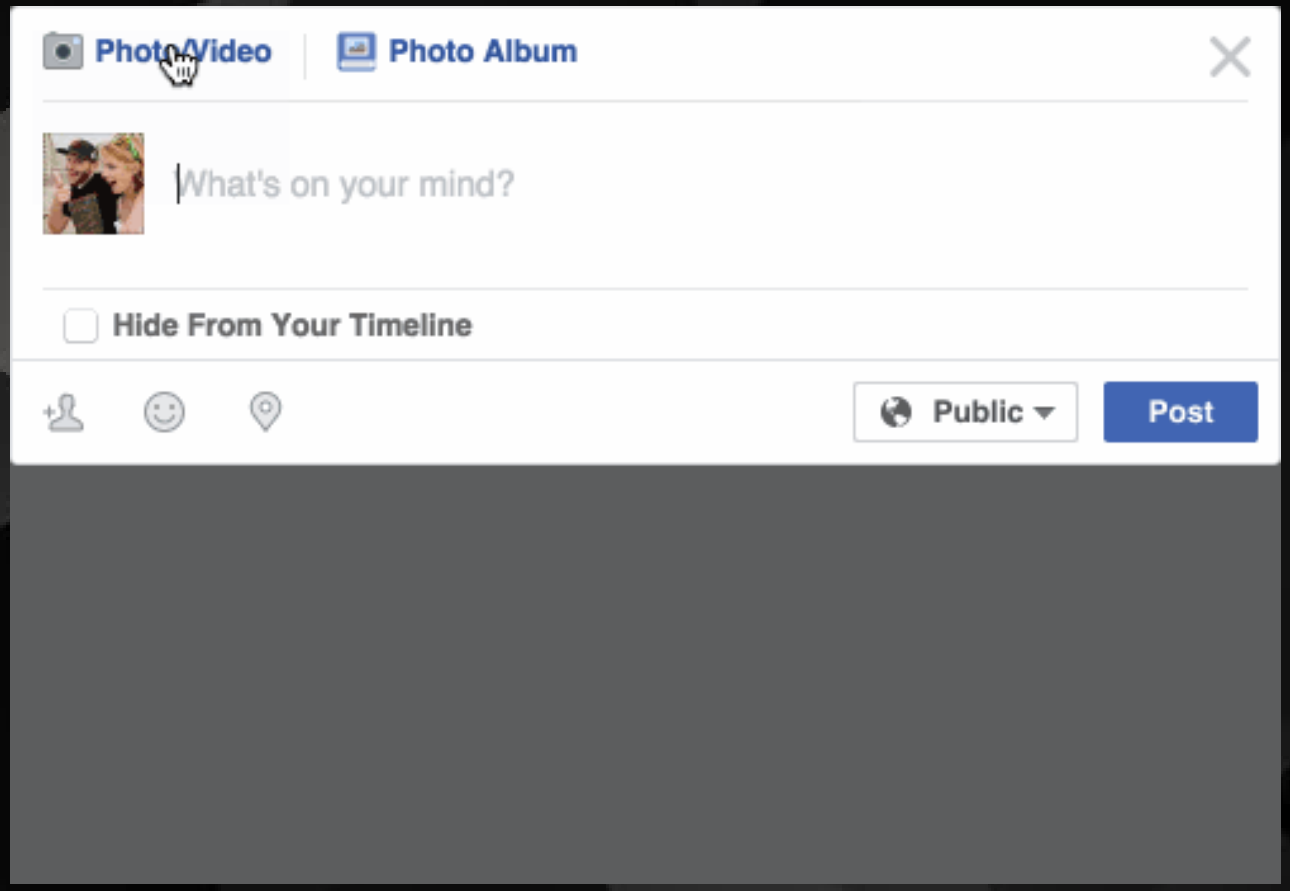
Our Application can have a conversation with people using NLP technology

Our Application is built to be interfaced with the Humanoid Robot

This type of Robot can be used in fields such as Therapy, Teaching and Personal Assistance

FACE RECOGNITION

Face recognition is a method of identifying or verifying the identity of an individual using their face. Face recognition systems can be used to identify people in photos, video, or in real-time.

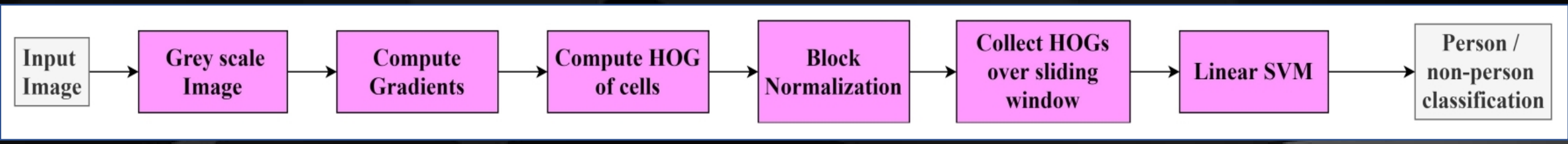


Facebook's algorithms are able to recognize your friends' faces after they have been tagged only a few times. It's pretty amazing technology — Facebook can recognize faces with 98% accuracy which is pretty much as good as humans can do!

Face Recognition Steps

1. Face Detection
2. Face Alignment
3. Encoding Faces
4. Comparing Faces

Step 1: Face Detection



a. We convert the Original RGB image into a Grey Scale image

Webcam resolution: 640 x 480

Step 1: Face Detection

b. Calculate Gradients

We calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

-1	0	1

-1
0
1

Next, we can find the magnitude and direction of gradient using the following formula. We then get 2 matrices – Gradient Direction and Gradient Magnitude

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

Why replace Pixels with Gradients?

If we analyze pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the *direction* that brightness changes, both really dark images and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve!

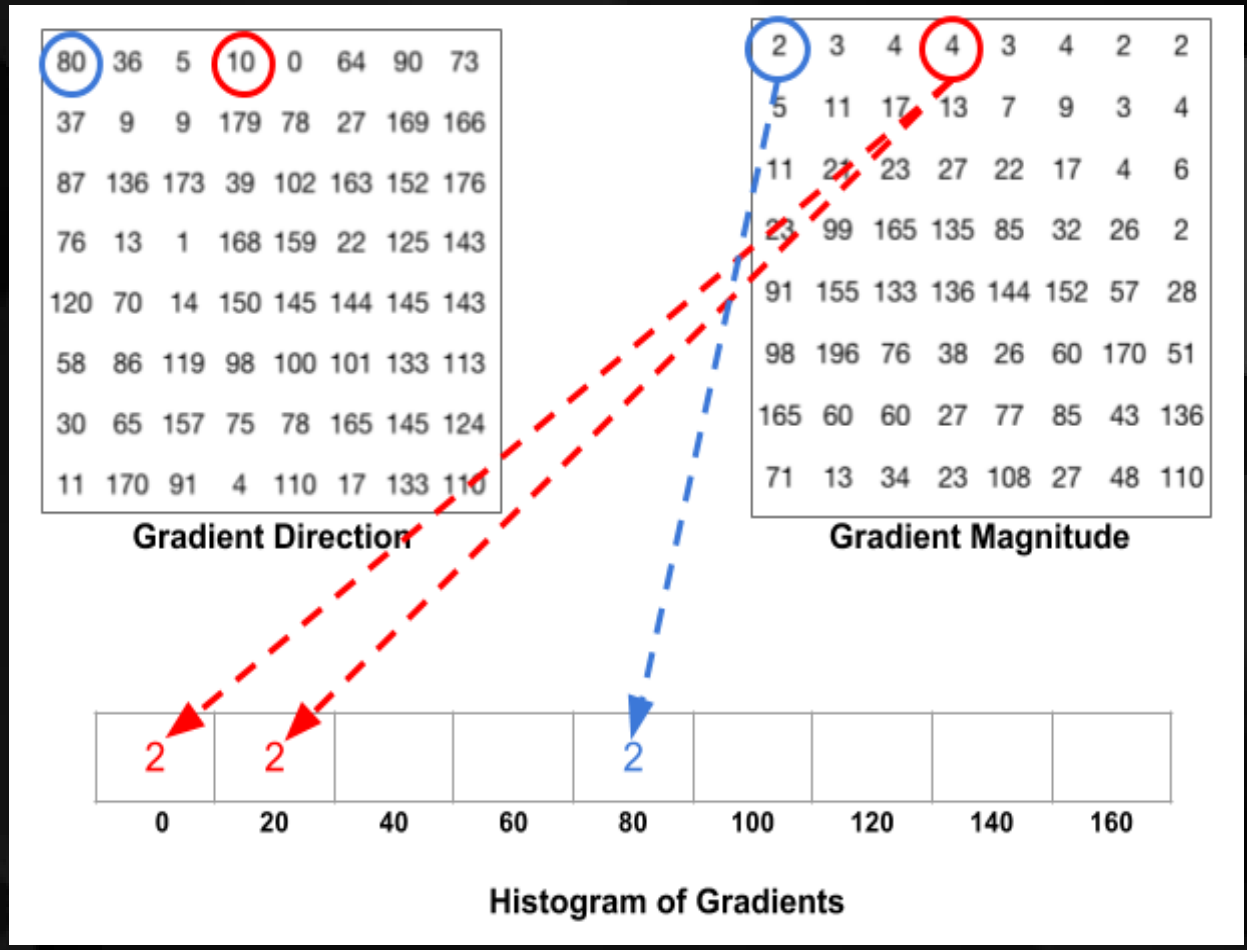
Step 1: Face Detection

c. Calculate the Histogram of Gradients in 8x8 cells

Divide the image into many 8x8 pixel cells. In each cell, the magnitude values of these 64 cells are binned and cumulatively added into 9 buckets of unsigned direction (no sign, so 0-180 degree rather than 0-360 degree; this is a practical choice based on empirical experiments).

For better robustness, if the direction of the gradient vector of a pixel lays between two buckets, its magnitude does not all go into the closer one but proportionally split between two. For example, if a pixel's gradient vector has magnitude 8 and degree 15, it is between two buckets for degree 0 and 20 and we would assign 2 to bucket 0 and 6 to bucket 20.

This interesting configuration makes the histogram much more stable when small distortion is applied to the image.



Step 1: Face Detection

HOG visualization

Step 1: Face Detection

d. 16 x 16 Block Normalization

In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half. Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations.

We normalize over a bigger sized block of 16×16. A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector. The block is normalized.

Let's say we have an RGB color vector [128, 64, 32]. The length of this vector is sqrt ($128^2 + 64^2 + 32^2$) = 146.64. Dividing each element of this vector by 146.64 gives us a normalized vector [0.87, 0.43, 0.22]. Now consider another vector in which the elements are twice the value of the first vector 2 x [128, 64, 32] = [256, 128, 64]. Normalizing [256, 128, 64] will result in [0.87, 0.43, 0.22], which is the same as the normalized version of the original RGB vector. You can see that normalizing a vector removes the scale.

Step 1: Face Detection

e. Collect HOGs over sliding window

A window of size 128 x 64 is used with 16 x 16 Blocks.

1. The normalized 36 x 1 vector of first Block is calculated.
2. The Block is then moved by 8 pixels and a normalized 36 × 1 vector is calculated over this Block and the process is repeated.
3. In one window we obtain (9x128x64) / (8x8) = 1152 x 1 vector

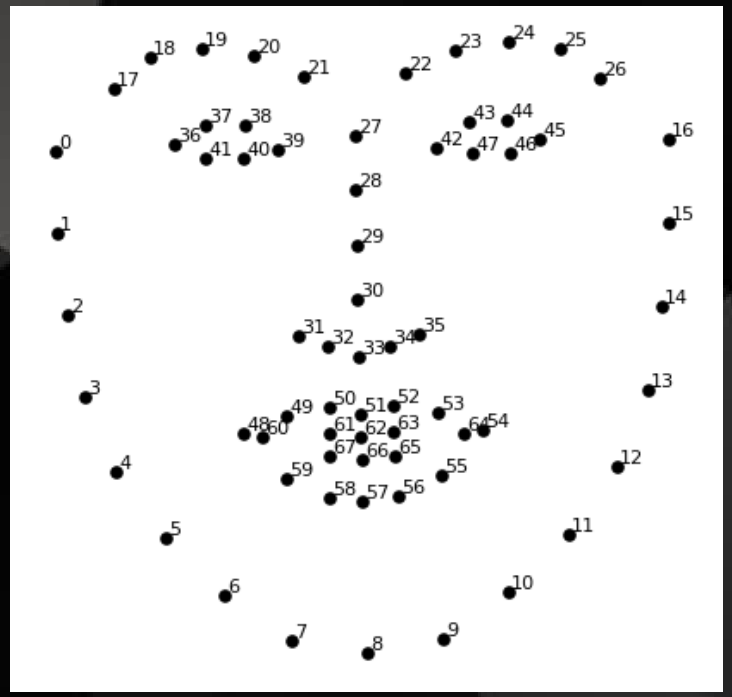
f. Linear SVM (C = 0.01)

At each window we extract 1152 x 1 HOG vector and apply Linear SVM classifier. If the classifier detects a face with sufficiently large probability, we record the bounding box of the window.

Step 2: Face Alignment - Posing and Projecting Faces

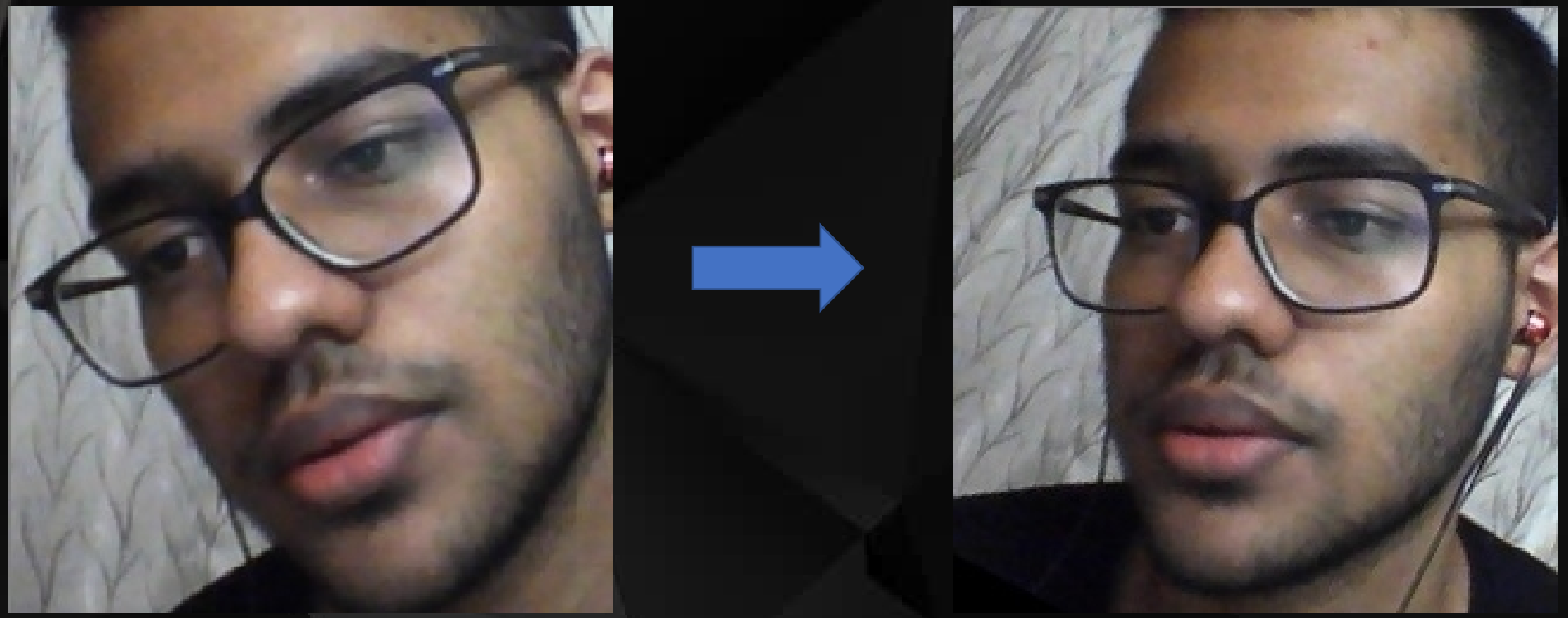
Face Alignment consists of 2 steps:
1. Face Landmark Estimation
2. Affine Transformation

First we need to find certain feature points. To do this, we have used an algorithm called **face landmark estimation**. The basic idea is we will come up with 68 specific points (called *landmarks*) that exist on every face — Eyes, Eyebrows, Nose, Mouth and Jawline. Then we will use ensemble of regression trees to be able to find these 68 specific points on any face.



Step 2: Face Alignment - Posing and Projecting Faces

Now that we know where the eyes and mouth are, we'll simply rotate, scale and shear the image so that the eyes and mouth are centered as best as possible. We won't do any fancy 3d warps because that would introduce distortions into the image. We are only going to use basic image transformations like translation, rotation and scale that preserve parallel lines (called affine transformation)



Step 3: Encoding Faces

There are 2 ways of Facial Recognition:

1. The simplest approach to face recognition is to directly compare the unknown face we found in Step 2 with all the pictures we have of people in our database.

The problem here is that in a huge database like Facebook, it can take hours to find the right Matching Face. But we need the matches in milliseconds.

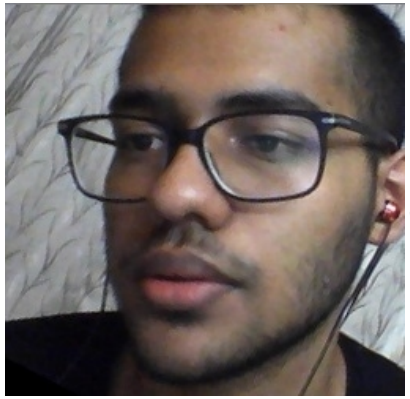
2. We extract a few basic measurements or features from each face. Then we could measure our unknown face the same way and find the known face with the closest measurements.

Ok, so which measurements should we collect from each face to build our known face database? Ear size? Nose length? Eye color? Something else?
It turns out that the measurements that seem obvious to us humans (like eye color) don't really make sense to a computer looking at individual pixels in an image. Researchers have discovered that the most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

Step 3: Encoding Faces

We pass the image of the face into a Convolutional Neural Network. This model is a ResNet network with 29 conv layers. It's essentially a version of the ResNet-34 network from the paper Deep Residual Learning for Image Recognition by He, Zhang, Ren, and Sun with a few layers removed. This network was trained on 'Labelled Faces in the Wild' dataset by Davis King with an accuracy of 99.38%. The network gives an output of a Feature vector with 128 dimensions.

Input Image



128 Measurements Generated from Image

0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

Step 4: Comparing Faces

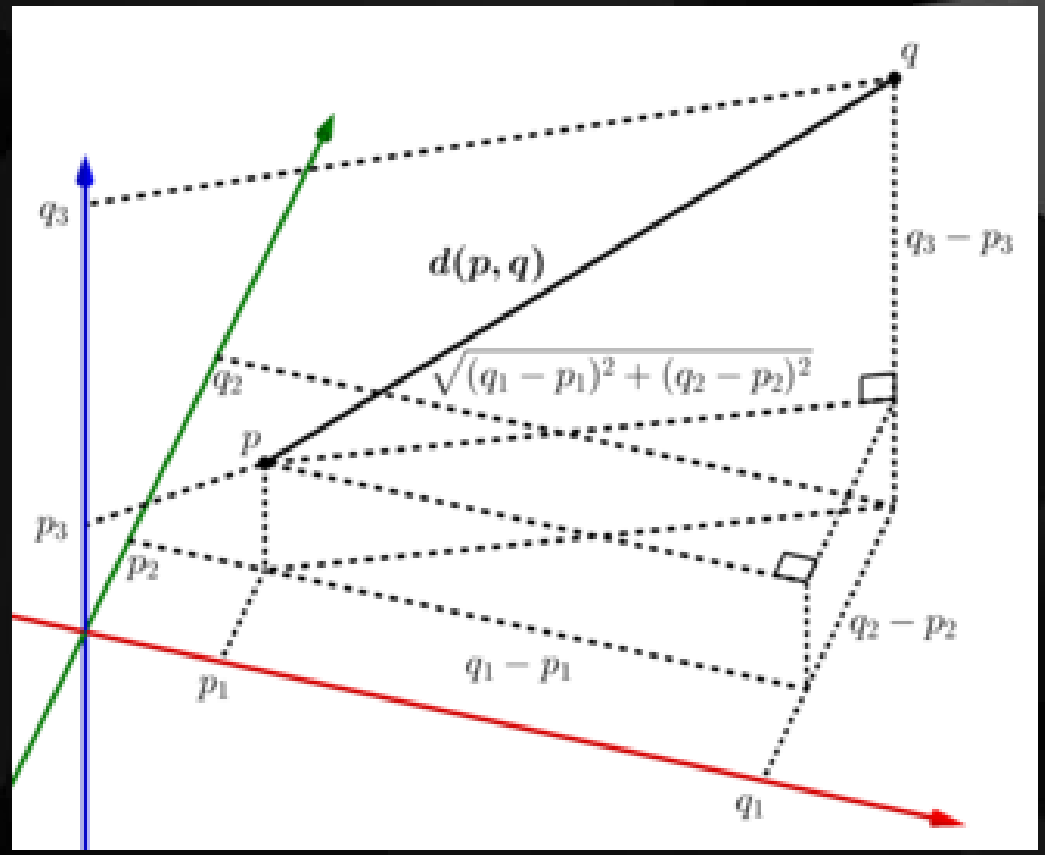
We find the Euclidean distances between the two faces and we associate this face with the closest match.

The **Euclidean distance** between points **p** and **q** is the length of the line segment connecting them.

If we don't find a match, we feed this new face into the database and generate its own identity.

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$



NATURAL LANGUAGE PROCESSING (NLP)

Natural language processing (**NLP**) is a subfield of linguistics, computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.
Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation.

What is a Chat Bot?

According to Oxford Dictionary, a chatbot is **“A computer program designed to simulate conversation with human users, especially over the Internet.”**

What can Chat Bots do?

- **Improve customer service**
- **Personalize communication**
- **Improve a response rate**
- **Automate repetitive tasks**

Types of Chat Bots

Depending on way bots are programmed, we can categorize them into two variants of chatbots: Rule-Based (dumb bots) & Self Learning (smart bots).

- **Rule-Based Chatbots**: This variety of bots answer questions based on some simple rules that they are trained on.
- **Self-Learning Chatbots**: This variety of bots rely on Artificial Intelligence(AI) & Machine Learning(ML) technologies to converse with users.

  Self-learning Chatbots are further divided into **Retrieval based and Generative.**

Retrieval based

The bot is trained to rank the best response from a finite set of predefined responses. The responses here are entered manually, or based on a knowledge base of pre-existing information.

Generative

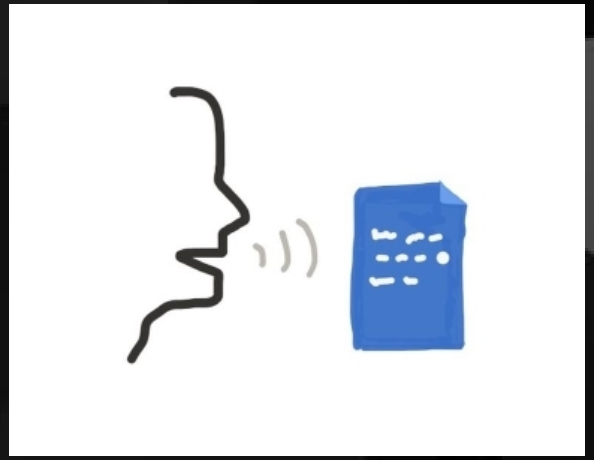
These chatbots are not built with predefined responses. Instead, they are trained using a large number of previous conversations, based upon which responses to the user are generated. They require a very large amount of conversational data to train.

Our Chat Bot Architecture

Here, the Bot will have conversations with the person standing in front of it.

This is a 3 step process:

1. Speech to Text conversion: The user speaks a sentence. This sentence is converted into text using the Google Speech to Text API



2. Text Processing: This text sentence is then processed using Google Dialogflow and Natural Language Tool Kit (NLTK)

Our Chat Bot Architecture

The bot can understand 5 types of intentions:

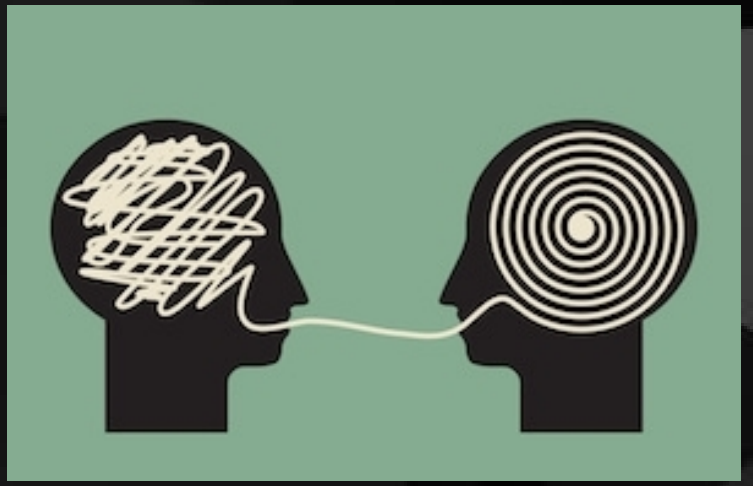
1. User is giving his information -- My name is Piyush

2. User is asking his information -- What is my name?

3. User is asking Bot information -- What are your hobbies?

4. Small Talk -- I hate you, tell me a joke

5. User wants general information -- Who is mahatma gandhi? what is coronavirus?



The chat bot identifies the Intention and generates a response for the input text. The output of this Step is in terms of text.

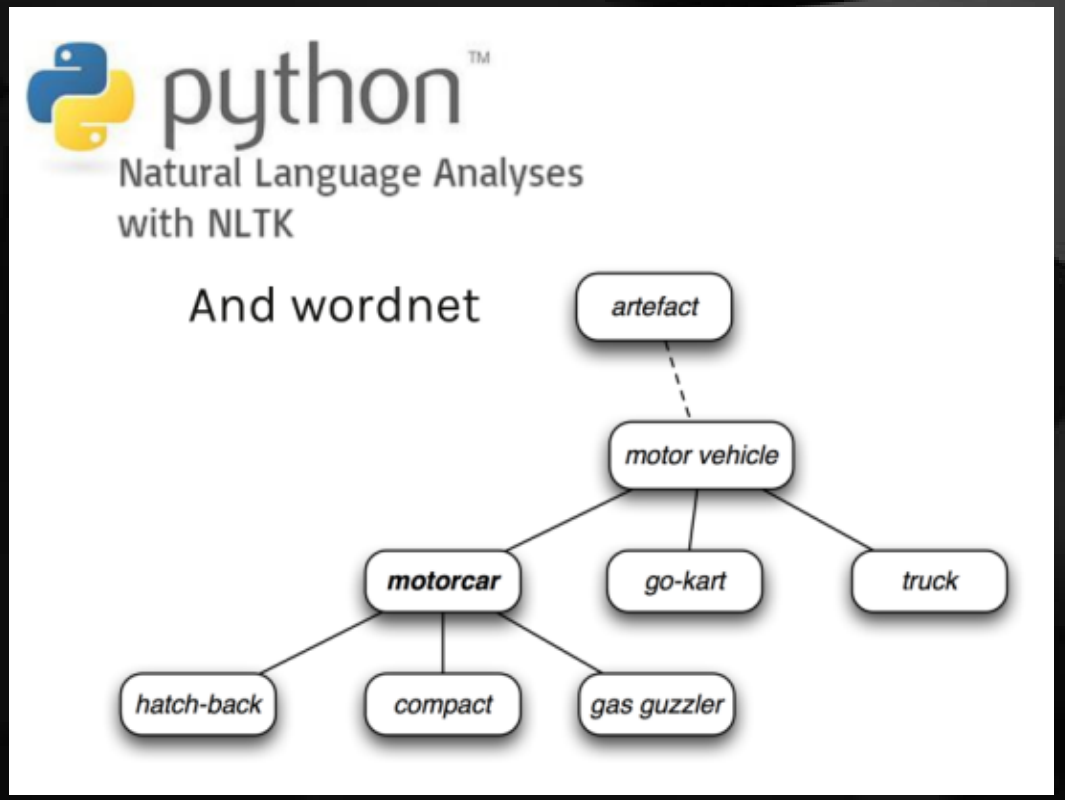
3. Text to Speech conversion: The output text is converted into speech by using Google Text to Speech API

Text Processing

We have used a combination of Natural Language Tool Kit (NLTK) and Google Dialogflow to do the job

1. Natural Language Tool Kit (NLTK)

The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology. NLTK will aid you with everything from splitting sentences from paragraphs, splitting up words, recognizing the part of speech of those words, highlighting the main subjects, and then even with helping your machine to understand what the text is all about.



NLTK processing

1. Tokenizing - Splitting sentences and words from the body of text.

```
EXAMPLE_TEXT = "Hello Mr. Smith, how are you doing today? The weather is
great, and Python is awesome. The sky is pinkish-blue. You shouldn't eat
cardboard."
```

Sentence Tokenizing - ['Hello Mr. Smith, how are you doing today?', 'The weather is great, and Python is awesome.', 'The sky is pinkish-blue.', "You shouldn't eat cardboard."]

Word Tokenizing:

```
['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing',
'today', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'Python', 'is',
'awesome', '.', 'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should',
"n't", 'eat', 'cardboard', '.']
```

NLTK processing

2. Filtering out Stop Words – Stop words are words which do not carry useful information in the sentence. Stop words are used in a sentence just for punctuation sake. Eg- 'with', 'they','so','a','the', 'too', 'only', 'those', 'i'

```
EXAMPLE_TEXT = ['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words', 'filtration', '.']

Filtered Text = ['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

3. Stemming – Stemming is used to normalize sentences.
Consider:
I was taking a ride in the car.
I was riding in the car.
This sentence means the same thing. The 'ing' in second sentence is not really required. So Stemming helps us find the basic form of words.

```
example_words = ["python","pythoner","pythoning","pythoned"]

output = ["python","python","python","python"]
```

NLTK processing

4. Part of Speech Tagging – This means labeling words in a sentence as nouns, adjectives, verbs...etc. Even more impressive, it also labels by tense, and more.

```
NN     noun, singular 'desk'
NNS    noun plural  'desks'
NNP    proper noun, singular     'Harrison'
VB     verb, base form     take
VBP    verb, sing. present, non-3d      take
CC     coordinating conjunction
WDT    wh-determinerwhich
JJ     adjective    'big'
```

Input Text = `['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'Python', 'is', 'awesome', '.']`

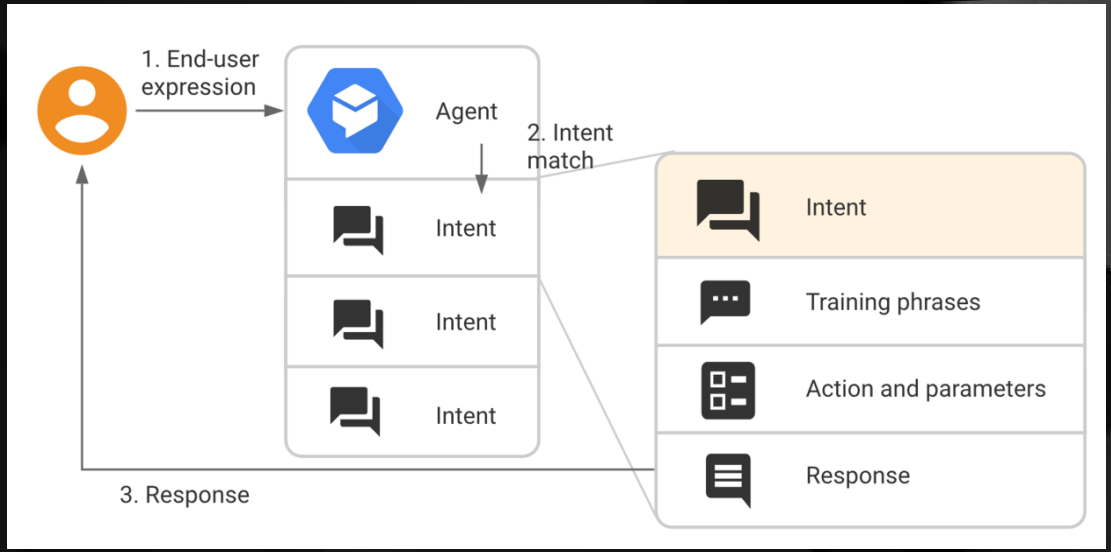
Output = [('Hello', 'NN')] [('Mr', 'NNP'), ('.', '.')] [('Smith', 'NNP')] [(',', ',')] [('how', 'WRB')] [('are', 'VBP')] [('you', 'PRP')] [('doing', 'VBG')] [('today', 'NN')] [('?', '.')] [('The', 'DT')] [('weather', 'NN')] [('is', 'VBZ')] [('great', 'JJ')] [(',', ',') [('and', 'CC')] [('Python', 'NN')] [('is', 'VBZ')] [('awesome', 'NN')] [('.', '.')]

Google Dialogflow

Dialogflow is a natural language understanding platform developed by Google, which is used to design and integrate a conversational user interface into applications.

An *intent* categorizes an end-user's intention for one conversation turn

Dialogflow matches the end-user expression to the best intent in the agent. Matching an intent is also known as *intent classification*.



Training phrases: These are example phrases for what end-users might say. When an end-user expression resembles one of these phrases, Dialogflow matches the intent.

Responses: You define text, speech, or visual responses to return to the end-user. These may provide the end-user with answers, ask the end-user for more information, or terminate the conversation.

Implementation Details

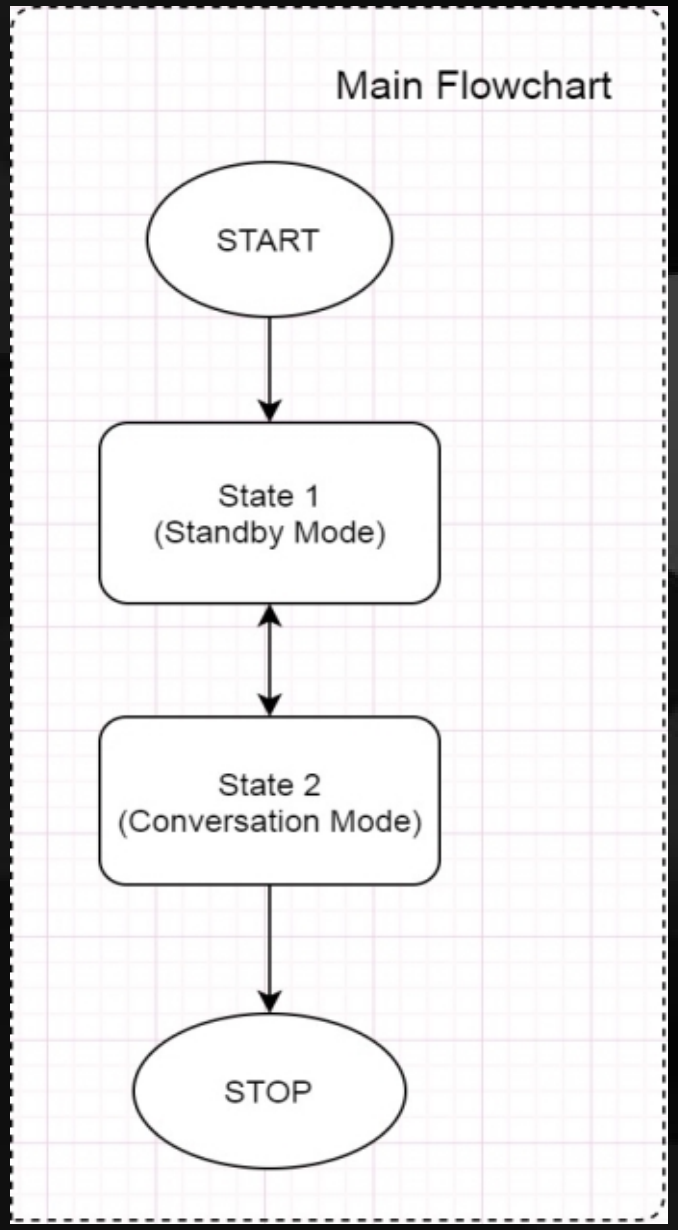
The code for this project was written entirely in **Python** programming language.

The Bot has 2 states.

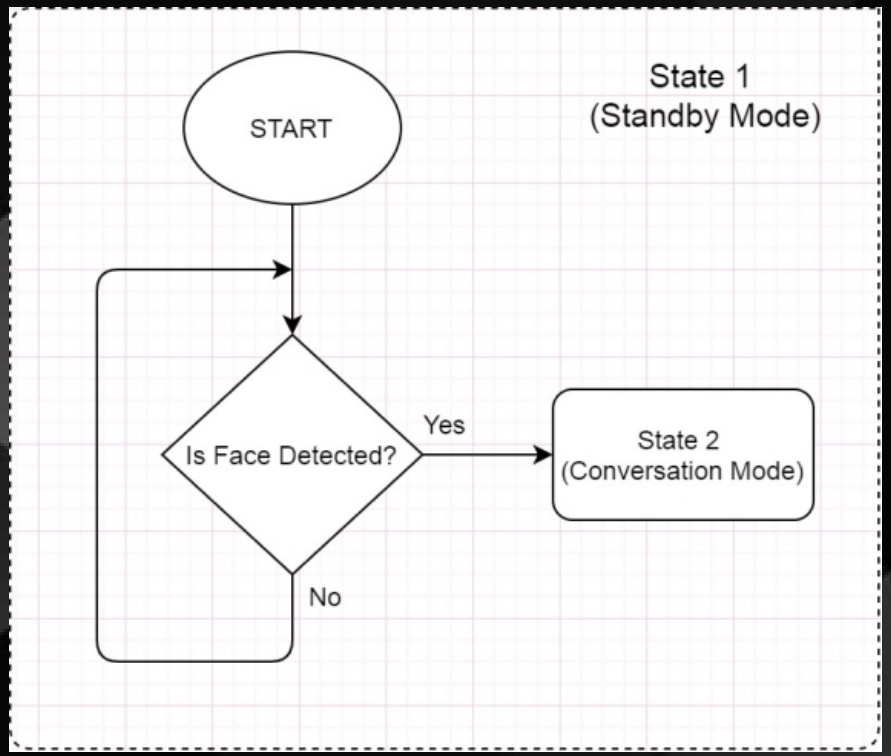
The first state is called Standby Mode. In this state, there is no person in front of the bot and the bot is waiting for a person to come in front of it.

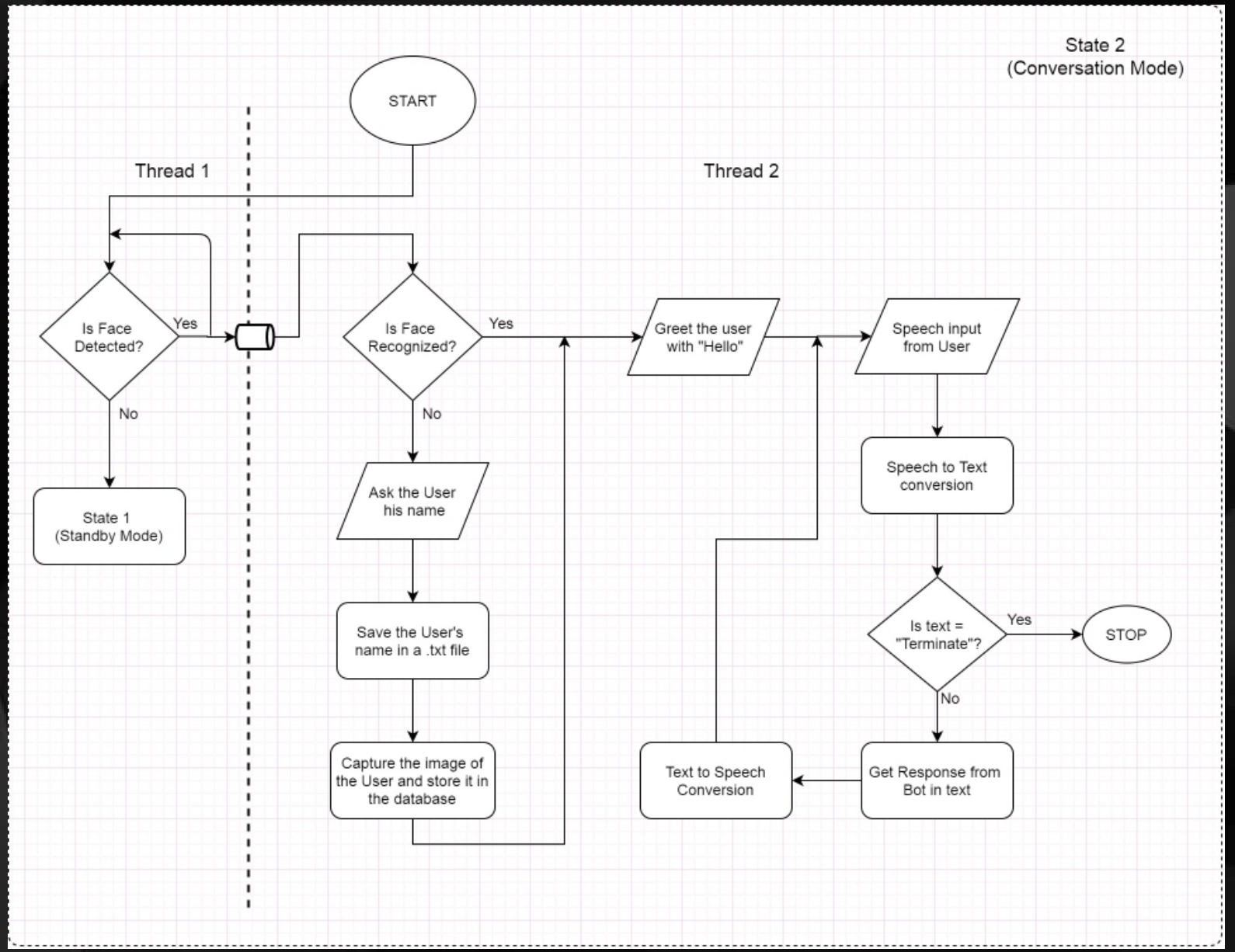
The second state is called Conversation Mode. In this state, there is a person in front of the bot and the bot converses with that person until the person either says Goodbye or he moves away from the screen.

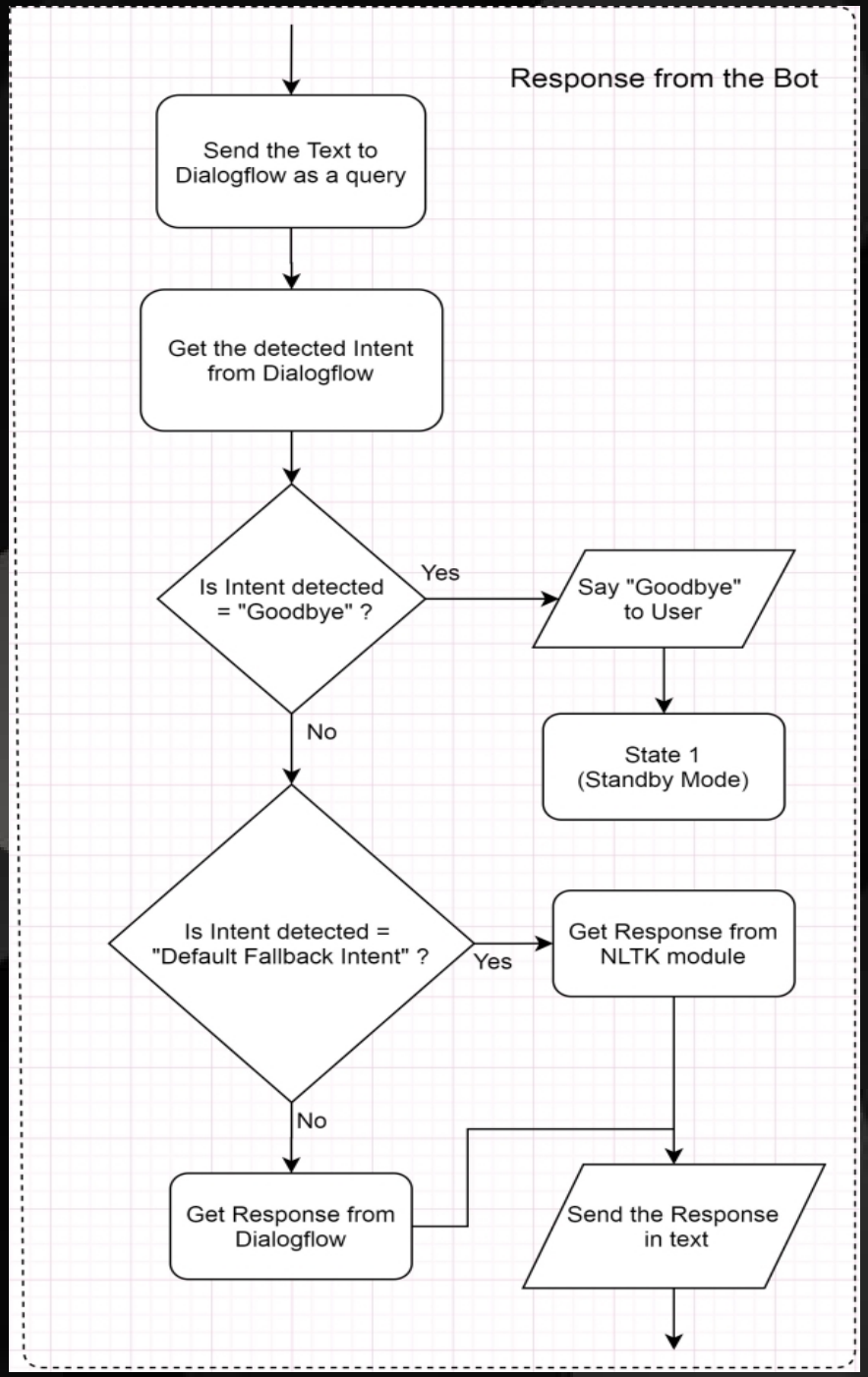
The reason for using the concept of states is because it makes the flow of the program error free and makes programming easier.

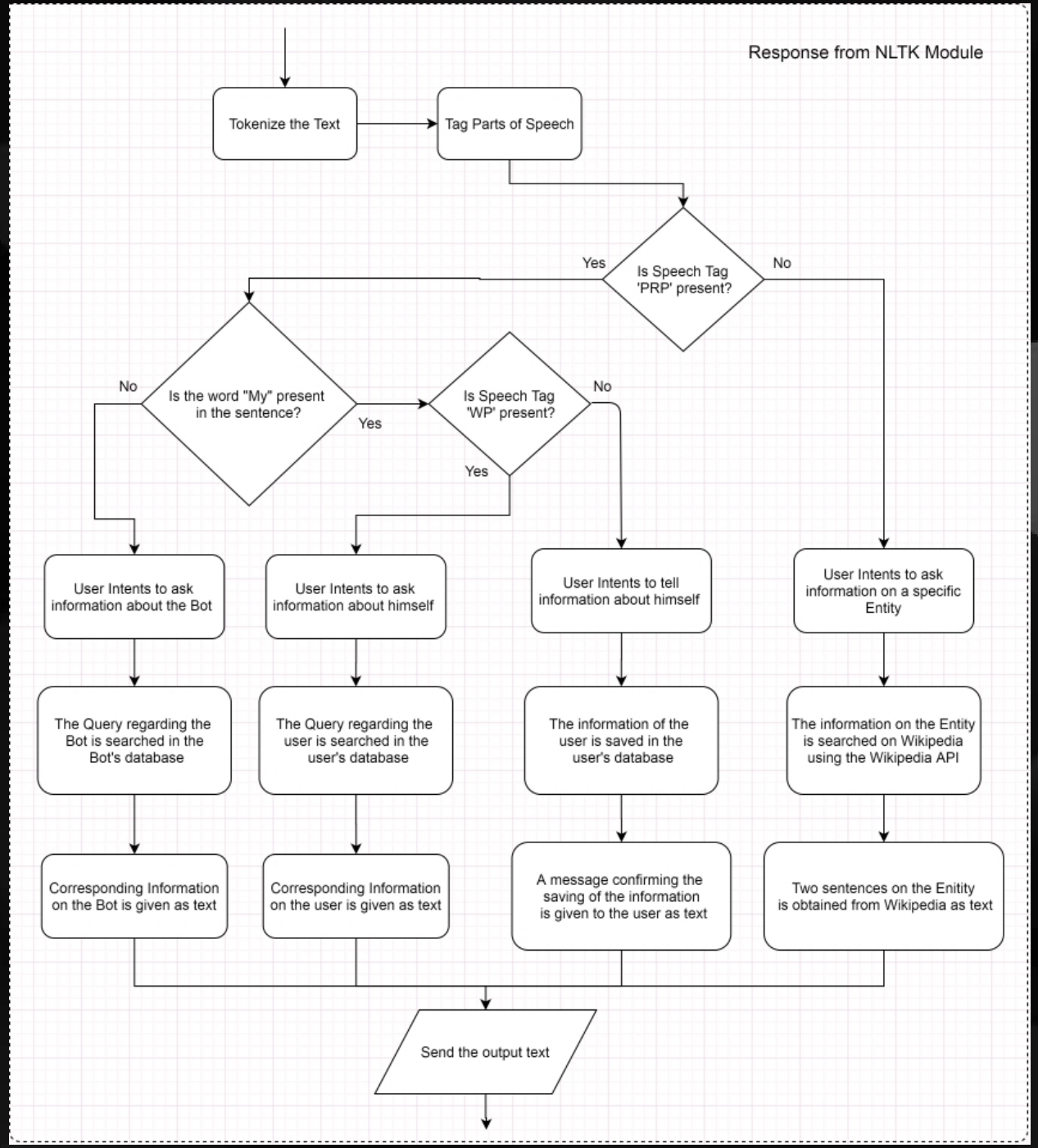


Implementation Details






Response from the Bot
Send the Text to Dialogflow as a query
Get the detected Intent from Dialogflow
Is Intent detected = "Goodbye" ?
Yes
Say "Goodbye" to User
State 1 (Standby Mode)
No
Is Intent detected = "Default Fallback Intent" ?
Yes
Get Response from NLTK module
No
Get Response from Dialogflow
Send the Response in text


Response from NLTK Module
Tokenize the Text
Tag Parts of Speech
Is Speech Tag 'PRP' present?
Yes
No
Is the word "My" present in the sentence?
No
Yes
Is Speech Tag 'WP' present?
No
Yes
User Intents to ask information about the Bot
User Intents to ask information about himself
User Intents to tell information about himself
User Intents to ask information on a specific Entity
The Query regarding the Bot is searched in the Bot's database
The Query regarding the user is searched in the user's database
The information of the user is saved in the user's database
The information on the Entity is searched on Wikipedia using the Wikipedia API
Corresponding Information on the Bot is given as text
Corresponding Information on the user is given as text
A message confirming the saving of the information is given to the user as text
Two sentences on the Enitity is obtained from Wikipedia as text
Send the output text

Conclusion

This Application can successfully recognize people through face recognition and can converse with them in speech in English language.

By deploying the Face Detection module in parallel to the NLP module makes the overall functioning of the Bot more stable and fast.

This Application can be interfaced with Humanoid Robots and Androids for a more realistic conversational experience.

The Application can be improved by adding more functionalities such as setting an Alarm, scheduling events in Calendar, etc.

Demo