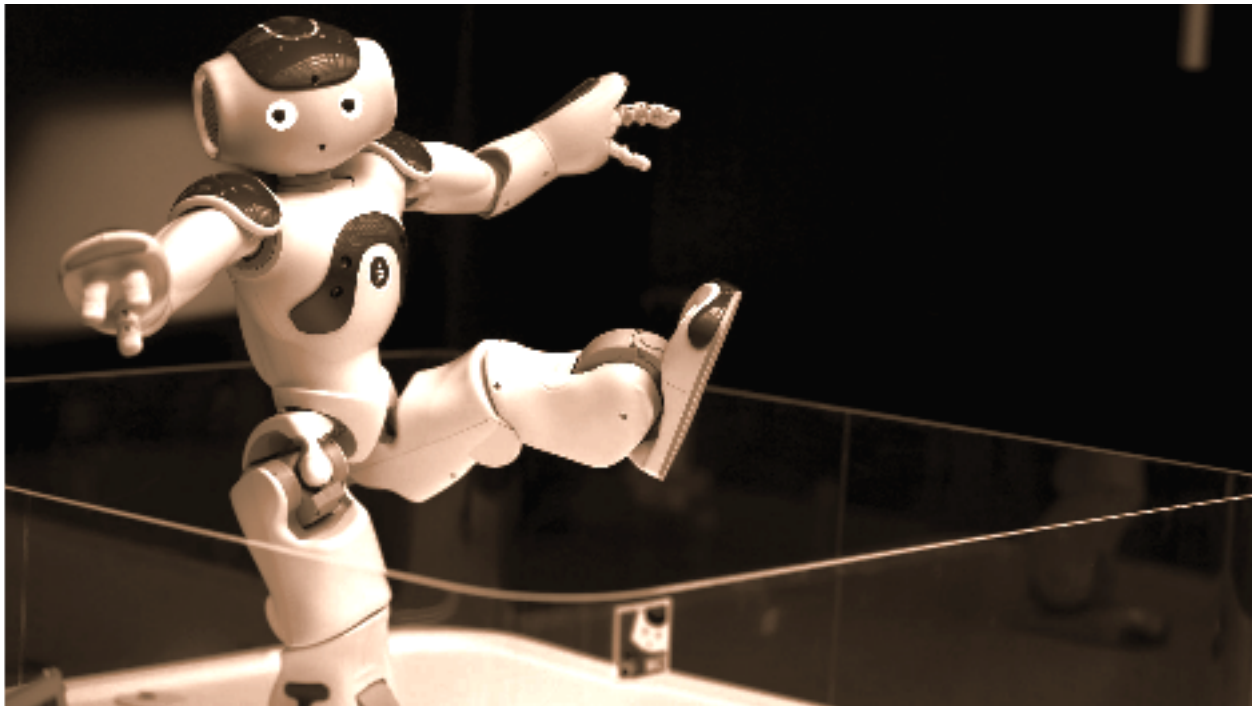
SPEAK NAO®

*A GUI Based Application for **Telepresence***



Network Embedded Applications

A Project Report

17.04.2019

Birla Institute of Technology and Science, Pilani
Rajasthan

Aman Sharma
2018H1030137P

Sachin PC
2018H1030140P

Harsh Bhojani
2018H1030128P

INTRODUCTION

Nao is an action-packed humanoid robot that stands 58 centimeters tall. It is a personal robot armed with a multitude of sensors, motors, cameras and an intuitive AI platform open to developers. Whether it's being used in a starring role for the "Robotics" show company or as a teacher companion in special education classrooms across the world, NAO is poised to interact with humans on a level never seen before. With 25 degrees of freedom, four directional microphones, and a highly sophisticated operating system, NAO is only limited by the vision of its developers. Designed mainly for education and research, Aldebaran Robotics created NAO as a way to personalize the learning experience. Equipped with an NAOqi 2.0 operating system and 30% better battery performance, Aldebaran's fifth edition humanoid can engage with users for up to 1 hour and 30 minutes.

Telepresence refers to a set of technologies which allow a person to feel as if they were present, to give the appearance of being present, or to have an effect, via telerobotics, at a place other than their true location. Telepresence requires that the users' senses be provided with such stimuli as to give the feeling of being in that other location. Additionally, users may be given the ability to affect the remote location. In this case, the user's position, movements, actions, voice, etc. may be sensed, transmitted and duplicated in the remote location to bring about this effect. Therefore information may be traveling in both directions between the user and the remote location.

OBJECTIVE

Our objective was to use the APIs and capabilities of Nao for Telepresence. We chose to build an application to enable voice based one-one communication between users via Nao.

THE APPLICATION

We used Python + Tkinter to create the user interface for our application.

Following are the different functions that are implemented:-

a. **Nao Say** Connection acknowledgment from Nao, it speaks if connection is established and running
b. **Say Something** This mode allows a person speak which is then converted to Text and sent to Nao. The bot then replicates the same, in its voice.
c. **Walkie-Talkie** This makes Nao record the audio at its end for a variable amount of time until it's touched which is taken as a cue for the person has stopped speaking. The code then plays the audio as user's side mimicking a Walkie-Talkie.
d. **Voice Streaming** The voice streaming feature allows the user to listen to whatever Nao's listening in its environment live from comfort of his/her room.

e. **Human Voice** Is an extension of walkie-talkie feature. The Nao plays the voice recorded at user's end for scenarios requiring human-voice at Nao's end for example when languages other than English are used.

USE CASES

- **Hospitality** : The bot can be used for various tasks involving interaction with guests. For example : Taking orders, resolving issues, handling enquiries.
- **Remote Lectures** : We can use the bot to stream live lectures by professors at one end of the world to classrooms in other end.
- **Classrooms:** The bot can be an interesting learning tool for small kids who can learn to speak using Bot's speech transcription feature.
- **Miscellaneous :** A person can use the bot as proxy whenever he/she could not be physically present at a location.

CONCLUSION

The Nao is an amazing piece of work. It is just like a blank slate with endless possibilities. We only used it's ALAudioDevice api for our audio related task, however we can easily build smart applications that encompass Audio, Video, Motion etc in coordination to enable Nao for more sophisticated Telepresence applications.

REFERENCES

1. https://www.softbankrobotics.com/emea/en/nao
2. https://en.wikipedia.org/wiki/Nao_(robot)
3. https://www.telegraph.co.uk/travel/hotels/articles/hotel-robot-room-service/
4. https://www.interesly.com/stop-worrying-love-robots/

APPENDIX

SPECIFICATION

Height	58 centimetres (23 in)
Weight	5.5 kilograms (12 lb)
Power supply	lithium battery providing 48.6 Wh
Autonomy	90 minutes (active use)
Degrees of freedom	25
CPU	Intel Atom @ 1.91 GHz
Built-in OS	NAOqi 2.8 (Linux-based)
Compatible OS	Windows, Mac OS, Linux
Programming languages	C++, Python, Java, MATLAB, Urbi, C, .Net
Simulation environment	Webots
Sensors	Two HD cameras, four microphones, sonar rangefinder, two infrared emitters and receivers, inertial board, nine tactile sensors, eight pressure sensors
Connectivity	Ethernet, Wi-Fi

CODE

```
def mode1():
    setParameter(tts, 100, 1)
    saySomething(tts, "HI! I'm connected")
```

```
def mode2():
    global recognizer, microphone
    response = recognize_speech_from_mic(recognizer,microphone)
    setParameter(tts, 90, 1)
    if(response["transcription"] != None):
        saySomething(tts, str(response["transcription"]))
    else:
        saySomething(tts, str(response["error"]))
```

```
def mode3():
    saySomething(tts, "Do you have any questions?")
    startRecording(record)
    ssh = paramiko.SSHClient()
    ssh.load_host_keys(os.path.expanduser(os.path.join("~", ".ssh", "known_hosts")))
    ssh.connect("10.42.0.18", username="nao", password="Naonao123*")
    sftp = ssh.open_sftp()
    sftp.get('/home/nao/record.wav', '/home/seekr/Desktop/NAO/recordTest.wav')
    sftp.close()
    ssh.close()
    playsound('/home/seekr/Desktop/NAO/recordTest.wav')
```

```
def mode4():
    saySomething(tts, "I'm connecting you to Gods")
    event_of_streaming()
```

```
def mode5():
    FORMAT = pyaudio.paInt16
    CHANNELS = 2
    RATE = 44100
    CHUNK = 1024
    RECORD_SECONDS = 10
    WAVE_OUTPUT_FILENAME = "voice_human.wav"
    audio = pyaudio.PyAudio()
    # start Recording
    stream = audio.open(format=FORMAT, channels=CHANNELS,
                        rate=RATE, input=True,
                        frames_per_buffer=CHUNK)
    print("recording...")
    frames = []
    for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
        data = stream.read(CHUNK)
        frames.append(data)
    print("finished recording")
    # stop Recording
    stream.stop_stream()
    stream.close()
    audio.terminate()

    waveFile = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(FORMAT))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(frames))
    waveFile.close()
```

```
    ssh = paramiko.SSHClient()
    ssh.load_host_keys(os.path.expanduser(os.path.join("~", ".ssh", "known_hosts")))
    ssh.connect("10.42.0.18", username="nao", password="Naonao123*")
    sftp = ssh.open_sftp()
    sftp.put('/home/seekr/Desktop/NAO/all_functions/voice_human.wav', '/home/nao/voice.wav')
    sftp.close()
    ssh.close()

    record_path = '/home/nao/voice.wav'
    saySomething(tts, "now next voice is of human!")
    startRecording(record)
    # ----------> playing the recorded file <----------
    fileID = aup.playFile(record_path, 0.7, 0)
```

```
def startRecording(record):
    print('start recording...')
    record_path = '/home/nao/record.wav'
    record.startMicrophonesRecording(record_path, 'wav', 16000, (0,0,1,0))
    time.sleep(5)
    record.stopMicrophonesRecording()
    print 'record over'

def event_of_streaming():
    ssh = paramiko.SSHClient()
    ssh.load_host_keys(os.path.expanduser(os.path.join("~", ".ssh", "known_hosts")))
    ssh.connect("10.42.0.18", username="nao", password="Naonao123*")
    command = "gst-launch-0.10 pulsesrc ! audioconvert ! vorbisenc ! oggmux ! tcpserversink port=5678"
    ssh.exec_command(command)
    os.system("vlc tcp://10.42.0.18:5678/")
    time.sleep(10)
    ssh.send('\x03')
    ssh.close()
```

```
def recognize_speech_from_mic(recognizer, microphone):
    # check that recognizer and microphone arguments are appropriate type
    if not isinstance(recognizer, sr.Recognizer):
        raise TypeError("`recognizer` must be `Recognizer` instance")

    if not isinstance(microphone, sr.Microphone):
        raise TypeError("`microphone` must be `Microphone` instance")
    # adjust the recognizer sensitivity to ambient noise and record audio from the microphone
    print("Start Speaking")
    with microphone as source:
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)
        print(type(audio))
    # set up the response object
    response = {
        "success": True,
        "error": None,
        "transcription": None
    }
    # try recognizing the speech in the recording
    # if a RequestError or UnknownValueError exception is caught,update the response object accordingly
    try:
        print("HI")
        print(type(audio))
        response["transcription"] = recognizer.recognize_google(audio)
        #response["transcription"] = recognizer.recognize_google(audio)
        #response["transcription"]= response["transcription"].encode("ascii")
        print(response["transcription"])
    except sr.RequestError:
        # API was unreachable or unresponsive
        response["success"] = False
        response["error"] = "API unavailable"
    except sr.UnknownValueError:
        # speech was unintelligible
        response["error"] = "Unable to recognize speech"
    return response
```