



DEGREE PROJECT IN ELECTRICAL ENGINEERING,  
SECOND CYCLE, 60 CREDITS  
*STOCKHOLM, SWEDEN 2018*

# **Developing a voice-controlled home-assisting system for KTH Live-in Labs**

**SHREYANS MALOO**

# Contents

<b>Acknowledgments</b>	<b>4</b>
<b>Abstract</b>	<b>5</b>
<b>Sammanfattning</b>	<b>6</b>
<b>List of Figures</b>	<b>9</b>
<b>List of Tables</b>	<b>10</b>
<b>1 Introduction</b>	<b>11</b>
1.1 Outline . . . . .	11
1.2 Problem Statement . . . . .	13
<b>2 Methodology</b>	<b>14</b>
<b>3 Background</b>	<b>15</b>
3.1 Speech to Text Services . . . . .	15
3.2 Communication Protocols . . . . .	20
3.3 Ethics and Society . . . . .	23
<b>4 Design</b>	<b>24</b>
4.1 Basic Model . . . . .	24
4.2 Map of Alternatives . . . . .	26
4.2.1 Voice Command . . . . .	26
4.2.2 Speech to Text (STT) Service . . . . .	27
4.2.3 Logic Engine . . . . .	27
4.2.4 Communication Protocol . . . . .	27
4.2.5 Final Output . . . . .	28

---

4.3	Failure Factors . . . . .	29
4.3.1	Speech to Text . . . . .	29
4.3.2	Text to Keywords . . . . .	31
4.3.3	Keyword to action . . . . .	32
<b>5</b>	<b>Implementation</b>	<b>34</b>
5.1	Raspberry Pi . . . . .	34
5.1.1	What is Raspberry Pi? . . . . .	34
5.1.2	Setting up the Pi . . . . .	35
5.2	IBM Bluemix . . . . .	44
5.2.1	Getting started with Bluemix . . . . .	44
5.2.2	Linking our Pi to our device in Bluemix . . . . .	48
5.3	Node-RED . . . . .	49
5.3.1	Setting up Node-RED on our Pi . . . . .	49
5.4	Prototype 1 . . . . .	51
5.5	Prototype 2 . . . . .	65
5.6	Prototype 3 . . . . .	79
5.6.1	Conversation service . . . . .	80
5.6.2	Node-RED . . . . .	91
<b>6</b>	<b>Results and Discussion</b>	<b>95</b>
<b>7</b>	<b>Conclusion and Future Work</b>	<b>99</b>

# Acknowledgments

I would like to reserve this space to thank my supervisor Elena Malakhatka for her endless support and dedication and for the brainstorming sessions that offered me guidance throughout all the stages of the project.

I would also like to express my gratitude towards my thesis examiner Prof. Carlo Fischione for the subject, the guidance and the opportunity to work in this field at the KTH Live-in Lab.

Most of all, I would like to thank my family and to all my friends for their support.

# Abstract

The following master thesis is conducted on behalf of KTH Royal Institute of Technology and KTH Live-in Lab with the purpose of developing a voice-controlled home-assisting system for the KTH Live-in Lab. The lab is being designed to serve as a testbed for products and services that can be tested and verified within an optimal space which can simulate a real-life usage. Being designed as a bridge between industry and academia, it aims to create a greater ease to which new products are tested and are researched while involving KTH students in the process. Having innovation at its core the KTH Live-in Lab needs a mode of communication between the user/occupant and the appliances in the space. That is why this thesis proposes to design a voice-controlled system that can control the appliances and execute the commands provided by the user. The system will be created around a Speech to text service and improving its performance through various modifications/integrations. The solution will be installed in the KTH Live-in Lab and integrated with the central controller once the sensor placement and controller installation is done.

To make the system more robust and accurate, a new variable called, “Failure Factors” were defined for a voice-controlled system. The prototypes were designed and improved with these factors as a basis. The main aim of the project is to make a system capable of handling a set of pre-defined simple commands. For testing purpose, only 3 appliances were considered – light, heater and music. Also, the output is observed on LEDs rather than on real appliances for the testing. These limitations were adapted to keep our focus on the prime motive of this project and that was to make the voice-recognition as consistent and accurate as possible. Future work will consist of making the system capable of handling complex user commands and having an active feedback mechanism such that the user can have conversation with the system.

# Sammanfattning

Följande magisteruppsats utförs på uppdrag av KTH Royal Institute of Technology och KTH Live-in Lab med syfte att utveckla ett röststyrt hemhjälpssystem för KTH Live-in Lab. Labbet är utformat för att fungera som testbädd för produkter och tjänster som kan testas och verifieras inom ett optimalt utrymme som kan simulera en verklig användning. Att vara utformad som en bro mellan industri och akademi, syftar det till att skapa en större lätthet för vilka nya produkter som testas och undersöks när de involverar KTH-studenter i processen. KTH Live-in Lab har en innovation som är kärnan i ett kommunikationsläge mellan användaren / passageraren och apparaterna i utrymmet. Det är därför som denna avhandling föreslår att designa ett röststyrt system som kan styra apparaterna och utföra kommandon som tillhandahålls av användaren. Systemet skapas runt ett tal till texttjänsten och förbättrar dess prestanda genom olika modifikationer / integreringar. Lösningen kommer att installeras i KTH Live-in Lab och integreras med centralstyrenheten när sensorplaceringen och kontrollenheten är klar.

För att göra systemet mer robust och noggrant definierades en ny variabel, "Failure Factors" för ett röststyrt system. Prototyperna utformades och förbättras med dessa faktorer som grund. Huvudsyftet med projektet är att skapa ett system som kan hantera en uppsättning fördefinierade enkla kommandon. För teständamål betraktades endast 3 apparater - ljus, värmare och musik. Dessutom observeras utsignalen på lysdioder i stället för på verkliga apparater för testningen. Dessa begränsningar var anpassade för att hålla fokus på huvudmotivet för projektet och det var att göra röstigenkänningen så konsekvent och korrekt som möjligt. Framtida arbete kommer att bestå i att systemet kan hantera komplexa användarkommandon och ha en aktiv återkopplingsmekanism så att användaren kan ha konversation med systemet.

# List of Figures

2.1	The Methodology Graph . . . . .	14
4.1	The Basic Model . . . . .	24
4.2	The Map of Alternatives . . . . .	26
4.3	The Final flowchart to be implemented . . . . .	28
5.1	Screenshot of the SDFormatter V4.0 . . . . .	36
5.2	Install screen we get first time we boot the Pi . . . . .	37
5.3	Configuration Screen of PuTTY . . . . .	40
5.4	Security warning generated by PuTTY . . . . .	41
5.5	Screenshot of vncserver command run via SSH (PuTTY) . . . . .	42
5.6	VNC Viewer welcome screen . . . . .	43
5.7	Screenshot of the desktop environment as visible on the remote device . . . . .	43
5.8	Screenshot showing the credentials for a Bluemix account . . . . .	44
5.9	Screenshot of the welcome page for Bluemix service . . . . .	45
5.10	Adding a new device . . . . .	46
5.11	Creating a device type named raspberryPi . . . . .	46
5.12	Adding new device, using the created device type . . . . .	47
5.13	Screenshot of the device information for device01 . . . . .	48
5.14	Running node-red-start . . . . .	50
5.15	Screenshot of the page with service-credentials . . . . .	53
5.16	Speech to text node details . . . . .	55
5.17	Editing the switch node for tasks . . . . .	56
5.18	Edit switch node for appliances . . . . .	57
5.19	The Node-RED flow for Prototype 1 (version 1.0) . . . . .	58
5.20	STT output for commands 1-4 . . . . .	60
5.21	STT output for commands 5-8 . . . . .	60

---

5.22	STT output for commands 9-12 . . . . .	61
5.23	Modified Task switch node . . . . .	62
5.24	Modified appliance on (and appliance off) . . . . .	63
5.25	The Node-RED flow for Prototype 1 (version 2) . . . . .	63
5.26	Flow for creating custom language model . . . . .	65
5.27	Create Customization node . . . . .	66
5.28	List customization node . . . . .	66
5.29	Debug node . . . . .	67
5.30	Get Customization node . . . . .	68
5.31	Node-RED flow for creating custom language model and adding data from corpora to it . . . . .	68
5.32	Add Corpus Node . . . . .	69
5.33	Get Corpora Node . . . . .	70
5.34	Train Node . . . . .	70
5.35	Node-RED flow for adding words and data to custom language model . . . . .	72
5.36	Add words node . . . . .	73
5.37	Node-RED flow for testing customized vs non-customized model	74
5.38	Non-Customized Speech to text node . . . . .	75
5.39	Customized Speech to text node . . . . .	75
5.40	Non-Customized vs Customized example 1 . . . . .	77
5.41	Non-Customized vs Customized example 2 . . . . .	77
5.42	Non-Customized vs Customized example 3 . . . . .	78
5.43	Performance after adding words . . . . .	78
5.44	Conversation service intents . . . . .	80
5.45	Conversation service entities . . . . .	81
5.46	Contents of the appliances entity . . . . .	81
5.47	The conversation service dialog . . . . .	82
5.48	Contents of "Hello" node . . . . .	83
5.49	Contents of "other appliances" node . . . . .	84
5.50	Contents of "turn on appliance" node . . . . .	85
5.51	Contents of "turn on" node . . . . .	86
5.52	Contents of "All appliances on" node . . . . .	87
5.53	Contents of "appliance on" node . . . . .	87
5.54	Contents of "Appliance" node . . . . .	88
5.55	Contents of "Goodbye" node . . . . .	89
5.56	Contents of "Anything else" node . . . . .	89
5.57	Snapshot of a dialog with the conversation service . . . . .	90



5.58 Node-RED flow for Prototype 3 . . . . .	91
5.59 Conversation service Node . . . . .	92
5.60 Watson Conversation Workspace details . . . . .	92
5.61 Conversation service handling multiple appliances . . . . .	93
5.62 Conversation service handling misinterpretations from customized STT . . . . .	94

# List of Tables

4.1	Summarized Failure Factors . . . . .	33
5.1	Performance - Prototype 1 (version 1) . . . . .	59
5.2	Performance - Prototype 1 (version 2) . . . . .	64
5.3	Performance - Prototype 2 . . . . .	79
5.4	Performance - Prototype 3 . . . . .	94
6.1	Evaluation - Prototype 1 . . . . .	96
6.2	Evaluation - Prototype 2 . . . . .	97
6.3	Evaluation - Prototype 3 . . . . .	98

# Chapter 1

## Introduction

In this chapter, we will get an overview of the factors that will come to shape the result of this thesis such as the present status of the home automation sector, and the motivation behind this thesis. It also contains a description of the problem and the steps needed to be taken in the attempt of solving it.

### 1.1 Outline

Home automation is evolving into something bigger than a connection between autonomous devices. It is moving towards systems and processes that are becoming more intelligent and can communicate with people. This whole relationship of us users with an autonomous system was foreseen by Mark Weiser [13], in 1991, where he predicted computers to increasingly enable the integration of simple objects, such as air conditioners, light switches and more, in unobtrusive way in the user's life. This evolution can be partly explained by the decrease in hardware production costs, the evolution in scientific research and the increase of economic advantages of the sector. The increase in investments in this sector has made it possible for automation to expand from just being restricted to high-value domains like industry and military applications, and enter our day to day life through access control devices, intelligent parking systems and smartphone applications. The focus being to provide well-being, to enhance the quality of life and to improve health and security services. Today, automation systems can combine information from several distributed sensors and actuators and use it. This environment, working with such huge amount of data, has led to new con-

cepts and research fields, like Big Data, IoT, Artificial Intelligence and others. Further, automation is evolving to new application domains like smart cities, transport, agriculture, and intelligent health. Coming to one of the main topics for this report, smart homes; it is a domain comprising familiar applications for users to merge all these new concepts and technology. The smart home automation is characterized by an infrastructure that enables intelligent networking of devices and appliances that use various wireless and wired technologies to provide seamless integration, which facilitate ease of use of house systems while creating a personalised and safe home space [14]. The smart home appliance market is projected to grow from 40 million dollars in 2012 to 26 billion dollars in 2019 [15]. This business opportunity is attracting several companies including General Electric, Cisco, Google and others. But no company has yet succeeded to launch home automation as a popular technology, despite the disparate activities in this industry. Some of the reasons would be [12], (a) cost: due to low demand, the existing systems have been expensive, (b) difficult to install: professionals needed to install and configure the system, (c) difficult to use: not so user-friendly control interfaces, (d) vendor dependency: different company appliances are not compatible with same system, (e) less functionality: most of the system are only able to monitor or control some functions, (f) not customized: very little possibility to customize with the needs of the user, (g) not secure: there has always been security issues and multiple-user problems related to such home automation systems. After looking at the bright prospects and the bottlenecks for the development of this sector, we will be trying to work towards developing an affordable, user-friendly, customizable home-assisting system in this report. With computers becoming more and more cheaper, we can ensure the affordability of our system by using a cheap yet powerful computer in Raspberry Pi. For making it user-friendly we have opted to go for a voice-controlled setup. The advances in speech technology and computing power over the last decade or so has created an increase of interest in the practical application of speech recognition. Speech being the primary mode of communication among humans, makes it more user friendly, faster and less cumbersome mean of communicating than through keyboards or other devices. For customizable, we will be discussing more as we proceed in this report.

## 1.2 Problem Statement

The problem statement being tackled in this thesis is, to develop a voice controlled home assisting system. Now, there are already several such products available in the market. But, the features that will make this project unique from all of them are as follows,

- This setup will be developed in-house and customized as per the application at KTH Live-in labs.
- A new variable named “Failure factor” will be established and using this, the efficiency of the developed system can be improved.
- Last but not the least, the place where this product will be used in the future, KTH Live-in Labs, is an exclusive project exempted from building permit regulations, providing unique possibilities for testing. The data collected by our setup will be used to come up with innovative living services. The exemption allows for wider range of data to be collected and processed than a commercial home-assisting setup.

# Chapter 2

## Methodology



Figure 2.1: The Methodology Graph

The first step is doing the market survey of the existing systems. This step involves doing a literature review of various studies done in the past on the speech to text services and the communication protocols available in the market.

The next step involves designing the basic model of our system. We use the findings from the market survey to generate a map of alternatives, comprising multiple options for each step in the basic model. Also, we select a relevant path for our project in this step; the one which will be used for implementation.

In the next step, we do some more literature review and define a new variable, “Failure factor” which represents the factors that may reduce the performance of our system.

Then we start with the implementation of our first prototype. We observe the results and then try to improve the design through a more advanced prototype 2 and 3, which is implemented in the next steps.

Finally, we analyse and evaluate all the results obtained. We also discuss the possibilities of improvement and implementing the idea in the real conditions at KTH Live-in Labs in the future.

# Chapter 3

## Background

When designing a voice-controlled home assisting system, there are 2 major areas where we have a wide range of options to choose from in the market today – the speech to text service and the communication protocol.

### 3.1 Speech to Text Services

A study was done on the available STT services in the market today. Based on this study, we will choose the one best suited for our application. The list along with a brief description of each is as follows,

- Based on the official web page of Google Cloud Speech API [4], it enables developers to convert audio to text by applying powerful neural network models in an easy to use API. It claims to recognize over 80 languages and variants and being able to successfully handle noisy audio from a variety of environments. Last but not the least, the Speech API supports any device that can send a REST or gRPC request including phones, PCs, tablets and IoT devices (e.g., cars, TVs, speakers). On the downside, it is not free to use this service, as the Cloud Speech API is priced \$ 0.006 per 15 seconds of audio processed after a 60-minute free tier on personal devices (for embedded devices as in our case, we need to contact them for approval and pricing). Unfortunately, with this solution one will not be able to change the “OK, Google” wake-up word.
- Pocketsphinx is an open source speech decoder developed under the

CMU Sphinx Project. It is quite fast and has been designed to work well on mobile operating systems such as Android as well as embedded systems (like Raspberry Pi). The advantage of using Pocketsphinx is that the speech recognition is performed offline, which means we don't need an active Internet connection. However, the recognition rate is on the poorer side. The most difficult issue mentioned above is: setting a custom wake-up word. Intuitively, it is better if we can avoid using a cloud service for this feature because the continuous connection with the Internet will negatively affect battery life. Instead of cloud services, we use PocketSphinx - an open source solution for Android. Thus, we can use PocketSphinx as an offline solution only for keyword recognition. Later, we'll require some cloud service to handle the request.

Pros:

- With Sphinx, it is possible to set a custom wake-up word.
- Sphinx works offline (lower battery consumption).

Cons:

- PocketSphinx is not accurate enough to get the effect we want to achieve.
  - There is a pause after Sphinx recognizes a keyword and launches the cloud service.
- A&T STT was developed by AT&T. The recognition rate is good, but it needs an active connection to work, just like Google STT. AT&T speech recognition REST API features:
    - Registration is required
    - Required “Premium Access” payment is \$99/year + Usage fees to access automatic speech recognition
    - Per documentation usage limitations is 1 request per second
  - Julius is a high-performance, open source speech-recognition engine. It does not need an active Internet connection, like Pocketsphinx. It is quite complicated to use because it requires the user to train their own acoustic models.



- Wit.ai STT is a cloud-based service provided to users. Like AT&T and Google STT, it requires an active Internet connection to work.
  - WIT is more about NLP (Natural Language Processing) than about plain-speech recognition.
  - Main focus, besides speech recognition, is to parse out spoken phrases and extract valuable information (e.g., some voice command). The goal is to have the system “understand” voice. For instance, when the user says, “Hi, robot! Please play me a Christmas song”, it should start playing “Jingle Bells.”
  - Github account is all that is needed to access WIT REST API
  - No account usage limitation

How wit.ai works:

- Provide a sentence we want the app to understand. Then either select an existing intent from the Community or create our own.
- Send text or stream audio to the API. In return, Wit.ai gets structured information.
- Wit.ai learns from usage and helps improve configurations.

Pros:

- Returns JSON.
- Already has many built-in intents.
- Ability to learn from the user.

Cons:

- Not stable enough. During the test, it was shut down after 30 sec [3].
  - Not so comfortable to use.
- Initially known as IBM’s Jeopardy winning AI service, Watson came forth for commercial applications at the beginning of 2013. With the growth of cloud and IoT, IBM Bluemix was launched as the go-to platform for all related services. Although speech analysis capabilities

were only added at the beginning of 2015, early research on Automatic Speech Recognition at IBM dates all the way back to the 60's. Currently, Watson's speech-to-text services provide transcriptions in 9 different languages. As mentioned in their webpage [5], recently it reached a new industry record of 5.5 percent word error rate (with human parity being lower than what any STT has yet achieved — at 5.1 percent). IBM STT was developed by IBM and is a part of the Watson division. It requires an active Internet connection to work. IBM Speech recognition REST API features:

- Registration in Bluemix is required
- Usage limitations of first 1000 minutes per month
- The American Nuance is a top-notch company in the field of speech recognition and synthesis. Coming forth with an already huge amount of industry specific solutions, they recently launched Dragon Drive, which is already implemented in some BMW vehicles. The service that is referred to here, however, is the Nuance Mix service. Launched at the end of 2015, the platform comes forth as the IoT and developer solution for their speech services, currently providing 18 different languages for speech-to-text analysis. Nuance provides many voice recognition and natural language processing services. It has a ready solution for mobile speech-recognition: VoCon Hybrid, which could solve one of our most difficult issue - custom keyword recognition. Pros:
  - A key advantage of this technology is that always-listening mode with keyword activation removes the need for a push-to-talk button.
  - All-inclusive main menu. Enables all commands to be spoken in a single utterance on the main menu.

Cons:

- Closed technology. It is not an open source API - if we want to use it in our project we need to contact Nuance and ask for samples to test it.
- Complicated documentation and set-up.
- Usage limitations of 5,000 requests per day

- Alexa Voice Service (AVS) is a cloud speech-recognition service from Amazon. It is used in Amazon's Echo. Here's how it works: "Alexa" is the wake-up word and starts the conversation. Our service [3] gets called when customers use our invocation name, such as: "Alexa, ask Lucy to say hello world." This example is a simple command-oriented one. ASK also supports more sophisticated multi-command dialogues and parameter passing. The above example would work like this:
  - "Alexa" is the wake word that starts the conversation.
  - "Ask...to" is one of the supported phrases for requesting a service.
  - "Lucy" is the invocation name that identifies the skill we want (in our case it's the name of our app).
  - "Say hello world" is the specific request, question, or command.

#### Pros

- Alexa provides a set of built-in skills and capabilities available for use. Examples of built-in Alexa skills include the ability to answer general knowledge questions, get the current time, provide weather forecast information and query Wikipedia, among others.
- Returns an mp3 with an answer.

#### Cons

- To get custom intents within AVS it is necessary to create, register and test them with the Alexa Skill Kit.
- Complicated documentation.
- To capture the user's utterances, the device needs to have a button to activate the microphone (push-to-talk). According to [3], they contacted Amazon and got information that far-field voice recognition or using a spoken word to trigger activation of the Alexa Voice Service is currently unavailable.
- The Api.ai platform lets developers seamlessly integrate intelligent voice command systems into their products to create consumer-friendly voice-enabled user interfaces. We made a test application [3] using Api.ai and it was closest in quality to Amazon Echo. Pros:

- Easy to implement, clear documentation.
- Ability to learn and adapt (machine learning).
- Complete cloud solution (only without keyword activation): Voice recognition + natural language understanding + text-to-speech.
- Returns a voice answer.
- User friendly.

Cons:

- It has only push-to-talk input.

## 3.2 Communication Protocols

A study was done on the available communication protocols in the market today. A couple of studies ([6] and [7]) on this topic were referred to, to get the basic idea of the communication protocols available. The list along with a brief description of each is as follows,

- X10 has been around for almost 40 years and uses the home wiring to communicate. It was supposed to be superseded by UPB but it didn't happen. Because of its age, the protocol is limited to simple, narrowband instructions and susceptible to electronic interference. This interference can be mitigated with filters though.
- UPB or Universal Powerline Bus, while similar to X10, was intended to be an X10 replacement, given its superior reliability (less susceptible to power line noise and increased range - it can transmit over one mile). The technology uses a home's existing power lines, which reduces costs a bit, to send signals that will control devices both inside and outside the home. One of the disadvantages of this technology is, it is difficult to combine it with the newer wireless technologies such as Wi-Fi, smartphones, etc. What's more, while the platform claims a 99% reliability factor, it offers a relatively low bandwidth, so performance can be slow. It also provides no encryption, meaning it is not quite as secure as wireless. The technical complexity of the system makes for a difficult user-setup experience. And the big downside: There are far fewer UPB-compatible devices than for other technologies.

- One of the most popular of the wireless home automation protocols, Z-Wave runs on the 908.42MHz frequency band. Because this is a much lower band than the one used by most household wireless products (2.4 GHz), it is not affected by their interference and “traffic jams.” A significant advantage of Z-Wave is its interoperability. All Z-Wave devices talk to all other Z-Wave devices, regardless of type, version or brand. Further, the interoperability is backward- and forward-compatible in the Z-Wave ecosystem; that is, Z-Wave products introduced today will work with Z-Wave products from a decade ago and with products in the future (although possibly with some limits on functionality). There are currently over 1,200 different Z-Wave -compatible devices on the market, giving consumers access to a wide range of options when automating their home. The advantage is that the protocol is simple to set up, requires very little power and can use each device as a repeater. The disadvantage is that just like Wi-Fi, some homes suffer from low reception.
- Introduced in 2005, Insteon devices communicate over both power lines and wirelessly, ensuring multiple pathways for messages to travel. Insteon is also X10 compatible, which means that users can add wireless capability to an existing X10 network; doing so can be an effective and cost-efficient way to make a full-blown transition to wireless. There are almost 200 different Insteon-enabled home automation devices available on the market (including the “hub” controllers). Moreover, these devices don’t have to be “enrolled” in the home automation network; they join the network as soon as they’re powered up, simplifying installation. There’s no practical limit to the size of an Insteon network; it’s not unusual to have more than 400 devices in a single installation. On top of all these, its dual-band mesh network turns all powerline-operated devices into repeaters, greatly extending signal range.
- There are myriad similarities between Z-Wave and ZigBee. Like Z-Wave, ZigBee is exclusively a wireless home automation protocol. One of the major drawbacks of this technology is the lack of interoperability between ZigBee devices, which often have difficulty communicating with those from different manufacturers. So, careful consideration should be made in terms of purchasing and product choice. ZigBee is a low-cost, low-power technology, meaning that the battery-operated

devices in a ZigBee network will enjoy a long life. Running on the 802.15.4 wireless communication standard, ZigBee also uses a mesh network structure that provides excellent range and creates rapid communication between ZigBee devices.

- We all know Wi-Fi and use it every day so it needs a very little introduction here. Boasting high bandwidth, Wi-Fi is already pretty much everywhere, so many manufacturers are enthusiastically making smart home devices to work with it. However, there are two key drawbacks: interference and bandwidth issues. If your house is full of Wi-Fi-connected gadgets (TVs, game consoles, laptops, tablets, etc.) then your smart devices must compete for bandwidth and may be slower to respond. Lastly, it is also hungry for power; consequently, battery-operated smart devices such as locks and sensors get drained much sooner than in other wireless environments.
- Bluetooth uses less power than Wi-Fi but has a shorter range. It is also another familiar protocol that we know and use already. The advantages of Bluetooth are interoperability and security. One of the advantages is the high data bandwidth (higher than ZigBee and Z-Wave but lower than Wi-Fi) while sucking up far less power than Wi-Fi. Conversely, it also has a limited range, so for devices that require constant connection—think motion sensors, security systems, etc.—it may not be the ideal platform. However, it has been reported [7] that the newest version of Bluetooth (Bluetooth Low Energy, or BLE) can form mesh networks, greatly extending its range. And with no central hub required, the convenience factor cannot be overlooked.
- Thread is a new wireless protocol for smart household devices. More than 250 devices can be connected on a Thread network and, because most devices meant to be connected to the network are battery-operated, it's very frugal on power. Using the same frequency and radio chips as ZigBee, it is intended to provide a reliable low-power and secure network that makes it simple for people to connect more than 250 devices in the home to each other. They can even be connected to the Cloud for ubiquitous access.

### 3.3 Ethics and Society

Home automation has been a very sensitive topic when it comes to ethics. A few of the areas that are of concern here are,

- Letting a machine collect personal data and analyse it. Consumers are sceptical about the way this data is handled by home-automation companies.
- The more the machine knows about the user, the more is the risk of losing it to someone who might for instance, hack the system.
- When it comes to voice-controlled systems, there is an unethical aspect related to it, about recording personal conversations. Although the companies claim that they only send the voice recording to their cloud platform once it captures the keyword, even though it is continuously recording the sound waiting for the keyword. These ethical issues from the consumer end has been one of the reasons for the lower acceptance of smart-home technologies by the consumer and thus slow growth of the smart home sector.

Coming to the KTH Live-in Labs, this would be a much-needed step stone for the home-automation companies. Being a testbed with some exemptions on the data it can work with, allows companies to try out their new ideas here or even develop ideas based on the research done here. Hence, the whole sector can bloom with the innovations, which was previously hindered by the unavailability of test subjects and test beds in this sector. The result would be a more consumer-centred innovations. Thus, eventually this project will be contributing to the whole home-automation community, which aims in making the society more comfortable and life easier for us human beings.

# Chapter 4

## Design

### 4.1 Basic Model

The basic model for a voice-controlled home-assisting system looks like this,

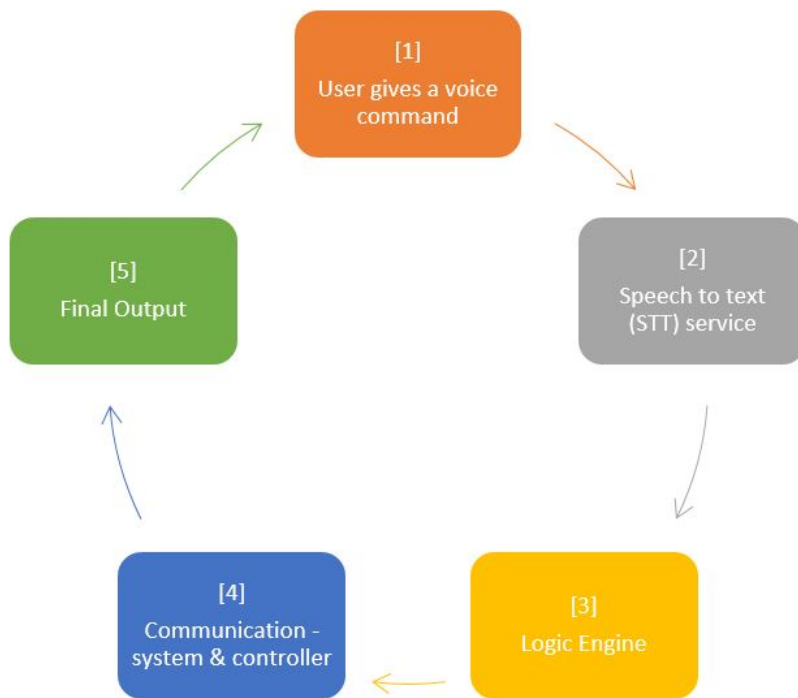


Figure 4.1: The Basic Model



The steps involved are,

- Voice Command: This step involves the user giving a voice command, which is recorded for further processing.
- Speech to text (STT) service: This step involves converting the speech recorded in the previous step into text for further processing.
- Logic Engine: In this step, the text is analyzed and the engine looks for certain keywords to make decisions.
- Communication: The decision made by the logic engine needs to be communicated to the controller, that is what happens in this step.
- Final Output: In this step the controller, once it receives the decision from the logic engine, controls the respective appliance to provide the output to the user. Hence, the loop completes.

## 4.2 Map of Alternatives

Generating a flowchart with the suitable alternatives we have for each step, we get the following,

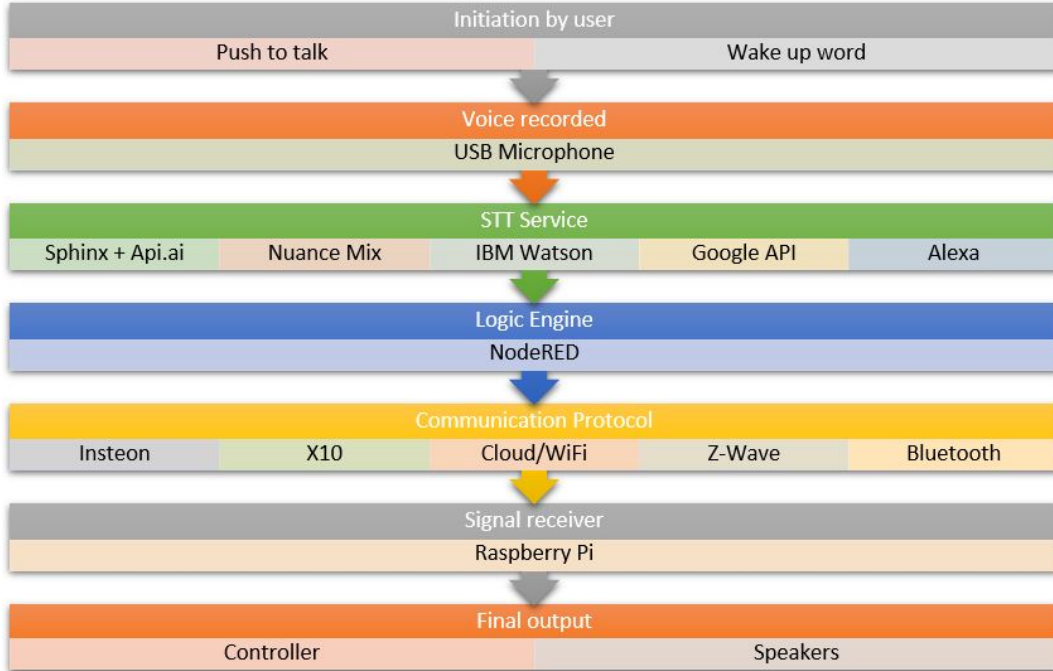


Figure 4.2: The Map of Alternatives

Now, we choose the best path suitable for our application before we start implementing,

### 4.2.1 Voice Command

The voice command from the user can be recorded by any means, be it an external USB microphone or an inbuilt microphone of a laptop. The focus here is to ensure the quality of the recorded voice is good, with least possible noise or disturbances. Hence, using a USB microphone with good noise cancellation would be a logical choice.

Another area of personalization here is to have either a push-to-talk technology or a wake-up word technology. The way they work is, in push-to-talk,

you need to press some button or icon in a GUI to start the recording, it is reliable in the way, it will only record when the user needs it to and there will not be any irrelevant requests transmitting. On the other hand, the wake-up word, is always listening for a keyword, and once it records that, it starts transmitting the information further. The disadvantage with this is, it consumes a lot of battery to always keep listening for a keyword. But then again, this technology gives a higher sense of autonomy when the user does not need to press any button to start the assistant.

The idea will be to first build a system with push-to-talk feature and then move on to wake-up word.

### 4.2.2 Speech to Text (STT) Service

Based on the study done in Background section of this report, I would like to test the IBM Watson STT for our voice-recognition system. Since, it looks promising, have been breaking records in recent times in this area [5], and the researchers at KTH Live-in Labs recommend using IBM Bluemix platform and IBM Watson. It even allows for a higher degree of personalization when compared to all other STT services, which is very important for our application.

### 4.2.3 Logic Engine

There is not a lot of choice when it comes to logic engine, as it will be pretty much decided by our choice of STT service. For instance, if we use IBM Watson STT, the preferable logic engine will be NodeRED, and the code can be executed on an embedded device let's say a Raspberry Pi (tiny and affordable computer).

### 4.2.4 Communication Protocol

Choosing a smart home communication protocol can be tricky business. Obviously, we want one that will support many devices, as well as one that offers the best possible device interoperability (the ability for devices to talk to each other). But there are also other factors to consider, such as power consumption, bandwidth and, of course, cost.

Since, we are moving ahead with the IBM cloud platform and the embedded device, Raspberry Pi supports wireless communication, we will be using

the WiFi communication protocol.

### 4.2.5 Final Output

The ultimate output is quite clear, the appliances. But to reach this, we need to control the controller that handles all the connected appliances. And to control the controller we need an intermediate device that listens to the logic engine and must be obviously compatible with the communication protocol to do that. For instance, going again with the IBM Watson example, we can use a Raspberry Pi where the NodeRED application will be running and which receives data from the IOT cloud, to further send signals to the controller.

For testing purpose, we will be using LEDs receiving signals from the GPIO pins of Raspberry Pi. These LEDs will be representing different appliances, for instance, light, music and heater.

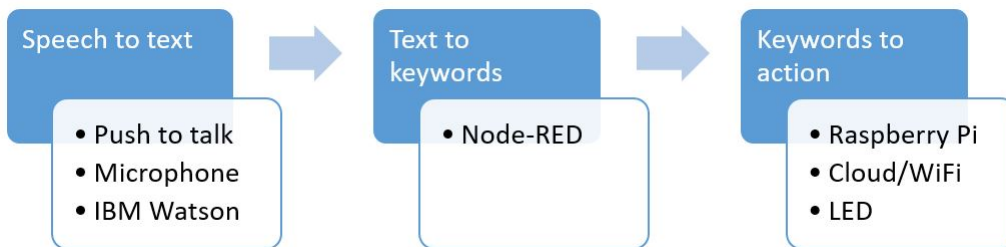


Figure 4.3: The Final flowchart to be implemented

## 4.3 Failure Factors

Here we will be looking at the various factors which can negatively affect the performance of our system. We call these factors, “Failure factors”. The intention is to eventually make our system as immune to these factors as possible.

### 4.3.1 Speech to Text

An Automatic Speech Recognition (ASR) is just like any other machine learning (ML) problem, where the objective is to classify a sound wave into one of the basic units of speech (also called a “class” in ML terminology), such as a word. The factors of failure here could be,

- The problem with human speech is the huge amount of variation that occurs while pronouncing a word. There are several reasons for this variation, namely stress on the vocal chords, environmental conditions and microphone conditions, to mention a few. To capture this variation, ML algorithms such as the hidden Markov model (HMM) along with Gaussian mixture models are used. More recently, deep neural networks (DNN) have been shown to perform better. One way to do ASR is to train ML models for each word. During the training phase, the speech signal is broken down into a set of features (such as Mel frequency cepstral coefficients, or MFCC for short) which are then used to build the model. These models are called acoustic models (AM). When a speech signal should be “recognized” (testing phase), features are again extracted, and are compared against each word model. The signal is assigned to represent the word, which has the highest probability value.
- This way of doing ASR works well for small vocabularies. When the number of words increases, we end up comparing with a very large set of models, which is computationally not feasible. There is another problem of finding enough data to train these models. The word model fails for large vocabulary continuous speech recognition tasks due to the high complexity involved in decoding as well the need for the high amounts of training data. To overcome this problem, we divide words into smaller units called phones. In the English language (and many Indian languages), there are approximately fifty phones that can be combined to make up any word. For example, the word “Hello” can be

broken in to "HH, AH, L, OW". The problem of ASR boils down to recognizing the phone sequence instead of a word. This requires building ML models for every phone. These models are called Monophone models. If one can do a good job of recognizing the phones, a big part of the ASR problem will be solved. Unfortunately, recognizing phones is not an easy task. If we plot the Fourier spectrum of a phone utterance, distinct peaks are visible. The peak frequencies are key indicators of a phone. If a scatter plot of the vowels are plotted, the spread is large and very often overlaps with one another, i.e. no clear boundaries can be drawn to differentiate the vowels. This overlap makes it hard for a ML algorithm to distinguish between phones.

- Even with good phoneme recognition, it is still hard to recognize speech. This is because the word boundaries are not defined beforehand. This causes problems while differentiating phonetically similar sentences. A classic example for such sentences are "Let's wreck a nice beach" and "Let's recognize speech". These sentences are phonetically very similar and the acoustic model can easily confuse between them. Language models (LM) are used in ASR to solve this problem.
- Another factor which bugs an ASR system is an accent. Just like humans, machines too have a hard time understanding the same language with different accents. This is because the classification boundaries previously learnt by a system for a particular accent do not stay constant for other accents. This is the reason why ASR systems often asks for user's location/speaking style (English-Indian, English-US, English-UK, for example) during the configuration process.
- The complexities described so far are part of natural speech. Even with such large complexities, recognizing speech in noiseless environments is generally considered a solved problem. It is the external influence such as noise and echoes which are bigger culprits. Noise and echoes are unavoidable interference while recording audio. Echoes happen due to the reflections of speech energy from surfaces such as walls, mirrors, and tables. This is not much of a problem when a speaker is speaking close to the microphone. But when spoken from a distance, multiple copies of the same signal are reflected and combined at different time delays and intensities. This will result in the stretching of phones across time and will end up corrupting the neighbouring speech information.

This phenomenon is called as smearing. The process of removing the smear is called dereverberation, which is a commonly used technique to address the reverberation problem.

- If the ASR requires internet connection to work, the poor connection or any problem in connection can be a factor of failure. The solution, could be to ensure continuous accessibility to internet or even look for offline options as much as possible but not at the expense of good performance.

### 4.3.2 Text to Keywords

The factors of failure here could be,

- The first factor of failure by the logic engine comes from the STT. If the STT misinterprets the user command, and the keywords provided by the user are not able to either reach the text phase in the correct form or not reach at all. The solution lies in making our STT more robust such that there are minimal misinterpretations.
- Assuming the STT works ideally (words transcribed with 100% accuracy), there still could be reasons for the system to fail; if the user provides fewer or no keywords than required. This can be tackled by having a feedback mechanism that re-starts the loop, i.e. the user is asked to rephrase the request with some keywords present, or in case there are fewer keywords the system could confirm with the user, the most probable inference made based on the provided keywords.
- Another factor of failure could come from the complexity of the user command. For instance, if the user gives multiple commands in the same sentence, like, “Turn on the lights and television”. The logic engine should be equipped to understand and handle such cases where the “turn on” keyword is used only once but two keywords for appliances are mentioned.

### 4.3.3 Keyword to action

The factors of failure here could be,

- If the output signal from the logic engine cannot reach the receiver, due to some communication problem, this could lead to the system failing to perform. The solution could be to firstly, have a stable communication protocol always; secondly, to have an alert mechanism to alert the user if the communication is broken and the command would not be able to reach the intended appliances.
- Now, if the signal reaches the receiver, the failure could occur at the link between the receiver and the controller. We need to ensure this link is intact too, and can again have an alert mechanism to notify if the controller is not listening to the signals from the receiver.
- Some applications may require the system to access certain readings from the sensor network. For instance, if the user says, “Turn off the lights if the room is empty”. Here, the recent readings from the occupancy sensor shall be accessed to verify if the room is empty or not. The factor of failure here could be the instability in communication between the sensor network and the controller (where we will be reading the data from). We go with the same solution as before for communication links; having an alert mechanism to notify broken links.
- Continuing with the same situation as the previous point, if the sensor itself is poor in quality, the readings can be wrong and thus again, the whole system might fail to perform in the expected manner. The solution here is straight, either we get better sensor or find alternative ways in which our final output would not be so dependent on this particular sensor.



Summarizing the failure factors in a table,

Category	Failure Factor	Solution
Speech to Text	<ul style="list-style-type: none"> <li>- Variations in human speech</li> <li>- Large vocabulary continuous speech recognition</li> <li>- Undefined word boundaries</li> <li>- Accent</li> <li>- External influence – noise, echoes</li> <li>- Poor Internet connection</li> </ul>	<ul style="list-style-type: none"> <li>- Machine Learning (Acoustic models, Monophone models and Language models)</li> <li>- User-input during configuration</li> <li>- Noise-cancelling microphone</li> <li>- Ensure better connectivity</li> </ul>
Text to Keyword	<ul style="list-style-type: none"> <li>- Misinterpretation from STT</li> <li>- Fewer or no keywords from user</li> <li>- Complex user command</li> </ul>	<ul style="list-style-type: none"> <li>- More robust STT</li> <li>- Feedback mechanism</li> <li>- More advanced logic engine</li> </ul>
Keyword to Action	<ul style="list-style-type: none"> <li>- Communication error</li> <li>- Poor sensor quality</li> </ul>	<ul style="list-style-type: none"> <li>- Stable communication; alert mechanism</li> <li>- Get better sensors</li> </ul>

Table 4.1: Summarized Failure Factors

# Chapter 5

## Implementation

The important components for the implementation – both hardware and software are stated as follows,

1. Raspberry Pi 3
2. IBM Bluemix account
3. Node-RED
4. USB Microphone (The one used in this project is, Yoga EM-310U)

### 5.1 Raspberry Pi

#### 5.1.1 What is Raspberry Pi?

The Pi is a tiny computer about the size of a credit-card, the board features a processor, RAM and typical hardware ports we find with most computers. Even though the basic model costs as low as 35\$, the possibilities of using it in our daily lives are endless. It can be used for controlling of hardware, as a media center, setting up camera projects, building games or most interesting of all, physical computing – which involves building systems using sensors, motors, lights and micro-controllers. The specifications of Raspberry Pi 3 - Model B (used in this project) are,

- SoC: Broadcom BCM2837
- CPU: 4x ARM Cortex-A53, 1.2GHz

- GPU: Broadcom VideoCore IV
- RAM: 1GB LPDDR2 (900 MHz)
- Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
- Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
- Storage: microSD
- GPIO: 40-pin header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- OS: The primary supported operating system is Raspbian, although it is compatible with many others. We have used Raspbian in this project.

### 5.1.2 Setting up the Pi

The steps are as follows,

1. Format the SD Card: To begin with, it's always a good idea to make sure we have formatted our SD card (recommended size at least 8 GB, we are using 32 GB in this project). We will need to make sure our computer has a built-in SD card reader, or we can use a USB SD card reader.
  - Firstly, we need to visit the SD Association's website (<https://www.sdcard.org/>) and download SD Formatter 4.0 for either Windows or Mac.
  - Then, we follow the instructions to install the software.
  - Then, we insert our SD card into the computer or laptop's SD card reader and make a note of the drive letter allocated to it, e.g. F:/.
  - Lastly, in SD Formatter, we need to select the drive letter for our SD card and format it.

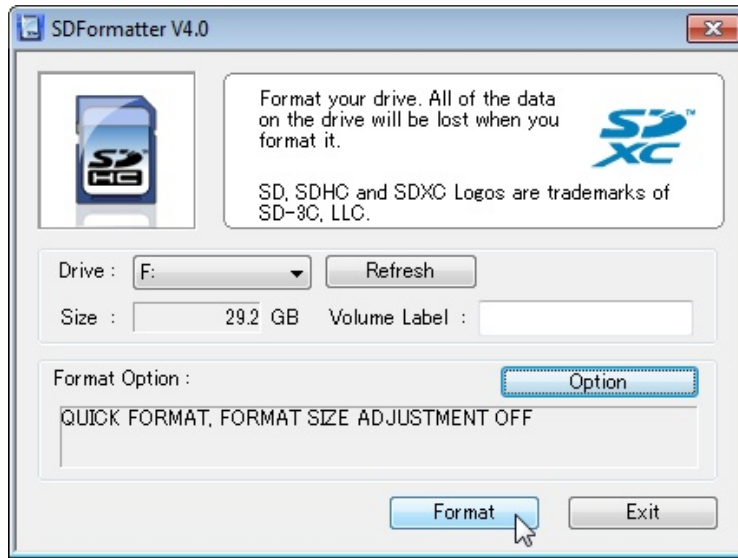


Figure 5.1: Screenshot of the SDFormatter V4.0

2. Install Raspbian with NOOBS:  
NOOBS stands for New Out Of Box Software, and if one has never played around with GNU/Linux before, then it's the best place to start.
  - Firstly, we need to visit the official Raspberry Pi Downloads page. (<https://www.raspberrypi.org/downloads/>)
  - Then, we click on NOOBS and then click on the Download ZIP button under 'NOOBS (offline and network install)', and select a folder to save it to.
  - We then, extract the files from the zip.
  - Once our SD card has been formatted, we can drag all the files in the extracted NOOBS folder and drop them onto the SD card drive. The necessary files will then be transferred to our SD card.
  - When this process has finished, we can safely remove the SD card and insert it into our Raspberry Pi.
3. Powering on the Pi for the first time:  
There are some additional hardware required for just the first time we boot our Pi.

- Once the SD card is placed into the SD card slot on the Raspberry Pi, we plug our keyboard and mouse into the USB ports on the Raspberry Pi.
- Next, we connect an HDMI cable from our Raspberry Pi to a monitor or TV. We need to make sure that the monitor or TV is turned on, and that we have selected the right input (e.g. HDMI 1, DVI, etc).
- If we intend to connect our Raspberry Pi to the internet, we plug an Ethernet cable into the Ethernet port, or connect a WiFi dongle to one of the USB ports (unless we have a Raspberry Pi 3).
- Finally, we connect the micro USB power supply. This action will turn on and boot our Raspberry Pi. We will now have to select an operating system (select Raspbian from the list) and let it install.

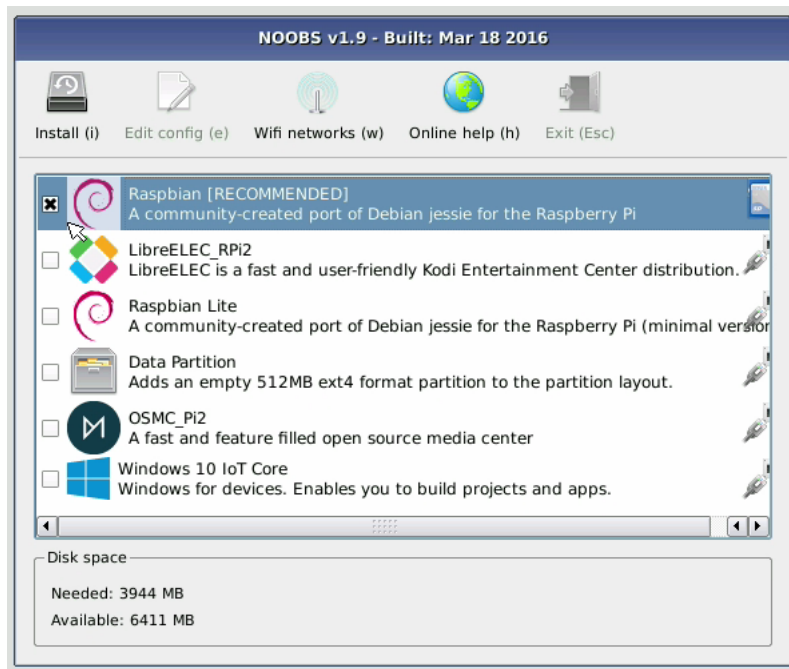


Figure 5.2: Install screen we get first time we boot the Pi

4. Enable SSH and VNC:

We can access the command line of a Raspberry Pi remotely from

another computer or device on the same network using SSH (Secure Shell). The Raspberry Pi will act as a remote device: we can connect to it using a client on another machine. We only have access to the command line, not the full desktop environment. For a full remote desktop, we use VNC (Virtual Network Computing). It is a graphical desktop sharing system that allows us to remotely control the desktop interface of one computer (running VNC Server) from another computer or mobile device (running VNC Viewer). VNC Viewer transmits the keyboard and either mouse or touch events to VNC Server, and receives updates to the screen in return.

Raspbian generally has the SSH server and VNC disabled by default. These can be enabled manually from the desktop:

- Launch Raspberry Pi Configuration from the Preferences menu
- Navigate to the Interfaces tab
- Select Enabled next to SSH and next to VNC
- Click OK

Alternatively, `raspi-config` can be used:

- Enter `sudo raspi-config` in a terminal window
- Select Interfacing Options
- Navigate to and select SSH
- Choose Yes
- Do the same for VNC
- Select Ok
- Choose Finish

Note: Our Pi is ready for the project now. If we do not have a screen available always and need to remotely access the Pi, we continue with the next steps.

#### 5. Finding the IP address of Pi:

As mentioned before, we can access the Raspberry Pi remotely from another computer or device on the same network. But we first need to know the IP address of our Pi.

- If we have a display at our disposal, we can connect our Pi to the display just to know the IP and then, we can start accessing the Pi remotely from our device. For this, we use the terminal (boot to the command line or open a Terminal window from the desktop) and simply type `hostname -I` which will reveal your Pi's IP address.
- It is even possible to find the IP address of our Pi without connecting to a screen using Router device list. In a web browser, we navigate to our router's IP address e.g. `http://192.168.1.1`, which is usually printed on a label on the router; this will take us to a control panel. Then logging in using our credentials (usually printed on the router or sent to the owner in the accompanying paperwork), we browse to the list of connected devices or similar (all routers are different), and we should see some devices that we recognise. Some devices are detected as PCs, tablets, phones, printers, etc. so we should recognise some and rule them out to figure out which is the Raspberry Pi. Also, we should note the connection type; if our Pi is connected with a wire there should be fewer devices to choose from.

6. Remotely access the command line of our Pi:

Once we know the IP we can move towards accessing our Pi remotely. For accessing only the command line, we use SSH. We will need to download an SSH client for this. Since the remote device used in this project was running on Windows OS, we have used the most commonly used client for Windows, called PuTTY.

- PuTTY does not include an installer package: it is a stand-alone .exe file. When we run it, we will see the configuration screen below:

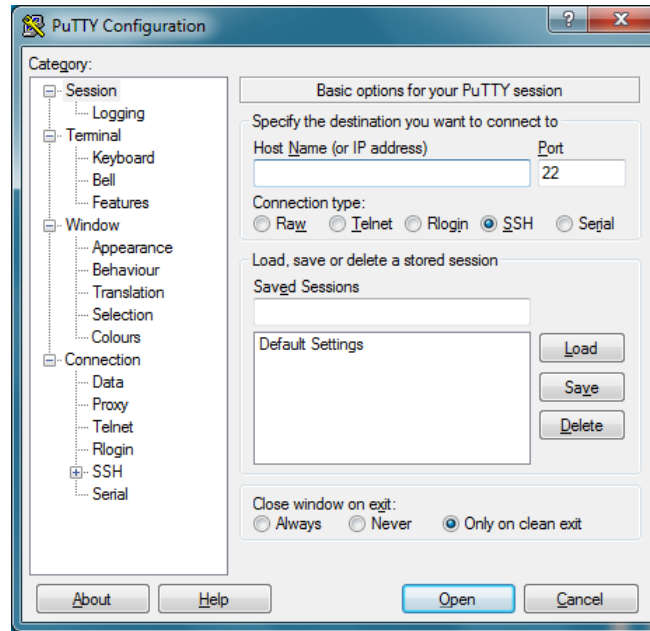


Figure 5.3: Configuration Screen of PuTTY

- Now, we type the IP address of the Pi into the Host Name field and click the Open button. If nothing happens when we click the Open button, and we eventually see a message saying Network error: Connection timed out, it is likely that we have entered the wrong IP address for the Pi.
- When the connection works, we will see the security warning shown below. We can safely ignore it, and click the 'Yes' button. We will only see this warning the first-time PuTTY connects to a Raspberry Pi that it has not seen before.



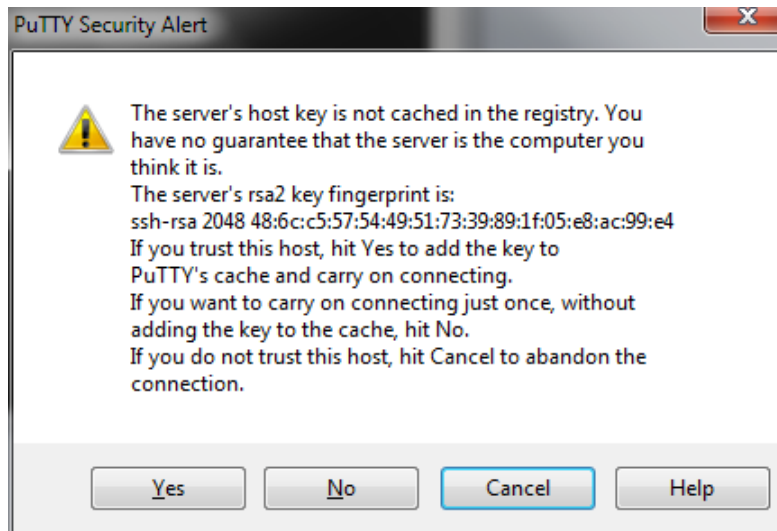
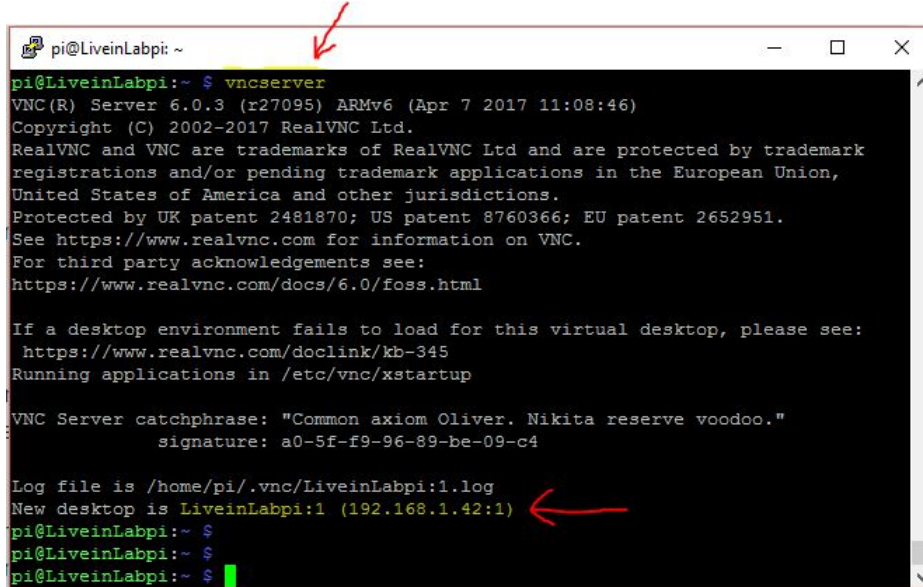


Figure 5.4: Security warning generated by PuTTY

- We will now see the usual login prompt. We need to log in with the same username and password we would use on the Pi. The default login for Raspbian is pi with the password raspberry. We should now have the Raspberry Pi prompt which will be identical to the one found on the Raspberry Pi itself.
- We can type exit at any moment to close the PuTTY window.

7. Remotely access the desktop environment of our Pi:

- The first step here is to create a virtual desktop. On our Raspberry Pi (using Terminal or via SSH), we run vncserver and make note of the IP address/display number that VNC Server will print to our Terminal (e.g. 192.167.1.42:1).



```
pi@LiveinLabpi: ~  
pi@LiveinLabpi:~ $ vncserver  
VNC(R) Server 6.0.3 (r27095) ARMv6 (Apr 7 2017 11:08:46)  
Copyright (C) 2002-2017 RealVNC Ltd.  
RealVNC and VNC are trademarks of RealVNC Ltd and are protected by trademark  
registrations and/or pending trademark applications in the European Union,  
United States of America and other jurisdictions.  
Protected by UK patent 2481870; US patent 8760366; EU patent 2652951.  
See https://www.realvnc.com for information on VNC.  
For third party acknowledgements see:  
https://www.realvnc.com/docs/6.0/foss.html  
  
If a desktop environment fails to load for this virtual desktop, please see:  
https://www.realvnc.com/doclink/kb-345  
Running applications in /etc/vnc/xstartup  
  
VNC Server catchphrase: "Common axiom Oliver. Nikita reserve voodoo."  
signature: a0-5f-f9-96-89-be-09-c4  
  
Log file is /home/pi/.vnc/LiveinLabpi:1.log  
New desktop is LiveinLabpi:1 (192.168.1.42:1)  
pi@LiveinLabpi:~ $  
pi@LiveinLabpi:~ $  
pi@LiveinLabpi:~ $
```

Figure 5.5: Screenshot of vncserver command run via SSH (PuTTY)

- To destroy a virtual desktop, we can run the following command:  
*vncserver -kill :display-number*  
This will also stop any existing connections to this virtual desktop.
- Next, we open a VNC Viewer on our remote device and type in the above address. Once we do this, we get an authentication screen, where we need to enter the username and password of our Pi. This will take us to the desktop environment.

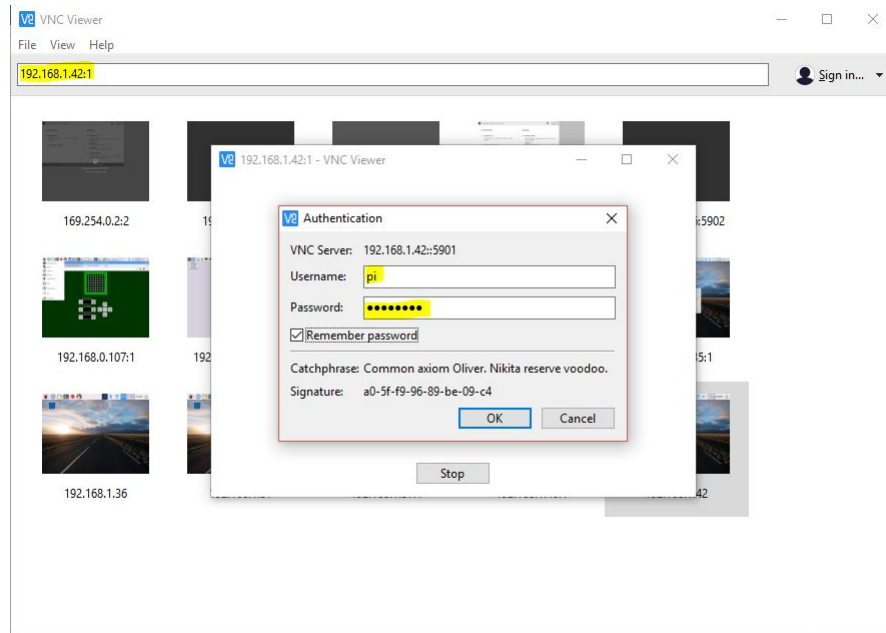


Figure 5.6: VNC Viewer welcome screen

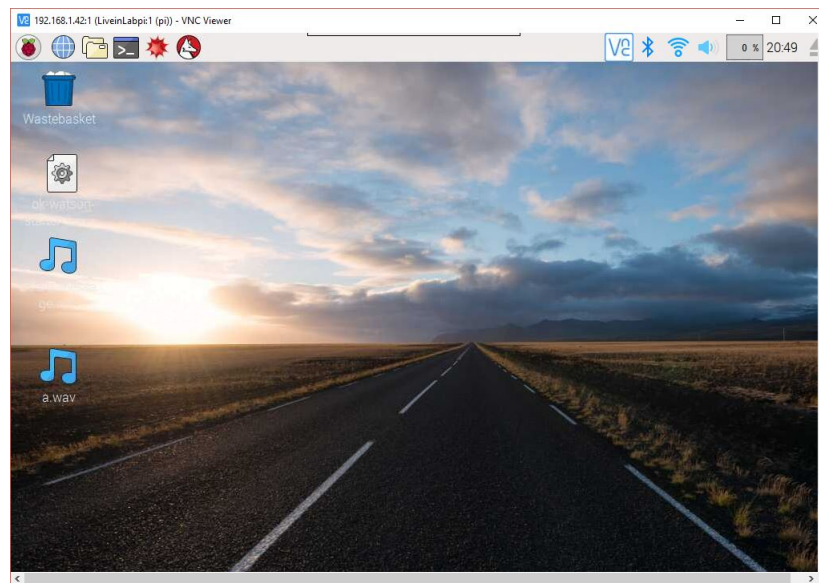


Figure 5.7: Screenshot of the desktop environment as visible on the remote device

## 5.2 IBM Bluemix

Bluemix is a cloud platform provided by IBM, that connects our hardware setup to the cloud. It makes it possible for us to access the data from our raspberry Pi from anywhere in the world. We just need the login credentials of the Bluemix account connected with our Pi. It has a wide range of services and APIs that can help us scale on virtual servers and build micro services based on event-driven models.

### 5.2.1 Getting started with Bluemix

To log into Bluemix, we head to the IBM Bluemix login page. We can sign up for an IBM ID and BlueMix from there. Once Bluemix loads, we select our region by clicking the top right-hand corner account icon. Then, if Bluemix requests that we create a space in that region, we do so. I named my space “test”.

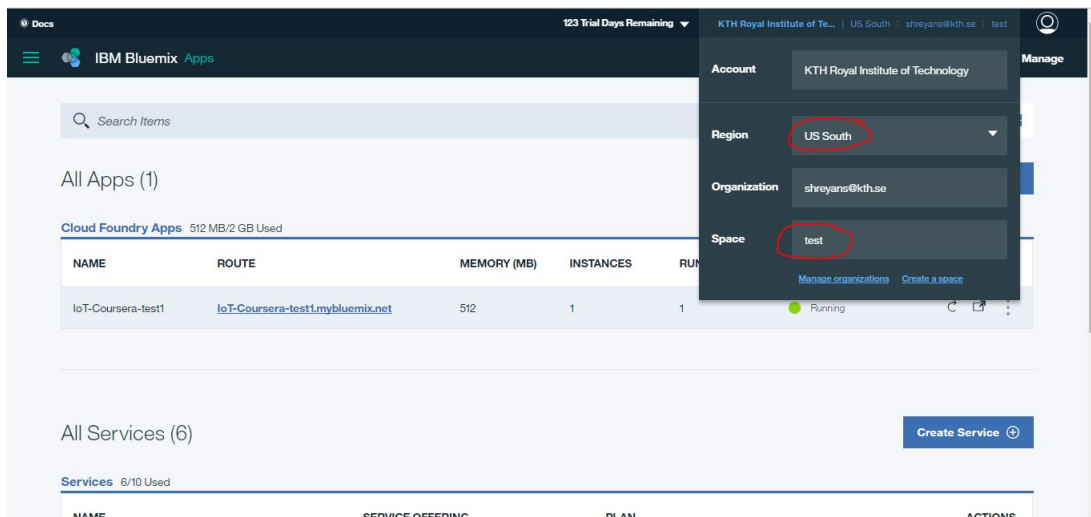


Figure 5.8: Screenshot showing the credentials for a Bluemix account

Then, we click on “Use Services or APIs” to find a good initial service for our app. In this screen, we need to find the “Internet of Things Platform” service and select it for our app. We then click “Create” on the next screen, we can change the “Service Name” if we want. Since, I started using Bluemix

through Coursera, my service has since been named, IoT-Coursera-test1-iotf-service and I am continuing with the same service for this project too. We scroll down on the welcome screen that appears and choose “Launch”.

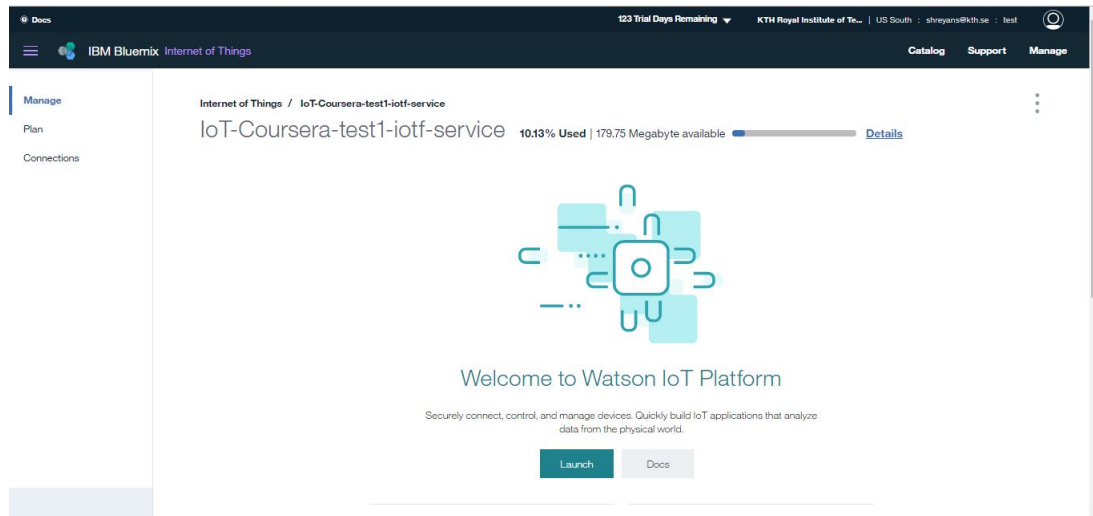


Figure 5.9: Screenshot of the welcome page for Bluemix service

Now we can add our Raspberry Pi to this new service by clicking “Add Device”. Then, click to “Create device type” if we are adding some device type for the first time. Else, we can choose the type from “Choose Device Type”.

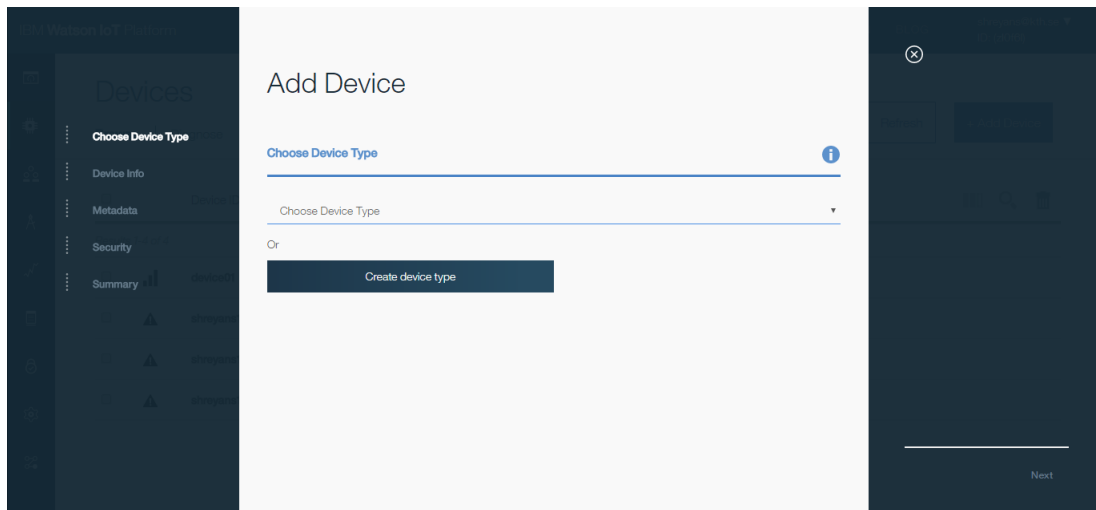


Figure 5.10: Adding a new device

Another screen will appear asking whether we want to create a device type or gateway type. We want a device type. Finally, we name our device type (I have named it “raspberrypi”). Underneath that, we can write a longer and more human readable description.

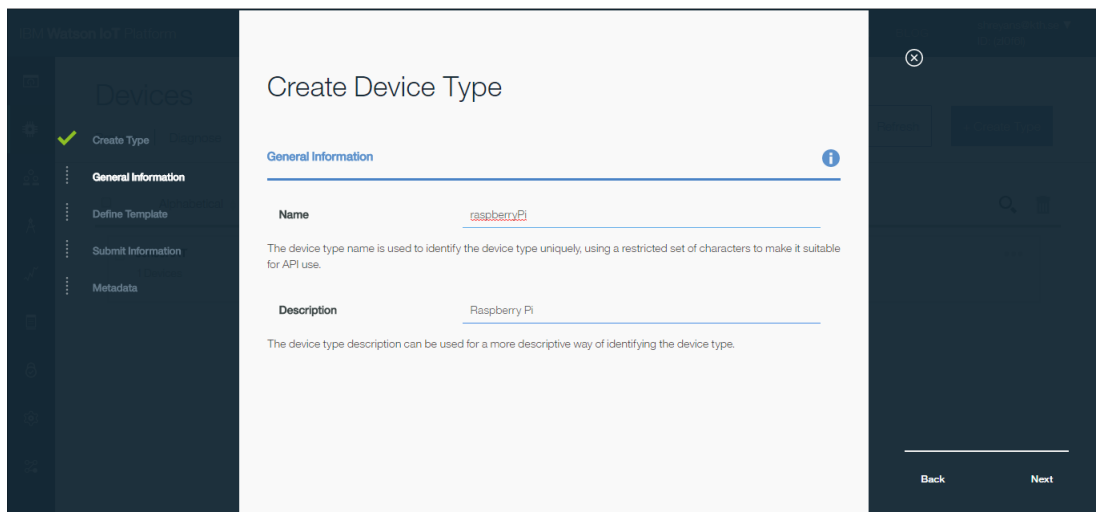


Figure 5.11: Creating a device type named raspberrypi

The next screen gives us options for our device template, providing fields

we can use for each device to define its characteristics. This is very much up to us and what device data we would like to record in this device type. We can add our own custom metadata in JSON format if we like. For this project, I have skipped this step. Now our device type is ready to be used. We should be back at the “Add Device” screen. This time, our new device type should be selected and click “Next”.

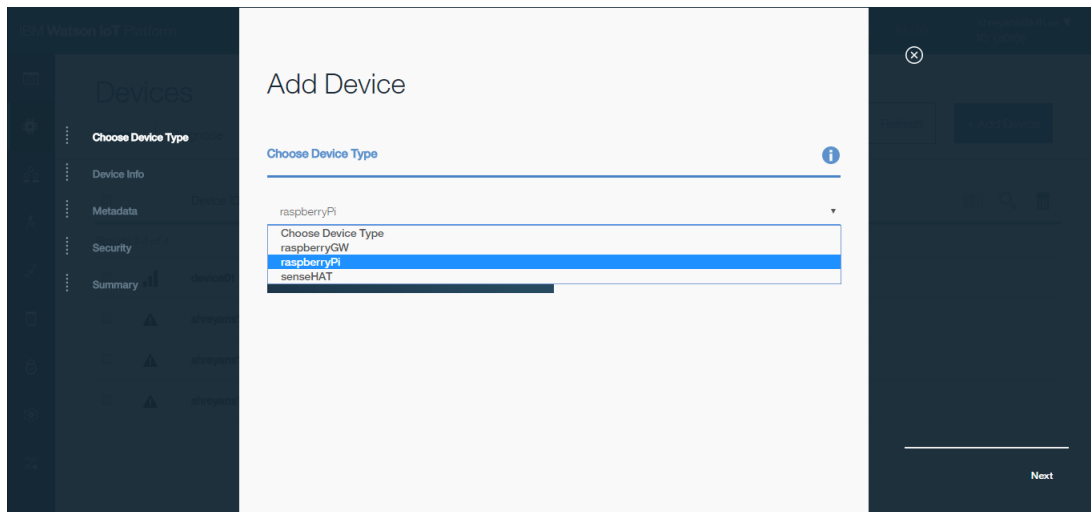


Figure 5.12: Adding new device, using the created device type

We now set up our individual device info for our Raspberry Pi into the Bluemix system. We give our device a unique ID (something that will be different to all other devices in our system). I have used the ID – device01. We can add any additional information if we want and then click “Next”. We can skip the metadata part again, unless there is specific data we want to store about our device. Then, we set up our authentication token. We can define a custom one or leave it blank for the system to automatically generate one for us. I have defined a custom one for my device. Once we click “Next”, we are asked to check the details to ensure they are correct and then click “Add”. The final screen will show all the device’s details, including the generated authentication token (or the one we put down for it). We must copy these details into a safe and easy to find place. Especially ensure that we have the authentication token saved somewhere that is easy to access as we cannot get this value ever again (we would have to delete and create a new device then). Once we have got all these values saved, close this pop-up

window.

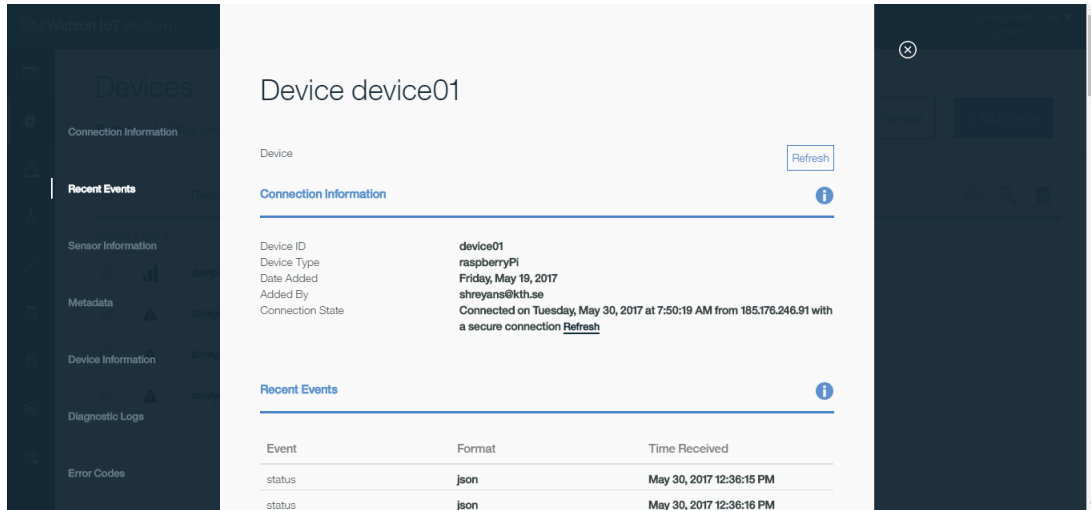


Figure 5.13: Screenshot of the device information for device01

### 5.2.2 Linking our Pi to our device in Bluemix

Now we want to link up our Raspberry Pi to the device we just set up in Bluemix. To do so, we need to first stop any Watson IoT service running on the Pi:

```
sudo service iot stop
```

Then, we type in the following to open up the Watson IoT config file for our Raspberry Pi (it will be created when we save the file if it does not already exist):

```
sudo nano /etc/iotsample-raspberrypi/device.cfg
```

Using the details we saved somewhere safe earlier, which should have looked like so:

```
Organization ID abcde
Device Type raspberrypi
Device ID device01
Authentication Method token
Authentication Token YOURTOKENWOULDBEHERE
```

We input them into our config file in this format:

```
# Device configuration file
org = abcde
```



```
type = raspberryPi
id = device01
auth-method = token
auth-token = YOURTOKENWOULDBEHERE
# End of Configuration file
```

We save those changes by pressing Ctrl + X and then typing “Y” when it asks if we would like to “Save modified buffer”. We keep the file name as is to write to the same file (/etc/iotsample-raspberrypi/device.cfg). Then hit enter if it shows the right filename.

Once that is saved, we are ready to set up Node-RED!

## 5.3 Node-RED

To do some more advanced things, we will install and run Node-RED, an environment that lets you work with connected devices and data without needing to delve into too much coding.

### 5.3.1 Setting up Node-RED on our Pi

We need to open a terminal on our Raspberry Pi and type in the following to update everything on our Raspberry Pi to the latest versions. Newer versions of Raspbian for the Raspberry Pi (Raspbian Jessie), come with Node-RED and Watson IoT already. However, it is recommended to update them all to get things to work correctly. So either way, we should update everything to be safe or install them from scratch if we don’t have them yet.

```
sudo apt-get update
```

Then we run this one too:

```
sudo apt-get dist-upgrade
```

If we have a Raspberry Pi 3 or any Raspberry Pi with Raspbian Jessie installed, we would not need to install Node-RED as it should already be there (and be updated to the latest version through that last command we just ran). But, if we do not have the latest version of Raspbian, we may need to install Node-RED. You can do this by first installing all its dependencies:

```
sudo apt-get install build-essential python-dev python-rpi.gpio
```

If we receive an error about “sudo: npm: command not found”, we will need to run the following to install npm first:

```
sudo apt-get install npm
```

Then, by installing Node-RED itself via npm:

```
sudo npm install -g --unsafe-perm node-red
```

In order to have access to the the IBM Watson IoT Node, we run this command too:

```
sudo npm install -g node-red-contrib-ibm-watson-iot
```

For some cases, the above command might not work and fail due to an error with the script referencing node rather than nodejs — this could be happening on Raspbian Jessie and if so, we don't need to worry as this is already installed for us on that version of Raspbian. If we would like to access Node-RED from our computer, rather than the Pi — we will need to know your Pi's local IP address. This can be found by the methods mentioned earlier (Finding the IP address of Pi). If all is installed successfully, we should be able to run Node-RED on our Pi using the following command:

```
node-red
```

```

Node-RED console
File Edit Tabs Help
^C30 May 18:15:19 - [info] Stopping flows
pi@LiveinLabpi:~$ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.1.42:1880
On Pi Node-RED works better with the Firefox browser

Use node-red-stop to stop Node-RED
Use node-red-start to start Node-RED again
Use node-red-log to view the recent log output
Use sudo systemctl enable nodered.service to autostart Node-RED at every boot
Use sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Started Node-RED graphical event wiring tool..
Welcome to Node-RED
=====
30 May 18:15:27 - [info] Node-RED version: v0.15.3
30 May 18:15:27 - [info] Node.js version: v7.10.0
30 May 18:15:27 - [info] Linux 4.4.50-v7+ arm LE
30 May 18:15:28 - [info] Loading palette nodes
pi : TTY=unknown ; PWD=/home/pi ; USER=root ; COMMAND=/usr/bin/python -u /usr/lib
/node_modules/node-red/nodes/core/hardware/nrgpio.py info
pam_unix(sudo:session): session opened for user root by (uid=0)

```

Figure 5.14: Running node-red-start

If we then go to either <http://127.0.0.1:1880> or [localhost:1880](http://localhost:1880) on our Pi itself or <http://your-pi-ip-address:1880> from another computer on the same network, we should see Node-RED ready and waiting. Check that within the interface, underneath both Input and Output, we should be able to see Watson IoT as an option.

Troubleshoot: If we get any error when we launch node-red (for instance, “connection was closed”), we need to first stop any Watson IoT service running on the Pi, by typing “sudo service iot stop” in another terminal.

## 5.4 Prototype 1

Since, the Raspberry Pi, IBM Bluemix account and Node-RED are all set up, we can start building the first Prototype.

The IBM Speech to Text service provides an API that lets us add speech transcription capabilities to our applications. To transcribe the human voice accurately, the service leverages machine intelligence to combine information about grammar and language structure with knowledge of the composition of the audio signal. The service continuously returns and retroactively updates the transcription as more speech is heard. But, this feature would not help us a lot, as we will be mostly dealing with small sentences rather than long continuous speech.

In prototype 1, we will test the Speech to Text service of IBM Watson for converting simple predefined set of voice commands into transcriptions and further take relevant action, i.e. to turn on/off the correct LED connected to the Pi.

Note: How to connect an LED to GPIO pins of a Raspberry Pi will not be covered in this report, these were done just for a visual demonstration. If you are still interested to implement that, then I would suggest referring to [10] for building a circuit.

The set of predefined commands are as follows,

- Turn on the light/s
- Turn off the light/s
- Switch on the light/s
- Switch off the light/s
- Turn on the heater/s
- Turn off the heater/s
- Switch on the heater/s

- Switch off the heater/s
- Turn on the music
- Turn off the music
- Play some music
- Stop the music

Now, we go through the steps taken for developing the Prototype 1:

1. Create Watson Speech to Text service in Bluemix

We go to the Bluemix Services page and find the “Speech to Text” service (we need to be careful not to choose “Text to Speech” that’s different). That should take us to the Speech to Text service Bluemix page. On that page, we’ll see various options for adding this service to our IBM Bluemix arsenal. We leave the app unbound; we can give the service a name and give the credentials a name. The only plan I had available was “Standard”, so I left that one as is too. Once we’re happy with our settings, we click “Create”. Once the service is created in our space, we’ll be taken to the page for that service. We click the “Service Credentials” menu item on the left to access the username and password we will need to give Node-RED to have access to our new IBM Watson Speech to Text service. We need to copy down the username and password from this page.

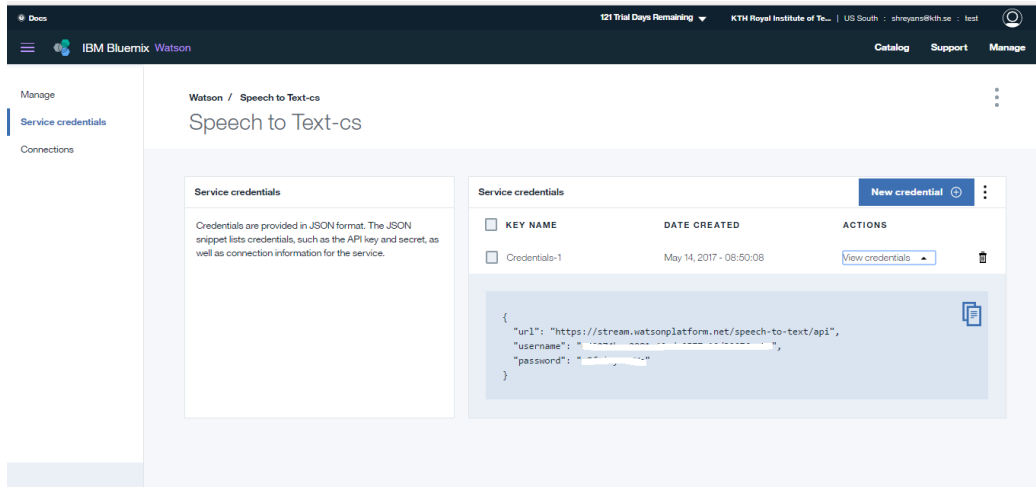


Figure 5.15: Screenshot of the page with service-credentials

## 2. Adding new services to Node-RED

In order to access the IBM Watson Text to Speech service in Node-RED, we will need to install some new nodes. To do so, we SSH into our Pi (or open the terminal from our Pi directly) and type in:

```
cd ~/.node-red
```

This brings us to the Node-RED app folder. From within here, we install a new collection of Node-RED nodes called `node-red-node-watson`. This includes access to a whole range of IBM Watson services, including the Speech to Text that we need. To install it, we run the following command on our Pi from the Node-RED folder:

```
sudo npm install node-red-node-watson
```

Next we install some additional modules on our Node-Red for audio recording using USB Microphone. We need to install `node-red-contrib-browser-utils` (collection of Node-RED nodes for browser functionality such as file upload, camera and microphone) and `node-red-contrib-media-utils` (collection of Node-RED media nodes using FFmpeg). To install these, we run the following command on our Pi from the Node-RED folder:

```
sudo npm install node-red-contrib-browser-utils
sudo npm install node-red-contrib-media-utils
```

For the new Node-RED node changes to come into effect, we need to restart Node-RED. To do so, we run the following two commands:

```
node-red-stop  
node-red-start
```

3. Start building the flow

The basic functionalities of our flow should be,

- Turn on/off the lights (represented by the LED connected to Pin 11)
- Turn on/off the heater (represented by the LED connected to Pin 15)
- Turn on/off the music (represented by the LED connected to Pin 18)

Note: These pin numbers are the “Physical pin numbers”, as can be referred from [10]. Also, if you are not using LEDs then, we can simply change the functionality to printing different outputs corresponding to different appliances.

After steps 1 and 2, we can now start building the flow in Node-RED. We go to either <http://127.0.0.1:1880> or [localhost:1880](http://localhost:1880) on a web browser in our Pi or <http://{your-pi-ip-address}:1880> from another computer on the same network, we should see Node-RED ready and waiting.

- (a) We select the “microphone” node from the input section and bring it on our palette. We can either give a name to this node by double-clicking on the node (I named it “USB Mic”), or leave it as it is.
- (b) From the IBM\_Watson section of nodes, we drag the “speech to text” node on to our palette. We double-click the node and give the node a name if we want; add the username and password saved earlier (service credentials) to connect it to our Bluemix service; select a language model (we are using “US English”); and quality (we are using “Broadband model”). Finally, we check the continuous speech checkbox, uncheck the speaker labels checkbox and click on Done. We connect the output from the microphone node to the input of speech to text node.

Figure 5.16: Speech to text node details

- (c) Next, we drag the “function” node from the function section and bring it on our palette. We double click on it and add the following piece of code:

```
msg.payload = msg.transcription;
return msg;
```

We give the node a name – “set payload” and click on Done and connect the output of speech to text node to the input of this function node.

- (d) Now, we bring a “debug” node from the output section on to our palette and connect it to the output of the “set payload”. This will help us in seeing the transcription of our speech in the debug tab of Node-RED environment.
- (e) Next, we bring in the “switch” node from the function section into our palette. We first need to check for the task asked in the command – if the user has asked to turn on or turn off, be it any appliance. We do this by putting conditions in the switch node as follows,

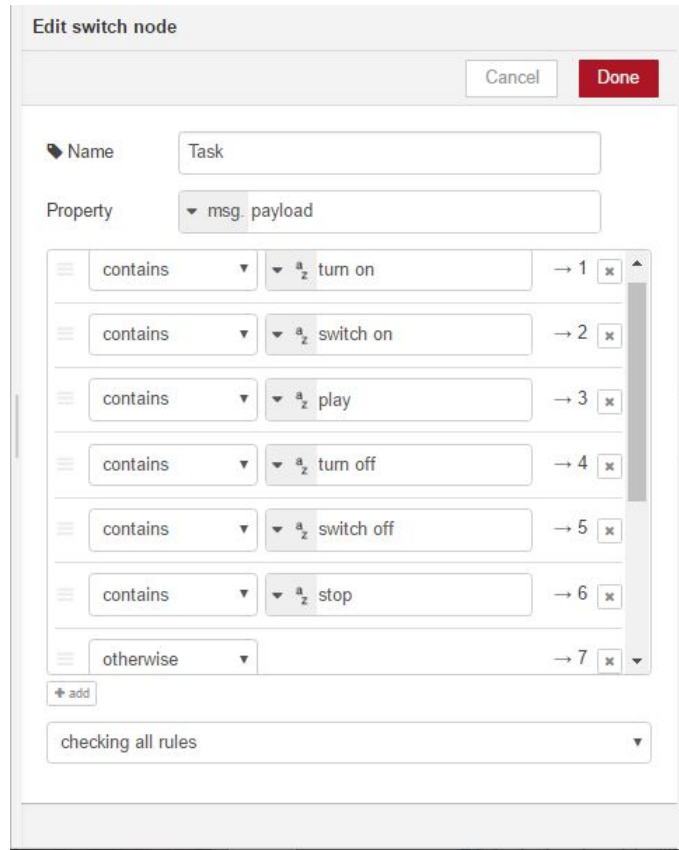


Figure 5.17: Editing the switch node for tasks

Then we click on Done and connect the output of “set payload” to the input of the switch node.

- (f) Now, we bring 2 more switch nodes in to palette. These are for checking now, which appliance is the user asking to control. The edit switch node for both looks the same except the name of the nodes “appliance on” and “appliance off”.



Figure 5.18: Edit switch node for appliances

- (g) Next we, create 2 function nodes for handling the case - when our flow reaches the “otherwise” condition in the switch nodes. These are named, “Error msg 1” (for the “Task” switch node) and “Error msg 2” (for the “appliances on” and “appliances off” switch nodes). The “Error msg 1” is connected to the last output of the “Task” switch node; and the contents are as follows:

```
msg.payload = "I did not get you. Please repeat!";
return msg;
```

The “Error msg 2” is connected to the last output of the “appliances on” and “appliances off” switch nodes; and the contents are as follows:

```
msg.payload = "I can only control light, heater or music";
return msg;
```

Then, we connect both these function nodes to debug nodes, such that the error message gets printed on the debug tab of the Node-RED environment.

- (h) Now, we bring in 6 function nodes, 3 setting msg.payload to 1:
- ```
msg.payload = 1;
return msg;
```
- We connect these 3 to first 3 outputs of “appliances on” switch node. And other 3 setting msg.payload to 0:
- ```
msg.payload = 0;
return msg;
```
- We connect these 3 to first 3 outputs of “appliances off” switch node.
- (i) Lastly, we connect the function nodes – 1s and 0s (created in previous step) to “rpi gpio” nodes from the Raspberry.Pi section. We connect them based on the appliances we assigned to each pin. Pin 11 – Lights, Pin 15 – Heater, Pin 18 – Music. The Node-RED flow looks like this:

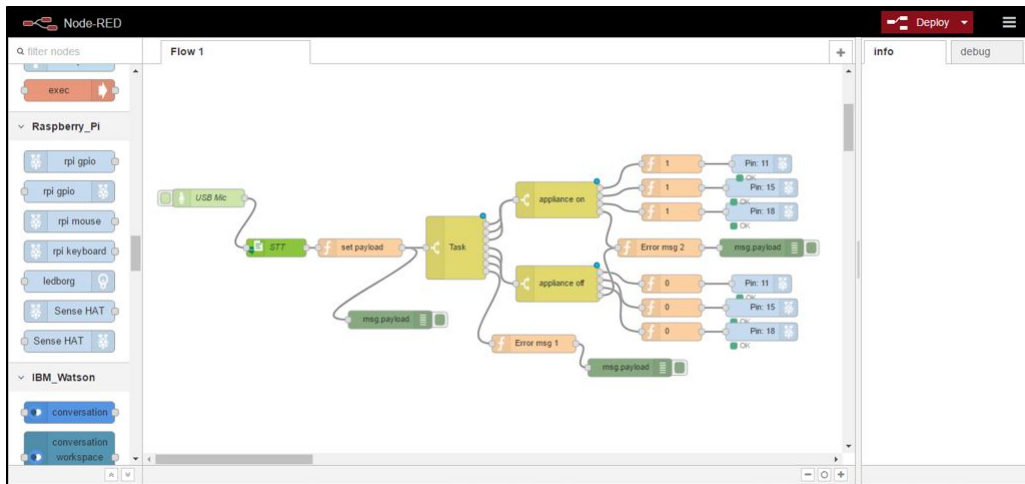


Figure 5.19: The Node-RED flow for Prototype 1 (version 1.0)

Note: If you are not using LEDs, simply replace steps 8 and 9 with connecting debug nodes to different outlets of “appliance on” and “appliance off”. Thus, we will see different messages on the screen instead of LED glowing on/off.

We now click on “Deploy” and we are all set to test our prototype. The flow goes like this:

- (a) Clicking the “USB Mic” node button toggles recording.
- (b) Then we record the voice command using USB Microphone (Play some music, Turn on light, etc.)
- (c) Microphone node sends a buffer of the recorded audio as the msg.payload object to Watson Speech to Text Node.
- (d) Then Watson converts the buffer into text.
- (e) “set payload” node moves the string output of speech to text to the msg.payload object as required by the subsequent nodes.
- (f) The text is analysed in “Task” and “appliance on/off” switch nodes, to look for keywords it might contain.
- (g) Based on the keywords, a signal is either sent to a GPIO pin (representing an appliance here), or to the output screen with some error message.

After testing numerous times, calling out each command at least 6 times, we got the following performance.

Command	No. of tries	Correct output (LED on/off)
Turn on the light/s	6	4
Turn off the light/s	6	3
Switch on the light/s	6	6
Switch off the light/s	6	6
Turn on the heater/s	6	2
Turn off the heater/s	6	3
Switch on the heater/s	6	5
Switch off the heater/s	6	5
Turn on the music	6	1
Turn off the music	6	2
Play some music	6	6
Stop the music	6	6
	<b>72</b>	<b>49 (68.05%)</b>

Table 5.1: Performance - Prototype 1 (version 1)

Let us have a look at a random set of outputs received from the Speech

to text, when I gave all the predefined commands in order from 1-12.



Figure 5.20: STT output for commands 1-4

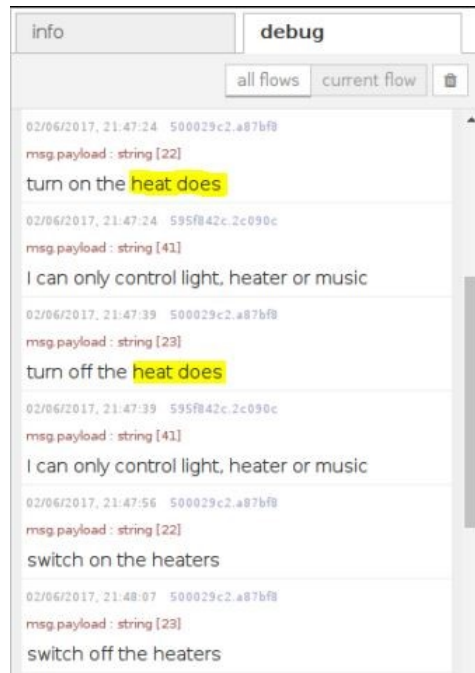
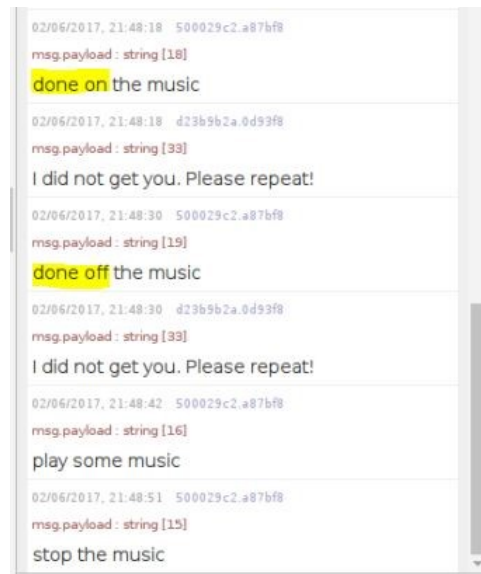


Figure 5.21: STT output for commands 5-8



```
02/06/2017, 21:48:18 500029c2.a87bf8
msg.payload: string [18]
done on the music

02/06/2017, 21:48:18 d23b9b2a.0d93f8
msg.payload: string [33]
I did not get you. Please repeat!

02/06/2017, 21:48:30 500029c2.a87bf8
msg.payload: string [19]
done off the music

02/06/2017, 21:48:30 d23b9b2a.0d93f8
msg.payload: string [33]
I did not get you. Please repeat!

02/06/2017, 21:48:42 500029c2.a87bf8
msg.payload: string [16]
play some music

02/06/2017, 21:48:51 500029c2.a87bf8
msg.payload: string [15]
stop the music
```

Figure 5.22: STT output for commands 9-12

After numerous runs, I found some common trends, which are mentioned as follows,

1. “Turn on” and “turn off” were misinterpreted very frequently, mostly as “done on”, “don on”, “done off” and “don off”. Hence, I added these to the Task switch node.

Figure 5.23 shows the 'Edit switch node' dialog box. The dialog has a title bar 'Edit switch node' with 'Cancel' and 'Done' buttons. It contains a 'Name' field with 'Task' and a 'Property' dropdown with 'msg. payload'. Below is a list of 11 items, each with a 'contains' dropdown, a text field, a right arrow, and a delete button. The items are: 1. contains, #\_2, turn on; 2. contains, #\_2, done on; 3. contains, #\_2, don on; 4. contains, #\_2, switch on; 5. contains, #\_2, play; 6. contains, #\_2, turn off; 7. contains, #\_2, done off; 8. contains, #\_2, don off; 9. contains, #\_2, switch off; 10. contains, #\_2, stop; 11. otherwise. At the bottom is an 'add' button and a dropdown menu showing 'checking all rules'.

Figure 5.23: Modified Task switch node

2. “Light” and “Heater” were often misinterpreted as “late” and “heat does” respectively. Hence, “late” was added and “heater” changed to “heat” in “appliances on” and “appliances off” switch node.

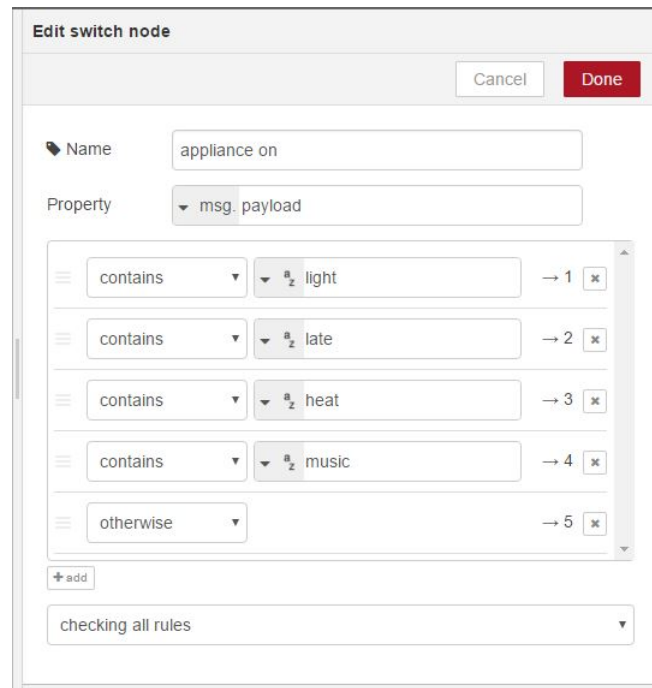


Figure 5.24: Modified appliance on (and appliance off)

Now the Node-RED flow looks like this:

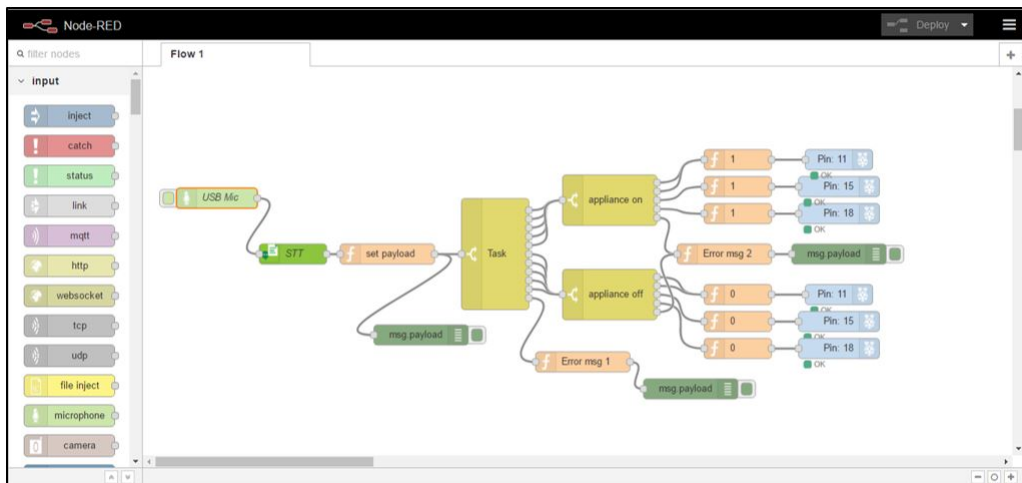


Figure 5.25: The Node-RED flow for Prototype 1 (version 2)

The performance table now can be seen below,

Command	No. of tries	Correct output (LED on/off)
Turn on the light/s	6	5
Turn off the light/s	6	4
Switch on the light/s	6	5
Switch off the light/s	6	6
Turn on the heater/s	6	4
Turn off the heater/s	6	5
Switch on the heater/s	6	4
Switch off the heater/s	6	4
Turn on the music	6	4
Turn off the music	6	5
Play some music	6	6
Stop the music	6	6
	<b>72</b>	<b>58 (80.55%)</b>

Table 5.2: Performance - Prototype 1 (version 2)

Although the performance did increase after this adjustment, but there was still a lot of scope for improvement. This prototype (Prototype-1) was designed for handling a set of pre-defined simple commands only, but it was not smart and most part of it was hard-coded. The intuition would say that, we should get 100% accuracy for pre-defined commands, since we can hard-code all the possible outcomes. But, we need to run a lot of tests to know the possible outcomes, as we cannot always predict what words are going to be interpreted by Speech to text service, because of accent or noise or other reasons. The most common ones were added based on the limited number of tests run (for instance, “late”, “done on”, “done off”, etc.), but not much effort was put in this area, as we would like to eventually make it interpret words based on the context also. Another shortcoming of this prototype is that it cannot handle minor modifications of the command, for instance, if the user says “turn the heater on”. If “turn” and “on” are separated, this system would not be able to handle it.

Thus, we move on to Prototype-2, where we plan to use the Watson’s ability to train itself based on the application, also the knowledge of the English language, to interpret the words based on the whole sentence (context) rather



than individual pieces of text.

## 5.5 Prototype 2

We begin building Prototype-2 with Speech to text customization.

The speech to text customization feature has been introduced so the service can learn specific input that is unique to one's use case. We train the service with sample sentences applicable to our domain. Most of the time the service will already have the words in its lexicon, but now it has context in which we use those words. It is then trained based on pronunciations of unusual words within the input which it otherwise would not know, for example names. The language model customization interface of the Speech to Text service is currently only available for US English and Japanese.

The steps for speech to text customization are as follows [11],

1. Create a custom language model

We start by adding 3 inject nodes (from the input section), 3 speech to text custom builder nodes (from the IBM\_Watson section) and one debug node (from the output section). Join them together like the following example:



Figure 5.26: Flow for creating custom language model

We do not need to configure the inject nodes. The Create Customization node is configured like shown below. We need to enter the exact same credentials as our Speech to text service. But, we can set our own names for the Custom Model Name and Custom Model Description. This creates the customization using the desired model.

The screenshot shows a dialog box titled "Edit speech to text custom builder node". At the top right are "Cancel" and "Done" buttons. The form contains the following fields:

- Name:** Create Customization
- Username:** 68227f1e-2221-4211-8227-10d39070c03c
- Password:** (masked with dots)
- Detect:** Create a custom language model (dropdown menu)
- Custom Model Name:** Bluemix04
- Base Model:** US English broadband model. (dropdown menu)
- Custom Model Description:** Bluemix Test 4

Figure 5.27: Create Customization node

The List Customization node is configured like shown below. This node allows us to view the customizations we have created.

The screenshot shows a dialog box titled "Edit speech to text custom builder node". At the top right are "Cancel" and "Done" buttons. The form contains the following fields:

- Name:** List Customizations
- Username:** 68227f1e-2221-4211-8227-10d39070c03c
- Password:** (masked with dots)
- Detect:** List Customizations (dropdown menu)

Figure 5.28: List customization node

The next step is to configure the debug node. The final customization node will be configured afterwards. We just need to change the output to ‘complete msg object’ and press ‘Done’. This allows us to view the full results of the inject nodes.

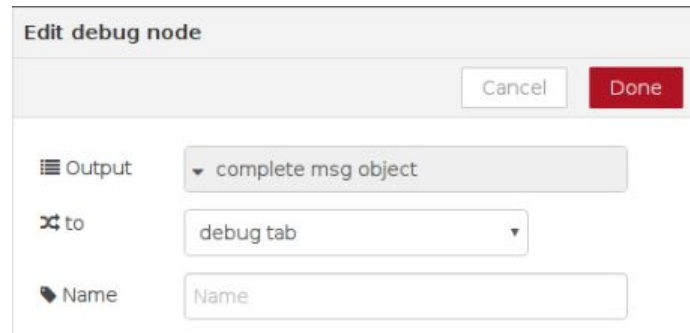


Figure 5.29: Debug node

Now, we can deploy our application and click the inject node for ‘Create Customization’ and then ‘List Customizations’.

In the debug tab we should see two ‘msg : Object’s. We look at the second one (from list customizations) and look for a ‘customization.id’. This is our customization so we shall make a note of the id as we will be needing it for the following steps.

Now, we can edit the final speech to text custom node by changing the ‘Detect’ field to ‘Get Customization’ and paste in the Customization ID we made a note of from the debug tab in the last step. Finally, we deploy the application.

**Edit speech to text custom builder node**

Cancel Done

Name: Get Customization

Username: 6822711a-0000-0000-0000-00000000003c

Password: .....

Detect: Get Customization

Customization ID: 25782250-5000-0000-0000-0000000000c7

Figure 5.30: Get Customization node

2. Add data from corpora to the custom language model

In this step, we add 1 file inject node (from input section), 2 inject nodes (from input section) and 3 more speech to text custom build nodes (from IBM\_Watson section) and join them to the debug node to look like shown below.

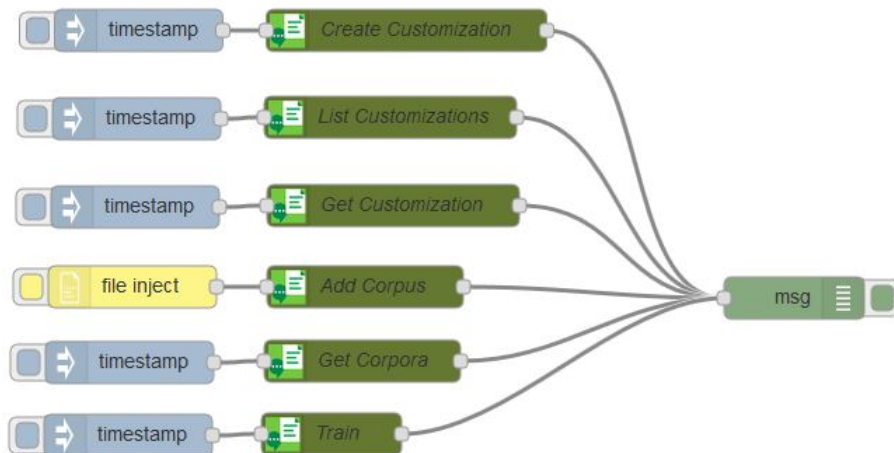
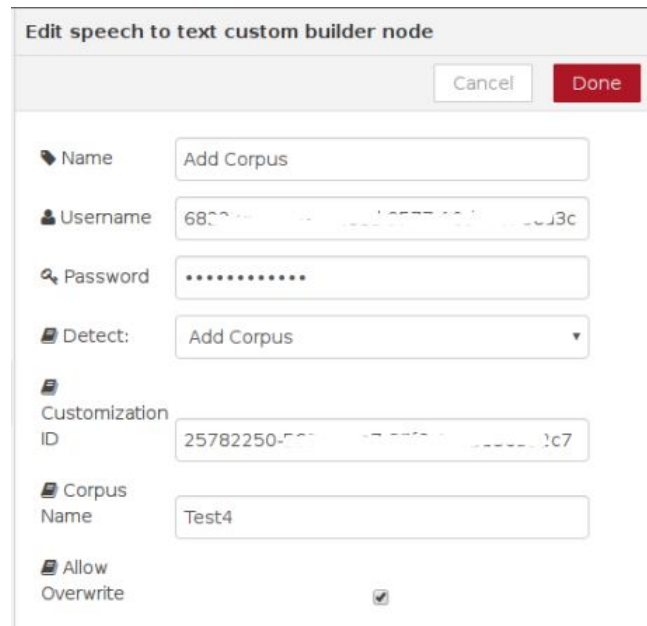


Figure 5.31: Node-RED flow for creating custom language model and adding data from corpora to it

We do not need to do anything with the file inject or inject node. In the speech to text custom node next to 'File Inject', we name it 'Add Corpus', change the 'Detect' field to 'Add Corpus' and paste in the customization ID we used in the previous step. Give the corpus a name in the final field. Note: The Corpus Name will not work with spaces. We then check the "Allow Overwrite" checkbox. If we leave it unchecked, we will not be able to add another Corpus to this customization.



The screenshot shows a dialog box titled "Edit speech to text custom builder node". At the top right are "Cancel" and "Done" buttons. The form contains the following fields and controls:

- Name:** A text input field containing "Add Corpus".
- Username:** A text input field containing "6800...".
- Password:** A text input field with masked characters (dots).
- Detect:** A dropdown menu currently showing "Add Corpus".
- Customization ID:** A text input field containing "25782250-...".
- Corpus Name:** A text input field containing "Test4".
- Allow Overwrite:** A checkbox that is checked.

Figure 5.32: Add Corpus Node

In the second speech to text node editor, we call this 'Get Corpora' and change the 'Detect' field to 'Get Corpora'. Again, we need to paste in the Customization ID and select 'Done'.

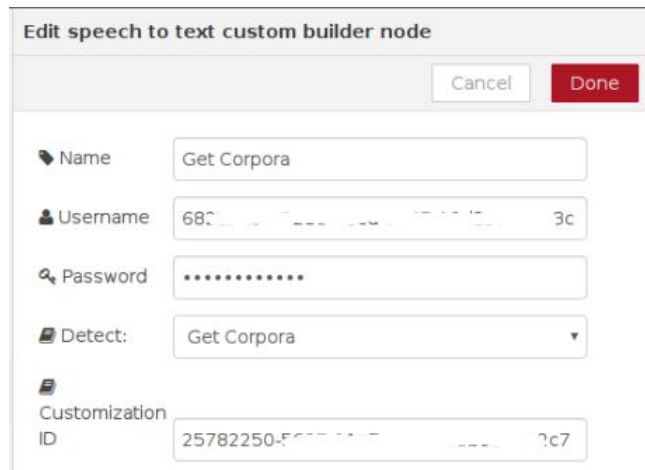


Figure 5.33 shows the 'Edit speech to text custom builder node' dialog box. The 'Name' field is 'Get Corpora'. The 'Username' field is '682...'. The 'Password' field is masked with dots. The 'Detect:' dropdown is set to 'Get Corpora'. The 'Customization ID' field is '25782250-5...'. There are 'Cancel' and 'Done' buttons at the top right.

Figure 5.33: Get Corpora Node

In the third speech to text node, we call this ‘Train’ and change the ‘Detect’ field to ‘Train’ and paste in the Customization ID.

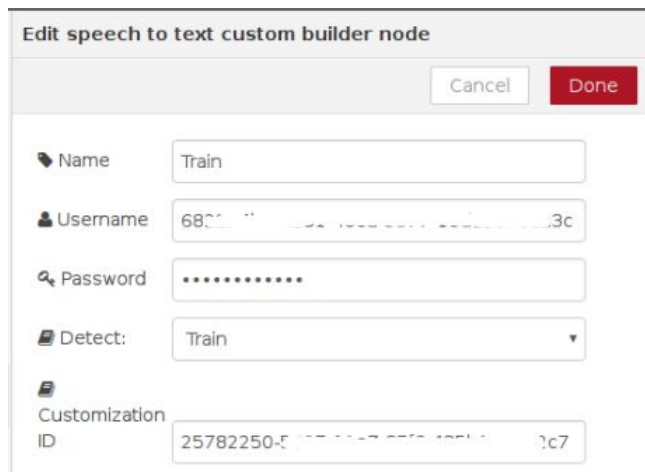


Figure 5.34 shows the 'Edit speech to text custom builder node' dialog box. The 'Name' field is 'Train'. The 'Username' field is '682...'. The 'Password' field is masked with dots. The 'Detect:' dropdown is set to 'Train'. The 'Customization ID' field is '25782250-5...'. There are 'Cancel' and 'Done' buttons at the top right.

Figure 5.34: Train Node

We now deploy our application. Then, we create a text file named appliances.txt to use as the Corpus. It contains sample sentences sample sentences applicable to our domain, providing a context for where the words can be used. We can keep building this text file and just need to

add corpus and train, everytime we add more use cases. Some of the sentences from appliances.txt are shown below,

- Can you please turn on the heater?
- Turn the heaters on.
- Turn on the lights.
- Turn the lights on.
- Turn on the music.
- Stop the heater.
- Switch the heater off.
- Switch off the light.
- Turn the lights off.
- Stop the music.
- My name is Shreyans Maloo.
- I work for KTH Live in labs.
- I study at KTH Royal Institute of Technology.
- My supervisor is Elena Malakhata.
- My examiner is Carlo Fischione.

We now, inject the appliances.txt file into the file inject node before ‘Add Corpus’. We may have to wait a while for the corpus to be processed.

Then we click the timestamp before ‘Get Corpora’ and we look at the debug tab for a status generated by ‘Get Corpora’ and that needs to read ‘analyzed’. If the status does not say ‘analyzed’ we might have to wait for few more seconds and then, click the timestamp again to check whether it is analyzed. This complete the step 2.

3. Add words to the custom language model

For this step, we add 1 file inject node (from the input section) and 1 speech to text custom build node (from IBM\_Watson section). We join these together and to the debug node.

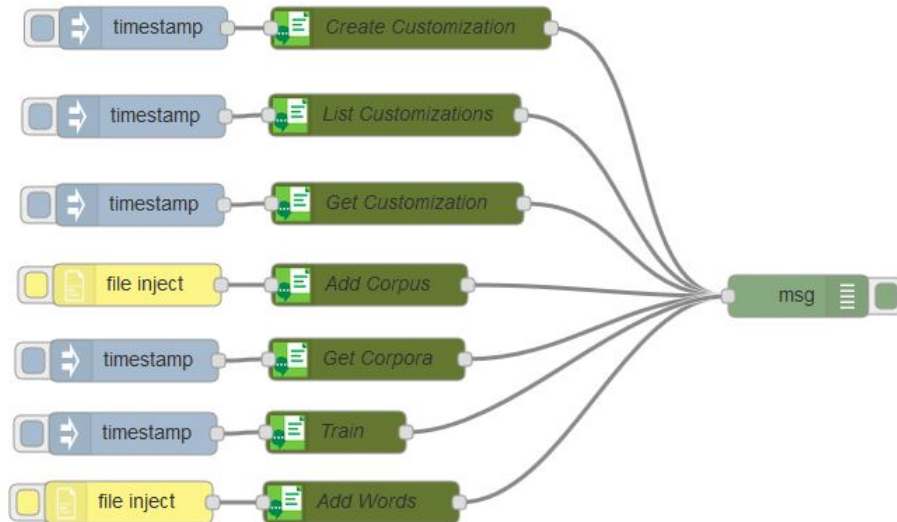


Figure 5.35: Node-RED flow for adding words and data to custom language model

We do not need to configure the file inject node. For the speech to text node, we can call it 'Add Words', change the 'Detect' field to 'Add Words' and paste in the Customization ID. Click 'Done' and deploy our app.



The screenshot shows a dialog box titled "Edit speech to text custom builder node". It has a "Cancel" button and a red "Done" button. The form contains the following fields:

- Name:** A text input field containing "Add Words".
- Username:** A text input field containing a long alphanumeric string: "6822...3c".
- Password:** A text input field with masked characters: ".....".
- Detect:** A dropdown menu with "Add Words" selected.
- Customization ID:** A text input field containing a long alphanumeric string: "25782250-5007...c7".

Figure 5.36: Add words node

Now we create a text file that includes pronunciation of some unusual words which might not be present in the lexicon of the service. The content of the file 'words.txt' looks like follows,

```
[ "word": "Shreyans", "translation": ["Shrae yaans"],
  "display_as": "Shreyans",
  "word": "Maloo", "translation": ["Maaloo"],
  "display_as": "Maloo",
  "word": "KTH", "translation": ["K T H"],
  "display_as": "KTH",
  "word": "Malakhatka", "translation": ["Malaa khaatkaa"],
  "display_as": "Malakhatka",
  "word": "Fischione", "translation": ["Fiskio nae"],
  "display_as": "Fischione" ]
```

We will be using this file in step 5.

#### 4. Train the custom language model

Here we just click the timestamp for 'Train' and wait for it to finish processing.

Note: We have not yet added the words, as we first need to compare the performance improvement just by adding data (sample use-cases) and then, we will add the words to improve the performance even further.

#### 5. Use the custom language model in a recognition request

We will now compare the difference between customized and non-customized results. To do this, we add 2 microphone nodes (from the input section), 2 speech to text nodes (from IBM\_Watson section) and 2 debug nodes (from output section) and join them together to look like the shown below. This can be above or below the existing nodes. At this point it is a good idea to refresh the page as the speech to text node may not present the 'Language Customization' field which we will need.

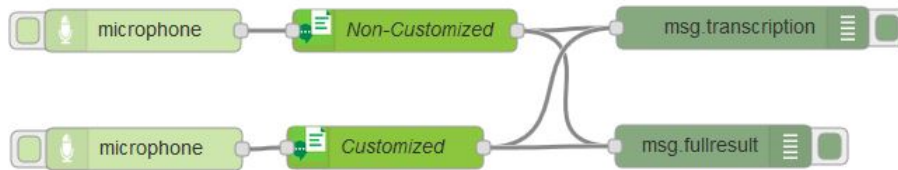


Figure 5.37: Node-RED flow for testing customized vs non-customized model

We do not need to do anything with the microphone nodes. For the first speech to text node, we name it 'Non-Customized', choose our language (US English) and select 'None' as the language customization. We select 'Broadband Model' for quality and check the continuous box. This will give us results without the customization.

For the second speech to text node, we must select US English or Japanese as these are currently only supported for the customization. We choose US English. Then, we choose the language customization we created in step 1 from the drop-down box (Bluemix04). We select 'Broadband Model' for quality and check the continuous box.

The screenshot shows a dialog box titled "Edit speech to text node" with a "Cancel" button and a red "Done" button. The form contains the following fields and options:

- Name:** A text field containing "Non-Customized".
- Username:** A text field containing "682274b-222-4444-9999-33c".
- Password:** A text field with masked characters "\*\*\*\*\*".
- Language:** A dropdown menu showing "US English".
- Language Customization:** A dropdown menu showing "None".
- Quality:** A dropdown menu showing "BroadbandModel".
- Continuous:** A checkbox that is checked.
- Speaker Labels:** A checkbox that is unchecked.

Figure 5.38: Non-Customized Speech to text node

The screenshot shows a dialog box titled "Edit speech to text node" with a "Cancel" button and a red "Done" button. The form contains the following fields and options:

- Name:** A text field containing "Customized".
- Username:** A text field containing "682274b-222-4444-9999-33c".
- Password:** A text field with masked characters "\*\*\*\*\*".
- Language:** A dropdown menu showing "US English".
- Language Customization:** A dropdown menu showing "Bluemix04".
- Quality:** A dropdown menu showing "BroadbandModel".
- Continuous:** A checkbox that is checked.
- Speaker Labels:** A checkbox that is unchecked.

Figure 5.39: Customized Speech to text node

For one of the debug tabs, we set the output to *msg.transcription*. This will output the text converted from speech. And the other debug tab output to *msg.fullresult*. This will output the full result of the speech to text function including confidence level.

Now we deploy our application and test the speech to text non-customized function by speaking a line from the appliances.txt file (preferably using words that are difficult to pronounce). Then we repeat the question using the customized function. Finally, we take note of the output from Watson in the debug tab. Here are a few examples passed through non-customized and customized respectively.

- There were cases where both non-customized and customized got the correct transcription, but the confidence level in customized was still higher (0.705 as compared to 0.429), as can be seen below. Use case: “Turn on the heater”.
- Then there were cases where non-customized misinterpreted some of the words whereas customized got the correct transcription, also the confidence level in customized was higher (0.88 as compared to 0.459), as can be seen below. Use case: “Turn off the light”.
- Then there were cases where we used some words which are out of vocabulary of the service (such as names). Here again the non-customized misinterpreted some of the words whereas customized got the correct transcription, also the confidence level in customized was higher (0.848 as compared to 0.781), as can be seen below. Use case: “My name is Shreyans Maloo and I’m doing my thesis under the supervision of Elena Malakhatka and my examiner is Carlo Fischione”.
- Finally, to handle such out of the vocabulary words, we can even train the service with the pronunciations of these words. We inject the words file (words.txt) as in Step 3 to the ‘Add Words’ inject node and repeat the same sentence. We observe an increase in the confidence level (it increases to 0.91). Use case: “My name is Shreyans Maloo and I’m doing my thesis under the supervision of Elena Malakhatka and my examiner is Carlo Fischione”.

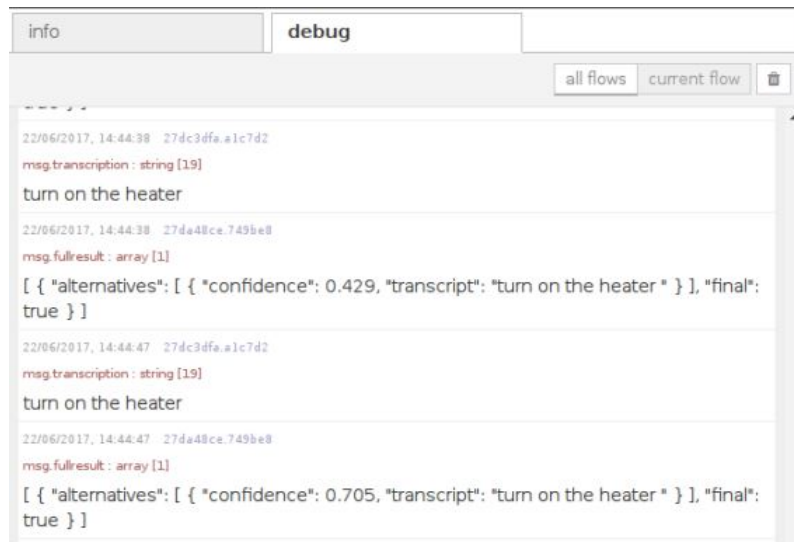


Figure 5.40: Non-Customized vs Customized example 1



Figure 5.41: Non-Customized vs Customized example 2

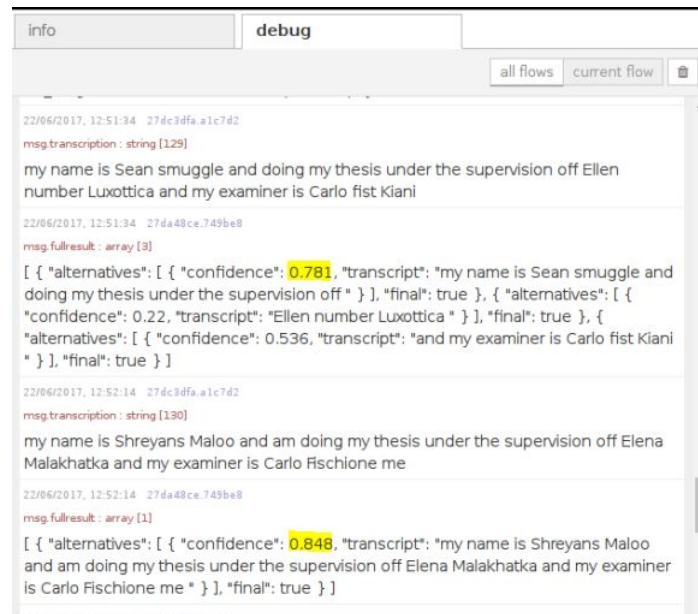


Figure 5.42: Non-Customized vs Customized example 3

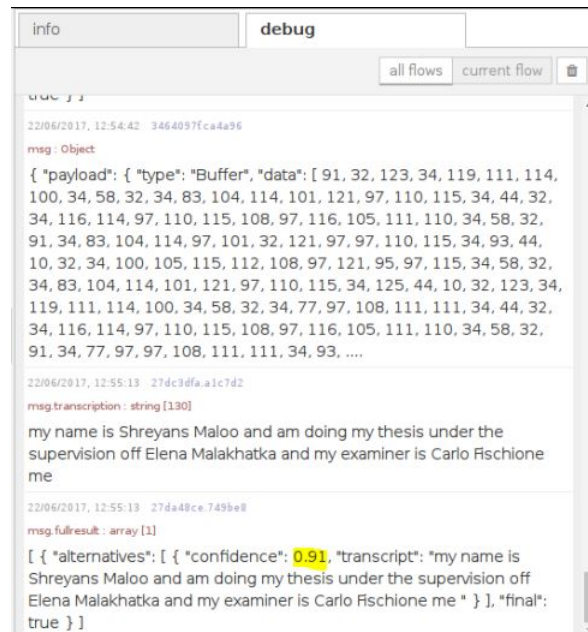


Figure 5.43: Performance after adding words

Finally, we try to measure the performance of prototype 2 for simple commands to see the improvement achieved from the previous prototype. This was done by, changing the STT in Node-RED flow of Prototype 1 by the Customized STT discussed above. The results are shown below,

Command	No. of tries	Correct output (LED on/off)
Turn on the light/s	6	6
Turn off the light/s	6	5
Switch on the light/s	6	6
Switch off the light/s	6	6
Turn on the heater/s	6	4
Turn off the heater/s	6	6
Switch on the heater/s	6	6
Switch off the heater/s	6	6
Turn on the music	6	6
Turn off the music	6	6
Play some music	6	5
Stop the music	6	6
	<b>72</b>	<b>68 (94.4%)</b>

Table 5.3: Performance - Prototype 2

The few errors, were again those of misinterpretation by STT, maybe due to human variations of accent or external influence of noise. To mention a few, we got “done on the heater” instead of “turn on the heater” and “it’s some music” instead of “play the music”. We believe, a knowledge of language and the context could help here, since the interpreted sentences makes no sense. Hence, we move to Prototype 3.

## 5.6 Prototype 3

The main component of prototype 3 is the conversation service module of IBM Watson. The conversation service integrates natural language understanding, to determine intent and entities, with dialog tools which allow for a natural conversation flow between our application and the users.

### 5.6.1 Conversation service

We start with creating a conversation. There is a tutorial available from IBM, which guides us through the whole process of launching the service, creating workspace, defining intents and entities and creating a dialog.

We will not be covering the steps of defining intents, entities and creating the dialog, but only the final conversation model we used in the project is represented by the pictures below,

- First, we define the intents we will be using in the dialog, “turn on”, “turn off”, “hello” and “goodbye”. Inside each category, we write various ways the user might intend for the task, for instance, for “turn on”, we write - “turn on the light”, “play some music”, “start the heater”, “can you switch on the light”, and others.

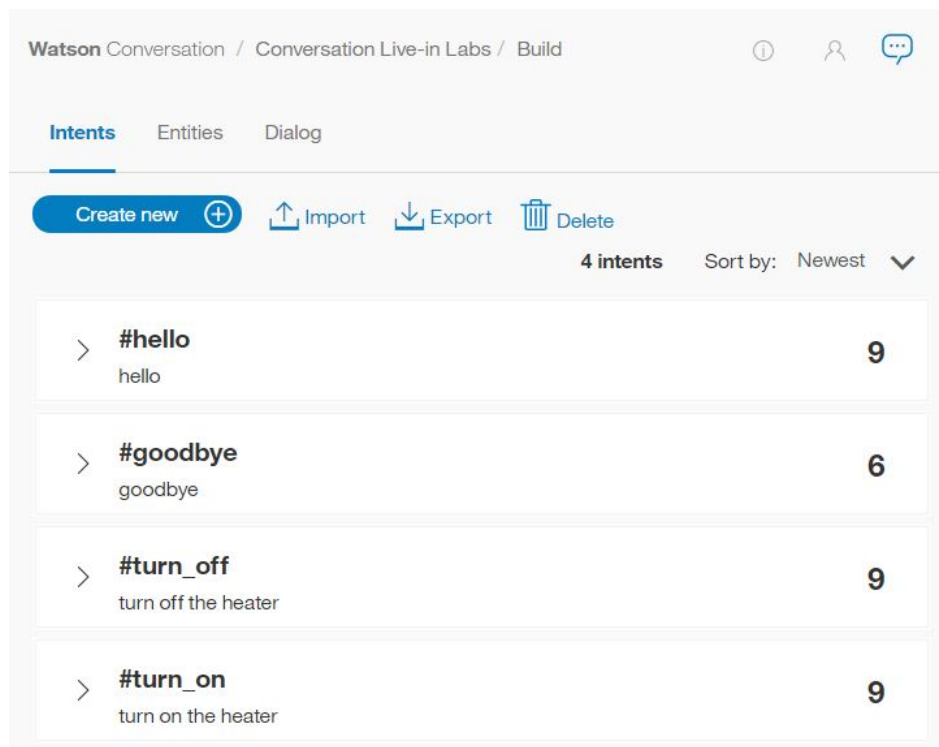


Figure 5.44: Conversation service intents

- Second, we define the entities we will be using in the dialog, “appliances” and “other appliances”. Inside each category, we write various



ways the user might mention the entity, for instance, for “music”, we write - “music system”, “stereo” and “radio”. There is a unique feature called, “Fuzzy Matching” which is in its BETA phase, we are still using it, as it is supposed to increase the ability of Watson to recognize misspelled entity values.

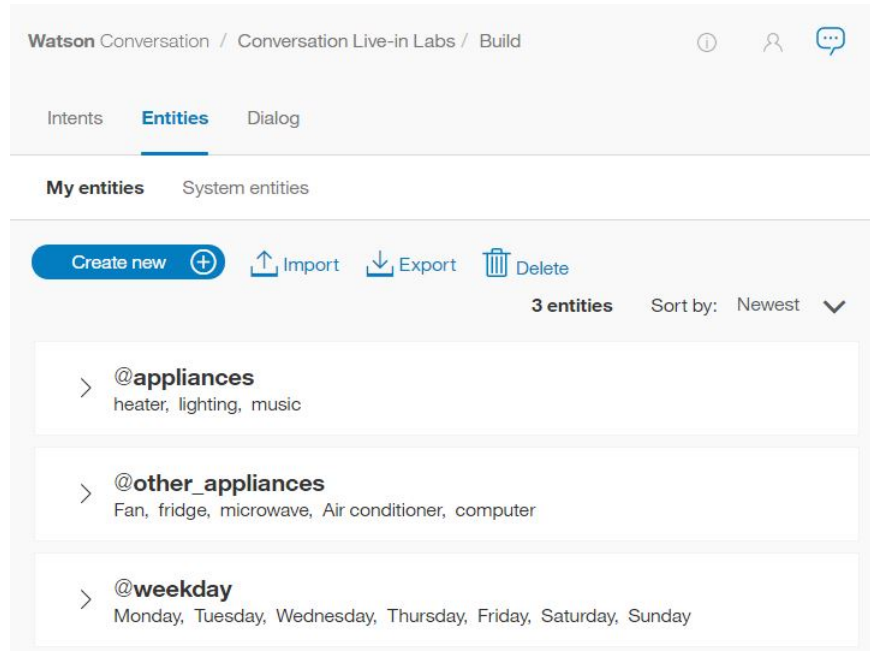


Figure 5.45: Conversation service entities



Figure 5.46: Contents of the appliances entity

- Now we move on to the dialog creation. The whole dialog looks like this,

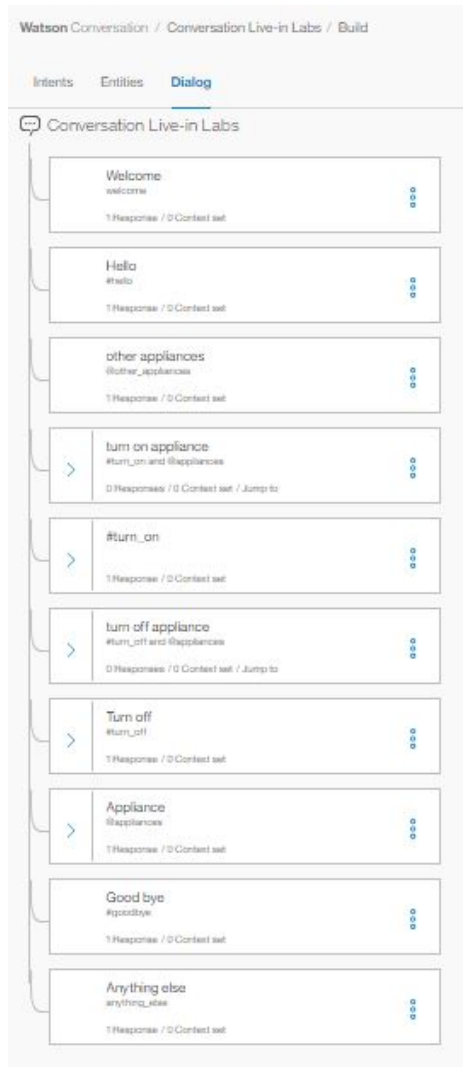


Figure 5.47: The conversation service dialog

We will be looking inside some of these nodes. First, the welcome node, it is by default a part of every dialog. We need to add some response, which is shown when a user enters a conversation. This will not be useful for our application. Second, we move on to the “Hello” node,

the contents of which are shown below. The purpose of this is to give a response, when the user greets it, saying, “hi” or “hello” or “good morning”. We can always add multiple responses to make the response more random, rather than the same reply everytime.

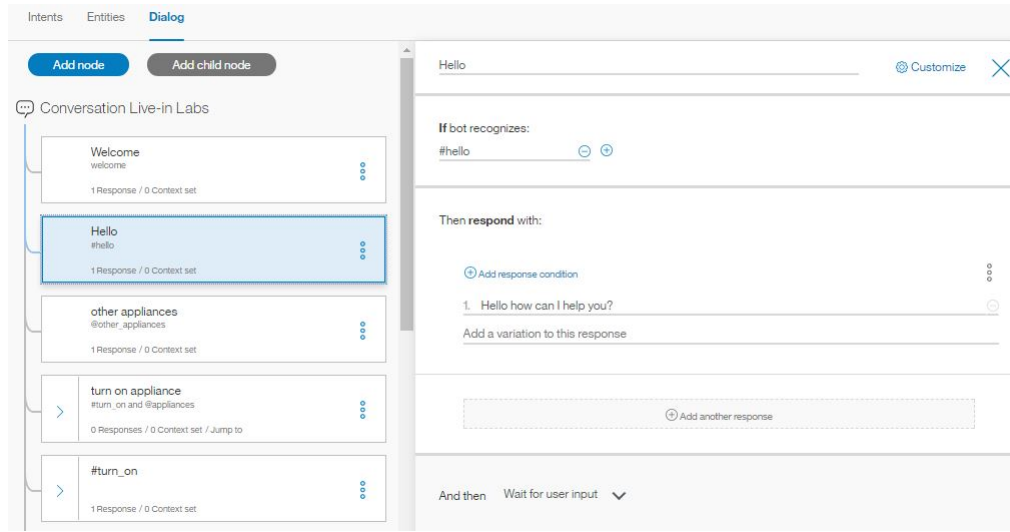


Figure 5.48: Contents of "Hello" node

Now, we move on to the “other appliances” node. The purpose of this node is to handle the commands, when the user asks to control an appliance other than light, heater or music. Since, the model is being designed only for these 3 appliances, we will give this feedback to the user. The contents are shown below,

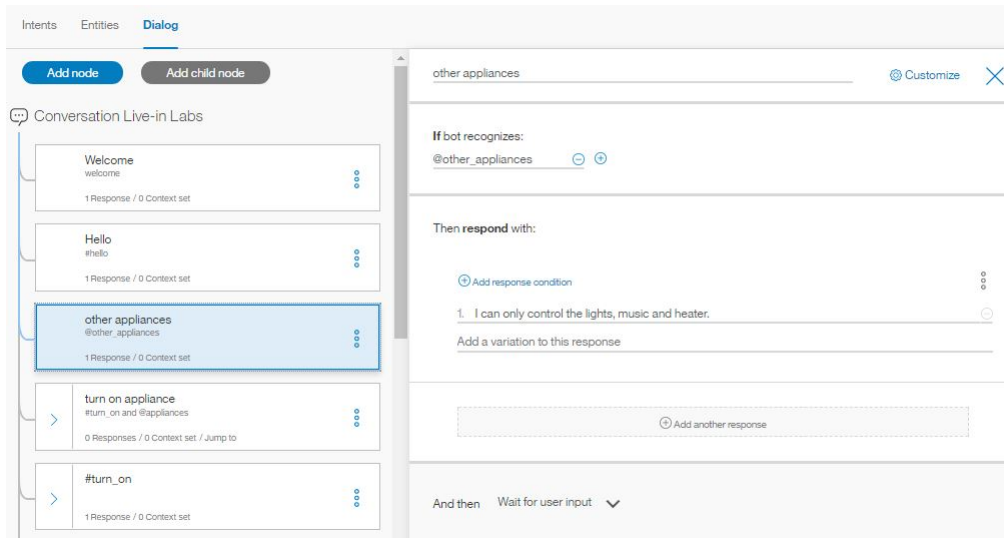


Figure 5.49: Contents of "other appliances" node

We then move to "turn on appliance" node. The purpose of this node is to check if the user has mentioned an intent of "turn on" along with one or more "appliances" entity. If yes, then it sends the control to the node – "All appliances on", it continues from there on. The contents are shown below,

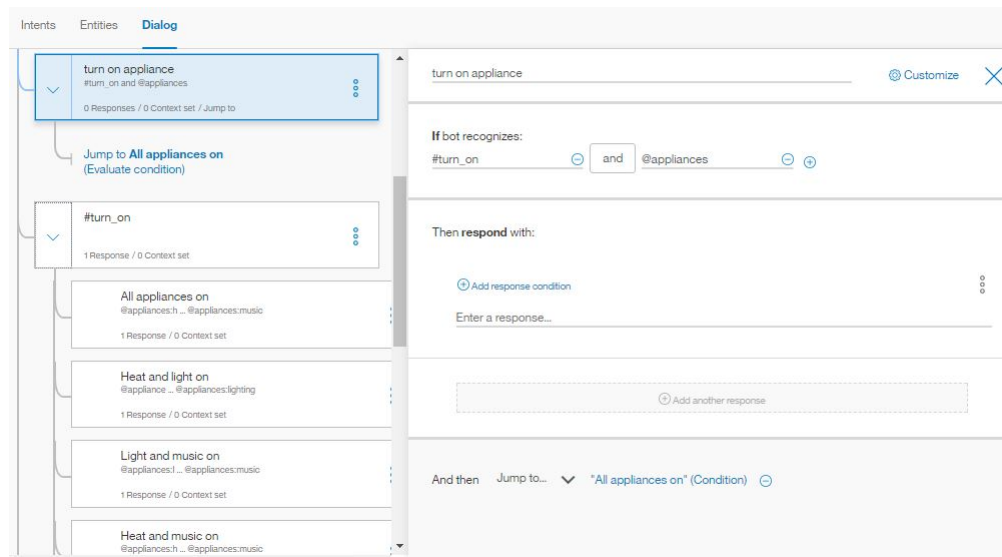


Figure 5.50: Contents of "turn on appliance" node

Now, we move to the next node, "#turn\_on". As the name suggests, we will look for the condition when, the user only mentions the intent of "turn on" but no entity along with it. We ask the user to rephrase the command and include an entity for the desired output.

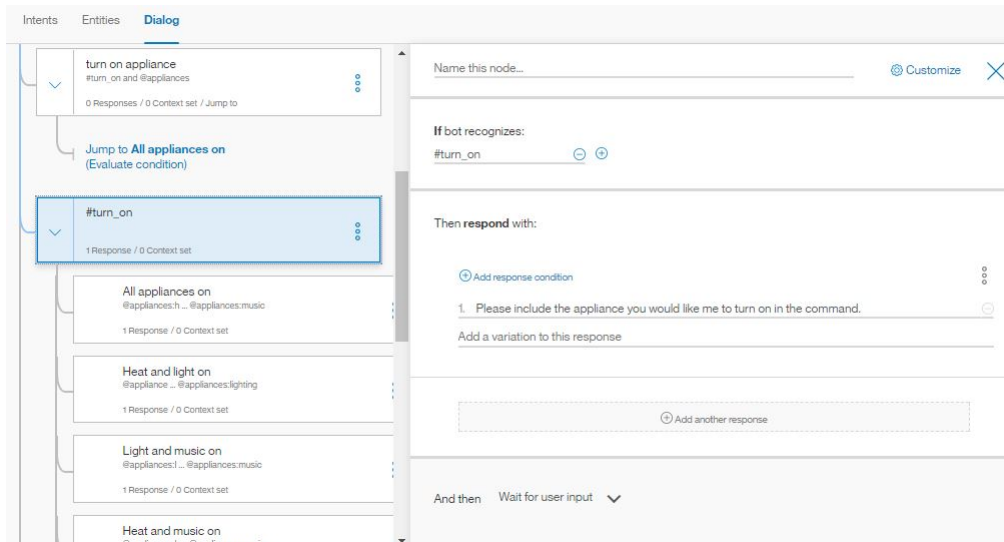


Figure 5.51: Contents of "turn on" node

Now, we handle the “appliance” entities, using the nodes “All appliances on”, “Heat and light on”, “Light and music on”, “Heat and music on” and “appliance on”. As the names suggest, “All appliances on” is to turn on all 3 appliances; “appliance on” is to control just one appliance which was mentioned, and other 3 are for controlling 2 appliances as mentioned in the node name. Below, we can see the contents of 2 of these nodes.

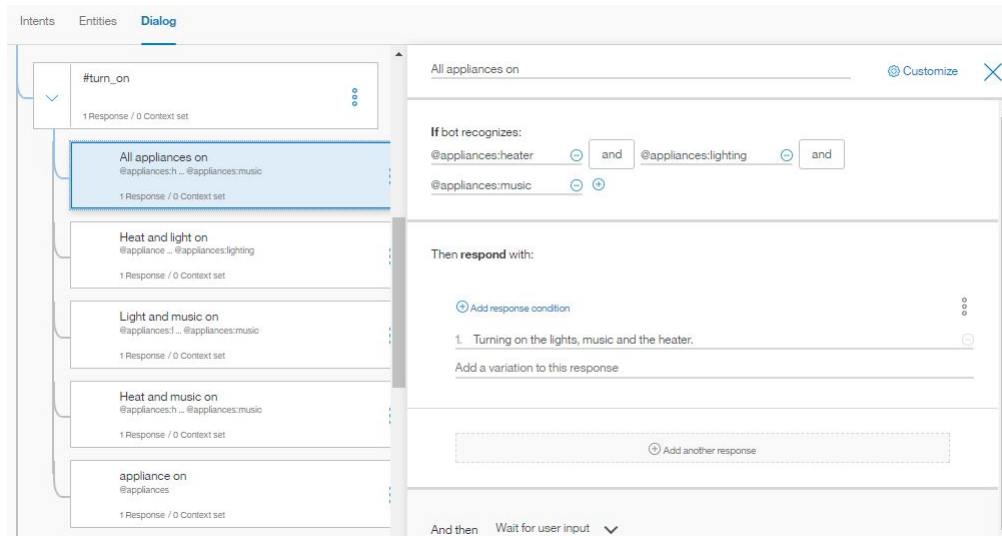


Figure 5.52: Contents of "All appliances on" node

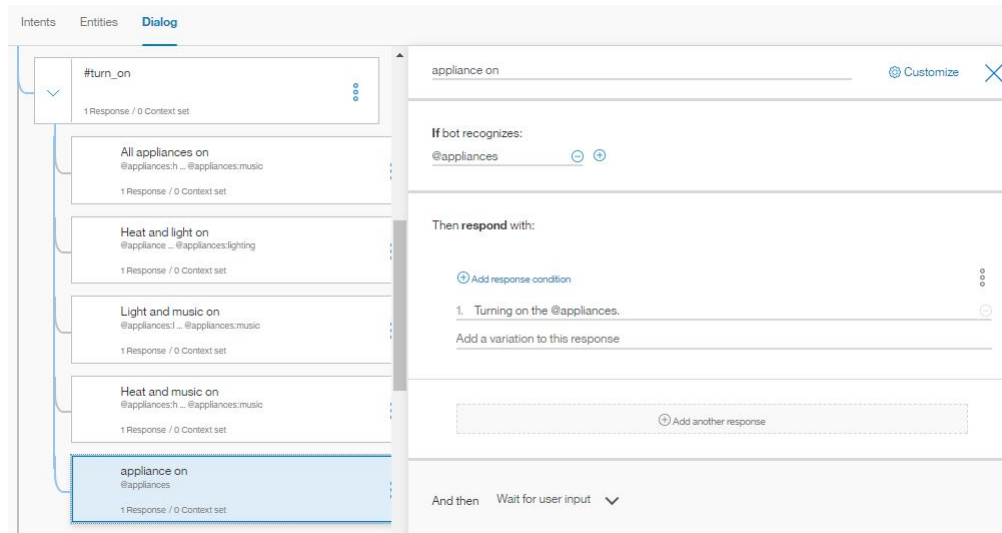


Figure 5.53: Contents of "appliance on" node

Similarly, we create nodes for turning off the appliances; "turn off appliances" and "turn off" consisting of similar "All appliances off", "Heat and light off", "Light and music off", "Heat and music off" and "appliance off".

Now, we move to the next node, “Appliance”. The purpose of this is to handle the case when the user just mentions an appliance name, but no intent attached to it “turn on” or “turn off”. We have a feedback mechanism, to ask the user to give complete commands.

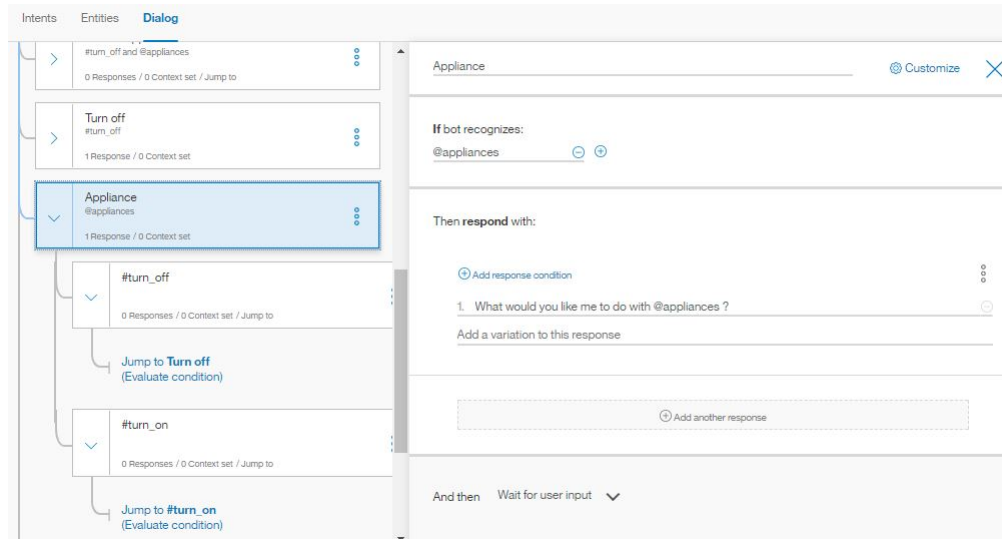


Figure 5.54: Contents of ”Appliance” node

Finally, we create a “Good bye” node. The purpose being, if the user gives an intent of “goodbye”, for instance, “goodbye”, “see you later”, “farewell”, and others, the model should be able to respond to that.



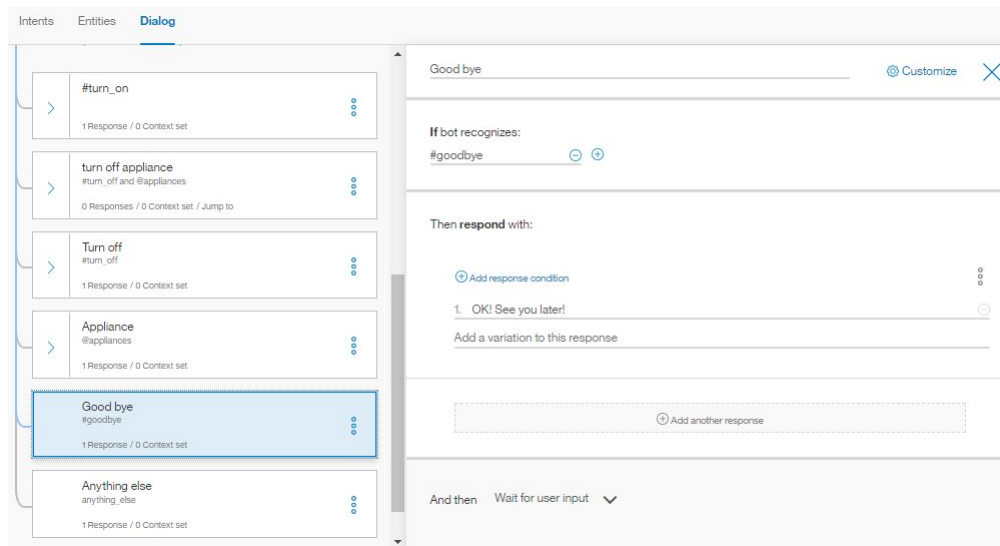


Figure 5.55: Contents of "Goodbye" node

Also, there is another node which is quite helpful to handle any command that does not meet any conditions in the whole dialog flow, called "Anything else". Here we add some responses to give a feedback to the user, so that he rephrases or changes his/her command.

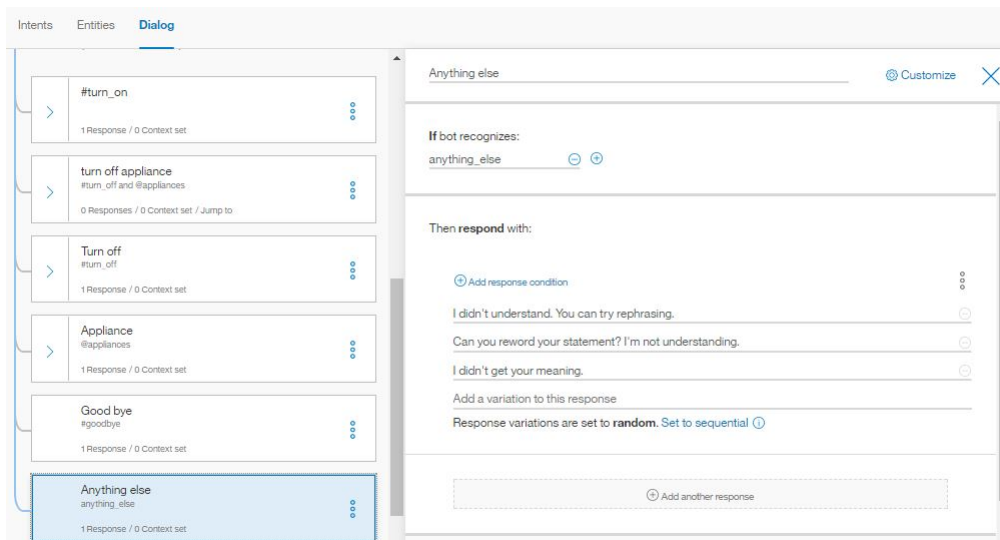


Figure 5.56: Contents of "Anything else" node

To conclude this sub-section, we add a snapshot of a small conversation with our model,

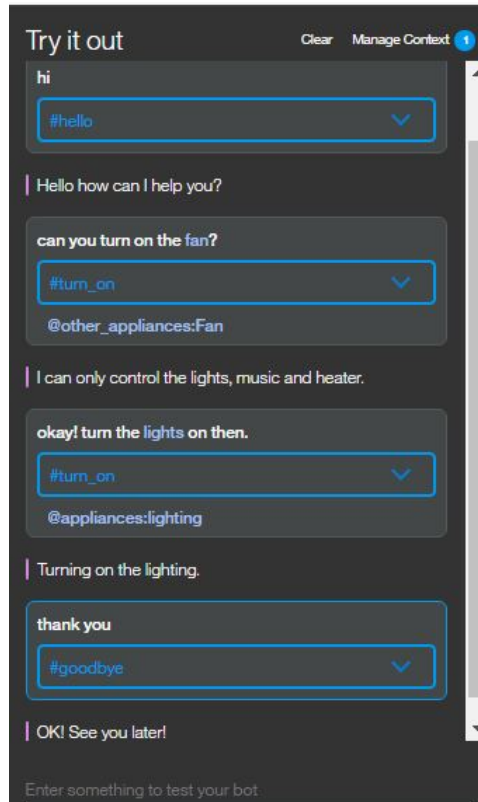


Figure 5.57: Snapshot of a dialog with the conversation service

One of the best features of conversation service, is the option to manually train the model. For instance, as we can see above (Fig 5.57), there is a drop-down menu, mentioning the intent or entity the model recognized for a command. If we think, it is wrong, we can change it to a different intent or entity or even mark it as irrelevant, the model learns from this, and next time a similar command appears it responds in the desirable manner. Once, the service is created, we move to Node-RED to use this service as a node.

### 5.6.2 Node-RED

We now build the flow for prototype 3 in Node-RED. We will be combining the flow from prototype-1, with using the customized STT as in prototype 2, and finally, we will even include the conversation service in the flow.

Everything is same as the prototype-1, except, we make some changes in STT node, the Task node and we add a couple more nodes. The Node-RED flow looks something like this,

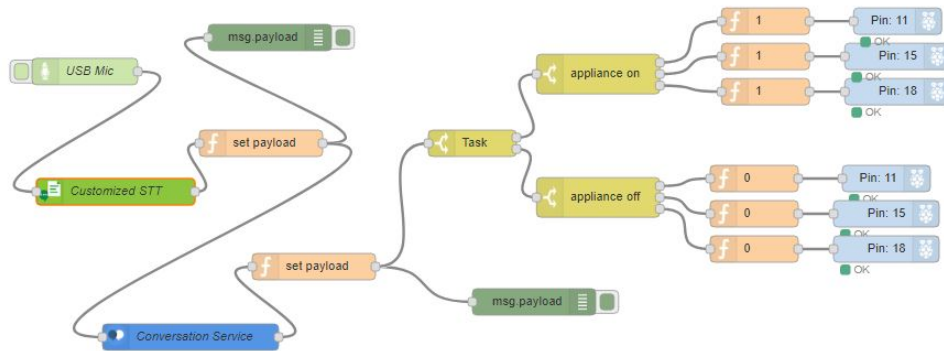


Figure 5.58: Node-RED flow for Prototype 3

The Customized STT node is same as the one we created for Prototype 2. So now, the speech is passed through the “Customized STT”, which is then passed through “set payload” as before. There is a change from the previous prototypes now, as we pass the payload through “Conversation Service” node. The contents of this node look like this,

Figure 5.59: Conversation service Node

The username and password can be obtained once we login in our IBM Bluemix dashboard and look for the conversation service in the list of services we have created. The workspace ID can be obtained, once we launch the conversation service tool in our browser and then look for the details of the workspace we are using for our application (Conversation Live-in Labs in our case).

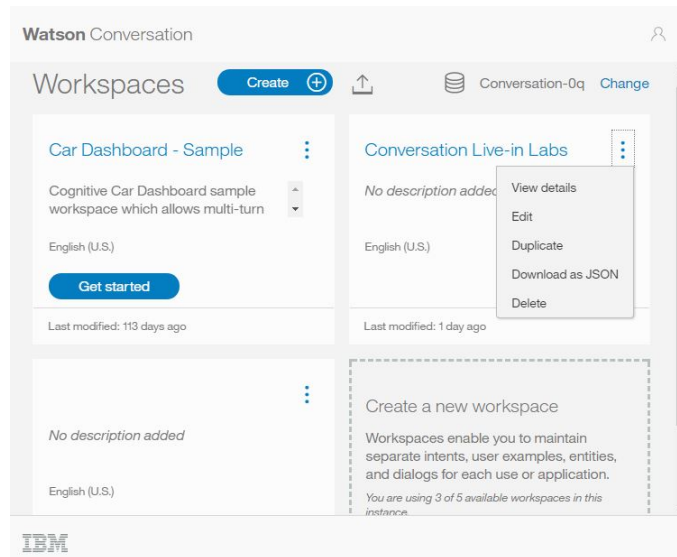


Figure 5.60: Watson Conversation Workspace details

Once these credentials are added, we are good to go ahead. We add

another “set payload” node, with contents as follows,  
*msg.payload = JSON.stringify(msg.payload.output.text);*  
*return msg;*

We need to convert the JSON output from the conversation service into string, for further processing. Now, this string goes in to “Task” node, where we segregate them based on the strings containing “Turning on” and those containing “Turning off”. After this, the flow is same as Prototype 1, where we look for appliances and turn them on/off as per the commands.

Now, the system can handle multiple appliances together, for instance, it can turn on/off 2 or all 3 appliances together if we want. Moreover, if the STT misinterprets some command, we can always teach the conversation service to respond in the desirable manner using the drop-down list and selecting the desirable entity or intent.

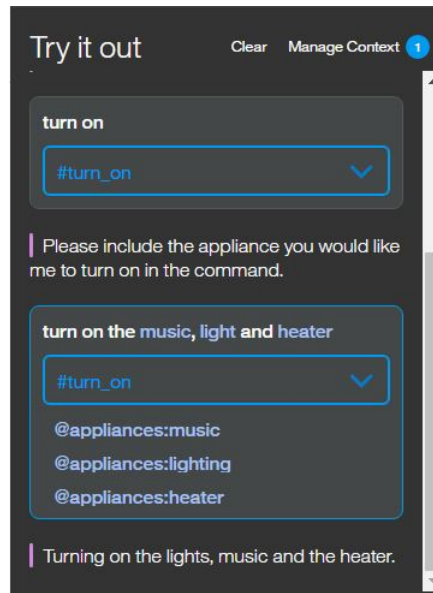


Figure 5.61: Conversation service handling multiple appliances

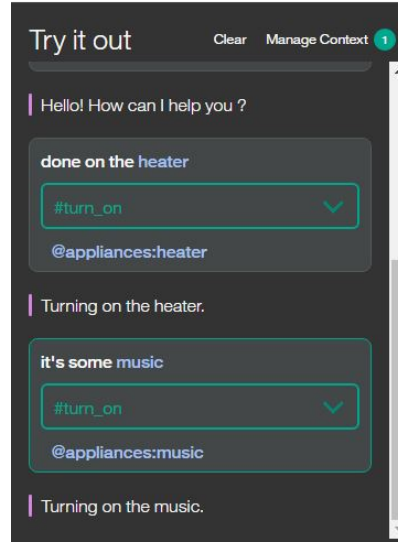


Figure 5.62: Conversation service handling misinterpretations from customized STT

Following the same simple command table, we run the tests for prototype 3 as well, and the outcome is as follows,

Command	No. of tries	Correct output (LED on/off)
Turn on the light/s	6	6
Turn off the light/s	6	6
Switch on the light/s	6	6
Switch off the light/s	6	6
Turn on the heater/s	6	6
Turn off the heater/s	6	6
Switch on the heater/s	6	6
Switch off the heater/s	6	6
Turn on the music	6	6
Turn off the music	6	6
Play some music	6	6
Stop the music	6	6
	<b>72</b>	<b>72 (100%)</b>

Table 5.4: Performance - Prototype 3

## Chapter 6

# Results and Discussion

Starting with the very first implementation of Prototype 1, the results were not good – only 68% success rate. We then moved towards hard coding certain outliers and could improve the results to 80% success, but still there were some drawbacks with this method – will fail for most of the variations. The performance of Prototype 1 against the failure factors is shown in the table below, In the following chapter the system will be undergoing an analysis by comparison with one standard used for certifying a healthy building.

Moving on to prototype 2, we included some more machine learning techniques in addition to the basic features available with the IBM Watson STT service. We added language models and some uncommon words along with the pronunciations; and trained the system with these. The performance increased as expected, success rate is 94.4%. The shortcomings remaining now would be, the system still does not understand the context of the command, for instance, if the user says “I am feeling cold”, the system should be smart enough to recommend turning the heaters on; no feedback loop has been implemented yet, which could make the system capable to ask for more information if needed. The performance of prototype 2 against the failure factors is shown below,

Finally, the prototype 3 was a further improvement to previous prototypes – 100% success rate for simple commands. Firstly, we had a better language model than before, allowing the system to understand the context better. Secondly, the option to train manually was a big advantage. We can now train the model to respond in desirable manner, for some common misinterpretations observed before. The performance of prototype 3 against the failure factors are shown below,

Category	Failure Factor	Able to handle
Speech to text	Variations in human speech	Yes. To some extent IBM Watson STT handles the variations.
	Large vocabulary continuous speech recognition	Not Applicable. We only dealt with short commands.
	Undefined word boundaries	Yes. To some extent handled by IBM Watson.
	Accent	Yes. To some extent handled by IBM Watson.
	External influence – noise, echoes	Yes. Noise-cancelling microphone
	Poor internet connection	Yes. Ensured stable connectivity, throws error if cannot connect to STT Service.
Text to keyword	Misinterpretation from STT	No. More robust STT needed.
	Fewer or no keywords from user	No. Missing Feedback mechanism
	Complex user command	No. More advanced logic engine needed.
Keyword to action	Communication error	Yes. Ensured stable communication; LED and debug mismatch acts as alert mechanism
	Poor sensor quality	Not Applicable.

Table 6.1: Evaluation - Prototype 1



Category	Failure Factor	Able to handle
Speech to text	Variations in human speech	Yes. Better than prototype 1, due to language model and uncommon words added.
	Large vocabulary continuous speech recognition	Not Applicable. We only dealt with short commands.
	Undefined word boundaries	Yes. Better than prototype 1, due to the inclusion of language model.
	Accent	Yes. Better than prototype 1, as user can add sounds of uncommon words in his/her accent.
	External influence – noise, echoes	Yes. Noise-cancelling microphone
	Poor internet connection	Yes. Ensured stable connectivity, throws error if cannot connect to STT Service.
Text to keyword	Misinterpretation from STT	No. But less misinterpretation than prototype 1, accuracy and confidence levels have increased.
	Fewer or no keywords from user	No. Missing Feedback mechanism
	Complex user command	No. But better than prototype 1, due to inclusion of language model.
Keyword to action	Communication error	Yes. Ensured stable communication; LED and debug mismatch acts as alert mechanism
	Poor sensor quality	Not Applicable.

Table 6.2: Evaluation - Prototype 2

Category	Failure Factor	Able to handle
Speech to text	Variations in human speech	Yes.
	Large vocabulary continuous speech recognition	Not Applicable. We only dealt with short commands.
	Undefined word boundaries	Yes. Better than prototype 1 and 2.
	Accent	Yes. Better than prototype 1 and 2.
	External influence – noise, echoes	Yes. Noise-cancelling microphone
Text to keyword	Poor internet connection	Yes. Ensured stable connectivity, throws error if cannot connect to STT Service.
	Misinterpretation from STT	Yes. We can keep training if an uncommon misinterpretation occurs.
	Fewer or no keywords from user	Yes. Feedback mechanism
Keyword to action	Complex user command	Yes. Better than prototype 1 and 2.
	Communication error	Yes. Ensured stable communication; LED and debug mismatch acts as alert mechanism
	Poor sensor quality	Not Applicable.

Table 6.3: Evaluation - Prototype 3

# Chapter 7

## Conclusion and Future Work

To conclude, we could achieve a good success rate when dealing with pre-defined short commands. The system could even handle words out of the common vocabulary, for instance, names of people, by adding them in the language model. But, there is still a lot of future work that can be done,

1. We can improve the “conversation” service module in IBM Watson to allow working with complex commands, where it should be able to remember the previous commands and build on that, for instance, if the user says “turn on”, the system asks for the appliance, and then if the user says “light”, it should turn it on, remembering the previous command was to turn on.
2. We can upgrade the push to talk feature to a wake-up word.
3. We need to improve the Node-RED flow such that we can handle different tasks together, for instance, the user asks for controlling 2 different appliances in different ways, for instance, “turn the lights on and stop the music”.
4. We can even work on providing a voice feedback mechanism instead of or in addition to text. Another service of IBM Watson, Text to Speech can be used for this purpose.
5. The last step, would be to try it on with the real sensors and controller. This real implementation shall begin with a smart room available at the “Q-building” in the KTH campus. Once that gets approved, it can be installed in KTH Live-in Lab.

# Bibliography

- [1] Kudryavtsev A (Jan, 2016). *Automatic Speech Recognition Services Comparison*. [Blog] Available at: <http://blog-archive.griddynamics.com/2016/01/automatic-speech-recognition-services.html> [Accessed 03/04/2017]
- [2] Vuylsteker B (Feb, 2017). *Speech Recognition - A comparison of popular services in EN and NL*. [Blog] Available at: <https://blog.craftworkz.co/speech-recognition-a-comparison-of-popular-services-in-en-and-nl-67a3e1b0cee6> [Accessed on 03/04/2017]
- [3] Averin P (Apr, 2016). *Voice Recognition Tools Review*. [Blog] Available at: <https://www.netguru.co/blog/voice-recognition-tools-review> [Accessed on 03/04/2017]
- [4] Google Cloud Speech API. [online] Available at: <https://cloud.google.com/speech/> [Accessed on 28/04/2017]
- [5] Saon G (March, 2017). *IBM achieves new record in speech recognition*. [Blog] Available at: <https://www.ibm.com/blogs/research/2017/03/speech-recognition/> [Accessed on 28/04/2017]
- [6] Kavanagh J (Feb, 2017). *Which smart home automation protocol is right for you?* [online] Available at: <http://www.tomsguide.com/faq/id-3218019/smart-home-automation-protocol.html> [Accessed on 26/04/2017]
- [7] Electronic House (Jan, 2016). *Home Automation Protocols: A Round-Up*. [online] Available at: <https://www.electronichouse.com/smart-home/home-automation-protocols-what-technology-is-right-for-you/> [Accessed on 26/04/2017]
- [8] Ravishankar, M (May, 1996). *Efficient Algorithms for Speech Recognition*. PhD thesis, Carnegie Mellon University.
- [9] Pallett, D (Sept, 1985). *Performance Assessment of Automatic Speech Recognizers*. National Bureau of Standards.
- [10] GPIO: MODELS A+, B+, RASPBERRY PI 2 B AND RASPBERRY PI 3 B [online] Available at: <https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/> [Accessed on 20/05/2017]

- 
- [11] *Speech to Text Customization*, GitHub repository [online] Available at: [https://github.com/watson-developer-cloud/node-red-labs/tree/master/basic\\_examples/speech\\_to\\_text\\_custom](https://github.com/watson-developer-cloud/node-red-labs/tree/master/basic_examples/speech_to_text_custom) [Accessed on 26/06/2017]
- [12] Obaid, T. and others (Feb, 2014). ZigBee based voice controlled wireless smart home system. *International Journal of Wireless & Mobile Networks (IJWMN)* Vol. 6, No. 1.
- [13] Toschi, G. Campos, L. and Cugnasca, C (Feb, 2017). Home automation networks: A survey. *Computer Standards & Interfaces* 50 (2017) 42-54.
- [14] A. Kumar, A. Mihovska, S. Kyriazakos, R. Prasad. Visible light communications (VLC) for ambient assisted living, *Wireless Personal Commun.* 78 (3) (2014) 1699–1717. Available at: <http://dx.doi.org/10.1007/s11277-014-1901-1> 00002. URL - <http://www.link.springer.com/article/10.1007/s11277-014-1901-1>
- [15] D.J. Fagnant, K. Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations, *Transp. Res. Part A: Policy Pract.* 77 (2015) 167–181.

TRITA EECS-EX-2018:46  
ISSN 1653-5146

[www.kth.se](http://www.kth.se)