B.E. Project Report

On

ROBOT MOVEMENT USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING

By

RISHIKESH SARJERAO BAMDALE (B150368508)

SHREEJEET SAHAY . (B150368615)

SONY SUNIL VANZE (B150368661)

Under the guidance of

PROF. V.S. KHANDEKAR

SINHGAD TECHNICAL EDUCATION SOCIETY-PUNE

तमसो मा ज्योतिर्गमय

Sinhgad Institutes

Department of Information Technology

Smt. Kashibai Navale College of Engineering, Pune-41

Accredited by NBA

SAVITRIBAI PHULE PUNE UNIVERSITY

2018-2019

Sinhgad Technical Education Society,

Department of Information Technology

Smt. Kashibai Navale College of Engineering , Pune-411041

Date:

CERTIFICATE

This is to certify that,

RISHIKESH SARJERAO BAMDALE (B150360508)

SHREEJEET SAHAY . (B150368615)

SONY SUNIL VANZE (B150368661)

of class B.E IT; have successfully completed their project work on "ROBOT MOVEMENT USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING", at Smt. Kashibai Navale College of Engineering, Pune in the partial fulfillment of the Graduate Degree course in B.E at the Department of **Information Technology**, in the academic Year 2018-2019 as prescribed by the Savitribai Phule Pune University, Pune.

Prof. V.S. Khandekar
Project Guide

Prof. R.H.Borhade
Head of the Department
(Department of Information Technology)

Prof. V.S.Khandekar
B.E. Div - 2
Project Coordinator

(External Examiner)
Savitribai Phule Pune Univeristy, Pune

Dr. A.V. Deshpande
Principal
Smt. Kashibai Navale College Of Engineering, Pune

ABSTRACT

Robotic movement is one of the most important matters of concern in the majority of industries today. Traditionally, Robots need to be given specific set of commands in order to carry out tasks, that have to be constructed via precise patterns and rules to dictate robots to correctly perform actions.

All the currently existing solutions like Voice Recognition Processor(VRP), Microsoft Speech SDK 5.1, etc translate by using word-to-word translation principle, and neither do they take into account the context of the full message, nor do they consider the expandability of the command text. These methodologies are tedious, and highly non-intuitive in nature.

This project work utilizes the potential of Natural Language Processing, and Machine Learning for the task of Human-Machine communication. The main aim of this project work is to propose a system that effectively translates a Natural Language input into a set of robot path trajectory commands, considering both the expandability of command text and context of the full message. The integration of machine learning allows the system to endlessly learn, and adapt to new commands in future.

Hence, the proposed system consists of a pre-processing function, command classification, parameter classification, and post-processing function, thereby utilizing the Neural Machine Translation(NMT) approach by using RNN encoding-decoding for classification purposes.

ACKNOWLEDGEMENT

It has been great honour and privilege for us to deliver project in the domains MACHINE LEARNING and NATURAL LANGUAGE PROCESSING, on the topic, "ROBOT MOVEMENT USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING".

We would like to take opportunity to express our humble gratitude to our guide, PROF. VARSHA S. KHANDEKAR, under whom we executed working on these seminars. Her constant guidance and willingness to share her vast knowledge made us understand this project topic and its manifestations in great depths and helped us to complete the assigned tasks.

Perfection is impossible to achieve, but hard work and dedication can achieve any goal. We have tried our level best to fulfill the requirements of the project-based seminar, but we could not have achieved our goal without the able guidance of HOD, Information Technology Department.

We wish to express our whole hearted thanks to our Principal for providing the modernized lab facilities in our institute.

We are also grateful to the members of the management of our college, who co-operated with us regarding some issues.

We are very much thankful to the lab attendants of our department, for providing all facilities and support to meet our requirements.

Finally, we convey our acknowledgement to all our friends and family members who directly or indirectly associated with us in the successful completion of the project.

We thank one and all.

RISHIKESH SARJERAO BAMDALE (B8109)
SHREEJEET SAHAY. (B8224)
SONY SUNIL VANZE (B8266)

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IEEE	INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS
IJCMI	INTERNATIONAL JOURNAL OF COMPUTATIONAL MATHEMATICAL IDEAS
NMT	NEURAL MACHINE TRANSLATION
RNN	RECURRENT NEURAL NETWORKS
SDK	SOFTWARE DEVELOPER’S KIT
VRP	VOICE RECOGNITION PROCESSOR
HMM	HIDDEN MARKOV MODELS
LSTM	LONG SHORT-TERM MEMORY
DNN	DEEP NEURAL NETWORK
API	APPLICATION PROGRAM INTERFACE
GNMT	GOOGLE’S NEURAL MACHINE TRANSLATION
GUI	GRAPHICAL USER INTERFACE
NLTK	NATURAL LANGUAGE TOOLKIT

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Problem On Hand

Automatic speech recognition by machine has been a goal of research for a long time. The First speech recognition come into existence in 1952 made of single spoken digits and with the help of acoustic signals. Since the invention and rise of robots, they are co-existing with humans, also heavily used in Industry than household purposes. So, communication and speech recognition is very demanding and futuristic research area, due to importance of communication with them. Nevertheless, to use a robot you have to communicate with it properly to give commands for tasks. The commanding of robot is most challenging task since rise of them.

We want to keep this communication as simple as possible as we do it with humans. To keep it crisp and efficient we should interact with them in natural way. Current scenario of communication is explicit programming and automation such as VRP (Voice Recognition Processor), Microsoft SDK, APIs. So Problem on Hand is nothing but to use effective methodologies to have commanding of robot movements in natural way, particularly in human language for efficient communication between human and robots.

1.2 Basic Concepts

Currently, robot commands for movement are given in precise patterns and rules for detection of their actions for movement and thereby completion of tasks. Natural Language Processing (NLP) and Machine Learning (ML) make this task effective, so we don’t need particular hardware and there is no use of Machine-level language.

Natural Language Processing with Machine Learning integration solves the problem of communication for commanding robots and we can process human-language with different NLP and ML techniques and algorithms. The ultimate concept and goal is, it should be easy for human-operator to simply tell the robot naturally in human language instead of programming it.

1.3 Project Objective

The important thing to achieve in this project and research is natural communication between human and robots, using Natural Language Processing. By integrating Machine Learning approach, the system is to put on endless-learning for adapting new set of commands. Neural Machine Translation with the help of Encoder-Decoder Algorithm technique is to be implemented. To achieve maximum accuracy for producing accurate robot command is also a part of project objective.

Hence, the objective of the project is to find precise trajectory for the movement of Robot based on Natural language input (Human language). A task of robot movement is nothing but the context of this project and research work. The final output of this work is the robotic command with command classification and parameter in form of vector, which will be used by robot for accurate movement for performing tasks.

1.4 Scope of Project Work

Nonetheless what are the objectives, Scope of Project is essential factor for deciding boundaries for the project and research work. The communication language for robot should be in English as a Natural-Language input. Currently, 6 different commands are to be implemented- ‘go’, ‘turn’, ‘wait’, ‘follow’, ‘stop’, ‘Speed’ . Also, when we give command in natural language to robot, the parameters should be implicitly mentioned in it for action or task.

1.5 Application

This project is basically application of Natural Language Processing with few Machine Learning (ML) techniques like NMT and Encoder-Decoder Algorithm. Further application of this project mainly lies in Industrial use of Robots for performing different mechanical tasks like welding, painting and assembling. Also this project and work is useful in basic movement for robot in house for basic tasks like small load transfer from one room to another room, cleaning of the house, in gardening and watering of plants.

Apart from above applications, there is a much more scope of future research involved in this project to enhance the capabilities and application of commanding robot using natural language for movement. Further this work can be used to add more commands for advanced usage like in for Space Technology and much more advanced fields.

CHAPTER 2
LITERATURE SURVEY

2.1 INTRODUCTION

Various old methodologies and different papers were studied, we found NLP with Encoder-Decoder (NMT) as the best approach for this implementation. Robots now-a-days are getting familiar with human day by day and this is the reason why we need a simple and robust system to communicate with them, as they're helping with daily human activities and also in industries. Robotic movement and Natural Language Processing has always been a subject of research since years and we found there is a lot of scope for exciting research to do.

Every previous implementation has some Pros and Cons, it's important to study them before we dive into our actual proposed methodologies.

TABLE 2.1. Literature Survey

Sr. No.	Reference	Description	Pros	Cons
1.	Commanding Mobile Robot Movement based on Natural Language Processing with RNN Encoder-Decoder (IEEE 2018)	Implementation of robotic movement using NMT	Use of NMT for classification of commands	Accuracy of this implementation is low
2.	Speech Recognition of Industrial Robot (IJCMI 2011)	Translation using HMM	HMM is robust method and accuracy is good	Memory is more used in this paper.
3.	Google's Neural Machine Translation System:	Human-to-machine	Accuracy is improved through GNMT, more	The architecture is more complex to implement.

	Bridging the Gap between Human and Machine Translation (IEEE 2016)	translation using Google's NMT	hidden layers are used.	
4.	Robot-by-voice: commanding an industrial robot using the human voice (Industrial Robot: An International Journal 2005)	Translation using Microsoft Speech SDK 5.1	Direct development of speech recognition system using Microsoft Speech SDK 5.1 application	Time consuming adjustment of process variables
5.	A voice command system for autonomous Robot Guidance (IEEE 2006)	Translation using VRP	Complexity of commanding system solved by using VRP	High Cost
6.	Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation (IEEE 2014)	RNN Encoding-Decoding implementation for learning linguistic phrase representation	High performance in statistical machine translation	Gradient vanishing problem
7.	Sequence to sequence learning with neural networks (IEEE 2014)	Use of DNN Encoder-decoder and LSTM for spoken language translation	High performance in sequence problems of machine learning	Requires a large amount of data

2.2 Related Works

In [1], the approach for human-machine translation is different from older methods which includes Voice Processor Recognition (VRP), Microsoft SDK and APIs which were inefficient. Neural Machine Translation (NMT) is primarily used as an more effective technique to solve problem of commanding for movement. In NMT, the input already consists of sequence of symbols in some language and the other computer program must convert this into sequence in another language. Here one language is natural English language and other is is robot command. The Recurrent Neural Network (RNN) Encoder-Decoder which is well-known approach of NMT is used for encoding and decoding of input message into meaningful vector. This, this work utilizes the concept of Natural Language Processing (NLP) and Machine Learning. In [2], the speech recognition is implemented on a processor, the no. of words (commands) to be executed depends on amount of memory allocated for storage. The processor used was HM2007 as a Voice Recognition Processor (VRP). HMM, which is one of the speech recognition approaches used for statistical pattern recognition. The disadvantage or limitation of this implementation is that, it’s based on word-to-word translation and also the command set depends on the memory of processor used in implementation. In [3], emphasis is on importance of Neural Machine Translation implemented by google over Statistical Machine Translation (SMT) i.e. Phrase-based translation. In this implementation, the recurrent networks are LSTM RNNs. These RNNs have 8 layers to encourage gradient flow. To process rare-words it uses sub-word units for inputs and outputs in our system. It reflects the robust implementation and performs well on huge data set with many pair of languages. This work presents the design and implementation of GNMT, a production NMT system at Google, that aims to provide solutions to the word-recognition problems. In this implementation, the recurrent networks are Long Short-Term Memory (LSTM) RNNs. Their LSTM RNNs have 8 layers, with residual connections between layers to encourage gradient flow. For parallelism, there is connection and the attention from the bottom layer of the decoder network to the top layer of the encoder network. To improve inference time, they employ low-precision arithmetic for inference, which is further accelerated by special hardware (Google’s Tensor Processing Unit, or TPU).

The history of making robots goes back to ancient times, but it wasn’t until the 1990s, the conversational robots MAIA, RHINO and AESOP started to appear, though in 1968, the first electronic speech synthesizer made in Japan. Today, the so-called social robots like Sophia or Poppy are capable of talking with humans, but still they are limited to many applications and not used widely. In early years systems, there was a Polly, which was made to help with touring the offices. The Polly used pre-defined answers and actions to people's response like 'start the tour'. A slightly more advanced TJ, was smart enough to respond to commands like 'go right', 'go left'. RHINO who was intelligent to respond to simple commands, but the mechanism was a fixed-programming defined verbal description. MAIA built to obey some commands and for delivering parcels, but similar in methodologies. The classical book, talks much about traditional human-robot interaction and natural-language processing methodologies Like semantic parsers which are theoretical approaches to solve Human-Robot Interaction problems. It also talks about the implemented systems to reduce the gap between problem and solutions.

The early development of robust implementation of mobile robots is also much contributed by NASA. Examples of early successes are Soviet Lunokhods and recent exploration of Mars's Surfaces. The Baxter assembly line robot which is the product of Rethink Robotics in Boston, they have extensively developed and done research in industrial robots i.e. telerobots. Defense Advanced Research Projects Agency's International Robot Challenge encompasses the exclusive techniques with respect to interactions of human with robots. MIT developed Kismet which is a "social-interactive" head which makes appropriate gestures based on interpretation of voice and face of another person. Mattel has developed a robot-doll i.e. "Barby Doll" having extraordinary speech and language recognition and it is connected to the company's server also. Research presents human feedback method for learning interaction for robots. Apple's SIRI eases the methodology of commanding with computer-based voice understanding and thereby taking a Human-Robot Interaction to a different level but still not in a real natural Methodologies for interactions in helping elder people in hospital.

Interestingly in 1980, LOGO language is developed by Papert to teach children programming. It shows that Human-Robot interaction is also used in the teaching activities which was a superior move at that time. Commanding in interaction was implemented through Voice Recognition Processor. Also, the implementation of an interactive system through Microsoft Speech SDK. Also, speech recognition of industrial robots is done through Hidden Markov Models. All these systems didn't use natural-language processing methods and they are all costly. Google's Neural Machine Translation system can be incorporated in Human-Robot Interaction, but still, it has not happened.

From all the above work, we can see that Human Robot Interaction systems, though not perfect, were always been a part of research since the early years. We expect to see a lot of inventions in interaction, based on natural-language processing which will be useful in applications mentioned in I. Conferences regarding HRI such as IEEE International Symposium on Robot & Human Interactive Communication since 1992 and ACM International Conference on HRI since 2007, have seen many papers and work, which show a tremendous amount of research is being done in Human Robot Interaction field.

To effectively deal with rare words, it uses sub-word units (also known as “word pieces”) for inputs and outputs in our system. Using word pieces gives a good balance between the flexibility of single characters and the efficiency of full words for decoding, and also sidesteps the need for special treatment of unknown words. It describes the in-detail implementation of Google’s NMT (GNMT) system which removes the disadvantage of typical Neural Machine Translation on accuracy, speed and robustness. Compared to previously phrase-based production system, this GNMT system delivers roughly a 60% reduction in translation errors on several popular language pairs.

So, at the last if we compare all this systems, VRP was the oldest one and currently GNMT is more advanced and accurate, but it's more complex. GNMT has more hidden layers, so efficiency increases slightly but slowness comes into picture for system and overall the architecture becomes complex for learning and implementation also. VRP was based upon acoustic signals and hardware and it uses more memory because of its physical implementation. However, simple RNN Encoder-Decoder with two hidden layers is simple and robust but it lacks accuracy. Accuracy is important parameter while comparing all these methodologies. For our research, we need basic, simple and more accurate NLP-based system for finding exact robotic movement.

In [5], a hardware-based approach for voice command recognition using Voice Recognition Processor (VRP) was proposed. While the VRP based approach is simple and compact, it relies on the word by word translation and has certain limitations in terms of expandability of command set. In [2], a speech recognition for an industrial robot has been implemented. The authors successfully implement a speech recognition system for an industrial robot with 7 unique commands where the response time of robot is within milliseconds. While this result shown a good performance, it still relies on the word by word translation principle. Moreover, the spoken command format is also highly specific and it does not provide the possibility to pass along the additional parameters with the spoken command. In a contrary, the work in [4] also present a voice command solution for industrial robot based on Microsoft Speech SDK

5.1. While the proposed system can accommodate both command and parameters, the command format is predefined and it is still based on the word by word translation principle which limits the use of natural language for the human operator. Recently, an approach that take the whole message as input and then translate it into an output message, so called the NMT, has shown better results compared to the word by word translation approach ([3][7]). The RNN Encoder-Decoder is one of the well-known NMT approaches which utilizes the encoding of the entire input message into a meaningful vector and then decode it into a target message. In [6] the RNN Encoder-Decoder was used to translate English sentences into French with good results. Hence, in this work we chose RNN Encoder-Decoder as an approach to translate from human natural language to robot command. A similar approach is used in [1] for Human-machine communication using neural machine translation.

2.3 Existing Methodology

The task to translate human natural language into machine command is highly challenged. While the word by word translation solutions do exist but they are mostly not applicable for natural language translation due to the fact that it does not take into account the context of the full message.

All the currently existing methodologies like Voice Recognition Processor(VRP), Microsoft Speech SDK 5.1, etc rely on word-to-word translation technique which neither consider the context of the full message nor the expandability of the command set.

On the contrary, [1] tells us the new way of human-machine communication by using potential of Natural Language Processing and Machine Learning, by using Neural Machine Translation approach like RNN encoding-decoding. Also, we can modify that system by improving the preprocessing function so that segmentation of the text is done in a more realistic and proper order.

CHAPTER 3
PROJECT STATEMENT

3.1 Problem Statement

To develop a software system for Robotic movement using Natural Language Processing and Machine Learning (use of Neural Machine Translation(NMT)) Technique. It receives natural language command input from human, namely text strings will generate the set of precise trajectory information, namely, the set of command and related parameters for the robot to perform the movement. The proposed software system can be divided into four main parts.

3.2 Proposed Methodology

The goal of this work is to create a software system that receives natural language input movement command from human, namely text strings, and produces the set of precise trajectory information, namely, the set of command and related parameters for the robot to perform the movement.

The proposed software system can be divided into four main parts. First, the Pre-processing function takes in the natural language command in a form of a raw input text from users and produces an output which is a list of command. As stated in the future scope of the base paper, we are going to modify the base paper [1] implementation to incorporate the correct segmentation of sentences without conjunction. Adding on further, we also try to change base paper [1] implementation by changing user input from text to voice. Then, the Command classification function utilizes the RNN Encoder-Decoder to processes the list of command and give out the list of classification command ID as an output. The Parameter classification function then selects the correspond RNN Encoder-Decoder for each command. Lastly, the Post-processing function puts together the complete robot movement commands and parameters based on the RNN encoded information.

Recurrent Neural Network comes into the picture when any model needs context to be able to provide the output based on the input. Sometimes the context is the single most important thing for the model to predict the most appropriate output. RNN remembers everything. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other. Let’s say you have to predict the next word in a given sentence, in that case, the relation among all the previous words helps in predicting the better output. The RNN remembers all these relations while training itself.

Training a RNN is similar to training a traditional Neural Network. We also use the backpropagation algorithm, but with a little twist. Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step but also the previous time steps. For example, in order to calculate the gradient at t=4, we would need to backpropogate 3 steps and sum up the gradients. This is called BPT. An Encoder-Decoder architecture was developed where an input sequence was read in entirety and encoded to a fixed-length internal representation. A decoder network then used this internal representation to output words until the end of sequence token was reached. LSTM networks were used for both the encoder and decoder. The idea is to use one LSTM to read the input sequence, one step at a time, to obtain large fixed-dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector.

At each time step, encoder reads the input at that time step and also add hidden state from the previous step. At the end of the input sequence, it outputs a hidden state that is meant to encode the entire input sequence of both semantic and structural regularities in the input sequence. Here in this type of process, there is no additional feature specific grammars syntax are applied and encoder RNN pretty much encodes the necessary features from the sequence itself. This is quite amazing in comparison with traditional statistical machine translation technique. Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder-decoders and consists of an encoder that encodes a source sentence into a fixed-length vector from which a decoder generates a translation.

The use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder-decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts like a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, a qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

CHAPTER 4

SYSTEM REQUIREMENTS AND SPECIFICATION

4.1 Hardware Requirements

- Computer System with Intel i5 or High Configuration
- Graphics processor (NVIDIA GeForce 940 MX or above)

4.2 Software Requirements

- Python3(3.5.2 or later)
- Tensor Flow (1.4.0)
- Keras
- NLTK

CHAPTER 5
SYSTEM DESIGN

5.1 Overall Architecture Diagram

Fig. 5.1 shows the overall architecture of the proposed system. An architecture diagram is a graphical representation of a set of concepts, that are part of an architecture, including their principles, elements and components. Software architecture involves the high level structure of software system abstraction, by using decomposition and composition, with architectural style and quality attributes. A software architecture design must conform to the major functionality and performance requirements of the system, as well as satisfy the non-functional requirements such as reliability, scalability, portability, and availability. A software architecture must describe its group of components, their connections, interactions among them and deployment configuration of all components.

The user interacts with the web interface on the monitor, and gives Natural Language Text Input, which is then processed by the business logic consisting of dictionaries, Machine Learning Model, main.py (module for that consists of all the three parts preprocessor, command/parameter classification and post processor). The database or corpus is used to form the model. And Finally, system outputs the robotic language commands to the user. In architecture (the discipline where we design buildings, not software) diagrams are regularly used as communications tools, but they can also be used as a way of testing different concepts quickly.

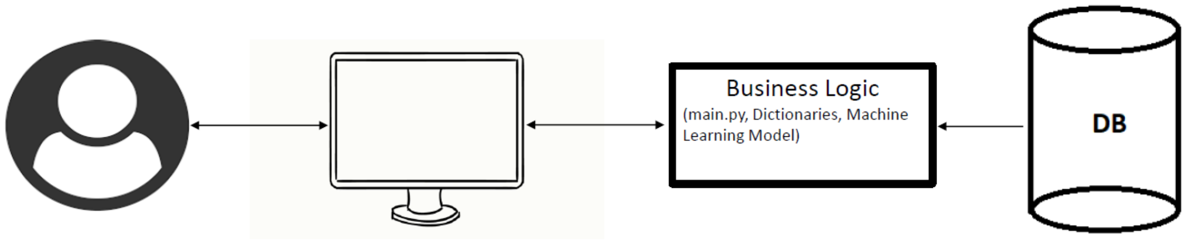


FIGURE 5.1. Overall Architecture Diagram

5.2 Use Case Diagram

In Fig. 5.2, User is a Primary actor, as it initiates the use of the system. User commands the Robot, which then reacts to the command, and in this whole process, user gives natural language text input, and then the software system converts natural language to robot language by pre-processing, command/parameter classification, and post-processing. Here, Pre-processing includes input segmentation, and may result in discarding of input. Hence, the former is a include relationship, while the latter one is an extended relationship with respect to "Natural to Robot Language conversion" use case. And finally after Natural to Robot language conversion, system outputs the commands in robotic language in order. This whole is required for our base use case "commands" to get executed.

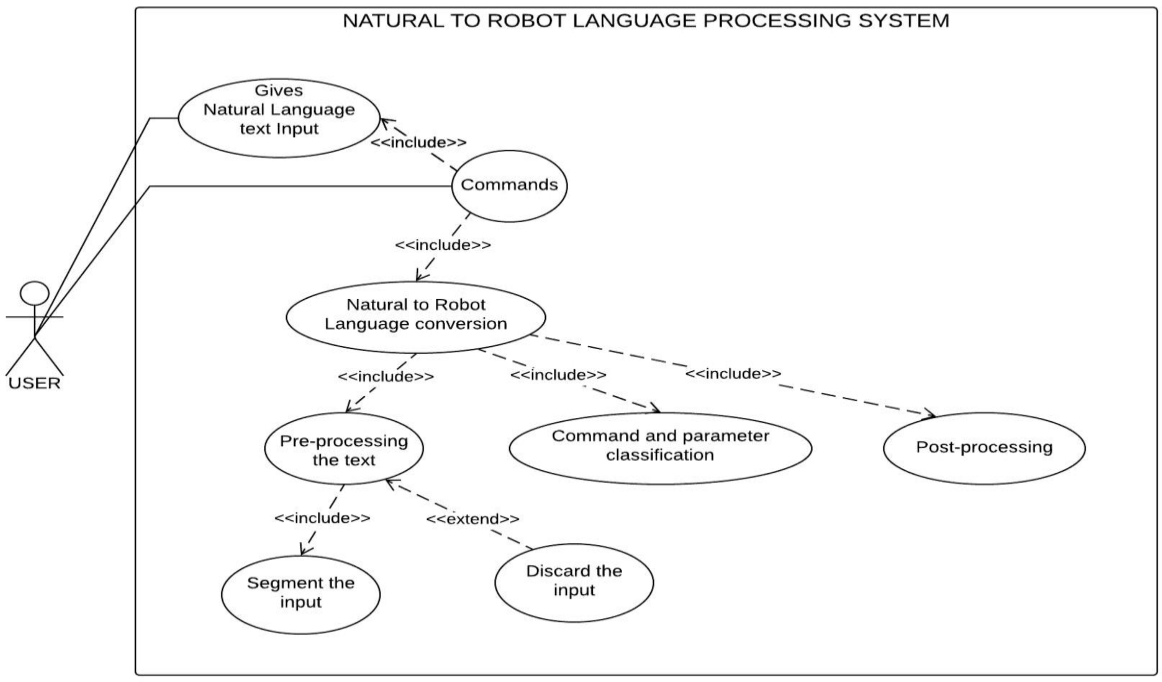


FIGURE 5.2. Use Case Diagram

5.3 Sequence Diagram

Fig. 5.3 shows sequence diagram. A sequence diagram, in the context of UML, represents object collaboration and is used to define event sequences between objects for a certain outcome. A sequence diagram is an essential component used in processes related to analysis, design and documentation. A sequence diagram is also known as a timing diagram, event diagram and event scenario. Object interactions usually begin at the top of a diagram and end at the bottom. In a sequence diagram, object interaction occurs through messages on the vertical and horizontal dimensions and are designated by horizontal arrows and message names. The initial sequence diagram message begins at the top and is located on the diagram's left side. Subsequent messages are added just below previous messages.

Sequence diagram messages may be subdivided by type, based on functionality. A lifeline, which indicates a role, is represented by a named rectangular box with a dashed line descending from the center of the diagram's bottom edge. Lifeline boxes represent participating sequence object instances. Blank instance names represent anonymous instances. User is the main object followed by the Interface, pre-processor, command/parameter classifier, and post-processor. There is basic sequence of activities required to be followed for successful use of this system.

User gives Natural Language text Input for robotic movement, from interface, it’s sent to pre-processor. Pre-processor segments the input. The segmented input list is then sent to Command/Parameter classifier, which encodes it into encoded command and parameter vectors, which are then sent to post-processor, which combinely decodes them to get robot language command vector, and then this vector is sent to interface from post-processor, which then displays it to the user.

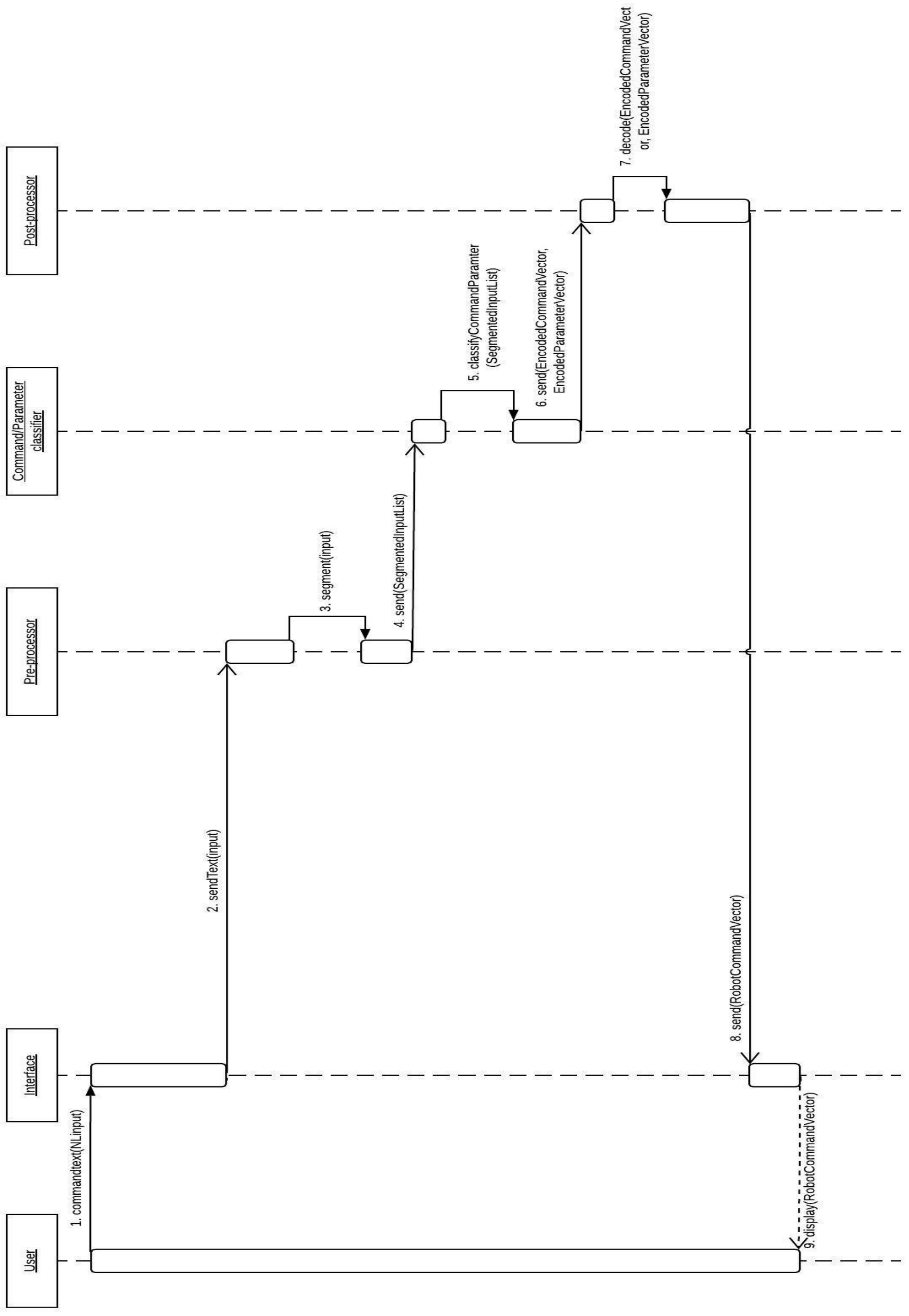

User
Interface
Pre-processor
Command/Parameter classifier
Post-processor
1. commandtext(NLinput)
2. sendText(input)
3. segment(input)
4. send(SegmentedInputList)
5. classifyCommandParamter (SegmentedInputList)
6. send(EncodedCommandVector, EncodedParameterVector)
7. decode(EncodedCommandVector, EncodedParameterVector)
8. send(RobotCommandVector)
9. display(RobotCommandVector)

FIGURE 5.3. Sequence Diagram

5.4 Activity Diagram

Fig.5.4 shows the activity diagram. Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc. The basic purposes of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single. The purpose of an activity diagram can be described as − draw the activity flow of a system, describe the sequence from one activity to another, and describe the parallel, branched and concurrent flow of the system.

The activity diagram is basically divided into 2 partitions, each corresponding to User, and the Software respectively. It starts from user giving natural language text input, followed by software preprocessing the input, and as an output of preprocessing, Segmented Input List object is passed to command/parameter classification, which then sends Encoded Command Vector and Encoded Parameter Vector to Post-processing, which finally, gives Robot Command Vector in robotic language as an output object to the Interface, which then displays it, and then finally, our system goes into the final state. Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types: ellipses represent actions, diamonds represent decisions, bars represent the start (split) or end (join) of concurrent activities, a black circle represents the start (initial node) of the workflow, an encircled black circle represents the end (final node).

Arrows run from the start towards the end and represent the order in which activities happen. Activity diagrams can be regarded as a form of a structured flowchart combined with a traditional data flow diagram. Typical flowchart techniques lack constructs for expressing concurrency.

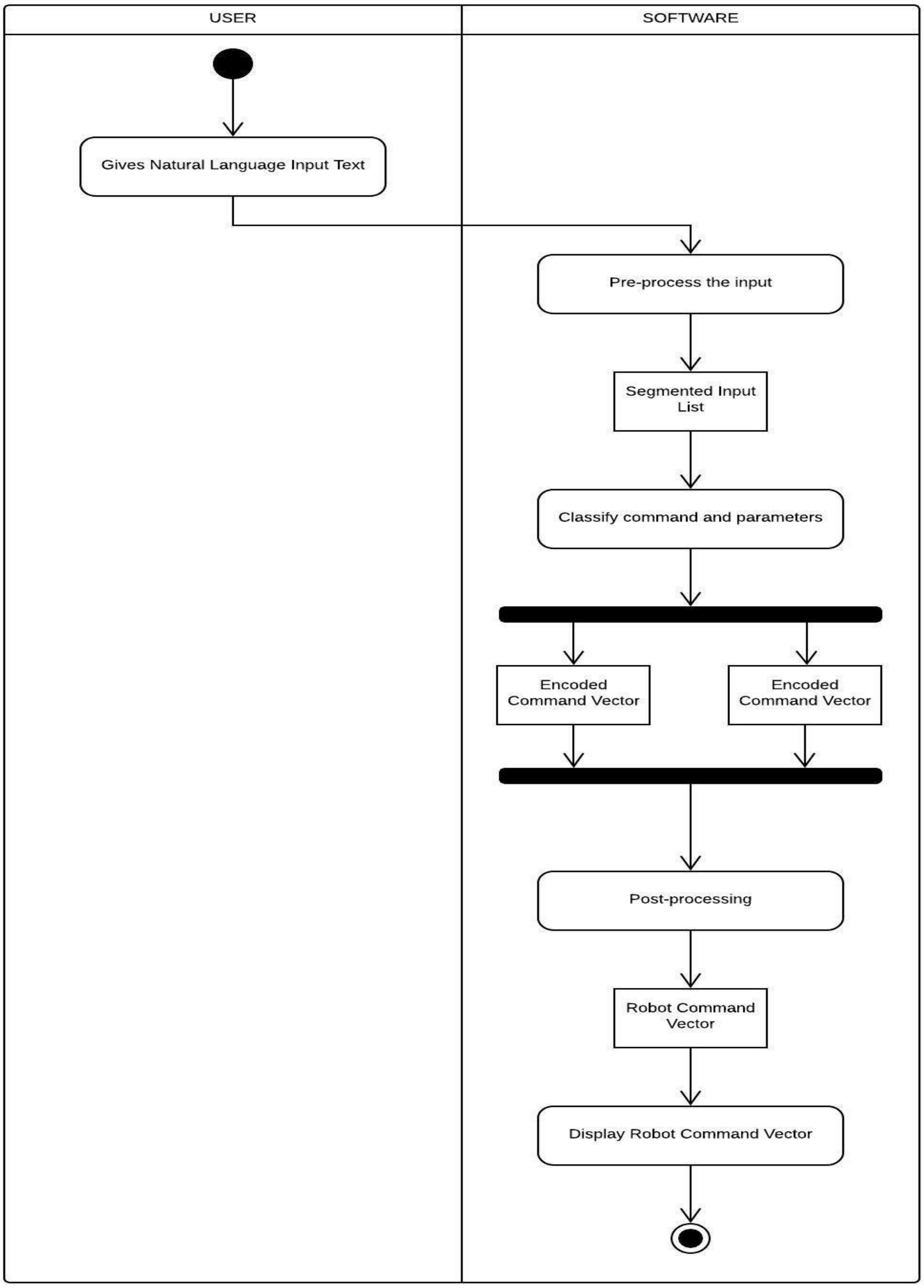


FIGURE 5.4. Activity Diagram

5.5 Class Diagram

In Fig. 5.5, Our system has the User_command class which has NLInput as attribute, and commandRobot() as operation, and it commands via SOFTWARE_SYSTEM, which has getCommand() as operation. User_Command commands SOFTWARE_SYSTEM. One system can have one or many user commands.

SOFTWARE_SYSTEM has Input_To_Command as a part, which cannot exist without the SOFTWARE_SYSTEM, hence the composition relationship. Input_To_Command has convertInputToCommand() as an operation, and it inherits from Preprocessor, Command_Parameter_Classifier, and PostProcessor classes. There is one-to-one relationship between SOFTWARE_SYSTEM and Input_To_Command.

SOFTWARE_SYSTEM also has Command as a part. Command has name, class and parameter as attributes. One system can have one or many commands. Also, One Command_Parameter_Classifier uses one or more commands.

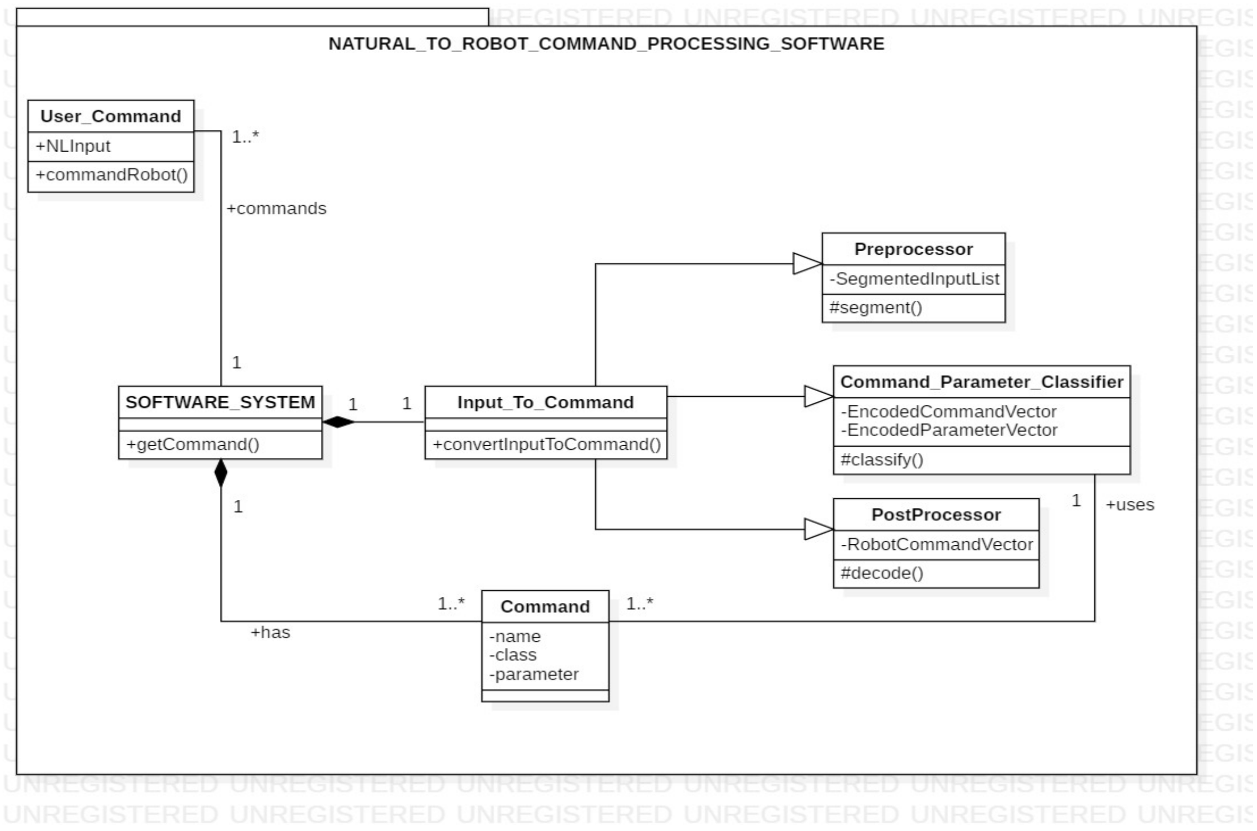


FIGURE 5.5. Class Diagram

5.6 Deployment Diagram

Fig. 5.6 shows deployment diagram. Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships. The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related. Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware. UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components. Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers. The purpose of deployment diagrams can be described as − visualize the hardware topology of a system, describe the hardware components used to deploy software components, and describe the runtime processing nodes.

Here, there are 2 basic modules. Web Page and NLP System are 2 important deployment modules. Web Page is basically an important module from where user gives input to whole system. The whole NLP system depends on and consists of Pre-processor, Command/Parameter classifier, Post-processor. Finally, Post-processor gives the Robotic command which is displayed by the web page to the user.

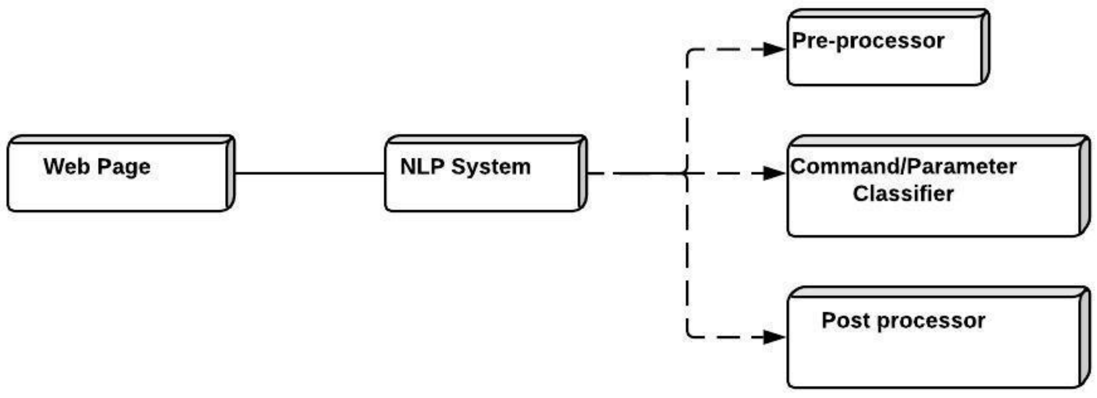


FIGURE 5.6. Deployment Diagram

5.7 Component Diagram

Fig. 5.7 shows the component diagram. Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node. Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems. Component diagram is a special kind of diagram in UM. Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc. Component diagrams can also be described as a static implementation view of a system. A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole. The purpose of the component diagram can be summarized as − visualize the components of a system, construct executables by using forward and reverse engineering, and describe the organization and relationships of the components.

Component diagram includes physical aspects of the system, which are Web Page, NLP System. NLP system depends on, and consists of Pre-processor, Command/Parameter classifier, and Post-processor. Web page provides the service of giving natural language input, and NLP System gives the robotic command output.

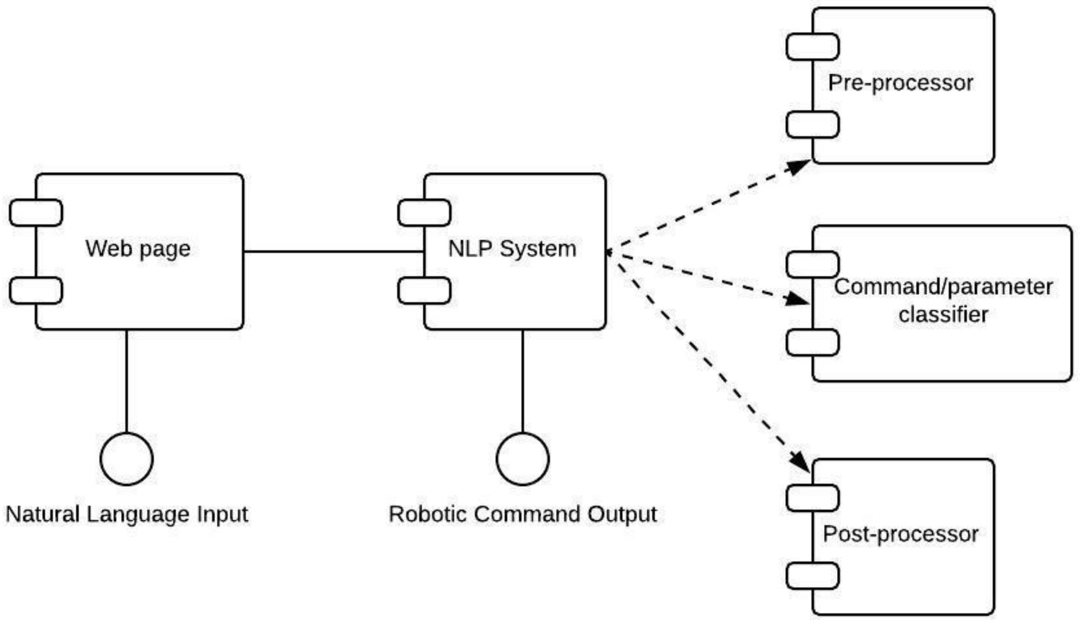


FIGURE 5.7. Component Diagram

5.8 Collaboration Diagram

Fig. 5.8 shows the collaboration diagram of the proposed system. Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received between the states/activities of a system. Structural organization consists of objects and links.

The use or purpose of collaboration diagram is that similar to sequence diagram. Although, the specific purpose of this UML diagram is to visualize how the objects are organized and how they are interacted.

User gives Natural Language text Input for robotic movement, from interface, it's sent to pre-processor. Pre-processor segments the input. The segmented input list is then sent to Command/Parameter classifier, which encodes it into encoded command and parameter vectors, which are then sent to post-processor, which combinely decodes them to get robot language command vector, and then this vector is sent to interface from post-processor, which then displays it to the user.

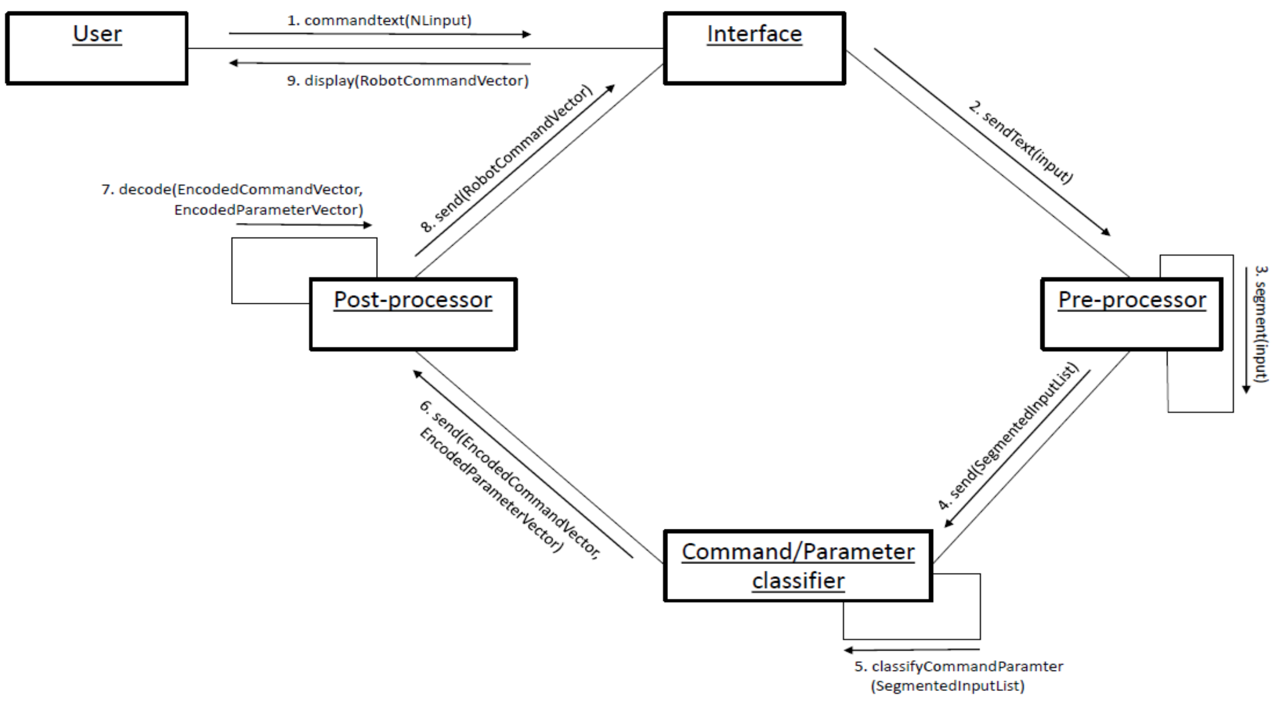

User
1. commandtext(NLinput)
9. display(RobotCommandVector)
Interface
2. sendText(input)
8. send(RobotCommandVector)
7. decode(EncodedCommandVector, EncodedParameterVector)
Post-processor
Pre-processor
3. segment(input)
6. send(EncodedCommandVector, EncodedParameterVector)
4. send(SegmentedInputList)
Command/Parameter classifier
5. classifyCommandParamter (SegmentedInputList)

FIGURE 5.9. Collaboration Diagram

5.9 Data Flow Diagram (Level - 0)

In Fig. 5.9, User and Interface are external entities. User gives Natural Language Text Input to the process “Natural to Robot Command Processing”, which then processes it, and returns “Robot Command Vector” to the interface.

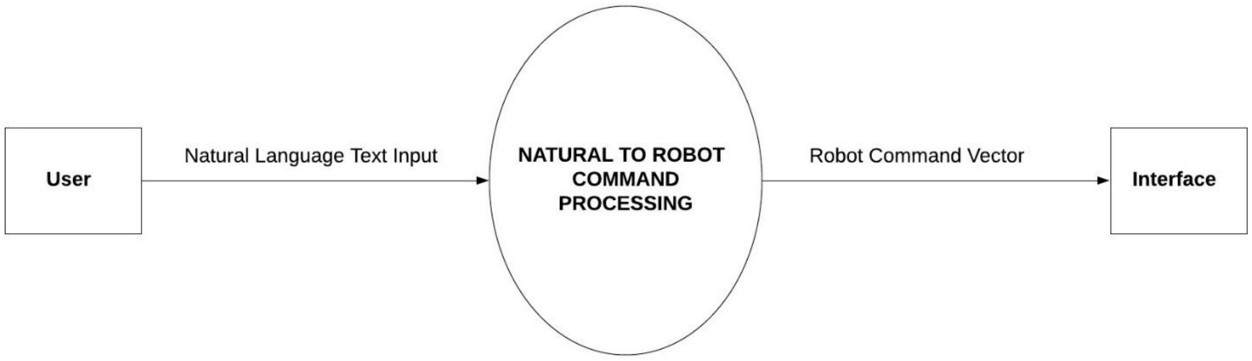

User
Natural Language Text Input
NATURAL TO ROBOT COMMAND PROCESSING
Robot Command Vector
Interface

FIGURE 5.9. Data Flow Diagram (Level – 0)

5.10 Data Flow Diagram (Level – 1)

In Fig. 5.10, User and Interface are external entities. User gives Natural Language text Input, which is pre-processed(1.0), to give the Segmented Input List, which is then sent to process of command/parameter classification(2.0) to get converted to Encoded command and parameter vector, which is post-processed(3.0) to Robot Command Vector in robotic language, which is finally given to the Interface to display.

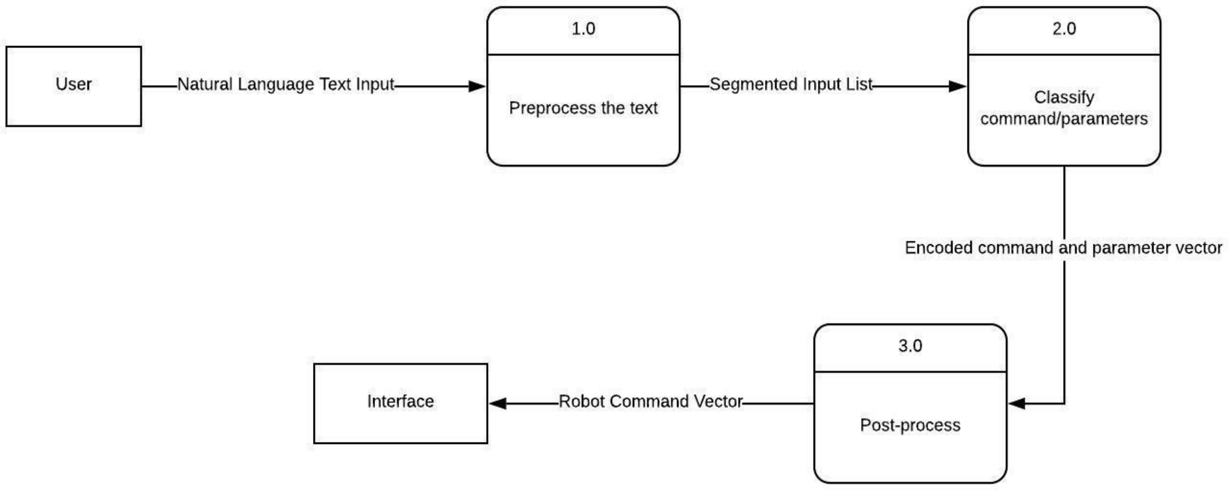


FIGURE 5.10. Data Flow Diagram (Level-1)

CHAPTER 6
METHODOLOGY USED FOR IMPLEMENTATION

6.1 Work Flow of The Proposed System

Fig. 6.1 shows the workflow of the proposed system. The designed system is an advancement of the currently available software systems for Human Robot Interaction. As discussed earlier, the currently available Human Robot Interaction systems like Voice Recognition Processor(VRP), Microsoft SDK, etc work such that robot commands for movement are given in precise patterns and rules for detecting their actions for movement and thereby completion of tasks. The designed system uses NMT or Neural Machine Translation technique using RNN Encoding-Decoding that combines both Natural Language Processing and Machine Learning, thus making the task of Human Robot Interaction effective, so that we don't need a particular robot language.

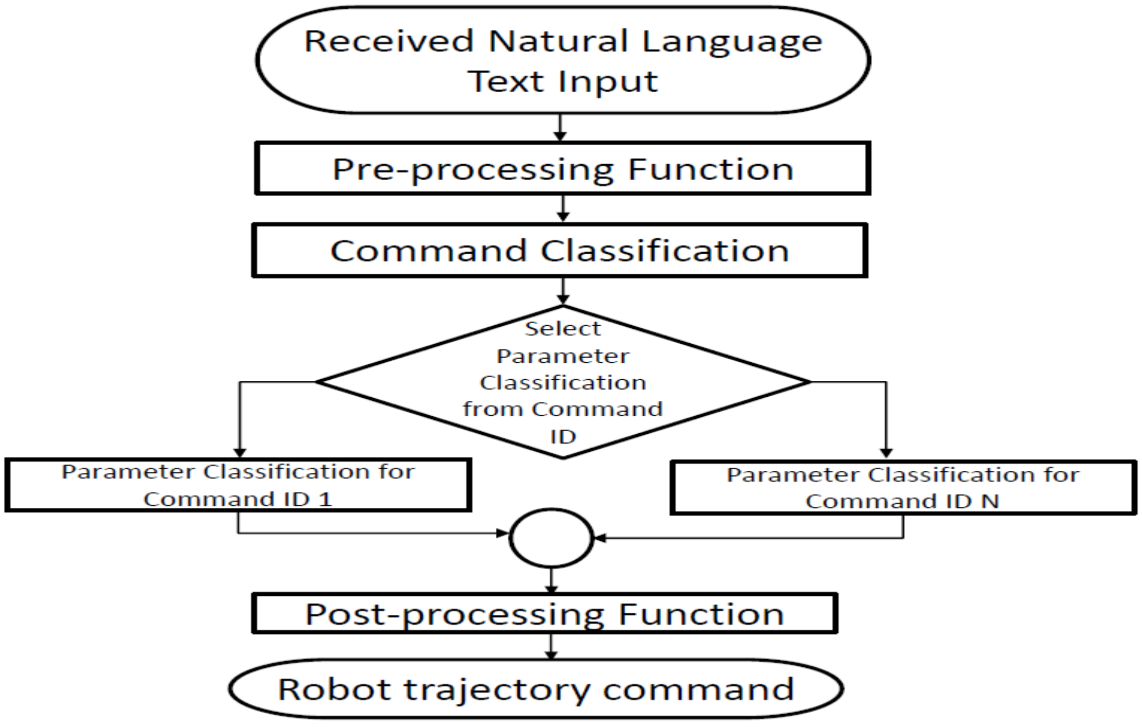

Received Natural Language Text Input
Pre-processing Function
Command Classification
Select Parameter Classification from Command ID
Parameter Classification for Command ID 1
Parameter Classification for Command ID N
Post-processing Function
Robot trajectory command

FIGURE 6.1. Work Flow of The Proposed System

What the system does is, firstly, it preprocesses the sentences considering conjunctions, and then sends all the preprocessed list of necessary words excluding conjunctions to Command/Parameter classifier, which first classifies the commands, and then its parameters and arguments, and forms an encoded command vector, which is then decoded back into the robotic language by the post-processor. If we type a sentence that doesn’t intend to command, then it is classified as nothing, and nothing is displayed on screen. In other cases, based on conjunctions used, commands along with their arguments are displayed on screen. The main purpose of the system is to enable user to not to remember precise patterns and rules while interacting with the robot, thus making it more human-like, and thus increasing the effectiveness, robustness, and efficiency of the human-robot communication.

6.2 Flowchart of The Proposed System

Fig. 6.2 shows the flowchart of the proposed system. Flowcharts are graphical representation of steps. It was originated from computer science as a tool for representing algorithms and programming logic, but had extended to use in all other kinds of processes. Nowadays, flowcharts play an extremely important role in displaying information and assisting reasoning. They help us visualize complex processes, or make explicit the structure of problems and tasks. A flowchart can also be used to define a process or project to be implemented. Flowcharts are used in designing and documenting simple processes or programs. Like other types of diagrams, they help visualize what is going on and thereby help understand a process, and perhaps also find less-obvious features within the process, like flaws and bottlenecks. A flowchart is described as "cross-functional" when the chart is divided into different vertical or horizontal parts, to describe the control of different organizational units. A symbol appearing in a particular part is within the control of that organizational unit. A cross-functional flowchart allows the author to correctly locate the responsibility for performing an action or making a decision, and to show the responsibility of each organizational unit for different parts of a single process. Flowcharts depict certain aspects of processes and are usually complemented by other types of diagram. For instance, Kaoru Ishikawa, defined the flowchart as one of the seven basic tools of quality control, next to the histogram, Pareto chart, check sheet, control chart, cause-and-effect diagram, and the scatter diagram.

We start by user entering the natural language text input on the web page, which is then preprocessed to form the segmented input list. This segmented input list is then given as an input to command/parameter classifier which then performs the classification and outputs the encoded command vector and encoded parameter vector. These 2 encoded vectors are then given as an input to post-processor which then decodes them to form the robot command vector, which is then displayed on the screen.

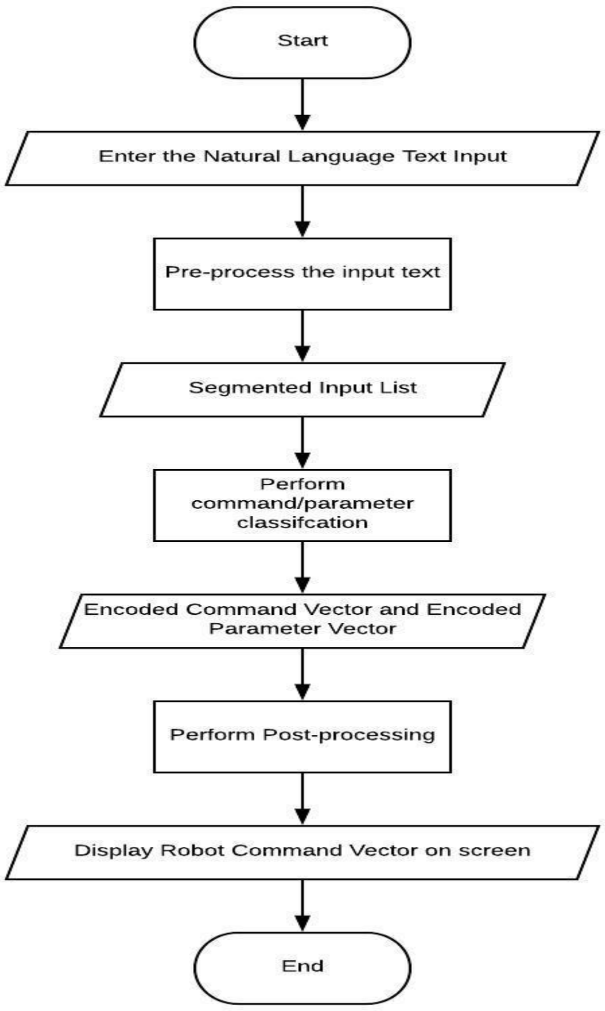

Start
Enter the Natural Language Text Input
Pre-process the input text
Segmented Input List
Perform command/parameter classifcation
Encoded Command Vector and Encoded Parameter Vector
Perform Post-processing
Display Robot Command Vector on screen
End

FIGURE 6.2. Flowchart of The Proposed System

6.3 Related Terms

Before learning about the algorithms applied for implementation, we need to understand about a few terms related to the proposed system. They are-

1. Recurrent Neural Networks
2. Encoder- Decoder Long Short-Term Memory Networks

1. Recurrent Neural Networks(RNN)- Recurrent Neural Network(RNN) are a type of Neural Network where the output from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence. See Fig. 6.3.

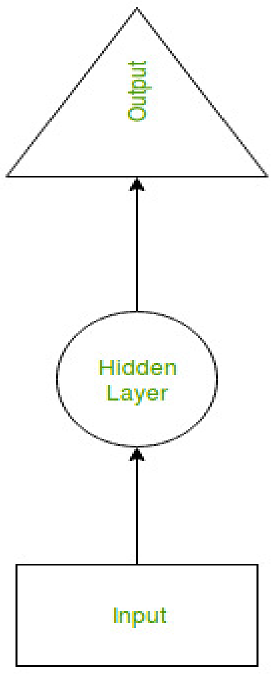


FIGURE 6.3. Basic RNN

RNN have a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

Now, we can do training through RNN by the following steps-

1. A single time step of the input is provided to the network.
2. Then calculate its current state using set of current input and the previous state.
3. The current state becomes previous state for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e. the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

Example of a Recurrent Neural Network- Suppose there is a deeper network with one input layer, three hidden layers and one output layer. Then like other neural networks, each hidden layer will have its own set of weights and biases, let’s say, for hidden layer 1 the weights and biases are (w1, b1), (w2, b2) for second hidden layer and (w3, b3) for third hidden layer. This means that each of these layers are independent of each other, i.e. they do not memorize the previous outputs. See Fig. 6.4.

Now the RNN will do the following:

a. RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous outputs by giving each output as input to the next hidden layer.

b. Hence these three layers can be joined together such that the weights and bias of all the hidden layers is the same, into a single recurrent layer.

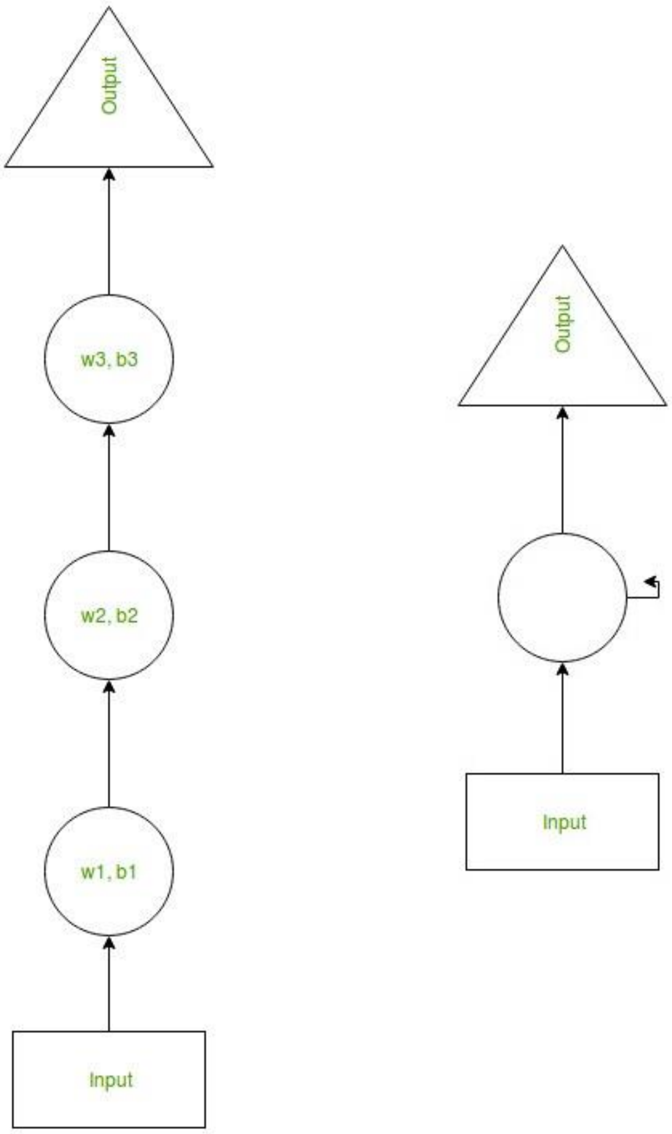


FIGURE 6.4. RNN Example

Advantages of RNN are-

1. An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. It is called LSTM.

2. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages of RNN are-

1. Gradient vanishing and exploding problems.

2. Training an RNN is a very difficult task.

3. It cannot process very long sequences if using tanh or relu as an activation function.

2. Encoder-Decoder Long Short-Term Memory Networks- Sequence prediction often involves forecasting the next value in a real valued sequence or outputting a class label for an input sequence. This is often framed as a sequence of one input time step to one output time step (e.g. one-to-one) or multiple input time steps to one output time step (many-to-one) type sequence prediction problem. There is a more challenging type of sequence prediction problem that takes a sequence as input and requires a sequence prediction as output. These are called sequence-to-sequence prediction problems, or seq2seq for short. One modeling concern that makes these problems challenging is that the length of the input and output sequences may vary. Given that there are multiple input time steps and multiple output time steps, this form of problem is referred to as many-to-many type sequence prediction problem.

One approach to seq2seq prediction problems that has proven very effective is called the Encoder-Decoder LSTM. This architecture is comprised of two models: one for reading the input sequence and encoding it into a fixed-length vector, and a second for decoding the fixed-length vector and outputting the predicted sequence. The use of the models in concert gives the architecture its name of Encoder-Decoder LSTM designed specifically for seq2seq problems. The Encoder-Decoder LSTM was developed for natural language processing problems where it demonstrated state-of-the-art performance, specifically in the area of text translation called statistical machine translation. The innovation of this architecture is the use of a fixed-sized internal representation in the heart of the model that input sequences are read to and output sequences are read from. For this reason, the method may be referred to as sequence embedding. In one of the first applications of the architecture to English-to-French translation, the internal representation of the encoded English phrases was visualized. The plots revealed a qualitatively meaningful learned structure of the phrases harnessed for the translation task.

On the task of translation, the model was found to be more effective when the input sequence was reversed. Further, the model was shown to be effective even on very long input sequences. This approach has also been used with image inputs where a Convolutional Neural Network is used as a feature extractor on input images, which is then read by a decoder LSTM. See Fig. 6.5.

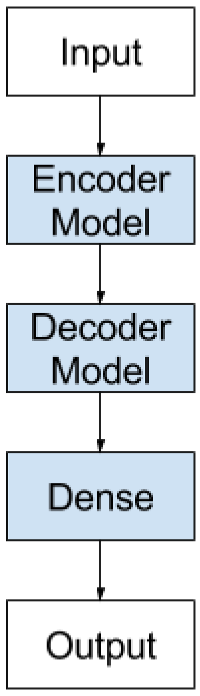


FIGURE 6.5. Encoder-Decoder LSTM Network

3. Categorical Cross-Entropy- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

Cross-entropy and log loss are slightly different depending on context, but in machine learning when calculating error rates between 0 and 1 they resolve to the same thing.

4. Accuracy- Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where,

TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

5. Categorical accuracy- It calculates the mean accuracy rate across all predictions for multiclass classification problems. For binary classification, the code for accuracy metric is:

```
K.mean(K.equal(y_true, K.round(y_pred)))
```

which suggests that 0.5 is the threshold to distinguish between classes. y_true should of course be 1-hots in this case. It's a bit different for categorical classification:

```
K.mean(K.equal(K.argmax(y_true, axis=-1), K.argmax(y_pred, axis=-1)))
```

which means "how often predictions have maximum in the same spot as true values".

6.4 Algorithms

We can understand the algorithm of the proposed system by dividing it into two parts-

1. For training and testing the model
2. For input from the user

1. For training and testing the model- We have considered these motion commands for our system - [NULL, FORWARD, BACKWARD, LEFT, RIGHT, STOP]. We created a labelled dataset or corpus consisting of the different sentences having this keywords withiut conjunctions. For the purpose of training the model and testing the model, we apply the following algorithm-

1. Import required libraries and functions.
2. Read the csv file of the dataset into a dataframe say "abc". Now, thus abc has 2 fields- sentence, and command.
3. Separate out abc.sentence into a new variable, say "texts".
4. Store "labels" by converting abc.command into categorical form. Now, this will result into a boolean matrix with each column corresponding to the abc.command, and each row representing the sentences, and it has 0s at the positions if the corresponding sentence doesn't have the corresponding keyword, and 1 otherwise.

5. Tokenize the texts, and create a dictionary such that it has all the unique words in descending order of frequencies with their respective indexes, and dump this dictionary of word-token pairs to a JSON file "dictionarylstm.json".

6. Also, convert texts into sequences of token indexes and pad them and store them in say "data".

7. Now, first store data into test_x(input) and labels into test_y(class in categorical form), and split this data by using train_test_split into X_train, X_test and Y_train and Y_test.

8. Now, form a classification model by applying RNN encoding-decoding using LSTM on X_train and Y_train, and evaluate it for loss and accuracy by testing the model on X_test and Y_test.

9. Then, write the model into a JSON file, say "classificationlstm.json", and save its weights in a HDF file, say "classificationlstm.hdf5".

<u>2. For input from the user-</u> For input from the user, we follow the following algorithm to obtain the output-

1. Take natural language text input from the user into a variable, say, "message".

2. Load the model created from JSON file "classificationlstm.json", and the weights of this model from HDF file "classificationlstm.hdf5".

3. Read all the considered conjunctions from "conjunctions.txt" file into a variable, say "conjunctions".

4. Preprocessing step

Segment the input message on the basis of conjunctions, and obtain the preprocessed input list and store it back into the variable "message".

5. Now, read "dictionarylstm.json" into a variable, say "dictionary_file", and from this, read word_index, i.e., word-token pairs.

6. Command/Parameter classification step

Apply model on message to predict the output command. And also, check for the parameter and value in message, e.g. 3 centimeters as value-3, parameter-cm.

7. Post processing step-

Then, for each segment, add to output a string with the syntax

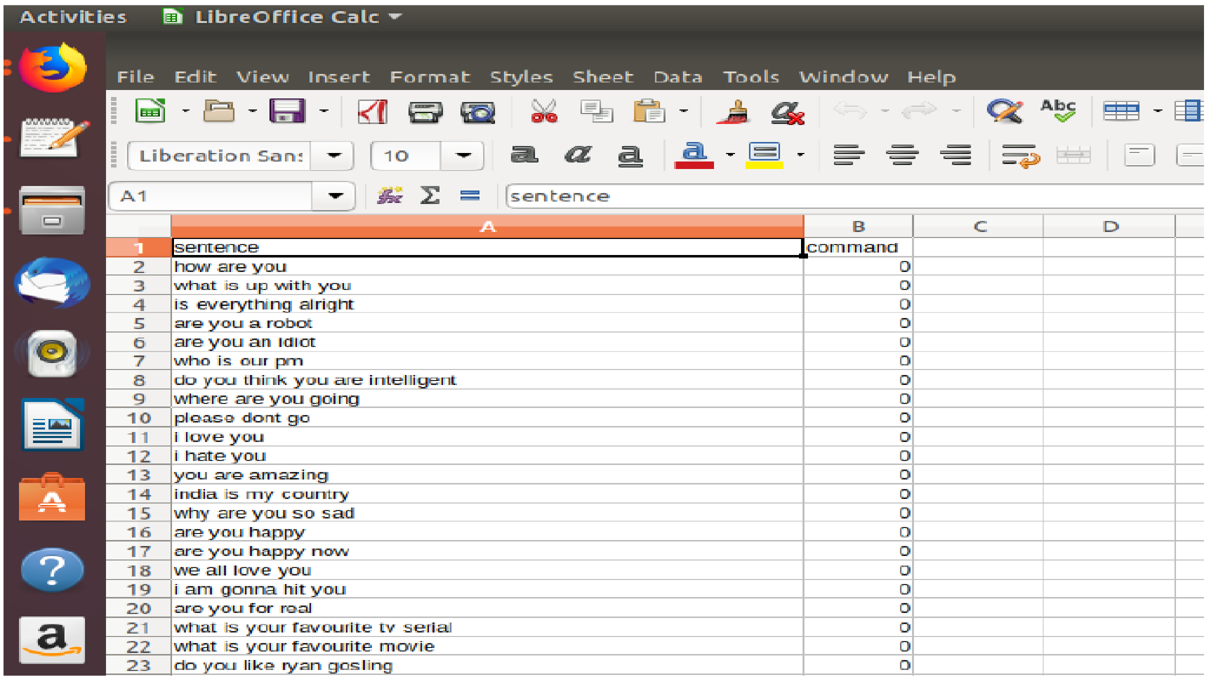
"<command>, ['<value>'], <parameter>"

and nothing for segment classified as NULL.

Eg.- "forward,['10'] , cm"

6.5 Data Analysis

As shown in Fig. 6.6, we began to design the proposed system by first designing the dataset or corpus with 660 entries, having fields as sentence and command. Sentence field consisted of sentences without conjunctions, or simply saying, simple sentences, which contained one of the considered motion commands- [Null, Forward, Backward, Left, Right, Stop] in them. Command field consisted of command class of the corresponding sentence like 0 for Null, 1 for Forward, 2 for Backward, 3 for Left, 4 for Right and 5 for Stop. This dataset or corpus is used to train the machine learning model. Null refers to those sentences which actually do not intend to command the robot for motion. For eg.- "How are you?" gets classified as Null as it does not intend to command the robot for motion. Based on training on this dataset, files dictionarylstm.json, classificationlstm.json and classificationlstm.hdf5 are generated. And also, for user input, we use conjunction.txt to get conjunctions for preprocessing the input.

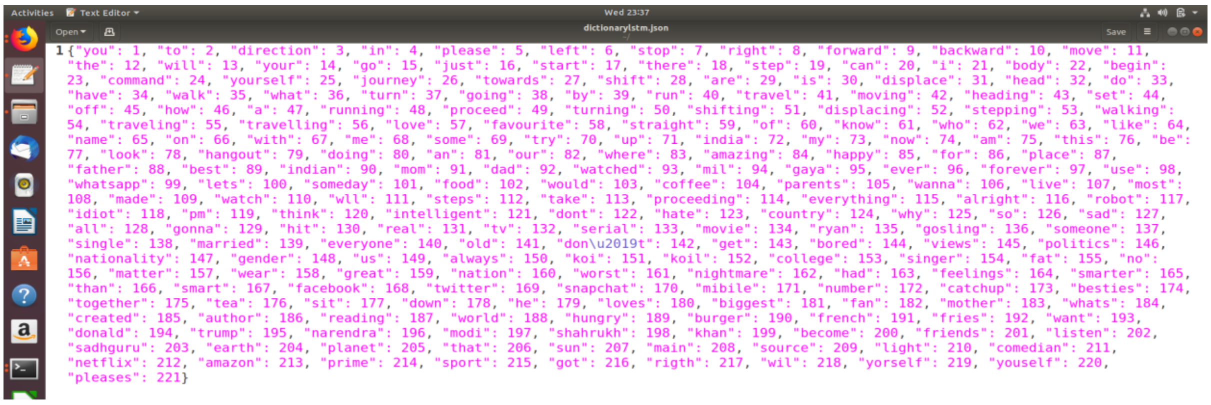


sentence

	A	B	C	D
1	sentence	command		
2	how are you	0		
3	what is up with you	0		
4	is everything alright	0		
5	are you a robot	0		
6	are you an idiot	0		
7	who is our pm	0		
8	do you think you are intelligent	0		
9	where are you going	0		
10	please dont go	0		
11	i love you	0		
12	i hate you	0		
13	you are amazing	0		
14	india is my country	0		
15	why are you so sad	0		
16	are you happy	0		
17	are you happy now	0		
18	we all love you	0		
19	i am gonna hit you	0		
20	are you for real	0		
21	what is your favourite tv serial	0		
22	what is your favourite movie	0		
23	do you like ryan gosling	0		

FIGURE 6.6. Dataset

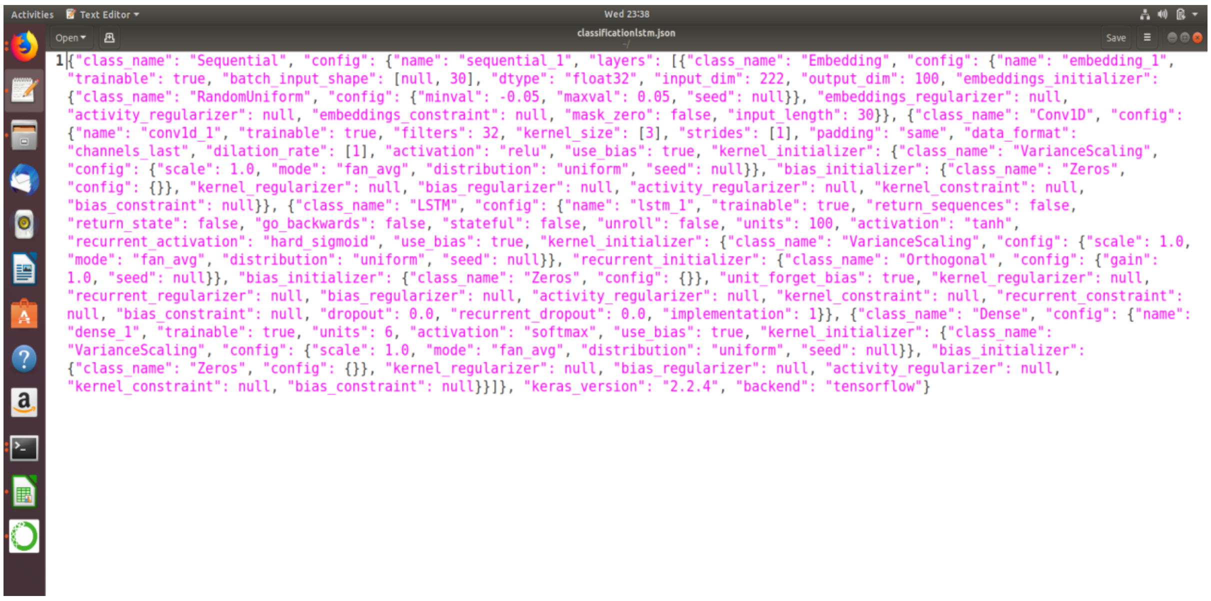
<u>1. dictionarylstm.json-</u> This file is generated by trainlstm.py, and consists of word-token pairs. It has all the unique words in descending order of frequencies with their respective indexes. See Fig. 6.7.



```
{"you": 1, "to": 2, "direction": 3, "in": 4, "please": 5, "left": 6, "stop": 7, "right": 8, "forward": 9, "backward": 10, "move": 11,
"the": 12, "will": 13, "your": 14, "go": 15, "just": 16, "start": 17, "there": 18, "step": 19, "can": 20, "i": 21, "body": 22, "begin":
23, "command": 24, "yourself": 25, "journey": 26, "towards": 27, "shift": 28, "are": 29, "is": 30, "displace": 31, "head": 32, "do": 33,
"have": 34, "walk": 35, "what": 36, "turn": 37, "going": 38, "by": 39, "run": 40, "travel": 41, "moving": 42, "heading": 43, "set": 44,
"off": 45, "how": 46, "a": 47, "running": 48, "proceed": 49, "turning": 50, "shifting": 51, "displacing": 52, "stepping": 53, "walking":
54, "traveling": 55, "travelling": 56, "love": 57, "favourite": 58, "straight": 59, "of": 60, "know": 61, "who": 62, "we": 63, "like": 64,
"name": 65, "on": 66, "with": 67, "me": 68, "some": 69, "try": 70, "up": 71, "india": 72, "my": 73, "now": 74, "am": 75, "this": 76, "be":
77, "look": 78, "hangout": 79, "doing": 80, "an": 81, "our": 82, "where": 83, "amazing": 84, "happy": 85, "for": 86, "place": 87,
"father": 88, "best": 89, "indian": 90, "mom": 91, "dad": 92, "watched": 93, "mil": 94, "gaya": 95, "ever": 96, "forever": 97, "use": 98,
"whatsapp": 99, "lets": 100, "someday": 101, "food": 102, "would": 103, "coffee": 104, "parents": 105, "wanna": 106, "live": 107, "most":
108, "made": 109, "watch": 110, "wll": 111, "steps": 112, "take": 113, "proceeding": 114, "everything": 115, "alright": 116, "robot": 117,
"idiot": 118, "pm": 119, "think": 120, "intelligent": 121, "dont": 122, "hate": 123, "country": 124, "why": 125, "so": 126, "sad": 127,
"all": 128, "gonna": 129, "hit": 130, "real": 131, "tv": 132, "serial": 133, "movie": 134, "ryan": 135, "gosling": 136, "someone": 137,
"single": 138, "married": 139, "everyone": 140, "old": 141, "don\u2019t": 142, "get": 143, "bored": 144, "views": 145, "politics": 146,
"nationality": 147, "gender": 148, "us": 149, "always": 150, "koi": 151, "koil": 152, "college": 153, "singer": 154, "fat": 155, "no":
156, "matter": 157, "wear": 158, "great": 159, "nation": 160, "worst": 161, "nightmare": 162, "had": 163, "feelings": 164, "smarter": 165,
"than": 166, "smart": 167, "facebook": 168, "twitter": 169, "snapchat": 170, "mibile": 171, "number": 172, "catchup": 173, "besties": 174,
"together": 175, "tea": 176, "sit": 177, "down": 178, "he": 179, "loves": 180, "biggest": 181, "fan": 182, "mother": 183, "whats": 184,
"created": 185, "author": 186, "reading": 187, "world": 188, "hungry": 189, "burger": 190, "french": 191, "fries": 192, "want": 193,
"donald": 194, "trump": 195, "narendra": 196, "modi": 197, "shahrukh": 198, "khan": 199, "become": 200, "friends": 201, "listen": 202,
"sadhguru": 203, "earth": 204, "planet": 205, "that": 206, "sun": 207, "main": 208, "source": 209, "light": 210, "comedian": 211,
"netflix": 212, "amazon": 213, "prime": 214, "sport": 215, "got": 216, "rigth": 217, "wil": 218, "yorself": 219, "youself": 220,
"pleases": 221}
```

FIGURE 6.7. Dictionarylstm.json

<u>2. classificationlstm.json-</u> This file is also generated by trainlstm.py, and consists of various details about the machine learning model formed by using encoder-decoder LSTM network, which is trained on the dataset or corpus. See Fig. 6.8.

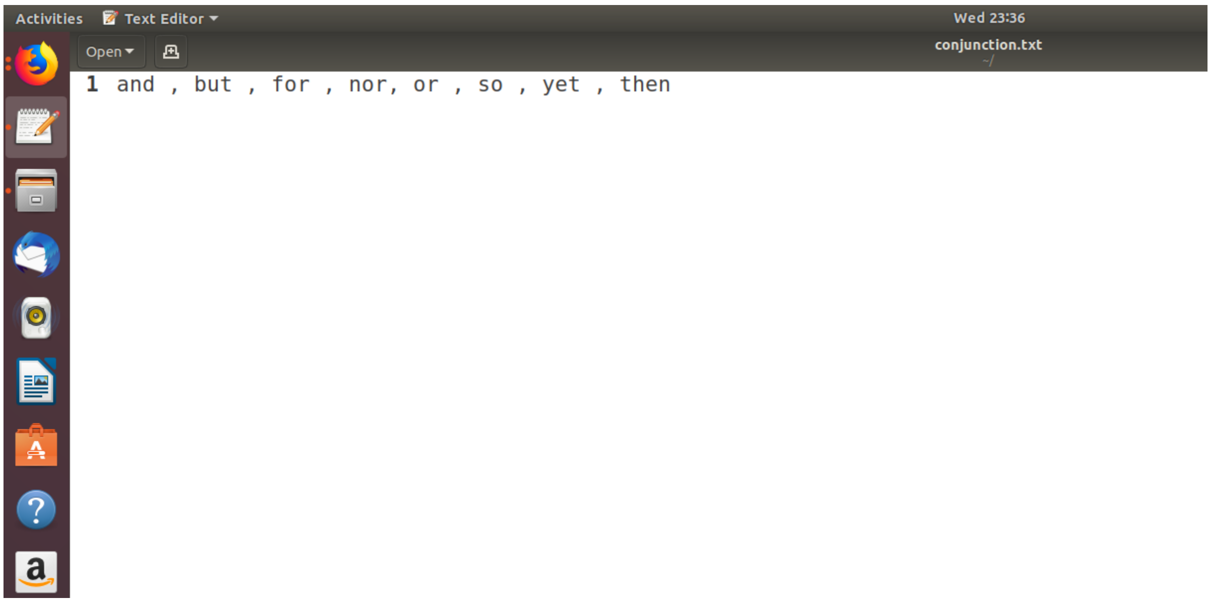


```
{"class_name": "Sequential", "config": {"name": "sequential_1", "layers": [{"class_name": "Embedding", "config": {"name": "embedding_1",
"trainable": true, "batch_input_shape": [null, 30], "dtype": "float32", "input_dim": 222, "output_dim": 100, "embeddings_initializer":
{"class_name": "RandomUniform", "config": {"minval": -0.05, "maxval": 0.05, "seed": null}}, "embeddings_regularizer": null,
"activity_regularizer": null, "embeddings_constraint": null, "mask_zero": false, "input_length": 30}}, {"class_name": "Conv1D", "config":
{"name": "conv1d_1", "trainable": true, "filters": 32, "kernel_size": [3], "strides": [1], "padding": "same", "data_format":
"channels_last", "dilation_rate": [1], "activation": "relu", "use_bias": true, "kernel_initializer": {"class_name": "VarianceScaling",
"config": {"scale": 1.0, "mode": "fan_avg", "distribution": "uniform", "seed": null}}, "bias_initializer": {"class_name": "Zeros",
"config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null,
"bias_constraint": null}}, {"class_name": "LSTM", "config": {"name": "lstm_1", "trainable": true, "return_sequences": false,
"return_state": false, "go_backwards": false, "stateful": false, "unroll": false, "units": 100, "activation": "tanh",
"recurrent_activation": "hard_sigmoid", "use_bias": true, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"scale": 1.0,
"mode": "fan_avg", "distribution": "uniform", "seed": null}}, "recurrent_initializer": {"class_name": "Orthogonal", "config": {"gain":
1.0, "seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "unit_forget_bias": true, "kernel_regularizer": null,
"recurrent_regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null, "recurrent_constraint":
null, "bias_constraint": null, "dropout": 0.0, "recurrent_dropout": 0.0, "implementation": 1}}, {"class_name": "Dense", "config": {"name":
"dense_1", "trainable": true, "units": 6, "activation": "softmax", "use_bias": true, "kernel_initializer": {"class_name":
"VarianceScaling", "config": {"scale": 1.0, "mode": "fan_avg", "distribution": "uniform", "seed": null}}, "bias_initializer":
{"class_name": "Zeros", "config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null,
"kernel_constraint": null, "bias_constraint": null}}]}, "keras_version": "2.2.4", "backend": "tensorflow"}
```

FIGURE 6.8. Classificationlstm.json

<u>3. classificationlstm.hdf5-</u> This file is an HDF file generated by trainlstm.py. All the weights used by the encoder-decoder LSTM network model are dumped into this file.

<u>4. conjunction.txt-</u> This file consists of conjunctions which are later used for preprocessing the user's Natural Language Text Input, and form Segmented Input List. See Fig. 6.9.

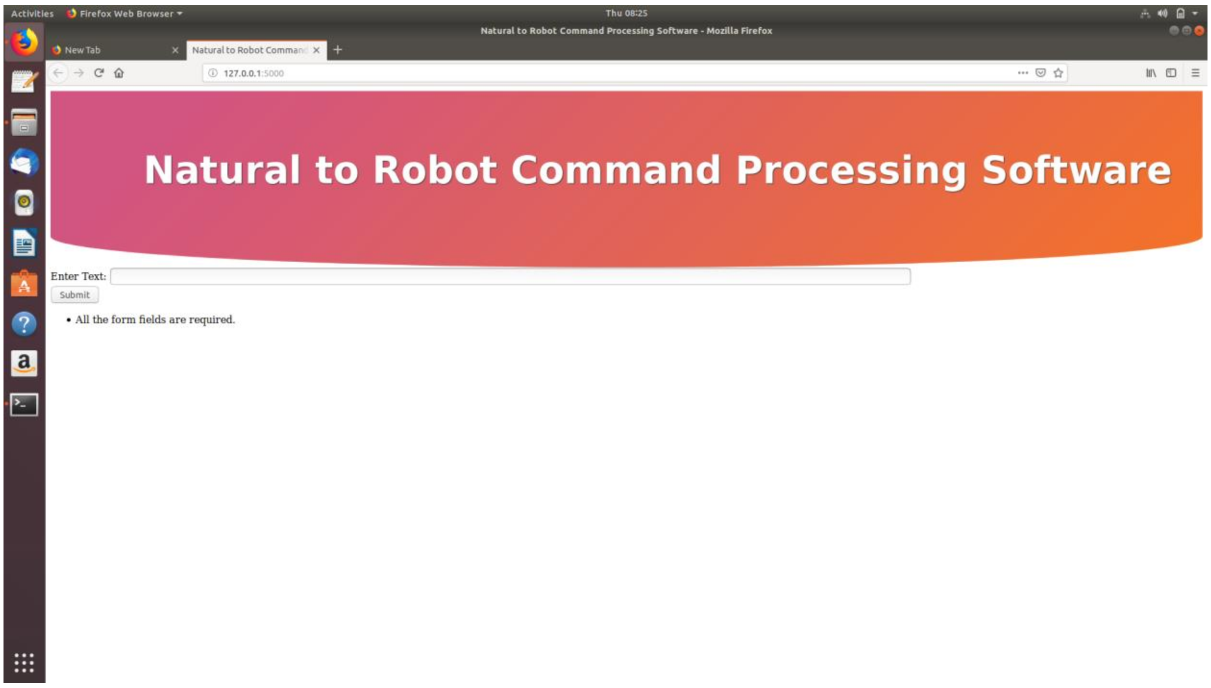


```
and , but , for , nor, or , so , yet , then
```

FIGURE 6.9. Conjunction.txt

6.6 Software System

1. Graphical User Interface- This is the user interface created via HTML template and flask connectivity through which user gives natural language text input, submits it, and receives robotic command vector output. See Fig. 6.10.



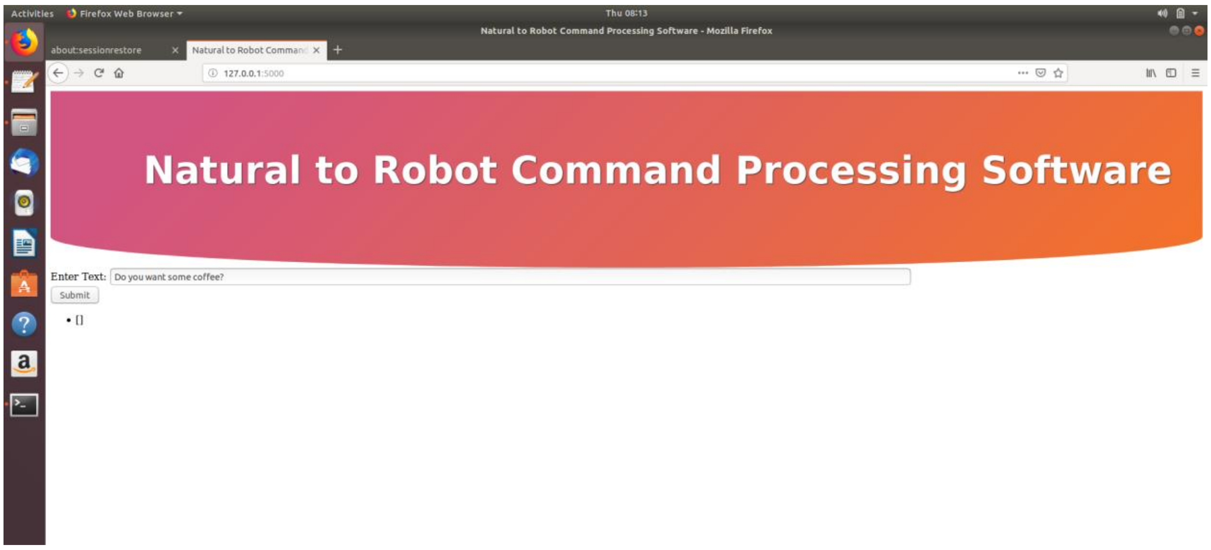
Natural to Robot Command Processing Software

Enter Text:

- All the form fields are required.

FIGURE 6.10. GUI

<u>2. Input is not intended to command the motion in robot-</u> This the case where input gets classified as Null, and we see “[]” on screen. See Fig. 6.11.

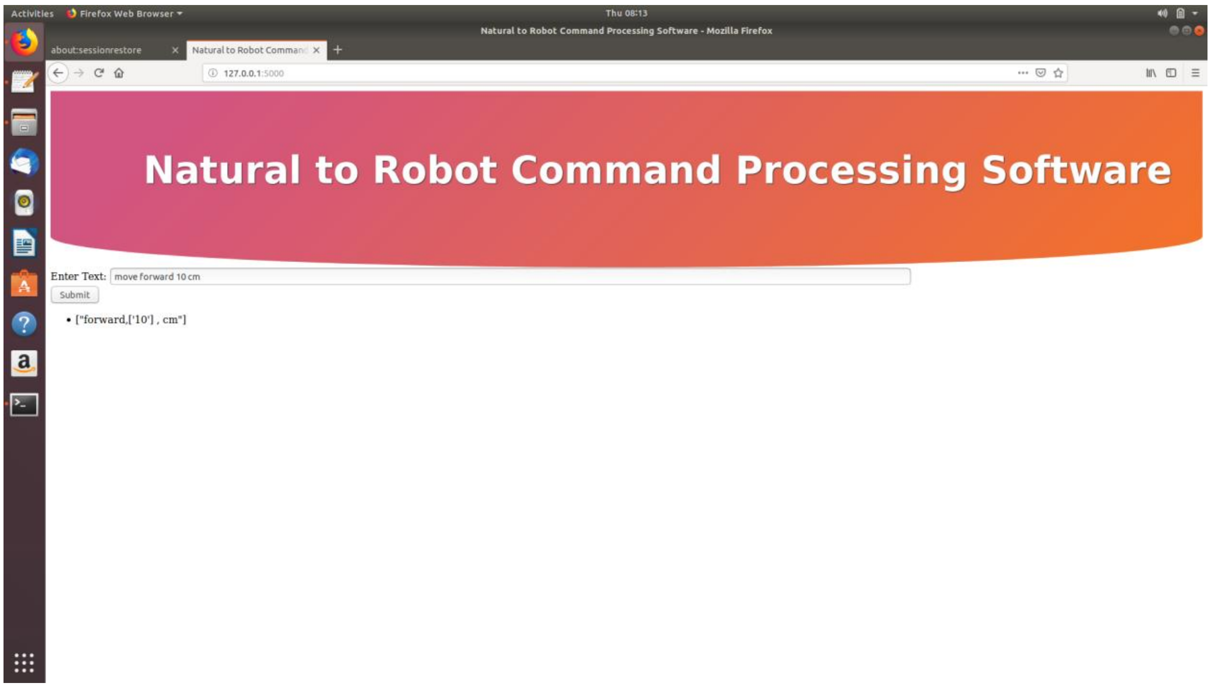

Activities
Firefox Web Browser
Thu 08:13
Natural to Robot Command Processing Software - Mozilla Firefox
about:sessionrestore
Natural to Robot Command
127.0.0.1:5000
Natural to Robot Command Processing Software
Submit

Enter Text: Do you want some coffee?

```
• []
```

FIGURE 6.11. Input Classifies as Nothing

<u>3. Simple sentence input to the robot-</u> This is simple sentence input without any conjunctions to the robot. See Fig. 6.12.

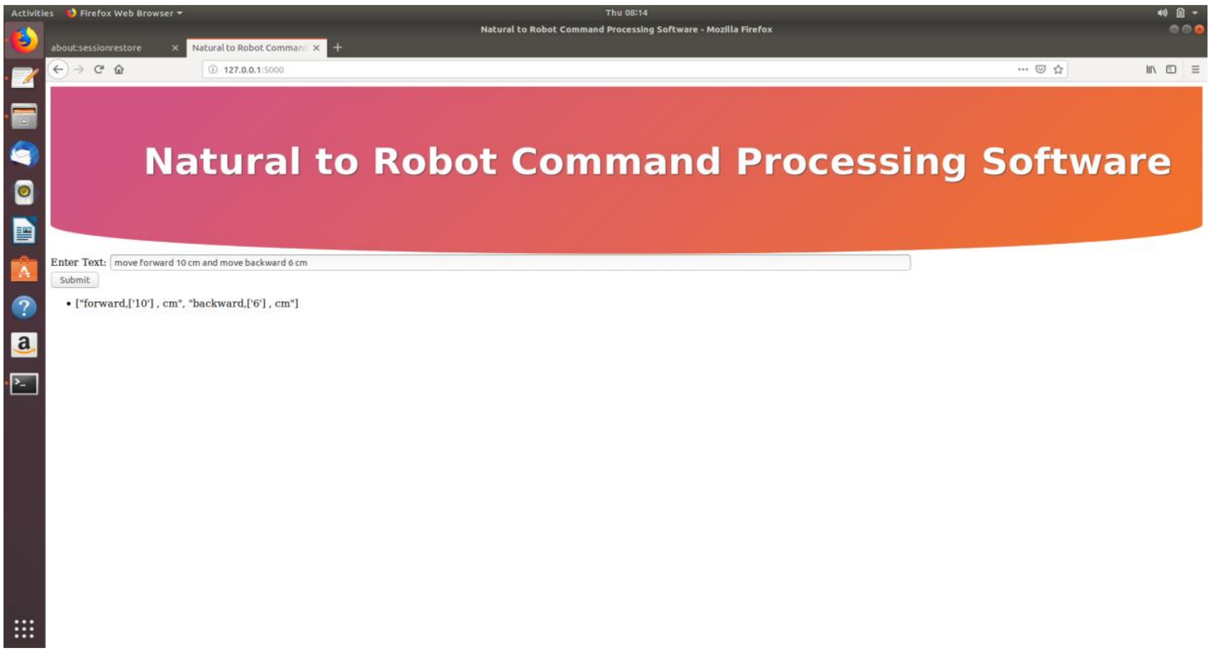

Activities
Firefox Web Browser
Thu 08:13
Natural to Robot Command Processing Software - Mozilla Firefox
about:sessionrestore
Natural to Robot Command
127.0.0.1:5000
Natural to Robot Command Processing Software
Submit

Enter Text: move forward 10 cm

```
• ["forward,['10'] , cm"]
```

FIGURE 6.12. Simple Sentence Input to The Robot

<u>4. Compound sentence input to the robot-</u> This is compound sentence input, i.e., sentence input with conjunctions to the robot, which undergoes pre-processing before command/parameter classification. See Fig. 6.13.

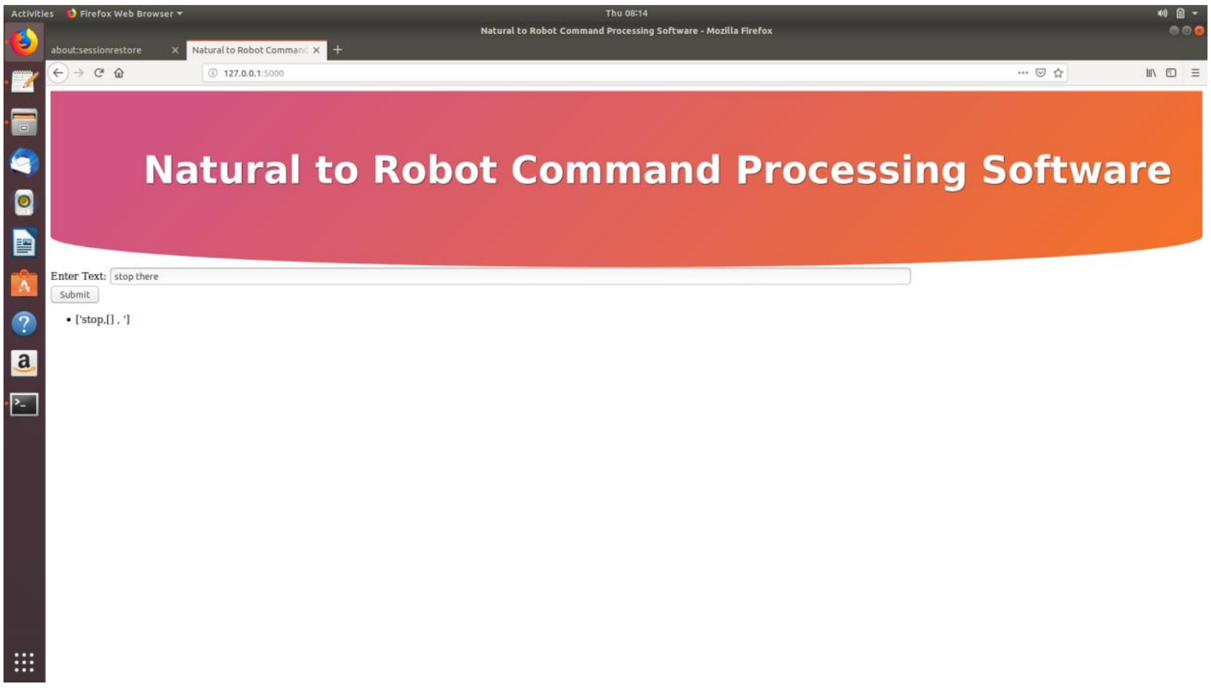

Activities
Firefox Web Browser
Thu 08:14
Natural to Robot Command Processing Software - Mozilla Firefox
about:sessionrestore
Natural to Robot Command
127.0.0.1:5000
Natural to Robot Command Processing Software
Submit

Enter Text: move forward 10 cm and move backward 6 cm

```
• ["forward,['10'] , cm", "backward,['6'] , cm"]
```

FIGURE 6.13. Compound Sentence Input to The Robot

<u>5. Stop as input to the robot-</u> This is the sentence input intended to stop the robot, and it can be combined with any other input via using conjunctions. See Fig. 6.14.


Activities
Firefox Web Browser
Thu 08:14
Natural to Robot Command Processing Software - Mozilla Firefox
about:sessionrestore
Natural to Robot Command
127.0.0.1:5000
Natural to Robot Command Processing Software
Submit

Enter Text: stop there

```
• ['stop,[] , ']
```

FIGURE 6.14. Stop as Input to The Robot

6.7 Results and Discussion

By understanding various results based on the system implementation, we get following graphs-

1. Loss v/s test_size Graph

In our proposed system, we have implemented categorical cross-entropy as a loss function. Based on that, we got the following values in Table 6.1. These values were generated during training by changing the parameter of test_size in train_test_split function, and then evaluating the model using model.evaluate function.

Table 6.1. Loss Values with respect to test_size

test_size in tran_test_split	Loss
0.1	0.00769
0.2	0.0309
0.3	0.0222
0.4	0.0062
0.5	0.0307
0.6	0.085
0.7	0.0695
0.8	0.488
0.9	1.11

If we compare loss and the size of the test dataset specified in test_size attribute of train_test_split, we get the below graph shown in Fig. 6.15.

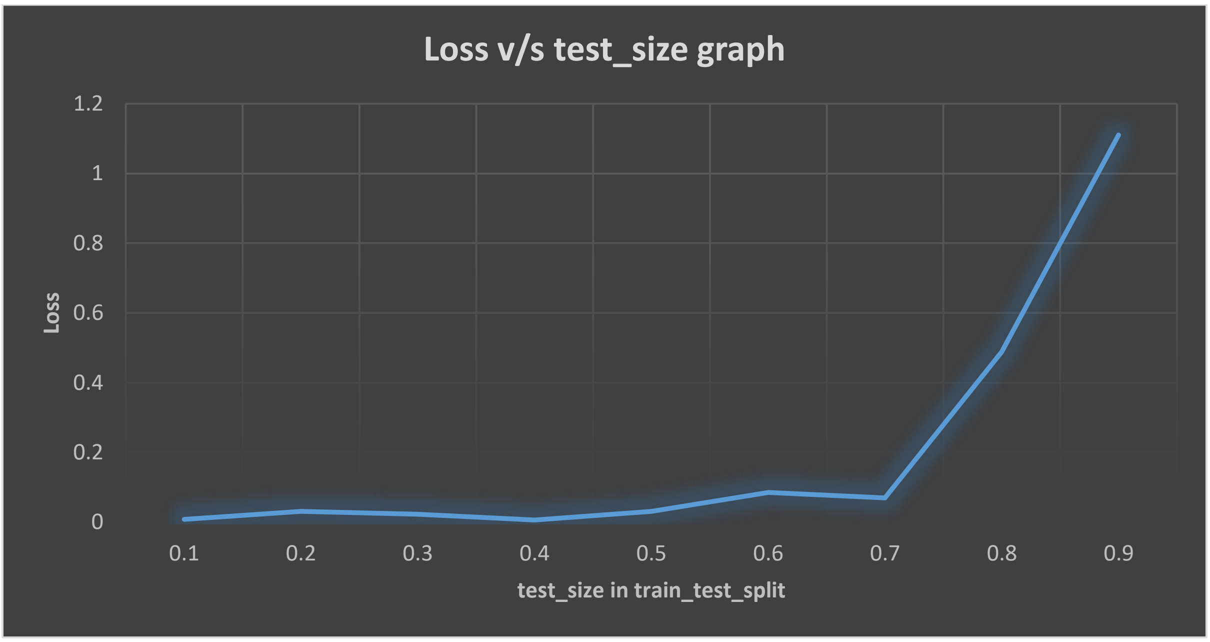


FIGURE 6.15. Loss v/s test_size Graph

Here, test_size denotes the size of the testing dataset considered for model evaluation, and loss is implemented via categorical cross-entropy function. It shows that if we increase the size of test dataset, loss increases.

2. Accuracy v/s test_size Graph

In our proposed system, we have implemented categorical accuracy as accuracy function. Based on that, we get the following values in Table 6.2. These values were generated during training by changing the parameter of test_size in train_test_split function, and then evaluating the model using model.evaluate function.

TABLE 6.2. Accuracy Values with respect to test_size

test_size in train_test_split	Accuracy
0.1	1
0.2	0.9898
0.3	0.9848
0.4	0.9878
0.5	0.8512
0.6	0.8014
0.7	0.7613
0.8	0.6517
0.9	0.5825

If we compare accuracy and the size of the test dataset specified in test_size attribute of train_test_split, we get the below graph shown in Fig. 6.16.

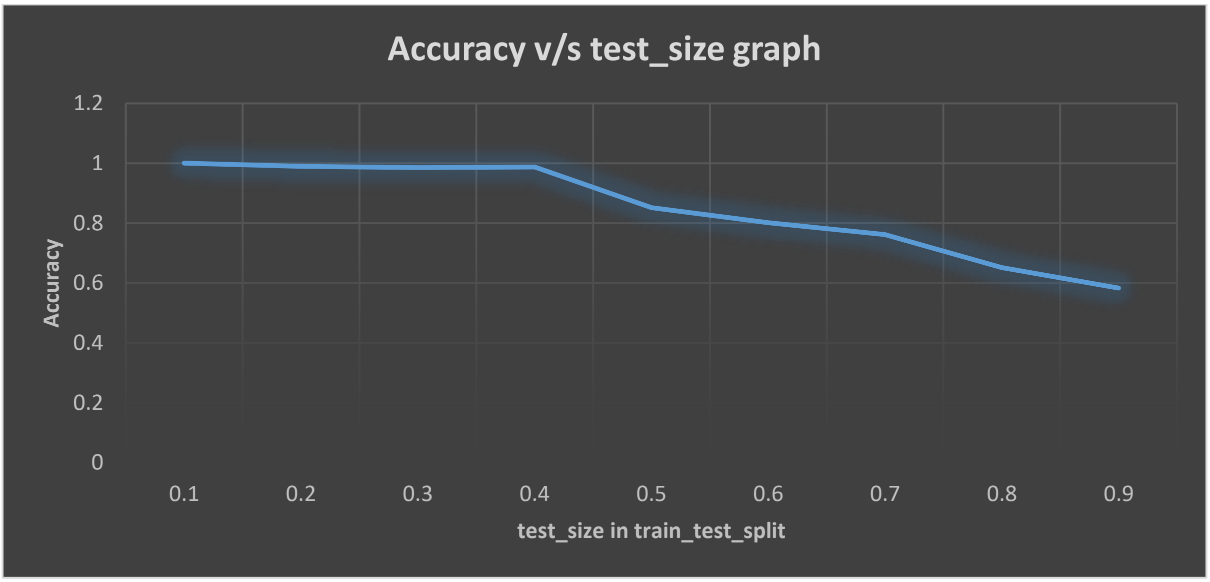
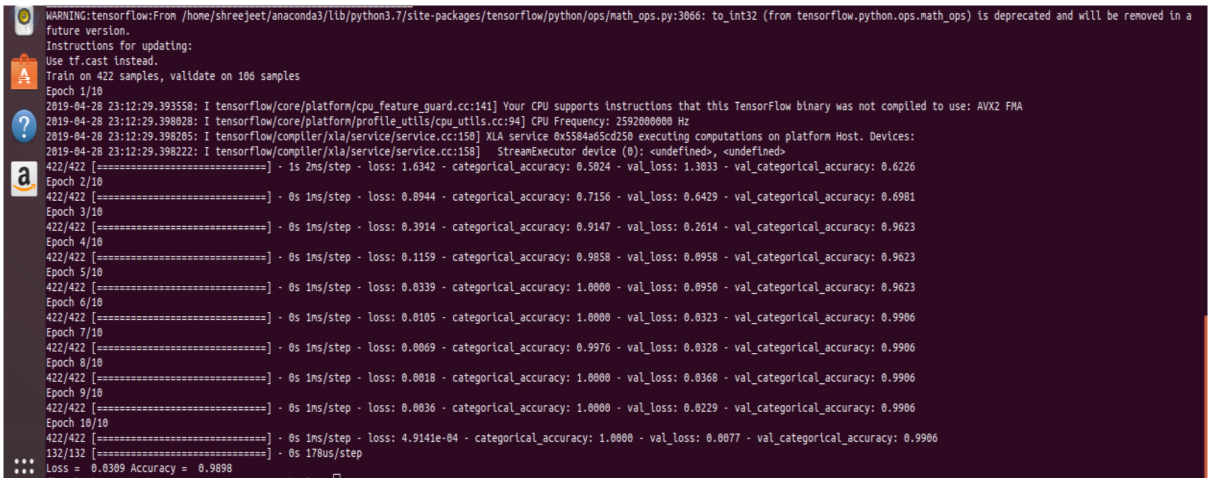


FIGURE 6.16. Accuracy v/s test_size Graph

Here, test_size denotes the size of the testing dataset considered for model evaluation, and accuracy is implemented via categorical accuracy function. It shows that if we increase the size of test dataset, accuracy decreases.

NOTE: The values for accuracy and loss were generated during training by changing the parameter of test_size in train_test_split function, and then evaluating the model using model.evaluate function. The model.evaluate function takes predicted and actual values as parameters and calculates accuracy and loss, based on loss and metrics given in model.compile function. In our case, loss is calculated via categorical cross-entropy, and accuracy via categorical accuracy function.

For eg.- While running trainlstm.py for test_size as 0.2, we get the following result on terminal. See Fig. 6.17.



```
WARNING:tensorflow:From /home/shreejeet/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Use tf.cast instead.
Train on 422 samples, validate on 106 samples
Epoch 1/10
2019-04-28 23:12:29.393558: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2019-04-28 23:12:29.398028: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2592000000 Hz
2019-04-28 23:12:29.398205: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x5584a65cd250 executing computations on platform Host. Devices:
2019-04-28 23:12:29.398222: I tensorflow/compiler/xla/service/service.cc:158]   StreamExecutor device (0): <undefined>, <undefined>
422/422 [==============================] - 1s 2ms/step - loss: 1.6342 - categorical_accuracy: 0.5024 - val_loss: 1.3033 - val_categorical_accuracy: 0.6226
Epoch 2/10
422/422 [==============================] - 0s 1ms/step - loss: 0.8944 - categorical_accuracy: 0.7156 - val_loss: 0.6429 - val_categorical_accuracy: 0.6981
Epoch 3/10
422/422 [==============================] - 0s 1ms/step - loss: 0.3914 - categorical_accuracy: 0.9147 - val_loss: 0.2614 - val_categorical_accuracy: 0.9623
Epoch 4/10
422/422 [==============================] - 0s 1ms/step - loss: 0.1159 - categorical_accuracy: 0.9858 - val_loss: 0.0958 - val_categorical_accuracy: 0.9623
Epoch 5/10
422/422 [==============================] - 0s 1ms/step - loss: 0.0339 - categorical_accuracy: 1.0000 - val_loss: 0.0950 - val_categorical_accuracy: 0.9623
Epoch 6/10
422/422 [==============================] - 0s 1ms/step - loss: 0.0105 - categorical_accuracy: 1.0000 - val_loss: 0.0323 - val_categorical_accuracy: 0.9906
Epoch 7/10
422/422 [==============================] - 0s 1ms/step - loss: 0.0069 - categorical_accuracy: 0.9976 - val_loss: 0.0328 - val_categorical_accuracy: 0.9906
Epoch 8/10
422/422 [==============================] - 0s 1ms/step - loss: 0.0018 - categorical_accuracy: 1.0000 - val_loss: 0.0368 - val_categorical_accuracy: 0.9906
Epoch 9/10
422/422 [==============================] - 0s 1ms/step - loss: 0.0036 - categorical_accuracy: 1.0000 - val_loss: 0.0229 - val_categorical_accuracy: 0.9906
Epoch 10/10
422/422 [==============================] - 0s 1ms/step - loss: 4.9141e-04 - categorical_accuracy: 1.0000 - val_loss: 0.0077 - val_categorical_accuracy: 0.9906
132/132 [==============================] - 0s 178us/step
Loss =  0.0309 Accuracy =  0.9898
```

FIGURE 6.17. Loss and Accuracy Values when test_size = 0.2

CHAPTER 7
TESTING

7.1 Test Plan

A Test Plan is a document describing software testing scope and activities. It is the basis for formally testing any software/product in a project. It is document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process. A master test plan that typically addresses multiple test levels. A phase test plan is a test plan that typically addresses one test phase.

TABLE 7.1. Test Plan

Test Plan Identifier	Master Test Plan for Natural to Robot Command Processing Software
Introduction	The objective of this plan is to perform unit testing, integration testing and black box testing on the software interface. The scope of this test plan is not limited, rather user can consider any test case. There is only one unit in all of the project and that is the software interface unit.
Test Items	Software Interface Unit/Module
Features to be tested	We need to test the above test item for the given naural language input and its portions like keywords, parameters and conjunctions.
Environmental needs	We need Ubuntu 16.04 and above versions as an operating system. Intel i5 or above processor for effective working of system. Also, we need Python3, Keras and NLTK installed.

7.2 Test Cases and Results

a. Unit Testing- Unit testing majorly focuses on the smallest unit of system or software that is the smallest part or module of the system. Its main focus is on the internal processing, logic and data structures in a particular module. Unit testing is achieved for multiple components concurrently. See Table 7.2.

TABLE 7.2. Unit Testing

Sr. No.	Part of Input	Unit Name	Expected Result	Actual Result	Status
1.	Keyword	Software Interface unit	Keyword should get recognized.	For every command, keyword gets recognized.	Pass
2.	Parameter	Software Interface unit	Any parameter should get recognized and accepted.	All parameters get recognized and accepted.	Pass
3.	Conjunction	Software Interface Unit	Any conjunction listed in conjunction.txt should get recognized, and input must be split at those points.	System recognizes the conjunctions and splits the input at those points.	Pass

b. Integration Testing- Integration testing is a systematic technique which is used for constructing the software architecture. It aims to detect uncover errors associated with interface. The different modules in this project were interfaced and tested in small increments, thus making the errors easy to isolate and correct. See Table 7.3.

TABLE 7.3. Integration Testing

Module	Input	Expected Result	Actual Result	Status
Software Interface Module	No Input	Display Error.	Error is displayed.	Pass
	Natural Language Input	Display Robot Command Vector.	Robot Command Vector is displayed.	Pass

c. Black Box Testing- It is a software testing method in which the design, internal structure or implementation of the software to be tested is not known to the tester. It is also known as Behavioural Testing because it deals with the behaviour of the system instead of going into depth of internal structure of system. See Table 7.4.

TABLE 7.4. Black Box Testing

Sr. No.	Test Case	Objective	Expected Result	Actual Result	Status
1.	Natural to Robot Language Conversion	Natural language input should get converted into robot command vector.	Display Robot Command Vector.	Robot Command Vector is displayed.	Pass

CHAPTER 8
CONCLUSION AND FUTURE SCOPE

8.1 Summary and Conclusion

Current technologies for commanding robots use word-by-word translation principles, which limits the scope of robotic movement, as they do not take into account the context of the full statement. Our system proposes the use of Neural Machine Translation(NMT) techniques which are more efficient for commanding robots using human language, with modification in base paper[1] implementation by changing user input from text to voice and also, improvement in pre-processing function to deal with wrong segmentation, for eg., in sentences without conjunction. Hence, by using NMT approach like RNN encoding-decoding, we can ease the process of Human-Machine translation by considering both context of the full message as well as the expandability of command set.

8.2 Future Scope

The software system developed can be implemented on an appropriate hardware platform on a moving bot, like raspberry pi, etc. But, due to less memory of this system, training will take time, and input and output will have delay in between. We can reduce this delay by using cloud for computations. Furthermore, the field of language understanding with Deep learning is expected to make a large impact in the coming few years while Reinforcement Learning with NLU and Deep learning will also advance in technology. As currently it's somewhat hard to teach robots natural language - We expect that RNN and MT should play a more important role in language understanding along with Reinforcement Learning. Reinforcement Learning is in frequent use for teaching movements to robots and in Game Playing, but we believe that it can be also used for language understanding with semantic parsing.

8.3 Limitations of Project Work

This project has considered a small dataset with 660 entries. We can build up more precise models by increasing the dataset size. Also, we have considered only 6 commands, but we can make this system for any number of commands we want. Also, though we have implemented the software part, we can select appropriate hardware platform having sufficient memory and processing to run this system, and make up a moving bot.

REFERENCES

[1] Wittawin Kahuttanaseth, Alexander Dressler, Chayakorn Netramai, “Commanding Mobile Robot Movement based on Natural Language Processing with RNN Encoder- Decoder” 5th International Conference on Business and Industrial Research (ICBIR), Bangkok, Thailand , 2018

[2] D. Rambabu, R. Naga Raju, B.Venkatesh, Speech Recognition of Industrial Robot. International Journal of Computational Mathematical Ideas, vol 3-no2-pp 92- 98, 2011

[3] Wu, Yonghui & Schuster, Mike & Chen, Zhifeng & V. Le, Quoc & Norouzi, Mohammad & Macherey, Wolfgang & Krikun, Maxim & Cao, Yuan & Gao, Qin & Macherey, Klaus & Klingner, Jeff & Shah, Apurva & Johnson, Melvin & Liu, Xiaobing & Kaiser, ukasz & Gouws, Stephan & Kato, Yoshikiyo & Kudo, Taku & Kazawa, Hideto & Dean, Jeffrey, “Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”,2016

[4] J. Norberto Pires, "Robot-by-voice: experiments on commanding an industrial robot using the human voice", Industrial Robot: An International Journal, Vol. 32 Issue: 6, pp.505-511, 2005

[5] Fezari, Mohamed & Bousbia Salah, Mounir, “A voice command system for autonomous robots guidance”, International Workshop on Advanced Motion Control, AMC. 2006. 261 - 265. 10.1109/AMC.2006.1631668,2006

[6] Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y, “Learning phrase representations using RNN encoder-decoder for statistical machine translation” In Conference on Empirical Methods in Natural Language Processing, 2014

[7] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 3104–3112

APPENDIX

Publications done on the project are-

[1] Rishikesh Bamdale, Shreejeet Sahay, Varsha Khandekar, “Natural Human Robot Interaction Using Artificial Intelligence: A Survey” 9th Information technology, Electromechanical and Microelectronics Conference, Jaipur, India, 2019