UNIVERSITÀ DEGLI STUDI
DI GENOVA

GROUP PROJECT - ACADEMIC YEAR 2020 / 2021

ROBOTICS ENGINEERING

MENTORE: A MOTIVATIONAL AND ENTERTAINING ONTOLOGY-BASED ROBOTIC SYSTEM FOR EDUCATION

Participants:
Aliya Arystanbek, Daulet Babakhan, Chetan Chand Chilakapati, Andrea Pitto, Syed Muhammad Raza Rizvi, Gabriele Reverberi, Sandeep Soleti, Srinivasan, Vishruth, Laiba Zahid

***Abstract*— Technology nowadays offers innovative approaches in various fields. Among these, our Group Project, *A Motivational And Entertaining Ontology-based Robotic System For Education* (also referred to as Mentore) proposes an alternative educational system that aims at improving the experience for teachers and students alike. Our work is based on ontology-driven Robotics, as it features a physical robot capable of performing convincing chitchatting with the students, performing human-like gestures in the meantime. While the robot aims at entertaining, teaching and also testing students knowledge, we additionally provide an intuitive and pleasing graphical interface for teachers to add new knowledge to the ontology (i.e. the knowledge and reasoning database), without them having to learn how to deal with ontologies directly. As our work could be potentially enriched or embedded in broader projects, we decided to share our code to the scientific community [1].**

I. Introduction

The aim of this project is to develop a conversational education system to help the Teachers and Students in carrying out educational activities. This education system will provide an Interface to the Teachers to add the information related to the Topics to teach, and on the other hand there will be a low-cost robot for the Students to have a conversation with, related to those topics for learning purposes.
The Teacher's interface is a simple-to-use program (II-A), whereas the student will be interacting with a TJBot.
For this project, an ontology based framework is used which will be populated with the knowledge by the Teachers using the GUI, and the Robot will extract knowledge from the Ontology to have a Natural language conversation with the students.

II. Implementation

A. *Graphical User Interface*

This Graphical User Interface (**GUI** from now on) will provide an interface to the Teachers to add data into the system which will be later on used in the conversation with the Students. The Teachers can add Subjects in the GUI which are the main topic about which the Teacher wants to hold the conversation about, for example Maths, Geography, History, etc. Fig. 1 shows the home page of the program. The Teacher can also add the sentences which needs to be spoken about the subject, which the Teacher entered earlier. These sentences can be of type "Positive", "Negative", and "Wait". The "Positive" sentence implies an affirmative response or a piece of fact about the subject entered. The "Negative" sentence implies that some negations are mentioned in the sentence related to the subject. A "Wait" sentence is one in which the system waits a response from the Student.
The Teacher can also add questions for the subjects, which will be asked from the Students during the conversation. The type of questions allowed in the GUI are, "Plain", "Goal", and "Contextual". The latter is the only one which also requires the Teacher to input a matching answer.

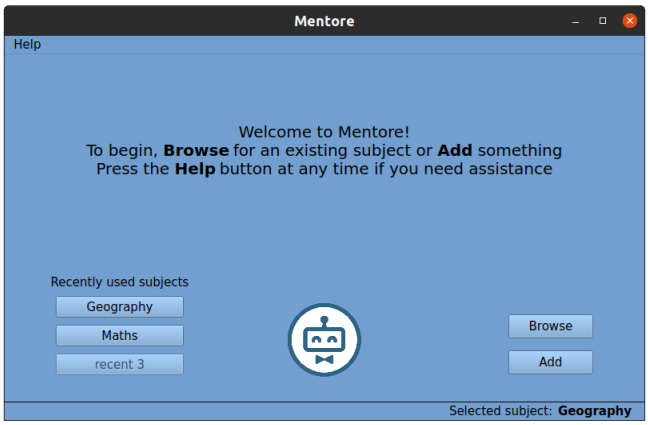

Fig. 1. Main page of the Mentore GUI

1) *Software Specifications:*

The following software specifications were used in the development of the GUI:

- Ubuntu OS, versions 18.04 & 20.04
- Python interpreter, versions 3.6 & 3.8.10
- PyQt5 libraries, versions 5.14.0 & 5.15.4
- Owlready2 library, version 0.33

2) *Overall Working:*

On the first page of GUI the Teacher can either select a pre-existing Subject, or can Add a new Subject in the ontology. The Subject can either be selected from the "Browse" button, or from the Recently used Subjects visible on the first page. Recently used Subjects is the list of Subjects

which were recently selected or modified by the Teacher. The list of recent subjects resets at each new launch of the GUI.

To Add a new Subject, the Teacher can click on the "Add" button, select "Add subject" and enter a new Subject which will be automatically added in the Ontology under the parent class "SchoolSubject". As soon as a Subject has been added, an individual of this Subject will also be created inside the Ontology and will be associated a Data Property "Likeliness", with an assigned value of 0.5 by default.

Once a new Subject has been selected, the Teacher can now Add Sentences or Questions under that subject. To Add a Sentence under a Subject, the Teacher can click on the "Add" button and choose "Add sentence". Then the Teacher can select the type of Sentence by checking the corresponding type. Once done, the Teacher can now Enter the Sentence in the Sentence Bar and click on the "Add" button. This sentence is now saved as an Object Property under that Subject.

Similarly, the Teacher can also add a question under a Subject by choosing "Add question" on the Add menu page. Then the Teacher can select the type of Question by checking the corresponding type. Once done, the Teacher can now Enter the question in the Question Bar and click on the "Add" button. This question is now saved as an Object Property under that Subject. For the question type "Contextual", the Teacher also needs to enter the answer.

At any point if the Teacher is stuck, they can access the Help menu given on the top left corner of the GUI.

3) ***Python Code Overview****:*

Separate Python files have been created for all the pages of the GUI that have all the contents of that particular page. All these Python files hold a children class of a parent class pageWindow that is present in the Python file pageWindow.py. All child classes inherit the signal function goto() from the parent class which is responsible for switching between pages.

The main idea is to have just one window open and update the contents of that window with the contents of the next window to be displayed. This depends upon the action performed by the Teacher, and all this is done in mentore.py. For defining a palette for each window, a separate Python file, mentorePalette.py has been created which holds all the properties of the palette. mentore.py is the main file responsible for all the actions mentioned below:

- Loading the contents of the Window to be displayed
- Using the palette for every window
- Receiving the data from the pages
- Updating the "Recently used subjects" list
- Updating the status bar with the currently selected subject
- Displaying the help menu
- Appending new information in the ontology through the functions defined in ontologyInterface.py

In the beginning, mentore.py loads the contents of the MainWindow on which the Teacher can interact. Along with the contents of the window, Help menu, Mentore Logo, and the Status bar displaying the Subject selected, are constantly displayed on the Window regardless of any page. The Ontology .owl file is loaded and a list of Subjects, already present in the Ontology, is created to be displayed in the Browse list. All the pages are then Registered and are associated a name to be called with later on.

Since every page also has a "Back" button, the mentore.py file also saves the name of the last page displayed so that if the Teacher clicks on the "Back" button, the contents of the window are updated according to the last window displayed. Every button on each page triggers a signal that is responsible for the next actions. Actions include passing the data to the mentore.py file, and updating the contents of the window.

If data is received from the Add Subject Page, Add Sentence page, or Add Question page, that data is sent to the mentore.py file first, and is the added into the ontology through the

ontologyInterface.py file.

B. *OWL Ontology*

The OWL hierarchy is designed in such a way to teach the students regarding School Subjects. For instance, we considered Geography, Maths and Physics subjects. In order to communicate with the students, random questions, answers and also a quiz game was designed regarding the discussed subjects in order to know the status of student's knowledge at the end of conversation.

*1) **Software Specifications**:* For the development of Ontology the following softwares are used:

- Windows 10
- Protege 5.5 [2]
- Java 1.8 older version
- Python 3.7.8

*2) **Design**:* The Ontology part has been modified using CARESSES.owl(developed by CARESSES) ([3], [4]) file which is renamed as CKB.owl. The ontology's hierarchy is very complex with a class named as SchoolSubject which are disjoint with multiple sub-classes like Geography, Maths and Physics. Each sub-class(subject) are given a set of sentences related to that particular subject, in a way that student would be able to gain knowledge on that subject. After designing the owl file it will be integrated with new-CKB.jar (Executable Java File) which has the algorithm designed according to hierarchy, Later, using the jar file user can start the conversation accordingly.

In order to make the conversation in a hierarchical way, we use the data properties like hasQuestion, hasPositiveSentence, hasNegativeSentence for questions and answers, whereas, hasQuestionContextual and hasQuestionContextualReply for making quiz game.

*3) **Data Properties**:* In contrast, a data property connects a single subject with some form of attribute data. Some of the data properties we use in particular are

- hasQuestion - Asks the Question
- hasPositiveSentence - Gives Positive sentence, if user's reply is positive
- hasPositiveSentenceAndWait - Gives Positive sentences, if user's reply is positive and wait for the user to respond
- hasNegativeSentence - Gives Negative sentence, if user's reply is Negative
- hasQuestionContextual - Asks the random question during quiz (End of the sentence is indicated by £1 to identify it is a question 1 and likewise...)
- hasQuestionContextualReply - Waits for the user answer and if it is correct, moves to next question, or, if wrong, it keeps asking the same until user replies correct answer (End of the sentence is indicated by £1_c for correct answer and £1_w for wrong answer and likewise...)

In order to identify the sentences, instances are added to each sub-class and will be assigned with the data property assertions such as, the keywords and the likeliness values [5]. In particular, we use two keywords for each instance to identify the topic(for example, "geography"-keyword1, "*"-keyword2). Keywords are mainly to used to trigger the topics, in order to jump from one topic to another, Whereas, likeliness values, which can range from 0 to 1, were left at their default value of "0.5". These values are dynamic and get modified according to user's reply. Low likeliness "0.0" is assigned for negative replies and high likeliness "1.0" for positive replies.

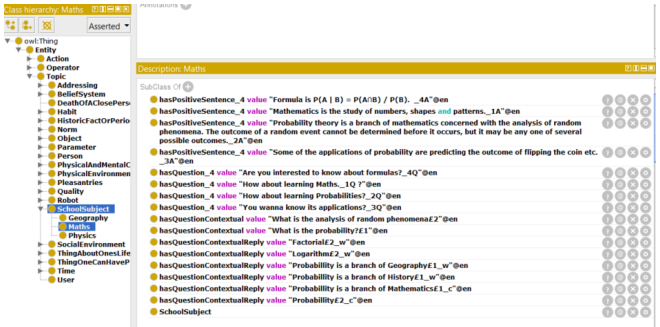

Fig. 2. Ontology - OWL File

*4) **Algorithm**:* Here, we're given to tackle some challenges in the existing algorithm developed by

CARESSES. In main, we use dialogue_manager.py to reply to user irrespective of likeliness values. Also, we work with test_dialogue_manager.py file on the server side whereas, TTS and STT module works on clients side to achieve replies. For instance, using this file each user is given a client ID for each conversation respectively.

*5) **Integrating and Launching** :* Initially, we run the CKB.owl using the command line java -jar new_CKB.jar -nationality Indian -virtual 1 in the terminal. Then the owl file generates a list of files like sentences.enu file which has all the topics and sentences in a text format, triggering_keywords to trigger the topics and some other list of files(.txt, .json, etc). All the files which are generated after running the owl file are rooted into a folder named as dialogue_tree.

After the integration, a folder named FLASK_CARESSES which has dialogue_manager.py, test_dialogue_manager.py, credentials.txt and other python files which are integrated with each other. Then before launching both scripts, check that the clients folder into Flask_CARESSES is empty, and remove the file credentials.txt from the same folder (this is only required the first time, to be sure that there are no initialization problems). Now, using the terminals we need to run dialogue_manager.py and on the other terminal test_dialogue_manager.py in order to start the conversation.

It is important to root the folder in such a way that dialogue_tree folder is inside the FLASK_CARESSES, and FLASK_CARESSES has empty clients folder and other python files, as all the files are interlinked with each other.

C. *Text To Speech and Speech To Text protocols*

1) Text To Speech (TTS): To give speaking functionality for TJbot, we need to convert the text data received from ontology into speech. For that, "pyttsx3" has been used, which is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3. The pyttsx3.drivers module will be able to load and use directly and its the best available driver for the platform, currently:

- Sapi5 - SAPI5 for Windows

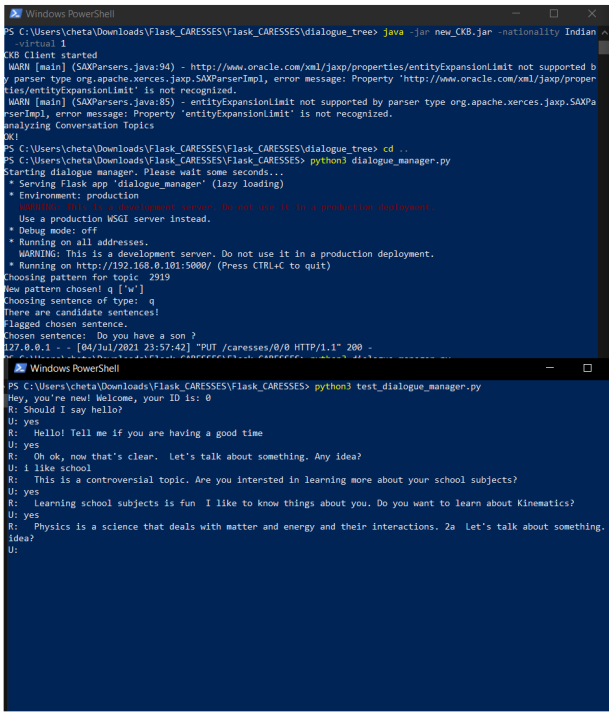

Fig. 3. Launching the Files and Developing the Conversation

- Nsss - NSSpeechSynthesizer for Mac OS X
- Espeak - eSpeak for other platform

Python code overview: First, It initializes the engine using the function pyttsx3.init(), and it gets a reference to an engine instance that will use the given driver. getproperty() gets the engine properties such as rate, volume, and voice for the speech data. Then Using the setproperty() function it sets all the values and parameters according to the user needs. By defining the voice id 1 for male and 0 for female in the setproperty() function for different voices.

say() function Queues a command to speak an sentence or line. The speech is output according to the properties set before this command in the queue.

2) Speech to Text (STT): To obtain user input over speech and feed it to the ontology, converting them from speech to text is essential. The text is given as input to the ontology, and the result from the system is displayed as text and speech using the same text to speech function.We have used Speech Recognition3.8 library in python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3. This library

is capable to support several recognised engines:

- CMU Sphinx (offline)
- Google Speech Recognition
- Google Cloud Speech API
- Snowboy Hotword Detection (offline)
- IBM Speech To Text

Dependencies: PyAudio 0.2.11 and SpeechRecognition 3.8.1 PyAudio is to get started with playback and record audio in Windows, Linux, and Mac OS in the python environment. PyAudio provides Python bindings for PortAudio, the cross-platform audio I/O library. With PyAudio, you can easily use Python to play and record audio on a variety of platforms. PyAudio is required to get input (Microphone) Speech Recognition uses google API in Python to recognise speech and convert them to text.
Python code overview: First, with recognizer(), initialize the audio recognition and Microphone() object here to read the audio from the default microphone. Then the duration parameter in the record() function is used to stop reading after 10 seconds and then upload the audio data to Google to get the output recognize_google() this function converts the audio received from the microphone to text. The output text is feed as input to the ontology. Also, you can recognize different languages by passing language parameters to recognize_google() function. The default language is set to English.

D. Hardware

The hardware part concept started from the TJbot design from IBM, but added several improvements regarding gestures, proxemics and facial expressions.

1) Tjbot: The TJBot (picture4), from the American company IBM, is an open-source project born on November 9, 2016. Its outer chassis can be built from laser-cutted cardbord, wood or polymer, or 3D printed. Its inner electronic components are:

- 1 Raspberry Pi Model 3
- 1 16GB NOOBS Card
- 1 USB Microphone
- 1 USB Speaker
- 1 micro-Servo
- 1 RGB LED
- Jumper wires

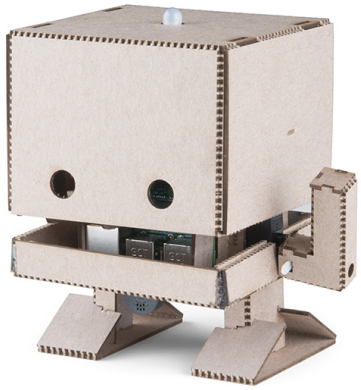

Fig. 4. The TJBot from IBM

As seen in picture 4, the TJBot only has one arm, moved by the servo displayed in the list above, without any other moving parts. The RGB LED placed on the head on the robot can change colour, a feature that can be used to display a basic setup of pre-coded expressions or emotions. This setup has the advantage to be very simple, but it does not guarantee a wide range of gestures to be displayed. The power supply is guaranteed by the USB cable, connected to a 5v transformer (the USB PC port can work as well)

2) Mentore: In this section, the Group 2 from Emaro RobEng 2019-2020 presents a prototype, specifically designed to better second the needs displayed in the Mentore Project requirements.

Starting from the TJBot concept, several motion features are added. Here can be found the list of the materials used to build the prototype:

- 1 Raspberry Pi Model 3B+
- 1 64 GB Micro SD card with Raspbian OS
- 1 USB Microphone
- 1 USB Speaker
- 4 micro-Servos
- 2 MG996 metal gear servos
- Jumper wires
- 2 aluminium servo brackets
- 1.5 mm section wires
- USB plug
- 400W Desktop computer power supply
- wooden main chassis
- polymer cover and appendices
- 1 3,5 inches capacitive touch screen

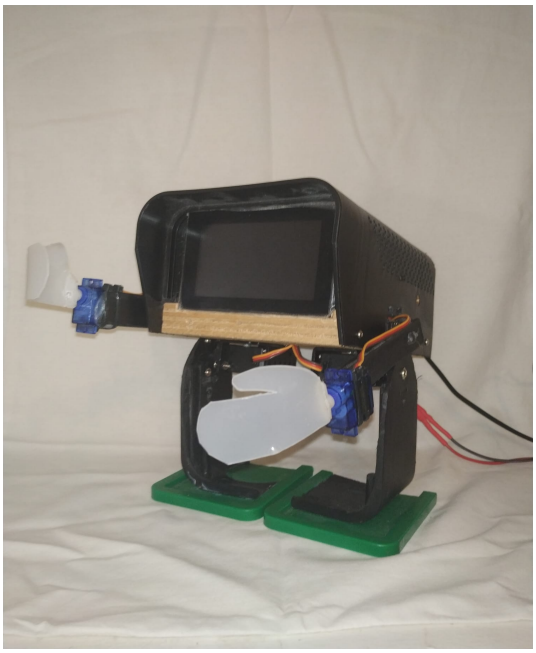

Fig. 5. Mentore, from Emaro RobEng group 2

Fig. 6. Mentore, from Emaro RobEng group 2

In this setup the main electronics is connected to the wooden chassis, including 2 of the micro servos moving the arms. Two additional servos are used to move the wrist mechanisms and make the polymer hands rotate. Below the main chassis, the two aluminium brackets hold in place the two MG996 servos, each one connected to a polymer leg.

The power supply (picture 6) consists in a 400 W modified Desktop computer power supply, in which only GND and +5V contacts have been used. This power supply is oversized when compared with the motion tasks to be accomplished. Further implementations will include the use of a properly dimensioned battery as main power supply. Considering the possible current overloads that can occur by connecting the power line of 6 servos to the Raspberry board, the only servo jumpers connected to the board are the control ones; GND and positive jumpers are instead connected straight to the power supply. Picture 7 displays a simplified scheme of the cable connections. For readability the connection of only one servo is included.

The screen can be used in further implementatons to display facial expression or visual content when required.

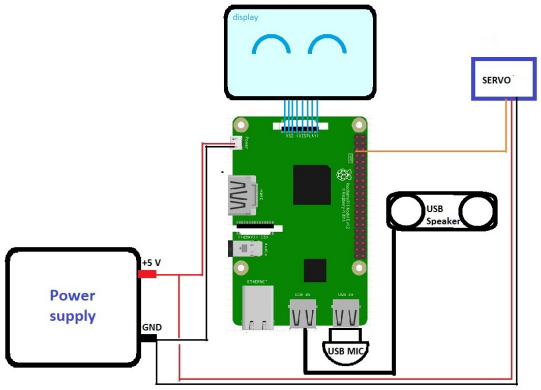

Fig. 7. Mentore, circuit scheme

III. Conclusions

In this Group Project we dealt with different environments and problems with the aim, in the end, of wrapping up everything together by delivering a fully fledged conversational robot, which exploits an embedded educational system for Teachers and Students to use. The GUI we developed features some useful capabilities, which allow Teachers to interact with ontologies without the need of them being expert in the field. We decided to focus on producing a robust and easy-to-use interface, that may then be expanded in the future with new features, notably the ability of

also letting the Teacher remove subjects, sentences and questions. We tried to design the GUI in order to make it flexible and easy to update, and also to give it intuitive and pleasant aesthetics. Regarding the ontology, on a whole, by developing the OWL file in a hierarchical way, student is able to gain knowledge regarding the school subjects. Using triggering keywords(for Eg., Maths), user can direct into his chosen subject and can start conversation. The conversation carries in a way, to better understand the topics of the subjects. Also, the user can test the status of his knowledge regarding the subjects covered by using the Question and Answer game. In future, the hierarchy can be designed in a more complex way to add more school subjects i.e., designing the hierarchy like a text book for students to better understand. The addition of a text-to-speech and speech-to-text functionality gives the TJbot ability to communicate with the user more naturally as humans communicate. This develops the overall system intuitive and communication between the user and system much more convenient and comfortable. This is designed to work offline without the internet, and making the system standalone. Regarding the hardware part, several improvements can still be done, such as implementing a better control and load distribution to overcome the servos jitter. Another possible improvement can be to properly dimension a battery, rather than using the wired separate power supply. Balance of the robot is overall quite good and a possible working code pattern could be implemented as well.

REFERENCES

[1] Aliya Arystanbek, Chetan Chand Chilakapati, Daulet Babakhan, Andrea Pitto, Syed Muhammad Raza Rizvi, Gabriele Reverberi, Sandeep Soleti, Srinivasan, Vishruth, and Laiba Zahid. Mentore Group Project repository. https://github.com/andreabradpitto/Mentore-Group-Project, 2021.

[2] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building owl ontologies using protégé 4 and co-ode tools edition1. 2. *The university of Manchester*, 107, 2009.

[3] Carmine T Recchiuto and Antonio Sgorbissa. A feasibility study of culture-aware cloud services for conversational robots. *IEEE Robotics and Automation Letters*, 5(4):6559–6566, 2020.

[4] Carmine Recchuto, Luna Gava, Lucrezia Grassi, Alberto Grillo, Marta Lagomarsino, Davide Lanza, Zijian Liu, Chris Papadopoulos, Irena Papadopoulos, Antonello Scalmato, et al. Cloud services for culture aware conversation: Socially assistive robots and virtual assistants. In *2020 17th International Conference on Ubiquitous Robots (UR)*, pages 270–277. IEEE, 2020.

[5] Barbara Bruno, Carmine Tommaso Recchiuto, Irena Papadopoulos, Alessandro Saffiotti, Christina Koulouglioti, Roberto Menicatti, Fulvio Mastrogiovanni, Renato Zaccaria, and Antonio Sgorbissa. Knowledge representation for culturally competent personal robots: requirements, design principles, implementation, and assessment. *International Journal of Social Robotics*, 11(3):515–538, 2019.