Contents

*These authors contributed equally to this work.

1 Introduction

Improving human-robot interactions and thus providing encouragement and helps to students with more technological creativity is an important topic facing the 21st century. How humans and robots engage in the creative process and contribute to creative productivity is a central research question connecting psychology and technology. Over the past few decades, research has explored the possibility of using social robots in a range of university campuses, including teachers and instructors [1]. The motivation behind these efforts ranges from exploring robots as technology to support learning to the concept of robots as a solution to a variety of challenges in services.

Figure 1: Sanbot Robot for School, Teaching Assistant Robot

In today's globalized context, more and more universities in non-English speaking countries are introducing courses in English as the language of instruction, and students from many countries do not have a good command of the local language at the beginning, which poses many challenges for their integration into campus life. Researchers have designed a number of robots and applications to support education through various methods. Our project aims to better integrate students from all over the world into campus life through language translation and conversation.
In fact, according to relevant studies, international students are at considerable risk of academic failure [2], and one of the very main reasons for this is the language barrier, as well as the inability of students to adapt to a new culture after entering it and not being present in class and thus not keeping up with the pace. But with the increasing number of students in recent years, school tutors are unable to serve everyone in detail. Thus we had the idea of making this pepper robot - DIAGo. In our project, you can see DIAGo, a pepper

robot, interacting with international students in a range of functions such as conversation, language translation, explanation of knowledge, and prompting of course-related information.
In this work, we also try to focus on several aspects of the Human-Robot Interaction (HRI) field and its importance in education. Especially the psychological improvements it brings to students when interacting with them. In particular, when we put this DIAGo robot in a specific college campus setting, the aim is to facilitate faster integration of international students, reduce the workload of campus teachers, and become part of the overall college environment.
The idea is to place the DIAGo on a university campus where there are international students from all over the world, many of whom are not fluent in the local language. Students can interact with the robot through a tablet (placed on the robot's chest) or they can talk to it directly. DIAGo can talk directly to students through English and do a lot of actions in the middle of talking to simulate a friend to talk to, thus relieving students' loneliness. DIAGo can also translate the English spoken by students into the local language, thus enabling students to become familiar with foreign language.
And when students encounter terms they don't understand, they can ask DIAGo directly to get an explanation from Wikipedia. Many students are unfamiliar with the campus environment and the course schedule, so DIAGo can give an introduction to the course and the time and place of the course so as not to waste students' time.
Then, to let DIAGo be more social and sustain a more natural interaction, we implemented several gestures. For example, whena the DIAlanguageshe student dialogue in a specific situation, the DIAGo will make some special movements to simulate the reaction of the real person dialogue, and the DIAGo will also make some entertaining movements to make the learning process more interesting.

Figure 2: Pepper robot

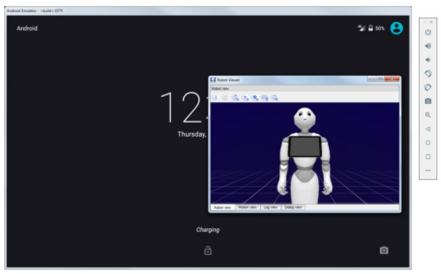

Figure 3: Pepper emulator

Our project was developed mainly in python using NAOqi SDK from Softbank robotics, to connect directly to the robot or do the simulation, which provides the needed APIs. We also used MODIM (Multi-modal Interaction

Manager) to start the server that we use for interacting with the tablet. Given the covid situation, we could not work on the physical robot, hence we used the SDK for the Android emulator, see Figure 2 because it allowed us to analyze the behavior of the robot in an almost real way based on our instructions. To summarize, the benefits of having an assistant robot like the one we propose in our prolanguages.
In summary, the benefits of having an assistant robot like the one we presented in the project are.

- Increase the mental health of students by providing continuous social contact and stimulation Interaction.
- Quickly guides students to the correct classroom when they are unfamiliar with the schedule.
- Reduces the teacher's workload.

Reasoning agents are another important aspect of our project. The basic concept of inference is that a new fact can be deduced in a given set of basic facts in a way that we have set up in advance. The parts usually included in reasoning are the study of ways to achieve a goal in the presence of various obstacles and the study of how to move from one state to another. To do this, there is some respect that need to be considered :

- World (or Domain) model
- Formalisms for world properties (including goals)
- Formalisms for actions
- Models for plans and strategies
- Algorithms

And in this project, we considered the goal of placing DIAGo in the actual place of DIAG, where there are many new students who cannot find the classroom and other facilities successfully, so DIAGo can perform the function of navigation by touching the tablet.

2 Previous Works

2.1 HRI: Previous works

2.1.1 Human-robot Social Interaction

As we know, the interactions between humans and robots aren't easy thing. In fact, there are many difficulties.

- Task Dynamic Analysis:
  The challenges of task planning and simulation in time, space, force, energy, and cost.
- Teaching a Robot and Avoiding Unintended Consequences:
  Rapid advances in computer-based speech understanding (e.g., Apple's SIRI) promises ease in commanding robots. But there is a great possibility for unintended consequences.
- Interfacing Mutual Mental Models to Avoid Working at Cross-Purposes:
  Eliciting mental models from humans of what robots can or should do, and combining that knowledge with the computer's model for purposes of planning and conflict avoidance, remains an HRI challenge.
- Role of Robots in Education:
  Understanding how people of different ages and abilities best learn from robots remains an important challenge that human factors should contribute to.
- Human Values:
  If we could automate everything, what would we want to be automated and what not be automated?

But there are some companies that have developed this kind of robot. Mattel has developed a new Barbie doll with an extensive speech and language recognition vocabulary that is linked via the Internet to the company server [3]. The doll is designed to carry on an extensive conversation with young girls or boys in areas of their interest. And Knox [4] presents a case study of applying a framework for learning from human feedback to an interactive robot. Knefel [5] questions whether one ever should fully trust a robot.Use of robot interaction in education is not a new idea. That effort evolved into the current commercial LEGO Mindstorms product for children.

2.1.2 Bio-inspired Robotics

After a long period of development, robots inspired by biological forms have come a long way. Some of the features are envisioned as follows.

- **Muscle-tendon complex**: Muscle-like motor functions can be realized, for example, with pneumatic artificial muscles or electro-active polymers. these actuation technologies are used in many bio-inspired robots.
- **Skin sensors**: Pressure-sensitive artificial skins relying on various transduction mechanisms (force-sensitive resistors as used on the hand;mems-based pressure sensors), type of substrate, and spatial resolutions have been developed. touch remains hard to mimic.

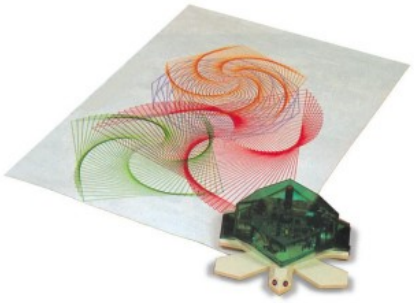

Figure 4: The LOGO language of Papert (1980) used robotic "turtles" as a means for children to learn elementary programs

- **Retina**: Bio-inspiration has also been applied to analog computing in visual sensors as well as a space-variant distribution of photoreceptors—typically a high density in the center, and a low density at the periphery.
- **Computing units**: Different approaches of distributed computation are being exploited in robotics projects, for example, sensor networks and distributed control for multi-robot systems.

The pepper robot can also be described as a robot designed based on bio-inspired design - with a human target, so you can see some of the generic features of this type of design reflected in the pepper.

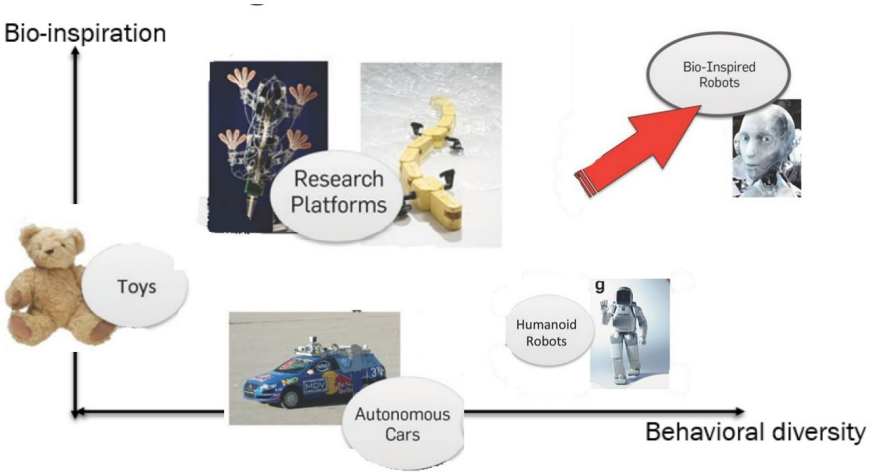

Figure 5: Define the biological resemblance and Behavioral diversity

2.1.3 Research on Joint Action

Joint action is very important for our project. Because the object of our project is to transform positive emotion from DIAGo to students, it means we design some entertainment functions (sensors). the students will also feel happy. there are two types of coordination- planned, and emergent coordination. planned coordination means agents' behavior is driven by desired outcomes of joint action. emergent coordination means perception-action couplings make multiple individuals act in similar ways. For Instance, pedestrians often fall into the same walking patterns, people engaged in conversation synchronize their body sway obviously, we want to stimulate joint action in students in emergent coordination. Schmidt presented an experiment in 1997[5], two persons sitting side by side moved a hand-held pendulum. One person used her left hand and the other person used her right hand so that the two pendulums were located in between the two persons. the result is: the relative phase between the two movements was much more frequently close to 0 and 180 during the second half of the trial than during the first half of the trial. It means Emergent coordination can sometimes interfere with individual action planning. So we can use the phenomenon, as DIAGo can achieve vary gestures, like dance, the happy emotion will infect the user, to relax students.

2.2 RA: Previous works

One of the most important goals of artificial intelligence is to develop intelligent agents, such as robots, capable of deliberating their course of action. Path planning is an important research focus task for reasoning agents. Historically, path planning has been divided into cases with and without maps. In the case of known maps of the environment, path planning is considered as an optimization problem whose goal is to find the best solution based on a given criterion (e.g., shortest path). Automated search is one way we can use it, which is a sequence of actions, so the search algorithm works by considering various possible sequences of actions. These possible sequences of actions starting from the initial state form a search tree, with the initial state of NODE at the root; the branches are actions and the nodes correspond to states in the state space of the problem. The state space of the problem[6]. The general approach we consider is called best-first search. , where a node is interpreted as a cost estimate according to an evaluation function f(n), so that the node with the lowest evaluation is expanded first. Most best-first algorithms include as a component of f a heuristic function that denoted as:
h(n)= estimated cost of the cheapest path from the state at node n to a goal state.
Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.
The problem, however, is that the above approach deals with atomic representations of states and thus requires good domain-specific ones to perform well. Hybrid propositional logic agents can find plans because it uses domain-

independent heuristics based on the logical structure of the problem. However, it relies on the ground (invariant-free) propositional reasoning, which means that it may have some problems with reasoning, which means it can be overwhelming when there are many actions and states. In response to this, planning researchers have settled on a factored representation—one in which a state of the world is represented by a collection of variables. We use a language called PDDL, the Planning Domain Definition Language. The language supports the following syntactic features:

- Basic STRIPS-style actions
- Conditional effects universal quantification over dynamic universes (i.e. object creation and destruction)
- Domain axioms over stratified theories
- Specification of safety constraints.
- Specification of hierarchical actions composed of subactions and subgoals.
- Management of multiple problems in multiple domains using different subsets of language features (to support sharing of domains across different planners that handle varying levels of expressiveness)[7].

3 Solution

Our work is developed entirely in a Linux environment, and because we use docker, we can create one or more independent containers. Thereby modifications to the whole system are not necessary, as we can work independently in these containers. The exact principle is achieved by providing an additional abstraction at the OS level, by virtualizing at the OS level, by executing the following commands

```
cd <this_repository>/docker
./run.bash
```

```
docker exec -it pepperhri tmux a
```

3.1 HRI: Solution

As mentioned above, our project is built in Python, whose role 1 lies in managing the libraries provided by NAOqi, and 2 is for communicating with HTML

files for managing the tablet. As described in Section 1, we use the Pepper robot, which uses the NAOqi operating system, an embedded Linux distribution developed specifically to meet the needs of SoftBank Robotics.
SoftBank Robotics provides several programs and libraries, known as NAOqi APIs, that enable the robot to come to life through conversations and other means.
For example, robots can talk, move, make all kinds of gestures, and even dance. All of the above makes the robot come to life. Our software is also architected to build on this foundation.
In order to communicate with the robot, first we need to start the NAOqi server, but in this project, it is activated when we open the SDK of the Android emulator. Then, with the help of the existing tool - the pepper tool - and SoftBank's documentation, we can achieve the entire interaction with the robot. For example, we can use these tools to move the robot, have conversations with people, move in the direction of people, and many other functions. Also, we use a lot of pepper tools in our work. Let us briefly analyze each of them in detail.

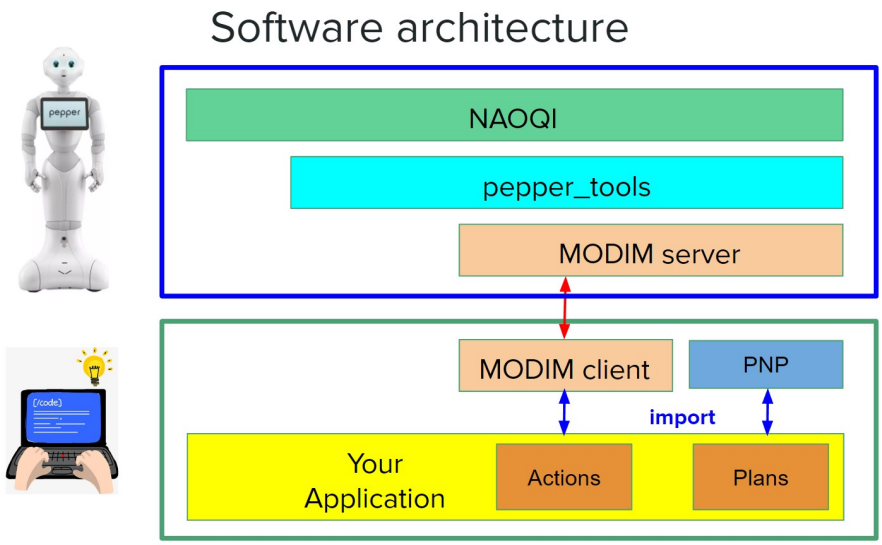

Figure 6: Define the "biological resemblance" and "Behavioral diversity"

- **ALTextToSpeech** is a module that allows bots to speak. We can set specific text-to-speech commands on it, or we can set a text-to-speech strategy, and the output we get from manipulating this module will be sent to the robot's loudspeaker.

- **ALMemory** is a centralized memory that stores all the information about the pepper hardware, the current state of the actuators and sensors. In

addition, this module can be used to store and retrieve named values and serve as a hub for distributing event notifications.

- **ALAudioPlayer** provides playback services (play, pause, stop and loop, etc.) for a wide range of audio file formats and associated generic functions. In all cases, the generated audio stream is sent to the robot's amplifier.

- **ALDialog** is a module that allows you to formulate the content of the robot's dialogues using certain strategies. By using a list of "rules" written and categorized in an appropriate way, you can give the robot conversational skills.

- **ALMotion** module Provides methods for facilitating robot motion. In particular, it contains four main groups of i) joint stiffness, ii) joint position, iii) walking (distance and speed control, world position, etc.), and iv) robot end effector control in a Cartesian system.

We also use MODIM, a service that is characterized by the fact that the specific functions of some interactions are defined first and then function, and because of this feature, these interactions can be easily invoked.
The available modes are: i) text-to-speech. ii) automatic speech recognition, allowing voice interaction with the robot. iii) writing text on the tablet touch screen. iv) Assign a title to the current mode and display it on the tablet touch screen. v) Generate buttons on the touchscreen layout. vi) Load any desired image on the tablet screen.
Also, it is possible to add many user definitions in the action file itself, for example adding commands in other languages than the default language (English). This could be a very important feature if we put Pepper in a multicultural environment.

3.2 RA: Solution

Based on the related work presented in the previous section, we decided to choose PDDL as our solution for the path planning problem. The PDDL is intended to express this objective world domain, what actions are available, what behaviors are possible, what the structure of the compound actions is, and what the effects of the actions are. But that alone is still not enough, because most planners also need some kind of guidance, i.e., about what actions to use in what situations to achieve what goals. We can decompose the language into several subsets of features, called requirements. Each domain defined using PDDL should declare which requirements it takes on.

```
    (define (domain museum-domain)
  (:requirements :strips)
  (:predicates (at ?x ?y) (adj ?x ?y) (ro ?x))

  (:action move
    :parameters (?a ?from ?to)
    :precondition (and (at ?a ?from)
           (adj ?from ?to)
           (ro ?a))
    :effect (and (not (at ?a ?from)) (at ?a ?to)))
)
```

```
    (define (problem museum-problem)
  (:domain museum-domain)
  (:objects sq-1-2 sq-1-3 sq-1-4 sq-1-5 sq-1-6 sq-1-7
            sq-2-0 sq-2-1 sq-2-2 sq-2-3 sq-2-4 sq-2-5
            sq-2-6 sq-2-7 sq-3-3 sq-4-0 sq-4-1 sq-4-2
            sq-4-3 sq-4-4 sq-4-5 sq-4-6 sq-4-7 robot
            )

  (:init (adj sq-1-2 sq-1-3) (adj sq-1-3 sq-1-2)
         (adj sq-1-3 sq-1-4) (adj sq-1-4 sq-1-3)
         (adj sq-1-4 sq-1-5) (adj sq-1-5 sq-1-4)
         (adj sq-1-5 sq-1-6) (adj sq-1-6 sq-1-5)

         (adj sq-2-0 sq-2-1) (adj sq-2-1 sq-2-0)
         (adj sq-2-1 sq-2-2) (adj sq-2-2 sq-2-1)
         (adj sq-2-2 sq-2-3) (adj sq-2-3 sq-2-2)
         (adj sq-2-3 sq-2-4) (adj sq-2-4 sq-2-3)
         (adj sq-2-4 sq-2-5) (adj sq-2-5 sq-2-4)
         (adj sq-2-5 sq-2-6) (adj sq-2-6 sq-2-5)
         (adj sq-2-6 sq-2-7) (adj sq-2-7 sq-2-6)

         ...

         (at robot sq-3-3)
         (ro robot)
  )
  (:goal (and (at robot sq-4-7)))
)
```

We can represent states by combining many fluxes that are ground, functionless atoms. The representation of states is carefully designed so that a state can be considered either as a conjunction of fluxes, which can be manipulated by logical inference or as a collection of fluxes, which can be manipulated by set operations.

```
    if __name__ == '__main__':
    a = Action('move', [['?ag', 'agent'], ['?from', 'pos'], ['?to',
                                          'pos']],
                       [['at', '?ag', '?from'], ['adjacent', '?from
                                                ', '?to']]
                                                ,
                       [['at', '?ag', '?to']],
```

```
                    [['at', '?ag', '?to']],
                    [['at', '?ag', '?from']])
print(a)

objects = {
    'agent': ['ana', 'bob'],
    'pos': ['p1', 'p2']
}
types = {'object': ['agent', 'pos']}
for act in a.groundify(objects, types):
    print(act)
```

Actions are described by a set of action patterns, ACTIONS(s) and RESULT(s, a) functions to represent the actions and the results that can result from the actions. The precondition defines the states in which the action can be executed, and the effect defines the result of executing the action. The very important part then is to figure out what changes and what remains the same. What is the result of an action and what stays the same? Classical planning focuses on the problem that most actions leave most things unchanged. PDDL does this by specifying the outcomes of actions.

4 Implementation

In this section, we will explain the implementation of our project in detail, describing step by step all possible interactions and the corresponding code. In particular, as we will see later, we try to make the interaction as multimodal as possible, handling it through multiple communication channels, written, voice, tablet, and motion. To make the robot more socially acceptable and to better serve the students.

4.1 HRI : Implementation

4.1.1 AI Communication

We make the voice communication between human and robot as the most basic function, which can close the distance between robot and human and make people trust DIAGo more. provide a good atmosphere for further interaction later.
At the very beginning, DIAGo will greet humans in a friendly way. Students can chat with DIAGo through talk plus conversation content.In this section, we use the dialogue window of the pepper robot simulator to simulate human voice input

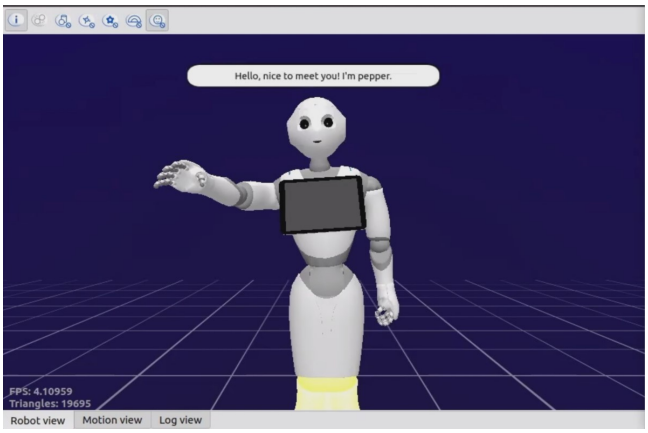

Figure 7: Define the "biological resemblance" and "Behavioral diversity"

Here we set up the basics of trying to connect to the Pepper and have some conversations, slowing them down considering that international students don't always speak English very well. A specific explanation of the ALTeextSpeech section can be found in the Solution section. The interaction begins with a welcome message. DIAGo is initiated via TTS mode. Also, in an asynchronous manner.DIAGo makes a "hello" gesture, moves its right arm, wrist, and hand Details about all the animations will be given in more detail later in the motion section. From now on, humans can interact with robot by writing. Only writing is allowed in the simulated environment, but in fact, it is also possible to have a direct conversation in the real robot. In the project demonstration, we interact in the dialogue view provided by the simulator.

```
if __name__ == "__main__":
parser = argparse.ArgumentParser()
parser.add_argument("--ip", type=str, default="127.0.0.1",
                    help="Robot's IP address. If
                    on a robot or a local Naoqi use
                    '127.0.0.1' (this is the default value)")
parser.add_argument("--port", type=int, default=9559,
                    help="port number, the default value is OK
                    in most cases")
parser.add_argument("--action", type=str, default='hello',
                    help="port number, the default value is OK
                    in most cases")
parser.add_argument("--sentence", type=str, default="hello",
                    help="Sentence to say")
parser.add_argument("--language", type=str, default="English",
                    help="language")
parser.add_argument("--speed", type=int, default=400,
                    help="speed")
args = parser.parse_args()
session = qi.Session()
try:
    session.connect("tcp://{}:{}".format(args.ip, args.port))
```

```
    except RuntimeError:
        print ("Can't connect to Naoqi at IP {} (port {}).
        Please check your script's arguments."
        " Run with -h option for help.".format(args.ip, args.port))
        sys.exit(1)
    strsay = args.sentence
    language = args.language
    speed = args.speed
```

As you can see, we use the event "Dialog/LastInput" from the ALMemory APIs.ALMemory APIs, which we will use to get the last answer of the robot based on the user input.

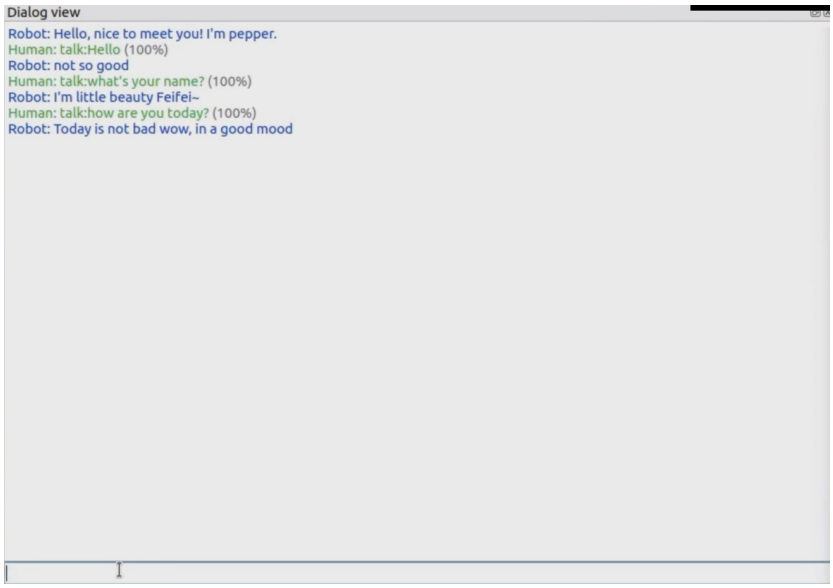

Figure 8: Dialog window

Typically, the event is handled by the ALMemory module, in which we link to an answer file. This is a powerful file supporting features such as weather, translations, jokes, lyrics, calculations, domain information/record/inclusion queries, IP queries, cell phone number attribution, etc.

```
    tts_service = session.service("ALTextToSpeech")
    tts_service.setLanguage(language)
    tts_service.setVolume(1.0)
    tts_service.setParameter("speed", speed)


    ALDialog = session.service('ALDialog')
    ALMemory = session.service('ALMemory')
    ALMotion = session.service("ALMotion")
```

```
    tts_service = session.service("ALTextToSpeech")
    audio_player_service = session.service("ALAudioPlayer")

    # Setup ALDialog
    ALDialog.setLanguage('English')
    vision = Vision()
    gesture = Gesture(ALMotion, doGesture, vision, tts_service, )
    # Start dialog
    ALDialog.subscribe('pepper')
    lastInput = ALMemory.subscriber("Dialog/LastInput")
    lastInput.signal.connect(dialog_0)

    main(session,args)
```

Conversations are managed through. Top files, which represent the topics of interaction.At this point, the interlocutor and DIAGo are still in a relatively unknown relationship, so at the beginning, we added a lot of human-robot interaction, and we designed DIAGo head, hand, and dance movements. We designed the head, hand, and dance movements of the DIAGo to more closely resemble the dialogue in a real-life situation.

```
def dialog_0(lastInput):
    text = lastInput.lower()
    print("text:", text)
    word = text.split(':')
    if 'talk' in word[0]:
        talk = word[1]
        talk = translate(talk, 'zh-CN')
        res = requests.post("http://api.qingyunke.com/api.php?key=
                                        free&appid=0&msg=" + talk
                                        )
        res = res.json()
        strsay = res['content']
        strsay1 = translate(strsay.encode('utf-8'), 'en')
        gesture.look_hand()
        tts_service.say(strsay1)
```

As the statistics show that many of the master programs offered at DIAG are taught in English, and as described in the introduction section, international students largely fail in their studies because of language issues, we believe it is important to add a translation function to DIAGo.

```
if 'translation'in word[0]:
        gesture.look_hand()
        talk = word[1]
        trans = translate(talk, 'it')
        tts_service.say(trans)
        print('Robot: ', trans)
```

We don't just want DIAGo to be a conversationalist, we want him to be a teacher, a robot that can really impart knowledge to the students, so we added the ability to query concepts, where students can gain knowledge by WIKI plus the term they wish to have explained. We found in practice that sometimes,

there are web contents that are not introductions and a bunch of codes, like this web page of Italy. So we set up the intercept content as the content of the first two p tags. Because of the loading speed, users sometimes have to wait for a long time to get an answer. So we designed DIAGo to keep waving this action while loading, in order to reduce the anxiety of waiting.

```
if word[0]=='wiki':
        gesture.look_hand()
        talk = word[1]
        url = 'https://en.wikipedia.org/wiki/' + talk.replace(' ',
                                                '_')
        web = requests.get(url)
        soup = BeautifulSoup(web.content,'lxml')
        find_all = soup.find_all('p')

        rule = re.compile('<[^>]*>|\n\n|\[|\]|, ')
        abstrict = rule.sub('',str(find_all[1:2]))
        abst = abstrict.split('.')
        for a in abst:
            if a == '\n':
                continue
            tts_service.say(rule.sub('', a))
            print( "Robot: "+rule.sub('', a))
```

There are two ways the most used MODIhumans

- **Direct call**, where each function can be called directly by im.executeModality (MODALITY, INTERACTION)command is called directly, specifying the desired interaction and what it will execute.
- **Invoke Action**, which we can work with indirectly through simple commands like im. execute(ACTION) or im.ask(ACTION).

In our project, we only included the second method because it is easier to read and neater. Also. MODIM allows having visual feedback of the work just done by rendering on our local web page all the things we implemented on its "touch screen" with the im.displayLoadURL() command.
Next is the code that calls the pepper tool and the SoftBank file, with which we can implement the entire interaction with the robot. The specifics of these modules are explained again specifically in this section.

```
def main(session, args):
    print("Hello, nice to meet you! I'm pepper.")
    tts_service.say("Hello, nice to meet you! I'm pepper.")
    gesture.doHello()
    stop_flag = False

    while not stop_flag:
        print("What do you want? You can choose to talk,
                                        translation, wiki, dance
                                        in dialog.")
```

```
        print("Or you can do the interaction, touch head, touch
                                            hand.")
        mod = raw_input("Please choose: head, hand, dance: ")
        word = mod
        if word == 'exit':
            stop_flag = True
        if word == 'head':
            gesture.touch_head()
            tts_service.say("???")
        if word == 'hand':
            gesture.touch_hand()
            tts_service.say("???")
        if word == 'dance':
            gesture.dance_0()
            audio_player_service.playFile("$HOME/playground/Pepper-
                                           Interaction/project-
                                           pepper/mytest/pop.wav
                                           ", _async=True)

    print("Byebye")
```

4.1.2 Course Guide

The beginning of the semester is a crucial period for new students to get familiar with the campus environment. Many new students are not familiar with the specific schedule of the courses and miss many of them, so this section aims to introduce the courses to new students through the tablet.
In fact, all the pages we see are actions, actions that we manage by asking and executing functions. It depends on whether we expect an answer from the user or not.

```
import ws_client
from ws_client import *

def i1():
    im.init()
    im.ask('welcome')  # wait for button
    q = random.choice(['animal']) #,'color'
    a = im.ask(q)
    if (a!='timeout'):
        im.execute(a)
        im.execute('goodbye')
    im.init()
if __name__ == "__main__":
    mws = ModimWSClient()
    # local execution
    mws.setDemoPathAuto(__file__)
    # remote execution
    # mws.setDemoPath('<ABSOLUTE_DEMO_PATH_ON_REMOTE_SERVER>')
    mws.run_interaction(i1)
```

when the tablet is running, we will see the following home page:

Figure 9: Home page

Figure 10: Ask for the start

At this point, we reach the core of the tablet. Here a number of specific courses are displayed and students can click on different courses to view them. What we have just described is shown below.

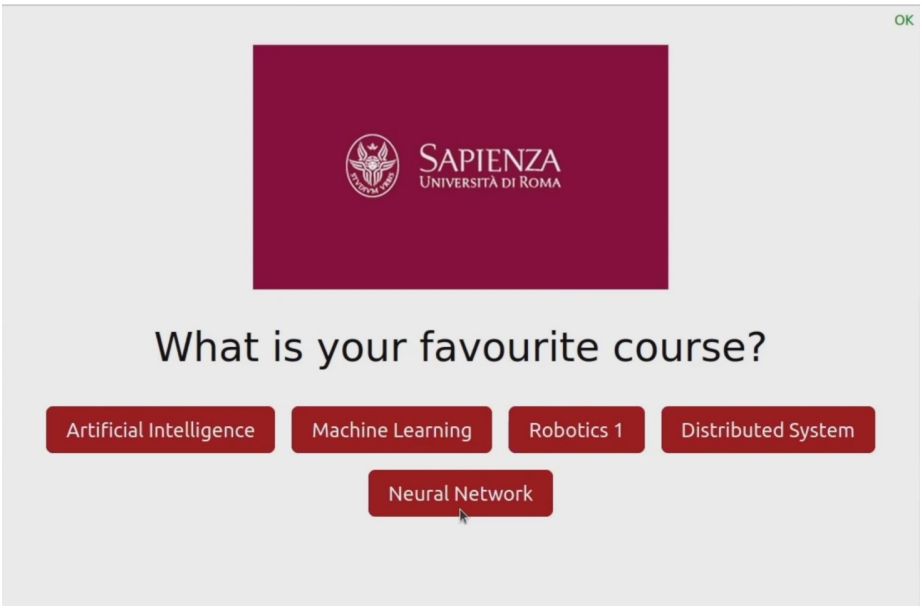

Figure 11: Choose course

Here is the basic introduction to the course, class location, and schedule.

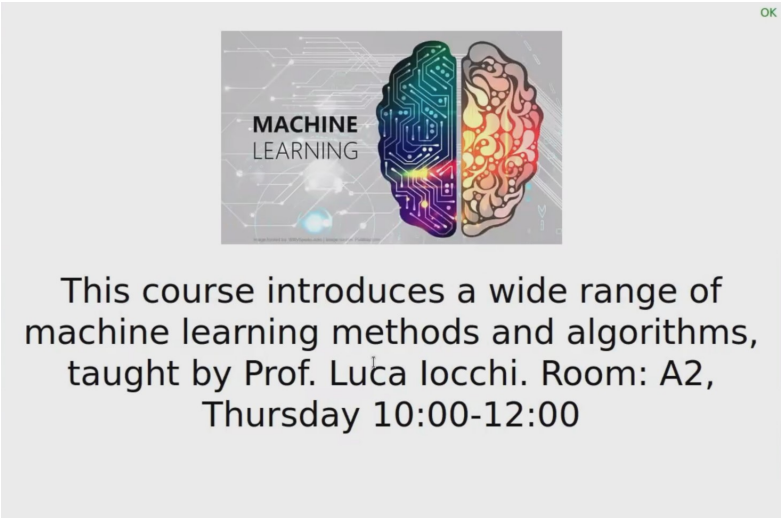

Figure 12: Course details

After the message has been displayed for a period of time, a Goodbye screen is displayed and automatically returns to the main screen.

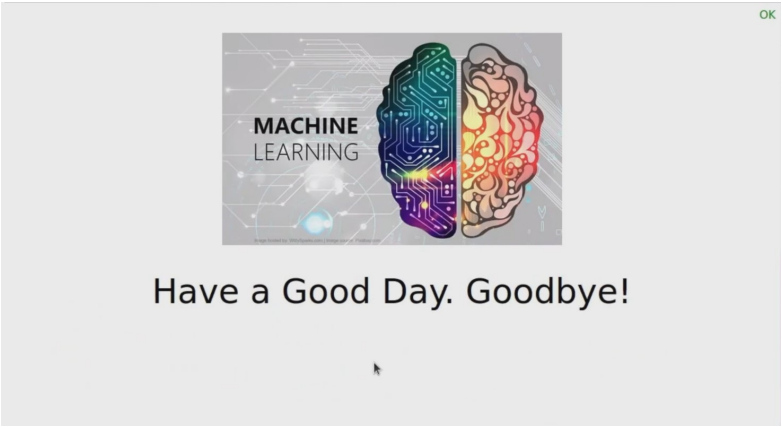

Figure 13: Goodbye

As we can see, we mainly use the ask function, but after the course information has been displayed for a while, we use the execute function. Finally, when the user finishes interacting with DIAGo using the tablet mode, it can always return to the text or sound interaction.

4.1.3 Action and Entertainment

DIAGo acts as a robot, and users can also interact with the robot directly through actions, such as DIAGo waving its arms while performing noun explanations. Or as an entertainment feature, DIAGo dances to music, or touches its head or hands.
As you can see from the code below, when the user is done using the tablet, or before using it, he can interact with DIAGo directly through the commands of the action.
If the user makes a command to dance, DIAGo will play the music that was played while dancing. At the same time. DIAGo will dance to the beat of the music, trying to attract the attention of the interlocutor and make him feel happy.

```
if word == 'exit':
            stop_flag = True
if word == 'head':
            gesture.touch_head()
            tts_service.say("???")
if word == 'hand':
            gesture.touch_hand()
            tts_service.say("???")
if word == 'dance':
            gesture.dance_0()
            audio_player_service.playFile("home/yourname/playground
                                          /Pepper-Interaction/
                                          project-pepper
            /mytest/pop.wav",
            _async=True)
```

The music starts when we call the function playFile of the ALAudioPlayer NAOqi module.

```
if 'dance' in word[0]:
            audio_player_service.playFile("/home/kantsen/playground
                                          /Pepper-Interaction/
                                          project-pepper/mytest
                                          /pop.wav", _async=
                                          True)
            gesture.dance_0()
```

DIAGo's dance movements are achieved by implementing specific hand gestures. Gestures are handled by the angle of the joints in the robot's body parts, or more precisely, by the design of Pepper's roll, pitch, and deviation angles. As an example, "LShoulderPitch" indicates the pitch angle of its left shoulder. We built the dance, touching the robot's head, and touching the robot's hand movements. The following code shows some examples of our actions.

```
    def dance_0(self):
```

```
        joints = ['HeadPitch', 'HeadYaw',
                  'RElbowYaw', 'RElbowRoll', 'RShoulderPitch',
                  'LElbowYaw', 'LElbowRoll', 'LShoulderPitch',
                  "RHand", "LHand",
                  "HipRoll", "HipPitch"]
        times = [0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3,
                                          0.3, 0.3]
        for i in range(50):
            angles = [-0.5, 0, 0.48, 2, 0.1, -2.53, -2, 0.2, 0, 0,
                                              -0.15, -0.17]
            self.ALMotion.angleInterpolation(joints, angles, times,
                                               True)
            angles = [0.5, 0, 1.9, 2, 0.1, -1.03, -2, 0.2, 0, 0, 0.
                                             15, -0.17]
            self.ALMotion.angleInterpolation(joints, angles, times,
                                               True)
```

In the gestures shown in the code above, we can see the number of times and angles for the movements of the different parts. You can see that the connection of one configuration to another, and therefore the execution of the actual motion, is handled by the ALMotion module. We can also see that the time we want to complete these gestures is required.

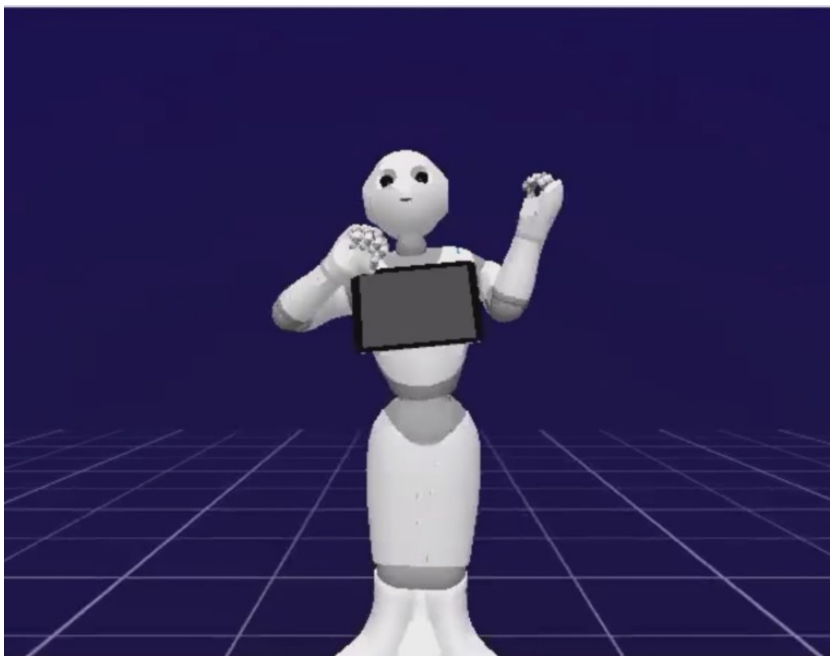

Figure 14: Pepper dance

Finally, the action of human-robot interaction is not just an expression from the robot to the human, but the human should also be able to express his or her

feelings to the robot. So we designed the robot to be able to feel human touch on specific body parts. If this happens, DIAGo will look around in confusion. The user can touch the robot's head, right arm, or left arm. The following code shows the described module.

```
    def touch_head(self):
            joints = ["HeadPitch", 'RElbowRoll', 'RElbowYaw', '
                                                RShoulderPitch']
            times = [1, 1, 1, 1]
            angles = [0.5, 4, 0.9, -0.5]
            self.ALMotion.angleInterpolation(joints, angles, times,
                                                True)
            angles = [0, 0, 1.47, 1.6]
            self.ALMotion.angleInterpolation(joints, angles, times,
                                                True)

    def touch_hand(self):
            joints = ["HeadYaw", "HeadPitch", 'RElbowRoll', '
                                                LElbowRoll']
            times = [0.8, 0.8, 0.8, 0.4]
            angles = [0.05, -0.1, 2.9, -1.2]
            self.ALMotion.angleInterpolation(joints, angles, times,
                                                True)

            angles = [-0.04, 0.2, 0, 0]
            self.ALMotion.angleInterpolation(joints, angles, times,
                                                True)
```

In the simulator, we can only imitate the user's touch on DIAGo in a virtual way, through the sTouched function, a fictional way, without a real robot at our disposal. In this case, to handle the touch, we use the NAOqi ALMemory module to input values to the corresponding sensor (the touched part of Pepper's body).

4.2 RA:Implementation

First, we need to write a server file, after that many files need to be called through the server file. If we get different destination inputs, we start planning the path. You can see that we output the path plan and can display it on the map, and the robot will say the path.

```
    if message == 'A1':
            path, paths = pathfind(rooms[message])
            write_path(message, 'The path founded: ' + path)
            drawpath(paths)
            say('The path founded: ' + path)

        elif message == 'WC':
            path, paths = pathfind(rooms[message])
            write_path(message, 'The path founded: ' + path)
            drawpath(paths)
```

```
            say('The path founded: ' + path)

        elif message == 'A2':
            path, paths = pathfind(rooms[message])
            write_path(message, 'The path founded: ' + path)
            drawpath(paths)
            say('The path founded: ' + path)

        elif message == 'A3':
            path, paths = pathfind(rooms[message])
            write_path(message, 'The path founded: ' + path)
            drawpath(paths)
            say('The path founded: ' + path)

        elif message == 'door':
            path, paths = pathfind(rooms[message])
            write_path(message, 'The path founded: ' + path)
            drawpath(paths)
            say('The path founded: ' + path)

        elif message == 'food':
            path, paths = pathfind(rooms[message])
            write_path(message, 'The path founded: ' + path)
            drawpath(paths)
            say('The path founded: ' + path)

    ...


        last_answer = message
```

On the tablet, if the path planning (i.e. room button) is selected and the JS file listens for this button, it is passed back to the server. The action files are called and displayed on the tablet with several destination options.

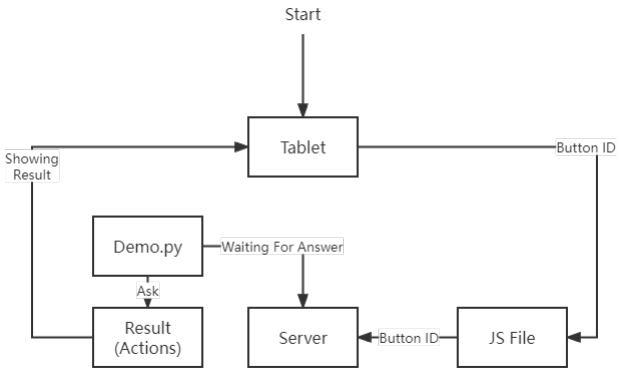

Figure 15: Flowchart

These files are responses to buttons with content inside that can be displayed on the web page.

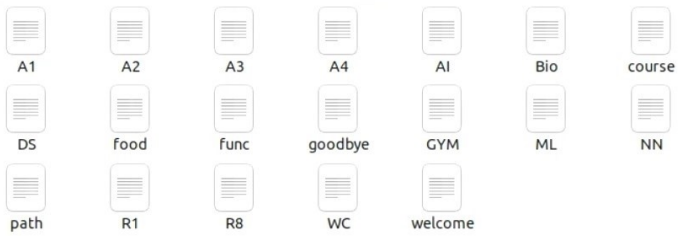

Figure 16: button files

When the user clicks a button on the web page, it responds to the user's action and sends the button information to the server file.

```
      function button_fn(event) {
  var bsrc = event.srcElement || event.originalTarget
  console.log('websocket button '+bsrc.id)
  wsrobot_send(bsrc.id);
}
```

Receive input in the server file, But at this point, the path file has not been generated yet, so the image file and path file do not exist yet.

```
    def on_message(self, message):
        global last_answer
        print('Input received:\n%s' % message)
        self.write_message('OK') # reply back to ws client
        # store answer for program client
```

Define the function for path planning, The definition of the domain, problem, and action files are defined or accepted by the server file, at which point we can start path planning.

```
    def pathfind(goal_str):
            domain = './path1/d1.pddl'
            problem = './path1/p1.pddl'
            planner = Planner()
            plan = planner.solve(domain, problem, goal_str)
            if type(plan) is list:
                path = ''
                paths = []
                paths.extend(plan[0].parameters[1:])

                for act in plan[1:]:
                    paths.append(act.parameters[2])


                paths_pos = ['({}, {})'.format(pos[3], pos[5]) for
                                                    pos in paths]
                path = ' -> '.join(paths_pos)
            else:
```

```
            print('No plan was found')
            exit(1)
        return path, [(int(pos[3]), int(pos[5])) for pos in
                                                paths]
```

As you can see, If we get different destination inputs, we start planning the path, you can see that we output the path plan and can display it on the map, and furthermore, the robot will say the path. Here is an example if our destination is classroom A1.

```
    if message == 'A1':
            path, paths = pathfind(rooms[message])
            write_path(message, 'The path founded: ' + path)
            drawpath(paths)
            say('The path founded: ' + path)
```

Output paths on the tablet, DIAGo details here on how to get from your current location to your destination location.

```
def write_path(file_name, path):
    with open('/home/robot/playground/html/sample1/actions/' +
                                        file_name, 'w') as f:
        f.write('IMAGE\n')
        f.write('<*,*,*,*>:  img/a_map.png\n')
        f.write('----\n')
        f.write('TEXT\n')
        f.write('<*,*,*,*>: ' + path + '\n')
        f.write('----\n')
        f.write('TTS\n')
        f.write('<*,*,*,*>: ' + path + '\n')
        f.write('----\n')
```

Plot the planned path on the map. The map shows how to get to the destination according to our planned path so that students who are not yet familiar with the terrain can understand it more clearly and intuitively.

```
def drawpath(paths):
  img = cv2.imread('/home/robot/playground/html/sample1/img/map.png
                                        ', 1)
  x_bias = -20
  y_bias = 50
  points = [[0 for i in range(8)] for j in range(8)]
  for i in range(0, 8):
    for j in range(0, 8):
      points[i][j] = ((j+4)*53 - x_bias, (i+3)*53 - y_bias)
  for i in range(len(paths) - 1):
    x1, y1 = paths[i][0], paths[i][1]
    x2, y2 = paths[i+1][0], paths[i+1][1]
    cv2.arrowedLine(img, points[x1][y1] ,points[x2][y2], color=(255
                                        , 0, 255), thickness = 2)
  cv2.imwrite('/home/robot/playground/html/sample1/img/a_map.png',
                                        img)
```

DIAGo says the planned path, as we showed in the HRI section about the powerful language capabilities of DIAGo, we can also speak out the paths we have planned.

```
def say(strsay):
    pip = '127.0.0.1'
    pport = 9559
    language = 'English'
    speed = 100
    session = qi.Session()
    try:
        connection_url = "tcp://" + pip + ":" + str(pport)
        session.connect(connection_url)
    except for RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + pip + "\" on
                                        port " + str(pport) +".\n
                                        "
               "Please check your script arguments. Run with -h
                                                option for help.")
        sys.exit(1)

    tts_service = session.service("ALTextToSpeech")

    tts_service.setLanguage(language)
    tts_service.setVolume(1.0)
    tts_service.setParameter("speed", speed)
    tts_service.say(strsay)
    time.sleep(1)
```

The following is a concrete example of path planning, we can go to the RA section by selecting the Find a Room button on the tablet.

Figure 17: Select pathfinding mode

The floor plan of DIAG is displayed on the tablet and we can choose which room we want to go to.

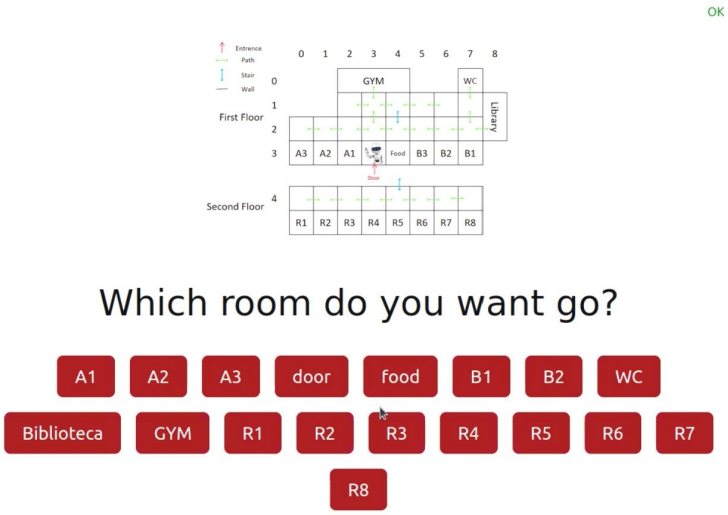

Figure 18: button files

After we choose the location we want to go, DIAGo tells us the path on the tablet and shows it through the map. It is also said to remind the students.

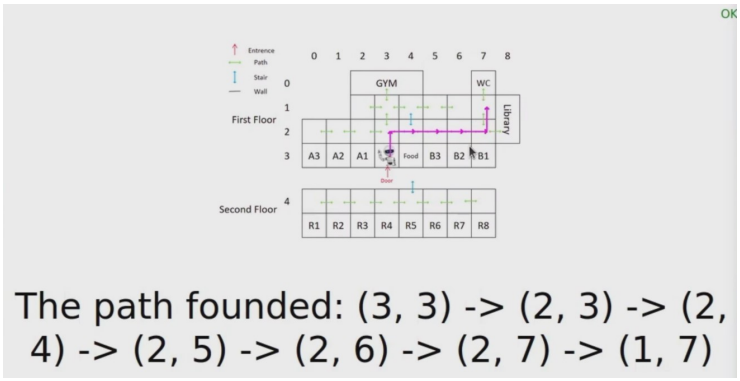

Figure 19: button files

5 Result

In this section, we mention the demo video we implemented, available at Google Drive. In it we demonstrate the different features of DIAGo, guiding the viewer through a comprehensive understanding of DIAGo's capabilities.

6 Conclusions

In recent years, we must admit that with the rapid development of robotics, robots are increasingly used in all aspects of education, security, and healthcare, and robots are increasingly capable of performing some difficult tasks. More and more people are proposing an architecture that relies on several modules belonging to different areas of artificial intelligence. This project aims to build a human-robot interaction and reasoning agent that will allow our pepper robot assistant, DIAGo, to interact with students. This project is one of the most different and interesting attempts we have made because the purpose of this project is not to demonstrate the technology of a single module, but to integrate robotics, machine vision, artificial intelligence, etc., not only considering the function of the robot but also the human factor, to build an effective and sophisticated overall system. There were many difficulties in designing this project, especially the interaction between humans and robots, how to find a balance between practicality and ethics, and we believe that only by considering robots as part of us, not as a bunch of cold parts, can we maximize the capabilities of robots and humans to meet the challenges of the world. There is a lot of room for expansion in this project, for example, we can enrich the interactive features of the robot and offer more detailed information such as query information. For the Reasoning Agent part, we can also combine machine learning technology to make our DIAGo more humanized and intelligent, so that students can have a better interactive experience.

References

[1] Tony Belpaeme, James Kennedy, Aditi Ramachandran, Brian Scassellati, and Fumihide Tanaka. Social robots for education: A review. *Science robotics*, 3(21):eaat5954, 2018.

[2] Michael John Paton. Why international students are at greater risk of failure. *International Journal of Diversity in Organizations, Communities & Nations*, 6(6):101–112, 2007.

[3] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Training a robot via human feedback: A case study. In *International Conference on Social Robotics*, pages 460–470. Springer, 2013.

[4] Hyunjin Kim. Trustworthiness of unmanned automated subway services and its effects on passengers' anxiety and fear. *Transportation research part F: traffic psychology and behaviour*, 65:158–175, 2019.

[5] Richard C Schmidt and Beth O'Brien. Evaluating the dynamics of unintended interpersonal coordination. *Ecological Psychology*, 9(3):189–206, 1997.

[6] SJ Russell, P Norvig, and JF Canny. Dd edwards, artificial intelligence: a modern approach, 1995.

[7] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.