UNIVERSITY OF SALERNO

DEPARTMENT OF INFORMATION AND ELECTRICAL ENGINEERING AND APPLIED MATHEMATICS

Master's degree in Artificial Intelligence and Intelligent Robotics

Cognitive Robotics Project:

Guardian of Shopping Mall Bot
Group 3

Lecturer:
Prof. Saggese Alessia

Students:
Mariniello Alessandro
Pepe Paolo
Rabasca Dario
Vicidomini Luigi

Contents

1 WP1 & WP2 - ROS Based Architecture

1.1 Detailed Architecture Schema

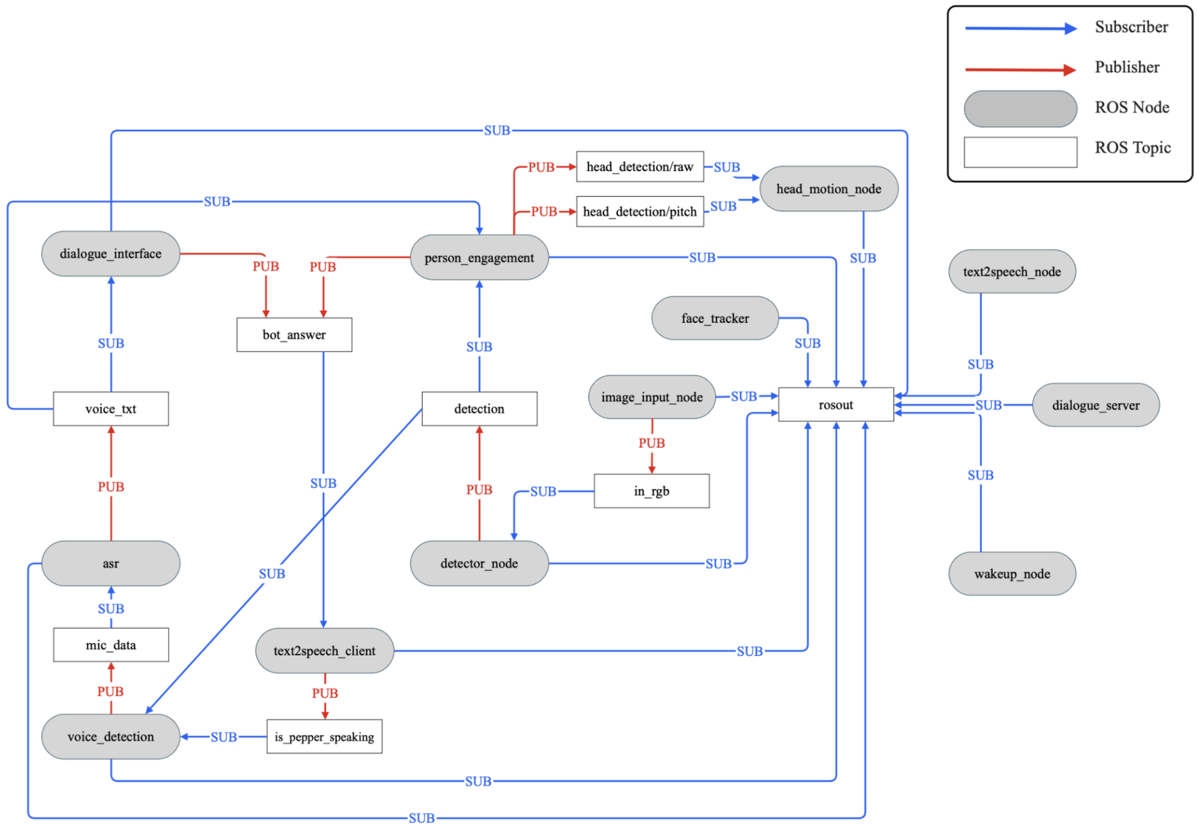


Figure 1: Proposed ROS architecture

1.2 Description of communication/roles and explanation of the implemented design choice

Here a description of the nodes, internal to our architecture.

1.2.1 Audio nodes

- **voice_detection**: uses the microphone to acquire the audio track of the user's speech. To prevent robot's voice detection and speaking sessions from overlapping, a "synchronization" procedure has been implemented, according to the publisher-subscriber methodology (on the topic "is_pepper_speaking"). In particular, whenever the robot has to start talking, the listening is paused and then resumed when the speaking session ends. In order to prevent the case where the robot is always listening, it was decided to allow the user to interact with Pepper only when the latter can see it, so this node also subscribes to the topic "detection" in which a Boolean value representing the presence of a person in the scene or absence is posted. When a person is detected, the node activates the microphone, in the other case it deactivates it. Everything that Pepper listens to is published on the "mic_data" topic.

- **asr**: This node receives the audio stream (via the "mic_data" topic) from the voice_detection node and uses the Google Speech Recognition API to synthesize audio tracks into text. If the recognizer recognizes in the received audio track a text, it publishes on the topic "voice_txt".

1.2.2 Rasa nodes

- **dialogue interface**: the node that manages people detection operations and chatbot use with Rasa. This node communicates with Rasa through the Dialogue.srv service. This service establishes synchronous communications between Ros and Rasa. This node waits for a message from the asr node on the "voice_txt" topic. Subsequently, it forwards the request to the Rasa server and publishes the received response on the "bot_answer" topic.

- **dialogue_server**: The node provides an intermediary service that forwards user requests to a Rasa server, subsequently receiving and handling the corresponding responses.

1.2.3 Engagement nodes

- **detector_node**: the node that identifies people; it makes use of a neural model used to accomplish the recognition. It uses the efficientdet_d1_coco17_tpu-32 module. This node receives images from the image_input_node on the "in_rgb" topic and performs person detection on them. If it detects a person for more than 5 frames (in our case 5 seconds, given that the image_input_node publishes at a rate of 1 frame per second (fps)), it publishes the value "True" on the "detection" topic; otherwise, if 10 consecutive frames pass without detecting any person, it publishes the value "False" on the topic.

- **person_engagement_node**: The node is responsible for managing Pepper's engagement logic. Specifically, when it receives a value on the "detection" topic, it triggers a callback function. Within this function, if a person is detected, it activates the microphone to allow Pepper to listen to the user. If the user starts speaking within five seconds, he is considered engaged; otherwise, Pepper initiates the conversation and completes the engagement. If an engaged person is no longer detected, the interaction is concluded with a farewell from Pepper, the microphone is turned off and Pepper's head comes back to its original position by publishing on the "head_detection/raw" and "head_detection/pitch" topic.

1.2.4 Pepper nodes

- **text2speech_node**: provides the text-to-speech (TTS) conversion service;

- **text2speech_client**: receives all messages from nodes and launch the text to speech;
- **image_input_node**: it uses the ALVideoDevice service in order to get the images from Pepper's camera and publishes the images on the "in_rgb" topic;
- **face_tracker**: enables Pepper to track the face of the person present in the scene;
- **head_motion_node**: a node that allows service calls for the movement of Pepper's head;
- **wakeup_node**: a node used to position Pepper in a standing position and return it to the rest position.

2 WP3 - RASA

2.1 NLU - Natural Language Understanding

For the NLU part, the project has the following structure: a training part of the model that consists of about eight hundred examples all written strictly "by hand". The part of writing, managing and improving the examples proved to be crucial for the operation of the chatbot. Indeed, in the course of the project's development, it was noticed that many of the errors the chatbot made were due to a lack of or incorrect generation of the sentences from which the model then trained.

2.1.1 Rules

In the **rules.yml** are present all the rules developed for the operation of the bot. Rules describe short pieces of conversations that should always follow the same path. Regarding the project specifically, the rules were used to let the model know when the form should start and to manage other behaviors. There are very simple rules that allow the robot to greet every time it is greeted, or that ask if the user concluded the search for a person wants to start another one by managing within the rules themselves the eventual response from the user. Regarding the rules for forms, when the "ask_with_no_info" intent is recognized, that is, when the user asks to find a person without providing any information about them there is a rule that automatically starts the form. On the other hand, if the "ask_with_info" intent is recognized i.e., when the user asks to find a person by also adding information about them, the chatbot will ask if the user wants to try to provide more information, if the answer is yes the form will start, otherwise all people with characteristics compatible with the initial request will be returned.In addition There is a rule that allows the form to be stopped when the stop_form intent is recognized i.e. when the user has run out of information about the person or when the user no longer wants to receive questions.When the form is active is present, and the user asks why all those questions (intent explain) the chatbot answers. Finally, there is the rule for activating and deactivating the form. When the form is deactivated, i.e., when all slots are filled, all people who match the information passed to the bot are returned. Then an action automatically starts to clean up the previously filled slots. As for the count part it is handled differently, as an implementation choice in fact the form is not triggered. This choice is motivated by the fact that it seemed more natural, from a conversational point of view, to directly return the number of people found after the count_people intent is recognized than to continue filling slots.

2.1.2 Domain

The **domain.yml** file it's the core of the NLU part , all the intents, entities and the actions that are implemented within the chatbot are present.

- **session_config**: Regarding the session information, the "session_expiration_time" parameter that defines the idle time in minutes after which a new session will start is set to 5, that is, after five minutes of idle time the previous session has expired. The carry_over_slot_to_new_session that determines whether existing set slots should be carried over to new sessions parameter is set to false.

- There is the define of a variable UNKNOWN, this is essential to handle slot filling when the user does not know a piece of information asked by the chatbot

2.1.3 Intents

There are standard Rasa intents such as: "greet", "goodbye", "affirm" and "deny" (to which several examples have been added) and other intents inserted for the developed chatbot.

- **ask_with_info**: It is the intent with the most examples. It covers the whole part of finding when information (entities) is already present in the sentence.
- **ask_with_no_info**: All examples involving searching for a person are specified, but when the user does not provide any information a priori.
- **count_people**: Is the intent in which all the examples regarding the "count" part developed in the bot.
- **doubt**: Is the intent defined to contain all examples concerning sentences in which the user makes explicit that he or she does not know the answer.
- **inform_info**: Contains all examples of user responses during the form.
- **affirm_with_entity** and **deny_with_entity**: They contain other useful examples for user responses during the bot, but they also present entities.
- **stop_form**: There are possible phrases that the user could use to stop the form.

2.1.4 Entities

- **gender**: That assumes male and female values.
- **hat**
- **bag**
- **color**
- **is_with**: This entity was inserted to handle everything related to entity negation, in fact it takes the value false if the entity with which it is grouped in the sentence is negated.
- **roi**: That assumes bar and market values.
- **aux_time_of_peristance**: This entity was introduced to classify the type of auxiliary that the user was adding to duration, which is needed later when queries need to be made to figure out which people to return.

- **shirt**: Represents all possible entities that represent clothes for the upper body.
- **pants**: Represents all possible entities that represent clothes for the lower body.
- **duration**: It represents the permanence time within one of the roi, and is extracted through the use of Duckling.
- **number**: It represents the number of times a person passes through one of the roi, and is extracted through the use of Duckling.

2.1.5 Form

It is used to collect info about the searched person, the slot to fill are:

- gender
- bag
- hat
- upper_color
- lower_color
- bar
- market
- time_of_persistence_bar
- time_of_persistence_market

2.1.6 Stories

The **stories.yml** file contains all the stories. It was decided to focus more on a "rule_based" chatbot as opposed to writing many stories, so as not to make the bot's conversations tied and a preset path. One of the stories implemented is "form interjection" that allows to avoid abnormal behavior during the form. If the form is active and one among count_people, aks_with_info or mood_greet intents is recognized, an action starts in which the robot explicates the fact that it must finish the form first. This story was implemented to avoid the fact that during the form the user could ask to search for another person or other anomalies that do not conform to the implementation choices. There are two other useful stories for when the slot is finished, in which case the chatbot asks if you want to start a new search through an action, to which the user can answer yes or no.

2.1.7 Configuration

The **config.yml** contains the configuration of the pipeline. The Rasa chatbot uses a pipeline of several natural language processing components to understand and generate text. The pipeline begins with a *Whitespace Tokenizer* and *Regular Expression Featurizer*, which break the text into individual words and extract features such as patterns in the text. Next, a *Lexical Syntactic Featurizer* and *Count Vectors Featurizer* are used to extract more features from the text, with the latter also analyzing character n-grams between 1 and 4. These extracted features are then passed to a deep learning-based classifier called the *DIETClassifier*, which is trained for 100 epochs. After this, the *DucklingEntityExtractor* is used to identify entities such as number and duration used for the roi and it also recognizes lettered numbers, and an entity synonym mapper is used to map synonyms of these entities. Finally, the *Fallback Classifier* is used to generate responses to the user. In addition to the NLU pipeline, the model also uses a set of policies including a *MemoizationPolicy*, *RulePolicy*, and *TEDPolicy* to generate responses.

2.1.8 Actions

Below are listed the actions that are used within the chatbot, that are present in the action directory:

- **submit_find**: The action involves querying a person search in the database, represented using a `.json` file, using the input requirements. For each person in the file, it checks if they meet the specified requirements. In the end, the chatbot responds with a sentence that can take one of three patterns:
  - Only one person found: The response will contain information about this person, specifically the places they have been.
  - More than one person found: The chatbot will display the number of people who meet the requirements and ask if further exploration is desired.
  - No person found: The chatbot responds by stating that it couldn't find any matching individuals.
- **submit_count**: The action is almost identical to the previous one, with the only difference being the response following the search, which can take the following forms:
  - Only one person found: The chatbot will respond by stating that it found only one person.
  - More than one person found: The chatbot will respond by specifying the number of individuals found who meet the requirements.
  - No person found: The chatbot will respond by stating that it couldn't find any individuals that meet the requirements.
- **correct_entities_values**: The action checks extracted values against the predefined values associated with the key `"values"` contained in the `entities.json`

file. This comparison is crucial for ensuring the accuracy of the extracted information. If the extracted value does not align with any of the correct values, a fuzzy matching function is invoked.
Fuzzy matching function compares all possible values of corresponding entities and inserts the best matching dictionary. Moreover, it also uses the `fuzzy.ratio` method to compare the entire word, beyond `fuzzy.extractOne`. It returns a `dict` with correct values.

- **suit**: The action checks for synonyms of *suit* and if they are found, the action sets `lower_color` and `upper_color` slots with the same values.
- **validate_form**: The action validates every input of the forms. If the user's intent is "*doubt*", it will set the slot with value `DOUBT`. Otherwise, it calls a function that checks if the value inserted belongs to the possible values of the current slot to validate.

3 WP4 - Detection and speech to text module

3.1 Detection module

Regarding the choice of the detection module, two different experiments were conducted within the project to determine which neural model performed best. Initially, an attempt was made to use the face detection module provided by the OpenCV library. However, during the testing of the ROS architecture with this configuration, a problem emerged: if a person approached or moved too far away from Pepper, the module lost sight of the person. Consequently, the detection_node published the "False" value on the "detection" topic, causing a crisis in the engagement node. On the other hand, using the efficientdet_d1_coco17_tpu-32 person detection module resulted in significantly better system performance. Even if the person moved away or, conversely, approached Pepper's camera more closely, the network still managed to detect the person from the bust (something that did not happen using a face detection module). Therefore, the design choice for the detection module leaned towards person detection, providing more robust and reliable results, especially in scenarios involving variable distances between Pepper and individuals. This decision enhances the overall functionality of the system, ensuring a more seamless engagement experience. One of the advantages of using the face detector is that engagement with the person occurs only when the person is close enough to be detected and is exclusively facing Pepper. The use of the object detector, however, has this drawback. Still, it was decided to use it because it proved to be more reliable. The face detector did not always detect the face in certain lighting conditions, and if the detected person moved, even slightly, their face would no longer be detected, resulting in the end of engagement.

3.2 Speech-to-text module

Regarding the speech-to-text module, similarly to the decision-making process regarding the detection module described in the previous paragraph, it was necessary to make a decision regarding the choice of voice recognition technology, choosing between OpenAI's Whisper and Google's speech recognition. After carefully evaluating the features, advantages, and disadvantages of both solutions, a thoughtful conclusion was reached. In the specific context of the application at hand, it became evident that utilizing Google's speech recognition service proves to be more advantageous and fitting. It was determined that, within the scope of the application, the use of Google's speech recognition service offers more benefits and is more suitable. The reasons are the following:

- Increased efficiency;
- Improved accessibility;
- Enhanced customer experience;
- Cost savings;
- Improved accuracy.

4 WP5 & WP6: Assessing Qualitative Examples and Quantitative Results

Regarding the testing of the entire system, various types of tests have been conducted, specifically targeting both Rasa and ROS.

4.1 Rasa

For testing rasa, both specific tests were written on the rules and form found in the test folder specifically in the test_stories.yml file, as well as tests evaluating the performance of the trained model. In the **test_stories.yml** file there are tests that have been written to test all the rules in the rules.yml,there is a test story that tests the form in its entirety and other test. The results of the implemented test stories can be found in the results folder in the failed_test_stories.yml file For tests concerning the model, the command **rasa test nlu –nlu data/nlu–cross-validation** was used. Cross-validation automatically creates multiple train/test splits and averages the results of evaluations on each train/test split. This means all data is evaluated during cross-validation, making cross-validation the most thorough way to automatically test the NLU model.The result of these tests can be found in the results folder.

The confusion matrix represents the accuracy of the DIET classifier in the recognition of entities during conversation with the user. As can be seen, the diet classifier errs in entity extraction a negligible number of times,this can also be observed in the histogram automatically generated by the tests.

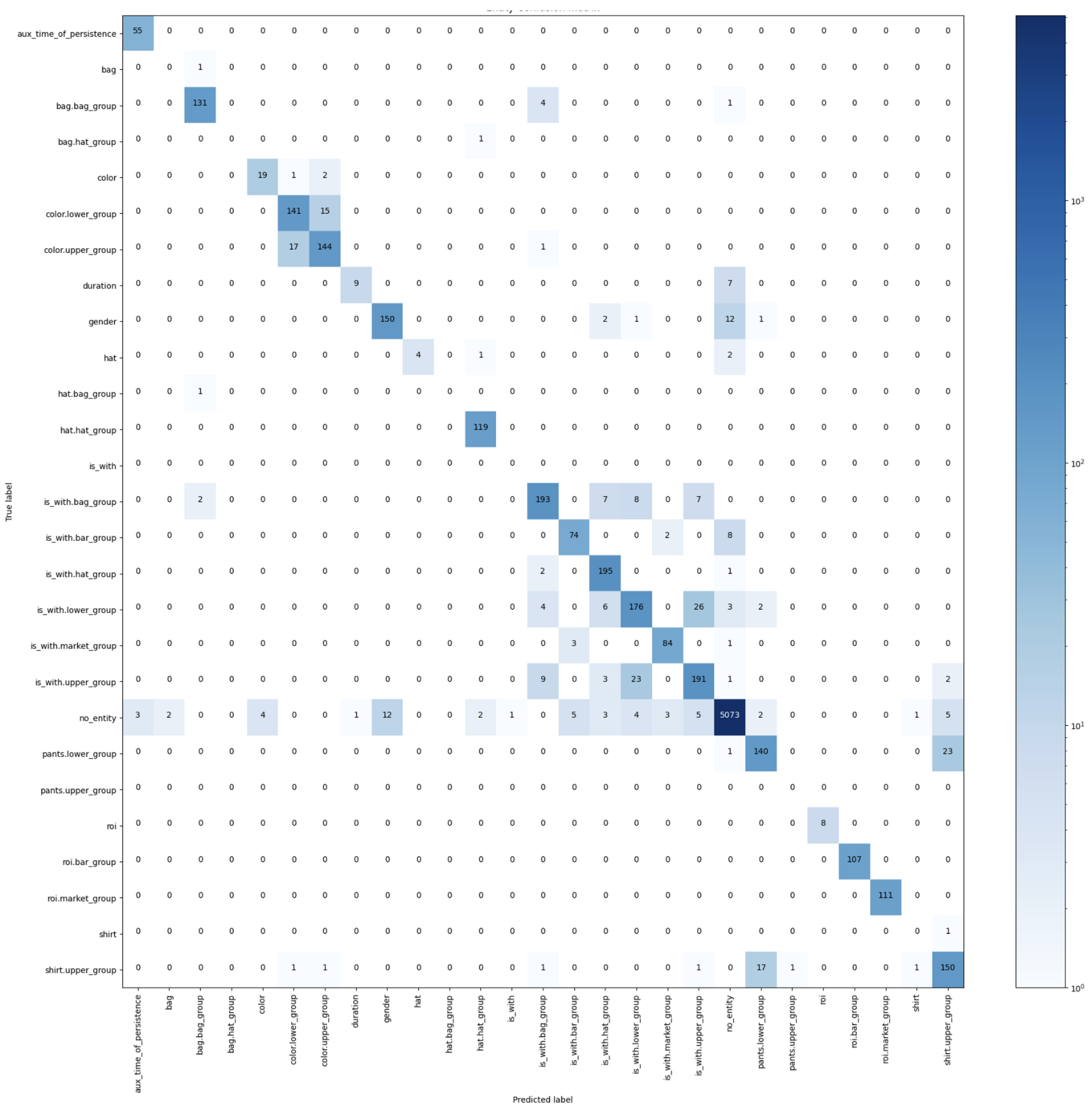


Figure 2: Entities confusion matrix

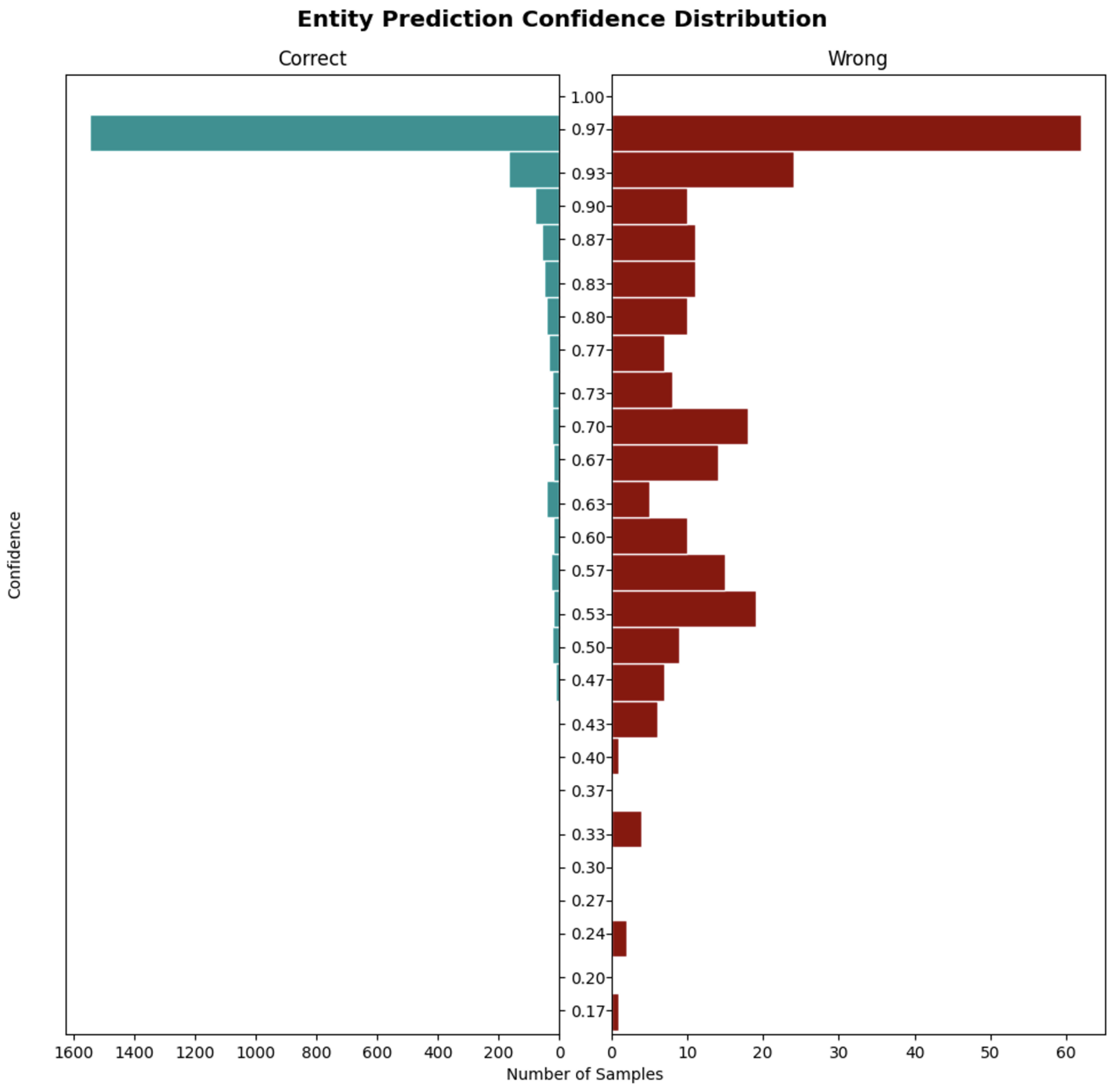
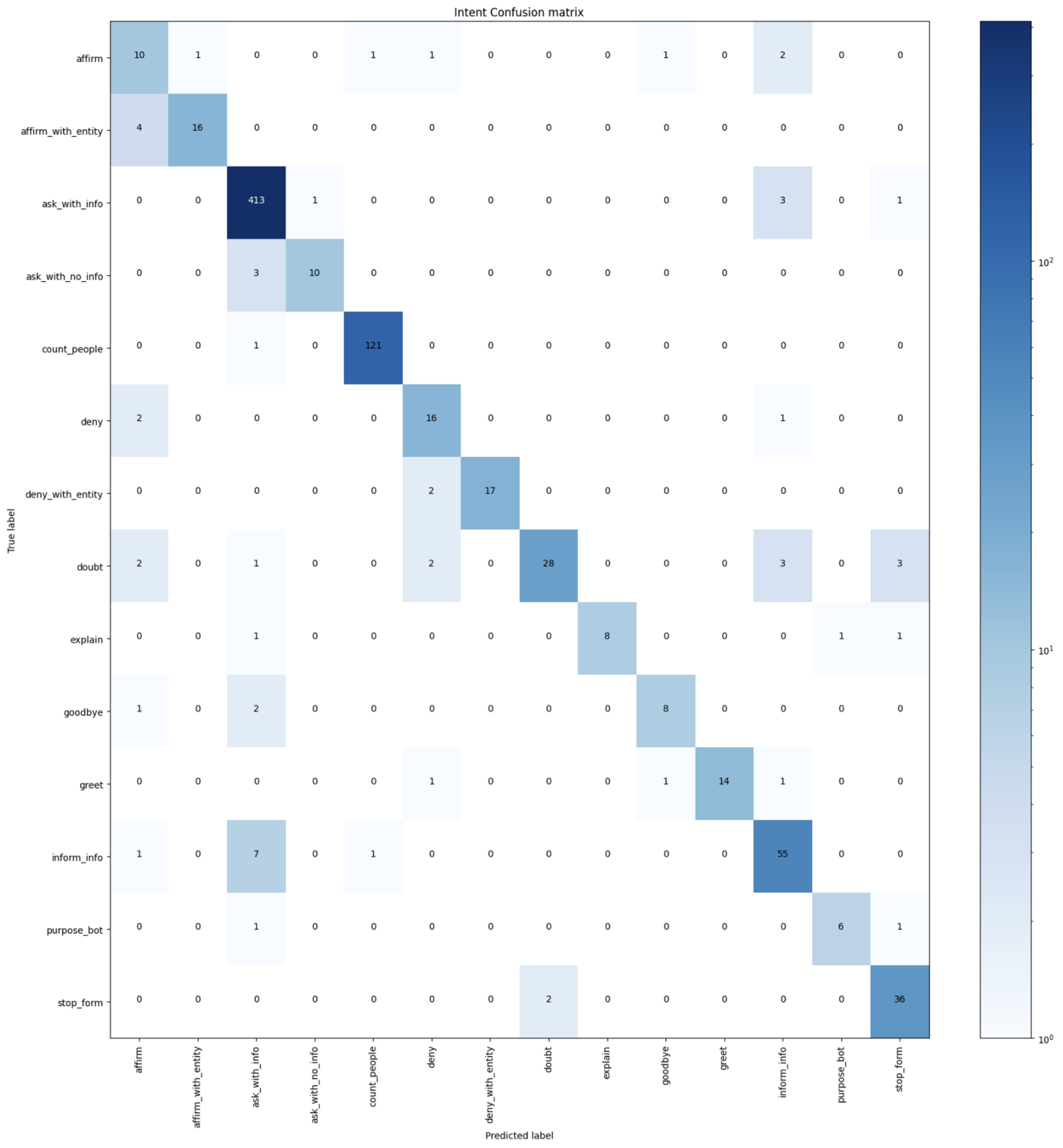


Figure 3: Entities confidence distribution

Similar discourse regarding the extraction of intents.



	affirm	affirm_with_entity	ask_with_info	ask_with_no_info	count_people	deny	deny_with_entity	doubt	explain	goodbye	greet	inform_info	purpose_bot	stop_form
affirm	10	1	0	0	1	1	0	0	0	1	0	2	0	0
affirm_with_entity	4	16	0	0	0	0	0	0	0	0	0	0	0	0
ask_with_info	0	0	413	1	0	0	0	0	0	0	0	3	0	1
ask_with_no_info	0	0	3	10	0	0	0	0	0	0	0	0	0	0
count_people	0	0	1	0	121	0	0	0	0	0	0	0	0	0
deny	2	0	0	0	0	16	0	0	0	0	0	1	0	0
deny_with_entity	0	0	0	0	0	2	17	0	0	0	0	0	0	0
doubt	2	0	1	0	0	2	0	28	0	0	0	3	0	3
explain	0	0	1	0	0	0	0	0	8	0	0	0	1	1
goodbye	1	0	2	0	0	0	0	0	0	8	0	0	0	0
greet	0	0	0	0	0	1	0	0	0	1	14	1	0	0
inform_info	1	0	7	0	1	0	0	0	0	0	0	55	0	0
purpose_bot	0	0	1	0	0	0	0	0	0	0	0	0	6	1
stop_form	0	0	0	0	0	0	0	2	0	0	0	0	0	36

Figure 4: Intents confusion matrix

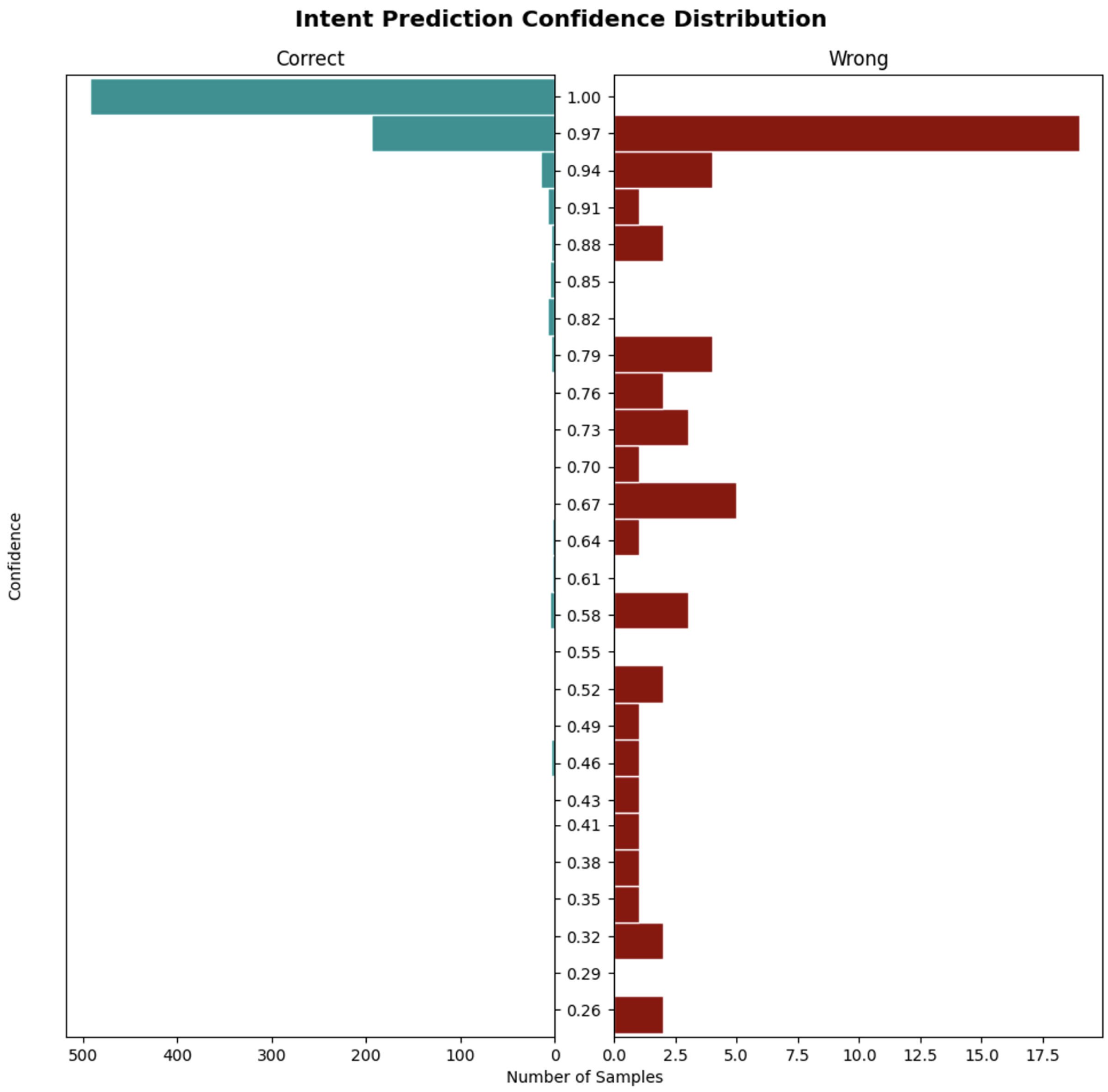


Figure 5: Intents confidence distribution

4.2 ROS

Each test is conducted within the following environment:

- The distance between the user and the robot is maintained at less than 2 meters.
- There are no ambient noises present during the conversation.
- Only one person at a time is permitted to engage in conversation with the robot.
- The conversation takes place in English.

Various live tests have been conducted to assess potential interactions. These include:

1. **Engagement Test**: To ensure that Pepper correctly engages with the person detected in the scene.
2. **Listening Test during Engagement**: To verify that Pepper listens exclusively when the person has been engaged.
3. **Listening Test without Engagement**: To ensure that Pepper does not listen when no person has been engaged.
4. **Listening Test while Pepper is Speaking**: To ensure that the microphone is deactivated while Pepper is uttering its phrase and to verify the microphone's reactivation when Pepper finishes speaking.
5. **Face Tracking Test**: To confirm that Pepper accurately tracks the face of the person in the scene.
6. **Head Position Reset Test**: To check if, when the person is no longer engaged, Pepper's head returns to its initial position.
7. **Engagement Start Test**: To ensure that, once engagement is initiated, Pepper activates the microphone for listening and to verify if, in the event the person does not interact with Pepper, it utters one of the possible "welcome" phrases.
8. **Engagement End Test**: To confirm if Pepper utters one of the possible "goodbye" phrases and deactivates the microphone when the engaged person is no longer present in the scene.

5 Conclusions and final considerations

Concluding:

From Rasa point of view, Pepper understands lots of questions and it responds according to rules. You can ask it to search a person of your interest and specify also what is wearing on top and underneath. It is capable of rejecting s ome words that don't belong to the domain of interest, but it does not always work well because it depends on what speech recognition has recognised and how it extracts the entities. Despite this, if you answer the questions it asks during the form, it helps you to find the person. Moreover, you can also stop the bot's questions, if you are no longer interested in searching for that person. The chatbot manages to understand complex phrases that contain more than two entities. If you wanted, you could ask a phrase with all entities.

From the ROS perspective, Pepper is capable of capturing user audio requests only if it has been identified, and in the absence of anyone in front of it, the microphone is correctly deactivated. The initiation of interaction and the subsequent start of the conversation between Pepper and the user pose no issues, as well as the restoration of the head position at the end of the interaction. However, there are some challenges related to synchronization among various ROS nodes, which slightly slow down the system. It's important to note that, in optimal environments, the speech-to-text module exhibits high performance and accurately translates spoken words into text. Nevertheless, in slightly noisier environments, the system starts to struggle, becoming confused and unable to capture the user's request clearly. This challenge may become particularly noticeable in contexts with background noise, indicating the need for further optimizations to ensure reliable translation even in acoustically complex environments.

6 Future developments

To make the robot more “human”, some improvements could be:

- Develop some robot gesture to better interact with the interlocutor
- Improve the synchronization mechanism among ROS nodes
- Implement a recognition module in such a way that, if a person exits the scene and then reappears, the conversation flow will not be interrupted. However, if a different person appears, a new conversation will be initiated.
- Enhance performance by integrating multiple example sentences into the nlu.yml file.
- Handle anomalous cases where the user makes requests that deviate from the chatbot’s normal control flow.
- Improve conversation flow so the interaction with Pepper is more natural

Develop a mechanism that resets all slots when switching from ask_with_info to count_people and vice versa. So the user can ask new questions in the same session- When the conversation flow is in the form and the user says something about other slots whether they have already been set or not, it keeps their value unchanged and the chatbot requests the current slot.

List of Figures