To-Do List with Pepper: Group 8

Faiella Ciro, Giannino Pio Roberto, Scovotto Luigi and Tortora Francesco

{c.faiella8, p.giannino, l.scovotto1, f.tortora21}@studenti.unisa.it

Jan, 2023

Department of Computer Engineering, Electrical Engineering and Applied Mathematics (DIEM), University of Salerno, Fisciano, Italy

1 WP1 & WP2 - ROS based architecture

1.1 Detailed architecture schema

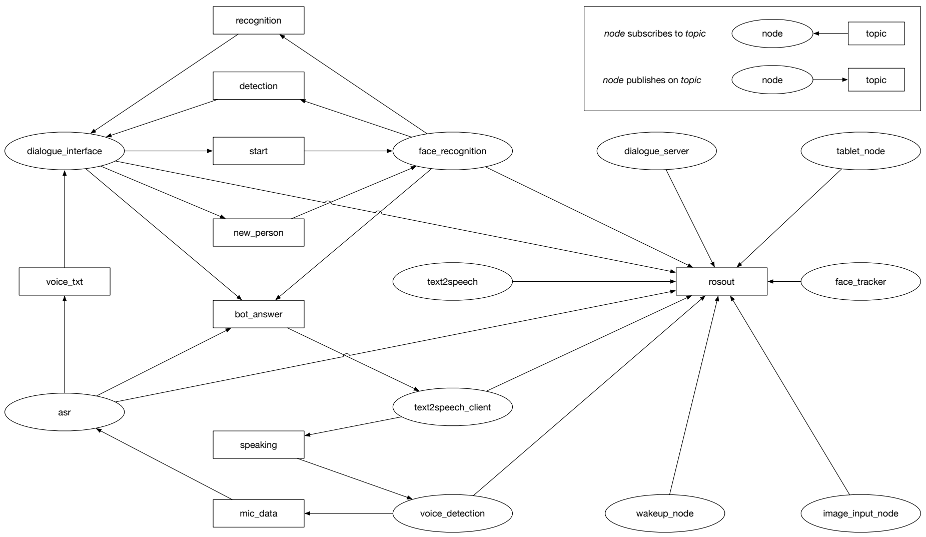

Figure 1: ROS architecture

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

Here a description of the main nodes, internal to our architecture:

- **dialogue_interface** is the main node that manages facial recognition operations and chatbot use with Rasa. This node communicates with Rasa through the *Dialogue.srv* service. This service establishes a synchronous communications between Ros and Rasa;
- **face_recognition** is the node that identifies people with face recognition; it implements a *neural model* used to accomplish the recognition. It uses the *vgg_face* module, with the *scipy cosine distance measurement* (more details at 3.1);
- **asr** use the Google Speech Recognition API [1] to synthesize audio tracks into text;
- **text2speech_client** receives all messages from nodes and launch the text to speech;
- **voice_detection** uses the microphone to acquire the audio track of the user's speech.

Within the figure 1, we can distinguish the communication, via topics, between the various nodes: the node sending the topic is the *Publisher*, while the one taking the topic information is the *Subscriber*. Several topics have been considered, due the fact that some operations must be performed before others, so a certain amount of synchronous communications were implemented. Specifically, the following topics were used for the communication between our nodes:

- **detection**: the face_recoginition node uses this topic to communicate the presence of a person in front of the camera, the dialogue_interface node receives the message;
- **recognition**: the face_recoginition node uses this topic to communicate the identity of the person in front of the camera, the dialogue_interface node receives the message;
- **start**: the dialogue_interface node uses this topic to tell face_recognition node when it is ready, this is used to start the face_recognition when the Rasa chatbot is already started, to avoid issues;
- **new_person**: the dialogue_interface node uses this topic to tell the face_recognition node the label of the new person in front of the robot;
- **bot_answer**: this topic is used by 3 publisher nodes (dialogue_interface, asr, and face_recognition) and one subscriber node (text2speech_client), the publisher nodes publish what they want to be said by the robot speaker, so in this way text2speech_client elaborates the request and processes it;
- **mic_data**: is used to receive from mic_data the audio track and send it to asr for processing;

- **speaking**: the speaking topic is used to handle the problem of self-listening (more details at 3.2), in fact text2speech_client communicates through this topic to voice_detection that it is speaking and then turns off the microphone.

2 WP3 - Dialogue management module

2.1 Explanation of the design choices within the NLU pipeline and DM module

The NLU pipeline was designed with one intent classifier and three entity extractor [2]. For the intent classification we use the DIET Classifier, trained with 100 epochs and *linear norm* as model confidence. We used three different entity extractor: Duckling, Spacy and CRF; the first one is used only for date and time extraction [3], Spacy for people's names [4] and CRF for the other entities [5]. Actually, Spacy can be used for more than just names, but we decided to use it in this way, with the following option: *dimensions: ["PERSON"]*. Three different extractors were chosen to be implemented, because each is specialized for the extraction of a particular type of data. DIET is built in such a way as to work simultaneously as both intent classifier and entity extractor, but was used only for intent classification. So, both DIET and CRF can be used for the same purpose, but, by comparing accuracy performance, we choose CRF for this Entity Extraction. The following is an extract of the *config.yml*:

```
  pipeline:
- name: SpacyNLP
  model: "en_core_web_md"
  case_sensitive: False
- name: SpacyTokenizer
- name: SpacyFeaturizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
  entity_recognition: False
  constrain_similarities: True
  model_confidence: linear_norm
- name: CRFEntityExtractor
- name: "DucklingEntityExtractor"
  url: "http://localhost:8000"
  dimensions: ["time"]
- name: "SpacyEntityExtractor"
```

```
    dimensions: ["PERSON"]
  - name: SklearnIntentClassifier
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
    constrain_similarities: True
    model_confidence: linear_norm
  - name: FallbackClassifier
    threshold: 0.2
    ambiguity_threshold: 0.1
```

Together with the pipeline, we used some policies [6]: the Memoization-Policy, which remembers the stories from the training data and checks if the current conversation matches the stories in the *stories.yml* file. TEDPolicy is used for next action prediction and entity recognition; it is trained with 100 epochs and the history parameter, which controls how much dialogue history the model looks at to decide which action to take next, was set to 5, so that the model has a sufficient number of previous dialogues to create a correct prediction. RulePolicy handles conversation parts that follow a fixed behavior by making predictions based on any rules defined in *rules.yml* file. Below is an extract from the *config.yml* file showing the policy definitions.

```
policies:
- name: MemoizationPolicy
- name: TEDPolicy
  max_history: 5
  epochs: 100
  constrain_similarities: True
  model_confidence: linear_norm
- name: RulePolicy
```

Rasa automatically generates some intents, e.g. greet or deny, which we decided to keep within the project; that are used to make the conversation more social with the user. Besides these, some custom intents required for the todo list have been implemented (e.g. change the user, add/modify/delete a task, etc.).

The intents we have used are as follows:

```
intents:
- greet
- goodbye
- affirm
- deny
- mood_great
- mood_unhappy
- bot_challenge
- thankyou
- inform_task
- inform_category
- manage_task
- inform_action
- view_task
- inform_person
- change_person
- delete_all
- who_am_i
```

Entities are structured pieces of information inside a user message; through these, we defined what our todo list needed. In particular, the information we needed is as follows:

```
entities:
- task
- category
- time
- purpose
- PERSON
```

2.2 Integration details

Manage different users We managed multiple users by storing their data in a database, so that we could keep track of each user's information. By connecting to the database, we could restart from the data of each past user. The chosen DBMS is SQLite3 [7]. Database storage seemed to us to be the fastest and most optimized way to save data and retrieve them. To manage multiple users, we defined intents and an entity. In particular, the entity *PERSON* represents the user, meanwhile, the intent *inform_person* served to name the user, while the intent *change_person* was useful for changing the user during the session. Moreover, the face re-identification can identify several users by training itself on dataset containing all the known people (i.e. the people that it already interacted with).

Insert a task When the user wants to insert a task, after the authentication phase is completed, he must specify the name of the activity and the category in which it must be inserted. The user can add a time too, in any kind of forms he wants (e.g. tomorrow at 8 am, today at half past nine, July 15th at midnight, etc.), thanks to Duckling extractor. When the time is entered, the bot asks for a reminder. If user chooses to use it then, when the time is up, the user can see that his task has expired.

Manage tasks After a task is inserted, the user can manage it e.g. deleting or updating. When the user wants to send removal request, he had to specify which task and then confirm the elimination of those task from the TODO list. Instead, when the user wants to modify a task, so doing a sort of update on it, he have to specify which task and then the user must re-insert the task with category, and time if needed.

3 WP4 - Face re-identification module

3.1 Explanation of the implemented design choice

For the people's identification we decided to use the face re-identification technique. We use as network the ResNet50 trained with VGGFace2 dataset [8], provided by *keras_vggface* [9]. When a user shows up, his face is taken as input in the ResNet50 (this network hasn't the decisional layer) and is taken as output the features of a given face. Then we compare the embedding of the features, extracted from the user face, with the embedding of each face that the network already know. Then is calculated cosine distance, because this distance metric is mainly used to find the amount of similarity between two data points. If a match is found, the identification conclude, otherwise, new training data were taken (shooting photos) and then the network is retrained generating a new association between the faces and the embedding spaces connected to them. To see the process by which the photo are taken, you can see the section test of "Unknown person enters the scene and says their name"(4.1) and the video (the YouTube link is at the end of the report). The face re-identification module starts working after the chatbot is started, to avoid any kind of problem that could occur in this phase.

3.2 Integration details

Robot voice The chatbot's responses are given 'by voice' by Pepper, using the speakers it is equipped with. To do this we used the service made available by Aldebaran, the node used is *ALTextToSpeech* [10]. The main problem due to the use of this system, is that while Pepper was talking, he could hear himself, so we handled it in this way: since the audio module always listen the background, after a user finished speaking, the microphone keep listen so, when Pepper say something, it has to be deactivated. Exactly for this reason, a topic (i.e. speaking) was designed to synchronize the activation and deactivation of the microphone. The text2speech node publishes on topic *speaking* whenever it finishes speaking; the voice_detection node has a callback related to subscribing of the topic *speaking*, the callback is nothing more than the start of listening; in this way, listening occurs only if the robot is not speaking.

Speech to text Very important to the realization of the whole system is the speech-to-text module, in order to have the user's voice synthesized into a string for the chatbot to use. *SpeechRecognition* library is used, which is based on the use of the Google Speech Recognition API [1].

Face tracking We implemented a face tracking module for both to make sure the user's face is always framed and to ensure that the robot is socially accepted. The example of the official NAOqi documentation was followed and adapted to our needs [11].

Web interface A Web site, in PHP, was built with a connection to the same DB used for task management (2.2), so that the user can view tasks and reminders on the tablet. Apache is used as the web server, launched on the same Linux computer on which Pepper's processes run. The ALTabletService was used to upload the URL to the tablet [12].

4 WP5 & WP6 - Test

All tests were carried out with Pepper during the various tests, on the various days of use in the laboratory. In contrast, the tests that have been delivered in the following report are carried out with Choreographe, due to circumstances beyond our control on the day when we had determined to film the results obtained. The test video with Pepper will be delivered the following day.

4.1 Qualitative evaluation planning and implementation

Each test is executed in the following environment:

- The distance between the user and the robot is less than 2 meters;
- There are no ambient noises during the conversation;
- One person at time can talk to the robot;
- The conversation happens in english.

Unknown person enters the scene and says their name There is initially an empty scenario with no people in it; a person comes in front of the robot, the robot asks who he/she is, since is an unknown person, the person says their name, the system starts the facial image acquisition phase, once the test is over the user should be recognizable by the system.

Known person is recognized The second scenario sees Pepper interacting with a known user, with the same initial condition of the first scenario: the social robot in this case has to recognize the user and initiate a conversation, calling the user by his name. As soon as the user is recognized, Pepper shows on its tablet the list of already submitted tasks related to the user.

Inserting a task The user has already been recognized and wants to enter a task, so he asks the system to store the task "send an email to prof" at the time "11.30" on "17-1-2023" in the "work" category, Pepper asks if he wants to enter a reminder related to this task, the user gives affirmative answer and Pepper enters the task and reminder into the database.

Editing a task The user wants to edit a task: first, Pepper has to find out which task and whether it is actually in the user's list; if the task is present, Pepper asks the user to re-enter the updated data, otherwise it says it has not found the task.

View to-do list The user has already been recognized and wishes to listen to the to-do list, Pepper responds by listing all the tasks. The user can already see the list on Pepper's tablet, because the list appears on the tablet since the recognition of the user (if the user is new, the tablet shows just a welcome message).

Delete a task Here the user wants to delete a single task in his to-do list, so asks Pepper to do it. Firstly, the robot search the task in the user to-do list and, if it find a match, it proceed to remove that task; otherwise, it answer back to the user that the task in not found.

Delete all tasks In this case, the user wants to delete all tasks already inserted in his to-do list: Pepper's response is to remove all the user tasks.

4.2 Quantitative evaluation planning and implementation

The system, in its entirety, has minor problems that may sometimes make the conversation not flow properly. In general, the problems are not that serious, especially if the environment used is ideal. Some problems relate to the recognition, by the Google Speech Recognition API, of the words spoken by the user. The facial recognition system is practically perfect under ideal conditions; if the video is subject to a lot of light or the user is too far away, facial recognition may perform poorly. The tablet responds well to input: it quickly shows the task list for the user and, when there are no people in front of it, it shows the list of known users. We could have used animations, or gestures, that would have made Pepper seem more welcoming and would have made the conversation more realistic[13]. However, this was not possible because these animations caused small movements of Pepper's head, which consequently interfered with face recognition. More generally, these animations also made the robot slower in responding. Improvements could be made regarding the update operation of a task: this is the most difficult operation as it requires many steps and the system we have defined only allows us to update using specific words. The operation is simple and intuitive when done via chat, i.e. by giving keyboard input; the difficulty arises when using the speech synthesizer, which is inclined to error.

Considerations on the tests:

- "Unknown person kenters the scene and says their name" test: the system in this case does not present problems; the only one we could imagine is when the Google Speech Recognition API does not understand properly the name of the user, but in the chosen environment it works well;
- "Known person is recognized" test: the system in this case does not present any problems, it is quite fast in recognising the person;
- "Inserting a task" test: here, the system works well, but there may be problems if the synthesiser recognises unspoken words by mistake; this leads to the entry of incorrect tasks;
- "Editing a task" test: this is the most challenging action because this operation is the longest; it is the less accurate in our system;
- "View to-do list" test: this operation is quite simple and the results are good;

- ”Delete a task” test: there are no problems with this operation;
- ”Delete all tasks” test: there are no problems with this operation.

In conclusion, here is the link to the video made on Choreographe:
https://youtu.be/7_Y4fi7ETKU
We also uploaded the video to the Group 8 Google Drive folder.

References

[1] *Speech to text with the Google Speech Recognition API.* URL: https://pypi.org/project/SpeechRecognition/.

[2] *NLU Model.* 2.x. URL: https://rasa.com/docs/rasa/2.x/tuning-your-model.

[3] *Duckiling Entity Extractor, for date and time extraction.* URL: https://rasa.com/docs/rasa/components/#ducklingentityextractor.

[4] *Spacy Entity Extractor, for person's names extraction.* URL: https://rasa.com/docs/rasa/components/#spacyentityextractor.

[5] *CRF Entity Extractor.* URL: https://rasa.com/docs/rasa/components/#crfentityextractor.

[6] *Policy.* 2.x. URL: https://rasa.com/docs/rasa/policies/#rule-policy.

[7] *SQLite.* URL: https://www.sqlite.org/index.html.

[8] Cao, Q., Shen, L., Xie, W., Parkhi, O.M., Zisserman, A. *VGGFace2: a dataset for recognising faces across pose and age.* 2018.

[9] Refik Can Malli. *VGGFace implementation with Keras Framework.* 2017. URL: https://github.com/rcmalli/keras-vggface.

[10] *ALTextToSpeech, text to speech pepper.* URL: http://doc.aldebaran.com/2-4/naoqi/audio/altexttospeech.html.

[11] *ALTracker, face tracking of NAOqi.* URL: http://doc.aldebaran.com/2-4/naoqi/trackers/index.html.

[12] *ALTabletService, to allow use the pepper tablet.* URL: http://doc.aldebaran.com/2-4/naoqi/core/altabletservice.html?highlight=altabletservice.

[13] *ALAnimationPlayer, animation of pepper.* URL: http://doc.aldebaran.com/2-4/naoqi/motion/alanimationplayer.html.