Guardian of the Mall: Improving Customers Experience in Shopping Malls with Pepper

Group 2: Giovanni Lamb[2168], **Palma Orlando**[0433], **Fabrizio Saturnino**[2328], and **Egidio Zottarelli**[2175]

[1]Dept. of Information Eng., Electrical Eng. and Applied Mathematics - University of Salerno, Italy

Abstract

This project introduces an AI-assisted robotic system, designed for shopping malls, with Pepper at its core acting as a guardian and interactive assistant. Pepper relies on a connected camera and video analytic system to detect people and recognize their attributes, such as gender and accessories. However, it does not process these data; instead, it accesses the information from a database. This allows Pepper to interact with visitors through spoken language, improving the customer experience by serving as an interactive interface to the sophisticated data processing system of the mall.

1 Introduction

In the evolving landscape of reactive technology, the fusion of advanced video analytics and robotic assistance has opened new frontiers for improving customer experience and operational efficiency. At the heart of this innovative ecosystem is **Pepper**, envisioned as a robotic sentinel for the mall. Pepper is not just a passive observer; it actively interacts with customers using natural language processing. However, it is important to note that Pepper does not directly process the recognition of pedestrian attributes. Instead, Pepper has access to a database that stores information processed by the video analytics system. This system is responsible for detecting people and identifying specific attributes, such as gender or the presence of accessories like bags or hats. Using the data acquired from the camera system database, Pepper can answer various queries related to the mall's environment. For example, it can provide real-time information about the number of people in the mall or identify people based on specific attributes such as gender or accessories.

2 ROS Based Architecture

In the following section, we look at the ROS architecture that underpins Pepper's effective operation. Specifically, the ROS architecture employs its two primary communication mechanisms, *topics* and *services*, to achieve this objective. This strategic use of ROS functionalities ensures that Pepper can efficiently process and respond to data and requests it encounters in the shopping mall environment.

2.1 Image Publisher Node

The **image_publisher_node** is developed to manage and publish video feeds from the camera, specifically designed to interface with Pepper. The node is adaptable to both the internal camera of Pepper and external camera sources, making it versatile in various operational settings. As designed, the node acts as a *publisher*. Its primary role is to publish the images captured by the camera on the `/image_feed` topic.

2.2 Face Detector Node

The **face_detection_node** serves as both a *publisher* and a *subscriber*. It is designed to detect faces in the image stream that is published on the `/image_feed` topic. To accomplish this, the node utilizes two important files from OpenCV's deep learning module:

- The `opencv_face_detector.pbtxt` file is a configuration file that outlines the structure of the neural network used for face detection. Details the layers and connections within the network.
- The `opencv_face_detector_uint8.pb` file, on the other hand, contains the pre-trained model with the weights of the neural network. This model is trained to identify faces within images.

As the node processes each image, it applies this neural network model to scan and detect faces. If a face is detected in an image, the note then calculates the bounding box for each detected face. Once these bounding boxes are determined, the node packages this information into a data structure known as Detection2DArray. This data structure is specifically designed to hold details of detected objects in a two-dimensional space, including their positions and sizes, which in this case are the positions and sizes of the bounding boxes around the faces. Finally, the node publishes this data on the `/face_detection` topic.

2.3 People Detector Node

The **people_detector_node** is similar to *face detector node*. It is designed to detect people in the image stream that is published on the `/image_feed` topic. The **MobileNetSSD** model is chosen for its efficiency and effectiveness in the real-time object detection detection task, making it suitable for scenarios where quick and accurate people detection is required. It strikes a good balance between speed and accuracy, making it well suited for robotic applications where real-time processing of camera input is essential. As the *face detector node*, once the bounding boxes are determined, the node packages this information into a data structure known as Detector2DArray and then published this data on the `/people_detection` topic.

2.4 Emotion Recognition Node

The **emotion_recognition_node** serves as both a *publisher* and *subscriber*. It is designed to receive images from the `/image_feed` topic and facial detections from the `/face_detection` topic. It utilizes a pre-trained deep learning model to analyze the detected faces and determine the predominant emotion among the seven predefined categories (surprise, fear, disgust, happiness, sadness, anger, and neutral). The acquired image is pre-processed, normalized, and passed to the neural network for classification. The analysis result, which includes the emotion label and a confidence score, is then published on the `/emotion_detection` topic in the form of an *EmotionImageArray* message.

2.5 Visualization node

The **visualization_node** serves as a *subscriber*. This node is connected with three topics: `image_feed`, `people_detection` and `emotion_detection`. This node has the aim of visualization the final result of these three topics in the final image.

2.6 Dialogue Server

The **dialogue_server** node is one of the main nodes for chatbot functionality. This node serves both as *publisher* and *subscriber*, in particular it publish on `/mic_data`, which is a topic for sending the data captured from the microphone, the next one is the subscriber topic `/ans_data` for reciving the converted audio to text file and send the question to the RASA chatbot with a json message, the response is get by the call of a service, and finally the result is publish on topic `/ans`.

2.7 Audio Speech Recognition Node

The **asr_node** is designed to perform speech recognition using the Google Speech Recognition API. The node gets data from a subscriber of the topic `/mic_data` and converts it into an AudioData object. The combination of the AudioData object and the favorite language is sent to the Google API and the result is published on topic `/ans_data`.

2.8 Dialogue Interface

The **dialogue_interface** is a node having the purpose of the visualization and the gesture of the response of the chatbot. The node has a subscription to a topic `/ans` which is used to get the data of the detected chatbot question. The **callback** function of the subscriber, extract the question of the person and the answer of RASA, print it in a terminal and call the text-to-speech and animation services.

2.9 Text to Speech Node

The **text2speech_node** is structured to configure the service *ALTextToSpeech* used by the **animation_node** through the *ALAnimatedSpeech*. In this node is possible to configure the language of Pepper with the following command:
`tts.setLanguage('English')` with `tts` as the service
The aim of this service is to get a request with a text message and the service provides to make Pepper say the text.

2.10 Animation Node

The **animation_node** is developed for perform the animation and the robot speech using the API service *ALAnimatedSpeech*. This service is able to gesture the **ALTextToSpeech** service of Pepper and reproduce animations based on the text that he elaborates and on the selected context. For this project, we selected a contextual behavior of his animations.

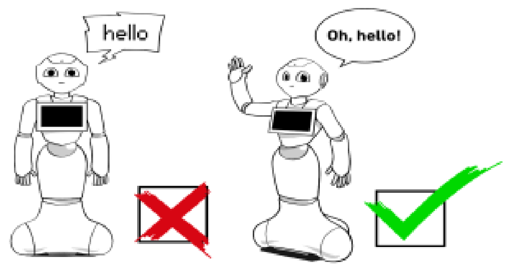

Figure 1: Depiction of Pepper's ideal behaviors implemented by us.

2.11 WakeUp Rest Node

The **wakeup_rest_node** is primarily designe to control Pepper's posture and motion. This node plays a crucial role in managing Pepper's behavior by utilizing the *ALMotion*, *ALRobotPosture* and *ALBasicAwareness* services provided by Pepper's software. The key functionalities of this node include:

- **WakeUp**: When the `wake_service` is called, the node uses the *ALMotion* service to wake up Pepper, turning on its motors.
- **Rest**: The node also provides a `rest_service` that allows Pepper to enter a resting state. When called, it uses the *ALMotion* service to put the robot in a resting position, conserving energy.

A key component of this node is its interaction using the *ALBasicAwareness* service. This service plays a crucial role in the robot's ability to track and respond to humans. When enabled, it allows the robot to track a person, enhancing its interactive capabilities.

3 Chatbot

At the heart of the project lies the **Chatbot**, which plays a pivotal role in facilitating human-Pepper interactions, and its functionality is of paramount importance. This means it should understand natural language, respond in a friendly and relatable manner. This approach enhances the overall user experience and makes interactions with Pepper more intuitive and enjoyable. As mentioned in the previous sections, Pepper incorporates a range of bodily animations, allowing for a more natural and less static conversation with the user. These animations add a layer of expressiveness to Pepper's interactions, making the robot's responses not only linguistically coherent but also visually engaging.

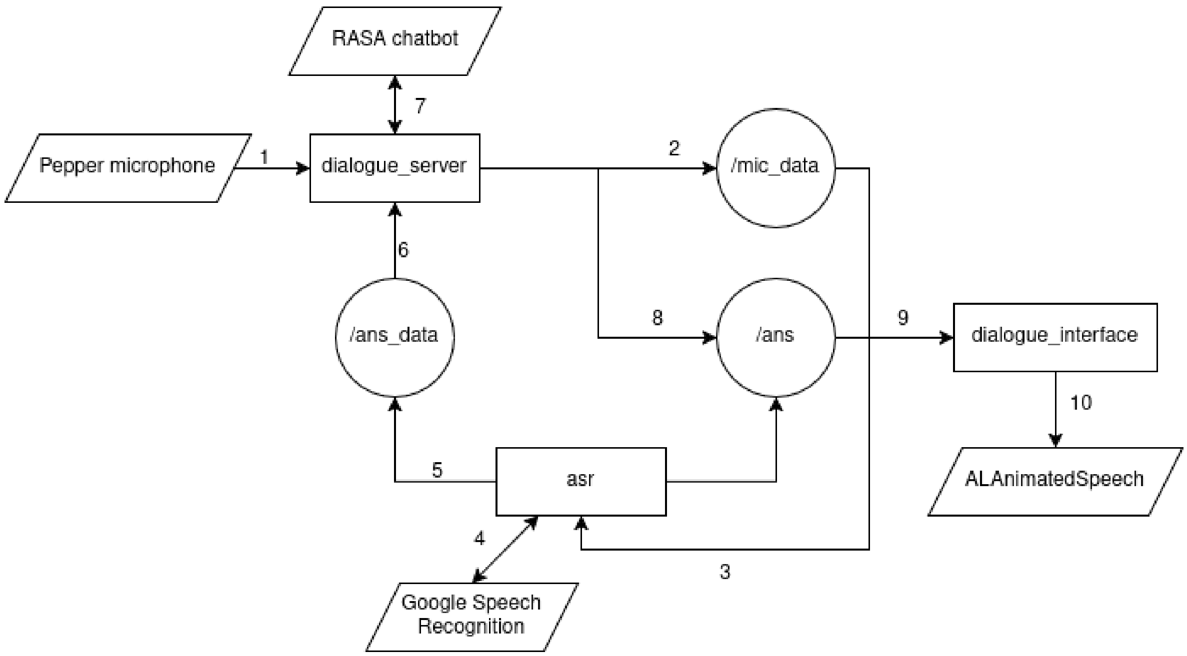

Figure 2: Scheme of the implemented chatbot

Based on the information gathered from the databases, Pepper possesses the capability to inquire about specific aspects related to a shopping mall enviroment and information on aspects related to the Artificial Vision contest. These inquires may encompass:

- **Count of Individuals in the Shopping Mall with Specific Attributes:** Pepper can determite the number of individuals in the shopping mall who meet a certain predefined criteria. These criteria can include attributes such as gender, the presence of a bag or of a hat. In addition to these attributes, Pepper also has the capacity to asses the number of passages trough four lines monitored by the camera system. In our case study, we consider the **Alpha line**, **Beta line**, **Gamma line**, **Delta line** as these lines.

- **Search for an Individual Meeting Specific Attributes**: In addition to counting, Pepper can also perform a search to locate a person within the shopping mall who matches specific attributes, as described above.

- **Count of groups that participated in the contest**: Pepper can determine the number of groups that took part in the contest, also doing a filtering on the number of groups that scored at least or less than a given score

- **Position in the ranking**: It can return who (group ID and members) achieved a given position (e.g. "who won the contest?").

- **Members of a group**: Can return the members of a given group or fellow members of a given member who participated in the contest.

- **Has participated**: Given a request about participation in the contest by a group or person, answer affirmatively or negatively.
- **Group number**: Pepper can provide the ID of the group to which a given member belongs or the ID of the group that reached a given position.
- **Score done**: It can return the score obtained by a given group or a given member of a group that took part in the competition.

The implementation of the operations for counting and searching people in the customer tracking system is facilitated by the `TrackingPeople` class. One of the key features of this class is its ability to update the internal data state based on input entities. The `update` method plays a pivotal role in this regard. It processes a list of entities, evaluating each one to determine its relevance to gender or line-related criteria. It effectively updates its field by leveraging language-specific characteristics in English, where negations precede nouns. This ensures accurate updates for gender data.

Box 1. Count People Task
This scenario illustrates the practical use of the chatbot in a `count_people` task. To begin, the chatbot initiates the conversation with a friendly greeting. Subsequently, the user engages with the chatbot by requesting information about the current number of people in the shopping mall. Upon completing the `count_people` task, the chatbot consistently follows a structured approach. It first inquires with the user whether they would like to either initiate a new search based on the data previously collected or proceed with a different task.

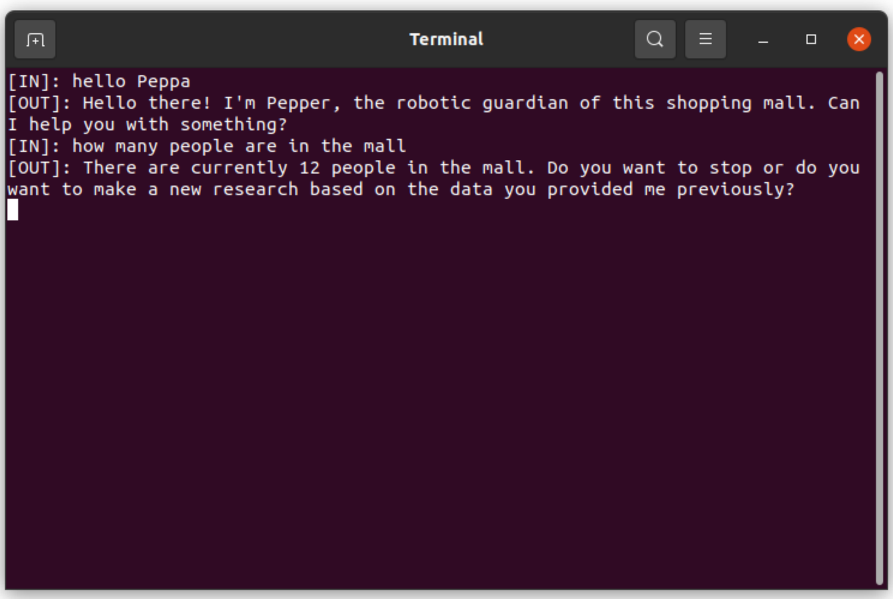

Figure 3: Example of the chatbot when we ask Pepper to count people in the mall

We can see that the chatbot functions correctly in this context. After recognizing the user's intent to conclude the interaction (the `stop` intent), it takes the appropriate action by resetting any slots that may have been filled during the previous requests. This behavior ensures a smooth transition between tasks and a user-friendly experience, as the chatbot is responsive to the user's cues and maintains a coherent conversation flow.

Box 2. Finding Person Task

This scenario illustrates the pratical use of the chatbot in a `finding_person` task. To achieve this goal, the chatbot requests the user to specify the values of several slots corresponding to attributes that Pepper acquires from the database, namely `gender`, `bag` and `hat`. To fill these slots, we employ a dynamic form that prompts the user to complete any remaining empty slots. For the case in the figure, the user asks Pepper for assistance in finding his daughter. The chatbot deduces from the word 'daughter' that the user is searching for a female individual, after Pepper asks if she is wearing a bag and if she's wearing an hat.

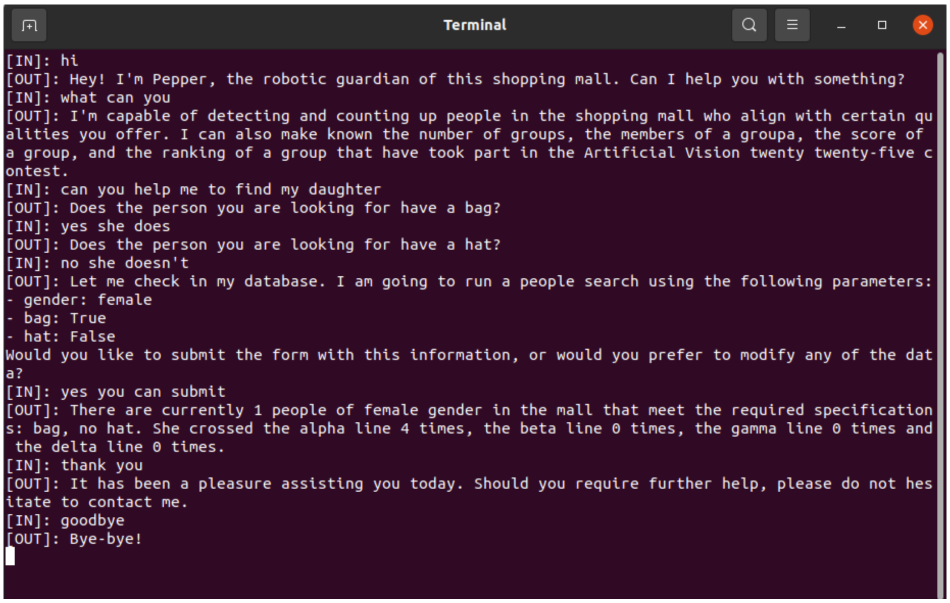

Figure 4: Example of the chatbot when we ask Pepper that we want to find a person

Once all the slots are filled, the chatbot performs a filtering operation to determine the subset of individuals who match the user's specified attributes by using the methods of the `TrackingPeople` class. In cases where the search yields no results, Pepper takes the initiative to call the human security personnel at the shopping mall to provide reassurance to the user.

Box 3. Contest Asking Task

This scenario illustrates the pratical use of the chatbot in a `asking_contest` task. For start we ask Pepper the winning group of the contest, and after a research on his database, he provides the correct reply. Subsequntly, the user engages with the chatbot by asking how many groups participated in the contest and he answers correctly. After that the user asks if group 11 took part in the contest and the bot correctly answers no, also providing information about the IDs of the groups that participated in the contest. Next we ask if group 10 participated in the contest, and receiving an affirmative answer from the bot, we then move on to ask who are the members, what score they made in the contest, and what position they achieved in the leaderboard (all questions pertaining to group 10, the last data provided by the user).

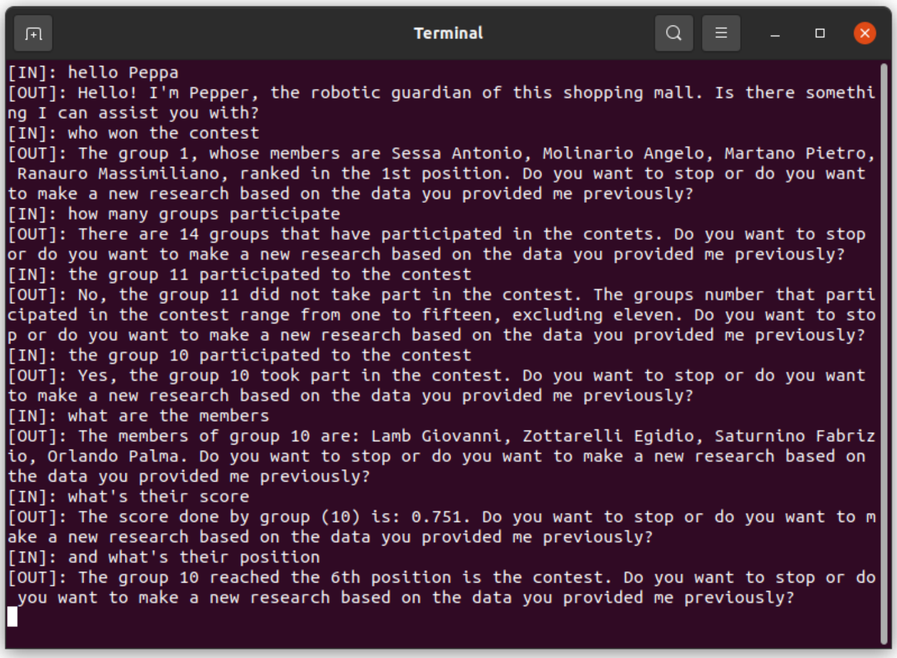

Figure 5: Example of the chatbot when we ask Pepper some informations about the contest

To ensure that the chatbot effectively interprets and responds to user inputs, a comprehensive and well-structured setup of *intents, slots, lookup tables, synonyms, stories, rules* as wellas a carefully chosen pipeline and policies is essential.

3.1 Domain

Each **intent** is integral to fostering an intuitive, user-focused conversational experience, adeptly catering to the diverse needs and expressions of users. Specifically:

- `affirm`: Detects affirmative responses from the user.
- `capabilities`: Helps the chatbot understand when a user is inquiring about its functionalities or the range of tasks it can perform.
- `count_people`: Activated when the user wants the chatbot to count or identify people, typically within a given context or set of parameters.
- `deny`: Opposite of `affirm`, this intent captures negative responses from the user.
- `doubt`: This intent is for situations where the user may be anxious and unable to recall certain attributes. In such cases, the chatbot always responds with reassuring phrases, assuring the user that it will do its utmost to locate the missing person.
- `finding_someone`: Triggered when the user's objective is to locate or identify a specific person.
- `goodbye`: Recognize when the user intends to end the conversation.
- `greet`: Used to detect greetings from the user.
- `inform`: This intent is crucial for scenarios where the user provides new information or updates previously provided information.
- `nlu_fallback`: Handles situations where the chatbot is unable to understand the user's input.
- `stop`: This intent is used when the user wants to halt a current process or interaction.
- `thanks`: Used to detect user gratitude.
- `count_groups`: Activated when the user wants the chatbot to count the groups that took part in the contest.
- `classification`: Activated when user is asked for ranking information.
- `have_partecipated`: Handles the situation when the user asks if a group or person has participated in the contest.
- `members_group`: Used when asking about members of a group or a person's teammates.
- `score_done`: Intent used when asking about a score achieved.
- `group_id`: Intent used when inquiring about the group number.

Entities are predefined categories of information that the chatbot can recognize and extract from user input. The entities in this file include:

- `clothing`: Used to identify mentions of clothing items in user input.
- `gender`: Identifies gender references.
- `negations`: Detects negations words of phrases, important for understanding the context of responses.
- `passages`: Used for identifying the number of passages or movements across a line.
- `line`: Detect lines of interest
- `mark`: Used to identify the score (number)
- `member_group`: Identifies the reference to a person
- `group_ID`: Used to identify the group ID

- `position`: Used to detect ranking position

In this project, the **slots** are strategically utilized to serve some primary functions: facilitating the form for searching a person, enabling multiple counting operations while retaining previously set parameters and allow sequential questions on the contest while retaining the parameter provided in the first question.

3.2 Training Data

For each intent defined in `nlu.yml` file, there are corresponding examples of user utterances. To enhance the model's accuracy in identifying intents and entities, we have incorporated lookup tables into our training data. Specifically, we created a **lookup table** for `gender`, which includes nouns that directly indicate a person's gender, such as *aunt, boys*, etc. Another table is for `clothing`, listing various garments to help deduce whether an item is a hat or a bar. The `negation` table contains negation terms, crucial for detecting negation in user utterances, such as *don't, not, no.* The table for `passagges` which consist in frequency expressions, such as *once, twice, twenty-five times*, etc. Another table is for `position`, listing various ordinal numbers for the ranking position, such as *first, second, third*, etc. .The table `group_ID` wich listing various cardinal numbers for the ID of the groups, such as *one, two, ten*, etc. The table `member_group` contains names and surnames of people, such as *Zottarelli Egidio, Lamb Giovanni, Saturnino Fabrizio, Oralndo Palma*, etc. Additionally the table `mark` listings a series of floating-point numbers for the score, such as *zero point five, zero point seven*, etc.
Synonyms play a significant role in enhancing the entity recognition process. By using synonyms, we group a set of related terms for the lookup table `gender`, for istance, terms like *woman, girls, mum*, etc., are grouped under the synonym `female`. To train the dialogue management model in our Rasa project, we employ a strategic blend of rules and stories. **Rules** are implemented to govern specific segments of conversations that are expected to follow a consistent and predefined path. On the other hand, **stories** are essentially examples of real conversations that guide the model in learning how to handle different conversation flows, including branching paths and varying user intents. To avoid the common challenge of omitting important events in stories, we leverage interactive learning. This approach, initiated via the `rasa interactive` command, allows us to directly engage with the bot, simulation real conversations and providing immediate feedback to refine the story writing process.

```yaml
- rule: Deactivate the form if intent is 'count_people'
  condition:
    - active_loop: find_person_form
  steps:
    - action: find_person_form
    - intent: count_people
    - action: action_reset
    - action: action_deactivate_loop
    - active_loop: null
    - slot_was_set:
      - requested_slot: null
    - action: action_count_people
    - action: utter_ask_stopping
```

Figure 6: Example of rule that defines a specific sequence of actions when the intent is `count_people` is detected during an active `find_person_form` loop

3.3 Pipeline and Policies

The **pipeline** defines a series of processing components for interpreting user input. This includes the `WhitespaceTokenizer` for tokenizing sentences, `RegexFeaturizer` and `LexicalSyntacticFeaturizer` for extracting features, and `CountVectorsFeaturizer` for vectorizing tokens. The `DIETClassifier` is used for intent classification and entity extraction, and it's configured with a specific number of training epochs and a similarity constraint. The `EntitySynonymMapper` maps different expressions to the same entity, while the `ResponseSelector` is used for selecting responses for specific intents.

```yaml
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true
- name: EntitySynonymMapper
- name: ResponseSelector
  epochs: 100
  constrain_similarities: true
- name: FallbackClassifier
  threshold: 0.7
  ambiguity_threshold: 0.1

# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies:
- name: RulePolicy
- name: MemoizationPolicy
- name: TEDPolicy
  max_history: 6
  epochs: 100
  constrain_similarities: true
```

Figure 7: Pipeline and Policies used.

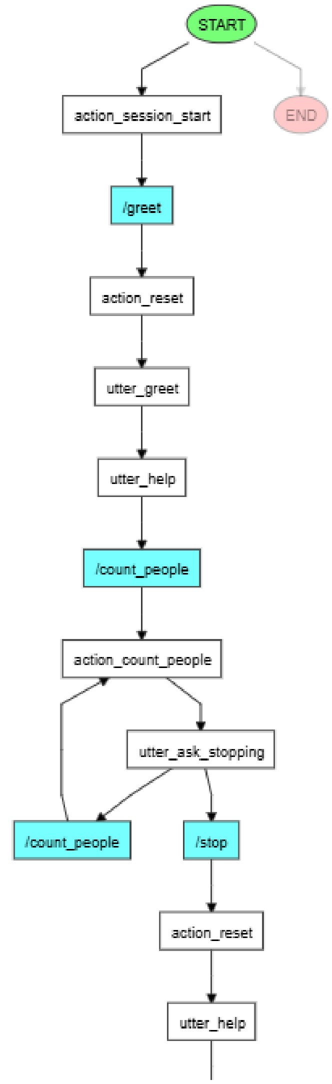

Figure 8: Example of Rasa conversation for intent `count_people`

The **core policies** define how the chatbot decides the next step in a conversation. The policies include a `MemoizationPolicy` for recalling specific conversation sequences, a `RulePolicy` for handling predefined conversation rules, and a `TEDPolicy` for predicting the next best action based on the conversation history.

4 Detector

This detector is designed to identify people, detect their faces, and classify their emotions in real-time. Using advanced computer vision and deep learning techniques, it can accurately recognize individuals, extract facial features, and analyze emotional expressions.

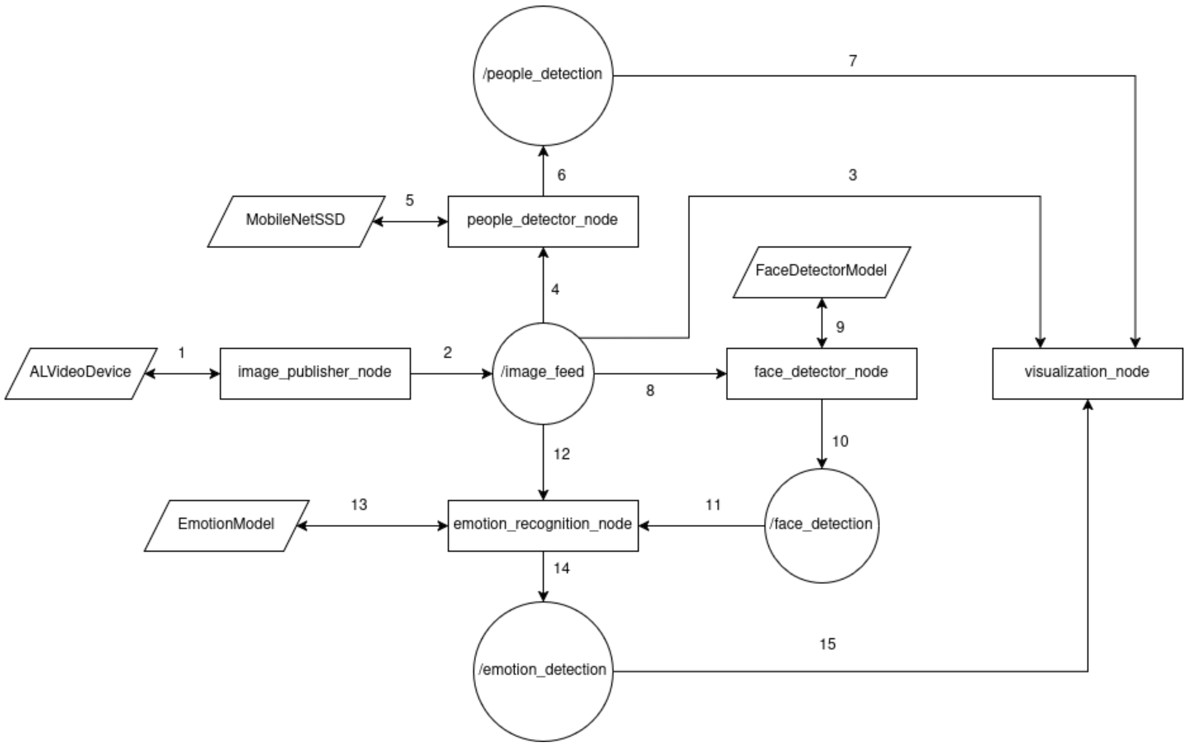

Figure 9: The complete scheme of detector

4.1 People Detection

We utilized `MobileNetSSD` for people detection because we found it to be superior to EfficientDetB0 in terms of inference performance. It is a deep learning model designed for object detection tasks, particularly optimized for performance on mobile and resource-constrained enviroments:

- **MobileNet**, the backbone of the model, is structured using depth-wise separable convolutions. This design significantly reduces the number of parameters and computational cost compared to traditional convolutional neural networks, making it efficient and fast, especially for real-time applications.
- The **SSD** part of the model allows for detecting multiple objects within the image in a single pass, providing both speed and accuracy.

4.2 Face Detection

For face detection in your application, we are using the **OpenCV Face Detector model**, which is an efficient and effective choice for such tasks. This model is built on deep learning techniques, utilizing a convolutional neural network (CNN) specifically tailored for detecting faces. In prectice, implementing the OpenCV Face Detector involves loading the model using specific files, typically a combination of a `.prototxt` file for the model architecture and a `.caffemodel` file containing the trained weights. After loading the model, it processes input images or video frames, detecting faces by identifying and outlining them with bounding boxes. This process is made efficient and adaptable to different image resolutions and conditions.

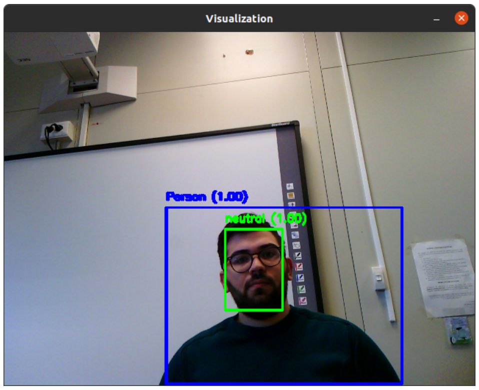

Figure 10: Example 1 of face and person detection with emotion recognition

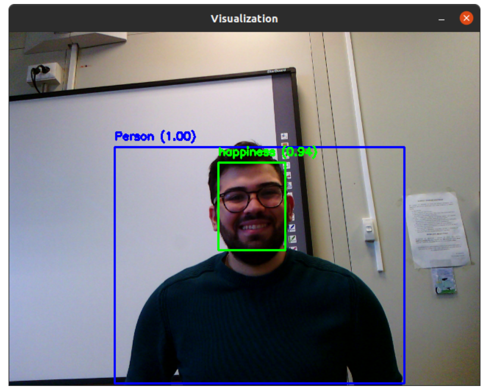

Figure 11: Example 2 of face and person detection with emotion recognition

4.3 Emotion Recognition

For emotion detection, we use a categorical model based on Paul Ekman's theory, which identifies six universal basic emotions (joy, sadness, anger, fear, disgust, and surprise) plus a neutral state. The model classifies emotions into discrete categories, allowing for immediate interpretation and quick decision-making. This approach is particularly advantageous for real-time applications. However, this method has limitations, as a predefined set of categories may not capture all variations. The processing pipeline for emotion recognition follows a series of steps to detect and classify emotions in facial expressions.

- **Face Detection:** This is the first step and is essential for locating and isolating the area of interest in the image. The node receives information about the positions of detected faces through the `/face_detection` topic published by the face_detector node. Each detection provides a bounding box that describes the area where the face is located.

- **Face Normalization:** After identifying the face, the images are resized to the input size required by the deep learning model, which is 224x224 pixels. Additionally, the face image is subtracted by the mean values to ensure greater uniformity and reduce the influence of lighting variations, improving the model's accuracy.

- **Feature Extraction:**In this phase, a pre-trained convolutional neural network (CNN) is used, loaded from the *emotion.hdf5* model. The CNN examines the face image and automatically learns relevant features, such as textures or contours. This process allows the system to adapt to variations in faces or expressions without requiring retraining.

- **Classification:** Once the facial features are extracted, the pre-trained model performs classification to assign an emotion to the detected face. Each face is associated with an emotion and a confidence score indicating how certain the model's prediction is. The emotions are then mapped to a predefined list of labels, and the result is published as a message on the `/emotion_detection` topic with information about the face's location, the emotion type, and the confidence score.

5 Test

This section provides a comprehensive evaluation of the project from both quantitative and qualitative perspectives.

5.1 Qualitative Test

Qualitative tests, on the other hand, focus on the quality and usability of the project. They aim to assess how well the system performs in terms of user experience, natural language understanding, and overall satisfaction.

5.1.1 Detectors Qualitative Test

This test allows for displaying real-time images captured by Pepper's camera (when `pepper_camera_on="True"`) or by the computer's camera (when `pepper_camera_on="False"`). It also shows any detected bounding boxes around faces and people and captures their emotions. To perform this test, you need to launch the `detector.xml` file as described in the next image. This script starts nodes for people, face and emotion detection, and by default, it can control a Pepper robot by waking up or putting it to rest and publishing images from its camera.

```
# Pepper ON
roslaunch detector detector.xml
# Pepper OFF
roslaunch detector detector.xml pepper_on:=False pepper_camera_on:=False
```


Figure 12: Command for launch the detector task

5.1.2 Animation Qualitative Test

This test allows Pepper to perform a specific animation. To do this, you need to provide Pepper with one of the animations listed at this link. To run this test, you should launch the `animated.xml` file as described below.

```
# Terminal 1
roslaunch all animation.xml
# Terminal 2
input="animation/Stand/Gesture/Hey_1"
rosservice call /animation_service "input: {data: '$input')"
```


Figure 13: This script requests Pepper to perform the Hey_1 gesture

5.1.3 Dialogue Qualitative Test

This test allows us to test the Rasa chatbot through ROS. To run this test, you should launch the `dialogue.xml` file displayed in the next figure. This launch file will create two terminals: one for the `dialogue_sever`, `rasa_actions`, and `rasa_server`, and another one for the `dialogue_interface`, facilitating the conversation between the user and the chatbot.

```
rosalunch chatbot dialogue.xml
```

Figure 14: This script starts the chatbot interface in ROS. With the command defined above, we can test also the functionalities of Speech-to-Text (S2T) and of Text-to-Speech(T2S).

- **Speech-to-Text:** This funcionality can be tested using both when using the PC's microphone (`mic_index="None"`) or Pepper's microphone. We have implemented voice detection and ASR modules as Publisher-Subscribe.
- **Text-to-Speech:** This functionality can be tested only using Pepper with a text to be played. We have implemented this functionality using the services.

5.2 Quantitative Test

Quantitative tests involve objective measurements and data-driven assessments of the project's performance.

5.2.1 Chatbot Quantitative Test

Rasa lets validate and test dialogues end-to-end by running through *test stories.* The use of test stories in the best way to have confidence in how our assistant will act in certain situations. For our project, we have developed a suite of multiple test cases that encompass a range of functionalities, from basic greetings forms to more complex tasks such as counting operations, from filling, and recognitions of fallback intents. Some of these test cases are integrated with each other. All of these tests have been successfully passed.
Once the assistant is deployed in the real world, it will be processing messages that it hasn't seen in the training data. To simulate this, we should always set aside some part of your data for testing. A solution is use **cross-validation**, which automatically creates multiple train/test splits and averages the results of evaluations on each train/test split, that is all data is evaluated during cross-validation. The command to launch is: `rasa test nul --nlu data/nlu --cross-validation`

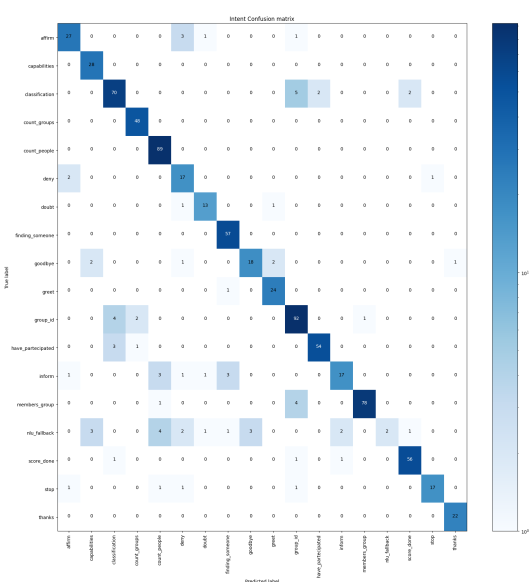

Figure 15: Confusion matrix for intent classification model

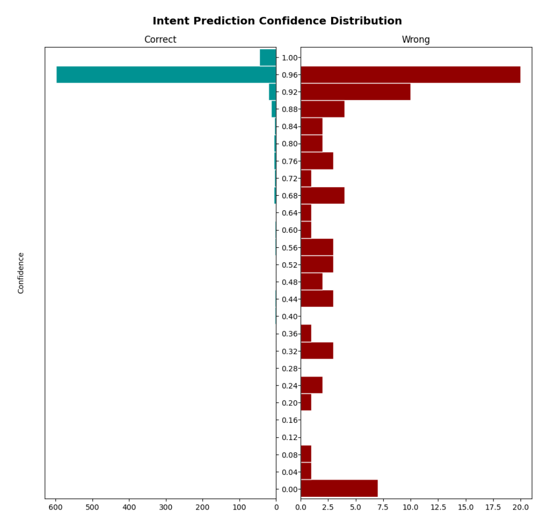

Figure 16: Confidence histogram for intent classification model

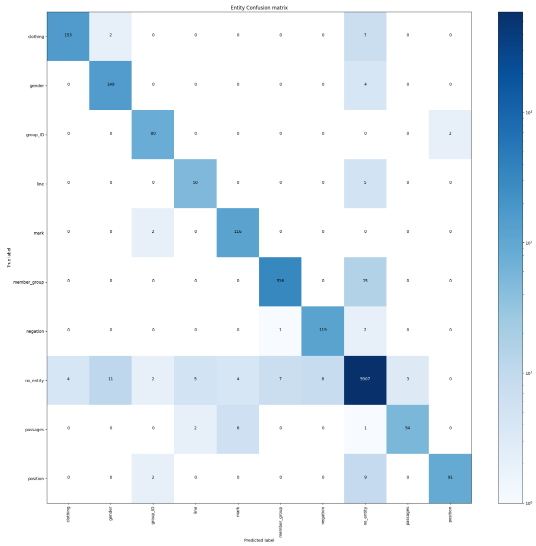

Figure 17: Confusion matrix for entity classification model

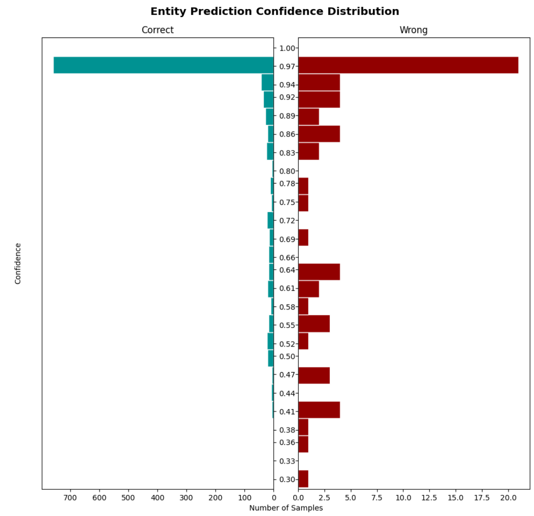

Figure 18: Confidence histogram for entity classification model

The `rasa train` script produces a report, a confusion matrix and confidence histogram for the intent classification model and also for the entity (DIET) classification model.

	ACCURACY	F1-SCORE	PRECISION
Train (Intent)	0.976 (0.001)	0.967 (0.001)	0.962 (0.002)
Test (Intent)	0.908 (0.016)	0.899 (0.019)	0.908 (0.022)
Train (DIET)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)
Test (DIET)	0.986 (0.007)	0.954 (0.020)	0.948 (0.017)

Figure 19: Intent and Entity evaluation results with cross-validation.

5.2.2 Detectors Quantitative Test

This test was designed to evaluate the effectiveness of detectors in identifying people, their faces and recognize emotions. To this end , three classes, `people_detector_test.py`, `face_detector_test.py` and `emotion_recognition_test.py`. These classes process a series of frames from different videos, executing detection and recognition on every single frame. The videos of people detection and face detection depict two distinct scenarios:

- The first scenario, labeled as `TestVideo0`, illustrates a situation where a person is already present in the scene and then leaves shortly thereafter.
- The second scenario, labeled as `TestVideo1`, illustrates a situation where a person is already present in the scene, then occluded the camera and shortly after disappear.

For the emotion detection instead, we have a scenario named `TestEmotion0` where an user pose with different face emotion. From these videos, we can derive the following metrics.

DETECTOR	PRECISION	RECALL	F1-SCORE
People Detector	1.0	0.93181	0.96470
Face Detector	0.9939	0.98192	0.98787
Emotion Detector	0.35576	0.35576	0.35576

Figure 20: Quantitative measures of the detector's performance

5.3 Final Test

This test allows testing the functionality of all implemented modules and, consequently, the overall behavior of Pepper. To execute this test, you should launch the `all.xml` file as demonstrated here:

```
# Pepper On
roslaunch all all.xml
# Pepper Off
roslaunch all all.xml pepper_on:=False pepper_camera_on:=False mic_index:=None
```
codesnap.dev

Figure 21: This script runs all the defined nodes, allowing you to test the complete behavior of Pepper. It also possible to test the complete behavior without Pepper

In particular, the test, consists of the following phases:

- **Greetings**: The person greets Pepper, and Pepper responds with another greeting gesture while uttering a welcoming phrase. In the greetings, Pepper introduces itselfas the robotic guardian of the shopping mall and asks how it can assist the person. Subsequently, the person inquires about Pepper's capabilities, and Pepper responds by explaining that it can conduce people searches based on specific attributes.
- **Finding People Task**: At this stage the person asks Pepper that he is looking for his daughter and if Pepper can help him. Pepper , guessing that it is a female, directly asks if, the person to be identified, is wearing a bag and then asks if she is wearing a hat. Having all the parameters at hand, he performs a search through his database to identify whether there is a person compatible with the attributes provided. If there is then Pepper will answer in the affirmative, if not, he will tell us that he is sorry and that he can help call mall security.
- **Contest Questioning Task**: In the next phase we ask Pepper some informations about the contest of Artificial Vision 2025, in particular we ask the winning group of the contest and the final score of group 4. After we ask Pepper some informations about our group like the name members, the score and the final position.
- **Count People Task**: Lately we ask Pepper to count how many people of gender male are in the mall and subsequently we ask the count of female in the mall. In the final part of the test we also ask Pepper the count of the people with a specific requirements, they must have gone through the alpha line.

The video of the final test is visible here.