Deliverable D6.6: Robot non-verbal behaviour system in target environments

Due Date: 30/08/2023

Main Author: HWU

Contributors: HWU, INRIA

Dissemination: Public Deliverable

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

DOCUMENT FACTSHEET

Deliverable	D6.6: Robot non-verbal behaviour system in target environments
Responsible Partner	INRIA
Work Package	WP6: Learning Robot Behaviour
Task	T6.3: Robot Non-verbal Behaviour System
Version & Date	30/08/2023
Dissemination	Public Deliverable

CONTRIBUTORS AND HISTORY

Version	Editor	Date	Change Log
0.1	HWU	20/08/2023	First Draft
0.2	INRIA	01/09/2023	Second Draft
0.3	INRIA	20/09/2023	Third Draft
1.0	HWU	25/09/2023	Final Draft

APPROVALS

Authors/editors	INRIA, HWU
Task Leader	HWU
WP Leader	INRIA

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

Contents

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

Abbreviations

Abbreviation	Meaning
A2C	Advantage Actor-Critic
AI	Artificial Intelligence
ARI	Social assistive robot used by the SPRING project
AP-HP	Assistance Publique – Hôpitaux de Paris (SPRING Partner)
BIU	Bar-Ilan University (SPRING Partner)
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CVUT	Czech technical university in Prague (SPRING Partner)
D3QN	Dueling Double Deep Q-Network
DDQN	Double Deep Q-Network)
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
ERM	ERM Automatismes Industriels (SPRING Partner)
FGD	Fréchet Gesture Distance
GRU	Gated Recurrent Unit
HRI	Human Robot Interaction
HWU	Heriot-Watt University (SPRING Partner)
INRIA	Institut National de Recherche en sciences et technologies du numérique (SPRING Partner)
MPC	Model Predictive Control
MPC	Multi Party Conversation
PAL	PAL Robotics (SPRING Partner)
PNP	Petri-Net Planner
RL	Reinforcement Learning
ROS	Robot Operating System
SAC	Soft Actor-Critic
SPRING	Socially Pertinent Robots in Gerontological Healthcare
TD3	Twin Delayed Deep Deterministic Policy Gradient
UNITN	University of Trento (SPRING Partner)
WP	Work Package (of the SPRING project)

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

Executive Summary

Deliverable 6.6 reports on the final software design and implementation of the robot non-verbal behaviour system for the target environment. This includes the interface between the non-verbal behaviour manager, the task planner, and the conversational system. The non-verbal behaviour system is compose of the Non-verbal Behaviour Manager and the Robot Non-verbal Behaviour Generation modules allowing to synthesise robot behaviour and to choose the appropriate non-verbal actions.

The work reported in this deliverable was carried out to fulfil the objectives of task 6.3 Robot Non-verbal Behaviour System. This system is responsible for taking the optimal non-verbal actions in the interaction, based on the dialogue state T5.3 and the overall plan T5.2.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

1 Introduction

The objective of Work Package 6 is to develop and implement methodologies enabling the robot to automatically:

1. Explore the environment
2. Move towards one or several persons
3. Attract the attention of the selected persons in order to facilitate face-to-face communication
4. Multi-party conversation management

In Task 6.3 of WP6 we developed the **robot non-verbal behaviour system** T6.3. This system is responsible for deciding the optimal non-verbal actions to take, based on the dialogue state T5.3 and the overall plan T5.2.

This deliverable presents the final software design and implementation of the robot non-verbal behaviour system for the target environment. This includes the interface between the non-verbal behaviour manager, the task planner, and the conversational system.

The robot non-verbal behaviour system of SPRING allows to synthesise the learnt robot behaviour and to choose the appropriate non-verbal actions for the robot to take during interactions. The Robot Non-verbal Behaviour System consists of 2 modules:

- The **non-verbal behaviour manager**, which interfaces with the high-level planner and conversational system to choose appropriate actions for the robot to take to manage the interactions.
- The **robot behaviour generation module**, which interfaces with the non-verbal behaviour manager to synthesise the optimal robot behaviour and controls the execution of the robot (non-verbal) actions during interactions.

This document describes the overall architecture for the robot non-verbal behaviour system, in Section 2. The robot behaviour generation module is described in Section 3. The non-verbal behaviour manager is described in Section 4.

The software will be released in the code repositories for WP 5[27] and WP 6 [28].

As per European Commission requirements, the repository will be available to the public for a duration of at least four years after the end of the SPRING project. People can request access to the software to the project coordinator at `spring-coord@inria.fr`. The software packages use ROS (Robotics Operating System) [29] to communicate with each other and with the modules developed in the other work packages.

2 Robot Non-Verbal Behavior System Architecture

Fig 2.1 presents the overall architecture for the robot non-verbal behaviour system.

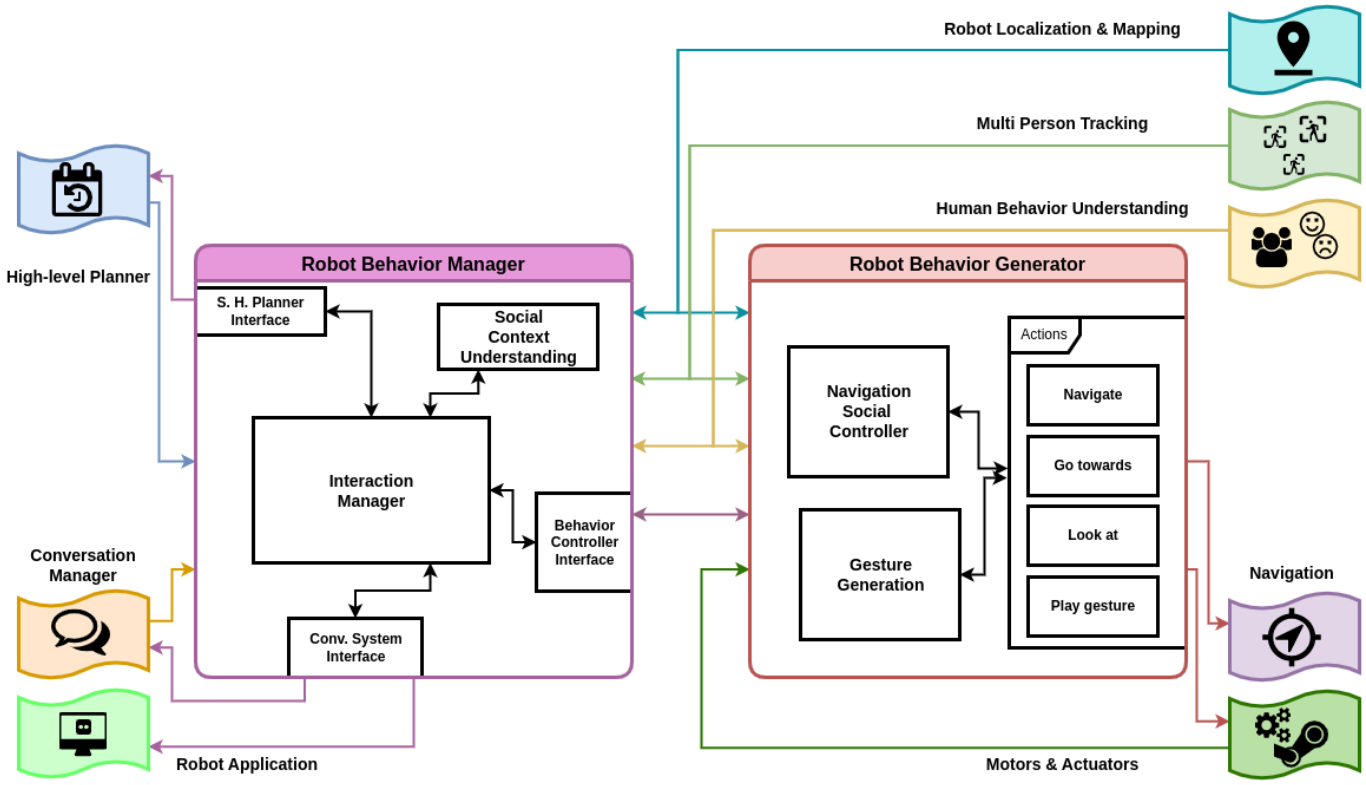


Figure 2.1: The Robot Non-verbal Behaviour System, which consists of the Non-verbal Behaviour Manager and Robot Non-verbal Behaviour Generation modules allowing to synthesise robot behaviour and to choose the appropriate non-verbal actions. The non-verbal behaviour manager, interfaces with the high-level planner and conversational system, and is responsible for deciding appropriate actions to take and managing the interactions. The robot behaviour generation module interfaces with the non-verbal behaviour manager and controls the execution of the robot (non-verbal) actions during interactions. The Robot Non-verbal Behaviour System modules take as input the high-level information provided by the other modules in SPRING, such as: robot localization and mapping, multi-person tracking, human behaviour understanding, conversation manager, high-level planner, and robot motors' and actuators' joint states. The outputs are given as robot actions, in the form of motor and actuator commands for gestures and navigation, as well as updates for the high-level plan and conversational dialogue state, and the robot application on the ARI tablet screen.

For the robot (ARI from PAL robotics) in the SPRING project, non-verbal behaviors are comprised of navigation and gesture (mainly arm, head, and eye movements) behaviors. These will allow the robot to navigate towards a person or a group of people to start a conversation; explore the environment while avoiding obstacles; move towards one or several persons in order to improve the quality of the sensory data (images and acoustic signals); attract the attention of the selected persons in order to facilitate face-to-face communication; and other action policies for multi-party conversation management.

These tasks are complex and based on high-dimensional variables such as the visual or auditory input from the robot cameras and microphones. The Non-verbal Behaviour System take as input the high level information provided by other work packages: robot localization and map of the environment (WP2), human behaviour understanding

(WP4), localization and identification of people with respect to the robot (WP3), dialogue state and the overall plan state (WP5), robot joint states (WP7), etc. Its outputs are given as robot actions, in the form of motor and actuator commands for gestures and navigation, as well as updates for the high-level plan and conversation dialogue state.

The robot non-verbal behaviour system is responsible for choosing the appropriate non-verbal actions to take and synthesise robot behaviour enabling multi-modal multi-person interaction and communication. It is made of a Non-verbal Behaviour Manager and a Robot Non-verbal Behaviour Generation modules, as illustrated in Fig 2.1.

The robot non-verbal behaviour manager consists of the *Interaction Manager* and the *Social Decision Making* components, and handles the interface between the *high-level planner*, the *conversational system* and the *robot behaviours*.

The robot behaviour manager module interfaces with the non-verbal behaviour generation through the *Robot Non-verbal Actions* servers. The behaviour manager handles high-level interaction decisions and the behaviour generation module controls low-level action execution. Fig 2.2 illustrates the components of the Robot Non-verbal Behaviour System as integrated into the the conversational system, and the high-level planner.

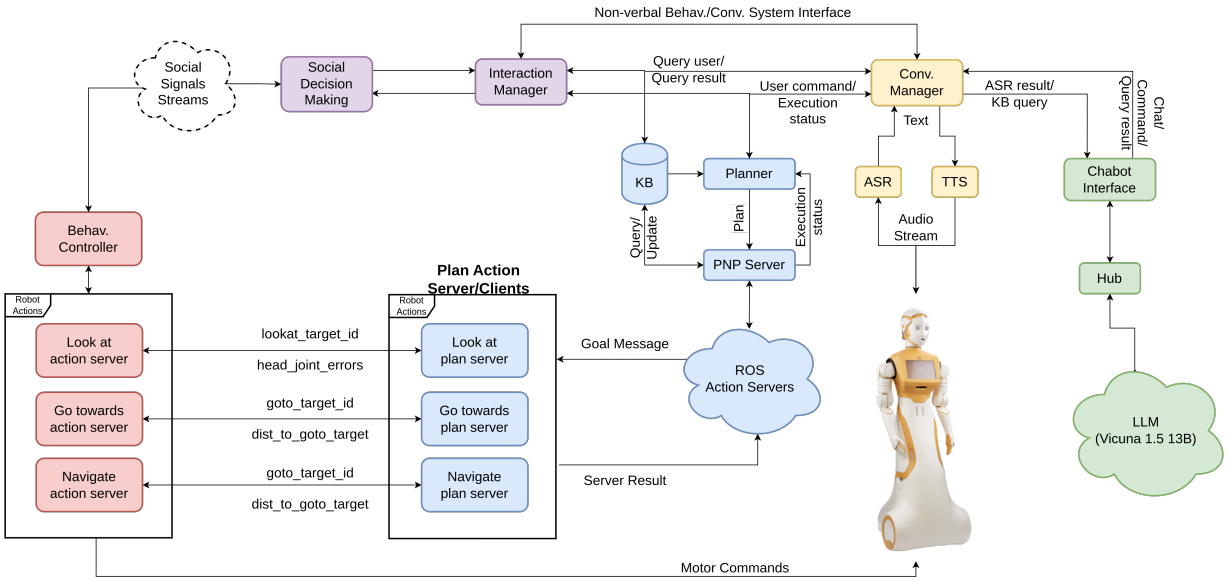


Figure 2.2: System architecture for the Robot Non-verbal Behaviour System; and the interface between the non-verbal behaviour manager, the task planner, and the conversational system. The non-verbal behaviour manager (in purple) the high-level planning framework (in blue), the conversation manager interface (in yellow) and the conversational system on the right (in green) with the robot non-verbal behaviour manager components (in red) on the left.

A description of the the Non-verbal Behaviour Manager and Robot Non-verbal Behaviour Generation modules is given in the next sections. Section 4 presents the non-verbal behaviour manager, whereas in Section 3 we describe the robot behaviour generation module.

3 Non-Verbal Behavior Generator

The robot's non-verbal behavior generation is in charge of the synthesis and execution of the robot's (non-verbal) actions during interactions. It controls the low-level positioning, navigation, and gesture execution of ARI, the SPRING robot. The following section introduces the general architecture, followed by a description of the navigation, and finally the gesture generation components.

3.1 General Architecture

The non-verbal behavior generation module consists of several ROS nodes responsible for different aspects of the non-verbal actions and modalities (Fig. 3.1). Additionally, ROS action servers exist which are used to control the behavior generation such as by giving a goal position the robot should navigate to. The robot behavior generation module interfaces with the non-verbal behavior manager through these action servers. The next section lists the different components and action servers. The following section describes its interface.

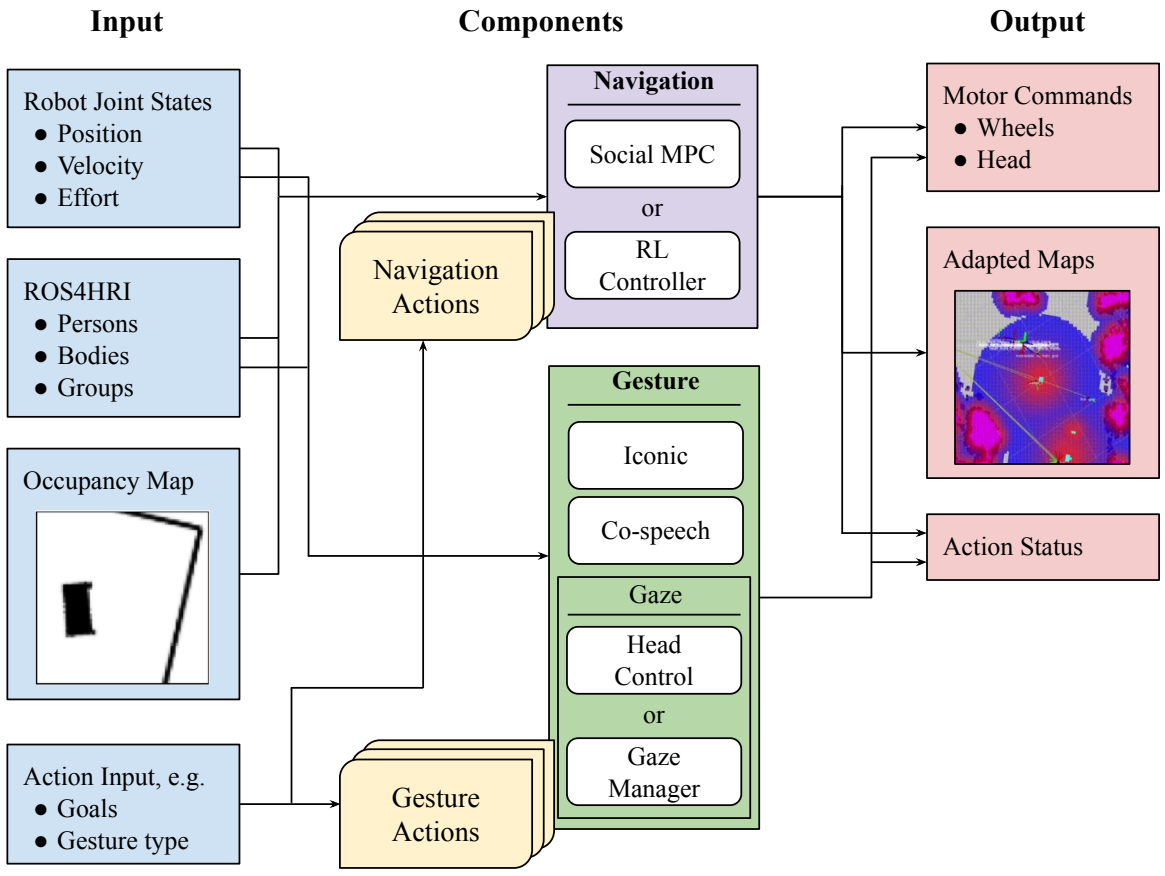


Figure 3.1: General Architecture of the Non-Verbal Behavior Generator.

Components

Several components are responsible for different aspects of behavior generation. These include components for navigation and gesture generation. The components themselves can be either a single ROS node or a combination of several nodes. Two main components exist. The first is responsible for navigation tasks and the second is for gestures. The sections of this report are organized accordingly.

- Navigation: human-aware navigation with obstacle avoidance (2 alternative controllers)
  - Social MPC: navigation based on a model predictive controller with a social space model.
  - RL controller: navigation behaviors trained with reinforcement learning.
- Gesture: different types of non-verbal gestures
  - Iconic Gestures: allows to play some iconic gestures, e.g. waving or pointing.
  - Co-speech automatically generated upper-body gestures correlated with the robot's speech.
  - Gaze: control of head and eyes (2 alternative controllers)
    * Head Control: controls head position to look at a direction.
    * Gaze Manager: uses the head position and the eyes to look at a direction.

3.1.1 Interface

The robot's behavior can be controlled via ROS Messages. For most navigation and gaze functionality different ROS action servers exist that receive commands and respond with status messages.[1] Moreover, the different components have configuration files that can be adapted to change most of their relevant parameters.

Inputs

List of input data (via ROS Messages):

- Robot joint states, including their position, velocity, and effort published at 50 Hz (joint_states ROS topic). This also includes the robot TF, i.e. the position of the robot in the world map.
- ROS4HRI messages: Information regarding persons, bodies, and groups (/persons/*, /bodies/*, /groups/* ROS topics). This includes their position relative to the robot.
- Local Occupancy Grid around the robot where cells indicate the probability of obstacles (/slam/grid_map ROS topic).
- Action Servers:
  - Navigation Actions: the actions move ARI to a target location, specified in the command. Targets can be persons, groups, or a specific position (x, y, θ)) in the local map. A timeout period (in seconds) when the actions should end can be given. The actions return the current distance to the target with the final time it took to execute the action.
  - Gesture Actions: actions trigger different types of gestures. For example, the 'Look at' actions align ARI's head to face a direction, person, or group. Additionally, a timeout period (in seconds) can be given which indicates when the actions should end. It returns a head joint error (pan and tilt) with the final duration it took to execute the action.

Outputs

The main output of the module is the actions sent to the low-level motor control of the robot. Moreover, the action servers provide messages regarding their current control status. Additionally, as part of the navigation module (Social MPC), several adapted occupancy grids are published. List of outputs:

- Motor commands:
  - Linear forward velocity (x-axis) and angular velocity (z-axis) of wheels: nav_vel (geometry_msgs/Twist ROS message).

[1] http://wiki.ros.org/actionlib

  - Pan and tilt angle positions of the head: /head_controller/command (trajectory_msgs/JointTrajectory ROS message)

- Status topics of action servers
- Adapted maps based on local occupancy grid:
  - Dilated map based on static obstacles: /slam/local_static_raw_map
  - Map of dynamic obstacles (detected with the RGBD torso front camera): /slam/obstacle_map
  - Dilated map based on static (/slam/obstacle_map) and dynamic (/slam/local_static_raw_map) obstacles (max over both): /slam/local_map
  - Cost map used by the social MPC based on the /slam/local_map: /slam/local_static_cost_map
  - Cost map built on the social spaces of the humans: /slam/local_social_cost_map
  - Dilated global map: /slam/global_map

3.2 Navigation

The main component of the behavior generator are the navigation modules. They perform human-aware navigation to move the robot to a goal position while taking obstacles and humans into account, joining a person or group, or following and guiding persons. Two alternative navigation modules are being developed. First, a Social Model Predictive Controller (Social MPC) plans the trajectory of the agent by taking the social spaces of humans and groups into account. Second, control behaviors (policies) that are trained via reinforcement learning (RL).

3.2.1 Social MPC

The Social MPC combines an MPC with a social model of human spaces. The general architecture for the MPC controller of the robot is described in SPRING Deliverable 6.1 [24]. Next, the general framework of MPC's will be introduced which is followed by its components: the cost function, the social space model, and how navigation targets are selected.

MPC

Model Predictive Control (MPC) is a control strategy [4] used in various fields, including robotics, engineering, or process control. It utilizes a dynamic forward model f of the target system (ARI robot) to predict its behavior over a certain time horizon, considering the current state and anticipated inputs. By optimizing a cost function J that captures desired performance and constraints, MPCs compute an optimal control trajectory for the system's inputs. As time progresses, only the first control action is applied to the system, and the optimization process is then repeated in a receding-horizon fashion, incorporating updated measurements and adjusting the control inputs. This approach allows MPCs to handle complex systems with constraints, time-varying dynamics, and disturbances, making it valuable for applications requiring precise and adaptable control in real-time.

The forward model f takes into account the robot state $\mathbf{x}$ and the control inputs $\mathbf{u}$ at time step t to predict the next time step $t+1$:

$$\mathbf{x}(t+1) = f(\mathbf{x}(t), \mathbf{u}(t)). \tag{3.1}$$

The model is formulated with discrete time steps, but the framework can be expressed in a continuous setting similarly. For the ARI robot, the state is defined over the joint angles of its wheels (α_1, α_2), its position (forward-direction: x, left-side-direction: y), and its clockwise orientation θ:

$$\mathbf{x}(t) = (\alpha_1(t), \alpha_2(t), \theta(t), x(t), y(t)).$$

All coordinates are in an ego frame, i.e. at time step $t = 0$: $x(0) = y(0) = \theta(t) = 0$. The motors of the robot's two wheels are velocity controlled, i.e. each motor takes a target velocity as input ($\dot{\alpha}_1(t), \dot{\alpha}_2(t)$). Using a simple coordinate change, these can be expressed by a linear velocity v and an angular velocity ω to control the robot:

$$\mathbf{u}(t) = (v, \omega).$$

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

The forward model f is given by:

$$\begin{pmatrix} x(t+1) \\ y(t+1) \\ \theta(t+1) \end{pmatrix} = \begin{pmatrix} x(t) - \Delta_t \sin(\theta(t))v \\ y(t) + \Delta_t \cos(\theta(t))v \\ \theta(t) + \Delta_t \omega \end{pmatrix}$$

where $\Delta_t \in \mathbb{R}$ is the duration of a discrete time step.

The forward model is used to find the optimal inputs $\mathbf{u}(t)$ to finally reach a given target state $\hat{\mathbf{x}}$ while minimizing a loss function $\mathcal{L}$ and considering some constraints g. The complete functional J that should be minimized is given by:

$$J_N(\mathbf{x}(0), \mathbf{u}(0), \ldots, \mathbf{u}(N-1)) = \sum_{t=1}^{N} \mathcal{L}(\mathbf{x}(t), \hat{\mathbf{x}}) + \lambda ||\mathbf{u}(t)^2|| \quad | \text{ s.t. } g(\mathbf{u}(0), \mathbf{u}(1), \ldots) \geq 0 \tag{3.2}$$

where N is the finite horizon, $\lambda ||\mathbf{u}(t)^2||$ is a regularization term to reduce large actions and improve the smoothness of the navigation, and the constraints $g(\mathbf{u})$ enforce a lower and upper limit ($\mathbf{u}_{\min}, \mathbf{u}_{\max}$) on the robot velocities:

$$g(\mathbf{u}) = \begin{cases} \mathbf{u} - \mathbf{u}_{\min} \\ \mathbf{u}_{\max} - \mathbf{u} \end{cases}.$$

The optimization iterates in a loop over the following steps:

1. Estimate/measure current state $\mathbf{x}(0)$;
2. Minimize J_N with respect to controls $\mathbf{u}(0), \ldots, \mathbf{u}(N-1)$, subject to (3.1);
3. Apply first control $\mathbf{u}$ for $t = 1$.

The term J_N is minimized in Step 2 with a sequential quadratic programming (SQP) algorithm for nonlinearly constrained gradient-based optimization [12]. For this purpose, the SLSQP implementation of SciPy[2] is used. To compute the gradients, an automatic differentiation jax library [3] is used.

The MPC control scheme allows for horizons (N) of a few seconds. A global planner is used to create a full path to the final goal position. The full path is split into a sequence of intermediate waypoints that are always within sight of the robot. These waypoints are given iteratively to the MPC as goal positions $\hat{\mathbf{x}}$. A fast marching algorithm [20] is used to compute the full path based on the static obstacle cost function described in the following section. It computes a path that would be optimal without the presence of persons in the scene.

Cost Function

The cost function $\mathcal{L}(\mathbf{x})$ of the MPC is represented as a cost map F_{MAP} over the (x, y) space. It takes into account a cost map for static obstacles F_{OBS} and a map for social spaces F_{SOC}:

$$F_{\text{MAP}}(x, y) = \lambda_{\text{OBS}} F_{\text{OBS}}(x, y) + \lambda_{\text{SOC}} F_{\text{SOC}}(x, y)$$

Static Obstacles The avoidance of static obstacles is integrated via a cost map. The robotic vision system provides a binary two-dimensional occupancy map:

$$F_{1,\text{OBS}}(x, y) = \begin{cases} 1 & \text{if obstacle at } (x, y) \\ 0 & \text{otherwise.} \end{cases}$$

From this map, a fast marching algorithm [19] computes the closest distance of each map point to an obstacle:

$$F_{2,\text{OBS}}(x, y) = \inf \{ |(x, y) - (x', y')| : F_{1,\text{OBS}}(x', y') = 1 \}.$$

The final obstacle avoidance cost map is given as follows:

$$F_{\text{OBS}}(x, y) = \begin{cases} 0 & \text{if } F_{2,\text{OBS}}(x, y) > r_{\text{BASE}} \\ r_{\text{BASE}} - F_{2,\text{OBS}}(x, y) & \text{if } 0 \leq F_{2,\text{OBS}}(x, y) < r_{\text{BASE}} \\ r_{\text{BASE}} & \text{if } F_{2,\text{OBS}}(x, y) < 0 \end{cases}$$

where r_{BASE} represents a margin to introduce a non-zero gradient in the cost function near the border to objects and ensures that since only the robot center point is considered on the map, the whole robot is not colliding with obstacles.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

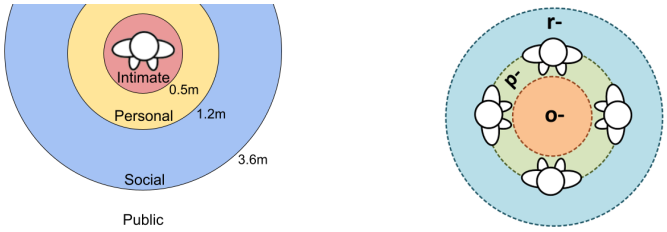


Figure 3.2: Social space models for individuals (right) and groups (left). Images adapted from [21].

Social Spaces The second cost map takes into account the social space of humans and groups. The goal is to avoid interfering with their spaces during navigation. The social space is based on the personal space model by Hall [9] for single persons and by the F-formation model by Kendon [11] for groups (Fig. 3.2).

Personal space refers to the emotionally and physically comfortable distance that individuals prefer to maintain between themselves and others in social interactions. It varies across cultures, contexts, and relationships, but generally encompasses a zone around an individual's body where they feel a sense of privacy, security, and control. Personal space serves to establish boundaries, reduce potential threats, and uphold a sense of personal autonomy.

An F-formation is a concept from social interaction analysis that refers to the spatial arrangement or pattern people naturally adopt when engaging in face-to-face conversations or group interactions. They consist of three different sub-spaces: 1) The o-space is the space in the middle of the group which is perceived as the interaction area. 2) The p-space is the space around the o-space where persons are located. 3) The r-space surrounds the group and is not considered to be part of the group interaction.

Based on these psychological theories a physical model of social space (F_{SOC}) is constructed following the paper by Truong and Ngo [30]. The model defines for each point around a human or a group a value of their social space. Generally, the closer a point is to a human the higher its social space value. The model depends on the state of humans, which is denoted by $\mathbf{p}_n$ (state of human n) with:

$$\mathbf{p}_n = (x_n, y_n, \theta_n, v_n, \omega_n), \text{ for } n \text{ in } \{1, \ldots, N_{\text{H}}\},$$

where x_n, y_n is its position, θ_n the orientation, and v_n, ω_n are the linear and angular velocities respectively. As for the robot's dynamics, by assuming constant linear and angular velocities, the future state of the person is approximated using:

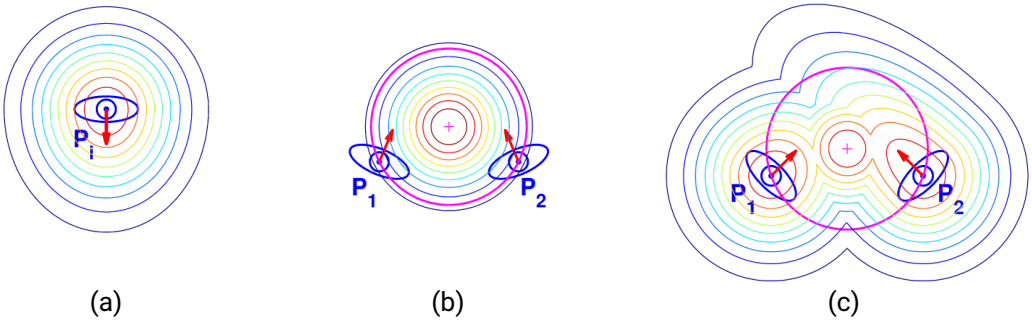
$$\begin{aligned} x_n(t+1) &= x_n(t) - \Delta_t \sin(\theta_n(t)) v_n \\ y_n(t+1) &= y_n(t) + \Delta_t \cos(\theta_n(t)) v_n \\ \theta_n(t+1) &= \theta_n(t) + \Delta_t \omega_n. \end{aligned}$$

For groups $\mathbf{g}_m$ the model takes into account the position (x_m, y_m) of the group centers and the group center width σ_m: $\mathbf{g}_m = (x_m, y_m, \sigma_m)$, for m in $\{1, \ldots, N_{\text{G}}\}$.

The social model differentiates between social spaces for individual humans (F_{PER}) and groups (F_{GRP}). The model for an individual humans $F_{\text{PER},i}$ takes into account the following factors:

- Distance to human: $d = \sqrt{(x - x_n)^2 + (y - y_n)^2}$

[2] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html



(a) (b) (c)

Figure 3.3: Different social space models. (a) F_{PER}: Model for individual persons. (b) F_{GRP}: Model of groups. (c) F_{SOC}: Combined model of persons and groups. Figures adapted from [31].

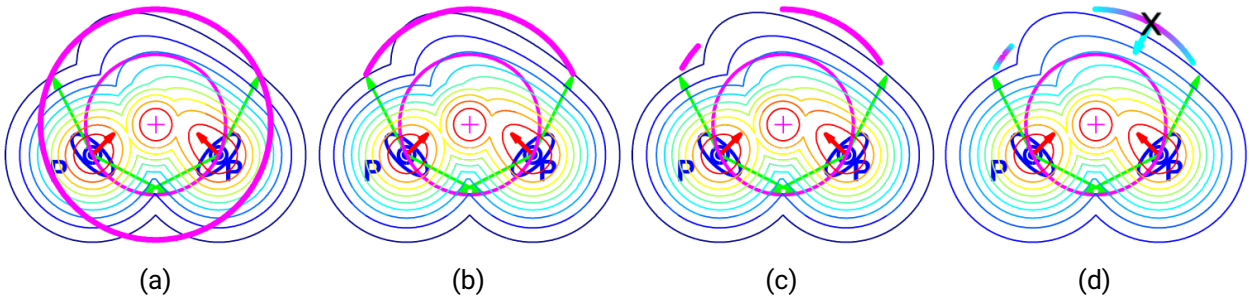


Figure 3.4: Estimation of the approaching pose. (a) The approaching area is the thick outer magenta circle. (b) Filtered approaching areas by human FOV, and (c) the social space F_{SOC}. (d) The final approaching pose (black x) is the central point of one of the remaining approaching area segments. Figures adapted from [31].

- Relative angle to human front-facing side: $\theta_d = |\operatorname{atan2}(\sin(\theta - \theta_n), \cos(\theta - \theta_n))|$ with $\theta = \operatorname{atan2}(y - y_n, x - x_n)$
- Velocity of the human: v_n
- State of the human: standing, moving, or sitting

Fig. 3.3 depicts the social space for different situations. Details about the function to compute the model are given in [31].

For groups, the model (F_{GRP}) takes into account the distance to the group center and its width using a Gaussian function:

$$F_{\text{GRP},m}(x,y) = w_g \exp\left(-\frac{\sqrt{(x-x_m)^2+(y-y_m)^2}}{\sqrt{2}\sigma_m}\right)$$

where w_g is a scaling factor (Fig. 3.3).

The final model takes the individual and group spaces into account by using their maximum social space value per position:

$$F_{\text{SOC}}(x,y) = \max\left(F_{\text{PER},1}(x,y), \ldots, F_{\text{PER},N_H}(x,y), F_{\text{GRP},1}(x,y), \ldots, F_{\text{GRP},N_G}(x,y)\right)$$

for N_H individuals and N_G groups.

Goal Target Selection

Depending on the scenario, the final goal can either be given as a position in world coordinates, or a target person or group. In the case of a person or group, the target position of the robot to join it is calculated based on their social space models (F_{SOC}) following the approach in [31]. The process of estimating the approaching pose is done in 4 steps (Fig. 3.4): a) To initiate the approaching area estimation, a circular region is defined around either the target individual or group. b) Subsequently, this circular area is filtered to identify the area that is in the field of view (FOV) of the target person or all group members. c) The remaining area is further filtered by the social space model with a maximum level: $F_{\text{SOC}} > \lambda_{\text{TRG}}$. d) The center of the closest remaining segment of the filtered area is selected as the goal position.

3.2.2 RL Controllers

The SPRING project explores an alternative navigation controller to the MPC that is trained via Reinforcement Learning (RL). RL is a machine learning paradigm focused on training agents to make sequences of decisions in an environment to maximize a cumulative reward. In RL, an agent interacts with an environment by taking actions, and based on these actions, it receives feedback in the form of rewards or penalties. The agent's goal is to learn a strategy, called a policy, that guides its actions to achieve the highest possible long-term reward. Through exploration and exploitation, the agent learns to map states of the environment to actions in a way that optimizes its performance.

The RL controller is trained using the 2D simulator that was developed for the SPRING project [25]. It simulates the robot, obstacles (e.g. walls or furniture), and humans. The simulated robot has the same high-level sensor and actuator components that are available on the ARI robot. This includes a semantic robot-centric map that tracks objects and humans which is mainly used for the RL controllers. The robot can be controlled via linear (forward, backward) and angular (rotation) velocity commands. The movement behavior of humans can be either fully scripted or controlled via a social force model [17]. For the force model, a goal position can be defined to which a human moves while trying to avoid collisions. This includes the control of groups of people to have coordinated movements

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

and gather around a given group center. For the RL training, a reward function can be defined based on the properties of the simulation including, for example, if a collision was detected or the distance of the robot to a goal position or human agent. After training the agent in simulation, it is transferred to the ARI robot.

Several RL algorithms have been implemented to train the agent, including DQN, DDQN, D3QN, DDPG, TD3, A2C, and SAC. DQN, DDQN, and D3QN are critic-only RL methods to learn optimal decision-making policies in environments with discrete actions. For this purpose, the actions are discretized into 28 pairs of linear and angular velocities. The available linear velocities are: $[0, 0.1, 0.2, 0.4]$ and the available angular velocities are: $[-0.4, -0.2, -0.1, 0, 0.1, 0.2, 0.4]$.

Both algorithms have a critic network that estimates Q-values, which represent the expected cumulative reward for taking a specific action a in a state s and following afterward a certain policy π:

$$Q^{\pi}(s_t, a_t) = \mathrm{E}\left[\sum_{k} \gamma^k r_{t+k}\right],$$

where $\gamma \in [0, 1)$ is a discount factor that rewards nearby rewards more than distant rewards. The agent's goal is to learn the optimal Q-function Q^* which follows the policy π^* that maximizes it. It can be defined using the Bellman Equation and by using for the next state s_{t+1} the action that maximizes the Q-function:

$$Q^*(s_t, a_t) = \mathrm{E}\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')\right].$$

Based on the optimal Q-function, the optimal policy can be acquired: $\pi^*(s) = \max_a Q^*(s, a)$. The agent learns the Q-function using Temporal Difference (TD) learning. Starting with an initial Q-value approximation, the algorithm iteratively explores the environment to collect observations, i.e. state transitions $(s_t, a_t, r_t, s_{t+1},)$ with the observed reward. Based on the observations it updates its Q-values using a temporal difference error, adjusting for the difference between expected and actual rewards:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \max Q(s_{t+1}, a') - Q(s_t, a_t)\right),$$

where $\alpha \in \mathbb{R}$ is a learning rate parameter. Over time, Q-values converge to reflect near-optimal policies, from which an agent can make action decisions. Deep RL algorithms approximate the Q-value using deep neural networks. These are updated in the same manner as for TD learning, but by using batches of observations. When the agent explores the environment it stores its observations in a memory called an Experience Replay Buffer (ERB). During the learning procedure, the agent samples experiences from the ERB and updates its neural network parameters using a Mean Squared Error (MSE) Loss:

$$\mathcal{L} = ||Q(s_t, a_t) - r_t + \max_{a'} \gamma Q(s_{t+1}, a')||^2$$

We explored 3 critic-only algorithms:

- DQN: The DQN (Deep Q-Network) algorithm uses experience replay and target networks to stabilize training by randomly sampling and learning from past experiences, reducing data correlation and mitigating overfitting [16]. DQN also utilizes a greedy policy with ϵ-greedy exploration, where it selects actions based on the Q-network's predictions. The algorithm updates the Q-network iteratively using TD learning.
- DDQN: The DDQN (Double Deep Q-Network) algorithm is a variation of DQN [10]. They differ in how they handle action selection and Q-value estimation during training. In DQN, a single neural network is used to approximate both the target Q-values and the Q-values used for action selection, leading potentially to an overestimation of Q-values due to the maximization operation in the TD error equation. DDQN, on the other hand, mitigates this overestimation issue by employing two separate neural networks: one for selecting actions and another for estimating target Q-values. This means that the target Q-values used for updates are derived from the Q-network that selects actions, resulting in a more accurate and stable Q-value estimation.
- D3QN: The D3QN (Dueling Double Deep Q-Network) algorithm is another variation of DQN [33]. It is based on the DDQN algorithm but has a network that divides the Q-value estimation into two parts: The value function and the Advantage function. The value function estimates how good it is to be in a state and the advantage function estimates the advantage of taking action in that state. The advantage function [1] is defined as: $A(s, a) = Q(s, a) - V(s)$. This approach learns state values V separately from the effect of action advantages A. It helps to identify states where actions have no impact and to represent better the difference in value between actions.

Besides the critic-only algorithms, we also explored actor-critic algorithms that learn policies for continuous actions. They have two networks compared to the critic-only agents (DQN, DDQN, D3QN). The first network, called 'actor', takes the state as input and predicts a distribution probability for taking actions (the policy). The second network is the 'critic'. It takes as input both the state and the action predicted by the actor and outputs the Q-value. We explored 4 different actor-critic algorithms:

- DDPG: The DDPG (Deep Deterministic Policy Gradient) algorithm can be described as the actor-critic equivalent of DDQN [16]. It has a target network for both the actor and the critic. DDPG is a model-free algorithm based on the deterministic policy gradient that can operate over continuous action space.
- TD3: The TD3 (Twin Delayed Deep Deterministic Policy Gradient) algorithm is an improvement upon the DDPG algorithm [5]. It introduces target policy smoothing to mitigate overestimation bias, incorporates delayed policy updates to prevent policy oscillations, and clips critic targets to ensure stable learning. TD3 is an off-policy algorithm that employs gradient descent methods for actor and critic network updates.
- A2C: The A2C (Advantage Actor-Critic) algorithm uses parallel environments to collect experiences, reducing data correlation, and optimizes the actor and critic networks simultaneously [15]. It computes advantages for each action to guide policy updates, leading to more stable and efficient learning. A2C is on-policy.
- SAC: What sets SAC (Soft Actor-Critic) apart is its emphasis on entropy regularization, encouraging the policy to be both deterministic and stochastic [8]. This encourages exploration and prevents the policy from becoming too deterministic prematurely. Additionally, SAC employs a maximum entropy framework, enabling the agent to learn not only to maximize rewards but also to maximize the policy's entropy, leading to more diverse and exploratory behaviors. This combination of elements makes SAC effective for complex environments where precise control and exploration are required, such as robotic tasks and autonomous systems.

The critic and actor components of the agents are represented by a deep neural network (Fig. 3.5). Several network architectures have been evaluated that differ in their number of neurons or the type of layers. They take as input the state, e.g. the egocentric occupancy grid map or similar information, and output Q-values and actions (for actor-critic methods). Both actor and critic have a common state encoder that we call the navigation encoder.

In general, agents were trained by allowing the agent to explore the environment for several episodes while updating its critic and policy. An episode starts in a random position of the simulated environment and has a random distribution of obstacles in it. An episode ends if the agent reaches its randomly sampled goal position, if it collides with an object, or when it reaches a maximum step limit. At each step, the critic (critic-only) or actor (actor-critic) is used to compute the action of the agent. The observed state transition is added to its ERB. After the step, the agent updates its critic and actor networks by sampling a batch of experiences (transition observations) from the ERB. The networks are then updated accordingly.

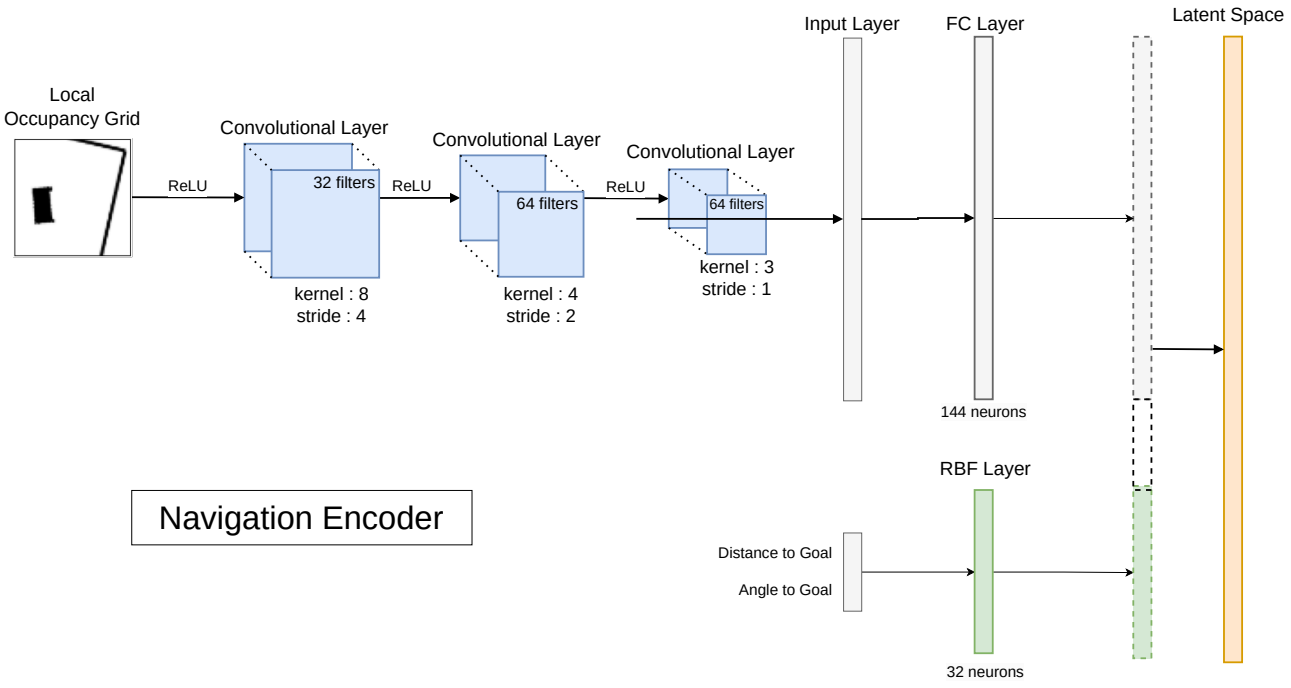


Figure 3.5: Navigation Encoder: The encoder takes as input the state and computes a latent representation. The encoded representation is then given to the actor and critic network to compute respectively the policy and the Q-value. The robot-centric occupancy grid is encoded with a CNN. The polar coordinates of the goal are encoded with a Radial Basis Function layer [2].

3.2.3 Experimental results for the Social MPC and RL agent

An initial comparison between the Social MPC and the RL agent has been conducted. On the figure 3.6, we can observe first that the MPC produces a trajectory that is smoother compared to the RL agent. This is probably because the MPC receives sub-goal positions from a global path planner. The MPC joined the goal in approximately 456 steps compared to 491 steps for the RL agent. There is not such a big difference which proves that the RL agent can be a valuable solution for navigation.

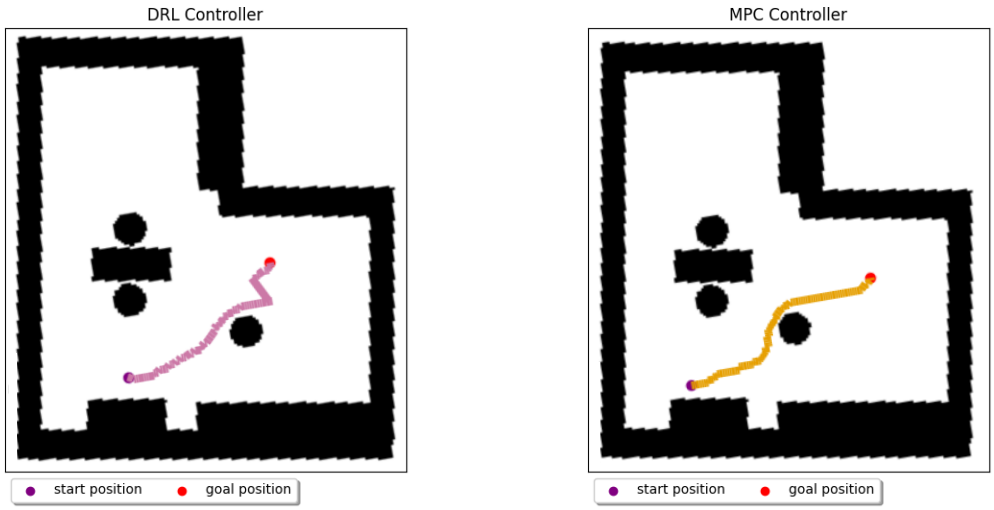


Figure 3.6: Comparison between MPC and DRL controller

3.3 Gesture Generation

A second modality of non-verbal behaviors are gestures. These involve on the ARI robot the movement of the arms, head, and eyes. Three different gestures types are distinguished:

- Iconic: These can be triggered to express a specific meaning.
- Co-speech: These gestures are automatically generated while ARI is speaking based on its spoken dialogue and audio.
- Gaze: The Gaze of ARI can be controlled via its eyes and head.

3.3.1 Iconic Gestures

Iconic gestures can be triggered to express specific meanings, such as welcome gestures (waving, bowing). The ARI robot has a library of gestures, including:

- Wave: wave the right arm.
- Bow: the head inclines downward and the arms go backward.
- Alive: several alive modes where only the arms are moving so that ARI looks alive.
- Nod: the head moves up and down.
- Handshake (right or left): greet somebody with a handshake.
- Point (right or left): point to the right/left with also the head and the arms indicating in this direction.
- Look around: the head moves right and left.

3.3.2 Co-speech Gesture Generation

Co-speech gesture generation refers to the process of producing nonverbal gestures, such as hand movements, facial expressions, and body postures, that accompany and complement spoken language during communication. These gestures are synchronized with speech and play a crucial role in communication. We implemented a neural architecture for the generation of upper body gestures for ARI (movement of arms and head) during conversations.

Generator Architecture

The architecture (Fig. 3.7) takes as input the utterances that ARI will speak as audio (waveform) and text and generates the movement of ARI's joints. The architecture is based on a Generative Adversarial Network (GAN) and trained on recorded video data from humans giving talks during TED conferences [35]. The GAN model is based on the architecture by [34]. It differs mainly in the text encoder that is replaced by a pretrained Transformer model based on

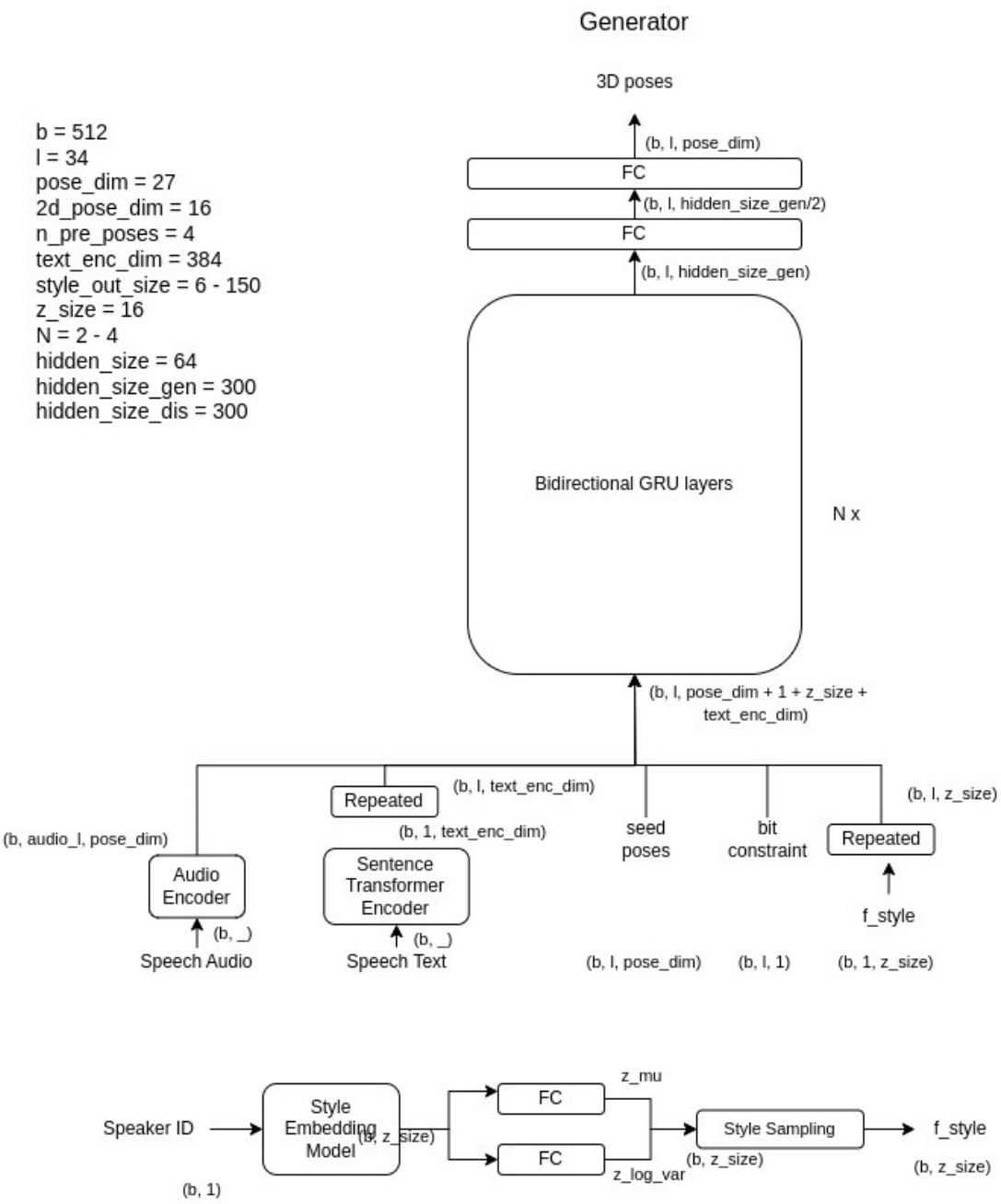


Figure 3.7: Co-Speech Gesture model Architecture

[32].

The generator network takes four inputs per data sample: a) The spoken text as a string. b) The spoken text as an audio wave. c) The initial poses of the speaker. These represent the joint poses of the previous 4 time frames so that the goal is to generate poses that continue from these. Poses are represented as directional vectors representing the relative positions of child joints from parent joints. Each pose encodes 9 joints where each joint is represented by a 3D vector ($p \in \mathbb{R}^{27}$). d) The ID ($\in \mathbb{N}$) of the speaker which is used to compute a style embedding to reflect inter-person variability. A specific ID can be selected during conversations to generate gestures similar to the speaker from the source dataset.

The text input is processed by a pretrained sentence encoder[3] based on the Transformer architecture in [32]. It outputs a sequence of sentence embeddings of variable length L where each embedding is a 384-dimensional vector ($f_{sentence} \in \mathbb{R}^{L \times 384}$). The sequence is averaged over its length L to have a fixed size input for the GRU units of the network using a pooling average procedure. The averaged embedding is given at each computational step to the GRU network.

The audio input is processed by cascaded one-dimensional convolutional layers with batch normalization and leaky ReLU activation. It generates a sequence of embeddings of length 34 where each embedding is a 32-dimensional vector: $f_{audio} \in \mathbb{R}^{34 \times 32}$.

The network receives for its first 4 computational steps the initial pose as an input with an additional bit input $b \in \mathbb{R}$ set to 1 that encodes that the input sequences are initial poses. Afterward, it receives for each generated pose $\hat{p}$ an input pose p where all values are zero. In this case, the bit input is also set to zero.

Finally, the network receives as input a style embedding f_{style} which is learned by a variational network. The speaker's ID is a one-hot vector. A set of fully connected layers maps the ID to a style embedding space $f_{style} \in \mathbb{R}^{34 \times 16}$. The network uses a sampling process similar to VAEs to learn the style mapping.

The GRU network consists of 4 bidirectional GRU layers followed by 2 fully connected layers. The first has the same size as the GRU layer's output and the second half of the GRU layer's output size. It generates an output of $T = 34$ poses where the first four poses are the initial poses.

We trained the proposed network model under various conditions on data of recorded gestures of human speakers from TED talks [35]. The generator network was trained with the help of a discriminator network. During the training, the discriminator network tries to identify if a certain gesture is either from the ground truth or generated. Please see [26] for details about the training procedure of the discriminator network.

Results

The gesture generation has been evaluated under different conditions. See [26] for details and further results. Table 3.1 shows results for networks that use different inputs. The main evaluation metric used is the Fréchet Gesture Distance (FGD), similar to the Fréchet Inception Distance used in image generation. FGD measures the difference between the distributions of real and generated gestures in a latent feature space, which is generated by an autoencoder trained to recreate input poses. The formula for FGD is given by:

$$\mathrm{FGD}(X, \hat{X}) = ||\mu_r - \mu_g||^{1/2} + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}\right) \tag{3.3}$$

where μ_r and Σ_r are the mean and variance of the latent feature distribution Z_r of real human gestures X , and μ_g and Σ_g are the mean and variance of the latent feature distribution Z_g of generated gestures $\hat{X}$.

The Yoon et al. model [34] is a predecessor of our model used as a baseline. It mainly differs in using a GRU network to process the input text, whereas our models use a transformer for this task. We compared models that use either GRU or Transformer layers. Results show that the GRU networks reach generally a better FGD score. Moreover,

[3] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Model	FGD (↓)
Yoon et al. [34]	3.729
Trimodal Model (GRU)	1.44
Audio Based Model (GRU)	1.88
Text Based Model (GRU)	3.52
Trimodal Model (Transformer)	4.10
Audio Based Model (Transformer)	5.04
Text Based Model (Transformer)	7.32

Table 3.1: Model Comparison

we compared the influence of the network inputs. The Trimodal models use as input the text (string) and audio (waveform) of the spoken dialogue, besides the style embedding. The Audio and Text models use only the text (string) or audio (waveform) besides the style. Having both (text and audio) as input provides the best results, but the audio input is more important than the text, as indicated by its better performance.

3.3.3 Gaze

Two alternative controllers for gaze exist. The first is part of the node responsible for the social MPC and controls the direction of the head. The second is an implementation by PAL that controls both the head and the eyes.

Head Control

The head control aligns ARI's face in a specific direction that can either a given direction or a person to look at. The controller can be interfaced via a 'Look at' action sever that takes as input either the ROS4HRI ID of the target person, target group, or a specific position (x, y) in the local or global 2D map. Additionally, it takes a timeout period in seconds when the actions should have ended. The action server returns the head joint errors (pan and tilt) and the final time it took to execute the action. The head is oriented towards the target using a simplified PID controller.

Gaze Manager

The second controller, the gaze manager, is delivered by PAL[4]. It coordinates the robot's eyes together with the head position (Fig. 3.8). It can be interfaced by publishing stamped points (geometry_msgs/PointStamped) to the /look_at ROS topic. It generates a natural-looking gaze behavior by first controlling the eyes to look at the target (by re-publishing the stamped point to the /robot_face/look_at topic). Then, the eyes manager publishes a TF frame called /current_gaze_target corresponding to where the eyes are currently looking at. Finally, the gaze manager monitors this frame and sends accordingly goals to the head controller, which transforms the cartesian look at target into a trajectory in joint space.

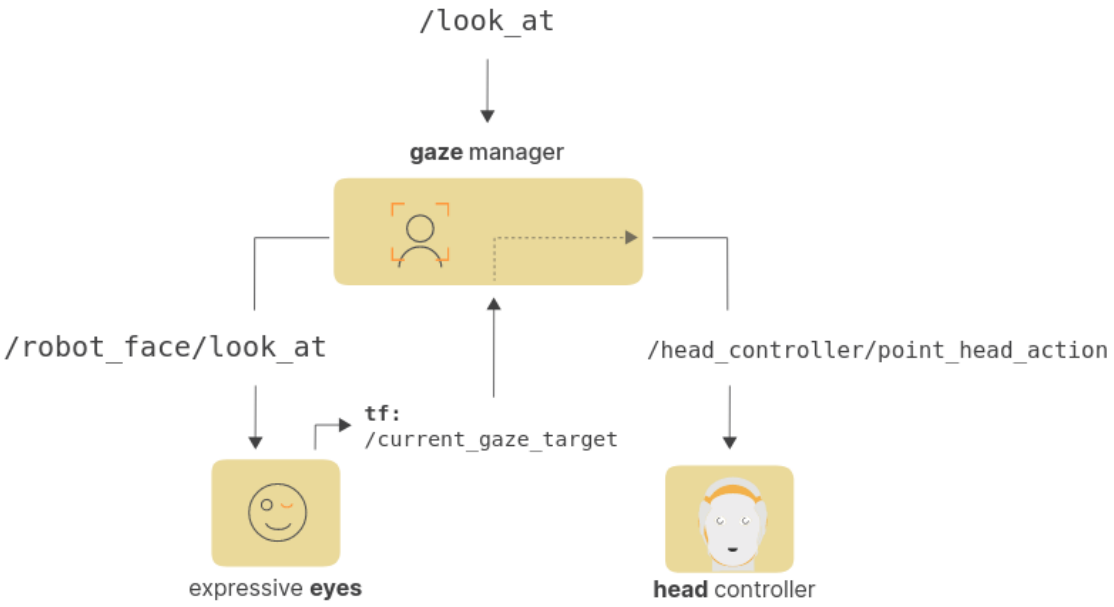


Figure 3.8: Gaze Manager that controls both eyes and head position.

[4] http://docs.pal-robotics.com/ari/sdk/23.1/expressiveness/attention-management.html

4 Non-Verbal Behavior Manager

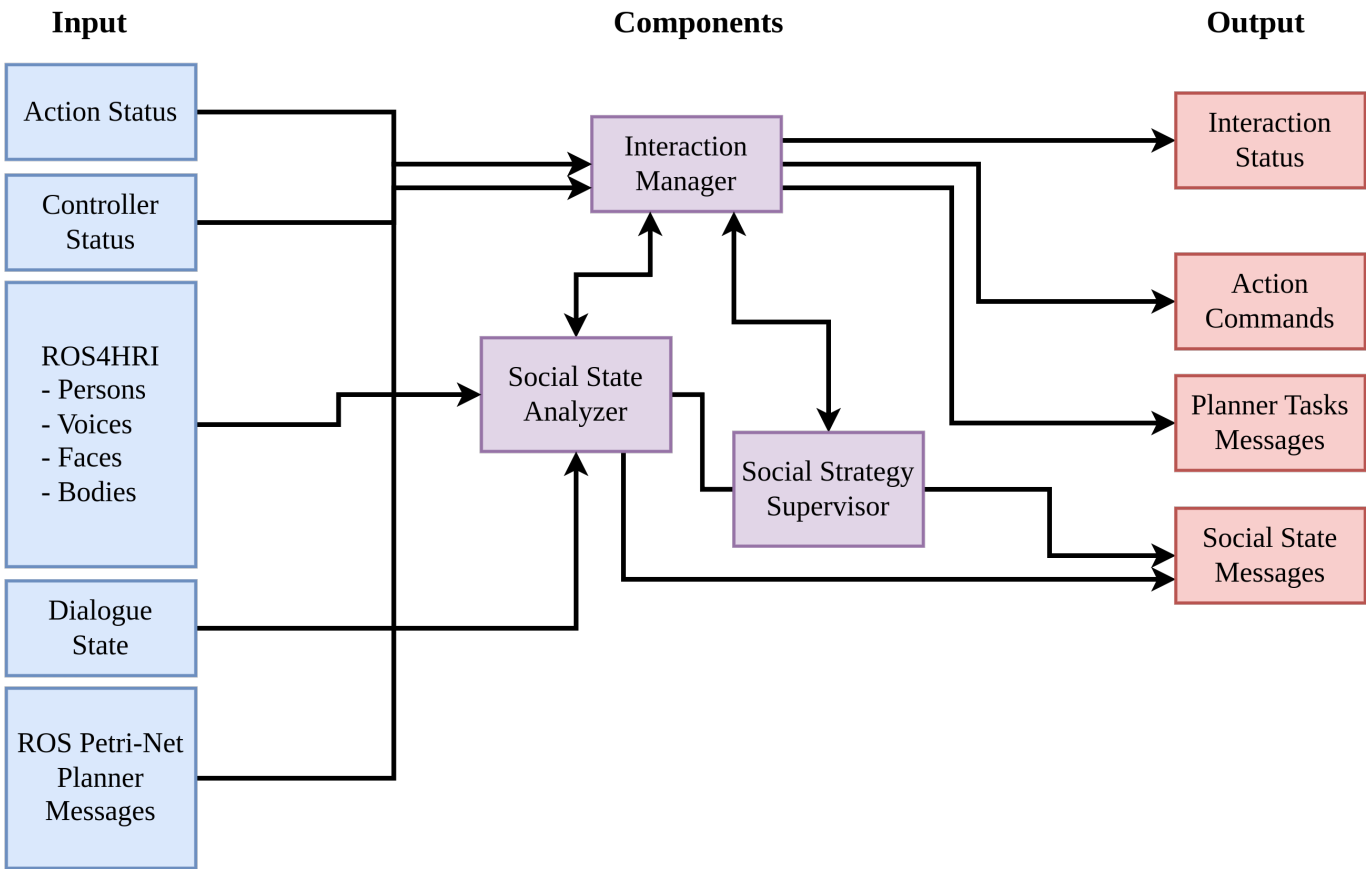


Figure 4.1: Overview of all nodes, topics, and messages in Non-Verbal Behavior Manager

The robot's non-verbal behaviour manager is responsible for deciding appropriate high-level social actions to take and to manage the interactions. The social interactions in SPRING required the behavior system components to interface with the social perception signals, the task planner, and the multi-user conversational manager, to enable situated interactions with multiple users at the same time.

4.1 General Architecture

The robot non-verbal behaviour manager consists of the Interaction Manager and various components for social scene understanding (Fig. 4.1), such as, robot gaze decision control, engagement and necessitude (or willingness) to interact estimation, dialogue states, etc.

The robot non-verbal behaviour manager is in charge of handling the interface between the high-level planner, the conversational system and the robot behaviours. The robot behaviour manager module interfaces with the non-verbal behaviour generation through the Robot Non-verbal Actions servers. The behaviour manager handles high-level interaction decisions and the behaviour generation module controls low-level action execution.

The next section lists the different components and action servers. The following section describes its interface.

Components

Several components are responsible for different aspects of the behavior manager. The components themselves can be either a single ROS node or a combination of several nodes.

- Interaction: Management of the human-robot interactions.
  - Planner Interface: manage communication with the high-level planer.
  - Dialogue Interface: manage communication with the conversational system.
  - Control Interface: manage communication with the robot behvavior controller.
  - Social Interface: manage communication with the social context understanding components.
- Social Context: analyse the social context, and provide high-level decisions of social actions to take.
  - Social State Analyzer: keeps the social state of interaction with the robot.
  - Social Strategy Supervisor: makes decisions on social actions to take during interaction.

4.1.1 Interface

Inputs

List of input data (via ROS Messages):

1. ROS petri-net messages: information regarding the state of the high-level planner tasks execution.
2. ROS4HRI messages: information regarding persons, voices, faces, bodies, and groups (/persons/*, /voices/*, /faces/* /bodies/*, /groups/* ROS topics). This includes their position relative to the robot.
3. Status topics of robot behavior controller and action servers: status and availability of the controller and the gesture and navigation action servers.
4. Dialogues state: messages from the conversational system status, including active conversations, and the output of the Natural Language Processing components.

Outputs

The main output of the module is the commands sent to the low-level behvavior control of the robot to execute an Action. In addition to this, the interaction manager and social context components provide messages about the status of the planner's tasks, the interaction and the social state. List of outputs:

- Action command to robot behavior controller action servers
- Interaction state messages with explanations of tasks status during interaction
- ROS Petri-Net Planner messages for controlling the execution of the high-level planner tasks.
- Social state messages with information of users in an interaction

4.2 Interaction Manager

The Interaction Manager handles the interface between the Robot Behaviour Generation modules (WP6) and the High-level Robot Task Planner and Conversational System (WP5). The Interaction Manager is itself implemented as an abstract controller for the ROS Petri-Net Planner presented in SPRING Deliverable 5.3 [23].

The Interaction Manager provides a number of "Interaction State" messages to allow monitoring the status and execution of the robot's task and plans during an interaction, combining information receive from the task planner, the dialogue arbiter and the robot controller, as well as social input signals from the body and face trackers, and the audio processing nodes.

With the Interaction Manager a Monitoring Application is provided as a web based interface for SPRING researchers to monitor messages from the execution state of the task plans, dialogue state and conversation messages, and the social state. The application also provides researchers with minimal overrides control to select and start/stop a robot's High-level Hierarchical Plans (see Deliverable 5.3 [23]) or start an Interaction/Conversation with users.

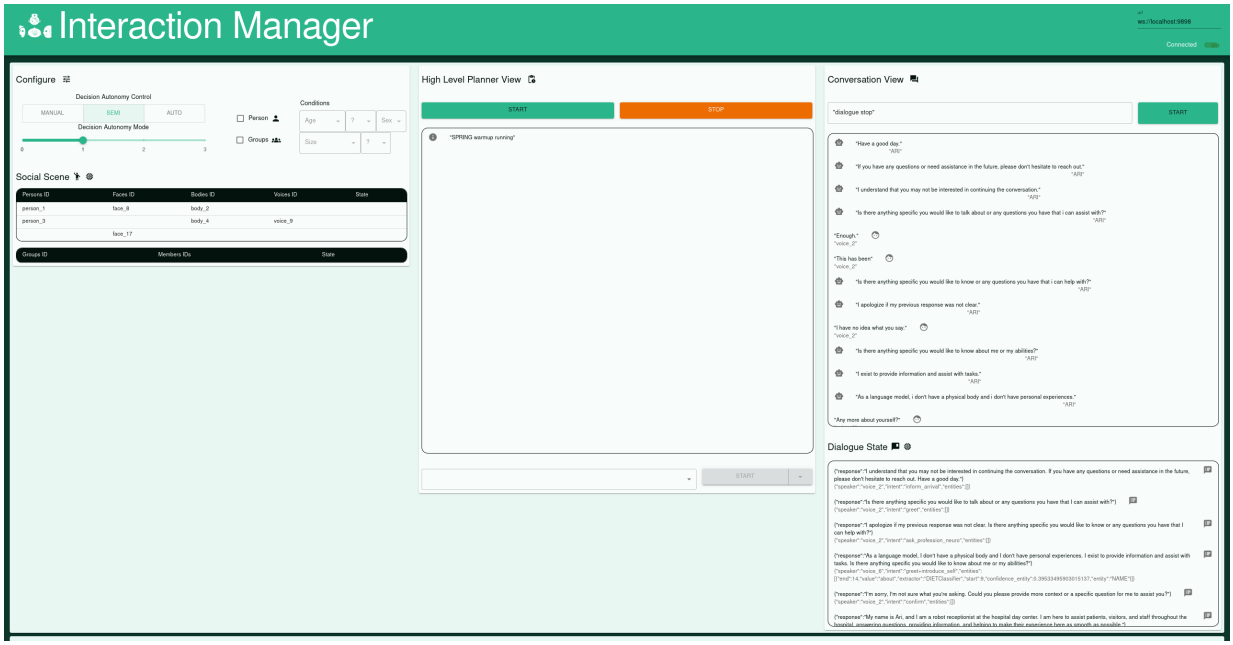


Figure 4.2: Screenshot for the Interaction Manager Monitoring Application. The Social Scene view display recognized humans interacting with the robot. The Conversation View display the state of the conversation dialogue. The Planner View display status messages from the high-level planner action.

High-level Planner Interface

The Interaction Manager, as a ROS Petri-Net controller, can start tasks and keeps track of them. The main functionality of the controller is to manage the currently available and running Petri-Nets and to provide functionality to send and receive information from/to a specific net.

For taking care of the running Petri-Nets plans, the controller interface provides information on the available nets. The other major functionality of the controller interface is to exchange data between the different plans and the social state representations provided by the social scene understanding components. The interaction manager populates the knowledge base for each plan execution according to the messages it receives from the social scene understanding components.

Deliverable [22, 23] presented the functionality of the ROS Petri-Net planner. A special plan 'SPRING warmup', have been develop to launch at the system startup, the social monitoring tasks for the social scene understanding components.

Dialogue State Interface

Both the Interaction Manager and the Conversation Manager are implemented as abstract controllers for the ROS Petri-Net (RPN) Planner presented in SPRING Deliverable 5.3 [23]. Each controller is in charge of its own type of RPN planner servers. In order to communicate with the controllers, we a use a knowledge base implementation. The Dialogue State Interface handles communication between the conversational system and the non-verbal behavior system in order to maintain and synchronize up date knowledge about the dialogue state and the interaction state for each controller. This includes the persons in the conversation/interaction, past dialogue history, etc.

Behaviour Generator Controller Interface

The non-verbal behaviours of the robot in SPRING are implemented as a collection of ROS Actions Servers, as presented in Chapter 3. The Robot Behavior Manager will handle/synchronise the different ROS Actions Servers from the Robot Behavior Generator. These are integrated through ROS [29] action server/client interface, as shown in Fig 2.2, for each robot non-verbal action/plan.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

Social State Interface

Thought the interface with the social scene understanding components the interaction manager is populates and maintains the planner's knowledge base with information about the interaction and social state, persons engage in interaction/conversation with the robot, etc.

4.3 Social Scene Understanding

In the SPRING project we need to be able to understand various individual and group situations and take appropriate decisions, e.g. identify persons that need assistance, engage in face-to-face multi-modal dialogue with a patient, a family member, a staff member, or with a party of them, etc.

Among the set of "social decisions" that are require for the SPRING robot non-verbal behaviour system we have 'Detect people arrival and departure', 'determines a person in the scene wants or requires the robot attention', 'decide when to go, start an approach or guidance action, adapt to persons in scene', 'decide who to look at, switch focus of attention during multi-party interactions', etc.

The SPRING robot needs the ability to track and ascribe social meaning to its sensory information. It must explore the environment and understand what the environment affords, including which objects, actions, events, and scene information can be extracted from the sensory data.

4.3.1 Who is taking part in the interaction?

The social scene understanding components are tasked with turning the continuous stream of messages produced by the low-level input and output components into discrete representations to describe multi-party interactions, devise social interaction plans, and support the high level planner and the conversational manager for maximizing the robot's execution strategies for social interaction and communication.

It must track the state of each agent, and track their conversations, determining what they are saying and to whom, where their attention is at, as well as predicting their goals, and their affective and emotional states, etc. In [7] we propose the representation of the social state into different domains, such as the the dialogue domain and and the behaviour domain.

Social State Modelling

The representation of the social state models the behaviour of the users and is a representation of the people interacting in the scene, which combines persistent data of the user for identification from the ROS4HRI person recognition topics tracking faces, bodies, and voices from the Audio/Visual data streams of the ARI robot.

Dialogue State Tracking

The model of the dialogue state is a representation of the conversation, tracking what has been said and by whom, the intents expressed, the entities mentioned, and the topics of conversation during interactions among multiple users and the robot.

4.3.2 When to start/stop an interaction?

Willingness to Interact

To equip social robots with the ability to detect a person who is willing to interact, a basic decision making system is implemented following the work of [18]. It is able to decide if a person, in the field of view of the robot, is looking to initiate a conversation with the robot.

The decision-making network in [18] uses nonverbal bodily social signals to determine engagement in human-human interactions, with a CNN in its decision-making process. The architecture (see Figure 4.3), comprises 9 layers in total, excluding the input layer. The first convolutional layer takes as input grayscale images from the robot cameras, resized to 198 × 198 pixels, and applies 16 filters 9 × 9 with stride 3. The second and the third layer apply 32 and 64 filters 5 × 5 respectively, with stride 1. Each convolutional layer is followed by a rectifier linear unit (ReLU) function and a max pooling layer with pool size 2 × 2 and stride 2. The last convolutional-max pooling layer is followed by a fully-connected layer of 256 units. The final output layer is a fully-connected linear layer of 3 units, giving as output the probabilities for each of the 3 available actions to be the right one to be chosen in the current situation. The choice of the network is the action associated with the label having the highest probability for the input image.

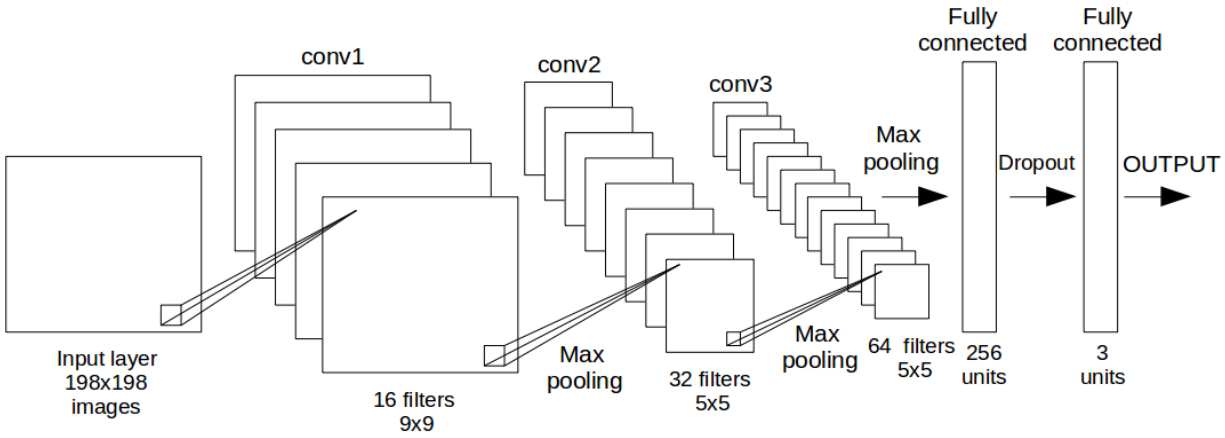


Figure 4.3: Architecture of the CNN (from [18]): 3 convolutional layers, each followed by a corresponding max pooling layer, and 2 fully-connected layers. The first convolutional layer applies 16 filters 9x9 with stride 3. The second and third apply 32 and 64 5x5 filters respectively, with stride 1. The last convolutional layer is followed by a dense layer of 256 units, after which a dropout layer is added. After applying the dropout, the output layer is a fully-connected linear layer of 3 units.

Engagement Estimation

Recognizing the level of engagement of humans during the interactions is an important capability for social robots. We implemented for the SPRING robot an engagement estimator build on the work of [6]. They proposed a regression model, utilizing CNN and LSTM networks, enabling the robot to compute a single scalar engagement estimator during interactions with humans from standard video streams, obtained from the point of view of an interacting robot. Their network architecture, depicted in Figure 4.4, is composed of two main modules: a convolutional module which extracts frame-wise image features and a recurrent module that aggregates the frame features over a time to produce a temporal feature vector of the scene.

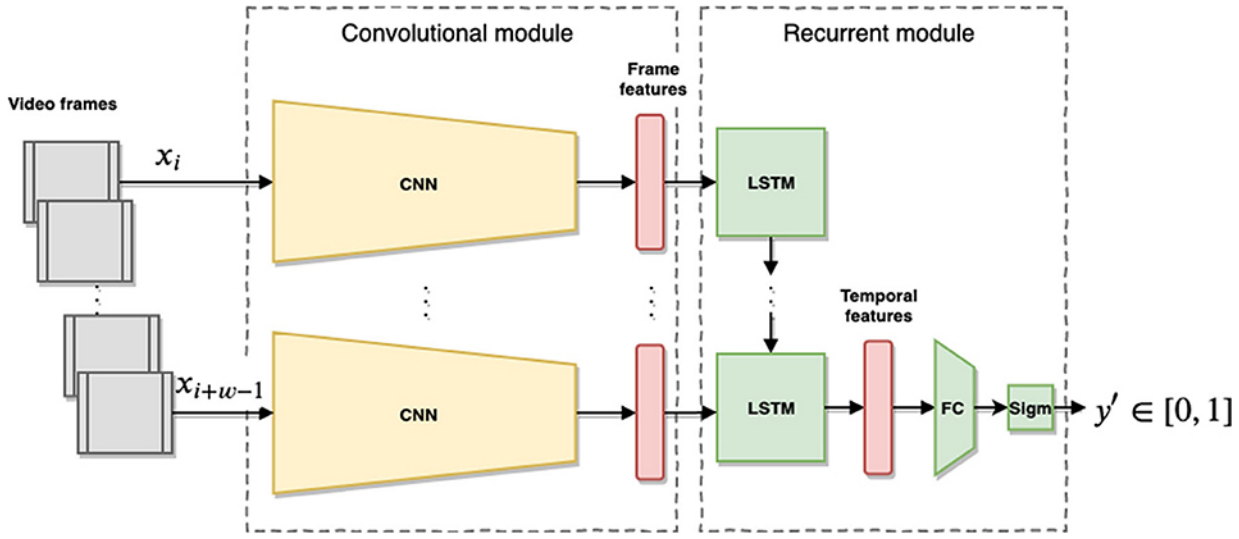


Figure 4.4: Overview of the model proposed in . The input is a video stream of interactions between the robot and humans collected in w size intervals. The frames xi are passed through the pre-trained CNN (ResNet) producing a per-frame feature vector which is then passed sequentially to the LSTM network. After w steps the LSTM produces a temporal feature vector which is passed to a FC layer with sigmoid activation to produce an engagement value y for the temporal window.

4.3.3 Whom is being talked to?

Addressee Detection

[13] developed a hybrid deep learning model composed of convolutional layers and LSTM cells for understanding to whom a speaker is addressing an utterance, by developing a deep learning hybrid model (CNN+LSTM) taking as input two visual information: the speaker's face and body pose. The model takes as input images portraying the face of the speaker and 2D vectors of the speaker's body posture and provides a Addressee Estimation in terms of addressee localisation in space, from a robot ego-centric point of view.

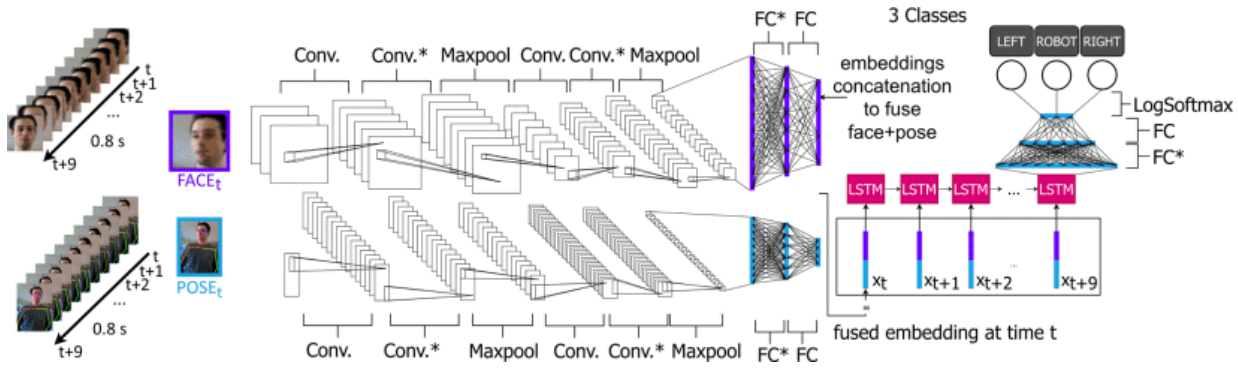


Figure 4.5: Illustration of the deep neural network for addressee estimation developed in [13]. Face images and body pose vectors are passed separately to two blocks of convolution, each including two 2d convolutional and one max-pooling layers. Then, the two embeddings resulting from fully connected layers are concatenated and sequences of 10 fused embeddings are passed to the lstm layer. The output is provided after two others fully connected layers and a logsoftmax layer. * represents leakyrelu activation function.

We will follow the approach presented in [13], taking advantage of the body and face tracking recognition signal from the other SPRING modules, to give our robot the ability to understand an utterance's addressee, by interpreting and exploiting non-verbal bodily cues from the speaker.

4.3.4 Where to look at during an interaction?

Gaze Target Decision Control

Another skill necessary for maintaining multi-party conversations is exhibiting natural, human-like gaze behavior. We take inspiration from the planning-based Gaze Control Systems (GCS) for HRI proposed in [14] to automate the gaze behavior of social robots. The model produces gaze behavior that is dynamic and differs in frequency and duration based on the state of the conversation by planning the priority for each potential gaze target (e.g., users or objects) in the environment incrementally (frame-by-frame) for a future rolling time window. We implemented a gaze decision strategy following this approach, combining information about the dialogue state with speaker information form the Audio/Visual signal coming from the voice, face and body signal topics with the ROS petri-net plan to sent gaze targets to the Robot Behavior generator gaze controller.

4.3.5 Knowing when to perform communicative actions during interaction

Conversation Gesture Selection

For effective socially situated interactions the SPRING robot must also 'know' when to perform communicative actions. We are using Large Language Models [36] to generate appropriate 'high-level' arm and head gestures to accompany the robot verbal responses to the user requests. This 'gesture commands' are sent as t the non-verbal behavior gesture generation action servers.

This project has received funding from the European Union's Horizon 2020
Research and Innovation Programme under Grant Agreement No. 871245.

5 Outputs

The software modules describe in this deliverable will be made available on the SPRING project Gitlab repositories for Work Package 5 [27] and Work Package 6 [28]. These will be available to the public for the duration specified in the SPRING project proposal.

Software repositories for the robot behaviour generator modules, described in Chapter 3 can be found on [28]. Software repositories for the robot behaviour manager modules, described in Chapter 4 can be found on [27].

As per European Commission requirements, the repositories will be available to the public for a duration of at least four years after the end of the SPRING project. People can request access to the software to the project coordinator at `spring-coord@inria.fr`. The software packages use ROS (Robotics Operating System) [29] to communicate with each other and with the modules developed in the other workpackages.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

Bibliography

[1] Leemon C Baird. Advantage updating. Technical report, Technical report wl-tr-93-1146, Wright Patterson AFB OH, 1993.

[2] Anand Ballou, Xavier Alameda-Pineda, and Chris Reinke. Variational meta reinforcement learning for social robotics. *Applied Intelligence*, pages 1–20, 2023.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[4] Eduardo F Camacho and Carlos Bordons. *Model predictive control*. Springer, 2007.

[5] Stephen Dankwa and Wenfeng Zheng. Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd international conference on vision, image and signal processing*, pages 1–5, 2019.

[6] Francesco Del Duchetto, Paul Baxter, and Marc Hanheide. Are you still with me? continuous engagement assessment from a robot's point of view. *Frontiers in Robotics and AI*, 7, 2020.

[7] Daniel Hernández García, Yanchao Yu, Weronika Sieinska, Jose L. Part, Nancie Gunson, Oliver Lemon, and Christian Dondrup. Explainable representations of the social state: A model for social human-robot interactions. *CoRR*, abs/2010.04570, 2020.

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[9] Edward Hall. The hidden dimension: man's use of space in public andprivate, 1969.

[10] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[11] Adam Kendon. *Conducting interaction: Patterns of behavior in focused encounters*, volume 7. CUP Archive, 1990.

[12] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[13] Carlo Mazzola, Marta Romeo, Francesco Rea, Alessandra Sciutti, and Angelo Cangelosi. To whom are you talking? a deep learning model to endow social robots with addressee estimation skills. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2023.

[14] Chinmaya Mishra and Gabriel Skantze. Knowing where to look: A planning-based architecture to automate the gaze behavior of social robots. In *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1201–1208, 2022.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[17] Claudio Pedica and Hannes Vilhjálmsson. Social perception and steering for online avatars. In *International Workshop on Intelligent Virtual Agents*, pages 104–116. Springer, 2008.

[18] Marta Romeo, Daniel Hernandez Garcia, Ray Jones, and Angelo Cangelosi. Deploying a deep learning agent for hri with potential "end-users" at multiple sheltered housing sites. In *7th International Conference on Human-Agent Interaction*, July 2019. 7th International Conference on Human-Agent Interaction, HAI 2019 ; Conference date: 06-10-2019 Through 10-10-2019.

[19] James A Sethian. A fast marching level set method for monotonically advancing fronts. *proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[20] James A Sethian. Fast marching methods. *SIAM review*, 41(2):199–235, 1999.

[21] Francesco Setti, Chris Russell, Chiara Bassetti, and Marco Cristani. F-formation detection: Individuating free-standing conversational groups in images. *PloS one*, 10(5):e0123783, 2015.

[22] SPRING Project. D5.1: Initial high-level task planner and conversational system prototype for realistic environments. https://spring-h2020.eu/wp-content/uploads/2021/06/SPRING_D5.1_Initial_High-level_Task_Planner_and_Conversational_System_Prototype_for_Realistic_Environments_vFinal_31.05.2021.pdf.

[23] SPRING Project. D5.3: High-level task planner in relevant environments. https://spring-h2020.eu/results/.

[24] SPRING Project. D6.1: Neural network architecture specification and design. https://spring-h2020.eu/wp-content/uploads/2021/03/SPRING_D6.1_Neural-network-architecture-specification-and-design_VFinal_24.02.2021.pdf.

[25] SPRING Project. D6.4: Software for generating multi-party situated interactions. https://spring-h2020.eu/wp-content/uploads/2023/07/SPRING_D6.4_Software_for_generating_multi_party_situated_interactions_VFinal_16.06.2022.pdf.

[26] SPRING Project. D6.5: Final neural network architectures. https://spring-h2020.eu/wp-content/uploads/2023/07/SPRING_D6.5_Final_neural_network_architectures_VFinal_08.11.2022.pdf.

[27] SPRING Project. Wp5: Spoken conversations repository. https://gitlab.inria.fr/spring/wp5_spoken_conversations.

[28] SPRING Project. Wp6: Robot behavior repository. https://gitlab.inria.fr/spring/wp6_robot_behavior.

[29] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. https://www.ros.org.

[30] Xuan-Tung Truong and Trung-Dung Ngo. Dynamic social zone based mobile robot navigation for human comfortable safety in social environments. *International Journal of Social Robotics*, 8(5):663–684, 2016.

[31] Xuan-Tung Truong and Trung-Dung Ngo. "to approach humans?": A unified framework for approaching pose prediction and socially aware robot navigation. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):557–572, 2017.

[32] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788, 2020.

[33] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[34] Youngwoo Yoon, Bok Cha, Joo-Haeng Lee, Minsu Jang, Jaeyeon Lee, Jaehong Kim, and Geehyuk Lee. Speech gesture generation from the trimodal context of text, audio, and speaker identity. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[35] Youngwoo Yoon, Woo-Ri Ko, Minsu Jang, Jaeyeon Lee, Jaehong Kim, and Geehyuk Lee. Robots learn social skills: End-to-end learning of co-speech gesture generation for humanoid robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4303–4309. IEEE, 2019.

[36] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.