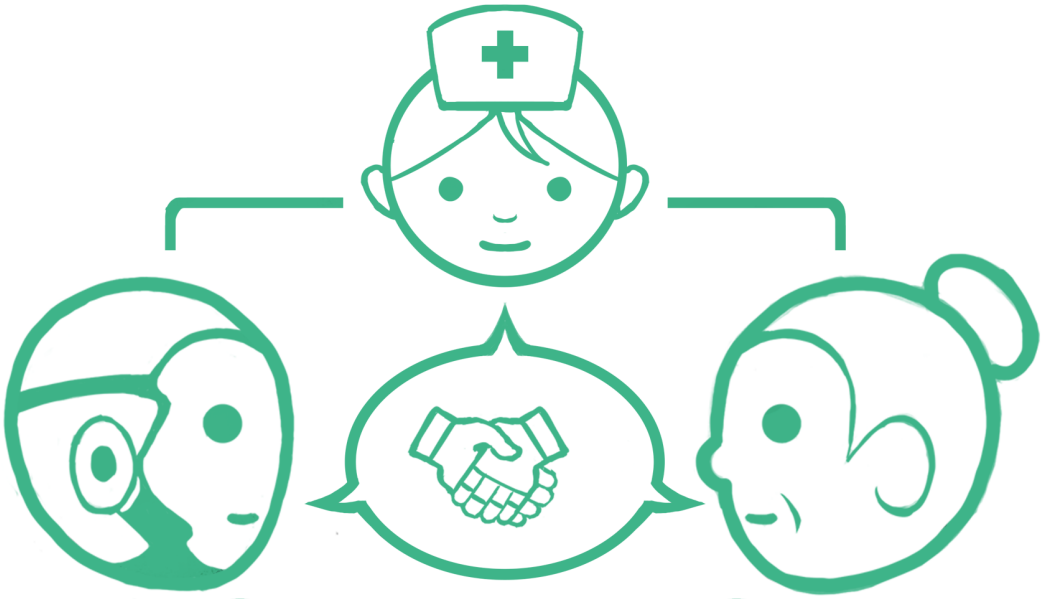

Deliverable D6.4: Software for Generating Multi-Party Situated Interactions

Due Date: v1.1 - 16/06/2022

Main Author: Chris Reinke (INRIA), Daniel Hernandez Garcia (HWU)

Contributors: Alex Auternaud (INRIA), Timothee Wintz (INRIA), Oliver Lemon (HWU), Nancie Gunson (HWU), Sara Cooper (PAL)

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	D6.4: Software for Generating Multi-Party Situated Interactions
Responsible Partner	INRIA
Work Package	WP6: Learning Robot Behaviour
Task	T6.2: Generation of Multi-party Situated Interactions
Version & Date	v1.1 - 16/06/2022
Dissemination	Public Deliverable

CONTRIBUTORS AND HISTORY

Version	Editor	Date	Change Log
1.0	INRIA, HWU, PAL	10/06/2022	
1.1	Chris Reinke (INRIA)	16/06/2022	Additional information in chapter executive summary. Added conclusion chapter. Minor format changes.

APPROVALS

Authors/editors	INRIA: Chris Reinke, Alex Auternaud, Timothee Wintz; HWU: Daniel Hernandez Garcia, Oliver Lemon, Nancie Gunson; PAL: Sara Cooper
Task Leader	INRIA
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

Abbreviations 4

Executive Summary 5

1 Introduction 6

2 3D Physical Robot Simulation 7

3 Multi-Party Interaction Simulation 9
- 3.1 Overview . . . 9
- 3.2 Architecture . . . 9
- 3.3 Human Agents . . . 11
  - 3.3.1 Movement and Navigation . . . 11
  - 3.3.2 Gaze Behavior . . . 12
  - 3.3.3 Speech . . . 13
- 3.4 Human Group Behavior . . . 13
  - 3.4.1 GroupNavigation Script . . . 14
  - 3.4.2 Simulation Results . . . 15
- 3.5 ARI Agent . . . 17
  - 3.5.1 Environment Mapping . . . 17
  - 3.5.2 Tracking of Humans and Groups . . . 18
  - 3.5.3 Detection of Faces and Related Human Properties . . . 18
  - 3.5.4 Speech Detection . . . 20
  - 3.5.5 Movement and Navigation . . . 20
- 3.6 GUI . . . 21
- 3.7 Scripting Interaction Scenarios . . . 22

4 Conversational Content Generator 25
- 4.1 Overview . . . 25
- 4.2 Dialogue Simulation Design . . . 25
- 4.3 Architecture . . . 26

5 Conclusions 27

Bibliography 28

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

Abbreviations

Abbreviation	Meaning
ABUS	Agenda-Based User Simulation
ARI	Social assistive robot used by the SPRING project
CCG	Conversational Content Generator
DM	Dialogue Manager
DOF	Degree Of Freedom
GCFF	Graph-Cuts for F-Formation algorithm
GUI	Graphical User Interface
HWU	Heriot-Watt University (SPRING Partner)
INRIA	Institut National de Recherche en sciences et technologies du numérique (SPRING Partner)
MPISim	Multi-Party Interaction Simulator
MUSE	Multi-User Simulation Environment
NLG	Natural Language Generation
NLU	Natural Language Understanding
NUS	Neural User Simulator
PAL	PAL Robotics (SPRING Partner)
ROS	Robot Operating System
SPRING	Socially Pertinent Robots in Gerontological Healthcare
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

This deliverable reports the results of SPRING task T6.2: Generation of multi-party situated interactions. The goal of the task is to provide a simulation environment for interactions between ARI, the robot used by the SPRING project, and human agents. The simulation allows to generate plausible human behavior (for example gaze), the synthesis of high-level sensor perception (for example the perception of emotions), the high-level motion control of ARI (for example to join a group of human agents), and the simulation of conversations between ARI and humans. The aim of the simulation is to support the tasks of SPRING work packages WP 5 (Multi-User Spoken Conversations with Robots) and WP 6 (Learning Robot Behaviour) by allowing the training and testing of their modules in a simulation environment to reduce the amount of time intensive real world experiments. The developed software consists of three modules: 1) a 3D Physical Robot Simulation[1] used for the realistic simulation of ARI, 2) a Multi-Party Interaction Simulation[2] for abstract simulations of ARI and its physical and verbal interaction with humans, and 3) a Conversational Content Generator[3] that simulates realistic dialogues between ARI and humans. The software is open access to the public at least until 4 years after the project ends.

[1] http://wiki.ros.org/Robots/ARI
[2] https://gitlab.inria.fr/spring/wp6_robot_behavior/multiparty_interaction_simulator
[3] https://gitlab.inria.fr/spring/wp5_spoken_conversations/conversational-user-bot-simulator

1 Introduction

The development of software for generating multi-party situated interactions is the goal of SPRING task T6.2 and part of WP 6 (Robot Learning Behavior). The goal of the software is to synthesize data that can be used for the training and evaluation of the machine learning architectures developed in T6.1 (Deep Architectures for Conversational System and Non-verbal Behaviour) [28] and for the conversational and high-level robot task manager of WP 5 (Multi-User Spoken Conversations with Robots) [26, 27]. The software was developed based on the specifications formulated in SPRING Deliverable D6.2 (Specifications of the generator of situated interactions) [29]. The software is composed of three modules:

1. **3D Physical Robot Simulation**: The simulation provides a realistic physical 3D model of SPRING's ARI[4] robot [2] to have a testbed for low level controller and perception modules.
2. **Multi-Party Interaction Simulation**: An abstract simulation of ARI and humans in a 2D environment to simulate physical and verbal interaction scenarios between them.
3. **Conversational Content Generator**: A generator of realistic dialogues between humans and ARI.

These modules are introduced in the next chapters individually.

Figure 1.1: The ARI robotic platform used for the SPRING project. Figure taken from [2]

[4] https://pal-robotics.com/robots/ari/

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

2 3D Physical Robot Simulation

A detailed physical 3D model of ARI was developed for the Gazebo[5] simulator by PAL. It models ARI's body and most of its actuators and sensors. Gazebo is compatible with ROS[6] and all components of the model use the same communication interface as is used on the ARI robot. The model allows to test motion controllers and sensor modules developed during the SPRING project. The simulation and its documentation can be found at and is at least open access to the public until 4 years after the project ends:

`http://wiki.ros.org/Robots/ARI`

The 3D model (Fig. 2.1) incorporates the following features:

- realistic model of ARI's 3D body
- actuators:
  - head controller (2 DOF[7]: pan, tilt)
  - left arm controller (4 DOF: shoulder, elbow)
  - right arm controller (4 DOF: shoulder, elbow)
  - left hand controller (1 DOF)
  - right hand controller (1 DOF)
  - mobile base controller (2 DOF)
- sensors:
  - head camera (Sony 8 MegaPixel RGB)
  - torso front camera (Intel RealSense D435i, RGBD + IMU)
  - torso back camera (Intel RealSense T265, RGBD + IMU)
  - front RGB-fisheye camera (ELP camera)
  - rear RGB-fisheye camera (ELP camera)
  - microphone-array

A set of Gazebo worlds that can be used as simulation environments for the ARI robot are publicly available at:

`https://github.com/pal-robotics/pal_gazebo_worlds`

Tutorials to learn how to use and manipulate the simulated model of the robot can be found at:

`http://wiki.ros.org/Robots/ARI/Tutorials`

The tutorials are used as a basis to start learning to program for ARI inside the SPRING project.

[5] `https://gazebosim.org`
[6] `https://ros.org/`
[7] DOF : Degree of freedom

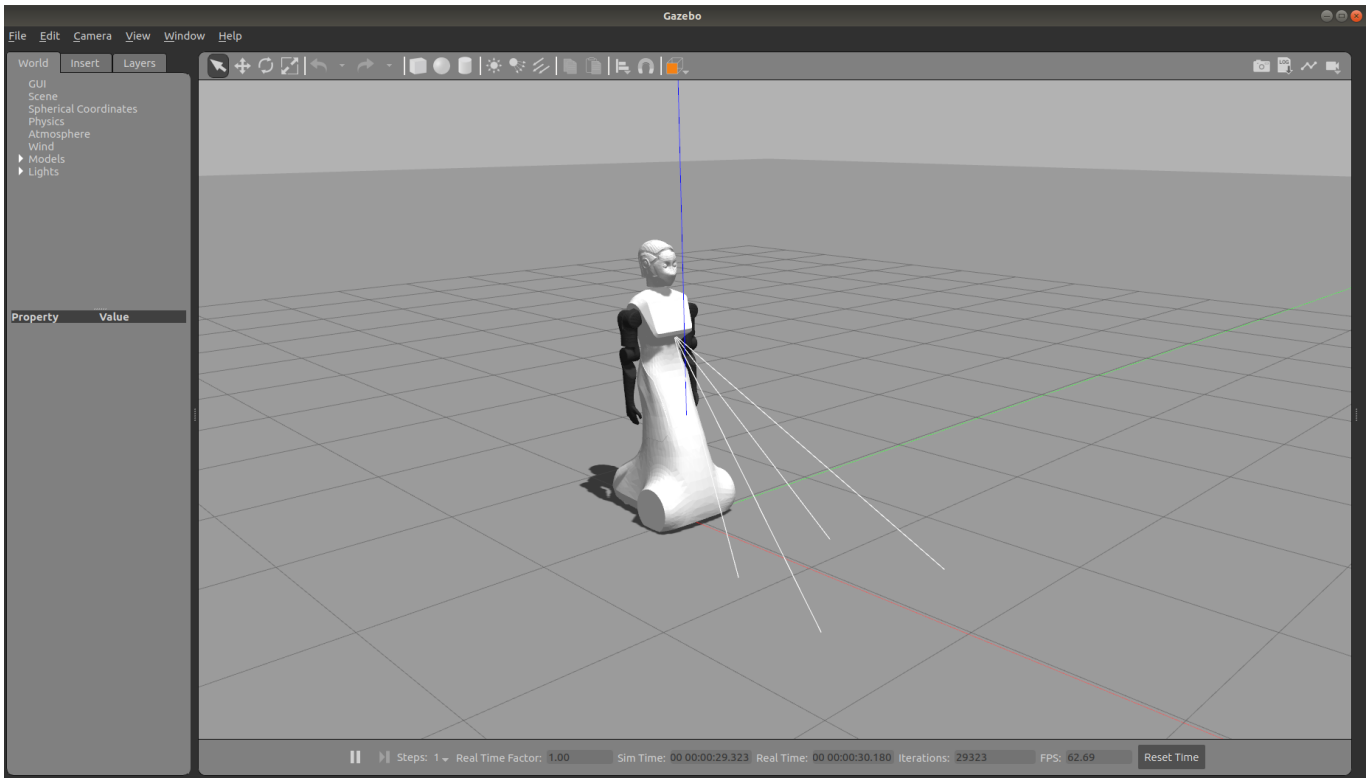

Figure 2.1: Simulation of ARI in the Gazebo environment. The white lines depict the field of vision of the forward positioned RGB-D camera.

3 Multi-Party Interaction Simulation

3.1 Overview

The multi-party interaction simulation (MPISim) is a high-level, abstract simulation of ARI and its environment including humans. It is developed by INRIA. Its purpose is to provide a simulation environment to train and test the high-level decision and behavior components of the SPRING project. It abstracts the environment using a 2D simulator. This reduces the complexity of creating interaction scenarios and having an improved computational performance compared to using the 3D simulation of ARI in Gazebo (Sec. 2). The relevant properties of ARIs environment and their abstract representations which form the input to higher decision and behavior components can be simulated. These include:

- position and orientation of ARI, humans, groups, and other objects such as walls, chairs, ...
- soft-biometric patterns of humans, such as age, gender, ...
- if humans wear a face mask or not
- gaze behavior of humans
- speech

The simulator is designed to be modular and to be easily extensible. All simulation elements (ARI, humans, objects) and the components that define their behavior (movement, gaze, speech, ...) can be easily added, removed, or changed. The code and documentation of the MPISim can be found at and is open access to the public at least until 4 years after the project ends:

`https://gitlab.inria.fr/spring/wp6_robot_behavior/multiparty_interaction_simulator`

The next section describes the general architecture of the MPISim. It is followed by sections that introduce the individual simulator elements: humans, human groups, ARI, the graphical interface (GUI), and how to script the behavior of the simulation.

3.2 Architecture

The simulator is an agent-based [14] 2D simulation developed for Python (Fig. 3.1). The physical simulation is done with PyBox2D[8] which is based on the PyGame[9] framework. It simulates several agents (ARI and humans) and objects. The behavior of agents and objects can be controlled via Scripts. Agents and objects have different properties and Components. The Components are used to control agents, generate their behavior such as gaze, and how they sense their environment. Components can be added in a modular way. Scripts control the simulation and give agents commands which they execute. Additionally, the simulation can have background Processes, for example, the visualization of the simulation. The next sections introduce these different elements: Simulation, Objects, Components, Scripts, and Processes.

Simulation The simulation provides the top layer interface to the simulator. It holds all other elements: Objects with Components, Scripts, and Processes. The simulation provides the functionality to run and reset a simulation. The physical simulation of 2D bodies including their movement and collisions is done via the PyBox2D framework.

A simulation runs in discrete steps. During each step the simulation calls a `step` method of each element of the simulation so that it can advance. First, the `step` method of Objects and their Generator Components are called.

[8] `https://github.com/pybox2d/pybox2d`
[9] `https://www.pygame.org`

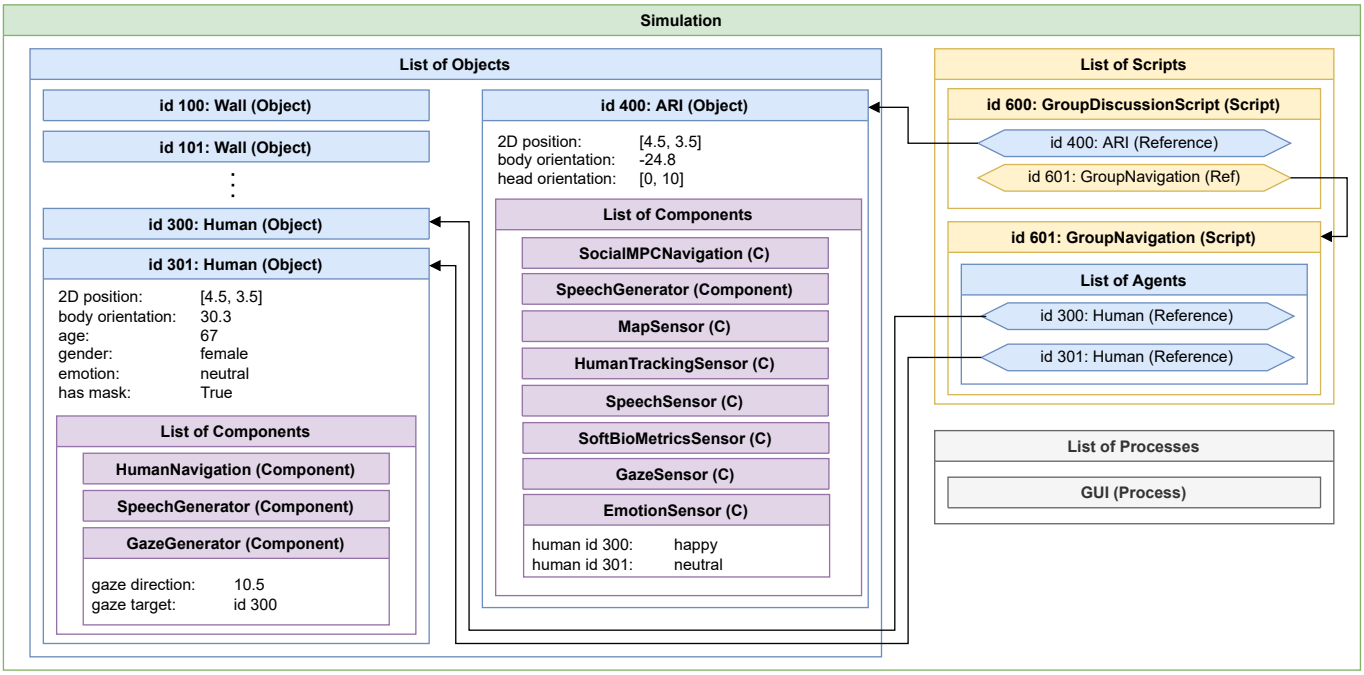

Figure 3.1: Architecture of the MPISim. The Simulation holds a list of Objects that represent physical entities such as walls, humans, and ARI. It also has a list of Scripts that define the behavior of the objects and a list of Processes such as the GUI.

These are responsible to generate the behavior, for example, to generate the movement in humans. Next, the Sensor Components of Objects are called which are usually responsible to sense properties of other Objects, for example, the ARI agent tracking the position of human agents around it. Finally the `step` method of Scripts and Processes are executed.

Objects An Object O represents a physical entity in the simulated environment. These include ARI, humans, walls, obstacles and other objects. Each Object has an unique identifier $i \in \mathbb{N}^+$. Each Object is connected to one or several PyBox2D *Bodies* $B = \{b_1, \ldots, b_m\}$ used to simulate their physical properties, such as mass and collision behaviors. Objects also have a list of connected Components $C = \{c_1, \ldots, c_n\}$. Components are modules that provide functionalities to an Object, such as producing gaze behavior or emotions for humans.

Components A Component C represents the functional elements of an Object. For ARI, these are, for example, its navigation control and its sensors such as the detector for soft-biometric patterns (age, gender). For humans, these are, for example, their movement control and their property generators such as their gaze behavior. Components are designed to be modular, so that they can be added and removed from Objects. This allows to create simulations where only the necessary parts are modelled. For example, a specific simulation might not require the gaze behavior of humans, thus all humans can be created without the Component for gaze generation. This modular philosophy also allows to easily create and add new Components, for example, an improved version of the gaze behavior to produce more natural gaze behaviors.

Scripts A Script is used to control the behavior of the simulation, including its Objects (ARI, humans, obstacles) and possibly other Scripts. Each Script has an unique identifier $i \in \mathbb{N}$. Usually Scripts also have a list of Objects which they control, for example, human agents that have a group conversation. Scripts can create Objects or remove them from the simulation. They can also create subscripts, which control, for example, only a subgroup of humans. Section 3.7 explains in more detail how Scripts are used to control scenarios. The control of the position and orientation of humans in a conversation group are also done via a special *GroupNavigation* Script, which is further explained in Section 3.4.

Processes Processes represent background processes. For example, the GUI (Sec. 3.6) is a Process. They can also be used to provide shared computational functions over several other entities, for example, to create a single map of the environment that is used for the navigation by all robotic agents without recreating such a map for each agent. For this purpose, a single Process is created that generates such a map which is then accessed by all agents.

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

3.3 Human Agents

Human agents are simulated as Objects. Each human H has the following properties:

- a unique identifier: $i \in \mathbb{N}^+$
- 2D body position: $p \in \mathbb{R}^2$
- body orientation: $\theta \in [-\pi, \pi]$
- soft-biometric patterns: age ($\mathbb{R}^+$), gender ({female, male, non-identified})
- does the person wear a mask: {true, false}
- emotions: {neutral, happy, angry, disturbed}
- parameters for the movement control:
  - step length: $\eta \in \mathbb{R}^+$
  - goal distance threshold: $\Delta_G \in \mathbb{R}^+$
  - personal distance: $\Delta_p \in \mathbb{R}^+$

Depending on the requirements additional properties can be assigned to humans (or Objects in general), such as their role (doctor, patient, ...).

The movement of people can be controlled by different methods. Moreover, different Components exist that can be added to a human agent to define specific behaviors. The following sections introduce the movement control and the Component for the gaze behavior generation.

3.3.1 Movement and Navigation

The movement behavior, i.e. the position and body orientation, of a human agent can be controlled via different methods:

- defining directly its position p and orientation θ
- defining a linear velocity on the position $v_p \in \mathbb{R}^2$ and orientation $v_\theta \in \mathbb{R}$
- the *HumanNavigation* Component where a goal position $p_G \in \mathbb{R}^2$ and orientation $\theta_G \in [-\pi, \pi]$ that should be reached can be defined
- the *GroupNavigation* Script (Sec. 3.4) to control the agent as part of a conversational group

The *HumanNavigation* Component allows to define a goal position and orientation. It uses a force model that is similar to the social force model [15] used for the group navigation (Sec. 3.4). It computes forces that change the position (f^H) and orientation (d^H) of a human H. The force on the position f^H is given by:

$$f^H = f^H_{\text{goal}} + f^H_{\text{repulsion}} \tag{3.1}$$

where the goal force f^H_{goal} pushes towards the goal position and the repulsion force $f^H_{\text{repulsion}}$ creates a repulsion to other nearby agents to avoid collisions with them. The force on the orientation d^H orientates the agent towards its movement direction until the goal position is reached. Then, it lets the agent orientate towards the goal orientation. It is defined by:

$$d^H = \begin{cases} \theta^H - \theta^H_{\text{move}} & \text{if } ||p^H - p_G|| > \Delta_G \\ \theta^H - \theta_G & \text{otherwise} \end{cases} \tag{3.2}$$

where θ^H is the current orientation of agent H, $\theta^H_{\text{move}} \in [-\pi, \pi]$ is the current movement direction and $\Delta_G \in \mathbb{R}^+$ is a threshold that defines when the agent reached the goal position. The forces are used to update the position of the human and its orientation by:

$$p^H_{t+1} = p^H_t + \eta f^H \tag{3.3}$$

$$\theta^H_{t+1} = \theta^H_t + \eta d^H \tag{3.4}$$

where $\eta \in \mathbb{R}^+$ is a positive step length. The next sections describe the goal and repulsion force on the agents position in detail. The index H of the human agent for which the forces are calculated is dropped for convenience.

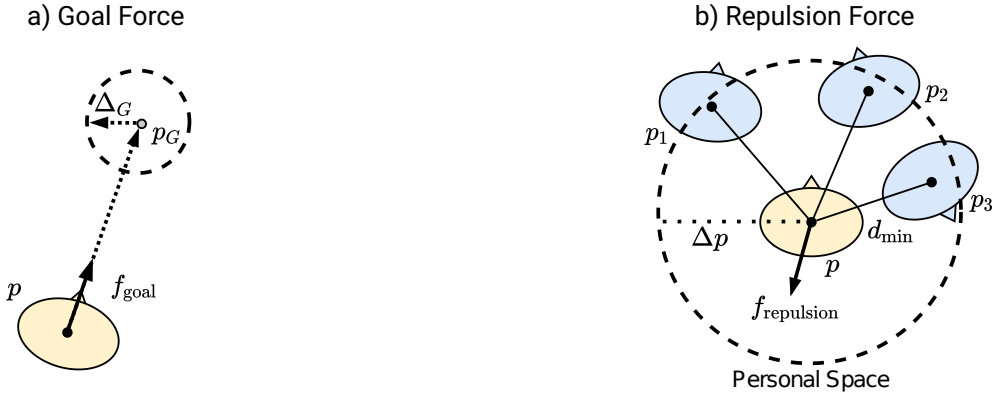

Figure 3.2: The goal force (a) attracts the human agent towards the goal position. The repulsion force (b) repels the agent from nearby agents that are too close.

Goal Force: The goal force f_{goal} (Fig. 3.2, a) drives the human agent towards its goal position p_G:

$$f_{\text{goal}} = \min\left(1, \frac{||p_G - p||}{\Delta_G}\right) \frac{p_G - p}{||p_G - p||} \tag{3.5}$$

The force is modulated by its distance to the goal position ($||p_G - p||$) and a threshold $\Delta_G \in \mathbb{R}$. If the distance to the goal is larger than Δ_G then the force has a magnitude of 1, otherwise the magnitude is linearly lowered to 0 until the goal is reached.

Repulsion Force: The repulsion force $f_{\text{repulsion}}$ (Fig. 3.2, b) creates a minimal distance between humans so that they avoid to bump into each other and maintain their personal space. The force is calculated based on the position of other agents (humans or robots) that are inside the personal area of the current human. The personal area is a circle with a radius equal to the personal distance $\Delta_p \in \mathbb{R}$. The force is given by:

$$f_{\text{repulsion}} = -(\Delta_p - d_{\text{min}})^2 \frac{R}{||R||} \tag{3.6}$$

where d_{min} is the distance to the closest individual. The sum of the vector differences between the position of the agent $p \in \mathbb{R}^2$ and each of the N_p agents positions $p_i \in \mathbb{R}^2$ inside its personal area is given by:

$$R = \sum_{i}^{N_p} (p_i - p) \tag{3.7}$$

3.3.2 Gaze Behavior

The gaze of humans can be controlled via the *GazeGenerator* Component. Properties of the Component:

- gaze direction: $\alpha \in [-\pi, \pi]$
- gaze target: identifier of a person or object ($\mathbb{N}^+$)
- properties to define a probabilistic gaze behavior model:
  - gaze targets: list with identifiers of potential targets $G_T = \{i_1, \ldots, i_m\}$ and their probabilities $P_T = \{\Pr(i_1), \ldots, \Pr(i_m)\}$, or a distribution D_T over directions
  - gaze change rate parameter: $\lambda \in \mathbb{R}^+$

The gaze direction and the gaze target can be either set directly or by a probabilistic gaze model. The model takes either a list of potential gaze targets as inputs or a distribution over gaze directions. For gaze targets $G_T = \{i_1, \ldots, i_m\}$ also a list of their individual probabilities $P_T = \{\Pr(i_1), \ldots, \Pr(i_m)\}$ must be provided with $\sum_{j=0}^{m} \Pr(i_j) = 1$. A new gaze target is then randomly chosen according to the probabilities. The gaze direction $\alpha \in [-\pi, \pi]$ is then the direction to the sampled target. Alternatively to a list of targets, a distribution over gaze directions can be provided from which the direction is sampled: $\alpha \sim D_T$. In this case, no gaze target is defined. The gaze is then kept on the sampled target

until it changes according to a Poisson process. Thus, the gaze duration $\Delta t \in \mathbb{R}^+$ (in seconds) follows an exponential distribution Exp(λ) having a probability density function of:

$$\Pr(\Delta t) = \lambda \exp^{-\lambda \Delta t} \tag{3.8}$$

where $\lambda \in \mathbb{R}^+$ is the rate parameter controlling how long a gaze duration is on average. The average time is equal to λ seconds.

3.3.3 Speech

Humans can generate speech using a *SpeechGenerator* Controller. The Controller has the following properties:

- current speech act: string which identifies the act such as: "WELCOME", "QUESTION:NAME", or "GOODBYE"
- current speech content: list with words of the speech act ("Bonjour. Je suis Alan.")
- current word: string
- history about: speech act, speech content
- word per minute rate: $wpm \in \mathbb{R}^+$

The Controller can be used to generate the speech by setting the current speech act and its content. The generator then plays the speech content word by word depending on the word per minute rate. The generator has a history of the speech acts and contents it has generated in the past together with the start and end time of each speech act. The Conversational Content Generator (Sec. 4) can be used to generate realistic speech acts and dialogues via the *SpeechGenerator* Controller by setting the speech act and content of human agents and ARI. See Sec. 3.7 for an example on how to use the *SpeechGenerator*.

3.4 Human Group Behavior

Several human agents can be organized in groups using the *GroupNavigation* Script. The Script controls the movement and orientation of their bodies. The group behavior is based on the concept of F-formations [7]. F-formations describe the arrangement of individuals in a group with respect to their positions and orientations. They are defined by three social spaces: o-space, p-space, and r-space (Fig. 3.3, a). The o-space is the empty space in the group center around which the individuals of a group are positioned. The p-space is the space in which the members of a group are positioned. The r-space is the space outside the group. A group can have different spatial arrangements: a) circular, where people are positioned in a circle around the o-space, b) vis-a-vis, where two people face each other, c) L-arrangement, where two people are positioned next to each other with a 90 degree angle, and d) side-by-side, where people are standing closely next to each other, often to look at a common object of interest.

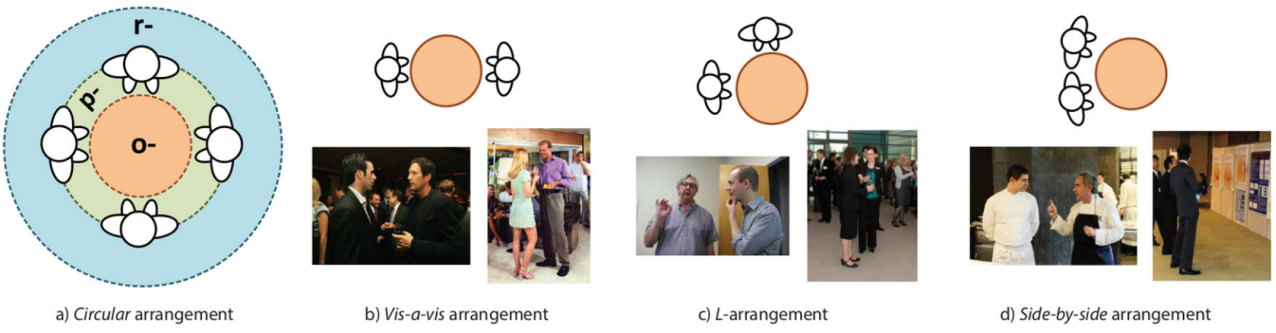

Figure 3.3: Overview of different F-formations that describe group formations. Each formation is defined by a o-space (orange), the space in front of all people of a group where they interact, a p-space (green) where they are located, and a r-space (blue), the space surrounding the group. Figure taken from [19].

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

3.4.1 GroupNavigation Script

For each group G exists a *GroupNavigation* Script which has the following properties:

- A unique identifier: $i \in \mathbb{N}^+$
- A list of human agents that are part of the group: $A = \{H_1, \ldots, H_m\}$
- The 2D goal position of the o-space center point: $o \in \mathbb{R}^2$
- The radius of the o-space: $s \in R$
- The personal distance of people: $\Delta_p \in \mathbb{R}$
- The social distance of people: $\Delta_s \in \mathbb{R}$
- The step length to update the agents position and orientation: $\eta \in \mathbb{R}$

The *GroupNavigation* Script models the movement of the human agents in its group via the social force model by [15]. The model uses three principle forces that are summed to define the translational ($F \in \mathbb{R}^2$) and rotational ($D \in \mathbb{R}$) movement of each human:

$$f^H = f^H_{\text{repulsion}} + f^H_{\text{equality}} + f^H_{\text{cohesion}} \tag{3.9}$$

$$d^H = d^H_{\text{equality}} + d^H_{\text{cohesion}} \tag{3.10}$$

The repulsion force is responsible for creating a minimal distance between humans. The equality force creates an o-space of the group by forcing the humans to group around it. Finally, the cohesion force attracts members of a group towards the group. The final forces are used to update the position of the human and its orientation by:

$$p^H_{t+1} = p^H_t + \eta f^H \tag{3.11}$$

$$\theta^H_{t+1} = \theta^H_t + \eta d^H \tag{3.12}$$

where $\eta \in \mathbb{R}^+$ is a positive step length. The next sections describe each principle force in detail. The index H of the human agent for which the forces are calculated is dropped for convenience.

Repulsion Force: The repulsion force $f_{\text{repulsion}}$ creates a minimal distance between humans so that they avoid to bump into each other and maintain their personal space. It is the same force as used for the navigation of single human agents (Eq. 3.6). Please see Section 3.3.1 for its details.

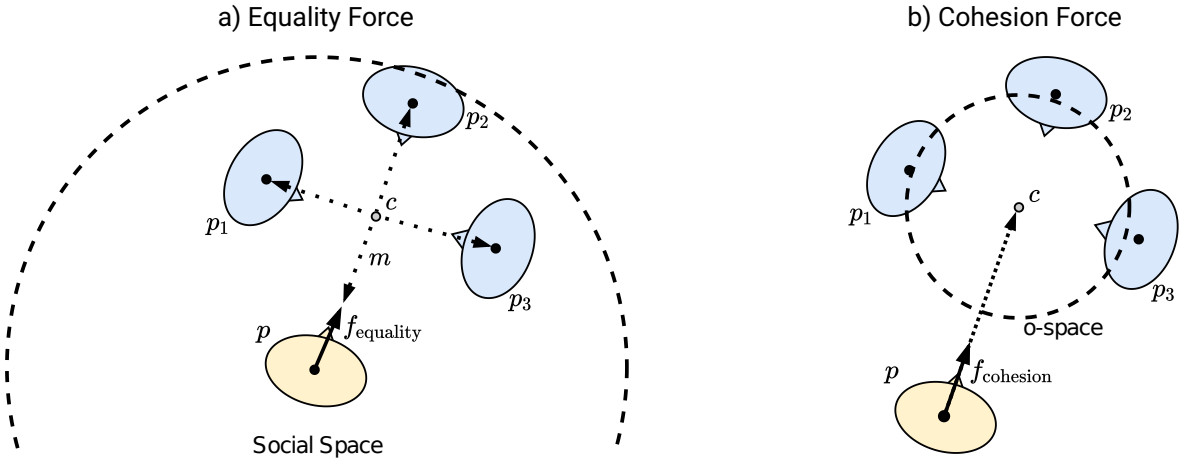

Figure 3.4: The equality force (a) attracts or repels human agents to form a group with nearby group members. The cohesion force (b) attracts or repels all agents to position around the o-space of the group.

Equality Force: The equality force f_{equality} is either attractive or repellent (Fig. 3.4, a). It forces the agent to position itself inside the groups p-space surrounding its o-space. The force depends on the agent's distance to the group center $c \in \mathbb{R}^2$ of all other group members that are in the nearby social space of the agent. The social space is a circular area around the agent with radius $\Delta_s \in \mathbb{R}$, its social distance. The group center is calculated based on the position $p_i \in \mathbb{R}^2$ of each of the N_s agents inside the agents social space, itself included:

$$c = \frac{1}{N_s + 1}\left(p + \sum_{i}^{N_s} p_i\right) \tag{3.13}$$

Based on the center, the mean distance m of all agents to the center can be computed:

$$m = \frac{||p - c||}{N_s + 1} \sum_{i}^{N_s} ||p_i - c|| \tag{3.14}$$

The center and the mean distance are used to define the equality force:

$$f_{\text{equality}} = \left(1 - \frac{m}{||c - p||}\right)(c - p) \tag{3.15}$$

The equality orientation is given by:

$$d_{\text{equality}} = \sum_{i}^{N_s} (p_i - p) \tag{3.16}$$

Cohesion Force: The final force, the cohesion force f_{cohesion}, pulls agents towards the o-space center of the group $o \in \mathbb{R}^2$ (Fig. 3.4, b). It depends on the difference and distance between the o-space center and the agents position $p \in \mathbb{R}^2$:

$$f_{\text{cohesion}} = \alpha \left(1 - \frac{s}{||o - p||}\right)(o - p) \tag{3.17}$$

where $s \in R$ is the radius of the o-space. The scaling factor $\alpha = \frac{N_a}{N_s + 1}$ reduces the magnitude of the force if the agent is surrounded by other agents of its group. The cohesion force is strongest for agents that are isolated from its conversation group. The cohesion orientation is given by:

$$d_{\text{cohesion}} = \sum_{i}^{N_a} (p_i - p) \tag{3.18}$$

3.4.2 Simulation Results

An example of a simulated group of human agents is illustrated in Fig. 3.5. The group consists of three human agents (ids 11, 33, and 11) starting in the middle of the left room (Fig. 3.5, a). They are assigned to a single *GroupNavigation* Script and its group center was set to a position at the top of the room next to the couch. After starting the simulation all three agents move towards the goal location (Fig. 3.5, b). Two agents (ids 33 and 11) arrive at first at the group location and form a F-formation in a L-arrangement (Fig. 3.5, c). When the third human agent arrives they make space for it by shifting to the top and all agents form a F-formation in a circular arrangement (Fig. 3.5, d).

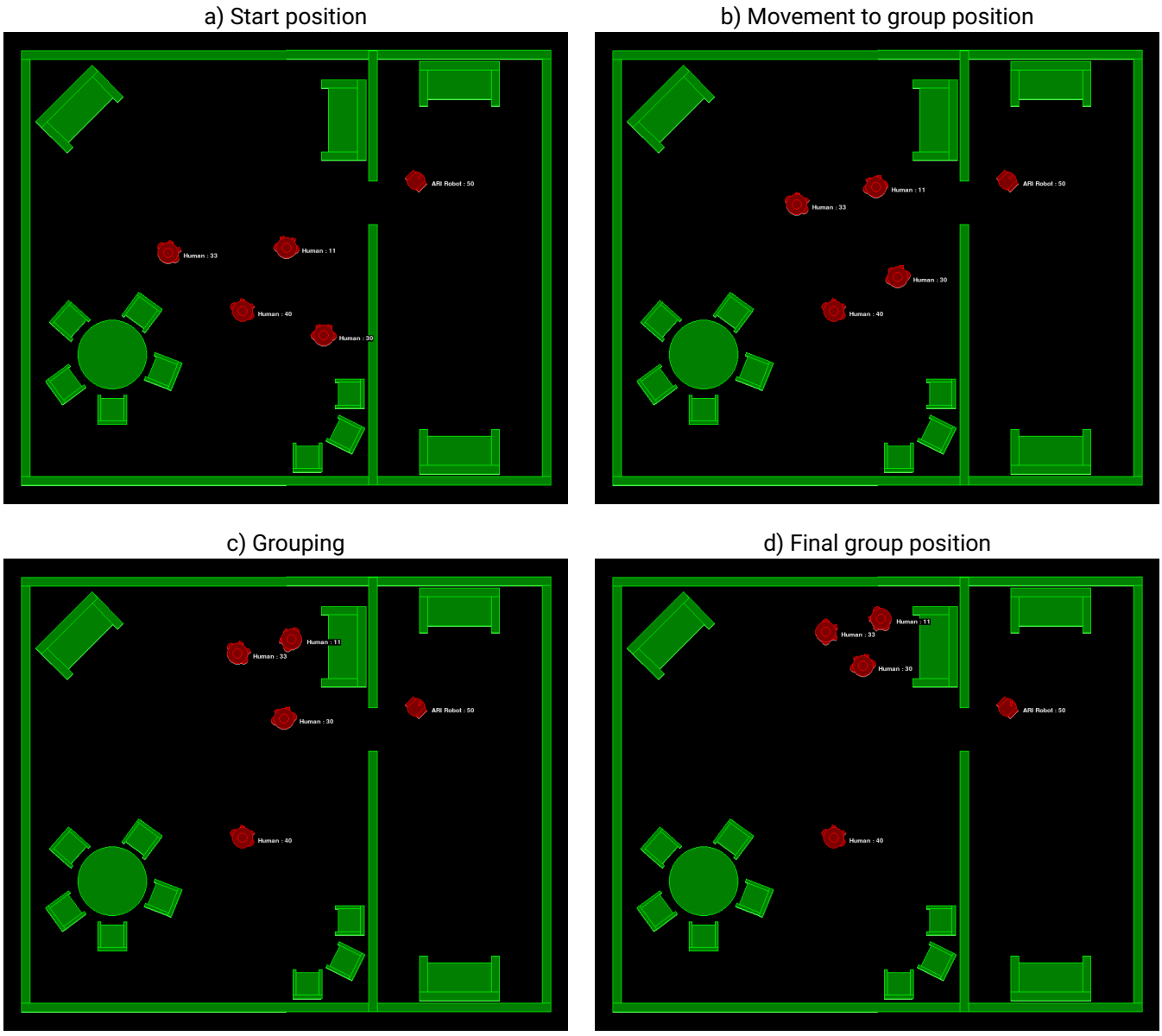

Figure 3.5: Group behavior of three human agents (id: 11, 30, 33) that start in the middle of the left room. The group center is at the right, top location of the room.

3.5 ARI Agent

The ARI agent is simulated via an Object. It has Components that generate its sensory input from the environment and that control its movement and speech behavior. The ARI Object has the following properties:

- a unique identifier: $i \in \mathbb{N}^+$
- 2D body position: $p \in \mathbb{R}^2$
- 2D body orientation: $\theta_B \in [-\pi, \pi]$
- pan head orientation: $\theta_H \in [-\pi, \pi]$

Sensor Components As part of the SPRING project (work packages WP2, WP3, and WP4), several perception modules are developed for the ARI robot that collect a variety of information:

- occupancy map of the environment (based on visual mapping from WP2 and torso RGB-D camera)
- 3D position tracking of humans in the environment (via the front fish eye camera)
- identification of different soft-biometric patterns including age or gender (via the head camera)
- identification if a mask is worn (via the head camera)
- face identification (via the head camera)
- auditory recording of speech from multiple persons with automated speech recognition (ASR), i.e. speech to text processing (via the microphones)

The MPISim provides Sensor Components for ARI that simulate the output of these modules based on the simulated 2D environment. Sensor Components are generally executed after the environment step and after the Generator Components of human agents are executed. They access the generated properties, for example, the gaze of a human, and *sense* it so that higher decision making modules can access this data. An important part of the Sensor Components is to decide if the properties of a certain human in the environment can be sensed, for example, if mask is worn can only be identified if a human is facing ARI. Moreover, they are used to introduce perception errors, for example, an error about the position of humans. The output of Sensor Components follow the ROS4HRI[10] standard developed by PAL that is used for the SPRING project.

Generator Components On the robotic ARI platform, higher decision making modules control the robot through modules that are developed during the SPRING project (work packages WP6 and WP7). The MPISim provides two Generator Components for the ARI agent to replicate their behavior. The first is the *SocialMPCNavigation* Component that can be used to navigate the robot to a goal position, or to join humans and groups. The second component is the *SpeechGenerator* used by ARI to talk. It is the same *SpeechGenerator* Component as human agents use which is described in Section. 3.3.3.

The following sections describe the different Sensor and Generator Components of the ARI agent in detail.

3.5.1 Environment Mapping

The *MapSensor* Component creates an occupancy map [13]. The occupancy map is a 2D map of the environment. It discretizes the space where each of its cells defines if this place is occupied by an obstacle or not (Fig. 3.6). On the ARI robot this map is created using the visual mapping procedures developed in WP2 (see SPRING Deliverable D2.2 [21], Chapter 3). Moreover, the map is updated based on the input from the RGB-D camera in the torso of ARI using the RTAB-Map framework[11] [10]. The *MapSensor* Component replicates this mapping process. For this purpose, it uses the a visualization of all objects in the simulation created with the PyGame framework[12] (Fig. 3.6, b). Please note, other agents such as humans are not part of the occupancy map as they are tracked separately. Based on the visualization, the occupancy of each cell in the map is computed (Fig. 3.6, c). The map is used by the *SocialMPCNavigation* Component (Sec. 3.5.5) for the navigation.

[10] http://wiki.ros.org/hri
[11] https://introlab.github.io/rtabmap/
[12] https://www.pygame.org

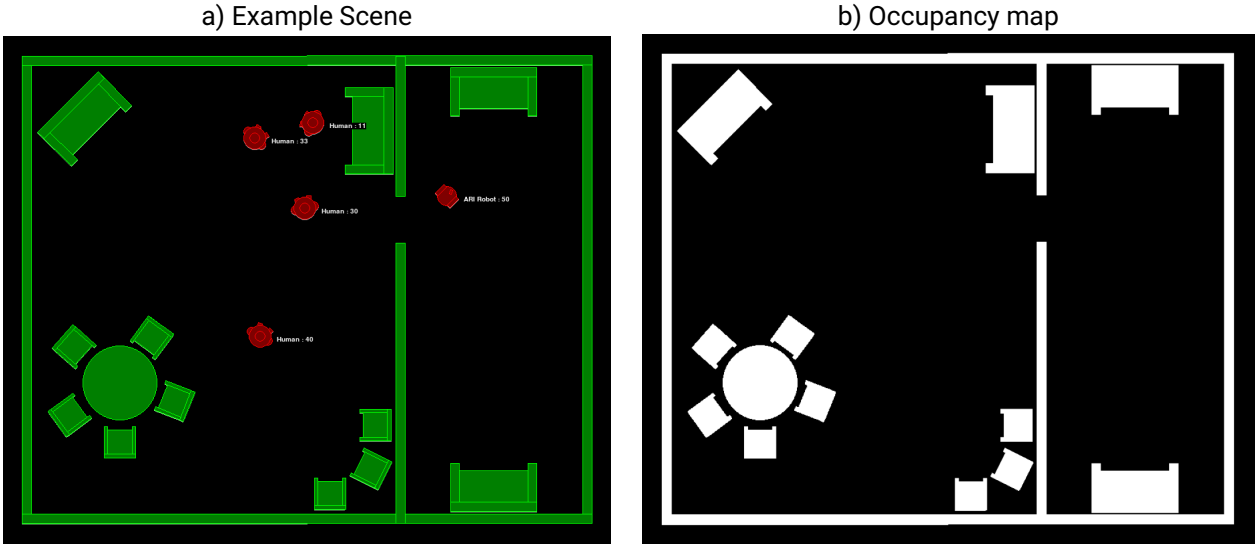

Figure 3.6: The occupancy map (b) of a simulated environment (a) defines objects that ARI has to avoid to collide with during its navigation. It entails all static objects such as walls and furniture (green). Agents (red) are not part of the occupancy map and tracked separately.

3.5.2 Tracking of Humans and Groups

One of the major task on the ARI robot platform is the identification and tracking of humans and groups of humans. For this purpose the front fish eye camera is used that is located in the torso of ARI. It has a forward view with an angle of 180 degrees. Based on the captured images the bodies of humans are detected and tracked using a deep neural model (see SPRING Deliverables 3.1 [22] and 3.2[23]). The tracking assigns each human body a consistent identifier over several time steps even if occlusions occur or if the person leaves shortly the field of view. The camera images are also used to identify the 2D posture of human bodies with the OpenPose[13] framework. The 2D posture is used to identify their 3D position and orientation.

In the MPISim the *HumanTrackingSensor* Component replicates this tracking process. It first identifies which human agents are in the field of view of ARI. This is done using raycasting with the Box2D[14] framework to detect the objects ARI can see within its field of view of the fish eye camera (Fig. 3.7). As the simulation has the unique identifiers i of human agents, these are used to generate consistent tracking ids even if occlusions occur or a human leaves the field of vision for some time. An error model can be applied to introduce perception errors that occur on the ARI robot platform during the tracking process. These errors include:

- identification errors: a person in the field of vision is not identified and tracked
- re-id errors: a person switches its id after occlusions, reentering the field of vision, or spontaneously
- position errors: the positions of humans are not correctly identified

Besides the tracking of individuals, a group detection module on the ARI robot platform also detects conversational groups. The detection is based on the F-formation theory [7] about the positioning of humans inside a group. See Figure 3.3 for an overview over different formation types. The Graph-Cuts for F-formation (GCFF) algorithm [19] is used to detect groups based on the position and body orientation of humans. In the MPISim the *GroupDetectionSensor* Component replicates this group detection process. It receives as input the perceived 2D position and orientation from the *HumanTrackingSensor* Component. It then uses the GCFF algorithm for the detection of groups. The algorithm is explained in detail in SPRING Deliverable D4.2 [25]. It outputs groups and their center position, i.e. the center point of their o-space.

3.5.3 Detection of Faces and Related Human Properties

Modules that concentrate on analysing the head and faces of humans are used on the robotic ARI platform to extract several types of information (see SPRING Devliverables D4.1 [24] and D4.2 [25] for details) such as:

[13] https://cmu-perceptual-computing-lab.github.io/openpose
[14] https://www.iforce2d.net/b2dtut/raycasting

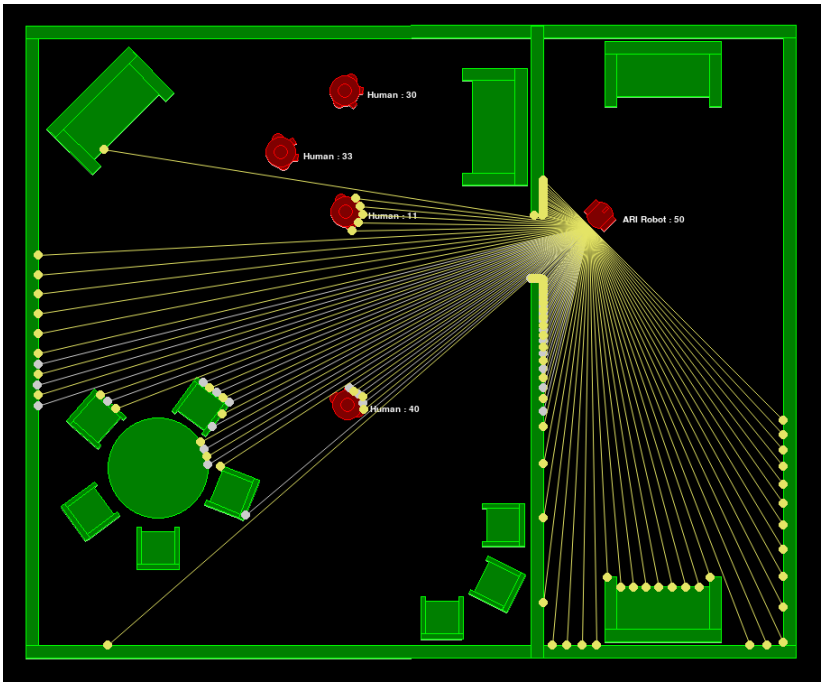

Figure 3.7: The raycasting procedure identifies which humans and other objects are visible for the front fish eye (yellow rays) and the head camera (grey rays) of ARI. Several rays are used that start at ARI and are projected outwards within the viewing angle of each camera. The objects that are first hit by the rays (yellow and grey dots) can be seen by the respective camera. In this example the fish eye and head camera can both see human 40. Only the fish eye camera can see human 11. Human 30 and 33 are not visible for ARI.

- face identification
- soft-biometric patterns including age or gender
- identification if a mask is worn
- emotions
- gaze

For this purpose the head camera of the ARI robot platform is used which has a viewing angle of 62 degrees. In the MPISim several Sensor Components replicate the identification of these properties. All Components first need to identify if a human face can be seen by the head camera. The same raycasting method as for the identification of human bodies with the front fish eye camera is used (Fig. 3.7). Furthermore, the orientation θ of the human agents that are in the field of view is used to identify if their faces are visible or if they are turned away from ARI. Different components are then used to extract specific information from the detected humans:

- *FaceTrackerSensor* Component: Identifies a person based on its identifier.
- *SoftBioMetricsSensor* Component: Extracts the age and gender of a person.
- *FaceMaskSensor* Component: Detects if the person wears a face mask.
- *EmotionSensor* Component: Detects the emotion of a person.
- *GazeSensor* Component: Identifies the gaze direction α and the ID of the gaze target from the *GazeGenerator* Component (Sec. 3.3.2).

Each component can have an error model to replicate potential perception errors. The models are probabilistic and can take into account the distance to the human agents, their orientation in relation to ARI, or if they are only partially visible due to occlusions or because they are at the border of the visual field.

3.5.4 Speech Detection

The ARI robot processes speech signals from multiple persons with its microphone array located in the torso of the robot. An automated speech recognition (ASR) module translates the recorded conversations into text. In the MPISim the ARI agent can recognize spoken speech in its surrounding via the *SpeechSensor* Component. Properties of the *SpeechSensor*:

- max listening distance $d \in \mathbb{R}^+$
- for each human in the surrounding:
  - current speech act
  - current word
  - history about: speech act, speech content

The Component provides the same output as the ASR of the ARI robot. It identifies all humans in the surrounding of the ARI agent within the max listening distance d. Humans that are occluded by walls, i.e. which are in a different room, are ignored. The *SpeechSensor* then reads out the current speech act and the current word from the *SpeechGenerator* Component (Sec. 3.3.3) of the identified humans. As for the other Sensor Components, an error model can be used to introduce perception errors such as wrongly perceived words or that the speaker is misidentified.

3.5.5 Movement and Navigation

The ARI agent can be either controlled directly or by using the *SocialMPCNavigation* Component that replicates the navigation module on the ARI robot. Controlling the ARI agents base, i.e. its body, by defining a linear velocity ($v_p \in \mathbb{R}$) and a angular velocity (v_{θ_B}) that control the 2D body position p towards the current orientation θ_B and the body orientation θ_B respectively. The pan of the head orientation (θ_H) can be controlled via a pan velocity ($v_{\theta_H} \in \mathbb{R}$).

Although a direct control exists, the more common way of controlling the ARI agent is via the *SocialMPCNavigation* Component. It replicates the interface and functionality of the social model predictive controller (SocialMPC) used by ARI's Behavior Manager for human-aware navigation (see SPRING Deliverable D6.3 [30] for more details). The SocialMPC allows to define either a goal position where ARI should go or a human agent, or a group that ARI should join. It then controls ARI through the environment taking into account obstacles, human agents, and their social space [5]. The *SocialMPCNavigation* Component replicates this behavior by using the SocialMPC for its navigation in the simulation environment. Its inputs are the position p and orientation θ_B of the detected humans and groups coming from the *HumanTrackingSensor* Component and the occupancy map of the environment from the *MapSensor* Component (Fig. 3.8, b). The occupancy map defines where objects are located which ARI has to naviage around. The positions and orientations of humans and groups are used to compute their social spaces which ARI should avoid to intrude (Fig. 3.8, c). A detailed description of the SocialMPC is provided in SPRING Deliverable D6.1 [28].

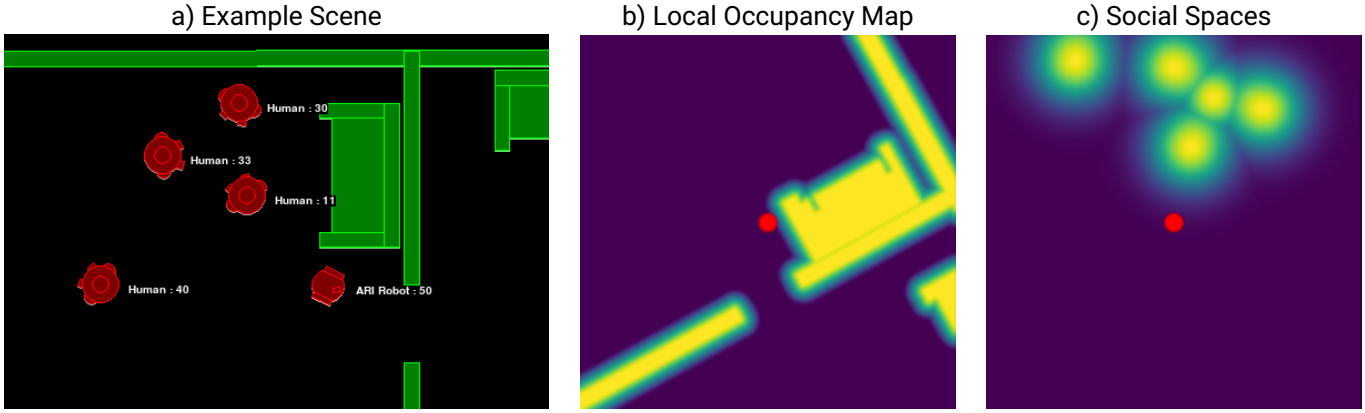

Figure 3.8: The *SocialMPCNavigation* uses a local occupancy map (b) around ARI that is based on the global occupancy map from the *MapSensor* (Fig. 3.6) to avoid obstacles. Based on the detection of humans and groups by the *HumanTrackingSensor*, their social spaces are modeled (c) which ARI should avoid to enter. The red dot in (b, c) shows the location of ARI.

3.6 GUI

The simulator has a graphical user interface (GUI) as a method to visualize it. The interface is based on the PyGame[15] framework. The GUI functions as a Process. It allows to visualize the following properties (Fig. 3.9):

- the PyBox2D bodies of all Objects including ARI, human agents, and objects such as walls or furniture
- properties of objects such as their identifier or the age, gender, or role of human agents
- the current dialog of humans and ARI

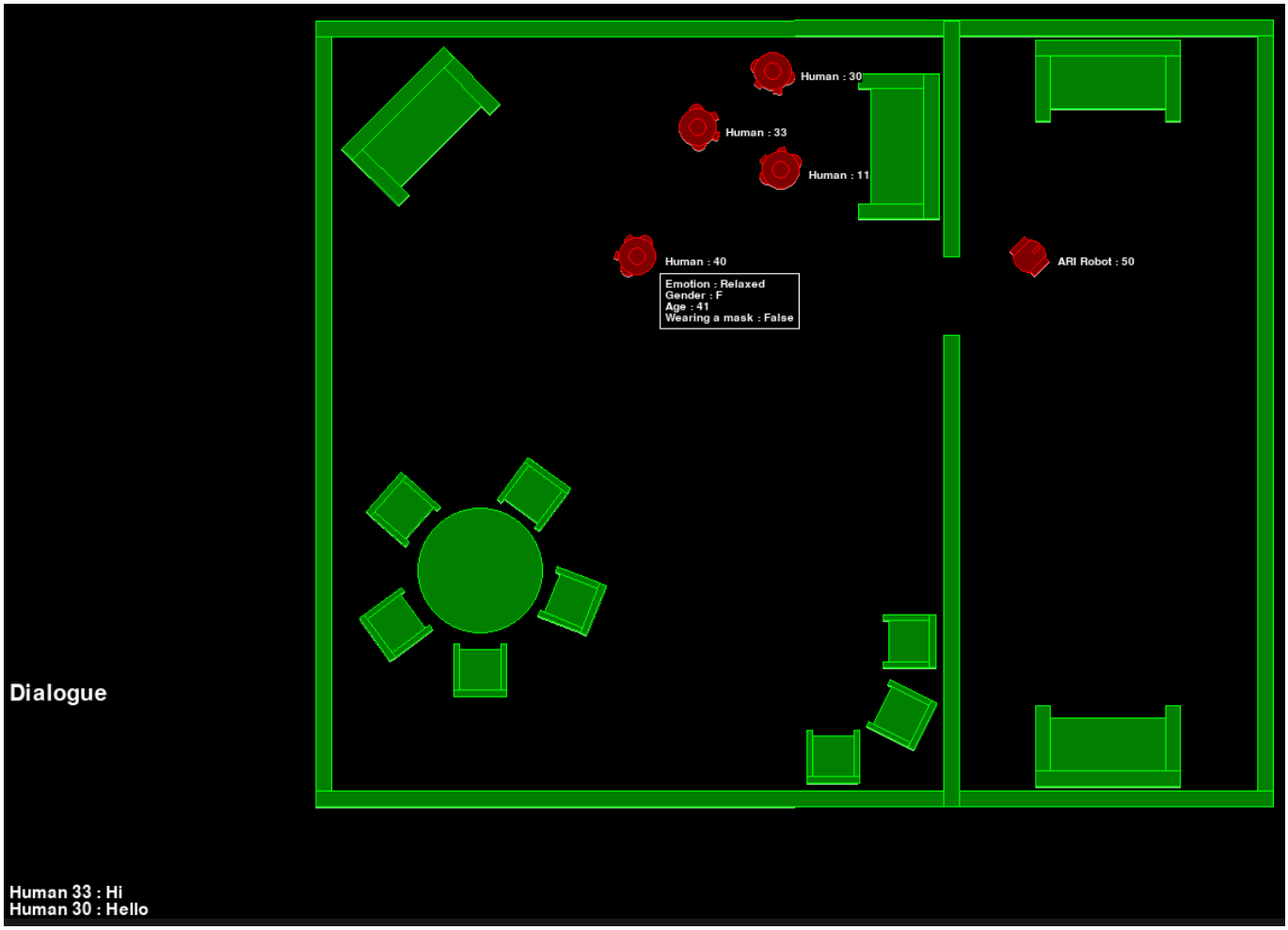

Figure 3.9: The GUI of the simulation. Static objects such as furniture and walls are drawn in green. Human agents and ARI are red. Hovering with the mouse over an agent shows its properties, here for Human 40. The dialogue between agents is shown in the lower, left corner.

[15] https://www.pygame.org

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

3.7 Scripting Interaction Scenarios

Interaction scenarios can be scripted via Scripts. Scripts are Python classes that have access to the simulation and all its elements, including Objects, their Components, Processes, and other Scripts. At each iteration of the simulation, the Script will be called (`step` method) after all other elements have been executed for this step. The Script has then the ability to check their properties and give them new commands, for example, to give a human a new goal position. It can also add or remove Objects.

As an example, the follwoing *GroupDiscussionScript* simulates an interaction where ARI joins a group of two people and has a short conversation with them. First, we set up the simulation by initializing it with a room having 4 walls. We add the *GroupDiscussionScript*, and runs the simulation, which will then use the *GroupDiscussionScript* to generate our interaction scenario:

```
import mpi_sim # import the MPISim Python package

# create an empty room
simulation = mpi_sim.Simulation(
    objects = [
        {'type': 'Wall', 'position': [6., 3.], 'orientation': 0., 'height': 6.}, # east
        {'type': 'Wall', 'position': [0., 3.], 'orientation': 0., 'height': 6.}, # west
        {'type': 'Wall', 'position': [3., 6.0], 'orientation': np.pi / 2, 'height': 6.2}, # south
        {'type': 'Wall', 'position': [3., 0.0], 'orientation': np.pi / 2, 'height': 6.2} # north
    ],
    processes = [{'type': 'GUI'}, {'type': 'Mapping'}]
)

# create the script and add it to the simulation
script = GroupDiscussionScript()
simulation.add_script(script)

# run the simulation, until the script stops the simulation
simulation.run()
simulation.close()
```

The *GroupDiscussionScript* itself adds all agents that interact and controls their behavior. First, when it is added to the simulation (`add` method) it creates two human agents (Paul and Irene) and adds them to a group. Furthermore, it adds ARI and defines that ARI should join the group by setting one of its members (Irene) as its navigation goal.

The agents are then controlled at each simulation step (`step` method) depending on the current state of the interaction. The script checks if certain conditions are fulfilled before going to the next state (similar to finite state machines[16]). After ARI reaches the group (closer than 1.5m to Irene), it welcomes them and asks if it can help (speech act = `'QUESTION:HELP'`). After ARI asked his question (checked via `previous_speech_act == 'QUESTION:HELP'`), Irene is asking for the time. ARI answers and they end the conversation. Finally, ARI leaves the group by navigating to another point in the environment. Fig. 3.10 illustrates the behavior of the script.

```
import mpi_sim
import numpy as np

class GroupDiscussionScript(mpi_sim.Script):

    def add(self):
        """Define what to do when script is added to simulation."""

        # add two humans (paul and irene)
        self.paul = mpi_sim.objects.Human(position=[0., 0.], orientation=0.)
        self.simulation.add_object(self.paul)
        self.irene = mpi_sim.objects.Human(position=[3., 0.], orientation=0.)
        self.simulation.add_object(self.irene)

        # add ARI
        self.ari = mpi_sim.objects.ARIRobot(position=[1., 4.], orientation=-3 / 4 * np.pi)
        self.simulation.add_object(self.ari)

        # create a group
        self.group = mpi_sim.scripts.GroupNavigation()
```

[16] https://en.wikipedia.org/wiki/Finite-state_machine

```
        self.simulation.add_script(self.group)
        self.group.add_agent([self.paul, self.irene])

        # define a goal for the group and let ari join the group by giving on person as target
        self.group.set_group_center([2.0, 3.0])
        self.ari.set_human_go_towards(self.irene)

        self.status = 'grouping'

    def step(self):
        """Define what to do in each time step."""

        # if ARI is reached: ask if he can help
        if self.status == 'grouping' and mpi_sim.utils.measure_distance(self.irene, self.ari) < 1.5:
            self.ari.speech.say('Hello, I am ARI. Can I help you?', act='QUESTION:HELP')
            self.status = 'ask_help'

        # ask ARI for the time
        elif self.status == 'ask_help' and self.ari.speech.previous_speech_act == 'QUESTION:HELP':
            self.irene.speech.say('Hello. Yes. What is the time?', act='QUESTION:TIME')
            self.status = 'ask_time'

        # after the question is asked, let ARI answer
        elif self.status == 'ask_time' and self.irene.speech.previous_speech_act == 'QUESTION:TIME':
            self.ari.speech.say('It is 14:30.', act='ANSWER:TIME')
            self.status = 'answer_time'

        # after ARI answered, thank it
        elif self.status == 'answer_time' and self.ari.speech.previous_speech_act == 'ANSWER:TIME':
            self.irene.speech.say('Thank you. Good bye.', 'GOODBYE')
            self.status = 'thank_ari'

        # finally, let ARI leave
        elif self.status == 'thank_ari' and self.irene.speech.previous_speech_act == 'GOODBYE':
            self.ari.speech.say('Good Bye', act='GOODBYE')
            self.ari.set_goal_position([1, 4.], -3 / 4 * np.pi)
            self.status = 'leave'

        # stop the simulation after ari left
        elif self.status == 'leave' and mpi_sim.utils.measure_distance(self.ari, [1., 4.]) < 0.5:
            simulation.stop()
```

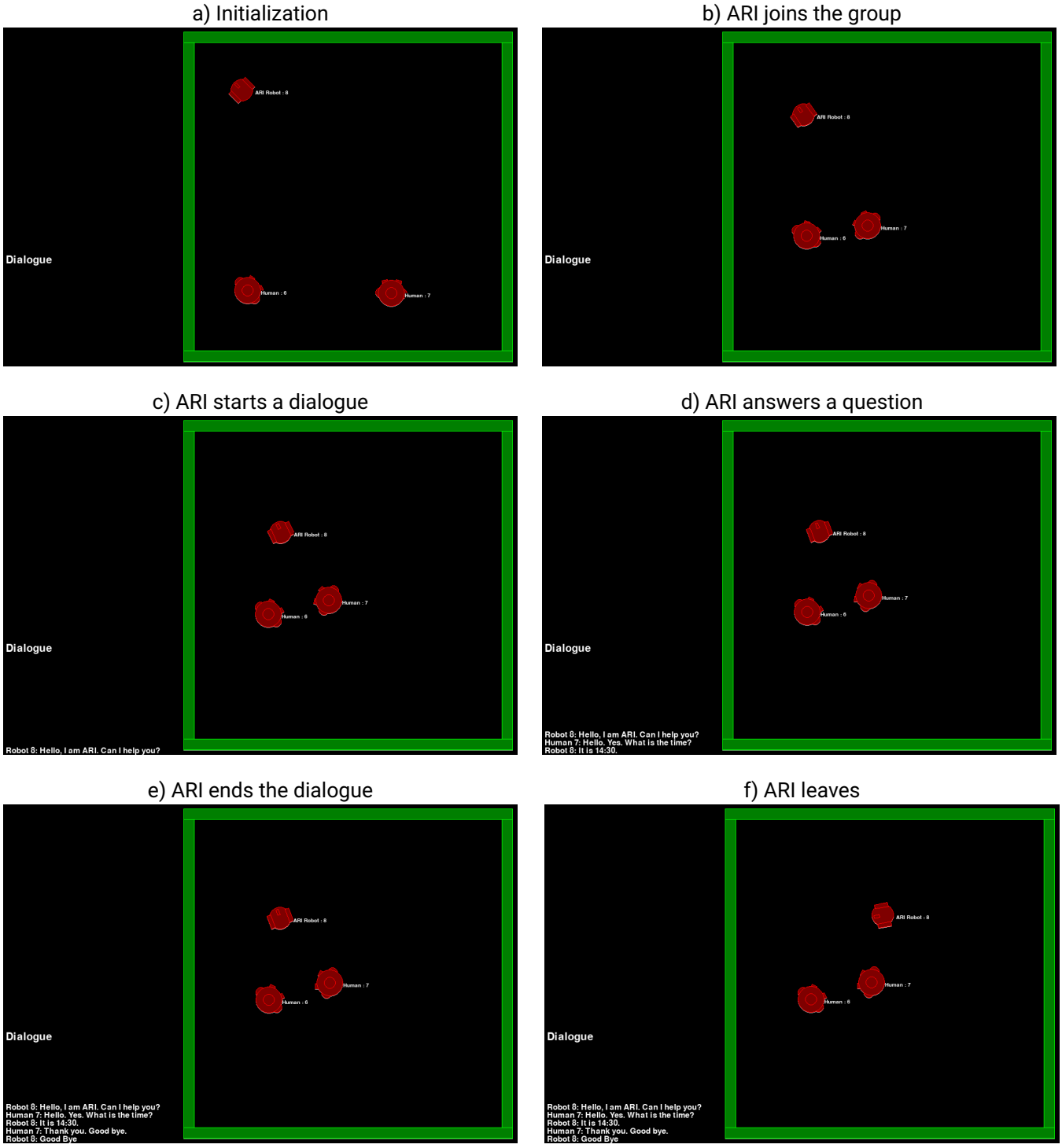

Figure 3.10: Behavior of the `GroupDiscussionScript` where ARI (starts in the top) joins a group of two people (start in the bottom). After joining the group (a, b), ARI asks if he can help (c) and then answers a question (d, e), before leaving the group (f).

4 Conversational Content Generator

4.1 Overview

The conversational content generator (CCG) is a user simulation tool to generate synthetic conversational content data to simulate the utterances that the different agents produce in a social context. It is developed by HWU. Its purpose is to provide a simulation framework for generating enough data so as to be able to test and train policies for the SPRING Multi-party Conversational System (T5.3), for example with Reinforcement Learning [31], based on the architectures presented in SPRING Deliverable D6.1 [28]; and the data generation framework, of the multi-party interaction simulator in Section 3, regarding it's conversational content, Section 3.3.3.

There has been a variety of prior work on user simulation for dialogue systems, as the use of user simulation is seen as a critical tool for automatic dialogue management design [18]. The primary goal is to generate dialogue behaviour realistic enough for training and testing a prototype system. These simulations have been used to train dialogue managers using Reinforcement Learning. The CCG will simulate users' behaviors in conversation. These include, for example:

- Simulate user's conversational goals (e.g. check in, then find the bathroom.)
- Simulate user's utterances in response to utterances from other agents (e.g. robot or other humans) - for example "Robot: How can I help you"; Human1: "I have an appointment at 10am"
- Simulate multiple users, for example a patient and their companion (Patient to companion: "Where do we need to go now?", Companion to patient: "I don't know")

The CCG is being developed as part of the T5.3 objective of developing the multi-party conversational system, for training the natural language understanding (NLU), natural language generation (NLG), and the dialogue manager (DM) modules. The CCG for the SPRING project will develop approaches such as the Multi-User Simulation Environment (MUSE) model of [6] using up-to-date tools and methods such as: [11, 12] [17], [20][18] and [8][19]. There are many possible interactions which need to be generated. Within the initial data collections of WP1 some of these interaction examples have been collected. Further, the design of the most recent data collection for multi-party conversations is reported in SPRING Deliverable D5.2 [27], with which we are able to develop such simulations in the next phase of the project. The code can be found in the SPRING repository and is open access to the public at least until 4 years after the project ends:

`https://gitlab.inria.fr/spring/wp5_spoken_conversations/conversational-user-bot-simulator`

4.2 Dialogue Simulation Design

User Simulators are one of the major tools that enable offline training of dialogue systems, most of which has considered only a single user within task-based settings. The challenges for the SPRING project are to extend such approaches to multi-agent settings with different roles (patient, carer, doctor...) and more complex agendas consisting of various different goals. For example a patient and carer may enter the waiting room together, the patient may ask the carer where they should sit, and the robot (overhearing this) could decide to offer to help find a seat. Or the patient could directly ask the robot questions such as where to find various locations or items (water, coffee etc) based on their goals.

Learning dialogue policies directly from the collected data from WP1 would have the problem that the state space that was visited during the collection of the data is limited, as using real users would require much more time and

[17] `https://github.com/MiuLab/TC-Bot`
[18] `https://github.com/wyshi/user-simulator`
[19] `https://github.com/kimdanny/user-simulation-t5`

effort. In addition, every time we modified a dialogue strategy we would have to repeat all experiments with human users from scratch, while a user simulation could allow an efficient and inexpensive way to test the performance of different dialogue policies [3]. A realistic user simulator, built based upon a corpus of example dialogues (from our data collection), can serve as an effective starting point to train the dialogue agents in an online fashion. Once agents master the simulator, they may be deployed in a real environment, or in a virtual one that allows to connect with real humans, like the `slurk` interaction server framework [4] – and continue to be trained online. This in turn will allow us to collect more data to refine the user simulator and iterate over the dialogue policy training.

There is no one standard way to build a user simulator [11]. Furthermore, as the SPRING use case requires a multi-task, multi-agent, multi-party setting, with different types of user roles that need to be simulated, there will not be a singular user agent simulator, but as many and different user bots are needed to simulate, train and test the dialogue policies for task T5.3. These user simulations will be developed inspired by previous research and state-of-the-art developments for user simulation following [20] survey, and insights, of different ways to build user simulators at the levels of DM, that governs the simulator's next move, and NLG, that translates the semantic output from the DM into natural language.

For NLG, the user simulator can operate either at the dialogue act level to select pre-defined templates, or retrieve user utterances from previously collected dialogues, or at the utterance level, to generate the utterance directly with a pre-trained language model. For the DM, a user simulator can adopt either an agenda-based approach or a model-based approach. The Agenda-Based User Simulation (ABUS) [17] is a popular approach, where a user is modelled as having a list of constraints or goals (the Agenda) which they wish to convey in a conversation (e.g. `destination = New York; depart-date= 12/4/22`). User utterances can then be simulated via probability distributions over user dialogue moves, given a prior system/robot dialogue move. On the other hand, data-driven neural user simulators have demonstrated superior performance to the previous approaches. For instance [9] introduces the Neural User Simulator (NUS) to learn realistic user behaviour from a corpus of recorded dialogues such that it can be used to optimise the policy of the DM of a spoken dialogue system. [11, 12] combined the benefits of both model-based and rule-based approaches following an agenda-based approach at the dialogue act level, and a sequence-to-sequence natural language generation (NLG) component is used to convert the selected dialogue acts into natural language. [8] propose a deep learning-based user simulator that predicts users' satisfaction scores and actions while also jointly generating users' utterances in a multi-task manner. It fine-tunes a T5 model [16] in a multi-task setting to make a user simulator that predicts users' satisfaction scores and actions, and generates users' utterances.

4.3 Architecture

Similar to the MPISim (Sec. 3), the design philosophy for the CCG is to be modular and to be easily extensible. The CCG will need to allow operation in 3 modes: as a conversational 'content source' for the *SpeechGenerator* Controller within the MPISim (Sec. 3.3.3); and as a 'stand-alone' user simulator for directly training a dialogue system policy; as a user 'bot' for testing a multi-party conversational system in an interactive setting with human users and other agents, such as the `slurk` interaction server framework [4].

In consequence, the core features of the CCG will follow a similar architecture to the one presented for the MPISim (Sec. 3.2), as well as the frameworks provided by the user simulators of [11] and the `slurk` framework [4]. The CCG system consists of three parts: agent and user simulator, and interaction script.

Simulation The simulation provides the top layer interface to the simulator, with a few common interfaces for agent user simulators. It implements all the rules and mechanism to issue the next user action based on the last agent action.

User (Agent) The user provides some common interfaces for the implementation of user bots (agents). All the agents will be derived from the `agent` class. Many different agents can exist, e.g. `rule_based_agent`, `neural_agent`, `t5_agent`, developed to fulfil different roles at different stages of the development.

Interaction Task (Script) The interaction task script is used to describe and control the behavior of the simulation. A task defines what an interaction looks and behaves like.

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

5 Conclusions

The goal of SPRING task T6.2 is to provide a simulation environment allowing to train and test modules controlling the non-verbal and verbal robot behavior. Three software modules have been developed for this task which support this goal on different levels:

1. The 3D simulation of ARI for Gazebo provides a realistic physical model of the visual sensors and actuators of the robot and their communication interface via ROS. It allows to evaluate visual sensor modules and low-level behavior controllers.
2. The Multi-Party Interaction Simulation provides an abstract simulation of ARI, humans, and their physical interaction. It allows to synthesize low-level behavior (for example, movement of humans) and perception (for example, tracking of humans). Interaction scenarios can be scripted such as shown in Section 3.7 and used to test, for example, if the navigation module of ARI is able to successfully join groups of humans. The simulation is also used to train non-verbal behavior controllers. For example, a modified version of the simulation was used to simulate the movement of human agents and to train a meta-reinforcement learning algorithm for human-aware navigation [1] (Sec. 4.3).
3. The Conversational Content Generator generates synthetic conversational content simulating utterances that agents produce in a social context. It allows to test and train policies for the multi-party conversational system.

All modules are designed to be modular and adaptable to changing requirements. They will be continuously updated and improved upon throughout the project based on the demands for the training and testing of SPRINGs non-verbal and verbal robot behavior modules.

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

Bibliography

[1] Anand Ballou, Xavier Alameda-Pineda, and Chris Reinke. Variational meta reinforcement learning for social robotics. *arXiv preprint arXiv:2206.03211*, 2022.

[2] Sara Cooper, Alessandro Di Fava, Carlos Vivas, Luca Marchionni, and Francesco Ferro. Ari: The social assistive robot and companion. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 745–751. IEEE, 2020.

[3] Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning user simulations for information state update dialogue systems. In *INTERSPEECH*, 2005.

[4] Jana Götze, Maike Paetzel-Prüsmann, Wencke Liermann, Tim Diekmann, and David Schlangen. The slurk interaction server framework: Better data for better dialog models, 2022.

[5] Edward T Hall, Ray L Birdwhistell, Bernhard Bock, Paul Bohannan, A Richard Diebold Jr, Marshall Durbin, Munro S Edmonson, JL Fischer, Dell Hymes, Solon T Kimball, et al. Proxemics [and comments and replies]. *Current anthropology*, 9(2/3):83–108, 1968.

[6] Simon Keizer, Mary Ellen Foster, Oliver Lemon, Andre Gaschler, and Manuel Giuliani. Training and evaluation of an mdp model for social multi-user human-robot interaction. In *Proceedings of SIGDIAL*, 2013.

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

[8] To Eun Kim and Aldo Lipani. A multi-task based neural model to simulate users in goal-oriented dialogue systems. *Association for Computing Machinery, New York, NY, USA*, 2022.

[9] Florian Kreyssig, Iñigo Casanueva, Paweł Budzianowski, and Milica Gašić. Neural user simulation for corpus-based policy optimisation of spoken dialogue systems. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, pages 60–69, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[10] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[11] Xiujun Li, Zachary C Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*, 2016.

[12] Xuijun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. End-to-end task-completion neural dialogue systems. In *Proceedings of The 8th International Joint Conference on Natural Language Processing*, 2017.

[13] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 116–121. IEEE, 1985.

[14] Muaz Niazi and Amir Hussain. Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics*, 89(2):479–499, 2011.

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

[16] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[17] Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York, April 2007. Association for Computational Linguistics.

[18] Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The Knowledge Engineering Review*, 21(2):97–126, 2006.

[19] 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.

[20] Weiyan Shi, Kun Qian, Xuewei Wang, and Zhou Yu. How to build user simulators to train rl-based dialog systems. In *Proceedngs of EMNLP*, 2019.

[21] SPRING Project. D2.2: Semantics-based localisation in realistic environments. Link.

[22] SPRING Project. D3.1: Audio-visual speaker tracking in realistic environments. Link.

[23] SPRING Project. D3.2: Audio-visual speaker tracking in relevant environments.

[24] SPRING Project. D4.1: Multi-modal behaviour recognition in realistic environments. Link.

[25] SPRING Project. D4.2: Multi-modal behaviour recognition in relevant environments.

[26] SPRING Project. D5.1: Initial high-level task planner and conversational system prototype for realistic environments. Link.

[27] SPRING Project. D5.2: Multi-party asr and conversational system in realistic environments.

[28] SPRING Project. D6.1: Neural network architecture specification and design. Link.

[29] SPRING Project. D6.2: Specifications of the generator of situated interactions. Link.

[30] SPRING Project. D6.3: Robot non-verbal behaviour system in realistic environments. Link.

[31] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.