Grant agreement no: FP7-600877

SPENCER:

Social situation-aware perception and action for cognitive robots

Project start: April 1, 2013

Duration: 3 years

DELIVERABLE 1.4

Software specification of the robot architecture

Due date: month 12 (March 2014)

Lead contractor organization: CNRS

Dissemination Level: PUBLIC

Contents

1 Executive summary

This document reports on the Software specification of the SPENCER robot architecture.

It summarizes the results of Task 1.4 (Software specification of the robot architecture). The objective of T1.4 is to build a detailed specification of the software architecture of the SPENCER robot. This includes identification of the main modules and interfaces and the way how they will be integrated.

It is based on a collaborative work of the project partners, and it is partly the result of the first SPENCER integration week (see Sec. 8). In order to come up with a valid architecture, we took into account not only the various components that will be provided by WP2 to WP5 but also the requirements analysis (T1.1) and the robot platform design (T1.2 and T1.3).

Also, we have built and discussed a number of typical SPENCER scenarios in order to refine and verify that the proposed architecture is a suitable framework to implement and integrate all the software components needed by the scenarios.

We have also made and confirmed choices in terms of middle-ware (ROS) and started to install and populate the robot programming infrastructure. The SPENCER software repository has been installed and already contains the description of the main data structures that will be used to communicate between robot software components.

2 Introduction

The SPENCER project aims to deploy a state-of-the-art robotic platform in a highly populated airport environment. The robot will be equipped with new cognitive capabilities in terms of perception, reasoning, planning, navigation, interaction and learning. It is absolutely necessary that the subsystems responsible for these capabilities work in integrated fashion for the robot to derive appropriate actions in real-time. This report presents the design of the software architecture for the SPENCER robot reflecting the true nature of the integrated subsystems and their interfaces. The overall goal of this report is to identify main software modules and define exact interfaces between these modules in order to support the entire process of the robot software development.

The document is organized as follows:

- First, we present the software architecture that has been obtained as a result of the specification process.
- Then, we provide a short description of the main modules.
- A set of scenarios are discussed in order to exhibit concretely various situations and how they can be managed by the SPENCER robot [1].
- The last sections present the tools that will be used as well as the messages definitions to be exchanges between SPENCER modules

3 Robot Architecture

Figure 1 shows the software architecture for SPENCER in terms of data and control flow. This diagram is a refined version of the preliminary software architecture defined in the SPENCER proposal.

During the kick-off meeting the consortium decided to use the flexible open-source Robot Operating System (ROS) framework as the middle-ware [2]. The software specification terminology we use here is already influenced by the usual terminology of ROS.

3.1 Design points

Main design points for the SPENCER architecture are:

- *Task-based software modules*, i.e. to define software modules that closely represent the tasks defined in the spencer proposal.
- *Grouping of software modules*. The software modules are grouped on the basis of typical high level procedures that run on a mobile robot. For example, tasks of people detection, posture and head pose estimation, and upper-body analysis are grouped as human detection.

[1]The scenarios provided here are preliminary in order to test the fact that the architecture is a pertinent framework to treat them. It remains, of course, to refine them and validate them in context. This work will be done in WP4 (Group-Level Analysis and User Studies) and WP6 (System Integration, Deployment, and Evaluation)

[2]`http://www.ros.org`

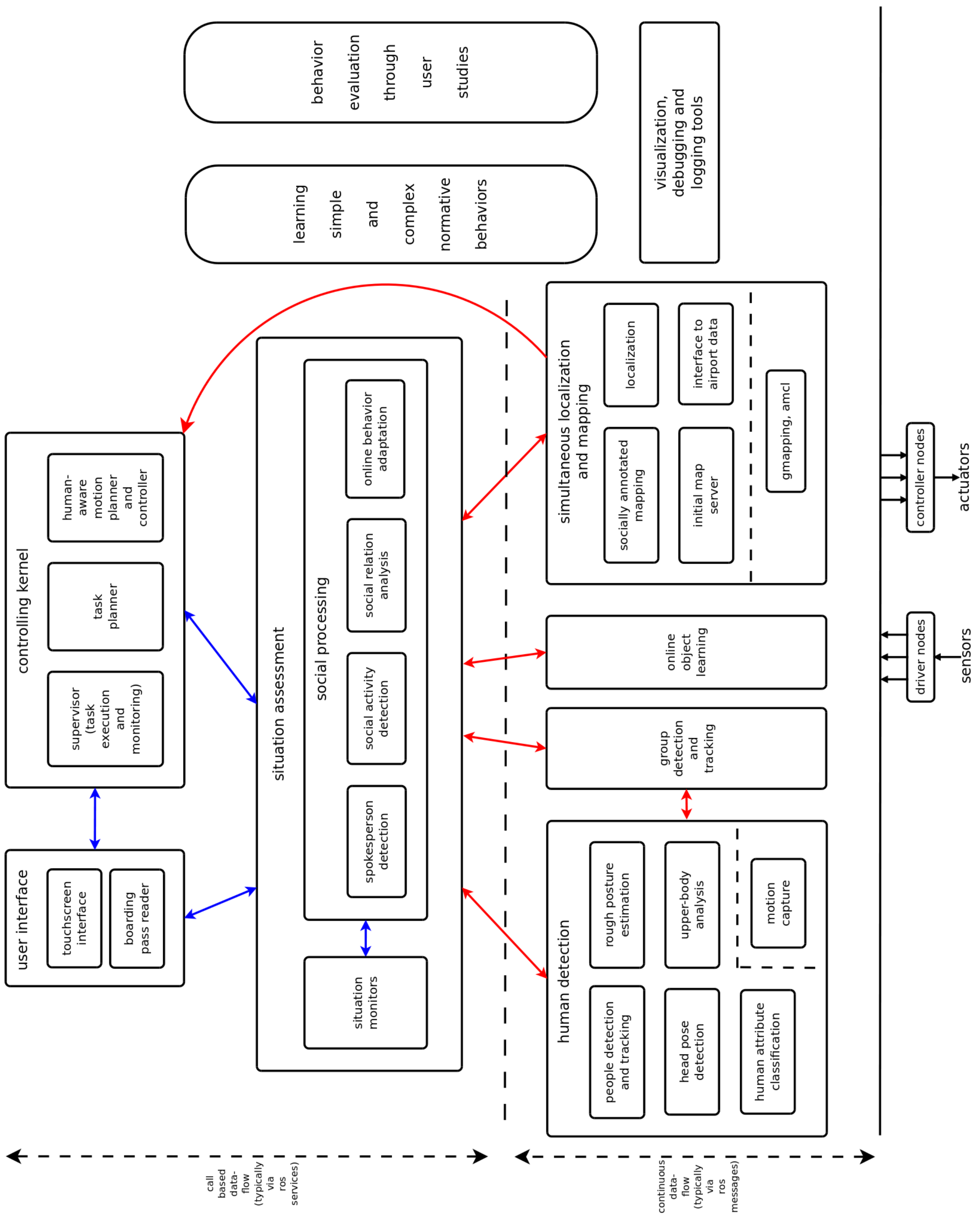

behavior evaluation through user studies
learning simple and complex normative behaviors
visualization, debugging and logging tools
controlling kernel
supervisor (task execution and monitoring)
task planner
human-aware motion planner and controller
user interface
touchscreen interface
boarding pass reader
situation assessment
social processing
spokesperson detection
social activity detection
social relation analysis
online behavior adaptation
situation monitors
simultaneous localization and mapping
socially annotated mapping
localization
initial map server
interface to airport data
gmapping, amcl
online object learning
group detection and tracking
human detection
people detected and tracking
rough posture estimation
head pose detection
upper-body analysis
human attribute classification
motion capture
controller nodes
actuators
driver nodes
sensors
call based data-flow (typically via ros services)
continuous data-flow (typically via ros messages)

Figure 1: Architecture for data and control flow

- *Incremental development*. Since new algorithms will be developed during the SPENCER time span, in the beginning, we have to use preliminary software that will be later on replaced by software from the consortium partners. For example, during the first year for the human-motion detection component a motion capture system was deployed to detect position of humans in the environment. Similarly, the open source software `openslam_gmapping` was used for geometric mapping during the experiments.

- *Continuous vs. call based data-flow*. Lower level processes of perception need to runs continuously and higher level processes should use a service-call based communication. Red arrows in the architecture diagram represents continuous data flow while blue arrows represent service-call based data and control flow.

- *Situation monitors module*: We have added a rather central module for calculating and reasoning about geometric facts. We are proposing a SPARK [3] like system for this purpose. However, the consortium is open to decide on alternatives.

3.2 Coordinate systems

There are three coordinate systems maintained on the robot platform (see Figure 2), in addition to one coordinate system per sensor. One is the robot frame, which is centered between the robot's wheels. One is the global "world" frame, which has its origin at a fixed point in the map. Additionally, there is an "odometry" frame which can be used to derive locally smooth trajectories for the robot. Track IDs from the group and person trackers (see Sections 4.9 and 4.13) will be published in the odometry frame, while the output of the mapping and localization system (Section 4.14) will be in the world frame. The mapping and localization system is responsible for maintaining the world-to-odometry transformation.

4 Module Description

The SPENCER consortium has decided on initial message definitions (when possible directly in ROS messages and services format). Following is the description and list of information needed and provided by various software components defined in the architecture.

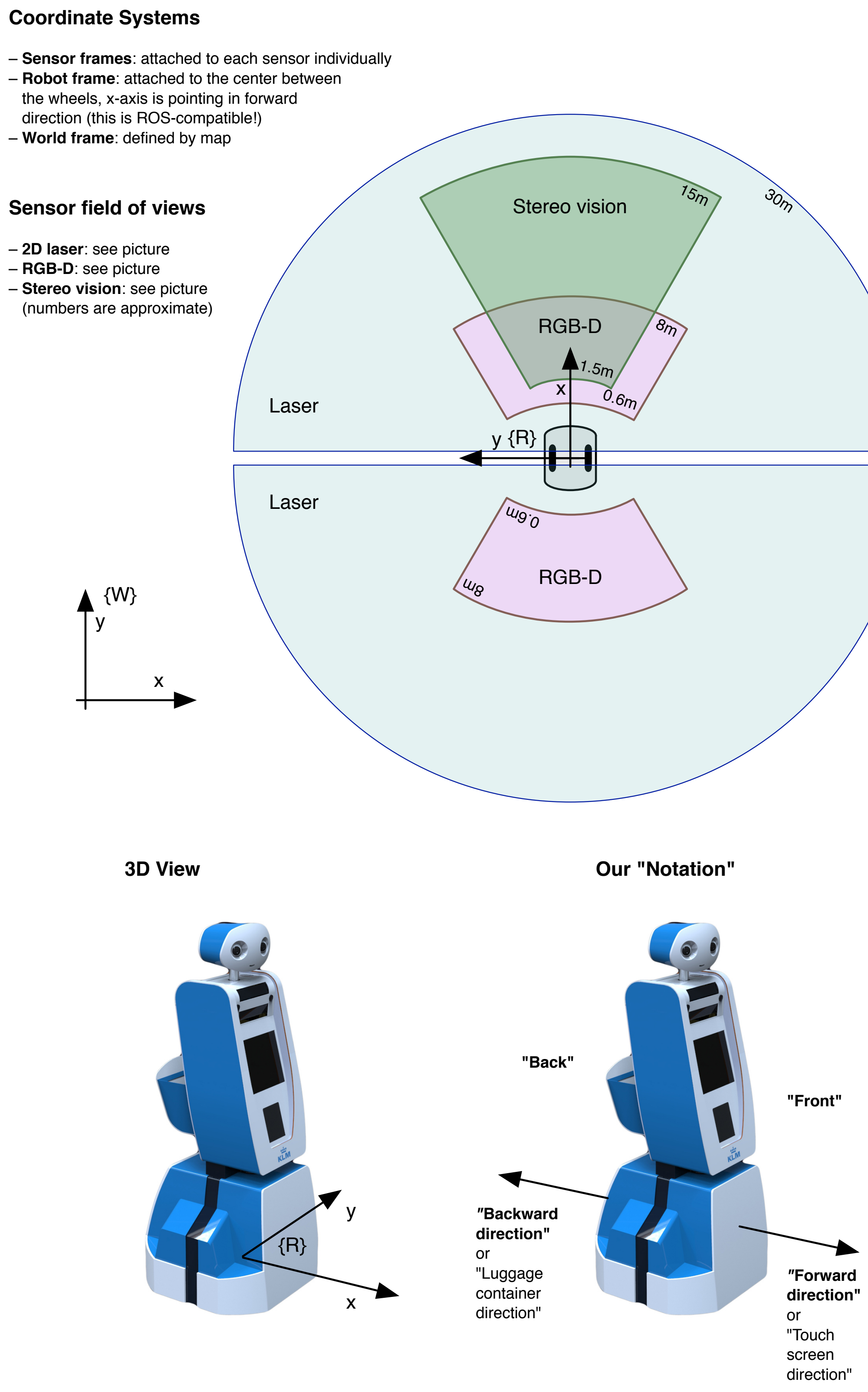
4.1 User interface

The precise user interface will be designed in the future. For now we envision a touch screen which allows the following list of interactions:

- Asking to be guided or to receive direction toward an interesting point (gates, monitors, toilets...).

- Starting, stopping, restarting or abandoning a task.

[3] `https://www.openrobots.org/wiki/spark`



Coordinate Systems

– **Sensor frames**: attached to each sensor individually
– **Robot frame**: attached to the center between the wheels, x-axis is pointing in forward direction (this is ROS-compatible!)
– **World frame**: defined by map

Sensor field of views

– **2D laser**: see picture
– **RGB-D**: see picture
– **Stereo vision**: see picture (numbers are approximate)

Figure 2: Coordinate systems used on the SPENCER platform.

- Displaying generic text strings.
- Displaying a portion of the map showing the robot path in the near future.

4.2 Supervisor

The supervisor is the module in charge of orchestrating the various system modules in order to complete a task. In a typical situation the supervisor should handle the engagement process of the users and then work with the task planner, motion planner and motion primitives to complete it. We can picture different modalities for the supervisor, such as a guiding phase, when the robot is actively guiding people to their destination, a ready phase, when the robot is ready to receive a new task, and a busy phase, where the robot can't be engaged by users. To better perform human-robot joint actions, the supervisor uses a set of collaborative planners, composed at the moment by POMDPs, to adapt its actions to user behaviors by estimating users intentions (e.g. engaged, not engaged in the task) and status (e.g. okay or having trouble following). Figure 3 shows the supervisor operations.

What it expects:

- Command requests. Received by the User Interface, the Airport Data-Management System or the Debugging System.
- Flight informations, in order to calculate plans and estimate the urgency of task by subtracting an estimate of the time to arrival at the destination (calculated by the Task Planner based perhaps on euclidean information included in the topological map) from the flight departure time. Received by the Airport Data System.
- Semantic information about user positions (e.g. user is close, far, out of range, oriented toward the robot, in a dangerous position for navigation, etc.). Received by the Situation Monitors.
- Information about user activities, received by the Situation Monitors.
- Changes to the topological map (e.g. corridors closed, crowd traffic jams). Received by the Airport Data System.
- Position of the robot in the topological map. Received by the SLAM module.
- Plan composed by a list of nodes to traverse and a cost of the plan, received by the Task Planner.
- Information about group of followers.

What it provides:

- Plan requests, composed by a destination node and a list of constraints. Sent to the Task Planner.
- Goals to the motion plan.
- Commands to the user interface to manipulate the interaction.
- Commands to manipulate other parts useful for interaction, such as the head and lights of the robot.
- Commands to the motor system to stop the robot in dangerous situations.

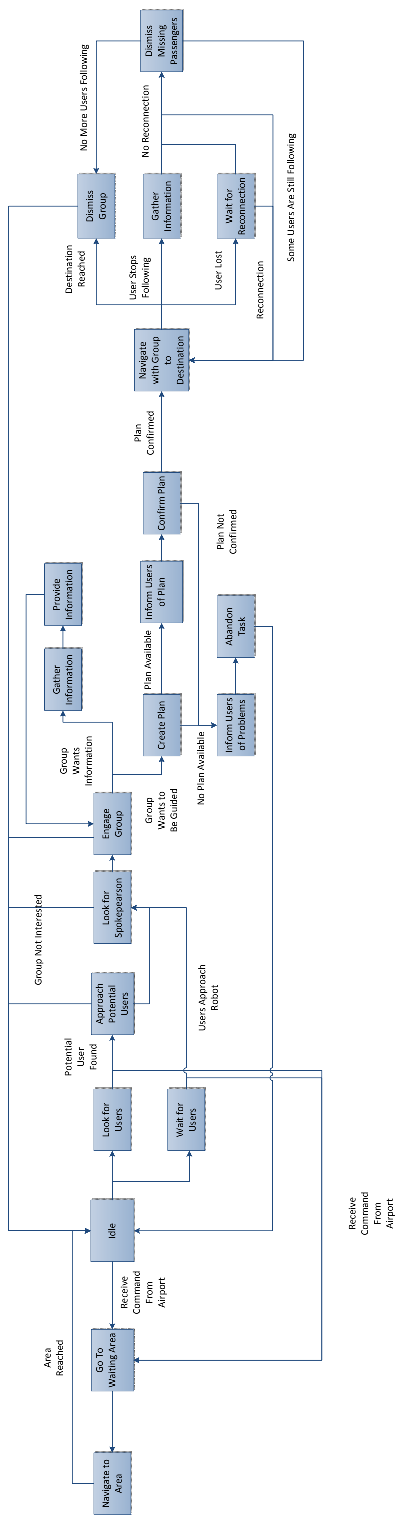


Figure 3: Supervisor scenario

4.3 Task planner

The task planner is the module tasked with creating a plan to achieve the current goal. At the moment we envision that the main task of the Task Planner will be to select the best path from a high-level semantic map (such as an annotated graph) representing semantic areas of the airport.

What it expects:

- A topological map composed by nodes with semantic information and edges with associated costs.
- Planning requests composed by a starting and ending node and a list of constraints.

What it provides:

- A plan composed by a list of nodes that the robot must cross to reach the destination and a cost.

4.4 Human-aware motion planner and controller

We have decided to use an architecture based on the separation between Global Planner and Local Planner. In the beginning, the ROS (`move_base`) package will be used. The planners take into account detected and tracked human positions and velocities, group positions and velocities as well as all the triggered behaviors from learning modules and it generates legible paths accordingly. The defined messages include the interface between the motion planner and the Social Processing Unit. The global planner can also receive a topological node as request to reach from the supervisor module (section 4.2), according to this request it takes the relative part of the map, and it generates the global path to achieve.

The global planner currently adopted is a variant of A*, implemented in the Move3D library at LAAS. The local planner aims to generate feasible commands to the mobile base of the robot. Currently at LAAS the local planner takes into account not only kinematics and non-holonomic constraints but also reactive behaviors to the presence of human beings.

We will write a wrapper between the robot ANT-API and the velocity command output of the local planner output. The head joints will be controlled by the supervisor.

What it expects:

- Position of the human relative to the robot position.
- List of groups (section 4.13).
- List and position of each person in a group.
- Types of grouping.
- Current geometric (matrix) map of the environment with navigable area and obstacles.
- Current position of the robot in the map.

- Airport information with objects grounded in geometric map (e.g. info screen, gate etc.
- Navigation goal.
- Triggered behaviors from learning module (section 4.15).

What it provides:

- Planned global path.
- Velocity commands for the robot base.

4.5 Situation assessment

Understanding the environment in which the robot interacts is a key issue in human-robot interaction. Therefore we want to include a human-centered situation assessment system. This system will be growing in terms of its capabilities during the time-span of the project. Therefore, the input to this system can be any perceptual data that we would like to process for supervision or other high level system. We see the situation assessment system working as an intermediator between low-level continuously running processes and decision making high-level systems. More precisely it will calculate and provide geometric and symbolic facts required for supervision system. Geometric facts will be calculated using small software modules called *situation monitors*. Input and output data-types for this system can be flexible and will be defined during the different phases of the project.

4.6 Spokesperson detection

Using nonverbal cues provided by the human detection modules and other social processing modules, the spokesperson module will identify a person in a group that can be approached and addressed.

What it expects:

- An array of detected groups (provided by the group detection and tracking), each with
  - A unique group track id.
  - The number of persons in the group.
  - An array of the members of the group (identified by unique person id, see below).
  - Position (x, y, z) of the center of gravity of the group (preferable at 30 Hz. or more).
- An array of detected persons (provided by the people detection and tracking), each with
  - A unique person id.
  - Position (x, y, z; preferable at 30 Hz. or more in world-coordinates).
  - Velocity (x, y, z; preferable at 30 Hz. or more in world-coordinates).
  - Length of the current motion track.
  - Human attributes (gender, age, height) and if possible the type of traveler (i.e., business traveler, family traveler, ...).

  - Positions of the upper-body joints (x, y, z; preferable at 30 Hz. or more in world-coordinates).
  - Length of the current upper-body track.
  - Head pose (angles in world-coordinates; preferably at 30 Hz. or more).
  - Length of the current head pose track.
  - A classification of who looks at whom for all persons in each group (e.g., for three persons this would be a 3x3 array with each entry x_{ij} between 0-1 indicating the likelihood that person *i* is looking in the direction of *j* person).
  - Rough posture estimation (i.e., body orientation, standing, sitting, ...).

What it publishes:

- A rank-ordered list of persons (using the unique person track id provided by the people detection and tracking) with a likelihood of being the spokesperson.

4.7 Social activity detection and social relation analysis

Possible activities that might be detected by this module are:

- ”Non-communicative” activities
  - Moving in a group.
  - Standing in a queue, or more specifically, in a queue.
  - Waiting for another person.
  - Group formation (merge, split).
  - Looking at a timetable / information screens.
  - Reading a newspaper.
  - Carrying something (suitcase).
  - Going to the bathroom.
  - Entering a shop / looking at goods in a shop.
- ”Interactive” activities
  - Talking to each other, talking to an airport official.
  - Taking a photograph (of each other / an object).
  - Pushing a wheelchair.
  - Interacting with a check-in kiosk.
  - Interacting with the robot itself.
  - Having a phone call.
  - Buying food.

What it expects:

- position of the human relative to the robot position.
- type of grouping (see social grouping, e.g. group shape, average age, ...).
- list of groups.
- list and position of each person in a group.

What it publishes:

- Single ROS message SocialActivity.msg containing:
  - Type as string, costants are defined in the ROS message
  - Detection confidence.
  - Trackids, IDs of all person tracks involved in the activity, might be one or multiple.

The Social relation analysis publishes the following two ROS messages:

- AllPersonRelations
  - Header which containes info time stamp and a unique id of the message.
  - Array of personRelations, which has one entry for each track-to-track combination in the current cycle.
- PersonRelations, which describes a relation between two human beings tracks
  - Track1id as integer.
  - Track2id as integer.
  - Array of string predicates, e.g. ”friendship”, ”couple”.
  - Array of strengths, between 0.0 and 1.0

4.8 Online behavior adaptation

The activities in this area (Task 5.4 “Online Behavior Adaptation”) only start in year 2 according to the SPENCER Gantt diagram.

4.9 People detection and tracking

This component will be an integrated component of the vision-based tracker by RWTH and the laser based tracker of ALU-FR. Discussion is open on how to integrate the trackers, e.g. fuse output of both trackers, or one tracker serves as input to the other.

Laser based tracker (ALU-FR):

What it expects:

- An array of detected persons, either from a laser-based detector which subscribes to a Laser-scan, or from any other detector via ROS e.g. RGB-D detector consisting of
  - A unique, monotonically increasing observation id.
  - Position with covariance.
  - And optionally a confidence value between 0.0 and 1.0

What it publishes:

- An array of tracked persons, each with
  - A unique person track id.
  - State (UNKNOWN, NEW, MATCHED, OCCLUDED, DELETED).
  - Age and creation time of the track.
  - Id of the underlying observation (detection).
  - If any 2D pose including orientation (if person is moving) plus twist (velocity) with co-variances.

Vision based tracker (RWTH):

The tracking/detection framework internally uses several ROS nodes: visual odometry, (fixed/estimated) ground plane, groundHOG, the upperbody detector, the pedestrian tracker and the pedestrian localization.

VisualOdometry: What it expects:

- Mono image
- Depth image
- Camera info

VisualOdometry: What it publishes:

- Motion matrix (VisualOdometry message type)
  - 4×4 float transformation matrix $(R \mid t)$

Got both a fixed ground plane and a ground plane estimating node. This is about the fixed one.

GroundPlane: What it expects:

- pan tilt unit

GroundPlane: What it publishes:

- GroundPlane message type

  - 3×1 float n ground-plane normal
  - 1 float d distance of ground plane to 0

The upperbody detector gives upper bodies using the depth map in real-time on one CPU core only (works best in close-range).

UpperBody: What it expects:

- ground plane (from GroundPlane)
- camera info
- depth image
- color image (for visualization)

UpperBody: What it publishes:

- UpperBodyDetections message type
  - bounding box array in image-space as int x, y, w, h
  - float distances, being the distance from the detection to the template (score of the measure)
  - float median depths
- PoseArray (array of ROS message type Pose)
- UpperBodyImage (for visualization)

GroundHoG detects people farther away using the GPU.

GroundHoG: What it expects:

- ground plane (from GroundPlane)
- color image
- camera info

GroundHoG: What it publishes:

- GroundHOG detections
- image (for visualization)

The main tracking component using the detections.

PedestrianTracker: What it excepts:

- color image

- camera info
- ground plane (from GroundPlane)
- GroundHOG detections (from GroundHOG)
- UpperBody detections (from UpperBody)
- Visual Odometry information (from VisualOdometry)

PedestrianTracker: What it publishes:

- PedestrianArray
  - array of float $traj_{x,y,z}$: positions on the track, projected to world-coordinate ground-plane
  - array of float $traj_{x,y,z}$-camera: positions on the track, projected to camera-coordinate ground-plane
  - array of float: orientation of the pedestrian
  - float speed: current speed of the person
  - float score: score of the track
  - int id: ID of the track
- pedestrian image (for visualization)

PedestrianLocalization: What it publishes:

- base link
- PedestrianArray (from PedestrianTracker)

PedestrianLocalization: What it publishes:

- PedestrianLocations
  - array of Pose (standard ROS type)
  - float distance
  - array of float angles
- markers (for visualization)

4.10 Rough posture estimation, Head pose detection and Upper-body analysis

These modules will use 2d and 3d camera information and will publish attributes about human posture and pose.

Rough posture estimation:

What it excepts:

- Tracked person id.
- Trajectory history.
- RGB-D image window showing the object in the current frame (obtained by detection or tracking).
- Current motion direction and velocity, relative to camera (to switch between different models for different viewpoints).
- In later stages: accumulated GCT for the object, containing information about the object's surface shape and its motion history.
- if available from detector, body part positions, in particular head position.

What it publishes:

- Person orientation (relative to the camera).
- Rough body posture class (standing, walking, sitting, lying, etc.).
- (In later stages: coarse skeleton pose (mainly head, torso, and leg positions) .

Head pose detection:

What it excepts:

- Tracked person id.
- Trajectory history.
- RGB-D image window showing the object in the current frame (obtained by detection or tracking).
- Current motion direction and velocity, relative to camera (to switch between different models for different viewpoints). *(if available from detector, body part positions, in particular head position).*

What it publishes:

- Head direction estimate (1D angle).
- Uncertainty of this estimate.

Upper body analysis:

What it excepts:

- Tracked person id.
- Trajectory history, including the person's current position in 2D (image coordinates) and 3D (robot/world coordinates).

- RGB-D image window showing the object in the current frame (obtained by detection or tracking).
- The tracked person's current motion direction and velocity, relative to camera (to switch between different models for different viewpoints).
- (in later stages: accumulated GCT for the object, containing information about the object's surface shape and its motion history).
- (if available from detector, body part positions, in particular head position).

What it publishes:

- Detailed skeleton joint positions for upper body (if the person is sufficiently visible)
- Uncertainty measure of this estimate (either globally or per joint)

4.11 Human attribute classification

What it expects:

- Observations from RGBD detector (either RWTH or ALU-FR), possibly cross-referenced with tracks to reduce false alarms.
- Cropped RGB and depth images from the RGBD sensors plus original rectangle coordinates.
- Possibly the point cloud, or a part thereof.
- Mapping of observed person id to tracked person id, so that it is clear to which track the cropped RGB/depth image belongs.

What it publishes:

- Gender (male / female) plus confidence value.
- Age group (e.g. infant, child, teenager, middle-age adult, elderly) plus confidence value.
- The associated observation and/or track id.

4.12 Group detection and tracking

What it expects:

- person tracks along with coordinates (see section 4.9). If person and group tracking are combined in a single, integrated tracker, this communication might happen internally instead of over ROS.

What it publishes:

- Pairwise social relationship strength $(0.0 - 1.0)$ of all tracks.
- Tracks array of tracked groups, each with:
  - A unique group track id (relatively stable while the group is being tracked).
  - Age and creation time of the group.
  - Center of gravity.
  - Array containing unique IDs of the person tracks belonging to the group 4.9.

4.13 Online Object Learning

This module applies online learning methods for moving objects in the airport environment. In contrast to other modules, it operates on objects (e.g. trolleys) rather than on detected humans.

What it expects:

- object tracks with coordinates and raw object information (point clouds, color information)

What it publishes:

- object class identifiers of unsupervised learned (clustered) objects

4.14 Localization and socially annotated mapping

This module provides the position of the robot and its orientation as well as the socially annotated map. The map will be continuously updated with statistical information about social activities ("in place A there is usually activity B going on") and expected occupancy at different time scales. The map will also be annotated with labels of individual places of interest (gates, information screens, etc.)

The module will also provide a topological map that explicitly stores the connectivity of nearby places in the airport, as an aid for path planning. A global path planner can use this topological map to quickly compute a rough path, and then inform the local motion planner of the more detailed metric grid maps along the way.

The SPENCER robot will use an initial deployment phase where it is manually guided through the airport to create an initial map. This map will be used for localization during the subsequent continuous operation, and will be updated with data learned from the social activities modules, as well as occupancy data from the robot's range sensors.

What it expects:

- Social annotations as defined in Section 4.7 and track IDs from Sections 4.9 and 4.13.
- Laser data from the two laser scanners
- Odometry data from the wheel encoders

- Possibly updated map information from airport (using a custom communication protocol)

What it publishes:

- The robot pose relative to the map with its covariance estimate
- Transformation information about the robot pose with respect to the world frame
- Transformation information about odometry frame with respect to the world frame
- Metric occupancy map (parametrized on the time scale of dynamics)
- Metric maps with social annotations (one topic per type of social activity)
- Metric map labeled with places of interest (gates, information screens, etc.)
- Topological map (to aid motion planning). The topological map includes nodes connected with edges. Each node stores information about its position in the metric map, as well as a unique node id, and is optionally connected to a place-of-interest label. Each edge stores information about its end points and the Euclidean distance between them, as well as a unique edge id.

4.15 Learning simple and complex normative behaviors

This module will provide schemes of behavioral norms for the robot to reproduce human social behaviors. This module will use low level as well as high level information provided by various other software modules.

What it expects:

- Position of the human relative to the robot position.
- Type of grouping (e.g. group shape, average age, ...).
- List of groups.
- List and position of each person in a group.
- Current geometric (matrix) map of the environment with navigable area and obstacles.
- Current position of the robot in the map.
- Airport information with objects grounded in geometric map (e.g. information screen, gate etc.).
- Current topological map of the environment, grounded in geometric map.
- Navigation goal.

What it publishes:

- CostMaps (or another custom defined type).
- (Local) movement primitives.
- Set of formally defined robot tasks, with conditional variants, that embed social norms.

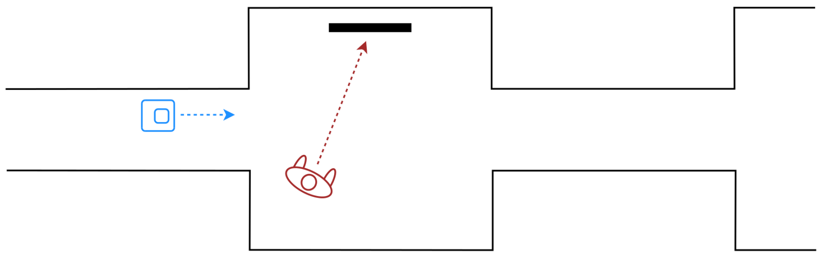


Figure 4: Micro-scenario 1

5 Scenarios

Until now we were only concerned about the SPENCER architecture in terms of data and control flow, as well as message definitions for data exchange between modules. Now we would like to discuss concrete scenarios to elaborate more on information exchange between modules, we start with some micro-scenarios below, and continue with some more complex scenarios in Sections 5.2– 5.4.

5.1 Micro scenarios

In Figure 4 the robot is moving through the lobby towards its goal and encounters a person crossing its path. The robot has to avoid this person. The navigation module has to determine whether or not the person is moving towards the screen shown in the figure for better human-aware motion planning. It expect the following information to be provided by the perception modules and map server:

- Position and direction of the person grounded in geometric map.
- Velocity of the person.
- Position of the desk grounded in the geometric map.

The second micro-scenario is shown in Figure 5. The robot is waiting for a person or group that it can help. A group of persons is in visible range of the robot. For the task planner to decide whether it should react to this group or not it expects the following information from the perception and social processing modules and the map server.

- A list of all persons within the visible range of the robot, with their position and orientation, with some (not too dynamic) IDs attached to all the persons.
- Groups of persons provided by the social processing module with IDs of persons and groups.
- Spokesperson ID from the group (if any).
- Information about desk and its position in the map.
- Position of the robot and its orientation.

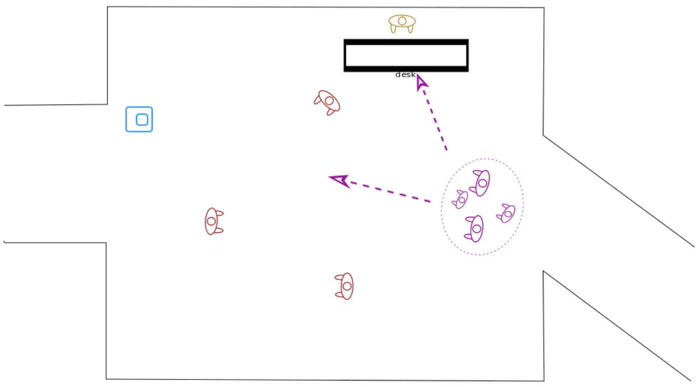


Figure 5: Micro-scenario 2

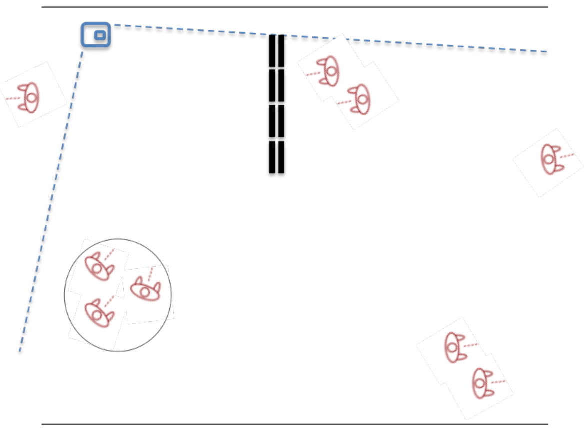


Figure 6: Micro-scenario 3

The following third micro-scenario concerns a wide corridor with information signs. The position of the robot could be such that it has a good view angle of passengers that approach the information signs. Those passengers are more likely to need assistance than passengers that are just walking by and could be approached by the robot.

This scenario involves:

- Tracking of individual passengers.
- Group detection by the social processing module.
- Social processing input (e.g., gaze and movement trajectories) for the passengers within a group for spokesperson detection.
- Information about the positions (and, possibly, the types) of the information signs on the map.

For evaluation of the SPENCER system, we have decided to define full scenarios for robot guiding tasks in the airport. The following descriptions of these scenarios are written from the perspective of the end user and are mainly relevant for the high-level task planner. These scenarios will provide a

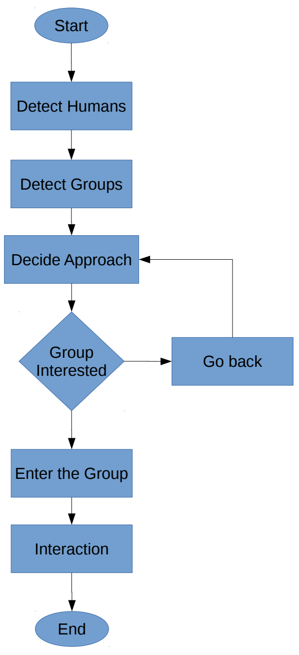


Figure 7: Full scenario 1

reference for SPENCER partners to discuss how they see what their respective software components will do in order to achieve the goals defined in the scenarios.

5.2 Scenario 1: Robot proactively reaching a group (of about 3 persons)

First full scenario is shown in Figure 7.

- Start condition: robot in interacting mode (perhaps showing a message on the screen).
- End condition: the robot starts an interaction with the group (other ending conditions could be request from the terminal to go to a specific place or problems to the robot).

State descriptions:

- Detect Groups:
  The robot is detecting groups of person, looking for people interested in interacting.
- Approach:
  The robot approaches the selected group.
- Enter the group:
  If the group shows signs of being interested in the interaction the robot completes its approaching phase, getting at an interaction distance.

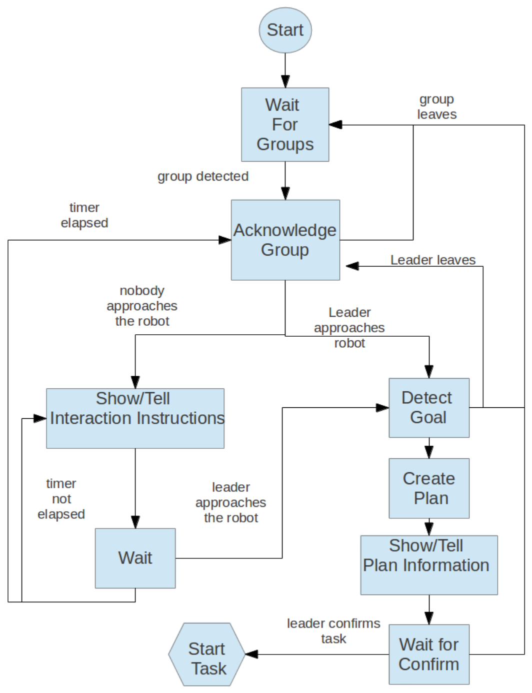


Figure 8: Full scenario 2

- Interact:
  The robot engages the group (see Scenario 2 for a detailed explanation of this phase).

5.3 Scenario 2: Robot available for interaction (i.e. Robot responding to an initial request from a group which is at several meters)

In this scenario (Figure 8) the robot is waiting for groups of people to approach him and start interacting. It will not actively navigate the area to engage people.

- Start condition: robot in interacting mode (perhaps showing a message on the screen).
- End condition: the robot starts an interaction with the group (other ending conditions could be request from the terminal to go to a specific place or problems to the robot).

State descriptions:

- Wait for Groups:
  The robot is not moving, or perhaps just having some basic routines of movement to position himself in different places in the area according to its needs. The robot analyzes the people surrounding him, looking for groups that are interested in interacting with it. There is also the possibility that a KLM staff member brings a group to the robot and drives the interaction process.

- Acknowledge Groups:
  A group is looking at the robot and approaching him. The robot starts turning its head to show that he has seen the group and while the group is approaching he turns his torso toward them. The robot then simply waits, looking at different members of the group (perhaps spending more time looking at possible leaders).
- Show/Tell Interaction Instructions:
  The group of people don't approach the robot close enough for interaction. The robot engages them, showing a message and speaking, to instruct them on how to interact with it.
- Wait:
  The robot waits for people to approach him for interaction. From this state it can try to engage them, going back to the show/tell info state or simply go back to acknowledge and look at them.
- Detect Goal:
  The group leader is approaching the robot to interact. He could push a button and then the goal detection phase would start (see notes after the state description)
- Create Plan:
  The robot creates a high-level plan to guide the group to its destination. This way, it can show the people how long it will take to guide them, if the task is feasible and perhaps the route that will be followed. In this phase the robot can also contact the airport system to inform that the group of passengers is arriving within an estimated time. The airport system could return an 'OK' , also giving the robot a maximum allowed delay, or a 'REJECT' making the robot abandon the task.
- Show/Tell Planning Information:
  The robot shows the group information related to the plan. If the plan is rejected it says that it can't guide the group, otherwise it may inform the passengers about the estimated time to reach the gate and give them information to place the luggage in the opposite space.
- Wait For Confirm:
  The robot waits for the leader to confirm the start of the task, perhaps by pressing a button on its screen. There is also the possibility that a KLM staff member confirms the start of the task.
- Go:
  The robot starts guiding the passengers.

Goal Detection Phase: There are different ways to handle this phase of the robot. Here are some possibilities (not necessarily mutually exclusive):

- Everybody in the group presents their boarding pass.
  - Advantages:
    * Clear understanding of the members of the group. Possibility to estimate subgroups based on surnames.
  - Disadvantages:
    * Engagement phase slower.

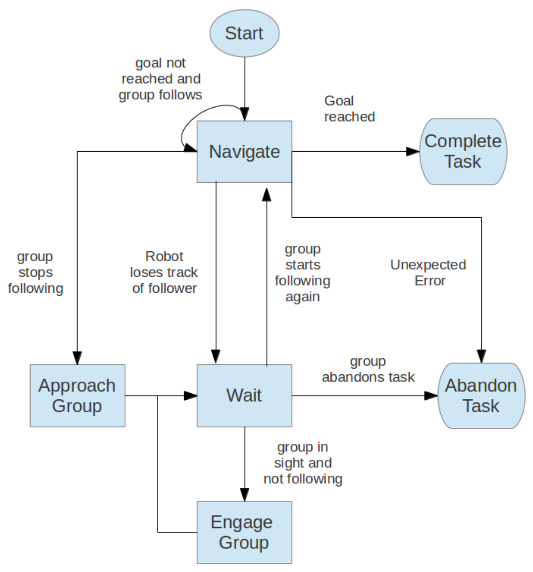


Figure 9: Full scenario 3

    * May lead to to situations where the robot must guide passengers with different goals, which leads to more interesting but complicated robot behaviors.

- The leader of the group presents the boarding pass
  - Advantages:
    * Fast engagement phase.
  - Disadvantages:
    * No clear understanding of the members of the group who is following the group.
- The leader of the group inputs the destination on the screen.
  - Advantages:
    * Doesn’t require boarding passes.
  - Disadvantages:
    * The user can do mistakes and input a wrong destination.

5.4 Scenario 3: guiding a group of people

In the scenario shown in Figure 9 the robot is guiding a group of people toward a goal. To simplify the scenario there are no obstacles in its path and the robot doesn’t have the luggage of the group.

- Start condition: robot in guiding mode. Group of people engaged and following.
- End condition: robot and group reach the goal or task is abandoned.

State descriptions:

- Navigate:
  The robot navigates the areas, following the plan already calculated in the engagement phase.
- Approach Group:
  If the group stops while following the robot, it turns back and approaches them.
- Wait:
  If the robot loses track of its followers or they stop and he his near them it could decide to wait for the passenger, depending on the distance to the gate and on the departure time.
- Engage Group:
  If the group has stopped following the robot and they are nearby him the robot can ask them if they have problems.
- Complete Task:
  The robot has reaches its goal with the group. The task is completed. If the robot has the luggage of some 'lost followers' it could place it at the destination gate.
- Abandon Task:
  There is a fail in the robot system or the group has abandoned the task (perhaps because they were away for too long or they tell the robot they are not interested anymore). If the robot has the luggage of some 'lost followers' it could place it at the destination gate.

6 Tools for SPENCER software

While defining the software architecture of the SPENCER-system, we also want to define software tools that will be used during the project.

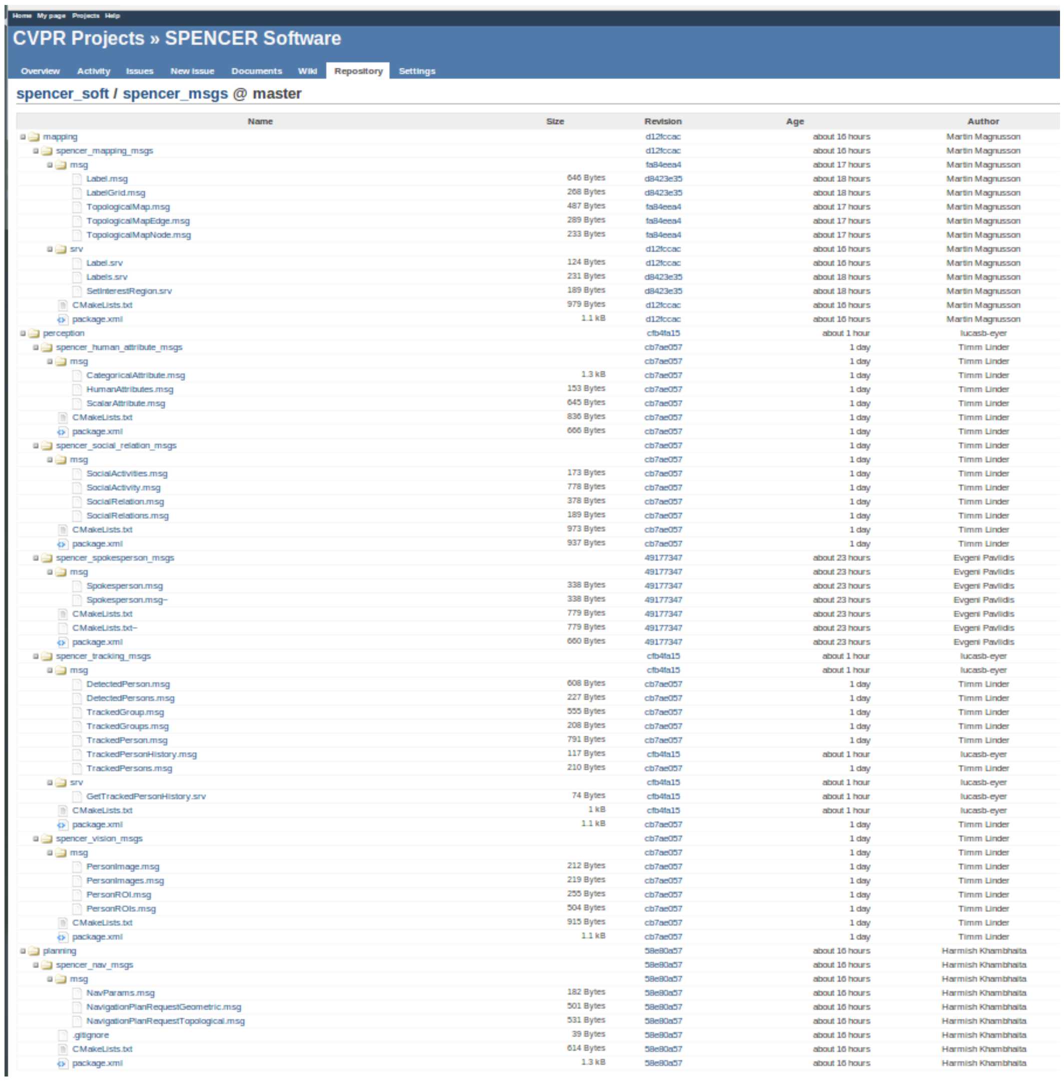
We have decided to use the following operating system and middle-ware the first SPENCER-system,

- Ubuntu 12.04 (with kernel 3.2 required for camera drivers)
- ROS Hydro as middle-ware
- Catkin as central build system
- Use TUM-redmine server as the central git repository for software

For easier software-module management we will separate the software repositories by conceptually different tasks. We will use tools like `rosinstall` and `wstool` as a way to provide easy instructions to clone and update multiple repositories.

We are looking into using the *MoveIt!* packages[4] with *Object Recognition Kitchen*[5] as a framework for manipulation and navigation planning. We are considering the use of these packages for

[4] `http://moveit.ros.org`
[5] `http://wg-perception.github.io/object_recognition_core`


Home My page Projects Help
CVPR Projects » SPENCER Software
Overview Activity Issues New issue Documents Wiki Repository Settings

spencer_soft / spencer_msgs @ master

Name	Size	Revision	Age	Author
mapping		d12fccac	about 16 hours	Martin Magnusson
spencer_mapping_msgs		d12fccac	about 16 hours	Martin Magnusson
msg		fa84eea4	about 17 hours	Martin Magnusson
Label.msg	646 Bytes	d8423e35	about 18 hours	Martin Magnusson
LabelGrid.msg	268 Bytes	d8423e35	about 18 hours	Martin Magnusson
TopologicalMap.msg	487 Bytes	fa84eea4	about 17 hours	Martin Magnusson
TopologicalMapEdge.msg	289 Bytes	fa84eea4	about 17 hours	Martin Magnusson
TopologicalMapNode.msg	233 Bytes	fa84eea4	about 17 hours	Martin Magnusson
srv		d12fccac	about 16 hours	Martin Magnusson
Label.srv	124 Bytes	d12fccac	about 16 hours	Martin Magnusson
Labels.srv	231 Bytes	d8423e35	about 18 hours	Martin Magnusson
SetInterestRegion.srv	189 Bytes	d8423e35	about 18 hours	Martin Magnusson
CMakeLists.txt	979 Bytes	d12fccac	about 16 hours	Martin Magnusson
package.xml	1.1 kB	d12fccac	about 16 hours	Martin Magnusson
perception		cfb4fa15	about 1 hour	lucasb-eyer
spencer_human_attribute_msgs		cb7ae057	1 day	Timm Linder
msg		cb7ae057	1 day	Timm Linder
CategoricalAttribute.msg	1.3 kB	cb7ae057	1 day	Timm Linder
HumanAttributes.msg	153 Bytes	cb7ae057	1 day	Timm Linder
ScalarAttribute.msg	645 Bytes	cb7ae057	1 day	Timm Linder
CMakeLists.txt	836 Bytes	cb7ae057	1 day	Timm Linder
package.xml	666 Bytes	cb7ae057	1 day	Timm Linder
spencer_social_relation_msgs		cb7ae057	1 day	Timm Linder
msg		cb7ae057	1 day	Timm Linder
SocialActivities.msg	173 Bytes	cb7ae057	1 day	Timm Linder
SocialActivity.msg	778 Bytes	cb7ae057	1 day	Timm Linder
SocialRelation.msg	378 Bytes	cb7ae057	1 day	Timm Linder
SocialRelations.msg	189 Bytes	cb7ae057	1 day	Timm Linder
CMakeLists.txt	973 Bytes	cb7ae057	1 day	Timm Linder
package.xml	937 Bytes	cb7ae057	1 day	Timm Linder
spencer_spokesperson_msgs		49177347	about 23 hours	Evgeni Pavlidis
msg		49177347	about 23 hours	Evgeni Pavlidis
Spokesperson.msg	338 Bytes	49177347	about 23 hours	Evgeni Pavlidis
Spokesperson.msg~	338 Bytes	49177347	about 23 hours	Evgeni Pavlidis
CMakeLists.txt	779 Bytes	49177347	about 23 hours	Evgeni Pavlidis
CMakeLists.txt~	779 Bytes	49177347	about 23 hours	Evgeni Pavlidis
package.xml	660 Bytes	49177347	about 23 hours	Evgeni Pavlidis
spencer_tracking_msgs		cfb4fa15	about 1 hour	lucasb-eyer
msg		cfb4fa15	about 1 hour	lucasb-eyer
DetectedPerson.msg	608 Bytes	cb7ae057	1 day	Timm Linder
DetectedPersons.msg	227 Bytes	cb7ae057	1 day	Timm Linder
TrackedGroup.msg	555 Bytes	cb7ae057	1 day	Timm Linder
TrackedGroups.msg	208 Bytes	cb7ae057	1 day	Timm Linder
TrackedPerson.msg	791 Bytes	cb7ae057	1 day	Timm Linder
TrackedPersonHistory.msg	117 Bytes	cfb4fa15	about 1 hour	lucasb-eyer
TrackedPersons.msg	210 Bytes	cb7ae057	1 day	Timm Linder
srv		cfb4fa15	about 1 hour	lucasb-eyer
GetTrackedPersonHistory.srv	74 Bytes	cfb4fa15	about 1 hour	lucasb-eyer
CMakeLists.txt	1 kB	cfb4fa15	about 1 hour	lucasb-eyer
package.xml	1.1 kB	cb7ae057	1 day	Timm Linder
spencer_vision_msgs		cb7ae057	1 day	Timm Linder
msg		cb7ae057	1 day	Timm Linder
PersonImage.msg	212 Bytes	cb7ae057	1 day	Timm Linder
PersonImages.msg	219 Bytes	cb7ae057	1 day	Timm Linder
PersonROI.msg	255 Bytes	cb7ae057	1 day	Timm Linder
PersonROIs.msg	504 Bytes	cb7ae057	1 day	Timm Linder
CMakeLists.txt	915 Bytes	cb7ae057	1 day	Timm Linder
package.xml	1.1 kB	cb7ae057	1 day	Timm Linder
planning		58e80a57	about 16 hours	Harmish Khambhaita
spencer_nav_msgs		58e80a57	about 16 hours	Harmish Khambhaita
msg		58e80a57	about 16 hours	Harmish Khambhaita
NavParams.msg	182 Bytes	58e80a57	about 16 hours	Harmish Khambhaita
NavigationPlanRequestGeometric.msg	501 Bytes	58e80a57	about 16 hours	Harmish Khambhaita
NavigationPlanRequestTopological.msg	531 Bytes	58e80a57	about 16 hours	Harmish Khambhaita
.gitignore	39 Bytes	58e80a57	about 16 hours	Harmish Khambhaita
CMakeLists.txt	614 Bytes	58e80a57	about 16 hours	Harmish Khambhaita
package.xml	1.3 kB	58e80a57	about 16 hours	Harmish Khambhaita

Figure 10: Screenshot of spencer software repository

future development of SPENCER software, because they provide a framework for message and data flow between components, as well as a convenient way to configure 3D perception sensors, switch planning algorithms, and low-level controller management. We will decide later in the project whether to use MoveIt! as configuration and communication framework for SPENCER.

7 Message definitions for software modules

During the first integration week of spencer, we have already started defining ROS messages and services for intermodule communication for the modules specified in the architecture. These message and service definitions are stored in a common software repository maintained at the TUM-redmine server. Figure 10 shows the structure of the repository.

8 First SPENCER Integration Week

This deliverable is also a great team effort by the SPENCER partners and the result of the first integration week which took place from April 7 to April 11, 2014, in Freiburg and Toulouse. Following the SPENCER integration strategy to have plenary and decoupled integration events, this week was a *decoupled integration event* in which the consortium split into two groups. The strategy is motivated by the experience that integration in smaller groups is typically more efficient. In total there are five integration weeks in SPENCER, two decoupled and three plenary events.

Impressions of the integration week are shown in Fig. 11.

Figure 11: The first SPENCER integration week. Top row: the perception group (ALU-FR, RWTH, TUM, UT) met in Freiburg. Bottom row: the planning and navigation group (CNRS, ORU, ALU-FR) met in Toulouse.