Deliverable D5.1
Iterative Integration and Evaluation Report

Consortium

UNIVERSITEIT VAN AMSTERDAM (UvA)
YDREAMS - INFORMATICA S.A. (YD)
IDMIND - ENGENHARIA DE SISTEMAS LDA (IDM)
UNIVERSIDAD PABLO DE OLAVIDE (UPO)
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE (ICL)

Grant Agreement no. **288235**
Funding Scheme: **STREP**

Project	
Project acronym:	FROG
Project full title:	Fun Robotic Outdoor Guide
Grant agreement no.:	288235
Funding scheme:	STREP
Project start date:	1 October 2011
Project duration:	30 September 2014
Call topic:	ICT-2011.2.1 Cognitive Systems and Robotics (a), (d)
Project web-site:	www.frogrobot.eu
Document	
Deliverable number:	D5.1 - revised
Deliverable title:	Iterative Integration and evaluation report
Due date of deliverable:	M36
Actual submission date:	M36 –30 September 2014
Editors:	
Authors:	UT, UPO, IDM
Reviewers:	UT, UPO, IDM
Participating beneficiaries:	All Partners
Work Package no.:	5
Work Package title:	Integration, Evaluation and Demonstrator
Work Package leader:	IDM
Work Package participants:	All Partners
Estimated person-months for deliverable:	
Dissemination level:	Public
Nature:	Report
Version:	Revised
Draft/Final:	Final
No of pages (including cover):	40
Keywords:	Integration, State Machine, Augmented Reality

D5.1 – Iterative Integration and Evaluation Report

Addendum – Iteration 2 - M36.

This report deals solely with the software Integration Framework. The first iteration of the deliverable, written by YDreams in April 2012, was in accordance with a request made during the negotiation sessions. It shows that, at the time, YVision was a good choice as Integration Platform.

During the Second Year Review YDreams reported that YVision was no longer being supported in the form that it had previously known and that, for the FROG Integration Framework, they would move over to their new YVision4Unity platform. YDreams left the project at the end of December 2013 (M27) without having satisfactorily implemented this framework on the FROG robot.

In the second half of April 2014, Randy Klaassen and Jan Kolkmeier (both from UT) took on the work of implementing an Integration Framework and AR Content for the FROG project. The behaviour and decisions of the robot have been implemented as a State Machine written in Python (2.7) running on the FROG's Interaction Computer. Content (the Front end) has been implemented in Unity. The ROSBridge[1] protocol over WebSockets has been used to send JSON-based commands to (other) ROS components and to subscribe to ROS topics to receive data necessary to implement the State Machine. Based on the received data and the state of the robot, the State Machine will decide what its next action will be.

UPO provided two simulators (described in D5.2). One was using recorded sensor and navigation data and one was given the possibility to navigate with a virtual robot on the map of the Royal Alcázar. Both simulators were used to test the implementation of the State Machine without connecting to the actual robot. These simulators were also used to simulate a mission on one of the FROG robots in order to test content.

The following pages describe the implementation of the new Integration Framework, the FROG State Machine.

Please note: This Framework has been used in integration and data collection sessions in Seville in May, June and September 2014. Evaluation of this framework in deployment will be reported in deliverables D5.4, D2.4 and Milestone MS4.

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

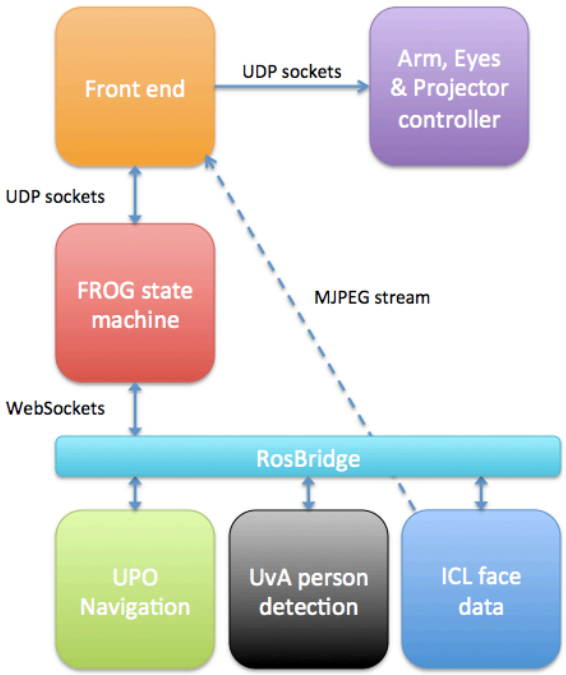
FROG State Machines

Randy Klaassen, 10 September 2014

Introduction

A state machine will specify the tour of the FROG. In this document we present details of the state machine and describe the behaviour in the different states. Transitions between states are the result of messages received from the robot via the ROSBridge or commands received from the front end. The front end is responsible for (dis)playing content on the FROG robot and has been developed using the Unity platform.

The FROG State Machine is written in Python 2.7. The state machines are specified using Fysom Python Finite State Machine. The FROG State Machine communicates with the other components of the FROG using a ROSBridge websocket server. Ws4py implementation is used to send and receive messages via the ROSBridge. We designed and implemented a protocol to communicate between the FROG state machine and the front end by a using UDP sockets. The arm (antenna), eyes and projector of the FROG are driven by their own controllers and are controlled by the front end.



Points of Interest

Six different Points of Interest (POI) have been identified. The robot will (dis)play content at each of these POIs. A map of the Royal Alcázar including the

POIs can be found in Figure 1. Table 1 gives an overview of the coordinates of the POIs.

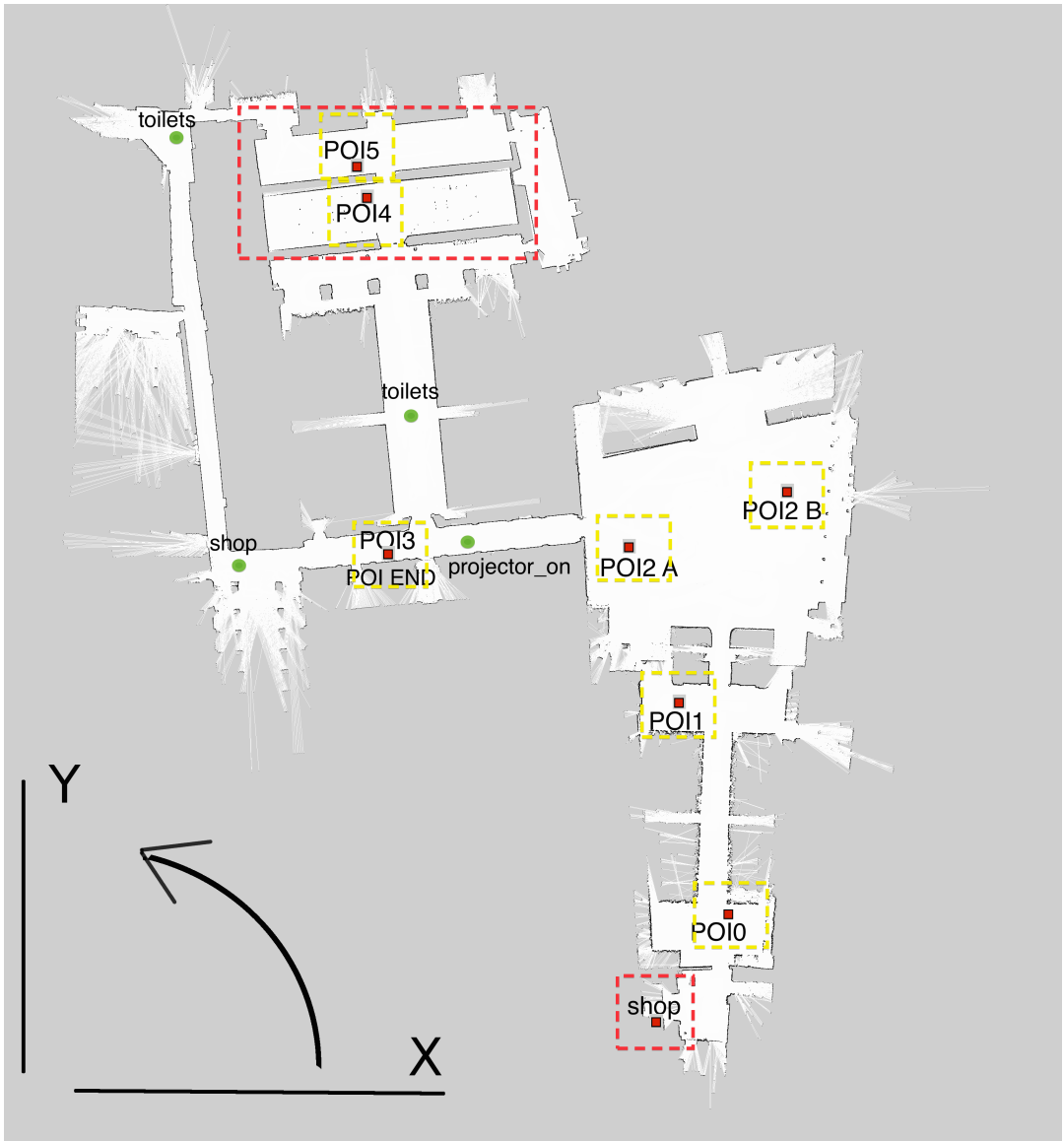


Figure 1 Map of the Royal Alcázar with the POIs

Table 1 Coordinates of the POIs

POI	X	Y	Z	W	Name
SHOP	95.89384	17.22943	-0.71942	0.69456	Shop
POI0	104.0537	32.9829	-0.89019	0.45226	Lion's Courtyard
POI1a[2]	97.19	63.02	0.898316	0.43934	Almohad wall
POI1b[3]	97.19	63.02	-0.56311	0.82638	Almohad wall
POI2a[4]	89.63559	87.67342	-0.77982	0.62599	Hunting Courtyard
POI2b[5]	114.4858	94.9391	-0.58442	0.80969	Hunting Courtyard

[2] time <= 13pm
[3] time > 13pm
[4] time <= 12pm

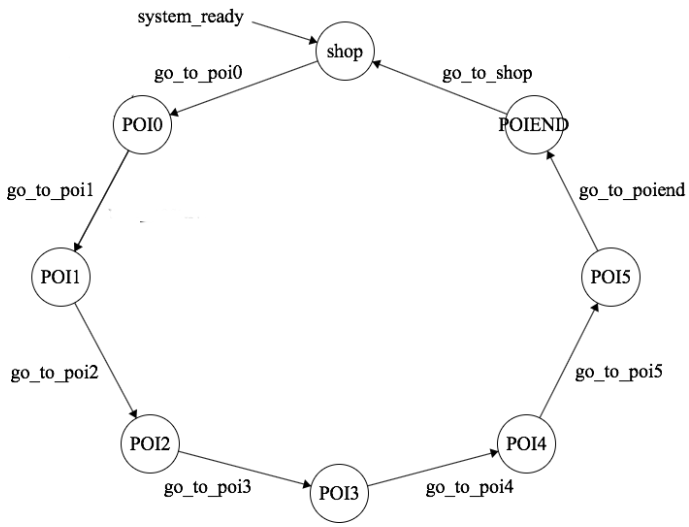
POI3	55.76009	85.4927	0.741422	0.671083	Crossing Courtyard
POI4	51.7105	135.9988	0.31531	0.94898	Tapestry Room
POI5	52.08933	140.9211	0.79449	0.60727	Vault Room
TEMP	*25.79437*	*144.70933*	*-0.76849*	*0.63985*	*Functional waypoint*
END	36.49437	83.50933	-0.76849	0.63985	Mounting block

State Machines

A complete tour of the robot will visit all of the POIs. When docked the robot is located in the shop near the entrance of the Alcázar. The tour starts at POI0 and when the last POI has been visited the robot will return to the shop for charging. A high level overview of the robot tour can be found in Figure 2. Three state transitions are possible:

1. `system_ready` -
2. `go_to_poi(x)` - go to the next POI by sending `/GotoWayPoint(POIx)` to the RosBridge

In the next section we will discuss the POIs in more detail.



Shop

The robot tour starts at the shop, where the robot will have its docking station. The shop is located at the entrance of the Royal Alcázar. As this is a crowded place, we will not look for potential participants for a tour at this location. This first thing to do is to move the robot to a location where it can start looking for participants. This location is called POI0 (Lion's Courtyard). Figure 2 shows the state machine that describes this scenario. `Frontend_idle` is a command sent by the front end when the system is ready to start the tour. After receiving this command, the robot will move to `POI0` (`go_to_poi0`) at the beginning of the Lion's Courtyard.

---

[5] time > 12pm

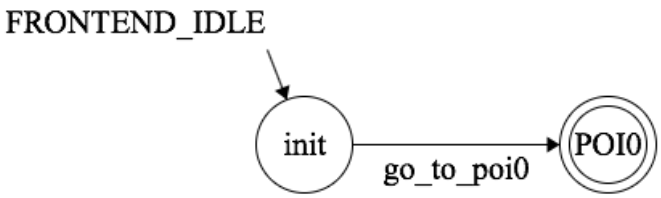


Figure 2 State machine @shop

Prior to being sent to the first Point of Interest, the FROG needs to be undocked from its docking station by sending an undock command. When the undocking has been successfully executed the State Machine receives an 'undocked' message from the FROG. When this message is received the State Machine will send a `go_to_poi0` command.

POI0

Point of Interest 0 is a special POI. At POI0 we can start looking for participants for the tour. Figure 3 shows the state machine that describes the process for finding and approaching participants, and for introducing the robot to a potential participant for a tour. After the introduction, participants can indicate whether they would like to join the tour or not by pressing one of the buttons on the screen. If the participant wants to join a tour, the tour will start by navigating to POI1. If the participants do not want a tour, the robot starts looking for new potential participants.

/start_introduction_ICL
During the introduction the tour guide will introduce the FROG robot and explain what the visitor can expect during the tour. The tour guide explains that the engagement level of the user can be measured during the interaction with the FROG and users have some time to play and try out the ICL face data stream. The result of the ICL face data stream will affect the way in which the robot asks the user to take a guided tour. If the user does not seem to be engaged (-1 ≤ engagement level < 0) the robot will ask the user to take a tour by asking the following questions: "*It seems you are not really interested in FROG. But maybe we measured wrong. Do you want to join us on a guided tour?*" If the user seem really engaged (0 ≤ engagement level ≤ 1)
State transitions possible here are: the robot asks the user to take a tour using the following sentences: "*It seems you are very interested in FROG! Do you want to join us on a guided tour?*"

/start_introduction
During the introduction the tour guide will introduce the FROG robot and explain what the visitor can expect during the tour. The tour guide explains that the engagement level of the user can be measured during the interaction with the FROG.

Figure 3 shows the state machine and the possible state transitions at POI0. The transitions will be explained here in more detail:

1. `person_detected` – receive `/UvA_PersonDetectionAndPose` with orientation towards the robot `(135° < orientation < 225°)` via RosBridge
2. `person_not_detected` – do not receive `/UvA_PersonDetectionAndPose` with orientation towards the robot `(135° < orientation < 225°)`
3. `person_approached` – received a positive response (`FROG_response = 2`) after sending an `/ApproachPeopleFar` command with ID of person who was detected to the RosBridge.
4. `person_not_approached` – received a negative response (`FROG_response = 3` or `4`) after sending an `/ApproachPeopleFar` command with ID of person who was detected to the RosBridge. Try to find new participant.
5. `start_intro` – when a participant is successful approached, send a `start_introduction(_icl)` command to the Unity front end to start the introduction.
6. finish_intro – when the Unity front end has finished the introduction, a `finish_introduction` command will be received
7. `user_want_tour` – when the participants want to join the tour, a `user_positive` command will be received from the Unity front end. The tour will start by navigating to POI1
8. `user_dont_want_tour` - when the participants do not want to join the tour, a `user_negative` command will be received from the Unity front end. Try to find new participant
9. `go_to_POI1` – tour will start at POI1 by sending a `/GotoWayPoint(POI1)` over the RosBridge

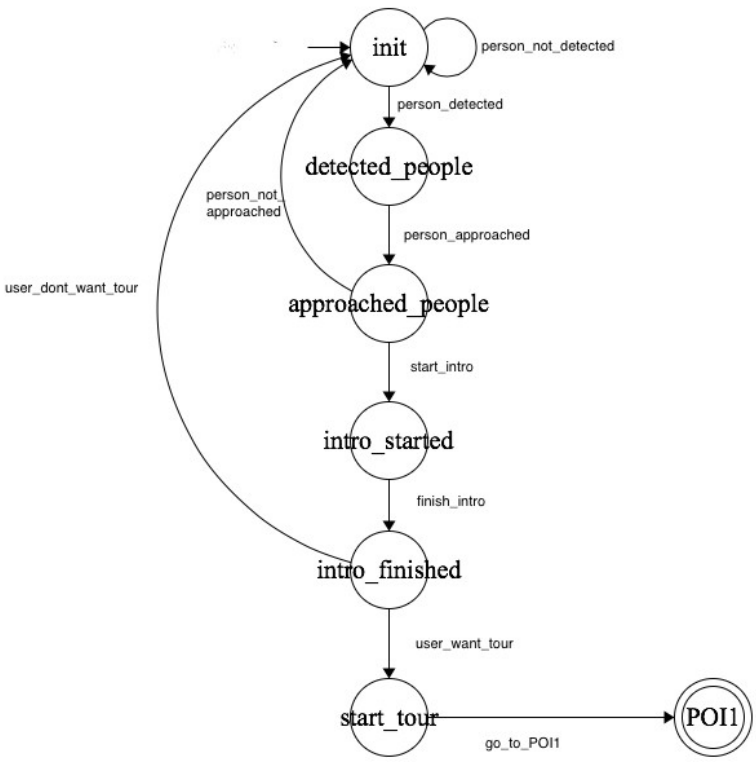


Figure 3 State machine of POI0

POI1

Point of interest 1 is located near the Almohad Wall. At this Point of Interest the robot tells about the Lion's Courtyard, it uses location based augmented reality to show the Hall of Justice and the Stucco Patio which are not accessible for the robot. The ICL face data stream is used to measure how much the users are engaged by the story told by the FROG. If the user seems to be engaged by the story told by the FROG (0 ≤ engagement level ≤ 1) more will be told about POI1 before going to POI2. When users are not engaged by the story told by FROG, a part of the story will be skipped and the tour will go straight to the next POI.

The Face data stream is started after finishing the navigation task and before starting the content of the POI. It is stopped when the content is finished. More details about the ICL face data stream and the use of it can be found in Section "ICL FaceDataStream". Figure 4 presents the state machine and possible state transitions for POIs that use the ICL face data stream.

The transitions will be explained here in more detail:

1. `start_poix` – When the FROG has finished its navigation task successfully it sends a `start_ poix` command to the front end to start the story of POIX.
2. `not_start_POI2` - When the FROG has finished its navigation task successfully but no participant is following the tour, FROG will go back to POI0 to try and find a new participant
3. `tell_more_poix` – When the engagement level of the user is between 0 and 1 FROG should present more details. It sends a `start_POIXb` command to the front end to start the extra content for that Point of Interest.
4. `finish_poix` – When the engagement level of the user is between -1 and 0, or when part two of POIX is finished. Send a `finish_POIX` command to the front end to stop the content of POIX.
5. `go_to_poix+1` – The tour will continue with POIX+1 by sending a `/GotoWayPoint(POIX+1)` to the RosBridge.

POI2

At POI2 the robot will stop in the Hunting Courtyard. Based on the time of the day the FROG will stop at POI2a or POI2b (see Figure 1). The ICL face data stream is used to measure how much the users are engaged by the story told by the FROG. If the user seems to be engaged by the story told by the FROG (0 ≤ engagement level ≤ 1) FROG will tell some more about POI2 before going to POI3. If the users are not engaged by the story told by FROG, part of the story will be skipped and the tour will go to the next POI.

Figure 4 presents the state machine and possible state transitions for POIs that use the ICL face data stream. State transitions possible here are the same as discussed at POI1.

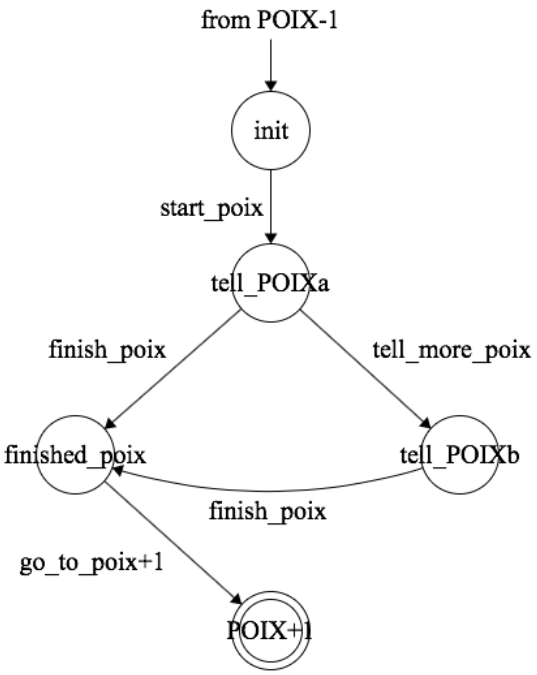


Figure 4 State machine of POIX with use of ICL face data stream

POI3

Point of Interest 3 is located near the entrance of the Crossing Courtyard. FROG will stop near a wall to use its projector to present a story about the Crossing Courtyard. The projector needs approximately 20 seconds to heat up to be ready to use. While navigating to POI3 and when FROG is close to POI3 the state machine sends a `projector_on` message to the projector controller to turn on the projector and show a blank screen.

Figure 5 presents the state machine and possible state transitions for POIs that are not using the ICL face data stream.

The transitions will be explained here in more detail:

1. `start_poix` – When the FROG has finished its navigation task successfully it sends a `start_poix` command to the front end to start the story of POIX.
2. `finish_poix` - When the content of POIX is finished, the state machine receives a `finsih_POIX` command from the front end.
3. `go_to_poix+1` – The tour will continue with POIX+1 by sending a `/GotoWayPoint(POIX+1)` to the RosBridge

POI4

Point of Interest 4 is located in the Tapestry Room. FROG will use location based augmented reality (AR) to explain the map of southwest Europe by drawing the flags and names of the countries on the map and rotating the map so that north is at the top.

Figure 4 presents the state machine and possible state transitions for POIs that are not using the ICL face data stream. State transitions possible here are the same as discussed for POI3.

POI5

Point of Interest 5 is located in the Vault Room where FROG will tell something about the room and play a quiz with its users.
Figure 4 presents the state machine and possible state transitions for POIs that are not using the ICL face data stream. State transitions possible here are the same as discussed at POI3.

POI END

POI END is the last Point of Interest of a complete FROG tour. FROG explains that this is the end of the tour, but that there is much more to explore on your own or with a human tour guide.
When this Point of Interest is finished the FROG will go back to the shop. When it arrives at the shop the State Machine will send a docking command and the FROG will drive to its docking station to charge. Figure 4 presents the state machine and possible state transitions for POIs that are not using the ICL face data stream. State transitions possible here are the same as discussed at POI3.

Location based

The State Machine keeps track of the current location of the FROG robot, which makes it possible to start content and change the behaviour of FROG while navigating in certain areas or interesting points between two Points of Interest. Three types of location-based adaptation of the behaviour of FROG or content have been implemented.

The behaviour of FROG will be adapted when the robot is in range of a Point of Interest to inform the users that it will soon finish navigating. FROG stops saying "*navigating to next location, please follow me*". Users will know that a new point of interest has almost been reached so they can be prepared for the start of the content when the POI is reached. The ranges of the POIs can be found in Figure 1 and are indicated by the yellow dashed lines around the POIs. These ranges can be adapted.

Another feature that can be adapted based on the location is the volume of FROG's audio. When FROG is indoors (near the docking station in the shop (volume is set to 50%) and in the Tapestry Room and Vault Room (volume is set to 80%)) it should lower its volume level so as not to disturb or bother other visitors, clients and (human) tour guides. The red dashed lines in figure 1 show the areas in which the volume is reduced.

A last feature that has been implemented based on the location of the FROG is the start of special content around places that can be useful for the visitors to know. FROG will tell the visitors where they can find the toilets, shop and bar while driving near these places.

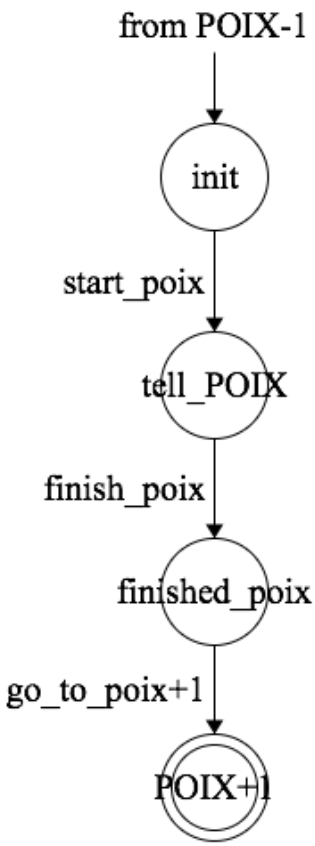


Figure 5 POIX

RosBridge

The FROG State Machine communicates with the other components via the RosBridge over WebSockets. The FROG State Machine is subscribed to the following topics to receive data on.

- `/diagnostic` – to keep track of the status of the electronics.
- `/UvA_PersonDetectionAndPose` – to react on the detection of persons.
- `/FROG_response` – to check the status of navigation commands.
- `/FROG_pose` – to keep track of the position of the FROG.
- `/ICL_FaceDataStream` – to receive the engagement levels and calculate the result for the engagement meter and for the decision whether or not more details need to be presented at POIs where the ICL face data stream is used.
- `/docking/state` – To keep track of the docking/undocking process. To start or finish a tour.
- `/PeopleUPO` – to react on the detection of persons by the laser scanner.

The FROG State Machine sends commands via the RosBridge to other components of the FROG for navigation, to control the ICL face data stream and to dock and undock the FROG. The following commands will be sent via the RosBridge

- `/GotoWayPoint` – to navigate the FROG to a position given the X, Y, Z and W coordinates.
- `/ApproachPeopleFar` – to approach a person that is detected at the entrance at the beginning of the tour given the ID of that person.
- `/ICL_command` – to start and stop the ICL face data stream.
- `/docking/dock` – to dock and undock the FROG at its docking station.

ICL FaceDataStream

Results of the ICL face data stream will be used for two purposes. It will be used to activate the engagement meter and to decide when the FROG should present more details about a Point of Interest or finish the current Point of Interest and continue the tour by navigating to the next Point of Interest.

The engagement meter will display the current engagement level of the user on the screen of the FROG when there is data available (when users are near the FROG and when users are facing the FROG). The engagement meter will be updated every two seconds. For every two seconds we determine the engagement level for the available data by using majority voting.

The decision for presenting more details about a Point of Interest is made based on the engagement level of the user. The results of the face data stream are stored from the start of the Point of Interest till the end of the first part of the content at a Point of Interest. Engagement levels are only stored when users are near the FROG and when users are facing the FROG. At the end of the first part of the content we calculate the engagement level of the first part of the content by using majority voting over the first and last 15% of the recorded data. These are the moments when users pay attention to FROG's screen.

Simulating missions

During the development of the FROG state machine and the front end we used the simulators from UPO to test the implementations (see D5.2). Two different simulators were used: one that was playing recorded sensor data from one of the FROG missions at the Royal Alcázar and one simulator that was capable of executing navigation commands and reacting on these commands by sending responses via the RosBridge and by showing a virtual FROG on a map of the Alcázar. Both simulators provide a RosBridge to connect and communicate with the simulator.

The front end, the FROG State Machine, and the controllers of the eyes, projector and antenna of the FROG, together with the UPO navigation simulator makes it possible to simulate complete missions of the FROG tour for Seville.

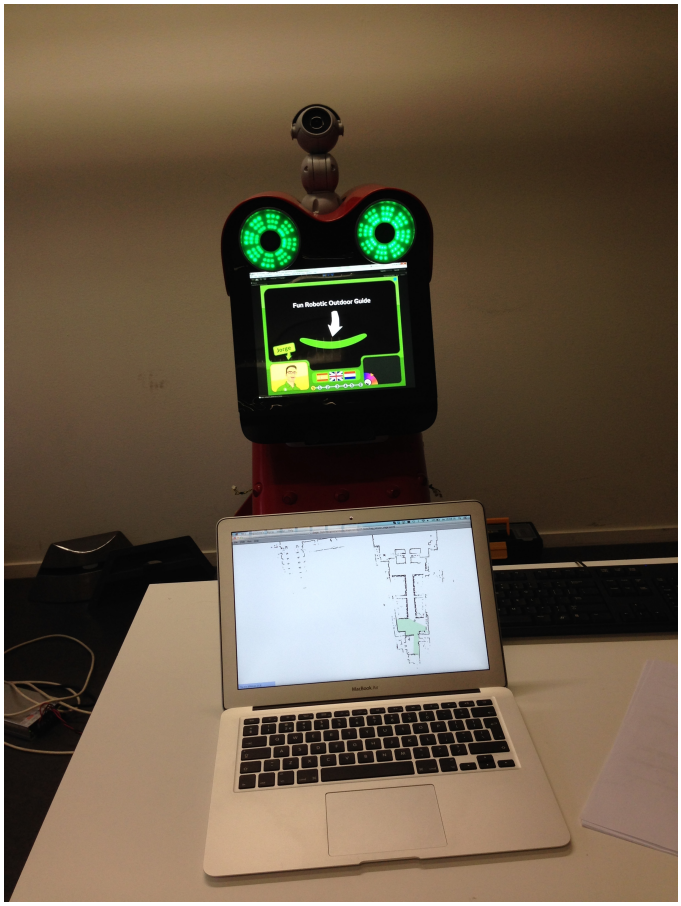


Figure 6 UPO navigation simulator, FROG state machine, the front end and the eyes, projector, and antenna controllers simulating a full tour for the Royal Alcázar

Contents

List of Figures

List of Tables

Abstract

This survey compares several software frameworks with the objective of selecting the most suitable to be adopted in the FROG EU project. The basic tools provided by these frameworks will be exploited during the project to construct algorithms in the areas of localization and navigation, people and emotion detection, and augmented reality to create an appealing interaction with tourists. The project objectives are presented and its modules are briefly described. The main characteristics of the studied frameworks are highlighted. The multimedia framework YVision is also presented and its key features emphasized. The key requirements for the FROG framework are identified and the advantages and disadvantages of each one are analysed. Finally, different solutions are presented and discussed.

1 Introduction

Since the construction of the first automatic machines, robotic agents have achieved an impressive progress, especially during the second half of the last century. Nowadays, they perform a number of tasks that range from industrial handling and home cleaning, to rescue missions in disaster sites and space exploration. Despite their notable evolution, service robots are still an emerging and promising market in areas so different as transportation of physically challenged people, tourist guiding, or telepresence.

The FROG project aims to develop an interactive outdoor robot guide for European cultural and historical sites. The project is divided into different areas, the navigation and localization algorithms, the detection of the affective and social behavior of tourists, and the interaction between the robot and the tourists. The FROG project covers a very wide range of vectors of knowledge. It needs to address the robot control problem, taking into account that the site might be full of tourists. But also needs to handle the identification and tracking of people, the detection of their emotional state, and the creation of an interesting personality for the robot. FROG resorts to state-of-the-art augmented reality techniques to offer the user appealing multi-media content about each point of interest.

Software frameworks provide functionalities that facilitate the development of computational applications. In particular, the FROG relies on these tools to provide the standard elements necessary for the development of new algorithms for each of the project vectors.

Each framework is target at a different area (e.g. robotics, multimedia applications, augmented reality). The development and wide-spreading of complex robots demanded great programming effort, from the driver-level software to navigation and path planning. To manage the complexity and increase the development speed of new robots, several robotic frameworks have been proposed. Such frameworks generally incorporate middleware, drivers, algorithms, tools, and utilities. A middleware is a resource that abstracts the user from low-level communication details. The aim of robotic frameworks is to perform the low-level interface with the hardware and to do high-level operations that give the robot awareness of the environment that surrounds it and enables it to act on that environment. It is desirable that a low-level layer should account for all the communications with the specific sensors and actuators, providing standardized data so that the implementation of the high-level algorithms does not depend on the sensor/actuator model and manufacturer, but only on the information provided. A planner or decisional layer is often also part of the software package. This layer is an event-driven supervisor that determines the sequence of tasks for execution and is reactive to events from the lower layers. It can be implemented by a discrete event system, such as finite state machines and petri nets. There are several robotic frameworks, each one meets the particular needs that the respective developers find most relevant. This work presents an analysis of some of the most widely adopted robotic frameworks, namely, Player, ROS, YARP, CARMEN, OROCOS, Orca, MRDS, URBI, MRPT, MOOS, OpenJAUS, ERSP, CLARAty, and GenoM.

Augmented reality denotes an online view of the real world augmented by computer-generated information such as sound, video, and graphics. This technology results in an enhanced perspective of the world by the user. This is far different from virtual reality, which replaces the real world with a simulated one. Total Immersion D'Fusion and Metaio Unifeye are augmented reality frameworks that provide the basic tools to develop augmented reality applications.

Game engines also provide useful tools such as rendering of 3D images and physics simulation. The game engine Unity provides tools for creating 3D video games or other interactive content such as architectural visualizations or real-time 3D animations.

In this survey, different frameworks will be evaluated with respect to important criteria for the project such as supported Operating Systems (OS), programming language, modular topology, communication protocol, license of distribution, availability of drivers for sensors and actuators, availability of control algorithms, quality of the support and when it was performed the last update to the framework. The existence of features such as a built-in discrete event system, image rendering, sound player, augmented reality, physics engine, statistical tools, and tracking of markers, faces and full bodies will also be studied.

We do not intend to point out the best framework, in fact, such framework, probably, does not exist, but rather to indicate the advantages and disadvantages of each one for the particular needs of the FROG project.

The remaining of this document is organized as follows. In Section 2 the FROG project is presented. Several frameworks will be described in Section 3. In Section 4 the key comparison criteria are identified and the frameworks evaluated. In Section 5, we will discuss the advantages and disadvantages of

different solutions. Finally, some concluding remarks will be presented in Section 6.

2 FROG Project

FROG (Fun Robotic Outdoor Guide) aims to develop a guide robot that engages tourists by exhibiting a friendly personality and behavior. The project comprises advances beyond the current state of the art in the areas of vision-detection, affective-computing, intelligent agent architecture, and dependable autonomous outdoor robot operation.

In order to meet its goals, the FROG robot requires a fascinating combination of "hard"-skills and "soft"-skills. The localization and navigation in crowded sites with irregular terrain presents a great challenge and is a totally open problem for the scientific community. These are "hard" requirements that call for new control and estimation techniques, and for efficient algorithms for human and obstacle detection. These problems need to be addressed in a robust way, always bearing in mind safety, especially for the humans, but also for the environment and for the robot itself. One of the most interesting aspects of this project is its "soft"-skills. The guiding task requires the assessment of the affective social behavior of the tourists. A captivating and friendly personality together with an appropriate reaction to the tourists behavior are key to meet the desired goals. Giving the robot these human-like behaviors is a major step forward the acceptance and integration of service robots into human communities and it will be one of the main contributions of the project.

The best way to address the project complexity and promote the reuse of the algorithms in other potential applications is by means of a modular architecture, where each module fulfils a specific task and where the modules communicate with each other through a middleware to exchange information. Figure 1 illustrates a provisional conceptual scheme of the FROG robot where some modules can be

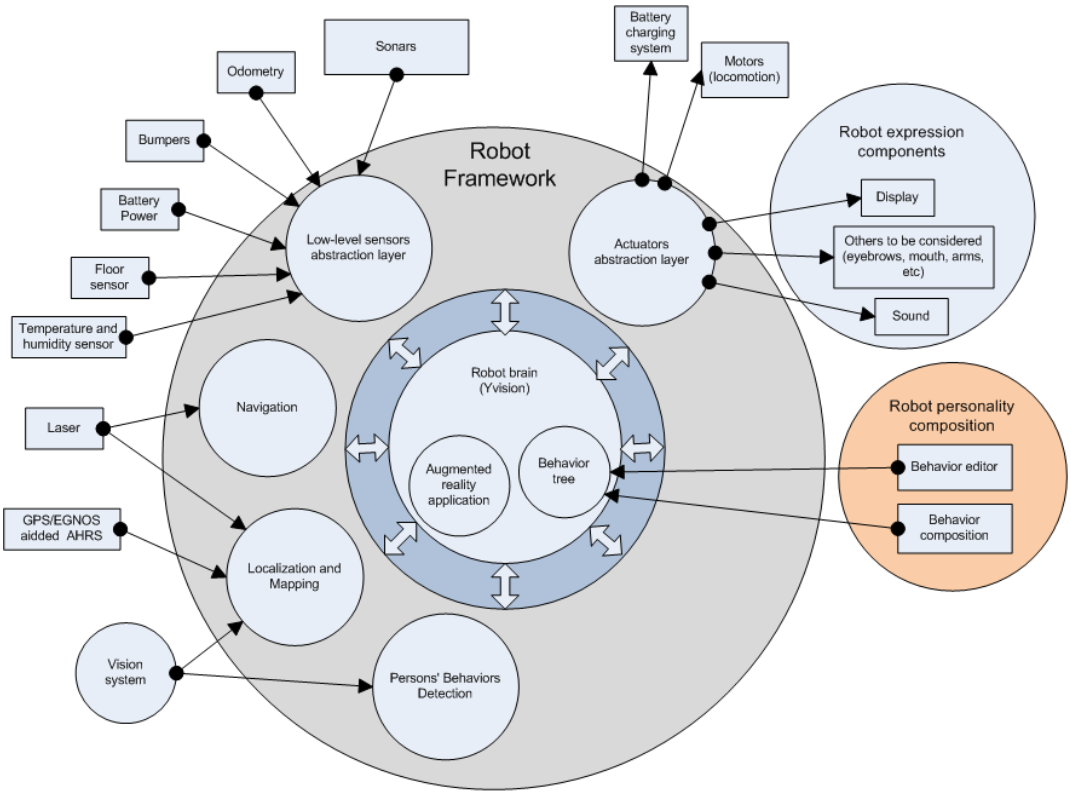


Figure 1: Provisional conceptual scheme of the FROG robot.

identified. The key components of the robotic software system that will be developed are individual modules that

- Detect obstacles, recognize certain objects such as pedestrians, and track them
- Constantly localize the robot in the environment,

- Detect affective behaviors of humans
- Plan routes for the robot, taking into account the humans and their behaviors
- Execute the planned trajectories considering all navigation related issues such as dynamic obstacles,
- Provide users with relevant content, and
- Interface with users in an engaging and enjoyable way.

It is important to note that not all modules will operate directly on the robot's hardware. A hardware abstraction layer will separate the software modules from the hardware, allowing the transfer of developed technology to other robots.

3 Software Frameworks

There are several frameworks available for robotics and augmented reality applications which will allow us to reduce the development time. In this section, we will present a brief description of software frameworks, which provide tools necessary for the FROG project. We include some of the best known robotic frameworks, namely, Player, ROS, YARP, CARMEN, OROCOS, Orca, MRDS, URBI, MRPT, MOOS, OpenJAUS, ERSP, CLARAty, and GenoM. Since a great part of the project is dedicated to the interaction with tourists and displaying information in an appealing way, we will describe two well known augmented reality frameworks, namely, Total Immersion D'Fusion and Metaio Unifeye, and the game engine Unity. We will also include the more general purpose framework YVision, which is targeted at areas such as simulation, computer vision, embodied agents, and embodied interaction.

3.1 Player

Player [13, 31] is a robot control interface that provides drivers for several sensors and robotic platforms and also some high-level algorithms.

The framework is based on a star topology and the server/client model: each instance of the Player server controls not only the connection establishment, but also the data transferred from (to) the sensors (actuators). Multiple robots would run multiple instances of the Player server. It supports multiple concurrent client connections to devices, and therefore, it is suitable for distributed and collaborative control.

The Player server can run on any POSIX platform (including ARM- and PPC- based Linux systems). It resorts to CMake to archive cross-compatibility and it can be built natively on Windows. A client program can run on any machine that shares a network connection with the robot and be written in any language that supports TCP sockets. Libraries in C, C++, Pyton and Ruby are officially supported, while Java, Ada and Octave are supported by third parties.

The Player project also includes two simulators, Stage and Gazebo. Stage is a multiple robot 2D simulator with Player interface, which includes several sensor models such as sonar, scanning laser range finder, pan-tilt-zoom camera, and odometry. The interface between Player and Stage allows to develop and refine different control strategies, which can afterwards be applied in real robots with few or even no changes. Gazebo is also a simulator compatible with Player and is suitable for 3D multiple robot simulation in outdoor and indoor environments. It provides realistic sensor feedback and physically plausible interactions between objects thanks to the Open Dynamics Engine. Stage is mainly targeted to simulate a large robot population in a 2D environment, while Gazebo is designed to simulate smaller robot populations but with higher fidelity and in a 3D environment.

The project is released under the GPL and LGPL. All code from the Player project is free to use, distribute and modify, but all derived work must be distributed under the same license. Further details about the different software licenses are given in Appendix 6.

3.2 ROS (Robot Operating System)

From the same developers of Player, ROS [14, 39] is an open-source robotic framework with functionalities that range from low-level PID control to high-level localization algorithms such as SLAM.

It has a distributed architecture based on independent nodes (executables) that communicate using messages transmitted over TCP/IP sockets. There is a Master node where the client nodes must be registered, but at runtime, the nodes are connected in a peer-to-peer topology. With this central node control, ROS provides for both client/server and publish/subscribe paradigms. The latter may be valuable at integration time.

ROS is developed and tested for Unix-based platforms, namely, Ubuntu and Mac OS X. Although the execution of the Master ROS node in Windows platforms is not yet fully possible, a ROS node can be written for any system that supports TCP/IP communication.

Both ROS and Player have a large community of users, which is an important asset as it provides valuable support and helps the development of new features. The number of ROS adopters grows steadily, and it might well be soon regarded as a de facto standard in the robotic environment. This framework inherits code from several other open-source projects. Many drivers, high-level algorithms, and the two simulators (Stage and Gazebo) from the Player project are available off-the-shelf. Procedures exist also to integrate those not included. Whereas Player offers more hardware drivers, ROS offers more implementations of algorithms. ROS is more powerful and flexible than Player, but, as usual, it means greater complexity. Among the supported hardware there is the Microsoft Kinect. Finally, this framework comes with several visualization, data logging and data replay tools, that are useful for debugging.

ROS is a free and open-source software distributed under a BSD license.

3.3 YARP (Yet Another Robot Platform)

YARP [20, 35, 29] is an open-source software library especially suitable for (but not limited to) humanoid robots.

This framework is modular and is composed by different processes that communicate via YARP ports. The YARP framework can work in two configurations: with a central node managing name resolution (central node control topology) or resorting to DNS/hard-coded name resolution (external/hard-coded control topology). In any of them, YARP allows application objects to provide services to each other (service oriented paradigm), and to establish direct connections to each other. Yarp can use several transports, namely those supported by the ACE library YARP uses. These include sockets, shared memory, pipes and corba RPC among others.

YARP itself provides some marshalling guidelines for compatibility, but does not enforce them. YARP is compatible with CMake in order to be cross-platform. Hence, processes can be executed on different OS, provided that the communication methods are available, which is usually the case.

The main programming language of YARP is C++, but it is compatible with many languages, such as Python, Java, Tcl, Lisp, Ruby, Pearl and others via SWIG bindings.

The existing drivers support sensors such as cameras, microphones, Microsoft Kinect and actuators such as motors. However, the available high-level algorithms are scarce. Although, there are YARP libraries that connect with Stage, the framework does not possess its own simulator. YARP is available under GPL and some of its components under LGPL.

3.4 CARMEN (Carnegie Mellon Robot Navigation Toolkit)

CARMEN [2, 36] is a robotic framework constituted by a collection of modules (processes) that provide a consistent interface and a basic set of primitives to control a variety of commercial robot platforms.

The CARMEN modules communicate using the Inter-Process Communication library [6], by R. Simmons and D. James, also developed at the Carnegie Mellon University. The IPC lib model involves an application independent central node (taking the functions of name and type server) and several application-specific processes. The IPC lib allows application objects to talk to each other using the publish/subscribe, service oriented and point-to-point communication paradigms. The IPC lib provides some additional features as well, such as data marshalling.

The modules are written in C but Java support is provided. Linux is the only supported OS.

The Carmen framework is organized in a three-layer architecture. The base layer governs hardware interaction and control, by providing an abstract set of base and sensor interfaces. It also provides low-level control loops for simple rotation and straight-line motion and integrates sensor and motion information to provide improved sensor odometry, and also allows for low-level collision detection (distinct from collision avoidance). The navigation layer implements intermediate navigation primitives including localization, dynamic object tracking, and motion planning. Unlike many other navigation systems,

motion control is not divided into high-level (strategic) planning and lower-level (tactical) collision avoidance; instead, all but the lowest-level motor control is integrated into a single module. This allows for more reliable motion planning, at a reasonable computational cost. Finally, the third layer is reserved for user-level tasks employing primitives from the second layer. The CARMEN navigation and mapping algorithms require the presence of a laser range finder.

The following mobile robot platforms are supported: iRobot ATRV, iRobot ATRVjr, iRobot B21R, ActivMedia Pioneer I, ActivMedia Pioneer II, Nomadic Technologies Scout, Nomadic Technologies XR4000 OrcBoard, and Segway. The sensors supported are: SICK LMS laser measurement system, SICK PLS proximity laser scanner, GPS receiver using the NMEA protocol, sonar (preliminary support), and Hokuyo IR Support (PB9).

CARMEN is an open software distributed under GPL restrictions.

3.5 OROCOS (Open Robot Control Software)

Targeted primarily at the field of robotic manipulators, OROCOS [12, 27, 40] is an open-source, modular framework for robot control.

The OROCOS project supports four libraries: the Kinematics and Dynamics Library (KDL), the Bayesian Filtering Library (BFL), the Real-Time Toolkit (RTT), and the Orocos Component Library (OCL). The KDL contains several basic functions such as kinematic chains, real-time inverse and forward kinematics, and Pyton bindings. Dynamic Bayesian Networks, Kalman Filters and Particle Filters are implemented in the BFL. OCL provides some ready to use control components. Finally, the RTT provides a C++ framework, targeting the implementation of (realtime and non-realtime) control systems.

Orocos was designed to be a good framework for in-process application components (what we termed the local communication domain). Orocos has support for very efficient inter-component communication in-process (lockfree buffers) and inbetween-process (message queues), but for multi-cpu/ multi-computer communication, external middleware is needed. Usual choices are CORBA and ROS transport.

No simulation environment is made available by the developers. On the other hand, advanced control algorithms for robotic arms are provided.

It is written in C++ and it has multi-platform support (using CMake) with Windows, Mac OS X and Linux. Extensions to other robotics frameworks, namely, ROS and YARP are also available.

Regarding licenses, the KDL and OCL software are licensed as LGPL software. Both the RTT and BFL software are licensed as GPL + runtime exception, which is exactly the same license as the GNU Standard C++ library.

3.6 Orca

Orca [11, 26, 33] is an open-source framework for developing component-based robotic systems. It provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks. Historically, Orca began as part of the OROCOS project funded by the EU, but eventually it became an individual project. Software reuse is stimulated by the definition of a set of commonly-used interfaces and by an attractive high-level API. It has a modular architecture based on Component-Based Software Engineering.

Orca relies upon ICE project as a middleware. ICE can be used in a centralized control, providing point-to-point, service oriented and pub/sub paradigms, or in external/hard-coded control topology, providing point-to-point and service oriented only. Marshalling and an interface definition language (SLICE) are available. Current transport are TCP/IP and UDP.

All components which are currently in the repository are written in C++. However, there are examples in Java, Python, and PHP. Slice interfaces can be compiled to C++, Java, Python, PHP, C#, Visual Basic, Ruby, and Objective C. Full Linux support is provided. Interfaces, core libraries and utilities, and some components compile in Windows XP and there are experimental builds in Mac OS X.

Regarding drivers, in the project repository, interfaces are available with GPS receiver, laser range finder, and cameras, and also some libraries useful for path planning applications. Orca can be used with Stage and Gazebo simulators.

The components are released under LGPL and GPL.

3.7 Microsoft Robotics Developer Studio (MRDS)

The software giant Microsoft has developed MRDS, which is a full Windows-based robotic framework with integration with Microsoft Visual Studio. Four main components are included in the package: the Concurrency and Coordination Runtime, the Decentralized Software Services, the Visual Programming Language, and the Visual Simulation Environment, which are separately available for use in commercial applications.

The contol topology of the framework can be centralized or external/hard-coded, providig service-oriented and pub/sub paradigms. Everything is abstracted by the Concurrency and Coordination Runtime, which makes it hard to find out the communication details.

Its main programming language is C++, but it is also compatible with Visual Basic and Iron Python. The more recent commercial version of MRDS is the 2008 R3 which is only compatible with Windows XP, Windows 7 and Windows Server. Recently, the version 4 Beta was released , but it is still only available for evaluation.

Very few useful robotic algorithms are included. On the other hand, MRDS can be used in a variety of configurations: the robots can be directly connected to the PC using RS-232, Bluetooth, USB, etc., can be used on a PC on board the robot, or can be used only with the simulator to test new algorithms and control techniques.

MRDS provides a 3D simulation environment, visual programming interface, and easy access to the robot sensors and actuators, including the Kinect system.

3.8 Urbi

Another software platform used to develop software for robotic and complex systems is Urbi [18, 23, 25]. This cross-platform software is written in C++ and it is based on the UObject distributed C++ component architecture, which is an event-driven script language. The UObject components can be called from urbiscript programming language, and appear as native objects that can be scripted to specify their interactions and data exchanges.

Relative to communications, Urbi exhibits a central node control architecture, providing application objects with both service oriented and publish/subscribe paradigms. It regularly uses TCP/IP as a transport. The later versions of Urbi 2.x integrate support for communicating with ROS topics and services. A bridge to YARP is provided as a module.

Regarding programming languages, Urbiscript provides parallelism and event-based programming, C++ like syntax, service oriented architecture, and client interfaces with Java and Matlab. Despite its benefits, it has the shortcoming that it still requires learning a relatively new language.

It is compatible with Linux, Mac OS X, Windows, and others, and can run on various processors: x86, ARM, mips, powerPC, etc.

The platform does not have its own simulation environment, but it is compatible with Webots, which is a commercially available (1900€Pro, 260€Edu) simulation software. Gostai, a French company, has several plug and play UObject components useful for robotics, e.g. voice recognition, voice synthesis, face detection (including a bridge with OpenCV), face recognition, SLAM, color blob detection, and SIFT based object recognition.

Urbi is open-source with GPL compatible license (GNU AGPL v3).

3.9 MRPT (The Mobile Robot Programming Toolkit)

The MRPT [9, 28] is composed by C++ libraries and a number of ready-to-use applications. It was developed by the research group MAPIR from the University of Málaga, with worldwide contributions. This framework aggregates efficient algorithms for SLAM, SIFT, detection and tracking of visual features, Bayesian inference, random number generators for a variety of probability distributions, Kalman filtering and particle filters for localization, among others.

MRPT is mainly intended for in-process application objects (local communication domain). For inter-process communication (remote communication domain), it provides some wrappers on sockets and serialization/deserialization facilities. Thus, the framework itself only provides for point-to-point remote connectivity, relying on an external/hard-coded control topology.

The C++ libraries are tested in 32bit and 64bit systems, and are compatible with Linux, Windows, and partially with Mac OS X. Good tutorials, a reference book, and a forum are available to provide support to the developers that use this framework.

The whole of the MRPT functionality is also made available as a ROS package, mrpt-ros-pkg. Drivers for some sensors such as cameras, laser range finder, GPSr, Microsoft Kinect, and rate gyros are build-in into the software.

MRPT is licensed under the GNU GPL version 3.

3.10 MOOS (Mission Oriented Operating Suite)

MOOS [8] which stands for Mission Oriented Operating Suite, is a set of C++ libraries and applications designed to facilitate research in the mobile robotic domain. The provided functionalities range over low-level, multi-platform communications, dynamic control, high precision navigation and path planning, mission logging and playback.

This framework exhibits a star topology, where several client applications connect to a central node. All communication happens via that central server. The network has no peer-to-peer communication, ensuring that the clients do not communicate directly with each other. The clients operate independently, in a publish/subscribe architecture entirely carried by the central node. The application objects can be distributed over any number of machines.

A Java version of MOOS client is available. Linux and Mac OS X are the only OS supported by MOOS. On the official webpage [8], a set of HTML and PDF documents can be found, which constitute the basic support for this software, since neither a forum, nor tutorials are available.

From the available information it was not possible to know the license of this framework.

3.11 OpenJAUS

OpenJAUS [10] is an open-source implementation in C++ of Joint Architecture for Unmanned Systems [1] (JAUS). JAUS is a standard chartered by de Department of Defense from the USA to define the communication and interoperation protocols of unmanned systems within a network. AUS is basically a common messaging protocol for robots.

JAUS employs a Service Oriented Architecture (SOA) approach to enable distributed command and control of these systems, which is said to allow efficient exchange of information between the CPUs of distributed systems. This framework can emulate the publish/subscribe paradigm also, by adding events to the service-oriented model. The framework relies only on an external control topology for name/address resolution, the rest being fully distributed. This is made possible by imposing a minimal set of discovery remote services to be implemented in every application object. It uses at least UDP and TCP/IP as a transport.

OpenJAUS runs under Linux and Windows. Regarding documentation, a forum can be found on its webpage [10] providing developer oriented support.

No information is provided about any available drivers for sensors or robotic platform. OpenJAUS does provide a library with all the standard JAUS messages, so to ease the integration of your robots within a JAUS network.

For commercial, government, or customer funded academic projects use, it is required to purchase one or more commercial developers' licenses. One for each person developing software with OpenJAUS. The commercial license of OpenJAUS 4.0 fee is around 2000€.

3.12 Evolution Robotics ERSP

The company Evolution Robotics has developed the robotic framework ERSP 3.1 [4]. This software package is targeted at R&D and Product Groups from companies worldwide. Some technologies included are: the visual pattern recognition software (ViPR™), visual localization and mapping system that enables a robot to use only one camera combined with wheel encoders (vSLAM®), an infrastructure, tools and facilities for handling and managing the robot hardware and software components (ERSA™).

ERSP is constrained to the local communication domain.

This framework includes a graphical composer, which can be used to create programs based of a number of reusable building blocks. The building blocks correspond to behaviors, which are activated/deactivated in run time by a higher-level program. However, no simulator is build-in into the package, and the range of supported hardware is small.

ERSP is available under the Windows 2000/XP Professional, and Linux: Debian Sarge, Fedora Core 3 and Red Hat 7.3.

3.13 CLARAty (Coupled-Layer Architecture for Robotic Autonomy)

CLARAty [3, 37, 38, 24] is a robotic platform built by NASA in collaboration with some universities and released as an open-source project. This software implements an architecture comprised of two layers, the Functional layer and the Decision layer. The Functional layer accounts for the hardware drivers, navigation algorithms, localization algorithms, and behaviors, while the Decision layer uses a declarative model-based approach to define activities. This layer incorporates a CASPER planner that, given the mission constraints, schedules all the activities required to perform the desired task.

CLARAty seems to have a central node control topology. Though communication between layers is based on sockets, it uses ACE as a middleware, so a wide variety of transports should be available.

The package includes a GUI and graphical display developed resorting to the Qt visual library.

This software is released under TSPA license, which renders its use in commercial applications impossible.

3.14 GenoM

GenoM (Generator of Modules) [5, 30, 34] is a tool to design real-time software architectures. It is mainly targeted at complex on-board systems, such as autonomous mobile robots or satellites. The GenoM's architecture is based on three layers, a Functional layer, an Execution Control layer and a Decisional layer. The Decisional layer planner and supervisor account for the planning of tasks and are reactive to external events. The GenoM's planner is based on an IxTeT temporal planner and the supervisor is based on a Procedural Reasoning System (OpenPRS). The Execution Control layer is composed by input and output buffers, by a model checker and by a system state database. The input buffer receives the requests from the Decisional layer. The output buffer sends requests to the Functional layer, and the system state database records the current and past state of the robot and its resources. Finally, the model checker performs the necessary actions according to the requests received from the Decisional layer. The Functional layer comprises the controllers and execution engines. The former manage the modules according to external requests, and the latter perform the activities required by the controllers.

GenoM allows encapsulating the operational functions on independent modules that manage their execution. A module is a standardized software entity that is able to offer services which are provided by a set of algorithms. Modules can start or stop the execution of these services, pass arguments to the algorithms and export the data produced. The framework is thus fully service-oriented, and its control topology has no central node, but relies on external/hard-coded name discovery and resolution. As a transport, it uses TCP/IP and shared memory.

GenoM modules run under various operating systems (Linux, BSD, Xenomai). The modules are automatically produced by GenoM using a common generic model of a module and a synthetic description of the considered module. This module description is elaborated using a very simple language that allows programmers to declare and describe its components: services, parameters, qualitative results, exported data, temporal and logical characteristics, etc. Modules provide two standard interface libraries in various programming languages (C, tcl, XML, open-prs).

Besides ease of development (the module description does not depend on the operating system and does not require specific knowledge on distributed systems), the generation of the modules guarantees that they fit with the generic common model. It is an important feature to handle large systems. The underlying standardization of the structure, the behavior and the interface of the modules allows the automation of their integration.

GenoM is compatible with the LAAS morse simulator and is distributed under a BSD license.

3.15 Total Immersion D'Fusion

Total Immersion D'Fusion [15] is an augmented reality framework that offers tools for tracking, image rendering, etc. D'Fusion tracking does not require markers (bar codes, or specific black and white shapes) and allows for robust tracking of 2D and 3D objects. Existing documents and products can be adapted for augmented reality applications, eliminating the need to create them from scratch. D'Fusion is able to manage and animate very complex 3D objects through a simple language allowing for development and maintenance. D'Fusion can also encrypt proprietary customer assets to guarantee security.

It is compatible with Windows PCs, and behaviors and interactions can be controlled using the scripting language Lua.

D'Fusion is only available as a commercial tool.

3.16 Metaio Unifeye

The augmented reality framework Metaio Unifeye [7] provides interesting tools for tracking, capturing and rendering. These include 2D image tracking, 3D object tracking, face tracking, sensor and offset calibration tools. This augmented reality framework is developed by the company Metaio and is target at professional applications for business and entertainment. The software allows the combination of real-world footages and images and 3D CGs. The tools provided are useful not only for visualization, but also for analysis of virtual- and real-world space, sales promotion, entertainment, industrial applications as well as consumer applications.

3.17 Unity 3D

The Unity [17], on the other hand, provides libraries for creating 3D video games or other interactive content such as architectural visualizations or real-time 3D animations. Unity's development environment runs on Microsoft Windows and Mac OS X, and the games it produces can be run on Windows, Mac, Xbox 360, PlayStation 3, Wii, iPad, iPhone as well as Android. It can also produce browser games that use the Unity web player plug-in. Unity consists of both an editor for designing content and a game engine for executing the final product. Unity is similar to Director, Blender game engine, Virtools, Torque Game Builder, and Gamestudio, which also use an integrated graphical environment as the primary method of development.

3.18 YVision

Nowadays, there is great demand for multimedia solutions focused on the user experience. New technologies such as parallel processing, stereo cameras, touch-screens, and smart phones have been developed to meet market needs. However, managing the increasing hardware complexity is a challenging task for programmers and developers. Good design and programming methodologies are key requirements to develop attractive and cost-effective solutions.

YVision [22, 21] is a powerful software composition framework, especially targeted at the development of multimedia and interactive applications. The whole package ensembles several state-of-the-art technologies such as 3D physics engine, computer vision and image processing algorithms, augmented reality, and automatic parallelization of independent tasks. The modular nature of YVision offers automatic thread handling, through Task Parallel Library integration. One does not have to worry about concurrency or coordination in multi-core systems as all this complexity is automatically managed. To tackle the challenges posed by the new multimedia interactive applications, the core of YVision is based on Behavior Trees [16], a programming paradigm successfully used to manage the artificial intelligence engine of video games. The current release includes several kinds of video input devices (webcams, ethernet cameras, capture boards), Human Interface Devices (such as mice, keyboards, joysticks, Arduinos or any other device that complies to the HID specification) and the Wiimote.

Windows Communication Foundation [19] is used in YVision as the medium for communication. WCF is designed using service oriented architecture principles to support distributed computing where services have remote consumers. Clients can consume multiple services; services can be consumed by multiple clients. Services are loosely coupled to each other. A WCF client connects to a WCF service via an Endpoint. Each service exposes its contract via one or more endpoints. An endpoint has an address (which is a URL specifying where the endpoint can be accessed) and binding properties that specify how the data will be transferred. Binding specifies what communication protocols are used to access the service, whether security mechanisms are to be used, and the like. WCF includes predefined bindings for most common communication protocols such as SOAP over HTTP, SOAP over TCP, SOAP over UDP, and SOAP over Message Queues.

Objects and behaviors are the basic elements which compose an application. The objects are self-containing and autonomous identities with different properties and characteristics. The interactions between different objects are encoded in the behaviors. A common interface promotes the behavior modularity and code reuse. Behaviors are composed hierarchically in a tree, where the leaves are reusable actions or conditions. Hierarchical logic allows the management of the complexity of the objects behaviors.

The YVision capabilities in parallel data-fusion and integration of different input devices as well as combining the different vectors that constitute a robot have proven to be befitting in the Santander

Headquarters' Robot visitor guide project, from YDreams. Hence, YVision is a strong alternative to the other robotic frameworks with clear advantages in modeling the robot behavior.

The framework provides an API based on XML and .NET languages. Thus, it is compatible with several platforms such as Windows, Mac OS X, iPhone, Android and WP7.

The libraries are not open-source but are free to use.

4 Comparison Criteria and Evaluation

Previously, we presented some of the most used robotic software frameworks. In this section, we will identify and describe the framework features that are important for the FROG project and compare the frameworks with respect to those criteria.

The following features are important:

- OS compatibility - The OS running in the robot can either facilitate or hinder the software development and the integration with the remaining hardware. The portability to mobile OS, such as Android and iOS, is also an important factor.
- Programming language - The programming language should be such that it simplifies the interoperability of the robot software. By adopting a modern language that efficiently manages resources such as memory, we reduce the development time and increase the software performance.
- Communication domain - The communication domain covered by a given framework will determine whether application objects will be able to seamlessly exchange data (in the same or different machines) out of the box, or will have to resort to external communication facilities. The former is quite desirable, in order not to have to invest development effort in this essential but basic functionality. On the other hand, integration of multiple frameworks might be easier if all of them are devoid of remote communication capabilities, so that a common communication system could be developed for all of them.
- Connection topology - The connection topology will let us know how data is exchanged between application objects. Having a star topology (all the data traversing a central node) might be good for one-to-many transmissions of small amounts of data, while having peer-to-peer connectivity will allow for higher bandwidth without bottlenecks.
- Control topology - The way the connection topology is controlled in the framework is important in order to quantify the flexibility of that topology. Frameworks with central nodes tend to greatly simplify the access to their services, usually managing it automatically, but a dedicated application object with a special role (the central node) is required. Frameworks free of the formal central node requirement will ultimately rely either on external servers for some purposes (DNS for name resolution, for example) or on static, preexisting configurations (such as hard-coded addresses).
- Communication paradigms provided - The different paradigms provided by the framework for the application objects to use will be useful in different scenarios. The service-oriented paradigm is perfect for transaction-like communications, while the publish/subscribe one tightly suits sensor data broadcasting. It could be said in short that the more communication paradigms available within one frame, the better.
- Underlying transport. Ultimately, there must be a transport allowing data exchange between application processes. Each transport and protocol has different properties regarding robustness, overhead, etc., which need to be considered in the design phase. Whether the framework uses sockets, shared memory, files, signals, message queues, pipes, message passing, memory-mapped file, or some combination, will be taken into account.
- Built-in simulator - Since the FROG project includes the development of a 3D simulator, this criterion is not crucial. Nevertheless, the existence of such simulator is always an advantage and it can be useful in the initial phase of FROG while the project simulator has not yet been developed.
- Distribution license - Software can be released under different licenses. Some licenses, even being open-source licenses, can pose limitations on the commercialization of derived products. For further details, see Appendix 6.

- Last update - This criterion indicates whether the framework development is still active. Choosing an abandoned framework can be highly disadvantageous, as it may rapidly become outdated and limit further updates to the robot.
- Support quality - This criterion is indicative of the assistance that one could obtain from the developers and from the rest of the community to solve any questions that might arise.
- Existence of a discrete event system - A compact and flexible discrete event system is crucial for the project success, because it will allow us to model and to encode in a perceptible way the complex behavior of a robot such as the FROG.
- Available drivers - The existence of device drivers facilitate the configuration and simplify the change of sensors and actuators. To evaluate this criterion we consider the existence of drivers for laser range finders (Laser), inertial measurement unities (IMU), and cameras (Cam.).
- Available algorithms - The existence of already developed algorithms will simplify the development phase of the FROG robot. It will allow component testing from the early beginning of the project and the comparison between the already existing algorithms and the new ones developed for the FROG project. We divide this criterion into localization algorithms (Loc.), local path planners (L.P.), global path planners (G.P.), and motion control (Motion).
- Image rendering and Audio - The robot guide developed for FROG has to display exciting information about the points of interest of the historical sites. To that end, the robot needs to have image rendering (Rend.) and audio capabilities (Audio) in order to play sound and display information.
- Augmented Reality - One of the most interesting characteristics of FROG is the symbiosis between the mobile robot world and the augmented reality world. Interactive augmented reality overlay capabilities experience and increase knowledge transfer as information is offered through multisensory interaction
- Physics Engine - The existence of a physics real-time engine is useful for remote visualization, as it will allow us to propagate the robot position and attitude when there is delayed communications. Moreover, it might be useful to develop a simulation environment, and to endow the augmented reality objects and the natural user interface with physical characteristics, such as velocity, inertial, drag, etc..
- Statistical Tools - The robot will receive a large amount of data that are required for navigation, people detection, and emotion detection. This data needs to be processed in order to extract useful information. For instance, for people detection, it is required to employ a statistical pattern matching technique for time-varying data.
- Marker Tracking - Detection and tracking of markers can be used to increase the precision required for the augmented reality component.
- Face Tracking and Full Body Tracking - This is a key requirement to enable the development of algorithms for identification of human communicative cues including facial expressions, gaze, head nods and shakes, body gestures and postures.

Tables 1 to 5[1] summarize the characteristics of each framework. In Table 1 the frameworks are compared with respect to OS compatibility and programming language. In Table 2 the frameworks are compared with respect to communication domain, connection topology, control topology, communication paradigms provided and underlying transport. In Table 3 the comparison is done with respect to simulation capabilities, license, date of the last update, support quality and existence of a discrete event system. The characteristics of the frameworks with respect to control algorithms and drivers are presented in Table 4. Finally, Table 5 summarizes some of the framework features, namely, image rendering, sound player, augmented reality, physics engine, statistical tools, and tracking of markers, faces and full body.

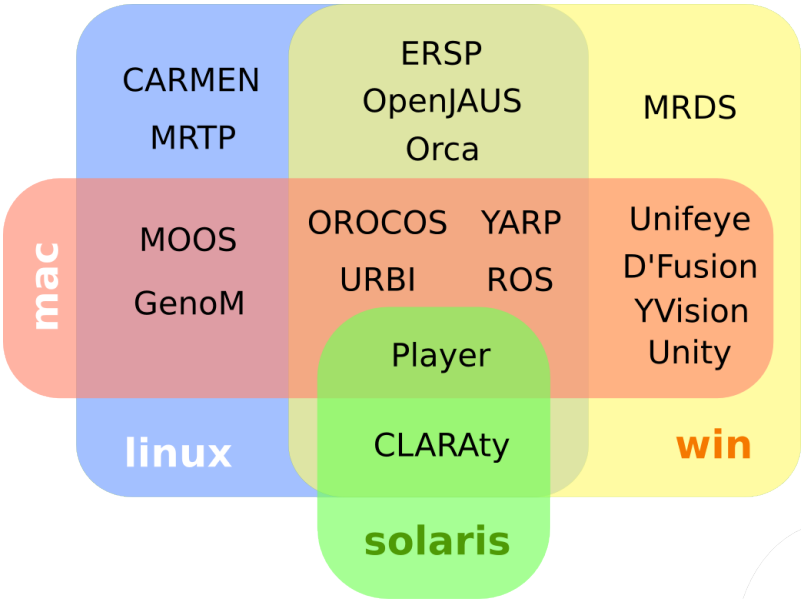


Figure 2: Venn diagram for the operative systems the frameworks can interact with. A framework in the "Linux" category implies compatibility with any of Ubuntu, Arch, Fedora, Gentoo, Suse, Slackware, Debian, Xenomai. A framework in the "Win" category implies compatibility with any of WindowsXP, Win7, WinCE. A framework in the "Mac" category implies compatibility with any of Mac OS X, BSD. A framework in the "Solaris" category implies compatibility with any Solaris version.

5 Discussion

In this section, we will discuss the advantages and disadvantages of each software framework. The FROG project goes beyond the classical mobile robot design. By developing a robot able to share the same space as tourists, capable of perceiving human emotions, via verbal and non verbal behaviors, and endowed with an engaging personality and social behavior, we will significantly expand the frontier of human-robot interaction and collaboration.

FROG is a multidisciplinary project, which will embrace a wide range of areas of knowledge. To accomplish the project goals on its different vectors, specific tools are required. Those are provided by different frameworks. Thus, in this study we compare software frameworks targeted at different areas, robotics, augmented reality, and game industry.

The characteristics and available tools of each framework are presented in the form of tables (Tables 1 to 5) for a clear comparison[2]. Table 1 compares the frameworks with respect to OS compatibility and programming language. If it is required that in the same system more that one software library coexist, it is important to guarantee that they are all compatible with a common OS. In addition, it would greatly benefit further extensions and reuse of the developed software, if the software framework were compatible with a wide variety of OS. The augmented reality frameworks, as well as, the Unity and the YVision frameworks are the ones that are supported in more OS. In particular, they are supported in mobile platforms, which are nowadays experiencing an impressive expansion. With respect to devolving language, most of the frameworks are compatible with C++, and some also with Python. The language C# can be used to develop applications using Orca, MRDS, Unity, and YVision. This language offers advantages such as being compiled to an intermediate language (CIL) independently of the target architecture and OS, automatic garbage collection, pointers no longer needed (but optional), definition of classes and functions can be done in any order, and applications can be executed within a restricted sandbox. MRDS and Unifeye provide visual programming tools to simplify the development process.

Table 2 compares the frameworks with respect to communication domain, connection topology, control topology, communication paradigms provided and underlying transport. For many robotic projects the topology and communication protocols are a key design parameter. For the FROG project it does not have the same importance, as it is a centralized system. However, it could be important to a future

[1] All tables can be found at the end of the document
[2] All tables can be found at the end of the document

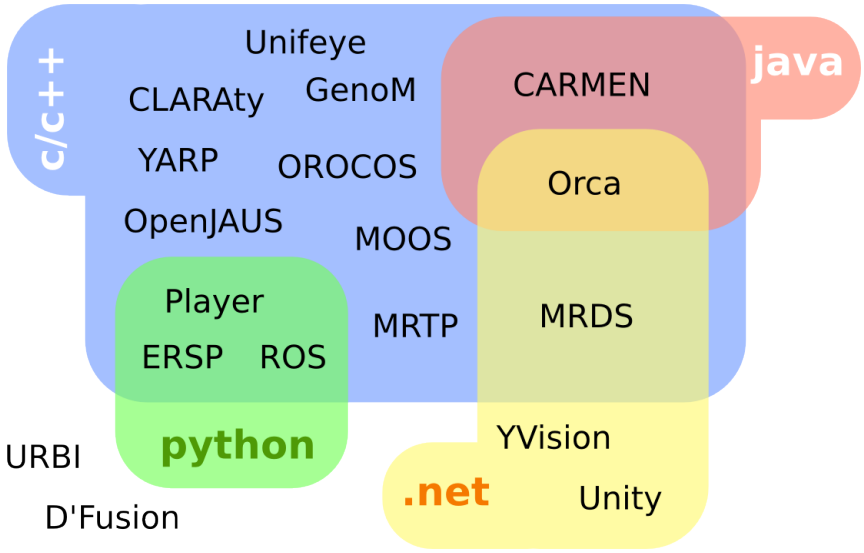


Figure 3: Venn diagram for the languages in which the frameworks can be accessed. Only four mainstream languages are shown, namely c/c++, java, python and any of the .Net family. This does not imply the frameworks APIs being limited to the languages shown, some being also available in TCL, XML, etc. Those frameworks with APIs solely available in other languages (such as Lua) are shown outside every bubble.

expansion of the project, such as, having several interconnected robots in the same site.

In Table 3 several other characteristics of the frameworks are displayed, namely, compatibility with simulator, release license, date of the last update, quality of the support and implementation of a discrete event system. Player, ROS, Carmen, URBI and GenoM are compatible with simulators. Unity and YVision provide a physics engine that easily allows the conception of a simulation environment. Most frameworks have release licenses incompatible with the development of commercial applications. Only ROS and GenoM are released under the BSD license, which is a permissive open-source software license compatible with its use in commercial products. The YVision framework is free to use and the YDreams developers have an extensive experience in using it. From the "Last-Update" column we can conclude that CARMEN, CLARAty, and MOOS are in an idle state, and in addition, they provide poor support, which are relevant shortcomings of these frameworks. The value of a discrete event system to complex applications, such as FROG, is evident. The frameworks ROS, CARMEN, MRDS, and CLARAty allow the user to construct finite state machines, while URBI, ERSP, and GenoM, provide an event-based programming language, tools for creation of behavior networks, and an event-driven supervisor, respectively. YVision provides a different discrete event system, which is based on behavior trees. This approach follows the latest trend in artificial intelligence for computer games, which is in the process of being adopted by the robotics community. This will result in a system architecture that can generate the appropriate behavior at any time, including in situations where information from input analysis is partial or missing completely, which might be a common case in the outdoor environment in which the FROG robot will operate.

Table 4 compares the studied frameworks with respect to the number of available control algorithms and device drivers. We conclude that YARP, OROCOS, URBI, MOOS, OpenJAUS, and naturally, D'Fusion, Unifeye, and Unity do not provided any implementation of algorithms for localization, path planning, and motion control. With respect to drivers, only Player, ROS, MRPT, and GenoM provide drivers for laser range finders, inertial measurement unities and cameras. Orca, MRDS, and YVision only offer drivers for some of these devices, and the remaining frameworks do not provide drivers for any of them.

Finally, Table 5 provides terms of comparison related to multi-media and augmented reality applications. Clearly, the robotic frameworks are not targeted at these applications which, on the other hand, are the core of D'Fusion and Unifeye. These facts are evident in the table. Despite that, Player and ROS provide some tools for augmented reality using external libraries. The game engine Unity provides rendering, audio support, as well as, a physics engine. The YVision is the only framework which has all the features present in this table. The great advantage of YVision comes from the conjugation and

articulation of several key features and tools for the FROG project.

The architecture of FROG can be conceptually divided into three different layers, the low-level control algorithms layer (comprising motor control, safety-stop, low-lever sensors abstraction layer), the middle-level control algorithms (comprising the navigation and localization systems), and the high-level behavior and interactive layer responsible for emotion detection and human-robot interaction via augmented reality tools.

After benchmarking several robotic frameworks, augmented reality frameworks, a game engine and the multi-purpose framework YVision, we propose two different strategies to be adopted in FROG.

In the first, the ROS framework is adopted as the base for the development of the low-level control layer and the middle-level control layer. Two other frameworks are suitable to be adopted for this purpose, Player and GenoM. The differences in the features of these three alternatives are not substantial. All three have good support of control algorithms and drivers, as well as good user support. The ROS project is being adopted at quite a rate, and it expansion is not foreseen to decline in the coming years. The implementation of the low-level control algorithms and the middle-level control algorithms using a framework as ROS facilitate their dissemination and benchmarking among the scientific community.

On the other hand, ROS is not so suitable for multimedia application. Hence, the high-level behavior and interactive layer have to be developed using other framework. Comparing D'Fusion, Unifeye, Unity, and YVision, the latter is the one that better answers the project needs. This alternative requires the development of a communication module that enables the exchange of information between ROS and YVision. In this approach ROS can also double as the middleware communication opening the opportunity to include several other frameworks that have bindings or integrate to ROS.

The idea behind the second alternative is to profit from the experience of YDreams and IDMind in developing the robotic guest assistants for the Santander headquarters in Madrid, Spain, which are based solely on YVision, and to pursue a similar strategy. The YVision capabilities will allow us to address all areas of the robot, from the locomotion and navigation algorithms to the complex interaction with visitors. Using the YVision as the base framework of all the FROG systems has several advantages. It will avoid possible communications problems between ROS and YVision, and promote a better integration and control over all the robot's systems. Using the 3D physics engine we can develop an advanced remote visualization system and a customized simulation environment especially developed to answer the project needs. The behavior trees will allow us to manage the complexity of the robot and tourists behaviors. YVision is developed by YDreams, so it is supported directly by the developers. Finally, by adopting this alternative the portability between different OS will not be an issue.

6 Conclusions

In this survey, we have evaluated fourteen different frameworks with respect to a set for criteria of recognized interest for the FROG project. The main objectives of the FROG project were briefly described and important features of the frameworks are identified. The studied frameworks were briefly described and their relevant features compiled in tables. The software composition framework YVision is also introduced and its main features are highlighted. Finally, we propose and discuss the advantages of two alternative solutions to be adopted in the FROG project, one based on a two frameworks solution, ROS and YVision, and other based solely on YVision.

Appendix

Terminology

This appendix describes the terminology adopted, especially the one employed when describing the communication capabilities. Please note that while meaningful and hopefully sensible terms have been chosen, the classification presented does not intend to be a complete or general taxonomy. It does, nevertheless, serve the framework discrimination purposes of the present document.

- Communication domain

  Local The communication methods used allow for comunication only between threads (shared address space) or processes, but always in the same machine.

Remote The communication methods used allow for comunication between threads and or processes, running in the same or in a remote machine.

- Connection topology

  Star All the data transfer traverses a central node

  Peer-to-peer The data is transfered directly between peer nodes. Notice this is true even for those frameworks where a central node is needed to establish/close the connections.

- How the services offered by the framewok are controlled

  Central Node Some services provided by the framework rely on the existence of an specially dedicated node.

  External/Hard-Coded The framework directly relies on the existence of external servers (DNS, for example) or resolution techniques (such as local hosts file, or address hard-coding) to provide its services.

- Communication paradigms provided by a framework

  None (point-to-point) The framework provides for the mechanisms to establish a direct link between two application objects. In general, before the communication establishment can be initiated, both application objects must be up and running (time coupled). Once the direct link has been established, every communication detail is up to them.

  Service oriented The framework provides for the mechanisms in order to establish and regulate a service oriented communication *between two application objects*, a la remote procedure call. This paradigm allows one application object to provide well defined services to other application objects. Service provider and service client may be tightly time-coupled (synchronous operation), or mildly coupled (asynchronous operation, message passing).

  Publish/Subscribe The framework provides for the mechanisms in order to allow the framework clients to publish in (or subscribe to) messages feeds. This paradigm allows one application object to act as a message producer (or consumer), independently of whether other application objects are consuming (or producing) the data. Producer and consumer are loosely coupled.

Software distribution licenses

Each license type gives the user and the developer different rights and restrictions. They can be targeted at open-source software or at commercial software. Some of the most used open-source software licenses are, the Apache License, Berkeley Software Distribution (BSD) License, the MIT License, GNU General Public License (GPL), GNU Lesser General Public License (LGPL), and the GNU Affero General Public License (AGPL). All open-source software licenses, allow the use for any purpose, the distribution and the modification of the software, under specific restrictions. The Apache license, the BSD license, and the MIT license are part of a class of open-source software licenses with minimal requirements about how the software can be redistributed, not requiring modified versions of the software to be distributed using the same license. Apache license is the longest of the three but it is more explicit in its terms. The GPL restricts all the derived works to be distributed under the same license terms. The LGPL apply the same restrictions as GPL on the programs governed under it, but it does not apply these restrictions to other software that merely link with the program. The AGPL is a slightly modified version of GPL and ensures the access to the source code of applications running on network servers. Developers can also designate its software as Technology and Software Publicly Available (TSPA). This license allows its distribution but explicitly forbids its use in any commercial application. For commercial products there is the End-User License Agreement (EULA), which is a legal contract between the user and author or publisher of the application. The EULA may differ from product to product and establishes the legal limitation of use and distribution of the software. For further details on the different software licenses, the reader is referred to [32].

Bibliography

[1] AS5684 JAUS service interface definition language. http://standards.sae.org/as5684a.

[2] CARMEN official website. http://carmen.sourceforge.net/.

[3] CLARAty official website. http://claraty.jpl.nasa.gov/man/overview/index.php.

[4] ERSP official website. http://www.evolution.com/products/ersp/.

[5] GenoM official website. http://www.openrobots.org/wiki/genom.

[6] *Inter-Process Communication: A Reference Manual*.

[7] Metaio Unifeye official website. http://www.metaio.com/software/sdk/.

[8] MOOS official website. http://www.robots.ox.ac.uk/ mobile/MOOS/wiki/.

[9] MRPT official website. http://www.mrpt.org/.

[10] OpenJAUS official website. http://openjaus.com/.

[11] Orca official website. http://orca-robotics.sourceforge.net/.

[12] OROCOS official website. http://www.orocos.org/.

[13] Player Project official website. http://playerstage.sourceforge.net/.

[14] ROS official website. http://www.ros.org/.

[15] Total Immertion D'Fusion official website. http://www.t-immersion.com/.

[16] Understanding behavior trees. http://aigamedev.com/open/article/bt-overview/.

[17] Unity official website. http://unity3d.com/.

[18] Urbi official website. http://www.urbiforge.org/.

[19] Windows communication foundation official website. http://msdn.microsoft.com/en-us/netframework/aa663324

[20] YARP official website. http://eris.liralab.it/yarp/.

[21] YVision official website. www.yvision.com/.

[22] A. Almada, G. Lopes, A. Almeida, J. Frazao, and N. Cardoso. Yvision: A general purpose software composition framework. In *Human-Computer Interaction New Trends 13th International Conference HCI*, pages 779–788, 2009.

[23] Jean-Christophe Baillie. Urbi: towards a universal robotic low-level programming language. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, Aug. 2005.

[24] Jean-Christophe Baillie. A reusable software framework for rover motion control. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Los Angeles, CA, USA, Feb. 2008.

[25] Jean-Christophe Baillie, Akim Demaille, Quentin Hocquet, Matthieu Nottale, and Samuel Tardieu. The urbi universal platform for robotics. In *Workshop Proceedings of SIMPAR 2008 Intl. Conf. on Simulation Modelling and Programming for Autonomous Robots*, Venice, Italy, Nov. 2008.

[26] A. Brooks, T. Kaupp, A. Makarenko, A. Orebäck, and S. Williams. Towards component-based robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*, 2005.

[27] Herman Bruyninckx. Open robot control software: the orocos project. In *IEEE International Conference on Robotics and Automation*, Seoul, Korea, May 2001.

[28] Jose Claraco. *Development of Scientific Applications with the Mobile Robot Programming Toolkit*. Oct. 2010. (online: http://www.mrpt.org/downloads/mrpt-book.pdf).

[29] Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robot. Auton. Syst.*, 56(1):29–45, Jan. 2008.

[30] S. Fleury, M. Herrb, and R. Chatila. GenoM: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International Conference on Intelligent Robotics and Systems*, Grenoble, France, 1997.

[31] Brian P. Gerkey, Richard T. Vayghan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, Jun. 2003.

[32] Andrew Laurent. *Understanding Open Source and Free Software Licensing*. O'Reilly Media, Aug 2004.

[33] A. Makarenko, A. Brooks, and T. Kaupp. Orca: Components for robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, 2006.

[34] A. Mallet, S. Fleury, and H. Bruyninckx. A specification of generic robotics software components: future evolutions of GenoM in the Orocos context. In *International Conference on Intelligent Robotics and Systems*, Lausanne, Switzerland, 2002.

[35] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. YARP: Yet Another Robot Platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48, 2006.

[36] M. Montemerlo, N. Roy, and S. and Thrun. Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (CARMEN) toolkit. In *Intelligent Robots and Systems, 2003. (IROS 2003)*, 2003.

[37] I.A. Nesnas. *Software Engineering for Experimental Robotics*, chapter The CLARAty Project: Coping with Hardware and Software Heterogeneity. Springer Tracts on Advanced Robotics. Springer, 2006.

[38] I.A. Nesnas. CLARAty: A collaborative software for advancing robotic technologies. Adelphi, MD, USA, Jun. 2007. NASA Science and Technology Conference.

[39] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[40] Diego Santini and Walter Lages. An open control system for manipulator robots. In *20th International Congress of Mechanical Engineering*, Seoul, Korea, Nov. 2009.

Comparison Tables

Table 1: Comparison of robotic frameworks with respect to OS compatibility and programming language.

Software	OS Compatibility	Programming Language
Player	Win, Mac OS X, Solaris, Linux	C, C++, Python and Ruby (client nodes in any lang supporting TCP sockets)
ROS	Ubuntu (Experimental in Windows, OS X, Arch, Fedora, Gentoo, OpenSUSE, Slackware, Debian, Android)	C++, Phyton and Lisp (client nodes in any lang supporting TCP sockets)
YARP	Win, Mac OS X, Linux(based on Cmake in order to be OS portable)	C, C++
CARMEN	Linux	C, but provides Java support
OROCOS	Win, Mac OS X, Linux	C++
Orca	Linux, (interfaces, core libraries, and some components compile in Win XP, based on CMake/Ice combination)	Java, C#, C++
MRDS	Win 7, XP, (CE5.0 and CE 6.0 on ver. 2008 R2)	C#, visual
URBI	Win, Mac OS X, Linux	UObject distributed C++ (urbiscript)
MRPT	Win, Linux (limited functionalities on Mac OS X)	C++
MOOS	Mac OS X, Linux	C++
OpenJAUS	Win, Linux	C++
ERSP	Win, Linux	Python, visual
CLARAty	Mac OS X, Solaris, Linux	C++
GenoM	Linux, BSD, Xenomai	C, tcl, XML, open-prs
D'Fusion	Windows (PC and mobile), Mac OS X and Symbian	Lua
Unifeye	Windows (PC and mobile), Mac OS, iPhone and Symbian	C++, visual programming
Unity	Win, Mac OS X, Xbox 360, PlayStation 3, Wii, iPad, iPhone, Android	C#
YVision	Win, Mac OS X, iPhone, Android, WP7	XML and .NET languages

Table 2: Comparison of robotic frameworks with respect to communication domain, connection topology, control topology, communication paradigms provided and underlying transport.

Framework	Domain	Topology	Control		Paradigms			Transport
			Central Node	External/Hard	Publish/Subscribe	Service Oriented	Point-to-Point	
Orca	Remote	P2P	✓	✓	✓[1]	✓	✓	TCP/IP, UDP
YARP	Remote	P2P	✓	✓	-	✓	✓	TCP/IP, UDP, shared memory, pipes [2]
YVision	Remote	P2P	✓	-	✓	✓	✓	TCP/IP, UDP
ROS	Remote	P2P	✓	-	✓	✓	✓	TCP/IP
CARMEN	Remote	P2P	✓	-	✓	✓	✓	TCP/IP
URBI	Remote	P2P	✓	-	✓	✓	N.F[3]	TCP/IP
MRDS	Remote	P2P	✓	N.F	✓	✓	-	TCP/IP
MRPT	Remote	P2P	-	✓	-	-	✓	Sockets
OpenJAUS	Remote	P2P	-	✓	✓	✓	N.F	TCP/IP, UDP
GenoM	Remote	P2P	-	✓	-	✓	-	TCP/IP, shared memory
Player	Remote	Star	✓	-	-	-	✓	TCP/IP
MOOS	Remote	Star	✓	-	✓	-	-	TCP/IP
CLARAty	Remote	N.F	✓	N.F	N.F	N.F	N.F	TCP/IP, UDP, shared memory, pipes[2]
OROCOS	Local	-	-	✓	-	-	✓	Lockfree buffers, message queue
ERSP	Local	-	-	✓	-	-	✓	N.F
D'Fusion	n/a[4]	-	-	-	-	-	-	-
Unifeye	n/a	-	-	-	-	-	-	-
Unity	n/a	-	-	-	-	-	-	-

[1] Only available when using the central node control topology.
[2] Uses ACE library, which includes many more transports.
[3] Not found.
[4] Not applicable. These are monolithic applications, with some plugin support.

Table 3: Comparison of robotic frameworks with respect to simulation capabilities, license, date of the last update, support quality, discrete event system.

Software	Simulator	License	Last Update	Support	Disc. Event Syst.
Player	Yes (2D and 3D)	GPL, LGPL	25-Nov-2010	Forum and documentation - good supporting community	N.F.[1]
ROS	Yes (2D and 3D)	BSD	2011	Forum and documentation - good supporting community	Hierarchical state machines (SMACH)
YARP	No	GPL, some components LGPL	21-Sep-2011	Mailing list and documentation - good supporting community	N.F.
CARMEN	Yes (2D)	GPL	Oct-2008	Limited	State machines
OROCOS	No	GPL, LGPL	2011	Wikipages and forum	State machines
Orca	No	GPL, LGPL	10-Nov-2010	Good documentation, but no forum	N.F.
MRDS	Yes (3D)	Commercial	17-Sep-2011 (v. 4 beta) 20-May-2010 (v. 2008 R3)	Professional support	State machines
URBI	Can be used with Webots	AGPL v3	17-Mar-2011	Forum and documentation	Event based prog. lang.
MRPT	No	GPL v3	4-Jun-2011	Forum and documentation	N.F.
MOOS	No	(unknown)	27-Jul-2010	Some documentation, but no forum	N.F.
OpenJAUS	No	Commercial (license 2k€)	(unknown)	Forum (few users) and some documentation	N.F.
ERSP	No	Commercial	(unknown)	Professional support	Behavior networks
CLARAty	No	TSPA	2008	Very limited	State machine
GenoM	Compatible with Gdhe	BSD	26-Apr-2011	Online documentation but no forum and no mailing list	Event-driven supervisor
D'Fusion	No	Commercial	(unknown)	Professional support	(unknown)
Unifeye	No	Commercial	(unknown)	Professional support	(unknown)
Unity	3D physics engine	Commercial	26-Jul-2011	Professional support	Available as plug-ins (e.g. behavior trees)
YVision	3D physics engine	not open-source but free to use	Out-2011	Tutorials and a forum	Behavior trees

[1] N.F. - not found

Table 4: Comparison of robotic frameworks with respect to control algorithms and drivers.

	Control algorithms				Drivers		
	Localiz.[1]	Local P.P.[2]	Global P.P.[3]	Motion Control	Laser[4]	IMU	Camera
Player	✓	✓	✓	✓	✓	✓	✓
ROS	✓	✓	✓	✓	✓	✓	✓
YARP	N.F.[5]	N.F.	N.F.	N.F.	N.F.	N.F.	✓
CARMEN	✓	✓	✓	N.F.	N.F.	N.F.	N.F.
OROCOS	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
Orca	N.F.	✓	✓	N.F.	✓	N.F.	✓
MRDS	✓	N.F.	N.F.	N.F.	✓	N.F.	✓
URBI	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
MRTP	✓	✓	✓	✓	✓	✓	✓
MOOS	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
OpenJAUS	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
ERSP	✓	✓	✓	N.F.	N.F.	N.F.	N.F.
CLARAty	✓	✓	✓	✓	N.F.	N.F.	N.F.
GenoM	✓	✓	✓	✓	✓	✓	✓
D'Fusion	n/a[6]	n/a	n/a	n/a	n/a	n/a	n/a
Unifeye	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Unity	n/a	n/a	n/a	n/a	n/a	n/a	n/a
YVision	✓	✓	N.F.	✓	N.F.	✓	✓

[1] Localization
[2] Local path planning
[3] Global path planning
[4] Laser range finder
[5] N.F. - resource not found
[6] n/a - not applicable

Table 5: Comparison of robotic frameworks with respect to image rendering, sound player, augmented reality, physics engine, statistical tools, and tracking of markers, faces and full body.

	Rend.[1]	Audio	A.R.[2]	Phy. Eng.[3]	Stat.[4]	Marker T.[5]	Face T.[6]	Body T.[7]
Player	✓[8]	✓	✓[8]	N.F.[9]	N.F.	✓	N.F.	N.F.
ROS	✓	✓	✓[10]	✓	N.F.	✓	✓	✓
YARP	N.F.	✓	N.F.	N.F.	N.F.	✓	✓	✓
CARMEN	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
OROCOS	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
Orca	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
MRDS	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	✓
URBI	N.F.	✓	N.F.	N.F.	N.F.	✓	✓	✓
MRTP	✓	N.F	N.F.	N.F.	N.F.	✓	N.F.	N.F.
MOOS	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
OpenJAUS	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
ERSP	N.F.	N.F.	N.F.	N.F.	N.F.	✓	N.F.	N.F.
CLARAty	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.	N.F.
GenoM	N.F.	N.F.	N.F.	N.F.	N.F.	✓	N.F.	N.F.
D'Fusion	✓	✓	✓	N.F.	N.F.	✓	✓	N.F.
Unifeye	✓	✓	✓	N.F.	N.F.	✓	✓	N.F.
Unity	✓	✓	N.F.	✓	N.F.	N.F.	N.F.	N.F.
YVision	✓	✓	✓	✓	✓	✓	✓	✓

[1] Image rendering and display
[2] Augmented reality
[3] Physics Engine
[4] Tools for Statistical analysis
[5] Marker tracking
[6] Face tracking
[7] Full body tracking
[8] Using ARDev library
[9] N.F. - resource not found
[10] Using ARToolkit