FROG

FUN ROBOTIC
OUTDOOR GUIDE

# Deliverable D5.2
# Simulation environment for FROG AR


**Consortium**

UNIVERSITEIT VAN AMSTERDAM (UvA)
IDMIND - ENGENHARIA DE SISTEMAS LDA (IDM)
UNIVERSIDAD PABLO DE OLAVIDE (UPO)
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE (ICL)
UNIVERSITY OF TWENTE (UT)



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






## DOCUMENT INFORMATION

| **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: | D5.2 |
| Deliverable title: | Simulation environment for FROG AR |
| Due date of deliverable: | M35 - Aug. 31, 2014 |
| Actual submission date: | September 2, 2014 |
| Editors: | |
| Authors: | UPO, UT |
| Reviewers: | All Partners |
| Participating beneficiaries: | UPO, UT |
| Work Package no.: | 5 |
| Work Package title: | Integration, Evaluation and Demostrator |
| Work Package leader: | IDM |
| Work Package participants: | All Partners |
| Estimated person-months for deliverable: | 4 |
| Dissemination level: | Public |
| Nature: | Prototype |
| Version: | 1.0 |
| Draft/Final | Final |
| No of pages (including cover): | 19 |
| Keywords: | Simulation, Integration, Evaluation |

# Contents

# List of Figures

# List of Tables

# 1 Introduction

The objective of task 5.2 is to develop a software simulation of the entire system in order to prototype and test the FROG robot. In the initial Description of Work, YVision was to be used as framework for the simulation, integrating a physics simulation and 3D rendering. At the point in time when YDreams left the project, only a simulation tool for scripting sequences for the FROG Eyes had been provided, and no YVision-based physical simulation was available.

Several tools have therefore been developed by UPO to overcome the situation. The main goal was to provide an easy way to integrate and test the State Machine and Augmented Reality components developed by UT before performing real experiments.

This document summarizes such tools, and it is organized as follows. Section 2 details the simulator that allows an external module to send navigation commands to the robot and receive responses the same way as in the real FROG robot. Section 3 describes the virtual machine provided, which includes the simulator and real data that can be also played back for testing. Then, Section 4 is dedicated to Sketching tool used for robot's eyes sequences and the document will end with conclusions in Section 5.

# 2 Simulator

The first tool required was a simulator of the robot itself. Instead of the initially planned realistic 3D physical simulation, the focus has been on providing a tool that eased the integration activities and allowed us to test the basic behaviors of the robot.

Thus, the simulator employed to emulate the FROG robot and its environment is the Stage Robot Simulator [5, 3]. Stage simulates a mobile robot or a population of mobile robots moving in a sensing and two-dimensional bitmapped environment. Various sensor models are provided, including sonar, scanning laser rangefinder, fiducials sensor, pan-tilt-zoom camera with color blob detection and odometry. Stage offered a compromise between high-fidelity simulations and computational demand. Actually, as it will be commented below, one of the objectives was the possibility of distributing the simulator to other partners by means of virtualized machines, which in general lack the ability to run realistic 3D simulations. In Figure 1, a capture of the Stage screen with a robot and some simulated persons is shown.

Thus, we use Stage inside ROS[1] [4] as a tool to check the localization and navigation system; to test the integration of these systems with the AR system, receiving the status and location of the robot; and checking the behavior control module, sending navigation commands in the same way that the actual robot does it.

## 2.1 ROS wrapper for Stage Simulator

The Stage Robot Simulator can be used in ROS through a wrapper. This way, the simulator can be used together with other ROS packages. As the navigation code has been developed under ROS, this lets us to perform simulation experiments with (nearly) the same software that is used in the real robot.

[1] ROS is a set of free libraries and tools for roboticists http://www.ros.org/


Figure 1: Example of a simulation in the Stage Robot Simulation.

The ROS node called 'stageros' wraps the Stage simulator, via the C++ library 'libstage'. This node exposes only a subset of Stage's functionality via ROS. Specifically, it maps the Stage models of type laser, and of type position to the following ROS topics:

- Odometry. The wrapper publishes in ROS the odometry data obtained by the Stage simulator.
- Laser range finder. It is also prepared to publish readings from one laser range finder per robot.
- Velocity commands. The wrapper is ready to receive velocity commands through the topic called 'cmd_vel' and transform them to the corresponding Stage movement commands.

In Figure 2, a brief diagram of the inputs and outputs of the Stage Simulator with the ROS wrapper is shown. First at all, we have to provide the definition of the working environment and the robot and other agents' features. The appendix explains how to establish the initial configuration of the work space and the robot features. Then, the system is ready to receive ROS velocity commands which the wrapper parses to Stage commands. Moreover, the system will continuously be publishing the corresponding odometry data and sensors data of the robot in the environment. But for the FROG project purposes, the functionality of the ROS wrapper is not enough, so some modifications and improvements have been implemented.

### 2.1.1 Modifications of the ROS wrapper

The first problem is that the ROS wrapper does not allow a robot to have more than one laser range finder onboard. In our case, the FROG robot has three laser sensors, so it was necessary to re-program this part in the ROS wrapper to allow more laser sensors in the same robot. The subscription method of the laser models was redefined and the way of publishing

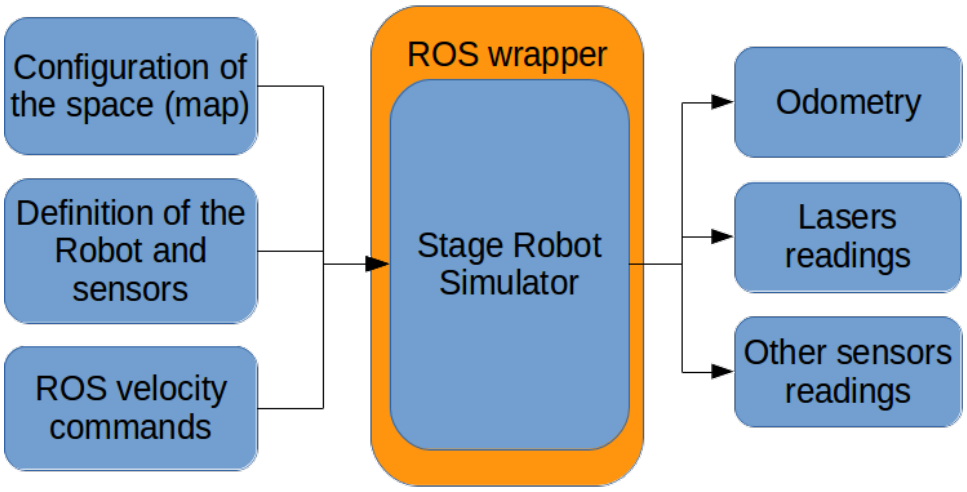


Figure 2: Diagram of the Stage Robot Simulator in ROS.

the data in ROS was split into different topics according to the number of lasers integrated in a single robot.

Another aspect required was the simulation of persons and other moving objects standing still or in motion within the environment at a simple level. For this purpose, Stage supports the use of "controllers", which are chunks of code that control simulated robots (persons in our case) from inside the simulator, instead of being on the other end of a ROS connection. This way, some behaviors and predefined routes were programmed and applied to some stage models. Therefore, we obtained other agents/persons moving around the environment independently of our robot controlled from ROS software.

Finally, in order to identify these models like "persons" we used the "fiducial" sensor of Stage. This sensor provides the location coordinates and orientation of other models that are detected and visible to the sensor in a predefined range. The ROS wrapper was programmed to take these data and publish the persons information in the same way that the real FROG robot does.

## 2.2 Software included with the simulator

### 2.2.1 Localization module

The simulator includes the localization module employed in FROG, strapped down as only 3 degrees of freedom are simulated in Stage (and no images). This module is based on Monte Carlo (or particle filter-based) localization approach, and is described in Deliverable D2.1 [1].

Even if the localization is provided in 2D, it allows one to perform basic testing on the augmented reality component. This is complemented with the ability to replay logged data with the full 6-DoF localization (see below).

### 2.2.2 Navigation module

The ROS wrapper for Stage is prepared to receive ROS movement commands. Thereby, the navigation system [2] can be executed in the same way in the simulator or in the robot. There-

fore, this converts the simulator into a useful tool for testing the navigation system. Together with the other modules presented, it is possible to test the communication and the navigation behavior in a proper way before performing tests in the real robot.

### 2.2.3 frog_adaptation_module

This ROS module has been developed as an intermediate software. It is meant to receive the high-level navigation commands from the UT's interaction computer, that control the behavior of the robot, and transform them to the proper ROS navigation commands. The high-level commands are "navigate to a specific point", "go to face a person detected", or "cancel the actual navigation goal", among others. Then, the ROS commands can be received and executed in the real robot or the simulator.

Furthermore, the frog_adaptation_module is in charge of sending the location of the robot, the persons detected and the navigation execution status to the State Machine in the interaction computer. This information is reported continuously in the proper format.

### 2.2.4 upo_advertise module

The upo_advertise module is responsible for advertising working message topics to other modules of the FROG system, allowing subscriptions or publications on those topics. Furthermore, this module monitors the number and frequency of messages sent and received for the different data streams, and can warn of abnormal rates as debugging tool.

## 3 Virtual machine for simulation and integration

As the different partners are coding under different environments and operating systems, and also as a way to avoid all the installation procedures, the previous simulator is provided within a virtualized computer, which can be run over virtualization clients like VirtualBox.

The virtual machine consists of an Ubuntu 12.04.1 LTS image with ROS Groovy pre-installed. The virtual machine takes 9Gb of space in HD and has been configured and tested with 3Gb of RAM. It has also been tested with 1GB of RAM, with slower operation.

VirtualBox 4.1.12 is the client used for testing the modules described in this document, running in an Intel® Core™ i7-2600 CPU@3.40GHz computer with 6Gb RAM.

The virtual machine comes with the described simulator installed, so the FROG robot can be simulated in any computer with the system requirements described and connected to the network (real or virtual). Furthermore, the virtual machine comes with a large part of the real navigation software running on the real robot, as indicated above. Some scripts were also prepared to make it easier to run different modules in charge of the communications, the simulation and the navigation.

An additional script is provided with the simulator to play real data stored in the virtual machine in ROSbag format[2] (to be used instead of simulator).

The main modules relative to communication and debugging are described below.

[2] http://wiki.ros.org/rosbag

## 3.1 Integration using ROSbridge

ROSbridge[3] provides a JSON protocol under websocket layer to communicate with non-ROS modules used by the rest of the partners (as discussed in Deliverable 5.1). The way partners can communicate each other is by connecting to the ROSbridge websocket server and publishing commands in topics[4] and receiving by subscription to other topics.

For instance, any external software can get the robot pose by subscribing to the topic */FROG_pose*, that is published by the simulator at 10Hz as described in Section 2. Messages received and sent follow the JSON format and ROS messages structure:

```
{
    "topic": "/docking/state",
    "msg": {
        "data": "aborted"
    },
    "op": "publish"
}
```

Furthermore, the virtual machine is delivered with an open-source Java websocket client[5] (see Fig. 3), which can be used for debugging websocket-based communications.

This client can be easily connected to ROSbridge and publish data or subscribe to topics by sending JSON strings. This way, it is possible to verify that ROSbridge is working correctly and the whole system is behaving as expected.

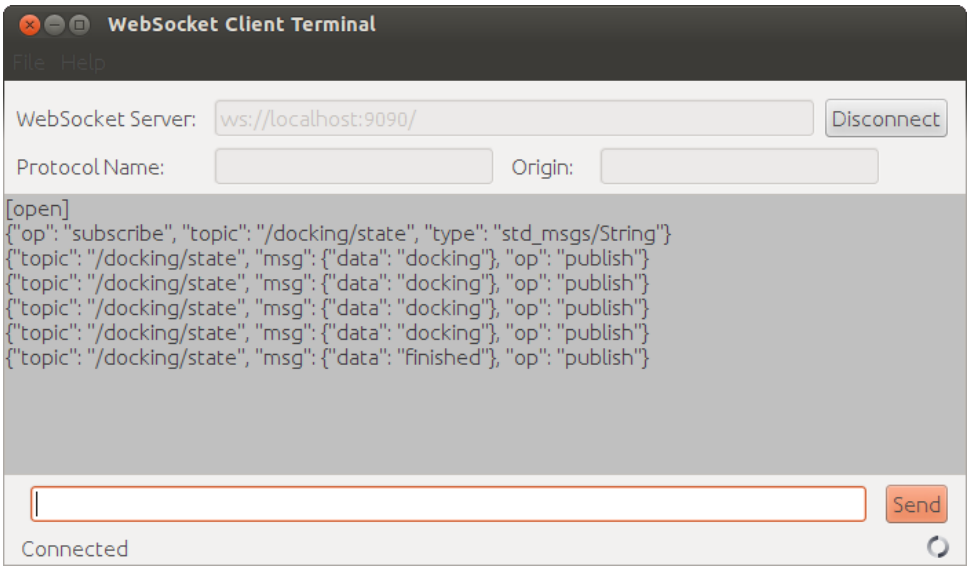


Figure 3: Java websocket client terminal connected to ROSbridge websocket client. First command requires subscription to topic */docking/state*, the following lines are messages sent by simulator through required topic.

[3] http://wiki.ros.org/rosbridge_suite
[4] http://wiki.ros.org/ROS/Concepts
[5] https://github.com/p-/WebSocket-Client-Terminal

### 3.2 Real data gathered

The virtual machine contains also 1878 seconds of real data gathered at the Royal Alcázar of Seville with the FROG robot and can be played by executing one script, starting to publish saved data of the robot. The modules installed in the virtual machine and other external modules can be executed over this recorded data to test algorithms and simulations in the same way of live execution. A short description of most important topics, number of messages and datatype[6] is shown in Table 1.

Table 1: Topics saved in real data bag at Royal Alcázar of Seville.

| Topic | Messages | Datatype |
|---|---|---|
| /FROG_pose | 18731 | FROG_msg/PoseRobot |
| /UvA_PersonDetectionAndPose | 3528 | FROG_msg/PersonsUVA |
| /diagnostic | 939 | frog_driver/Diagnostic |
| /imu/data | 174840 | sensor_msgs/Imu |
| /scan360 | 51136 | sensor_msgs/LaserScan |
| /scanback | 75150 | sensor_msgs/LaserScan |
| /scanfront | 75327 | sensor_msgs/LaserScan |
| /scanvtcal | 18867 | sensor_msgs/LaserScan |
| /tf | 210571 | tf/tfMessage |

- `/FROG_pose` publishes the pose of the robot in the map frame, following the ROS format `geometry_msgs/PoseWithCovariance`, and including the status of the localization (normal or lost).
- `/UvA_PersonDetectionAndPose` is the topic that contains information about people detected by the stereo pair cameras.
- `/diagnostic` contains data of living state of the FROG driver, mainly for debugging.
- the scans topics publish measurements from different laser rangefinders (frontal, back, tilted and a $360^o$ virtual laser created from the combination of the frontal and rear lasers).

## 4 Sketching tool

Besides simulating the robot and the navigation behaviors, it was required to be able to design and test different animations of the FROG eyes, one of the ways the robot shows its own internal "emotions".

### 4.1 The Eyes Animation Maker

Animations of the eyes of the FROG robot can be designed with the Eyes Animation Maker. The Eyes Animation Maker was developed by YDreams to make sequences for the LED lights of the eyes. All LEDs (2 x 96) in the eyes can be used separately. YDreams made a tool

[6] Default datatypes defined at http://wiki.ros.org/common_msgs

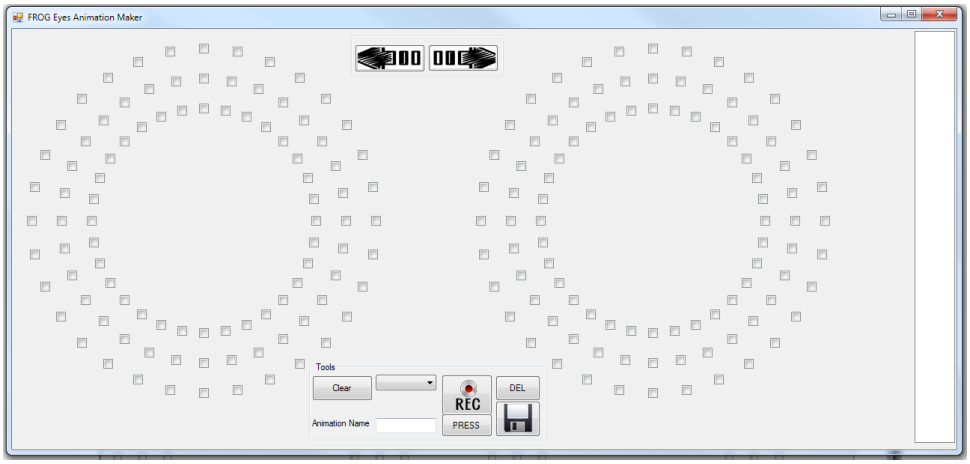


Figure 4: Eye Animation Maker developed by YDreams. Different led configuration and patters can be saved to make each eye animation.

that clearly shows both eyes and the three rings of LEDs in each eye. Fig. 4 shows the user interface of the Eyes Animation Maker.

## 4.2 Designing an eye animation

The user interface of the Eye Animation Maker represents the two eyes of FROG. Each eye has 32 check boxes in each ring which represent the LEDs of the eyes. An eye animation is a sequence of patterns. These patterns can be designed by selecting or deselecting the check boxes. The LEDs can be turned on or off by selecting or deselecting the check boxes. Fig. 5 gives an example of a designed pattern in the Eye Animation Maker and the result of it when executed on the Frog.

The design for of one of the eyes can be copied to the other eye by using one of the arrows in the top of the user interface, but the two eyes do not have the same pattern. The clear button can be used to turn off all the LEDs of one or both eyes. When a pattern is finished, it can be saved as one of the patterns in a sequence by clicking the record button. Patterns are stored as a bitmap. Fig. 6 shows an example of a pattern stored as a bitmap. The last pattern of the sequence can be deleted using the delete button. When the sequence is finished, it can be saved by giving it a name and clicking the save button. The sequences are saved as images in a folder named as the sequence (e.g. "nice").

## 4.3 Executing the eye animations

When the design of the patterns and sequences are finished they are stored as an animation. The eyes of FROG are controlled by the Eyes Controller. The Eyes Controller is responsible for turning on and off the eyes, controlling the brightness of the LEDs of the eyes and for displaying the animations designed with the Eyes Animation Maker. The Eyes Controller is listening to commands sent over a network connection.

An animation can be executed by calling the function *animate_eyes(string name, integer times, integer speed)* of the eyes controller. Name is the name of the animation and should match the name of the sequence. Times will specify how many times the animation will be played (-1 will loop the animation forever) and the speed will specify the time it will take to play one

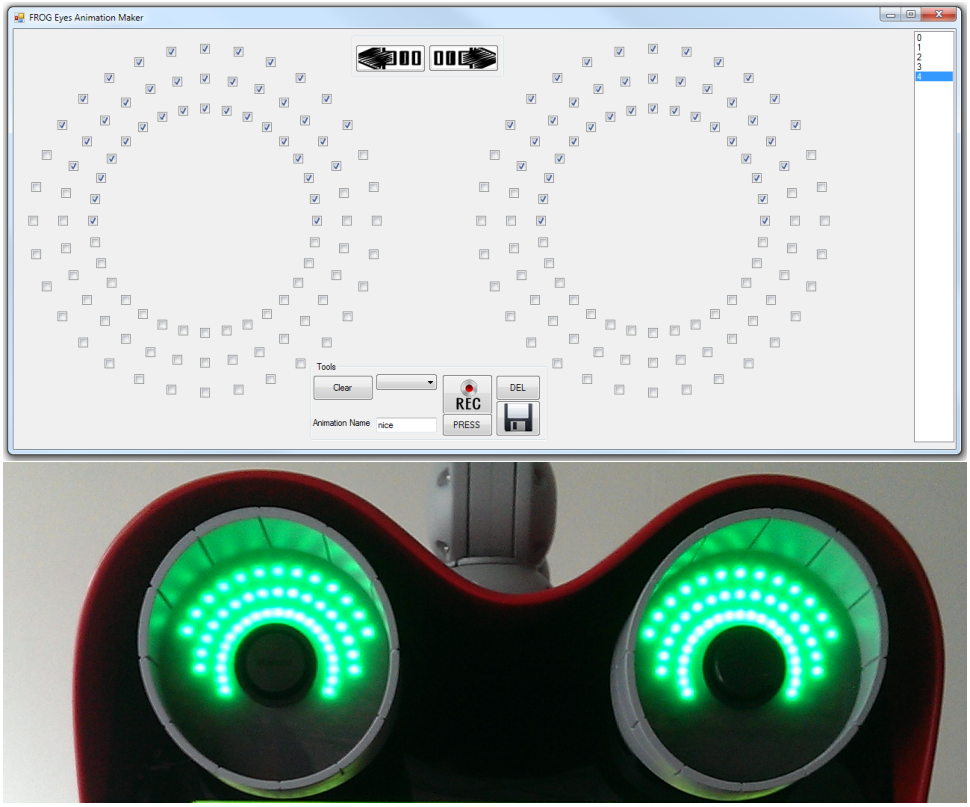


Figure 5: Example of an eye animation in the editor and the result on FROG's eyes.

Figure 6: Example of pattern made with the Eyes Animation Maker and saved as a bitmap.

sequence in milliseconds. The animation of the eyes can always be stopped by calling the function *stop_eyes()*. When the animation of the eyes is stopped, a default pattern belonging to the animation will be displayed.

# 5 Conclusions

The simulator, composed by Stage and the FROG driver, allows other partners to connect to a virtual robot and test sending commands to control the robot navigation behavior while simulating the environment around the robot, including a real map of the Royal Alcázar of Seville. With both navigation control and robot and environmental responses it is possible to test the AR software by executing specific content at corresponding places

For testing the complete behavior of the robot, a virtual machine has been provided with the simulator and real modules already tested in the FROG robot, related to communication between partners, localization and navigation.

The Java websocket client mentioned above proved to be very helpful in debugging tasks, due to its simplicity and the fact of being a cross-platform application that can easily connect to a ROSbridge websocket server, send and receive commands and verify the correct operation of the complete software.

# Appendices

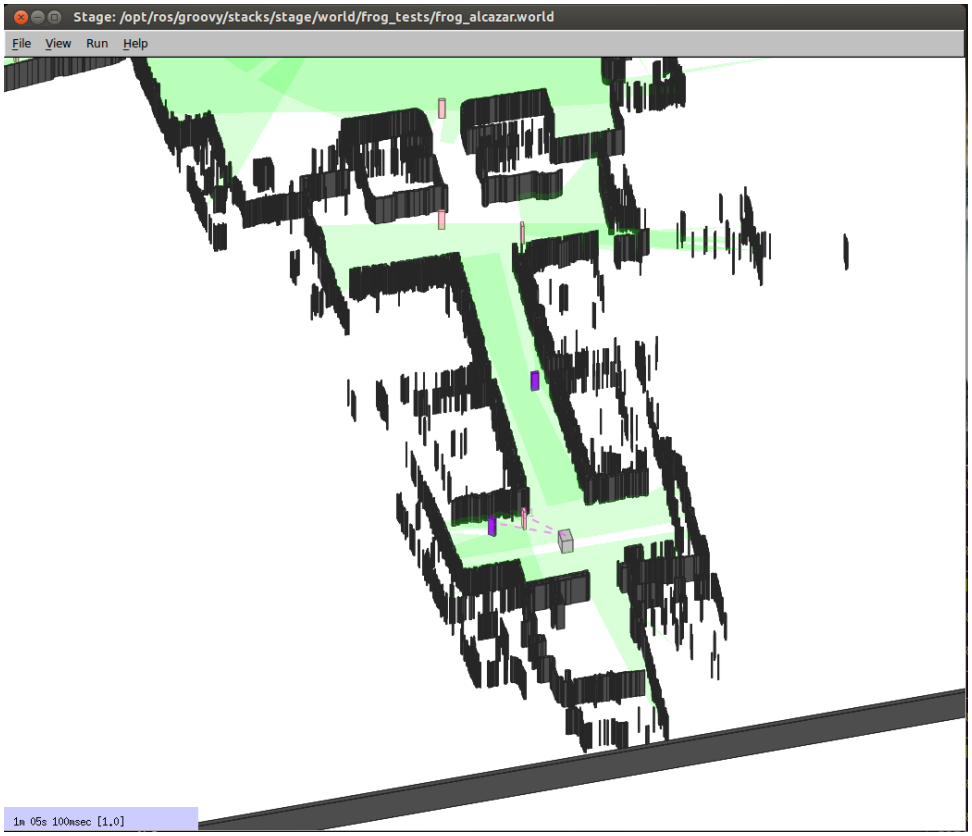


Figure 7: Stage simulation with defined world and robot. It reacts to received commands and publishes information relative to robot movement and sensors' measurements according to the simulated world. The map used for simulation is a low resolution version of the map built of the Royal Alcázar of Seville.

## Robot and environment definitions in Stage

Stage simulates a world as defined in a file with extension .world, as seen in Fig. 7. This file tells Stage everything about the world, from obstacles (usually represented via a bitmap to be used as a kind of background), to robots and other objects. Therefore, we must provide in this file all the information that defines as closely as possible the project environment (The Royal Alcazar of Seville) and the FROG robot features.

### Map definition

For the simulation in Stage we can use the same map that is used in the real system for localization and for navigation. Then, we define the properties of the obstacles in the map, such as color and which sensors can detect them. Moreover, we can indicate the window size, the point of view of the scene, and some information that we can show in the Stage screen. Here, a snippet of the world file corresponding to the scene configuration is shown.

```
#———— MAP PROPERTIES ————————————————————————————————
define floorplan model
(
color "gray30"
boundary 0
gui_nose 0
gui_grid 0
gui_move 0
gui_outline 0
fiducial_return 0
laser_return 1
ranger_return 1
obstacle_return 1
)
# set the resolution of the underlying raytrace model in meters
resolution 0.05 # meters/pix
interval_sim 100 # simulation timestep in milliseconds

#—— STAGE WINDOW ——————————————————————————————————————-
window
(
size [ 1863 1056 ] # in pixels
# camera options
center [ 72.560 43.712 ]
rotate [ 0.000 0.000 ]
scale 10.263 # pixels per meter
# perspective camera options
pcam_loc [ -32.981 27.945 33.200 ]
pcam_angle [ 0.000 -3093.972 ]
# GUI options
show_data 1
show_flags 1
show_blocks 1
show_clock 1
show_footprints 0
show_grid 0
show_occupancy 0
show_tree 0
pcam_on 0
screenshots 0
)

#load an environment bitmap ———————————————————————————————
floorplan
(
name "alcazar-stage"
bitmap "maps/maps_Alcazar/map.pgm"
size [143.600 150.800 2.500] # x pix*0,05 m/pix = meters
pose [ 71.800 75.400 0.000 0.000] )
```

### FROG Robot and sensors definition

First, we can define the properties of the sensors of the FROG robot. In this case, we use laser range finders and a fiducial model. Below, the portion of the world file with the sensors definition is presented.

```
#—— HOKUYO LASER ————————————————————————————-
define hokuyolaser ranger
(
sensor(
# laser-specific properties
# settings for Hokuyo UTM-30LX
range [ 0.0 30.0 ] # range of 30 meters
fov 180.0
samples 270
)
color "purple"
size [ 0.070 0.070 0.050 ] # dimensions from data
sheet )

#—— FIDUCIAL SENSOR ————————————————————————————
define fidu fiducial
(
color "purple"
size [ 0.070 0.070 0.050 ]
range_min 0.4 #meters
range_max 8 #meters
range_max_id 8 #meters
fov 180.0 #field of view degrees
)
```

Once the sensors are defined, we can define the properties of the FROG robot such as acceleration and velocity limits, odometric errors, size or weight, as it can be seen below. Moreover, we can indicate the sensors that the robot has and their location and orientation.

```
#——- FROG BASE ————————————————————————————————-
define frog_base position
(
color "gray" # Default color.
drive "diff" # Differential steering model.
gui_nose 1 # Draw a nose on the robot so we can see which way it points
obstacle_return 1 # Can hit things.
ranger_return 0.500 # reflects sonar beams
blob_return 1 # Seen by blobfinders
fiducial_return 2 # Seen as "1" fiducial finders
# alternative odometric localization with simple error model
localization "odom" # Change to "gps" to have impossibly perfect, global odometry
odom_error [ 0.05 0.05 0.0 0.05 ] # Odometry error or slip in X, Y, Z and Theta
# (Uniform random distribution)
# four DOF kinematics limits
# [ xmin xmax ymin ymax zmin zmax amin amax ]
velocity_bounds [-0.8 0.8 0 0 0 0 -90.0 90.0]
acceleration_bounds [-1.0 1.0 0 0 0 0 -90 90.0 ]
)

# ———————— The FROG standard configuration ——————————————————-
define frog frog_base
(
# Actual size
size [1.100 0.800 1.200]
# Estimated mass in KG
mass 80.0
# sensors
hokuyolaser(pose [0.420 0.000 -1.000 0.000]) #pose[x y z yaw]
hokuyolaser(pose [-0.420 0.000 -1.000 180.000])
fidu()
)
```

### Person models definition

To simulate the persons that will be detected by the fiducial sensor or the robot, we create a model in a similar way that we did for the FROG robot but indicating the person features.

```
#—— PERSON ————————————————————————————————-
define person position
(
color "pink" # Default color.
drive "diff" # Differential steering model.
gui_nose 1
obstacle_return 1 # Can hit things.
ranger_return 0.500 # reflects sonar beams
blob_return 1 # Seen by blobfinders
localization "gps"
localization_origin [0 0 0 0] # Start odometry at (0, 0, 0).
# four DOF kinematics limits
# [ xmin xmax ymin ymax zmin zmax amin amax ]
velocity_bounds [-2.0 2.0 0 0 0 0 -90.0 90.0 ]
acceleration_bounds [-1.5 1.5 0 0 0 0 -90 90.0 ]
# Actual size
size [0.20 0.45 1.78]
# The center of rotation is offset from its center of area
origin [0.0 0.000 0.000 0.000]
# Estimated mass in KG
mass 70.0
hokuyolaser(pose [ 0.10 0.000 -1.200 0.000 ])
)
```

Finally, at the end of the world file, we indicate the robots and persons and their locations in the map. Moreover, for each person we indicate the "Stage controller" that will control the movement of the mobile persons. In the following snippet of the world file, an example of this is shown. We use two kinds of person movement controllers: the "wander" controller, that makes the person wander on the map and avoiding the obstacles; and the "goal_directed" controllers, that moves the person from the initial location to a final location and repeat the process.

```
frog( pose [ 102.400 23.850 0.000 90.000 ] name "frog")
person( pose [ 102 61 0.000 0 ] name "person1" ctrl "wander" fiducial_return 10)
person(
pose [ 104 52 0.000 -90 ]
name "person2"
fiducial_return 11 #value returned by the fiducial sensor when it detects this person
ctrl "goal_directed 0.8 repeat 100 26.5 50 104 52 50"
# goal_directed ’cruise_velocity’ ’repeat or once’ x_goal’ ’y_goal’ ’waitduration’
)
person(
pose [ 102.5 20.5 0.000 45 ]
name "person3"
fiducial_return 12
ctrl "goal_directed 0.9 repeat 102 57.9 50 102.5 20.5 50"
# goal_directed ’cruise_velocity’ ’repeat or once’ x_goal’ ’y_goal’ ’waitduration’
)
```

# Bibliography


[1] FROG Consortium. Deliverable D2.1: Robot 6DOF precise localization component. https://www.frogrobot.eu/wordpress/wp-content/uploads/2014/03/d2.1-v1.1.pdf.

[2] FROG Consortium. Deliverable D2.2: Path planning and execution component for efficient and human-aware navigation. https://www.frogrobot.eu/wordpress/wp-content/uploads/2014/01/d2-2.pdf.

[3] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, June 2003.

[4] 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.

[5] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, pages 189–208, 2008.