FROG

FUN ROBOTIC
OUTDOOR GUIDE

Deliverable: D2.4

AR Robot Application component

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)
UNIVERSITY OF TWENTE (UT)

Grant agreement no. **288235**

Funding scheme **STREP**

UNIVERSITEIT TWENTE.

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:	36 Months
Call topic:	ICT-2011.2.1 Cognitive Systems and Robotics (a), (d)
Project web-site:	www.frogrobot.eu

Document	
Deliverable number:	D2.4
Deliverable title:	AR Robot Application component
Due date of deliverable:	M23 – 31 August 2013
Actual submission date:	M23 – 29 August 2013
Editors:	YD, UT
Authors:	YD, UT
Reviewers:	UT
Participating beneficiaries:	2,3,4,5,6
Work Package no.:	2
Work Package title:	Robot Control, Navigation and Location Based Content
Work Package leader:	UPO
Work Package participants:	2,3,4,5,6
Estimated person-months for deliverable:	15
Dissemination level:	Public
Nature:	Other
Version:	Final
Draft/Final:	Draft
No of pages (including cover):	31
Keywords:	Augmented Reality, robot software frameworks

Table of Contents

Summary

The goal of this deliverable is to report the implementation details of the Augmented Reality module in the FROG project as well as to provide preliminary results of the performance evaluation. Since the initial conceptualization of the project, Augmented Reality has been considered a distinctive feature that can enhance the attractiveness of a robot guide. While the social aspects of the robot behavior are built upon the analysis of human guides, augmented reality is clearly a content presentation strategy that is unique to computational media. The FROG robot was designed to include hardware and software components that allow and facilitate the presentation of this type of content. It includes a monitor, a video projector, a laser pointer, LED lights and speakers that are directly controlled by this module, supplying services that are used by the Behavior module.

Considering that Mobile robots, such as FROG, and Augmented Reality share a crucial characteristic, localization, this module makes use of the precise 6DOF localization information provided by the navigation module in order to calculate multimedia content localization.

Although the AR module makes use of the localization information from UPO, it was decided that it would have its own coordinates system to achieve a higher independency. The objective of this strategy was to facilitate the decoupled development of the several software modules and so that the AR module can be independent from the robotic framework where it is applied. This could also lead to a software product of its own as a functionality that can be added to other robots, making use of different localization and coordinates systems. This adequacy is easily obtained by establishing a transformation matrix to each system.

The relationship and communication of the AR module with the other modules in the system follows the publish/subscribe model defined in the integration framework.

This module is also responsible for playing the expressive animations of the antenna. A specific tool was developed to design these animations.

1. Introduction

The Augmented Reality (AR) Module is responsible for providing services on Augmented Reality and Multimedia. These services are mainly requested by the Behavior Module, but they can also be triggered when a set of predefined conditions are satisfied. These services are the following:

- Show an image
- Show a 3D model
- Play a video
- Play a sound
- Point the laser
- Play antenna animation

The FROG robot provides two different image output devices with which imagery content can be displayed: video projection and monitor. The AR module is also responsible for displaying the content through the adequate output, according to the request.

The module also contains services to retrieve information about space located items from the point of view of the Robot. These services are used by the Behavior Module by request or when a set of conditions are met.

As an example, the robot in an arbitrary position and orientation can use the antenna to point in the direction of the exit or the nearest toilet (this will not be implemented but illustrates an example of the module services potential). The system can also ask the module if some item in real space is within a specific range of directions relative to the robot.

2. AR and localized content

During execution, the module converts references to Locations in Real Space, back and forth between the robot's local point of view at any moment, and an external absolute referential. For AR purposes, some references are projected according to the model of the output device it is destined to use.

The notion of the robot's current position and orientation is obtained from UPOLOC module, in regular updates.

In configuration time, Locations can be defined in Real Space, so they are known in the global system and can be used as route points, points of interest, and locations for content. Content items may be attached to Locations in Real Space, by the use of Smart Objects, which will be explained further on in this document.

3. Antenna

Fig. 1 – Antenna/Laser pointer rendering

The FROG robot includes an antenna which is functionally a robotic arm with a laser pointer at its end. This antenna on top of the FROG robot fulfills several objectives. It includes a laser pointer in order to point and highlight specific points in space and it is an expressive vehicle to convey information about the robot internal affective state (much like a dog tail).

3.1 As a pointing device

The coordinates of any given Locations in the real world can be computed relative to the robot. Then the antenna can point to that Location using inverse kinematics.

3.2 As an expression component

Being an articulated device, the antenna can perform gestures in order to express emotions. Animations can be recorded, which can be read by this module, and used as affective expressive content. These animations can be performed upon request or when a set of predefined conditions are met.

4. Integration in the robot framework

Most of the interaction the AR module has in the robot framework is with the Robot Behavior Module. Apart from the Robot Behavior Module, the relationship with other modules of the system is made through the general integration framework.

4.1 The Content Layer

A layer of protocol logic and conventions was developed, in order to provide a common notation to modules interacting with AR and Multimedia features included in this module.
Over this layer a set of objects can be defined, which are valid through different parts of the framework, and can be used by other modules in the global framework. Two of the most important structures are Smart Objects and Routines.

4.2 Smart Objects

Smart Objects are software objects that can aggregate content (data) and the logic (code) to manipulate it. A smart object contains all the information about itself so that it may be “consumed” by the robot framework. An example can be an object that includes a 3D model with all the logic for its exhibition and, eventually, the possible user interactions.
An important advantage of using Smart Objects, is the fact that a single Player can be developed and used generically, since it is the Smart Object who will be “telling” the player what to do with it. This avoids the development of a different player for each pair of content/behavior type.
Smart Objects benefit from some autonomous execution, which provides a considerable degree of protection from affecting one another in their execution.
Smart Objects can be known and referred to, throughout the framework, by the modules that include the Content Layer. In this case, only the behavior module includes this layer, however, the AR module was built so that any other can include it. An example could be a Smart Object that is played by the navigation module when a specific position is reached or an even lower level in an alarm situation.
For most Smart Objects, the routine is expected to manipulate the Content Objects assigned to them.

4.3 Behaviors and Routines

Behaviors are logical units that can execute at the AR Module with some degree of independence. They can be requested to execute or be triggered by the occurrence of a pre-specified event; in this case they are also called Routines. Other Behaviors can be associated with Smart Objects, these Behaviors are triggered by object activation. A set of contextual information is passed into each Behavior execution, this set includes information acquired from outside the module and module internal information. A Behavior may define a set of properties that can be used to pass values into a particular execution.

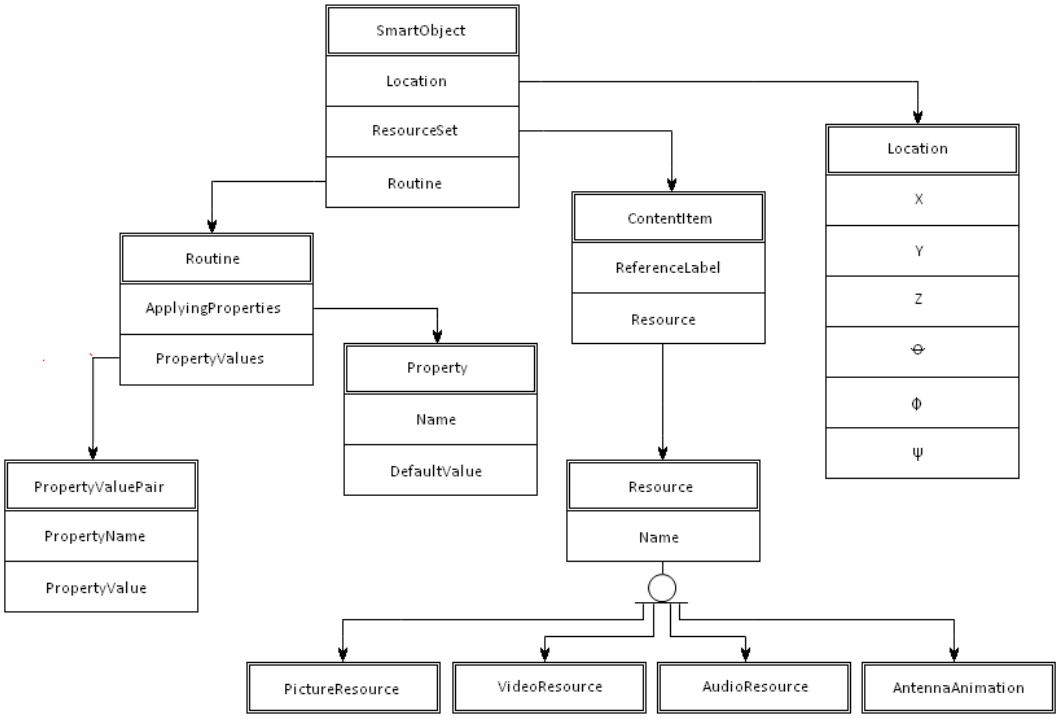
Every Routine execution request has an identifier which should be unique for each requester. The requester should subscribe the *ExecutionStarted*, *ExecutionFinished* and *ExecutionCancelled* topics, so it can be notified of the start, the finishing or the cancellation of the execution requested.

Behaviors associated with Smart Objects are expected to access the content of the object and possibly the location.

Other type of Behaviors, called Questions also define a set of possible answers. The requester subscribes the Answer topic to receive the answer.

4.4 Relationships between types

Some of the most important types are represented in the diagram below, along with the relationships between them.



5. Managing Content

The AR module expects specifications and configurations of content in a set of text files in predefined file system locations.

Typically, raw content items, such as Videos, Images and the like are specified for loading. Then more complex structures are defined with reference to content items provided.

The main Media Resource Types we have considered for FROG are: Picture resource, Video resource, Sound resource, Antenna animation Resource.

5.1 Content Files

Raw content items and other objects of the Content Layer are stored in text files in the local file systems.
An example of a Smart Object with a routine and a few content items, is presented in annex 2.

5.2 Locations

In order to assign a location to a Smart Object, the latter has to be configured with the location coordinates.
The coordinates can be acquired with the help of a calibrated plane of the area, as provided by this module. By picking a location on the plane, their coordinates are displayed. If the object is to be placed at a specific height, this value must be added to the coordinates, because only locations at floor level can be picked from the plane.

5.3 Content Manager

Content management software was developed to ease the task of specifying configuring and relating content.

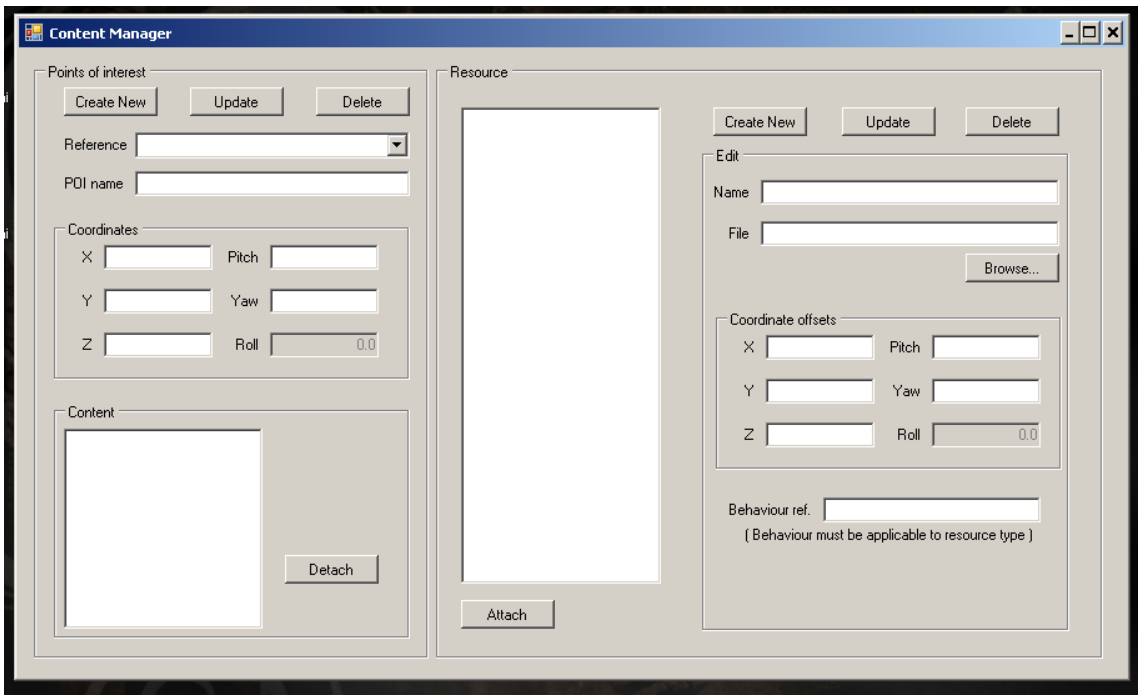


Fig. 2 - backoffice content manager GUI

Using this software, content files can be specified to be loaded into the system, Smart Objects can be created, locations can be given, and routines attached. The software updates the content data files accordingly.

6. Particular coordinate space

So that content level configurations, do not get their meanings altered by changes in the navigation system, content locations are defined in a separate coordinate space.

At runtime, locations specified in the global referential, like the position updates from the navigation module, must be converted into this Particular coordinate space before being used. Before leaving the Content Layer, coordinates passed unto the global framework must be converted to the Navigation Referential.

6.1 Space Transform

Two real space points were defined, to be used as calibration references. Every time a new scenario is mapped and every time the navigation referential changes, the new coordinates for the calibration reference points must be provided. The new transform can then be calculated.

This transform is determined by a 4x4 matrix. A point can be multiplied by such a matrix, after being converted into homogeneous coordinates and interpreted as a row vector.

Conversion of point into homogeneous coordinates

$P(p_x, p_y, p_z) \rightarrow P_h(p_x, p_y, p_z, 1)$

Multiplication by the transform matrix

$$\begin{matrix} p_x & p_y & p_z & 1 \end{matrix} \times \begin{matrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{matrix} = \begin{matrix} q_x & q_y & q_z & 1 \end{matrix}$$

The considered referentials have the same scale and their y-axes are parallel. The final transform must include a rotation about the y-axis a translation. A handedness conversion must also be included, since the referentials have opposite handedness.

Annex3 contains the calculations of the transform extraction for a specific case at the Real Alcazar scenario. The extraction process is explained next.

6.2 Transform extraction

Based on the coordinates of reference points specified in UPOLOC referential, named $Ref1_U$ and $Ref2_U$, a matrix is calculated at module startup, for converting points from Content space to UPOLOC space, and its inverse for converting back from UPOLOC space to Content space.

U : Referential used by UPOLOC

C : referential used by AR Module (Content space referential)

$Ref1_U$, $Ref2_U$: reference points in coordinates of U.

$Ref1_C$, $Ref2_C$: reference points in coordinates of C.

6.2.1 Left/Right Handedness

UPOLOC uses a right-handed referential, while the Content space referential is left-handed. The reference point coordinates from U are first mirrored about the xy-plane, considering a left handed referential we will call U_L, whose y and x-axis coincide with those of U, but the z-axis is directed in the opposite way. All further operations in the transform extraction method are used as defined for left-handed coordinate systems. At last, the handedness conversion will be included into the final transform matrix.

Rotation angles are negated in handedness conversion, in order to be preserved between left and right-handed rotation conventions.

Mirroring a point about the xy-plane is equivalent to negating the z component of the coordinates or applying the handedness conversion matrix:

$$H = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Calculating the U_L version of the reference points:

$Ref1_{U\_L} = Ref1_U \times H$

$Ref2_{U\_L} = Ref2_U \times H$

6.2.2 Rotation from C to U_L

By convention, the xz-planes of both referentials are parallel. This means in terms of rotation, that referentials U_L and C can only differ by yaw.

To determine the angle from C to U_L the vector from Ref1 to Ref2 is calculated in both versions:

$\overrightarrow{V_C}$ = $\text{Ref2}_C - \text{Ref1}_C$

$\overrightarrow{V_{U\_L}}$ = $\text{Ref2}_{U\_L} - \text{Ref1}_{U\_L}$

The angle between $\overrightarrow{V_C}$ and $\overrightarrow{V_{U\_L}}$ is then computed. The direction of the rotation is calculated using the cross product vector, which is collinear with the y-axis:

$$\overrightarrow{cProd} = \overrightarrow{V_C} \times \overrightarrow{V_{U\_L}}$$

$$\text{signal} = \frac{\overrightarrow{cProd}.\text{y}}{|\overrightarrow{cProd}.\text{y}|} \quad .$$

The rotation magnitude is calculated using the dot product between vectors:

$$\text{angle} = \text{acos}\left(\frac{\overrightarrow{V_C} \cdot \overrightarrow{V_{U\_L}}}{||\overrightarrow{V_C}|| \times ||\overrightarrow{V_{U_L}}||}\right) \quad .$$

The angle rotation angle around the y-axis, θ:

$$\theta = \text{signal} \times \text{angle} \quad .$$

The matrix representing this rotation is a y-axis rotation matrix:

$$M_{rot} = \text{Rot}_y(\theta) = \begin{matrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

6.2.3 Translation from C to U_L

To compute the translation vector from C origin to U_L origin, one of the reference points is chosen and its coordinates in C referential, are rotated back to align with U_L referential.

$$Ref1_{CRot} = Ref1_{C} \times M_{rot} \quad .$$

T translation vector will be:

$$\vec{T} = Ref1_{U\_L} - Ref1_{CRot}$$

and the translation matrix will be:

$$M_{trl} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \vec{T}_x & \vec{T}_y & \vec{T}_z & 1 \end{matrix} \quad .$$

6.2.4 The transform matrix

The transform matrix from C to U_L can now be calculated:

$$M_{C2U\_L} = M_{rot} \times M_{trl} \quad .$$

The final transform matrix transforms from referential C to referential U, so it has to include the handedness transform.

$$M_{C2U} = M_{C2U\_L} \times H \quad .$$

For converting back from U to C the inverse of M_{C2U} is used:

$$M_{U2C} = M_{C2U}^{-1} \quad .$$

7. Evaluation of the AR application performance and the effectiveness of the interaction strategies

In order to evaluate the AR application performance and the effectiveness of the interaction strategies a field test was held during the integration and data collection week in the Lisbon Zoo in July 2013.

In order to prepare this field test, the table with multimedia content of Deliverable 2.3 was used and complemented with suggestions for robot interaction strategies that resulted from research in WP4. Two storyboards were worked out in more detail for the test. The first storyboard used a wall projection in the Tigers' Valley and the second storyboard used the antenna to point with the laser at an information panel in the Primates' Temple. The test was a Wizard of Oz test hence the movements of the robot and the sound were manually operated.

Unfortunately, shortly before the field test started it appeared that the speakers of the robot did not work. There was not enough time to find the cause of this defect (which later appeared to be quite easily solvable). A laptop was used for the sound instead. Consequently the volume of the sound was far too low and the robot could not be heard, neither by the visitors of the zoo, nor by the person operating the robot, so the field-test mainly failed.

The most important thing we learned from this test is that the zoo is a very noisy environment. At most places the background noise from animals, birds, waterfalls and people is very loud. Therefore it is recommended to use powerful speakers that can be directed to the visitors that are being addressed by the robot.

Later we learned that the robot's speakers would not have succeeded in overcoming the noise either. At the time of writing IDMind is looking for more powerful speakers and YDreams is adapting their presentations to ambient noise levels at both sites.

Annex 1 - Types in AR and in AR-Behavior relationship

- **question** (questionId) - Predefined routine, to which a user-chosen answer is expected.

- **answer** (answerId) - Predefined answer, which can be one of the expected answers for each defined question.

- **resource** (resId) - Playable on one or more displays depending on resource type. Can be associated to a behavior and a location with an object.

resource types:

.**sound** - A sound or sequence of sounds. These resources may represent noises, speech tokens or music.

.**graphical object** - 3D model, 2D object with some content (video, text, etc.) or other visual effect.

.**antenna animation** - predefined sequence of commands to FROG antenna in direct or inverse kinematics fashion.

- **object** (oId) - Object with a location in the world. May contain one or more playable resources, and a behavior. The behavior may reference to the resources and the location as well as contextual data.

- **display** (displayId) - May be one of { Screen, Projector1, AllValidDisplays, Antenna, Speaker 1, Speaker 2, Speakers }

- **logicalPredicate** (predicateId) – Predefined or derivative predicate whose value may depend totally or partially on module inputs.

- timeElapsed (timeInterval);

- momentReached (dateTime)

- inRangeOf (location, radius)

- isPeopleDetected

- and (predicateId, predicateId)

- or (predicateId, predicateId)

- not (predicateId)

derivative predicates may be composed based on predefined predicates. Predicates are provided to behaviors as part of the context.

- **behavior** (bId) - Basic or derivative behavior, applicable to a specific domain of resource types.

basic behaviors:

_ display (parameters: duration, repeat) (applies to: sound, graphical, antennaAnim)

- displayAt (parameters: same as play plus set of displayId's) (applies to: sound, graphical)

- stop/turnOff (no parameters) (applies to: sound, graphical, antennaAnim)

- changeLocation/pointTo (parameters: worldCoordinateSet or predefinedPlace) (applies to: sound, graphical, antennaAnim (make antenna point at location))

- onCondition (parameters: predicateId);

- sequence (params: beh_1, ..., beh_N, failPolicy)

- parallel (params: beh_1, ..., beh_N, failPolicy)

derivative Behaviors : constructs based on other behaviors, suitable for more complex procedures.

default behavior – predefined behavior, executes at module start.

- **behaviorProperty** (propId) – placeholders where to pass parameters to and from behaviors invoked.

Behaviors define a set of applicable Properties and default values for each one. If one of the properties is not mentioned at execution start, the default value for that property will be used.

Properties:

Duration (time)

Location (oId or coordinates)

ScreenPosition (2D coordinates)

Question (questionId)

Answer (answerId)

Object (oId)

Display (displayId)

StartPolicy (startPolicy)

ExecId (execId)

…

- **execId** - Denotes a particular execution of a behavior

- **executionResult** (resultId) - One of { finished, canceled}

- **startPolicy** - One of { Now Next, DontCare }

Module Inputs/Outputs (subscribe/publish)

- [In] Start

  Same as Execute[DefaultBehavior { StartPolicy = Now|Dontcare; ExecId = null}]

- [In] Stop

  Same as: Cancel[ExecId = null]

- [In] Execute [behId, startPolicy, execId, {Property0 = val0, Property1 = val1, …}]

- [Out] Execution Started [ExecId = execId]

- [Out] ExecutionFinished [{ ExecId = execId; Result = resultId}]

- [In] LocationUpdate [coordinates]

- [In] PeopleTracked [{poepleTracked = [(id0, cords, gesture), (id1, cords, gesture), …] }]

- [In] Cancel [ExecId = execId]

- [In] Question [QuestionId = questionId]

- [Out] Answer [QuestionId = questionId, Result = resultId]

Annex 2 - Content file examples

Resource File: “Tilemap.res”

```
{
        "Name" : "Tilemap",
        "Filename" : "tilemap04.jpg",
        "CoordinateOffsets" :
        {
                "X" : 0.0,
                "Y" : 1.233,
                "Z" : 0.0,

                "Pitch" : 0.0,
                "Yaw"   : 0.0,
                "Roll"  : 90.0,
        },
        "TimeOffset" : 0.0
}
```

ResourceFile: “TileSong.res”

```
{
        "Name" : "TileSong",
        "Filename" : "tilemusic.mp3",
        "CoordinateOffsets" :
        {
                "X" : 0.0,
                "Y" : 0.0,
                "Z" : 0.0,
                "Pitch" : 0.0,
                "Yaw" : 0.0,
                "Roll" : 0.0,
```

```
        },
        "TimeOffset" : 4.0
}
```


_____

Routine File: “ProjectTilesWhilePlayingMusic.routine”

_____


```
{
        "Name" : "ProjectTilesWhilePlayingMusic",
        "Reference" : "ProjectTilesWhilePlayingMusic",
        "ApplicableProperties" :
        [],
        "ContentProperties" :
        [
                {
                        "Name" : "Tilemap_IMAGE"
                },
                {
                        "Name" : "TileMusic_AUDIO"
                },
                {
                        "Name" : "TileAnimation_VIDEO"
                }
        ],
}
```


_____

Smart Object File: “TapestryRoomWall.sobj”

_____


```
{
        "Name" : "TapestryRoomWall",
        "Reference" : "TapestryRoomWall",
        "CoordinateOffsets" :
        {
                "X" : -225.452,
                "Y" : 0.0,
```

```
                "Z" : -183.038,
                "Pitch" : 0.0,
                "Yaw" : 0.0,
                "Roll" : 0.0,
        },
        "BehaviourReference" : "ProjectTilesWhilePlayingMusic",
        "ContentPropertyValues" :
        [
                {
                        "PropertyName" : "TileMap_IMAGE",
                        "Value" : "Tilemap04.jpg"
                },
                {
                        "PropertyName" : "Tilemusic_AUDIO",
                        "Value" : "TileSong.mp3"
                }
        ]
}
```

Annex 3 – Coordinates Transformation matrix

The example here refers to the Real Alcazar scenario. The reference points chosen are signaled in the following picture:

Fig. 3 – Coordinates spaces referential at the Real Alcazar

The coordinates of the chosen points:

In C referential:

$Ref1_C$ (-73.24401, 0.19, 57.36475)

$Ref2_C$ (-66.32367, 0.19, 65.12097)

In U referential:

$Ref1_U$ (107.2, 0.19, 93.40)

$Ref2_U$ (108.6, 0.19, 83.10)

Converting coordinates from U to U_L

$Ref1_{U\_L} = Ref1_U \times H$ = (107.2, 0.19, -93.40)

$Ref2_{U\_L} = Ref2_U \times H$ = (108.6, 0.19, -83.10)

Calculating $\vec{V}$ vectors for both C and U_L referentials

$\overrightarrow{V_C}$ = Ref2$_C$ – Ref1$_C$ = $\vec{V}$(6.920341, 0.0, 7.756218)

$\overrightarrow{V_{U\_L}}$ = Ref2$_{U_L}$ – Ref1$_{U_L}$ = $\vec{V}$(1.400002, 0.0, 10.3)

Extracting the rotation

$\overrightarrow{cProd}$ = $\overrightarrow{V_C} \times \overrightarrow{V_{U\_L}}$ = $\vec{V}$(0.0, -60.42082, 0.0)

signal = $\frac{\overrightarrow{cProd}.y}{|\overrightarrow{cProd}.y|}$ = $\frac{-60.42082}{60.42082}$ = -1

angle = $\frac{\overrightarrow{V_C} \cdot \overrightarrow{V_{U\_L}}}{||\overrightarrow{V_C}|| \times ||\overrightarrow{V_{U_L}}||}$ = -0.593411863 (≅ 34°)

θ = signal × angle = 0.593411863 (≅ 34° clockwise)

M_{rot} = $\begin{matrix} 0.8290376 & 0 & 0.5591928 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5591928 & 0 & 0.8290376 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$

Extracting the translation

Ref1$_{CRot}$ = Ref1$_C$ × M_{rot} = $\vec{V}$(-92.8, 0.19, 6.600011)

$\vec{T}$ = Ref1$_{U_L}$(107.2, 0.19, -93.40) - Ref1$_{CRot}$(-92.8, 0.19, 6.600011)

= (200.0, 0.0, -100.0)

M_{Trl} = $\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 200.0 & 0 & -100.0 & 1 \end{matrix}$

Transform matrix for converting coordinates from C to U

$$M_{C2U} = M_{rot} \times M_{Trl} \times H = \begin{matrix} 0.8290376 & 0 & -0.5591928 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5591928 & 0 & -0.8290376 & 0 \\ 200.0 & 0 & 100.0 & 1 \end{matrix}$$

Transform matrix for converting coordinates from U to C

$$M_{U2C} = M_{C2U}^{-1} = \begin{matrix} 0.8290376 & 0 & -0.5591928 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5591928 & 0 & -0.8290376 & 0 \\ -109.8882 & 0 & 194.7423 & 1 \end{matrix}$$

Annex 4 – Frog State Editor

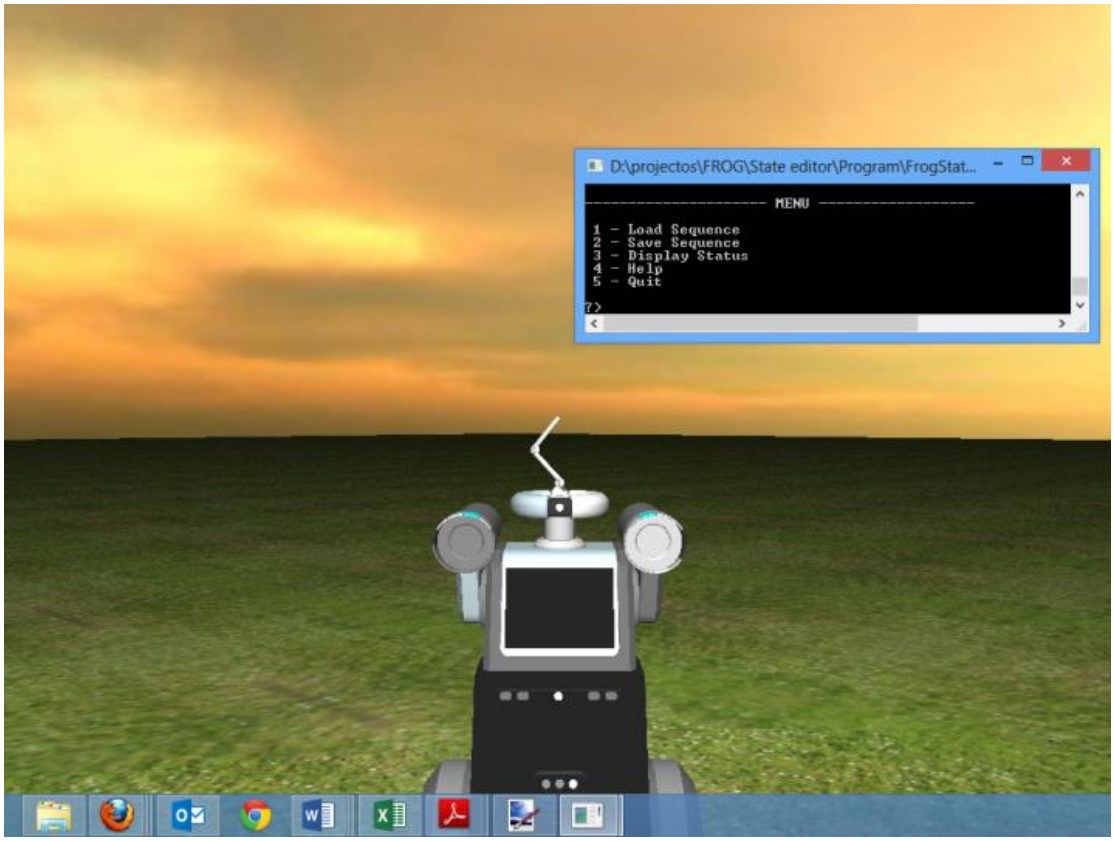


Fig. 4 – FROG state editor

Features in Frog State Editor :

- viewing the robot model by rotating around it approaching or moving away from it;
- changing state by moving the antenna and changing the screen state;
- construct a **state** sequence by adding and removing state;
- play the whole current sequence or play just one step and go back one step;
- save sequences to files, load sequences from files

Windows

Frog State Editor uses two windows:

- a graphics window, which normally starts on top and where the robot is displayed;
- a text console where some more general commands can be given.

Graphics Window

Moving the camera

In the Graphics window the camera view can rotate around the robot by keeping the right mouse button down while moving the mouse.
The view can get near or move away from the robot by turning the mouse wheel.

Changing the robot state

The three joints of the antenna can be actuated:

Base Joint – keys ‘z’ and ‘x’;
Middle Joint – keys ‘q’ and ‘a’;
Top Joint – keys ‘w’ and ‘s’.

The screen state can be changed:~

Screen off – key ‘1’;
Screen displaying video – key ‘2’;
Screen displaying smile – key ‘3’.

Editing the sequence

Initially, there’s just an empty sequence. A sequence can be loaded from file or built up by adding states. Current state of the robot can be added to current sequence by pressing ‘m’ key and last state can be removed from sequence by pressing ‘n’.

To play the current sequence press ‘p’. While sequence is playing no state can be added or removed nor can the sequence be stepped.

Initially, the current state is the first in the sequence. The transition to next state can be played or Current can be moved back to previous, as long as these positions exist.

To Step forward press ‘right arrow’ key , to go back one step press ‘left arrow’ key.

Console Window

A menu is displayed in the console Window. An option from the menu can be chosen by entering the corresponding digit and pressing Enter.

The following options are on the menu:

1 – Load sequence from file. When this option is chosen, the user is asked for a filename to load. The file must exist at the 'Sequences' folder, within the program folder.

2 – Save sequence to file. The user is asked to provide a name for the file. A file with that name must not exist. Note that in the case a sequence is loaded, changed and saves again it must be saved to different name or the original file must be renamed or removed.

3 – Display status, displays info about the current sequence editing : filename, number of states in the sequence and the current state.

4 – Help. Displays key functions.

5- Quits the application.