TEC2014-53176-R HAVideo (2015-2017)

High Availability Video Analysis for People Behaviour Understanding

D1.3 v2

Simulator documentation

Video Processing and Understanding Lab

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Supported by

AUTHORS LIST

Juan carlos SanMiguel Juancarlos.sanmiguel@uam.es

HISTORY

Version	Date	Editor	Description
0.9	16 December 2015	Juan Carlos San Miguel	Final Working Draft
1.0	17 December 2015	José M. Martínez	Editorial checking
1.9	24 June 2017	Juan Carlos San Miguel	Final Working Draft v2
2.0	25 June 2017	José M. Martínez	Editorial checking

CONTENTS:

1. Introduction

This document summarizes the work during the first year of the project for the task T1.3 “Development and maintenance of a camera network simulator.” (WP1 Video analysis framework) whose goal is to maintain a simulation environment for research on (wired and wireless) camera networks. This simulator will enable quick prototyping and testing of the prototypes developed in this project.

The following milestones have been defined during the first year of the project:

- MS1.3.1 simulator requirements
- MS1.3.2 Upgrades of the simulator
- MS1.3.3 Integration of 3D environments

1.1. Document structure

The document is structured in the following chapters:

- Chapter 2: Simulator requirements
- Chapter 3: Overview of the Wise-Mnet simulator
- Chapter 4: Upgrades of the simulator
- Chapter 5: Integration of 3D environments
- Chapter 6: Conclusions and future work

2. System requirements

This section describes the requirements of the camera network simulator and the required software packages.

2.1. Requirements description

The objective is to develop a simulator based on WiSE-Mnet for smart cameras. It will address the limitations of sensor network simulators and integrate advanced features of smart cameras such as collaborative processing, power management and the integration of 3D environments. In this milestone, we present the requirements of the overall system:

The design requirements of the complete system (see Figure 1) are as follows:

- A camera is defined as a four-component system with sensing, processing, communication and motion capabilities. Each camera acts as an independent node in the network and cooperate with other nodes to complete tasks.
- Generalized sensor data-types to define any type of physical process information (e.g. visual data such as frames, metadata, packets exchanged through network)
- Communication mechanisms: idealistic and real. Real communication should consider network protocols and existing delays. Idealistic communication is useful to test prototypes for collaborative processing.
- Smart camera resources: hardware descriptors of camera components are needed to define the capabilities of each camera. The cameras must have a “resource manager”
- Measurement of power consumption by creating models for each camera component.
- Provide templates for quick development. All standard functions (e.g. sensing frames, sending/receiving packets,…) must be in-built in the camera node so future developments can focus on the processing aspect of the camera network. Therefore a class-based design will be followed.

To ease the development of these requirements, the “overall system” has been divided into the camera network simulator Wise-Mnet (“simulador de redes de cámara”) and the 3D environment (“Motor gráfico 3D”), as seen in Figure 1.

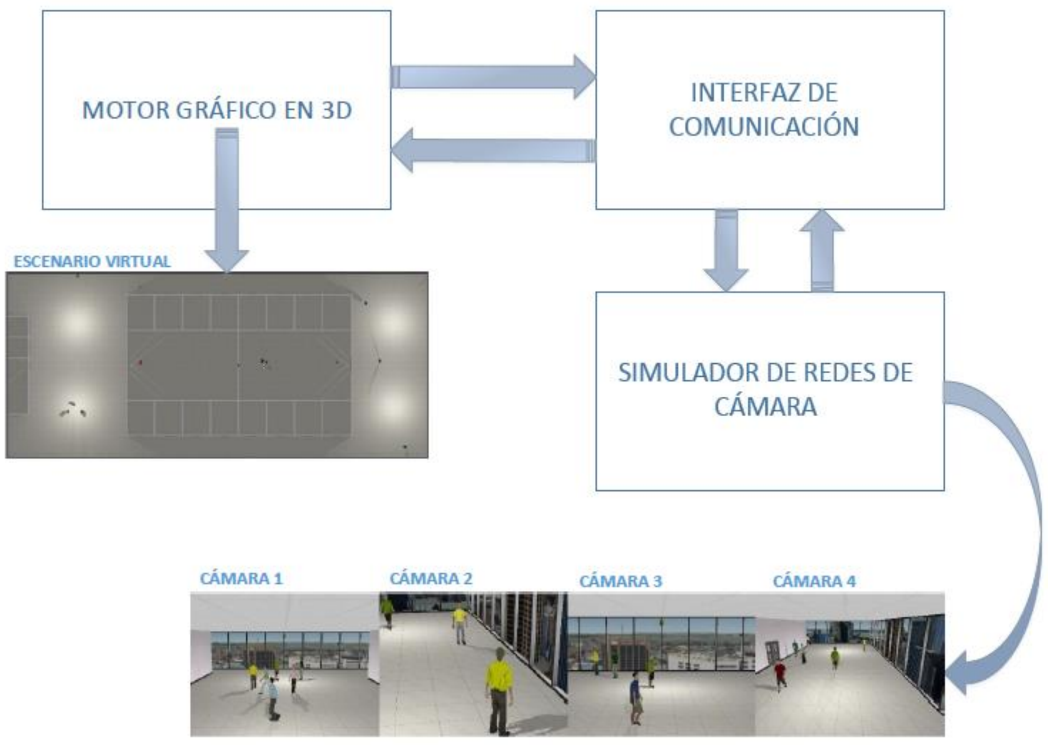


Figure 1. Overall system integrating Wise-Mnet ("simulador de redes de cámara") and 3D environments ("Motor gráfico 3D")

3. Overview of WiseMNet

For the camera network simulator, we use Wise-Mnet as the core simulator where all the improvements and required features for the project are to be implemented. Here we provide a brief description of its capabilities.

WiSE-MNet is based on Castalia/Omnet++ and enables the modeling of the communication layers, the sensing and distributed applications of Wireless multimedia sensor networks (WMSNs), i.e. networks with sensors capturing complex vectorial data, such as video and audio. The environment is designed to be flexible and extensible, and has a simple perception model that enables the simulation of distributed computer-vision algorithms at a high-level of abstraction. It is available at http://www.eecs.qmul.ac.uk/~andrea/wise-mnet.html

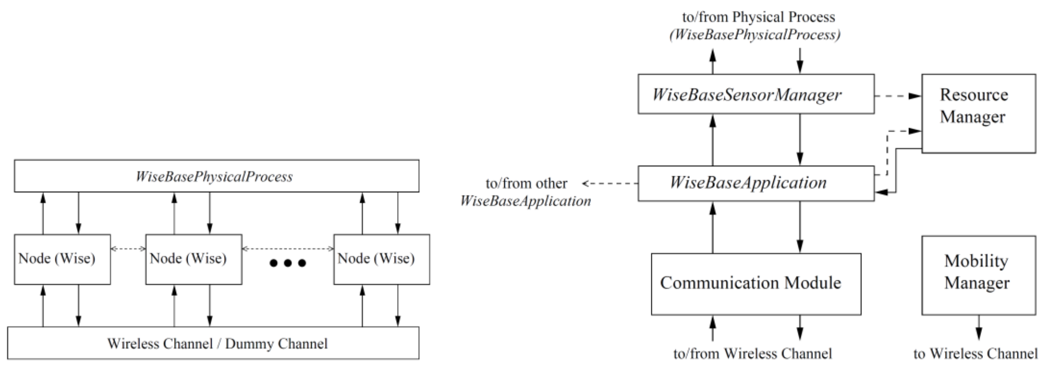
3.1. Main features

The main features provided are:

- Generic discrete-event simulation engine
- Generic modules interactions can be defined
    - behavior is coded in C++
    - interconnections/composition specified through a Network Description (NED) language
    - parameters can be set through configuration files
- Highly flexible and extensible with external libraries
- Network elements: nodes, protocols, channels – pro- vided (externally) as simulation models (INET, MiXiM, Castalia)

3.2. Node structure

The overall structure of the network and node model is depicted in Figure 2. The structure of the node model contains the same modules as the Castalia simulator (sensor, application, resource, communication and mobility). However, WiSE-MNet extends Castalia's modules to provide functionalities for multimedia networks.



(a) Network Model (b) Node (Wise) Model

Figure 2. WiSE-MNet network and node model overview

3.3. Sensing

WiseMovingTarget: This module extends the WiseBasePhysicalProcess base class/module to implement a moving target in a 2D ground plane. Targets are currently represented as (bounding) boxes and can move according to different types of motion: linear, circular, linear-circular and random. This configuration of the 2D target behavior can be established in the omnetpp.ini file (settings) of the defined simulation.

WiseVideoFile: This module extends the WiseBasePhysicalProcess base class/module to implement the capture process of a live video stream via files stored. The path to the video file must be defined in the *.ini file of the simulation

WiseCameraManager: This module extends the WiseBaseSensorManager module that implements the sensing logic of the node's camera. The module is strongly related to the type of physical process we are using. The WiseCameraManager has been designed to support different types of sensing through the WiseCameraHandler mechanism, allowing the user to easily add different camera models (e.g. projection models). We currently support only the WiseCameraDetections model, which is a simplified projection model that assumes a top-down facing camera observing targets modeled according to the WiseMovingTarget module.

3.4. Communication

In WiSE-MNet , communication is done via packets whose format is encoded in *.msg files. These packets depend on the developed application (e.g. the tracking algorithm) and contain all the variables and data to be exchanged among nodes. *Note that OMNeT++ automatically generates two files *_m.cc and *_m.h for every defined packet when the compilation of the project starts*. These two files should not be modified.

Idealistic communication mechanisms: There are two "idealistic" communication mechanisms that have been introduced: the *WiseDummyWirelessChannel* and the *DirectApplicationMessage*. The first one changes the network properties (to idealistic) seemingly from the application point of view, the second one is rather a "magic" direct information exchange channel.

3.5. Processing

The processing is performed in the application layer of the node. Therefore, each node can implement a different application layer or all the nodes can have the same processing routines. Note that this processing layer does not only correspond to tracking algorithms as other distributed algorithms can be implemented. The selection of the application layer is done in the **omnet.ini** file and its configurations depends on the defined parameters for the layer.

Types: The processing in WiSE-MNet can be performed via two mechanisms:

- On demand via **fromNetworkLayer** function. This mechanism correspond to replies to received messages from other network nodes.
- Periodically via timers using the **timerFiredCallback** function. This function corresponds to repetitive tasks that the node has to perform (e.g. grab a video frame and analyze its content every second). The timer type and alarm period have to be defined.

4. Upgrades of Wise-MNet

4.1. Description of upgraded features

The Wise-Mnet simulator has been extended according to the needs of the project resulting in a new release. The research activities have been the following:

- Development of required resources for WiSE-MNet to use an Integrated Development Environment (IDE) which enables quick prototyping.
- Improvements of original WiSE-MNet version: local processing, synchronization, live video feed and in-node video analysis.
- Improvements of the 2D sensing modes: now directional sensing can be used for Wise-Mnet defined by the depth, orientation and angle of view.
- Documentation of code and user guide (installation and development).
- Implementation of distributed tracking algorithms based on Kalman and Information Consensus Filter for single target tracking. Implementation of Information Consensus Filter algorithm for multi-target tracking using nearest neighbor approach.

4.2. Included algorithms

The list of included algorithms is as follows:

- **WiseCameraDPF** It is a WiseCameraSimplePeriodicTracker that implements a distributed particle filter algorithm. The algorithm uses a sequential aggregation mechanism, exchanging the partial posterior approximated with Gaussian Mixture Models. For more details the reader can refer to [1].
- **WiseCameraKCF** It is a WiseCameraSimplePeriodicTracker that implements a distributed Kalman filter algorithm. The algorithm uses a consensus mechanism, exchanging the final state among camera neighbors. For more details the reader can refer to [2]
- **WiseCameraICF** It is a WiseCameraSimplePeriodicTracker that implements a distributed Kalman filter algorithm via its equivalent information matrix formulation. The algorithm uses a consensus mechanism, exchanging the weighted final state among camera neighbors (information vector and matrix). For more details the reader can refer to [3]
- **WiseCameraICF**-NN It is a WiseCameraSimplePeriodicTracker that extends WiseCameraICF for multiple targets. For the association stage, the algorithm uses a nearest-neighbor approach so tracks at one time-step are linked with following one. For more details the reader can refer to [3].

4.3. Developing your own application

For creating a new application, the *WiseCameraSimplePeriodicTracker* base class is provided which extends WiseBaseApplication. This class contains basic functions to initialize resources, send/receive messages from/to network (or direct node-to-node communication) and handling of control messages. Prior to process packets, this class

discovers the communication graph for each node (i.e. neighbors nodes using network communication).
The new distributed algorithm has to extend the *WiseCameraSimplePeriodicTracker* in order to use the provided functionality. Please check the already defined classes *WiseCameraICF* or *WiseCameraKCF* for examples of applications developed based on *WiseCameraSimplePeriodicTracker.*

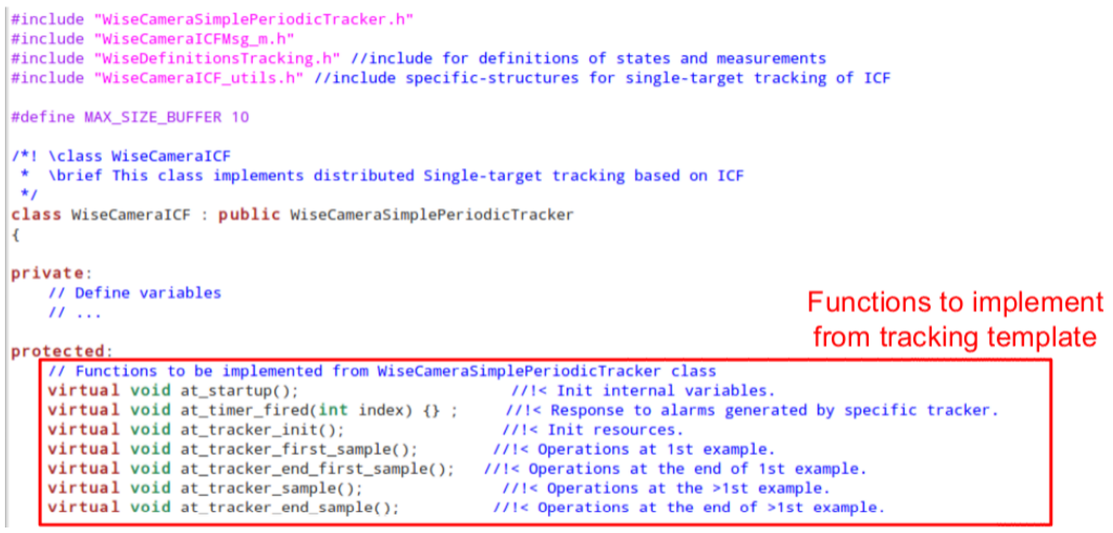
4.3.1. Required files

At least, one file of the following types is required:
Source files: WiseCameraXXXX.ned Description of your application using NED language
WiseCameraXXXX.h Include file with the header of the application
WiseCameraXXXX.cc Source file with the code of the application
WiseCameraXXXMsg.msg Packet definition to exchange among nodes
Simulation files: omnetXXX.ini Configuration of the simulation for the new application

4.3.2. Steps

The steps for developing a new application are:

1. Define the new application as a new class (extending WiseBaseApplication for generic processing or WiseCameraSimplePeriodicTracker for a distributed tracker).
2. Create the structures and classes to be used within your new application.
3. Implement the functions *startup* and *finishSpecific* for the new application requirements.
4. Implement the function handleSensorReading to handle the data provided by the SensorManager.
5. Implement the functions fromNetworkLayer and handleDirectApplicationMessage to handle received packets from, respectively the network and direct node-to-node communication.
6. Functions handleNetworkControlMessage, handleMacControlMessage and handleRadioControlMessage are optional.
7. Implement the logic of your application via the processing functions (see following Figure).



```
#include "WiseCameraSimplePeriodicTracker.h"
#include "WiseCameraICFMsg_m.h"
#include "WiseDefinitionsTracking.h" //include for definitions of states and measurements
#include "WiseCameraICF_utils.h" //include specific-structures for single-target tracking of ICF

#define MAX_SIZE_BUFFER 10

/*! \class WiseCameraICF
 *  \brief This class implements distributed Single-target tracking based on ICF
 */
class WiseCameraICF : public WiseCameraSimplePeriodicTracker
{

private:
    // Define variables
    // ...

protected:
    // Functions to be implemented from WiseCameraSimplePeriodicTracker class
    virtual void at_startup();                        //!< Init internal variables.
    virtual void at_timer_fired(int index) {} ;      //!< Response to alarms generated by specific tracker.
    virtual void at_tracker_init();                  //!< Init resources.
    virtual void at_tracker_first_sample();         //!< Operations at 1st example.
    virtual void at_tracker_end_first_sample();    //!< Operations at the end of 1st example.
    virtual void at_tracker_sample();               //!< Operations at the >1st example.
    virtual void at_tracker_end_sample();          //!< Operations at the end of >1st example.
```

Figure 3. Functions to be implement for developing a new application

Note that the development of new application layers do not need to modify the sensing (WiseMovingTarget, WiseVideoFile and WiseCameraManager) and communication (WiseDummyWirelessChannel and Wireless-Channel) modules.

5. Integration of 3D environment - OVVV

After defining the requirements of the camera network simulator in section 2 which is based on the Wise-Mnet simulator (http://www.eecs.qmul.ac.uk/~andrea/wise-mnet.html), this milestone describes the development of a camera simulator based on 3D environments.

5.1. Description of work

To create a camera simulator based on 3D environments, the following action points have been defined:

1. Search and selection of related tools to simulate and edit 3D environments
2. Creation of the development enviroment
    a. Installation of tools and libraries needed
    b. Integration of the tool to simulate cameras and to simulate 3D enviroments
3. Development of the camera functionality
    a. Create/delete/modify a camera
    b. Move a camera
    c. Transmit/receive images from cameras
    d. Extension to multi-camera enviroments
4. Development of a multi-camera controller: this sub-system will allow to provide an interface between applications and the simulator
5. Experiments
    a. Resources: Memory, CPU,…
    b. Interconnectivity: bandwidth, FPS received,… Outcomes

For the 3D environment generator, we used the simulator ObjectVideo Virtual Video tool (OVVV) which allows to load different 3D environments based on the steam engine. Moreover, the DiVA distributed video analysis framework was used to provide the interconnection functionality between cameras. The system developed is described in the following figure:

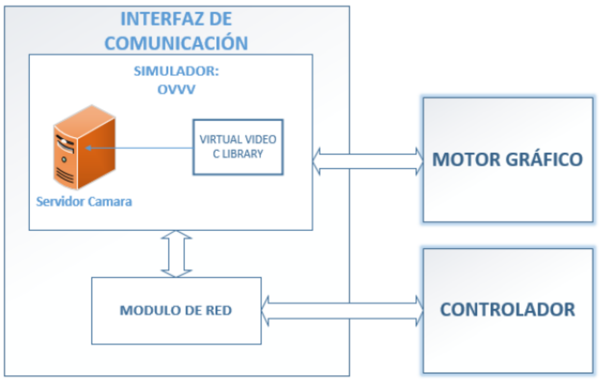


Figure 4. System developed to integrate 3D environments ("Motor gráfico 3D") within camera simulators

5.2. Outcomes

The developed system allows to control multiple cameras installed in virtual environments 3D simulated realistic scenarios. In addition, this system allows quick development and remotely control of cameras, creating a separation between the cameras management, simulator virtual environment and responsible for the control of the cameras.

As a result, this work has been performed through the final degree thesis of Luis Perez Llorente (Degree in Computer Science), defended in July 2015. Full details for the design, experiments and conclusions are available at the report of this degree thesis.

5.3. Sample results

The following figures shows an example of a moving camera:

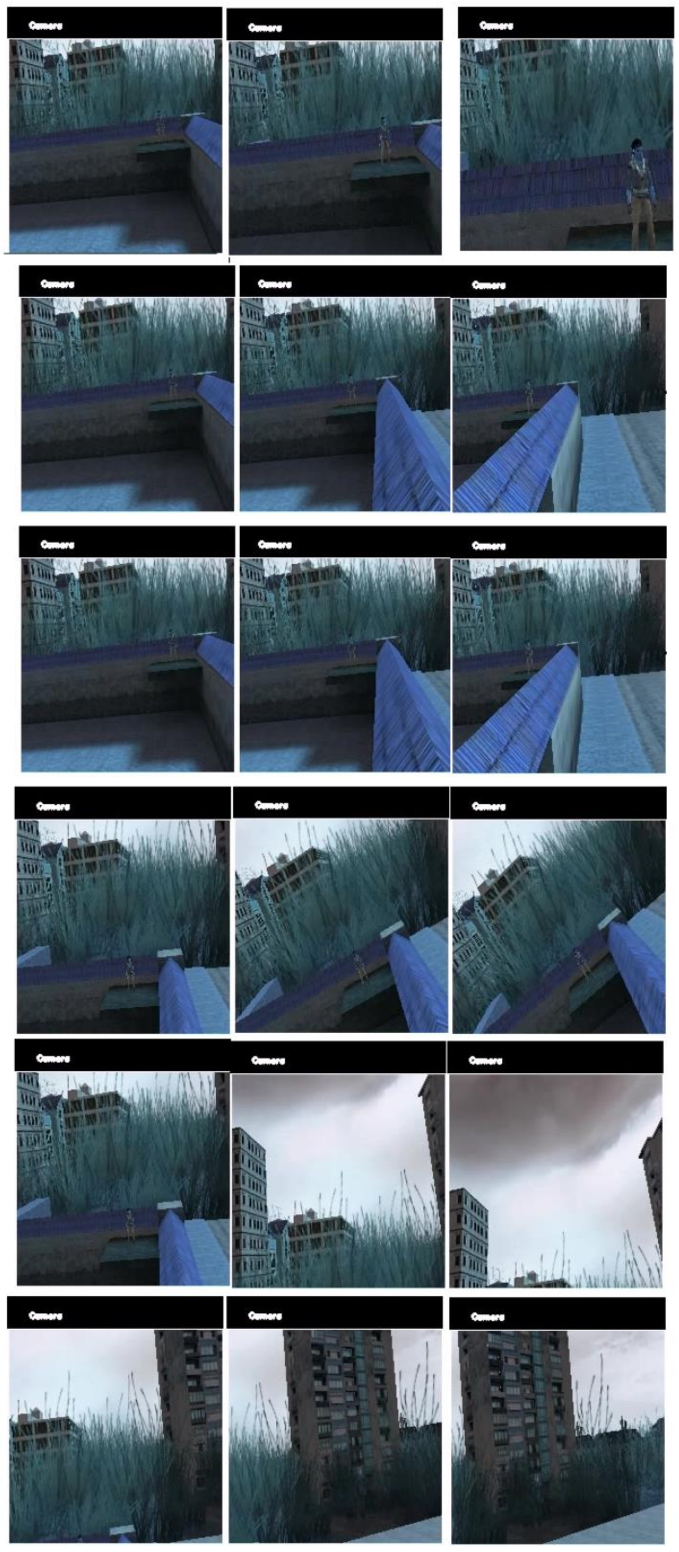


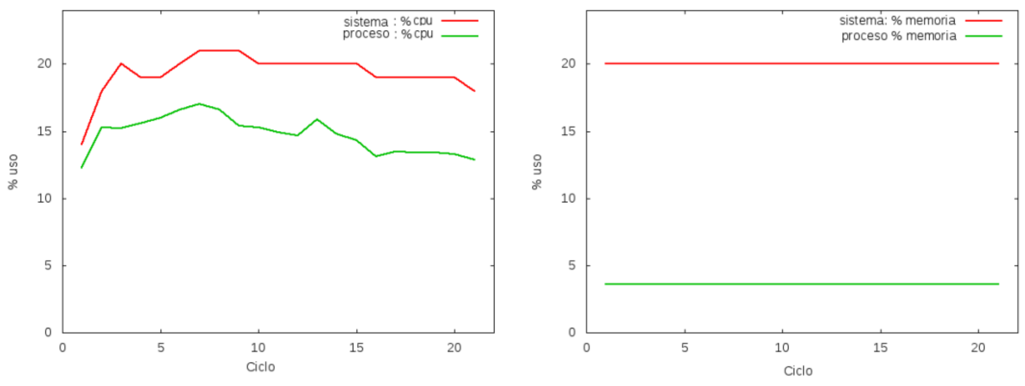
Figure 5. Sample frames from a moving camera created in the 3D environment

5.4. Experiments

A set of experiments was performed to:

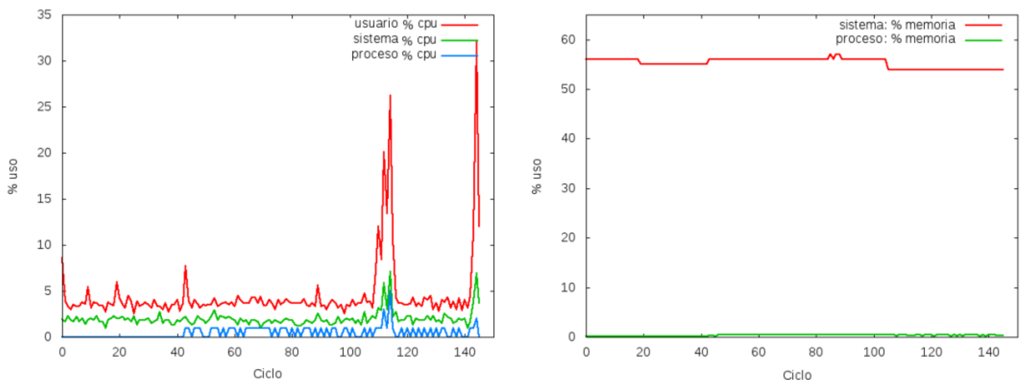
- Test the use of memory and CPU
- Test the visualization of images a different resolutions (bandwidth)
- Test the modifications of camera properties
- Test the performance of the controller application

Thus a dummy application was created to connect to the created camera simulator. An example of the obtained results is shown in the following figure:



(a) Rendimiento CPU. (b) Rendimiento memoria.

Figura 5.1: Memoria y CPU utilizada por el Controlador.



(a) Rendimiento CPU (b) Rendimiento memoria.

Figure 6. Memory and CPU utilization of the dummy program created to test the simulator

6. Integration of 3D environment – Unity3D

After completing the first development of the 3D environment for simulating multi-camera systems with virtual data, an improved version was developed based on a more recent game engine (Unity 3D) as opposed to the previous one employed (source, dated from 2007).

This work has been developed in the context of the undergraduate thesis "Sistema multi-camara distribuido basado en Unity" and the posterior contract in the HAVIDEO project as technician (both by Mario Gonzalez Jimenz). In this document we provide a summary of the main achievements which resulted in the 'Multi-Camera System Simulator' (MSS).

6.1. Description of work

The main objective of this work is to develop a simulator which provides virtual video from multiple cameras. This virtual system must be able to manage several distributed cameras in real time, configure camera's properties and possibility to transmit data (e.g. frames) to external applications. The Unity game engine3 is used as the starting point to simulate realistic 3D environments which is extended with the desired multi-camera functionality. In order to achieve the main objective, the following sub-goals are defined:

- To study of the related state of the art including related multi-camera simulators and available game engines to motivate the selection of Unity.
- To design and implement a system for controlling cameras remotely with different location configurations (e.g. position, orientation) and acquisition parameters (e.g. frame rate, resolution) to get information from the 3D environment.
- To design and implement a system for supporting the simultaneous connection of several third party applications (e.g. computer vision algorithms) to the multi-camera simulator. This system uses a client-server architecture and considers a server (integrated within the simulator), a communication protocol and an API that will be integrated and used by future applications (i.e. clients).
- To design and implement a proof-of-concept example containing a 3D scenario modeling the hall of the 'Escuela Politécnica Superior' ('A' building) and two simple application making use of the simulator using standard computer vision libraries such as OpenCV.
- To evaluate the functionality and performance of the system developed.

6.2. Main outcomes

The main outcome is a simulation tool which allows to handle multiple cameras into a virtual scenario. Additionally, broadcast messages can be generated from each camera (by sending frames through sockets) to external applications and vision algorithms. By using the Unity game engine where we find the appropriate features for the start point developing our system, we build up a complete multi-camera visual simulator. This simulator is referenced as ’Multi-Camera System Simulator’ (MSS). In Figure 7, the MSS architecture are outlined with a multi-client server design. The Client-Server architecture makes remote work possible as well as local work. With the API client library developed, clients are able to communicate and to receive information through the methods included.

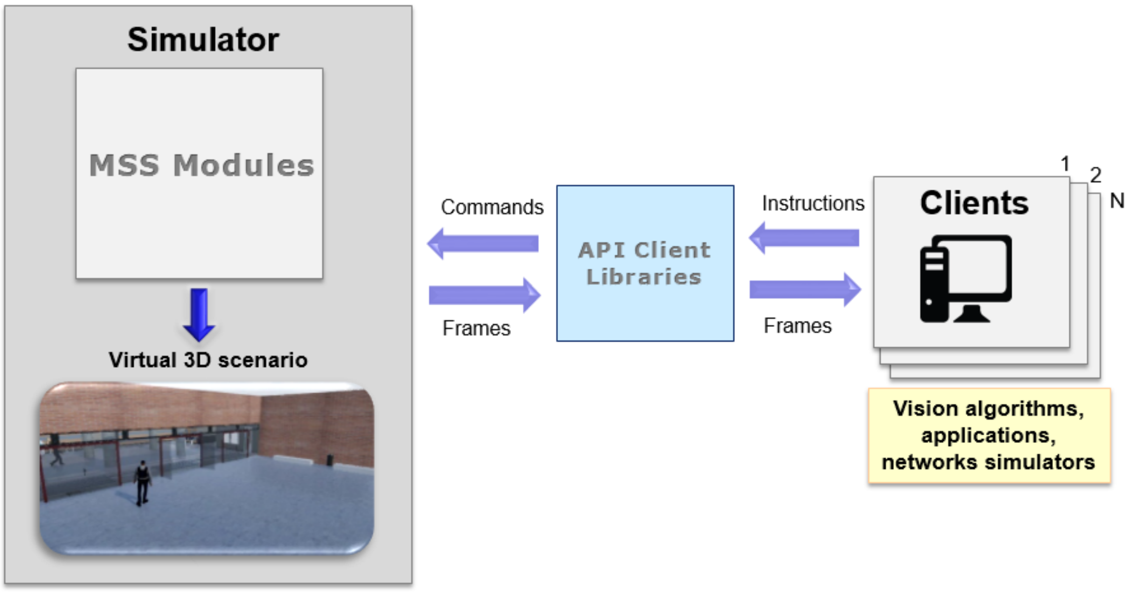


Figure 7. MSS incorporates a Client-Server architecture allowing multiple connections

The MSS modules extend the native Unity’s features with a multi-camera system, an asynchronous server TCP and more. These modules work as a plug-in, making possible to incorporate the simulator in an existing Unity project or in a new one. To incorporate it in an existing project, you only need to copy the MSS’s source folder which contains all the MSS’s classes and code in to the root Unity project folder.

The development of this simulator is complex due to the implication of a variety of technologies that work for specific purposes: image processing, multithreading, sockets, synchronization logic, GPU programming and more. To an optimal implementation, we make a modular design (Figure 8). The simulator is composed by three modules:

- **Virtual Word**: this module is responsible for all operations related to the virtual world and therefore a direct integration with the game engine (Unity). It contains the virtual cameras, the cameras controller and all logic which implies using the Unity Scripting API. By the Unity technical limitations, this entire module works sequentially, synced and following an object oriented programming. This module involves mainly GPU programming and synchronization logic.
- **Buffer**: the main purpose of this module is to store frames in main memory temporarily. In order to get the maximum performance possible, this module is implemented with multithreading. As it is explained in the next chapter, before storing a frame in main memory is necessary an image conversion. So, this module involves multithreading and image processing.

- **Server**: this module contains an asynchronous server that allows multiple client connections. It receives commands from clients, processes them and gives a response. In order to attend the clients immediately, this module also is implemented with multithreading. In this way, with independent threads, the performance of the previous modules does not affect to server and the connection logic. In this modules is involved sockets, multithreading and a *ThreadPool* which is explained in the next chapter.

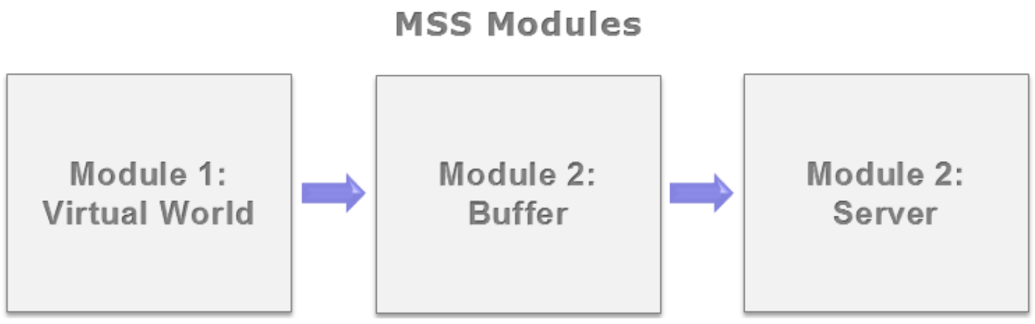
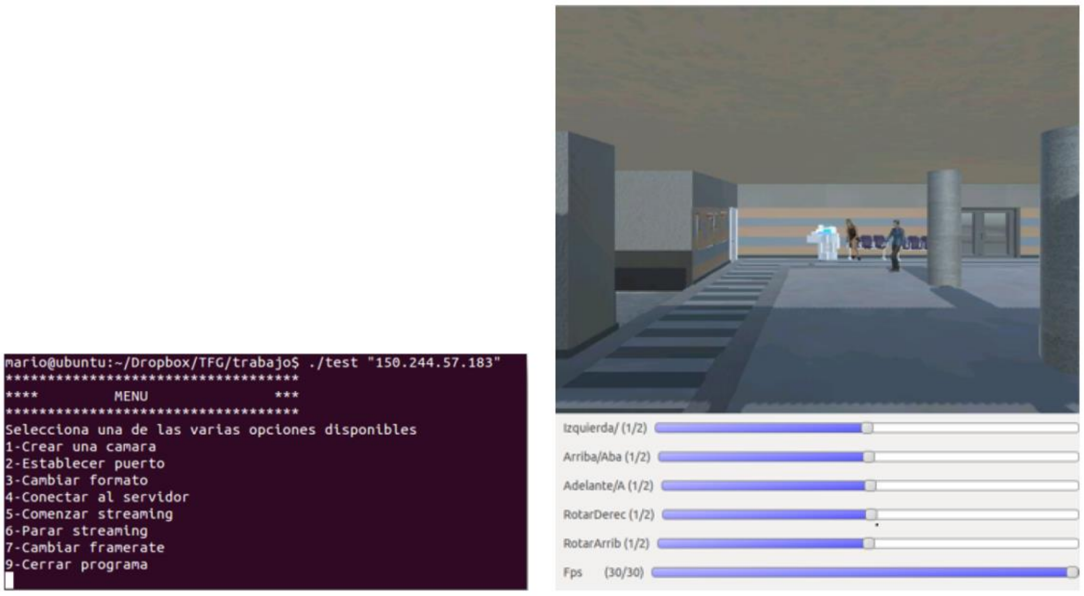


Figure 8. MSS Modules logic view.

6.3. Sample results

In order to test the functionality of the simulator, a dummy console application for Linux is built using the API provided. This application connects to the simulator trough an IP passed it on argument, and allows to create a camera, to generate a broadcast and to handle it visually trough the User Interface library. Once the application is started, a minimalist menu appears on console (Figure 9) where you can find options to establish connection to the MSS simulator, create a single camera, change its properties and delete the camera from the simulator.

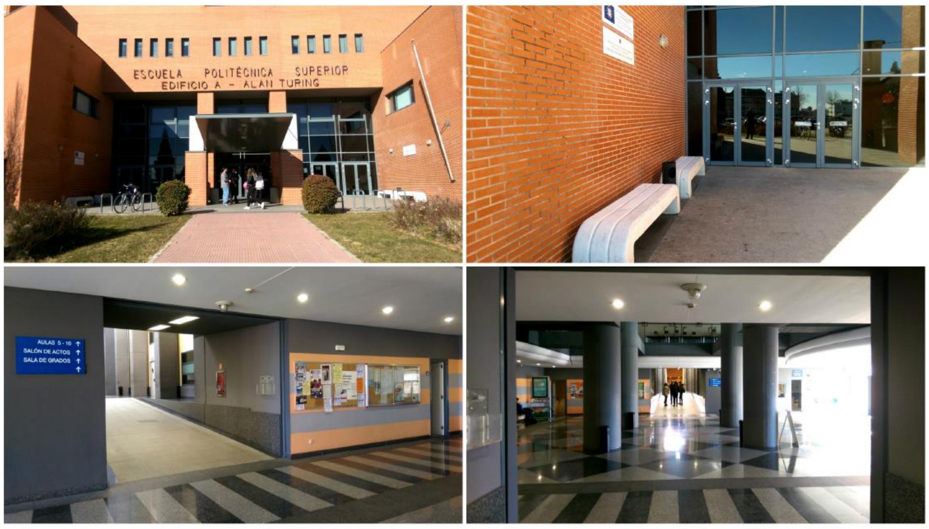


```
mario@ubuntu:~/Dropbox/TFG/trabajo$ ./test "150.244.57.183"
**********************************
****          MENU            ***
**********************************
Selecciona una de las varias opciones disponibles
1-Crear una camara
2-Establecer puerto
3-Cambiar formato
4-Conectar al servidor
5-Comenzar streaming
6-Parar streaming
7-Cambiar framerate
9-Cerrar programa
```

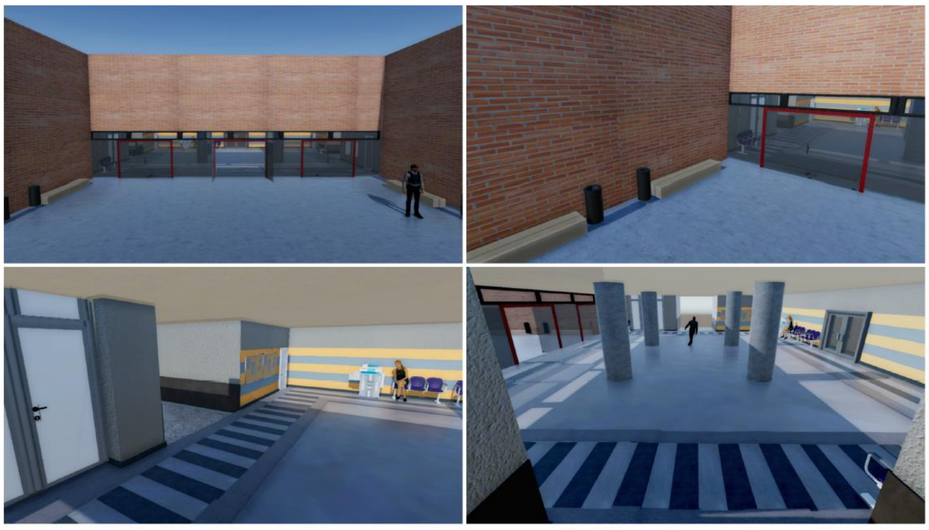
(a) Menu using the console. (b) Dummy user interface.

Figure 9. Screenshot of the dummy application.

In all experiments, we use the modeled scenario for the EPS building. Examples for the pedestrians included in the scenario and comparisons between real pictures and the modeled scenario are provided in Figure 10.



(a)



(b)

Figure 10. Comparative between (a) pictures of the 'Alan Turing' building and (b) virtual scenario modeled.

6.4. Experiments

As experiments, we have tested the following:

- Check the overall system and functionality developed
- Evaluate the performance of the frame generation process
- Evaluate the performance of the frame coding process (JPEG or PNG)
- Test the computational resources employed.
- Monitor the network usage in different situations to test a distributed computing environment.

The following figures provide examples of the experiments:

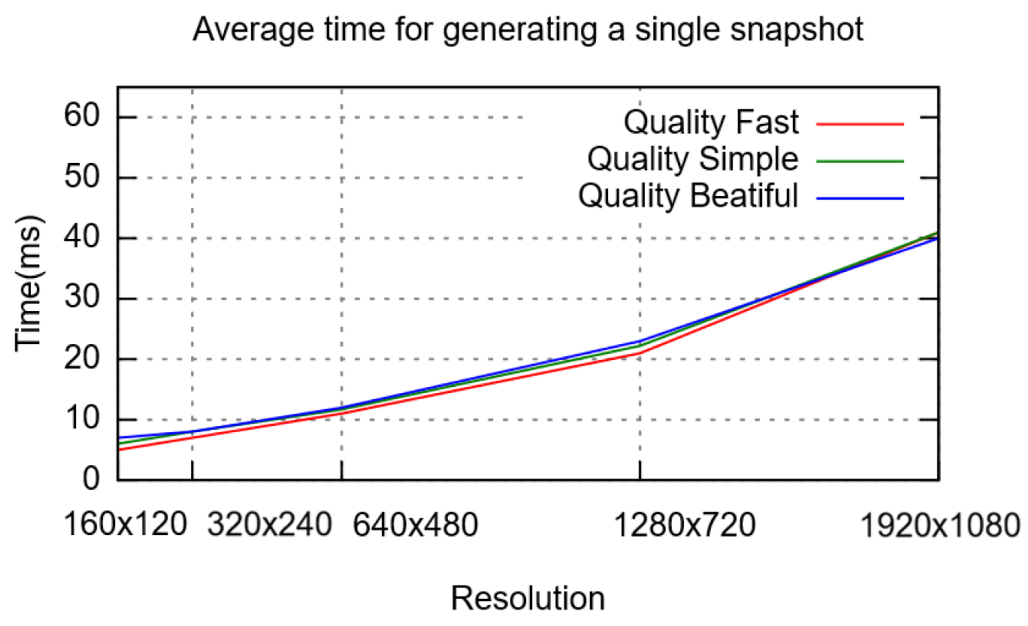


Figure 11. Time to generate one RAW frame for different resolutions and graphic qualities.

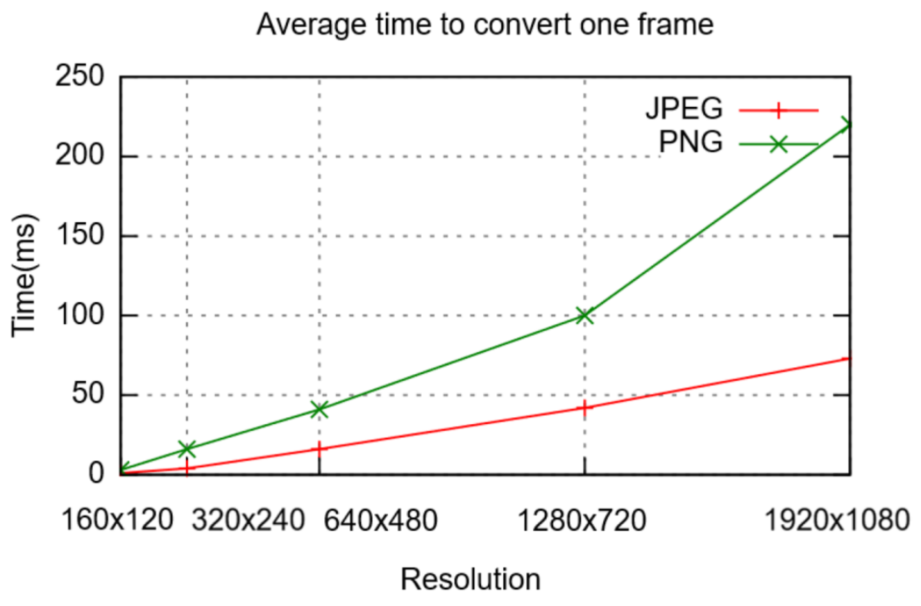


Figure 12. Average conversion time for different resolutions and encoders (JPEG and PNG).

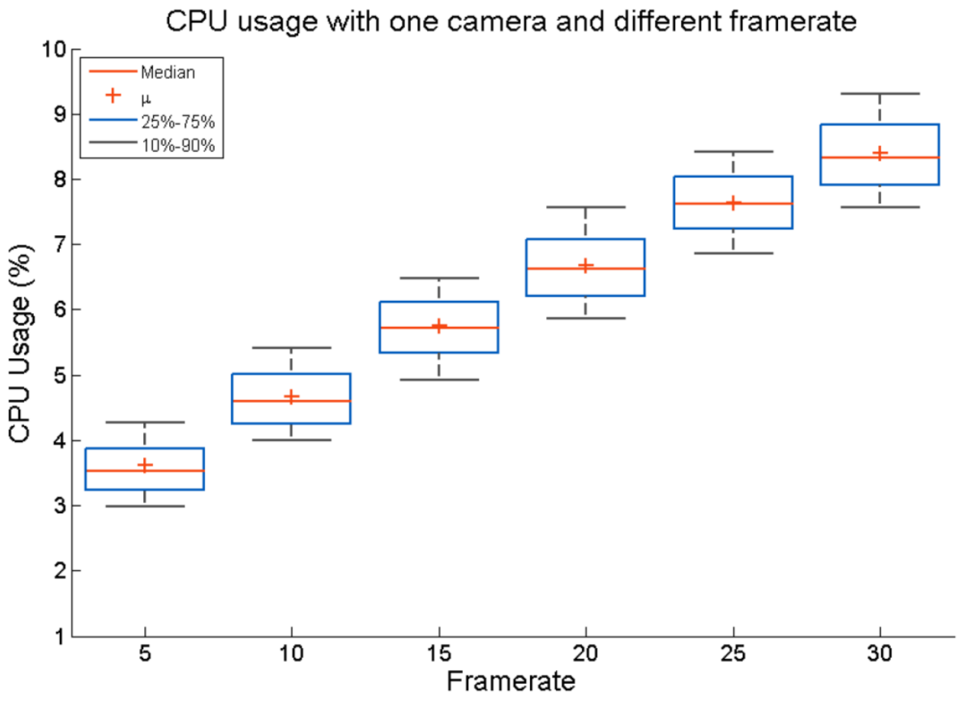


Figure 13. CPU usage with one camera and different framerates.

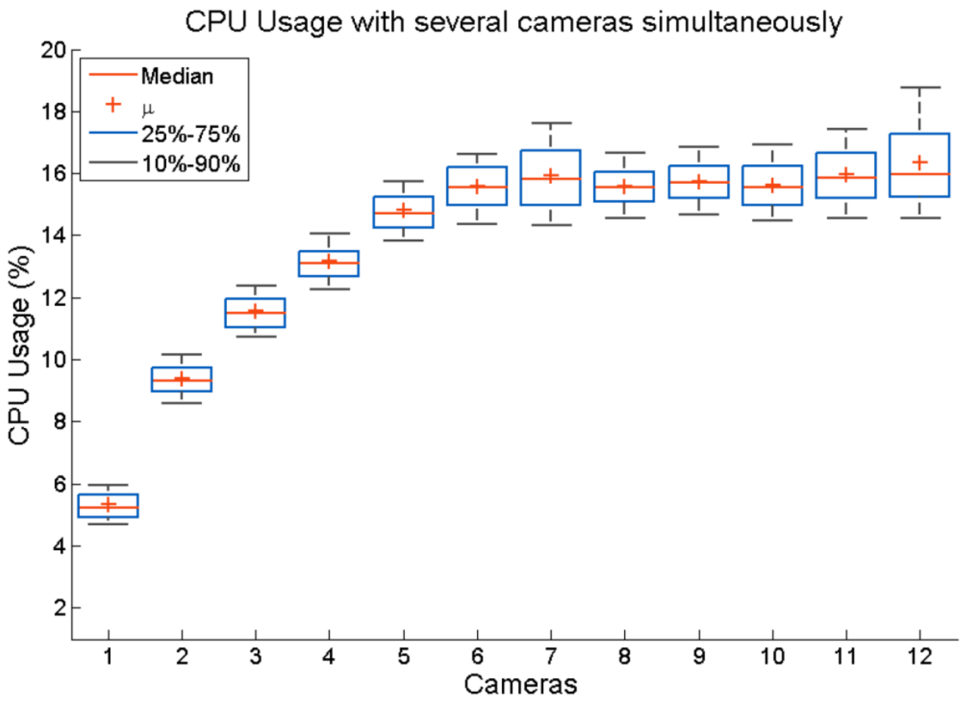


Figure 14. CPU usage with several cameras simultaneously configured at 640x480 @ 25 fps.

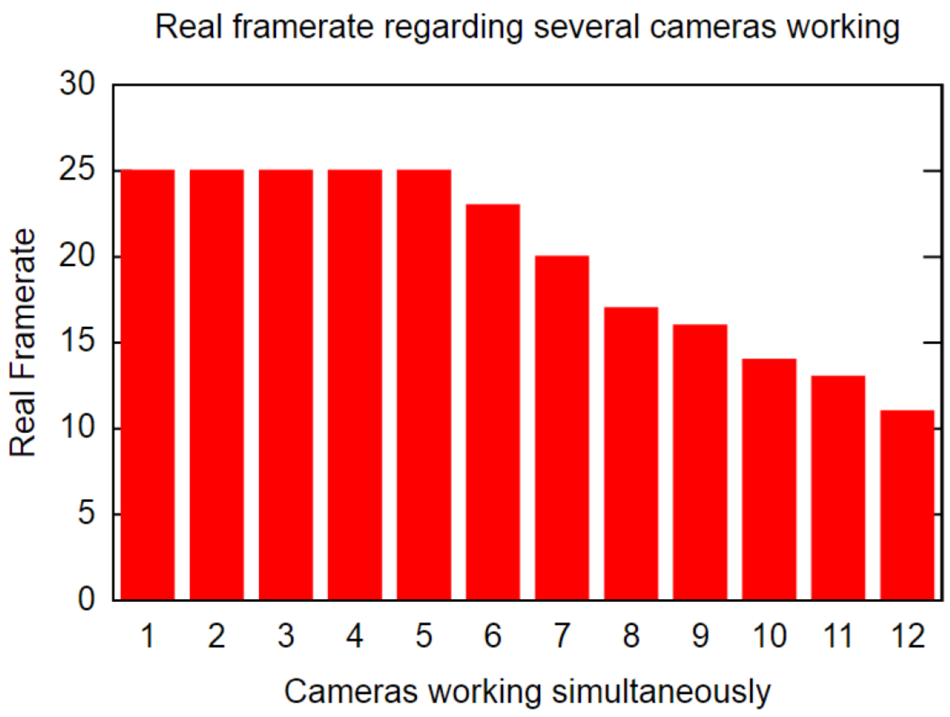


Figure 15. Real Framerate regarding several cameras working simultaneously. All cameras have the same configuration: 640x480 @ 25. We expect 25 fps until it reach the maximum of frames that our system is able to generate in one second.

7. Conclusions

During the first and second years, an existing simulator for camera networks (Wise-Mnet) has been upgraded with interesting features to achieve the project goals. Moreover, two 3D environments have been developed and integrated within a camera simulator. However, a complete system containing both the 3D environment and Wise-Mnet has not been developed yet. Moreover, the 3D environment developed using the OVVV tool provides synthetic data which is far from representing realistic visual data and therefore, Unity3D has been employed to address this limitation.

Therefore, the future lines of work would focus on documenting the developed software and the creation of repositories for its distribution.

8. References

[1] J. Farrell A. Kamal and A. Roy-Chowdhury. Information weighted consensus filters and their application in distributed camera networks. IEEE Transactions on Automatic Control, 58(12):3112–3125, Dec 2013.
[2] A. Kamal, J. Farrell, and A. Roy-chowdhury. Information consensus for distributed multi-target tracking. In Proc. of the IEEE Int. Conf. on Computer Vision and Pattern Recognition, pages 2403–2410, Portland (USA), 25-27 Jun. 2013.
[3] C. Nastasi and A. Cavallaro. Distributed target tracking under realistic network conditions. In Proc. of Sensor Signal Processing for Defence (SSPD), pages 1–5, London (UK), 28-29 Sept. 2011.
[4] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In Proc. of the IEEE Int. Conf. on Decision and Control, pages 5492–5498, San Diego (USA), 12-15 Dec. 2007.
[5] Geoffrey R. Taylor, Andrew J.Chosak, and Paul C. Brewer. Ovvv: Using virtual worls to design and evaluate surveillanve systems. Computer Vision and Pattern Recognition and 2007. CVPR '07. IEEE Conference on, pages 1_8, Junio 2007.
[6] Juan C. SanMiguel, Jesús Bescós, José M. Martínez, and Álvaro García. DiVA: A distributed video analysis framework applied to video-surveillance systems. WIAMIS '08. Ninth International Workshop on Image Analysis for Multimedia Interactive Services. IEEE,, pages 207_210, Mayo 2008.
[7] C. Nastasi and A. Cavallaro. Wise-mnet: an experimental environment for wireless multimedia sensor networks. Proc. of Sensor Signal Processing for Defence (SSPD), Septiembre 2011