Demo: VSNsim - A Simulator for Control and Coordination in Visual Sensor Networks

Melanie Schranz and Bernhard Rinner
Institute of Networked and Embedded Systems
Alpen-Adria Universität Klagenfurt, Austria
{firstname}.{lastname}@aau.at

ABSTRACT

The analysis and evaluation of concepts in the research fields of visual sensor networks (VSNs) suffer from the low number of simulation possibilities. In this paper we present a simulator, the VSNsim, dedicated for evaluating control and coordination strategies in VSNs. It is built with the game engine Unity3D and has a very user friendly handling. The algorithms locally running on the sensor nodes of the VSN can be implemented in C#, JavaScript or Boo. Due to graphical user interface and the 3D implementation, our simulator is a tool that can be intuitively applied and extended to a researcher's need.

Categories and Subject Descriptors

Computer systems organization [**Embedded and cyber-physical systems**]: Sensor Networks; Computer systems organization [**Architectures**]: Other architectures—*Self-organizing autonomic computing*; Software and its engineering [**Software organization and properties**]: Contextual software domains—*Virtual worlds software*

Keywords

Visual sensor networks, Virtual worlds, Distributed systems

1. INTRODUCTION

Visual sensor networks (VSNs) are constituted of spatially distributed smart cameras. The cameras work autonomously, retrieve the observations from the environment and process them locally. By exchanging the retrieved information among the other cameras in the network or a specific neighborhood, they are able to further process aggregated observations for an optimized output.

In general, VSNs address different research fields including image processing, networking as well as control and coordination. For evaluating image processing techniques, researchers usually revert to pre-recorded datasets from physical camera networks or virtual rendered video streams. The latter is provided by a simulation environment proposed by Qureshi and Terzopoulos in [1] and following works in [2] and [3]. They present a 3D synthetic environment to simulate real-life scenes involving pedestrian tracking with two PTZ-cameras and one static camera.

For evaluating networking aspects, several well-known network simulators like the OMNeT++[1], ns-2[2] or GNS3[3] are widely used. Their main focus is to simulate issues on a very deep network protocol level (routing, TCP, IP, multicast protocols) for wired or wireless communication.

For evaluating control and coordination aspects no convenient simulation framework is available. Thus, researchers mostly implement the control and coordination algorithms in Matlab or other programming languages, as in Java by Esterle et al. in [4].

We established a new simulator for the evaluation of control and coordination strategies. The VSNsim provides a user friendly 3D environment for evaluations before implementing the algorithms on a real camera network. First of all, the reasons for developing the VSNsim was to create a tool capable of analyzing control and coordination strategies by abstracting the physical layer, networking protocols and image processing tasks of the nodes in a VSN. The current version of the simulator models multiple cameras with parallel execution behavior. Nevertheless, the simulator is easy in installation, use and extension of the simulation environment. Moreover, it provides a pseudo-realistic environment with multiple GUI elements. Such an evaluation environment motivates researchers to encourage their creativity in designing various scenarios and thus, analyzing different behavior of their algorithms.

The remainder of this paper is organized as follows: Section 2 describes the implementation details of VSNsim. In Section 3 we briefly sketch the application we evaluated with VSNsim. Further, Section 4 explains the way in which VSNsim is presented at the conference and finally, Section 5 concludes the paper and gives an outlook on further improvements on the presented simulator.

2. THE VSNSIM

A screenshot of the VSNsim is shown in Fig. 1, which includes three selected camera views and some control buttons. The current simulation environment provides 26 smart cameras placed in 14 emulated office rooms. The cameras have overlapping field of views (FOVs) and the object identi-

[1] http://www.omnetpp.org/
[2] http://www.isi.edu/nsnam/ns/
[3] http://www.gns3.net/

ICDSC '14, November 04 - 07 2014, Venezia Mestre, Italy

http://dx.doi.org/10.1145/2659021.2669475

fication as well as all other processing tasks concerning control and coordination run locally on the cameras. The corresponding algorithms can be easily written in C#, JavaScript or Boo. The cameras in the simulator retrieve the object's information all two seconds. The applied processing tasks are executed with ease within this time.

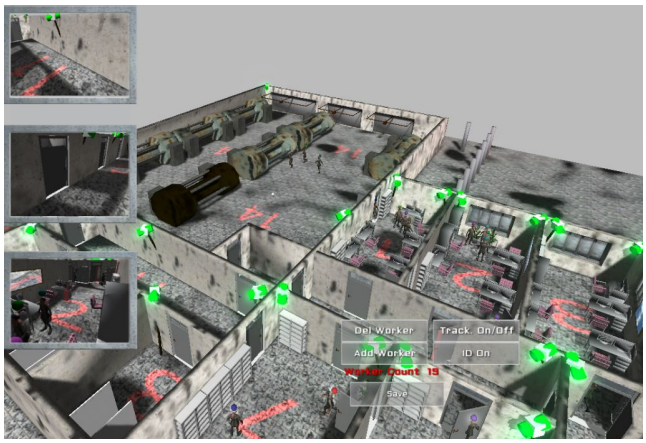


Figure 1: A screenshot of the VSNsim.

The main VSNsim building blocks, e.g., for office rooms, moving objects or smart cameras, are the so-called game objects[4]. A game object refers to an entity and consists of several components describing the object's attributes and behavior. These components are structuring elements representing, e.g., an object's tag, transform, collider, renderer or behavior through scripts. These game objects are built and operate independently from each other.

2.1 Environmental Structure

The office rooms in the simulator (Fig. 2a) are built up with the previously described game objects as low-poly models. They define walls equipped with colliders for collision detection. The colliders are built as box objects to achieve a minimized polygon count. This avoids collisions of the moving objects with the walls. They do not have a significance during run time. Passing of moving objects through walls is enabled by deploying animated doors. The doors are also triggered by additional colliders. The office equipment corresponds to typical interior decoration in office rooms with desks, chairs, shelves, plants and toilets (Fig. 2b). This furnishing also uses colliders to be visible to the moving objects. The textures used for the offices have a standard size of 512x512 pixels. All shadows in the scene are pre-rendered to save computation time.

2.2 Object Movement

Fig. 2c represents an object which is realized as a game object with selected texture. The object's movement is based on two strategies. i) It follows pre-defined waypoints as indicated with yellow circles in Fig. 2c. The object's trajectory together with its velocity are defined offline. ii) The objects move randomly and are attracted by some "object's needs" which help to guide them from room to room. Their trajectory is determined during run time. Collisions between moving objects are avoided with collision queries and recalculations of the path. The only limitation for the object movement is given with the defined navmesh in the simulation environment—an abstract data structure generated offline through colliders to aid the moving objects in path-finding while reducing computational effort.

[4]`http://docs.unity3d.com/ScriptReference/GameObject.html`

2.3 Smart Cameras

A smart camera together with its view is shown in Fig. 2d. It is realized as a so-called prefab object. A prefab is a clone and placed multiple times—in our case 26 times—to the scene having the same attributes and behavior. It is constituted of an game object combining the component "camera" to it. This component is available via the Unity3D game engine. Within the engine it is a device to capture and display the world to the player. Each prefab is able to perform computation locally and independent of the other prefabs. Through a common interface—represented by a component attached to the prefabs—they exchange their locally retrieved and processed data among themselves.

The object identification is realized by the raycast method provided by Unity3D. The raycast method has a similar operation as radar. If a camera has an object in its FOV, it gets the object's coordinates together with its ID transmitted through an internal message.

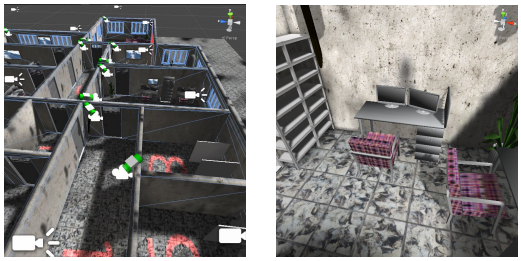
Further processing tasks related to cooperative control, etc. are added as further components directly to the smart camera game object. As already mentioned, they can be written in C#, JavaScript or Boo, whereby the game object is able to handle scripts written in multiple programming languages. Moreover, at each processing step, the smart cameras are storing their camera ID, a time stamp and selectable information into a file with spreadsheet format. Already during the simulation time it is possible to request the stored data for further analyses.

2.4 Graphical User Interface

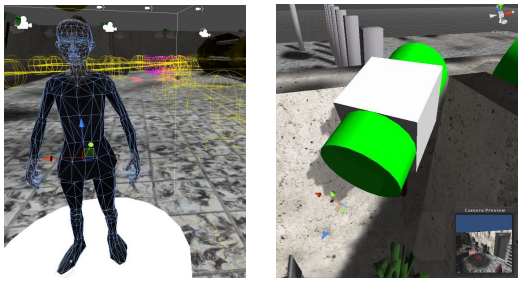
The simulator starts with a 2D GUI allowing the user to initialize several variables of the simulator. In our application each camera stores the coordinates and the ID of an object, if it is in its FOV, together with time and a simulated resource parameter. Thus, first of all we can select a location for data storage locally on the PC. Another setting can be done on the standard deviation of the objects ground truth serving as input to the individual cameras. Further, we can choose between two indoor and one outdoor scene. After starting the simulator, a window as in Fig. 1 appears. We have three additional windows, where camera views can be shown as necessary. These views provide further understanding on the scene and are not considered as any input. By selecting a camera in the simulator during run time, its view will arrange itself into the list of given windows. Further, there are several buttons visible to add or delete objects, switch tracking of objects on/off and save the observations.

3. SIMULATION EXAMPLE: RESOURCE-AWARE STATE ESTIMATION

As example, we demonstrate the VSNsim functionality on a resource-aware cluster-based protocol for smart cameras in VSNs, as proposed in [5]. The dynamic clustering protocol considers the available resources and a visibility parameter



(a) (b)



(c) (d)

Figure 2: The individual parts of the simulator: 2a) the setup of the office rooms, 2b) the equipment within the office rooms, 2c) the structure of a moving object together with the pre-defined waypoints and 2d) a smart camera together with its preview.

in order to distribute the state estimation process accordingly. It is designed in a lightweight way leading to a minimal number of messages to be exchanged and thus, spare a node's resources.

We used the simulator to compare it to the fully distributed approach of [6]. In our evaluation we consider a single room of the VSNsim equipped with nine cameras. The object is following pre-defined waypoints—called the ground truth. Within this simulation each camera gets its individual observations from the object by a random modification of the ground truth. In the evaluation we set the modification value randomly to a standard deviation of three length units. Thus, we showed in [5] that the state accuracy in the cluster-based approach suffers due to the reduced number of exchanged information. Nevertheless, our approach outperforms the distributed one in terms of a reduced communication and storage consumption.

4. ICDSC DEMO

During the demonstration we will show the main possibilities for the usage of the simulator. We implemented three different scenarios: i) an indoor scene, where the objects follow a specific need to move from office to office, ii) an indoor scene, where the objects follow pre-defined waypoints and iii) an outdoor scene also based on waypoint movement. Further on, we will show how to extend the simulator using your own coordination and control algorithms. Finally, we will demonstrate how to apply simple image processing algorithms on the simulated camera views.

A short video of the current version of the simulator is presented on `www.youtube.com/watch?v=r7E1NRSccjg`.

5. CONCLUSION AND FUTURE WORK

The presented demonstration shows the possibility of evaluating a smart camera network in a very efficient and user friendly way. Within a 3D game engine we can generate a simulation environment for coordination and cooperation control algorithms in VSNs.

Open issues are additional user interface actions during runtime like defining the number of active cameras, allowing for application appropriate scenes, etc. Additional enhancements are related to simulating PTZ and mobile cameras as well as the applicability of vision-based algorithms.

6. REFERENCES

[1] F. Z. Qureshi and D. Terzopoulos, "Smart Camera Networks in Virtual Reality," in *First ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, 2007, pp. 87–94.

[2] W. Starzyk, A. Domurad, and F. Z. Qureshi, "A Virtual Vision Simulator for Camera Networks Research," in *Ninth Conference on Computer and Robot Vision (CRV)*, 2012, pp. 306–313.

[3] W. Starzyk and F. Z. Qureshi, "Software laboratory for camera networks research," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 3, no. 2, pp. 284–293, 2013.

[4] L. Esterle, P. Lewis, M. Bogdanski, B. Rinner, and X. Yao, "A socio-economic approach to online vision graph generation and handover in distributed smart camera networks," in *Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Aug. 2011, pp. 1 –6.

[5] M. Schranz and B. Rinner, "Resource-Aware State Estimation in Visual Sensor Networks with Dynamic Clustering," 2014, under Review.

[6] B. Song, C. Ding, A. Kamal, J. Farrell, and A. Roy-Chowdhury, "Distributed camera networks," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 20 –31, 2011.