CamSim: A Distributed Smart Camera Network Simulator

Lukas Esterle*, Peter R. Lewis†, Horatio Caine†, Xin Yao† and Bernhard Rinner*
*Institute of Networked and Embedded Systems, Alpen-Adria Universität Klagenfurt and Lakeside Labs, Austria
†CERCIA, School of Computer Science, University of Birmingham, United Kingdom
{lukas.esterle|bernhard.rinner}@aau.at, {p.r.lewis|hxc931|x.yao}@cs.bham.ac.uk

***Abstract*—Smart cameras allow pre-processing of video data on the camera instead of sending it to a remote server for further analysis. Having a network of smart cameras allows various vision tasks to be processed in a distributed fashion. While cameras may have different tasks, we concentrate on distributed tracking in smart camera networks. This application introduces various highly interesting problems. Firstly, how can conflicting goals be satisfied such as cameras in the network try to track objects while also trying to keep communication overhead low? Secondly, how can cameras in the network self-adapt in response to the behaviour of objects and changes in scenarios, to ensure continued efficient performance? Thirdly, how can cameras organise themselves to improve the overall network's performance and efficiency? This paper presents a simulation environment, called *CamSim*, allowing distributed self-adaptation and self-organisation algorithms to be tested, without setting up a physical smart camera network. The simulation tool is written in Java and hence allows high portability between different operating systems. Relaxing various problems of computer vision and network communication enables a focus on implementing and testing new self-adaptation and self-organisation algorithms for cameras to use.**

***Keywords*-distributed smart cameras; simulation; software; self-organisation; self-adaptation; computer vision**

I. Introduction

Smart cameras allow the pre-processing of video data on the camera instead of sending it to a remote server for further analysis. This allows not only the execution of distributed algorithms on the cameras, but also enables each camera to build up its own awareness of itself and its environment. Setting up such a network of smart cameras in the real world has various difficulties. First of all, deploying the network itself can be challenging. This is especially true if a larger network of cameras is being investigated, as cost can be prohibitive. Furthermore, legal issues vary from country to country, and setting up cameras only in a laboratory environment would limit the number of used cameras drastically. Finally, both cameras and computer vision algorithms are prone to fail during runtime and the study of self-adaptation and self-organisation techniques through repeatable experiments in a real environment can prove quite difficult.

Our presented simulation tool, *CamSim*[1] has been used extensively in our previous research [1]–[4]. This simulation environment focuses on infusing virtual smart cameras with distributed algorithms. As such, it can be used to test and compare self-organising camera control techniques. The key benefits of *CamSim* are:

1) Ease of generating test scenarios, with cameras and objects limited only by computer memory.
2) Camera behaviour, using an economic and pheromone inspired approach [1] is implemented, as well as several communication strategies.
3) Several bandit solvers are implemented to provide meta-management at the camera level, selecting between communication strategies dynamically at runtime [2].
4) All aspects of camera behaviour, including bandit solvers, communication strategies and pheromone learning can be replaced using reflection mechanisms.

To support easy implementation, testing and comparison of distributed algorithms for self-adaptation and self-organisation of the network, we made assumptions to relax the real operating environment of multi-camera applications. Smart camera networks are usually connected using either Ethernet or Wi-Fi. These communication channels can be highly reliable and therefore in our simulation we assume perfect communication channel without any communication loss. The controlled environment allows the analysis of deployed algorithms very easily. Due to the simulator's use of discrete time steps, problems and anomalies can be identified, and debugging is easier than in real camera networks. Also, the exact position of all objects and the states of every camera can be extracted at any time step for further analysis. The uncertainties of computer vision algorithms have been removed from the simulation environment. This allows the user to focus on the development of algorithms to self-adapt cameras to a certain scenario and to self-organise the entire network of smart cameras.

The rest of this paper describes the basic features of *CamSim* in more detail. The next section will discuss the simulation environment in general and then focuses on the abilities of the cameras. Section II-B will describe the behaviour of objects in the environment. Finally, we will give an overview on how to use *CamSim* in Section III.

II. CamSim

This section describes the basic features of *CamSim*. More details are available in the documentation of the simulation

[1]CamSim is available at `https://github.com/EPiCS/CamSim`

environment.

Since *CamSim* is implemented in Java, it is highly flexible and portable. It is able to simulate a large number of cameras and objects limited only by available computer memory. *CamSim* can be run with or without a graphical user interface, to allow both user interaction and visual inspection, and batch running of experiments, for example on a cluster. Figure 1 shows a screen shot running a simple scenario with five cameras and a single object. For each scenario, the simulation environment has a predefined size, depicted as a thin blue line. All objects, illustrated as black dots, and all cameras, shown as green dots, have to be placed within this simulation environment. The labels for cameras and objects can be turned on or off as desired.

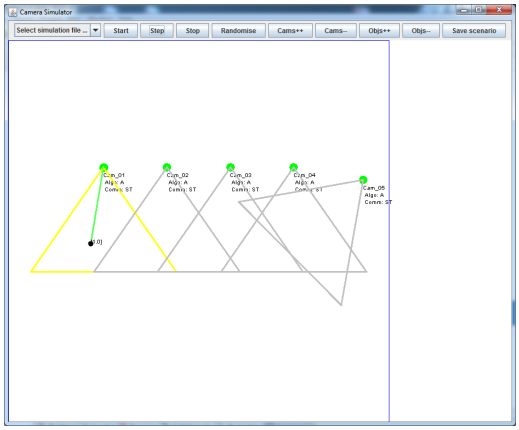

Figure 1. *CamSim* screen shot, showing 5 cameras, their fields of view, and one object, being tracked by the left-most camera.

A. Cameras

Each camera and acts as a completely autonomous agent without central control. Furthermore, each camera has its own field of view (FOV) and a unique name. This FOV is a circular segment, though illustrated as a triangle. This triangle turns from grey to yellow as soon as an object is within the respective camera's FOV. The FOV of a camera can have an arbitrary size and viewing angle. Cameras can communicate with other cameras in the network using message passing. Since *CamSim* simulates smart cameras, each camera in the network can be provided with a specific (possibly different) behaviour. For our distributed tracking application as presented in [1] we implemented six different communication strategies. Cameras can also fail for a limited time or the remainder of the simulation. When a camera fails, it can either recover previously learnt knowledge or start from scratch to build up new knowledge again.

B. Objects

Objects to be tracked are depicted as black dots. Every object has unique features which are used to identify and distinguish the object among the others. An object can have either a predefined path to follow, or an initial direction, moving in a straight line until it reaches the boundary of the simulation environment. To keep the number of objects constant, objects do not leave the simulation environment but bounce back in a random direction and continue in a straight path. A green line between the object and the camera indicates the object is currently processed by the respective camera.

III. Usage

The simulator can be used with or without parameters. Parameters can set various values for the simulations. The most important one is setting a scenario file. Scenarios are specified using XML, using tags such as `<simulation>`, `<cameras>` and `<objects>`. The parameter `-t` sets the number of time steps in a simulation without graphical user interface. `-o` allows the specification of an output filename while `-a` and `-c` predefine a specific algorithm and communication schedule respectively. `--no-gui` starts the simulation without graphical interface. Finally, `-h` prints out a text explaining the usage of the simulation environment again including the possible parameters. The use of bandit solvers can be defined within the scenarios files.

Even though *CamSim* was developed with smart camera networks in mind, the simulation environment can easily be extended to various distributed networks with resource constraints, since many of the techniques implemented are not specific to smart camera networks. Full usage details are contained within the provided help files and tutorials.

Acknowledgment

This work was conducted in the EPiCS project and received funding from the European Union Seventh Framework Programme under grant agreement n° 257906. `http://www.epics-project.eu/`

References

[1] L. Esterle, P. R. Lewis, X. Yao, and B. Rinner, "Socio-economic vision graph generation and handover in distributed smart camera networks," *ACM Transactions on Sensor Networks*, vol. 10, no. 2, 2014, to appear.

[2] P. R. Lewis, L. Esterle, A. Chandra, B. Rinner, and X. Yao, "Learning to be different: Heterogeneity and efficiency in distributed smart camera networks," in *Seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE Computer Society Press, 2013, p. 10.

[3] 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 *Proceedings of the Fifth ACM/IEEE International Conference on Distributed Smart Cameras*. IEEE Press, 2011, pp. 1–6.

[4] L. Esterle, P. R. Lewis, B. Rinner, and X. Yao, "Improved adaptivity and robustness in decentralised multi-camera networks," in *Proceedings of the Sixth ACM/IEEE International Conference on Distributed Smart Cameras*. IEEE Press, 2012, pp. 1–6.