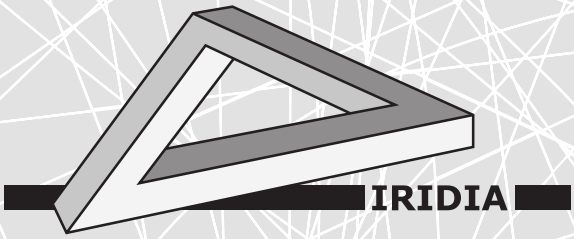

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

IRIDIA's Arena Tracking System

A. Stranieri, A.E. Turgut, M. Salvaro,
G. Francesca, A. Reina, M. Dorigo, and M.
Birattari

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2013-013

November 2013
Last revision: July 2014

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2013-013

Revision history:

TR/IRIDIA/2013-013.001	November 2013
TR/IRIDIA/2013-013.002	July 2014
TR/IRIDIA/2013-013.003	July 2014

IRIDIA's Arena Tracking System

Alessandro Stranieri alessandro.stranieri@gmail.com
Ali Emre Turgut ali.turguti@gmail.com
Mattia Salvaro mattia.salvaro@studio.unibo.it
Gianpiero Francesca gianpiero.francesca@ulb.ac.be
Andreagiovanni Reina areina@ulb.ac.be
Marco Dorigo mdorigo@ulb.ac.be
Mauro Birattari mbiro@ulb.ac.be
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium.

November 2013

1 Introduction

This document describes the multi-camera tracking system developed for the IRIDIA laboratory, which is targeted for multi-robot experiments in large experimental set-ups. The system is deployed in a large room exclusively dedicated to this kind of experiment, and we refer to this room as the Robotics Arena. The tracking system here presented is referred to as the Arena Tracking System, or ATS.

The purpose of this document is to provide an overview of this system, in terms of which hardware has been employed, which software has been developed and what use cases are offered to the researcher.

The first section is dedicated to the description of the hardware and the setup of the system. The second section presents the software architecture of the system. The third session presents the integration of the tracking system with the the ARGoS [4] simulator that enables the creation of augmented reality for the robots.

1.1 Motivations

One of the main focus of the research carried out at IRIDIA laboratory is on Swarm Robotics [1] and multi-robot applications. Experiments in Swarm Robotics involve large number of robots that navigate through the environment, sense it and act on it.

The initial purpose for the development of the tracking system here presented is to provide a tool that allows a researcher to record and control the state of the experiment throughout its complete execution. We refer to the state of the experiment as the position of all the individuals of the robot swarm at a particular time. The positions are computed with respect to a global frame of reference. Other than experimental analysis, the ATS has the additional functionality of implement virtual sensing. This means that as a robot navigates

through the environment, the system can virtualize the sensed environment enhancing the robot capabilities.

The main requirements of this system are: 1) provide the possibility to follow the evolution of the experiment across a large area; 2) record the state of the experiment with a good time resolution; 3) integrate the ATS within ARGoS to create a communication infrastructure between the ATS, the swarm simulator and the real swarm. The integration ot the two entities is called IRIDIA Tracking System, or ITS.

Several benefits spring by the integration of the ATS into ARGoS. The power of the simulator combined with the ATS opens a brand new branch of possible experiments, and at the same time eases the data gathering. For example, it allows to run experiments that involve sensors or actuators that the robots do not have onboard, or to prototype sensor and actuators that is desirable to mount on the robots in the future. From the data gathering point of view, using ARGoS integrated with the ATS during the experiment allows us to compute the evaluation function during the experiment execution, rather than calculate it on a further step.

The integration of ARGoS into the ATS is detailed in section 4.

2 Hardware

The ATS is composed of a set of cameras whose collective field of view covers the entire Robotics Arena. The cameras feed images through an Ethernet connection to a dedicated computer, the Arena Tracking System Server, located outside the Robotics Arena. This computer hosts and runs the API that a researcher can use to record an experiment. This section is dedicated to the description of the Robotics Arena set-up and the hardware on which the ATS is based.

2.1 Cameras

The Robotics Arena is a large room around $990cm$ large and $705cm$ deep, which can host a single multi-robot experiment across the whole area, or even several experiments in parallel, if divided along the shorter side.

While an experiment is in progress, its State is computed by applying recognition and decoding steps on a set of images acquired by an array of cameras deployed on the Robotics Arena's ceiling. This array consists of 16 cameras, arranged in a 4- by-4 matrix. The arrangement of the cameras has been drawn in order to have full coverage of the experimental area, and it has been tailored to the area size and the cameras' specifications.

For the choice of the camera, we opted for a *Prosilica GC1600 C*, manufactured by Allied Vision Technologies and shown in Figure 1. This particular camera has been chosen as it presented a favorable compromise between resolution, focal length, ease of interfacing and speed of transmission. Some of the technical specifications of the camera are given in Table 1.

The positioning of each camera within the Robotics Arena has been determined following these criteria: 1) The maximization the area of the Robotics Arena covered by the field of view; 2) Resolution useful for symbol decoding on the plane occupied by the top of the robots.

Figure 1: Prosilica GC 1600C

The final layout of the cameras with the distances between each camera is presented in Figure 2. Each camera has been placed on a wooden structure specifically manufactured and at a height of $243cm$ from the ground. With this configuration, each camera covers an area of about $350x260cm$, with a spatial resolution on the ground of about $4.6px/cm$

2.2 Network configuration and Computer Host

The cameras employed in the Arena Tracking System transmit the acquired images through an Ethernet interface linked through a 1Gbit connection to a dedicated switch installed in the Robotics Arena, which mounts a module for 10Gbit/s connection to the Arena Tracking System Server. The Arena Tracking System Server hosts 16 Intel® Xeon(R) CPU E5-2670 at 2.60GHz. Each core features the Intel® Hyper-Threading Technology, which enables it to run two threads per core. The Operating System is GNU/Linux Ubuntu 12.04.1 LTS for 64 bits architectures.

3 Arena Tracking System

The Arena Tracking System's software architecture is built as a three layer structure presented in Figure 3. The bottom layer consists on the QT Framework and the Halcon library [3]. Halcon is a library by MVTec Software GmbH, used in for scientific and industrial computer vision applications. It provides an extensive set of optimized operators and it comes along with a full featured IDE that allows for fast prototyping of computer vision programs, camera calibration and configuration utilities. The library also provides drivers to easily interface with a broad range of cameras. On top of those libraries lies an appli-

Prosilica GC 1600	
Interface	IEEE 802.3 1000baseT
Resoultion	1620 x 1220
Max frame rate	15 fps
Color modes	Gray Scale, RGB
Dimensions ((L x W x H in mm))	59x46x33
Mass	97 g

Table 1: Camera Specifications

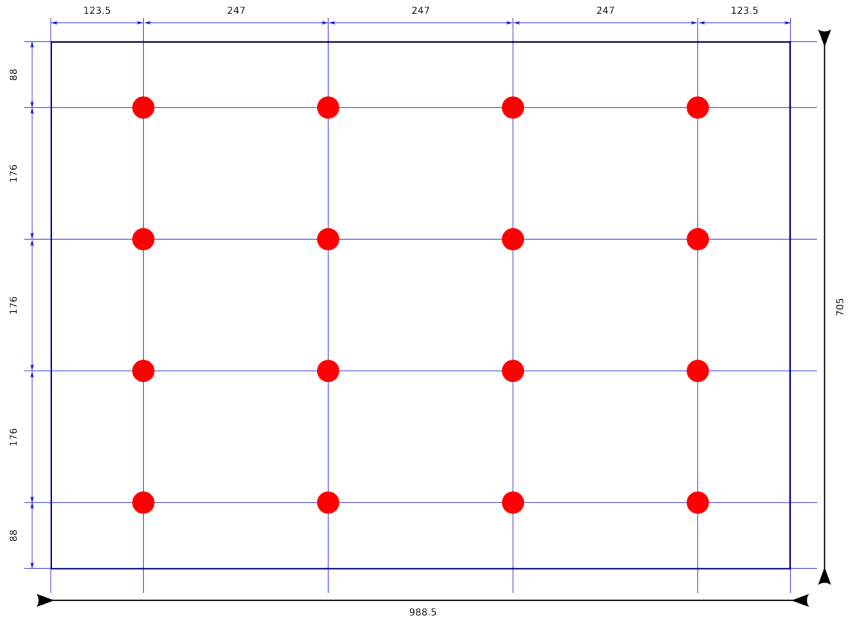

Figure 2: Layout of cameras (distances in cm)

cation specific library layer, called Arena Tracking System API. This API offers the possibility to configure different aspects of a tracking session, such as the cameras to be used, the objects to detect and the global frame of reference to in which the robots' positions have to be computed. The application level provides the researcher with two tools to support his or her experiments: a viewer for ATS standalone utilization, and a server to allow interaction with ARGoS running on a remote machine.

This section provides an overview over ATS API level and ATS Application level.

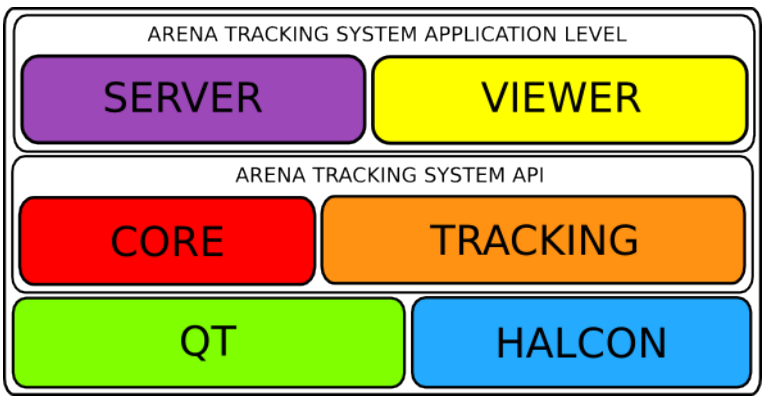

Figure 3: Arena Tracking System Software Architecture

3.1 Arena Tracking System API

This layer is based on Qt and Halcon libraries and provides a set of domain specific functions for tracking applications. The ATS API consists of two modules: Core and Tracking. The Core module provides the data types that are related to the representation of a tracking session state, whereas the Tracking module provides the tools to configure a tracking session and to extract the experiment state instances. The remainder of this subsection will describe in detail the functionalities of the ATS API layer.

3.1.1 The ArenaState Class

The use of the Arena Tracking System API revolves around two main classes: ArenState and ArenaStateTracker.

Given a running experiment that involves several robots navigating across the Robotics Arena, the ArenaState type represents the state of the experiment at a given time step. Figure 4 provides a simplified view of this type, and its components. An object of the ArenaState class contains all the elements detected at the timestep of creation. Each element is represented by its own ID, the position in the image where it was acquired and the position in the world frame of reference. There are two ways of accessing the elements detected. The method GerArenaElements gives a simple list of all the robots detected across the whole Experimental Arena. The other possibility, is to access the detected robots under a specific camera, by requesting access to the camera's specific tracker with the GetCameraState method.

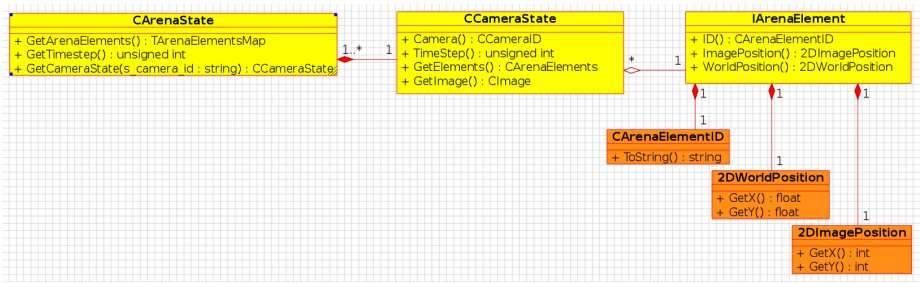

Figure 4: ArenaState class diagram

3.1.2 The ArenaStateTracker Class

A new instance of the ArenaState class can be created from the GetArenaState method of the ArenaStateTracker class. This class represents the configured set of resources that make up for the particular instance of a tracking session. Figure 5 contains the class diagram of the ArenaStateTracker class and its components.

The ArenaStateTracker's job is to configure and orchestrate a set of CameraStateTracker instances. Each CameraStateTracker is associated to an image source, such as a camera, and the operators to detect the imaged robots. It is basically a virtual device that performs all the steps (see Figure **??**) to output a sequence of elements within a specific field of view. A CameraStateTracker is composed of (I) an ImageGrabber, which encapsulates the logic to configure an

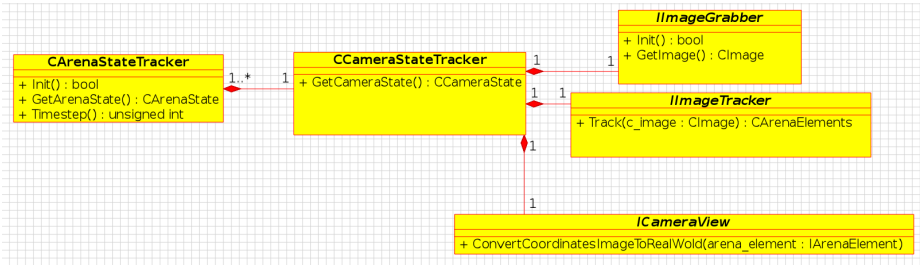

Figure 5: ArenaStateTracker Class Diagram

image source and acquire an image; (II) an ImageTracker, which is responsible for detecting the robots in the input image, decode their ID and output a list of objects that represent the robots' status; (III) a CameraView, whose job is to convert the image positions of a list of detected objects into the coordinates of a world reference system, given the external and internal parameters of a particular camera.

3.1.3 Detection and optimization

As mentioned in the previous section, an instance of the ArenaStateTracker class applies a detection and decoding operator on the images acquired at a given timestep. In order to detect the robots in action during an experiment while they are navigating, we designed a type of marker that allows for easy detection and easy decoding even when the image is relatively small. An example of the marker is depicted in Figure 6. These markers are applied on top of each robot involved in an experiment (see Figure 7 for an example with e-puck robots [2]) and carry their ID encoded in binary as matrix of black and white cells. Each time the detection step is performed, each frame acquired by the ArenaStateTracker is scanned for occurrences of the marker. Each time a marker is detected, the inner matrix is decoded and converted to an ID.

Figure 6: Marker Example

Figure 7: Markers on e-puck robots

The detection of markers is a very time consuming operation due to the high resolution of the camera images (i. e., 1620 x 1220 pixels) Although the Tracking System Server consists of 16 cores and Halcon libraries can be parallelized on those cores, the marker detection on one single image takes on average over 100 milliseconds. In multi camera experiments, the total tracking time of one frame is the detection time multiplied by the number of cameras in use. This datum would break another requirement which is the good time resolution. To have good time resolution and to provide the robots with real time virtual sensor

data, we need to synchronize the time spent for one step tracking with the clock of ARGoS. It had been empirically proved that 100 milliseconds is a good period for timesteps in ARGoS. To achieve effectiveness in ARGoS integration, the average tracking time per image must be around 100 ms. Which is the case only with single camera experiment. Therefore, optimization in image processing is needed. The optimization heuristic implemented to reach the required level of efficiency consists in processing the whole image only at a certain periodic timestep, called keyframe (See Figure 8). In the timesteps between two keyframes, the detection is performed only in the neighbourhood of a previously detected marker (See Figure 9). The size of the neighbourhood is a parameter that the researcher is able to set in the experiment configuration file. It is good practice to set the size of the neighbourhood greater or equal at the maximum distance that a robot can cover in 100 ms.

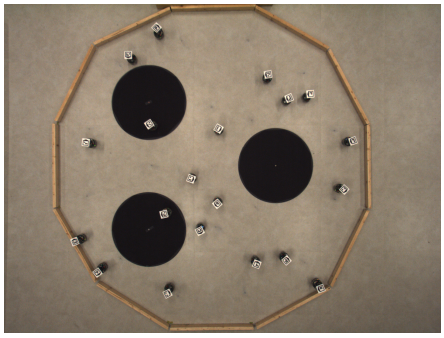

Figure 8: Image domain processed in a keyframe

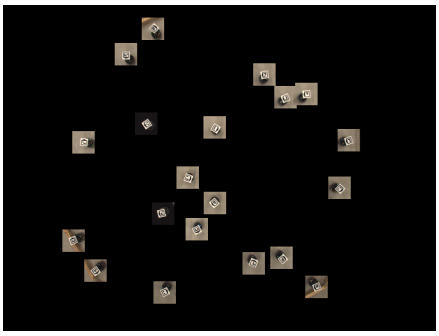

Figure 9: Image domain processed in a optimized frame

This technique is consistent under the hypothesis that robots move at a given maximum speed and therefore the maximum distance covered in 100 ms is easily computable. The periodic keyframe helps to recover the robots that might be lost by the ATS during the experiment. In fact, our experimental data demonstrate that the reliability of the ATS on marker detection is constrained to keyframe period and neighbourhood size (see Table 2). Therefore, lower the period, higher the reliability of the system, but also lower time efficiency (cf. also Figure 10 and Figure 11).

Both the keyframe period and the size of the neighbourhood are configuration parameters that can be tuned according to the requirements of each

ATS marker detection reliability			
Keyframe period (timesteps)	Neighbourhood size (cm)	Mean	Median
1	–	19.25725	20
5	30	19.05596	19
5	11	19.06481	19
5	10	19.02939	19
9	11	19.01369	19

Table 2: ATS marker detection reliability

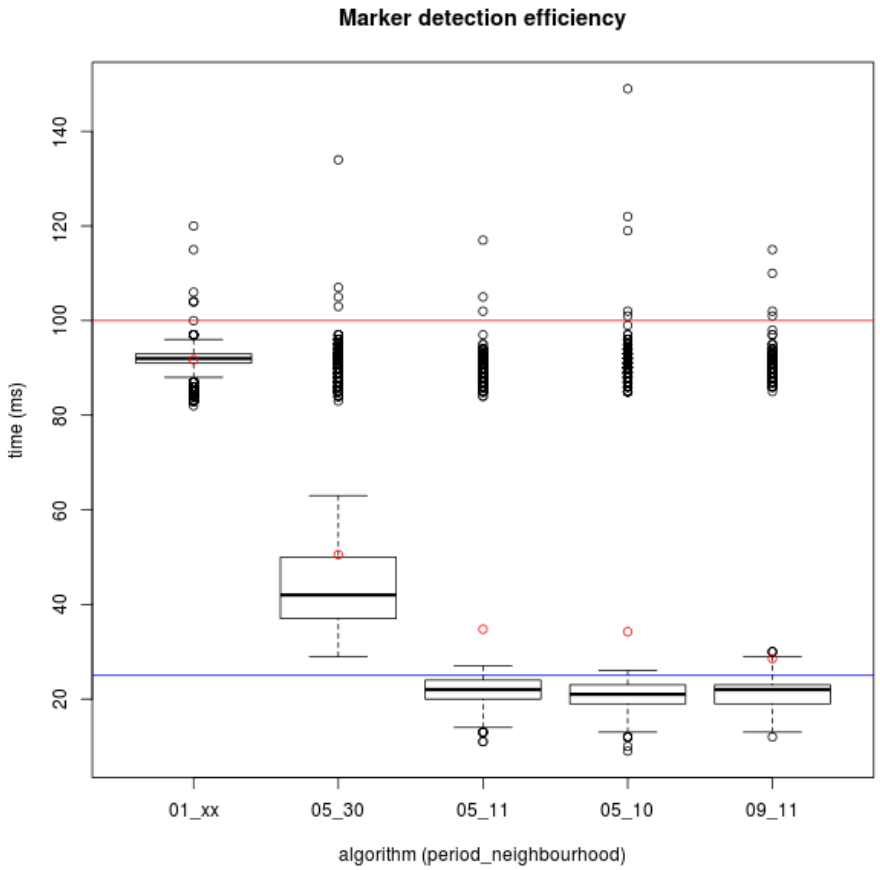

Figure 10: ATS marker detection efficiency in ms. The red line at 100 ms is a required upper bound for efficiency performances. The blue line at 25 ms is an ideal upper bound that allows a four camera experiment to meet the requirement of 100 ms for timestep. The red points represent the average computation time for each combination of keyframe period and neighbourhood size.

particular experiment. The researcher is in charge to settle the trade-off between time efficiency and accuracy on robots detection. Time efficiency grows by growing the keyframe period or reducing the neighbourhood. The same actions lead to a drop of the accuracy of robots detection. Setting the keyframe period to 1 is equivalent to disable the optimization (see section 3.1.4).

3.1.4 Configuration

Two other important classes of the API are ResourceManager and ExperimentManager. These two classes are responsible for declaring the set of resources that a researcher wishes to have available for image acquisition and marker detection, and for configuring a tracking session. Instances of these classes can load a tracking session's configuration from XML files.

The ResourceManager class can be used to read a XML file containing the

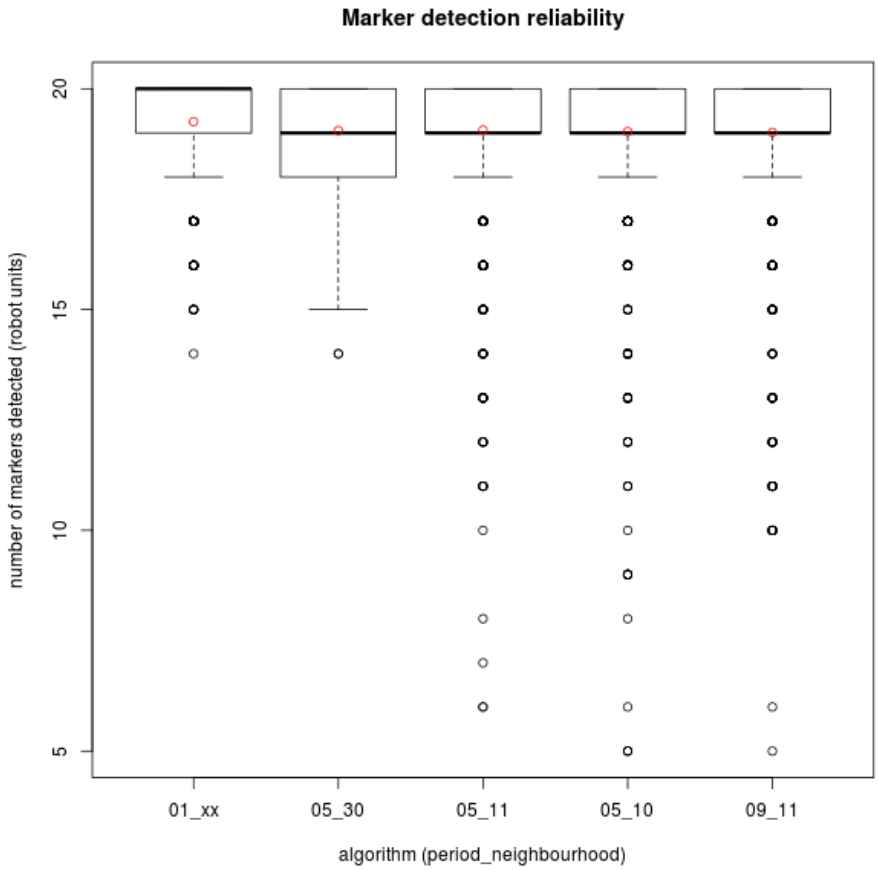

Figure 11: ATS marker detection reliability on a set of 20 robots

definition and the parameters of cameras, detectors and image transformations available for the session. Each of these elements is mapped to a unique ID, which can be used to acquire the object which controls it. An XML tree of a resources configuration files contains a root node, *arena_tracking_system*, and three children nodes: *grabbers*, *trackers* and *cameras*. The grabbers node contains the definition of every possible image source available in the experiment. It can be either a camera or a directory of images. For each grabber of type camera, in the cameras section must be added a node called *camera* that contains the path to the camera calibration files. Also, for each grabber of type camera a set of parameters is defined. In order to tune the image grabber properly in the desired condition of the environment, the researcher might like to change the parameters involved with the brightness of the image: *exposure_time_abs* and *gain_raw*. By raising the gain, the image results brighter but more noisy. In turn, by increasing the exposure time, the risk is to have blurred markers due to the movement of the robots. In the trackers section all the available trackers are listed in nodes called *tracker*. A useful parameter for the researcher here is the *robot_height* parameter. This parameter allows a researcher to reuse the

same set of markers on robots of different height, provided that in the same experiment the set of robots is homogeneous.

The ExperimentManager is instead used to read an XML file that describes the configuration of the ArenaStateTracker, in terms of its components. The researcher who wishes to configure an ArenaStateTracker instance has to simply specify in the XML file how many CameraStateTracker objects compose the ArenaStateTracker and, for each of them, specify the resources they should use calling them by their unique ID. An example of XML tree for an experiment configuration file includes a root node called *arena_tracking*, and two children node: *arena_state_tracker* and *experiment_record*. The arena_state_tracker node includes three attributes and a collection of nodes called *camera_state_tracker*, one for each grabber used in the experiment. The camera_state_tracker must include the specification of the grabber, a corresponding camera view if the grabber is of type camera, and the tracker. Back to the arena_state_tracker attributes, the attribute *server_port* defines the port to which the server is bound in the Tracking System Server in case the researcher decides to use the ATS this way (see section 3.2.2). The attributes *opt_key_frame_period* and *opt_square_size* are optimization parameters (see section 3.1.3). The former defines the period in timesteps of the key frame occurrence, the latter is the dimension in centimetres of the side of the square that forms the area around the markers on which the image domain is cropped.

3.2 Arena Tracking System Application Level

As stated earlier, there are two ways in which a researcher can be interested in using the ATS. One is to monitor the real time evolution of the experiment on all the cameras involved, controlling the beginning and the end of the recording. For this purpose, an application called Viewer has been created. The other implemented usage is to monitor and control the experiment through ARGoS, with all the benefits already mentioned. For this purpose, the Tracking System Server communicates with a client built ad-hoc within ARGoS.

In this section, the two applications will be presented in detail. Both of them must be launched with a path to a resource XML file and a path to an experiment configuration XML file as respectively as first and second parameter.

3.2.1 Viewer

The Viewer is an application that allows a researcher to visualize the progress of the experiment through the images of the active cameras. It supports any possible subset of cameras among those specified in the resources XML file, and let the researcher start and stop the visualization of the images. The recording of the images must be enabled in the XML configuration file in order to be effective in the Viewer application.

The GUI is composed of three main parts: (I) the control part, (II) the informative part and (III) the visual part, a part in which all the cameras are shown according to their position in the camera grid in the arena (see Figure 12). In the control part, there are buttons that the researcher can use to start and stop the tracking. The recording button is shown only for completeness, but its status is bounded to the corresponding attribute *"record_images"* in the XML configuration file. The refresh of the visual part heavily affects the performance

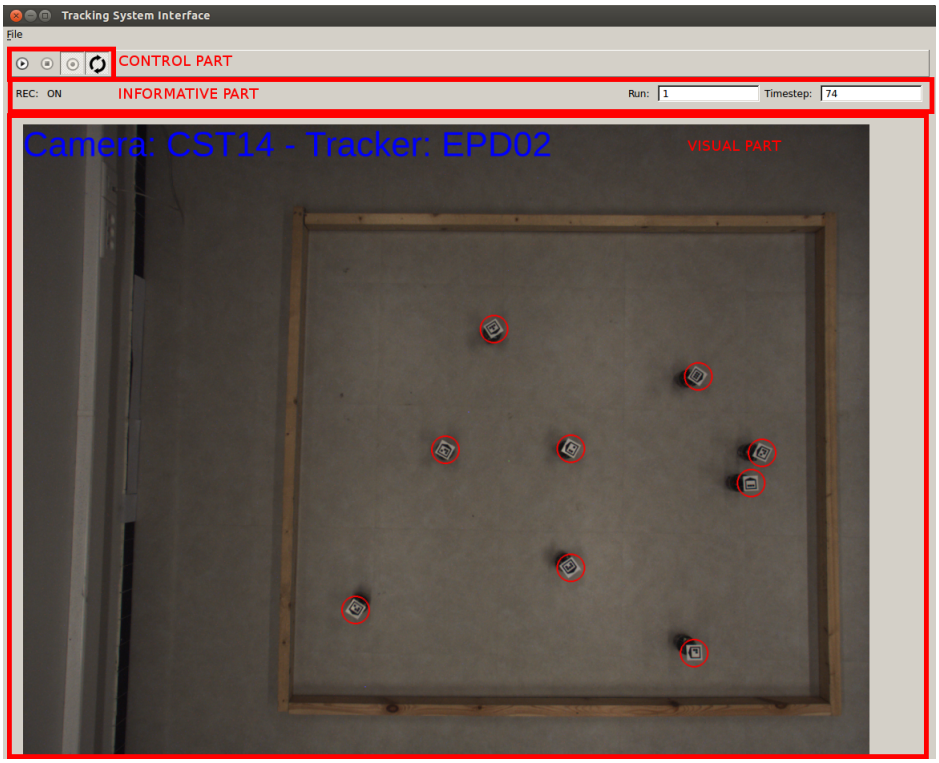

Figure 12: Viewer GUI

of the tracking step. The refresh toggle button allows the researcher to enable or disable the refresh of the visual part, in order to gain in terms of performance. In the informative part, some information about the status of the experiment is shown, such as enabled or disabled recording of images, current run number and current timestep. The meaning of run can be set in the XML configuration file by the attribute *"change_root_directorty_on_start"*. If the attribute is true, each run is a session that lasts from one start to the next stop. Otherwise the run will be only one, but with the possibility to suspend and restart the tracking and the recording when needed. In the first case, the timestep counter restarts at each new run, in the second case the counter continues from the last value. In the visual part, all the views of the camera involved in the experiment are displayed in a grid that reflects the disposition of the cameras in the arena. Each view reports its camera ID and tracker ID as reported in the resources XML file. It is possible to focus and enlarge the view of one or more cameras by clicking on the desired views. A pop up window with enlarged picture will appear.

3.2.2 Server

The Server application is the interface between the ATS and ARGoS. The application binds a socket on any network interface of the Tracking System Server to the port specified in the attribute *"server_port"* in the experiment configuration XML file. If no attribute is specified, the server will bind by default

to the port 4040. After binding, the server will enter its main loop. The loop consists of two simple steps: waiting for a connection and an inner loop that executes commands coming from the client. The inner loop ends when the client disconnects, then the control returns to the outer loop that starts again waiting for a new connection. There are up to now three commands that the client can ask the server to execute, but for the developer is very easy to extend this list. Those commands are *start experiment*, *stop experiment* and *one shot*. Start and stop experiment respectively enable and disable tracking. The server exploits the ATS API to perform tracking and sends to the client the Arena State at any timestep the tracking is enabled. One shot in turn sends the Arena State only once.

4 Integration into ARGoS

The most suitable mean of communication between the robot swarm and the Tracking System is ARGoS [4], a robot swarm simulator developed at IRIDIA. ARGoS is a modular system based on plugins. The plugin developed to interface ARGoS with the ATS is a physics engine, called Iridia Tracking System Physics Engine. Unlike a regular physics engine, the ITS Physics Engine does not calculate the position of each robot, but it receives the robot positions directly from the ATS. Once ARGoS places all the robots in the space, it is possible to exploit its power to send information about the virtualized environment to the robots through virtual sensors onboard. Furthermore, it would be possible for the robots to act back on the simulated environment thanks to virtual actuators. This last feature is not yet implemented and must be studied in the future.

The data flow for the architecture is illustrated in Figure 13. Notice that the virtual sensor data and the virtual actuator data flow in opposite directions, forming a loop between ARGoS and the robots.

In this section we describe how the communication between the ATS, ARGoS, and the swarm takes place, how real robots can sense virtual environment, and how it is possible to extend the set of virtual sensors by implementing specific virtual sensors. Finally, suggestions about possible future work are listed.

4.1 Architecture

The ITS Physics Engine plugin is the cornerstone of the interactions between ARGoS, the ATS and the robot swarm. As stated above, the goal of this physics engine is not to calculate the Arena State, but to open a channel of communication towards the Tracking System Server and obtain the Arena State from it. At the same time, the ITS Physics Engine is responsible for the communication between the simulated environment and the robot swarm for virtual sensing. To achieve these requirements, at the beginning of its execution the physics engine spawns two threads. One of them executes the Argos ITS Client, and the other one the Virtual Sensor Server. On robot side, the entity in charge of the communication with the Virtual Sensor Server is a thread called Virtual Sensor Client.

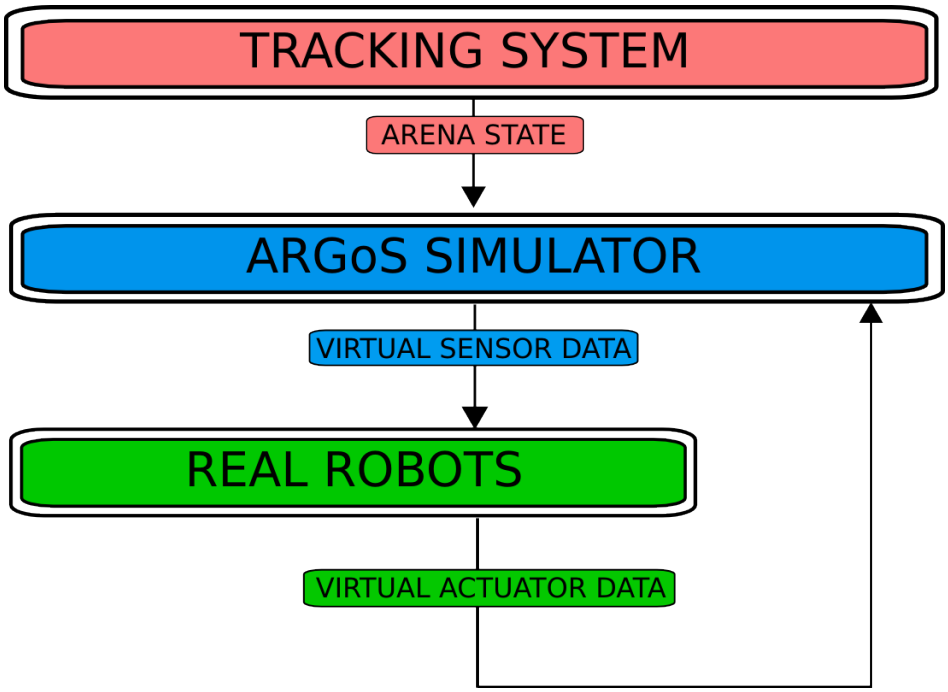

Figure 13: Communication flow in the system

4.1.1 Argos ITS Client

The Argos ITS Client is the client that connects to the Tracking System Server which must be already running. After the connection, the client sends a command to the server and waits for the answer. Normally, the Argos ITS Client asks the server to perform the tracking, and waits for the Arena State from the server. When the client receives a new Arena State, it updates a data structure called Arena State Struct, which is the data structure that the ITS Physics Engine reads during its update step. It is worth to note that the timesteps of the ATS and ARGoS are not synchronized, but thanks to the multi thread architecture the physics engine always obtains the latest data available.

4.1.2 Virtual Sensor Server

The Virtual Sensor Server is the endpoint of the communication with the swarm. Its task is to accept robot client connections at any time and send the data related to the virtual sensors mounted on the robot, and stored in a data structure called Virtual Sensor Data. The Virtual Sensor Data must be sent to the swarm at a specific time, synchronized with the update step of the physics engine. The Virtual Sensor Data is basically a collection of pairs, where one element is the robot ID, and the other is a byte array containing the data that must be sent at the end of the current update step. The byte buffer includes, for each virtual sensor to be updated, one byte for the sensor ID followed by a number of bytes that represents the updated data. The number of data bytes is fixed for each sensor, and known by the Virtual Sensor Server and the Virtual Sensor Client

on the robot thanks to a shared table called Virtual Sensor Register. Therefore, the length of the byte array sent does not need to be fixed. This strategy allows to save bandwidth by preventing to send data related to certain sensors. For example a virtual sensor can implement the policy to send data only if the variation of the reading is greater than a threshold.

4.1.3 Virtual Sensor Client

On robot side, the Virtual Sensor Client is again a thread that receives a new Virtual Sensor Data at each update step of ARGoS. The data are received in form of byte array and then converted in appropriate data structure by the specific virtual sensor. The deserialization made by the Virtual Sensor Client consists in consuming the byte array by reading one byte for the sensor id, retrieve the specific virtual sensor data size from the Virtual Sensor Register, then update the Virtual Sensor Data for the specific sensor with the portion of byte array of the given size. The deserialization is then completed when the update step of the specific virtual sensor is called and the byte array is transformed into the particular data structure of the sensor. This approach allows maximal flexibility because data is treated as generic byte array outside the virtual sensor.

4.2 Virtual sensing

The virtual sensing is one of the most powerful feature of the system, emerging from the integration of the ATS in ARGoS. As stated earlier, the virtual sensing allows real robots to feel the characteristics of a virtual environment in a sort of augmented reality. This peculiarity is interesting under different points of view. On one hand, it is possible to virtualize the environment in an easy way, creating scenarios that are difficult to reproduce in the real arena, or not possible at all. For example, it is possible to simulate obstacles of any shape without actually build them. Moreover, it is possible to create an evolving environment (e.g. movable obstacles, lights, color spots on the floor...) which is actually not possible to have in a regular arena. On the other hand the researcher can virtualize parts of the robot itself such as sensors and actuators that are not currently onboard the robots. This possibility is useful in case of a feasibility study for a new sensor/actuator. In fact it is now possible to study the characteristics of the wanted sensor, prototype is as a virtual sensor, and also test it in real experiments. Thanks to the testing the so designed virtual sensor can be validated and committed for hardware design, or it can be modified in order to better satisfy the requirements.

4.3 Virtual sensor implementation

The steps to take to implement a virtual sensor are different if the researcher wants to develop a brand new virtual sensor, or to virtualize a sensor that is already onboard the robots and already implemented as a couple of real and simulated sensors in ARGoS. The first case is a generalization of the second, where one preliminary step is needed.

The structure of the ITS Physics Engine reflects the tree structure of ARGoS. Inside main directory plugins, there are two directories: simulator and

robots. The researcher is intended to work inside the robots directory. The robots directory contains a directory generic, and a directory for each robot ARGoS can be built for. In each of those directories there are three directories: control_interface, simulator, and real_robot. The directory simulator contains all the files needed to use the robot in simulation. The directory real_robot contains all the files that form the library to be upload on the real robots. The directory control_interface includes the common interfaces that are used both on the real robots and in the simulator.

To virtualize an existing sensor, the researcher is required to implement the pair of classes one for the real_robot and one for the simulator. Also, the researcher must insert an entry in the virtual sensor register, located in generic/control_interface, assigning a unique ID to the virtual sensor. Finally, the virtual sensor must be added to the list of suitable sensors in the method InsertSensor of the class CRealEPuckITS. The real robot virtual sensor class must implement the same control interface of the sensor that the researcher is virtualizing, along as the real_virtual_sensor interface. The simulator virtual sensor class must also implement the same control interface as the corresponding simulated sensor, and the generic_virtual_sensor interface.

To virtualize a brand new sensor, the researcher must implement the corresponding control interface and enable the sensor's name in the initialization of the robot. After that, the user can follow the same steps described above for existing sensor virtualization.

4.4 Future work

Future development of the ITS includes a study oriented to the implementation of virtual actuators. The realization of virtual actuators will enhance the virtualization power of the ITS, closing the loop shown in Figure 13. Thanks to the addiction of the virtual actuators the robots will be able to sense and act on a virtual environment at the same time.

5 Concluding remarks

This document is not the final version and it will be updated along with the technical development of the Arena Tracking System.

References

[1] M. Dorigo, M. Birattari, and M. Brambilla. Swarm robotics. *Scholarpedia*, 9(1):1463, 2014.

[2] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stéphane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[3] MVTech Software GmbH. Halcon library website. URL `http://www.mvtec.com/halcon/`. Last checked on November 2013.

[4] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.