2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

INTERACTIVE AUGMENTED REALITY FOR UNDERSTANDING AND ANALYZING MULTI-ROBOT SYSTEMS

Fabrizio Ghiringhelli[1] Jérôme Guzzi[2] Gianni A. Di Caro[2]
Vincenzo Caglioti[1] Luca M. Gambardella[2] Alessandro Giusti[2]

Abstract— **Once a multi-robot system is implemented on real hardware and tested in the real world, analyzing its evolution and debugging unexpected behaviors is often a very difficult task. We present a tool for aiding this activity, by visualizing an Augmented Reality overlay on a live video feed acquired by a fixed camera overlooking the robot environment. Such overlay displays live information exposed by each robot, which may be textual (state messages), symbolic (graphs, charts), or, most importantly, spatially-situated; spatially-situated information is related to the environment surrounding the robot itself, such as for example the perceived position of neighboring robots, the perceived extent of obstacles, the path the robot plans to follow. We show that, by directly representing such information on the environment it refers to, our proposal removes a layer of indirection and significantly eases the process of understanding complex multi-robot systems. We describe how the system is implemented, discuss application examples in different scenarios, and provide supplementary material including demonstration videos and a functional implementation.**

I. INTRODUCTION

Let us consider the example in Fig. 1a, which shows a multi-robot system which performs a foraging task while adopting a bio-inspired collision avoidance algorithm [1]. The algorithm designers observe the behavior of the system, which appears in general terms correct, but plagued by occasional unexpected collisions, which were not observed in simulation; however, robots were individually debugged and shown to work correctly. Is the collision avoidance algorithm itself to blame? Is there an issue with sensing or actuation, for one or multiple robots? Analyzing and debugging a deployed multi-robot system is a daunting task, and very few tools are available to address the problem. This paper presents a practical, interactive system for visualizing the internal, normally hidden state of the swarm, overlaid in real-time over a live video feed acquired from a fixed camera; in other words, we provide an augmented-reality view of the robots' environment. The result is shown in Fig. 1b: it allows the user to easily determine whether robot perceptions match their surroundings, and whether higher-level internal states for each robot are consistent with each other, with the user's expectations, and with the ongoing activity in the swarm. In the specific case, the algorithm designers could quickly verify that the IR-based sensing subsystem of one robot was occasionally affected by outlier measurements, which only appeared when observing one specific neighbor, thus leading to a temporary corruption of the internal state which caused the sporadic issue.

For similar analysis and debugging purposes, software developers routinely make use of a variety of tools, such as prioritized logging messages or automatically-generated visualizations (plots, dashboards or diagrams). These tools allow one to efficiently monitor and analyze the evolution of parts of the system's state which would normally be hidden (internal state): graphical representations, in particular, are useful to summarize and make sense of huge amounts of debugging data, which would be unmanageable otherwise.

Unfortunately, the same tools are not very useful for analyzing what's happening to a real deployed robot. Indeed, since by definition a robot interacts with its surroundings, its internal state is normally meaningful only if considered alongside with the physical environment. For example, in order to understand whether a perception module (such as an obstacle detector) works correctly, the analyst needs to compare the module's outputs with the perceived entity (the actual presence and location of the obstacle), and possibly study the evolution of the outputs as the environment changes (e.g. the robot or obstacle are moved).

Analyzing multi-robot systems is even more challenging, because in spite of the behavior of each single robot is simple and well understood, the system still needs to be analyzed as a whole. This is often the case in *robot swarms*, where tightly-interacting robots share an environment and collaborate to solve a single task: then, the behavior (and internal state) of each robot is only meaningful when the robot is observed in the context of the rest of the swarm.

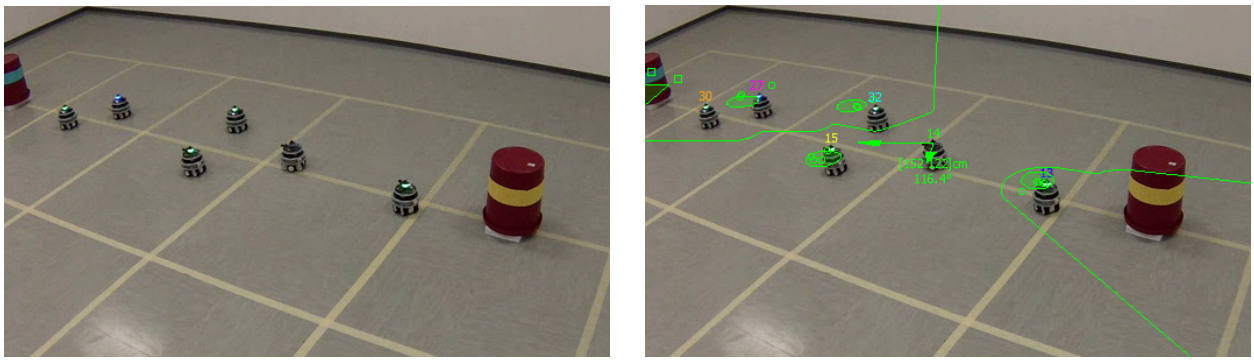
Our system includes two interacting components.

- A practical and efficient 2D visual tracker which uses active markers (i.e. LEDs blinking with a known pattern) to detect, identify and follow each individual robot in the image. If the robots expose odometry information, our tracker can take advantage of it for increasing tracking accuracy and tracking through long occlusions: furthermore, in this case a single blinking led per robot is sufficient to keep track of the robot's position *and orientation*.
- An interactive graphical interface which augments the camera view with information exposed by each robot, namely: *a)* textual or symbolic representations of the internal state, displayed next to the robot itself; *b)* spatially-situated data (such as raw or filtered sensor

[1]F. Ghiringhelli and V. Caglioti are with DEIB, Politecnico di Milano, Italy.

[2]A. Giusti, J. Guzzi, G. A. Di Caro and L. M. Gambardella are with IDSIA, USI/SUPSI, Lugano, Switzerland.

This research was partially supported by the Swiss National Science Foundation (SNSF) through the National Centre of Competence in Research (NCCR) Robotics (`www.nccr-robotics.ch`).



(a) **(b)**

Fig. 1: (a) a single frame taken from the live video feed, acquired by a fixed camera overlooking the robot arena. (b) The same frame augmented with tracking information, identifiers of each visible robot, and up-to-date spatial state information exposed by robot 14. The ellipses and circles indicate the position of the other robots, with the former being a filtered version of the latter, whereas the squares indicate the position of the landmarks. By looking at the ellipses we can see that robots 13 and 15 are correctly located, robots 32 and 27 are slightly mislocated and robot 30 is not detected at all.

readings, points, directions, past or planned paths), which the robot expresses in its own reference frame, overlaid on the actual environment the robot lies in. Furthermore, our system implements simple and intuitive interaction modalities for filtering relevant data for display, and sending commands to individual robots by clicking on them in the video feed. Commands may also be parametrized with real-world points, which are selected in the interface by clicking on specific points of the environment, and automatically translated by our system to the robot's own frame of reference.

The former component – tracking – is discussed in Section III; the **main contribution** of the paper lies in the latter component – i.e. the interactive augmented-reality visualization – which is described in Section IV. We present a real-time implementation of the entire system (available as supplementary material at `http://bit.ly/swarmvis`), and demonstrate its application to several swarm robotics research scenarios in Section V and supplementary videos.

II. RELATED WORK

A. Tracking for multi-robot systems

Robot tracking is a fundamental tool in robotics research; many off-the-shelf solutions exist in different price, accuracy and functionality ranges. The most common high-level systems [2], [3] are based on infrared (IR) reflectors and multiple synchronized IR cameras but, especially when only 2D tracking information is required (as is often the case with ground robots), inexpensive systems based on ordinary cameras and visual fiducial markers are often adopted [4], [5], [6]. Using active markers – such as LEDs blinking in a known time-based pattern – is an appealing solution for overcoming the issues of 2D fiducial markers, which has been recently adopted for ultra-low-latency 3D tracking of quadcopters [7] using special dynamic vision sensors [8].

In this paper, we use a simpler implementation of time-coded active markers, imaged by a single ordinary fixed camera, which we describe in Section III. This approach is suitable to our scenario because it is inexpensive, can be setup easily, and has minimal requirements in term of camera specifications and scene illumination conditions. In contrast, using 2D fiducial markers normally requires higher camera resolution [4], because the marker image needs to be large enough to be decoded. IR-based systems, on the other hand, are much more accurate and informative, but are expensive and require to setup and calibrate at least one dedicated IR camera. Our system, instead, solves the tracking problem directly on the video feed which is displayed to the user. The downside of our tracking approach is that, unless at least two leds are mounted on each robot, the orientation of the robot is not directly observed, but has to be inferred from the robot's motion and odometry data (see Section IV).

Note that the core contribution of the paper, described in Section IV, is not tied to this specific tracking approach, but could be trivially adapted to exploit any of the more sophisticated systems referenced above.

B. Augmented reality for the analysis of complex systems

Overlaying robot internal states on the robot's environment is an important aid to the development process, and this idea has in fact been implemented for *simulators* such as Player [9] and ARGoS [10]. Our approach implements the same ideas in the real world.

A preliminary demonstration of a similar approach has been implemented on real robots [11] by physically *projecting* the robot data over the environment, for human-robot interaction purposes; the idea of mixing robot data on the real environment has also been explored using virtual reality headsets [12]. Several earlier works [13], [14] also explored augmented interfaces for improving human-robot collaboration and tele-control.

III. TRACKING ALGORITHM

Our system relies on a simple yet robust visual tracking approach for identifying and following each visible robot which is assumed to move on a planar surface. Each robot is equipped with an RGB led (*beacon*), which is controlled

in a known, pre-defined temporal pattern alternating two specific colors (in the following, green and blue). Since it is unique for each robot, such pattern allows to determine the ID of each detection, automatically discarding spurious tracks. As with most tracking algorithms, our approach is composed of two distinct phases, namely *initial localization and identification* and *robot tracking*.

A. Localization and Identification

The goal of the initial localization algorithm is to locate and identify all robots in the scene, thereby providing a reliable starting point for the subsequent tracking algorithm.

Let $K = \{(r, \tau_b, \tau_g)_i \mid i = 1, \ldots, n\}$ be a set of n keys, where each key k_i is a three-tuple characterizing the robot r_i by:

- the duration of the beacon *blue* blink interval τ_b expressed in number of frames
- the duration of the beacon *green* blink interval τ_g expressed in number of frames

Let the m-tuple $\Gamma_m = (\gamma_1, \gamma_2, \ldots, \gamma_m)$ denote a sequence of m video frames, where γ_j refers to the image frame at time t_j which immediately follows (in time) γ_{j-1}.

The initial localization algorithm receives a set K of keys and a sequence Γ_m of image frames as input, and returns a set of m-length position tracks of each key matching a robot's blink code, where the position is expressed by the (x, y) pixel coordinate of the robot's beacon center on the image plane.

The algorithm maintains a set of candidates B which is first populated with the beacons located in γ_1: $B \leftarrow B_1$. This is obtained by means of a fast frame-based *beacon detection* routine, implemented as a bright object detector based on intensity thresholding and connected component analysis. Note that such a simple approach often returns spurious detections (i.e. reflections or other light sources): this is expected, and the subsequent steps of the algorithm take care of removing such detections. For a given frame, the *beacon detection* routine returns the image coordinates of each bright spot ($b.\boldsymbol{pos}$), along with its average RGB color in a small region around its center ($b.\boldsymbol{col}$). Let $T(B)$ be a generic data structure storing the position track and color track of all candidates in B.

Next, the algorithm goes through an iterative procedure over the remaining $m-1$ frames that keeps updating $T(B)$, eventually by adding new tracks as new candidates are detected along the way. During the i-th iteration it first locates the set B_i of beacon candidates in γ_i, then constructs a *tracking relation* from B to B_i. Let $R_{n_r \times n_c}$ be the incident matrix of such a binary relation, where $n_r = |B|$ and $n_c = |B_i|$. Then $R(x, y) = 1$ if the pair of points $b.\boldsymbol{pos}$ and $b_i.\boldsymbol{pos}$ are in *motion correspondence* to each other, meaning that they represent the motion of the candidate b from frame γ_{i-1} to γ_i. Conversely, $R(x, y) = 0$ if such points belongs to two distinct candidate tracks. The construction of R is driven by the assumptions that the location of the robot do not change notably across two adjacent frames (proximity constraint).

Since the algorithm guarantees the consistency of R, i.e. each row and column of R has at most one 1-element, it turns out that only three distinct scenarios need to be handled:

- a *non-zero row* of R identifies one single pair of points in motion correspondence.
- a *zero row* of R means that the corresponding candidate b was not located in γ_i. In such case, the algorithm's policy is to "keep alive" the existing path by replicating $b.\boldsymbol{pos}$ in γ_i.
- a *zero column* of R reveals that none of the existing paths extends to the point $b_i.\boldsymbol{pos}$. This implies that b_i should be considered as a new beacon candidate, which therefore must be added to B and tracked just as any other existing candidate.

Once all m frames have been processed, $T(B)$ holds the tracked data (both position and color) of all beacon candidates in B. To decide which candidates are indeed actual robots, the algorithm applies a two-stage filtering process which first removes unreliable candidates, and then discards those with a non-matching key.

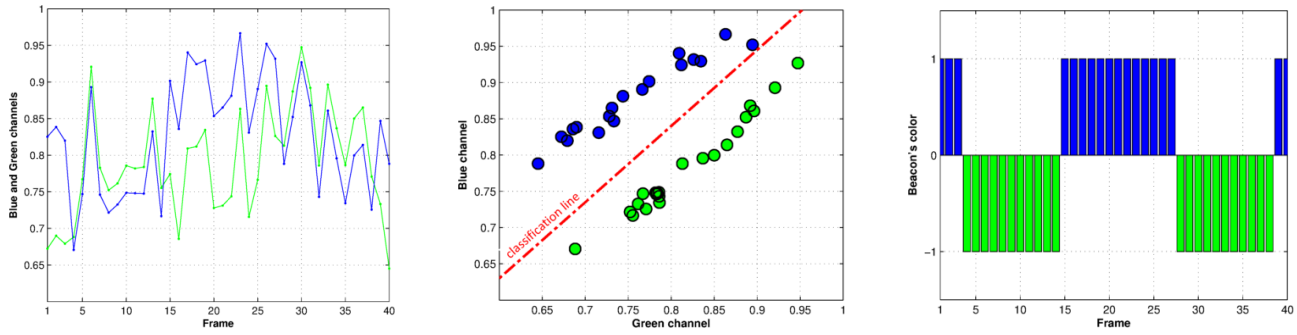
The filter's first stage is achieved by computing the *track quality index* (TQI in the following) of each candidate's position track. The TQI is a measure of the reliability of a track and is defined as the portion of all frames in the track where a beacon was actually detected. This filtering stage removes from B all candidates with a TQI smaller than a predefined threshold value, namely *Track Quality Index Threshold* (`TQIThr`).

The set of candidates with a valid TQI is further processed by means of a *key matching* algorithm which, for every candidate in B, seeks the best matching key in K. The first task involves generating a blink code from each respective candidate's m-length color track. Essentially this is a *classification step*, where each point of the RGB color track is assigned to a class describing the corresponding LED color, i.e. *blue-led* or *green-led*. This is achieved through a *linear binary classifier* learned through logistic regression during the system calibration, using as features the blue and green components of the observed color. An example of blink code generation is shown in Fig. 2.

Once all blink codes have been generated, the next task is to associate each blink code to the best matching key. Let $K^c = \{\boldsymbol{k}_i^{\boldsymbol{c}} \mid i = 1 \ldots n\}$ be the set of blink codes and $B^c = \{\boldsymbol{b}_j^{\boldsymbol{c}} \mid j = 1 \ldots n_b\}$ the set of generated blink codes, where n_b denotes the number of candidates in B which passed through the TQI filter. A circular cross correlation is performed on every possible correspondence $(\boldsymbol{k}_i^{\boldsymbol{c}}, \boldsymbol{b}_j^{\boldsymbol{c}}) \in K^c \times B^c$. The peak value of the cross correlation is taken as a measure of similarity between $\boldsymbol{k}_i^{\boldsymbol{c}}$ and $\boldsymbol{b}_i^{\boldsymbol{c}}$. We denote this coefficient by $d_{(i,j)}$.

The *correlation coefficient matrix* holds the correlation coefficients of all correspondences $(\boldsymbol{k}_i^{\boldsymbol{c}}, \boldsymbol{b}_j^{\boldsymbol{c}})$:

$$\text{CCM} = \begin{bmatrix} d_{(1,1)} & \ldots & d_{(1,n_b)} \\ \ldots & \ldots & \ldots \\ d_{(n,1)} & \ldots & d_{(n,n_b)} \end{bmatrix} \tag{1}$$



(a) (b) (c)

Fig. 2: Example of blink code generation from a 40-point color track ($m = 40$). The input data is the green and blue raw values of the RGB color track (a). Each track's point is described by its green and blue features and classified using a binary linear model in either *blue-led* or *green-led* (b). The blink code is obtained as the temporal sequence of such classifications (c).

Given CCM, the algorithm determines an optimal mapping between the set of known key codes K^c and the set of observed blink codes B^c. Each $(\boldsymbol{k}_i^c, \boldsymbol{b}_j^c)$ map entry requires that its respective correlation coefficient $d_{(i,j)}$ be larger than a predefined similarity threshold, namely `KeyCorrThr`. In addition, we ensure that a blink code is mapped to at most a single key, and vice-versa. The beacon candidates whose blink code was mapped to a key code represent the final set of localized robots, which is the output of the initial localization algorithm.

For example, suppose `KeyCorrThr` $= 0.9$ and consider the following CCM, wherein the bold values highlight associations of key codes (rows) to blink codes (columns):

$$\text{CCM} = \begin{bmatrix} 0.20 & 0.60 & \mathbf{0.93} & 0.10 & 0.23 \\ 0.60 & 0.85 & 0.63 & 0.37 & 0.56 \\ \mathbf{0.91} & 0.60 & 0.20 & 0.94 & 0.37 \\ 0.86 & 0.46 & 0.10 & \mathbf{0.97} & 0.92 \end{bmatrix}$$

$\boldsymbol{k}_2^c$ doesn't map to any blink code because all its correlation coefficient values are below `KeyCorrThr`. Furthermore, no mapping was found for $\boldsymbol{b}_5^c$, as its unique potential match is with $\boldsymbol{k}_4^c$, which however is more tightly correlated with $\boldsymbol{b}_4^c$.

B. Robot tracking

The final set B of robots localized in frame γ_m yields the initialization for a multi-target Kalman filter which is used to follow such robots in subsequent frames: a) the state of each target is its position and apparent velocity in the image b) the measurements of each frames are the outputs of the *beacon detection* routine. Measurements are associated to the nearest target, and tracks are discarded if no measurements are associated for a given number of frames.

IV. AUGMENTED REALITY OVERLAY

During execution, each robot exposes odometry data as well as textual or graphical information to be overlaid on the video feed: in our implementation, this happens through TCP packets sent to from the robot to the visualization server, but different mechanisms (such as ROS messages) may be more suitable in other environments. Each message specifies the originating robot's ID – which allows the visualization server to identify the corresponding robot tracked in the video image.

Our system allows robots to expose three types of information to be overlaid to the video feed.

- Textual information, such as logging and debugging messages.
- Symbolic information, i.e. simple graphical representations which summarize parts of the robot's internal state, which have no relation with the spatial environment around the robot.
- Spatially-situated data, i.e. entities which have a specific location in the environment with respect to the robot.

Textual and symbolic information is displayed next to the robot in the video feed and is not affected by the robot orientation or the camera viewpoint, akin to *billboarding* [15] in computer graphics. In contrast, spatially-situated data is drawn on the video feed using the robot's frame of reference, and is therefore affected by camera perspective and robot orientation, as in augmented reality systems. Robots take advantage of the geometry primitives provided by the Well-known-text [16] standard to represent spatially-situated information as text.

A. Reference Frames

In the following, we refer to three 2D reference frames (see Fig. 5).

- *Image Frame*: a left-hand coordinate system with the origin located at the top-left corner of the image, the x axis along the upper border and the y axis along the left border.
- *World Frame*: a user defined right-hand coordinate system placed on the ground plane.
- *Robot Frame*: a right-hand coordinate system centered at the robot's position, with the x axis oriented forwards.

Homography H_{IW} maps the image frame to the world frame. It is computed offline by the user, by clicking on four points lying on the floor, at a known location relative to each other (e.g. as the corners of an A4 sheet of paper). This allows to recover the homography between the image frame and an horizontal metric reference frame lying on the

floor, and does not need to be repeated unless the camera is moved. Subsequently, the user can isometrically adjust the world frame as needed by clicking on a point on the floor (which sets the origin) and dragging in a direction (which sets the orientation). The latter operation is only required if the user needs robot positions to be expressed according to a specific origin.

The transformation H_{WR} linking the world frame to a robot's frame is a 2D roto-translation which is updated in real time as the robot's position and orientation are tracked by the system. Besides, if additional calibration data were provided – e.g. the real and apparent height of a robot in known positions on the floor – then the system can estimate the ground plane and adjust the origin of each Robot Frame accordingly, as shown in Fig. 7a.

In order to draw spatially-situated content generated by the robot, the points expressed in the Robot Frame are premultiplied by H_{IR}^{-1}, where $H_{\text{IR}} = H_{\text{WR}} \cdot H_{\text{IW}}$ (see Fig. 5). Note that H_{IW} is fixed and system-wide, whereas H_{WR} is continuously updated and specific to each visible robot.

B. Recovering robot orientations

As previously described, we track a single point for each robot. On one hand, this allows our tracking system to require minimal setup, handle small robots, and cope with different resolution and quality of the input video stream. On the other hand, a single point only identifies the location of the robot in the image, but not its orientation. Robot orientation is necessary for correctly drawing spatially-situated information exposed by a robot, which is expressed with respect to the robot's own frame of reference.

Robot orientation is tracked by merging each robot's odometry information with visual tracking data. In particular, at any time the tracking module measures a significant motion of the robot; we can define the robot's Instant Velocity Vector (IVV) as the smoothed derivative of its tracked position. Comparing such information with the robot odometry for the same time allows the system to fix the orientation of the robot. Once the orientation is known, odometry information can be integrated even if no translation is visible, until an IVV can be observed again.

For example, consider the case of a robot that is exposing its odometry information while moving forwards. The tracking module observes the beacon's motion in a direction d in the World Frame, which enables it to rotate the Robot Frame so that its x axis points towards d. Now, imagine the robot stops and begins to rotate around its vertical axis; in such situation the tracking module only detects a stationary beacon and thus is unable to determine an IVV. However, integrating the odometry information enables the system to align the Robot Frame with the actual direction the robot is pointing at. Because odometry may drift in time or generally be unreliable, the system will re-align the robot frame to the IVV as soon as significant motion is visible in the video frames. A diagram of the whole system is depicted in Fig. 4.

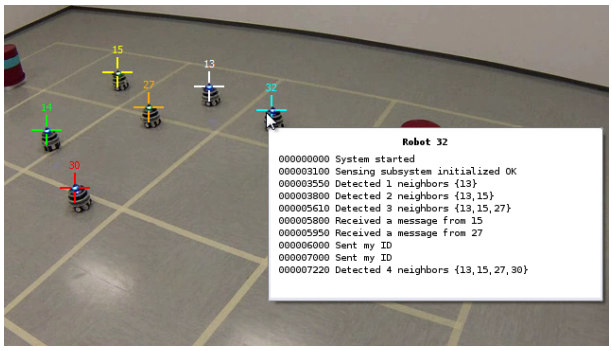


Fig. 3: Example of debug message history shown by hovering a specific robot.

C. Interaction Modalities

Our system implements simple interaction modalities which allow the user to explore the available data, and interact with the robot system by taking advantage of the augmented video feed. More specifically, every piece of displayed information is linked to a specific class: for example, all raw sensing data for a each sensor are grouped under the same class, whereas higher-level state information is associated to a different class. The user can interactively manipulate the visibility (and graphical properties, such as color and transparency) of individual information classes, in order to focus on the aspects under analysis.

In addition, every piece of information is associated to its originating robot and can be viewed simply by hovering or clicking the robot's image in the video feed. This allows the analyst to remove irrelevant information from view when studying the behavior of a specific robot (most figures in the paper were shot in this mode).

The interface can also be configured to *send* messages to specific robots by clicking on their image in the video feed (e.g. instantly disabling a robot that is misbehaving). This removes a layer of indirection, and can prove very useful in

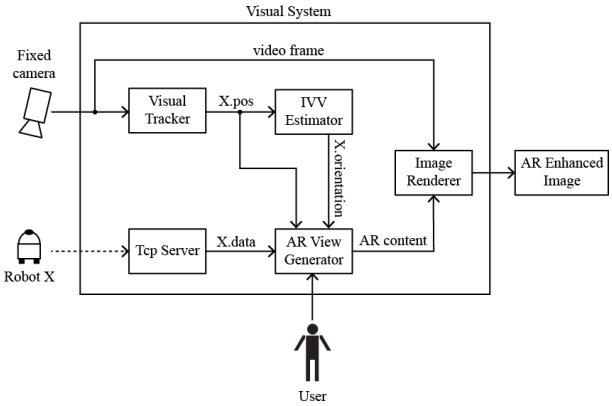


Fig. 4: Logical representation of the entire system. The position data provided by the visual tracker are merged with odometry data received from each robot; this fixes the robot's orientation and defines the robot coordinate frame in the image. Data sent from each robot is then overlaid to create the augmented reality view, which the user can customize as needed.

a variety of applications, especially when many robots are in view.

The transformations computed by the system are primarily used in order to map positions expressed in the reference frame of a specific robot to the image frame, which is needed for drawing spatially-situated data. However, by inverting the same transformations we can achieve the opposite goal, i.e. translating a point in the image to the corresponding point on the floor, expressed in the robot's own reference frame. This mechanism allows us to implement more sophisticated interaction modalities, in which a message to a given robot is parametrized with one or more real-world points, expressed in the robot's own coordinate system. For example, clicking on a robot and dragging towards a specific point on the floor can trigger the transmission of a message instructing the robot to move towards that waypoint.

V. IMPLEMENTATION AND USAGE EXAMPLES

The robot tracking and visualization modules are implemented as a single cross-platform multithreaded C++ application that can process and augment a 30FPS VGA video stream in real time on an average Intel i5 laptop. A preliminary release and demonstration videos are provided as supplementary material at `http://bit.ly/swarmvis`.

A simple application in a single-robot scenario is illustrated in Fig. 6, where orange lines depict the robot's knowledge of the surrounding environment (which can be considered satisfactory in this case), derived from self-localization in a known map. The white circle shows the expected position of the person (not directly visible to the robot at this time, but correct nonetheless) as well as its inferred motion direction, which due to a software problem is far wrong. Because of this issue, the scenario results in a collision.

Sample augmented views for a more complex swarm robotics scenario are reported in Fig. 7. In particular, Fig. 7a

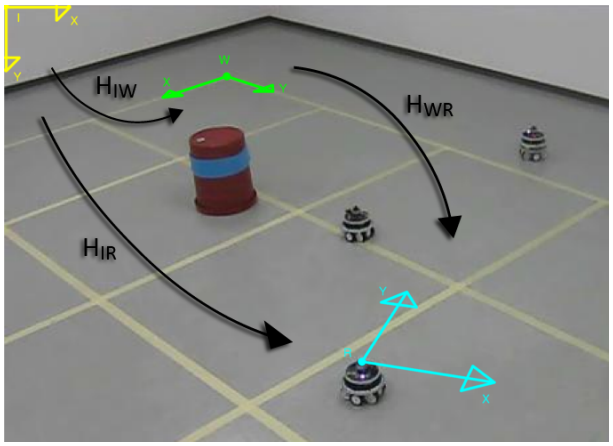
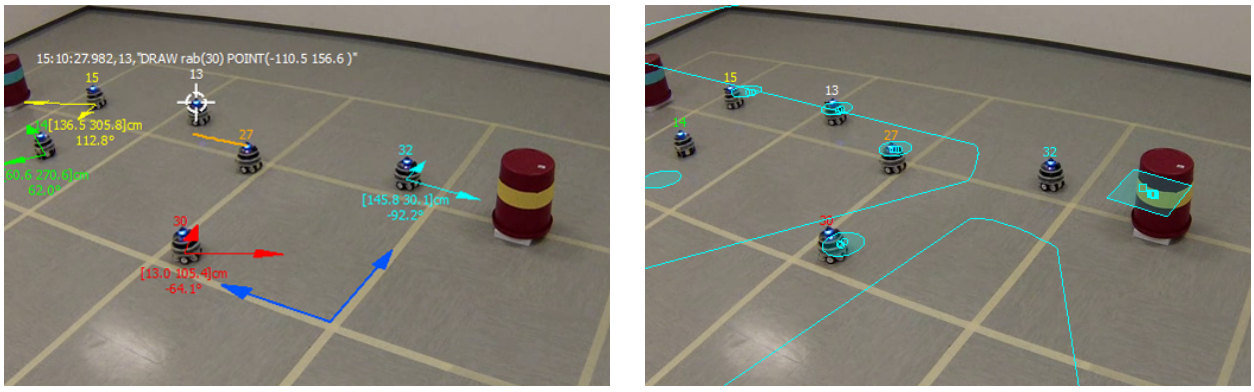


Fig. 5: Coordinate frames and related transforms. Mappings from the Image frame (I) to the World frame (W) and from the World frame to the Robot frame (R) of one specific robot; such transformations are defined respectively by the matrices H_{IW} (fixed and system-wide) and H_{WR} (continuously updated and specific to each visible robot).

Fig. 6: Illustration of simple augmented-reality data overlayed in a single-robot human-perception scenario (see text).

displays the tracked positions and reference systems associated to each visible robot. Fig. 7b displays all spatially-situated data generated for this application by robot 32. Small circles indicate the raw sensing information (acquired through a dedicated robot-mounted IR-based range-and-bearing system) concerning the position of each visible robot. Larger ellipses represent higher-level state information resulting from filtering such raw measurements. We can notice that robot 14 is not directly visible by robot 32 (i.e. no raw observations are available), which is expected since it is occluded by robot 27. Still, the internal state of robot 32 includes an (slightly wrong) estimate of the position of robot 14. Similarly, small squares and the large quadrilateral represent respectively raw and filtered information of the landmark's position (the upside down basket with the yellow band), detected through a computer vision approach which in this case works satisfactorily well. The solid continuous line represents an high-level internal state of the reactive obstacle avoidance algorithm running on the robot [17]. In particular, for each direction, the line represents the position where the robot expects to first collide with an obstacle, under the assumption that all obstacles maintain their current velocity. Note that, at the time the screenshot was taken, robots 30 and 27 were moving towards the yellow landmark, and robot 32 was aware of this fact (although this is not explicitly represented in the view). Therefore, the depicted high-level internal state matches our observations of the environment. Fig. 1 shows the similar setting using the same symbology, for a different robot and video frame. The entire processed video is available as supplementary material.

Finally, in Fig. 8 we illustrate the overlay of symbolic, non spatially situated data, in the context of a distributed gesture recognition problem [18]. Here, by means of its embedded camera each robot is observing the subject while showing one of four possible gestures to the swarm. The swarm needs to collaboratively reach a final decision on a single gesture, by aggregating the opinions of each robot. Here, the overlay graphically represents such opinions, in form of a bar plot of probability values for each class (the correct gesture in this case being class 4). We can easily observe that off-center robots produce wrong, uncertain opinions, whereas robots



(a) (b)

Fig. 7: Example of augmented reality enhanced view. Each robot is assigned a unique color which is used to display its AR content. (a) shows a variety of AR elements. At the bottom of the image, drawn in light blue color, is the world frame. Each robot's label is displayed right above the beacon. Robots 15, 14, 30 and 32 show their spatial reference system (placed on the estimated ground plane) along with the position coordinates and the orientation angle in the world frame. The position track is shown for robot 27, whereas robot 13 shows the tracking region and the last message sent. In (b) spatially-situated data generated by robot 32 are displayed (see text).

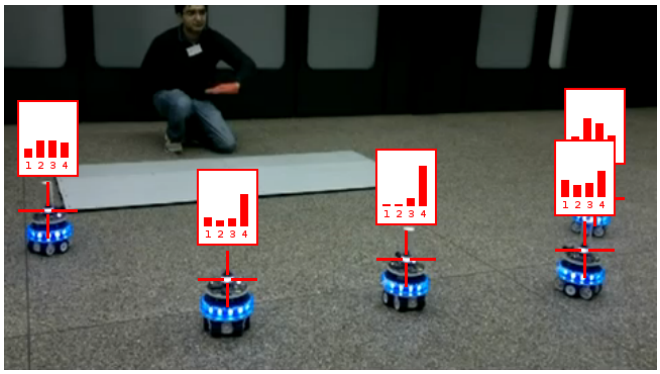


Fig. 8: Illustration of symbolic, non-spatially-situated data displayed over each corresponding robot.

close to the center have a better viewpoint.

VI. CONCLUSIONS

We presented a practical approach for visualizing the internal, normally hidden state of a deployed multi-robot system, displayed in real-time over a live video feed acquired by a fixed external camera overlooking the robots' environment. Demonstration videos and various examples show that the system can represent an useful aid in developing, understanding and debugging multi-robot systems, which is a complex task for which very few tools are available. Supplementary material provides a functional implementation of the system for ground robots, which makes use of blinking LEDs for robot localization, identification and tracking. The system is currently being finalized by integrating it in the ROS framework, taking advantage of the standardized publish-subscribe message passing mechanism for the interaction among robots, the tracking module, and the visualization module. We expect that a ROS-enabled release will open the way to adoption by other researchers dealing with multi-robot systems.

REFERENCES

[1] J. Guzzi, A. Giusti, L. Gambardella, and G. A. Di Caro, "Bioinspired obstacle avoidance algorithms for robot swarms," in *Proc. of the Int. Conf. on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS)*, 2012.
[2] "Optitrack," http://www.naturalpoint.com/optitrack/.
[3] "Vicon," http://www.vicon.com/.
[4] S. Ceriani, G. Fontana, A. Giusti, D. Marzorati, M. Matteucci, *et al.*, "Rawseeds ground truth collection systems for indoor self-localization and mapping," *Auton. Robots*, vol. 27, no. 4, pp. 353–371, 2009.
[5] N. Correll, G. Sempo, Y. L. D. Meneses, J. Halloy, J. louis Deneubourg, *et al.*, "Swistrack: A tracking tool for multi-unit robotic and biological research," in *Proc. of IROS*, 2006, pp. 2185–2191.
[6] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Proc. of ICRA*, 2011, pp. 3400–3407.
[7] A. Censi, J. Strubel, C. Brandli, T. Delbruck, and D. Scaramuzza, "Low-latency localization by active led markers tracking using a dynamic vision sensor," in *Proc. of IROS*, 2013, pp. 891–898.
[8] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 × 128 120 db 15 μs latency asynchronous temporal contrast vision sensor," *J. of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.
[9] T. H. J. Collett and B. MacDonald, "Augmented reality visualisation for player," in *Proc. of ICRA*, 2006, pp. 3954–3959.
[10] C. Pinciroli, V. Trianni, R. OGrady, G. Pini, A. Brutschy, *et al.*, "Argos: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm intelligence*, vol. 6, no. 4, pp. 271–295, 2012.
[11] F. Leutert, C. Herrmann, and K. Schilling, "A spatial augmented reality system for intuitive display of robotic data," in *Proc. of the Int. Conf. on Human-Robot Interaction (HRI)*, 2013, pp. 179–180.
[12] M. Daily, Y. Cho, K. Martin, and D. Payton, "World embedded interfaces for human-robot interaction," in *Proc. of the Annual Hawaii Int. Conf. on System Sciences*, 2003.
[13] M. Fiala, "A robot control and augmented reality interface for multiple robots," in *Proc. of the Canadian Conf. on Computer and Robot Vision (CRV)*, 2009, pp. 31–36.
[14] P. Milgram, S. Zhai, D. Drascic, and J. Grodski, "Applications of augmented reality for human-robot communication," in *Proc. of IROS*, 1993, pp. 1467–1472.
[15] I. Ragnemalm, "Billboards," http://www.computer-graphics.se/TSBK07-files/pdf13/8c.pdf.
[16] "Wtk," http://www.opengeospatial.org/standards/sfa.
[17] J. Guzzi, A. Giusti, L. Gambardella, G. Theraulaz, and G. Di Caro, "Human-friendly robot navigation in dynamic environments," in *Proc. of ICRA*, 2013, pp. 423–430.
[18] A. Giusti, J. Nagi, L. Gambardella, and G. Di Caro, "Cooperative sensing and recognition by a swarm of mobile robots," in *Proc. of IROS*, 2012, pp. 551–558.