Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference

Overwatch: An Educational Testbed for Multi-Robot Experimentation

D. Michael Franklin and **Lynne E. Parker**
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
dfrankl7@utk.edu and parker@eecs.utk.edu

Abstract

Educators who wish to engage their students in multi-agent experimentation and learning need an inexpensive multi-robot system that leverages existing equipment and open-source software. This paper proposes Overwatch as an inexpensive educational tool for teaching and experimenting in multi-robot systems. The interaction of multiple agents within a single environment is an important area of study. It is vital that agents within the environment perceive other agents as intelligent, acting within the environment as cooperative teammates or as competitive members of another team. To do so, the system must meet three goals: first, to allow multiple robots to communicate and coordinate; second, to localize within a shared global coordinate system; third, to recognize their teammates and other teams. The cost and scale of such experimental platforms places them outside the reach of many educational institutions or limits the number of agents that are interacting within the system (Liu and Winfield 2011). The goal of Overwatch is to create an experimental platform for multi-agent systems that is comprised of much smaller, albeit less capable, robots, many of which are prevalent in academic institutions already. Making use of available open-source libraries and utilizing lower cost robots, such as Scribblers, allows for experiments with many agents. This enables Overwatch to fit into the budget limitations of an academic setting. The Overwatch platform provides the Scribblers with global localization capabilities. This paper presents the system in detail and includes experiments to show its ability to localize, interact with other agents, and coordinate behaviors with these other agents. Additionally, the details to setup this system are also included.

Introduction

With the rise of larger and more complicated multi-agent systems comes the need for experimental testbeds that can test such systems. Existing solutions for large scale multi-agent interaction often involve great expense. In order to place these systems within budget for an educational environment Overwatch was created. Overwatch, a name derived from the military term for the overseeing command group, is a near real-time multi-agent coordination system that allows simple, limited robots to behave intelligently and interactively. The Overwatch toolkit, provided here, offers a practical solution using standard computational resources, inexpensive Scribbler robots, and open-source software.

Similar to the E-2 Hawkeye and UAVs utilized in modern combat and rescue operations, Overwatch takes the high-ground as an overhead camera that can observe the entire operation space and can identify and track the individual agents within this space, communicating independently with each one. The system can, by knowing each agent's identification, work with multi-agent teams as well. This system provides sensor-limited robots (i.e., those without internal navigation or localization capabilities) the functionality of sensor-capable robots within this defined area. These multiple agents, bearing markers and under the control of Overwatch, can perform higher-level strategic interaction. This high-level coordination is exhibited in the experiments documented here.

The paper is organized as follows: Related Work examines the foundational and comparable work, Methodology examines the system implementation, Experimentation and Results shares the proof-of-concept experiments and findings, Conclusions summarizes the results of this work, the Future Work lists the on-going objectives, and the Toolkit links to the software, videos and samples.

Related Work

The Institute for Personal Robotics in Education (IPRE) (IPRE 2007) is instrumental in developing the educational usages for the Scribbler robots, shown in Figure 1. They have developed curricula that utilize the Scribbler robots when mated with the Fluke board. Their research is one of the reasons that there are so many of these robots out in the academic arena. They have focused on students working one-on-one with a single robot for educational purposes. Overwatch takes these same robots and expands them into an experimental platform for multi-agent interactive systems.

In (Liu and Winfield 2011), the researchers propose a mostly hardware-based approach to creating a swarm of inexpensive robots (using the e-puck (EPFL 2011) robots (about $1100US)). Their system relies on additional hardware running on the robots and an expensive vision tracking system (the Vicon system (Vicon 2011), that starts around

$50kUS). These systems are out of the price range of most academic settings.

One of the largest and most complete examples of a similar system is that used in Robocup Soccer (Robocup 2012). Here the organizers use various sizes of robots (the SSL being closest to Scribbler size). They use two cameras to cover their large arena. The RoboCup SSL Vision system is specified only by protocols, so the robots and cameras vary in cost. A sample budget for a team, (Stevens Institute of Technology 2010), shows a cost of approximately $500US per robot and $950 per camera. This state-of-the-art system is solid and widely used, but it is more expensive and requires more custom building and coding.

For comparison, the Overwatch system utilizes standard HD webcams (around $100US) and Scribbler robots (from $100US to $200US).

Methodology

The Overwatch system seeks to create a multi-faceted testbed for experimentation in multi-agent systems that is modular and multi-tiered, offering flexibility and interoperability.

Robots

The agents in this scenario are Scribbler robots (Parallax 2012). The Scribbler, from Parallax, Inc., is a sensor-limited robot having three light sensors and two line-following sensors mounted on the bottom. This configuration is enhanced with the addition of the IPRE board (IPRE 2007) that mounts directly on the serial port of the Scribbler. This board provides IR sensors, a color camera, and a Bluetooth-over-serial connection to wirelessly connect to the Scribblers. The Scribblers have a limited ability to store programming information, so it is not practical for them to run their own independent code or manage their own inter-agent communication.

While there are several sensors on the robot, it does not have the ability to localize. It cannot return its x, y, z coordinates in real-space nor its θ heading. Because of this, it must rely on blind-reckoning (a poor approximation of dead-reckoning due to a lack of wheel-encoders) or limited guidance (e.g., the light from a flashlight or blob tracking) to move about. This is not true navigation - the robot is unaware of its current location, its location within the world, or its heading. Instead, this is reactive open-loop control, like obstacle avoidance, rather than true navigation where the robot is tracking along a map. In computer vision applications the limited view of the camera makes it challenging to track where the robot is, where other robots are, and where goal positions are. The overhead of such systems weigh heavily on these computationally simple robots. Externally-supplied information is needed to overcome this. Additionally, the Scribblers cannot reliably track straight lines (given the differential of the low-cost motors and a lack of wheel-encoders) which is only exacerbated as the battery runs down (see Failure Modes). The large thin plastic wheels are prone to slip. These motors have trouble initiating motion when the input is below a certain power threshold, so they require a surge to get started. All of these factors combine to show that the Scribbler, while a nice and inexpensive platform, cannot independently function as a reliable agent in a more complicated multi-agent system.

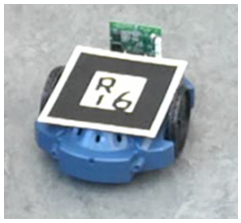

Figure 1: Scribbler with Fluke and AR Marker

In the standard Scribbler applications the robot is often connected to a single computer. For Overwatch, more than one Scribbler needs to be connected at a time. This requires understanding how the particular OS on the target computer handles Bluetooth connections. Each Fluke has its own ID, and as each system recognizes and connects to these boards over Bluetooth, it is assigned a MAC address. This becomes the system link to a communications port (e.g., "Com60" in Windows, or "/dev/rfcomm14" in Linux). These communications ports become the channels across which Overwatch communicates with each robot. The particulars of this setup are found in the linking material (with a step-by-step guide) at the end of this paper.

The Camera

Overwatch is enabled by having the high-ground view of the arena, so an overhead camera is used. The camera determines the size of the arena (based on its resolution and field of view). For this toolkit a single overhead camera is used, but the system could utilize multiple independent cameras and merge their perceived worlds into one larger world to allow for an increased arena size while maintaining the same resolution. The camera used for these experiments was the Microsoft HD (720p) webcam (Figure 2). The webcam is not fixed and can be relocated to an overheard position, an angled position, or even a point-of-view (POV) position without compromising the performance of Overwatch.

Figure 2: MS HD Webcam

In choosing a camera the effective resolution of the optics must be considered. Any camera can be used, but it must provide images with sufficient detail to recognize the markers on the agents.

Marker Recognition

There are many methods for marker recognition. The most common, by far, is ARToolkit (ARToolkit 2008). This toolkit is built upon the required OpenCV platform (which supplies the computer vision framework for interacting with the graphics hardware). Together, these two pieces of software allow the computer to interpret video, check for AR markers, and draw graphics on those markers. The original ARToolkit is no longer being developed (by the original creators) and has been supplanted by ARToolkit Plus. This newer version is a significant improvement over the original but it is less compatible across platforms. As a result, Overwatch is designed around ARToolkit so that it is maximally compatible. There are other AR packages available that may be of interest to educators specializing in other languages. The first is PyARTK, which works in Python. This toolkit was intended to be a port of ARToolkit that would work natively in Python. The next is OpenCV AR. This toolkit is a native set of AR tools inside of OpenCV. For those interested in mobile development, there are also a few other lite versions of AR, like Aruco and Augment. While these may add functionality or expandability to Overwatch, they do not have the processing power and communications ability to control multiple robots and coordinate the communications necessary for the system.

Computing Environment

The computing environment also presents challenges to the system. The robots have a software platform, called Myro, that simplifies communications and commands with the Scribblers. The Myro software is written in Python, though it has also been ported to C++ (Hoare et al. 2011). The ARToolkit software is written natively in C, while the interaction with the arrays of robots and arrays of marker patterns is better facilitated in C++ (especially when considering these elements as objects in a multi-threaded environment). The computing environment must also be robust enough to support programming with multiple threads, large memory transactions, high-throughput algorithmic calculations, and high-speed graphics capabilities. Another consideration is the operating system. As mentioned previously, Windows greatly simplifies the coordination of multiple Bluetooth-enabled Scribblers, but both it and Linux (and the MacOS) provide solid connections. With all of these considerations in mind, the goal was to have Overwatch be available without constraint, so it will work in Python, C, or C++, in Windows, Linux, or on the Mac. The code requires slight alteration in the device names of the robots (e.g., from "COM60" to "/dev/rfcomm14"), but is portable across platforms.

Overwatch System

Overwatch creates an arena that is defined by the resolution and field of view of an overhead camera. In our implementation the camera is mounted to the ceiling at a height of 9ft in the lab. This gives a field of view of about 15ft by 12ft and thus defines our arena. For these experiments the markers were produced from the ARToolkit. The augmented reality tools provide marker recognition and can be used to determine the marker's (and thus, the agent's) position in a relative camera space. The system creates an array of robot objects. Because of this, any student familiar with coding in Myro for a single robot can now simply loop through the robot array to talk to all of them. This provides an easy transition to multi-robot systems.

In Overwatch the target and goal information for the world is calculated centrally. Specific move commands are then sent to each Scribbler. This shifts the sensor-limited robot into simulating a sensor-capable robot. While this implementation is centralized, the system can work passively in a decentralized mode by only providing information that each agent could normally have. The system works as follows: first, the system creates a data structure that holds the robot objects. Second, it creates a data structure that holds the various marker patterns that will identify the robots, targets, and other objects within the arena. Third, Overwatch uses ARToolkit to recognize markers by searching the current video frame for large black squares. Any such square will get the attention of ARToolkit. Once the target is acquired, it is inspected for the predetermined patterns and is matched on the greatest matching likelihood (Figure 1 shows such a marker). Recognized markers are entered into the visible marker array. Once all markers within the frame have been located, this array is returned to the code for processing. This processing is integral to Overwatch. Fourth, each marker is compared with its target marker to determine location and orientation. Fifth, the markers (i.e., the robots) are checked for collisions with targets or other markers and the traffic is directed according to the current overall goal (i.e., strategy) of the system. This also establishes communications within the teams or across teams through the Overwatch central command. This communication ability provides functionality, such as organizing the agents to avoid collisions with teammates based on priorities, changing the target being sought, or implementing a new strategy based on the current information from the environment.

The process of checking each marker for location and orientation is complicated by the various coordinate systems. First, the robot and target markers are recognized in the camera coordinate system (Figure 3). The orientation in this space is $\pm 180^\circ$. Next, the robot's localization information has to be transformed into the marker coordinate system (Figure 4). Here the angle to the target and the bearing angle have to be calculated and corrected, as shown in Equations 1 - 5. Finally, to achieve any reasonable navigation within the real world each of these markers has to be located in a shared real-world coordinate system (Figure 5).

In this final space the angles are measured in a full circle, 360°, as opposed to the split $\pm 180^\circ$ system of the camera and marker coordinate system. This makes navigation identical for each marker, so they can now understand where they are and where any other markers are. The robots can now navigate to any marker. Overwatch calculates this quickly (around 15Hz), so the system is able to correct robot 'wandering' (where the motors are misaligned or aren't calibrated), handle moving targets, and recover from collisions that move the robot off its path.

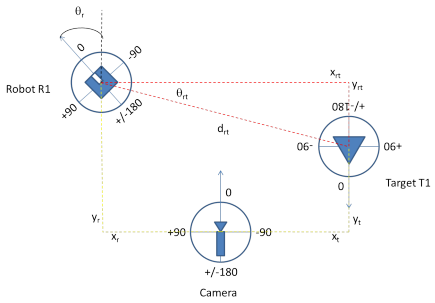

Figure 3: Camera-Space

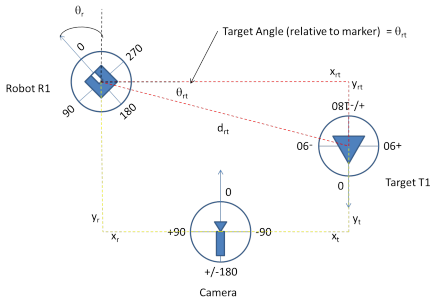

Figure 4: Marker-Space

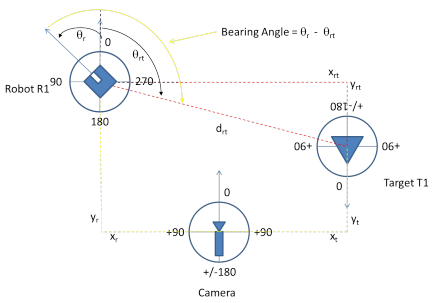

Figure 5: Real-Space

$$\theta_{rt} = \tan^{-1}\left(\frac{y_{rt}}{x_{rt}}\right) \tag{1}$$

$$\theta_{r(corrected)} = 2\pi + \theta_r \tag{2}$$

$$\theta_{bearing} = \theta_{r(corrected)} - \theta_{rt} \tag{3}$$

$$\theta_{bearing} = \theta_{bearing} \mod 2\pi \tag{4}$$

$$\theta_{bearing} = \begin{cases} \theta & \text{if } \theta < \pi \\ -(2\pi - \theta) & \text{otherwise} \end{cases} \tag{5}$$

Experiments and Results

There are three claims that need to be tested. The first is that Overwatch is accurate and consistent in localization and navigation. The second is that it can control multiple robots simultaneously and efficiently. The third is that Overwatch can coordinate strategic behavior within a team of robots. One additional goal is to show scalability and robustness through each of these experiments.

The first experiment is navigation and localization using Overwatch vs. blind-reckoning. In this experiment the blind-reckoning robots are driven, at 100% velocity, forward for 6s, turned right for 0.45s, then forward for 3s, then another right turn for 0.45s, then forward for 6s. This should result in the three sides of a rectangle that ends back in line with the target, displaced by about 110cm. The target zone was determined in a way that favors the blind-reckoning: three runs were made as directed, then the average finish point was used as the target location. Once the target was placed at this location, this experiment was run 7 times for each control method.

The accuracy results show, in Figure 6, that the blind-reckoning method averaged 435.86mm, with a σ of 149.23, from the center of the target. The Overwatch runs averaged only 83.14mm, with a σ of 7.06, from the center. It should be noted, as indicated by the line in the graph, that the center of the target is 76mm from the edge on average, so the Overwatch navigation ended, as directed, at the edge of the target. These results are a specific example shown by this case to demonstrate this issue with blind-reckoning.

An examination of the traces shown in Figures 7 and 8 shows visually what the numbers state. These traces show

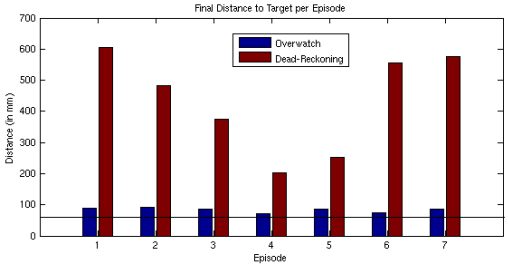

Figure 6: Final Distance to Target: Blind-Reckoning vs. Overwatch

that the blind-reckoning pathways were inconsistent across each run, whereas the Overwatch pathways were consistent. Overwatch will be consistent to any given target without respect to the number of waypoints because each target acquisition and navigation is handled independently. Any accumulated tracking error will thus be erased in seeking the next target or waypoint. Seeking the target after one waypoint or after ten waypoints will bring about the same results.

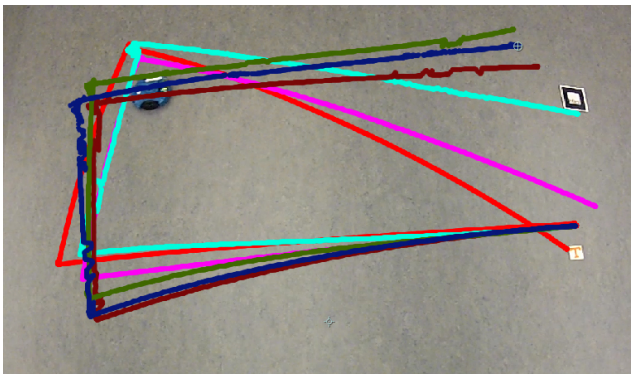

Figure 7: Blind-Reckoning Traces

These experiments show at least two critical points: first, they demonstrate the overall inconsistency of blind-reckoning in sensor-limited robots (these runs were with fresh batteries; the same trials with older batteries were even less consistent); second, they show the difference that Overwatch can make not only with accuracy (distance to target)

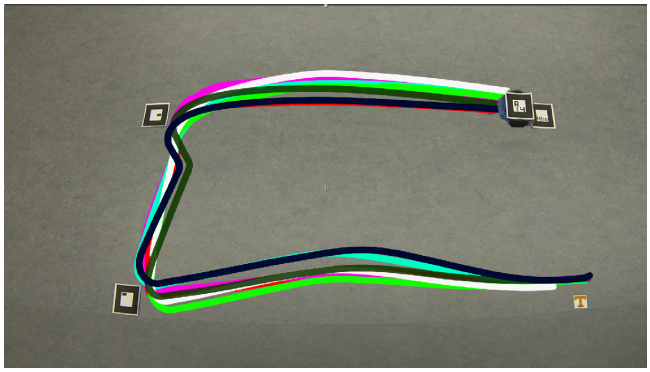

Figure 8: Overwatch Traces

but with consistency (σ of distances to target) and proper path planning.

In the second experiment, multiple robots were tested on multiple teams for scalability and reliability. In these experiments, the system was running in C++ and Python, in Windows and in Linux, on a Core2 T7200 laptop running at 2GHz with 2GB RAM. To test the scalability, multiple robots were run from one machine. For this experiment an arena was constructed that enclosed two teams of 6 robots each, for a total of 12 robots, running an obstacle avoidance program.

Each of the robots was moving and being controlled independently by the system. In this scenario, the robots were passing back their IR sensor data to Overwatch and it was interpreting this sensor data and sending back steering instructions. These robots were not being tracked yet, just being controlled to test the throughput of the system and the control methods. Further testing of this scenario showed that the system works with up to 7 robots per machine on 3 separate machines simultaneously for a total of 21 robots interacting in the arena. From a reliability perspective each robot stayed under system control, even when the robot itself was having issues (see Failure Modes).

The video frame rate, the speed to recognize markers, and the speed of communication form the limiting factors for the cycles per second (Hz) of the system. On the computer mentioned the system averages 14.86Hz with σ of 2.35. The high was close to 25Hz, with the low around 12Hz. There was also no noticeable signal depreciation with the Bluetooth communication saturation as the experiment scaled up to 21 robots as the speed hovered around this same average. The system runs almost identically in round-robin, where each robot is given instruction one at a time, and in multi-threaded operations. While multi-threaded performance may be better over time, it is important to note that students and researchers working with Overwatch do not have to program in threads to use the system. The scalability of the system is supported by no appreciable decline with 21 total robots running.

The third experiment was designed to show that Overwatch can coordinate the behavior of a team intelligently. The scenario envisioned was one of a squad of 5 robots where 3 of them are seekers and the other 2 are 'on-station' as defenders to block opponents. The 3 seekers are tasked with finding the target and converging there. When the seekers all arrive at the target the entire team should disperse. Figures 9, 10 and 11 show the frames from the video of this experiment. The initial starting positions of the robots are shown in Figure 9. Figure 10 shows the traces of the robots from the starting positions to the point at which the seekers converge on the target. Finally, Figure 11, shows the traces as the robots disperse. The seeker robots converged within the target zone 92% of the time, and the orbiting defenders stayed on station 96% of the time. This shows that these simple robots can exhibit coordinated behavior.

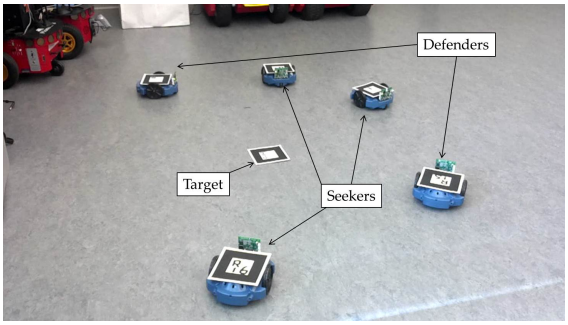

Figure 9: 5-Robot Trial: Initial Position

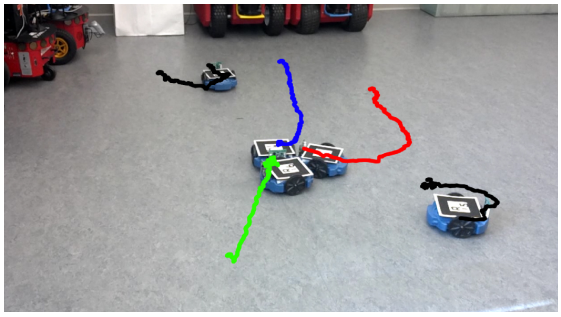

Figure 10: 5-Robot Trial: Seek and Defend

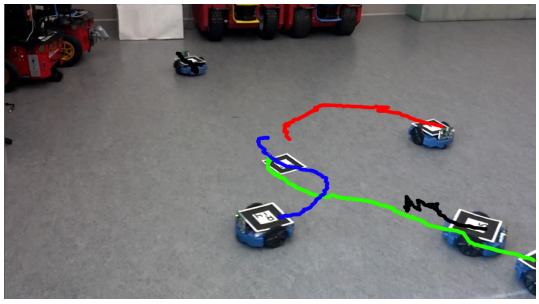

Figure 11: 5-Robot Trial: Disperse

Failure Modes

In these experiments, and in general experience, there are several failure modes that occur. First, Scribblers are prone to motor and alignment wandering, general inconsistency over the life of the battery, and the 'kidnapped robot' syndrome (where a robot moved by an outside force is unaware of the change in its location or orientation). Overwatch helps overcome these alignment and dislocation issues by using fast tracking (around 15Hz) to re-localize the robot in its new orientation and position - an important feature for users who need their robots to perform with robustness and consistency.

The Bluetooth connection can be problematic in syncing up, but once it is connected it performs well. Most issues are resolved by resetting the robot and/or clearing the channel. A built-in re-sync could overcome this initial issue.

There are issues where the tracker loses the markers, but it generally catches them again within a few frames. If this occurs too close to the edge of the video frame the robot will head off course and not be recoverable. To resolve this there are a few implementations. First, the robot movements can be stored with a small memory. When this robot's marker is lost it simply issues the reverse commands to bring the robot back in-bounds. Second, the robot can plot a curving path from its last known coordinates to the target coordinates. In addition, there can be an issue when a robot covers the target's marker. As a general guideline, this can be overcome by using safety zones (stop short at a certain distance) to halt the robots.

Conclusions

When selecting the robotic platform, the software, and the computers, a conscious effort was made to find the lowest common denominator. The system performs even better on higher-performance computers. Newer versions of the software promise faster marker recognition, increased accuracy, and more intricate patterns in the markers. As a result, the performance should improve with these upgrades.

Additionally, by only changing the actual robot connection and move commands (contained in functions in the code), it could be easily reconfigured to work similarly with other robots needing this kind of performance upgrade (e.g., e-puck (EPFL 2011), Scribbler2, Mindstorm NXT, BOE-BOT, any other custom robots). For example, there is a new version of the Scribbler, the Scribbler 2, that features an improved processor and better sensors. While this newer version has many improvements, it still does not have the abilities that Overwatch provides. It will still work with Overwatch (in fact, Parallax notes that it will still use the same IPRE board used in these experiments).

These results show that, with Overwatch, it is possible to take low-cost, sensor-limited robots and perform real-world experiments with them wherein they act like higher functioning, and more expensive, robots. This platform leverages these small, prevalent robots into stand-in's for the much more expensive and less common robots in research today. Overwatch thus offers researchers and educators a robust and scalable tool to explore the space of multi-agent systems within the constraints of their current budgets. In fact, if the robots are already present, and a camera available, there is no cost to implementing this system.

Additionally, it is noteworthy that Overwatch, because it has total knowledge of all agents, communications, strategies, etc., can then be constrained to simulate many different environments. It could isolate the team communications from each other so that they are not aware of the internal processing of the other team's actions or simulate a Master Controller where all information, communication, and strategy is centralized. Alternately, none of the agents could have any internal information about any other agents. Many different paradigms can be configured and enforced within the Overwatch system.

Future Work

For future work, this toolkit can be expanded in scale, mixture of robots, and control strategies. It will be used as a testbed for strategy implementation, multi-agent interaction, and strategy inference and learning. This tool can be expanded to reach a broader audience with the inclusion of more marker-related technologies, testing with various additional robot manufacturers, and with higher-performance computational power.

Toolkit

The toolkit, with step-by-step installation instructions and links, the hardware setup (with the detailed Bluetooth instructions and examples), and sample Overwatch control software, along with the accompanying videos are located at: http://dilab.eecs.utk.edu/Overwatch

References

ARToolkit, I. 2008. ARToolkit, University of Washington, HITLab, http://www.hitl.washington.edu/artoolkit.

EPFL. 2011. Ecole Polytechnique Federale de Lausanne, http://www.e-puck.org/index.php.

Hoare, J.; Edwards, R.; MacLennan, B.; and Parker, L. 2011. Myro-C++: An Open Source C++ Library for CS Education Using AI. *Proceedings of the 24th International Conference of the Florida Artificial Intelligence Research Society (FLAIRS 2011)*.

IPRE. 2007. Institute for Personal Robots in Education, www.roboteducation.org.

Liu, W., and Winfield, A. F. 2011. Open-hardware e-puck linux extension board for experimental swarm robotics research. *Microprocessors and Microsystems* 35(1):60 – 67.

Parallax, I. 2012. Scribbler Robot, from Parallax, http://www.parallax.com/ScribblerFamily/tabid/825/Default.aspx.

Robocup. 2012. Robocup Soccer Tournament, http://www.robocup.org.

Stevens Institute of Technology. 2010. RoboCup Soccer SSL: Platform Design Phase VI Final Report. *Report for Senior Design Project* 1(1):1 – 110.

Vicon, I. 2011. Vicon Inc., http://www.vicon.com.