A Multi-Vehicle Framework for the Development of Robotic Games: The Marco Polo Case

Brent Perteet, James McClintock, and Rafael Fierro

Abstract— **This paper presents a multi-vehicle platform and framework for robotics education and research. The system is designed as an educational tool for introducing children to engineering and robotics and is composed of hardware and software components that allow users to easily design and implement sophisticated robotic behaviors. We formally introduce the robotic game Marco Polo as a problem that mimics the pursuit-evasion game often played by children in swimming pools. Specifically, we address the question of finding a pursuit strategy under the condition of intermittent communication. Finally, we present an implementation of the Marco Polo game.**

I. INTRODUCTION

As robots become more ubiquitous in everyday life, humans must learn to interact and work with these machines. University engineering curricula have begun to incorporate robotics as a practical application of theory. However, additional work is required to introduce younger students to the problems and challenges that robotics present. The main contributions of this paper are twofold: (i) a framework that has been created as a tool for educational robotics development; and (ii) a game developed for the platform called Marco Polo. The original application of the framework was for a project called "Robotic Games" [1] which has a goal of introducing elementary school children to robotics and encouraging their interest in the science, technology, engineering, and mathematics fields. As the framework evolves, it is also proving to be useful for studying other topics such as human-robot interaction [2] and motion planning. This result occurs because many of the same features that are needed in a teaching platform are also useful for research in these areas.

The remainder of this paper is organized as follows. We begin by discussing relevant related work in Section II. In Section III we present key features of the framework. In Section IV we describe two of the games that have been developed on the platform. Section V presents a case study on Marco Polo including a discussion of a special type of the pursuit-evasion problem that must be solved to implement the game. In Section VI we discuss future work. Finally, concluding remarks are provided in Section VII.

This work is supported in part by NSF grant CAREER #0348637, and by the U.S. Army Research Office under grant DAAD19-03-1-0142 (through the University of Oklahoma)

B. Perteet, J. McClintock, and R. Fierro are members of the MARHES Lab, School of Electrical & Computer Engineering, Oklahoma State University, Stillwater, OK 74078-5032, USA (e-mail: {brent.perteet, james.mcclintock, rfierro}@okstate.edu).

II. RELATED WORK

Many researchers are actively working in the area of educational robotics, which focuses on motivating and encouraging students to explore opportunities and careers in science, technology, engineering, and mathematics. Researchers have used robotics and developed materials to introduce engineering in general to elementary and secondary students [3]. Additionally, several researchers have participated in competitions such as RoboCup and RoboFlag, which involve robots playing various games including soccer and capture-the-flag [4].

With respect to the robotic game, *Marco Polo*, a great deal of work has been done related to pursuit-evasion and target tracking. The authors in [5] discuss basic motion planning and control strategies for multi-robot systems. The Marco Polo game has motivated a multi-robot localization problem introduced in [6]. In [7], the authors develop a decentralized motion coordination algorithm for tracking groups of dynamic targets. Strategies to search for moving targets in a two-dimensional plane are considered in [8], [9]. In [10] the authors show that a pursuer can detect an arbitrarily fast evader using a randomized strategy. The evader can be captured by two pursuers solving a *lion-and-man* problem assuming that at least one pursuer is as fast as the evader.

III. THE ROBOTIC GAMES FRAMEWORK

The Robotic Games framework represents the combination of several hardware and software elements that are integrated with a flexible software architecture.

A. Hardware Components

The primary hardware components of the platform include an Evolution Robotics Scorpion Robot and a Toshiba M400 Tablet PC. The architecture also makes use of desktop computers and other common devices such as a wireless router and PDA.

1) The Scorpion Robot: The robotic platform used to implement this framework is the Scorpion Robot from Evolution Robotics [11]. The robot features a variety of simple sensors for gathering information about the environment including infrared range-finding sensors, a contact-sensing bumper, a camera, and high-resolution optical encoders on the motors. All sensors except for the camera connect to a central controller called the Robot Control Module (RCM). A servo controlled differential drive system, which is also controlled using the RCM, provides locomotion. High-level control is accomplished by a Tablet PC that is mounted on the robot and connects to the RCM and camera by USB. Other USB devices such as a wireless joystick or other USB sensors may be connected to the Tablet PC as well.

2) A Tablet PC Computer: A Toshiba M400 Tablet PC is mounted on the Scorpion Robot to provide high level processing and control functions. Though the Scorpion may be used with any suitable notebook computer, a Tablet PC has features that are particularly useful on a platform designed for education. Because the screen rotates, the display is visible while the robot is running such that a 3D face that displays emotions and reacts to the environment might be rendered on the screen. This feature allows the robot to interact with humans in a more natural way, which enhances its capabilities as a educational tool. Also, the Tablet PC supports pen input, which provides another method for interacting with the robot.

3) Other Hardware: Many applications of this framework benefit from other common devices. Specifically, the games that we have developed use desktop computers, wireless joysticks, a wireless access point, and a PDA. The access point allows for centralized communication and enables the robots to communicate with desktops for user input. In some games, desktops are not needed but a mechanism to coordinate all of the robots (start, stop, pause the game, etc.) is required. In these instances, a PDA with wireless communication capabilities works well.

B. Software Components

This framework combines the powerful robotics processing and control algorithms available in the Evolution Robotics Software Platform (ERSP) with the Trolltech Qt application framework. These two libraries are integrated using an architecture that we designed to allow developers to easily create new games and demonstrations.

1) The ERSP Library: The ERSP library is an extensive software platform for developing high-level controllers while abstracting the developer from the underlying hardware. The platform consists of an advanced API for the C++ programming language. The API is available for both the Microsoft Windows and Linux platforms. We have chosen to use the Windows version for development of the Robotic Games.

The ERSP API includes a comprehensive set of advanced algorithms particularly for vision and navigation. At the center of the vision module is the ViPR^{TM} (Visual Pattern Recognition) algorithm. Among other things, this algorithm provides behaviors which handle object recognition, motion flow, and color segmentation which are useful for detecting objects and their pose, movement, and skin color. The most significant component of the navigation module is the implementation of the vSLAM^{TM} (Visual Simultaneous Localization and Mapping) algorithm [12]. This algorithm uses input from the wheel encoders and camera to accomplish the simultaneous localization and mapping function. The navigation module also contains support for path planning, obstacle avoidance, exploration, and population of an occupancy grid map.

2) The Qt Application Framework: Qt is an open-source, C++ application framework that is produced by Trolltech. The library supports several platforms including Windows, Linux, and MacOS and provides a rich set of classes for application development. For example, the Qt library contains object models for XML processing, multi-threading, networking, GUI design, and plug-in development. Qt also allows window based GUIs to be developed easily. Classes for common window interface controls such as menus and buttons may be combined to create advanced, event-driven interfaces.

3) The Robotic Games Library: The Robotic Games library consists of a set of software modules that are designed to manage games which conform to the Robotic Games software architecture. Specifically, games are designed using C++ with the support of the Qt and ERSP libraries. The architecture uses Qt's plug-in capabilities to manage the games. Thus, new games should provide a plug-in interface which includes a set of common methods (start, stop, pause, etc.) The software architecture defines and manages two categories of hardware: Displays (desktop computers); and Robots (Scorpion and Tablet PC). As part of the plug-in interface, each game specifies the number of displays and robots that are required.

The Robotic Games library consists of two software components: the Game Manager and the Coordinator. A block diagram of the system architecture is shown in Figure 1. The Game Manager runs on robots and displays. This application is able to load and configure games that conform to the plug-in interface. The Coordinator, on the other hand, may run on either a desktop computer or on a PDA. This application is used by game referees to manage the games. Communication between the Coordinator and Game Managers is accomplished with simple text messages which are sent on top of the standard UDP/IP and TCP/IP networking protocols. Upon execution and at the referee's request, the Coordinator queries the network using the UDP protocol for robot and display client discovery. Clients running the Game Manager listen for these discovery queries and respond by broadcasting their presence on the network. The Coordinator receives these responses and registers each robotic node in an internal database. Using the Coordinator application, the referee may select a game for the system to run. The Coordinator then queries the game's plug-in for a list of configuration parameters and dynamically creates forms to configure the game. Finally, the referee may select a button to begin the game and the Coordinator instructs each Game Manager to configure and begin execution.

IV. Game Examples

Using the framework discussed, several games have been developed for the Robotic Games project. The games are designed to be educational, demonstrating various aspects of the state-of-the-art in robotics and engineering.

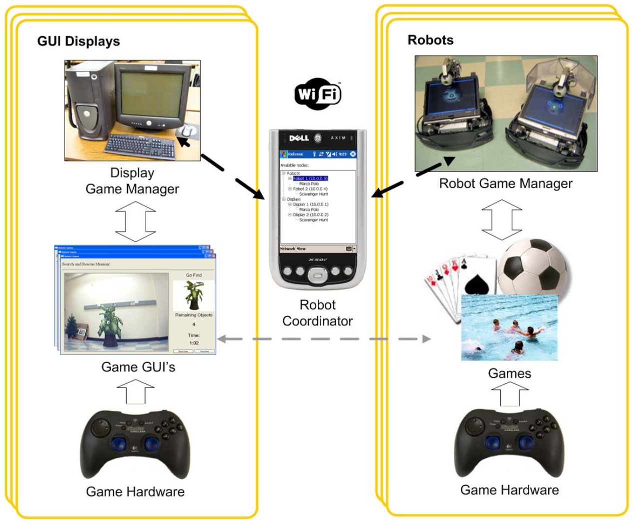

Fig. 1. The Robotic Games System Architecture.

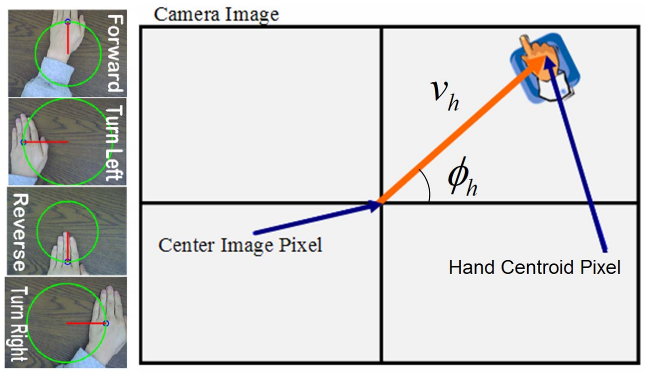

Fig. 2. The Obstacle Course game uses hand movement for robot control.

A. Obstacle Course

In the Obstacle Course game, teams of children guide a robot through a game area filled with obstacles. Obstacles might include cones to traverse, traps to avoid, or objects to locate. The goal of the game is to complete the course as quickly as possible. Each robot is given a limited amount of energy E. The game artificially limits the robot's speed to

$$v_\ell = E v_{\max}/E_{\max},$$

where $E_{\max}$ and $v_{\max}$ are constants representing the robot's maximum energy and linear velocity respectively. As the robot moves, its energy is depleted as

$$\dot{E} = -(v/v_\ell)^2,$$

where v is the robot's current velocity. Thus, driving as fast as possible, $v = v_\ell$, reduces the robot's energy the fastest. When the robot's energy is depleted, it must "rest." While resting, the robot cannot move, but its energy is restored at a constant rate. To make the game more interesting, we have integrated a vision based hand recognition system to maneuver the robot. This system uses the HandVu library [13] to determine the pixel location of a hand's centroid within a camera frame. A vector v_h is calculated from the center of the image to this point. This vector is first normalized to the display size and then the angle ϕ_h is calculated as shown in Figure 2. To control the robot, values for its linear speed v and angular speed ω are taken as

$$\begin{aligned} v &= v_h \sin\phi_h \\ \omega &= v_h \cos\phi_h. \end{aligned}$$

This mapping is designed to simulate joystick control. For example, when the hand is at the top of the image and horizontally centered, the robot moves forward. If the hand is on the left side of the image and vertically centered, the robot spins left in place.

B. Marco Polo

Marco Polo is a pursuit-evasion game in which the pursuer receives information about the evaders' location in random time intervals. The game uses two Scorpion robots operating in a leader-follower configuration. The pursuer robot is autonomous while the evader robot is controlled by a participant using a wireless joystick. The goal for the human player is to evade capture by the autonomous pursuer robot for as long as possible. In this game, the evader robot is defined as "caught" when the separation distance between the two robots' centers falls below a threshold value for some length of time. At the beginning of the game, the two Scorpion robots are placed together at random locations within the game area. The player is given a wireless joystick, informed about the rules of the game, and instructed begin moving. Once the game is running, the evader robot immediately begins responding to commands from the player's joystick. At a constrained random interval, the pursuer robot announces "Marco" over its laptop computer's speakers and uses the computer network to request the position of the evader. Once the pursuer receives the evader's location for the first time, it begins moving. Following this, the communication process described above is repeated. The human player, generally attempts to maneuver around the pursuer robot in such a way as to maximize the distance between the two. The strategy for a pursuer with incomplete information based on intermittent cooperation is a challenging research problem that is relevant to this game.

V. A Marco Polo Case Study

In this section, we present a formal definition of the Marco Polo problem together with a pursuit strategy when the evader is constrained to straight-line motion. We also describe an experimental implementation of the game using the framework discussed above.

A. Problem Statement

From a game design standpoint, the goal of Marco Polo is to capture a group of intelligent evaders as quickly as possible using a team of autonomous pursuers that have intermittent knowledge of the evaders' locations. The remainder of this section presents a formal definition of this problem including the assumptions that will be made and the terminology that will be used to describe the game.

Let the game area be referred to as $\mathcal{S}$ and its boundary as $\partial\mathcal{S}$. We assume that $\mathcal{S} \subset \mathbb{R}^2$, and that it is convex and bounded. Associated with $\mathcal{S}$ is a fixed coordinate frame $\mathcal{F}_\mathcal{S}$. The pursuers or mobile sensor agents are nonholonomic vehicles that can be modeled using the unicycle model

$$\begin{aligned} \dot{x}_p^i &= v_p^i \cos\theta_p^i, \\ \dot{y}_p^i &= v_p^i \sin\theta_p^i, \\ \dot{\theta}_p^i &= \omega_p^i, \end{aligned} \tag{1}$$

where $(x_p^i, y_p^i, \theta_p^i) \in SE(2)$ is the position and orientation of pursuer i with respect to $\mathcal{F}_\mathcal{S}$ and $u_p^i = [v_p^i \quad \omega_p^i]^T$ represents the input to pursuer i. In addition, pursuers are bound by certain kinematic and dynamic constraints. Specifically, we assume that the maximum linear velocity $V_{p\max}$ and angular velocity $\Omega_{p\max}$ of all pursuers is known.

The model of the evader or target agents is given by

$$\begin{aligned} \dot{x}_\tau^j &= v_\tau^j \cos\theta_\tau^j, \\ \dot{y}_\tau^j &= v_\tau^j \sin\theta_\tau^j, \end{aligned} \tag{2}$$

where $(x_\tau^j, y_\tau^j) \in \mathbb{R}^2$ is the position of target j, v_τ^j is uniformly distributed in $[0, V_{\tau\max}]$ and θ_τ^j is uniformly distributed in $[0, 2\pi)$. In our experimental implementation, the evader is tele-operated by a human. In this case, uniform distributions may not be the best choice. This human-robot interaction is a topic of further research.

Obstacles may be placed in the game area as long as certain conditions are met. Suppose the j^{th} obstacle obstructs a certain region in $\mathbb{R}^2$ denoted as $\mathcal{O}_j$. Valid obstacles will meet the following conditions: (i) $\mathcal{O}_j$ is a convex region in $\mathbb{R}^2$, (ii) $\mathcal{O}_j \subset \mathcal{S}$, and (iii) the minimum distance from any point $p \in \mathcal{O}_j$ to a point $q \in \mathcal{O}_i$ $(i \neq j)$ or to a point $s \in \partial\mathcal{S}$ is greater than W. Here W refers to the diameter of the smallest circle which can completely surround the largest participating robot's projection onto $\mathbb{R}^2$. This ensures that there are no regions in the game area that can be reached by some robots but not by others.

Let N be the total number of pursuers and M the total number of active targets that are participating in the game. All targets are considered active at the beginning of the game. Once a target is caught, it becomes inactive. Inactive agents must either move to a position where they represent valid obstacles and remain still or be removed from the playing field $\mathcal{S}$. Let p_i $(\tau_i) = [x_{p(\tau)}^i \quad y_{p(\tau)}^i]^T \in \mathbb{R}^2$ refer to the position of the i^{th} pursuer (target) robot with respect to $\mathcal{F}_\mathcal{S}$. When there is no danger of confusion, p_i (τ_i) may simply be used to refer to the robot itself as well. The notation $\mathcal{P} = \{p_1, p_2, p_3, \ldots\}$ is used to refer to the set of all pursuers and $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \ldots\}$ refers to the set of all targets. All agents in $\mathcal{P}$ and in $\mathcal{T}$ must have initial positions within $\mathcal{S}$ and cannot leave $\mathcal{S}$. Let e_{ji} be the Euclidean distance from the j^{th} target position, τ_j, to the closest pursuer, i.e., $e_{ji} = \min d(\tau_j, p_i)$. Then the pursuer i is said to *capture* the target j when $e_{ji} < \varepsilon$. The threshold value ε is called the *capture threshold* for an interval Δ_c called the *capture timeframe*.

The pursuers receive information about the position of each target intermittently. Let σ_i be a random variable representing the time period between communications for the i^{th} target. At the instant of communication from target i, its exact position within $\mathcal{S}$ is known by the pursuer agent. Following this, the target may continue to move but the sensor agent receives no updated information until the next communication from target i. Based on the previous discussion, the problem in Marco Polo can be stated as follows:

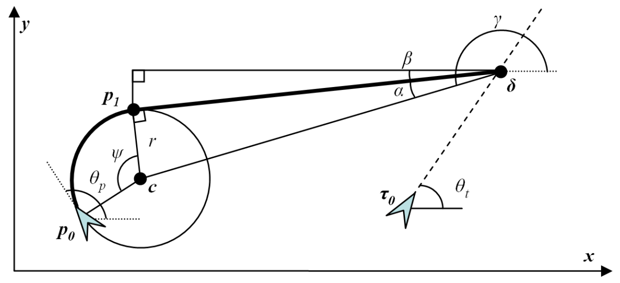

Fig. 3. Interception of target

Problem 5.1: Given a set $\mathcal{P}$ of N pursuers and a set $\mathcal{T}$ of M target robots within a specified game area $\mathcal{S}$ and meeting all of the assumptions outlined in this section, choose values $u_p^i = [v_p^i \quad \omega_p^i]^T$ for all pursuers in $\mathcal{P}$ subject to the pursuers' dynamic and kinematic constraints which minimize the time t_c required to capture all targets in $\mathcal{T}$.

B. Pursuit Strategy

In this section we propose two very simple potential pursuit strategies for the single pursuer, single target pursuit problem described in the previous section. These solutions include (i) moving to the last known target location and (ii) assuming the target is moving in a straight line with constant velocity and intercepting along that line.

The first solution moves the pursuer to the last known point of the target. This strategy may be most useful when the pursuer is far from the target and the communication interval is small enough. That is, this strategy would be employed to move the pursuer close enough to the evader to employ a more sophisticated strategy. The second approach shown in Figure 3 is a simple solution based upon the geometry of the problem taking into account the kinematic constraints of the pursuer. This approach is strongly dependent on the controller used to guide the nonholonomic vehicle to a target waypoint. In this case, the assumed controller is based on the potential field controller (PFC) presented in [14]. The strategy also assumes that the target is moving with both constant velocity and heading. The strategy attempts to intercept the target at a point δ which is a function of the interception time t_c which is the time for the pursuer and target to reach that point. The initial states of the pursuer and target are $p_0 = (x_p, y_p, \theta_p)$ and $\tau_0 = (x_\tau, y_\tau, \theta_\tau)$, respectively. The interception point is defined as

$$\delta(t_c) = \begin{bmatrix} x_\tau + t_c v_\tau \cos\theta_\tau \\ y_\tau + t_c v_\tau \sin\theta_\tau \end{bmatrix}, \tag{3}$$

and the time to interception becomes

$$t_c = \frac{r\psi(t_c) + \|c - \delta(t_c)\| \cos\alpha(t_c)}{v_p},$$

where the distance traveled by the pursuer is the distance along the arc p_0p_1 plus the straight line distance between p_1 and δ. The arc radius is the same as the turn radius of the pursuer and is defined as $r = \frac{v_p}{\omega_p}$. Using the geometric definition, the interception point δ in (3) which is passed to the controller may be solved numerically using an iterative algorithm such as that presented in [15] or a Newton method.

C. Simulation Results

In order to test the performance of these individual strategies, a Matlab/Simulink simulator is developed for the single pursuer, single target case. The simulator is able to use a joystick input such that an intelligent driver may control the target agent.

Several basic maneuvers were performed using the joystick and a simple waypoint extraction algorithm. These maneuvers, which are performed by the target in each simulation, include (i) a "figure 8", (ii) a spiral, and (iii) random walk. For the simulations, the pursuer and target agents are given maximum linear and angular velocities with the pursuer's greater than the target's. Also, in order to guarantee that the simulation ends in finite time, the velocity of the target is reduced with time as $v_\tau = v_{\tau 0} e^{-\frac{t}{\mu}}$ where μ is the time constant and $v_{\tau 0}$ is the target's initial maximum linear velocity. The broadcast interval of the target's position to the pursuer is a random variable σ whose distribution is uniform in $[\sigma_{\min}, \sigma_{\max}]$.

A total of 100 simulations were performed for each strategy/maneuver pair. Time to capture is the metric used for comparison in each case. Simulation data is omitted here due to space limitations; however, in each of the maneuvers by the target, the line interception strategy outperformed the more native strategy of moving to the last known location.

D. Experimental Implementation

Marco Polo requires two robots; however, the evader robot is relatively simple using only a wireless joystick to set the robot's velocity inputs. Since the pursuer robot is autonomous, its implementation is more challenging requiring an answer to the three fundamental questions in autonomous mobile robotics: 1) Where am I? 2) Where am I going? and 3) How do I get there? This experimental implementation as well as the pursuit strategy is shown in the paper's accompaning video.

1) Where am I?: Section V indicates that the pursuer robot receives information about the location of the evader at a random interval. The Scorpion's encoder based odometry alone cannot accurately provide this information over long distances. Marco Polo uses the vSLAM$^{\text{TM}}$ algorithm that is included with ERSP. The algorithm extracts distinguishing features from camera images to correct odometry error. To ensure that robots communicate in the same reference frame, each use a common vSLAM$^{\text{TM}}$ map. Thus, this common map of the game area is generated first. At the beginning of each game, all pursuers wander around the room attempting to determine their positions in the game area by watching for scenes in the map. After the pursuer determines its location, it begins tracking the evader. Evaders also watch for scenes in the map as game participants maneuver them. In summary, the vSLAM$^{\text{TM}}$ module provides a method for accurately tracking each robot's location throughout the game even if they start at random locations within $\mathcal{S}$.

2) Where am I going?: This is the question that motivated most of the research related to Marco Polo. However, one key issue here relating to the experimental implementation of Marco Polo is the method used for communication between the two robots. Since both robots have Tablet PC with wireless network interfaces, using TCP/IP over a standard wireless network is simple and effective. Also, both ERSP and Qt provide networking modules that make communication relatively easy. We use the ERSP networking in our implementation.

3) How do I get there?: Answering this question involves designing a low-level controller that is capable of generating values of v and ω that will maneuver the robot from its current location to a goal location within the game area. Since the goal location depends on the low-level controller that is chosen, a high-level control strategy must take into account the low-level controller that is used or supply the needed control values rather than the goal location. The pursuit strategy presented above is designed for use with an attractive potential field controller [14]. Suppose that pursuer i is located at (x_p^i, y_p^i) and must travel to the goal location $r_i = (x_r^i, y_r^i) \in \mathcal{S}$. We define the distance error $\ell_i = d(p_i, r_i)$ as the Euclidean distance between pursuer i and its target. Let $\phi_i = \text{arctan2}(y_r^i - y_p^i, x_r^i - x_p^i)$ represent the angle of a vector connecting the pursuer to its target. Then, we define the angular error as $\psi_i = \theta_p^i - \phi_i$ where $\psi_i \in (-\pi \quad \pi]$. Based on this notation, the proportional control law is given by

$$u_p^i = K_p \zeta_i, \tag{4}$$

where $\zeta_i = [\ell_i \quad \psi_i]^T$ is the error vector and $K_p = \text{diag}(k_v, k_\omega)$ is a diagonal matrix of control constants.

4) Software Integration: To complete the game, the design techniques discussed above are integrated into a Qt plug-in DLL that may be configured and managed using the Robotic Games Coordinator and Game Manager. The plug-in is designed using four modules including: a vSLAM$^{\text{TM}}$ module, a networking module, a joystick control module, and an autonomous control module. The game referee uses the Coordinator to specify which of the two robots will be the pursuer and which will be the evader. Both robots load the vSLAM$^{\text{TM}}$ and networking modules together with one of the control modules. All three modules run in separate threads and communicate using a thread-safe event system that is built into ERSP.

The autonomous control design is best described using the notation of a hierarchical hybrid system [5]. An automaton representation of this system is shown in Figure 4. This

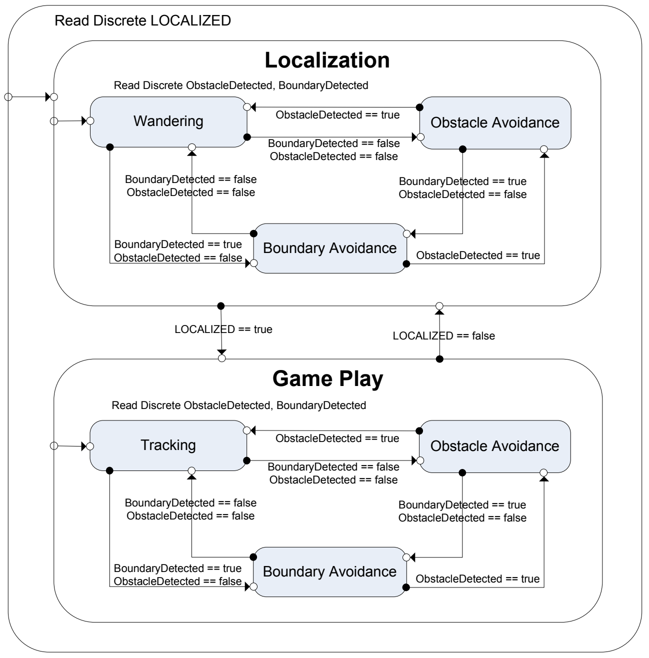
Read Discrete LOCALIZED
Localization
Read Discrete ObstacleDetected, BoundaryDetected
Wandering
ObstacleDetected == true
Obstacle Avoidance
BoundaryDetected == false
ObstacleDetected == false
BoundaryDetected == false
ObstacleDetected == false
BoundaryDetected == true
ObstacleDetected == false
BoundaryDetected == true
ObstacleDetected == false
Boundary Avoidance
ObstacleDetected == true
LOCALIZED == true
LOCALIZED == false
Game Play
Read Discrete ObstacleDetected, BoundaryDetected
Tracking
ObstacleDetected == true
Obstacle Avoidance
BoundaryDetected == false
ObstacleDetected == false
BoundaryDetected == false
ObstacleDetected == false
BoundaryDetected == true
ObstacleDetected == false
BoundaryDetected == true
ObstacleDetected == false
Boundary Avoidance
ObstacleDetected == true

Fig. 4. A hybrid automaton representing the autonomous control module.

design consists of two outer states that switch based on the vSLAM^{TM} module's confidence in its location estimation. If this value is below a threshold, the *Localization* state is active. Otherwise, the robot switches to the *Gameplay* state. Each outer state contains three inner states including obstacle avoidance, boundary detection, and goal seeking. The goal in the *Localization* state is simply wandering. This state is implemented using an ERSP behavior and is used to enhance the vSLAM^{TM} module's ability to localize. In the *Gameplay* state, the goal is to approach a goal point that is calculated based on the pursuit strategy as discussed above. When one of the obstacle avoidance or boundary avoidance states is active, the current goal is ignored and v and ω are adjusted to prevent leaving the boundary or colliding with an obstacle.

VI. Future Work

We plan to add support for additional sensors to the platform. The framework will support any hardware that has a driver for the ERSP framework. Currently, we are integrating the Evolution Robotics NorthStar sensor which may be used for more accurate indoor localization. Additionally, we are investigating integrating the designed framework with open-source robot control libraries to be compatible with other multi-vehicle platforms. Finally, because the platform is an excellent resource for studying issues related to human-robot interaction, we plan to continue research in this area.

VII. Conclusions

In this paper, we have presented a framework for robotics education and research. The benefits of using this framework include providing a uniform hardware and software interface for all applications and simplfying the procedure for giving demonstrations. The hardware available on the Scorpion robot makes it flexible enough for exploring a wide variety of robotics related issues. Furthermore, the software libraries provide native support for this hardware including built-in robotics behaviors and algorithms that can be combined to create sophisticated robotics applications.

Although the framework was originally intended for education, we quickly discovered that a designer who is writing a robotic "game" faces many of the same challenges and problems as a someone that is creating real-world robotics applications. In fact, several of the questions that we encountered while creating the games motivated ongoing research projects. Marco Polo is an excellent example where the main research issue centers on how to control a pursuer robot given intermittent knowledge of the evaders. We have provided some preliminary results based on constrained evader motion. Finally, we have successfully implemented Marco Polo using this framework. Surveys from a recent science camp rated Robotic Games the most educational and informative activity of the day. Additional information related to Robotic Games may be found on the MARHES web site [1].

References

[1] (2007) MARHES laboratory. [Online]. Available: http://marhes.okstate.edu

[2] J. Chestnutt, P. Michel, K. Nishiwaki, J. Kuffner, and S. Kagami, "Intelligent joystick for biped control," in *Proc. IEEE Int. Conf. Robot. Automat.*, Orlando, FL, May 2006, pp. 860–865.

[3] M. Matarič, "Robotics education for all ages," in *Proceedings, AAAI Spring Symposium on Accessible, Hands-on AI and Robotics Education*, Palo Alto, CA, March 22-24 2004.

[4] B. Browning, J. Searock, P. E. Rybski, and M. Veloso, "Turning segways into soccer robots," *Industrial Robot*, vol. 32, no. 2, pp. 149–156, 2005.

[5] R. Fierro, L. Chaimowicz, and V. Kumar, "Multi-robot cooperation," in *Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications*, S. Ge and F. Lewis, Eds. Boca Raton, FL: CRC Press, 2006, ch. 11.

[6] E. B. Martinson and F. Dellaert, "Marco polo localization," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 2, Taipei, Taiwan, September 14-19 2003, pp. 1960–1965.

[7] T. H. Chung, J. W. Burdick, and R. M. Murray, "A decentralized motion coordination strategy for dynamic target tracking," in *Proc. IEEE Int. Conf. Robot. Automat.*, Orlando, Florida, May 2006, pp. 2416–2422.

[8] Z. Tang and Ümit Özgüner, "On non-escape search for a moving target by multiple mobile sensor agents," in *Proc. American Control Conf.*, Minneapolis, MN, June 14-16 2006, pp. 3525–3530.

[9] T. G. McGee and J. K. Hedrick, "Guaranteed strategies to search for mobile evaders in the plane," in *Proc. American Control Conf.*, Minneapolis, MN, June 14-16 2006, pp. 2819–2824.

[10] V. Isler, S. Kannan, and S. Khanna, "Randomized pursuit-evasion in a polygonal environment," *IEEE Trans. on Robotics*, vol. 21, no. 5, pp. 875–884, 2005.

[11] (2007) Evolution robotics. [Online]. Available: http://www.evolution.com

[12] N. Karlsson, E. D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich, "The vSLAM algorithm for robust localization and mapping," in *Proc. IEEE Int. Conf. Robot. Automat.*, Barcelona, Spain, April 2005, pp. 24–29.

[13] (2007) Handvu: Hand gesture recognition. [Online]. Available: http://www.movesinstitute.org/~kolsch/HandVu/HandVu.html

[14] J. Clark and R. Fierro, "Cooperative hybrid control of robotic sensors for perimeter detection and tracking," in *Proc. American Control Conf.*, vol. 5, June 2005, pp. 3500–3505.

[15] D. Le, "An efficient derivative-free method for solving nonlinear equations," *ACM Trans. Math. Softw.*, vol. 11, no. 3, pp. 250–262, 1985.