Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea • May 21-26, 2001

Pursuit-Evasion Games with Unmanned Ground and Aerial Vehicles

René Vidal Shahid Rashid Cory Sharp Omid Shakernia Jin Kim Shankar Sastry

Department of EECS, University of California Berkeley, Berkeley CA 94720
{rvidal,rashid,cssharp,omids,jin,sastry}@eecs.berkeley.edu

Abstract

This paper presents the implementation of a hierarchical architecture for the coordination and control of a heterogeneous team of autonomous agents. We consider the problem of having a team of agents pursue a second team of evaders while building a map of the environment. The control architecture emphasizes the autonomy of each agent yet allows for coordinated efforts among them. We address the technical challenges and implementation issues of multi-agent operation. Finally we present experimental results of a pursuit-evasion game scenario between unmanned ground and aerial vehicles.

1 Introduction

The BErkeley AeRobot (BEAR) project [3] is a research effort at the University of California, Berkeley that encompasses the disciplines of control, hybrid systems theory, computer vision, sensor fusion, communication, game theory and multi-agent coordination.

This paper highlights the efforts of the BEAR project in multi-agent research from an implementation perspective. We consider a group of agents acting as *pursuers* attempting to capture a group of *evaders* within a bounded but unknown environment (see Figure 1). The classical approach to these pursuit-evasion games is to first build a map of the terrain and then play the game in a known environment. Several techniques have been proposed for map building: In [15] an algorithm is proposed for maximum likelihood estimation of the map of a region based on noisy measurements obtained by a mobile robot. However, systematic map building is time consuming and computationally expensive, even in the case of simple two dimensional rectilinear environments [4]. On the other hand, most of the literature in pursuit-evasion games, see e.g. [9], assumes worst case motion for the evaders and an accurate map of the environment. In practice, this results in overly conservative pursuit policies if applied to inaccurate maps built from noisy measurements.

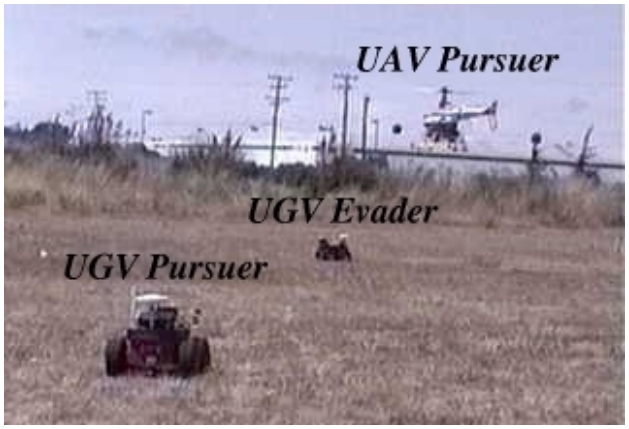

Figure 1: UAV-UGV Pursuit Evasion Game

Recently, the pursuit-evasion game and map building problems have been combined in a probabilistic framework [5], which avoids the conservativeness inherent to the classical worst-case approaches and takes into account inaccuracies in sensor information. The basic setup considers multiple pursuers trying to capture a single evader undergoing random motion. In [5] it is shown that, under certain assumptions, there exists a "persistent" pursuit policy that guarantees that the evader can be captured in finite time with probability one. In [11, 12] the basic scenario is extended to consider supervisory agents, such as a helicopter, that can estimate the position of the evader but not capture it. The approach has also been extended to multiple evaders, under the assumption that each one can be identified separately. The case where the evaders actively avoid the pursuers has been investigated in [6].

We present the implementation of a pursuit-evasion game between ground and aerial robots following the approaches in [5, 11, 12]. Section 2 describes the scenario and Section 3 describes a hierarchical architecture for multi-agent coordination and control adapted from [7, 8, 10, 16]. Section 4 describes the implementation of the architecture, Section 5 presents the experimental results and Section 6 concludes the paper.

2 Pursuit-Evasion Scenario

This section describes the pursuit-evasion scenario considered in this paper. We first outline the theoretic foundations for the problem, including map building and pursuit policies (See [5, 11, 12] for details) and then propose a visual-based algorithm for estimating the position of obstacles and evaders.

2.1 Rules of the Game

Consider a scenario in which the environment is a two dimensional grid $\mathcal{X}$ with n_c square cells. $\mathbf{x}_p \subset \mathcal{X}$ ($\mathbf{x}_e \subset \mathcal{X}$) is the set of cells occupied by the n_p pursuers (n_e evaders) and ω the set of obstacle locations. The motion of all the agents is constrained to either remain in the current cell or move to a cell in $\mathcal{A}(x)$, the set of (up to eight) cells adjacent to x. Aerial pursuers can move to any cell in $\mathcal{A}(x)$ (they can fly over obstacles an evaders), but ground pursuers and evaders are restricted to move to empty cells.

Each pursuer collects information about $\mathcal{X}$ at discrete time instants $t \in \mathcal{T} \triangleq \{1, 2, \ldots\}$. Each measurement $\mathbf{y}(t), t \in \mathcal{T}$ is a triple $\{\mathbf{v}(t), \mathbf{e}(t), \mathbf{o}(t)\}$, where $\mathbf{v}(t) \subset \mathcal{X}$ denotes the measured positions of the pursuers and $\mathbf{e}(t) \subset \mathcal{X}$ ($\mathbf{o}(t) \subset \mathcal{X}$) is a set of cells where each evader (obstacle) was detected. Sensor information is assumed to be perfect for the cells in which pursuers are located, but not for the adjacent cells. A simple sensor model based on the probability of false positives $p \in [0, 1]$ and false negatives $q \in [0, 1]$ of an agent detecting an evader in adjacent locations is considered. Also, we assume that pursuers have perfect knowledge of their own locations, that is $\mathbf{v}(t) = \mathbf{x}_p(t)$.

Since pursuers are able to identify each evader, they keep one map for each evader and one map for the obstacles. It is possible to have one pursuer collect all the measurements, build the maps and broadcast them to the rest of the team, or each pursuer build its own map and share its measurements. If the communication network is perfect (no packet loss, no transmission time or delay), all the pursuers have identical maps. Otherwise, each pursuer will update its maps with the information available to it.

Whenever an evader is captured, which happens when a ground pursuer and an evader occupy the same cell, the captured evader is removed from the game and its map is no longer updated. Aerial pursuers are not allowed to capture an evader.

2.2 Updating Evader and Obstacle Maps

Since each evader is identified separately, we can assume $n_e = 1$. Let $p_e(x, \tau|Y_t)$ be the (conditional) posterior probability of the evader being in cell x at time τ, given the measurements $\mathbf{Y}_t = Y_t$ taken up to time t. Similarly, let $p_o(x|Y_t)$ be the (conditional) probability of having an obstacle in cell x given the measurements Y_t. At time t we have an estimate of the evader map, $p_e(x, t|Y_{t-1})$ and an estimate of obstacle map $p_o(x|Y_{t-1})$. We obtain a new measurement $\mathbf{y}(t)$ and wish to estimate $p_o(x, Y_t)$ and $p_e(x, t+1|Y_t)$. We do this in three steps: First we compute $p_e(x, t|Y_t)$ as:

$$\begin{cases} 0 & x \in \mathbf{o}(t) \cup \mathbf{v}(t) \setminus \mathbf{e}(t) \text{ or } \exists \bar{x} \neq x : \bar{x} \in \mathbf{v}(t) \cap \mathbf{e}(t) \\ \alpha p_e(x, t|Y_{t-1}) p^{k_1} (1-p)^{k_2} q^{k_3} (1-q)^{k_4} & \text{otherwise} \end{cases} \quad (1)$$

where α is a normalizing constant chosen so that $\sum_{x \in \mathcal{X}} p_e(x, t, Y_t) = 1$; k_1 is the number of pursuers that reported in $\mathbf{e}(t)$ seeing the evader at cells other than x that are adjacent to their one (false positives); k_2 is the number of pursuers that reported in $\mathbf{e}(t)$ not seeing the evader at cells other than x that are adjacent to their one (true negatives); k_3 is the number of pursuers that reported in $\mathbf{e}(t)$ not seeing the evader at the cell x adjacent to their one (false negatives); and k_4 is the number of pursuers that reported in $\mathbf{e}(t)$ seeing the evader at the cell x adjacent to their one (true positives). $\sum_{i=1}^{4} k_i$ must be equal to the number of cells adjacent to any of the pursuers positions in $\mathbf{v}(t)$.

Second, we compute $p_o(x|Y_t)$ as:

$$\begin{cases} \frac{(1-q)p_o(x|Y_{t-1})}{(1-q)p_o(x|Y_{t-1}) + p(1-p_o(x|Y_{t-1}))} & x \in \mathcal{A}(\mathbf{x}_p(t)) \cap \mathbf{o}(t) \\ \frac{q p_o(x|Y_{t-1}}{q p_o(x|Y_{t-1}) + (1-p)(1-p_o(x|Y_{t-1}))} & x \in \mathcal{A}(\mathbf{x}_p(t)) \setminus \mathbf{o}(t) \\ 1 & x \in \mathbf{x}_p(t) \cap \mathbf{o}(t) \\ 0 & x \in \mathbf{x}_p(t) \setminus \mathbf{o}(t) \\ p_o(x|Y_{t-1}) & \text{otherwise} \end{cases} \quad (2)$$

Finally, in order to compute $p_e(x, t+1|Y_t)$ pursuers assume a Markov model for the motion of the enemy which is determined by the probability $\rho \in [0, 1/8]$ that the evader moves to an empty adjacent cell. Therefore, the evader map is updated as:

$$p_e(x, t+1|Y_t) \approx (1 - |\mathcal{A}(x)|\rho) p_e(x, t|Y_t) + \rho p_o(x, Y_t) \sum_{\bar{x} \in \mathcal{A}(x)} p_e(\bar{x}, t|Y_t). \quad (3)$$

2.3 Pursuit Policies

Given the current estimate of the location of obstacles and evaders, the pursuers need to choose where to move. A natural objective would be to minimize the expected value of the time of capture. However, such an objective is a complicated function of the pursuit policy, which makes the optimization not suitable for real-time applications. Therefore, we consider the following suboptimal policies:

Greedy Policy: here each pursuer moves to the adjacent cell with the highest probability of containing

an evader over all the evader maps, that is

$$\mathbf{x}_p(t+1) = \underset{x \in \{\mathcal{A}(\mathbf{x}_p(t)) \cup \mathbf{x}_p(t)\}}{\operatorname{argmax}} \max_{i=\{1 \ldots n_e\}} p_{e(i)}(x, t+1|Y_t) \quad (4)$$

where $p_{e(i)}(x, t+1|Y_t)$ represents the probability of evader i being in cell x at time $t+1$ given the measurements Y_t. Notice that this policy is advantageous in scalability, since it assigns more importance to local measurements by searching only in $\{\mathcal{A}(\mathbf{x}_p(t)) \cup \mathbf{x}_p(t))\}$ regardless of the size of the environment $\mathcal{X}$.

Global Maximum Policy: one disadvantage of the greedy policy is that the location of the evader does not have an immediate effect on selecting $\mathbf{x}_p(t+1)$ when $\mathbf{x}_e(t)$ is far from $\mathbf{x}_p(t)$. The global maximum policy intends to improve this by searching the entire environment. Each pursuer sets a trajectory $traj$ towards the global location in $\mathcal{X}$ with the highest discounted probability of containing an evader over all the evader maps, that is

$$\mathbf{x}_p(t+1) = traj(\underset{x \in \mathcal{X}}{\operatorname{argmax}} \max_{i=\{1 \ldots n_e\}} \frac{p_{e(i)}(x, t+1, Y_t))}{d(x, x_p(t))} \quad (5)$$

where the discount factor d is the distance between x and $\mathbf{x}_p(t)$. One of the disadvantages of this policy is that multiple pursuers may try to move to the same cell. Simulation results [11, 12] show that this can be solved with an appropriate combination of the two policies.

2.4 Visual based estimation of the position of obstacles and evaders

To build a probabilistic map of obstacles and evaders, each agent needs to detect their location in the 3D world. Here, we assume that ground and aerial pursuers (UGV and UAV, respectively) know their orientation and are equipped with a camera to sense the environment. Then, the 3D location of obstacles and evaders is estimated by triangulation, given their observed positions in the image plane, the orientation of the camera and the orientation of the observer.

Figure 2 shows the coordinate frames and Euclidean motions involved in the calculation of the 3D position of obstacles and evaders. The coordinate frames are labeled as: (a) Inertial frame, (b) UAV frame, (c) Camera base, (d) Camera head, and (e) UGV frame.

Let $g_{ij} \triangleq (R_{ij}, p_{ij}) \in SE(3)$ represent the relative orientation (rotation and translation) of frame i with respect to frame j. Let $\hat{w} \in so(3)$ be the skew symmetric matrix associated with axis $w \in \mathbb{R}^3$. Also let (e_1, e_2, e_3) be the usual basis for $\mathbb{R}^3$. If the observer is a UAV, then $g_{ab} = (\exp(\hat{e_3}\psi) \exp(\hat{e_2}\theta) \exp(\hat{e_1}\phi), p_{ab})$, where (ψ, θ, ϕ) are the estimates of the yaw, pitch and roll angles of the helicopter and $p_{ab} \in \mathbb{R}^3$ is the estimate of its position. $g_{bc} = (R_{bc}, p_{bc})$ is a predefined (known) transformation and $g_{cd} = (\exp(\hat{e_2}\alpha) \exp(\hat{e_1}\beta), 0)$, where (α, β) are the estimates of the pan and tilt angles of the camera. Then, the orientation of the camera head with respect to the fixed inertial frame is then given by:

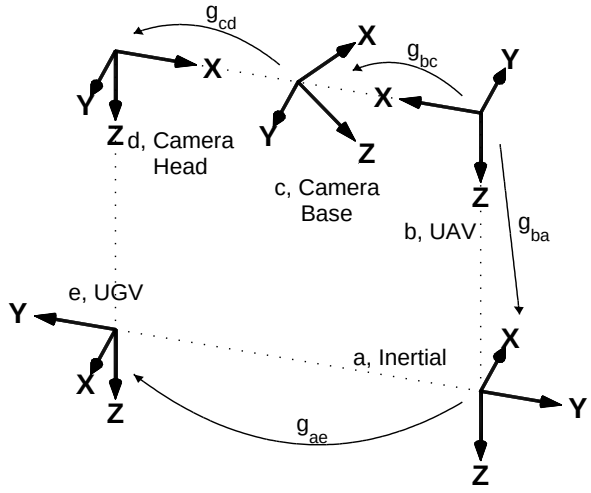

Figure 2: Coordinate frames for visual based estimation of the positions of obstacles and evaders

$$(R_{ad}, p_{ad}) = (R_{ab}R_{bc}R_{cd}, R_{ab}p_{bc} + p_{ab}). \quad (6)$$

Let f be the estimate of the zoom factor of the camera, (w, h) be the (known) width and height of an image in pixels and (fov_x, fov_y) be the (known) fields of view of the camera for $f = 1$. Then the camera calibration matrix is given by:

$$A = \begin{bmatrix} \frac{fw}{2\tan(fov_x/2)} & 0 & w/2 \\ 0 & \frac{fh}{\tan(fov_y/2)} & h/2 \\ 0 & 0 & 1 \end{bmatrix}. \quad (7)$$

Let $\mathbf{x}$ be the estimate of the position of an obstacle (evader) in the image plane. Then its 3D position is obtained as:

$$q = \frac{(z_0 - e_3^T p_{ad})}{e_3^T R_{ad} A^{-1} \mathbf{x}} R_{ad} A^{-1} \mathbf{x} + p_{ad} \quad (8)$$

where z_0 is the (fixed) z-coordinate of the evader on the ground, assuming a flat terrain.

If obstacles (evaders) are being observed by a ground agent, equation 8 can still be applied with minor changes. Replace frame (b) by frame (e), the UGV frame. Then $R_{ae} = \exp(\hat{z}\gamma)$, where γ is the estimate of the heading of the UGV, p_{ae} is the estimate of the position of the observer and g_{ec} is also a predefined (fixed) transformation.

3 System Architecture

The efforts and design of the BEAR project are based on a hierarchical architecture of control [7], which segments the control of each agent into a number of different layers of abstraction as shown in Figure 3. The layers of abstraction allow for the same high-level intelligent control strategies to be applicable to both UAVs and UGVs. By abstracting away the details of sensing and control of each agent, we gain the interoperability of a unified framework for high level intelligent pursuit policies across all platforms. We first give an overview of the high levels of abstraction for the control architecture, then dive into the details of implementation for each agent.

3.1 Strategic Planner & Map building

The *strategic planner* handles the selection and control of tasks at the highest level. It maintains a state-space of the system useful for mission planning and then tasks the robots according to mission objectives. It is at the strategic planner level that inter-agent communication takes place. In order to effectively and efficiently incorporate information from other agents, yet maintain the autonomy of each individual agent, the strategic planner must be carefully designed. State information maintained by the strategic planner is used by the *tactical planner* for motion control of the agent. Each agent will make observations of the environment using sensors and through communication with other agents, and then decide a course of action (to map the environment or attempt to capture an evader, depending on the scenario). For this paper the details of the map builder and strategic planner have already been described in Sections 2.2 and 2.3 on pursuit-evasion games.

The *trajectory planner* is responsible for the design of a realizable trajectory for each agent, based on a detailed dynamic model of the vehicle and the set of way-points given by the tactical planner. It is at this level safety routines, such as obstacle avoidance, also reside. The trajectory planner provides a set of way-points to the *regulation layer*. The regulation layer uses classical control techniques (fuzzy control on the UGVs [1] and multivariable control on the UAVs [14]) to guide the agent to the desired way-point and send the tracking error back to the trajectory planner in case rescheduling is necessary. The specific details of regulation layer for the UGV and UAV agents will be described in Section 4.

3.2 Communication Architecture

While the autonomy of each agent is central to our design, some coordination within a group of agents

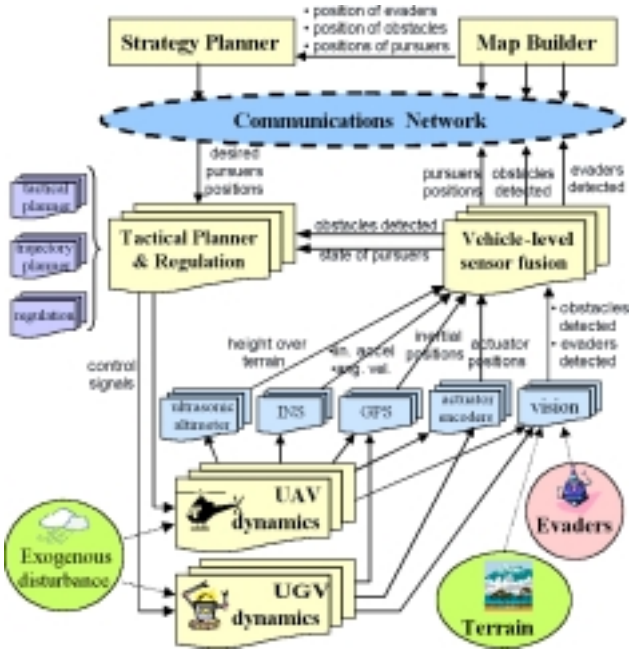

Figure 3: System Architecture

greatly enhances performance and thus communication between agents is necessary. The original theoretical work in the BEAR group assumed that each agent was fully aware of the state, position, and observations of every other agent. In effect a perfect, infinite-bandwidth, delay-free communication channel was modeled.

For a real implementation this is not realizable, so a simple scheme was developed for the remote agents to communicate with each other over wireless Ethernet. Information exchange between agents is purposefully made minimal and the algorithms employed by the agents are robust to the presence or absence of information from other agents. This ensures that each agent will still be able to operate autonomously and makes the system, as a whole, robust to communication loss or loss of an agent.

4 Implementation on UAV / UGV

This section describes the tactical planner, regulation layer, as well as the hardware and software systems for the BEAR project's fleet of UAVs and UGVs.

4.1 Common Hardware/Software

The BEAR project's UAVs and UGVs share many components for sensing and communication. Each agent has two on-board computers: the tactical planner is implemented on one computer while the trajectory generator and regulation layers are implemented

on the other. The commonality among all the UAVs and UGVs lies in the strategic layer and the following components:

- Ayllu multi-agent communication/control system
- WaveLAN wireless Ethernet
- NovAtel MillenRT2 GPS
- LittleBoard Pentium 233MHz running Linux
- Color tracking vision system

Ayllu: Ayllu is a tool for development of distributed control systems for groups of intelligent mobile robots. An Ayllu system consists of a group of behaviors which run in parallel and a set of connections through which messages are passed between behaviors. The behaviors may all be instantiated on one host computer, or may be spread across the network. Thus a single computer might control a number of robots, or a group of robots could be controlled in a fully distributed manner. Each behavior has a set of output ports which are connected at run-time to input ports of either local or remote behaviors [1].

Communication: Each robot has a Lucent WaveLAN wireless Ethernet card operating at 11Mbs. The wireless LAN is setup such that agents share information using either TCP/IP sockets or Ayllu. A limitation of the wireless LAN is that communication between two agents is lost if they are more than 100m apart. We are currently working with SRI to integrate a mobile routing scheme that allows agents to automatically detect when connectivity with another agent is lost [2]. In such a situation, packets are routed through a third agent inside the radius of operation of the network cards if such an agent exists.

GPS: Each agent is able to keep track of its position and orientation based on internal sensors: dead-reckoning from wheel motion for the UGVs and inertial navigation system (INS) for the UAVs. Due to drift, the accuracy of internal measurements of both the dead-reckoning and INS systems deteriorates rapidly over time. Therefore, we incorporated a NovAtel GPS MillenRT-2, which gives a 2cm accuracy for the position of the agents with respect to a global coordinate frame. We developed software for a simple sensor fusion strategy to continuously correct the dead-reckoning and INS sensors with the GPS measurements. The whole system operates at 10Hz.

Vision System: Each agent is equipped with a LittleBoard Pentium 233MHz *vision computer* running the Linux OS. The vision computer controls a Sony EVI-D30 pan/tilt/zoom (PTZ) color camera and an Imagenation PXC200 frame-grabber which acquires

Figure 4: UGV based on Pioneer 2-AT Ground Robot

video at 30Hz. We developed software for controlling and sensing the state of PTZ camera at 30Hz (See [13] for details). We use the ActivMedia Color Tracking System (ACTS) [1] which, in combination with the color camera and the frame grabber, allows our applications actively track up to 320 colored objects at 30 frames per second. For our pursuit-evasion scenario, we use the vision system to identify and locate obstacles and evaders based on their colors.

4.2 Unmanned Ground Vehicles

Here we describe the hardware/software components specific to our unmanned ground vehicles. Our UGV fleet consists of two ActivMedia Pioneer 2-AT [1] all terrain ground robots (see Figure 4), which in addition to the common hardware described above, are endowed with a 20MHz Siemens 88C1 microprocessor running P2OS (Pioneer 2 Operating System). The micro-controller implements the *regulation layer* which is responsible for low level control of the robot. It manages the motors, position encoders and user accessories such as sonars, compasses, grippers, etc.

The *tactical planner* and *trajectory planner* are implemented by the vision computer with either the Ayllu or Saphira software (see below). The vision computer (client) communicates with the micro-controller (server) through a serial connection. The vision computer receives the current state of the robot (position, heading, translational velocity, rotational velocity, sonar readings, etc.) and sends motion commands (move forward, rotate, etc.).

Ayllu: Ayllu has a standard set of predefined behaviors for the low level control and sensor interpretation of the Pioneer robots. These behaviors propagate information such as sonar ranges, dead-reckoning information, compass readings, etc., and accept messages for direct motion control [1].

Saphira: Saphira is an open architecture for behavior based mobile robot control developed by SRI International and ActivMedia [1]. It consists of a set of routines for communicating with and controlling a robot from a host computer. Saphira provides a facility for implementing *behaviors* as sets of fuzzy control rules. Behaviors have priority, activity-level and other state variables that mediate their interaction with other behaviors. For example, a routine can check whether a behavior has achieved its goal, and two behaviors can control the same actuator with the control actions being combined according to fuzzy rules.

4.3 Unmanned Aerial Vehicles

Here we describe the hardware/software components specific to our unmanned aerial vehicles. Our UAV testbed is a Yamaha R-50 helicopter (see Figure 5), which in addition to the hardware/software common to the UGVs, is endowed with a Boeign DQI-NP inertial navigation system (INS), ultrasonic height sensors and a Pentium 233MHz *navigation computer* running QNX real-time OS. The navigation computer is responsible for sensor management, wireless communication and low-level flight control. The *trajectory planner* and *regulation layer* are implemented by the navigation computer using classical control techniques based on μ-synthesis as described in [14].

The vision and navigation computers communicate with each other over a 115200bps RS232 serial link. Currently the communication link is only used for the vision system to gather state information from the navigation system. In the near future, we will implement the already existing QNX version of the *tactical planner* on the vision computer, which will send way-point commands for the trajectory planner and regulation layers of the navigation computer to follow.

Figure 5: UAV based on Yamaha R-50 Helicopter

5 Experimental Results

As a first stage towards implementing the full pursuit-evasion scenario, we divided the problem in two stages, each one consisting of a simplified version of the scenario.

In the first stage, we implemented the algorithm for building a probabilistic map of obstacles on two UGVs. Here the location of the obstacles is not estimated, but artificially generated by a ground station.

In the second stage, we implemented the pursuit-evasion scenario on one UAV and 2 UGVs. In this case the location of the evader is estimated by both the ground and aerial pursuers. However, none of them builds a map of the evader. The ground pursuer goes directly to the estimated position of the evader.

5.1 Map Building

In this scenario there is a ground station that stores a real map of the terrain. The ground station establishes a communication link with each robot, receives their current positions and sends the positions of obstacles located in cells adjacent to each robot's location. This process is meant to simulate each robot's sensing its surrounding environment. Then each robot communicates with the other to exchange its (artificial) measurements for the purpose of coordinated map building. Afterwards, both agents update their probabilistic maps of the environment based on the sensor readings, according to equation (2). Given their maps, the robots move using a greedy strategy designed to minimize the entropy of the obstacle map (as opposed to maximizing the probability of catching an evader).

We implemented the scenario on two Pioneer robots which run Saphira for motion control. Each robot runs a Saphira micro-task to estimate its "continuous" position from GPS and dead-reckoning measurements. That position is quantized to coordinates in the map grid with a cell of size 2×2m. Each agent runs a Saphira behavior to go the desired position on the map. In parallel, each agent runs a Saphira obstacle avoidance behavior based on sonar information. However, sonar information is not included in the map building process.

Map building experiments were run outdoors at the University of California-Berkeley Richmond Field Station. The map building system performed well in the field tests on the Pioneers. The agents were able to share their readings on the environment and build an accurate map of obstacle locations. The map building strategy is robust to modifications in movement due to lower-level hierarchical segments (i.e. obstacle avoidance, improper regulation). The movement strategies, however, are not as robust. Agents have a tendency to

be unable to move around obstacles for a number of iterations. Additionally, the agents tend to focus more on a local than a global search.

5.2 Pursuit-Evasion Game

We implemented a simplified version of the pursuit-evasion scenario for one UAV pursuer, one UGV pursuer and one UGV evader. In this setup, no agent builds a map of obstacles or evaders, nor moves according to the policies described in Section 2. Instead, the aerial agent estimates the position of the evader then commands the UGV pursuer to move directly to that location. The details of our experiment follow.

Although the aerial pursuer has the capability of pursuing a moving ground evader, the existing tactical planner on the navigation computer of the UAV has not been integrated with the vision computer. Currently, the vision computer is not able to send commands to the navigation computer, so for our initial experiment the evader remains stationary.

For identification purposes, the evader is painted with a known, unique, identifiable color. The UAV is flown in manual mode by a human operator until the color tracking software (ACTS) and on-board camera detect the evader. At this point, the UAV is placed in a set-point hover-control mode to autonomously maintain its position. However, its position and orientation may vary even in this mode due to exogenous disturbances. To compensate for this effect, the vision computer must continuously obtain updates from the navigation computer of the UAV and control the PTZ camera to maintain visual lock on the evader. The latter is done using the control strategy described in [13]. Then, using the state of the UAV, state of the camera, and calibrated image coordinates of the evader, the vision computer estimates the evader's ground position in the inertial frame as described in Section 2.4. By fact that it is also engaged as an Ayllu server, the vision computer transparently broadcasts over TCP/IP the evader position to the UGV pursuer.

Depending on recently received information, the UGV pursuer controls its motion using a set of Ayllu behaviors. If none of the pursuing team detects the evader, the UGV rotates in place, hoping to eventually obtain visual lock. If the UGV directly detects the evader with its on-board camera, it estimates the position of the evader using its own position, pan/tilt state of the camera, and calibrated image coordinates of the evader. Then, given that the pursuer has current information on the state of the evader, it engages a motion-intercept behavior, where position estimates gathered first-hand override estimates from the UAV. The pursuer concurrently runs a simple Ayllu obstacle avoidance behavior based on sonar readings. The UGV dynamically adapts its motion to avoid detected obstacles, but does not record those obstacle locations in a map.

Figure 6 shows the results of our pursuit-evasion experiment. The actors begin in an initial configuration (a). Once the UAV detects the evader, the vision computer controls the PTZ camera to center the evader in the image. Upon detection of the evader, the UGV pursuer engages a motion-intercept behavior (c) until it finally captures the evader (d).

6 Conclusions and Current Research

This paper presented the implementation of a real-time platform for muti-agent control of ground and aerial vehicles. The implementation was based on a hierarchical control architecture that is applicable to a heterogeneous group of agents and allows for coordinated efforts among them.

The architecture was successfully applied to a subset of the pursuit-evasion game scenario described in the paper, in which a team of agents pursues a second team of evaders while building a map of the environment. Overall, our experiments show the possibility and create the foundation for using a grid based approach for multi-agent control in pursuit-evasion games.

We are currently implementing the full pursuit-evasion game scenario by combining the experimental results of this paper. We have recently acquired three additional ground robots for a pursuit-evasion game between two ground evaders, one aerial and three ground pursuers.

We are also working on a new implementation of the strategic planner for the centralized case. The idea is to integrate the strategic planner implemented in our Matlab/Simulink simulation platform [12] with the hardware and lower level control of the ground and aerial robots through the wireless communication network. In this way, we will be able to use the same implementation of the strategic planner in both simulations and experiments. This will give us great flexibility to test new control strategies in our pursuit-evasion game experiments by simply changing a few lines of Matlab code.

Acknowledgment

The authors would like to thank the BEAR team, especially Hyunchul Shim. We also want to thank João Hespanha, John Koo and Frank Hoffmann. This research was supported by the ONR grants N00014-97-1-0946 and N00014-00-1-0621, and ARO MURI grant DAAH04-96-1-0341.

(a)

(b)

(c)

(d)

Figure 6: Results from our pursuit-evasion experiment. (a) Initial configuration, where no pursuer detects the evader. Once detected by the UAV, the vision computer controls the camera to center the evader in the image (b). Then, the UGV in engages a motion-intercept behavior (c), until it finally captures the evader (d).

References

[1] *ActivMedia, Inc.* `http://robots.activmedia.com`.

[2] B. Bellur and R. Ogier. A reliable, efficient topology broadcast protocol for dynamic networks. In *Proc. of IEEE Infocom*, pages 178–186, 1999.

[3] BErkeley Aerial Robot (BEAR) Project homepage. `http://robotics.eecs.berkeley.edu/bear`.

[4] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *Journal of the ACM*, 45(2):215–245, March 1998.

[5] J. Hespanha, H. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games. In *Proc. of 38th IEEE CDC*, pages 2432–2437, Dec. 1999.

[6] J. Hespanha, M. Prandini, and S. Sastry. Probabilistic pursuit-evasion games: a one-step Nash approach. In *Proc. of 39th IEEE CDC*, pages 2272–2277, Dec. 2000.

[7] T. Koo, F. Hoffmann, H. Shim, B. Sinopoli, and S. Sastry. Hybrid control of an autonomous helicopter. In *IFAC Workshop on Motion Control*, pages 285–290, Grenoble, France, Oct. 1998.

[8] T. Koo, B. Sinopoli, A. Sangiovanni-Vincentelli, and S. Sastry. A formal approach to reactive system design: A UAV flight management system design example. In *IEEE International Symposium on Computer-Aided Control System Design*, pages 522–527, 1999.

[9] S. LaValle, D. Lin, L. Guibas, J-C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 732–742, 1997.

[10] G. Pappas, C. Tomlin, J. Lygeros, D. Godbole, and S. Sastry. A next generation architecture for air traffic management systems. In *Proc. of 36th IEEE CDC*, pages 2405–2410, Dec. 1997.

[11] S. Rashid. Design and Implementation of Multi-Agent Control: Pursuit & Map Building. Master's thesis, UC Berkeley, 2000.

[12] S. Rashid and J. Kim. Multiple-agent probabilistic pursuit-evasion games in 2.5D. Technical Report Memo No. UCB/ERL M99/34, UC Berkeley, 1999.

[13] C. Sharp, O. Shakernia, and S. Sastry. A vision system for landing an unmanned aerial vehicle. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2001.

[14] H. Shim, H.J. Kim, and S. Sastry. Hierarchical control system synthesis for rotorcraft-based unmanned aerial vehicles. In *Proc. of AIAA Conference on Guidance, Navigation and Control*, Denver, 2000.

[15] S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots*, 31(5):1–25, 1998.

[16] C. Tomlin, G. Pappas, J. Lygeros, D. Godbole, and S. Sastry. Hybrid control models of next generation air traffic management. In *Proc. of Hybrid Systems: 4th International Conference*, Oct. 1996.