

Decentralized Sensing and Tracking for UAV Scheduling

Valentino Crespi^a, Wayne Chung^b and Alex B. Jordan^b

^aCalifornia State University, 5151 State University Drive, Los Angeles, CA, USA 90032;

^bThayer School of Engineering, Dartmouth College, Hanover, NH, USA 03755;

ABSTRACT

This paper presents a fully automated and decentralized surveillance system for the problem of detecting and possibly tracking mobile unknown ground vehicles in a bounded area. The system consists ideally of unmanned aerial vehicles (UAVs) and unattended fixed sensors with limited communication and detection range that are deployed in the area of interest. Each component of the system (UAV and/or sensor) is completely autonomous and programmed to scan the area searching for targets and share its knowledge with other components within communication range. We assume that both UAVs and sensors have similar computing and sensing capabilities and differ only in their mobility (sensors are stationary while UAVs are mobile). Gathered information is reported to a base station (monitor) that computes an estimate of the global state of the system and quantifies the quality of the surveillance based on a measure of the uncertainty on the number and position of the targets overtime. The present solution has been achieved through a robotic implementation of a software simulation that was in turn realized under the principles of a novel top-down methodology for the design of provably performant agent-based control systems. In this paper we provide a description of our solution including the distributed algorithms that have been employed in the control of the UAV navigation and monitoring. Finally we show the results of an novel experimental performance analysis of our surveillance system.

Keywords: Unmanned Aerial Vehicle, UAVs, Distributed Computing, Autonomous Agents, Target Tracking, Surveillance

1. INTRODUCTION

Upon their appearance, *unmanned aerial vehicles* (UAVs) were employed in aerial reconnaissance military missions. These flying vehicles were characterized by a fully centralized navigation control. The mode of deployment was to launch them from a battleship sailing near the coast, and fly them remotely towards the inland, over geographic barriers, such as mountains to provide visual information of what could not be seen from aboard. This idea is similar to the way reconnaissance balloons were employed during the military conflicts of the nineteenth century in America and in Europe.

A natural evolution of this tactical employment of UAVs was to deploy a large number of UAVs to carry out missions of surveillance and target tracking [1, 2]. The work in [1, 2] proposed a fully decentralized control system for the navigation of UAVs on patrol of a given area with charges of surveillance and tracking of unknown (possibly mobile) targets. The obtained solution was intended to provide a large fault-tolerant autonomous system composed of fully automated, unmanned, easy-to-deploy, and cheap (expendable) vehicles.

In this paper we present a nontrivial robotic implementation of the distributed algorithm introduced in [1, 2] for the following:

Instance: A battlefield consisting of a terrain and a set of potential targets that move along unknown trajectories.

Problem: Deploy a number of scouts (flying or grounded vehicles) and a number of fixed sensors having limited sensing and communication range and bandwidth with the following objective and optimization requirements:

- **Objective:** Know number and position of targets over time.
- **Optimization/Trade-off:** Drive scouts along unpredictable trajectories minimizing the consumption of fuel per cell visit.

Our initial solution was devised for a scenario in which all the UAVs were flying at different altitudes to prevent collisions and there were no fixed sensors. In this paper UAVs are substituted with grounded (wheeled iRobots), and there are also fixed sensors that are treated as stationary UAVs. In other words we assume that UAVs and fixed sensors possess similar computing and sensing capabilities. Also both their sensing and communication range are assumed to be limited.

Our approach is based on a top-down methodology developed to design *provably performant* agent-based control systems [1–3]. This was a first response to the problem of supplying agent-based technology with a general unified model for performance estimation [2, 4] and is based on a three-stage process:

1. The first stage of the design is characterized by the assumption that each autonomous subcomponent of the system has full knowledge of the global state. This means that we looked for a centralized solution. Specifically, we produce a navigation algorithm that is run locally by each UAV but with global knowledge of all the events that occur in the large area being monitored. One way to simulate this is by assuming the existence of an omniscient supermonitor (e.g. a satellite with unbounded transmission range and bandwidth) that records all the events occurring locally and allows all the UAVs and sensors to learn about them.
2. The second stage is to remove the supermonitor and replace global resources with local resources compatible with the constraints of the given distributed environment. This introduces communication issues due to the fact that now vehicles must acquire information through interactions with other locally reachable components in order to infer the most likely global state of the system. The result of this stage is a navigation algorithm that runs locally and uses local information. It records all the events that occur within range and maintains a representation (hypothesis) of the global state. When two UAVs/sensors come into contact, an exchange of knowledge and experience takes place on a peer-to-peer basis.
3. The final stage consists of calibrating the obtained distributed solution. We want to determine the minimum amount of information that needs to be exchanged between peers in order to keep the uncertainty on the global state (relative to the ideal condition of complete centralization) below a certain bound, which in turn depends on a global objective that the entire monitoring system needs to fulfill. An example of global objective is a function that quantifies how much the system knows about the position of the unknown targets over time. As exact knowledge of the position of the targets at any time is unrealistic, given that we deploy a finite number of scouts with limited radius of visibility, we are forced to accept a margin of uncertainty that needs to be properly quantified. This is attained through the notion of (α, β) -currency [5] that was already successfully employed in the framework of the Web. A crucial problem in that context would be the determination of the time needed for a search engine to reindex the entire Web in order to maintain an “up-to-date” database of pages. Alternatively we may define a risk function that instead quantifies the cost of not knowing enough about the position of the targets over time.

Instances of our methodology can be found in [6–8].

The rest of the paper is organized as follows. In Section 2 we provide a high-level description of the distributed algorithms for the UAV navigation. This includes also a specification of what kind of information pertaining individual knowledge of the global state is exchanged when UAVs and/or sensors come into contact. Section 3 contains the definition of the quantitative metrics that we use to measure how well our problem is solved. In particular we focus on the objective and define (α, β) -currency to quantify knowledge of the position of the targets. Then in Section 4 we describe in detail how the various distributed algorithms have been implemented in Java and then embedded in the hardware. Finally, in Section 5 we display the results of our experimental analysis aimed at studying the values of (α, β) -currency as a function of various parameters of the system such as communication range and complexity. In particular, one of our aims, in the prospective, is to show how to determine the minimum amount of information needed to be exchanged between UAVs and sensors in order for the monitoring system to keep the global “level of surveillance” sufficiently high.

2. UAV NAVIGATION AND INTERACTION

Each UAV and sensor maintains a hypothetical representation of the system’s state locally. This consists of a list L_T of sighted targets with time stamped information about their identity and position, plus other data used to model their kinematics, a list L_S of other UAVs or sensors that have come into contact and wish to exchange part of their experience, and finally information \mathcal{P} about the observable space necessary for UAVs to plan their journey. A detailed specification of L_T , L_S and \mathcal{P} requires a description of a) how the kinematics of targets are modeled and b) how the observable space is discretized.

Modeling Target Kinematics

At this stage of our study we assume that targets can move in any direction and perform a very hard-to-predict trajectory at bounded speed. If they do not follow a rectilinear path at all time, but rather make turns once in a while, then we can model their covered radial distance as a function $\rho(t) = h \cdot t^a$ that fulfills:

$$c \cdot \sqrt{t} \leq \rho(t) \leq d \cdot t$$

where \sqrt{t} represents the radial distance of a particle that describes undergoes symmetric Brownian motion [9, 10] and $d \cdot t$ represents a particle that follows a rectilinear path.

Discretization of the Observable Space

The observable space is partitioned into a grid of cells. Let $P_{i,j}$ denote the position of the center of cell $c = (i, j)$ in the $C \times C$ grid. Cells are given dynamic priorities that increase with time and are reset when they are visited (observed). Each scout s_i needs to select a cell as destination through a randomized choice. So, at time t scout i either has been assigned a destination cell or has to select a new cell by sampling from a probability distribution that is defined through the dynamic priorities and the Euclidean distances $\{d(s_i, P_c) \mid c \in C \times C\}$. The selection of a cell is irrevocable and the selecting scout has to reach its destination before making a new selection. Each scout travels rectilinearly at constant speed from its last visited cell to its current destination cell. We are also investigating more sophisticated UAV kinematics based on the application of the theory of artificial potential functions in the solution of path and trajectory planning [11–15]. Those techniques proved to be effective in dealing with problems of coordinated multi-agent navigation [2, 7]. The quantity $p_c(t)$, the dynamic priority of cell c at time t can be defined, in a discrete fashion, as:

$$\begin{cases} p_c(t+1) := p_c(t) + \delta \cdot w_{s(c)} & \text{cell } c \text{ is not visited at time } t \\ p_c(t+1) := 0 & \text{cell } c \text{ is visited at time } t \end{cases}$$

The matrix $W = \{w_{s(c)} \mid c \in C \times C\}$ is intended to be a matrix of static priorities that depend only on the “status” $s(c)$ of the cells. Each UAV and sensor maintains a local *priority table* $\mathcal{P} = \{p_c\}$.

Observe that the “status” of cells affects the derivative $\delta \cdot w_{s(c)}$ of the priority functions and so it can be used to control the evolution of the probability distributions and attract UAVs in certain areas. For example, assume that the status of each cell can be either “N” for *normal* or “H” for *high priority* and define $w_N \ll w_H$. Then the following event-driven procedure illustrates how to update the status of cells using information about target observations and their kinematic model:

UpdateStatus($\rho(t), \rho_0, r$)

- When a scout locates a target T_j at position $y_j(t)$, it sets an initial “radius of uncertainty”, ρ_0 . All the cells whose center P_c verify $\|P_c - y_j(t)\| \leq \rho_0$ are given the high priority derivative, i.e., $w_{s(c)} = w_H$. Then an event $e = (T_j, t + \rho^{-1}(\rho_0))$ is locally generated.
- If target T_j is not located by the same scout within time $[t, t + \rho^{-1}(\rho_0)]$ then event e will fire locally and the radius of uncertainty reviewed as:

$$\rho_1 = (1 + r) \cdot \rho_0 .$$

Finally a new event $e' = (T_j, t + \rho^{-1}(\rho_1))$ will be locally generated. So, now, a larger set of cells (those within range from $y_j(t)$) will be marked by a high priority derivative.

- If, on the contrary, the same target is located again by the same scout, then its radius of uncertainty will be reset to ρ_0 and a related local event generated.

Unattended sensors do not navigate. However, they maintain the same information as UAVs: target lists, scout lists and priority tables that they can exchange with UAVs when they travel by. In this way sensors facilitate cooperation by allowing indirect exchange of information among UAVs that do not cross each other's path but have visited the same area in different time frames.

In summary, each UAV and sensor needs to maintain locally the following information:

- Current destination cell (for UAVs only).
- A priority table $\mathcal{P} = \{p_c\}$.
- A list of targets $L_T = \{(id, y, t_s, \rho(t), \rho_i, e)\}$, where id , y and t_s are identity and time stamped position of last observation while $\rho(t)$, ρ_i and e are current kinematics model, radius of uncertainty and pending event as described above.
- A list of UAVs and sensors, $L_S = \{(id, y, t_s)\}$, where id , y and t_s are identity and time stamped position of last contact.

Deciding which information needs to be exchanged is a delicate matter. In our current implementations, UAVs and sensors exchange local priority tables \mathcal{P} and their target lists L_T . A policy to merge values of \mathcal{P} based on status and absolute value of the priorities is also implemented. The information in the target list is used to determine uniquely the status of the cells (in particular the position of the last observation y and the current radius of uncertainty ρ_i).

3. METRICS AND MEASUREMENTS

In our research we are interested in a quantitative analysis of the performance of our algorithms based on the following metrics:

1. *Application performance with respect to the Objective Invariant.*

We have already anticipated in the introduction that knowledge of the position of the targets at any time would be unrealistic in our system. So, we need to formalize our objective in order to account for the quantity of acceptable uncertainty that we are prepared to allow.

So, the main objective of our problem is to keep the value of a proper risk function below a certain level. We define this risk function in terms of (α, β) -currency [1, 5] of the observable space. This metric provides a quantification of the quality (currency) of the gathered information.

We can in principle discretize the observable space (e.g., the battlefield) into a bidimensional grid of observable cells. We say that our knowledge about the state of a particular cell is β -current at time t if that state has not changed between the time it was last observed (e.g., visited) and time $t - \beta$. And we say that the Target Surveillance System, at a certain time t , is (α, β) -current if the probability that a cell selected uniformly at random is β -current is at least α .

If we now denote with \mathcal{C} the set of all observable cells, then a natural definition for the risk function would be

$$R(t) = \sum_{i \in \mathcal{C}} r_i \cdot (1 - \alpha_i(\beta, t)),$$

where r_i is the cost of not possessing of β -current information about the state of cell i and $\alpha_i(\beta, t)$ is the probability of cell i being β -current at time t . So, we may define our objective by requiring that $R(t) \geq R_0$ for all $t \geq 0$ [1].

In order to measure (α, β) -currency we need to estimate how the state of each cell changes over time with respect to a set of events. In our model the state of a cell can be:

- **Empty:** no targets are inside it.
- **Invaded:** a target has entered it;
- **Abandoned:** a target that was already inside it left.

So, in order to measure the probability α that the system is β -current we need to imagine that each cell is equipped with a sensor that is able to record all those events with proper timestamps. This is attained, as we will see, through the use of a centralized monitor that receives all the broadcast events generated by UAVs, sensors and targets. Then, given β , at each time $t > \beta$, we compute the ratio of the cells that are β -current. A cell is declared β -current if events have occurred between the time it was last observed by any scout and time $t - \beta$. As we would like to keep α as close to 1 as possible, it is of particular interest the study of the variation of α as the UAVs' communication range decreases. Alternatively, instead of relating α to the communication range we may relate it to the rate of information irradiated over a bounded space by two UAVs during their contact (*bit - meter/sec*), and more generally to the *transport capacity* of the entire UAV/sensor network [16–18]. This quantity accounts for bandwidth limitations (rate), limited communication range (max distance) and the amount of data in bits that needs to be exchanged.

The role of the omniscient monitor is to estimate a theoretical value of α that will be acknowledgeable to Scouts, sensors and to a real monitoring base-station. This is due to the fact that we have assumed that targets broadcast their position and identity. In future work we will develop mathematical models for the rate of changes of the state of the cells that will allow realistic estimates of (α, β) -currency relative to a single scout and to a global monitor. And those estimates will be matched with the theoretical values computed in ideal conditions in order to assess the goodness of the models themselves.

2. Predictability and stability of the trajectories.

This metric is important in that if the potential enemy can predict the journey of the scouts within a certain time frame, it will be able to adopt undesirable counter-measures.

Let us consider the following event generation process: each time an observable cell is visited by any scout we record the corresponding cell id. Then as the system evolves we will be able to collect a (potentially infinite) sequence of cell id's. Our idea is to study the compressibility of such sequences in order to estimate the entropy of the stochastic source that has generated it. Maximum entropy would then correspond to total unpredictability of the next cell to be visited or, equivalently, to space aperiodicity of the trajectories*. We measure space aperiodicity by applying information theoretical tools. In particular we run our simulations recording the sequence of cell ids as they are visited. Then we use Lempel-Ziv coding schemes to compress the string of such recordings and finally we compute the compression ratio. These values provide us with a measure of the regularity of the string.

Roughly speaking, the idea behind the Lempel-Ziv Universal algorithm (see [19]) is to provide a shorter representation of a given sequence of characters by removing all the possible redundancies. The algorithm in question is optimal in that it attains a compression ratio that matches the information entropy of the source that has generated it. So, for example, a binary string composed only of, say, ones: $s_n = 1111111\dots$, corresponds to a generating source defined by the probability distribution: $p(0) = 0, p(1) = 1$ whose entropy is 0. It is not hard to provide a logarithmically long representation for such strings whose length $l(c(s_n))$ clearly verifies: $\lim_{n \rightarrow \infty} l(c(s_n))/n = 0$. On the other hand, if we consider a totally random binary sequence: $t_n = 01001101\dots$, corresponding to the probability distribution $p(0) = p(1) = 1/2$, whose entropy is maximum: 1, we see that there is no way to produce a compressed representation due to complete lack of regularities in t_n for all n . In that case $\lim_{n \rightarrow \infty} l(c(t_n))/n = 1$.

3. Energy consumption of UAVs per cell visit.

In our model we assume that energy and fuel consumption of a single UAV is proportional to the arc length of its trajectory. Thereby, in our analysis we estimate the lengths of those trajectories by adding lengths of segments of polygonal curves that approximate the actual curves.

*We focus on space aperiodicity since time periodicity can be easily broken by randomizing the speed of the scouts (within their capabilities).

In our previous investigations [1, 2] we have studied trade-offs between energy consumption and predictability of the trajectories. In this paper we are not concerned with the consumption of energy but we are more focused on studying the variation of (α, β) -currency as a function of the amount of information that scouts and sensors exchange when at close range and of the unpredictability of their trajectories.

4. IMPLEMENTATION

In order to test and verify our agent based algorithms for surveillance, a software simulation was created. This simulation allowed us to establish quickly the veracity of our algorithms as well as to test individual components and modules of the code. We first created a centralized simulation, where the scouts, sensors and targets all existed as object entities within the simulation. These actors performed all of their duties as children of the central simulation framework. Using this centralized code as a blueprint, a decentralized simulation was generated. Each actor within the simulation was transferred to a separate program that communicated through the network. The decentralized simulation provided a realistic test bed as well as allowed us to quickly bridge our framework onto the robots.

After successfully implementing a distributed simulation of our system, we desired to demonstrate it in a real-world situation. By embedding the distributed components of the simulation into physical hardware, we were able to show that our system is robust and fast enough to control physical systems.

4.1. Software Simulation

4.1.1. Centralized software

The centralized software simulation was created with the Java programming language. This allowed us to take advantage of the of the existing object-oriented API, while not tying the simulation to any particular platform. The centralized simulation was designed for modularity, allowing us to reuse much of the code for the distributed and robot programs. The simulation was comprised of a framework object, global monitor object, scout/sensor object, and target object. The framework object is responsible for all of the graphics, user input, and configuring the scout/sensors and targets.

The framework is the parent to all of the other objects within the simulation. In addition to the user-application interactions, the framework provides the means for the various agents to interact. The framework stores a common priority map of the space, and observation tables. This shared view allows the scouts and sensors to quickly access information about target sightings and global priority map. The target and scout tables are represented with hash-tables using an identification number as the key value. Each key value is associated with the grid location and time of the observation. The priority map is represented by a simulated two-dimensional array. Each cell of the grid is stored as an item in the array, specifically its priority, rate of priority growth, and time or last visit. All of this information is used by the scouts and sensors when they are determining which location to visit next.

The scouts and sensors are represented by the same object class. In our experiments, we have assumed that both the mobile scouts and the static sensors will have the same observation, detection, and processing capabilities. In this case, the sensors are simply immobile scouts. The scouts travel toward their destination cell, detecting sensors, scouts, and targets as they traverse the battlefield. When the scout reaches its destination, the scout selects a new destination based on the updated priority map, hash-tables and its routing algorithm. The cell is only marked as visited if the scout pass over the center of the cell. If a target enters the detection range of the scout, an observation is reported and stored in the hash-table. Friendly observations of other scouts and sensors are recorded as well. All of the scouts use the same priority map and hash-tables, allowing global knowledge of the battlefield. Because the targets identify themselves, the targets are not completely adversarial in the centralized version of the software.

The target moves to random destinations selecting a new random destination once it completes its travel. The target is only visible when it is within the scouts and sensors observation range. The target (once observed by the scout or sensor) is uniquely identified and added to the hash-table and displayed on the global monitor.

The global monitor is responsible for calculating the (α, β) -currency of the globe. Using the global monitor, we are able to determine the effectiveness of the algorithms used in scheduling and surveillance. The global monitor

in conjunction with the framework allows us an optimal view of the battlefield. While the global monitor is not very important in the centralized simulation of the software, it is essential for the distributed simulation.

4.1.2. Distributed software

The distributed software simulation was the next evolution of the test bed. The modular design of the centralized simulation allowed us to migrate the individual objects into stand-alone applications. In the distributed test bed, the scouts, sensors and targets are run as separate applications. These programs simulate radio communications through modified network multicasts. Porting the centralized simulation to a distributed application required us to make several major adjustments and added much complexity. The centralized simulation was broken down into several separate applications.

- **Scout/Sensor:** Standalone application simulating UAVs and ground sensors
- **Target:** Simulated ground vehicle that drives along unpredictable trajectories.
- **Global Monitor:** Monitors communications and determines (α, β) -currency of the world.

The scout and sensor applications are based on the code of the centralized application except now each agent is responsible for maintaining its own tables and priority map. The scouts are now operating independently of other sensors and scouts. Although they are only directly aware of their own observations, agents are able to share observation tables and priority maps with other agents. The sharing of information is vital for the scouts to maximally observe the targets within the space. In addition to sharing information about past observations, redundant destination selection should be avoided.

When two agents enter into communication range each agent makes a connection to the other. The connection are single directional. The agents each maintain a communications server and client. Each agent's client connects to the other agent's server and uploads their hash-tables and current, complete priority map. This single direction communications allows the agents to prioritize their information and send as much data as is possible while both agents are in communication range. An optimal method for agent communications is being developed, but is not yet apart of the test bed. The scouts and sensors are able to communicate further then their observation range. This method of information exchange allows the test bed to scale and operate in a completely distributed manner.

Scouts travel in the same way as in the centralized simulation. The Target travels around the battlefield and is observed by both the scouts and sensors. When the target is observed, the global monitor is notified and the priority map is updated accordingly.

The global monitor provides a near omniscient view of the world. In the distributed simulation, the global monitor needs to be informed of each independent agent's observations and knowledge. In the distributed simulation, the scouts and sensors inform the global monitor whenever an observation is made. The global monitor can then compute the (α, β) -currency on all known information. This (α, β) -currency can be compared against the (α, β) -currency from each agent. By comparing the quality of information of each agent to the baseline value calculated by the monitor, we can measure the performance of the algorithms and validity of the system.

Communications and observations are essential in the distributed simulation. Each application needs to be able to simulate traversing the battlefield and observing and communicating with the other agents. In the distributed simulation each agent is instantiated with the same battlefield map. The scouts and sensors are then given their starting position. The sensors will remain in their location and begin sensing the environment. The scouts will select their destination and begin their search and scheduling. All of the agents, including the target, are a part of the same multicast network. In order to simulate radio communications and observation limitations, each agent multicasts its current position to all of the other agents in the network. Each agent has a guardian function that examines all incoming multicasts. The guardian compares the agent's current position to the position of the sender. If the sending agent is within communication range, the message is accepted. If the sending agent is beyond the communication range, the message is dropped. The target is not able to receive

multicast messages sent by the scouts or sensors. The constant stream of agent positions allows us to gain an omniscient view of the battlefield, something that is impossible in the real world. By comparing the global monitor to the omniscient view, we can see how far we deviate from ultimate knowledge.

4.2. Robotic Simulation

4.2.1. Hardware

In order to simulate the functionality of UAVs in order to test our system, we used four ARTV-Mini robotic platforms, produced by iRobot, Inc.. Each robot is based on a Pentium-III CPU with 128 megabytes (MB) of SDRAM, a full EBX motherboard, a shock-mounted 6.4 gigabyte (GB) hard drive with integrated video, ethernet, and sound, as well as an 802.11b wireless access point. The robots run a modified version of the Linux RedHat 6.2 Deluxe operating system to control the computer and a software package called RFlex to control the embedded hardware. The RFlex software, when combined with the Mobility C++ interface supplied by iRobot, provides a unique programming environment in which all robot components are controlled and monitored as software objects. This object-oriented approach to robot control simplifies integration with our existing software by allowing us to create instances of objects like MotorController, which provides direct access to the motors. The computing power and architecture available on these platforms makes them an attractive option for a test bed for our distributed simulation software.

By taking advantage of the robots' wireless access points, we are able to create a wireless network that allows the robots to exchange information with each other and with sensor nodes. By performing range-limitation in the simulation, we are able to create a realistic model of battlefield communication conditions.

Each robot is also equipped with a variety of sensor devices. The robots each have a Garmin GPS receiver unit, which allows us to determine the robot's location, bearing, and path history very precisely. In the event that the GPS signal is not available, the robot is equipped with a flux-gate compass. Using this compass in conjunction with the robot's wheel encoders allows us to perform dead-reckoning estimates of position relative to a known starting position. Combined with GPS, this provides a good positioning system for the controller. The robot also has an array of ultrasonic sensors that can be used for collision avoidance or local sensing. In the final iteration, our system will use acoustic sensors on each robot and sensor laptop in order to verify our system works in a realistic sensing environment.

4.2.2. Software

In order to control the robots using our existing Java program, it was necessary to design and implement a multi-threaded C++ program to provide a socket interface for sending commands to our robot via the C++ framework provided by Mobility. This interface program opens a TCP/IP socket that accepts commands as strings that follow a specific protocol we designed. The command set includes 'go to position' and 'turn to heading' commands, as well as velocity commands. The socket server also responds to status requests and returns information about dead-reckoned position, heading, velocity, collision danger, and GPS position. In addition, this interface program also performs closed-loop control of the robot. This is necessary in order to implement the pose commands specified in the command protocol. Currently, a simplistic control algorithm has been implemented that slows the robot down as it approaches the specific point or compass bearing towards which it is headed.

The distributed nature of our simulated system made the transition to a physical deployment relatively straightforward. The simulated movements of the scouts were replaced by commands to the robots drive system. The drive system would return the robots GPS position.

In order to provide a more realistic test environment, we left the communications gating in place from the original simulated system. Initially, since all the clients in the distributed system were able to communicate over the network, it was necessary to introduce a gating function that prevented two entities from communicating when they are out of range. This was accomplished on each client by calculating the distance to each peer on the network and dropping packets from peers that exceeded some threshold. By leaving this limitation in place, we are able to use a smaller space as a scaled version of a large battlefield.

Another crucial consideration was how clients should communicate as they move past each other. There are many factors in this decision, such as limited bandwidth, limited time to communicate, limited fidelity on the channel, and security of transmissions. In the initial iteration, the naïve approach was taken where each client sent as much of its scout tables, target tables, and priority maps as possible before the robots left communication range. In later implementations, we can improve the efficiency of this communication by addressing some or all of the limiting factors mentioned above. For example, in the case where the entities have limited time to communicate, the ideal communication protocol would have a robust system of prioritization that makes sure the most important information is exchanged first. In the case where there is limited bandwidth, the best protocol would send the most information in the fewest number of bytes. Many other scenarios can be imagined, and an optimal communication protocol will address as many of them as possible.

Another factor in the physical implementation of the system is the problem of sensing. For the software simulation, we used the entities’ position update broadcasts to decide whether scouts could see targets using a method similar to the gating we did for limiting communication and observation range. In our physical system, this simulated ranging is replaced with an acoustic sensor. When the sound intensity crosses some threshold value, the sensor is said to have made an observation in that cell.

4.2.3. Design of Experiment

In order to test our software, a series of experiments were performed on a set of test bed systems. Each sensor client ran on a laptop computer, while the scout and target clients were run on the built-in computers in the iRobot ATRV-Mini robots. For the purposes of our software simulation, we used a setup with one target, three scouts, and three sensors. In each setup we were able to record the α value of the system over time to a file using the global monitor. The simulations were run until the α reached a steady state value. In one experimental setup, we varied the β value and recorded the α over time. Another experiment involved adjusting the communication range of the scouts and sensors and recording the α as time progressed. Before recording the α values, several tests were completed to determine the time necessary to reach steady state. The battlefield was configured as a 10 by 10 grid of cells.

5. EXPERIMENTAL ANALYSIS

Several experiments were completed with different β values to determine the effect on α . Intuitively, we expect the α values to increase as the system is given a larger and larger grace period. The scouts in this simulation had an observation range of 100 units and a communications range of 150 units. These values allowed the scouts and sensors to observe targets 1 cell away, and to communicate to up to 2.5 cells away. In addition to yielding greater mean α values in the steady state, the longer grace period also prevented the α value from dropping until after β time units had elapsed. This was also expected, as any change to the world during the first β units is not taken into account until the grace period expires.

Table 1. Mean α values for different β times

β Value	Mean α
10	0.45906924
100	0.4818092
500	0.61439877
1000	0.76558679

When the system was run with a β value of 1000, the ability of the sensors and scouts to maintain an accurate, distributed view of the world was impressive. Even though the sensors and scouts had a limited observation and communication range, they were able to maintain reliable accounts of the global state.

In addition to varying the β values, experiments were conducted with different communication ranges. Changing the range of communications allows the scouts and sensors to exchange their distributed tables and priority

maps more or less frequently. Several runs were recorded with communication ranges ranging from a single cell to the entire battlefield. When the communication range was set to cover the entire battlefield, the distributed agents approximated the centralized simulation, since each scout and sensor was in constant contact with every other scout and sensor. The observation range of the scouts and sensors was kept constant at 100 units. The communication range of 600 units allowed the scouts to communicate with every scout due to the placement of the sensors. With such a large communication range, the sensors acted as relays, passing messages to each other and to passing scouts.

Table 2. Mean α values for difference Communication Ranges

Communciation Range	Mean α
100	0.49916573
150	0.41969182
200	0.47584511
250	0.46952354
300	0.48807955
600	0.45740783

Table 2 demonstrates that it is possible, in principle, to trade communication range with the number of deployed UAVs (the extreme case being a sufficiently high number of UAVs and sensors that cover all the observable cells with null communication range but reporting observations directly to a base-station). In this particular experiment three UAVs and three sensors were deployed in order to monitor a system perturbed only by one target. The value of α remained essentially stationary as the communication range varied. When the communication range was limited to the size of the observation range, the scouts and sensor rarely exchanged information. Each scout and sensor acted independently, with the result that the scouts created priority maps that approximated a normal distribution centered on the last known observation of the target. Currently, we are performing experiments whereby the number of UAVs and sensors becomes comparable or smaller than the number of targets in order to better appreciate fluctuations of α as the communication range is varied. The purpose of this analysis would be to determine the communication range such that a desired $\alpha \geq \alpha_0$ for given β, α_0 .

6. CONCLUSIONS AND FUTURE WORK

- Our scheduling algorithm appears operatively effective in so far as sensors and UAVs are able to cooperate in order to produce a nonnegligible probability that the system is β -current. Moreover, this condition seems to match our intuitive notion of surveillance and tracking, namely UAVs concentrate their patrols around areas that have high probability of containing an already spotted target.
- (α, β) -currency seems consistent with our intended quantification of the situational awareness of the system. In particular the values of α increases toward 1 as β increases.
- Changing the communication range while keeping β constant does not necessarily affect α . For example, in Table 2, α is not monotonic as one may expect because a preponderant deployment of surveilling resources (three UAVs and three targets) was committed to intercept a single target. This resulted into an overcompensation for poor communication range that has flattened an appreciable fluctuation of α .
- New experiments, in progress, are involving several targets, evenly distributed on the battlefield and few UAVs to verify how α quantifies the general coordination of the UAVs and sensors as we expect. This trade-off analysis aims at determining critical communication ranges.

- All these experiments described in the above paragraphs are only the starting point to investigate communication energy versus navigation energy trade-offs. In other words, what happens if we measure the global energy spent by the UAVs and by the sensors to exchange their data and also to navigate in order to produce a fixed number of target detections and maintain $\alpha \geq \alpha_0$ for given α_0 and β ? It is intuitive that there is a correlation between the consumption of energy spent in navigation and the amount of coordination attained through data exchanges.

The system currently assumes the targets identify themselves to the scouts and sensors. This is not a reasonable assumption to make in a realistic battlefield environment. In order to disambiguate the target sightings and correlate them into tracks, it becomes essential to use a system for evaluating process models. The Process Query System developed at Dartmouth [20] would provide the ability to identify and track adversarial targets. More work needs to be done for the use of a PQS system to provide feedback to scouts based on the evaluation of metrics like (α, β) -currency. In this case, scouts and ground sensors form a large distributed sensor system that reports/publishes events to the PQS. PQS in turn processes the gathered information, evaluates (α, β) -currency, and computes better priority tables for scouts.

ACKNOWLEDGMENTS

The authors would like to thank Professor George Cybenko for his invaluable suggestions and guidance that have made this work possible. Research partially supported by the DARPA TASK program under grant number F30602-00-2-0585. Points of view in this document are those of the authors and do not necessarily represent the official position of the sponsoring agencies or the U.S. Government.

REFERENCES

1. L. Caffarelli, V. Crespi, G. Cybenko, and I. Gamba, "On Stochastic Distributed Algorithms for Target Surveillance," *Unpublished manuscript*, August 2003.
2. L. Caffarelli, V. Crespi, G. Cybenko, I. Gamba, and D. Rus, "Stochastic Distributed Algorithms for Target Surveillance," in *Proceedings of the ISDA2003, Tulsa, Oklahoma*, August 2003.
3. V. Crespi and G. Cybenko, "Agent-based Systems Engineering and Intelligent Vehicles and Road Systems," *Darpa Task Program white paper*. <http://actcomm.thayer.dartmouth.edu/task/>, Apr. 2001.
4. G. Cybenko, "Agent-Based Systems Engineering," *Darpa Task Program Research Proposal*. <http://actcomm.thayer.dartmouth.edu/task/>, Oct 2000.
5. B. Brewington, "Observation of Changing Information Sources," *Ph.D. Dissertation, Dartmouth College*, 2000.
6. V. Crespi, G. Cybenko, and D. Rus, "Decentralized Control and Agent-Based Systems in the Framework of the IRVS," *Darpa Task Program paper*. <http://actcomm.thayer.dartmouth.edu/task/>, Apr 2001.
7. V. Crespi, G. Cybenko, D. Rus, and M. Santini, "Decentralized Control for Coordinated flow of Multiagent Systems," in *Proceedings of the 2002 World Congress on Computational Intelligence. Honolulu, Hawaii*, May 2002.
8. V. Crespi and G. Cybenko, "Decentralized Algorithms for Sensor Registration," in *Proceedings of the 2003 International Joint Conference on Neural Networks (IJCNN2003), Portland, Oregon*, July 2003.
9. W. Feller, *An Introduction to Probability Theory and its Applications*, John Wiley & Sons, Inc, Publishers, 1957.
10. I. Karatzas and S. Steven, E., *Brownian Motion and Stochastic Calculus (2nd edition)*, Springer Verlag.
11. Y. K. Hwang and N. Ahuja, "Gross Motion Planning – A survey," *ACM Computing Surveys* **24**, pp. 219–291, Sept. 1992.
12. J. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
13. E. Rimon and D. E. Koditschek, "Exact Robot Navigation using Artificial Potential Functions," *IEEE Transactions on Robotics and Automation*, October 1992 **8**(5), pp. 501–518, 1992.
14. C. I. Connolly and J. B. Burns, "Path Planning using Laplace's Equation," in *Proceedings of the International Conference on Robotics and Automation*, pp. 2102–2106, 1990.

15. D. Keymeulen and J. Decuyper, "The Fluid Dynamics Applied to Mobile Robot Motion: The Stream Field Method," in *Proceedings of the International Conference on Robotics and Automation*, pp. 378–386, 1994.
16. P. Gupta and P. R. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory* **IT-46**, pp. 388–404, March 2000.
17. P. Gupta and P. R. Kumar, "Internets in the Sky: The Capacity of Three Dimensional Wireless Networks," *Communications in Information and Systems* **1**, pp. 33–49, January 2001.
18. P. Gupta and P. R. Kumar, "Towards an Information Theory of Large Networks: An Achievable Rate Region," *IEEE Transactions on Information Theory* **49**, pp. 1877–1894, August 2003.
19. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley Interscience, 1991.
20. V. Berk, W. Chung, V. Crespi, G. Cybenko, R. Gray, D. Hernando, G. Jiang, H. Li, and Y. Sheng, "Process query systems for surveillance and awareness," in *7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)*, Orlando, Florida, pp. 27–30, July 2003.