IEEE Copyright Statement:

Distributed Control Applications Within Sensor Networks

Bruno Sinopoli, *Student Member,IEEE,* Courtney Sharp, Luca Schenato, *Student Member,IEEE*
Shawn Schaffert, *Student Member,IEEE* and Shankar Sastry, *Fellow,IEEE*

Robotics and Intelligent Machines Laboratory, UC Berkeley, Berkeley, CA, USA

{sinopoli,cssharp,lusche,sms,sastry}@eecs.berkeley.edu

Sensor networks are gaining a central role in the research community. This paper addresses some of the issues arising from the use of sensor networks in control applications. Classical control theory proves to be insufficient in modeling distributed control problems where issues of communication delay, jitter, and time synchronization between components are not negligible. After discussing our hardware and software platform and our target application, we review useful models of computation and then suggest a mixed model for design, analysis and synthesis of control algorithms within sensor networks. We present a hierarchical model composed of continuous time-trigger components at the low level and discrete event-triggered components at the high level.

***Index Terms*— distributed control, sensor network, DPEG, pursuit evasion game, TinyOS, NesC, mote, Mica, embedded**

I. Introduction

Sensor Networks (SN) are gaining a role of importance in the research community. Embedded computers are well settled in our lives, in our houses, in our cars, and in our work environments.

Embedded systems, by definition, interact with the physical world. They are sensors, actuators, and controllers which are programmed to perform specified functions. As the range of applications grows, the need arises to network several embedded systems to perform incrementally complex tasks. The automotive domain is an excellent illustrative example. Here several embedded systems interact to provide a safe, comfortable driving experience.

Recent developments in MEMS technology have provided us with a wealth of cheap, customizable, embedded sensor systems capable of wireless communication among each other. The advantage of wireless sensor networks is enormous – deploying and maintaining a network of thousands of nodes is impractical considering the thousands of miles of wire that would be needed for the connections. Several hardware platforms are available, developed by both startups [1], [2], [3] and universities [4].

Applications in various fields of research are being developed. Interesting ongoing projects include extensive experimentation of structural response to earthquakes [5], habitat monitoring [6], and intelligent transportation systems [7]. Other important fields of applications include home and building automation, and military applications. Self configurable, ubiquitous, easy to deploy, secure, undetectable sensor networks are an ideal technology to employ in intelligence operations and war scenarios for detecting movements of enemy troops and artillery, and for monitoring and managing friendly resources.

The research community has quickly acknowledged the importance of large scale ad hoc networks, and developed several services to support applications. Time services [8] provide the network with a globally consistent notion of time, localization services [9] allow computing nodes to acquire their coordinates relative to each other, routing services [10], [11] reliably deliver packets while dynamically adapting to the ever changing network topology, and tracking services [12], [13] follow objects moving through the network.

System design and implementation on such a versatile platform introduces a series of issues. The longevity of these networks requires a stable software platform capable of self configuration, self upgrade, and adaptation to changing environmental conditions. Another set of issues arises when a sensor network is used for control applications. This is the thrust of this paper. Throughout our discussion, we will see several keys issues presenting themselves time and time again: location determination, time synchronization, reliable communication, power consumption management, cooperation and coordination, and security.

The goal of our research is to design robust controllers for distributed systems that violate typical control assumptions. Designed controllers will be evaluated on a distributed control application testbed. Among the wealth of available applications, we have selected a pursuit evasion game (PEG) application. In our particular application, the sensor network is deployed in the environment where the game is played and cooperates with the pursuers' team.

This application includes many interesting research problems in the areas of tracking, control design, security, and robustness. For a PEG, the sensor network must be capable of multiple vehicle tracking that can distinguish pursuers from evaders. Furthermore, the network needs to have a dynamic routing structure to deliver information to pursuers in minimal time. Since the game will be played in a distributed fashion, distributed sensing, control, and actuation need to be accounted for during controller design. To prevent the evader's team from intercepting sensitive information, the network must provide security features. Finally, since any one node of a sensor network can fail, control algorithms should

This research is supported by DARPA under grant F33615-01-C-1895.

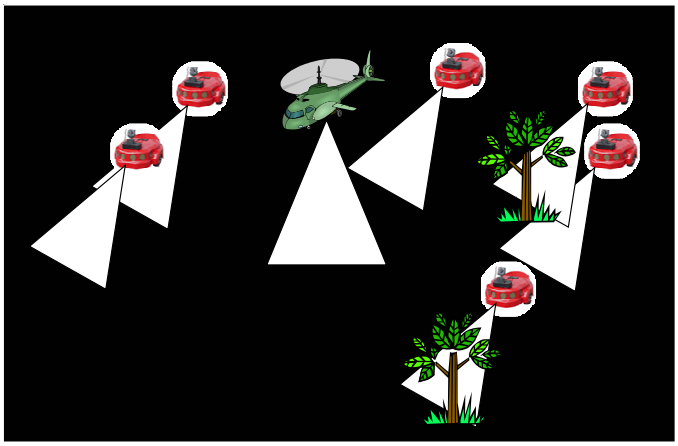


Fig. 1. Pursuit-Evasion Game: what pursuers really see.

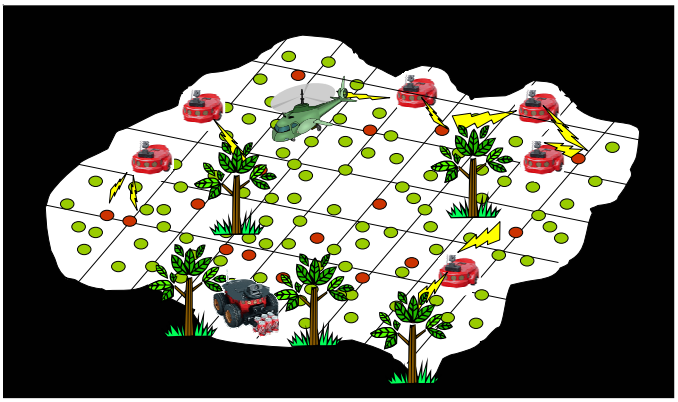


Fig. 2. Pursuit-Evasion Game: sensor network increases visibility.

show graceful performance degradation.

II. Pursuit Evasion Games

The framework of PEGs captures fundamental features for modeling multi-agents in cooperative robotics and has been an active area of research in the past decades. In this section, we give a brief overview of the research history on PEGs, describe the advantages of adding SNs to standard PEGs, and enumerate additional issues that arise when using SNs.

A. PEG Overview

Pursuit-Evasion Games (PEGs) are a mathematical abstraction arising from numerous situations which addresses the problem of controlling a swarm of autonomous agents in the pursuit of one or more evaders. Typical examples are search and rescue operations, surveillance, localization and tracking of moving parts in a warehouse, and search and capture missions. In some cases, the evaders are actively avoiding detection as in capture missions, whereas in other cases their motion is approximately random as in rescue operations.

Different versions of PEGs have been analyzed according to different frameworks and assumptions. Deterministic PEGs on finite graphs have been extensively studied [14], [15]. In these games, the playing field is abstracted to be a finite set of nodes and the allowed motions for the pursuers and evaders are represented by edges connecting nodes. An evader is captured if both the evader and one of the pursuers occupy the same node. One of the most important problems arising from this framework is the computation of the *search number*, i.e., the smallest number of pursuers necessary to capture a single evader in a finite time, regardless of the escaping policy adopted by the evader. It has been shown that this problem is NP-hard [15], [16]. This approach is limited only to worst case motions of the evaders, and it is in general overly pessimistic. A great deal of research has focused on how to reduce a continuous space into a discrete number of regions, each to be mapped into a node of the graph, so that the game on the reduced graph is equivalent to the original game in the continuous space. For example LaValle *et al.* proposed a method of decomposing the continuous space into a finite number of regions for *known* polygonal environments [17] and simply connected, smooth-curved, two dimensional environment [18].

Another active area of research deals with PEGs where the environment is *unknown*. In this framework, an additional *map-learning* phase is required to precede the pursuit phase. The map-learning phase is, by itself, time-consuming and computationally intensive even for simple two-dimensional rectilinear environments [19]. Moreover, inaccurate sensors complicate this process and a probabilistic approach is often required [20].

Finally, a recent approach to PEGs has dealt with combining map-learning and pursuit into a single problem. This is done in a probabilistic framework to avoid the conservativeness inherent in worst-case assumptions on the motion of the evader. A probabilistic framework also naturally takes into account inaccurate sensor readings, uncertain *a priori* map of terrain, and evaders motion policies [21], [22].

B. Sensor Networks in PEGs

The use of a sensor network can greatly improve the overall performance of a PEG. Pursuers have a relatively small detection range. They usually employ computer vision or ultrasonic sensors, providing only local observability over the area of interest. This constraint makes designing a cooperative pursuit algorithm harder because lack of complete observability only allows for suboptimal pursuit policies. See Figure 1. Furthermore, a smart evader is difficult to catch unless appropriately detected.

Communication among pursuers may be difficult over a large area. Lack of communication, even partially, among pursuers is a major disruption for any pursuit policy. Because of the expense of unmanned vehicles, it is unrealistic to deploy a large number of them to continuously monitor a large region.

With sensor networks, complete visibility of the field and communication over a long radius is possible. See Figure 2. Global pursuit policies can then be used to efficiently find

the optimal solution regardless of the level of intelligence of the evader. Also, with a sensor network, the number of pursuers needed is likely a function exclusively of the number of evaders and not to the size of the field.

This DPEG scenario exposes a number of issues fundamental to any sensor network. Resolving these issues is complicated by the desire to make the solutions robust even in a dynamic, ad-hoc network.

Time – The notion of time presents two distinct problems. First, coordinating sensing and actuating in the physical world requires either a sense of global time or the ability to resolve different time measurements to a meaningful representation. Second, many existing design techniques assume that the computation of control and the processing of sensing and actuation occur within a negligible amount of time; thus, requiring new design and analysis techniques for sensor networks.

Communication – It is expected that a network of motes will span a spatial area significantly greater than a single mote's maximum communication area. For a mote to send a message to another, distant mote, intermediate motes must be able to relay the message. Additionally, because motes can go offline without warning, the underlying communication protocol must be robust to network changes.

Location – Sensing and actuating events in the physical world must be paired with the relative or absolute location of the mote to be useful to control algorithms. That location must be assumed, provided, or deduced.

Cooperation – Tasks that require the combined effort of two or more motes, such as any form of distributed sensing or distributed computing, require protocols and structures that provide handshaking, coordination, and possibly hierarchy.

Power – Energy is a valued resource in a sensor network. Service and performance guarantees provided by a sensor network must be balanced against overall power consumption.

Security – To prevent numerous potential abuses of a sensor network, a communication security layer must provide known guarantees for access control, message integrity, and confidentiality.

When developing control applications on a sensor network platform, we are particularly interested with issues related to time, communication, and location. We will focus on these issues throughout.

C. Distributed PEGs

To start our distributed pursuit evasion game (DPEG) scenario, the motes comprising the sensor network are deployed onto the playing field in a sleep state. The mote sensor network then goes through an initialization and calibration stage for bootstrapping their provided services. The pursuers and evaders then enter the playing field and remain within the field for the duration of the game.

The sensor network provides a variety of services to both pursuers and other sensor motes: time synchronization, localization, moving entity (pursuer or evader) estimation, etc. For the purpose of the game, the sole goal of these services is to produce estimates on the positions, velocity, and identity of entities in the playing field. This information is time stamped and routed to all pursuers in the playing field. The pursuers have onboard computation facilities comparable to a laptop computer. We may choose to have the pursuers communicate through a separate robust channel to coordinate to capture the evader when and if that channel is available.

When all evaders are captured (a capture occurs when a pursuer is "close enough" to it), the game ends. A base station is outside the playing area and provides logging and visualization services.

Mote Type	WeC	Rene	Rene2	Dot	Mica
					
Date	Sep-99	Oct-00	Jun-01	Aug-01	Feb-02
Microcontroller (4MHz)					
Type	AT90LS8535		ATMega163		ATMega103/128
Prog.Mem.(KB)	8		16		128
RAM (KB)	0.5		1		4
Communication					
Radio	RFM TR1000				
Rate (Kbps)	10				10/40
ModulationType	OOK				OOK/ASK

Fig. 3. Evolution of motes from the Berkeley TinyOS group.

III. Implementation

Our implementations span hardware, software, and various application scenarios to explore and demonstrate distributed control via sensor networks. In the hardware section, we discuss our current embedded network devices. Then, in the software section, we review our new programming language, operating system, and system service architecture. Finally, we survey our current and future testbeds for interacting and learning at the whole-system level.

A. Hardware

The hardware platform developed by the TinyOS group at Berkeley consists of numerous, small, extendible embedded network devices. Each device has limited power, computation, and storage resources – significantly limited when compared to modern desktop computer systems. The goal of each hardware platform is to provide computation, sensing, actuation, and communication resources embedded in miniature packaging. By making the conscious design decision to significantly limit the resources available per mote, we leave the door open for reaching the goal of dust-sized devices.

The current platforms are designed to be both modular and flexible; providing ease in re-targeting motes to new and unanticipated applications while allowing for significant code reuse. Figure 3 shows the evolution of the base computation modules. In particular, the most recent transition from the Rene2/Dot to Mica (Figure 4) gave at least a four-fold increase in program memory, RAM, and radio transmission

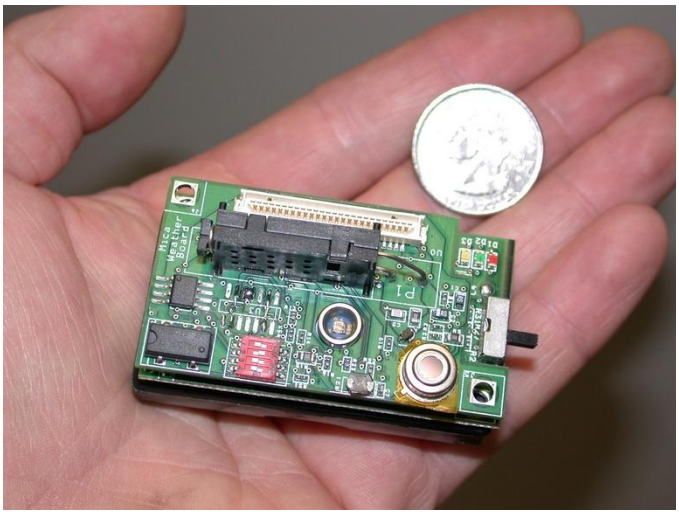


Fig. 4. Mica mote ($2'' \times 1'' \times \frac{3}{4}''$) with attached weather board module.

rate. All motes have otherwise had some form of an 4 MHz, 8-bit Atmel microcontroller and an RFM TR1000 radio.

Add-on boards for the motes may be designed for general purpose sensing or targeted toward a particular application. For instance, the weather sensing board has humidity, barometric pressure, infrared, temperature, and light sensors and is used for experiments on Great Duck Island [23] in Maine. And, a motor/servo and whisker/accelerometer board were developed for COTS-BOTS [24] (Figure 5) for controlling off-the-shelf miniature cars. We also have various general-purpose sensor boards that have some combination of photodiodes, temperature sensors, magnetometers, accelerometers, microphones, and sounders.

The overall modularity of these devices comes at the cost of size. A device targeted at large-scale deployment can do away with the add-on connector and supporting circuitry. The resulting space savings in the current platforms easily allows for a final form-factor with diameter smaller than a quarter.

All together, the hardware platforms have been sufficient to meet the needs of both research and experimentation.

B. System services

We build our embedded software with NesC [25], a new, open-source programming language developed at Berkeley. NesC extends the standard C language with semantics and syntax for component-based architectures. Component behaviors are described with bidirectional interfaces that either provide commands or require the dependent to handle events. Components are statically wired together to form a whole program or system; which when compiled with a whole-program compiler, allow for greater optimizations and efficiency. Whole programs also match well with formal analysis tools for verifying system functionality.

Berkeley's open-source embedded operating system, TinyOS [26], provides basic system services, such as communication and simple process scheduling, and access to

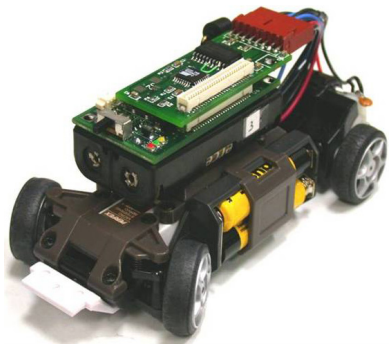


Fig. 5. COTS-BOTS developed by Sarah Bergbreiter and Kris Pister.

hardware components, such as sensor and actuators. It is specifically designed for extremely resource-limited devices that have only a few kilobytes of memory. TinyOS is written in NesC using a component-based architecture with layered access to hardware resources, which provides robustness, flexibility, and extensibility.

Using NesC and TinyOS as building blocks, we have been working with a number of other groups on the NEST project funded by DARPA to develop a coherent architecture of system services to help solve fundamental sensor network stumbling issues. The crucial services we have currently identified are estimation, grouping, localization, power management, routing, service coordination, and time synchronization. We feel that these components will facilitate a large set of rich and adaptive applications.

To address time issues within a sensor network, we propose a time synchronization API that supports two time management protocols: a global NTP-like synchronization protocol, and a local time protocol with the means to transform time readings between individual motes. It is expected that NTP-like global synchronization will offer lower precision time measurements, but otherwise provide an immediately available global time on the mote. Local transformations between individual mote "time-zones" has the advantages of higher precision between pairs of motes, being able to back-calculate synchronized times for past events, and guarantees monotonicity in local time by not directly modifying the local clock [8]. Various applications can have vastly different time synchronization requirements, and we feel these two methodologies together can more adequately serve a broad set of applications.

To address communication issues within a sensor network, we propose a general routing framework that supports a number of routing methodologies. First, because sensor networks primarily sense and interact with phenomena in the physical world, routing to geographic regions is expected to be the common-case. Second, to assist in routing packets around physical obstacles, routing based on geographic direction is expected to be useful. Third, the more obvious case of routing to symbolic network identifiers is reserved for dynamically routing to physically moving destinations within the network. Finally, the general-case of constraint-based routing provides

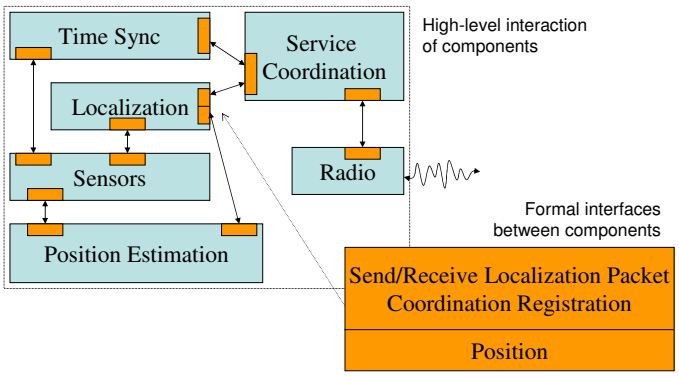


Fig. 6. Sample component architecture demonstrating the design methodology.

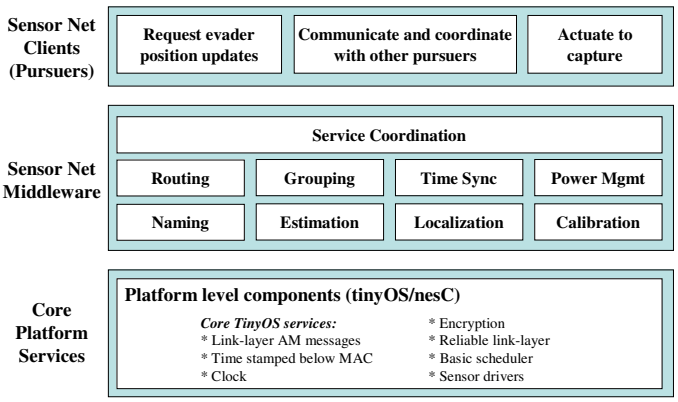


Fig. 7. Relationship between proposed services and components.

means to route based on arbitrary criteria, such as power level, sensor values, and so on.

To resolve the physical location of motes in a sensor network [9], we propose a top-to-bottom localization framework. A localization service requires a broad set of coordination and processing stages between motes: coordinated sensors and actuators, group data management, and computation. Separating localization into a number of distinct components that work together allows for an amount of heterogeneity in the sensor network that may be necessary given the limited resources of the motes.

To address the issues of coordination between motes, we propose both application targeted grouping algorithms and general purposing grouping services. A group management service must provide means to send and receive data from a group, the ability to join and leave a group, and leader election. For tracking a moving evader in a PEG scenario, decisions to join and leave groups can be tied to sensor readings. This simplifies the handshaking and decision process, allowing for overall lower overhead. There are concerns that these services in the general case impose significant overhead on a sensor network.

Issues of power management are on the agenda but are currently unaddressed in the architecture. Issues of security are being solved at the operating system level, providing transparent authentication, encryption, and concealment.

Our methodology for creating an infrastructure for these services is to first specify a set of *prototypes* that define abstract programming interfaces for classes of components and services. Developers then create *components* that instantiate an *algorithm* using one or more prototypes. Some components may behave as *services* whose execution and behavior are managed by a central coordinator. Finally, interaction between components must be formalized by specifying *protocols* and *types*. Figure 6 illustrates this methodology in a sample architecture that shows the interactions and protocols between components, services, a service coordinator, sensors, and radio channel.

Figure 7 further shows the relationship between these services, components, TinyOS, and a DPEG application layer.

C. Testbeds

Our current experimental platform is functional but limited when compared to the scope of a full DPEG scenario. It is the result of a focused effort to produce a solution for a set of particular goals rather than to provide a general framework. To that end, it exists more as a proof that a highly constrained DPEG solution is achievable and that NesC and TinyOS provide a suitable platform for development.

Figure 8 shows the setup for that platform. A human remotely controls a miniature car, and the sensor network remotely controls a pan-tilt-zoom camera to track the car. Because we have not yet integrated a self-localization service on the motes, the sensor network is a uniform grid of 25 motes, where each mote presumes its location given its network address. Each mote shares its location with its local neighborhood, which is necessary both for position estimation and geographic-based routing. When a mote detects change in its local magnetic field, it broadcasts its readings to its local neighbors and records similar broadcasts from other, nearby motes. In this way, local behaviors are expected; we are currently not attempting to aggregate readings from the entire network to produce a single, global estimate. The mote with the highest reading is implicitly elected the leader, who calculates a position estimate from its cached neighborhood readings. That estimate is sent via reliable geographic-based multi-hop routing to a base station mote, which relays it to a camera mote. The camera mote performs the actuation necessary to point toward the estimated location.

What we would like to do is to use the sensor network software architecture to implement this scenario in a more versatile, general framework. We are looking forward to a more complete, outdoor PEG scenario, shown in Figure 9. Beyond that scenario, we look forward to expanding our understanding of whole-system behavior through formalism and parameterization of distributed sensor networks.

IV. Methodology

Our initial DPEG implementation has provided valuable insight into the pressing issues that a control design methodology must address, and we will use these ideas to inform our

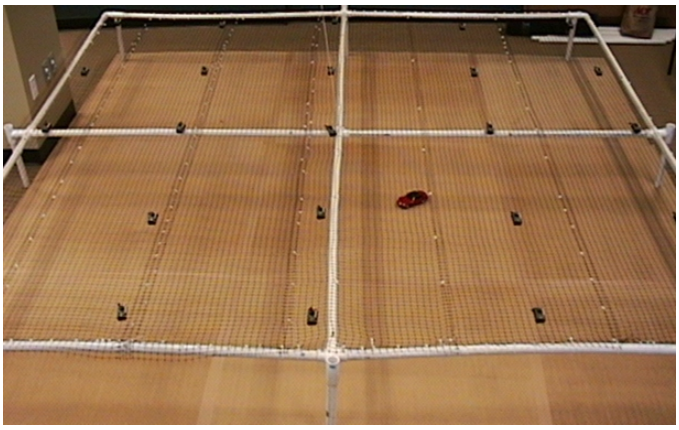


Fig. 8. Indoor sensor-based tracking testbed.

Fig. 9. Outdoor DPEG testbed.

proposed design methodology. In this section, we will first review existing design approaches that address the issues of scalability and distribution. During this discussion, we will be interested in extracting the essence of existing algorithms while abstracting away the particular choice of model. Following this, models of computation will be explored that are useful for describing such systems. Finally, we will discuss our proposed design methodology that will be applied to the next DPEG implementation.

A. Scalability and Distributed Control

Distributed control systems are an integral part of our world and have been studied in many different contexts ranging from biology to artificial intelligence to control systems. Naturally occurring distributed systems such as ants searching for food, bacteria foraging, and the flight formations of some birds have been well studied by biologist and are beginning to receive more attention from other communities interested in distributed algorithms. Indeed, the artificial intelligence community has considered such systems in more abstract terms for several years. Additionally, the continuous time control community has addressed many of the features that distinguish distributed control systems from classical centralized control systems.

Nature provides us with several good examples of distributed control in action. For example, schooling in fishes [27] and cooperation in insect societies [28] exhibit complex collective patterns arising from rather simple individual behavior. These social behaviors have been argued to improve food search, predator avoidance and colony survival for the species as a whole rather than for the individual. Some researchers have been turning to such examples to gain insight into these naturally optimized distributed algorithms. Investigating bacteria foraging of E. coli, Passino [29] has developed a distributed optimization algorithm. The algorithm models how E. coli bacteria move in a solution as they collectively search for nutrients and avoid toxins to reach an optimal state where the collection of bacteria is satisfied with their surroundings.

The artificial intelligent community has addressed such systems under the title of distributed agents for several years [30]. Some researchers in this community have developed approaches such as free market systems [31] that mimic our own trade system. In this architecture, each agent, which could be a robot with a specialized ability, bids on a particular task based on its cost function which combines the robot's reward and effort. It is even possible for robots to become leaders who bid on tasks and then subcontract the task out to several other robots.

The continuous control community has wrestled with distributed systems for many years in the realm of process control, and has independently addressed many of the caveats of distributed systems such as jitter compensation and scheduling. Martí et al [32] have identified the types of jitter that can occur in distributed systems and investigated compensation techniques. Their method first analyzes whether on-line or off-line compensation is needed. If on-line compensation is feasible, then the parameters of the control law are dynamically updated according to the next time the controller will be executed. Other researchers have reformulated the typical scheduling problem as a dynamic system so that the techniques of control theory may be applied [33]. In [34], a centralized scheduling rule is replaced with local instantiations of integral controllers that are shown to drive the state to a viable solution.

B. Models of Computation

The impossibility of characterizing these systems within the classical control framework raises the need to select one or more models of computation (MOCs) in order to accurately analyze distributed control problems in sensor networks. Our hope would be that such a combination captures the continuously changing dynamics of the environment, the distribution of resources, and the discrete nature of the hardware. To address this issue more specifically, we investigate several common MOCs, including discrete event, continuous dynamical systems, discrete-time dynamical systems, hybrid automata, synchronous reactive languages, and data-flow models. For each of these, we consider its advantages and its drawbacks with respect to control applications within sensor networks.

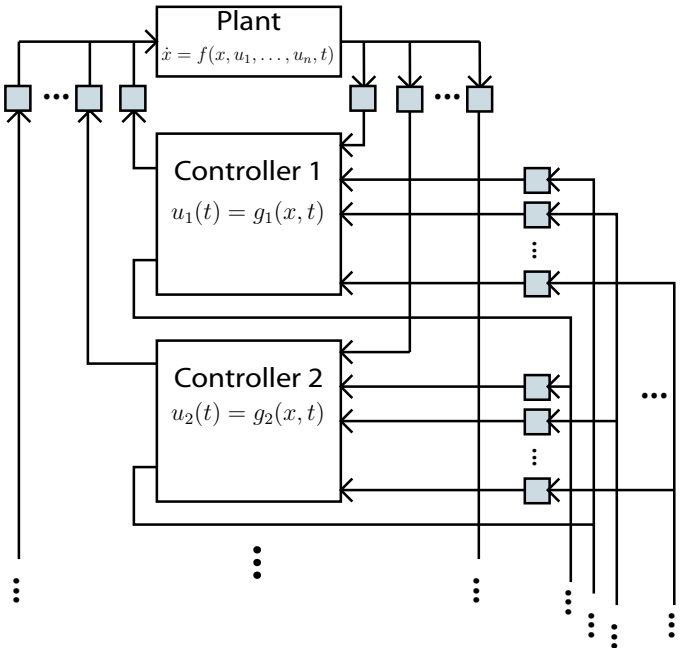


Fig. 10. Section of a distributed continuous control MOC with sensing, actuation, and communication jitter. Shaded blocks represent a time delay.

Continuous time dynamical systems [35], [36] are a well studied formal model. Key properties such as stability and reachability can be deducted using available analytical and numerical methods. Controllers can be designed to meet desired specifications. Additionally, they are familiar to the control community, and hence preferred for control applications. However, for distributed control applications in sensor networks, this theory is not able to capture communication delays, time skew between clocks, or discrete decision making. Since all the variables are continuous, it is difficult to model such discrete phenomena. Additionally, controllers must be implemented on microprocessors, and control must be piecewise constant.

To describe the controller's piecewise constant nature, we turn to *discrete time dynamical systems* [35], [36]. However, we are again limited to characterizing systems without mode changes. Additionally, this MOC assumes periodic activation of the controller with instantaneous computation of the control law which is not preserved by the underlying platform. This model does not directly address sensing and actuation jitter, but it can be taken into account by augmenting with time delays between the plant and the controller. This approach assumes that the control law is computed synchronously on each node every T seconds, but different sensing and actuation jitters are allowed for each node. This model is useful when we assume that the process scheduler running on each node can ensure synchronous operation. Additionally, the system can be modified to distribute control computation across nodes with state communication between them, as shown in Figure 10.

The multi-modal nature of such systems can be described by a *hybrid automaton* [37]. These systems nicely account for both the "continuous flow" and discrete jumps of such systems. Note, that "continuous flow", or just flow, in a hybrid automaton may be modeled by either differential equations or difference equations. They allow the system to evolve according to the flow with occasional discrete transitions. Additionally, with each discrete transition, the equations governing the flow are allowed to change. Difference equations allow such a model to capture the piecewise constant nature of the controller. Mode changes can then be characterized by the discrete dynamics, where all the discrete properties of our application must be encoded. The discrete dynamics are similar to finite state machines in that encoding many discrete variables leads to a discrete state explosion problem and quickly becomes unmanageable for sensor networks.

To consider MOCs more appealing for algorithms, we can consider *discrete event systems* [38]. Such a model works well for mode changes or task scheduling and characterizes the hardware platform nicely, as well. It also allows for the system to be event-triggered, which is often the case in sensor networks. However, it does not support continuous variables, and given the discrete nature of variables we again run up against a state explosion problem when modeling a large number of nodes. Finally, such systems generally do not correlate time-steps of the model with real-time.

Dataflow [39] MOCs are intended to describe data transformations. In particular, they are useful for characterizing several communicating processes. However, this paradigm is awkward for control since it generally considers the relationship between sequences of inputs and sequences of outputs, rather than the evolution of the output for each input signal in turn. In general, when composing several dataflow models in a feedback loop, the result may not be deterministic [40].

Another set of common modeling paradigms are *synchronous reactive languages*, such as Signal [41], Lustre [42], and Esterel [43]. These languages support a broad range of formal verification tools to aid in debugging. Additionally, it is possible to generate code for the platform directly from the synchronous reactive language. However, we again find that there is no relation between time-steps of the language and real-time. Furthermore, synchronous reactive systems presume the existence of a global clock and that time-steps, and hence the computation of fixed points, happen instantaneously. This MOC is not appropriate because it is not congruent with the event triggered nature of sensor networks. Finally, such a model can be counter-intuitive since it searches for a fixed point at every step.

C. Design Approaches

In the previous two sections we described different approaches to address scalability and synchronus/asynchronous systems. Our approach to scalability for DPEGs will rely heavily on distributed processing of sensors readings in order to get good estimates of positions and velocities of both evaders and pursuers. The control of each pursuer dynamics is performed within the pursuer itself based on network

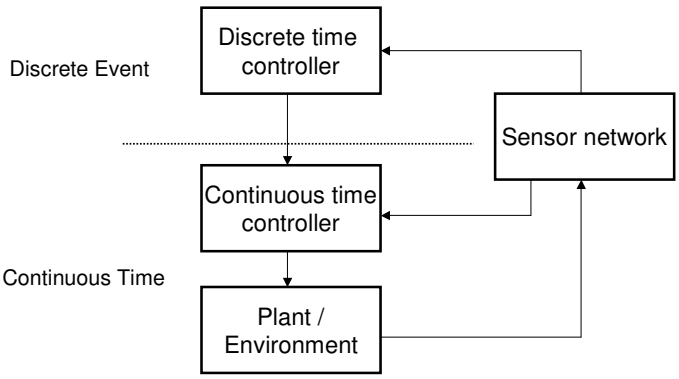


Fig. 11. A hierarchical system representation

readings, but higher level coordination will be distributed between all the pursuers to maximize robustness to adversarial attack. In order to address the issues arising from the fact that DPEGs include synchronous and asynchronous dynamics, several ad-hoc solutions are available. To compensate for nonuniform time-delays, one approach is to buffer the incoming data for a certain amount of time such that most of the data has arrived. With this approach, the problem has been reduced to the classical control problem of driving a system with a fixed time delay. However, this result comes at the price of suboptimal performance. As for missing data, the most common solutions are either using the most recent data regardless of its exact time of arrival, or estimating the most probable measurement that is consistent with previous measurements and the dynamics of the system.

Some issues related to the event-triggered nature of distributed control have been addressed by the hybrid system control community. Here the idea is to develop a formalism that combines the best of control theory and state machine theory [44], [45], [46]. Although few analytical results are available today, this rather intuitive and promising approach is an active area of research.

Time synchronization research for sensor networks has been intense, yielding promising results [8]. In our model, we confidently assume that sensors readings come with accurate time-stamp. Also we assume that sensors know their location in space. A localization service ensures that the nodes in a deployed network can compute their location relative to each other [9]. With these two assumptions, we use the standard control formalism with sensor networks. A choice of a model is critical to the design of controllers for such systems. In dealing with complex applications such as DPEGs, control must be exercised at several levels and a hierarchical system seems to be the natural modeling choice. A graphical representation is shown in Figure 11.

At the low level, the continuous time dynamics of the system need to be captured. Since the implemented controllers are digital, the model is discretized to yield a discrete time control system. At this level, the system is time based, in the sense that time triggers each transition. At each time step, an observation, generated from a sensor reading, needs to be

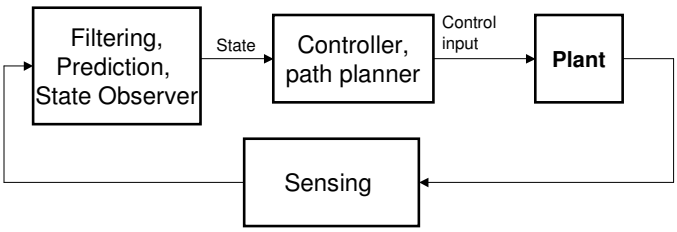


Fig. 12. Low level controller

provided to the controller, which will in turn produce an input to the dynamics of the system, via an actuator. In standard control problems, the sensors are physically attached to the plant; therefore, it is assured to receive a sensor reading at each time step. In the case of SNs, the sensing is distributed. This means that it may take some time for the observation to reach its destination since packets over the network are subject to delay and loss. Additionally, the control law needs some information about the plant to compute the next input, which will heavily rely on state estimation, prediction, and smoothing. In the absence of an observation, we will make use of the model alone to provide state estimation for control. In this way, late packets can be used to improve current estimate. Several methods can be used for estimation from Kalman to particle filtering. A graphical representation of the low level controller is shown in Figure 12.

At the higher level, the system is event based. In this domain, the control reacts to one or more events, sequences of which are called behaviors. Events are detected by the sensor network and transmitted to a discrete controller that generates the appropriate reaction. Each reaction is then transmitted to the lower level by changing the control objective to agree with the new specifications. Once again events occur in an asynchronous fashion, making formal analysis difficult. To work with such events, we implement the system using a synchronous reactive language, where behaviors can be verified and mapped to our asynchronous platform, making sure the verified properties are preserved. The problem of mapping behaviors from different domains has been tackled in several different ways. We follow the approach of Benveniste [47] by designing controllers in a synchronous fashion, verifying the behavior, and then de-synchronizing the algorithm to be implemented on the asynchronous target architecture. The advantage of this approach also includes the possibility of automatically generating embedded code directly from a high level specification language, thus enormously speeding up the development phase. A graphical representation of the design flow is shown in Figure 13.

V. Conclusions

In this paper, we presented an overview of research activities dealing with distributed control in sensor networks. We introduced sensor networks and related research issues. We then presented our hardware and software platforms while proposing an open architecture to help develop rich distributed

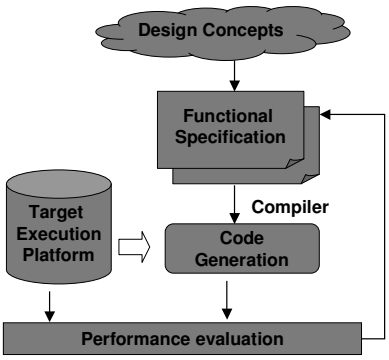


Fig. 13. A graphical representation of the proposed design methodology

applications. We presented an overview of the theoretical issues facing researchers interested in using sensor networks for distributed control applications. We identified key properties that cause classical control theory to fail. We suggested a general approach to control design using a hierarchical model composed of continuous time-triggered components at the low level and discrete event-triggered components at the high level. For the future, we will focus on implementation, verification, and testing of our methodologies in distributed control systems on our proposed DPEG testbed.

ACKNOWLEDGMENT

We would like to thank the TinyOS team for providing a great hardware and software platform. Additionally, we would like to thank the entire NEST team at Berkeley and Intel Research-Berkeley. Finally, we would like to thank the following individuals: Sarah Bergbreiter, Eric Brewer, David Culler, David Gay, Jason Hill, Barbara Hohlt, Chris Karlof, Phil Levis, Sam Madden, Kris Pister, Joe Polastre, Naveen Sastry, Robert Szewczyk, Rob von Behren, David Wagner, Matt Welsh, Kamin Whitehouse, and Alec Woo. This research is supported by DARPA under grant F33615-01-C-1895.

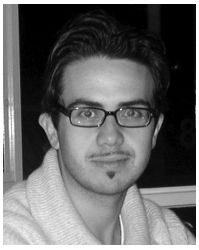
REFERENCES

[1] Crossbow home page. http://www.xbow.com/.
[2] Millennial home page. http://www.millennial.net.
[3] Sensoria home page. http://www.sensoria.com.
[4] K. S. J. Pister, J. M. Kahn, and B. E. Boser, "Smart dust: Wireless networks of millimeter-scale sensor nodes," 1999.
[5] Seismic sensor research. http://www.berkeley.edu/news/media/releases /2001/12/13_snsor.html.
[6] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," Atlanta, GA, Sept. 2002.
[7] A. N. Knaian, "A wireless sensor network for smart roadbeds and intellignet transportation systems," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.
[8] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, USA, Dec. 2002.
[9] S. Klemmer, S. Waterson, and K. Whitehouse. (2000, Dec.) Towards a location-based context-aware sensor infrastructure. http://guir.berkeley.edu/projects/location/Location.pdf.
[10] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Mobile Computing and Networking*, 1999, pp. 263–270.
[11] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad-hoc mobile wireless networks," in *IEEE Transactions on Personal Communications*, Apr. 1999.
[12] F. Zhao, J. Shin, and J. Reich, "Information-driven dynamic sensor collaboration for target tracking," *IEEE Signal Processing Magazine*, vol. 19(2), Mar. 2002.
[13] L. Guibas, "Sensing, tracking and reasoning with relations," *IEEE Signal Processing Magazine*, vol. 19(2), Mar. 2002.
[14] T. Parsons, "Pursuit-evasion in a graph," in *Theory and Application of Graphs*, Y. Alani and D. Lick, Eds. Springer-Verlag, 1976, pp. 426–441.
[15] M. Megiddo, S. Hakimi, M. Garey, S. Johnson, and C. Papadimitriou, "The complexity of searching a graph," *Journal of the ACM*, vol. 35, no. 2, pp. 18–44, 1988.
[16] R. does not help to seach a graph, "The complexity of searching a graph," *Journal of the ACM*, vol. 41, no. 2, pp. 224–245, 1993.
[17] S. LaValle, D. Lin, L. Guibas, J. Latombe, and R. Motwani, "Finding an unpredictable target in a workspace with obstacles," in *IEEE Int. Conf. Robotics and Automation*, 1997.
[18] S. LaValle and J. Hinrichsen, "Visibility-based pursuit-evasion: The case of curved environments." in *IEEE Int. Conf. Robotics and Automation*, 1999.
[19] X. Deng, T. Kameda, and C. Papadimitriou, "How to learn an unknown environment I: The rectilinear case," *Journal of the ACM*, vol. 45, no. 2, pp. 215–245, 1998.
[20] S. Thrun, W. Burgard, and D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Machine Learning and Autonomous Robots (joint issue)*, vol. 31, no. 5, pp. 1–25, 1998.
[21] J. Hespanha, H. Kim, and S. Sastry, "Multiple-agent probabilistic pursuit-evasion games," pp. 2432–2437, 1999.
[22] H. Kim, R. Vidal, D. Shim, O. Shakernia, and S. Sastry, ". a hierarchical approach to probabilistic pursuit evasion games with unmanned ground and aerial vehicles," pp. 634–639, 1901.
[23] Great duck island homepage. http://www.greatduckisland.net/.
[24] S. Bergbreiter and K. Pister. Cots bots homepage. http://www-bsac.eecs.berkeley.edu/~sbergbre/CotsBots/cotsbots.html.
[25] Nesc homepage. http://nescc.sourceforge.net/.
[26] Tinyos homepage. http://webs.cs.berkeley.edu/tos/.
[27] J. Parrish, S. Viscido, and D. Grunbaum, "Self-organized fish schools: an examination of emergent properties," *The Biological Bulletin*, vol. 202, June 2002.
[28] J. Deneubourg, A. Lioni, and C. Detrain, "Dynamics of aggregation and emergence of cooperation," *The Biological Bulletin*, vol. 202, June 2002.
[29] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *IEEE Control Systems Magazine*, vol. 22, no. 3, June 2002.
[30] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, 2nd ed. The MIT Press, 2000.
[31] M. Dias and A. Stentz, "Opportunistic optimization for market-based multirobot control," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2002.
[32] P. Martí, G. Fohler, K. Ramamritham, and J. M. Fuertes, "Jitter compensation in real-time control systems," *Real-Time Systems Symposium*, December 2001.
[33] M. Caramanis and A. Sharifnia, "Near optimal manufacturing flow control design," *The International Journal of Flexible Manufacturing Systems*, vol. 3, no. 3/4, 1991.
[34] V. V. Prabhu and N. A. Duffie, "Nonlinear dynamics in distributed arrival time control," *IEEE Transactions on Control Systems Technology*, vol. 7, no. 6, November 1999.
[35] F. M. Callier and C. A. Desoer, Eds., *Linear System Theory*, 1st ed. Springer-Verlag, 1991.
[36] C.-T. Chen, Ed., *Linear System Theory and Design*, 3rd ed. Oxford University Press, 1999.
[37] C. Tomlin, J. Lygeros, and S. Sastry, "Computing controllers for nonlinear hybrid systems," in *Hybrid Systems: Computation and Control*, ser. LNCS, F. W. Vaandrager and J. H. van Schuppen, Eds. Springer Verlag, 1999, no. 1569, pp. 238–255.
[38] E. A. Lee, "Modeling concurrent real-time processes using discrete events," *Annals of Software Engineering, Special Volume on Real-Time Software Engineering*, vol. 7, pp. 25–45, 1999.
[39] W. B. Ackerman, "Data flow languages," *Computer*, vol. 15, no. 2, 1982.

[40] E. A. Lee, "A denotational semantics for dataflow with firing," Electronics Research Laboratory, University of California, Berkeley, Tech. Rep. UCB/ERL M97/3, January 1997.
[41] A. Benveniste and P. L. Guernic, "Hybrid dynamical systems theory and the signal language," *IEEE Transactions on Automatic Control*, vol. 35, no. 2, May 1990.
[42] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice, "LUSTRE: A declarative language for programming synchronous systems," 1987.
[43] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Science of Computer Programming*, vol. 19, no. 2, 1992.
[44] R. Alur and T. Henzinger, "Modularity for timed and hybrid systems," in *CONCUR97: Concurrency Theory*, 1997.
[45] M. S. Branicky, V. S. Borker, and S. K. Mitter, "A unified framework for hybrid control: Model and optimal control theory," in *IEEE Transactions on Automatic Control*, vol. 43, January 1998.
[46] J. Lygeros, C. Tomlin, and S. Sastry, "Controllers for reachability specifications for hybrid systems," *Automatica*, vol. 35, no. 3, March 1999.
[47] A. Benveniste, "Some synchronization issues when designing embedded systems from components," *Emsoft 2001*, 2001 Oct.

Bruno Sinopoli (Student Member, IEEE) received his Laurea in electrical engineering in 1998 from the University of Padova in Italy. He is currently pursuing his Ph.D in electrical engineering from the University of California at Berkeley under the supervision of Professor Shankar Sastry. His research interests include sensor networks, design of embedded systems from components, distributed control, and hybrid systems.

Cory Sharp received his B.S degree from the University of Oklahoma in computer engineering in 1997, and his M.S. degree from the University of California at Berkeley in electrical engineering in 2000. He is currently working as a research specialist in the Intelligent Machines and Robotics Laboratory at the University of California at Berkeley. His research interests include design of distributed and networked embedded systems, composable architectures, distributed programming languages, and computer vision.



Luca Schenato (Student Member, IEEE) was born in Treviso, Italy, in 1974. He received the Dr. Eng. degree in electrical engineering from the University of Padova in 1999. Since 1999 he has been with the Department of Electrical Engineering, University of California at Berkeley, where he is currently pursuing a doctoral degree. His interests include modeling of biological networks, insect locomotion, millirobotics and avionics.

Shawn Schaffert (Student Member, IEEE) received the B.S. degree in electrical engineering from the University of Nebraska, Lincoln, in 1998, and the M.S. degree in electrical engineering from the University of California, Berkeley, in 2001. In 2001, he worked as an intern at Xerox PARC, Palo Alto, California, investigating the complexity of constrained optimization problems. Currently, he is working towards the Ph.D. degree at the University of California, Berkeley, under the supervision of his research advisor, Professor Shankar Sastry. His research interests include embedded systems, distributed control, hybrid systems, and robust control in sensor networks. Mr. Schaffert is a member of Tau Beta Pi and Eta Kappa Nu.

S. Shankar Sastry (Fellow, IEEE) received the M.S. degree (honoris causa) from Harvard University, Cambridge, MA, in 1994, and the Ph.D. degree from the University of California, Berkeley, in 1981. From 1980 to 1982, he was an Assistant Professor at Massachusetts Institute of Technology, Cambridge. In 2000, he was Director of the Information Technology Office at the Defense Advanced Research Projects Agency, Arlington, VA. He is currently the NEC Distinguished Professor of Electrical Engineering and Computer Sciences and Bioengineering and the Chairman of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His research interests are embedded and autonomous software, computer vision, computation in novel substrates such as DNA, nonlinear and adaptive control, robotic telesurgery, control of hybrid systems, embedded systems, sensor networks, and biological motor control. Dr. Sastry was elected into the National Academy of Engineering in 200 "for pioneering contributions to the design of hybrid and embedded systems." He has served as Associate Editor for IEEE TRANSACTIONS ON AUTOMATIC CONTROL, IEEE CONTROL SYSTEMS MAGAZINE, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS.