April 2004

Distributed Sensor Databases for Multi-Robot Teams

Anthony Cowley
University of Pennsylvania

Hwa-Chow Hsu
University of Pennsylvania

Camillo J. Taylor
University of Pennsylvania, cjtaylor@cis.upenn.edu

Distributed Sensor Databases for Multi-Robot Teams

Abstract
In this paper we describe our implementation of a distributed sensor database that was designed to support the activities of teams of mobile robots as they explore an environment. Importantly, this approach effectively separates the process of acquiring sensor data from that of exploiting it. This allows us to develop applications where robots and human users can automatically discover and utilize sensor measurements acquired by other robots in the team. We also explain our approach to implementing distributed queries, an important capability that allows us to perform queries in a way that makes best use of the limited available communication bandwidth. Finally, we briefly describe how we have used this system to support situational awareness tasks.

Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

Distributed Sensor Databases for Multi-Robot Teams

Anthony Cowley, Hwa-Chow Hsu, Camillo J. Taylor

GRASP Laboratory – University of Pennsylvania, Philadelphia, PA, USA, 19104
{acowley, hwahsu, cjtaylor}@grasp.cis.upenn.edu

Abstract— **In this paper we describe our implementation of a distributed sensor database that was designed to support the activities of teams of mobile robots as they explore an environment. Importantly, this approach effectively separates the process of acquiring sensor data from that of exploiting it. This allows us to develop applications where robots and human users can automatically discover and utilize sensor measurements acquired by other robots in the team. We also explain our approach to implementing distributed queries, an important capability that allows us to perform queries in a way that makes best use of the limited available communication bandwidth. Finally, we briefly describe how we have used this system to support situational awareness tasks.**

I. Introduction

A number of current research efforts, including our own, aim to develop technologies that would facilitate the deployment of teams of mobile robots into an environment to support a range of environmental monitoring and surveillance tasks. [4], [3], [2], [11], [1], [6] Figure 1 shows some of the members of our team of unmanned ground and air vehicles which have been developed to tackle these kinds of distributed data gathering and analysis tasks. When these vehicles are deployed in an environment they form a network of processors, sensors and actuators that can collectively gather enormous volumes of information about the environment which can be visualised and analyzed by human operators or other automated systems.

A technical challenge that one faces in effectively deploying such systems is the problem of collating and exploiting the volumes of data that the robots acquire and generate as they explore the environment. One approach to this problem which offers some promise is to adopt some of the techniques and ideas developed in the context of distributed databases to arbitrate access to sensor data. Madden et al [12], [10] demonstrated how database techniques could be used to effectively and efficiently abstract the flow of information from an array of sensory motes in the environment. The abstractions provided by the database approach allow us to reformulate the problem of extracting data from the sensor array in terms of database queries which are handled in a distributed manner.

In this paper we consider the problem of extending distributed database ideas to handle the robotic nodes of the type shown in Figure 1 which have substantially greater capabilities than the motes considered in previous work [10].

Fig. 1. The sensor database implementation described in this paper was designed to support the activities of distributed teams of mobile robots. This figure shows some of the UAVs and UGVs in our team.

In this framework, each robot is equipped with gigabytes of storage, a powerful microprocessor and a motion control system. Typical sensor suites on a node may include GPS units, Inertial Measurement Systems, temperature sensors, altimeters, and imaging systems such as cameras, range finders and radars. It is important to note that the sensor arrays on the nodes provide volumes of data at a bandwidth that dwarfs the communication capabilities of the network. (Consider for example 10 robots each collecting imagery at a rate of 15 MegaBytes/sec while communicating over a shared 10MegaBit/sec 802.11b network.) This fact prompts us to consider approaches where each of the robotic nodes stores the results of various sensor processing operations in a local database and sensor queries are distributed over the network and executed on the nodes that host the data.

The distributed sensor database approach described here offers a number of advantages: firstly this approach effectively separates the process of acquiring sensor information from the process of analyzing and exploiting the data. This provides a useful degree of flexibility since it means that data acquired by one robot can be used for a number of different purposes which could not be adequately predicted at design time. For example, a UGV operating in the environment can query other UAVs in the theater to find aerial imagery taken within its vicinity so that it can localize itself or identify relevant targets in the scene. Since our implementation supports dynamic discovery of data sources, this query could be answered by any UAV with the requisite data without either of the robots having to know about the other's existence prior to the initiation of their transaction.

0-7803-8232-3/04/$17.00 ©2004 IEEE

This approach also allows us to provide users with a flexible system for analyzing data collected by the sensor array. Here the user can answer questions about the scene by formulating queries which are processed in a distributed manner on the robot network.

Another important aspect is that it allows us to make effective use of the limited communication bandwidth by distributing query processing and data analysis over the network. Our implementation allows us to move sensor analysis computations over the network to the data instead of shipping reams of data over the network to a central processing station. If for example the user were interested in detecting changes in a particular area of the environment by analyzing image data taken over a period of a few hours he could formulate a query which would encapsulate all the relevant image processing applications. This query would then be relayed to the nodes that had the required image data and the data intensive processing would be carried out on those processors with the final results being shipped back to the user.

In the sequel we describe the implementation of our distributed sensor database system and explain some applications that we have implemented on top of this framework.

II. Implementation

Our current work on distributed sensor databases builds on ROCI (Remote Objects Control Interface), our software framework for distributed embedded software applications [5]. The building blocks of ROCI applications are self-contained, reusable modules. Each module encapsulates a process which acts on data available on its inputs and presents its results on well defined outputs. Thus, complex tasks can be built by connecting inputs and outputs of specific modules. These connections are made through a pin architecture that provides a strongly typed, network transparent communication framework. A good analogy is to view each of these modules as an integrated circuit (IC) that has inputs and outputs and does some processing. Complex circuits can be built by wiring several ICs together, and individual ICs can be reused in different circuits. ROCI modules have been developed for a wide range of tasks such as: interfacing to low level devices like GPS units and cameras, computing position estimates based on GPS, IMU and odometry data, acquiring stereo panoramas, platform motion control, online map building and GPS waypoint navigation.

ROCI modules are further organized into tasks. A ROCI task is a way of describing an instance of a collection of ROCI modules to be run on a single node and how they interact at runtime. Tasks represent a family of modules that work together to accomplish some end goal – a chain of building blocks that transforms input data through intermediate forms and into a useful output. A task can be defined in an XML file which specifies the modules that are needed to achieve the goal, and the connectivity between these modules. Tasks can also be defined and changed dynamically, by starting new modules and connecting them with the inputs and outputs of other modules.

The wiring that connects ROCI modules is the pin communication architecture. Pin communications in ROCI are designed to be network transparent yet high performance. Basically, a pin provides the developer with an abstract communications endpoint. These endpoints can either represent a data producer or a data consumer. Pins in the system are nothing more than strongly typed fields of the module class, and thus connecting a producer pin to a consumer pin is as simple as adding a reference to the producer to the consumer's input collection. Pin communication allows modules to communicate with each other within a task, within a node or over a network seamlessly. The type system enforces pin compatibility at run time which makes it impossible to connect inputs and outputs of differing types.

The core control element in the ROCI architecture is the ROCI kernel. A copy of the kernel runs on every entity that is part of the ROCI network (robots, remote sensors, etc.). The kernel is responsible for handling module and task allocation and injection. It allows applications to be specified and executed dynamically, by connecting available pins and transferring code libraries to the nodes as needed.

The kernel is also responsible for managing the network and maintaining an updated database of other nodes and modules in the ROCI network. In this way, ROCI acts as a distributed peer to peer system, much like Napster. Nodes can be dynamically added and removed from the network and information about these nodes and the modules running on them is automatically propagated throughout the system without the need for a central repository.

Importantly, the modules in the system are self describing so that the kernel can automatically discover their input and output pins along with any user settable parameters. These features of the ROCI architecture facilitate automatic service discovery since a module running on one ROCI node can query the kernel database to find out about services offered by modules on other nodes and can connect to these services dynamically.

A simple example of a ROCI application is shown in Figure 2. Here a collection of ROCI modules has been connected together to implement a GPS waypoint control system. The outputs of a GPS module and a stereo camera system are provided as inputs to a waypoint control system which in turn outputs velocity commands to a PID loop which interfaces to the low level servo control system of

the robot. The waypoint control system is also connected to a browser module running on another computer in the network which is being used to monitor the status of the controller and provide a manual override capability.

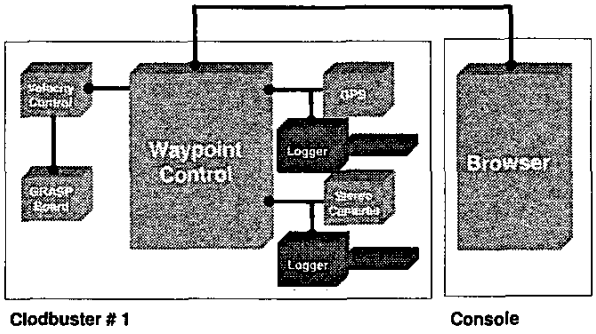

Fig. 2. ROCI Architecture: tasks are composed of modules and run inside nodes. The pin communication system allows seamless communication between modules within the same task, modules in different tasks, or modules on different nodes. Logger modules can be attached to the outputs of any module to create a timestamped record of that modules output.

A. Logger Modules

Our sensor database system is implemented on top of ROCI through the addition of logger modules. These logger modules can be attached to any output pin and record the outputs of that module in a timestamped log which can be accessed by external processes. These logger modules appear to the system as regular ROCI modules which means that they can be started and stopped dynamically and can be discovered by other ROCI nodes on the system. This last point is particularly salient since it means that robots can learn about the records available in other parts of the network at run time as those resources become available.

Since logger modules can be attached to any output pin, there is no meaningful distinction between "low level" sensor data such as images returned by a camera module and "high level" information such as the output of a position estimation module. Any data that is relevant to a task can easily be logged through the addition of a logger module.

The generic logger module logs all incoming data based on time, an index relevant and meaningful regardless of the data type. Additional indexing methods that are specific to a particular data type are easily implemented by creating a new type of logger module that inherits from the general logger and is explicitly usable only with the expected data type. For example, a logger module that records the output of a GPS unit may also support efficient indexing based on position.

Figure 3 shows a schematic view of the information available from logger modules running on a single ROCI node. Using time as a common key provides a simple mechanism for correlating information from different channels. Consider, for example, the problem of obtaining all of the images that a robot acquired from a particular position. This can be implemented efficiently by first indexing into the GPS log to find the times at which the robot was at that location and then using those times to index the image log to pull out the images taken from that vantage point.

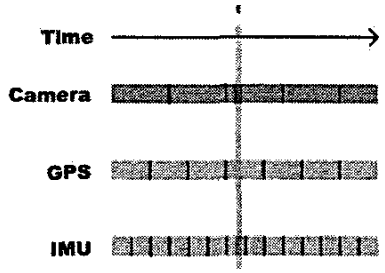

Fig. 3. The logger modules can be used to store the output of any module in a timestamped archive. Since time is a common index it can be used to correlate information derived from various channels.

Having time as a common index also obviates the need for a fixed database schema on the robots. Different logger modules can be added or removed from a node as needed without having to perform complex surgery on a global table of sensor readings. Since the logger processes do not interact directly, they can be started and stopped, added and removed independently of each other.

B. Query Processing

Once a relevant data log has been found on the network, one must then face the problem of executing a query to extract information from that archive. It is often the case that the volume of data stored in a log makes it unattractive to transfer the data over the network for processing. In these situations we can take advantage of the fact that the facilities provided by ROCI can be used to support distributed query processing. Consider the example of a UAV that stores a log of images acquired as it flies over a site. If a process on a UGV wanted to access this data to search for particular targets in the scene it would be impractical to transfer every image frame to the ground unit for processing. Here it makes sense to consider sending an active query to the UAV requesting it to process the images and send the target locations back to the UGV.[1] This can be accomplished by developing a ROCI module that extracts the targets of interest from UAV imagery and then sending this module to the UAV as part of a query. The ROCI kernel on the UAV would

[1]Mohammed (the query) goes to the mountain (the data) instead of vice versa.

then instantiate a task and use this module to process the data in the image log returning the results to the UGV.

Sophisticated queries that involve chaining together the results of many processing operations or combining information from several logs can be handled through precisely the same mechanism. The query takes the form of a network of ROCI modules that carry out various phases of the query. The modules in this task are distributed to appropriate nodes on the network and the final output is returned to the node that initiated the request. This approach allows us to dynamically distribute the computation through the network in order to make more efficient use of the limited communication bandwidth.

Another feature of this approach is that it promotes code re-use since the modules that are developed for carrying out various data processing and analysis operations online can also be used to implement queries on stored data logs. This is important not just by virtue of facilitating rapid development, but also by the robustness and familiarity users have with the component modules used in all aspects of a ROCI deployment. By making the same framework pervasive throughout the development pipeline, users are able to concentrate their efforts on improving core techniques because the code only needs to be written once. Once the code has been written, users setting up robot behaviors work from the same toolbox as those formulating queries at run time and throughout post-processing.

The notion of query stages combined with the strong type system underlying ROCI module inputs and outputs immediately opens the door for a multitude of queries that make use of functionality already used by robot behaviors. For example, a robust localization routine may be run on all robots as they move around the environment. This routine must update relatively quickly to allow the robot to navigate in real-time, thus necessitating that it only consider readily available data. However, a user or autonomous agent may require an alternate estimation of a robot's location at a particular time in the past, perhaps utilizing newly acquired data. This can be achieved by designing a query wherein a localization routine, possibly another instance of the original routine, is connected to not only locally collected data, but also to any number of data processing routines, also specified by the query body, running on any number of other nodes. This localization may take a relatively long time to execute, and may not be suitable for real time control, but it is available to any programmed behavior or human operator that requests it. This query, while complex, automatically benefits from the shared toolbox provided by the consistent design framework. Processing modules that already exist on data hosts need not be transmitted, while others are downloaded from peers on an as-needed basis. The query itself is analogous to a behavior task: it specifies processing modules and how they connect. The ROCI kernel handles the work of ensuring that modules exist on the nodes that need them, and that those modules are properly connected.

III. Applications

In the GRASP Laboratory we have, to date, experimented with a database generated by a heterogeneous team of ground and air robots equipped with cameras, GPS receivers, IMU readers, altimeters and other sensors. Using a visualization module, called the ROCI Browser, the collected sensor data can be fused and displayed to the user in an intuitive way. A typical view provided by the browser is shown in Figure 4. Upon start up, the Browser queries the kernel to discover the data logs provided by all of the nodes in the network.

The Browser has a set of visualization plug-ins, each designed to perform a specific visualization task. In this example, two of these plug-ins are used, a Map plug-in, which reads an image file from disk along with corresponding geo-registration data, and an image position plug-in, which is used to present the collected data.

The connections between the database modules and the visualization plug-ins can be established manually by the user or through an XML script. Once the connections are established, a screenshot as shown in Figure 4 can be generated in the following manner: First, a query for time stamps only is made to the camera loggers, which maintain logs of all images taken by the robots. Next, using the time stamps retrieved by this query, the locations corresponding to each image are requested from the position loggers. These locations are then plotted in the browser on top of a geo-registered overhead image of the target area, which allows for quick and intuitive filtering of the images by the user. Once a location of interest is determined, a click on the image icon results in the retrieval and display of the corresponding image (Figure 5). The interface can be used to recover aerial views recovered by the UAVs and terrestrial stereo panoramas acquired by the UGVs.

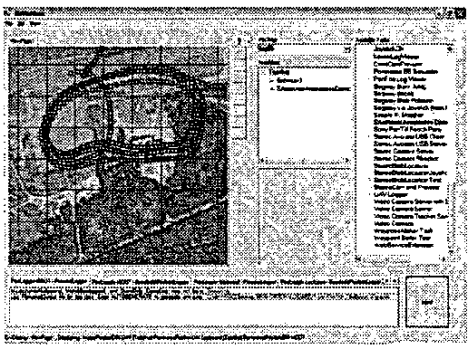

Fig. 4. This figure shows the main browser interface. Locations for which images are available are plotted on top of a satellite image of the area.

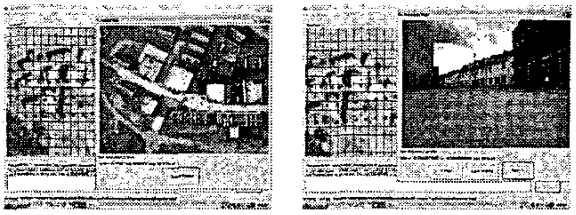

Fig. 5. Individual views of the scene captured by the UAVs and terrestrial views collected by the UGVs can be called up from the database by clicking on the plotted locations in the browser

Furthermore, by using IMU data retrieved from the IMU logger (using the time stamp from the image), locations in the captured image can be translated to real world coordinates. Thus, the captured images can be geo-registered and used for robot navigation as well.

It is easy to imagine other uses of these databases. Since the GPS logger modules support efficient indexing on position, we can use this capability to request all images of a certain target taken from a particular orientation and standoff distance over a certain period of time. Such a query would combine information from the GPS, IMU and image logs on multiple robots. This type of query can be seen as a way for an agent to sift all the data on the network down to a manageable set that is relevant to the agent. This particular query is realized by first requesting time indices from every GPS log on the network corresponding to positions within some threshold distance of the agent's location of interest. These indices can be used to index into IMU logs on the responding nodes to obtain orientation information which allows the query logic to filter out times when the sensor was oriented away from the target location. The remaining set of time indices can then be used as the body of a query to other sensor logs to isolate the actual sensor readings (e.g. images, range maps, etc.) when the target location was in view. This sensor data can be returned to the agent that originated the query, or the selected sensor data could instead be used as the input to a processing module, or chain of modules, which, like the query itself, executes on the data hosts themselves. The outputs of those modules could then be shipped to wherever they are needed by attaching the appropriate pins.

IV. Conclusions

In this paper we have described an approach to implementing distributed database capabilities on a network of mobile robots. Importantly, this approach allows us to separate the process of acquiring data from the process of exploiting it. This enables us to develop applications where the robot team members and the human users automatically discover and exploit measurements acquired by other robots in the team.

The ROCI framework also provides a convenient substrate for supporting distributed queries. Complex queries can be supported by injecting a collection of modules into the network to carry out the required processing. This feature allows us to distribute query processing throughout the network and to make more efficient use of the limited communication bandwidth.

By applying distributed database methods and techniques, the architecture presented here frees designers from having to create a static, all-encompassing communications scheme capable of satisfying a set of prespecified query types. Instead, individual developers are able to utilize all sensor network resources in a modular, dynamic fashion through the use of modularly constructed distributed database queries.

The approach has been implemented on our mobile robot team and been used to successfully support situational awareness tasks.

V. References

[1] R. Alur, A. Das, J. Esposito, R. Fierro, G. Grudic, Y. Hur, V. Kumar, I. Lee, J. Ostrowski, G. Pappas, B. Southall, J. Spletzer, and C. Taylor. A framework and architecture for multirobot coordination. In D. Rus and S. Singh, editors, *Experimental Robotics VII, LNCIS 271*. Springer Verlag, 2001.

[2] T. Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, College of Computing - Georgia Institute of Technology, 1998.

[3] Maxim A. Batalin and Gaurav S. Sukhatme. Sensor network-based multi-robot task allocation. In *IROS*, page 1939, October 2003.

[4] Sarah Bergbreiter and K.S.J. Pister. Cotsbots: An off-the-shelf platform for distributed robotics. In *IROS*, page 1632, October 2003.

[5] Luiz Chaimowicz, Anthony Cowley, Vito Sabella, and Camillo J. Taylor. Roci: A distributed framework for multi-robot perception and control. In *IROS*, page 266, 2003.

[6] A. Das, R. Fierro, V. Kumar, J. Ostrowski, J. Spletzer, and C. J. Taylor. Vision based formation control of multiple robots. *IEEE Transactions on Robotics and Automation*, 18(5):813–825, October 2002.

[7] A. Das, J. Spletzer, V. Kumar, and C. J. Taylor. Ad hoc networks for localization and control. In *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[9] B. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric. Most valuable player: A robot device server for distributed control. In *Proceedings of the IEEE/RJS International Conference on Intelligent Robots and Systems*, pages 1226–1231, 2001.

[10] Joseph M. Hellerstein, Wei Hong, Samuel Madden, and Kyle Stanek. Beyond average: Towards sophisticated sensing with queries. In *Information Processing in Sensor Networks*, March 2003.

[11] D. MacKenzie, R. Arkin, and J. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52, 1997.

[12] Samuel R. Madden, Michael J. Franklin, Joseph Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, June 2003.

[13] D. Martin, A. Cheyer, and D. Moran. The open agent architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.