FAULT TOLERANT AUTONOMOUS MOBILE ROBOTIC SYSTEMS

Dale Lord (Student) and Kurt Kosbar (Advisor)
Telemetry Learning Center
Department of Electrical and Computer Engineering
University of Missouri – Rolla
Rolla, MO 65409-0040

ABSTRACT

Recent emphasis has been placed on mobile robotics performing in unstructured environments. This realm of operations requires many different algorithms to interpret the various situations. This not only requires a system that is able to support, and facilitate, the fusion of the results, but it also needs to be tolerant of system errors. In modern operating systems, separate processes are able to fail without affecting other processes. Using this ability, along with fault tolerant inter-process communications, and supervisory process managers, allows the total system to continue to operate under adverse conditions. While this paper focuses primarily on the challenges faced by mobile robotics, the approach can be extended to a wide range of systems which must autonomously identify and adapt to failures/situations.

KEY WORDS

Mobile robots, Operating systems, Autonomous Systems, Fault-Tolerant Systems

INTRODUCTION

Robotics, by its very definition, needs to be able adapt to various situations. Robotic arms in industrial environments are just reprogrammed to a different sequence of operations to handle new situations since the environment around the robot is controlled. For mobile robotics, making the operational environment easy for the robot to navigate is not practical.

In mobile robotics, even the first step of identifying what is the ground and how far the ground extends is difficult. This determination of traversable floor can be done with many different

sensors. Air based sonar, for example, sends out a tone above human hearing and listens for the echo from nearby objects. The time it takes to receive the echo is directly proportional to the distance to the object reflecting the sound. While this works for most situations the angular uncertainty and multi bounce reflections make sonar not a complete solution for mobile robot navigation. While laser range finders, and other systems, may provide superior results, there are always situations where a sensor can be deceived. Often the various sensor systems have different enough errors, that combining the results generally is not much better than using the best of the sensors. Using a simple weighted average based on the general error also does not help with situations where a sensor returns erroneous values.

A better solution is to identify situations where sensors have problems, and use another sensor that is not deceived in this situation. For example, air based sonar sent into a corner will often reflect off of one wall onto the other wall and then travel back to the sensor. This extra travel deceives the sensor into thinking that the corner is further away. It would be better to identify corners, and switch to another sensor in that situation.

This type of operation, identifying a situation and implementing a customized solution, is useful throughout unstructured robotics. There may be situations where a door blocks the path of the robot, but the robot may have the ability to open the door. The system must first identify the situation, then execute of a sequence of operations to open the door. The operating system must facilitate these types of operations in an efficient manner. This general approach can be applied to areas outside of robotics, where systems need to autonomously identify, and adapt, to a variety of errors or situations.

The architecture described in this document, uses the Linux computer operating system to manage the different processes, while utilizing shared memory, signals, and message queue libraries, to provide inter process communications.

There are many other robotic operating systems [1-4]. The most popular being the Player / Stage system. The Player / Stage [1] system operates over TCP/IP sockets. While this allows processes to be distributed over a network the overhead of sending sensor information over a network to multiple client processes can be a problem. Image data from a vision system can consume a processor just copying the data from one location to another let alone sending it across TCP/IP.

Carmen [2-4] uses an inter process communication system called IPC which also uses TCP/IP for its underlying communications. IPC provides subscription and command based interfaces and manages the underlying inter process communications. While IPC makes robotic type communications easy, it still is not practical for large image data.

The Saphira [5] system was developed by SRI International. This system was purchased by ActivMedia, which is now MobileRobots, who renamed the system, ARCS inside. This commercially available system is closed to outside development and appears to operate as a few large processes.

While robotic operating systems can be written as single process the unstructured nature of mobile robotics requires a system that is able to encompass a variety of situations. A robotic operating system must be able to be extended to deal with more situations and not reduce the reliability of the original system.

SYSTEM ARCHITECTURE

Many environments require widely different methods, and multiple sensors, to be pre-processed to obtain the information required for robotic navigation and operation. Even when the domain of the navigation is controlled, it still becomes apparent that a dynamic multi-tasking system is needed. This paper presents a robotic operating system architecture for mobile robotics. The specific modules of robot navigation and obstacle avoidance are subsequently discussed within this framework.

SYSTEM FLOW

All the modules in the robot system are triggered by an external input, or another module. This type of external event input could be a user requesting action; a reaction to visual sensor stimuli, or any other changes in the robot's status. Server processes are used to collect various input data, and trigger sensor processes to transform and analyze the data. It is important to note that these sensor transform processes are kept separate from the server process, and task processes, for a reason. Often the robot may have several tasks to complete, and multiple goals to achieve. These tasks and goals may need the same transform, on the sensor data. By keeping the transform process separate from the tasks, the transform is only performed once. If it is not being used, the process is stopped and unloaded, saving resources. This type of data flow and processing is the beginning of a web of processes, and tasks that collect refine and analyze data for mobile robotics.

If the environment where the robotic task is to be done is controlled (such as in the case of some industrial robots) it may be possible to write a single process, which accesses all of the available sensor data, and controls all the actions of the robot. However the environment of an autonomous mobile robot is not controlled. For the robot to accomplish its objectives, it should be adaptable to this unstructured environment. A sample schematic of the proposed robotic system architecture is shown in Figure 1. Each of the sensors (such as video, sonar, user input) is controlled by sensor server, and provides raw data which is analyzed by sensor processes to extract useful sensory information. Note that some raw data from a given sensor may be

processed by multiple sensor processes, to extract different information. For example, video data can be processed by a motion flow process to detect moving objects, while the same raw data is being processed by texture segmentation process to detect the extent of the floor. The useful information from sensor processes are fed to task processes, which has the responsibility for a specific task such as obstacle avoidance. Note that the same sensory information can be used by multiple tasks, and each task may also act using information from multiple sensors. The behavior coordinator has the responsibility to register different behaviors in the system, as well as schedule different outputs based upon the response of the behavior processes. The knowledge database contains this registry of behaviors, and control flow for the robotic system behavior coordinator. Any active behavior of the robot is registered with the task coordinator, which in turn adds the triggers to the existing processes to accomplish the behavior.

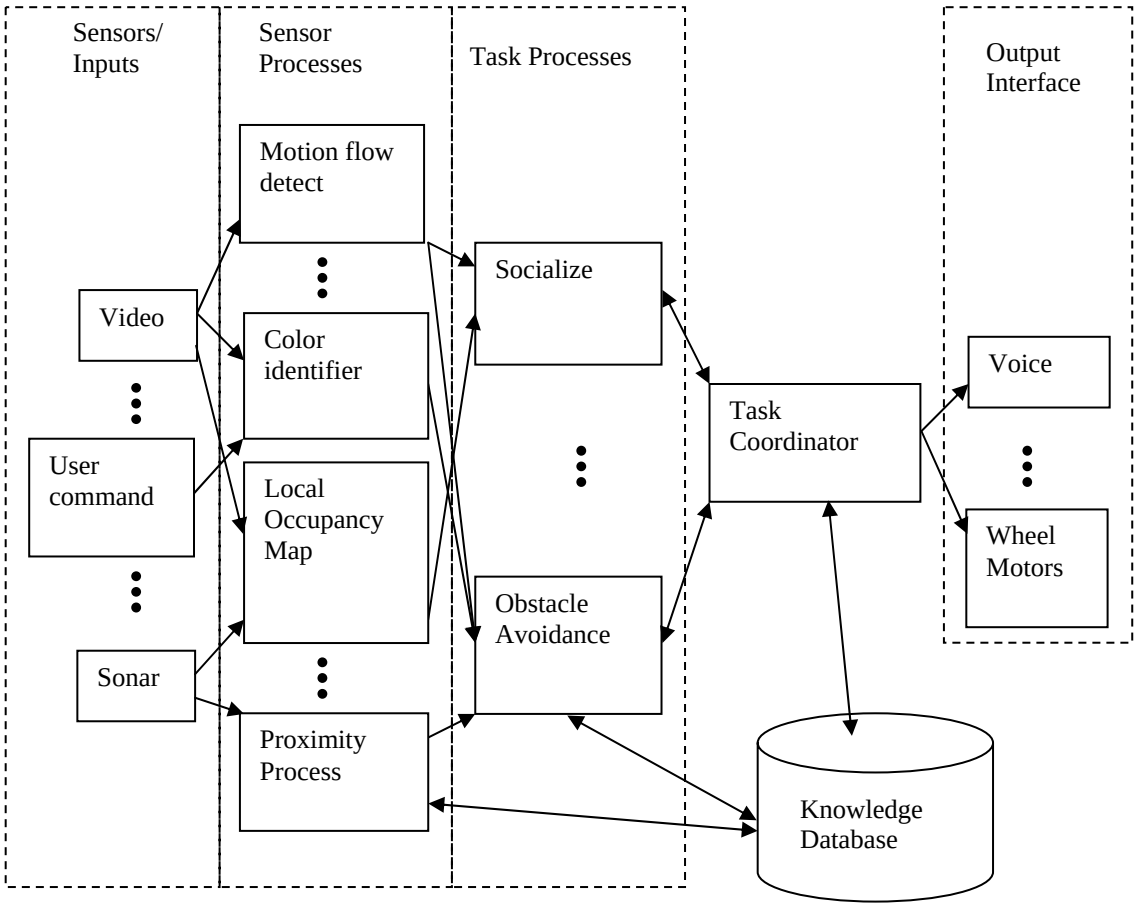

Figure 1. System architecture for robotic interactions

SENSOR SERVERS

Often there are multiple tasks that could require sensor information. For example, the robot may be using the vision information to estimate his position, while at the same time it is required to visually track an object. Without a vision server, allowing for concurrent connections to the vision data, these two tasks would have to be integrated into a single process, or take turns using the video resource. This may not be difficult for two tasks, but in general there could be dozens of visual tasks that occur at the same time, and each task would require different amounts of processing time with different data. This multiple access problem is solved by the use of sensor information servers.

Vision Server

One problem with allowing multiple processes to access the same image information is some processes might take longer than a single frame time period to work with the image. If the image capturing were held until all processes were done with the image data, then the capture rate would not be fast enough for some high speed sensor processes, like motion detection. While making a copy of the image is possible, copying takes a significant amount of processor time, and would be required for each process that needed the image. The best solution is to implement a ring buffer. The video capture is done independent of the image data usage so the frame rate is always at thirty frames per second.

This is done by filling the ring buffer with frames using the capture card's Direct Memory Access controller. Once the buffer is full, normal operation commences. The server blocks (waits) until the next to the last frame is done transferring from the capture card. The capture card continues with the last frame, however before it is done, the next frame for it to transfer is queued to replace the next oldest frame in the ring buffer. In this method, the frames are available for the maximum amount of time, and the DMA controller always has the next frame to transfer before it is done with its current frame (Figure 2).

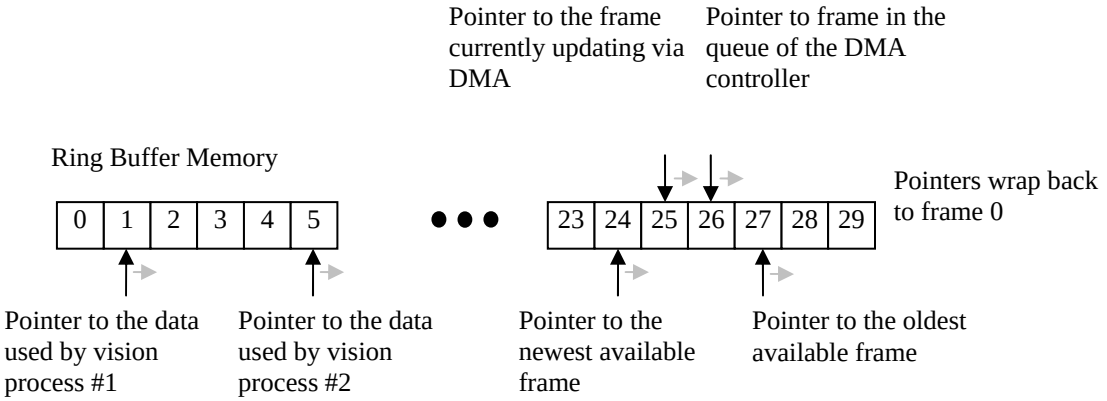

Figure 2. Video ring buffer

The size of the ring buffer, multiplied by the time per frame, gives the new maximum time for processing an image. Typically thirty frames are held so that there is a complete second to process the image data.

SENSOR PROCESSES

Some sensors give useful information directly from their outputs. For example, sonar uses the time the acoustic signal requires to travel to a target, and bounce back to the transducer. This time is proportional to the distance to the target. Cameras, on the other hand, are primarily used to reproduce a scene on a screen, and there is little knowledge gained from the raw data of an image. Processing is required to gather edges, segment areas and recognize features. This process of going from raw data to usable knowledge is highly dependent on the sensor and the knowledge that is desired. The set of algorithms discussed here, are processes associated with image/video sensors, since there is an enormous amount of information contained in an image. It is important to note that the image sensor processing done in these algorithms is not typically performed on the entire image of each frame. Most often there is a priori knowledge of what is expected in a view, for example earlier processing or sonar data. It is much more efficient to just verify these expectations, or search in a small area about the expectation location. Even in situations where nothing is known about the scene in advance, it is better to process the center or the bottom of the image first, since often this is the most important area of the view. An even better approach would be to concentrate on the area in the image where motion is detected. However this is especially difficult when the robot is in motion. The motion of the robot needs to be compensated for in the view, in order to maintain areas that have already been processed and to be able to detect motion in the view.

INTER PROCESS COMMUNICATIONS

With the modules of the robotic system being separate, the system gains stability - unfortunately the communication between processes becomes more difficult. Depending on the data and synchronous needs two types of communication are used.

Message Queues

The first common communication between processes, is where one needs to request data, or command action, by another process. This type of communication also may, or may not, need to be synchronous. The robotic system implements this communication using message queues. A general purpose queue is set up for inter process communications. Server processes generally listen for a targeted message to modify, or perform an action. The message queue library allows for messages to be checked without blocking the process flow when a message is

not available. This allows servers to continue to receive the external sensor information.

Client processes can send messages to this queue in two different modes. The client could send a message and not worry about when the server receives the message, or the client can send a message and wait until the message is received.

Shared Memory

With sensor data not being processed by the sensor servers, there may be a large amount of data that needs to be accessible by client processes. Generally, this would be too much to pass through message queues, and would lead too much time building and sending messages. The fastest one way communications possible on a single machine is shared memory. Data such as video frames which contain nearly one megabyte require this speed of communication for this distributed architecture.

Generally client processes register a dependency with the server process, and send a message for the type of data that is required. The server then continues to post the requested information to

DATABASE KNOWLEDGE

For the robot to react autonomously, it needs to have a preconceived, or previously acquired, knowledge of its environment. This knowledge consists of two aspects. First, the robot must know about behaviors and tasks and when to trigger them, and second, some tasks often need a database of prior knowledge such as navigation maps, occupancy maps, shape descriptions, etc.

Many tasks need prior knowledge of sensor data. For example, to navigate intelligently in an environment, a map of the area is needed, along with identifiers to localize the robot. This data can be rather large, so the relational database MySQL is used to store the information. By using a relational database, information is automatically indexed, sorted and searchable using the well defined “Structured Query Language”. This database has proven to be successful for searching Terabytes of information quickly, so it is well suited for storing the robot’s knowledge. The knowledge that is contained in the database, is information such as paths and nodes of navigation; information about different markers and objects in the environment are also stored in this database.

TASK COORDINATION

With the robot having various tasks and behaviors occurring simultaneously, the robot must manage its outputs. One example is when the robot is required to go to a specific location. While the robot is executing this command there may be another request for the robot to be present at another location. In this situation the location requests should be combined into a single navigation path for the robot, however in the current implementation there are locks that prevents a new task from taking control of outputs while they are in use. This locking is done using mutual exclusion locks for multi-threaded protection. Each of the robot's different outputs can receive multiple requests and most of them can benefit from future work to combine requests.

When a new task is added to the system, there are server triggers that need to be notified. For example, if a behavior is added to take a picture every time the robot is in the computer room, a trigger would be added to the navigation task to execute taking a picture when the robot enters the computer room. Each task and behavior implements an interrupt service routine and registers itself with the triggering processes. The triggering processes send a user interrupt to the client task to notify it of the pending data or robot state.

SYSTEM SERVICES

With the robotic system startup, two processes that manage the system are started. The first process is a system scheduler. Any task that needs to be performed on a regular basis can be registered in a database. The scheduler process reads this database and triggers the appropriate tasks.

Along with scheduled services the system starts servers for client processes requesting data or triggers. This not only allows for server processes to have the correct permissions to access robotic hardware it also allows for clients to not have to worry about the status of the servers when starting. If the server is not currently running, the system starts the server and registers the client as a dependant. If another client is already using a server, the new client is registered as a dependent, and the server shares the data and triggers events with both clients.

CONCLUSION

While many of the current robotic operating systems have attributes in common, it is important to note that reliability and performance are the key attributes. Reliability of a system is improved by separating different aspects of the algorithms into independent processes. This, along with robust communications, prevents a single failure from disabling the entire system.

Sensor systems, such as vision, produce large amounts of data, and can provide a large variety of information about the surrounding environment. Using a distributed process architecture works well with vision data, however the inter process communications needs to be particularly efficient. Using shared memory allows for near zero communications cost for processes on the same computer.

REFERENCES

[1] Brian Gerkey, Richard T. Vaughan and Andrew Howard. "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems" Proceedings of the 11th International Conference on Advanced Robotics, pages 317-323, Coimbra, Portugal, June 2003

[1] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. Artificial Intelligence, 101:99–141, 2000.

[2] S. Thrun. Robotic mapping: A survey. In Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann, 2002.

[3] Montemerlo, M. and Roy, N. and Thrun, S. Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit. Proceedings of the Conference on Intelligent Robots and Systems (IROS) 2003

[4] K. Konolige and K. L. Myers, "The saphira architecture for autonomous mobile robots," in AI-based Mobile Robots: Case studies of successful robot systems, D. Kortenkamp, R. P. Bonasso, and R. Murphy, Eds. MIT Press, 1996

[5] Guilherme N. DeSouza and Avinash C. Kak, “Vision for Mobile Robot Navigation: A Survey,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no. 2, pp. 237-267 Feb 2002

[6] David G. Lowe. "Object Recognition from Local Scale-Invariant Features," *iccv*, p. 1150, Seventh International Conference on Computer Vision (ICCV'99) - Volume 2, 1999

[7] Manian, V., Vasquez, R., Katiyar, P., Texture Classification Using Logical Operators, IP(9), No. 10, October 2000, pp. 1693-1703

[8] M. Mirmehdi and M. Petrou, "Segmentation of color textures," IEEE Trans. Pattern Anal. Machine Intell., vol. 22, pp. 142--159, Feb. 2000.

[9] Toshio Uchiyama, Naoki Mukawa, Hiroshi Kaneko. "Estimation of Homogeneous Regions for Segmentation of Textured Images," *icpr*, p. 7084, 15th International Conference on Pattern Recognition (ICPR'00) - Volume 3, 2000.

[10] J. Canny, "A computational approach to edge detection, " IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, pp. 679--698, 1986.

[11] Smith, S.M. and Brady, J.M. "SUSAN - A New Approach to Low Level Image Processing" Technical Report TR95SMS1c, Oxford University, 1995.

[12] K. Jack, Video Demystified: A Handbook for the Digital Engineer. HighText Publications Inc., 1993.

[13] William K. Pratt, Digital Image Processing: PIKS Inside, Third Edition, 2001, John Wiley & Sons, Inc.

[14] Rafael C Gonzales, Richard E. Woods, Digital Image Processing, 1993, Addisio-Wesley Publishing Company

[15] B. K. P. Horn, Robot Vision. Cambridge, MA: M.I.T. Press, 1986.

[16] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multirobot localization. Autonomous Robots (ARJ), 8(3):325–344, 2000.

[17] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. Artificial Intelligence (AIJ), 128(1-2):99–141, 2001.