Spinoza: A Stereoscopic Visually Guided Mobile Robot

Vladimir Tucakov, Michael Sahota, Don Murray, Alan Mackworth,
Jim Little, Stewart Kingdon, Cullen Jennings, Rod Barman *

Laboratory for Computational Intelligence
Department of Computer Science
University of British Columbia
Vancouver, British Columbia, CANADA V6T 1Z4
tucakov@cs.ubc.ca 604-822-6625 604-822-5485 FAX

Abstract

Our mobile robot, Spinoza, embodies a sophisticated real-time vision system for control of a mobile robot in a dynamic environment. The complexity of our robot architecture arises from the wide variety of tasks that need to be performed and the resulting challenge of coordinating multiple distributed, concurrent processes on a diverse range of processor architectures including Transputers, digital signal processors, and a workstation host. The system handles sensing, reasoning, and action components of a robot distributed over these architectures, and responds to unpredictable events in an unknown dynamic environment. Spinoza relies heavily on its capability to perform real-time vision processing in order to perform task such as mapping, navigation, exploration, tracking, and simple manipulation.

1 Introduction

Our mobile robot, Spinoza, embodies a sophisticated real-time vision system for control of a responsive mobile robot. Balancing the real-time constraints of a robot in a dynamic environment challenges the limits of both technology and our scientific understanding of embedded systems. Dynamic environments are unpredictable, asynchronous, and require a low latency in response, while visual information processing require high data-rate communications and significant computation.

*This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Networks of Centres of Excellence Institute for Robotics and Intelligent Systems, Project IS-6.

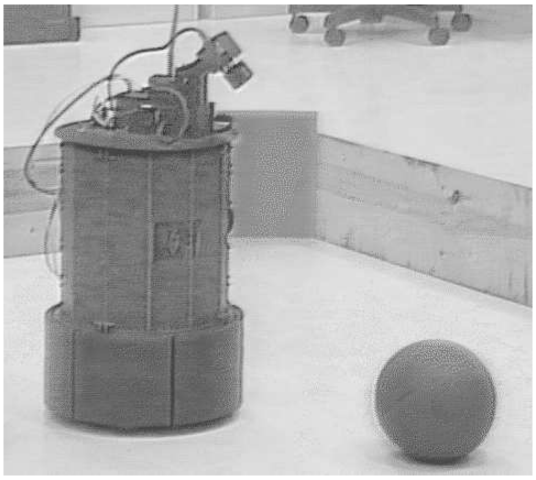

Figure 1: Spinoza: the visually guided mobile robot

Spinoza, as seen in Figure 1, is a self-contained robot, with host support. It consists of a Real World Interface (RWI) B-12 base with an RGB (colour) camera, mounted on a Directed Perception pan-tilt platform, on top, and trinocular monochrome stereo cameras in the body.

To provide a context for the design issues involved in this system, we begin by describing previous work in our lab on developing vision-based robotics systems that are antecedents of Spinoza. Also we present our research goals and a description the tasks Spinoza is to perform. Section 2 describes how the functional requirements of Spinoza and the development environment shapes the the choice of computational architecture and communication protocols. Section 3 describes the robot hardware. Section 4 describes in detail the workings of all the components of the robot from the hardware to the software. We finally present the experimental results and conclude with a discussion about the future of mobile robotics.

1.1 History of the Dynamo project

Spinoza is being built as part of a long-term project intended to develop a new approach to the specification, design and implementation of robotic systems. We will describe the elements of the system architecture that appeared in several of these robotic systems.

The UBC Vision Engine[13] is a general-purpose vision system that consists of multiple architectures: pipelined (a Datacube MaxVideo200) image processor and a MIMD multicomputer (20 T800 2MB Transputers, connected via a crossbar). These are connected by a bidirectional video-rate interface. The Vision Engine has been used in a range of visually-guide robots, such as an eye-head system: a pair of cameras on a pointable platform[15]. The eye-head follows a moving object, with no knowledge of its target, using dense optical flow input computed on the MaxVideo200. The Transputer system processes the flow data and controls the eye-head platform for vergence, pan and tilt.

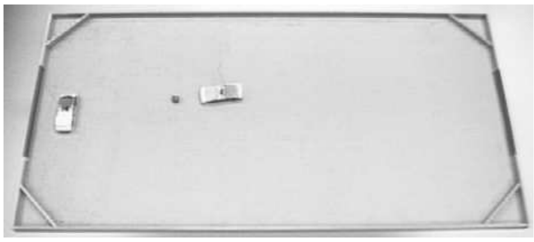

Figure 2: Dynamites: soccer playing robots

Another predecessor to Spinoza was the Dynamite testbed, shown in Figure 2. The Dynamite testbed is a collection of independently controlled mobile robot vehicles that play soccer [3, 16]. It has been used to explore novel reactive strategies for control[19] as well as for ideas on control, specification, and reasoning about real-time systems[20]. The system demonstrates offboard vision processing and distributed computation. The vision component was originally prototyped on the MaxVideo200 in the Vision Engine with the control programs running on the Transputers. Currently the system is realized as simple custom hardware to process RGB signals, followed by run-length encoding and centroid calculation on Transputers. A single offboard camera sensor communicates its signals to the centralized sensor processor. The sensor processor provides positional information to the control processes for each competing soccer player at 60Hz (once per image field) with a lag of at most 5 ms after the end of field. The structure of the full system is shown in Fig. 3. Four robots can be controlled concurrently.

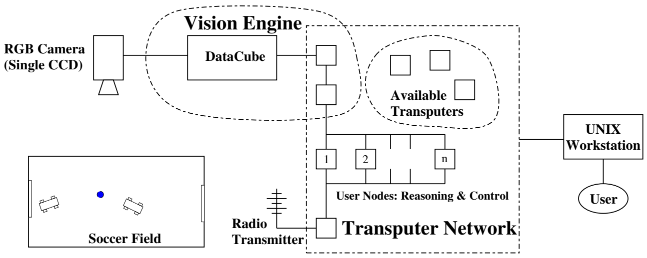

Figure 3: Dynamite Architecture

The "remote brain" idea, offboard visual processing, was used for soccer players because of size/weight limitations, and for our initial work with Spinoza: ROLL, (Real-time Onboard Localization with Landmarks) identifies its position in real-time, using passive visual localization of a single landmark[14]. ROLL used a set of TI C40 DSPs and transmitters to support offboard visual processing.

1.2 Project Goals

The scientific goals of the project include determining the power of vision as a sensing mode for a robot, how to integrate ongoing sensing, and how much knowledge and reasoning is needed for planning actions. In general, we seek to delve into the interaction between processing, perception, and action in a dynamic environment.

Other goals include learning about capabilities of situated/embedded vision, integration of technology, and exploration of dynamic environments. We wish to develop a flexible computing environment that can handle multiple visual tasks, processing modes, and cameras to support a visually guided robot that can operate outdoors.

1.3 Robot Tasks

At the highest level, we would like to build a robot that can, either under program control, or full or limited teleoperation, to act as a remote physical agent, to perform a range of tasks, including finding lab members, identifying whether equipment is busy, to check the status of the lab, to guide tours, to find things. More concretely, the robot's functions that have been completed to date include mapping, navigation, exploration, tracking, and simple manipulation. The requirements include fast vision processing, flexible controllers, high bandwidth communication, and support for high-level processes.The types of visual tasks required are as follows:

- blob detection—isolation of coloured objects for recognition and tracking
- stereo—passive distance estimation
- pointable camera—for tracking
- optical flow—computing perceived motion for segmentation, recognition, and obstacle detection
- tracking—visual servoing
- integration of multiple modes—cooperation among sensing systems (cameras) and vision processes

2 Models

In our research we are interested in developing formal theories that can provide systematic design and analysis methodology for perceptual robotic systems.

2.1 Robot Models

We need practical and formal design methods for building integrated perceptual robots. A robot is, typically, a hybrid intelligent system, consisting of a controller coupled to its plant. The controller and the plant each consist of discrete-time, continuous-time or event-driven components operating over discrete or continuous domains. The controller has perceptual subsystems that can (partially) observe the state of the environment and the state of the plant. Vision as a passive sensing system is cheap, reliable and biologically validated, so we are pushing the use of vision for mobile robots as far as we can.

The structure of the robot follows standard models [1, 22] that decompose the robot into sensing, reasoning, and action subsystems, each realized at a hierarchy of scales. The finest scale handles control loops with a 100 Hz rate and a 10ms time horizon, and operates synchronously. Each coarser scale reduces the rate by a factor of 10 and increases the time horizon by a factor of 10. At the highest level, the time horizon is on the order of 10s of seconds, and the system operates asynchronously. In practice the data flows up and down the time and space hierarchy are implemented as streams of messages passing asynchronously through the system.

The finest level is also responsible for reaction to stimuli such as looming objects, our "knee-jerk" reflex, which avoids approaching objects, in a dynamic environment, where transport of information to high levels would delay response unacceptably.

2.2 Programming Models

Robots are a typical class of hybrid systems. One of the most important challenges facing us is to develop theoretical and practical tools for designing hybrid embedded intelligent systems.

The hierarchical spatial and temporal structure of the levels in the standard robotic (such as Albus's NASREM architecture) map onto a set of software levels. The reasoning component dominates at the highest asynchronous level, while the lower levels, with control loops, are concerned with sensing (vision and robot base) and action (actuation and movement of the robot base.

We are working toward an implementation that will specify the controller for the robot using the terminology of Constraint Nets [21], but at present we use the layered approach to distribute the computation in a set of software modules that match the elements of the layers.

Much of the complexity of our robot architecture arises from the nature of the tasks and the challenge of coordinating multiple distributed, concurrent processes on a wide range of architectures. A robot engages in multiple asynchronous activities.

2.3 Development Environment

Spinoza has evolved, with new communication, computation, and sensing capabilities added over time. We have pursued an incremental strategy, extending its abilities to handle new tasks. To limit the degree of low-level programming, as well as to reduce the impact of changing hardware, a stable software interface to robot services was required. The actual robot hardware is varied, and cannot easily be mastered, thus a certain level of abstraction from the actual implementation was necessary.

The separation between abstraction and implementation, argued for by these considerations, leads to a model where the robot is isolated from the program development system by an interface. Controllers can be designed and tested in isolation from the actual robot; for example the development of controllers for the robot soccer players was facilitated by realistic simulation of the soccer players and their environment. This dovetails nicely with our desire to include teleoperation, the human intelligence and decision-making capability, in the robotic system.

2.4 Model/Hardware map

Hardware	Task	Comm.	Scale
C40s	Blob Detection	synch	> 10 Hz
	Stereo Vision	synch	> 1 Hz
Transputers	Robot Control	synch	> 10 Hz
Host	Teleoperation	asynch	< 1 Hz
	Mapping	asynch	< 2 Hz
	Planning	asynch	< 0.5 Hz

Figure 4: Levels and scales of hardware and associated tasks

Figure 4 illustrates how the hierarchy of robot tasks is mapped to actual hardware. Low-level synchronous high-speed control loops are run on specialized on-board hardware for performance. Higher level task such are reasoning and mapping are run on the host asynchronously in an event-based manner.

3 Hardware Architecture

How do we realize a system that meets our constraints and has the capabilities we need? One method of specifying robotic systems has been the “reactive” situated approach that exploits regularities of the task and environment of the robot[9, 2]. Typically these systems have simplified the sensing capabilities of the robot so as to meet the physical and cost limitations, suited to the task and environment. Horswill[10] has implemented a more general, but inexpensive processor, with limited capabilities. Others[6] move much of the signal and image processing offboard.

We chose to build a large amount of our computing requirements onboard our mobile robot. Figure 6 shows the computing system. A VME card cage, visible in Figure 5, holds four INMOS Transputer processors plus two Texas Instruments TMS320C40 digital signal processors. Our experience with the Vision Engine[13] showed that Transputers are suitable for implementing real-time controllers because they have low-latency communication capabilities and built-in lightweight scheduling and context switching. However, they lack the computational power and communication bandwidth for vision processing. For this reason we use C40 DSP processors, which support 25 MFlops computation plus 20MB/s communications, for the image acquisition and processing functions onboard the robot.

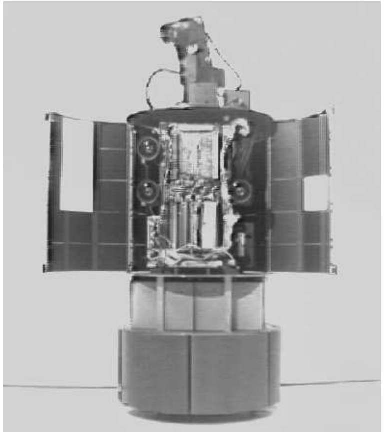

Figure 5: View of Spinoza's Hardware

Spinoza's C40 vision system is composed of an RGB video frame grabber and a specialized image processing module. The frame grabber can simultaneously grab from either the pan-tilt mounted colour camera or the three greyscale stereo cameras. The image processing module is a VIPTIM from Traquair Data Systems. The VIPTIM contains a cascaded pair of INMOS A110 convolvers that perform a 6x7 convolution at 10 Mpixels per second. Since the bulk of our early vision computation is filtering and matching, the hardware convolver greatly accelerates the overall processing speed of the vision system.

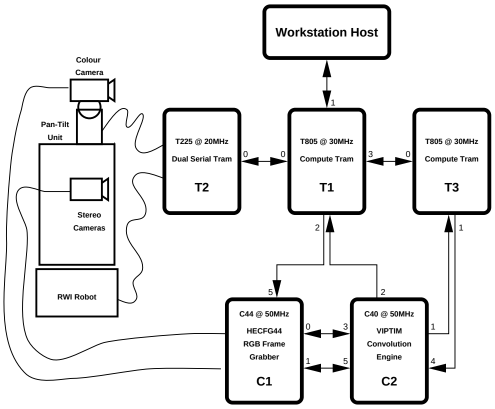

Figure 6: Spinoza Hardware

The Transputers are the heart of Spinoza's onboard computing system. They communicate with the C40 vision system, the workstation host and, through a serial Transputer module, the B12 base and pan-tilt unit controllers.

The host workstation is a Sun Ultra 1 connected to the robot through a Transputer link interface on its parallel port. The host can reset, boot and communicate with the robot at a speed of 20Mb/s bidirectionally.

The original version of Spinoza was tethered: the limitations of battery capacity and the large power demands of the onboard computers meant that power had to be supplied via a cable during extended use (longer than half an hour). The goals of the robot, however, include activity throughout the research labs and the entire building, hence we needed untethered operation. We added an additional battery pack so the robot is now capable of over two hours of untethered operation.

Untethered communication to the host workstation is through a spread spectrum radio modem with a bandwidth of 1.6 Mb/s. This raw bandwidth is reduced by the necessary layers above the raw transmission layer that provide reliable handling of packets. Currently the system gets 80KB/s across the radio modems. The transition to the wireless operation required the down scaling of diagnostic reporting, which often includes images. We are currently developing the ability to switch between tethered and radio modem operation "on-the-fly." This will allow high-speed communication for system development.

4 Software Architecture

The robot software was implemented on a variety of hardware architectures described in the previous section. The design of the software was challenging due to constraints posed by robot tasks as well as hardware limitations. Issues such as amount of available computational power and the communication bandwidth were closely examined.

Figure 7 presents the software architecture of the robot. The dashed line in the figure represents the physical separation between the robot and the host.

Software implemented on the robot is in charge of sensing and robot controls. The software implemented on the host does data integration, reasoning, and interacts with a human operator.

4.1 Vision Services

There are four cameras on board Spinoza: a colour camera ("top") on the pan-tilt unit (PTU) provides a pointable colour input useful for tracking; three monochrome cameras ("left", "right", and "upper") in a static "L" configuration are used for stereo ranging. The first DSP, called the grabber, grabs colour (RGB) images and stereo camera information from the three monochrome cameras. The stereo information is passed on to the second DSP for processing

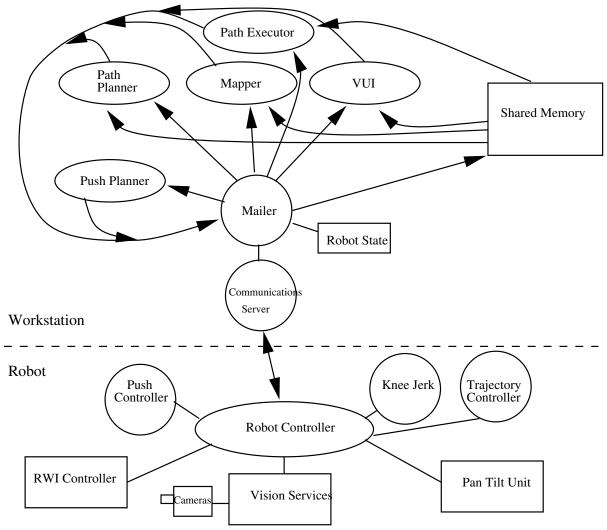

Figure 7: Spinoza Software Architecture

while the color information is used to find coloured blobs. Timely delivery of blob information is assured by having the VIP TIM probe the Grabber for blob information during stereo computation, which takes much longer than blob detection. The interconnections between these and the robot controller are shown in Figure 8.

The grabber regularly switches between two three-input camera systems. In one the three inputs are the left, right, and upper cameras, the trinocular inputs. In the other, the top RGB camera sends three signals containing Red, Green, and Blue separations.

Colour blob tracking is performed by first segmenting a colour image to a binary map. The centroid of all "on" pixels is the centroid of the target; speed requirement necessitate this simplification. While the blob is being detected, the DSP concurrently passes on the stereo images to the VIP TIM which performs trinocular stereo.

The reliability of stereo data is paramount in obstacle avoidance—stereo is computed in trinocular format, requiring slightly more computing, but with a useful increase in reliability [8]. Dense stereo citeBulLitPog89a,OkuKan93a permits obstacle avoidance without segmentation or interpretation as would be required by line-based stereo [18]. Trinocular stereo compares image patches along a fixed range of disparities, among three cameras roughly aligned in an "L" shape. Horizontal scene structures may be ambiguous from the left-right comparison, but will be separated by the upper-right comparison. Both comparisons create combined measure of support for a particular depth.

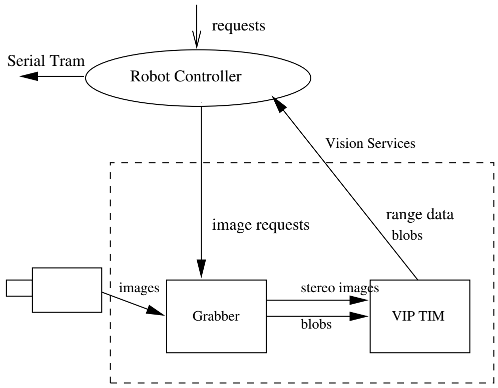

Figure 8: Vision Server

The VIP TIM DSP corrects for warping of images due to lens distortion and aligns the geometry so that the epipolar lines are alighted to the x and y axis. Cameras are calibrated [12] and the images correction mapping is computed off line, using Matlab. Images are first smoothed, then down sampled and corrected via a large table. This implements a "soft" calibration that can be redone on demand.

The stereo is then computed a multi–baseline correlation method[17]. This is implemented using the A110 convolver to do the stereo correlation.

This replaces stereo previously implemented on the Datacube system which could operate at 15Hz, but the Datacube does not fit into an embedded system [13]. Optical flow [4] can be implemented in similar fashion to the stereo on the VIP TIM, to support obstacle avoidance based on flow [5].

Sample Image Stereo Depth Image

Figure 9: Results of the stereo algorithm

Figure 9 presents an example of the results obtained by the stereo algorithm. The brighter shades of grey represent points in the scene that are closer to the robot. Likewise the darker shades of grey represent the points further away. The black areas of the image represent points for which distance can not be determined accurately. The system processes 128x128 pixel images at 20 disparities at 2 Hz.

4.2 Robot controller

The robot controller receives requests from the communications server. The controller coordinates the outputs sent to the robot actuators, suppresses redundant commands and repeat commands that have not been fully executed.

The robot controller regularly requests vision services such as blobs and stereo range data, at varying rates, depending on the task. The data is passed back to the host and distributed to the task modules. By initiating regular requests for stereo data, recent data is always available to service host requests without the otherwise unacceptable delay. Similarly, the robot controller regularly updates all other robot state information.

The robot controller communicates directly to the RWI controller, which controls the RWI base. The unit transformations and the communication through the serial port on the serial Tram are transparent. The serial Tram connected to the robot controller points the pan/tilt unit during tracking and pushing.

The stereo vision data sent from the vision services to the Robot controller may contain depth values that indicate that an object is "too close". The robot controller recognizes this situation, and acts to stop the forward progress of the robot. This tight loop between the RWI controller and the vision services must be implemented on the robot to minimize delay in reaction.

Paths come down from the path plan generator on the host as waypoints in the robot's coordinate system. To direct the motion of the robot along a smooth trajectory, the trajectory controller divides such a path into a sequence of tightly monitored commands that smoothly combine rotation and forward movement. Likewise a tight control loop is necessary to control the robot's movement while pushing objects along a specified trajectory. These capabilities: obstacle avoidance using stereo data, smooth trajectory execution, and obstacle pushing are all examples of the tight synchronous control loops required at the lowest level of software in our robot model.

4.3 Host Communication Server

The communication module provides message and data passing capabilities both robot $\leftrightarrow$ host and host interprocess communication (IPC). Several issues had to be considered in the design to meet various competing requirements.

The first problem, is that the robot–host bandwidth is a scarce resource, much in demand. If every time a host process requires robot state information a request is sent to the robot, the system would quickly bog down. To eleviate this, we had to design a system that would limit robot queries and reuse the results as much as possible.

Another difficulty was the number of host processes. There are many processes, which often require to communicate with several others. If each process had to create a separate connection to all other processes, the number of links would increase quadratically and communication management would quickly become unwieldy. Also, to minimize the impact of future changes in hardware and software architecture, we want to minimize the direct knowledge of system configuration required by each process.

The solution was to use a central data blackboard process we dubbed the "mailer". This process provides IPC through the mechanism of Unix message queues. This provides each process its only avenue to data and state information. Any process can request data by sending a request to the mailer via the mailer's message queue. It then performs a hanging read on its message queue until the requested data is available. Request take 3 forms:

New item the requested data or item must be queried from the robot. This request is restricted as much as possible.

Next item the process wishes to be provided with this data when it is next submitted (as new) to the blackboard

Last item the most recent data of this type is requested and provided without delay.

In addition to requests, a process may provide an updated data product for the blackboard. Host processes generally follow a "producer-consumer" model. That is, the process waits for the next issue (or production) of its input data, performs its task, then produces a resultant item for other processes to consume.

This design has many advantages. One is that it is host CPU friendly. Each time a process is complete, rather than polling communications or repeating operations on already processed data, it will suspend until new data arrives. This conserves system resources. The system provides message passing that can work both as events in an event-driven system, or as a synchronization method between essentially asynchronous processes.

It also means it is easy to test and debug software, even if the robot is not available. It is easy to simulate robot state messages, and the mailer can run on any workstation. It also has the advantage that modules have no other point of contact that the mailer, and thus require no knowledge of system configuration, other than the address of the mailer message queue. New processes can be added at anytime and will automatically have access to information available.

For example, when a user requests the robot to move to a new location by clicking on the displayed map, the user interface produces a "goal update". This unblocks the path planner which has posted a request for the "next goal". The path planner makes the path, and posts a "path update". The path executor is in turn waiting for the "next path". It receives the new path and issues robot commands appropriately, again through the mailer.

4.4 Mapper

As described in Section 4.1, dense depth images are regularly constructed by the trinocular stereo vision service on-board the robot. These depth images can be reduced to represent the nearest obstacles by projecting all sensed points down through a vertical column to the plane of the floor. The depth image is reduced to a single row of disparities representing the closest obstacle as seen from a top-view perspective. This 2-D map has high angular resolution; however, range uncertainty varies proportionally with the depth. This representation is much smaller than full depth images and are much cheaper to send to the host.

In the host, the radial depth maps are routed by the mailer to the mapper module. The mapper application integrates these directional range maps into a 2-D map represented by an occupancy grid [7]. Such a map is represented by a tessellation of the mapped space into a grid. The value of each grid is related to the probability that this space is occupied by any part of an obstacle. The "mapper" initializes the map to contain only values at 50% probability, indicating that the entire space is unknown. As new range maps arrive, the mapper updates the occupancy grid so that each cell contains an updated probability that the cell is occupied by an object. Every point between the current position of the robot and the nearest obstacle in a given direction is marked clear. The probability of the cell at the given range is updated, combining its previous value with the uncertainty of the range estimate. Cells beyond the object detected are unaffected.

In a sense, the mapper acts as a smart memory, that integrates the information over time into a coherent whole, and buffers data between the vision service and the client.

4.5 Path planner

As its name suggests, the path planner produces paths for the robot to follow. As such, it is used by those processes that move the robot from one position to another. The path planner takes as input a map of the environment produced by the mapper, a goal position, and an initial position (specified as triples of X, Y, and theta values). It returns a sequence of X, Y, theta triples that denote significant waypoints along the path generated by the planner.

Paths are generated in the following manner. A simple wavefront expansion[11] algorithm is used to generate a unimodal, potential field between the goal and initial position. Then, an initial path is generated by following the gradient of this field. Waypoints along the path are generated by starting at the initial position and selecting the last point on the path for which there exists an unobstructed, straight line path from the initial state to the goal. This point is marked as a waypoint and the process is repeated from the new waypoint until the goal position can be seen.

4.6 Task module

The task module is the end-user programming environment that allows the use of all lower-level functionalities. Programming at this level is sequential, ie. get an image, move, get an image, turn right etc. While high level programming is sequential, the servers communicating information from other modules run in parallel. The purpose of the task module is to shelters the developer from changes in the underlying hardware of the robot. Ideally the code written in this module would need only recompiling, when a change in the hardware is made.

4.7 Visual User Interface

This module, the Visual User Interface (VUI), is an example of a user level application. VUI provides a graphical user interface to some of the task modules and well as to the interface to the robot abstraction. The interface can access the state of the robot including its battery charge, odometry, images as currently seen. The VUI can display the map of the environment. By pointing and clicking user can specify a new goal location for the robot.

5 Results

Spinoza has demonstrated a number of tasks it can accomplish. These tasks include, chasing a brightly coloured ball, avoiding collisions with dynamic objects, pushing a box along a specified path and exploring and mapping a static environment.

5.1 Ball Chasing

The ball chasing demonstration was designed to show that all levels of onboard architecture can work coherently. The task of the robot was to find a brightly coloured ball and keep it in the field of view. The body of the robot moved slower than the pan-tilt camera. The robot took advantage of the pan-tilt camera when the body of the robot was not pointing towards the ball. When the body of the robot was pointing towards the all, the robot would move until it reached a specified distance from the ball (about 30cm). Robots task at that point was complete. If the ball was moved the robot would chase it again. The challenge of this robot task was in synchronizing the communication between control modules as well as doing fast vision processing.

5.2 Obstacle avoidance

While colour camera was used for detecting dynamic objects in the environment, three cameras in the robot body were used for stereo vision. The results from these cameras produced a depth map of the environment. The performance of the stereo algorithm was first tested by letting the robot move forward in its environment. If an obstacle was encountered the robot would choose to move left or right depending on where the obstacle was. Obstacles were defined as any object that is closer than a specified distance (about 30cm). The robot was able to avoid static objects in the environment as well as dynamic objects such as people walking in front of the robot.

5.3 Box Pushing

The robot can interact with its environment by pushing objects. To demonstrate this the robot was programmed to push a box along a specified path. The challenge of this task was in locating the position of the box relative to the robot and controlling the motion of the box. The pan-tilt camera was used to locate the position of the box and the bump panels were used to push the box. The robot was able to push the box along a specified triangular path. The box was kept at all times within 5 cm of the specified path.

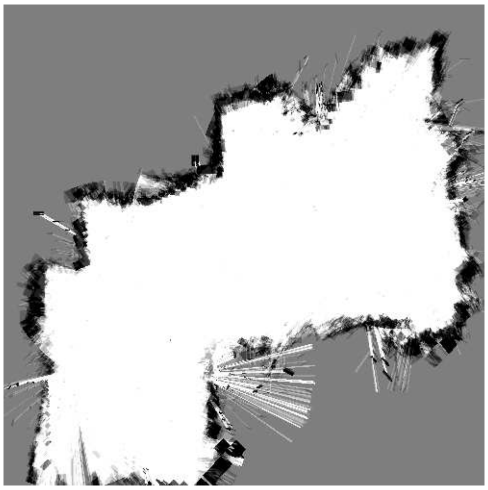

Figure 10: Occupancy Grid Map

5.4 Mapping

An example of a map built autonomously by Spinoza is shown in Figure 10. The grey regions of the map represent areas unseen by the robot. The white regions are areas where it is known that no obstacles exist, and the black regions are locations of known obstacles. This map was made on a 500×500 grid, each grid representing a 2×2 cm square. The robot was able to autonomously explore and map out this region in less than 10 minutes.

6 Conclusion

Spinoza demonstrates a sophisticated real-time vision system for control of a responsive mobile robot, operating in dynamic environments. It is a complex system, coordinating multiple distributed, concurrent processes on a wide range of architectures, and performing a range of asynchronous activities. Its design represents the resolution of conflicting design requirements: high processing capability and telerobotic guidance, under the limitations of a mobile robot: power, heat, space, and communication bandwidth.

The performance gap between specialized digital signal processors and standard personal computers is rapidly shrinking. Therefore the tradeoffs of using embedded hardware versus conventional computer hardware need to be closely examined. Our experience suggests that the costs, in terms of development time and debugging time, are restrictively high using C40s and Transputers. In the future we plan to experiment with Intel based CPUs for vision processing as well as robot control.

References

[1] James Albus. *Brians, behaviour, and robotics.* BYTE Publications, 1981.

[2] R. L. Andersson. *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control.* MIT Press, Cambridge, MA, 1988.

[3] R. Barman, S. Kingdon, J.J. Little, A.K. Mackworth, D.K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang. DYNAMO: real-time experiments with multiple mobile robots. In *Intelligent Vehicles Symposium*, Tokyo, July 1993.

[4] H. Bulthoff, J. J. Little, and T. Poggio. A parallel algorithm for real-time computation of optical flow. *Nature*, 337:549–553, February 1989.

[5] David Coombs, Martin Herman, Tsai Hong, and Marilyn Nashman. Real-time obstacle avoidance using central flow divergence and peripheral flow. In *Proc. 5th International Conference on Computer Vision*, pages 276–283, June 1995.

[6] Gregory Dudek, Paul Freedman, and Ioannis M. Rekleitis. Just-in-time sensing: efficiently combining sonar and laser range data for exploring unknown worlds. In *Proc. IEEE Conf. on Robotics and Automation, 1996*, pages 667–672, April 1996.

[7] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6):46–67, June 1989.

[8] C. Hansen, N. Ayache, and F. Lustman. Towards real-time trinocular stereo. In *Proc. 2nd International Conference on Computer Vision*, 1988.

[9] I. D. Horswill and R. A. Brooks. Situated vision in a dynamic world: Chasing objects. In *AAAI-88*, pages 796–800, St. Paul, MN, 1988.

[10] Ian Horswill and Masaki Yamamoto. A $1000 active stereo vision system. In *Proc. Workshop on Visual Behaviors*, pages 107–111, 1994.

[11] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer, 1991.

[12] R. K. Lenz and R. Y. Tsai. Techniques for calibration of the scale factor and image center for high accuracy 3-d machine vision metrology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:713–720, 1988.

[13] J. J. Little, R. A. Barman, S. J. Kingdon, and J. Lu. Computational architectures for responsive vision: the vision engine. In *Proceedings of CAMP-91, Computer Architectures for Machine Perception*, pages 233–240, December 1991.

[14] James J. Little. Vision servers and their clients. In *Proc. 12th International Conference on Pattern Recognition*, pages 295–299, October 1994.

[15] James J. Little and Johnny Kam. A smart buffer for tracking using motion data. In *Proc. Workshop on Computer Architectures for Machine Perception*, pages 257–266, December 1993.

[16] Rod A. Barman Michael K. Sahota, Alan K. Mackworth and Stewart J. Kingdon. Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 3690–3663, 1995.

[17] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–363, 1993.

[18] S. B. Pollard, J. Porrill, J. E. W. Mayhew, and J. P. Frisby. Matching geometrical descriptions in three-space. *Image and Vision Computing*, 5:73–78, 1987.

[19] Michael K. Sahota. Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. In *Proc. 12th National Conference on Artificial Intelligence*, pages 1303–1308, 1994.

[20] Y. Zhang and A. K. Mackworth. Modeling behavioral dynamics in discrete robotic systems with logical concurrent objects. In S. G. Tzafestas and J. C. Gentina, editors, *Robotics and Flexible Manufacturing Systems*, pages 187–196. Elsevier Science Publishers B.V., 1992.

[21] Y. Zhang and A. K. Mackworth. Constraint nets: A semantic model for real-time embedded systems. *Theoretical Computer Science*, 138:211–239, 1995.

[22] Y. Zhang and A. K. Mackworth. Synthesis of hybrid constraint-based controllers. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II, Lecture Notes in Computer Science 999*, pages 552–567. Springer Verlag, 1995.