The BIRIS server: integration of a three dimensional range sensor into a Harmony-based realtime architecture

Green, D. A.; Blais, F.

Report (National Research Council of Canada. Radio and Electrical Engineering Division : ERB), 1992-04

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

ERB-1036

NRC·CNRC

The BIRIS Server: Integration of a Three-dimensional Range Sensor into a Harmony-based Realtime Architecture

D.A. Green and F. Blais
April 1992

Canada

NRC No. 33183

The Institute for Information Technology (IIT) is committed to assisting industry across Canada to achieve excellence in information technology software and systems by promoting, supporting, and undertaking focused, industry-oriented research and development programs in these areas. To achieve this mission, IIT works in collaboration with industrial, educational, government, and other organizations.

There are four elements to the Institute's research program: software engineering and knowledge-based systems as part of software technology; sensor-based automation and systems integration as part of systems technology.

There are four laboratories in the Institute, with each laboratory responsible for one research element.

The Institute invites inquiries with regard to its mission and research program and will be pleased to discuss any proposal for research or collaborative activity. Contact the Industrial Liaison Office at (613) 993-2491.

L'Institut de technologie de l'information (ITI) est chargé d'aider l'industrie partout au Canada à atteindre l'excellence en matière de logiciels et de systèmes de la technologie de l'information en faisant la promotion de programmes de recherche-développement focalisés et tournés vers l'industrie, de même qu'en appuyant et en entreprenant de tels programmes. Pour s'acquitter de sa mission, l'ITI collabore avec des organisations de l'industrie, de l'enseignement et du gouvernement et avec d'autres organismes.

Le programme de recherche de l'ITI comporte quatre éléments : génie logiciel et systèmes à base de connaissances dans le cadre de la technologie des logiciels; automatisation à base de capteurs et intégration des systèmes dans le cadre de la technologie des systèmes.

L'Institut compte quatre laboratoires; chaque laboratoire est chargé d'un domaine de recherche.

L'Institut se fera un plaisir de répondre à toute question concernant sa mission et son programme de recherche, et de discuter de tout projet de recherche ou d'activité conjointe. Communiquer avec le Bureau de liaison industrielle au (613) 993-2491.

Additional copies are available free of charge from:

Editorial Office, Room 301
Institute for Information Technology
A.G.L. McNaughton Building (M-50)
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R6

Des exemplaires supplémentaires peuvent être obtenus gratuitement à l'adresse suivante :

Bureau des publications, Pièce 301
Institut de technologie de l'information
immeuble A.G.L. McNaughton (M-50)
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R6

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

The BIRIS Server: Integration of a Three-dimensional Range Sensor into a Harmony-based Realtime Architecture

D.A. Green and F. Blais
April 1992

Contents

Abstract/Résumé v
Introduction 1
Functionality of the BIRIS Server 1
  The BIRIS Sensor 1
  The BIRIS Server 1
System Configuration — Hardware 2
  Overview 2
  Interrupts 2
Interface Mechanics 3
  Command and Control 3
  Access to BIRIS Data 4
System Configuration — Software 4
  Tasks 5
  Future Tasks 6
High-level Software Interface 6
  Messages from Client Tasks 6
  Messages from Internal Tasks 7
  Messages from Internal Tasks Reporting System Errors 8
Initialization of the BIRIS Server 8
Summary 8
References 8
Appendix A. The Structure Definition of the Server Initialization Record 11
Appendix B. An Example of the Server Initialization Record 12
Appendix C. BIRIS Server Design Details 13
  Drive Mechanism and Scan Control 13
  Driving to the HOME Position using the BIRIS_HOME Case 13
  Command Transfers from the Server to BIRIS 13
  Transfer of Data from BIRIS to the Server 13
  Synchronization of the Number of BIRIS Data Acquisitions with the Scan Length 14
Appendix D. Proposed Changes 15
  Continuous Operation 15

Figures

1. BIRIS configuration on the mobile robot. 2
2. A portion of the data structure used for BIRIS control. 3
3. Organization of BIRIS data. 4
4. Harmony task configuration. 5

Abstract

This report describes the method and rationale that was used to interface the BIRIS 3-D laser range sensor to a multiprocessor system that runs under the Harmony realtime operating system. The report is primarily concerned with a description of a "server" that presents to the application programmer a clear abstraction of the BIRIS system so that control of the device and access to its data are uncomplicated. The system described here is a component of a mobile robot system being developed by the Autonomous Systems Laboratory at NRC.

Résumé

Ce rapport décrit la méthode et les principes qui ont présidé à l'interfaçage entre le télémètre laser 3D BIRIS et un système à multiprocesseur qui tourne sous le système d'exploitation en temps réel Harmony. Le rapport porte surtout sur la description d'un «serveur» qui présente au programmeur d'applications une abstraction nette du système BIRIS, de sorte que la commande du dispositif et l'accès à ses données sont des opérations simples. Le système décrit est une composante du système robotique mobile en cours de développement par le Laboratoire d'automatique du CNRC.

Introduction

The NRC mobile robot project [1] has relied upon a sonar array and until recently a simple laser slit scanner as range sensors to support ongoing experiments in sensor-based collision avoidance. BIRIS [2], which was developed by the Autonomous Systems Laboratory at NRC, is now replacing the laser slit scanner and is being integrated into the Harmony[1]-based system architecture [3–5]. This has occurred because BIRIS can provide full 3-D range data that the slit scanner is incapable of supplying and because it offers better noise immunity than the slit scanner. The BIRIS system is now being installed on the vehicle.

BIRIS in its present form projects a slit of laser light with a 22° vertical dispersion that is observed by a CCD camera fitted with a custom built double aperture lens. This lens produces a split image of the surface that is illuminated by the projected laser line. The separation of the images of the line at each point is related to the distance to the illuminated surface. The number of range samples obtained from the image of a single projected line is programmable; currently it is 64. In this implementation, the sensor head is scanned mechanically by a DC servo motor which is computer controlled.

The BIRIS system configured for this application offers short range sensing limited to about 3.5 m. This is adequate for local navigation purposes. At present, two immediate applications are being considered: the first is the use of BIRIS for geometric mapping of the local environment for path planning collision avoidance, the second is the integration of BIRIS range data directly into the certainty grid which is used for reactive collision avoidance [6] to improve its resolution and performance. Other applications include initial studies in environment modeling for localization purposes.

The initial implementation of the BIRIS system on the mobile robot uses a single laser projector and two video fields per acquisition (i.e., an acquisition rate of 1/30 s) and only a single degree of freedom (azimuth scanning). Elevation scanning can be added, if required, since an uncommitted servo controller is available. Furthermore, only 2-D data are currently made available with each acquisition since only the range to the closest point along the entire projected laser line is actually transferred to the Harmony system. This is a software limitation which will be changed so that full 3-D data will be available.

This report is primarily concerned with a description of a "server" that has been designed and implemented to interface the BIRIS system to the runtime multiprocessor. The role of the server is to present to the application programmer a clear abstraction of the BIRIS system so that control of the device and access to its data are uncomplicated.

Functionality of the BIRIS Server

BIRIS will be used primarily as a source of realtime range data for local navigation purposes. However, it is expected that there will also be a need, as determined by some users, to supply range data and vehicle position data to remote systems for off-line analysis related to environment modeling. It is expected that both needs can be satisfied through the Harmony system — using the BIRIS server described here, the K2A server, and a planned Ethernet server.

The BIRIS Sensor

The BIRIS sensor is described in detail in ref. 2. Physically it consists of a sensor head and a processor system. The head is comprised of a small CCD camera with a double aperture lens and a 19.5-mW laser projector mounted on a DC servo motor. The processor is configured from a number of commercial VMEbus products and is packaged in a single VME rack.

The BIRIS Server

The abstraction of BIRIS that is presented to the application programmer is determined by the BIRIS server. In its initial form, the server presents a very simple abstraction. The application task can instruct BIRIS to acquire data using either a single scan mode or a continuous scan mode of operation. Furthermore, BIRIS data in polar form (range vs. azimuth) that have already been acquired can be transferred to the client task on request. Clockwise and/or counterclockwise scan limits must be explicitly provided with each scan request. Auxiliary requests allow the scanning velocity or other motor control parameters to be changed. Note that requests for scan data occur asynchronous to

[1]Mark reserved for the exclusive use of Her Majesty the Queen in right of Canada by the National Research Council of Canada./ Marque déposée réservée à l'usage exclusif de sa Majesté du chef du Canada par le Conseil national de recherches du Canada.

actual data acquisition. The initial implementation of the server responds by supplying the latest complete scan. This results in an inherent delay between acquisition and presentation of data when continuous scans are being acquired. The magnitude of the delay is determined by the scan duration. Because this delay may jeopardize the performance of the robot in time-critical collision avoidance applications, methods to improve the presentation of the data will be considered in the near future.

Note that the BIRIS server is derived directly from the Slit server that was written for the laser slit scanner. Differences include the removal of vehicle position data from the server since in recent applications these data are obtained directly from the K2A server that controls the mobile robot, the removal of the event generation mechanism, and the removal of the proximity alarm (hazard detection) mechanism.

System Configuration — Hardware

Overview

Figure 1 illustrates the system hardware configuration. Two independent VMEbus chassis are linked through a bus window provided by a VME–VME (A32/D32 IRQ1* – IRQ7*) bus coupler (Bit 3, model 412) [7].

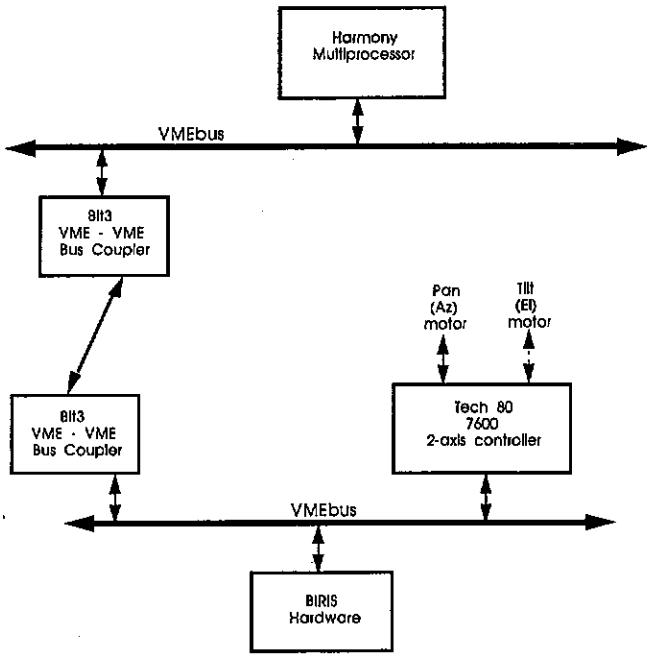

Figure 1. BIRIS configuration on the mobile robot.

The separation is necessary because the large bus bandwidth demanded by the BIRIS Datacube hardware would seriously degrade the multiprocessor performance if they were to operate on a common shared bus. The bus coupler is configured to provide a memory mapped window from the Harmony multiprocessor into the BIRIS system. Furthermore, a variety of interrupts are conveyed from BIRIS to the multiprocessor for synchronization purposes.

Panning of the BIRIS sensor head is directly managed by tasks running under Harmony using a model 7600 dual axis servo controller manufactured by Technology 80 Inc. [8]. The second axis is available for tilt control if required in the future. The controller includes on-board low power drive amplifiers. An interrupt announces the end of the current azimuth scan. (In the future, elevation motions will also raise interrupts.) During initialization, the zero index of each drive is found by polling in order to define a 0° azimuth ("home") position.

Interrupts

Since Harmony systems are preemptive in nature, it is essential that all interaction with the outside world be interrupt driven if Harmony is to be used to full advantage. In this way, an efficient task structure can be created to respond effectively to external events; this is the essence of realtime systems. Consequently, a number of interrupts are provided in the BIRIS system to satisfy this requirement.

Note that Harmony distinguishes between logical interrupts and physical interrupts. A number of logical interrupts may share the same physical interrupt level since they are demultiplexed in Harmony by an interrupt service routine. In Harmony, a logical interrupt is used to trigger a task that is blocked waiting for that interrupt. The interrupts are listed below (listed as device codes as they appear in the source code).

BIRIS Interrupts

Note that in normal operation, after a command is issued, a number of BIRIS_DATA_DEV interrupts, each announcing the completion of a single acquisition, will occur and will be followed by a BIRIS_CMD interrupt that defines the end of an acquisition sequence.

- BIRIS_DATA_DEV (physical level 2) — issued by BIRIS to indicate the completion of a single data acquisition. Note that this interrupt is asserted by

BIRIS unconventionally during an acquisition and not after that acquisition is complete. Therefore, to avoid acquiring invalid data, this interrupt is used by the interrupt service routine to indicate that the previous acquisition is available.

- BIRIS_CMD (physical level 2) — issued by BIRIS to indicate the completion of an acquisition sequence. A new BIRIS command must not be issued until this interrupt has been asserted and serviced.

Motor Controller Interrupts

- MOTOR_END_OF_SCAN (physical level 3) — issued by the motor controller to announce the completion of an azimuth motor scan.

Failure Interrupts

- BIT3_PERR (physical level 2) — interrupt issued by the bus coupler to indicate that a Parity error has been detected during a transfer across the bus interface.

- BIT3_BERR (physical level 2) — interrupt issued by the bus coupler to indicate that a bus cycle initiated by the bus coupler on the BIRIS VMEbus has failed due to a Bus Error.

The last two interrupts are used to announce fatal hardware failures.

Physically, all interrupts associated with BIRIS are bus vectored. A local interrupt vector register which resides on the bus coupler board that is plugged into the multiprocessor VMEbus contains the interrupt vector for all level 2 (BIRIS) interrupts. The interrupt vector for the motor controller is physically located in an interrupt vector register on the motor controller circuit board.

Interface Mechanics

Command and Control

The BIRIS system used here is a prototype that has been configured mainly from commercial hardware manufactured by Datacube. This system is controlled by a 68020-based single-board computer manufactured by Motorola that operates under the OS9™ operating system from Microware. This board supervises the BIRIS system. The system contains a nonvolatile RAMdisk which is used by the 68020 supervisor for automatic initialization of BIRIS.

A data structure that is created and managed by the 68020 supervisor can be used to control BIRIS remotely. This structure also appears in the Harmony multiprocessor memory map, by way of the bus coupler. The structure, Fig. 2, contains a command register, a status register, and a variety of other components which are used to point to data structures within BIRIS or to enable or disable specific BIRIS features. In short, remote control of the BIRIS system involves manipulating this data structure.

To issue a typical command to BIRIS, for example, a parameter string is copied to the PARAM[] element, and a code is written to the command register. If the _Biris_cmd_int_enable flag has been asserted, then an interrupt will be generated to the multiprocessor bus when actions resulting from the current command have been completed. The interrupt service routine will demultiplex the interrupt in part by examining the command register in the structure. If it is nonzero, then the interrupt is assumed to be a Biris Data interrupt. If it is zero, then a command interrupt has occurred since it is cleared by BIRIS on completion of command execution.

```
typedef struct BIEXEC_COM                    /* subset of BIRIS definition */
{
    Int_16 COMMAND;                          /* command register */
    Int_16 STATUS;                           /* status register */
    Int_16 ACQ_LINE;                         /* acquisition number */
    char   PARAM[BIEXEC_PARAM_SIZE];         /* parameter string */
    Int_16 CURRENT_FRAME;                    /* current frame in this sequence */
    Int_16 CURRENT_PROJ;                     /* current projection line in this acquisition */
    Int_16 _Biris_cmd_int_enable;            /* command interrupt enable flag */
    Int_16 _Biris_data_int_enable;           /* data int enable flag */
    Int_16 _Biris_motor_access;              /* enables BIRIS access to motor */
};
```

Figure 2. A portion of the data structure used for BIRIS control.

Access to BIRIS Data

A data array appears in the BIRIS memory map (Fig. 3). This memory also appears in the memory map of the multiprocessor by way of the bus coupler. A pointer to the base of this array (Biris_range_ptr) is obtained by the remote system by issuing a command to BIRIS: "BI_GET_VME_ADDR_DIST". The pointer is returned as an ASCII string in the PARAM[] element. Range data arrays and intensity data arrays alternate in BIRIS memory. The size (Biris_frame_size) of each array, i.e., either a range data array or an intensity data array, is determined by issuing a "BI_GET_VME_INC_FRAME" command to BIRIS.

The command used to initiate a BIRIS acquisition requires a parameter that specifies the number of acquisitions to be performed. This parameter is written directly to the PARAM[] element. In the server, the number of acquisitions for each scan is computed from the duration of the requested scan (see the last item in Appendix C). A single acquisition contains a fixed number of data points. At present there are 64 points and the first element of the range array contains the closest sample in that acquisition. In the initial implementation of the BIRIS server, these are the only range data that are returned. The ACQ_LINE element of the control structure contains the index number of the current acquisition in the sequence. It also is the index used in the azimuth array to find the motor azimuth at the time that that acquisition was made.

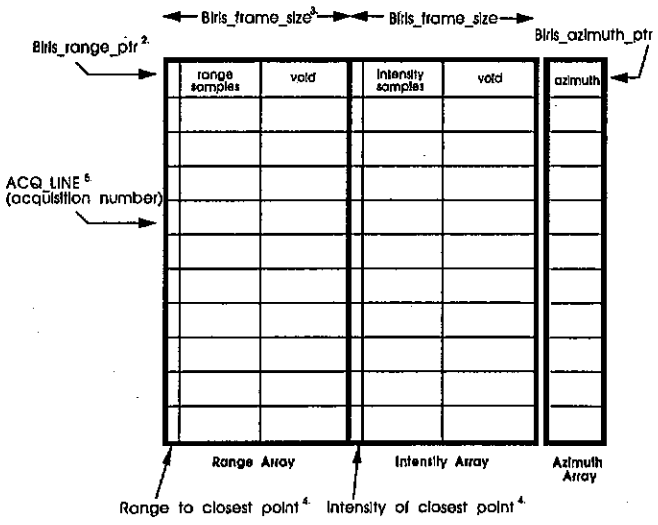

Figure 3. Organization of BIRIS data. Notes: 1. The number of samples in each acquisition is related to the acquisition rate: 64 samples @ 2 fields/acq. (30 Hz rate); 128 samples @ 3 fields/acq. (20 Hz rate). 2. The base of the range array is obtained from the BIRIS command: "BI_GET_VME_ADDR_DIST." 3. The BIRIS_frame_size is obtained from "BI_GET_VME_INC_FRAME" command. 4. The first element of each array contains the range or intensity of the closest point for that acquisition. 5. Each entry is indexed by the acquisition number in the current sequence.

In summary, with the arrival of a BIRIS data interrupt, the current acquisition can be located in BIRIS memory from the relationship

[Biris_range_ptr
+ (2 * ACQ_LINE * Biris_frame_size)]

Some special cases exist for ACQ_LINE. During startup, ACQ_LINE is coded as –1 and the associated acquisition is discarded. The remaining acquisitions are read by indexing directly with ACQ_LINE. At the end of an acquisition sequence, ACQ_LINE is coded as –128.

System Configuration — Software

The Harmony task structure that has been implemented is outlined here. The BIRIS server communicates with eight low-level tasks (Fig. 4). Typical of all servers, the BIRIS server is normally blocked awaiting a message from either a client task, a notifier task, or one of its worker tasks. In Harmony, a notifier task is a task that normally waits for a specific interrupt and then, by sending a message, notifies another task that the interrupt has indeed occurred. With the arrival of a message at the server, an appropriate case statement is executed and the server again blocks awaiting the next request. The next section "High-level Software Interface" describes the messages that are handled by the BIRIS server; this section describes the role of the various low-level tasks that are part of the BIRIS server. Note that the application programmer is not aware of these tasks.

The BIRIS server synchronizes the acquisition of BIRIS data and the control of the scanning motor. When a request for a scan is received, the server initiates a sequence of actions which ensure that the notifier tasks are all blocked awaiting the interrupts which will occur at the conclusion of the scan, that the motor is issued the correct drive command, and that BIRIS is issued the correct acquisition command. At the end of the scan, BIRIS will report with an interrupt that the acquisition sequence has finished and the motor controller will also report with an interrupt

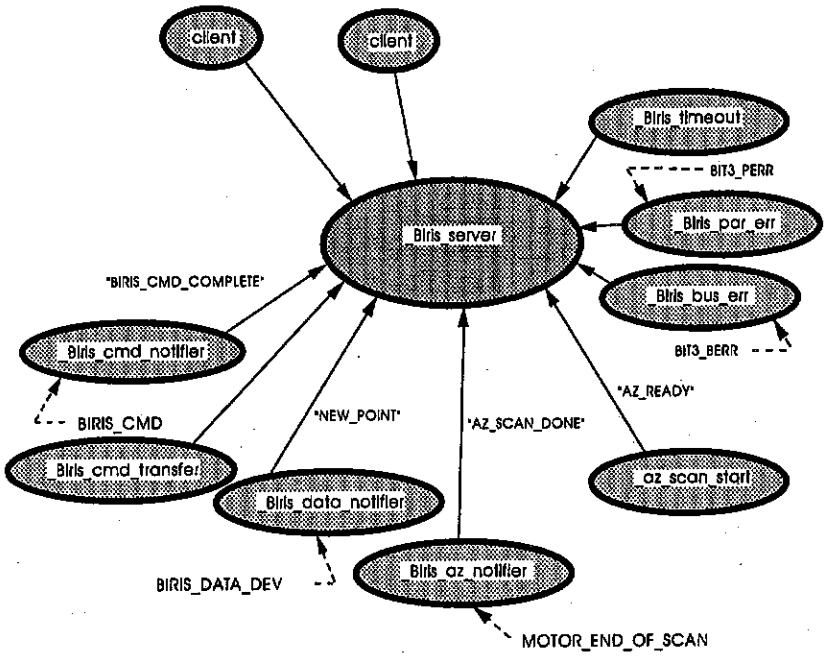

Figure 4. Harmony task configuration. Note: each notifier task is normally blocked awaiting its interrupt.

when motion has terminated. These actions are synchronized internally by the server before a new BIRIS acquisition can be initiated. Note that the only difference between a single scan request and a continuous scan request is that, in the continuous case, the next sequence is initiated automatically. In all cases, incoming data are buffered according to scan direction.

Tasks

_Biris_cmd_notifier

This notifier task receives the BIRIS_CMD_COMPLETE interrupt from BIRIS indicating that the last command transferred to the BIRIS system has completed. It is used in conjunction with the AZ_SCAN_DONE message from the _Biris_az_notifier task to determine that a BIRIS scanning operation has completed; i.e., all data have been received AND the motor has stopped.

_Biris_cmd_transfer

This worker task is normally blocked waiting for a reply from the server. When the reply message does arrive, it contains the next command to be transferred from this task to BIRIS. The use of a separate task here provides a simple method to guarantee synchronization with the BIRIS system.

_Biris_data_notifier

This task receives interrupts at the video frame rate (1/30 s) and sends a NEW_POINT message to the BIRIS server task at this rate. This notifier collects the latest range data from the BIRIS system and the current azimuth data from the motor controller. These data are included in the NEW_POINT message to the server.

_Biris_az_notifier

This notifier task is normally blocked awaiting an interrupt from the azimuth servo controller indicating completion of the scan. Upon receipt of the interrupt, the task will send an AZ_SCAN_DONE message to the server. This is used along with the BIRIS_CMD_COMPLETE message to synchronize the operation of the motor and the BIRIS system in preparation for the next scan.

_az_scan_start

This worker task is used to start a new azimuth scan. It is normally blocked waiting for a reply message from the BIRIS server. When the server wishes to start a scan, it will transfer a message to this task which will specify the destination of the next scan. The task then unblocks and starts a scan directly by communicating with the motion controller. It then sends the AZ_READY message to the BIRIS_server and blocks to indicate that it is again ready.

_Biris_timeout

If the BIRIS timeout mechanism has been enabled, this task will be created dynamically whenever a BIRIS command is issued. The task uses the hardware clock to generate an interrupt after a 4 s delay. If the clock times out, then this task is dispatched by the arrival of that interrupt and will immediately send a message to the BIRIS server to signal a timeout failure. Normally, this task is destroyed by the BIRIS server before the clock times out and so an error report is avoided.

_Biris_parity_err

This notifier task is created to report communication errors that may occur in the VME–VME bus coupler. In practice, these errors do not occur except in cases of fatal hardware failures. This task blocks and awaits a Parity Error interrupt from the bus coupler. If such an interrupt is observed, a message is transferred to the BIRIS server.

_Biris_bus_err

This notifier task is used to report fatal Bus Errors that occur on the BIRIS VME chassis due to a failed bus cycle initiated by the bus coupler. In practice, these errors do not occur except in cases of fatal hardware failures. This task blocks and awaits a Bus Error interrupt from the bus coupler. If such an interrupt is observed, a message is transferred to the BIRIS server.

Future Tasks

The following tasks will be required if a tilt axis is to be added to the BIRIS sensor head. They are similar to the motor tasks listed above except that they are intended to be used for the tilt motor (elevation scan) rather than the pan motor (azimuth scan). Note that, for each change in elevation angle, a new BIRIS calibration is required in order to express the range measurements in a single Cartesian frame of reference (the vehicle frame of reference or the world coordinate system). This can be implemented within the BIRIS system by reloading the internal lookup tables. A calibration must exist for each angle of elevation that is chosen.

_el_scan_notifier

Similar to above except that it applies to the elevation scan.

_el_scan_start

Similar to above except that it applies to the elevation scan.

High-level Software Interface

The abstraction presented by the BIRIS server is realized through a set of predefined messages which are sent to the server by client tasks. These messages are concerned with configuring and controlling the sensor, requesting BIRIS range data, and managing the routine activities of a server such as opening and closing connections.

A number of internal tasks are used by the BIRIS server to interact with the BIRIS system through hardware interrupts and a memory mapped interface. These tasks also use messages to communicate with the BIRIS server. In addition, several message types are used to report to the server the occurrence of fatal system errors. These messages are also outlined in this section.

In the future, when elevation scanning is added, a new message must be defined to control the servoing of the elevation axis and to make the necessary changes to the BIRIS system lookup tables so that the system remains calibrated. Data acquisitions will not occur during elevation changes.

Messages from Client Tasks

REQUEST_POINT

This message requests the latest range and azimuth data from the BIRIS server. In the initial implementation, the data will be composed of a single range sample representing distance to the closest object illuminated by the laser projector and the corresponding azimuth reading. In the next implementation of the server, a full vector of range samples will be returned representing a sequence of spatial data samples observed along the line illuminated by the laser projector.

REQUEST_SCAN_DATA

This message is used by a client to request a full scan of BIRIS data. In the initial implementation, range data are double buffered by the server. As new data are being received by the server, data from the last complete scan are available to any client task that requests them. This introduces a potentially unacceptable delay between the acquisition of data and the availability of that data. This issue will likely be addressed in future versions of the server.

BIRIS_TIMEOUT_ENABLE

This message enables a timeout mechanism that will notify the server if the BIRIS system fails to respond because of a hardware failure. (See BIRIS_TIMEOUT_ERR message.)

BIRIS_TIMEOUT_DISABLE

This message disables the BIRIS timeout mechanism.

BIRIS_MOTOR_PARAMS

The servo motor that scans the BIRIS sensor head is controlled directly by an HCTL-1000 general purpose motion control IC manufactured by Hewlett Packard [9]. A number of parameters must be supplied to the controller before it can operate. These default parameters are normally presented automatically during initialization of the BIRIS server. They include the sample time, the digital filter pole, the digital filter zero, the digital filter gain, the maximum motor velocity, and the maximum motor acceleration. This message to the BIRIS server allows a client task to dynamically adjust the motor control parameters regardless of the default values.

BIRIS_SINGLE_SCAN_RQST

This message is used by a client task to trigger a single scan of the BIRIS sensor. The destination angular position of the scan motor is contained in the message. Note that, when elevation control is added to the system, any change in sensor elevation must be completed before an acquisition scan is executed. This is necessary in part to ensure that the appropriate lookup table has been loaded into the BIRIS system so that the sensor remains calibrated.

BIRIS_CONT_SCAN_RQST

This message from a client task triggers continuous scanning of the BIRIS sensor. The angular limits of the scans are specified in the message. Each sweep of the sensor, either clockwise or counterclockwise, produces a buffer of data. Hence one buffer exists for clockwise sweeps and a second for counterclockwise sweeps. The destination angular positions (CW and CCW) of the scan motor are contained in the message. Note that, when elevation control is added to the system, any change in sensor elevation must be completed before an acquisition scan is executed.

BIRIS_STOP_SCAN_RQST

This message is used to terminate continuous scanning at the completion of the current scan cycle.

BIRIS_HOME

This message from a client task requests that the sensor be driven to the straight-ahead position (0° azimuth). No BIRIS data are acquired during this motion.

OPEN_REQUEST

This message from a client task is a request to open a connection to the BIRIS server. This functionality is typical of all Harmony servers whereby connections can be changed dynamically. In addition to the usual mode of connection, the BIRIS server also supports a special connection type called a "DAT" or data connection. This type is used to notify the server that the requesting task is to be issued data automatically when a buffer is acquired.

CLOSE_REQUEST

This message is used by a client task to close a connection to the server.

READY_FOR_DATA

A task that has opened a DAT connection to the server will send this message type to indicate that it is ready to receive an unsolicited buffer of data from the server. The data will be sent to the task when they become available.

Messages from Internal Tasks

NEW_POINT

The _Biris_data_notifier() task will send this message to the server at the completion of each BIRIS acquisition. In the initial implementation it will include a single range sample and the corresponding azimuth, whereas in the future, it will include a full vector of range data and the azimuth.

CMD_TRANSFER_RDY

This message is used by the _Biris_cmd_transfer() worker task to announce that it is now available to transfer another command to the BIRIS system. It plays only a synchronization role, allowing the task to block and await the next _Reply() from the server. No record is made of the fact that it has reported.

AZ_READY

This message from the _az_scan_start() worker task is sent to announce to the server that the task has become ready to accept the next motor control command. When the server is ready to drive the motor, it "replies" a message to this task containing the next motor destination. The task then activates the motor controller.

AZ_SCAN_DONE

This message from the _Biris_az_notifier() task indicates that the latest motor action has been completed (the MOTOR_END_OF_SCAN interrupt has been received).

BIRIS_CMD_COMPLETE

This message from the _Biris_cmd_notifier() task announces the completion of a BIRIS command (the BIRIS_CMD interrupt has been received).

Messages from Internal Tasks Reporting System Errors

BIRIS_TIMEOUT_ERR

This message from the _Biris_timeout() task indicates that a fatal hardware error in the BIRIS system has occurred.

BIRIS_BUS_ERR

This message from the _Biris_bus_err() notifier task indicates that a bus cycle on the remote BIRIS VME chassis initiated by the bus coupler has failed. It is most likely due to a hardware failure within the bus coupler or the BIRIS system.

BIRIS_PARITY_ERR

This message from the _Biris_parity_err() notifier task indicates that a communication error has occurred on the bus coupler.

Initialization of the BIRIS Server

Creation of the BIRIS server by an application program at runtime is achieved by a Harmony system call, _Create_server(). Internally this results in the creation, in the proper sequence, of the internal tasks that together makeup the server.

Most servers in Harmony make use of a server initialization record that supplies parameters to the server at startup. This is done to simplify the process of adapting the server to different applications since all of the application specific parameters are located in a single compact file. Birisinit.h (Appendix A) contains the definition of the structure and an example of a declaration can be found in Birisinit.c (Appendix B). For the BIRIS server, the initialization record is a structure containing, amongst other things, the global indices of all of the tasks that are internal to the BIRIS server, the name of the bootfile (found on the internal BIRIS RAMdisk) to be used by BIRIS during initialization, a vector containing the default motor parameters, a variable containing the sample rate of the BIRIS system, and the maximum range value beyond which range data will be discarded. Therefore, at the completion of startup the server and all of its associated hardware will be initialized appropriately.

Summary

This report has described the initial development of a server which was built to integrate the BIRIS system into a Harmony runtime environment being used on an experimental mobile robot. While some improvements have already been suggested in the report, experimentation with the complete robot system will no doubt identify others. Depending on results obtained and interest from other parties, a new version of the BIRIS system offering higher density packaging, lower power requirements, and better performance may be developed as a result of these efforts.

References

1. R. Liscano, D. Green, A. Manz, L. Korba, and S. Lang. The NRC mobile robot project. *Proceedings of the Third International Workshop on Military Robotic Applications*, Medicine Hat, Alta. Sept. 9–12, 1991. NRC 31837.

2. F. Blais, M. Rioux, and J. Domey. Compact three dimensional camera for robot and vehicle guidance. *Opt. Lasers Eng.* 10, 227–229, 1989.

3. D. Green, R. Liscano, and M. Wein. Real-time control of an autonomous mobile robot using the Harmony operating system. *Proceedings of the Fourth IEEE International Symposium on Intelligent Control*, Albany, N.Y. Sept. 24–26, 1989. pp. 374–378. NRC 30554.

4. R. Liscano and D. Green. "REDY", An architecture for a sensor-based mobile platform. *Proceedings of the Canadian Conference on Electrical and Computer Engineering.* Ottawa, Ont. Sept. 3–6, 1990. pp. 58-1-1 to 58-1-7. NRC 31746.

5. W.M. Gentleman, S.A. MacKay, D.A. Stewart, and M. Wein. *Using the Harmony Operating System: Release 3.0.* NRC/ERA-377. National Research Council of Canada, Ottawa, Ont. February 1989. NRC 30081.

6. A. Manz, R. Liscano, and D. Green. A comparison of realtime obstacle avoidance methods for mobile robots. *Lecture Notes in Control and Information Science Series*, Springer Verlag. Experimental Robotics II Symposium, Toulouse, France. June 25–27, 1991. In press. NRC 31826.

7. *A32/D32 VME–VME Adaptor Model 412*. Bit 3 Computer Corp., Minneapolis, MN, Rev. 1.1, 8/89. Pub. No. 82902216.

8. *Model 7600 VMEbus Motion Controller User's Manual*. Technology 80 Inc., Minneapolis, MN. January 1990. Doc. No. 7600UM2.

9. *HCTL-1000 General Purpose Motion Control IC*. Hewlett Packard, Palo Alto, CA. Doc. No. 5954-8471.

Appendix A. The Structure Definition of the Server Initialization Record

```
struct BIRIS_INIT_REC
{
    struct INIT_REC      BIRIS_HDR;
    char                 BIRIS_NAME[32];
    uint_32   BIRIS_CMD_NOTIFIER_INDEX;      /* global index of cmd notifier task */
    uint_32   BIRIS_CMD_TRANSFER_INDEX;      /* global index of cmd transfer task */
    uint_32   BIRIS_DATA_NOTIFIER_INDEX;     /* global index of data notifier task */
    uint_32   BIRIS_AZ_NOTIFIER_INDEX;       /* global index of az notifier task */
    uint_32   AZ_SCAN_START_INDEX;           /* global index of az scan start task */
    uint_32   BIRIS_TIMEOUT_INDEX;           /* global index of timeout task */
    uint_32   BIRIS_BUS_ERR_INDEX;           /* global index of bus error task */
    uint_32   BIRIS_PAR_ERR_INDEX;           /* global index of parity error task */
    char      BOOT_FILE[32];                 /* BIRIS boot filename */
    struct MOTOR_PARAM  MOTOR;               /* motor operating parameters */
    int_16   BIRIS_DATA_RATE;                /* BIRIS sample rate ( ms ) */
    uint_16  BIRIS_MAX_RANGE;                /* range data exceeding this limit are discarded */
};
#define I_BIRIS_INIT_REC 1
```

Appendix B. An Example of the Server Initialization Record

```
struct BIRIS_INIT_REC bs_init =
{
    sizeof(struct BIRIS_INIT_REC),          /* size */
    I_BIRIS_INIT_REC,                       /* MSG_TYPE */
    0,                                      /* next I_INIT_REC */
    "BIRISSERV:",
    BIRIS_CMD_NOTIFIER,                     /* global indices */
    BIRIS_CMD_TRANSFER,
    BIRIS_DATA_NOTIFIER,
    BIRIS_AZ_NOTIFIER,
    AZ_SCAN_START,
    BIRIS_TIMEOUT,
    BIRIS_PAR_ERROR,
    BIRIS_BUS_ERROR,
    "/dd/users/biris/biniticl.par",         /* BIRIS boot filename */
    {0xAF, 0x7F, 0xEF, 0x3F, 0x2, 0x3322 }, /* motor operating parameters */
                                            /* sample, zero, pole, gain, max vel., max accel. */
    30,                                     /* Biris acquisition rate in ms */
    0x7FFF,                                 /* Biris max range limit - tbd */
};
```

Appendix C. BIRIS Server Design Details

Drive Mechanism and Scan Control

The BIRIS system uses a DC servo motor to pan the CCD camera. A commercial VMEbus motor controller manufactured by Technology 80 and based on the HP HCTL-1000 controller chip is used. A 1000 line incremental position encoder is used in the ×4 mode yielding a count resolution of 4096 counts per revolution (0.088°/count).

ALL MOTIONS ARE CONSTRAINED TO LIE WITHIN ±360° (±0×1000).

This is implemented by the function _Scan_limit(), which truncates all motor destinations presented to the motor controller. The maximum motion in either direction is two full rotations: −359° to 0 to +359°.

Representation

deg	binary degrees	hex
0	0	0 (dead ahead)
45	512	0x200
90	1024	0x400
180	2048	0x800
360	4096	0x1000

Unlike the Slit scanner where the camera was mechanically scanned continuously and where direction had to be computed from the azimuth data, each BIRIS scan is initiated in software. An interrupt is used to announce the completion of a scan. Two interrupts occur: MOTOR_END_OF_SCAN from the motor controller and BIRIS_CMD, which occurs at the completion of a BIRIS acquisition sequence. For continuous scanning operation, both are used to determine when the subsequent scan can be started.

Driving to the HOME Position using the BIRIS_HOME Case

BIRIS_HOME does not force an immediate motor motion unless the motor is stationary and no motion requests are pending. Hence, if a request is issued during a single scan, it remains pending until that scan is complete. To drive to the HOME position during a continuous scan operation, the BIRIS_STOP_SCAN_RQST must be issued first to terminate the scanning, then the BIRIS_HOME request can be issued.

The reason for this has to do with the operation of the HCTL-1000 motion control IC. If a BIRIS_HOME request is issued during a normal motion, the motor will switch instantaneously to the Position Control Mode (i.e., without velocity control) and will drive at high speed to the home position. This is undesirable. The design of this server ensures that the motor will drive to the HOME position using the Trapezoidal Profile Control mode that always limits the motor velocity to the specified value (the default motor maximum velocity is specified in the server initialization record).

Command Transfers from the Server to BIRIS

The following describes how commands are transferred to BIRIS. After a command is issued, BIRIS remains busy until command completion. Completion is announced by a BIRIS_CMD interrupt which occurs after the command element of the BIEXEC_COM structure is cleared. The _Biris_cmd_notifier task is used to await the command interrupt and notify the server so that the next command can be issued. A variable delay of from tens of microseconds to tens of milliseconds occurs between command presentation and command completion. This is due in part to the fact that commands are presented to BIRIS asynchronously and are synchronized internally to the vertical sync rate. To accommodate this, a separate task is used to transfer a command to BIRIS. The _Biris_cmd_transfer() task is a typical worker task that reports to the BIRIS server to indicate that it is ready to transfer a command to BIRIS. Typical operation then involves

- replying to the _Biris_cmd_notifier() task so that it will be blocked on an _Await_interrupt() before the next command is issued
- replying to the _Biris_cmd_transfer() task so that the actual command is issued

Transfer of Data from BIRIS to the Server

This server acquires and buffers BIRIS data. Data arrive at half the vertical sync rate of the video camera (1/30 s) and are announced by an interrupt. The _BIRIS_data_notifier() task awaits the interrupts and sends the current data to this server. The data include the current azimuth referenced to the turret of the vehicle and a calibrated range reading. Note that, unlike the slit scanner with its wide field of view, no "heading" calculation is required with BIRIS.

Synchronization of the Number of BIRIS Data Acquisitions with the Scan Length

The number of BIRIS acquisitions for each scan is adjusted automatically to match the given scan length. The function _Get_acq_rate() computes the constant of proportionality from the motor setup parameters and from the BIRIS sample rate. These values are contained in the initialization record. _Get_acq_rate() returns a value in units of encoder counts/acquisition. Dividing a given scan length (encoder counts) by this constant yields the number of BIRIS acquisitions for that scan. (Since the constant is less than 1.0, the reciprocal is used to avoid floats.)

Appendix D. Proposed Changes

Continuous Operation

The next generation BIRIS system may incorporate a continuous mode of operation. This would eliminate the 240 ms latency (startup delay) currently encountered with each new BIRIS acquisition. Typical operation would require sending the server a START_CONTINUOUS message which would produce a continuous sequence of acquisitions. Each acquisition would have an associated interrupt. Termination of this mode would be provided by a STOP_CONTINUOUS message that would cease all acquisitions immediately. The BIEXEC_COM structure would be used to point to the most recent acquisition.

Implications

Under continuous operation, data interrupts must be disabled at the end of a motor scan and re-enabled after the next scan is started. Even so, multiple samples at constant azimuth would still occur. A new flag is required to identify the continuous mode (a further description appears in the AZ_SCAN_DONE case in the Birisserver.c source code). Single scans can also benefit from this mode since the BIRIS latency is eliminated. Changes will have to be made to the BIRIS_SINGLE_SCAN_RQST and BIRIS_CONT_SCAN_RQST cases. In both cases, Biris_continuous_mode will also be used to determine if a scan can be issued immediately since Biris_ready will be FALSE in the continuous case.