IN-63-CR
243098
p-79

(NASA-CR-185934) ROBOTIC SPACE SIMULATION INTEGRATION OF VISION ALGORITHMS INTO AN ORBITAL OPERATIONS SIMULATION (Houston Univ.) 79 p CSCL 09B

N90-27395

Unclas
G3/63 0243098

Robotic Space Simulation

Integration of Vision Algorithms into an Orbital Operations Simulation

Daniel C. Bochsler

LinCom Corporation

October 30, 1987

Cooperative Agreement NCC 9-16
Research Activity No. AI.6

Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

Robotic Space Simulation

Integration of Vision Algorithms into an Orbital Operations Simulation

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by LinCom Corporation under the supervision of Daniel C. Bocshsler. Joseph Giarrantano, Associate Professor of Computer Science and Information Systems at the University of Houston - Clear Lake, was the RICIS technical representative.

Funding has been provided by the Mission Planning and Analysis Division, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston - Clear Lake. The NASA Technical Monitor for this activity was Timothy Cleghorn, Mission Planning and Analysis Division, NASA/JSC.

ROBOTIC SPACE SIMULATION

INTEGRATION OF VISION ALGORITHMS INTO AN ORBITAL OPERATIONS SIMULATION

DOCUMENTATION PRODUCED DURING CURRENT WORK EFFORT

LinCom CORPORATION
30 OCTOBER 1987

May 29, 1987

INTERNAL MEMORANDOM

29 May 1987

To: Dr. Jani

From: Bill Othon

Re: Implementation of Vision Sensor algorithms received from Rice University

---

I have looked over the first package of C-language routines received from Rice. The package includes 12 routines involved in wireframe construction and identification. Also, there is code which defines two wireframe objects, a ball and a box, and data files associated with the C files.

A driver routine was included in the files to run many, but not all, the 'vision' routines. The driver called the following routines:

1) transfor.c: rotates and translates a given wireframe object. The rotation and translation parameters are user-input, and the object is defined in `database.c'. The input object is changed, but the original orientation is still available in 'database.c'.

2) wiredraw.c: draws a 2D image of a given 3D wireframe object on a tektronics-like terminal. Hidden faces are identified by evaluating the y-component (perpendicular to image plane) of the points making up a given face. If the y-component is larger than a certain set value, the face associated with the point is defined as not visible.

3) wiregnr.c : converts a 3D wireframe object into a 2D image which is a projection of the original (rotated) object. Again, non-visible faces are determined and not included in the image. The total number of visible faces and edges is determined, and edge connectivity among the faces is defined (ie. if two faces share an edge, variable = 1, else variable = 0).

4) obinit.c : calculates the relationship between the faces of the original rotated object. Calculates angles between the normals of the faces, and the moments, invariants, and tensors of each face. This information is stored in the object data structure.

5) owmatch.c : conducts the actual comparision between the predefined 3D object in the database and the 2D rotated image of the object. For each face on the object, the moment invariant of the object face is compared to the moment invariant of a given face of the 2D image. If the invariants are sufficiently close (arbitrary), the object and image faces are input to the 'grow' subroutine, the heart of the comparison process. 'grow' is a recursive routine which tries to match the input faces (or root faces) with their surrounding, adjecent faces. In other words, if both the object root face and image root face are surrounded by the same faces (as determined by the algorithm), the image is assumed to be a subgraph of the object. The output of the subroutine is just a (match/no match). No orientation identification algorithms have been identified. It may be possible to reverse the role of the transform subroutine, so instead of using user-input parameters to get a transformed object, these parameters may be backed out from an image. More work is being conducted to fully understand the comparision procedure.

These programs represent all the algorithms used by the driver routine. It seems that the purpose of this driver is to varify the ability of the comparision algorithm to successfully identify a known object.

Two other programs were included in the software package. `ipc.c' and 'gipc.c' are the image point coorespondance routines, defined by a paper authored by Rice's Sunil Fotedar and Dr. Rue de Figueiredo. These routines are used in the determination of motion parameters of a moving object from moving camera data. The documentation identifies a number of case options and associated algorithms involved in the operation of these codes, but these files were not included in the package from Rice.

What I plan to do in the short term is remove the wireframe building routines from the driver code and run the program without graphics, to see if it works. Actually, the graphics associated with the driver are for display purposes only, and input no information to the driver routine.

Apparently, we still need to receive algorithms which can read a 2D image (from graphics) and can convert the image into a form which can be used to compare it to models in the object library. Also, we need to find out which algorithms should be used to calculate the orientation of an object, once the associated image has been identified.

Attached to this memo is information on each of the routines delivered by Rice.

DEFINITION OF RICE FILES FOR VISION SENSOR

****

'he object files all routines shown here (except for driver routines) ire kept in an archive library /mnt/bamieh/cleanproj/lib. This iame should be used in the with the cc comand. ill routines were compiled with the -g option for debuging.

****

---

amieh.h : This file contains all the headers used. It contains type definitions, and also stdio and math.h. In this directory it is used by #include "bamieh.h".

---

atabase.c: The database which contains the initial object discritptions, New objects are added here. An object is declared as an extern polyhedron variable, and any programs that use the object should be compiled and linked to the database.

-ansform.c: contains the routine

```
polyhedron  transform(dx,dz,theta,phi,psi,object)
float dx,dz,theta,phi,psi;
polyhedron object;
```

which rotates and translates a polyhedron "object" in 3-space by transforming the coordinates in the array vert[], the other parameters of a polyhedron are invariant.

dx,dz : translation in the x and z axes respectively.

theta,phi: spherical coordinates of the axis of rotation theta is the angle in the x,y plane.

psi : the magnitude of the counterclockwise rotation about the axis.

object : the structure containing the polyhedron.

all angles should be in degrees. transform returns a polyhedron structure which is the rotated object.

-aw.c: Contains the function

ORIGINAL PAGE IS OF POOR QUALITY

```
hdraw(hoffset,voffset,wndsize,object)
int hoffset,voffset,wndsize;
polyhedron object;
```

which draws a polyhedron "object" on a tektronix like terminal, assuming orthogonal projection on the

... from the negative
y axis at the x,z image plane.
The screen origin is at the lower left-hand corner, the vertical and horizontal scales are 799 and 1024 respectively. The object is drawn in a square window specified by the following parameters:

hoffset,voffset: the horizontal and vertical coordinates of the window origin, respectively.

wndsize : the length of the window side.

This routine does hidden line removal for a convex object.

---

draw.c: Contains

```
draw.c(hoffset,voffset,wndsize,object)
```

Same as hdraw but does not do hidden lines.

---

polymnts.c Contains

```
polygon2  polymnts(face)
polygon2 face;
```

which calculates the moments of a 2D polygon "face" and returns the same 2D polygon but with the moments entries appropriatly filled. The highest order of moments calculated is determined by MORDER in the file bamieh.h, this constant also effects the type defenitions.

---

invariants.c: Contains

```
polygon2  invariants(face)
polygon2 face;
```

which calculates the invariants given the moments which should already be in face. It returns the original 2D polygon face with the invariants in their proper places.

---

tensor.c Contains

```
float  Ttensor(face,index);
polygon2 face; int index;

float  Vtensor(face,index);
polygon2 face; int index;
```

ORIGINAL PAGE IS
OF POOR QUALITY

which calculates the tensors T and V (see paper). Index specifies which component of the tensor is to be calculated, either 1 or 2.
These tensors are calculated in the most brute force way imaginable, if they become a bottleneck, they probably can be improved substantialy.

winit.c: Contains

```
wireframe  winit(immap)        /* wireframe initializer */
wireframe immap;
```

Initializes wireframe by symmetrizinn the edges matrix. It also adds the moments invariants and tensors of every face in the wireframe. The original wireframe plus every thing added is returned.

---

obinit.c Contains

```
polyhedron  obinit(object);      /* object initializer */
polyhedron object;
```

Intitializes the object by symmetrizing the edges matrix and adding the moments, invariants, and tensors to every face. These last there are computed in the plane in which each face lies.
The initialized object is returned.

---

iregnr: Contains

```
wireframe  wiregnr(theta,phi,psi,object)
float theta,phi,psi;
polyhedron object;
```

which genterates a wireframe of a polyhedron "object" viewd with a rotation of theta,phi,psi. The function returns the wireframe it generates. To find the visible faces it basically uses a code similar to that of hdraw, except that faces that are only very slightly visible (i.e. almost perpendicular to the image plane) are not included in the wireframe.

---

redraw.c: Contains

```
wiredraw(theta,phi,psi,object)
float theta,phi,psi;
polyhedron object;
```

This is similar to hdraw except that it draws the object exactly as the wireframe generator "wiregnr" sees it, i.e. faces that are only slightly visible are not included. The arguments are the same as hdraw.

---

match.c: Contains "object to wireframe matching"

```
correspondence  owmatch(object,immap)
polyhedron object;
wireframe immap;
```


which looks for a possible match between the object and a wireframe. The results of the match is returned as a correspondence struct.

mtest.c   A driver routine. Self explanatory.

ORIGINAL PAGE IS
OF POOR QUALITY

June 17, 1987

Simulation of Robotics Space Operations

Principal Investigator: Yashvant Jani
LinCom Corporation

Computer-based simulations of activities in low earth orbit play a vital role in the research and development of space missions, especially generation scenarios. In order to successfully plan and analyst future space activities, these simulations will be required to model and integrate vision and robotics operations with vehicle dynamics, and proximity operations procedures. The basic objective of this project is to configure and enhance the orbital operations simulation (OOS) as a testbed for robotics space operations.

The vision sensor is comprised of many subsystems, which will; 1) Detect the presence of orbiting space vehicles, using camera data, 2) Identify an unknown vehicle being scanned by a camera, 3) Identify the position, attitude, and rates of a scanned object, and 4) Track a vehicle along its flight path.

Each of these capabilities could be used for a wide range of orbital operations, including proximity operations of vehicles, traffic control, and collision avoidance. Additionally, the vision sensor when integrated with robotics, would allow robotics-enhanced, free-flying vehicles, like the Orbital Maneuvering Vehicle (OMV), to conduct autonomous missions including vehicle repair and retrieval. By using the vision sensor, autonomous vehicles could identify desired targets, track their motion and attitude, and dock with the target. The vision sensor could also identify damage, and provide visual data required for work with Remote Manipulatc System (RMS).

The vision sensor will be mathmatically modeled, and included in a general orbital operations simulator. * By coupling the vision sensor with vehicle dynamics and the orbital environment, the simulation will be used as a test-bed for the development, and optimization of vision-related operations procedures. The simulation can use a number of different orbital vehicles during testing of vision techniques, including shuttle, OMV, and eventually Space Station. Topics that can be explored through simulation include range requirements, resolution constraints, data extraction and analysis techniques, and integration of vehicle flight software and vision-derived environment and tracking information.

*The vision data processing algorithm will be implemented as a flight software which can be scheduled according to the processing requirements.

SIMULATION OF ROBOTICS SPACE OPERATIONS

LinCom

AGENDA

- OBJECTIVES
- ARCHITECTURE OF OOS
- VISION SENSOR SIMULATION
- VISION PROCESSING ALGORITHM
- CURRENT STATUS
- FUTURE PLANS

LinCom

OBJECTIVES

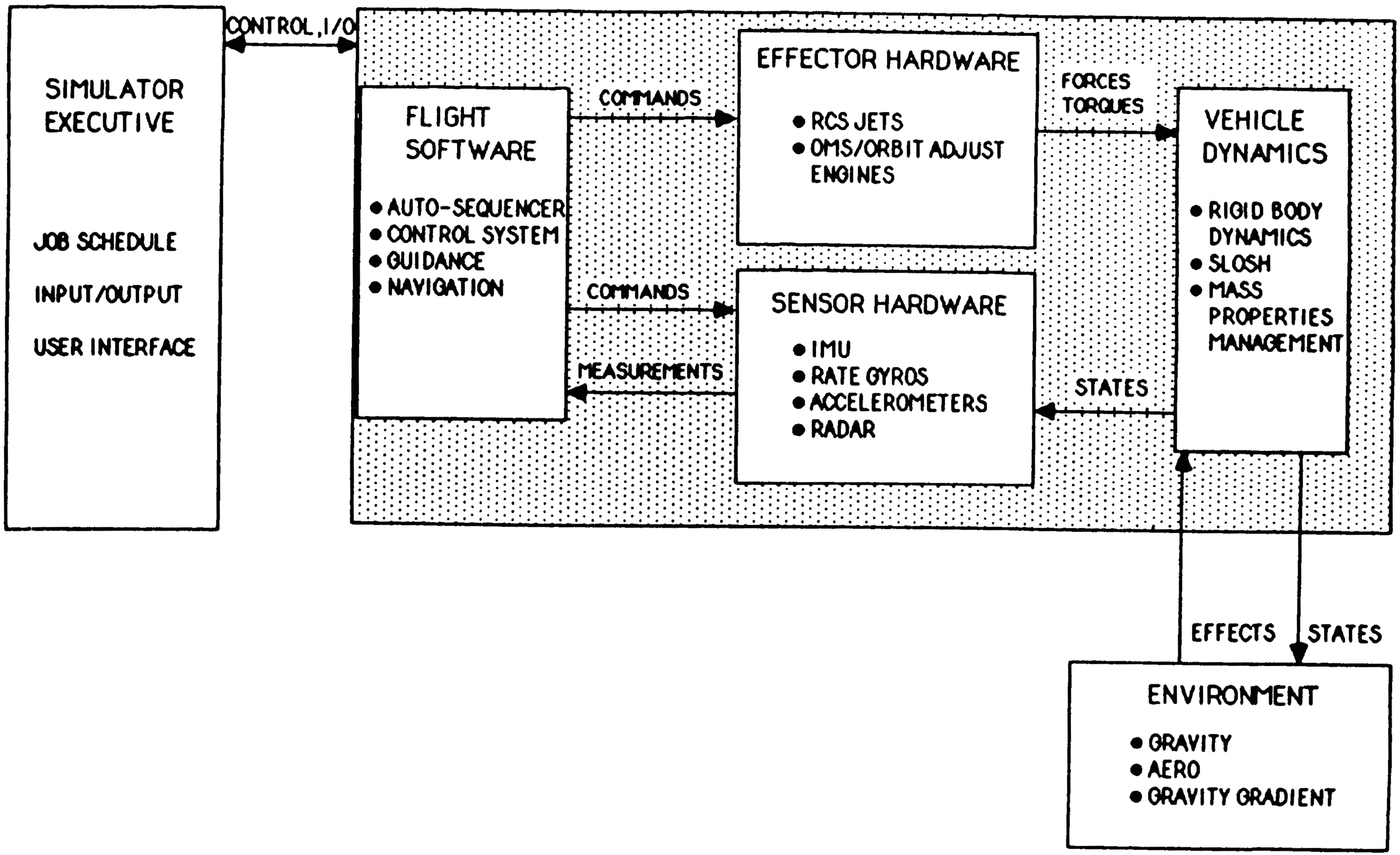
- CONFIGURE AND ENHANCE OOS AS A TESTBED FOR ROBOTICS SPACE OPERATIONS
- START WITH A VISION SENSOR, RELATED VISION ALGORITHMS, AND INFORMATION PROCESSING SOFTWARE DEVELOPED AT RICE UNIVERSITY
- IMPLEMENT THESE ALGORITHMS IN OOS
- VALIDATE AND DEMONSTRATE WITH GRAPHICS SYSTEM

LinCom

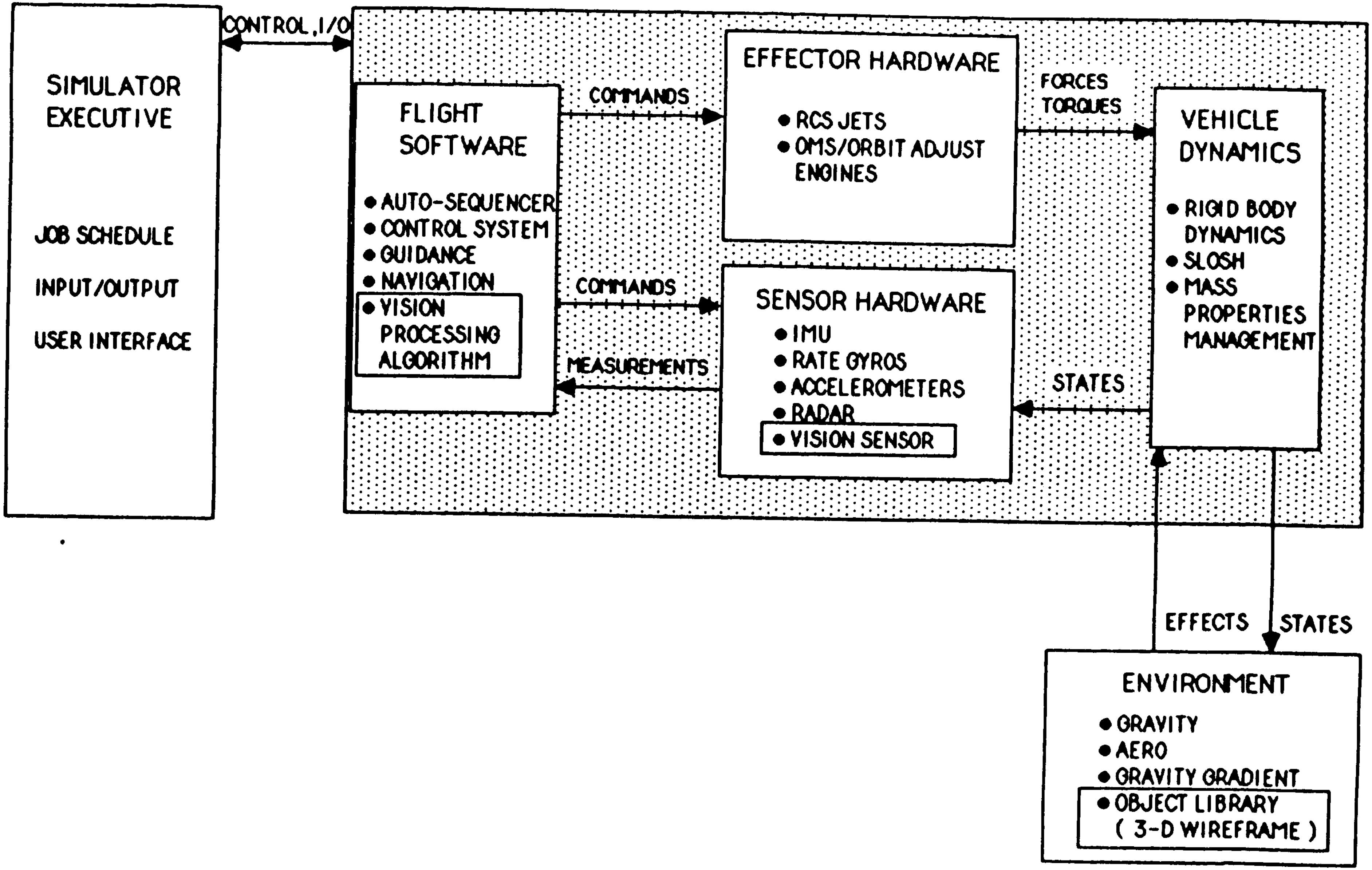
ARCHITECTURE OF OOS

- GENERAL ARCHITECTURE
- VISION SENSOR IMPLEMENTATION

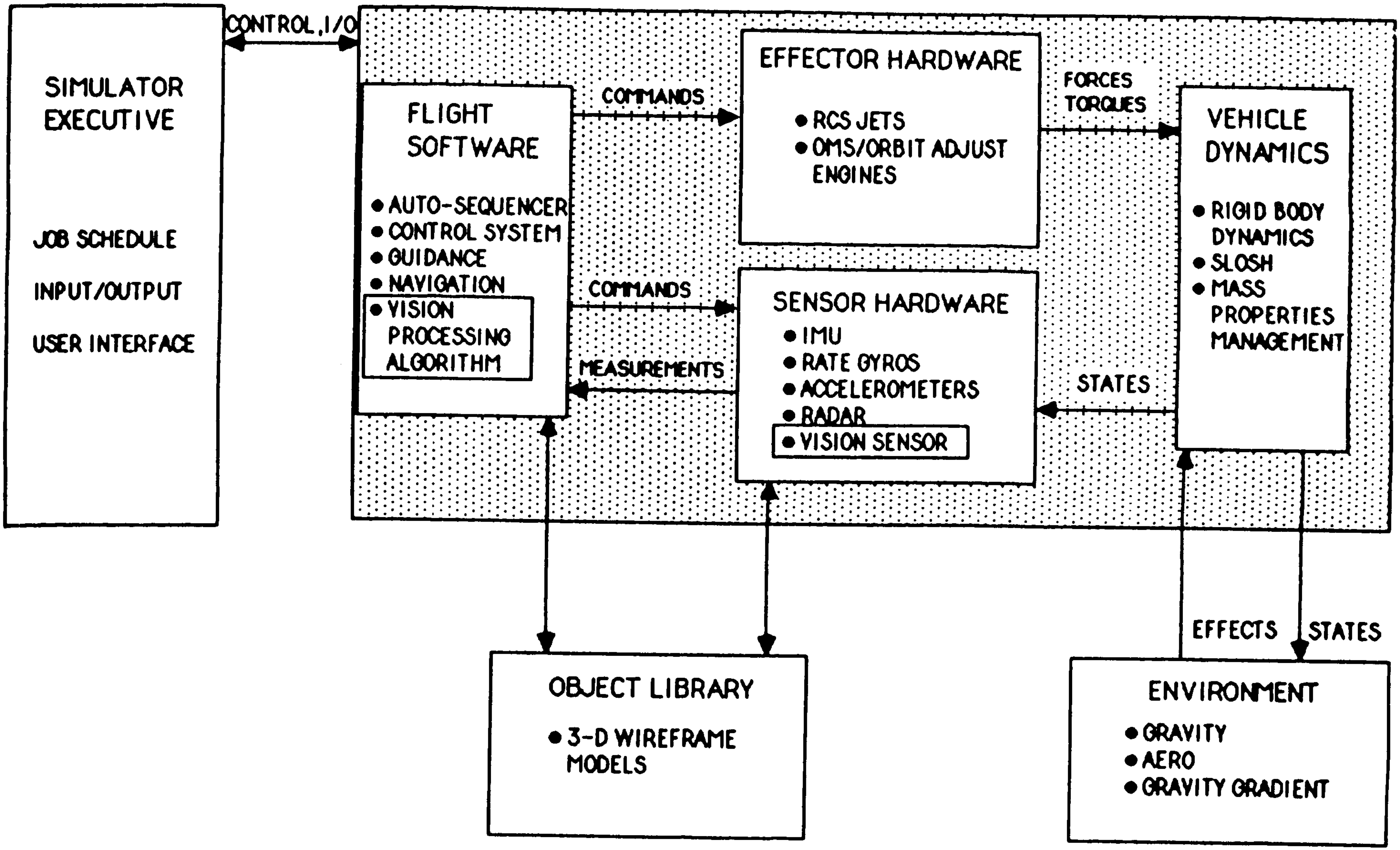
LinCom



ORBITAL OPERATIONS SIMULATOR/OMV IMPLEMENTATION

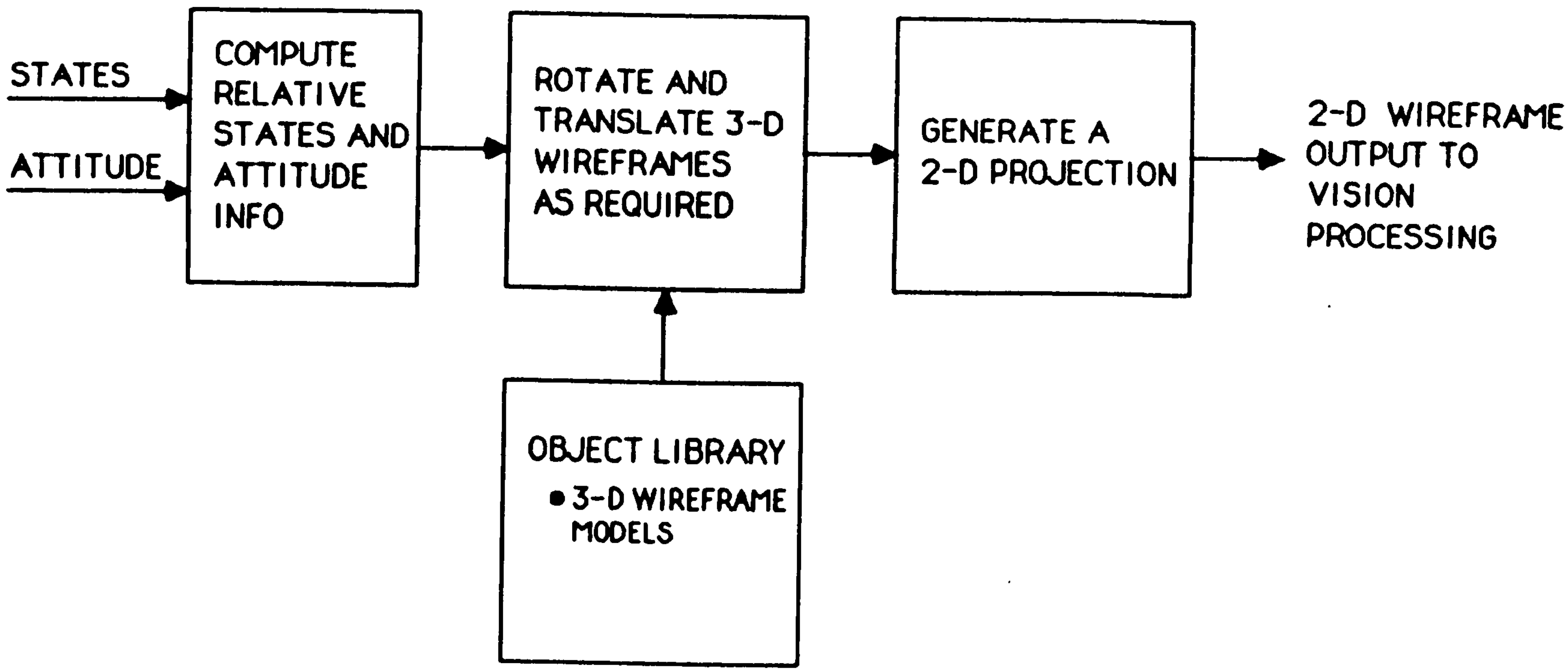


ORBITAL OPERATIONS SIMULATOR/OMV IMPLEMENTATION



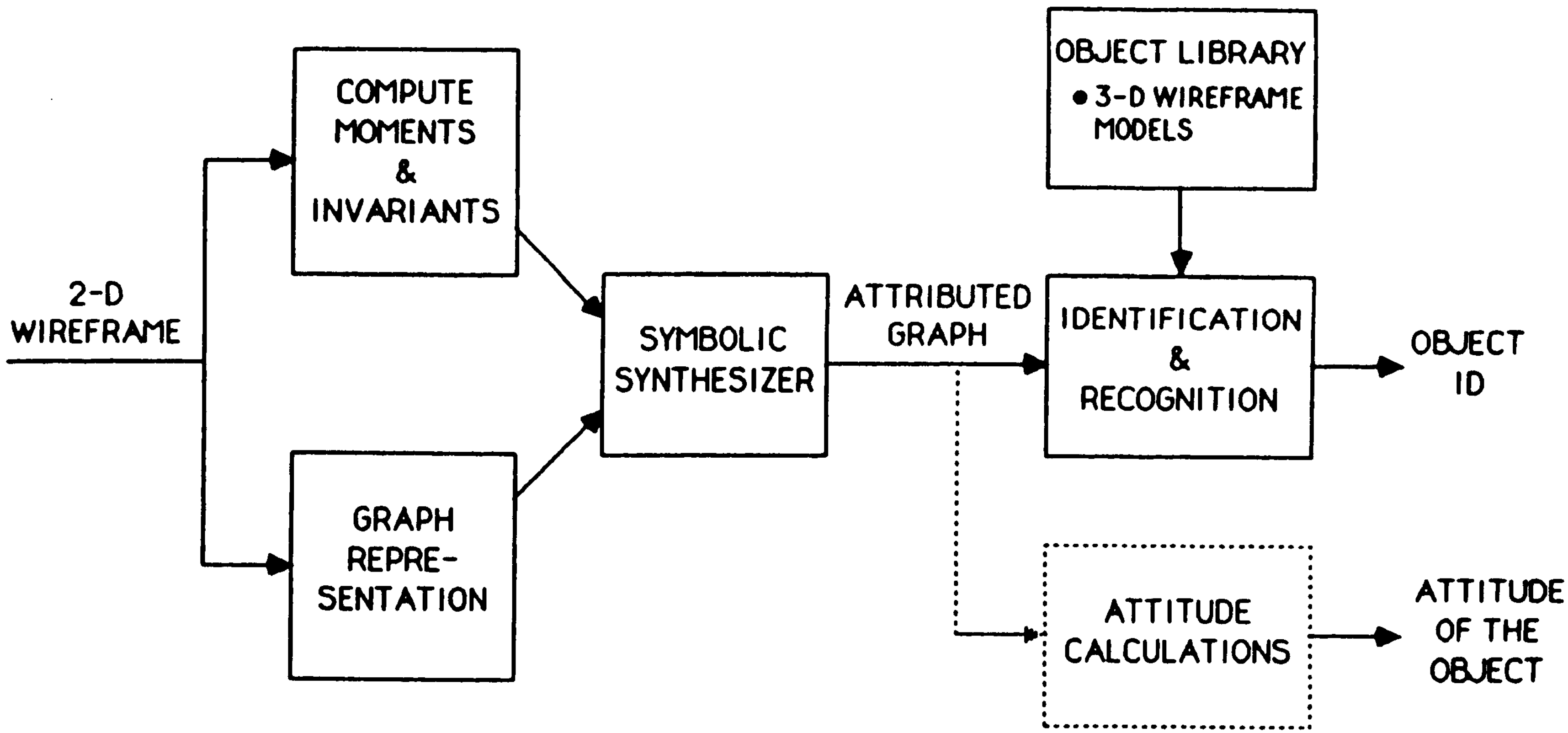
ORBITAL OPERATIONS SIMULATOR/OMV IMPLEMENTATION

SIMULATION OF VISION SENSOR



LinCom

VISION PROCESSING ALGORITHMS



LinCom

LinCom

CURRENT STATUS

- TECHNICAL INFORMATION EXCHANGE & COORDINATION WITH RICE HAS BEEN EXCELLENT
- REQUIREMENTS ANALYSIS FOR SIMULATION NEAR COMPLETION
- DESIGN AND IMPLEMENTATION IN OOS IS CONTINUING

LinCom

FUTURE PLANS

- IMPLEMENT SENSOR & PROCESSING ALGORITHMS
- CREATE THE OBJECT LIBRARY (3-D WIREFRAME DATABASES) FOR THE OOS
- VALIDATE THE IMPLEMENTATION FOR OBJECT IDENTIFICATION
- DEMONSTRATE THE CAPABILITY

END OF PRESENTATION

LinCom

August 1, 1987

INTERNAL
MEMORANDOM

1 August 1987

To: Dr. Jani

From: Bill Othon

Re: Status Update of Implementation of Vision Sensor algorithms

---

While I was on leave, the complete code for Rice's GIPC (motion parameter determination) and MIAG (object identification) algorithms were delivered to LinCom. The GIPC code includes a small menu and algorithms for all methods of motion determination outlined in the reference document. The MIAG also includes a menu for user input. A number of object models (with vertex, face, and edge information) were included for testing.

These codes are currently being examined and evaluated for modifications which would be necessary before inclusion into OOS. David and I intend to get the stand-alone versions of these codes running, and to test output from the code with available reference material. Thus, we can be sure the code is running correctly before integration with OOS.

One small note: Apparantly, the computer hardware at Rice has different capabilities and resources than the HP9000 at LinCom. Consequently, the two algorithms are not running smoothly at this time. Some of the arrays in the GIPC routine are dimensioned arbitrarily large, and may be overwriting memory. This problem is currently being examined.

August 11, 1987

SIMULATION OF ROBOTICS SPACE OPERATIONS

STATUS OF INTEGRATION OF VISION ALGORITHMS INTO OOS

AUGUST 11,1987

MEETING WITH VISHAL MARKANDEY, RICE UNIVERSITY- 7/30/87

Dr. Yashvant Jani and William Othon met with Vishal Markandey of Rice University on 7/30/87. Vishal brought with him graphics algorithms for extracting wireframe and vertex information from the image of an object, produced by a camera. These routines were not fully completed, and development and modification of these algorithms continues at Rice. However, LinCom will begin analysis of the routines for future integration into OOS.

A copy of a single value decomposition (SVD) routine was given to Vishal at the meeting. David Myrick translated this SVD routine from FORTRAN listing to 'C' code. The translation was necessary because the routine was not available to LinCom, and it was part of a prepackaged math library at Rice which could not be transferred. The routine is used in the Generalized Image Point Correspondence (GIPC) algorithm, which extracts motion parameters of a moving object from information provided by a moving camera.

THREE MAIN AREAS OF VISION ALGORITHM DEVELOPMENT

Vishal described the three main areas of vision algorithm development going on at Rice. The three main areas are: 1) preprocessing of raw camera (pixel) data, 2) object recognition from preprocessed data, and 3) determination of the attitude and attitude rates of a observed object (figure 1).

1) Preprocessing

Preprocessing algorithms transform the raw camera pixel data of a scanned object into a graphical representation of the object. This representation can then be used by other vision algorithms to identify the object and define its position and attitude. There are several elements to the preprocessing phase:

NOISE REMOVAL- Every frame taken from a camera in pixel format will have noise which is unassociated with the object being scanned. This noise can be filtered out based on the abruptness of the change in "gray-level", or intensity, of the pixels. If this gray-level change is sufficiently abrupt from one pixel to the next, and there is no continuity in intensity, the pixel is defined as noise and filtered out. If the intensity from one pixel to the next is continuous, or changing gradually, the pixels are assumed to be part of the pictured object. A gaussian filtering routine is used to remove the high frequency noise (i.e. abrupt, non-continuous change in pixel intensity).

REGION GROWING- Some of the characteristics of an object, such as writing, emblems, or windows, are interpreted as polygons by the vision algorithms. These polygons appear as dark regions inside lighter areas. To prevent these polygons from being identified as faces, a region growing technique is used. Region growing increases white areas of the image, but the shape of the region is maintained. As the routine continues through more iterations, the shape of the image becomes more uniform. Eventually, windows and writing are erased from the image. The transformed image is larger, but the shape is maintained so that identification and attitude can still be determined.

EDGE DETECTION- Edge detection involves the building of a 2D wireframe based on the image of the sighted object. This building can be done using vertex detection schemes (after identification of straight lines) or contour following (to define object faces).

Both schemes use changes in gray-level to define lines or faces. After edge detection, the image is transformed into a wireframe image, with two levels of intensity: background and the lines of the object.

GRAPH BUILDING- This is the final step in preprocessing. The various faces and vertices of the wireframe image are defined and stored in a GRAPH STRUCTURE. The moment invariants of the identified faces are then calculated. Together, the graph structure and the associated moment invariant information are known as the ATTRIBUTED GRAPH.

2) RECOGNITION

After defining the various components and invariants of the wireframe image (whether through simulation or real camera data), the image can be identified with an object in the object library. The Moment Invariant/Attributed Graph algorithm (MIAG) matches the moment invariants of the wireframe object with those of a specific object in the object library. Since these moment invariants remain constant for a given polygon, regardless of rotations, wireframe polygons and object faces can be compared and possible matches identified. Once the possible matches are found, then the relationships between the "root" face and adjacent object faces and the "root" polygon and adjacent wireframe polygons are checked. If all adjacent faces and polygons match, the wireframe image is defined as a subgraph of the object, and therefore identified. Otherwise, a new object from the library can be tested or the matching process stopped. (see figure 2)

Currently, only polygonal shapes can be identified. A future extension to the MIAG should allow for edges of various shapes: circular, elliptic, etc. To do this, the graph structure must have information about the connectivity of faces, and information about the contours of connected faces.

3) ATTITUDE/ATTITUDE RATES

Currently, two algorithms are under development for determination of attitude and attitude rates based on processed camera data. These algorithms are based on different schemes, and no comparison of efficiency, accuracy, or speed has yet been made.

The MIAG algorithm calculates tensors based on polygon geometry. These tensors can be used to calculate the change in attitude between an identified camera image and an object in the library (at some reference attitude). Also, an estimate of the translation vector can also be determined. For information about the algorithm, refer to "General Moment Invariants and Their Application to 3D Object Recognition from a Single Image" by B. Bamieh and Prof. Rui de Figueiredo.

The second method of attitude extraction under development is called Generalized Image Point Correspondence (GIPC). The algorithm determines the rotation and translation (to a scale factor) of a moving object in some reference frame, from data provided from a moving camera. The routine requires: 1) 8 or more unique points defined on the object, before and after motion, 2) the transformation between the two image coordinate frames, and 3) the transformation between the original image coordinate frame and the reference frame. This method is explained fully in the reference "Determination of Motion Parameters of a Moving Object From Moving Camera Data" by S. Fotedar and Prof. Rui de Figueiredo.

CAMERA MODEL INTEGRATION IN OOS

A software model of an OMV-based camera is being developed at LinCom. This model will simulate the sensing capabilities and hardware constraints of a camera. Characteristics of the camera model will include range, field of view, and focal length. Additionally, the camera will be integrated with a target-tracking algorithm, to define the motion of a camera with two gimbals (pitch and yaw).

The camera model will be used in simulations where the two-dimensional image data is simulated and not derived from actual camera input. The translations and rotations of the target (i.e. camera image) will be fed directly from the dynamics routines to a transformation routine. This routine will transform a library model of the target (at some reference orientation) and define the points which are visible on the 2D camera image. These data can then be fed to the other vision algorithms for object identification and determination of motion parameters.

CURRENT STATUS OF VISION INTEGRATION WORK AT LinCom

* The SVD routine has been translated from the FORTRAN listing to 'C' code. The output was validated. An interface was created between the new SVD and OOS-compatible code to be used with GIPC algorithm.

* GIPC and MIAG codes received from Rice are currently being tested. Comparison checks are being run between reference document data and output from LinCom code.

* The GIPC 'C' code is being modified to match OOS code conventions.

* Integration of validated GIPC and MIAG into vision subsystem structures in OOS is being developed. Also being developing is a camera model, with hardware restrictions (i.e. range, viewing cone, etc.) and target-tracking ability.

* The preprocessing algorithms delivered by Rice (7/30) will be analyzed. Future plans include integration of these vision algorithms (in OOS) with graphics for data retrieval and visual depiction of vision techniques.

August 27, 1987

Preparation of Vision for Integration Into the OOS

William David Myrick

LinCom Corporation

August 27, 1987

Table of Contents

1. Introduction

The Vision program supplied to LinCom by Rice University has been modified for use in the Orbital Operations Simulator. This required LinCom to add a "singular value decomposition" (svd) routine to its library. Also, several changes were made to the Vision program itself (see Reference 1). The orginal program invoked LINPACK routines. The program was changed to used LinCom's linear algebra routines. To allow use of the OOS's logging functions and other routines, the program's array structure was changed although it was kept conceptually the same. The end result is a cleaner program that is OOS compatible.

2. Singular Value Decomposition Algorithm

The Vision algorithm requires a singular value decomposition routine. Rice invoked a LINPACK routine which is part of their computer library. Unfortunately, LinCom's library was lacking in an svd routine. A LINPACK svd routine written in FORTRAN was used as a basis for LinCom's svd. This required that the program be converted from FORTRAN to C. This conversion proved to be quite tedious due to the unstructured nature of the original program. Some of the problems encountered included the passing of two-dimensional arrays to subroutines expecting one-dimensional entities, unstructured usage of GOTO statements, and mathematical manipulations of array indeces which had to be changed to meet C array conventions.

FORTRAN and C store arrays differently. Since some routines pass two-dimensional arrays to routines expecting one-dimensional arrays, a direct conversion was not possible. The Orbital Operations Simulator, which is written in C, stores all arrays one-dimensionally. Multi-dimensioned arrays are conceptually stored columnwise, as in FORTRAN (see Fig. 1). Actual C storage is done rowwise. Since FORTRAN stores multi-dimensional arrays columnwise and the OOS conceptually stores arrays columnwise, svd was converted to store arrays conceptually columnwise. This is done in the following manner:

```
x[i][j] = x[ i + j * Row_dimension_of_x ].
```

The advantages of singly-dimensioning arrays may not be

1	2	3	4	5	6	7	8	9

Actual FORTRAN Storage

0	1	2	3	4	5	6	7	8

Actual C Storage

1	4	7
2	5	8
3	6	9

FORTRAN Storage (Columnwise)

0	1	2
3	4	5
6	7	8

C Storage (Rowwise)

0	3	6
1	4	7
2	5	8

OOS Storage (Columnwise)

FIGURE 1: Array Storage Methods for FORTRAN and C

intuitively obvious. The biggest advantage is that it allows the programmer to overcome C's inability to allow flexible array sizing in subroutines in which an array is passed as an argument. In other words, when an array is received as an argument in a subroutine, it must be declared with dimension sizes in the second and higher dimensions. For example, the three-dimensional double precision array "x" passed to a subroutine must be declared in that subroutine as follows:

```
double x[][2nd_Dimension_Size][3rd_Dimension_Size] ; .
```

Since x is actually a pointer to a string of linearly stored memory, 2nd_Dimension_Size and 3rd_Dimension_Size must be set at compile time to tell the program how to access the array elements. The first dimension size need not be given. This allows the programmer who uses singly-dimensioned arrays flexibility to conceptually redimension arrays at execution time. It also keeps the programmer from creating arbitrarily huge multi-dimensional arrays in the hopes that such a monstrosity would take care of "all" situations.

FORTRAN array indices normally start at one whereas the C convention is to start indeces at zero. In most cases, this difference in conventions is dealt with easily. However, svd has many array manipulations which require careful tracking of indices.

The biggest obstacle in the conversion was the "GOTO" statement. In most cases, these were easily replaced by "IF-THEN-ELSE" blocks. Other cases were more thought provoking. For example, the subroutine "SNRM2" was redone with two switch statements inside a while loop (see Appendix A). Elimination of the GOTO statement results in cleaner, structured, and more readable code. This is advantagious since structured code is more easily converted to other languages, such as Ada.

Nevertheless, there is now has a workable C version of svd. The program was verified by testing matrices whose eigenvalues were already known (see Reference 2). The two test cases are as follows:

Case 1:

$$\begin{vmatrix} 7 & 3 & 1 \\ 3 & 4 & 2 \\ 1 & 2 & 3 \end{vmatrix}$$

whose eigenvalues are 9.433551, 3.419421, and 1.147028.

Case 2:

$$\begin{vmatrix} 4 & 2 & 3 & 7 \\ 2 & 8 & 5 & 1 \\ 3 & 5 & 12 & 9 \\ 7 & 1 & 9 & 7 \end{vmatrix}$$

whose eigenvalues are 23.04466, 7.450091, 3.739112, and -3.233881.

The svd program output for these two test cases are listed in Appendix B.

3. The Vision Program

Several changes were made to the original Vision program provided by Rice University. It has been reworked to use LinCom's svd routine, matrix multiplier, matrix inverter, and other routines that do basic linear algebra. The program has also been converted to one-dimensional array storage using the same conventions as described in the svd section. Other changes were made to improve readability and to eliminate unnecessary memory allocation.

Minor changes were made to the original program to allow interfacing with LinCom's svd routine and matrix inverter. This temporarily gave LinCom a working version while the program was undergoing restructuring to conform to OOS standards. The original program arbitrarily set one-hundred as the maximum number of points that could be stored for a given object. This allowed allocation of 100 by 100 size arrays. So much memory was being allocated that the HP9000 Unix operating system killed execution of the program. The maximum was finally reset to 25 so that the program could operate without a memory failure. This reduced memory allocation by at least an order of magnitude. Arbitrarily huge working matrices were being created for the sole purpose of interfacing with LINPACK linear algebra routines. Since LinCom's linear algebra routines do not require huge working spaces, these temporary working matrices were eliminated in the OOS version.

The program was changed to work with one-dimensional arrays.

This allowed direct usage of LinCom's linear algebra and svd routines. It also led to better memory management since arbitrarily huge temporary working matrices could be eliminated. Many arrays were shrinked dramatically since their size was no longer dependent on the LINPACK linear algebra array conventions. For example, the array "XY" in many of the GIPC routines was reduced from 100 by 100 to 100 by 8. The array "SF" was reduced from 100 by 100 to 8 by 8 (see Reference 1).

In the original program, many variables and arrays were allocated and never used. This may be due to the fact that many of the subprograms are quite similar in function and probably originated as copies of each other with some variables used and others not used. Nevertheless, variables and arrays that were originally declared and never used were eliminated. This greatly improved the memory management situation.

Aesthetic changes were made to improve program readability. Loops were made to conform to regular C standards. Comments are currently being added to improve the understandability of the program.

At its completion, the new one-dimensional version has been run and compared with results from the old version. The new version emulates the old version. The maximum number of object points has been reset to 100 without killing memory.

4. Conclusion

There is now a clean, working version of the Vision program that can be integrated into the OOS. There is also an svd routine written in the C language that could be used for future programs.

APPENDIX A

Sample FORTRAN Program With C Conversion

```
      DOUBLE PRECISION FUNCTION SNRM2(N,SX,INCX)
      INTEGER NEXT
      DOUBLE PRECISION SX(1),CUTLO,CUTHI,HITEST,SUM,XMAX,ZERO,ONE
      DATA ZERO, ONE /0.0D0, 1.0D0/
      DATA CUTLO, CUTHI / 8.232D-11, 1.304D19 /

      IF(N .GT. 0) GO TO 10
         SNRM2 = ZERO
         GO TO 300

 10   ASSIGN 30 TO NEXT
      SUM = ZERO
      NN = N * INCX

      I = 1
 20      GO TO NEXT,(30, 50, 70, 110)
 30   IF( DABS(SX(I)) .GT. CUTLO) GO TO 85
      ASSIGN 50 TO NEXT
      XMAX = ZERO

                           PHASE 1.  SUM IS ZERO

 50   IF( SX(I) .EQ. ZERO) GO TO 200
      IF( DABS(SX(I)) .GT. CUTLO) GO TO 85

                           PREPARE FOR PHASE 2.
      ASSIGN 70 TO NEXT
      GO TO 105

                           PREPARE FOR PHASE 4.

 100  I = J
      ASSIGN 110 TO NEXT
      SUM = (SUM / SX(I)) / SX(I)
 105  XMAX = DABS(SX(I))
      GO TO 115

                     PHASE 2.  SUM IS SMALL.
                               SCALE TO AVOID DESTRUCTIVE UNDERFLOW.

 70   IF( DABS(SX(I)) .GT. CUTLO ) GO TO 75

                        COMMON CODE FOR PHASES 2 AND 4.
                        IN PHASE 4 SUM IS LARGE.  SCALE TO AVOID OVERFLOW.

 110  IF( DABS(SX(I)) .LE. XMAX ) GO TO 115
         SUM = ONE + SUM * (XMAX / SX(I))**2
         XMAX = DABS(SX(I))
         GO TO 200

 115  SUM = SUM + (SX(I)/XMAX)**2
      GO TO 200

                   PREPARE FOR PHASE 3.

 75   SUM = (SUM * XMAX) * XMAX
```

```
C
C
C     FOR REAL OR D.P. SET HITEST = CUTHI/N
C     FOR COMPLEX      SET HITEST = CUTHI/(2*N)
C
 85   HITEST = CUTHI/DBLE( N )
C
C                  PHASE 3.  SUM IS MID-RANGE.  NO SCALING.
C
      DO 95 J = I.NN,INCX
      IF(DABS(SX(J)) .GE. HITEST) GO TO 100
 95      SUM = SUM + SX(J)**2
      SNRM2 = DSQRT( SUM )
      GO TO 300
C
 200  CONTINUE
      I = I + INCX
      IF ( I .LE. NN ) GO TO 20
C
C              END OF MAIN LOOP
C
C              COMPUTE SQUARE ROOT AND ADJUST FOR SCALING
C
      SNRM2 = XMAX * DSQRT(SUM)
 300  CONTINUE
      RETURN
      END
```

```
#include        <stdio.h>          /* UNIX STANDARD INPUT AND OUTPUT        */
#include        <math.h>           /* STANDARD MATH LIBRARY                 */
#define         PRE_PHASE_CHECK  1        /* Check absolute value of sx[i]
                                             to see if >= cutlo            */
#define         PHASE_1          2        /* Scan zero components          */
#define         PHASE_2_AND_4    3        /* Component nonzero and <= cutlo or
                                             >= ( cuthi / n )              */
#define         PHASE_3          4        /* Component > cutlo             */
#define         LEVEL_1          5        /* 'next_calc' top level         */
#define         LEVEL_2          6
#define         LEVEL_3          7
#define         LEVEL_4          8
#define         LEVEL_5          9
#define         LEVEL_6          10

ouble snrm2( n, sx, incx )

nt n ;          /* INPUT: Input vector dimension                           */
nt incx ;       /* INPUT: Increment of x                                   */
ouble sx[] ;    /* INPUT: Vector sx                                        */


nt next_phase ;         /* Flag indicating next_phase phase of
                                   algorithm                               */
nt next_calc ;          /* Flag indicating which sequential step algorithm
                           level must execute                              */
nt i ;                  /* Loop counter                                    */
nt j ;                  /* Loop counter                                    */
nt n_x_incx ;           /* Vector dimension times increment of x           */

ouble xmax ;            /* Absolute value of array element                 */
ouble hitest ;          /* cuthi / double( n )                             */
ouble zero = 0.0 ;              /* Machine dependent 'zero'                */
ouble one = 1.0 ;               /* Machine dependent 'one'                 */
ouble cutlo = 4.441e-16 ;       /* Machine dependent                       */
ouble cuthi = 1.304e19 ;        /* Machine dependent                       */
ouble sum ;                     /* Algorithm parameter                     */

( n <= 0 ) return( zero ) ;

xt_phase = PRE_PHASE_CHECK ;
xt_calc = LEVEL_1 ;
m = zero ;
x_incx = n * incx ;
= 0 ;

r( ;; ) {

itch( next_phase ) {
   case PRE_PHASE_CHECK :
        if( fabs( sx[i] ) > cutlo ) {
              next_calc = LEVEL_2 ;
              break ;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        }
        else {
             next_phase = PHASE_1 ;
             xmax = zero ;
        }
    case PHASE_1 :
        if( sx[i] == zero ) {
             next_calc = LEVEL_6 ;
             break ;
        }
        else if( fabs( sx[i] ) > cutlo ) {
             next_calc = LEVEL_2 ;
             break ;
        }
        else {
             next_phase = PHASE_2_AND_4 ;
             next_calc = LEVEL_4 ;
             break ;
        }
    case PHASE_2_AND_4 :
        if( fabs( sx[i] ) > cutlo ) {
             next_calc = LEVEL_1 ;
             break ;
        }
    case PHASE_3 :
        if( fabs( sx[i] ) <= xmax ) {
             next_calc = LEVEL_5 ;
             break ;
        }
        else {
             sum = one + sum * ( xmax / sx[i] ) * ( xmax / sx[i] ) ;
             xmax = fabs( sx[i] ) ;
             next_calc = LEVEL_6 ;
             break ;
        }
    default :
        fprintf(stderr,"PROGRAM FAILED IN 'snmr2'- switch 'next_phase /n" ) ;
        exit(1) ;
        break ;
     /* end switch( next_phase ) */



itch( next_calc ) {
    case LEVEL_1:
        sum = sum * xmax * xmax ;
    case LEVEL_2 :
        hitest = cuthi / ( ( double ) ( n ) ) ;
        for( j = i ; j < n_x_incx ; j += incx ) {
             if( fabs( sx[j] ) >= hitest ) {
                  next_calc = LEVEL_3 ;
                  break ;
             }
             sum = sum + sx[j] * sx[j] ;
        }
        if( next_calc != LEVEL_3 ) return( sqrt( sum ) ) ;
```

```
        case LEVEL_3 :
             i = j ;
             next_phase = PHASE_3 ;
             sum = ( sum / sx[i] ) / sx[i] ;
        case LEVEL_4 :
             xmax = fabs( sx[i] ) ;
        case LEVEL_5 :
             sum = sum + ( sx[i] / xmax ) * ( sx[i] / xmax ) ;
        case LEVEL_6 :
             i = i + incx ;
             if( i < n_x_incx ) break ;
             else {
                  return( xmax * sqrt( sum ) ) ;
             }
        default :
             fprintf(stderr,"PROGRAM FAILED IN snmr2 switch 'next_calc'/n") ;
             exit ( 1 ) ;
             break ;
        /* end switch( next_calc ) */

        /* end for( ;; ) */
```

Appendix B

Test Results of SVD Algorithm in C Language

```
$ cat out file
        C  VERSION OF SVD - OUTPUT FILE (case 1)


              X MATRIX -- before entering svd

   7.0000000000        3.0000000000        1.0000000000
   3.0000000000        4.0000000000        2.0000000000
   1.0000000000        2.0000000000        3.0000000000


              S MATRIX  (Eigenvalues on diagonal)

   9.4335510972        0.0000000000        0.0000000000
   0.0000000000        3.4194211565        0.0000000000
   0.0000000000        0.0000000000        1.1470277463


              U MATRIX  (Left Singular Vectors)

  -0.7881466819        0.5572297562        0.2613805780
  -0.5423013758       -0.4278539749       -0.7230838084
  -0.2910910950       -0.7116431514        0.6393981541


              V MATRIX  (Right Singular Vectors)

  -0.7881466819        0.5572297562        0.2613805780
  -0.5423013758       -0.4278539749       -0.7230838084
  -0.2910910950       -0.7116431514        0.6393981541


                     U x U_TRANSPOSE

   1.0000000000        0.0000000000        0.0000000000
   0.0000000000        1.0000000000        0.0000000000
   0.0000000000        0.0000000000        1.0000000000


                     V x V_TRANSPOSE

   1.0000000000        0.0000000000        0.0000000000
   0.0000000000        1.0000000000        0.0000000000
   0.0000000000        0.0000000000        1.0000000000


              U x S x V_TRANSPOSE = X (as desired)

   7.0000000000        3.0000000000        1.0000000000
   3.0000000000        4.0000000000        2.0000000000
   1.0000000000        2.0000000000        3.0000000000
```

ORIGINAL PAGE IS OF POOR QUALITY

cat out file

C VERSION OF SVD - OUTPUT FILE (case 2)

X MATRIX -- before entering svd

4.0000000000	2.0000000000	3.0000000000	7.0000000000
2.0000000000	8.0000000000	5.0000000000	1.0000000000
3.0000000000	5.0000000000	12.0000000000	9.0000000000
7.0000000000	1.0000000000	9.0000000000	7.0000000000

S MATRIX (Eigenvalues on diagonal)

23.0446725411	0.0000000000	0.0000000000	0.0000000000
0.0000000000	7.4501012997	0.0000000000	0.0000000000
0.0000000000	0.0000000000	3.7391178738	0.0000000000
0.0000000000	0.0000000000	0.0000000000	3.2338917147

U MATRIX (Left Singular Vectors)

-0.3456580386	-0.2872994068	0.6787287859	-0.5807812035
-0.3117015538	0.8489634413	0.3749567016	0.2037417203
-0.6883556583	0.1076069808	-0.6174607983	-0.3651429691
-0.5563534392	-0.4302799831	0.1322001119	0.6984648289

V MATRIX (Right Singular Vectors)

-0.3456580386	-0.2872994068	0.6787287859	0.5807812035
-0.3117015538	0.8489634413	0.3749567016	-0.2037417203
-0.6883556583	0.1076069808	-0.6174607983	0.3651429691
-0.5563534392	-0.4302799831	0.1322001119	-0.6984648289

U x U_TRANSPOSE

1.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	1.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	1.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	1.0000000000

V x V_TRANSPOSE

1.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	1.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	1.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	1.0000000000

U x S x V_TRANSPOSE = X (as desired)

4.0000000000	2.0000000000	3.0000000000	7.0000000000
2.0000000000	8.0000000000	5.0000000000	1.0000000000
3.0000000000	5.0000000000	12.0000000000	9.0000000000
7.0000000000	1.0000000000	9.0000000000	7.0000000000

References

1. Fotedar, Sunil and de Figueiredo, Rui J. P. Software Implementation of Image Point Correspondence Algorithms For Motion Parameter Determination. Dept. of Electrical and Computer Engineering, Rice University, Houston. Technical Report EE8709. April 29, 1987.

2. Hornbeck, Robert W. Numerical Methods. Quantum Publishers, New York, 1975. Pages 258, 260, 26.

September 25, 1987

IMPLEMENTATION OF VISION ALGORITHMS IN DOS

STATUS UPDATE

9/25/87

FUNCTIONAL CAMERA MODEL

INPUTS:

vehicle state from dynamics
target state from dynamics
target polygon model from object library
  vertices, no. of vertices
  faces, no. of faces
  connectivity of faces
  moment invariants of faces

ALGOTITHM:

- calculate actual target position in camera frame
- if target not previously seen
  check if in range (if not, status = "not seen" and exit)
  check if in field of view
  (if not, status = "not seen" and exit)
  if in range, wait certail lag time before aquisition
  (if time in range is less than lag time,
  status = "not seen" and exit)
- if target previously seen but moved out of range,
  lose sighting and exit

***** OBJECT SEEN *****

- rotate object model from library to match vehicle dynamics
- extract wireframe
  no. of faces
  definition of vertices of each face
  face connectivity

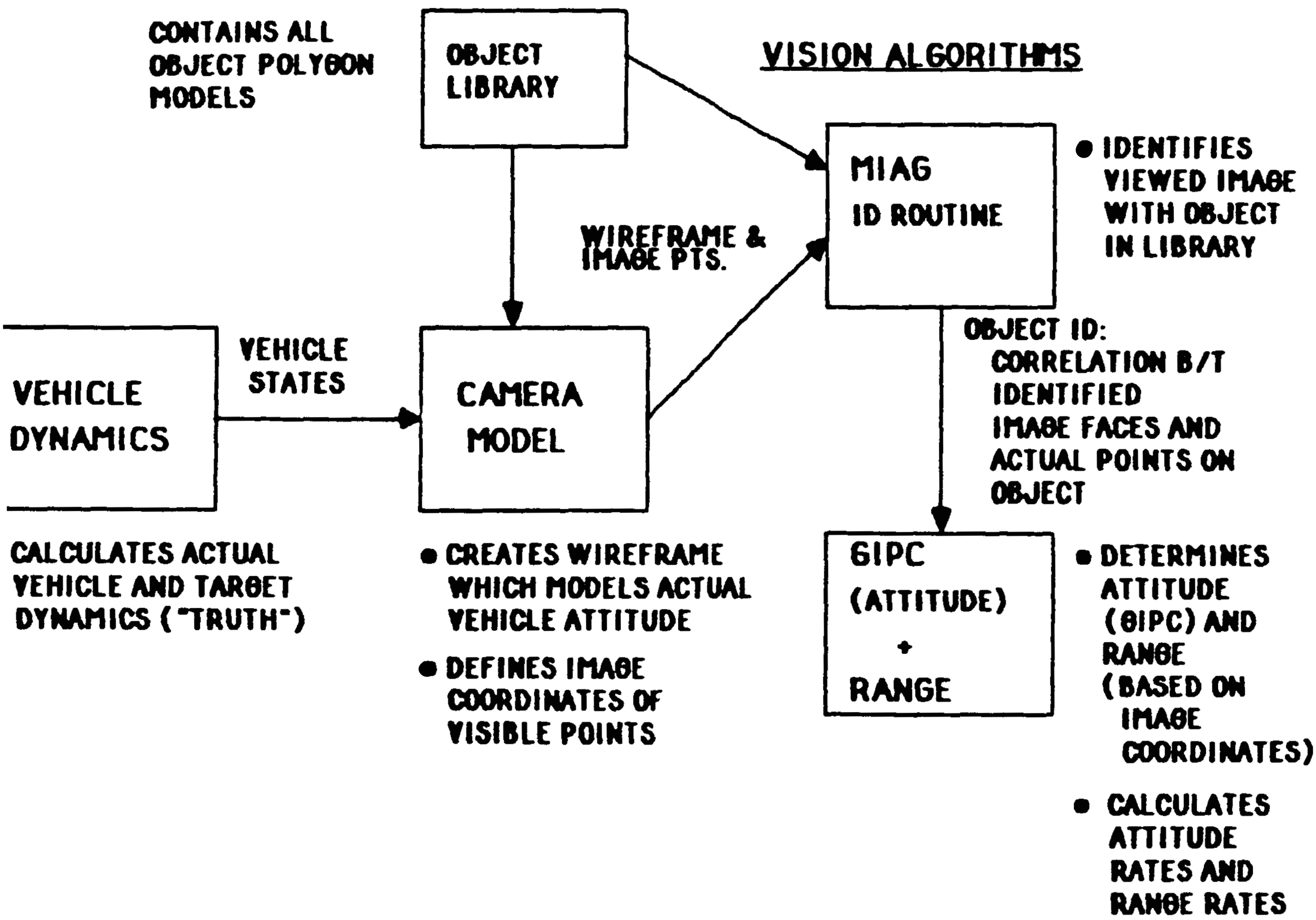
***** ASSUMPTIONS: 1) POLYGONAL OBJECT
2) PERFECT WIREFRAME EXTRACTION

- identify visible points of object (from wireframe)
- calculate image points based on actual object position, coordinates of visible points in object frame, and lens focal length

FUNCTIONAL CAMERA MODEL

OUTPUT:

wireframe for MIAG identification routine
image points for GIPC attitude determination and range & range rate determination



CONCEPTUAL FLOW DIAGRAM OF VISION MODELS IN OOS

CURRENT STATUS

- o A functional camera model is built. This conforms to OOS standards. Code is ready for integration testing and inclusion into OOS.

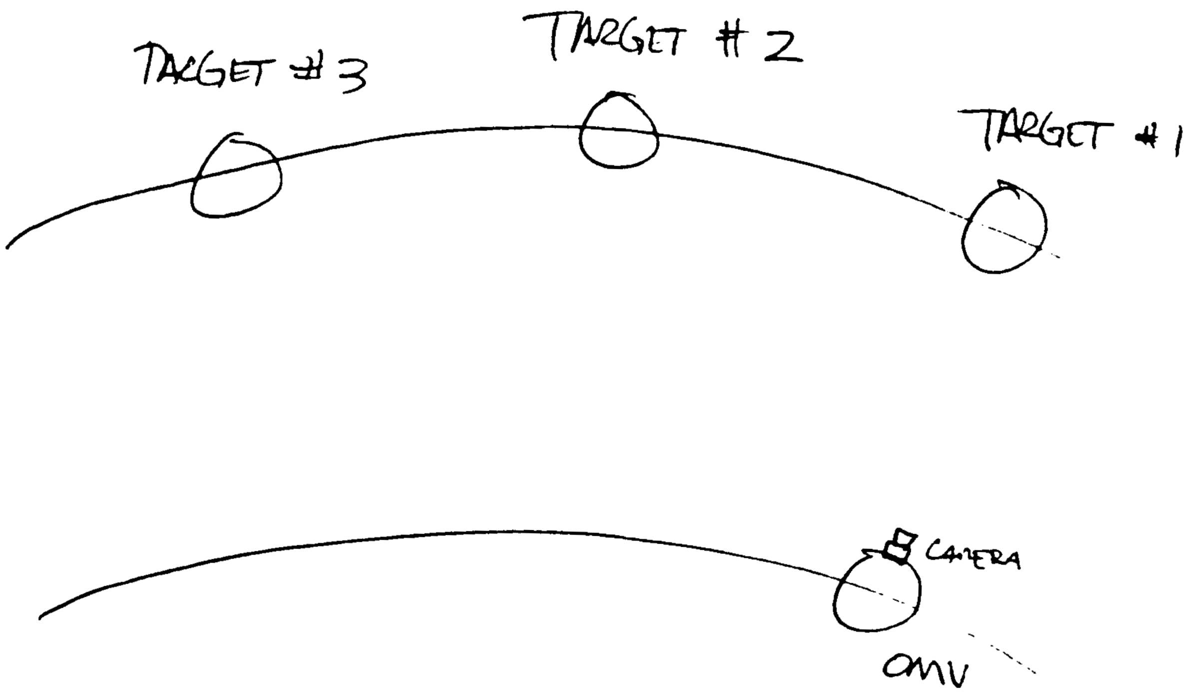
- o MIAG identification routines have been modified to conform to OOS standards. Code is ready for integration testing and inclusion into OOS.

- o LinCom is currently developing procedures to combine the MIAG identification information with the GIPC attitude determination routine. Currently, the GIPC requires that the image points from the two frames (before and after rotation) be specifically paired , which it does artificially within the software. However, MIAG provides a coorespondence map between the wireframe faces and the object faces in the library. This map can be used to uniquely identify image points with respect to the object model. Thus, GIPC knows which points it has and can determine the attitude of the object.

- o After initial testing of the integrated vision system, random noise will be added to the vertices of the wireframe and to the image points in the camera model. This small noise will simulate error in wireframe extraction and image point identification.

ORIGINAL PAGE IS
OF POOR QUALITY

[illegible] OOS SCENARIO



- OMV IN CIRCULAR ORBIT
- THREE TARGETS IN HIGHER CIRCULAR ORBIT, WITHIN CAMERA RANGE
- AS OMV PASSES UNDERNEATH TARGET, CAMERA VIEWS TARGET. VISION ALGORITHMS DETERMINE I.D., ATTITUDE & ATT. RATES, RANGE & RANGE RATE
- AS OMV MOVES IN ITS ORBIT, TARGET CHANGES

  [redacted]

  OMV RECIEVES "UPLINK" - ESTIMATED POSITION OF NEW TARGET
- VIEW 3 TARGETS IN ALL

October 15, 1987

SIMULATION OF ROBOTIC SPACE OPERATIONS

INTEGRATION OF VISION ALGORITHMS INTO AN ORBITAL OPERATIONS SIMULATION

YASHVANT JANI, PhD
WILLIAM L. OTHON

LinCom CORPORATION

RICIS Symposium
15 October 1987

AGENDA

- OBJECTIVES
- USE OF SIMULATION
- INTEGRATION OF ROBOTICS / VISION ALGORITHMS INTO AN ORBITAL OPERATIONS SIMULATION
- CURRENT EFFORT: INTEGRATION OF VISION ALGORITHMS FROM RICE UNIVERSITY WITH ORBITAL MANUVERING VEHICLE (OMV) MODEL
- PROJECT STATUS
- FUTURE EFFORT

OBJECTIVES

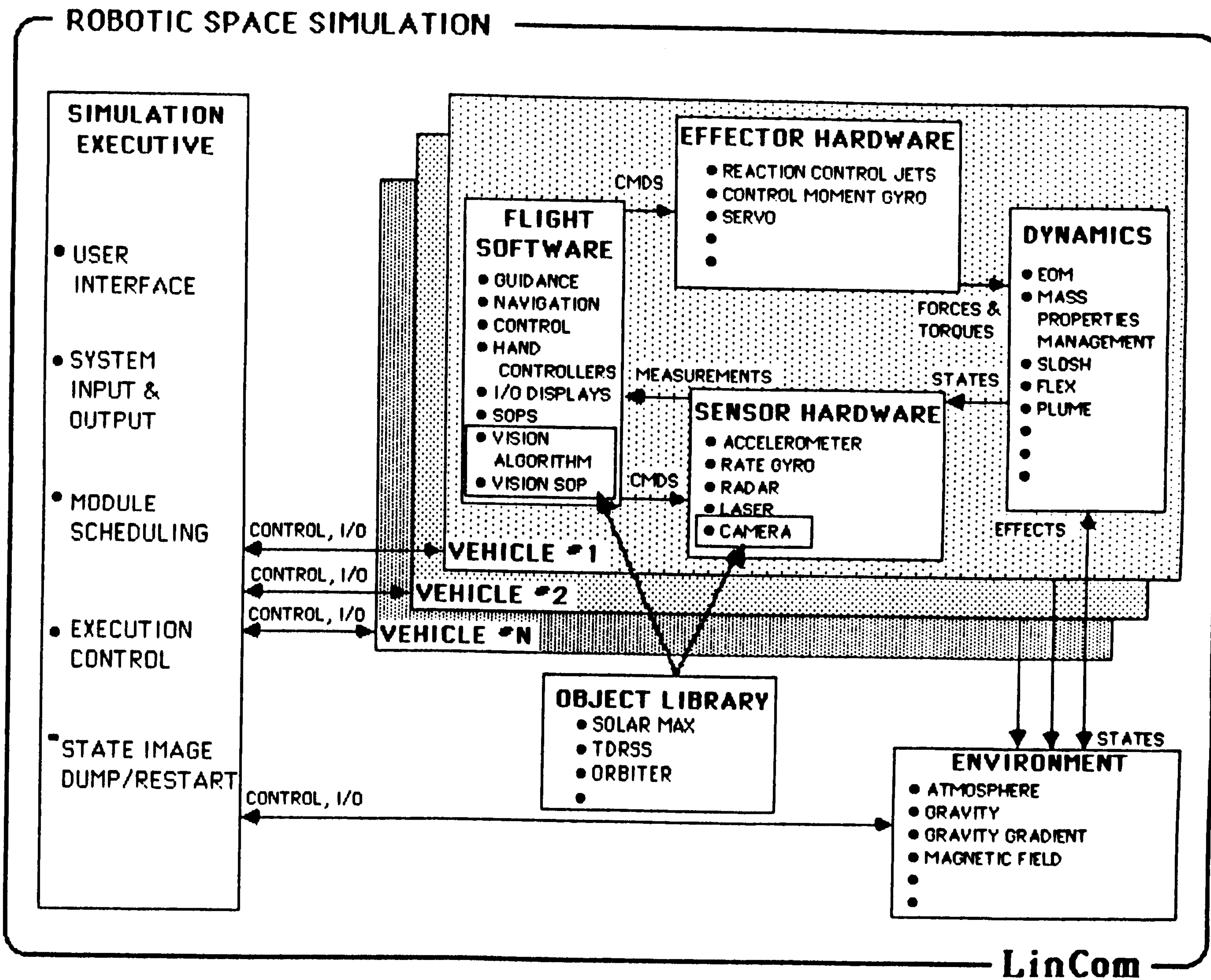
- DEVELOP A *TESTBED* FOR INTEGRATION OF ROBOTICS SUBSYSTEMS AND SPACE VEHICLES SIMULATION
  - IMPLEMENT VISION/ROBOTICS ALGORITMS
  - PERFORM SYSTEMS INTEGRATION ANALYSIS
- STUDY OPERATIONAL ASPECTS OF ROBOTIC SPACE SYSTEMS AND MISSIONS

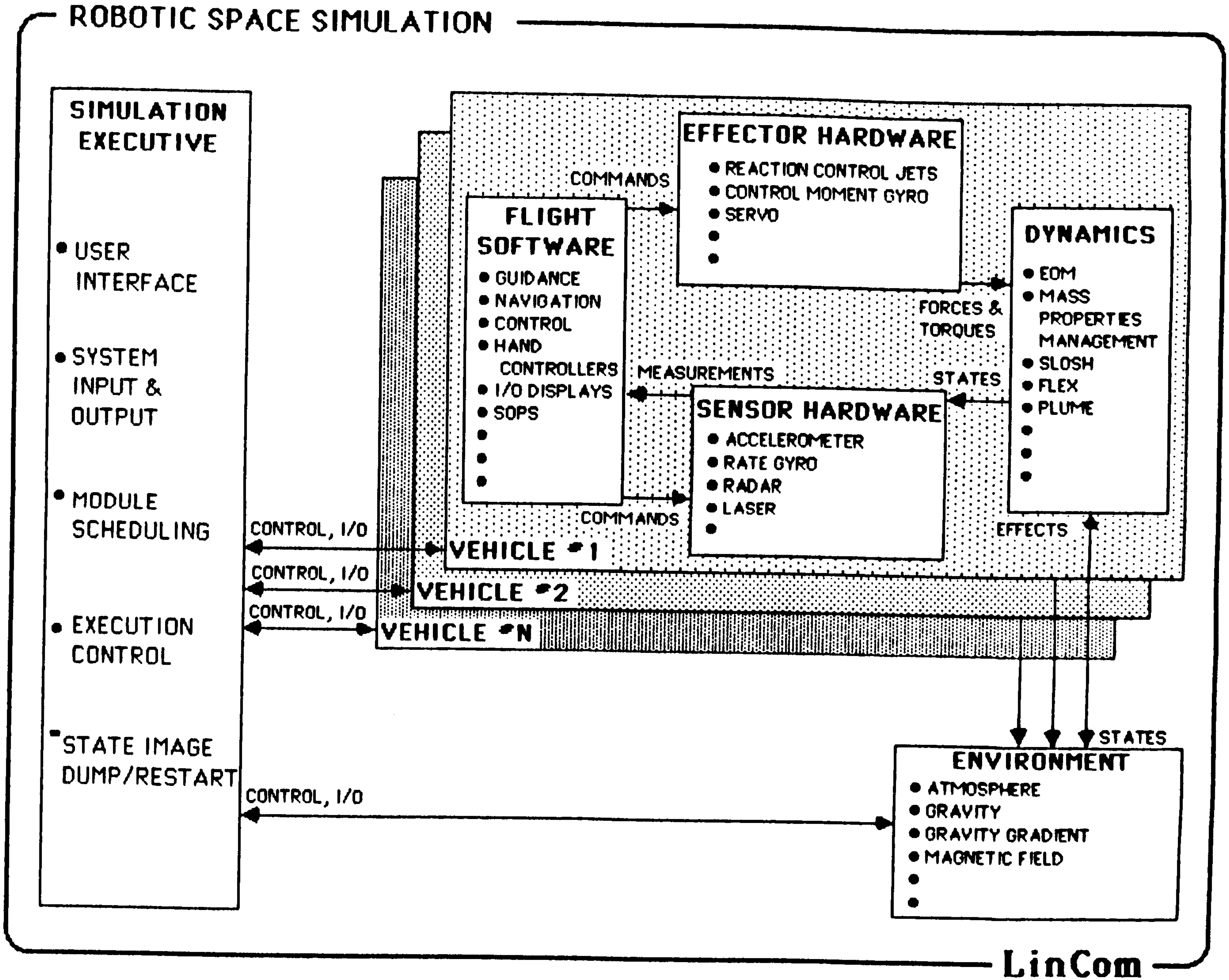
USE OF SIMULATION

- PRE-FLIGHT ANALYSIS
  - DEFINITION OF MISSION REQUIREMENTS
  - PERFORMANCE ENVELOPES
  - FLIGHT ASSESSMENT
- DEVELOPMENT OF MISSION SCENARIOS
  - OPERATIONS
  - PROCEDURES
  - INTEGRATION OF SEVERAL VEHICLES AND SUBSYSTEMS INTO A COORDINATED SCENARIO
- INTRODUCTION OF NEW VEHICLES / SUBSYSTEMS
  - SPECIFICATION AND ANALYSIS
  - SUBSYSTEMS REQUIREMENTS ANALYSIS

INTEGRATION OF ROBOTICS/VISION ALGORITHMS INTO AN ORBITAL OPERATIONS SIMULATION

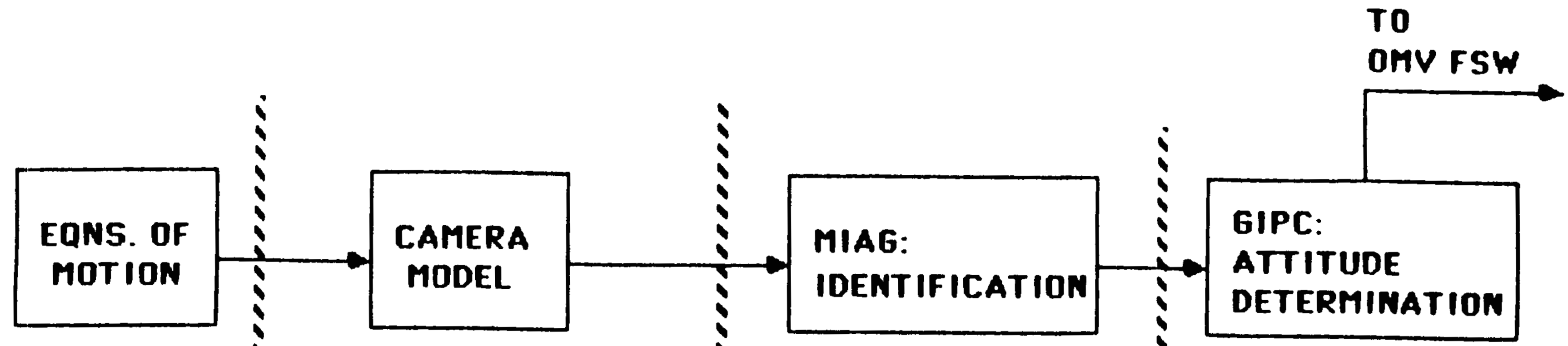
- TESTBED REQUIREMENTS
  - MODULARITY
  - RAPID PROTOTYPING
  - FIDELITY
- ROBOTICS COMPONENTS IN OOS
  - VISION
  - REMOTE MANIPULATOR SYSTEM (RMS)
  - AUTOMATED FLIGHT / EXPERT SYSTEMS


ROBOTIC SPACE SIMULATION
SIMULATION EXECUTIVE
• USER INTERFACE
• SYSTEM INPUT & OUTPUT
• MODULE SCHEDULING
• EXECUTION CONTROL
• STATE IMAGE DUMP/RESTART
CONTROL, I/O
CONTROL, I/O
CONTROL, I/O
CONTROL, I/O
FLIGHT SOFTWARE
• GUIDANCE
• NAVIGATION
• CONTROL
• HAND CONTROLLERS
• I/O DISPLAYS
• SOPS
• VISION ALGORITHM
• VISION SOP
CMDS
EFFECTOR HARDWARE
• REACTION CONTROL JETS
• CONTROL MOMENT GYRO
• SERVO
FORCES & TORQUES
DYNAMICS
• EOM
• MASS PROPERTIES MANAGEMENT
• SLOSH
• FLEX
• PLUME
MEASUREMENTS
STATES
SENSOR HARDWARE
• ACCELEROMETER
• RATE GYRO
• RADAR
• LASER
• CAMERA
CMDS
EFFECTS
VEHICLE #1
VEHICLE #2
VEHICLE #N
OBJECT LIBRARY
• SOLAR MAX
• TDRSS
• ORBITER
STATES
ENVIRONMENT
• ATMOSPHERE
• GRAVITY
• GRAVITY GRADIENT
• MAGNETIC FIELD
LinCom



ROBOTIC SPACE SIMULATION
SIMULATION EXECUTIVE
• USER INTERFACE
• SYSTEM INPUT & OUTPUT
• MODULE SCHEDULING
• EXECUTION CONTROL
• STATE IMAGE DUMP/RESTART
CONTROL, I/O
CONTROL, I/O
CONTROL, I/O
CONTROL, I/O
FLIGHT SOFTWARE
• GUIDANCE
• NAVIGATION
• CONTROL
• HAND CONTROLLERS
• I/O DISPLAYS
• SOPS
COMMANDS
EFFECTOR HARDWARE
• REACTION CONTROL JETS
• CONTROL MOMENT GYRO
• SERVO
FORCES & TORQUES
DYNAMICS
• EOM
• MASS PROPERTIES MANAGEMENT
• SLOSH
• FLEX
• PLUME
MEASUREMENTS
STATES
SENSOR HARDWARE
• ACCELEROMETER
• RATE GYRO
• RADAR
• LASER
COMMANDS
EFFECTS
VEHICLE #1
VEHICLE #2
VEHICLE #N
STATES
ENVIRONMENT
• ATMOSPHERE
• GRAVITY
• GRAVITY GRADIENT
• MAGNETIC FIELD
LinCom

ROBOTIC SPACE SIMULATION



VEHICLE DYNAMICS	SENSORS	FSW	FSW
- PROPAGATES EQUATIONS OF MOTION OF TARGET AND CHASER (OMV)	- CHECKS IF OBJECT IS WITHIN RANGE AND IN FIELD OF VIEW - CALCULATES THE ORIENTATION OF THE TARGET IN CAMERA FRAME - FUNCTIONAL WIREFRAME EXTRACTION ROUTINE	- MATCHES TARGET WIREFRAME WITH MODELS IN OBJECT LIBRARY	- UNIQUELY IDENTIFIES POINTS USING MAP FROM MIAG - DETERMINES ATTITUDE OF OBJECT IN CAMERA FRAME
OUTPUT: "TRUE VEHICLE STATE"	OUTPUT: WIREFRAME OF ROTATED AND TRANSLATED OBJECT MODEL	OUTPUT: TARGET ID AND COORESPONDENCE MAP BETWEEN TARGET AND CHASER	OUTPUT: ATTITUDE AND RATE OF TARGET

LinCom

CURRENT EFFORT

INTEGRATION OF VISION ALGORITHMS WITH ORBITAL MANUVERING VEHICLE (OMV) MODEL

- VISION ALGORITHMS FROM RICE UNIVERSITY
  - OBJECT IDENTIFICATION
    - MOMENT INVARIANT/ATTRIBUTED GRAPH (MIAG):
  - ATTITUDE DETERMINATION
    - GENERALIZED IMAGE POINT CORRESPONDENCE (GIPC):
    - MIAG EXTENSION (TENSORS)
- OMV MODEL
  - RIGID BODY DYNAMICS
  - REACTION CONTROL SYSTEM (RCS) JETS
  - OMV FLIGHT SOFTWARE (CONTROL SYSTEM, GUIDANCE, ETC)
  - CAMERA MODEL
    - FOCAL LENGTH , RANGE , FIELD OF VIEW
    - EXTRACTION OF 2D WIREFRAME (LOW-LEVEL IMAGE PROCESSING)

CURRENT STATUS

- ALGORITHMS IMPLEMENTATION COMPLETE
  - CAMERA MODEL
  - FUNCTIONAL WIREFRAME EXTRACTION
  - MIAG IDENTIFICATION AND GIPC ATTITUDE DETERMINATION IN OOS
- INTEGRATION TESTING IN PROGRESS
  - MODULE INTERFACES COMPLETE
  - NEW EVENT-DRIVEN OMV SEQUENCER GENERATED
- TEST CASE DESCRIPTION
  - THREE VEHICLES IN SAME ORBIT
  - OMV WITH CAMERA IN LOWER ORBIT
  - AS OMV APPROACHES TARGET, THE VISION ALGORITHMS WILL IDENTIFY OBJECT AND COMPUTE ATTITUDE AND ATTITUDE RATES

FUTURE EFFORT

- PERFORMANCE ANALYSIS OF VISION ALGORITHMS
  - INTRODUCE NOISE, ERROR, AND LAG TIME INTO WIREFRAME EXTRACTION ROUTINE
  - ANALYZE RATE OF INPUT FROM VISION ALGORITHMS TO OMV FSW (i.e. PROCESSING SPEED REQUIRED FOR APPLICATIONS)
  - ANALYZE ACCURACY REQUIREMENT FOR ORBITAL OPERATIONS
  - EXPAND OBJECT LIBRARY
- HARDWARE/SOFTWARE TESTING IN LABORATORY WITH PROCESSING TIME ANALYSIS
- INTEGRATE OTHER VISION / ROBOTIC ALGORITHMS INTO OOS
  - NEW ATTITUDE DETERMINATION ROUTINES
  - RMS ALGORITHMS
    - KINEMATICS
    - SERVOS AND GYROS
- INTEGRATE VISION/ROBOTICS ALGORITHMS WITH OTHER MODELS
  - VISION + RMS + MMU = AUTONOMOUS ROBOT
  - VISION + SPACE STATION => TRAFFIC CONTROL

END OF PRESENTATION