AD-A260 567

ARO 26813.5-EL

2

PURDUE UNIVERSITY
SCHOOL OF AERONAUTICS AND ASTRONAUTICS

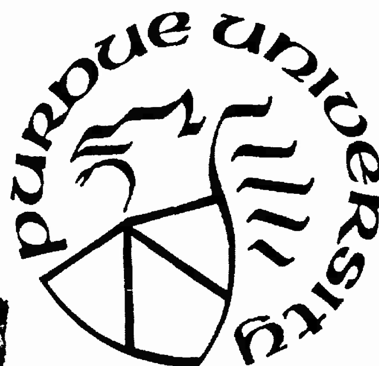
A Hierarchical Target Extraction, Recognition, and Tracking (HiTert) System

by
Jun Lu
Dominick Andrisani, II
and M. Fernando Tenorio

March 31, 1992

Final Report to the U. S. Army Research Office
Contract No. DAAL03-89-K-0086

DTIC
ELECTE
FEB 19 1993
S E D



DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

93-03350

1282 Grissom Hall
West Lafayette, Indiana 47907-1282

93 2 18 048

A Hierarchical Target Extraction, Recognition, and Tracking (HiTert) System

by
Jun Lu
Dominick Andrisani, II
and M. Fernando Tenorio

March 31, 1992

Final Report to the U. S. Army Research Office
Contract No. DAAL03-89-K-0086

School of Aeronautics and Astronautics,
and School of Electrical Engineering
Purdue University
West Lafayette, IN
47907

Accesion For
NTIS CRA&I X
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
3/31/92

3. REPORT TYPE AND DATES COVERED
Final 1 Apr 89 - 31 Mar 92

4. TITLE AND SUBTITLE
A Hierarchical Target Extraction, Recognition and Tracking (HiTert) System

5. FUNDING NUMBERS
DAAL03-89-K-0086

6. AUTHOR(S)
Jun Lu, Dominick Andrisani II, and M. Fernando Tenorio

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
School of Aeronautics and Astronautics
and School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 26813.5-EL

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This research contributed to the development of a hierarchical target extraction, identification, and tracking system based on imaging sensors. The work suggests that passive tracking of ground targets is a desirable possibility. Our work involved 1) computer rendering of a color image database containing 1000 images of a maneuvering tank; 2) the use of image derived data to help track a violently maneuvering tank; 3) the use of the Cantata Visual programming Language to design the multiple interconnected algorithms in an intuitive and extensible manner; 4) the delivery of this software to the U. S. Army Armament Research and Development Center (ARDEC) at Picatinny Arsenal, New Jersey; and 5) presenting of a one day short course to engineers at ARDEC concerning the design and use of the software. This final report describes that software. Our research has suggested the need for the following future work: 1) improved realism in the computer generated image database; 2) development of additional higher level image processing and tracking modules using Cantata; and 4) implementation of a way to communicate between competing algorithms.

14. SUBJECT TERMS
target tracking, image processing, optical tracking

15. NUMBER OF PAGES
124

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

1 Introduction **1**

1.1 Structure of the HiTert System 1

1.2 Scenario Under Consideration 3

1.3 Some Implementation Issues 5

1.3.1 A Generic Subsystem 5

1.3.2 Non-real Time Simulation 7

1.4 Development Environment 9

1.4.1 Hardware Architecture 9

1.4.2 Interprocess Communication(IPC) 9

1.4.3 Cantata Visual Programming Language 10

2 Installation Guide **11**

2.1 Platform Requirements 11

2.2 Unpack the Source 12

2.3 Installation Environment Variables 12

2.4 Compilation and Installation 13

3 Getting Started **14**

3.1 X Window System 14

3.2 Khoros System 14

3.3 Environment Variables for the HiTert System 15

3.4 Command Line Interface 17

3.5 Cantata Visual Language Interface 19

4 User's Guide **21**

4.1 Overview of the Software Modules 21
4.2 Data Control Module 24
4.2.1 Introduction 24
4.2.2 Command Line Options 25
4.2.3 I/O File Specification 28
4.3 World-view Camera 35
4.3.1 Introduction 35
4.3.2 Command Line Options 36
4.3.3 I/O File Specification 39
4.4 Tracking Camera 41
4.4.1 Introduction 41
4.4.2 Command Line Options 42
4.4.3 I/O File Specification 43
4.5 Image Processor 44
4.5.1 Introduction 44
4.5.2 Command Line Options 44
4.5.3 I/O File Specification 49
4.6 Tracker 52
4.6.1 Introduction 52
4.6.2 Command Line Options 52
4.6.3 I/O File Specification 53
4.7 Predictor 55
4.7.1 Introduction 55
4.7.2 Command Line Options 56
4.7.3 I/O File Specification 57
4.8 Error Analysis 58
4.8.1 Introduction 58

4.8.2 Command Line Options 59
4.8.3 I/O File Specification 63
4.9 Instrumentation Module 65
4.9.1 Introduction 65
4.9.2 Command Line Options 67
4.9.3 I/O File Specification 69

5 World-view Image Database **71**
5.1 Introduction 71
5.2 Trajectory Data 71
5.3 Scenario 74
5.4 Image Generation 75
5.5 I/O File Specification 76
5.5.1 Command Line Options 76
5.5.2 Input Data File 80
5.5.3 Output Description File 81
5.5.4 Image Files 83

6 Conclusions **84**
6.1 Summary of Achievements 84
6.2 Future Research 85

A α-β-γ Tracker Design **87**
A.1 Introduction 87
A.2 Target State Estimator Equation 88
A.2.1 State Equation 88
A.2.2 Measurement Equation 89
A.2.3 Discretization 89
A.3 Target State Estimator Gains 90
A.4 Predictor Equation 92

A.5 Numerical Results 92
A.6 Conclusions 95

B Dartboard Analysis **96**
B.1 Definitions 96
B.2 The Hit Point 98
B.3 Coordinate Transformation 99

C Geometrical Transformation **101**
C.1 Introduction 101
C.2 Viewing Transformation 102
C.2.1 World to Screen Transformations 102
C.2.2 Scaling Compensation 104
C.2.3 Perspective Transformation 104
C.2.4 Screen Transformation 105
C.2.5 Concatenation of Transformations 107
C.3 Screen to World Transformations 108

D Manual Pages **110**

Bibliography **122**

List of Figures

1.1 Structure of HiTert system 2

1.2 Tracking scenario 4

1.3 Trajectory data 6

1.4 A generic subsystem 7

1.5 World and camera views 8

4.1 Software modules and data flows 22

4.2 Definitions of the dartboard 59

4.3 Definition of the burst interval 60

4.4 The window arrangement. 66

5.1 Scenario under consideration 73

5.2 Trajectory data 74

A.1 α-β-γ tracker RMS prediction error 93

A.2 α-β tracker RMS prediction error 94

A.3 α-β tracker RMS estimation error($t_p = 0$ second) 95

B.1 Dartboard schematic 97

C.1 World to eye coordinate transformation 102

C.2 Perspective transformation 106

C.3 Screen transformation 107

Chapter 1

Introduction

In this chapter, the underlying concepts of the Hierarchical Target Extraction, Recognition and Tracking(HiTert) System will be addressed. These concepts are essential for understanding the rest of the material. Some implementation issues will also be briefly discussed.

1.1 Structure of the HiTert System

The HiTert system is aimed to study a hierarchical target extraction, recognition and tracking system based on passive sensors, which could be integrated with other battlefield resources[1]. The HiTert system consists of the mutually beneficial subsystems running different algorithms and executing in parallel at several hierarchical levels. Together these subsystems cooperate in seeking the solution for the complex problem beyond the capability of any one of the subsystems. The HiTert system is composed of four hierarchical levels as shown in Figure 1.1:

- Preprocessing Level: Imagery acquiring, noise filtering and compensation of the effect of the imaging system[3].

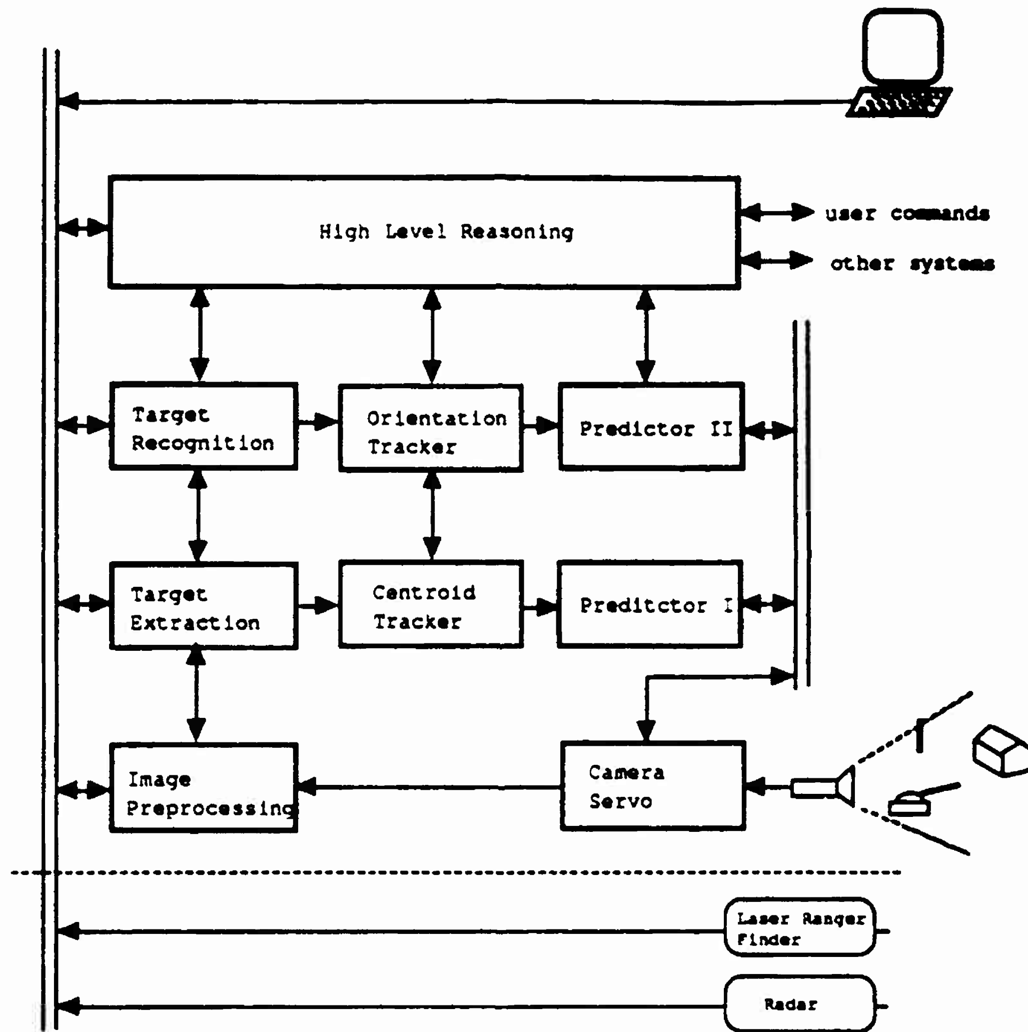


Figure 1.1: Structure of HiTert system

- Low Level: Target extraction and tracking. The tracker at this level mainly uses primitive information such as the target centroid position, while the image processor at this level classifies the pixels of images and separates the target from the background.
- Middle Level: Target recognition and tracking. The image processor at this level is responsible for yielding target orientation information and object identification. These refined results are utilized by the tracker at this level. The processing at this level has a longer time period than that at the low level processing.
- High Level: Command, Control and Communication. This high-level reasoning module is responsible for the coordination of low-level subsystems, information fusion, user interaction, and interaction with other battle field resources. The blackboard architecture allows the information sharing, subsystem coordination and integration with other battle field resources.

1.2 Scenario Under Consideration

Figure 1.2 shows the simple battle field scene being considered. It consists of a flat terrain, a moving tracked vehicle, a house, a generic tree. Such a simple scene still possesses the characteristics of a complicated battle field environment. For instance, certain obstacle avoidance strategies will be adopted by the driver in the decision-making of the driving logic; overlapping of the target and ground obstacles may pose ambiguity problems for the image processor to resolve. However, such a simple scenario ameliorates the complexity of scene rendering and also makes it more amenable to study and analysis.

It should be noted that to be consistent with the Doré 3D graphics system[11], which is used to generate the world view images, the camera coordinate system is such that the "camera" looks along the negative Z_c

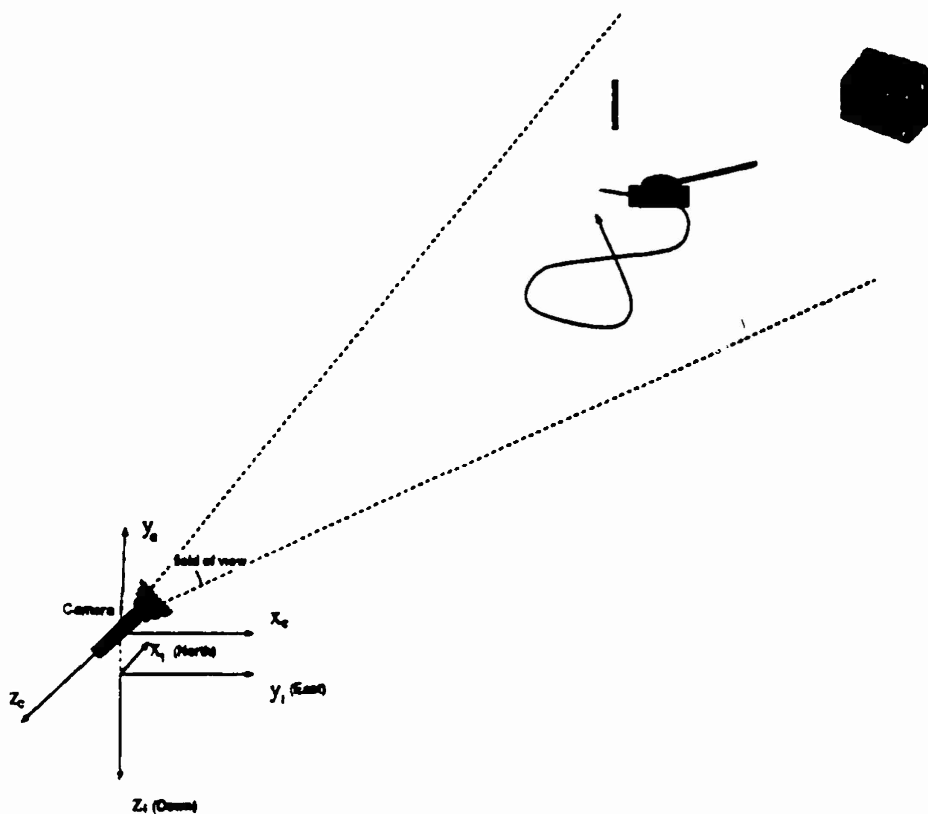


Figure 1.2: Tracking scenario

direction and x_c, y_c are mapped to the horizontal dirction(left to right) and vertical direction(down to up) of the screen.

One set of trajectory data is shown in Figure 1.3. It is believed that the target may do any maneuvering as the driver deems necessary to avoid the threat in a real environment. A maneuvering is realizable as long as the certain dynamic constraints are satisfied:

- The acceleration and deceleration can not exceed the thrust capability as determined mainly by the characteristics of the propulsion system and the terrain sustaining capability.

- Under the no-skidding condition, the target lateral acceleration should not exceed the terrain sustaining capability determined by the lateral frictional coefficient:

$$a_n = \omega v \leq \mu_t g$$

  where a_n is the centripetal acceleration, ω the angular rate, v the forward speed, μ_t the lateral frictional coefficient, g the gravitational constant.

1.3 Some Implementation Issues

1.3.1 A Generic Subsystem

The HiTert system spans several research fields such as target dynamics modeling and tracking, image processing, artificial intelligence and database. Many subproblems are being actively studied. The full implementation of such a hierarchical system is difficult. And further in-depth study of HiTert is needed. As a result, we break down the complex problem into simple ones, concentrating on building a testbed for the subsystem operating at

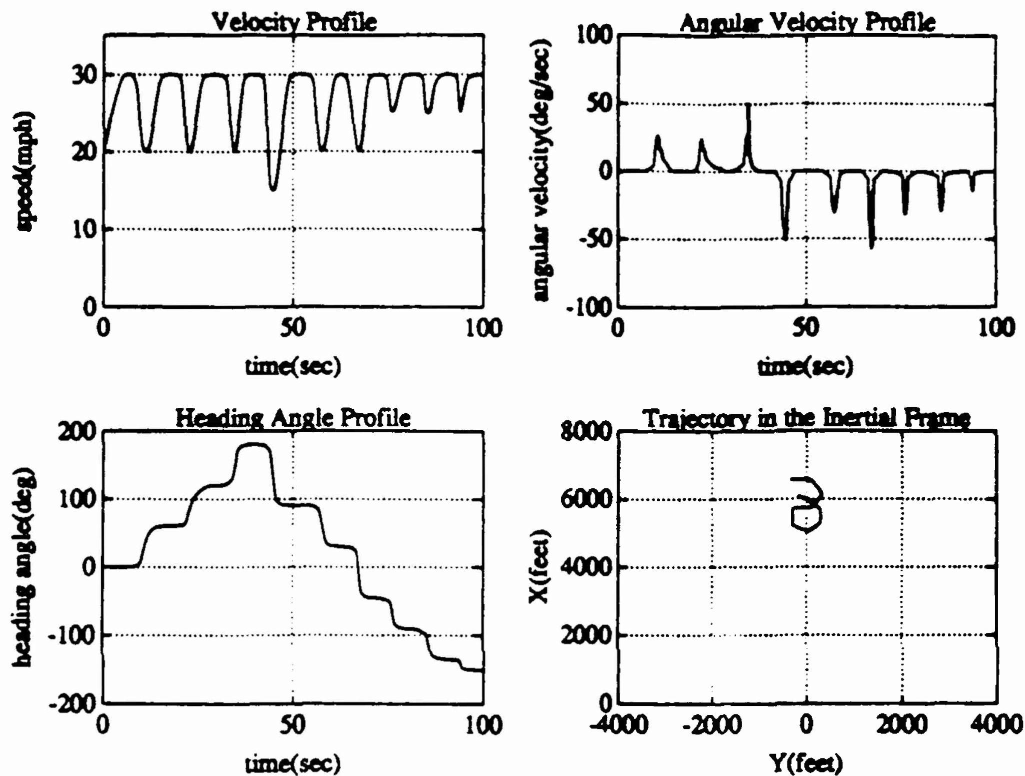


Figure 1.3: Trajectory data

different levels. From the point of view of information flow, a subsystem at a certain level has the information loop as shown in Figure 1.4. Such a testbed is simple enough to implement, yet still flexible enough for further development. With such a testbed, different modules or algorithms can be ***plugged in*** with minimum efforts. This allows the easy comparisons and selections of different algorithms, and it also makes it possible to quickly prototype a subsystem.

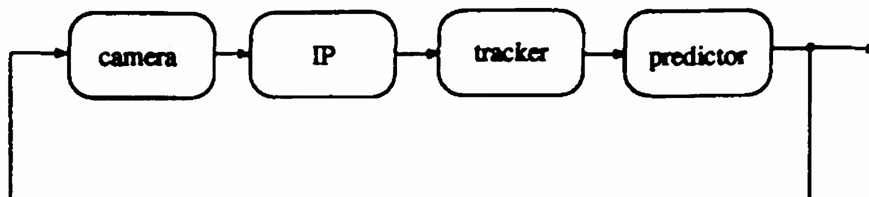


Figure 1.4: A generic subsystem

1.3.2 Non-real Time Simulation

Optical imaging system is one of the important components of the HiTert system. Computer graphics principles are utilized to generate the images for a simulated battle field scene. Real-time simulation of a battle field scene is tempting, but requires special hardware, which is prohibitively expensive. Also, segmenting images is very CPU-intensive and special hardware is also required if the real-time response is desired. Because of the unavailability of the special hardware due to its prohibitively high cost, it is decided to perform CPU-intensive work off-line, i.e. a sequence of images are *pre*generated and *pre*segmented and the results are stored on disk. The raw images as well as segmented images are made available, on demand, to the HiTert system during its operation.

The basic idea behind this approach is as follows. A *fixed* world-view "camera" is introduced for the purpose of generating a temporal sequence of world images. This "camera" is assumed to have sufficiently large field of view such that the target always moves within the field of view for the time period of interest. The tracking camera, which is the imagery acquiring system in HiTert, is able to *pane* commanded by the tracking system. If both the world-view camera and the tracking camera are located at the same position, if the world-view camera has a sufficiently large field of view, and if the tracking camera has sufficiently small panning angles, then the image

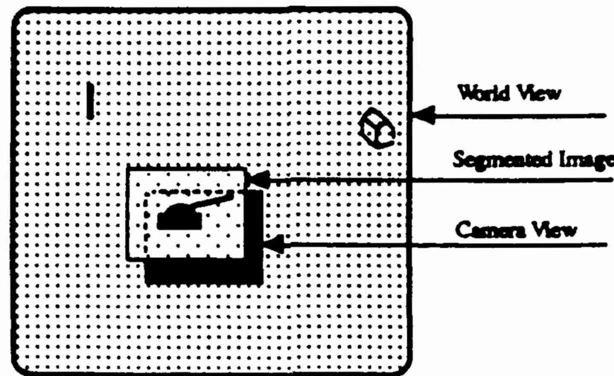


Figure 1.5: World and camera views

as seen by the tracking camera is just a subimage of the world-view image. This is illustrated in Figure 1.5.

The presegmentation is done on a subimage that is centered around the target. Portion of this subimage may overlap with the image of the tracking camera. It should be noted that if HiTert has a poor performance in operation, the tracking camera may be commanded to look at a wrong region. In this case, the presegmented image and the tracking camera image may have no overlapping at all. If HiTert's performance is very poor, the tracking camera's boresight may completely fall outside of the viewing angles of the world-view camera. Conversely, the symbiotic resonance of the tracking system and image processing system can yield a tight lock on target[2]. Therefore, with this non-real time simulation approach, in which the CPU-intensive jobs are preprocessed , HiTert system can be implemented on a general-purpose digital computer; and reasonably fast on-line responses can be achieved; yet the dynamic performance of the HiTert system can still be fully evaluated.

The presegmentation allows the implementation of sophisticated time-consuming image processing algorithms. This non-real time simulation ap-

proach also makes it possible to study tracking and image processing algorithms separately.

It should be pointed out that the world-view "camera" is *fictitious*. Its main purpose is to generate the world-view image database.

1.4 Development Environment

1.4.1 Hardware Architecture

The School of Aeronautics and Astronautics has clusters of Sun workstations ranging from Sun 3/50 to Sparc 1, running vendor-enhanced UNIX[1] operating system(SunOS 4.x). These workstations are part of the nodes on the Local Area Network, which is connected to the Engineering Computing Network(ECN). ECN consists of varieties of platforms, ranging from workstations to supermini computers, connected via ethernet.

The primary development work is done on a Sun workstation, with the *de facto* industry standard X Window System[2] Version 11, Release 4. The network environment as well as the network-transparency of X windowing system allows us to explore the distributed processing in the HiTert system. Also whenever possible, more dedicated computers are used. For example, the image database is generated on a Stardent 3000 machine, a supermini graphics computer.

1.4.2 Interprocess Communication(IPC)

On current UNIX systems, processes can communicate with one another via a variety of methods, including shared file pointers, signals, files, pipes, FIFO's, semaphores, messages, shared memory and Berkeley sockets[6, 7]. Shared memory provides the fastest IPC mechanism. However, the communicating

[1]UNIX is a registered trademark of AT&T Bell Laboratories.
[2]The X Window System is a trademark of MIT.

processes must physically reside on the same machine. Also, shared memory makes the communication interface complicated, while software modules are constantly evolving. BSD socket-based IPC mechanism has the same drawback in communication interface design as shared memory, although it allows communication among the processes running on different machines on the Internet. To simplify the communication interface, files are chosen as the IPC mechanism for the HiTert system at this stage. Files provides IPC for the processes running on the machines with the same Network File System. Also, there are standard C libraries on all systems supporting the C language[4, 5], and most people are familiar with I/O on files. Another important factor in choosing files as the IPC mechanism is the availability of the X11-based Cantata visual programming language coming with the Khoros system[8]. Cantata provides a graphical user interface for the UNIX processes communicating via files, thus simplifying our work in the graphical user interface design.

1.4.3 Cantata Visual Programming Language

One of the major components of the Khoros system is its Cantata visual programming language. Cantata provides a graphical user interface for the conventional command line options interface of UNIX programs. Graphically expressed and visually oriented, Cantata consists of following graphical elements: a *workspace*, *forms*, *glyphs*, and *connections*. To build a Cantata application, the user selects the desired programs(or *processing nodes*), places the corresponding glyphs on the workspace, and then interconnects the glyphs from upstream to downstream to indicate the data flow of processing. The built-in control constructs, as well as its expression parser and dynamic execution scheduler make Cantata behaves like a *visual shell*. Because of these features, Cantata is selected as a visual presentation tool for our UNIX command line interface programs.

Chapter 2

Installation Guide

In this chapter, some highlights of the installation procedures will be presented to reduce the installation efforts on the user's part. Currently, all the modules are implemented in the C Language, though some Fortran and Lisp modules may be included in the future. Efficiency and portability have been taken into account. All the programs have been tested on Sun3 and Sparc workstations.

2.1 Platform Requirements

A Sun workstation of 3/60 or later model is recommended but not mandatory. Other hardware supporting C and UNIX environment is just fine, although some additional porting efforts may be needed. However, in order for all the modules to run, the X window system(version 11, release 4) and the Khoros system must already be installed. We shall assume this to be true in the following.

The HiTert system takes about 10 mega bytes of disk space and need 8 MB main memory (RAM) to achieve a reasonable real-time response because of the large(900 $\times$ 900 pixels) images involved.

2.2 Unpack the Source

To unpack the tar file **hitert.tar.Z**, go to the directory in which the HiTert source tree is to be installed. Note that this directory does not have to be in the Khoros source tree. Then type

```
example% zcat hitert.tar.Z | tar xvf -
```

This should result in HiTert source directory **hitert**.

2.3 Installation Environment Variables

The environment variable **HITERT_HOME** must be set to the *full path* of the HiTert source directory. For example, if you are using csh-shell, type

```
csh-example% setenv HITERT_HOME /home/gus2/luj/hitert
```

If you're using ksh, bash, or sh shell, type

```
sh-example% HITERT_HOME=/home/gus2/luj/hitert
sh-example% export HITERT_HOME
```

Note in the the above examples, **/home/gus2/luj/hitert** should be modified to reflect the change of the full path name of the HiTert source directory. From now on we shall refer to this directory, following a UNIX shell's syntax, by symbol **$HITERT_HOME**.

To compile all the programs, you should also have the environment variable **KHOROS_HOME** set properly to the directory where the Khoros system is installed. This is necessary for linking with the Khoros libraries in compiling some programs. Section 3.2 has some information on how to set **KHOROS_HOME**. For further information on how to set this environment variable, please consult Khoros reference manual.

2.4 Compilation and Installation

After you have unpacked the HiTert source and set the environment variables, issue the following command while in **$HITERT_HOME** directory:

```
example% ./InstallMe
```

InstallMe is a shell script intended to automate the installation process. In the case such a global automated compiling procedure fails, you may want to compile the programs individually. To compile a program, you may need to modify the corresponding *Imakefile*[1], and then update *Makefile* by issuing commands:

```
example% rm -f Makefile
example% makemake  # Khoros program. Not xmkmf
example% make all
```

After the successful compilation of the programs, you need to copy or move the resulting executable programs to a directory. This directory should be included in your shell's search path by modifying the environment variable **PATH** as necessary. You may also want to install manual pages to the appropriate directory and modify the **man**(1)'s environment variable **MANPATH** so that HiTert's UNIX manual page directory is in **man**(1)'s search path.

[1]The *Imakefile* is Khoros-flavored and has the difference with the *Imakefile* of X11R4 from MIT.

Chapter 3

Getting Started

3.1 The X Window System

It is assumed that you have some basic working experience in a windowing environment, and that you know how to login a workstation with a bitmap display and start the X window system. This procedure varies from the site to site. Usually, there is a site-customized shell script program available to help the user to invoke `xinit(1)` and a set of applications such as **`xterm(1)`**, **`xclock(1)`** and etc.

3.2 The Khoros System

It is also assumed that now both Khoros and HiTert system have been successfully installed and you have some exposure to **`cantata(1)`**, a visual language environment in Khoros system. If you know how to start Khoros programs from your home directory, you may skip this section and go to Section 3.3.

To access Khoros programs, you need to set the environment variable **KHOROS_HOME** to the full path of the directory in which Khoros is installed(this directory will be referred to by the symbol **$KHOROS_HOME**). In **$KHOROS_HOME**,

there should be a file named `.khoros_env` for csh users. You may need to add the following lines in your `.login` or `.cshrc` file:

```
setenv KHOROS_HOME /home/gus3/khaos
source $KHOROS_HOME/.khoros_env
set path = ($KHOROS_HOME/bin $path)
```

If you are using other shells such as **ksh**, **bash** or **sh**, you need to modify `$KHOROS_HOME/.khoros_env`. In your home directory, create a file `.khoros_env.sh` which contains the following lines

```
KHOROS_HOME=/home/gus3/khaos        # edit this as necessary
KHOROS_MAIL=$USER
KHOROS_LOG=$HOME/khoros_cmd.log
KHOROS_VERBOSE=no
KHOROS_CACHE_SIZE=4194304
KHOROS_CACHE=no
TMPDIR=${TMPDIR-/tmp}
export KHOROS_HOME KHOROS_MAIL KHOROS_LOG KHOROS_VERBOSE \
       KHOROS_CACHE_SIZE KHOROS_CACHE TMPDIR
```

Then in your `.profile`, add the following statements:

```
. $HOME/.khoros_env.sh
PATH=$KHOROS_HOME/bin:$PATH; export PATH
```

3.3 Environment Variables for the HiTert System

First, you need to set the shell environment variable `HITERT_HOME` to the directory in which the `hitert` source tree resides. For example, if you are using csh, just type the following to the shell prompt:

```
example% setenv HITERT_HOME  /home/gus2/luj/aro/hitert
```

If you are a ksh, bash, sh user, enter the following at the shell prompt:

```
example% HITERT_HOME=/home/gus2/luj/aro/hitert
example% export HITERT_HOME
```

In the above examples, **/home/gus2/luj/aro/hitert** need to be modified reflect the changes to the actual environment. If you want to avoid doing this every time after you login, you may add the corresponding statements to your **.login** or **.cshrc** file if you are a csh user, or to your **.profile** file if you are a ksh, bash or sh user.

You also need to check to see if the shell environment variable **PATH** includes the directory in which the HiTert programs are installed. If not, you need to modify **PATH** such that the shell's search path includes this directory. For example, if the HiTert programs are installed in the directory **$HOME/bin**, you may modify your shell **PATH** variable in the following way:

```
# for csh user
setenv PATH $HOME/bin:$PATH
```

or

```
# for ksh, bash, sh user
PATH=$HOME/bin:$PATH; export PATH
```

You may also want to set your environment variable **MANPATH** so that it includes the directory in which the manual pages for the HiTert system are installed. For example, if the manual pages for the HiTert system are installed in the directory **$HOME/man/man1**, you may set **MANPATH** variable this way:

```
# for csh user
setenv MANPATH /usr/man:$HOME/man
```

or

```
# for ksh, bash, sh user
MANPATH=/usr/man:$HOME/man; export MANPATH
```

To see if you have correctly set the environment variables for the HiTert system, just type

```
example% hitert -help
```

If you are able to see the help message from `hitert`, congratulations. If your shell is unable to find the executable program `hitert`, you need to check if `PATH` indeed includes the directory in which the HiTert executable programs reside. If `hitert` reports an error, you need to make changes accordingly.

Similarly, if `MANPATH` has been set correctly, you should be able to see the on-line manual pages following this example:

```
example% man hitert
```

3.4 Command Line Interface

The HiTert programs can be invoked as an individual program by typing the command or program name with command line options. For example, if the file `prediction.dat` contains prediction data from the α-β-γ tracker/predictor, and if the file `exact.dat` contains the reference exact data for the target positions, you can invoke `errAnaly` to the perform error analysis by typing:

```
example% errAnaly -pred prediction.dat -exact exact.dat \
-showDart 0 -outSpec spec -outErr error.dat
```

errAnaly will compute the errors in each inertial x, y, z component, and store the error data in the file **error.dat**. File **spec** will contain the statistics about the errors.

This is the conventional way to invoke a UNIX program, familiar to most UNIX users. In this approach, a *shell* is acting as the *interface* between the user and the *kernel*, even though you may not be aware of this. Except for **xhitert**, all other HiTert programs can be executed by using the command line interface from any terminal without getting into X windowing environment.

If you want to get some simple help from a program, just type

```
example% errAnaly -help
```

This yields

```
Usage: errAnaly
        -pred <predfile>          -- data file from the predictor
        -exact <exactfile>        -- exact data file for comparison.
        [-tbf <tbf>]              -- track-before-fire time.
        [-tburst <tburs>]         -- burst interval.
        [-showDart <1 or 0>]      -- computes the dartboard errors.
        -outSpec <filename>       -- output spec filename.
        [-outErr <filename>]      -- prediction err data filename.
        [-help]                   -- print out this help message.
```

If your manual pages are properly installed and the environment variable **MANPATH** is set properly, you may get a more detailed description by using UNIX **man**(1) command. For example,

```
example% man errAnaly
```

UNIX stdin(standard input) can be used for any one input *file* by specifying "-" for the file name. Similarly, stdout(standard output) can be used for any one output *file* by specifying "-" for the file name. As a result, the output of one program can be *piped* to the input of another program. For example,

```
example% predict -i tracker.dat -o - -tp 2 | errAnaly -pred - \
 -exact exact.dat -showDart 0  -outSpec spec -outErr error.dat
```

invokes the `predict`, which performs prediction based on the tracker output(state estimates) `tracker.dat`. The prediction results are piped, as the input, to the program `errAnaly`, which does the error analysis.

Experienced users may notice that there are some differences in the handling of command line options between the HiTert programs and Khoros programs. In the HiTert programs, the Khoros program `ghostwriter` has *not* been used to generate the code for handling command line options; instead a much more efficient scheme is utilized to parse the command line options. Standard Khoros flags "[-U] [-P] [-A [file]] [-a [file]]" are not supported by the HiTert programs, since we ourselves found little use of them.

3.5 Cantata Visual Language Interface

To access cantata visual language interface for the HiTert system, you need first start up the X window system from your workstation. Then from an `xterm(1)`(an X-based terminal emulator) type

```
example% hitert
```

This should bring up the cantata visual language interface with the customized *forms* for the HiTert system. All the HiTert programs can be accessed from the pull-down menu "HiTERT" on the *master form*. On-line

documentation for the corresponding programs can be invoked by pressing the "Help" button on the *pane*.

Within the the cantata visual language environment, accessing the HiTert programs is the same as accessing other Khoros programs. You should have little problem in using HiTert programs if you are familiar with `cantata`. Please consult Khoros tutorial or reference manual for the assistance in using `cantata`.

Novice users may find it helpful to use `hitert2`. This program brings up the cantata visual language interface as well as a preassembled and stored *workspace* as shown in Figure 4.1. This should serve as a good example as how to assemble individual HiTert programs together to perform a team work. It should also serve as an extensible foundation upon which a user can configure HiTert's behavior by modifying parameters.

Chapter 4

User's Guide

The modules in HiTert System will be explained in the following sections. Chapter 5 is dedicated to the discussion on the generation of the image database. Interested reader may read that chapter before this one.

4.1 Overview of the Software Modules

Figure 4.1 shows the primary modules comprising a generic subsystem in HiTert: *dataControl*, *w_camera*, *t_camera*, *ipc*, *tracker*, *predictor*, *errAnaly* and *xhitert*. *dataControl* module controls starting and terminating time, and etc. for the HiTert system. *w_camera* is the world-view camera, responsible for retrieving the world-view images. *t_camera* simulates the servomechanism for the tracking camera, receiving command signals from the tracker and pointing the tracking camera to the appropriate direction. *ipc* is the centroid image processor, responsible for retrieving presegmented images. *tracker* does the state estimation. *predictor* does the prediction based on the state estimation results from the tracker. *errAnaly* performs the error analysis for the overall tracking system. *xhitert* is the instrumentation module, responsible for the displaying the world-view image, tracking camera image, segmented image, prediction results, error analysis results, and etc.

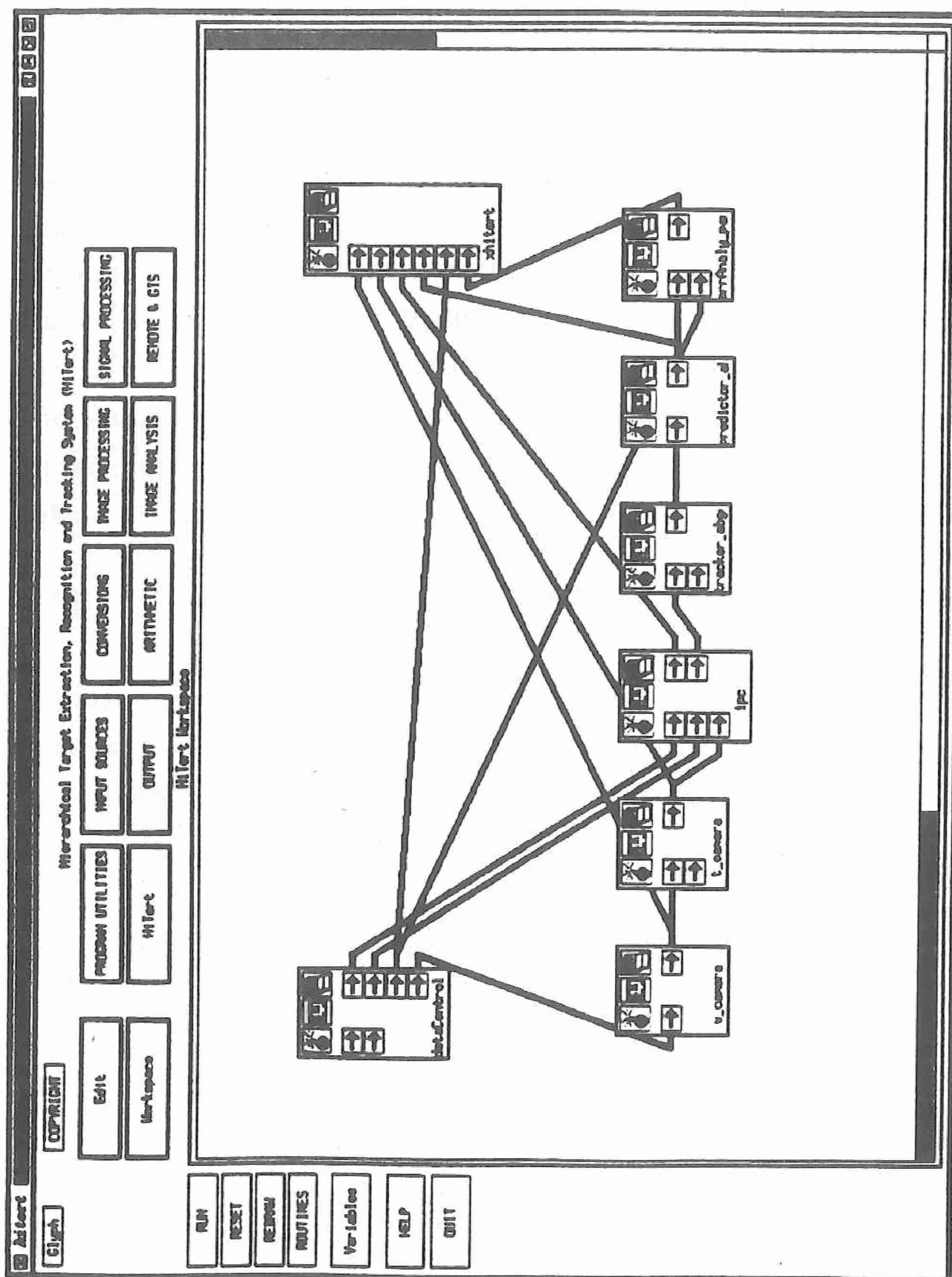


Figure 4.1: Software modules and data flows

There are two functional modes for the HiTert system: the *batch mode* and *loop mode*. In the *batch mode*, each module operates on *all* the data for the whole time history of the interest and outputs *all* relevant data, and then it stops execution[1]. Whereas in the *loop mode*, each module operates on the data one time step at a time and will resume the execution when next data frame is available.

Right now the HiTert system only functions in the *batch mode*. This is done primarily because files have been chosen as the IPC mechanism to fit into the cantata visual environment. Using intermediate files as the IPC mechanism is most suitable for the processes communicating and executing in *serial* or in *batch mode*. It is possible to implement the *loop mode* on top of `cantata`.However, additional mechanisms such as file locking techniques must be introduced to synchronize and safeguard the communication of processes executing in *loop*. Also in the *loop* mode, each time intermediate temporary communication files need to be recreated and each module need to be restarted by `cantata` through `fork(2)`. This would incur much overhead[2]. Another hindrance for implementing the *loop mode* is that more powerful computer hardware[3] is needed for displaying images in succession.

It should be pointed out that in order to avoid creating or passing multiple intermediate *data* files, *spec files* are used frequently in the HiTert system. The *spec files* are introduced as the information *carriers*. Such a *spec file* differs from the ordinary *data* file in that the main purpose of the *spec file* is to bundle all the necessary *pointers* to other *data* files together in a single file, while all other *data* files retain their own formats and integrity. For example,

[1] In *batch mode*, `w_camera` only retrieves one frame from the image database, and `xhitert` only display one frame of image to save the time and computer resources. This is important particularly for the monochrome display for which `xhitert` has to do CPU-intensive dithering to the the color image.

[2] More appropriate IPC mechanism for the *loop mode* is to use BSD sockets, which however is not supported by cantata visual language environment.

[3] For example, a Sparc-station with 16 MB RAM and 8-bit color display is required.

w_camera creates a world view image *spec* file. This file contains the *sequence number* of the original input image and the *path* of the world-view image. By accessing this *spec* file, other processes can not only access the image data by opening the file with the specified *path* but also know the corresponding time by appropriately interpreting the *sequence number* in the *spec* file.

When multiple *data* files need to be accessed by the communicating processes, using the *spec* file is particularly convenient, since the *spec* file encapsulates all the information together as a *single* file. Introducing *spec* files not only greatly simplifies and cleans up the communication interface design, but also have the potential to greatly reduce the communication overhead, since now multiple processes can access the same *data* files without having to actually pass large data files around.

4.2 Data Control Module

4.2.1 Introduction

In the *batch mode*, each HiTert program operates on *all* the data contained in the input file(s). Very often, however, one wants the HiTert system to run over a contiguous portion of the data in the file. In order to avoid the unnecessary complexity of other HiTert programs, `dataControl` is introduced as a stand-alone *data control* module to isolate the problem. `dataControl` is responsible for selecting a portion of time history over which the HiTert system operates. It takes the description file for the world-view and segmented image database as input, and outputs the selected segment description files and the corresponding exact trajectory data file. It also outputs a *spec* file which contains the sequence numbers and the data records from the description files at the starting and terminating time. This way, each HiTert program still operates on *all* the data contained in the input file(s), yet the HiTert system is able to operate only over the specified time period as controlled by

dataControl.

4.2.2 Command Line Options

Synopsis

dataControl -imgInfo *imgInfoFile* **-segInfo** *segInfoFile* **-t0** *start-time* **-te** *end-time* **-oimgInfo** *out-imgInfoFile* **-osegInfo** *out-segInfoFile* **-traj** *out-exact-traj* **-spec** *out-spec* **[-help]**

Options

-imgInfo *imgInfoFile* Required argument. *imgInfoFile* is the description file for the world-view image database of the whole time history. This file consists of the data records, with each data record containing items (see Section 5.5.3):

- time at which the image is to be generated. This is identified by a " *keyword=value*" pair, with the keyword being **time**.
- the image identifier which maps to the corresponding image file. This is identified by a " *keyword=value*" pair, with the keyword being **image**.
- the image width in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **width**.
- the image height in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **height**.
- the tank's state at that instant which includes

$$t \quad x_I \quad y_I \quad z_I \quad \dot{x}_I \quad \dot{y}_I \quad \dot{z}_I \quad R \quad S \quad T \quad \dot{R} \quad \dot{S} \quad \dot{T}$$

- the camera's state which includes

$$x_{cI} \quad y_{cI} \quad z_{cI} \qquad \dot{x}_{cI} \quad \dot{y}_{cI} \quad \dot{z}_{cI} \; R_c \; S_c \; T_c \; \dot{R}_c \; \dot{S}_c \; \dot{T}_c$$

fov $\dot{fov}$ $hither$ yon $\dot{yon}$

where x_{cI}, y_{cI}, z_{cI}, $\dot{x_{cI}}$, $\dot{y_{cI}}$, $\dot{z_{cI}}$ denote the position and velocity components of the camera at the instant in global inertial coordinate system. R_c, S_c, T_c, $\dot{R_c}$, $\dot{S_c}$, $\dot{T_c}$ refer to the yaw, pitch , roll angles and their rates. fov and $\dot{fov}$ refer to the field of view(in degrees) and its time rate of change(in degrees per second), respectively; $hither$, $\dot{hither}$ denote the position of the front clipping plane and its rate in the camera coordinate system; yon, $\dot{yon}$ position of the back clipping plane and its rate in the camera coordinate system. The camera coordinate system is such that its initial orientation aligns with the inertial coordinate system, and the camera is always looking towards the negative z direction of the body-fixed right-handed coordinate system.

- range of the target to camera and range rate.

-segInfo *segInfoFile* Required argument. *segInfoFile* is the description file for the segmented image database of the whole time history. The file is composed of the data records of the following form:

```
time=2  image=s1.seg0020
        seg_ulx=90  seg_uly=316  seg_width=256  seg_height=256
        tcx=221.809  tcy=444.128  tvx=46.5792  tvy=5.43854
        out_ulx=-1  out_uly=-1  out_width=256  out_height=256
        size=46.8957  scale=2  mag=5.45893  proj=2
        roll=0  pitch=0  yaw=1.5609
        estroll=0  estpitch=0  estyaw=1.5708
```

-t0 *start-time* Required argument. *start-time* specifies the time at which the tracking system should start to run. This *floating* number has the

unit of ***seconds***.

Example: -t0 10

-te *end-time* Required argument. ***end-time*** specifies the time at which the tracking system should terminate. This *floating* number has the unit of *seconds*. By specifying ***start-time*** and ***end-time***, one can ask system to run over the selected segment of the trajectory of interest instead of the whole time history.

Example: -te 50

-oimgInfo *out-imgInfoFile* Required argument. ***out-imgInfoFile*** is the selected segment of the description file for the world-view image database. It has the same format as the input file *imgInfoFile*.

-osegInfo *out-segInfoFile* Required argument. *out-segInfoFile* is the selected segment of the description file for the segmented image database. It has the same format as the input file *segInfoFile*.

-traj *out-exact-traj* Required argument. *out-exact-traj* is the file containing the selected segment of *exact* trajectory. This file has the following format:

$$\begin{array}{cccc} t & x(t) & y(t) & z(t) \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

where t is the *current* time, $x(t)$, $y(t)$ and $z(t)$ the *exact* target position coordinates at t. These *floating* numbers t, $x(t)$, $y(t)$ and $z(t)$ have the following units, respectively:

seconds *meters* *meters* *meters*

-spec *out-spec* Required argument. *out-spec* is the output *spec* file containing information corresponding to the starting and terminating points.

Each of the two data records in the file *out-spec* consists of the following items in order: A sequence number, a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

`-help` Optional argument. When this argument is specified, **`dataControl`** prints out a brief help message and exits gracefully.

4.2.3 I/O File Specification

-imgInfo *imgInfoFile*

The input image the description file *imgInfoFile* consists of the data records, with each data record containing items (see Section 5.5.3):

- time at which the image is to be generated. This is identified by a " *keyword=value*" pair, with the keyword being `time`.
- the image identifier which maps to the corresponding image file. This is identified by a " *keyword=value*" pair, with the keyword being **`image`**.
- the image width in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **`width`**.
- the image height in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **`height`**.
- the tank's state at that instant which includes

$$t \quad x_I \quad y_I \quad z_I \quad \dot{x}_I \quad \dot{y}_I \quad \dot{z}_I \quad R \quad S \quad T \quad \dot{R} \quad \dot{S} \quad \dot{T}$$

- the camera's state which includes

x_{cI} y_{cI} z_{cI} $\dot{x_{cI}}$ $\dot{y_{cI}}$ $\dot{z_{cI}}$ R_c S_c T_c $\dot{R}_c$ $\dot{S}_c$ $\dot{T}_c$
fov $\dot{fov}$ $hither$ yon $\dot{yon}$

where x_{cI}, y_{cI}, z_{cI}, $\dot{x_{cI}}$, $\dot{y_{cI}}$, $\dot{z_{cI}}$ denote the position and velocity components of the camera at the instant in global inertial coordinate system. R_c, S_c, T_c, $\dot{R}_c$, $\dot{S}_c$, $\dot{T}_c$ refer to the yaw, pitch , roll angles and their rates. fov and $\dot{fov}$ refer to the field of view(in degrees) and its time rate of change(in degrees per second), respectively; $hither$, $\dot{hither}$ denote the position of the front clipping plane and its rate in the camera coordinate system; yon, $\dot{yon}$ position of the back clipping plane and its rate in the camera coordinate system. The camera coordinate system is such that its initial orientation aligns with the inertial coordinate system, and the camera is always looking towards the negative z direction of the body-fixed right-handed coordinate system.

- range of the target to camera and range rate.

The data records are separated by one or more *white space characters*(blanks, tabs, newlines). The data items in each data record are also separated by one or more *white space characters*.

A Typical abridged *imgInfoFile* having two data records may look as follows:

```
time=2.000000  image=s1.img0020  width=900  height=900
         2000.100000 -79.946000 -1.400000 0.109320 11.046000 0.000000
         1.560900 0.000000 0.000000 0.005000 0.000000 0.000000
         0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
         0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
         9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
         2001.703368 -0.331933
```

```
time=2.100000  image=s1.img0021  width=900  height=900
        2000.200000 -78.837000 -1.400000 0.109860 11.141000 0.000000
        1.560900 0.000000 0.000000 0.035000 0.000000 0.000000
        0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
        0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
        9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
        2001.759304 -0.329001
```

-segInfo *segInfoFile* [1]

The input file *segInfoFile* contains a brief description of the parameters active at the time each image is segmented. Each segmented image has an associated record of the form:

```
time=0.5  image=s1.seg0005
seg_ulx=40  seg_uly=40  seg_width=256  seg_height=256
tcx=130.0  tcy=130.0  tvx=60.0  tvy=80.0
out_ulx=4  out_uly=4  out_width=256  out_height=256
size=100  scale=2.0  mag=2.56  proj=5
roll=0  pitch=0  yaw=1.5609
estroll=0  estpitch=0  estyaw=1.5708
```

The information in this file pertains strictly to the segmented image. Information regarding the pre-segmentation image, for example its dimensions, is not recorded in this file; instead it is in "s1.info".

Each segmented image has a sequence number formed from the last 4 characters of its name; **time** is this sequence number divided by 10.

The entire input image is not segmented; only pixels falling in a segmentation window, measuring **seg_height** by **seg_width**, centered at the last estimated position of the target are considered. The upper left hand corner

[1] By Craig Codrington

of this window is situated at pixel coordinates (**seg_ulx**,**seg_uly**) of the input image. The new target position, relative to the input image, is taken to be the centroid (**tcx**,**tcy**) of the pixels classified as target, and an indication of its size is given by the variances **tvx** and **tvy** of these pixels in the x and y directions, respectively. In fact,

$$\mathbf{size} = \sqrt{\mathbf{tvx}^2 + \mathbf{tvy}^2} \tag{4.1}$$

The segmentation is mapped to an output window, measuring **out_height** by **out_width**, and centered at the new position of the target in the segmentation window. The upper left hand corner of this window is situated at pixel coordinates (**out_ulx**,**out_uly**) of the input image. The contents of the output window are what is written to the segmented image file **image**.

This is not the whole story, however, for there are 5 possible ways to map the segmentation window to the output window. The particular mapping chosen is given by **proj**. Some of these mappings (one to many, many to one, and fuzzy many to one) attempt to scale the target to a consistent size in the output window. For these mappings, the **scale** gives the fraction of the output window that corresponds to one **size** unit in the segmentation window. For such mappings, the output window is virtual – it does not correspond to any particular location in the input image, so **out_ulx** and **out_uly** are both set to -1. For scaled mappings, the magnification **mag** is defined as

$$\frac{\left(\frac{\mathbf{out_height}+\mathbf{out_width}}{2}\right) * \mathbf{scale}}{2 * \mathbf{size}} \tag{4.2}$$

For other mappings, **mag** is set to 1. A brief description of each mapping is given below.

one to one (**proj** = 1) : The dimensions of the segmentation and output windows are ignored; the entire input image is segmented and the complete segmentation is written to **image**.

one to many (**proj** = 2) : For each pixel O in the output window, find the pixel S in the segmentation window which is mapped to it, taking account of **scale** and **size**, and set the class of O to the class of S.

many to one (**proj** = 3) : Initialize all pixels in the output window to non-target. For each target pixel S in the segmentation window, find the pixel O it is mapped to in the output window, taking account of **scale** and **size**, and set O to target.

fuzzy many to one (**proj** = 4) : This mapping is like the previous one except that *all* segmentation window pixels are mapped to the output window; the class of an output window pixel is then the class of the segmentation window pixel mapped to it. If more than one segmentation window pixel is mapped to a single output window pixel O, the class to which the largest number of such segmentation window pixels belong becomes the class of O.

orthogonal (**proj** = 5) : For each pixel O in the output window, find the pixel S in the segmentation window which is mapped to it, without scaling, and set the class of O to the class of S.

The fields **roll**, **pitch**, and **yaw** give the target's true orientation, as indicated in "s1.info", while **estroll**, **estpitch**, and **estyaw** give it's estimated orientation, derived from it's computed moments by indexing into "momentfile" using a nearest neighbor strategy.

-oimgInfo *out-imgInfoFile*

out-imgInfoFile has the same data format as *segInfoFile*.

-osegInfo *out-segInfoFile*

out-segInfoFile has the same data format as *segInfoFile*.

-traj *out-exact-traj*

The *data records* making up *out-exact-traj* are separated by one or more mixed *white space characters*. Within each *data record*, there are three *floating* numbers t, $x(t)$, $y(t)$ and $z(t)$ which are also separated by one or more mixed *white space characters*. t is the *current* time and has the unit of *seconds*; $x(t)$, $y(t)$ and $z(t)$ the *exact* target position coordinates at t, and each have the unit of *meters*.

A typical *exactfile* having 5 *data records* may look as follows:

```
2.000000 2000.100000 -79.946000 -1.400000
2.100000 2000.200000 -78.837000 -1.400000
2.200000 2000.200000 -77.718000 -1.400000
2.300000 2000.200000 -76.590000 -1.400000
2.400000 2000.200000 -75.453000 -1.400000
```

out-exact-traj is an *output* file from `dataControl` and has the same *format* as the input file *exact-traj*.

-spec *out-spec*

out-spec is the output *spec* file containing information corresponding to the starting and terminating points. Each of the two data records in the file *out-spec* consists of the following items in order: A sequence number, a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*). All data fields are separated by one or more mixed white space characters. A typical output spec file *out-spec* may look as follows:

```
0
time=0.000000  image=s1.img0000  width=900  height=900
        2000.000000 -100.000000 -1.400000 0.000000 8.938900 0.000000
```

```
         1.570800 0.000000 0.000000 -0.645000 0.000000 0.000000
         0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
         0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
         9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
         2002.504682 -0.446386
time= 0.000000 image= s1.seg0000
        seg_ulx= 0 seg_uly= 0 seg_width= 256 seg_height= 256
        tcx= 164.716000 tcy= 444.135000 tvx= 45.836500 tvy= 5.356840
        out_ulx= -1 out_uly= -1 out_width= 256 out_height= 256
        size= 46.148500 scale= 2.000000 mag= 5.547310 proj= 2
        roll= 0.000000 pitch= 0.000000 yaw= 1.570800
        estroll= 0.000000 estpitch= 0.000000 estyaw= 1.570800
500
time=50.000000  image=s1.img0500  width=900  height=900
         1687.500000 -90.137000 -1.400000 -13.408000 -0.067162 0.000000
         3.146600 0.000000 0.000000 -0.200000 0.000000 0.000000
         0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
         0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
         9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
         1689.912994 -13.385272
time= 50.000000 image= s1.seg0500
        seg_ulx= 16 seg_uly= 318 seg_width= 256 seg_height= 256
        tcx= 144.383000 tcy= 446.439000 tvx= 13.450800 tvy= 6.849180
        out_ulx= -1 out_uly= -1 out_width= 256 out_height= 256
        size= 15.094200 scale= 2.000000 mag= 16.960100 proj= 2
        roll= 0.000000 pitch= 0.000000 yaw= 3.146600
        estroll= 0.000000 estpitch= 0.000000 estyaw= 3.154900
```

4.3 World-view Camera

4.3.1 Introduction

v_camera is the program for the *world-view camera* module. It is responsible for retrieving an image based on the user specifications on the directory and the format of a file name string. It uncompresses the image file as required by the user. It should be pointed out that writing large uncompressed images files(approximately $1MB$ per world-view image) incurs a great deal of disk I/O overhead. Because of this, Khoros routine readimage() for reading viff image has been extended so that the extended routine can automatically uncompress the image in cache memory and pipe the uncompressed data to image reading process. This extended routine is employed by other programs, which makes image uncompressing unnecessary.

Very often a set of related data files are stored in the same directory. Besides sharing the same directory, these files have common *prefix*, varying *index* or *sequence numbers* and common *suffix*. In other words, the *paths* of the files have the following format:

directory/PrefixSequence-numberSuffix

No assumption has been made as to how the prefix, sequence and suffix are separated. If they are indeed separated by some character such as "." or "-", the characters must be explicitly specified either in the prefix or suffix part. Take `my-img-011.Z` for example. The prefix is `my-img-`, the sequence number `11`, the suffix `.Z`

v_camera is ideal for retrieving a set of related files: all the desired target input files must have the same prefix and suffix(if any) in addition to having same directory. Only the sequence or index numbers are allowed to vary. The sequence or index number comes from the input spec file, which is the

output spec file of the `dataControl` module.

4.3.2 Command Line Options

Synopsis

w_camera [-dir *directory*] [-prefix *prefix*] [**-num_width** *num-width*] [**-suffix** *suffix*] -inSpec *inSpec* -outSpec *outSpec* [-img *imgfile*] [-uc <1 or 0>] [-help]

Options

-dir *directory* Optional argument. *directory* specifies the directory in which a desired input file resides.

Default: the current directory.
Example: -dir $HITERT_HOME/data/images

-prefix *prefix* Optional argument. It specifies the string that *precedes* the sequence or index number in a file name.

Default: null
Example: -prefix s1.img

-num_width *num-width* Optional argument. *num-width* specifies the width that the numeric sequence number takes. If *num-width* is greater than the number of the *digits* that *sequence-number*[1] has, *sequence-number* will be *prepended* with 0's(zeros) in formating the file name string. For example,

```
example% w_camera -inSpec inSpec -outSpec outSpec -num_width 4
```

[1]specified via -inSpec *inSpec*

If *sequence-number* is 123, **w_camera** will try to find the file with the name 0123 in the current working directory. If *num-width* is less than the number of the *digits* that *sequence-number* has, *num-width* has no effect. For example,

```
example% w_camera -inSpec inSpec -outSpec -num 111 -num_width 2
example% w_camera -inSpec inSpec -outSpec -num 111
```

In these two examples, if *sequence-number* is 123, **w_camera** will try to find the file with the name 123 in the current working directory.

Default: unspecified(as long as effective *sequence-number* is)
Example: **-num_width 4**

-suffix *suffix* Optional argument. This option specifies the string in the file name that *follows* the sequence digit string. If *suffix* is ".Z", the input file is assumed to compressed by **compress(1)**.

Default: null
Example: **-suffix** .dat

-inSpec *inSpec* Required argument. *inSpec* specifies the input *specification* file. *inSpec* should be the output spec file from the **dataControl** module. It contains the following data items in order: A sequence number, a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

-outSpec *outSpec* Required argument. *outSpec* specifies the output *specification* file. It is almost the same as the input *inSpec*, except that a filename is appended to the sequence number in the file. More specifically, *outSpec* contains the following data items in order: a sequence number,

a image filename(full path), a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

`-img` *imgfile* Optional argument. *imgfile* specifies the file to contain the same data as those contained in the *input file*, which is specified via *directory*, *prefix*, *sequence-number*, *sequence-number*, *suffix*. If this option is not specified, the default path name for *imgfile* is `/tmp/w_cam.$USER`, where `$USER` is the login name of the user.

Default: `/tmp/w_cam.$USER`
Example: `-img ~luj/tmp/w_image`

`-uc` 1 *or* 0 Optional argument. This switch has effect only if the input image file has the suffix ".Z". If the input image file is compressed and `-uc 1` is specified, the image image will be uncompressed by using `uncompress(1)` to produce the output image file. If the input image file is compressed but `-uc 0` is specified, the image will not be uncompressed and the output image will be tagged with ".Z" to indicate that the file is still compressed.

Default: 1
Example: `-uc 0`

`-help` Optional argument. When this argument is specified, `w_camera` prints out a brief help message and exits gracefully.

A complete example may look as follows:

```
example% w_camera -inSpec inSpec -outSpec ~luj/tmp/outSpec \
         -dir $HITERT_HOME/data/images \
         -prefix s1.img -num_width 4 -suffix .Z \
         -img ~luj/tmp/w_image -uc 0
```

This instructs **w_camera** to read the input spec file **inSpec** and to find out the input image file **$HITERT_HOME/data/images/s1.img0010.Z** and store the input image as the file **~luj/tmp/w_image**. Since the input file name has the trailing string .Z and -uc 0 is specified, the output images will be **~luj/tmp/w_image**.Z and **~luj/tmp/w_image1**.Z, provided that the **inSpec** has two data records.

4.3.3 I/O File Specification

In this section, the format of input and output *files* are explicitly specified. Any other programs wishing to communicate with **w_camera** via files must follow the *protocols* or formats specified herein.

Input Data File

The input *data* file as specified by *directory*, *prefix*, *sequence-number*, *num-width* and *suffix* can be in any format, no interpretation is done on the content of this file. However, if the file name as specified above has the trailing suffix ".Z", the input data file is considered to have been compressed using **compress**(1).

-inSpec *inSpec*

This input spec file is the same as the output spec file of the **dataControl** module. See Section 4.2.3.

-outSpec *outSpec*

It is almost the same as the input *inSpec*, except that a filename is appended to the sequence number in the file. More specifically, *outSpec* contains the following data items in order: a sequence number, a image filename(full path), a data record from the image description file at *start-time*(or *end-time*), a

data record from the description file for the segmented image database at *start-time*(or *end-time*). All data fields are separated by one or more *white space characters*. A typical *outSpec* file may look as follows:

```
0 /tmp/w_img.luj
time=0.000000  image=s1.img0000  width=900  height=900
         2000.000000 -100.000000 -1.400000 0.000000 8.938900 0.000000
         1.570800 0.000000 0.000000 -0.645000 0.000000 0.000000
         0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
         0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
         9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
         2002.504682 -0.446386
time= 0.000000 image= s1.seg0000
        seg_ulx= 0 seg_uly= 0 seg_width= 256 seg_height= 256
        tcx= 164.716000 tcy= 444.135000 tvx= 45.836500 tvy= 5.356840
        out_ulx= -1 out_uly= -1 out_width= 256 out_height= 256
        size= 46.148500 scale= 2.000000 mag= 5.547310 proj= 2
        roll= 0.000000 pitch= 0.000000 yaw= 1.570800
        estroll= 0.000000 estpitch= 0.000000 estyaw= 1.570800
500 /tmp/w_img.luj1
time=50.000000  image=s1.img0500  width=900  height=900
         1687.500000 -90.137000 -1.400000 -13.408000 -0.067162 0.000000
         3.146600 0.000000 0.000000 -0.200000 0.000000 0.000000
         0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
         0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
         9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
         1689.912994 -13.385272
time= 50.000000 image= s1.seg0500
        seg_ulx= 16 seg_uly= 318 seg_width= 256 seg_height= 256
        tcx= 144.383000 tcy= 446.439000 tvx= 13.450800 tvy= 6.849180
```

```
out_ulx= -1 out_uly= -1 out_width= 256 out_height= 256
size= 15.094200 scale= 2.000000 mag= 16.960100 proj= 2
roll= 0.000000 pitch= 0.000000 yaw= 3.146600
estroll= 0.000000 estpitch= 0.000000 estyaw= 3.154900
```

-img *imgfile*

imgfile is the actual output image ***data*** file. *imgfile* specifies the file to contain the same data as those contained in the ***input file***, which is specified via ***directory***, ***prefix***, ***sequence-number***, ***sequence-number***, ***suffix***, with the following exception: if the specified input file has the trailing string "`.Z`" and `-uc 1` is specified, then it is assumed that the input file has been compressed by using `compress(1)`, and `w_camera` will filter the input file through `uncompress(1)` and store the results as *imgfile*.

4.4 Tracking Camera

4.4.1 Introduction

Conceptually, the tracking camera takes as its input the world view image and the control signals from the tracker, and then finds out the subimage from the world view image. By doing this, `t_camera` effectively mimics a tracking camera. Models of the dynamics of the camera servomechanism may be incorporated.

However, in the *batch mode*, each individual program executes once and then dies or exits. There is no mechanism for the tracking camera to get any actual dynamic control signals from the tracker, since `track` executes *after* `t_camera`. Therefore, in the *batch mode*, the "control signals" are prespecified by the user, and this information is passed to the display module so that the viewing box of the tracking camera can be displayed in the world view image. The main purpose for having such a rather dumb tracking

camera in the batch mode is that the HiTert system can be more easily modified to a ***loop mode*** system with dynamic feedback.

Currently, no dynamic model is built-in for `t_camera` to simulate the servomechanism. A perfect and static model is assumed, i.e., the camera can look at whatever direction instantly. Note, however, the dynamics of the servomechanism may be simulated by modifying the control signals in this ideal model.

4.4.2 Command Line Options

Synopsis

`t_camera -world` *worldSpec* `-noise` *noise-intensity* `-o` *out_spec* **`[-help]`**

Options

`-world` *worldSpec* Required argument. The input spec file *worldSpec* should be the same as the output spec file of **`w_camera`** module. *worldSpec* contains the following data items in order: a sequence number, a image filename(full path), a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

-noise noise-intensity Required argument. *noise-intensity* specifies the file containing three floating numbers representing white noise intensities for x, y and z directions respectively. The exact target position as obtained from *worldSpec* are perturbed with Gaussian random values with their standard deviations determined by noise intensities, respectively. The perturbed values are used to simulate where the actual tracking camera is looking at. By changing the input noise intensities, one may mimic the tracking cameras with different tracking accuracies.

-o ***out_spec*** Required argument. The output spec file *out_spec* is almost the same as the input spec file *worldSpec*, except that additional information regarding where the the tracking camera is looking at the pixel location of the boresight in the world-view image.

-help Optional argument. When this argument is specified, **t_camera** prints out a brief help message and exits gracefully.

4.4.3 I/O File Specification

-world *worldSpec*

The input file *worldSpec* is the world view image *spec* file created by **w_camera**. The file format has been specified in Section 4.3.3.

-noise *noise-intensity*

The input file *noise-intensity* is an ASCII file that contains the following three *floating* numbers separated by one or more mixed *white space characters*(blanks, tabs, newlines):

$v_1 \quad v_2 \quad v_3$

where v_1, v_2 and v_3 are the standard deviations of the tracking residual errors in the cartesian inertial coordinates. The units for v_1, v_2 and v_3 are *meters*, *meters* and *meters*, respectively.

A typical input file *track* may look as follows:

```
0.5     0.5     0.12
```

-o ***out_spec***

The output file *out_spec* consists of following data items: a sequence number, a image filename(full path), three inertial cartesian coordinates indicating

where the tracking camera is looking at, two integers representing the pixel location of the tracking camera boresight in the world-view image, a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

4.5 Image Processor

4.5.1 Introduction

Conceptually, the image processor segments the input images from the tracking camera and outputs the segmentation results, such as the centroid pixel location, line-of-sight(LOS) of the target, etc. However, segmentation is very CPU-intensive and is not suitable for on-line interactive simulation. As a result, all the images have been pregenerate and segmented.

The centroid image processor `ipc` reads the description files for the world-view and segmented image databases and performs the on-line measurement from the segmentation results and target range information. Given the target pixel location and camera states, the line-of-sight of the target can be computed. With additional target range information, target locations can be found.

4.5.2 Command Line Options

Synopsis

ipc -imgInfo *imgInfoFile* **-segInfo** *segInfoFile* **-inSpec** *inSpec* **[-dir** *directory***] [-prefix** *prefix***] [-num_width** *num-width***] [-suffix** *suffix***] -outSpec** *outSpec* **-meas** *measurement* **[-img** *imgfile***] [-uc <1 or 0>] [-help]**

Options

`-imgInfo` *imgInfoFile* Required argument. It specifies the segment of the description file for the world-view image database. This file should be the output of the data control module. It contains following items (See Section 4.2.3):

- time at which the image is to be generated. This is identified by a " *keyword=value*" pair, with the keyword being **`time`**.
- the image identifier which maps to the corresponding image file. This is identified by a " *keyword=value*" pair, with the keyword being **`image`**.
- the image width in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **`width`**.
- the image height in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **`height`**.
- the tank's state at that instant which includes

  $$t \quad x_I \quad y_I \quad z_I \quad \dot{x}_I \quad \dot{y}_I \quad \dot{z}_I \quad R \quad S \quad T \quad \dot{R} \quad \dot{S} \quad \dot{T}$$

- the camera's state which includes

  $$x_{cI} \; y_{cI} \; z_{cI} \quad \dot{x}_{cI} \; \dot{y}_{cI} \; \dot{z}_{cI} \; R_c \; S_c \; T_c \; \dot{R}_c \; \dot{S}_c \; \dot{T}_c$$
  $$fov \; \dot{fov} \; hither \; yon \; \dot{yon}$$

where x_{cI}, y_{cI}, z_{cI}, $\dot{x}_{cI}$, $\dot{y}_{cI}$, $\dot{z}_{cI}$ denote the position and velocity components of the camera at the instant in global inertial coordinate system. R_c, S_c, T_c, $\dot{R}_c$, $\dot{S}_c$, $\dot{T}_c$ refer to the yaw, pitch , roll angles and their rates. fov and $\dot{fov}$ refer to the field of view(in degrees) and its time rate of change(in degrees per second), respectively; $hither$, $\dot{hither}$ denote the position of the front

clipping plane and its rate in the camera coordinate system; *yon*, $\dot{yon}$ position of the back clipping plane and its rate in the camera coordinate system. The camera coordinate system is such that its initial orientation aligns with the inertial coordinate system, and the camera is always looking towards the negative z direction of the body-fixed right-handed coordinate system.

- range of the target to camera and range rate.

-segInfo *segInfoFile* Required argument. It specifies the segment of the description file for the segmented image database. This file should be the output of the data control module. The file is composed of the data records of the following form (See Section 4.2.3):

```
time=2  image=s1.seg0020
        seg_ulx=90  seg_uly=316  seg_width=256  seg_height=256
        tcx=221.809  tcy=444.128  tvx=46.5792  tvy=5.43854
        out_ulx=-1  out_uly=-1  out_width=256  out_height=256
        size=46.8957  scale=2  mag=5.45893  proj=2
        roll=0  pitch=0  yaw=1.5609
        estroll=0  estpitch=0  estyaw=1.5708
```

-inSpec *inSpec* Required argument. The input spec file *inSpec* is the output spec file from the tracking camera. It consists of following data items: a sequence number, a image filename(full path), three inertial cartesian coordinates indicating where the tracking camera is looking at, two integers representing the pixel location of the tracking camera boresight in the world-view image, a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*). See Section 4.4.3.

-dir *directory* Optional argument. *directory* specifies the directory in which a desired segmented input image file resides.

Default: the current directory.

Example: `-dir $HITERT_HOME/data/images/scene1_seg`

-prefix *prefix* Optional argument. It specifies the string that *precedes* the sequence or index number in the file name.

Default: null

Example: `-prefix s1.seg`

-num_width *num-width* Optional argument. *num-width* specifies the width that the numeric sequence number takes. If *num-width* is greater than the number of the *digits* that *sequence-number*[1] has, *sequence-number* will be *prepended* with 0's(zeros) in formating the file name string.

-suffix *suffix* Optional argument. This option specifies the string in the file name that *follows* the sequence digit string. If *suffix* is ".Z", the input file is assumed to compressed by `compress(1)`.

Default: null

Example: `-suffix .dat`

-outSpec *outSpec* Required argument. The output spec file *outSpec* is almost the same as the input spec file *segInfoFile*, except it has the segmented image file name following the word-view image filename. More specifically, *outSpec* contains the following data items in order: a sequence number, a image filename(full path), a segmented image filename(full path), a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

[1]specified via `-inSpec` *inSpec*

-meas *measurement* Required argument. *measurement* is the file containing the measurement data about the cartesian coordinates of the target. This file has the following format:

t *x-measure* *y-measure* *z-measure*
⋮ ⋮ ⋮ ⋮

where *x-measure*, *y-measure* and *z-measure* are position measurements of the target centroid at time *t*. These *floating* numbers *t*, *x-measure*, *y-measure* and *z-measure* have the following units, respectively:

seconds *meters* *meters* *meters*

`-img` *imgfile* Optional argument. *imgfile* specifies the file to contain the same data as those contained in the *input file*, which is specified via *directory*, *prefix*, *sequence-number*, *sequence-number*, *suffix*. If this option is not specified, the default path name for *imgfile* is `/tmp/ipImg.$USER`, where `$USER` is the login name of the user.

Default: `/tmp/w_cam.$USER`
Example: `-img ~luj/tmp/ip_img`

`-uc` 1 *or* 0 Optional argument. This switch has effect only if the input image file is compressed and has the suffix ".Z". If the input image file is compressed and `-uc 1` is specified, the image image will be uncompressed by using `uncompress(1)` to produce the output image file. If the input image file is compressed but `-uc 0` is specified, the image will not be compressed and the output image will be tagged with ".Z" to indicate that the file is still compressed.

Default: 1
Example: `-uc 0`

4.5.3 I/O File Specification

-imgInfo *imgInfoFile*

The input ASCII file *imgInfoFile* is the segment of the world-view image database description file of the interest. It should be the output of the `dataControl` module. See Section 4.2.3.

-segInfo *segInfoFile*

The input ASCII file *segInfoFile* is the segment of the segmented image database description file of the interest. It should be the output of the `dataControl` module. See Section 4.2.3.

-inSpec *inSpec*

The input spec file *inSpec* is the output spec file from the tracking camera. See Section 4.4.3.

-outSpec *outSpec*

The output spec file *outSpec* contains the following data items in order: a sequence number, a image filename(full path), a segmented image filename(fullpath), a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*). All these data fields are separated by the mixed *white space characters*(blanks, tabs, newlines). A typical output spec file may look as follows:

```
0 /tmp/w_img.luj /tmp/ipImg.luj
1999.836125 -100.268080 -1.292417 163 445
time=0.000000  image=s1.img0000  width=900  height=900
        2000.000000 -100.000000 -1.400000 0.000000 8.938900 0.000000
```

```
          1.570800 0.000000 0.000000 -0.645000 0.000000 0.000000
          0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
          0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
          9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
          2002.504682 -0.446386
time= 0.000000 image= s1.seg0000
         seg_ulx= 0 seg_uly= 0 seg_width= 256 seg_height= 256
         tcx= 164.716000 tcy= 444.135000 tvx= 45.836500 tvy= 5.356840
         out_ulx= -1 out_uly= -1 out_width= 256 out_height= 256
         size= 46.148500 scale= 2.000000 mag= 5.547310 proj= 2
         roll= 0.000000 pitch= 0.000000 yaw= 1.570800
         estroll= 0.000000 estpitch= 0.000000 estyaw= 1.570800
500 /tmp/w_img.luj1 /tmp/ipImg.luj1
1687.603661 -89.648265 -1.655478 146 445
time=50.000000  image=s1.img0500  width=900  height=900
          1687.500000 -90.137000 -1.400000 -13.408000 -0.067162 0.000000
          3.146600 0.000000 0.000000 -0.200000 0.000000 0.000000
          0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
          0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
          9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
          1689.912994 -13.385272
time= 50.000000 image= s1.seg0500
         seg_ulx= 16 seg_uly= 318 seg_width= 256 seg_height= 256
         tcx= 144.383000 tcy= 446.439000 tvx= 13.450800 tvy= 6.849180
         out_ulx= -1 out_uly= -1 out_width= 256 out_height= 256
         size= 15.094200 scale= 2.000000 mag= 16.960100 proj= 2
         roll= 0.000000 pitch= 0.000000 yaw= 3.146600
         estroll= 0.000000 estpitch= 0.000000 estyaw= 3.154900
```

-meas *measurement*

The output ASCII data file *measurement* consists of measurement data. Each instant is associated with a data record consisting of the four floating numbers separated by the mixed white space characters:

t *x-measure* *y-measure* *z-measure*

The *data records* making up *measurement* are separated by one or more mixed *white space characters*(blanks, tabs, newlines). Within each *data record*, there are four *floating* numbers *t*, *x-measure*, *y-measure* and *z-measure*, which are also separated by one or more mixed *white space characters*. *x-measure*, *y-measure* and *z-measure* are the *position* measurements of the target centroid at the time *t*. The units of *t*, *x-measure*, *y-measure* and *z-measure* are *seconds*, *meters*, *meters* and *meters*, respectively.

A typical abridged measurement data file having 5 data records may look as follows:

```
2.000000 2000.109179 -79.822001 -1.498194
2.100000 2000.205959 -78.792146 -1.485431
2.200000 2000.203332 -77.740134 -1.485436
2.300000 2000.213978 -76.335661 -1.508155
2.400000 2000.211513 -75.260915 -1.515856
```

Input Data File

The input *data* file as specified by *directory*, *prefix*, *sequence-number*, *num-width* and *suffix* can be in any format, no interpretation is done on the content of this file. However, if the file name as specified above has the trailing suffix ".Z", the input data file is considered to have been compressed using `compress(1)`.

-img *imgfile*

imgfile is the actual output image ***data*** file. *imgfile* specifies the file to contain the same data as those contained in the *input file*, which is specified via *directory*, *prefix*, *sequence-number*, *sequence-number*, *suffix*, with the following exception: if the specified input file has the trailing string "`.Z`" and `-uc 1` is specified, then it is assumed that the input file has been compressed by using **`compress(1)`**, and `ipc` will filter the input file through **`uncompress(1)`** and store the results as *imgfile*.

4.6 Tracker

4.6.1 Introduction

`track` is the *tracker* program, responsible for on-line estimations of the tracker states, based on the measurement data from the *image processor*. **`track`** is an α-β-γ *Kalman filter* using the position measurements of the target centroid. The built-in Kalman filter gains are: 0.8790, 0.8790, 0.8790. To see how the gains are selected, interested reader may refer to Appendix A.

The inertial reference coordinate system is shown in Figure 1.2. Even thought the *terrain* in the scenario being considered is flat, **`track`** does state estimation for the x, y *and* z components, because the measurement output from the image processor is inherently 3-D, i.e. it contains three measurment outputs at each sampling instant.

4.6.2 Command Line Options

Synopsis

`track -ip` *ip-file* `[-ic` *ic-file*`]` **`-o`** *out-file* **`[-help]`**

Options

`-ip` *ip-file* Required argument. *ip-file* is the data file containing the position measurements from the *image processor*. It should be the output from the image processor. *ip-file* has the following format:

t *x-measure* *y-measure* *z-measure*
$\vdots$ $\vdots$ $\vdots$ $\vdots$

where *x-measure*, *y-measure* and *z-measure* are position measurements of the target centroid at time t.

`-ic` *ic-file* Optional argument. The *ic-file* contains the *initial conditions* for the states of the tracker. It consists of the following data:

$$t_0 \quad \hat{x} \quad \dot{\hat{x}} \quad \ddot{\hat{x}} \quad \hat{y} \quad \dot{\hat{y}} \quad \ddot{\hat{y}} \quad \hat{z} \quad \dot{\hat{z}} \quad \ddot{\hat{z}}$$

where $\hat{x}$, $\dot{\hat{x}}$, $\ddot{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$ and $\ddot{\hat{z}}$ are the *initial conditions* at the starting time t_0. If this option is not specified, `track` will set $\hat{x}$, $\dot{\hat{x}}$, $\ddot{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$ $\hat{z}$, $\dot{\hat{z}}$ and $\ddot{\hat{z}}$ all to zeros as the default values for the initial conditions.

`-o` *out-file* Required argument. *out-file* contains the state estimation results. It has the following format:

$$\begin{array}{cccccccccc} t & \hat{x} & \dot{\hat{x}} & \ddot{\hat{x}} & \hat{y} & \dot{\hat{y}} & \ddot{\hat{y}} & \hat{z} & \dot{\hat{z}} & \ddot{\hat{z}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

where $\hat{x}$, $\dot{\hat{x}}$, $\ddot{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$ $\hat{z}$, $\dot{\hat{z}}$, $\ddot{\hat{z}}$ are the estimated states at the time t.

`-help` Optional argument. When this argument is specified, `track` prints out a brief help message and exits gracefully.

4.6.3 I/O File Specification

-ip ***ip-file***

The input ASCII file *ip-file* is the the measurement data file from the image processor. See Section 4.5.3.

-ic *ic-file*

ic-file is the initial condition file, consisting of following numbers:

$$t_0 \quad \hat{x} \quad \dot{\hat{x}} \quad \bar{\hat{x}} \quad \hat{y} \quad \dot{\hat{y}} \quad \bar{\hat{y}} \; \hat{z} \quad \dot{\hat{z}} \quad \bar{\hat{z}}$$

where $\hat{x}$, $\dot{\hat{x}}$ and $\bar{\hat{x}}$ are the position, speed and acceleration of the target at time t_0. t_0 has the units *seconds*; whereas $\hat{x}$, $\dot{\hat{x}}$ and $\bar{\hat{x}}$ have units *meters*, *meters/seconds* and *meters/(seconds)*^{2}, respectively. Similar remarks apply to $\hat{y}$, $\dot{\hat{y}}$, $\bar{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$ and $\bar{\hat{z}}$.

The 10 *floating* numbers t_0, $\hat{x}$, $\dot{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\bar{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$ and $\bar{\hat{z}}$ are separated by one or more mixed *white space characters*. A typical *ic-file* may look like this

```
 0.0
 2.0000e+03    0.0000e+00    0.0000e+00
-1.0000e+02    0.0000e+00    0.0000e+00
       -1.4    0             0
```

-o *out-file*

out-file is the output data file produced by `track`. The file contains the estimation results for the states of the α-β-γ *Kalman filter*. *out-file* consists of the *data records* of the form:

$$t \quad \hat{x} \quad \dot{\hat{x}} \quad \bar{\hat{x}} \quad \hat{y} \quad \dot{\hat{y}} \quad \bar{\hat{y}} \; \hat{z} \quad \dot{\hat{z}} \quad \bar{\hat{z}}$$

The *data records* making up *out-file* are separated by one or more mixed *white space characters*(blanks, tabs, newlines). Within each *data record*, there are 10 *floating* numbers t, $\hat{x}$, $\dot{\hat{x}}$, , $\bar{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\bar{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$, $\bar{\hat{z}}$. $\hat{x}$, $\dot{\hat{x}}$ and $\bar{\hat{x}}$ are position, speed and acceleration of the target in inertial x component at time t_0. t has the units *seconds*; whereas $\hat{x}$, $\dot{\hat{x}}$ and $\bar{\hat{x}}$ have units *meters*, *meters/seconds* and *meters/(seconds)*^{2}, respectively. Similar remarks apply to $\hat{y}$, $\dot{\hat{y}}$, $\bar{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$, $\bar{\hat{z}}$.

A typical abridged *out-file* having 5 *data records* may look like this

```
2.0 2000.139048 0.067167 -0.042767
```

```
       -79.399693 11.831204 1.019657
        -1.500131 0.045759 0.060516
2.1 2000.130446 0.028983 -0.080865
       -78.425036 11.453766 0.481003
        -1.494273 0.054009 0.062987
2.2 2000.171127 0.106619  0.015452
       -77.462902 11.085122 0.012754
        -1.484086 0.070346 0.074265
2.3 2000.198152 0.144723 0.056529
       -76.494522 10.771683 -0.340856
        -1.477363 0.076240 0.072543
2.4 2000.220910 0.168341 0.076715
       -75.338722 10.917937 -0.138229
        -1.484948 0.048538 0.033268
```

4.7 Predictor

4.7.1 Introduction

`predict` is the predictor program, responsible for computing the future position of the target. The *predict-ahead* or *lead* time is the time of the expected flight time of the projectile intercepting the target. `predict` is used in conjunction with the α-β-γ *Kalman filter*. The prediction equation used by `predict` is as the follows:

$$\hat{x}_i(t + t_p|t) = \hat{x}_i(t) + \dot{\hat{x}}_i(t)t_p + 1/2\ddot{\hat{x}}_i(t)t_p{}^2$$

where t is the *current* time, t_p the *predict-ahead* or *lead* time, $\hat{x}_i(t + t_p|t)$ the *predicted* target position at the future time $t + t_p$ given the estimates at the present time t; $x_i(t)$, $\dot{\hat{x}}_i(t)$, $\ddot{\hat{x}}_i(t)$ are the estimates of the current position,

speed and acceleration, respectively. The subscript i applies to each x, y and z component, respectively.

4.7.2 Command Line Options

Synopsis

`predict -i` *in-file* `[-tp` *tp*`] -o` *out-file* `[-help]`

Options

`-i` *in-file* Required argument. *in-file* is the input file containing the state estimation results from the α-β-γ tracker. The file has the following format:

$$\begin{array}{cccccccccc} t & \hat{x} & \dot{\hat{x}} & \ddot{\hat{x}} & \hat{y} & \dot{\hat{y}} & \ddot{\hat{y}} & \hat{z} & \dot{\hat{z}} & \ddot{\hat{z}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

where $\hat{x}$, $\dot{\hat{x}}$ and $\ddot{\hat{x}}$ are the position, speed and acceleration of the target at time t_0. t has the unit of *seconds*; whereas $\hat{x}$, $\dot{\hat{x}}$ and $\ddot{\hat{x}}$ have units of *meters*, *meters/seconds* and *meters/(seconds)*^{2}, respectively. Similar remarks apply to $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$ and $\ddot{\hat{z}}$, respectively. These data are separated by one or more mixed *white space characters*.

`-tp` *tp* Optional argument. *tp* is the *predict-ahead* or *lead* time, which is the expected flight time of the projectile intercepting the target. *tp* is a *floating* number and has the unit of *seconds*.

Default: 2

Example: `-tp 1`

`-o` *out-file* Required argument. *out-file* is the output data file produced by `predict`. This file has the following format:

$$\begin{array}{ccccc} t & t_f & \hat{x}(t_f|t) & \hat{y}(t_f|t) & \hat{z}(t_f|t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

where t is the *current* time, t_f the *future* time($t_f = t + t_p$), $\hat{x}(t_f|t)$, $\hat{y}(t_f|t)$ and $\hat{z}(t_f|t)$ the *predicted* target position coordinates at t_f based on the *current* state estimates at t.

-help Optional argument. When this argument is specified, **predict** prints out a brief help message and exits gracefully.

4.7.3 I/O File Specification

-i *in-file*

This input file *in-file* to **predict** is the output state estimation file from **track**. The file format has been specified in Section 4.6.3.

-o *out-file*

out-file is the output data file produced by **predict**. The file consists of the data *records* of the form:

$$\begin{matrix} t & t_f & \hat{x}(t_f|t) & \hat{y}(t_f|t) & \hat{z}(t_f|t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{matrix}$$

The *data records* making up *out-file* are separated by one or more mixed *white space characters*(blanks, tabs, newlines). Within each such a *data record*, t is the *current* time, t_f the *future* time($t_f = t + t_p$), $\hat{x}(t_f|t)$, $\hat{y}(t_f|t)$ and $\hat{z}(t_f|t)$ the *predicted* target position at t_f based on the *current* state estimates at t. These *floating* numbers are separated by one or more mixed *white space characters*. t, t_f, $\hat{x}(t_f|t)$, $\hat{y}(t_f|t)$ and $\hat{z}(t_f|t)$ have the following units, respectively:

seconds *seconds* *meters* *meters* *meters*

A typical abridged *out-file* having 5 data records may look as follows:

```
2.000000 4.000000 2000.187848 -53.697971 -1.287581
2.100000 4.100000 2000.026682 -54.555498 -1.260281
2.200000 4.200000 2000.415269 -55.267150 -1.194864
```

```
2.300000 4.300000 2000.600656 -55.632868 -1.179797
2.400000 4.400000 2000.711022 -53.779306 -1.321336
```

4.8 Error Analysis

4.8.1 Introduction

errAnaly is the program that performs the *error analysis* for the HiTert system. **errAnaly** takes as its inputs the prediction data and the reference exact data, and computes prediction errors as well as the statistics on the error data.

Two types of errors are computed depending on the command line option specified: *inertial component errors* and *dartboard errors* . The *inertial component errors* are computed by finding the differences between the *predicted* target positions and the *actual* target positions in each inertial component x, y and z, respectively. More specifically, the *inertial component errors* are defined as follows:

$$\epsilon_i(t) = \hat{x}_i(t) - x_i(t)$$

where $\epsilon_i(t)$ is the *error* at t, $\hat{x}_i(t)$ the *predicted* inertial target position coordinate, $x_i(t)$ the *exact* inertial target position coordinate. The subscript i applies to each inertial x, y and z component, respectively. These errors are used subsequently to perform the *error statistics*.

The *dartboard errors*, on the other hand, are the *perspective projections* of the *inertial error vectors* onto the *dartboard* as seen by an observer[1]. Figure 4.2 shows the *dartboard* at an arbitrary instant t. Point $\hat{P}$ is the *predicted* position of the target at t, P_0 the *exact* position, O the location of the observer. The *dartboard* π is centered on the point P_0, the *inertial error vector* at this instant is given by $\overrightarrow{P_0\hat{P}}$. The perspective projection of $\overrightarrow{P_0\hat{P}}$

[1]Currently, the observer is fixed on the *tracking camera*.

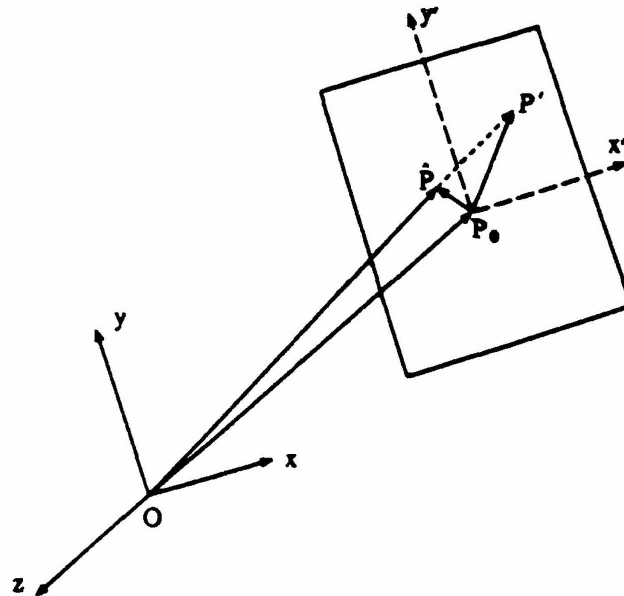


Figure 4.2: Definitions of the dartboard

onto the dartboard plane π is given by $\overrightarrow{P_0P'}$. Note that the intersection of the line $O\hat{P}$ and the dartboard plane π is P'. The components of $\overrightarrow{P_0P'}$ in the $x'y'$ plane are the *dartboard errors*. Appendix B has the derivations on how to find the *dartboard errors*. Please note that the *dartboard errors* are mainly intended for graphical displaying purpose. The *error statistics* are still computed based on the *inertial component errors*.

4.8.2 Command Line Options

Synopsis

`errAnaly -pred` *predfile* `-exact` *exactfile* `[-showDart` *showDart*`]` `[-tbf` *tbf*`]` `[-tburst` *tburst*`]` `-outSpec` *outSpec* `[-outErr` *outErr*`]` `[-help]`

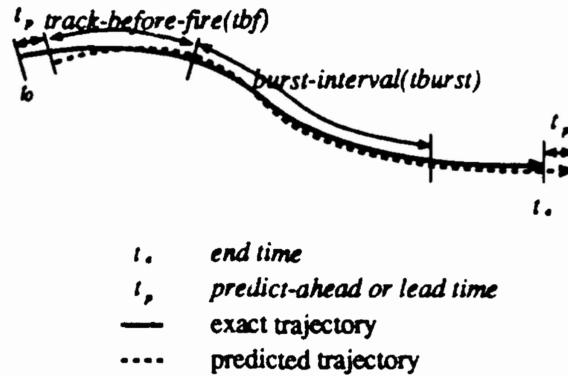


Figure 4.3: Definition of the burst interval

Options

-pred *predfile* Required argument. *predfile* is the data file containing the prediction results from **predict**. The file has the following format:

$$\begin{array}{ccccc} t & t_f & \hat{x}(t_f|t) & \hat{y}(t_f|t) & \hat{z}(t_f|t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

where t is the *current* time, t_f the *future* time($t_f = t + t_p$), $\hat{x}(t_f|t)$, $\hat{y}(t_f|t)$ and $\hat{z}(t_f|t)$ the *predicted* target position coordinates at t_f based on the *current* state estimates at t. These *floating* numbers t, t_f, $\hat{x}(t_f|t)$, $\hat{y}(t_f|t)$ and $\hat{z}(t_f|t)$ have the following units, respectively:

seconds seconds meters meters meters

-exact *exactfile* Required argument. *exactfile* contains the exact positions of the target. This file has the following format:

$$\begin{array}{cccc} t & x(t) & y(t) & z(t) \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

where t is the *current* time, $x(t)$, $y(t)$ and $z(t)$ the *exact* target position coordinates at t. These *floating* numbers t, $x(t)$, $y(t)$ and $z(t)$ have the

following units, respectively:

seconds meters meters meters

`-showDart` *showDart* Optional argument. This *boolean* argument *showDart* specifies whether or not the *dartboard errors* should be computed. If *showDart* is *false*, i.e., 0(zero), **errAnaly** will compute the *inertial component errors* and write the error data to a file(see option "`-outErr` *outErr*''); otherwise, **errAnaly** will compute the *dartboard errors* and and write the error data to a file. Note the error data computed are intended for graphical displaying purpose. The statistics are *always* computed based on the *inertial component error* data.

Default: 1
Example: -showDart 0

`-tbf` *tbf* Optional argument. *tbf* is the *track-before-fire time*(See Figure 4.3). *tbf* is a *floating* number and has the unit of *seconds*.

Default: *tp*
Example: `-tbf 5`

`-tburst` *tburst* Optional argument. *tburst* is the *burst interval* over which projectiles are fired to intercept the target(See Figure 4.3). *tburst* is a *floating* number and has the unit of *seconds*. The relevant statistics are computed based on the data during this *burst interval*.

If *tburst* is not specified, the default value of $(end_time - start_time - tbf)$ is used, where *start_time* and *end_time* are the instants at which the tracking system starts and terminates, respectively. *start_time* and *end_time* are controlled by the *data control* module. **errAnaly** determines the values of *start_time* and *end_time* from the numbers in the input data files.

If *tburst* is specified and is greater than $(end_time - start_time - tbf)$, *tburst* will be set to $(end_time - start_time - tbf)$ by `errAnaly`.

Default: $(end_time - start_time - tbf)$

Example: `-tburst 10`

`-outSpec` *outSpec* *outSpec* contains the name or path of the file containing computed error data, and prediction error statistics. The file has the following format:

t_0	t_e	t_p	tbf	$tburst$
np	n_{tbf}	n_{burst}	n_{burst_end}	
$max_{\vert\epsilon_x\vert}$	m_{ϵ_x}	rms_{ϵ_x}		
$max_{\vert\epsilon_y\vert}$	m_{ϵ_y}	rms_{ϵ_y}		
$max_{\vert\epsilon_z\vert}$	m_{ϵ_z}	rms_{ϵ_z}		
$max_{\vert\epsilon_{az}\vert}$	$m_{\epsilon_{az}}$	$rms_{\epsilon_{az}}$		
$max_{\vert\epsilon_{el}\vert}$	$m_{\epsilon_{el}}$	$rms_{\epsilon_{el}}$		
$showDart$				
$errorDataFilename$				

where t_0, t_e, t_p, tbf and $tburst$ are the *start_time*, *end_time*, *predict-ahead time*, *track-before-fire time*, *burst interval*, respectively; $max_{|\epsilon_i|}$, m_{ϵ_i}, rms_{ϵ_i} are the maximum absolute error, *mean* and *RMS (Root Mean Square) value* of errors in inertial x, inertial y, inertial z, azimuth and elevation components, respectively; *showDart* is a boolean having the meaning as that specified under the option "`-showDart` *showDart*". *errorDataFilename* is the path of the file containing the error data(see below).

`-outErr` *outErr* Optional argument. *outErr* specifies the name or path of the file to contain the computed error data. If this option is not spec-

ified, the default path name for *outErr* is `/tmp/w_cam.$USER`, where `$USER` is the login name of the user.

The format of the file *outErr* depends on the boolean flag *showDart* specified. If *showDart* is *false*, i.e., 0, *outErr* contains the *inertial component errors* and has the following format:

$$\begin{array}{ccccc} t & t_f & \epsilon_x(t_f|t) & \epsilon_y(t_f|t) & \epsilon_z(t_f|t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

If *showDart* is *true*, i.e., *non-zero*, *outErr* contains the *dartboard errors* and has the following format:

$$\begin{array}{cccc} t & t_f & d_x(t_f|t) & d_y(t_f|t) \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

Default: `/tmp/errAnaly.$USER`

`-help` Optional argument. When this argument is specified, **`errAnaly`** prints out a brief help message and exits gracefully.

4.8.3 I/O File Specification

-pred *predfile*

predfile is an *input* file to `errAnaly`. This file contains the prediction results from `predict`. See Section 4.7.3.

-exact *exactfile*

The input ASCII file *exactfile* contains the reference exact data of the target trajectory. It should be the output of `dataControl`. See Section 4.2.3.

A typical *exactfile* having 5 *data records* may look as follows:

```
2.000000 2000.100000 -79.946000 -1.400000
2.100000 2000.200000 -78.837000 -1.400000
2.200000 2000.200000 -77.718000 -1.400000
```

```
2.300000 2000.200000 -76.590000 -1.400000
2.400000 2000.200000 -75.453000 -1.400000
```

-outSpec *outSpec*

exactfile is an *output* file from **errAnaly**. The file contains the following data, which are separated by one or more mixed *white space characters*(blanks, tabs, newlines).

t_0	t_e	t_p	tbf	$tburst$
np	n_{tbf}	n_{burst}	n_{burst_end}	
$max_{\|\epsilon_x\|}$	m_{ϵ_x}	rms_{ϵ_x}		
$max_{\|\epsilon_y\|}$	m_{ϵ_y}	rms_{ϵ_y}		
$max_{\|\epsilon_z\|}$	m_{ϵ_z}	rms_{ϵ_z}		
$max_{\|\epsilon_{az}\|}$	$m_{\epsilon_{az}}$	$rms_{\epsilon_{az}}$		
$max_{\|\epsilon_{el}\|}$	$m_{\epsilon_{el}}$	$rms_{\epsilon_{el}}$		
$showDart$				
$errorDataFilename$				

where t_0, t_e, t_p, tbf and $tburst$ are the *start_time*, *end_time*, *predict-ahead time*, *track-before-fire time*, *burst interval*, respectively, and they have the unit of *seconds*; $max_{|\epsilon_i|}$, m_{ϵ_i}, rms_{ϵ_i}(i takes x, y, or z) are the maximum absolute error, *mean* and *RMS(Root Mean Square) value* of the errors in inertial x, y and z components, respectively, and they have the unit of *meters*; when i takes az or el, $max_{|\epsilon_i|}$, m_{ϵ_i}, rms_{ϵ_i} are the errors in the azimuth and elevation components, respectively, and they have the unit of *radians*; $showDart$ is a boolean having the meaning as that specified under the option "**-showDart** *showDart*". $errorDataFilename$ is the path of the file containing the error data and is the same as the file name *outErr*.

-outErr *outErr*

outErr is an *output* file from **errAnaly**. The contents or format of the file depends on the *boolean* flag *showDart* specified via the command line option.

If *showDart* is *false*, i.e., 0, *outErr* contains the *inertial component errors* and is composed of the *data records* of the form:

$$t \quad t_f \quad \epsilon_x(t_f|t) \quad \epsilon_y(t_f|t) \quad \epsilon_z(t_f|t)$$

These *data records* are separated by one or more mixed *white space characters*(blanks, tabs, newlines). Within each *data record*, there are 5 *floating* numbers t, t_f, $\epsilon_x(t_f|t)$, $\epsilon_y(t_f|t)$, $\epsilon_z(t_f|t)$, which are also separated by one or more mixed *white space characters*. t is the *current* time, t_f the *future* time($t_f = t + t_p$); $\epsilon_x(t_f|t)$, $\epsilon_y(t_f|t)$, $\epsilon_z(t_f|t)$ are the *inertial* component *prediction* errors at t_f based on the *current* state estimates at t. t, t_f, $\epsilon_x(t_f|t)$, $\epsilon_y(t_f|t)$, $\epsilon_z(t_f|t)$ have the units of *seconds*, *seconds*, *meters*, *meters* and *meters*, respectively.

If *showDart* is *true*, i.e., *non-zero*, *outErr* ontains the *dartboard errors* and is composed of the *data records* of the form:

$$t \quad t_f \quad d_x(t_f|t) \quad d_y(t_f|t)$$

These *data records* are separated by one or more mixed *white space characters*. Within each *data record*, there are 4 *floating* numbers t, t_f, $d_x(t_f|t)$ and $d_y(t_f|t)$, which are also separated by one or more mixed *white space characters*. t is the *current* time, t_f the *future* time($t_f = t + t_p$); $d_x(t_f|t)$ and $d_y(t_f|t)$ are the *dartboard component errors*. t, t_f, $d_x(t_f|t)$ and $d_y(t_f|t)$ have the units of *seconds*, *seconds*, *meters*, *meters* , respectively

4.9 Instrumentation Module

4.9.1 Introduction

xhitert is the *instrumentation module* program, responsible for displaying world-view images, tracking camera images, segmented images and error

analysis results. The program must be executed under the X11 windowing environment.

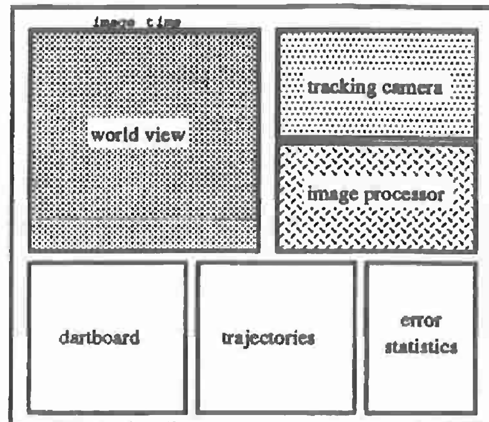


Figure 4.4: The window arrangement.

During its execution, `xhitert` will pop up a window of almost a full-screen size. This window consists of 6 subwindows(See Figure 4.4). The world-view image displayed has been shrinked by a factor of 2 in order to fit into the window. The red box in the world view image indicates where the tracking camera is looking at. Also displayed are the corresponding tracking-camera image, which is not scaled, and the segemented binary image, which is *normalized*. In the *batch mode*, only images corresonding to a *single* user specified instant are displayed[1].

In the subwindow displaying the trajectories, the red curve is the *predicted* trajectory, while the green curve is the exact trajectory.

`xhitert` is built on top of Xlib. It runs much faster on a display supporting 8-bit *pseudo-color* than on a monochrome display, since on monochrome display `xhitert` has to dither the large color images to black and whtie ones.

[1]However, the *dartboard* and trajectory windows display the results for the whole time period of the operation.

4.9.2 Command Line Options

Synopsis

xhitert -world *worldSpec* **-camera** *cameraSpec* **-ip** *ipSpec* **-pred** *predfile* **-exact** *exactfile* **-errSpec** *errSpec* **[-display** *display***] [-demo** *demo***] [-help]**

Options

-world *worldSpec* Required argument. *worldSpec* is world-view image *spec* file. It should be the output spec file of the **w_camera** module. It contains the following data items in order: A sequence number, a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

-camera *cameraSpec* Required argument. It should be the output spec file of **t_camera** module. *cameraSpec* consists of following data items: a sequence number, a image filename(full path), three inertial cartesian coordinates indicating where the tracking camera is looking at, two integers representing the pixel location of the tracking camera boresight in the world-view image, a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

-ip *ipSpec* Required argument. It should be the output spec file from the **ipc** module. *ipSpec* contains the following data items in order: a sequence number, a image filename(full path), a segmented image filename(fullpath), a data record from the image description file at *start-time*(or *end-time*), a data record from the description file for the segmented image database at *start-time*(or *end-time*).

-pred *predfile* Required argument. *predfile* is the *input* file containing the prediction data from `predict`. It should be the output of the predictor. This file has the following format:

$$\begin{array}{lllll} t & t_f & \hat{x}(t_f|t) & \hat{y}(t_f|t) & \hat{z}(t_f|t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

where t is the *current* time, t_f the *future* time($t_f = t + t_p$), $\hat{x}(t_f|t)$, $\hat{y}(t_f|t)$ and $\hat{z}(t_f|t)$ the *predicted* target position coordinates at t_f based on the *current* state estimates at t.

-exact *exactfile* Required argument. *exactfile* is the *input* file containing the reference *exact* trajectory data. It should be the output of the **dataControl** module. This file has the following format:

$$\begin{array}{llll} t & x(t) & y(t) & z(t) \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

where t is the *current* time, $x(t)$, $y(t)$ and $z(t)$ the *exact* target position coordinates at t. These *floating* numbers t, $x(t)$, $y(t)$ and $z(t)$ have the following units, respectively:

$$\begin{array}{llll} seconds & meters & meters & meters \end{array}$$

-errSpec *errSpec* Required argument. *errSpec* is the error *spec* file from **errAnaly**. *input spec* file has the following format:

$$\begin{array}{lllll} t_0 & t_e & t_p & tbf & tburst \\ np & n_{tbf} & n_{burst} & n_{burst_end} & \\ max_{|\epsilon_x|} & m_{\epsilon_x} & rms_{\epsilon_x} & & \\ max_{|\epsilon_y|} & m_{\epsilon_y} & rms_{\epsilon_y} & & \\ max_{|\epsilon_z|} & m_{\epsilon_z} & rms_{\epsilon_z} & & \\ max_{|\epsilon_{az}|} & m_{\epsilon_{az}} & rms_{\epsilon_{az}} & & \\ max_{|\epsilon_{el}|} & m_{\epsilon_{el}} & rms_{\epsilon_{el}} & & \\ showDart & & & & \end{array}$$

errorDataFilename

where t_0, t_e, t_p, *tbf* and *tburst* are the *start_time*, *end_time*, *predict-ahead time*, *track-before-fire time*, *burst interval*, respectively; $max_{|\epsilon_i|}$, m_{ϵ_i}, rms_{ϵ_i} are the maximum absolute error, *mean* and *RMS (Root Mean Square) value* of errors in inertial x, inertial y, inertial z, azimuth and elevation components, respectively; *showDart* is a boolean having the meaning as that specified under the option "`-showDart` *showDart*" in `errAnaly`. *errorDataFilename* is the path of the file containing the error data.

`-display` *display* Optional argument. *display* specifies where to display the images and the results. It is a string of the format "host:display:screen".

Default: `$DISPLAY`

Example: delta:0

`-demo` *demo* Optional argument. This *boolean* flag *demo* specifies whether or not to display the tracking results in a *pseudo real-time mode*. If *demo* is *false*, i.e., 0, all the tracking results will be *blasted* to the screen almost at the same time; if *demo* is *true*, however, `xhitert` will be in the *pseudo real-time mode*, displaying the tracking results one at a time.

Default: 0

Example: -demo 1

4.9.3 I/O File Specification

-world *worldSpec*

worldSpec is an *input spec* file produced by `v_camera`. See Section 4.3.3.

-camera *cameraSpec*

cameraSpec is an *input spec* file produced by **`t_camera`**. See Section 4.4.3.

-ip *ipSpec*

ipSpec is an *input spec* file produced by `ip`.

-pred *predfile*

predfile is an *input* data file produced by **`predict`**. See Section 4.7.3.

-exact *exactfile*

exactfile is an *input* data file containing the exact reference trajectory data. See Section 4.8.3.

-errSpec *errSpec*

errSpec is an *input spec* file produced by **`errAnaly`**. See Section 4.8.3.

Image Data File

The *input* image data file as specified by *ImageFileName* in the *worldSpec* must be a Khoros *viff* image. See Section 4.3.3.

Error Data File

The *input* error data file as specified by *errorDataFilename* is produced by **`errAnaly`**. The error data file has the format as specified in Section 4.8.3.

Chapter 5

World-view Image Database

5.1 Introduction

One of the important part in the simulation of the HiTert system is to simulate the image acquiring system. Various factors and considerations lead to the decision of building an image database by pregenerating a temporal sequence of images. Individual images in the database can be can be retrieved or played back for the study of image processing algorithms and the tracking system. With the tool of the image generation program, different scenarios can be simulated to study the HiTert system. In this chapter, discussions will be concentrated on the specific scene that has been constructed and the images of which have been actually stored on the disk.

5.2 Trajectory Data

The scenario under the consideration is illustrated in Figure 1.2 and reproduced in Figure 5.1. First, the geometric trajectory that a land vehicle, equipped with certain driving logic, may traverse is envisioned or proposed. It is then endowed with the time information, i.e. the velocity and the heading angle profiles of a generic tank along the trajectory are designed. Note

that on a flat terrain the velocity and the heading angle profiles completely determines the trajectory, given the initial conditions. It is assumed that heading direction of the vehicle is along the tangential direction of the trajectory. This assumption is generally valid for a tracked vehicle(e.g. tank) with no skidding. Differentiation of the heading angle with respect to time yields the angular velocity of the tank. The product of the angular velocity and the speed at a point along the trajectory gives the centripetal acceleration of the tank. This centripetal acceleration depends on the lateral frictional characteristics between the terrain and the track, and is constrained, under the no-skidding condition, by the following equation:

$$a_n = \omega v \leq \mu_t g$$

where a_n is the centripetal acceleration, ω the angular velocity of the tank, v the forward speed, μ_t the lateral frictional coefficient, g the gravitational constant.

The velocity profile is designed in such a way that its time rate of change falls within the acceleration and deceleration capability of a generic tank(e.g. M1 tank). From the velocity profile and heading angle profile, the trajectory can be constructed through the numerical integration. The constructed trajectory can further be examined to see if it is indeed the desired one.

Since a target may take any maneuvers as the driver deems necessary in the real environment, the trajectory produced from the above is realistic as long as the dynamic constraints are not violated.

The trajectory data used in generating image database are plotted in Figure 1.3 and is reproduced in Figure 5.2. One of the important features in the plots is that when the tank makes a turn it decelerates, while when it travels along a straight path it accelerates or keeps at a higher speed.

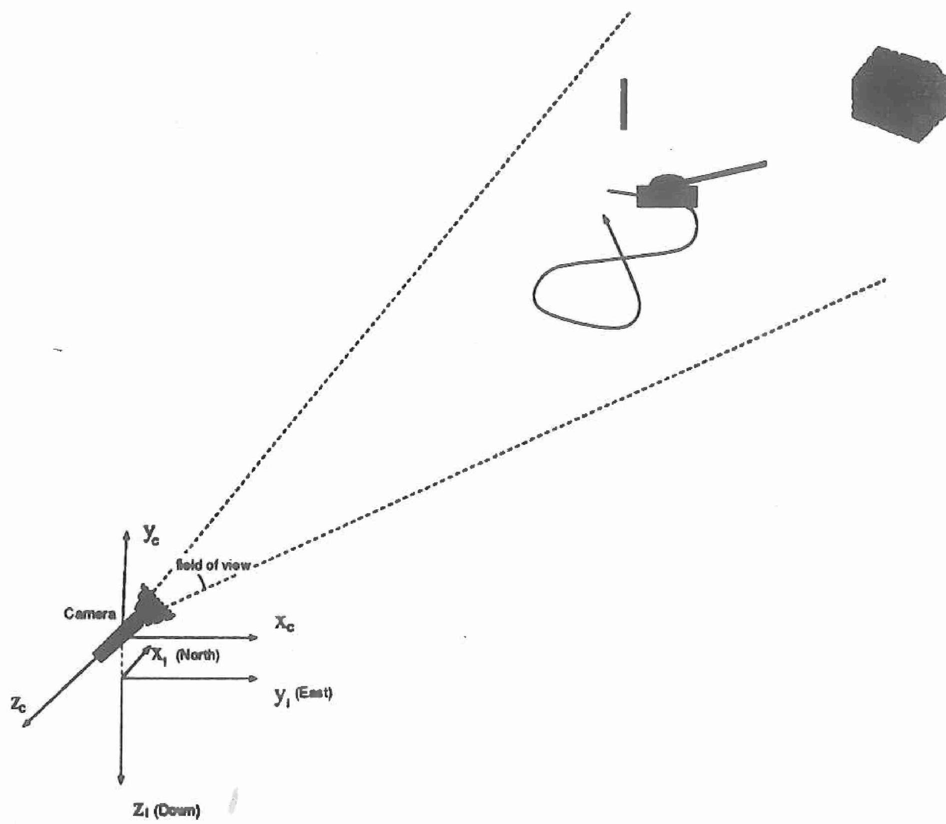


Figure 5.1: Scenario under consideration

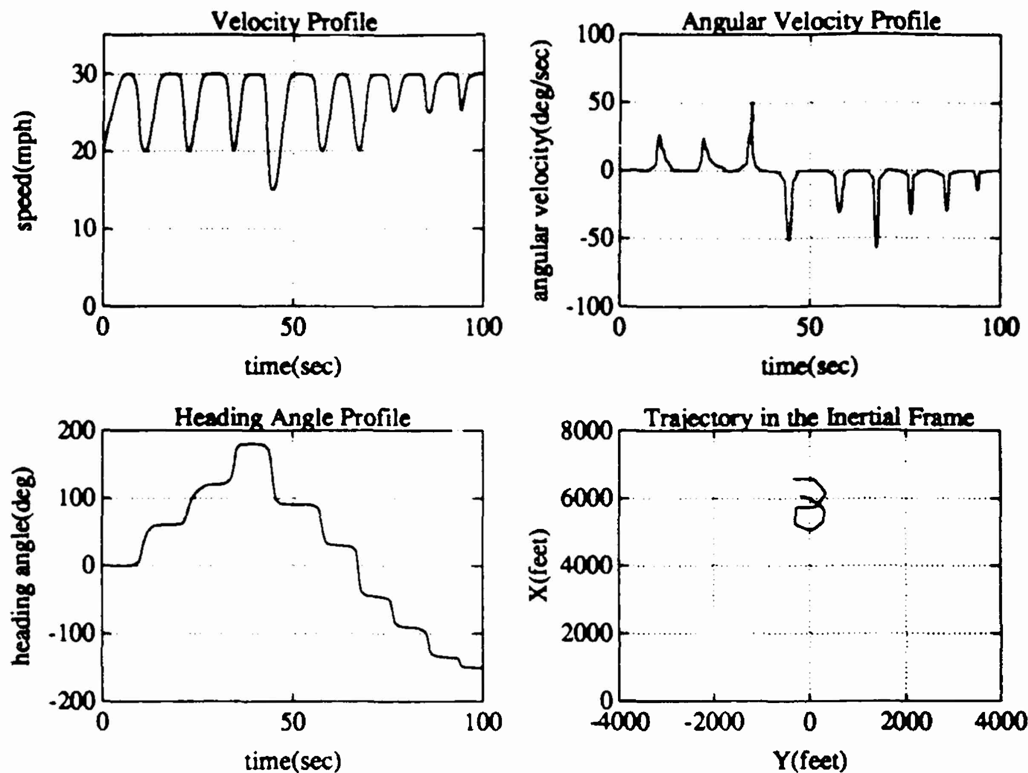


Figure 5.2: Trajectory data

5.3 Scenario

The tracking scenario is illustrated in Figure 5.1. The global inertial coordinate system is fixed on the flat earth, with positive x pointing to North, y to East and z vertically down.

At $t = 0$ seconu, the tank is located at $(6561.7, -328.1, 0.0)$ feet or $(2000, -100, 0)$ meters, traveling east with a speed of 20 mph. The subsequent states/maneuvers of the tank can be seen from Fig.1. The house is located at $(6561.7, 262.5, 0)$ ft and the tree at $(6594.5, 295.3, 0)$ ft.

The camera, fixed at $(0.0, 0.0, -16.4)$ ft, is 5 meters(16.4 ft) above the ground and has a constant view angle of 9 degrees. The depth of the field is determined by the hither and yon clipping planes of the camera. The hither and yon clipping planes are 1 meter(3.3 ft) and 2500 meters (8202.1 ft), respectively, away from and in front of the camera.

5.4 Image Generation

A C program named **tank** has been written to do image rendering. **tank** is built on top of the 3-D graphics library Doré(Dynamic Object Rendering Environment). **tank** has two interfaces with Doré: one for the *dynamic renderer*, the other for the *production renderer*. The dynamic renderer can achieve near real-time response, but requires special-purpose graphics hardware. As a result, the dynamic renderer interface of **tank** only runs on the Stardent machine. The production renderer, on the hand, relys on the software for scene rendering, and can run both on a Sun and Stardent machine, provided the Doré library is available.

The images were generated with the aid of Doré on Stardent 3000(titan P3) machine. Titan P3 is a supermini graphics computer. It has 1280×1024 color display which supports double buffering and 24-bit-per-pixel image. Because of the hardware problem, the dynamic renderer could not be run in multi-user mode. As a result, the ray-tracer was used to generate the images. At the time of image generation, the machine's configuration was 3 cpus with 256MB RAM.

Since it was expected that the resulting images would also be displayed on a Sun workstation, which typically has a screen size of 1152×900 pixels, the dimension of the images was chosen to be 900×900 pixels.

tank's production renderer takes the trajectory data as input and generates a frame of image for each sampling instant. The trajectory data were

sampled at the .1 second from 0 second to 100*th* second inclusive. 1001 images were generated after 64 hours(38-hour cpu time) of running at the highest elevated user priority.

The images generated by the Doré production renderer are in Doré raster file format and had 24 bits per pixel, with each RGB channel having 8 bits. A 900×900 image in Doré rasterfile format takes about 2.43 MB disk space. This means that the total images would consume 2.43 GB(2430 MB) disk space, which is prohibitively expensive and unfeasible. For this reason, at the run time, each Doré raster image was converted to ***viff*** format image and a color compression code was introduced, which compressed the 24-bit-per-pixel image to 8 bits per pixel. This resulted in a reduction of 2/3 disk storage, and only 810 MB storage was needed. The 8-bit images were further compressed using unix file compressing utility `compress(1)`, resulting in about another 95% compression. The images are finally in 8-bit viff format, stored as `*.Z` files, which consume only 4 MB of disk space. Compared to 2430 MB storage originally required, this is a very significant reduction. The whole conversion process is automatically done by the image-generation program if the command line option "-c" to the program is specified.

5.5 I/O File Specification

5.5.1 Command Line Options

Synopsis

tank [**-x** *xstart*] [-y *ystart*] [-W *width*] [-H *height*] [-procs *processors*] [**-dt** *devicetype*] [-i *datafile*] [-o *imgfile***imgfile**] [**-debug**] [c] [**-track**] [**-help**]

Options

-x *xstart* Optional argument. It specifies the horizontal position in pixels of the upper left corner of the Doré window relative to upper left corner of the root window. This option has effect only for the dynamic renderer running under the windowing environment and will be ignored the image is written to a file.

Default: 0

Example: `-x 20`

-y *ystart* Optional argument. It specifies the vertical position in pixels of the upper left corner of the Doré window. This option has effect only for dynamic renderer running under the windowing environment and will be ignored the image is written to a file.

Default: 0

Example: `-y 20`

-W *width* Optional argument. Specifies the width in pixels for the Doré window. In the case that the image is written to a file, the image width can be larger than the screen width.

Default: 128

Example: `-W 512`

-H *height* Optional argument. Specifies the height in pixels for the Doré window. In the case that the image is written to a file, the image width can be larger than the screen height.

Default: 128

Example: `-H 512`

-procs *processors* Optional argument. Specifies the number of processors to be used. The default is that only 1 CPU is going to be used(i.e.

processors takes the value of 0). The value of 1 tells Doré to run in a multiprocessor mode, but only with one processor. This mode is usually used for debugging purpose and is less efficient than specifying 0(true uniprocessor operation). In the case that the number of processors specified are more than that of machine's actual configuration, the actual number of CPUs will be used.

Default: 0

Example: `-procs 2`

-dt *devicetype* Optional argument. Specifies the type of rendering devices to be used. *devicetype* can only be one of the following strings:

- **`ardentx11`** A dynamic X11 window device will be used. This option is valid only if the underlying hardware supports the dynamic renderer.
- **`rasterfile`** The production renderer will be used and the output of the image is written to a disk file.
- **`seq_rasterfile`**. The production renderer will still be used but a sequence of image files will be produced. In this case the corresponding output data file must be specified. See below.

Default: **`ardentx11`**

Example: **`-dt seq_rasterfile`**

`-i` *datafile* Optional argument. Specifies the input data file which contains the states of the tank. The position and orientation information will be used to position the tank in the image. The velocity information will be used, together with the camera's state, to compute the range rate, which will be output to the description file.

Since right now the camera is fixed, the input data file only contains the state information about the tank, while the camera's state information

is hard-coded in the program. This relieves the burden of inputting data about camera's state information. The input data file consists of *data records* of the following form:

$$t \quad x_I \quad y_I \quad z_I \quad \dot{x}_I \quad \dot{y}_I \quad \dot{z}_I \quad R \quad S \quad T \quad \dot{R} \quad \dot{S} \quad \dot{T}$$

where x_I, y_I, z_I, $\dot{x}_I$, $\dot{y}_I$, $\dot{z}_I$ denote the position and velocity components of the tank at the instant t in global inertial coordinate system. R, S, T, $\dot{R}$, $\dot{S}$, $\dot{T}$ refer to the Eulerian angles and their rates.

Default: `dynamics.dat`
Example: `-i ~/data/tank.dat`

`-o` *imgfile* Optional argument. Specifies the filename for the output image. This option should be specified only if the production renderer is used and the output of the image is written to a disk file. This option also depends on the specification of "`-dt` *devicetype*".

- If "`-dt rasterfile`" is specified, the default file name for the image is `tank.img`.
- If "`-dt seq_rasterfile`" is specified, this option must be specified and take the following format:

  `-o s`*sceneNumber*`.img`*imageNumber*

  For example, if "`-o s1.img10`" is specified, a sequence of image files will be produced: `s1.img0010`, `s1.img0011`, `s1.img0012`, ... depending the number of data records contained in the input data file *datafile*. In this case the default is `s1.img0`.

`-c` Optional argument. This option should be specified only if the production renderer is used and the output of the image is written to a disk file. If this option is specified, the output Doré raster image file will be converted to *viff* image format and the resulting *viff* image is com-

pressed from 24-bits-per pixel to 8 bits per pixel. This viff byte image is further compressed using **compress(1)**.

-debug Optional argument. If specified, **tank** will print out some debugging information.

-track Optional argument. This option is valid only if a X11 dynamic device is used. If the dynamic renderer is used and this option is specified, a primitive tracker will be started and communication between the tracker and the camera is established via BSD sockets. This way the target will be always locked at the center of camera view.

-help Optional argument. When this argument is specified, **tank** prints out a brief help message and exits gracefully.

5.5.2 Input Data File

The input data file to the image generation program contains the complete state information about the tank and the camera at the sampling instants. Since right now the camera is fixed, the input data file only contains the state information about the tank, while the camera's state information is hard-coded in the program. This relieves the burden of inputting data about camera's state information. The input data file consists of *data records* of the following form:

$$t \quad x_I \quad y_I \quad z_I \quad \dot{x}_I \quad \dot{y}_I \quad \dot{z}_I \quad R \quad S \quad T \quad \dot{R} \quad \dot{S} \quad \dot{T}$$

where x_I, y_I, z_I, $\dot{x}_I$, $\dot{y}_I$, $\dot{z}_I$ denote the position and velocity components of the tank at the instant t in global inertial coordinate system. R, S, T, $\dot{R}$, $\dot{S}$, $\dot{T}$ refer to the Eulerian angles and their rates.

Note that program can handle the input data file quite flexibly. The input data record can be written almost any way the user wants, as long as there is the right number of data fields and those fields are separated

by ***white space characters***(space, tab, newline character(s)). A typical input data record can take, for example, the following form:

t
$x_I \; y_I \; z_I$
$\dot{x}_I \; \dot{y}_I \; \dot{z}_I$
$R \; S \; T$
$\dot{R} \; \dot{S} \; \dot{T}$

5.5.3 Output Description File

The output of the program consists of two parts: a single description file and a temporal sequence of raster image files. The description file and those image files constitute the image database. All the files reside in the same directory. The primary image file retrieving tool is **w_camera**, which is described in Section 4.3. The more sophisticated image database management is left for the future development.

The description file will be described in this section and the raster image files in next section.

The quantities in the description file are in SI units(i.e. length in meters, time in seconds, mass in kilograms, angle in radians) unless explicitly stated otherwise. The description file contains the following information:

- time at which the image is to be generated. This is identified by a " *keyword=value*" pair, with the keyword being **time**.
- the image identifier which maps to the corresponding image file. This is identified by a " *keyword=value*" pair, with the keyword being **image**.
- the image width in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **width**.

- the image height in number of pixels. This is identified by a " *keyword=value*" pair, with the keyword being **height**.
- the tank's state at that instant which includes

$$t \quad x_I \quad y_I \quad z_I \quad \dot{x}_I \quad \dot{y}_I \quad \dot{z}_I \quad R \quad S \quad T \quad \dot{R} \quad \dot{S} \quad \dot{T}$$

- the camera's state which includes

$$x_{cI} \; y_{cI} \; z_{cI} \quad \dot{x_{cI}} \; \dot{y_{cI}} \; \dot{z_{cI}} \; R_c \; S_c \; T_c \; \dot{R}_c \; \dot{S}_c \; \dot{T}_c$$
$$fov \; \dot{fov} \; hither \; yon \; \dot{yon}$$

where x_{cI}, y_{cI}, z_{cI}, $\dot{x_{cI}}$, $\dot{y_{cI}}$, $\dot{z_{cI}}$ denote the position and velocity components of the camera at the instant in global inertial coordinate system. R_c, S_c, T_c, $\dot{R}_c$, $\dot{S}_c$, $\dot{T}_c$ refer to the yaw, pitch , roll angles and their rates. fov and $\dot{fov}$ refer to the field of view(in degrees) and its time rate of change(in degrees per second), respectively; $hither$, $\dot{hither}$ denote the position of the front clipping plane and its rate in the camera coordinate system; yon, $\dot{yon}$ position of the back clipping plane and its rate in the camera coordinate system. The camera coordinate system is such that its initial orientation aligns with the inertial coordinate system, and the camera is always looking towards the negative z direction of the body-fixed right-handed coordinate system.

- range of the target to camera and range rate.

A typical data record in the output description file may look like this:

```
time=0.500000  image=s1.img0005  width=900  height=900
2000.000000 -95.388000 0.000000 0.044456 9.504300 0.000000
1.566100 0.000000 0.000000 0.006405 0.000000 0.000000
0.000000 0.000000 -5.000000 0.000000 0.000000 0.000000
```

```
0.000000 -0.002778 0.000000 0.000000 0.000000 0.000000
9.000000 0.000000 -1.000000 0.000000 -2500.000000 0.000000
2002.279668 -0.816753
```

5.5.4 Image Files

Image Data Format

The images in the database are in Khoros *viff* image format. For detailed information about viff image file format, please refer to the Khoros Reference Manual.

Image File Name

The image file names have the following format:

`s`N`.img`$XXXX$

For example, `s1.img0000`, `s1.img0001`, `s1.img0002`, ..., `s1.img1000`. *s1* stands for *scene 1*; `img0001` corresponds to the image at $.1th$ second, `img0002` at $.2th$ second, ..., `img1000` at $100th$ second.

Chapter 6

Conclusions

6.1 Summary of Achievements

With the very limited resources, we have successfully established the non-real time simulation capability for the passive, optically-based hierarchical tracking system. The achievements include:

- An image generation tool, which allows us to generate the temporal sequence of images of different scenarios.
- An image database and image retrieval tool(`w_camera`). The scenario for the established image database is simple, yet possesses the complexity to test the HiTert system against a real environment.
- An image processor, which segments the images and outputs target centroid position information.
- A tracker, which performs the optimal estimation of the target states.
- A predictor, which attempts to predict the future position of the target, based on the state estimation results from the tracker.
- An error analysis module, which can compute the *dartboard errors* in addition to computing the error statistics for the tracking system.

- An instrumentation module, which displays the status of the HiTert system during its operation. More specifically, this instrumentation module can display the world-view image, tracking camera image, segmented image, projectile hit points, trajectories, and error analysis results.

- A set of file-based IPC protocols and a generic subsystem, which consists of the world-view image database, tracking camera, image processor, tracker, predictor, error analysis and displaying/control module.

With such a testbed, we can *plug in* a different image processor, tracker or predictor with minimum efforts to study the different image processing and tracking algorithms. This generic subsystem lays the foundation upon which we can build multi-level tracking system and further study various aspects of the HiTert system.

The non-real time simulation allows the pre-generation of raw images and pre-segmentation of those imageries. It also the allows the separate study for image processing and tracking algorithms. These achievements realize the hierarchical target tracking system proposed in 1989.

6.2 Future Research

This research on the Hierarchical Target Extraction, Recognition and Tracking System is inherently multi-disciplinary in nature and covers diverse active research fields. We have built a prototype for the HiTert system. More in-depth study is needed in further exploring the multi-disciplinary environment existing at Purdue. Many problems need to be addressed in the full implementation of the HiTert system. These problems include:

- Battle field scene simulation:
    - Terrain rendering for the rough surface.

  - Target motion-blur simulation.
  - Environment simulation.

- Image database management.
- Image Processing: Extraction of the target orientation information.
- Tracking: More advanced tracker need to be developed
  - Utilization of elevation data from a digital terrain map.
  - Utilization of orientation information
- Coordination of the subsystems.

Appendix A

α-β-γ Tracker Design

A.1 Introduction

The classical problem of the fire control system is the accurate prediction of the future position of a target at the time of the projectile intercept. Once the future position of the target has been obtained, the corresponding gun pointing lead-angles can be determined. Current approaches to the solution of this problem typically consists of two parts: estimation and prediction[12, 13]. Kalman filtering techniques are employed to estimate the velocity and acceleration of the target based on the noisy measurements of the target positions. Based upon the estimate for the velocity and acceleration, a prediction of the future position of the target can be ascertained.

One of the key issues in the modeling of the target dynamics has been how to model the target acceleration[12, 13, 14, 15, 16]. Singer[12] used the temporally exponentially correlated acceleration model, viewing the accelerations as the perturbations to the constant velocity trajectory. Berg[13] proposed the addition of the acceleration correction term to the Singer model, which represents an adaptive estimate of the mean target jerk(time rate of the change of the target acceleration). Kendrick et al.[14] suggested to use the orientation measurements to improve the estimate of the air-borne target

acceleration. Andrisani et al.[15, 16] further extended the idea of incorporation of orientation measurements into the trackers by developing nonlinear dynamic models of fixed-wing aircraft and helicopters for extended Kalman filters[19].

Thus far, the ground vehicle tracking remains to be an open research area. Because of the resource constraints, however, it is not our intent to address advanced trackers for the ground vehicles here. Instead, a simple non-adaptive α-β-γ tracker will be developed for the prototype HiTert system.

A.2 Target State Estimator Equation

A.2.1 State Equation

In a single physical dimension, the target state equations may be represented by

$$\dot{x} = Fx + Gw \tag{A.1}$$

where

$$x = \begin{bmatrix} x_i \\ \dot{x}_i \\ \ddot{x}_i \end{bmatrix}, \; F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \; G = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

x is the target state vector; x_i, $\dot{x}_i$ and $\ddot{x}_i$ are the inertial position, velocity and acceleration in ith component of the inertial Cartesian coordinate system. A is the system matrix, B the input vector; w is the zero-mean white noise process with intensity Q. It is assumed that the target jerking term as modelled by the white noise intensity Q is the same in each inertial Cartesian coordinate. Also assumed is that the exogenous white noise process in each dimension is independent of or uncorrelated with each other.

A.2.2 Measurement Equation

A single passive optical sensor can only yield the bearing angles or additionally bearing angle rates. The absence of range measurements will lead to the lack of observability in certain scenarios[17]. To remedy this, one may assume availability of either the range measurements or another passive optical sensor to triangulate the target. For now, we assume for simplicity that the position measurements are available indirectly via the transformation of the measurement data. Also the measurements are discrete in nature and are available at the sampling instants. With these assumptions the measurement equation can be written as

$$z(k) = Cx(k) + v(k) \tag{A.2}$$

where z is the measured position of a single dimension of the inertial cartesian coordinate systems, x the state vector, v the zero-mean white noise process with intensity R; C is the measurement vector and has the following value:

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

A.2.3 Discretization

Let T denote the sampling period. It is well known[20, 21] that the solution to the linear time invariant state equation (A.1) is

$$\begin{aligned} x(t+T) &= e^{FT}x(t) + \int_t^{t+T} e^{F(t+T-\tau)}Gw(\tau)d\tau \\ &= e^{FT}x(t) + \int_0^T e^{F\sigma}Gw(t+T-\sigma)d\sigma \end{aligned} \tag{A.3}$$

Since

$$F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

is in Jordan form, A is nilpotent. It is easy to see that

$$e^{Ft} = \begin{bmatrix} 1 & t & t^2/2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

Also note that under the zero-order hold approximation,

$$w(t+T-\sigma) = w(t) \quad \text{for } 0 \leq \sigma \leq T \tag{A.5}$$

Therefore

$$\begin{aligned} \int_0^T e^{F\sigma} G w(t+T-\sigma) d\sigma &= \int_0^T \begin{bmatrix} 1 & \sigma & \sigma^2/2 \\ 0 & 1 & \sigma \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w(t) d\sigma \\ &= \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix} w(t) \end{aligned} \tag{A.6}$$

Finally, combining the Equations (A.3)–(A.6) gives, with the slight abuse of the notations, the state equation can be written as

$$x(k+1) = Ax(k) + Bw(k) \tag{A.7}$$

where

$$A = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix}$$

Using the observability test[20, 21], it is easy to see that the $\{C, A\}$ in Equations (A.2) and (A.7) form an observable pair.

A.3 Target State Estimator Gains

Kalman filter provides the optimal solution in the sense of minimizing the mean square estimation error. In general, it can also be easily implemented.

The Kalman filter state equations are[21]

$$\bar{x}(k+1) = A\hat{x}(k) \tag{A.8}$$

$$\hat{x}(k) = \bar{x}(k) + L(k)[y(k) - C\bar{x}(k)] \tag{A.9}$$

where

$$L(k) = M(k)C^T(CM(k)C^T + R)^{-1} \tag{A.10}$$

$$P(k) \triangleq E\{[x(k) - \hat{x}(k)][x(k) - \hat{x}(k)]^T\}$$

$$= M(k) - M(k)C^T(CM(k)C^T + R)^{-1}CM(k) \tag{A.11}$$

$$M(k+1) \triangleq E\{[x(k+1) - \bar{x}(k+1)][x(k+1) - \bar{x}(k+1)]^T\}$$

$$= AP(k)A^T + BQB^T \tag{A.12}$$

$$M(0) = E\{\tilde{x}(0)\tilde{x}^T(0)\} \tag{A.13}$$

The steady-state Kalman filter gain is given by

$$L = MC^T(CMC^T + R)^{-1}$$

$$M = A(M - MC^T(CMC^T + R)^{-1}CM)A^T + BQB^T$$

It should be noted that the state estimator governed by Equations (A.8) and (A.9) is also known as the *current estimator*[21] because the estimate of $x(k)$ uses the measurements up to and including those at k. However, the image based measurement process is time-consuming compared to the sampling period T. Thus, it is not realistic assuming the measurement $y(k)$ is available for the estimation of $x(k)$. Therefore, the one-step ahead *prediction estimator* should be used instead. Substituting (A.9) into (A.8) yields state equation for the *prediction estimator*

$$\bar{x}(k+1) = A\bar{x}(k) + F(k)[y(k) - C\bar{x}(k)] \tag{A.14}$$

where

$$F(k) = AL(k) \tag{A.15}$$

A.4 Predictor Equation

A simple second-order predictor is employed using the following equation

$$\hat{p}_i(t + t_p|t) = \hat{p}_i(t) + \hat{v}_i(t)t_p + \frac{1}{2}\hat{a}_i(t)t_p^2 \qquad \text{(A.16)}$$

where t_p is the projectile time-of-flight, $\hat{p}_i(t)$, $\hat{v}_i(t)$ and $\hat{a}_i(t)$ are the current estimate for the position, velocity and the acceleration in the ith inertial Cartesian coordinate, respectively. Implicit in the above prediction equation is the assumption that the target's linear velocity and acceleration are constant during the time-of-flight interval.

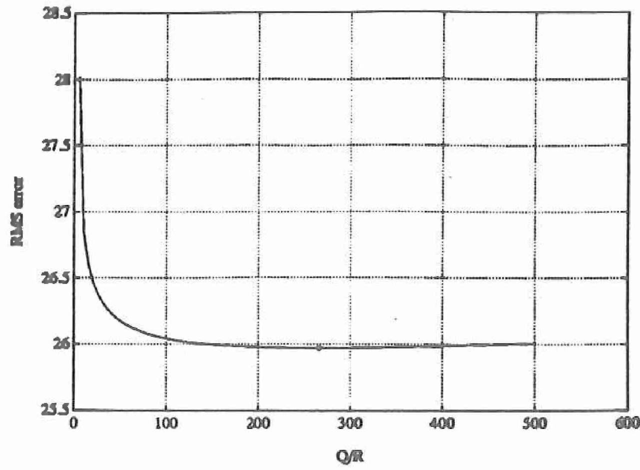
The performance of a predictor is often evaluated in terms of the prediction error. The performance metric used here is the average prediction error distance, which is defined as

$$\epsilon_d = \sqrt{\frac{\sum[(x - \hat{x})^2 + (y - \hat{y})^2 + (z - \hat{z})^2]}{N}} \qquad \text{(A.17)}$$

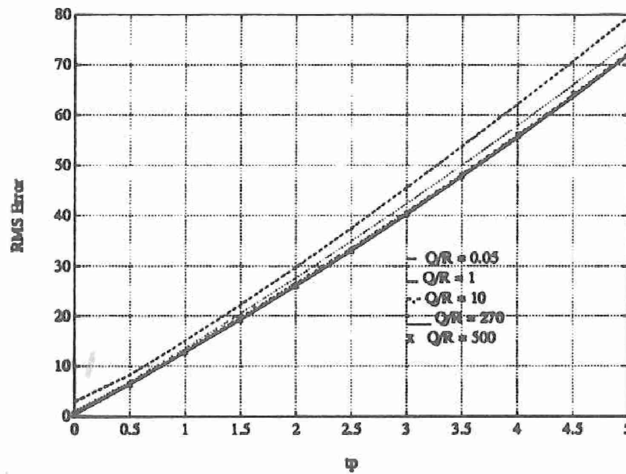
where $(\hat{x}, \hat{y}, \hat{z})$ is the *predicted* position of the target, (x, y, z) the corresponding exact position of the target, N the number of the prediction data points.

A.5 Numerical Results

Taking the image processor capability into account, the measurement noise covariance is assumed to have the value $R = 0.1\ m^2$. The white noise covariance Q modelling the target jerking is somehow arbitrary. In fact, Q is a design parameter at our disposal. Figure A.1-a shows the RMS 2-second ***prediction*** error ϵ_d vs. the ratio of Q/R. The suboptimal prediction occurs at $Q/R = 270$. The corresponding optimal Kalman filter gain is 0.5057, 1.1352 and 1.2755. Figure A.1-b shows the parametric behavior of the ***prediction*** error ϵ_d vs. the prediction lead time t_p, when the design parameter Q/P varies. For comparison, Figure A.2 shows the RMS prediction error for an

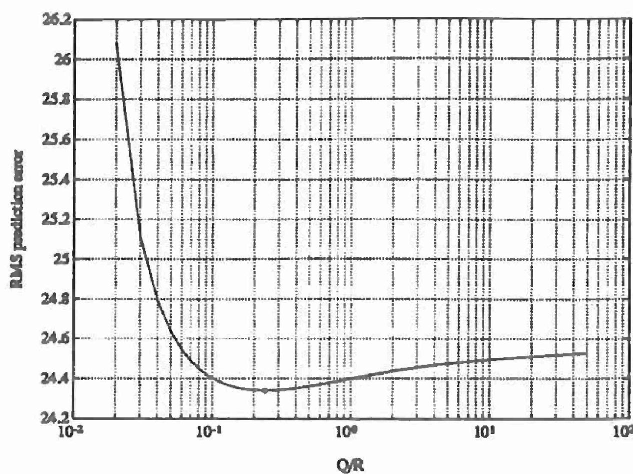


(a) Influence of $Q/R(t_p = 2$ second)

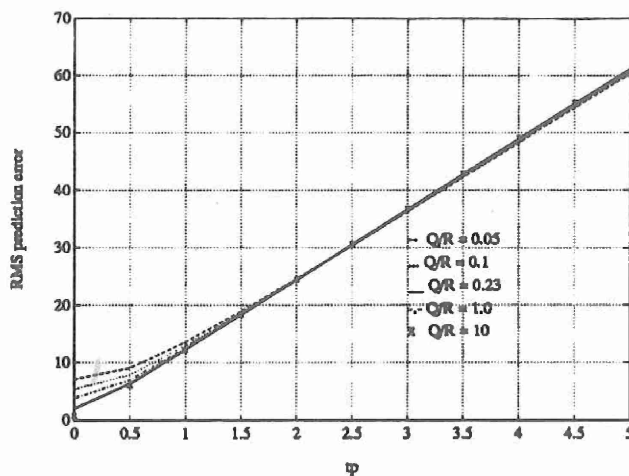


(b) Parametric behavior

Figure A.1: α-β-γ tracker RMS prediction error



(a) Influence of $Q/R(t_p = 2 \text{ second})$



(b) Parametric behavior

Figure A.2: α-β tracker RMS prediction error

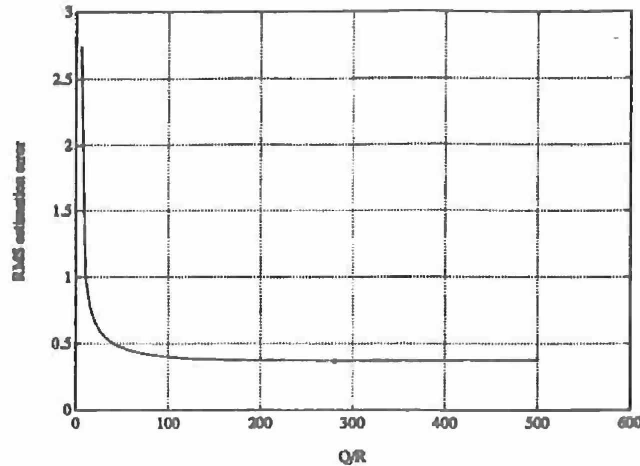


Figure A.3: α-β tracker RMS estimation error($t_p = 0$ second)

α-β tracker. It can be seen from the plots that the suboptimal α-β tracker can achieve the approximately same performance as the the suboptimal α-β-γ tracker. This is mainly because the non-adaptive nature of the trackers, which are employed for the state estimation for the whole time history of the trajectory consisting of the various segments of different maneuvers. However, it should be noted that the behavior of the *prediction* error versus Q/R may not be the same as the state *estimation* error versus Q/R. Figure A.3 show the α-β tracker RMS estimation error($t_p = 0$ second) versus the ratio Q/R. Note that Figure A.3-a and Figure A.3 has the different abscissa scale.

A.6 Conclusions

A simple α-β-γ tracker/predictor has been designed to minimize the prediction error. Comparison shows that the α-β-γ tracker and α-β tracker can achieve the approximately same performance. This suggest that advanced tracker design method need to be studied and the ground vehicle tracking still remains to be an open research area.

Appendix B

Dartboard Analysis

While the prediction errors in each inertial Cartesian components are good metrics of the performance of a tracking system, it is often desirable to have the tracking results graphically displayed in a form suitable for human consumption. This is particularly important for the operator of a fire control system. The dartboard analysis presented here mimics the *fictitious* target-carried dartboard as seen by an observer.

B.1 Definitions

Figure B.1 is a schematic at an arbitrary instant t. Point P_0 is the *exact* position of the target at t, Point $\hat{P}$ the *predicted* position at t[1], O the location of the observer. The *up* direction of the observer at the time t is specified by the unit vector $\vec{u}$.

The *eye coordinate system* $Oxyz$ is determinated as follows: the positive z direction is opposite to the vector $\overrightarrow{OP_0}$, i.e.,

$$\vec{z} = -\overrightarrow{OP_0}/|\overrightarrow{OP_0}| \tag{B.1}$$

[1]The prediction was performed at $(t - t_p)$, where t_p is the predict-ahead or lead time.

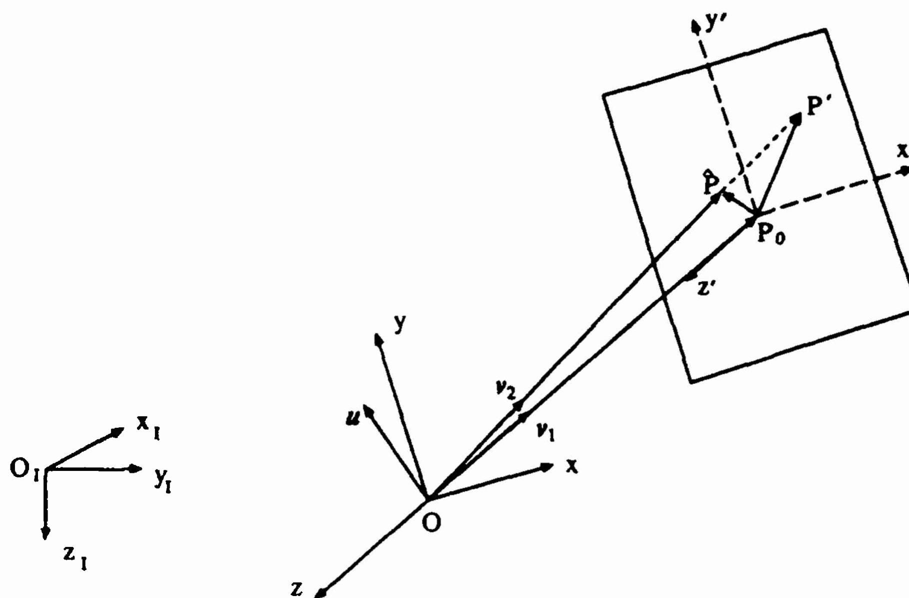


Figure B.1: Dartboard schematic

the x-axis is determined by

$$\vec{z} = \vec{u} \times \vec{z} \tag{B.2}$$

and the y-axis completes the triad of the right-hand coordinate system:

$$\vec{y} = \vec{z} \times \vec{x} \tag{B.3}$$

Note that the *up* direction is not necessarily perpendicular to the line of sight(LOS) of the observer. The only requirements for the *up* direction vector $\vec{u}$ are that $\vec{u}$ is in the "*vertical*" plane determined by the y and z axes and that $\vec{u}$ has a positive component along the y axis.

The *dartboard plane* π contains the point P_0 and is perpendicular to the vector $\overrightarrow{OP_0}$. The x'-axis, y'-axis and z'-axis of the *dartboard coordinate system* $P_0x'y'z'$ are parallel to the Ox, Oy and Oz axes, respectively.

The ***hit point*** P' of a projectile is the intersection of the line $O\hat{P}$ and the dartboard plane π. The vector $\overrightarrow{P_0P'}$ is the ***perspective projection*** of the ***prediction error vector*** $\overrightarrow{P_0\hat{P}}$ onto the dartboard plane π. The vector $\overrightarrow{P_0P'}$ is the ***dartboard error vector***. Resolving the vector $\overrightarrow{P_0P'}$ in the coordinate system $P_0x'y'z'$ yields the ***dartboard error components***[1].

It should be pointed out that both the location and the orientation of the dartboard plane π change with the time. The location and ***up*** direction of the observer also change with the time if the observer is traveling with a moving vehicle. However, in computing the dartboard error vector $\overrightarrow{PP'}$, it is assumed that the observer and the dartboard are ***fixed*** during the time interval that it takes for a projectile to travel from P to P'.

B.2 The Hit Point

We shall derive the vector equation for finding the ***hit point*** P'. In the following, we use an arrow above a single upper case letter to denote the ***position*** vector for the corresponding point. All the ***position*** vectors have the common starting point, say, O_I, which is the origin of the inertial reference frame.

Let $\vec{v_1}$ and $\vec{v_2}$ be the ***unit direction*** vectors for $\overrightarrow{OP_0}$ and $\overrightarrow{O\hat{P}}$, respectively. The vector form equation for the line $O\hat{P}$ is given by

$$\vec{P} = \vec{O} + \rho\vec{v_2} \tag{B.4}$$

where $\vec{P}$ is the ***position*** vector of an arbitrary point P on the line, ρ a real scalar quantity[2], $\vec{O}$ the ***position*** vector of the point O.

The vector form equation for the dartboard plane π is

$$(\vec{P} - \vec{P_0}) \cdot \vec{v_1} = 0 \tag{B.5}$$

[1]The error component in the z' axis is always zero

[2]The absolute value of ρ is the distance between the points P and O.

where $\vec{P}$ is the *position* vector of an arbitrary point P on the plane, $\vec{P_0}$ the *position* vector of the point P_0. Substituting Equation B.4 into B.5 gives

$$(\vec{O} + \rho\vec{v_2} - \vec{P_0}) \cdot \vec{v_1} = 0 \tag{B.6}$$

Solving Equation B.6 for ρ yields

$$\rho = \frac{(\vec{P_0} - \vec{O}) \cdot \vec{v_1}}{\vec{v_1} \cdot \vec{v_2}} \tag{B.7}$$

Substituting ρ back into Equation B.4 gives the *hit point* P'

$$\vec{P'} = \vec{O} + (\frac{(\vec{P_0} - \vec{O}) \cdot \vec{v_1}}{\vec{v_1} \cdot \vec{v_2}})\vec{v_2} \tag{B.8}$$

The solution for the *position* vector $\vec{P'}$ of the the *hit point* P' is independent of any coordinates system, since vectors are coordinates system independent[1].

B.3 Coordinate Transformation

The *coordinates* of the hit point position vector $\vec{P'}$ in Equation B.8 can be most conveniently resolved in the inertial Cartesian coordinate system $O_I x_I y_I z_I$, as all the relevant vector quantities in the right hand side of Equation B.8 are usually expressed in the inertial coordinate system. Given the inertial coordinates of these vectors, finding the coordinates for the position vector $\vec{P'}$ is quite trivial.

However, to find out the *dartboard errors*, the inertial coordinates for $\vec{P'}$ must be transformed into the the *dartboard coordinate system* $P_0x'y'z'$. This transformation can be done in the two steps: translation and then rotation.

First, the inertial coordinate system $O_I x_I y_I z_I$ need to be translated to the origin P_0 of the *dartboard coordinate system*. The application of this

[1]All the position vectors, however, have the same reference point.

translation transformation to the hit point P' effectively yields the dartboard error vector $\overrightarrow{P_0P'}$, where

$$\overrightarrow{P_0P'} = \vec{P'} - \vec{P_0} \tag{B.9}$$

Next, we need to apply the rotation transformation to the dartboard error vector $\overrightarrow{P_0P'}$. Since the ***dartboard coordinate system*** has the same orientation as the eye coordinate system $Oxyz$, we can use the direction matrix of the coordinate system $Oxyz$ relative to the inertial coordinate system $O_I x_I y_I z_I$. The direction cosine matrix which transforms from $O_I x_I y_I z_I$ to $Oxyz$ is given by

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \tag{B.10}$$

where x_1, x_2 and x_3 are the components of the unit vector $\vec{x}$ in the inertial x_I, y_I and z_I directions, respectively. Similar remarks apply to y_1, y_2, y_3, z_1, z_2 and z_3. The unit vectors $\vec{x}$, $\vec{y}$ and $\vec{z}$ of the ***eye coordinate system Oxyz*** are given by Equations B.1 ~ B.3.

Concatenating the translation and rotation, we have the ***dartboard error components*** x', y' and z' given by

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} (\overrightarrow{P_0P'})_1 \\ (\overrightarrow{P_0P'})_2 \\ (\overrightarrow{P_0P'})_3 \end{bmatrix} \tag{B.11}$$

where $(\overrightarrow{P_0P'})_1$, $(\overrightarrow{P_0P'})_2$ and $(\overrightarrow{P_0P'})_3$ are the inertial components of the databoard error vector $\overrightarrow{P_0P'}$ in the x_I, y_I and z_I directions, respectively.

Appendix C

Geometrical Transformation

C.1 Introduction

Geometrical transformation is one of the core parts of 3-dimensional computer graphics. In addition to its importance in the modelling and rendering object hierarchies, geometrical transformation is also an integral part of the image-based tracking system. For example, given a world coordinate (x_w, y_w, z_w) what is the corresponding point in the image ? Conversely, given a screen coordinate (x_s, y_s) what is the corresponding line-of-sight(LOS) in the world ? To answer these questions, a series of geometrical transformations must be performed to yield the solution.

A thorough discussion of the subject is beyond the scope. We shall outline the highlights of the geometrical transformation by deriving or summarizing the key formulas from an engineering point of view, leaving aside other graphics issues such as clipping, hidden surface removal and lighting models. Interested readers may refer to Foley et. el.[9] or Newman et. el.[10].

Since the images are generated with the aid of 3D graphics system Doré[11], discussions will follow Doré conventions when necessary.

C.2 Viewing Transformation

C.2.1 World to Eye Coordinate Transformations

In order to simplify the transformations of world coordinates into the graphical output device coordinates, the world coordinates are first transformed into the *eye* coordinates system. The eye coordinate system relative to the

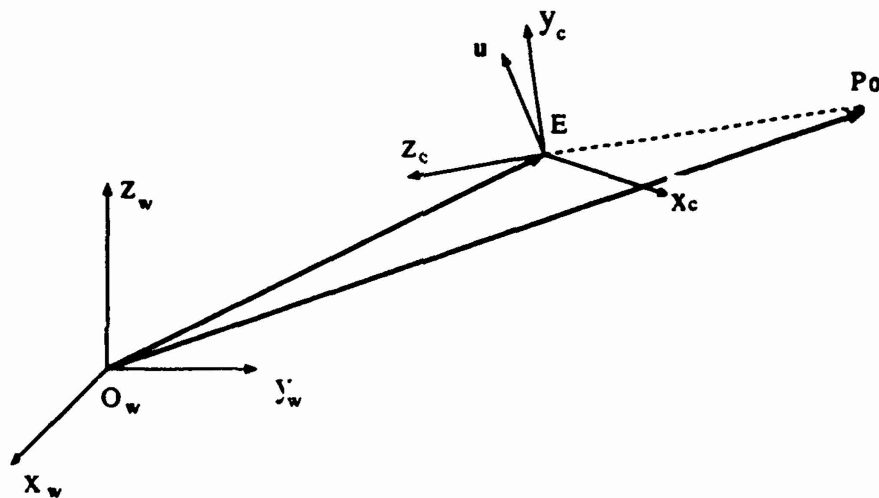


Figure C.1: World to eye coordinate transformation

world coordinate system is completely determined by the location and orientation of the eye coordinate system. The eye point or the location of the camera is given by the position $\vec{E}(E_x,\ E_y,\ E_z)$ (See Figure C.1). Suppose that the camera is looking at P_0(the center of interest). To be consistent with Doré 3D graphics system, which is used to generate the world view images of a simply battle field scene, the positive direction of the z_c axis is *opposite* to the camera boresight. Also assume that the *up* direction of the camera is

given by the unit vector $\vec{u}$[1]. The orientation or the unit vectors $\vec{l}$, $\vec{m}$ and $\vec{n}$ of the camera coordinate system are then given as follows:

$$\vec{n} = -\frac{\overrightarrow{EP_0}}{|\overrightarrow{EP_0}|} \tag{C.1}$$

$$\vec{l} = \vec{u} \times \vec{n} \tag{C.2}$$

$$\vec{m} = \vec{n} \times \vec{l} \tag{C.3}$$

A *column* vector will be used to represent the coordinates of a point. The *direction cosine* matrix, which transforms the world coordinates of a fixed point into the eye or camera coordinates, is given by

$$R = \begin{bmatrix} l_x & l_y & l_z \\ m_x & m_y & m_z \\ n_x & n_y & n_z \end{bmatrix} \tag{C.4}$$

where l_x, l_y and l_z are the components of the unit vector $\vec{l}$ of the x_c axis along the world x_w, y_w and z_w axes, respectively. Similar remarks apply to $\vec{m}$ and $\vec{n}$.

The transformation from the world to eye coordinates consists of two steps: translation and rotation. Under the *homogeneous* coordinates[9, 10], the associated matrices are:

$$M_T(E_x, E_y, E_z) = \begin{bmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 1 & -Ez \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C.5}$$

$$M_R = \begin{bmatrix} l_x & l_y & l_z & 0 \\ m_x & m_y & m_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C.6}$$

[1] $\vec{u}$ is not necessarily the same as the "up" direction of the world. The only requirement for u is that u lies in the half plane determined by the z_c axis and the *positive* y_c axis.

Concatenation of the two matrices, i.e. *pre*multiplying M_R with M_T, yields

$$M_{w2e} = \begin{bmatrix} l_x & l_y & l_z & -\vec{l}\cdot\vec{E} \\ m_x & m_y & m_z & -\vec{m}\cdot\vec{E} \\ n_x & n_y & n_z & -\vec{n}\cdot\vec{E} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (C.7)$$

C.2.2 Scaling Compensation

In order to avoid the nonuniform scaling in the horizontal and vertical directions, the following scaling transformation is performed after the world to eye transformation:

$$M_{scale} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (C.8)$$

where

$$\begin{cases} s_x = height/width; \quad s_y = 1 & \text{if } width > height \\ s_x = 1; \quad s_y = height/width & \text{if } width <= height \end{cases}$$

width and *height* are the sizes of an image.

C.2.3 Perspective Transformation

In order to achieve the 3-D effects on a 2-D graphical output device, the next step is to perform the ***perspective transformation*** in the eye coordinate system. The 3D ***viewing frustum*** is determined by the field of viewing angle, the hither and yon clipping plane(See Figure C.2 - a). In Doré, the camera is located at the origin and looks towards the negative direction of the z-axis of the camera coordinate system. Thus, the hither and yon planes must be specified by the *negative* floating numbers h and y, respectively($|h| \leq |y|$). In order to facilitate the clipping process, the viewing frustum is also transformed ino the *canonical viewing volume*[9]. This canonical viewing

volume extends evenly 2 units of distance in the x and y directions, and extends in the negative z direction 1 unit of distance. See Figure C.2 - b.

The transformation which maps the hither and yon clipping planes into the planes $z = 0$ and $z = -1$ in addition to the perspective projection is given by

$$M_{persp} = \begin{bmatrix} \frac{1}{\tan\frac{\alpha}{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\frac{\alpha}{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{1-h/y} & \frac{-h}{1-h/y} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (C.9)$$

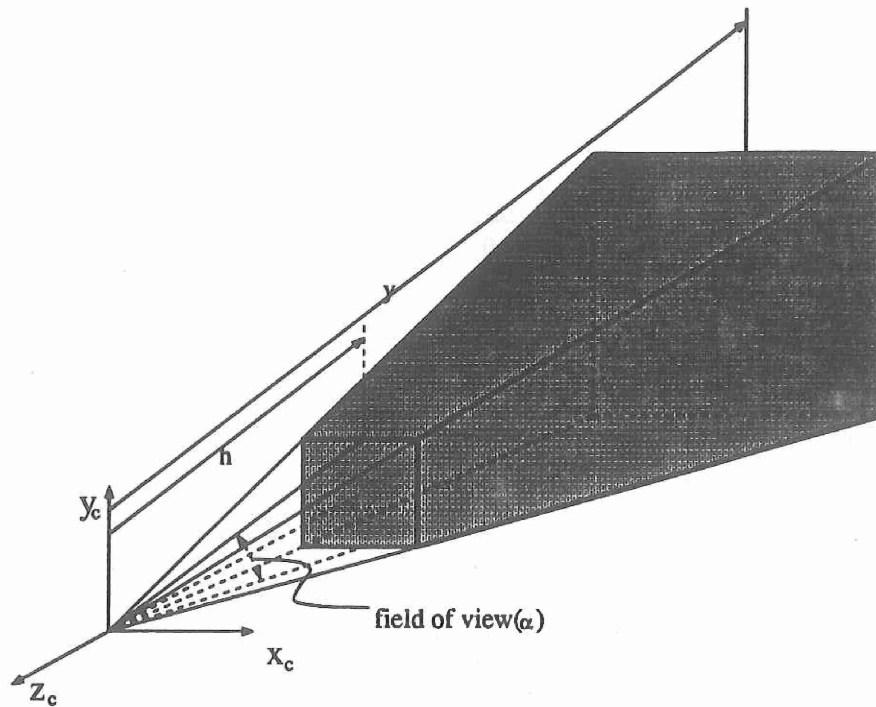
It should be pointed out that this transformation is *nonlinear:* a line will not, in general, be mapped into a line. A line-of-sight(LOS) originating form the eye point will, however, be mapped into a line parallel to the z axis. This observation is important. It plays an key role in the inverse process of determining the LOS given the screen coordinates.

C.2.4 Screen Transformation

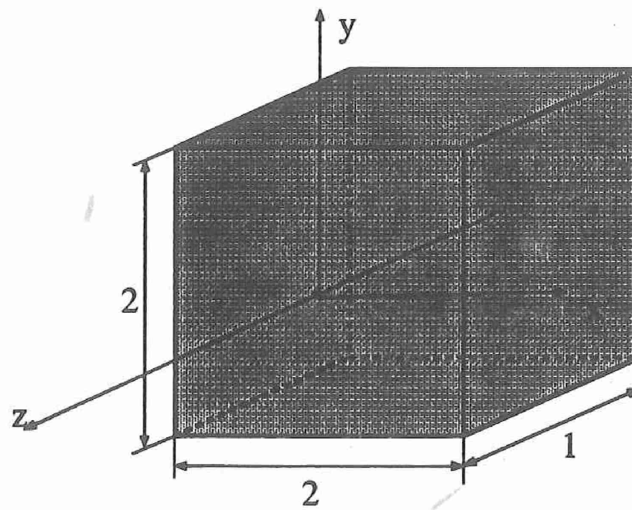
After the perspective projection and clipping, the *frontmost* in the canonical volume is mapped onto the physical 2D graphical output device. Let us consider the case where the 2D graphical output device is an X window. According to the convention in X, the upper left hand corner of the window has the coordinates $(0,0)$. The positive x_s direction is from left to right, while the positive direction for y_s is vertically downward. See Figure C.3.

Consider the rectangular output area of width W and height H with its centroid located at (x_c, y_c). The transformation which maps the normalized 2×2 screen to the above physical graphical output device is given by

$$M_{screen} = \begin{bmatrix} W/2 & 0 & 0 & x_c \\ 0 & -H/2 & 0 & y_c \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (C.10)$$



(a) Viewing frustum



(b) Canonical viewing volume

Figure C.2: Perspective transformation

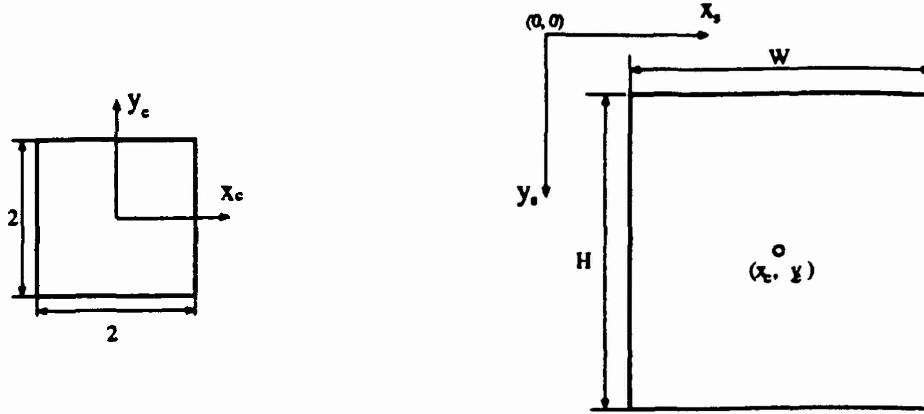


Figure C.3: Screen transformation

C.2.5 Concatenation of Transformations

The overall world to screen transformation is the concatenation of above series of transformations:

$$M_{w2s} = M_{screen} M_{persp} M_{scale} M_{w2e} \tag{C.11}$$

Carrying out the matrix multiplications yields:

$$M_{w2s} = \begin{bmatrix} al_x - x_c n_x & al_y - x_c n_y & al_z - x_c n_z & -a(\vec{l} \cdot \vec{E}) + x_c(\vec{n} \cdot \vec{E}) \\ bm_x - y_c n_x & bm_y - y_c n_y & bm_z - y_c n_z & -b(\vec{m} \cdot \vec{E}) + y_c(\vec{n} \cdot \vec{E}) \\ cn_x & cn_y & cn_z & -c[(\vec{n} \cdot \vec{E}) + h] \\ -n_x & -n_y & -n_z & \vec{n} \cdot \vec{E} \end{bmatrix} \tag{C.12}$$

where

$$a = \frac{W s_x}{2 \tan \frac{\alpha}{2}} \tag{C.13}$$

$$b = -\frac{H s_y}{2 \tan \frac{\alpha}{2}} \tag{C.14}$$

$$c = \frac{1}{1 - h/y} \tag{C.15}$$

C.3 Screen to World Transformations

Given the screen coordinates (x_s, y_s), it is insufficient to find the point in the world coordinates system, as the depth value z_s is unknown. However, one can find the LOS of the corresponding point in the world coordinate system. As remarked in Section C.2.3, given the fixed x_s, y_s, the screen coordinates (x_s, y_s, z_s) will yield the same LOS for any values $-1 \leq z_s \leq 0$. With the range information, one can determine the corresponding point in the world coordinate system.

The inverse mapping from the screen to world coordinate is given by

$$\begin{aligned} M_{s2w} &= M_{w2s}^{-1} \\ &= M_{w2e}^{-1} M_{scale}^{-1} M_{persp}^{-1} M_{screen}^{-1} \end{aligned} \tag{C.16}$$

where

$$\begin{aligned} M_{w2e}^{-1} &= M_T^{-1}(E_x, E_y, E_z) M_R^{-1} \\ &= M_T(-E_x, -E_y, -E_z) M_R^T \\ &= \begin{bmatrix} 1 & 0 & 0 & E_x \\ 0 & 1 & 0 & E_y \\ 0 & 0 & 1 & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_x & m_x & n_x & 0 \\ l_y & m_y & n_y & 0 \\ l_z & m_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} l_x & m_x & n_x & E_x \\ l_y & m_y & n_y & E_y \\ l_z & m_z & n_z & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

$$M_{scale}^{-1} = \begin{bmatrix} s_x^{-1} & 0 & 0 & 0 \\ 0 & s_y^{-1} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{persp}^{-1} = \begin{bmatrix} \tan\frac{\alpha}{2} & 0 & 0 & 0 \\ 0 & \tan\frac{\alpha}{2} & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -\frac{1-h/y}{h} & -\frac{1}{h} \end{bmatrix}$$

$$M_{screen}^{-1} = \begin{bmatrix} \frac{2}{W} & 0 & 0 & -\frac{2}{W}x_c \\ 0 & -\frac{2}{H} & 0 & \frac{2}{H}y_c \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Carrying out matrix multiplications yields

$$M_{s2w} = \begin{bmatrix} \tilde{a}l_x & \tilde{b}m_x & \tilde{f}E_x & \tilde{c}l_x + \tilde{d}m_x - n_x + \tilde{e}E_x \\ \tilde{a}l_y & \tilde{b}m_y & \tilde{f}E_y & \tilde{c}l_y + \tilde{d}m_y - n_y + \tilde{e}E_y \\ \tilde{a}l_z & \tilde{b}m_z & \tilde{f}E_z & \tilde{c}l_z + \tilde{d}m_z - n_z + \tilde{e}E_z \\ 0 & 0 & \tilde{f} & \tilde{e} \end{bmatrix} \quad \text{(C.17)}$$

where

$$\tilde{a} = a^{-1} = \frac{2}{Ws_x}\tan\frac{\alpha}{2} \quad \text{(C.18)}$$

$$\tilde{b} = b^{-1} = -\frac{2}{Hs_y}\tan\frac{\alpha}{2} \quad \text{(C.19)}$$

$$\tilde{c} = -\tilde{a}x_c \quad \text{(C.20)}$$

$$\tilde{d} = -\tilde{b}y_c \quad \text{(C.21)}$$

$$\tilde{e} = -1/h \quad \text{(C.22)}$$

$$\tilde{f} = \tilde{e}(1 - h/y) \quad \text{(C.23)}$$

As a check, one can verify that the following holds:

$$M_{w2s}M_{s2w} = M_{s2w}M_{w2s} = I$$

Appendix D

Manual Pages

The manual pages, in the standard UNIX on-line **man(1)** page format, are listed here for reference. These man pages are intended to be a quick reference for the usage of the command line options. The specification of the file formats are omitted.

If you have set your environment variable MANPATH *appropriately, you* can access manual pages on-line by typing, for example,

```
example% man hitert
```

NAME

dataControl – **Data** Control module

SYNOPSIS

dataControl -imgInfo *imgInfoFile* -segInfo *segInfoFile*

-t0 *start-time* -te *end-time* -oimgInfo *out-imgInfoFile* -osegInfo *out-segInfoFile* -traj *out-exact-traj* -spec *out-spec*

DESCRIPTION

dataControl takes the description files for the world view image database and the segmented image database, respectively, as inputs. It outputs the selected segments of the description files, exact trajectory data file and the spec file containing the information for the starting and terminating point.

OPTIONS

-imgInfo *imgInfoFile* Required argument. *imgInfoFile* is the name of the description file for the world-view image database of the whole time history.

-segInfo *segInfoFile* Required argument. *segInfoFile* is the name of the description file for the segmented image database of the whole time history.

-t0 *start-time* Required argument. *start-time* specifies the time at which the tracking system should start to run.

-te *end-time* Required argument. *end-time* specifies the time at which the tracking system should terminate. By specifying *start-time* and *end-time*, one can ask system to run over the selected segment of the trajectory of interest instead of the whole time history.

-oimgInfo *out-imgInfoFile* Required argument. *out-imgInfoFile* specifies the name of description file for the selected segment of the world-view image database.

-osegInfo *out-segInfoFile* Required argument. *out-segInfoFile* specifies the name of description file for the selected segment of the segmented image database.

-traj *out-exact-traj* Required argument. *out-exact-traj* specifies the name of file containing selected segment of exact trajectory data.

-spec *out-spec* Required argument. *out-spec* specifies the name of file containing information corresponding to the starting and terminating points.

SEE ALSO

hitert(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

NAME

w_camera – Returns the sequence of user specified input files.

SYNOPSIS

w_camera [-dir *directory*] [-prefix *prefix*] [-num_width *num-width*] [-suffix *suffix*] -inSpec *inSpec* -outSpec *outSpec* [-img *image-file*] [-uc <1 or 0>] [-help]

DESCRIPTION

w_camera returns an desired input file according to the user specified directory, prefix, sequence number, suffix. The sequence numbers are extracted from the *inSpec* file. The *inSpec* file consists of the sequence numbers, data records from the description file for the world-view image database and data records from the description file for the segmented image database. The *outSpec* consists of the sequence numbers, image file names, data records from the description file for the world-view image database and data records from the description file for the segmented image database. If the input filename has ".Z", it is assumed to be in the compressed format. The compressed input files will be uncompressed to produce the output image file if the "-uc 1" option is specified. (this is also the default). However, if "-uc 0" is specified and if the input files have the tag ".Z", the compressed image files will be copied to produce the output image files. In this case, the output image files are tagged with ".Z" to indicate that they are compressed files.

No assumption has been made that the prefix, sequence and suffix are separated by the character period ".". If they are indeed separated by such a character, they should be explicitly specified either in the prefix or suffix part.

OPTIONS

[-dir *directory*] Optional argument. This option specifies the directory from which the input file is to be picked up. It is an optional argument. Default: the current directory.

[-prefix *prefix*] Optional argument. This option specifies the part in the input file name that precedes the sequence number. It is an optional argument. Default: null.

[-num_width *num-width*] Optional argument. The width that the numeric sequence takes may be specified through this optional argument. If the *num-width* is greater than the number of the *digits* the *sequence-number* (as specified in -num) has, the *sequence-number* will be *prepended* with zero's(0's) so that the width of the sequence string will be the same as *num-width*. Default: the sequence string is as long as the the number of digits that the sequence number has.

[-suffix *suffix*] Optional argument. The option specifies the part of the filename that *follows* the sequence string. If the last two characters of the suffix string are ".Z", then the input file will be assumed to in compressed format, and uncompressed data file will be produced using *uncompress(1)*. Default: null

-inSpec *inSpec* Required argument. *inSpec*

specifies the name of a input *specification* file. This file is not actual output "data or image" file; rather, it contains data records of the form: *sequence-number* followed by a data record from the description file for the world-view image database and a data record from the description file for the segmented

image database, which are separated by the white space characters(space, tab, newline).

-outSpec *outSpec* Required argument. *outSpec*

specifies the name of a output *specification* file. This file is not actual output "data or image" file; rather, it contains data records of the form: *sequence-number*, followed by the file name for the corresponding output image, followed by a data record from the description file for the world-view image database and a data record from the description file for the segmented image database, which are separated by the white space characters(space, tab, newline).

[**-img** *imgfile*] Optional argument. *imgfile* specifies the name of the output image files. If this option is not specified, the default path name for *imgfile* is */tmp/w_cam.$USER*, where $USER is expanded to the login name of the user. If more than one image sequences are specified in the *inSpec*, the output image files will have the names *imgfile*[.Z], *imgfile*1[.Z], *imgfile*2[.Z], ...

EXAMPLES

```
% w_camera -dir $HITERT_HOME/data/images/scene1 \
-prefix s1.img -num_width 4 -suffix .Z \
-inSpec inSpec -outSpec outSpec
```

SEE ALSO

hitert(1), vseqdir(1), t_camera(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

NAME

t_camera – **Batch**-mode tracking camera module in HiTert System

SYNOPSIS

t_camera -world *world-spec* -noise *noise* -o *outSpec* [-help]

DESCRIPTION

t_camera , conceptually, takes world view image and tracking commands(where should camera look at ?) and outputs the acquired image to the image processor for segmentation and to the instrumentation module for displaying purpose. However, since in batch-mode imageries are presegmented, tracking camera *image* to the image processor is not really needed. However, the image processor need to know the tracking camera commands(where is camera looking at ?) in order to compare with the presegmented image to output the appropriate overlapping area of the images. Also since the the world view image is available to the instrumentation module, the instrumentation module may display appropriate subimage according to the tracking camera commands. Therefore no tracking camera image is really needed. For implementation efficiency, t_camera does not output images. It merely make its its state information available to other down-stream modules.

OPTIONS

-world *world-spec* The world-spec contains the information about the time and the filename corresponding world-view image, as well as the data records from the description file for the world-view image database and the data records from the description file for the segmented image database.

-noise *noise* Required argument. It specifies the name of the file containing three floating numbers representing white noise intensities for x, y and z directions respectively. The exact target position as obtained from *world-spec* are perturbed with Gaussian random values with their standard deviations determined by noise intensities, respectively. The perturbed values are used to simulate where the actual tracking camera is looking at. By changing the input noise intensities, one may mimic the tracking cameras with different tracking accuracies.

-o *outSpec* Required argument. It specifies the name of the output spec file.

[-help] *t_camera* will print out brief help message if this option is specified. Note this option is designed to give a quick help to the user via unix command line option. It does not work with *cantata(1)*.

SEE ALSO

hitert(1), w_camera(1), xhitert(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

NAME

ipc – Image Processor.

SYNOPSIS

ipc -imgInfo *imgInfoFile* -segInfo *segInfoFile* -inSpec *inSpec*

[-dir *directory*] [-prefix *prefix*] [-num_width *num-width*] [-suffix *suffix*] -meas *measurement* -outSpec *outSpec* [-img *image-file*] [-uc <1 or 0>] [-help]

DESCRIPTION

ipc computes the target centroid position according to segmentation results and range information that are contained in *imgInfoFile* and *segInfoFile*. It also returns an desired output file according to the user specified directory, prefix, sequence number, suffix. The sequence numbers are extracted from the *inSpec* file.

If *suffix* is ".Z", it is assumed that the corresponding input file is in the compressed format. The corresponding output image files will be uncompressed or remain compressed depending on the option specified for "-uc". If the output image file remain compressed, the output image files are tagged with ".Z" to indicate that they are compressed files.

No assumption has been made that the prefix, sequence and suffix are separated by the character period ".". If they are indeed separated by such a character, they should be explicitly specified either in the prefix or suffix part.

OPTIONS

-imgInfo *imgInfoFile* Required argument. This option specifies the description file for the world-view image database.

-segInfo *segInfoFile* Required argument. This option specifies the description file for the segmented image database.

-inSpec *inSpec* Required argument. This option specifies an input file to the program. This input spec file comes from the output spec file of the tracking camera.

[**-dir** *directory*] Optional argument. This option specifies the directory from which the input file is to be picked up. It is an optional argument. Default: the current directory.

[**-prefix** *prefix*] Optional argument. This option specifies the part in the input file name that precedes the sequence number. It is an optional argument. Default: null.

[**-num_width** *num-width*] Optional argument. The width that the numeric sequence takes may be specified through this optional argument. If the *num-width* is greater than the number of the *digits* the *sequence-number* (as specified in -num) has, the *sequence-number* will be *prepended* with zero's(0's) so that the width of the sequence string will be the same as *num-width*. Default: the sequence string is as long as the the number of digits that the sequence number has.

[**-suffix** *suffix*] Optional argument. The option specifies the part of the filename that *follows* the sequence string. If the last two characters of the suffix string are ".Z", then the input file will be assumed to in compressed format, and uncompressed data file will be produced using *uncompress(1)*. Default: null

-inSpec *inSpec* Required argument. *inSpec* specifies the name of a input *specification* file. The *inSpec* file has possibly several data records, with each data record consisting of a sequence number, a filename specifying the world-view image, the point(center of interest - COI) that the tracking camera is looking at, the screen coordinates of COI in the world-view image, a data record from the description file for the world-view image database and a data record from the description file for the segmented image database, all of which are separated by the white space characters(space, tab, newline).

-outSpec *outSpec* Required argument. *outSpec* specifies the name of a output *specification* file. The *outSpec* has the same number of data records as *inSpec* file. But each data record in *outSpec* is slightly different from that in *inSpec*. Each data record in *outSpec* consists of the sequence numbers, the world-view image filename, the filename for the segmented image, the point(center of interest - COI) that the tracking camera is looking at, the screen coordinates of COI in the world-view image, the data record from the description file for the world-view image database and data records from the description file for the segmented image database, all which are separated by the white space characters(space, tab, newline).

[-img *imgfile*] Optional argument. *imgfile* specifies the name of the output image files. If this option is not specified, the default path name for *imgfile* is */tmp/ipImg.$USER*, where $USER is expanded to the login name of the user. If more than one image sequences are specified in the *inSpec*, the output image files will have the names *imgfile*[.Z], *imgfile*1[.Z], *imgfile*2[.Z], ...

[-uc <1 or 0>] Optional argument. This boolean flag specifies whether or not to uncompress the input file(s) specified. If *suffix* is ".Z", it is assumed that the corresponding input file is in the compressed format. It will be uncompressed to produce the output image file if the "-uc 1" option is specified. (this is also the default). However, if "-uc 0" is specified and if the input files have the tag ".Z", the compressed image files will be copied to produce the output image files. In this case, the output image files are tagged with ".Z" to indicate that they are compressed files. This option has no effect if the input file has no tag ".Z".

EXAMPLES

```
% ipc -imgInfo ../dataControl/imginfo \
-segInfo ../dataControl/seginfo -inSpec inSpec \
-dir $HITERT_HOME/data/images/secne1_seg \
-prefix s1.seg -num_width 4 -suffix .Z \
-meas measurement -outSpec outSpec
```

SEE ALSO

hitert(1), vseqdir(1), w_camera(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

NAME

track – An alpha-beta-gamma tracker

SYNOPSIS

track -ip *ip_data* -ic *ic_file* -o *out_filename*

DESCRIPTION

track takes noise-contaminated measurement data as input and gives the optimal estimation of the state using Kalman filter theory.

OPTIONS

-ip *ip_data* Required argument. *ip_data* is the name of the file which contains the measurement data as obtained from the image segmentation. The measurement data file consists of the data records of the form:

t *x-measure* *y-measure* *z-measure*
⋮ ⋮ ⋮ ⋮

[-ic *ic_file*] Optional argument. *ic_file* is the name of the file which contains initial conditions for the tracker. It has the following data items, all of which are separated by the white space characters.

-ic *ic-file* Optional argument. The *ic-file* contains the *initial conditions* for the states of the tracker. It consists of the following data:

$$t_0 \quad \hat{x} \quad \dot{\hat{x}} \quad \ddot{\hat{x}} \quad \hat{y} \quad \dot{\hat{y}} \quad \ddot{\hat{y}} \quad \hat{z} \quad \dot{\hat{z}} \quad \ddot{\hat{z}}$$

where $\hat{x}$, $\dot{\hat{x}}$, $\ddot{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$ and $\ddot{\hat{z}}$ are the *initial conditions* at the starting time t_0. If this option is not specified, **track** will set $\hat{x}$, $\dot{\hat{x}}$, $\ddot{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$ $\hat{z}$, $\dot{\hat{z}}$ and $\ddot{\hat{z}}$ all to zeros as the default values for the initial conditions.

-o *out_filename* Required argument. *out_filename* is the name of the file which contains optimal state estimations. It consists of the data records of the form:

$$t \quad \hat{x} \quad \dot{\hat{x}} \quad \ddot{\hat{x}} \quad \hat{y} \quad \dot{\hat{y}} \quad \ddot{\hat{y}} \quad \hat{z} \quad \dot{\hat{z}} \quad \ddot{\hat{z}}$$
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮

where $\hat{x}$, $\dot{\hat{x}}$, $\ddot{\hat{x}}$, $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$ $\hat{z}$, $\dot{\hat{z}}$, $\ddot{\hat{z}}$ are the estimated states at the time t.

SEE ALSO

hitert(1), predict(1), xhitert(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

NAME

predict – The alpha-beta-gamma tracker

SYNOPSIS

predict -i *in-filename* [-tp *sec*] -o *out-filename*

DESCRIPTION

predict performs *sec* ahead prediction based on the state estimation from the alpha-beta-gamma tracker.

OPTIONS

-i *in-filename* Required argument. *in-filename* is the name of the file which contains the state estimation of the alpha-beta-gamma tracker, the "best" estimation about the target's position, velocity and acceleration. It consists of the data records of the form:

$$\begin{array}{cccccccccc} t & \hat{x} & \dot{\hat{x}} & \ddot{\hat{x}} & \hat{y} & \dot{\hat{y}} & \ddot{\hat{y}} & \hat{z} & \dot{\hat{z}} & \ddot{\hat{z}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

where $\hat{x}$, $\dot{\hat{x}}$ and $\ddot{\hat{x}}$ are the position, speed and acceleration of the target at time t_0. t has the unit of *seconds*; whereas $\hat{x}$, $\dot{\hat{x}}$ and $\ddot{\hat{x}}$ have units of *meters*, *meters/seconds* and *meters*/(*seconds*)^2, respectively. Similar remarks apply to $\hat{y}$, $\dot{\hat{y}}$, $\ddot{\hat{y}}$, $\hat{z}$, $\dot{\hat{z}}$ and $\ddot{\hat{z}}$, respectively. These data are separated by one or more mixed *white space characters*.

[**-tp** *sec*] Optional argument. *sec* is the expected time of flight of a projectile intercepting the target.

-o *out-filename* Required argument. *out-filename* is the name of the file which contains predicted target position. It consists of the data records of the form:

$$\begin{array}{ccccc} t & t_f & \hat{x}(t_f|t) & \hat{y}(t_f|t) & \hat{z}(t_f|t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

where t is the *current* time, t_f the *future* time($t_f = t + t_p$), $\hat{x}(t_f|t)$, $\hat{y}(t_f|t)$ and $\hat{z}(t_f|t)$ the *predicted* target position coordinates at t_f based on the *current* state estimates at t.

SEE ALSO

hitert(1), track(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

NAME

errAnaly – Performs error analysis for the tracking system.

SYNOPSIS

errAnaly -pred *predfile* -exact *exactfile*
[-showErr <*1 or 0*>] -outSpec *outSpec* [-outErr *outErr*] [-help]

DESCRIPTION

errAnaly computes error metrics for the tracking system, using the prediction data and exact data.

OPTIONS

-pred *predfile* Required argument. It specifies the file that contains the prediction data from the predictor.

-exact *exactfile* Required argument. It specifies the file that contains the exact data for comparison.

-showDart *showDart* Optional argument. If *showDart* is 0, *errAnaly* will compute the inertial component errors and write these errors to a data file(see option "-outErr *outErr*"). If *showDart* is 1, *errAnaly* will compute the dartborad errors and write these errors to a data file(see option "-outErr *outErr*"). These error data are intended main for graphical displaying purpose. The statistics are are always computed based on the inertial component errors. Default: 1

-tbf *tbf* Optional argument. *tbf* is track-before-fire time in seconds. Default: 0

-tburst *tburst* Optional argument. *tburst* is the burst interval in seconds. Default: from starting point to the terminating point.
Example: -showDart 0

-outSpec *outSpec* Required argument. It specifies the name of the file that contains the output c file, which contains statistics of the *errors* and the filename of the actual error data file.

-outErr *outErr* Optional argument. *outErr* specifies the name of the file which contains the actual error data. Default: /tmp/errAnaly.login

SEE ALSO

hitert(1), predict(1), xhitert(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

NAME

xhitert – Instrumentation/display module for the HiTert System

SYNOPSIS

xhitert –world *world* –camera *camera_state* –ip *ip_outfile* –pred *predictor_data* –exact *exact_trajectory* [–display *display*]

DESCTIPTION

xhitert is a program which runs under X window system. It graphically displays the run-time status of the HiTert system. Graphically, xhitert has the following components in its top level window:

an image of a *world view*.

an image as seen by the *tracking camera*.

an image segmented by the *image processor*.

a *dart-board*, which displays the prediction errors.

exact *trajectory* and the predicted *trajectory*.

tabulated numerical results of the *error analysis*.

Functionally, *xhitert* take the inputs from the "camera", which provides the world view image and the state of the tracking camera so that the *xhitert* can find, from the world view image, the subimage corresponding to the tracking camera. It also takes as its input the segmented from the image processor and displays all the input images. Besides, *xhitert* takes as its inputs the tracker/predictor results, performing error analysis, mimicking dart-board and display exact and predicted trajectories.

This version is designed to work in the *batch-mode* of the HiTert system.

OPTIONS

–world *world* Required argument. It specifies the ASCII file which contains the *image-sequence-number* followed by the world-view-image *filename*. *image-sequence-number* is used to carry the time information that the image corresponds to. *time* and *filename* are separated by white space character(s).

–camera *camera* Required argument. It specifies the ASCII file which contains the tracking camera state.

–ip *ip* Required argument. It specifies the ASCII file which contains the measurement as segmented by the image processor.

–pred *predicted_results* Required argument. It specifies the file which contain the prediction results.

–exact *exact* Required argument. It specifies the file which contains the exact trajectory data.

–display *host:diplay:screen* Optional argument, which specifies where to display *xhitert*'s graphical output.

–demo <*0 or 1*> Optional argument. This boolean flag specifies whether or not to display the tracking results in a *pseudo real-time mode*. If *demo* is *false*, i.e., 0, all the tracking results will be blasted to the screen almost at the same time; if *demo* is *true*, however, *xhitert* will be in the *pseudo real-time mode*, displaying the tracking results one at a time.

-help Optional argument. When specified, *xhitert* will print out biref help message.

SEE ALSO

X(1)

AUTHORS

Jun Lu, Dominick Andrisani II, Purdue University School of Aeronautics and Astronautics

Bibliography

[1] M. F. Tenorio and D. Andrisani II, *A Hierachical Approach to Target Recognition and Tracking*, Research Proposal, Schools of Electrical and Aerospace Engineering, Purdue University, West Lafayette, IN, 1989.

[2] D. Andrisani II, M. F. Tenorio, J. Lu and F. P. Kuhl "A Hierachical Approach to Passive Robust Target Tracking," *Proceedings of the Ninth Meeting of the Coordinating Group on Modern Control*, Picatinny Arsenal, N.J., Oct. 24–25 1989.

[3] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd ed., vol. 1 and 2, Academic Press, New York, 1982.

[4] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, New Jersey, 2nd ed., 1988.

[5] B. W. Kernighan and R. Pike, *The UNIX Programming Environment*, Prentice-Hall, New Jersey, 1984.

[6] M. J. Rochkind, *Advanced UNIX Programming*, Prentice-Hall, New Jersey, 1985.

[7] W. R. Stevens, *UNIX Network Programming*, Prentice-Hall, New Jersey, 1990.

[8] Khoros Group, *Khoros Manuals*, vols. 1 and 2, Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM, 1991.

[9] J. D. Foley, A. Van Dam, *Computer Graphics: Princiles and Practice*, Addison-Wesley, Reading, Mass., 1990.

[10] W. M. Newman, R. F. Sproull, *Principle of Interactive Computer Graphics*, McGraw-Hill, New York, 1979.

[11] Stardent Computer, Inc., *Doré Reference Manual*, 1989.

[12] R. A. Singer, "Estimating Optimal Tracking Filter Performance for Manned Maneuvering Targets," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-6, pp. 473–383, July 1970.

[13] R. Berg, "Estimation and Prediction for Maneuvering Target Trajectories," *IEEE Trans. on Automat. Contr*, vol. AC-28, pp. 294–304, Mar. 1983.

[14] J. D. Kendrick, P. S. Maybeck and J. G. Reid, "Estimation of Aircraft Target Motion Using Orientation Measurements" *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-17, pp. 254-259, Sept. 1986.

[15] D. Andrisani II, F. P. Kuhl and D. Gleason, "Estimation of Aircraft Target Motion Using Orientation Measurements," *IEEE Trans. on Aerosp. Electron. Syst.*, vol. 17, pp. 533–538, Mar. 1986.

[16] J. D. Schierman and D. Andrisani II, "Tracking Maneuvering Helicopters Using Attitude and Rotor Angle Measurements," *Journal of Guidance, Control and Dynamics*, vol. 13, pp. 929–935, Nov.-Dec. 1990.

[17] S. A. R. Hepner and H. P. Geering, "Observability Analysis for Target Maneuver Estimation via Bearing-Only and Bearing-Rate-Only Measurements," *Journal of Guidance, Control and Dynamics*, vol. 13, pp. 977–983, Nov.–Dec. 1990.

[18] R. J. Fitzgerald, "Simple Tracking Filters: Closed-form Solutions," *IEEE Trans. on Aerosp. Electron. Syst.*, vol. AES-17, pp.781–785, Nov. 1981.

[19] A. Gelb (ed.), *Applied Optimal Estimation*, M.I.T. Press, Cambridge, MA, 1974.

[20] T. Kailath, *Linear Systems*, Prentice-Hall, New Jersey, 1980.

[21] G. F. Franklin, J. D. Powell and M. L. Workman, *Digital Control of Dynamic Systems*, Addison-Wesley, Reading, Mass., 1990.