Requirements and Execution Plan

David E Smith • Mike Kasper • Ryan Raub

2008/02/13 Rev 1.1

Table of Contents

Introduction

Synthetic Aperture Radar (SAR) is a modern radar imaging technique. The key constraint of this technology is that it can only be used from moving platforms while aimed at stationary targets. Fortunately, these instruments are commonly employed in this manner. However, issues arise during their development, as a moving platform is also required during testing. A common solution is to simulate movement by sending the SAR instrument inertial navigation data. Current simulations are hard-coded to devices that can send this data to the SAR instrument reliably at a constant desired rate. Unfortunately, the required rate and format of input vary among SAR instruments, and these devices only work for a single variation. This project aims to develop an application that provides a more flexible solution, where simulated aircraft movement can be sent to a wider range of SAR instruments.

Problem Statement

Testing of Synthetic Aperture Radar (SAR) is currently done manually using Inertial Navigation Systems (INS), which can currently play back pre-recorded flight data or simulate actual flight data to the SAR for testing. There are several different protocols for data transfer for each SAR interface; each requiring a specific INS that is not interchangeable or configurable. This makes the testing environment very rigid and time-consuming to set up and run. Recorded Flight data can only be played back over the same protocol upon which it was collected, which results in an even narrower selection of test cases available. In summary, the current testing environment is rigid and could benefit greatly from a more flexible solution.

Solution Statement

An INS simulator will feed the SAR simulated data calculated from a flight path defined by the user; which allows for an improvement to there testing environment for their SAR product. The INS simulator will have two main components; the User Interface and the Navigation Simulation System. The User Interface will allow the user to interactively manipulate a flight plan that the Navigation Simulator will run. The output from the Navigation Simulation can be in several forms, which are defined by a modular output. The final module will control the synchronization of data processing in the core module, which allows for a dynamic timing interface. This software application will take the place of several hardware INS systems currently in use, and provide a more convenient and flexible interface for testing.

Functional Specifications

GUI must reside on remote machine
The application will be designed to operate either over a network or on the local system. This provides the user with flexibility, and the time-critical system with some relief from potential load.

Requirements:
- Communication over some network is needed
- A process to select a remote core module to communicate with

Saving and Loading waypoints to/from a file
The application will take in a lot of its input data via waypoints. These waypoints must be either hand-enterable via the GUI or readable from a file on disk. This provides the user with the ability to save waypoints for future use, or generate waypoints independently of the application.

Requirements:
- Must be conducted from GUI
- Standard format must be established
- Provide some means to create and output new files
- Raw data must passed to and interpreted by core

Visual display of flight status
Visual feedback is critical for the application's usability. This includes a two-dimensional graphical model of the flight as it progresses. Providing this will give the user vital feedback about the current status of the simulation.

Requirements:
- Core must also be able to communicate with the GUI client

Modular I/O formats
Since it will be difficult to predict all possible interfaces this application could potentially be used with in advance, modularity of output is critical. The application must be designed such that a variety of formats can be selected, and the task of creating a new output format for the application is as trivial as possible. This is a critical component of the specification, as the product is nearly useless without it.

Requirements:
- I/O functionality must be extracted from the core

Flight path changes at runtime
In order for the simulation to be as efficient as possible, the user must be able to make adjustments to the flight path while the simulation is in progress. This will be performed via the GUI, and allow the user to test a wider variety of situations without resetting the simulation.

Requirements:
- The core must receive the data to process from the client at runtime
- Client-side viewing and editing of flight paths
- Restrictions on what can and can not be changed

Flight Control
The GUI must be able to communicate user adjustments to the core module in order for changes to the simulation to take effect. A communication layer must be established to allow for rapid adjustments to the plane in flight.

Requirements:
- The GUI to be able to communicate and control the core module

Constraints

Precise Output Metering
The application must be able to output data at regular intervals based on a provided signal to ensure that it can perform its task effectively. If the application is unable to reliably meet this need, it will be potentially useless.

Execution Plan

Initial Communication (early Jan '08)
The first objective of was to establish initial contact with our clients. This was done throughout the month of January, by email and teleconference. During this time, we introduced ourselves and gave an overview of our skills and experience as a team and as individuals. We also got to know the clients we would be working with throughout the rest of this project's development.

Determine Requirements (late Jan – Feb '08)
Our progress currently resides at the end of this stage of the development plan. During this time, we obtained the specifications and requirements desired by our clients. To achieve this we have conducted several teleconferences and exchanges emails and other documents. The clients made a recent request to have the application have real-time functionality. This, expectedly, extended our discussion and delayed our progress. Our current focus has been on general functionalities of the program, and not much has been mentioned on the performance specifications of the GUI. This will be better defined in our upcoming trip to their facilities on Friday, February 8th 2008.

Establish Development Environment (Feb '08)
This milestone of development simply requires decisions to be made on the development framework, programming language(s), and code repositories. Because our final program must reside on both remote computers as well as real-time boards, we must also determine how testing can be conducted to properly assess the correctness of our

implementation. This too will be worked on during our upcoming trip to meet with the clients, as we will be shown their current technologies and devices.

Architecture Design (Feb '08)
After establishing the requirements, we will be able to begin designing the project. In this first stage of design, we will focus on the architecture and class relations. UML diagrams will be exchange during this time, assuring that we, as well as the clients, have a mutual understanding of how we will be implementing the solution. It is important that we establish this information first, as it may continue to add constraints and requirements to our GUI.

Interface Design (Feb '08)
This second part of design will focus on the user interface. We will discuss the desired look of the application, as well as the flow of its tasks. The exchanging of GUI prototype sketches and walkthroughs will be conducted to help steer our early design efforts in the right direction.

Implementation: Stage One (late Feb – early March '08)
This initial stage of implementation will focus on developing critical areas of functionality. These may include the communication between devices, use of new languages and technologies to employ real-time behavior, and output of data at various speeds. This time will be devoted to understanding our risks as well as gaining a better understanding for future development. Final code may not be produced at during this stage; instead, this will most likely produce proof-of-concepts.

Implementation: Stage Two (March '08)
The majority of actual implementation will be done during this stage. Our focus here is to provide all the core functionality and finish a complete prototype. We may begin by working on modules independently, after establishing an initial basic communication, and only combine them towards the end of this stage. An expert review might be conducted after this stage's completion, to better guide us in the next stage of implementation.

Implementation: Stage Three (late March – early April '08)
A week of final implementation will be devoted to providing additional functionality. This will probably focus on the GUI client, as the other modules will most likely require no more functionality than those defined in the specification documents. Ideas for these additions will result from the reviews and comments from our result of the previous stage of implementation. These last-minute features will be what we have decided will make the use and future development of program more efficient.

User Testing (early April '08)
Our exact plan for user testing will be determined around April 1st, but we plan on devoting at least two weeks to conduct actual testing. We anticipate the testing regime, in addition to the few expert reviews conducted in earlier stages of development, to include the following:

Expert Reviews: Additional tests of our application and recent changes made in the final stages of implementation. This will generally not be conducted in person with the clients, given our geographical constraints. We believe this will be acceptable because we only desire that major bugs/problems be reviewed here.

Moderated Testing: We hope to make another trip to see our clients, and sit down with them as they walk through each of the program's major processes. This will aid us in the fine-tuning of our application and hopefully result in a product that completely satisfies our clients.

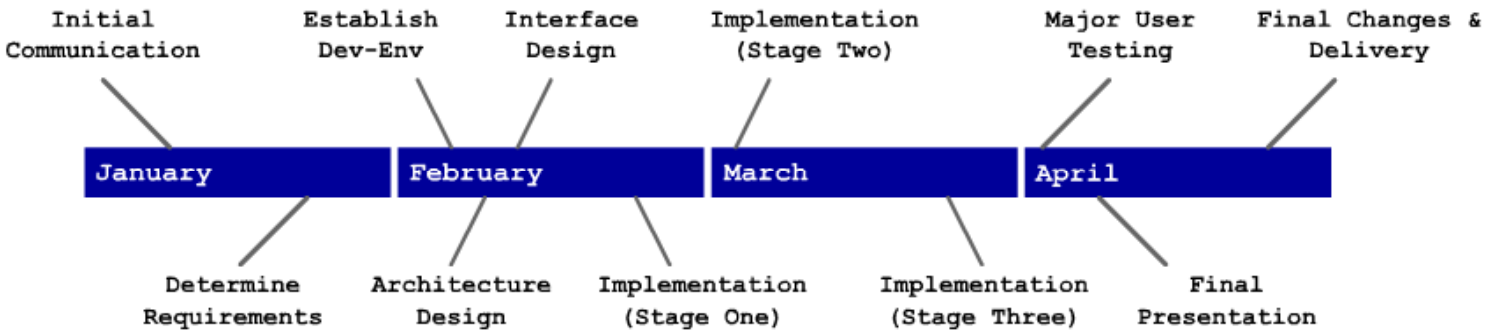
Performance Testing: We may also conduct some performance tests, but this depends on whether we can establish some concrete performance requirements with our clients.

Final Presentation (April 16th – 18th '08)
Having completed the majority of our project, we will devote the few days before our final presentation, on the 18th of April, to preparing slides and creating the poster.

Possible Changes
There is current discussion as to the relationship between the synchronization module and other modules, and whether its modular design adds avoidable complications system.

Lastly, it is possible that we need more time to complete the user testing than the two weeks allotted. If this occurs, we will have to complete more testing and final polishing of our application after the final presentation and before the end of the semester. This will give us an additional 2-3 weeks to complete user testing.

Timeline



Appendix A: Input/Output Specification

- ASCII file, tab delimited
- all fields are decimal values
- max resolution of 256Hz for time stamp
- resolution of 10e-3 for altitude
- resolution of 10e-9 for all other fields
- field order:

TimeStamp	(s)
Lat	(degrees)
Lon	(degrees)
Altitude	(ft)
Velocity East	(ft/s)
Velocity North	(ft/s)
Velocity Up	(ft/s)
RollAngle	(degrees)
PitchAngle	(degrees)
YawAngle	(degrees)

- minimum input of two waypoints and a velocity
- birds-eye view of plane in flight on path
- load waypoints from file
- flight speed in range of 40 mi/s – 200 mi/s (default: 100 mi/s)
- no curves needed in flight path, rotation is instant

Appendix B: Input Example

Waypoint	Alt	Lat	Lon	Speed	Roll	Pitch	Yaw
0	16000	38	129	200	0	0	0
1	16000	38	126	200	0	0	0
2	16000	36	126	200	0	0	0
3	16000	36	129	200	0	0	0
4	16000	38	129	200	0	0	0
5	16000	36	126	200	0	0	0
6	16000	36	129	200	0	0	0
7	16000	38	126	200	0	0	0

Appendix C: Output Example

Time	Lat	Lon	Alt	Vel E	Vel N	Vel Up	Roll	Pitch	Yaw
1.0000	38.0000	129.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2.0000	38.0000	126.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3.0000	36.0000	126.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000
4.0000	36.0000	129.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5.0000	38.0000	129.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000
6.0000	36.0000	126.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000
7.0000	36.0000	129.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000
8.0000	38.0000	126.0000	16000	200.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Appendix D: Modular Design Example Diagram

This is a diagram illustrating an example implementation that would meet the required specifications for application modularity. Note the independence between the graphical user interface and the core system, as well as the separate output and synchronization modules that can potentially be replaced as needed to satisfy new operating conditions.

This diagram is exploratory and is intended as an example only.