REAL-TIME UAV VISUALIZATION USING A FLIGHT SIMULATOR

E.R. Price and G.K. Egan

Department of Electrical & Computer Systems Engineering
Monash University 3800
Melbourne, Australia

Abstract

This paper describes a real-time visualization tool for a UAV based on Microsoft Flight Simulator. The tool display consists of a 3D simulation of the aircraft's position and orientation as well as an instrument panel which displays flight parameters and alarms deemed most relevant by the pilot in the UAV's various flight modes. The tool is intended to bring some comfort to the pilot when the aircraft is beyond visual range.

Biography

E. R. Price is a graduate of the Department of Electrical & Computer Systems Engineering and completed this research as part of his undergraduate engineering programme.

Professor G.K. Egan is Professor of Electrical & Computer Systems Engineering and Director of the Centre for Telecommunications and Information Engineering (greg.egan@eng.monash.edu.au).

Also published AIAC12, Melbourne, Australia, 16-22 March 2007

Introduction

AN unmanned aerial vehicle (UAV) is a self-descriptive term used to describe the latest generation of pilotless aircraft. The modern UAV originated in the early 1970s and promises to transform both military and civilian aerospace operations. The attractiveness of UAVs is their ability to do the dirty, dangerous and dull jobs autonomously.

The Aerobotics© (Aerial Robotics) Research Group at Monash University is interested in all aspects of the design, construction and application of electrically powered UAVs. The research group's current aircraft transmit a significant amount of information while in flight. The aim of this project was to display this data within a flight simulator package to obtain a pilot's eye view of the aircraft's behavior in real-time to assist when flying the aircraft visually at long-range and to give some confidence in the autopilot's operation when the aircraft is beyond visual range. Four main objectives were identified in order to achieve this aim. The overall system diagram is shown in Figure 1.

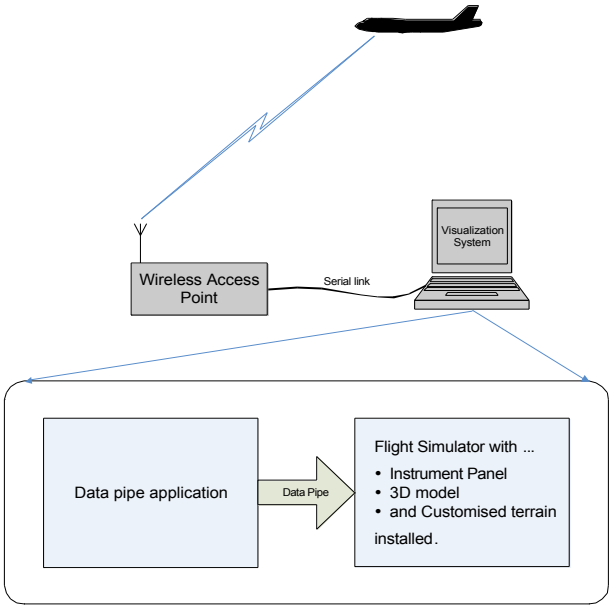

Figure 1. Complete system diagram, showing how the four components of the visualization system interact with the existing components of the system.

An application to pipe flight data to a flight simulator

A Windows application was written to receive real-time data from the UAV, convert this data into the format required by Flight Simulator, and then pipe it to an instance of Flight Simulator. The application also implements some smoothing and interpolation to the data as the data is only sent from the UAV once every second whereas the flight simulator requires about 20 frames per second for smooth simulation.

Constructing a 3D model of the UAV

A model of the UAV was made using a graphical modeling package. This model was then converted to a new flight simulator aircraft so that when simulating the UAV it would actually appear as the UAV rather than one of the standard aircraft supplied with the flight simulator.

Designing a graphical instrument panel

An instrument panel was designed that graphically displays all the relevant flight data, such as:

- Position and orientation (latitude, longitude, altitude, heading, pitch, roll, etc).
- Battery voltages and warning alarms
- Temperature levels and warning alarms
- Navigational information

The content and layout of the panel was designed in consultation with those who fly the UAV with the aim of designing a panel, which displays the most important information in an easy to read format without overloading the pilot with unnecessary information.

Customizing the terrain

The standard terrain used within Microsoft Flight Simulator contains major roads and landmarks but is mostly just computer generated from knowledge of the population density in a particular area. To provide more realistic terrain, satellite and/or aerial photos were collected for the areas in which the UAVs are flown and were mapped to the terrain surface.

Choice of Flight Simulator Software

Two possible choices of flight simulator software were considered, FlightGear and Microsoft Flight Simulator 2004. The FlightGear flight simulator is an open-source, multi-platform, cooperative development project [1]. Microsoft Flight Simulator 2004 is a proprietary flight simulator developed by Microsoft Corporation [2]. The features of both flight simulators were researched and it was concluded that both simulators provided all the necessary features to meet the objectives of this project. However it was also concluded that the documentation provided with Microsoft Flight Simulator 2004 was more comprehensive and understandable than that provided with FlightGear. Given the limited timeframe that was available to complete this project Microsoft Flight Simulator 2004 was chosen.

FlightGear does however provide greater flexibility than Microsoft Flight simulator since it is an open-source project, and likely to be a better choice for a more in-depth project.

Data Pipe Application

In order to drive Microsoft Flight Simulator with external data from the UAV a small Windows application was written. This application reads in the flight data from a selected source (either from a log file or from the serial port), converts the data into the format required by Microsoft Flight Simulator, and sends this data to the simulator via a pipe.

Input data formats

The application was required to input flight data stored in two different formats. The first format is a text log file produced from a MicroPilot MP2028 autopilot [3]. The second format is a binary data stream, which is sent via the serial link in real-time. As well as being able to receive this data stream from the serial port in real-time, the application was given the functionality to read this data stored in a binary file. This allows a flight to be recorded and played back at a later time.

The data, which is sent by the ground station via a serial link, comes in the form of three C data structures. The set of data structures are sent once every second.

Reading data from the serial port

The data from the serial port is read in-between sending frames to Flight Simulator. It is essential that the program execution does not wait at the serial port because Flight Simulator requires a new frame every 50ms or so (depending on the frame rate), and if it doesn't receive the next frame in time the simulation will just hang there until it does.

The serial port operates at 4800 baud and each packet is of about 200 bytes long, which gives a transmission time of about 330ms. Thus, a complete packet cannot be received in-between sending a frame to Flight Simulator. Instead, only the data already in the serial port buffer is read before sending the next frame to the simulator. Then, when the complete packet has been received the data is copied into the data structures and the relevant data is extracted and converted to the format required by Flight Simulator.
An alternative approach would have been to have a multithreaded program, one thread to read data from the serial port and one thread to pipe data to Flight Simulator. This however would have added unnecessary complexity to the design.

The overall scheme for reading real-time data is shown in Figure 2.

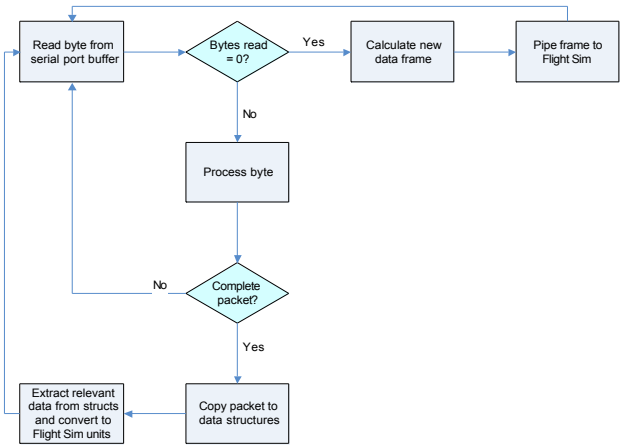

Figure 2. Algorithm for reading data from the serial port.

The Flight simulator recorder file format

In Microsoft Flight Simulator, the "video" recorder captures data about the simulation and stores it in a .fsr file. When piping external data to Microsoft Flight Simulator the same file format must be used. A .fsr file is organized around frames of data representing a snapshot of the simulation and environment at a moment of time. The data recorded in each data-frame is indexed into a dictionary of properties and objects specified at the top of the data file.

Microsoft Flight Simulator internal parameters

The names of all the internal Flight Simulator parameters are provided with the *Panels SDK*, available from the Microsoft Flight Simulator website [2]. These names must be used when defining the parameters in the properties section of the .fsr file. There is no way to add new parameters.

Some of the UAV flight variables such as battery voltages and temperatures had no equivalent parameters to use in Microsoft Flight Simulator. To pass such variables to Flight Simulator, and hence be able to display them on the panel, other unimportant Flight Simulator parameters had to be found to represent these variables. Although there are hundreds of internal Flight Simulator parameters, it was found that only a very limited number of them could be modified externally, mostly only those that directly relate to the simulation. For example, airspeed and aircraft position could be modified whilst next waypoint position could not. This presented a problem, as there were a limited number of parameters in which to store all the UAV flight data. To overcome this problem the many Boolean variables were stored bitwise within the one variable.

Creating a pipe to Microsoft Flight Simulator

Included with the *Netpipes SDK*, available from the Microsoft Flight Simulator website [2], is sample code for collecting data from one instance of Flight Simulator and playing it back in real time in another instance of Flight Simulator, to drive views of the same flight on two computers. Thus, the classes from this sample were used to create a pipe to Microsoft Flight Simulator and play the simulation.

Data interpolation and smoothing

When the data is read from a file a moving average technique is used, with two data points behind and two data points ahead being stored to calculate the values for each data frame. Between each data point a number of interpolated points are calculated equal to the number of frames sent to Flight Simulator per second. Then, from these interpolated points, a moving average curve is calculated using a specified number of points either side.
When the data is read from the serial port in real-time a moving average technique is again used but only those points preceding the current point are used, otherwise an unnecessary delay in the simulation would be introduced.

Graphical user interface

A graphical user interface (GUI) was designed for the data pipe application by using the Microsoft Foundation Class Library (MFC). It allows selection of the input source (either a log file or the serial port), selection of an input file (if the source is a log file), and the option to save the simulation as a Flight Recorder File or save the data stream if reading from the serial port. It also provides dialog boxes to set the initial altitude and serial port parameters (Figure 3).

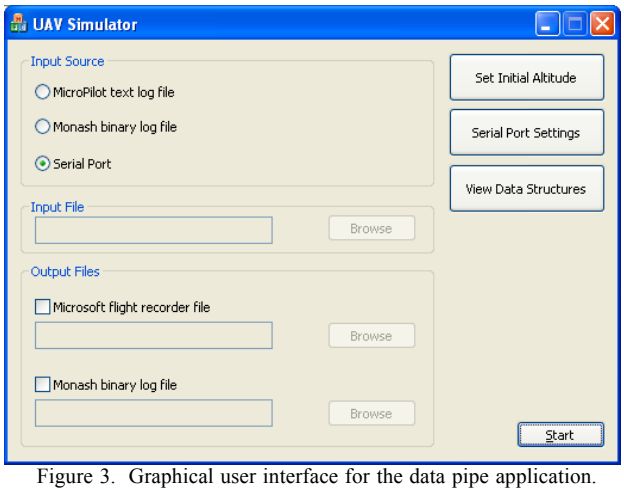

Figure 3. Graphical user interface for the data pipe application.

The initial altitude is required because the altitude on the UAV is initialized to zero before takeoff, whilst the flight

simulator requires an altitude above sea-level. Hence the initial height of the UAV above sea level must be known and set with this dialog box; otherwise the UAV is likely to be placed under the terrain's surface.

When the View Data Structures button is pressed a property sheet is created which displays the current values of all the fields contained in the data structures sent from the UAV. This provides a means of viewing the less important data fields, which are not displayed on the graphical instrument panel.

3D Model of The UAV

To add to the realism of the simulation a graphical model of the UAV was created and converted to a Microsoft Flight Simulator aircraft (Figure 4). The model was created using the 3D modeling package *gmax* from Discreet [4]. This software package was chosen as Microsoft Flight Simulator provides an SDK with an application to convert a model created within *gmax* to a Flight Simulator aircraft. The SDK also provides details on how to design aircraft within *gmax*.

Textures

Taking photos of every surface of the UAV and mapping these to the model as textures improved the realism of the model. To be used within Flight Simulator the textures were converted into *mipmaps*. A mipmap is a sequence of textures, each of which is a progressively lower resolution representation of the same image. The height and width of each image, or level, in the mipmap is a power of two smaller than the previous level. A high-resolution mipmap image is used for the UAV when it appears close. Lower resolution images are used, as the UAV appears further away. Mipmapping decreases the time required to render a scene and also improves the scene's realism [5]. However they do require more memory.

Animations

Two animations were also added to the model; a spinning propeller, and moving wing flaps. For Microsoft Flight Simulator to recognize these animated parts it uses a naming convention which is described in the *gmax Aircraft Creation SDK*, available from the Flight Simulator website [2]. The speed of the propeller is determined from the throttle parameter. Likewise the position of the wing flaps is controlled by an internal parameter.

Model scale

It was found that with the model at the correct scale (approx 1.5 m wingspan) it would not display in the aircraft preview window, and would also partly disappear in the simulation window at certain view angles. The graphics generator within Flight Simulator is just too crude for models of that small a scale to display correctly.

To overcome this problem the scale of the model was increased to have a wingspan of about 5 meters, which resulted in the model displaying correctly. It was not important to have the model at the correct scale, as it is simply a visual of the aircraft's position and orientation during flight.

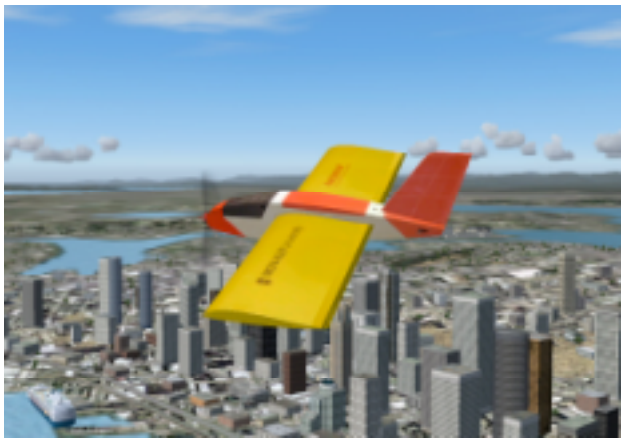

Figure 4. 3D graphical model of the UAV whilst in flight.

Instrument Panel

A large amount of information is captured by the UAV during flight. Some of this data is very useful to the person piloting the aircraft, whilst other information is not immediately relevant. The objective of this part of the project was to consult with pilots to determine which data would be useful and how this data would best be displayed.

Programming the panel gauges

Microsoft Flight Simulator has traditionally used gauges programmed in C. However, in Flight Simulator 2002 a new kind of gauge was introduced, the XML gauge. Currently, in Flight Simulator 2004, both C gauges and XML gauges are supported, although the majority of the standard panels still use C gauges. The shift is however towards XML gauges and thus the panel for the UAV was built using XML gauges to increase the chances that it will be compatible with future versions of Microsoft Flight Simulator.

Figure 5. Graphical instrument panel for the UAV.

An XML gauge in Flight Simulator consists of three parts:

1) The first part provides generic information about the gauge: its name, its background image, and its size.
2) The second part of the XML gauge is a list of elements of the gauge (marked by <Element> tags). Each element describes a part of the gauge, be it a needle, a switch, or a display with numbers. Each element is a bitmap image or a text string. The gauge elements are transformed by further child elements. For example, <Visible> specifies when the element is visible and

<Rotate> rotates the element.

3) The third part of the XML gauge describes the gauge's mouse rectangles, which can be used to display tool tips or to obtain user input from a mouse click.

Designing the graphics for the panel

The graphics of the panel and its gauges (Figure 5.) were designed using the drawing tools in Adobe Photoshop to create vector shapes (mathematically defined lines and curves), which could be easily manipulated and resized as desired. Once designed the images were converted to 8-bit bitmap images as required by Microsoft Flight Simulator. When transparency was required the background color was set to true black (0,0,0 RGB value). True black is made to appear transparent in the simulation.

Custom Terrain

To improve the realism of the terrain in the simulation, satellite images and aerial photos of some of the areas in which the UAV is flown were collected and mapped to the terrain. This was done using the re-sampler tool provided within the *Terrain SDK*, available from the Microsoft Flight Simulator Website [2].

Image requirements

The raw image required by the re-sampler must be either a 24-bit per pixel Windows .bmp or a 32-bit per pixel Targa .tga file. The latitude and longitude of the northwest corner of the image must be known as well as the spacing (in decimal degrees) between the image pixels. The spacing between pixels however can be calculated from the latitude/longitude values of the image corners and the image size. The extent of each terrain texture pixel is 4.75 meters in Flight Simulator. The re-sampler process will filter the raw image to the right size. If the raw image is less than 4.75 meters per pixel, some detail is lost. This is unfortunate as much of the detail from a high quality aerial photo is lost during the re-sampling process; major landmarks such as roads and buildings are still clearly visible at this resolution however.

Digital elevation models

As well as providing a tool to map images to the terrain, Microsoft Flight Simulator also provides a tool for using Digital Elevation Models (DEMs) to create a more accurate terrain mesh. A DEM provides a digital representation of a portion of the earth's elevation points over a two-dimensional surface. Currently, in Australia, accurate 3-second models are only available with the payment of a licensing fee [6]. For this project it was considered unnecessary to obtain these models, the standard elevation model of the simulator was considered to be adequate.

Conclusion

A visualization tool for a UAV has successfully been designed and implemented using Microsoft Flight Simulator. The Flight Simulator instance flew according to the data that it received and gave the correct position and orientation.

The system includes an application to receive data from the UAV and pipe it to the simulator, a graphical instrument panel which was designed using Flight Simulator's XML gauge system to display important information about the state of the UAV, a 3D model of the UAV, and a set of customized terrain textures obtained from satellite and aerial images.

Further testing is being conducted in real-time whilst flying the UAV.

This project has demonstrated the feasibility of using an existing flight simulator platform to provide a visualization of a UAV's position and orientation, as well as displaying crucial flight information, to the pilots on the ground.

Acknowledgment

We wish to thank the members of the Aerobotics Research Group at Monash University [7] and, in particular, the Chief Test Pilot Mr. Raymond Cooper for conducting the test flights and his general advice on human factors.

References

1. FlightGear, [online], Available: http://flightgear.org, September 2005, (Accessed September 2005).
2. Microsoft Corporation, "Microsoft Flight Simulator SDK", [online], Available: http://www.microsoft.com/games/flightsimulator/fs2004_downloads_sdk.asp, 2005, (Accessed April 2005).
3. MicroPilot, [online], Available: http://www.micropilot.com, 2005, (Accessed April 2005).
4. Autodesk, [online], Available: http://www.discreet.com, 2005, (Accessed May 2005).
5. Microsoft Corporation, "Texture Filtering with Mipmaps", [online], Available: http://msdn.microsoft.com, 2004, (Accessed June 2005).
6. Australian Government Geoscience Australia, [online], Available: http://www.ga.gov.au, July 2005, (Accessed September 2005).
7. Monash Aerobotics, [online], Available: http://www.ctie.monash.edu.au/hargrave/aerobotics.html, 2002, (Accessed February 2006).