Android for Pilots: Integration with the FlightGear Flight Simulator

Alex Perry
FlightGear

Look here /\ on the yoke!

Talk #16294
room Berlin I
11 June 2010
at 16:00.
Outline

• Problem - How do we design and test applications for pilots?
• Solution - Use a virtual aircraft environment

• Android, operating system for small devices
• QEMU, general purpose processor emulator
• FlightGear, Flight Simulator
• SimGear, reality simulation framework
• Open Scene Graph, visual representation engine

• Challenges, in addition to working with open source

• Opportunities for Further Work
• Conclusions
Why Android in Airplanes

Flying an airplane is a high workload (in addition to being fun), so it is worth moving tasks from the busy times to more relaxed phases of flight, or ideally even before the plane takes off.

There's lots of PDA utilities (web and local) for pre-flight tasks, but fewer for use during flight ...when automation would help.

Hardware choice really matters. Full sunlight to total darkness, vibration, damp, altitudes where humans need oxygen, etc etc.

it is hard to write good apps for use in-flight for exactly the reasons why they're so valuable. The environment and flight related workload is so different from sitting comfortably on the ground that it is easy to make poor design decisions...
People are able to do two tasks concurrently only in very limited circumstances

- asrs.arc.nasa.gov/publications/directline/dl10_distract.htm
- Even if they are skillful in performing each task separately
- Conscious control is required in four situations:
  1. when the task is novel,
     - the task is perceived to be critical, difficult, or dangerous,
     - when an automatic process must be overridden to prevent habit capture [i.e. expectation to deviate from routine], or
     - to choose among competing activities.
- Some [app data] entries involving one or two keystrokes can be performed quickly and may be interleaved with other cockpit tasks.
USA accident statistics for 2005

General Aviation: 1,670 total in 2005

Will Android help, or make it worse?


<table>
<thead>
<tr>
<th>Human Performance and Explanatory Causes/Factors 2005</th>
<th>All Accidents</th>
<th>Fatal Accidents</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Accidents</td>
<td>1,372</td>
<td>275</td>
</tr>
<tr>
<td>Human Performance Issues</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aircraft handling/control</td>
<td>990</td>
<td>229</td>
</tr>
<tr>
<td>Planning/decision</td>
<td>489</td>
<td>106</td>
</tr>
<tr>
<td>Use of aircraft equipment</td>
<td>148</td>
<td>22</td>
</tr>
<tr>
<td>Maintenance</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>Communications/information/ATC</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Meteorological service</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Airport</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Dispatch</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Underlying Explanatory Factors</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Qualification</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>Physiological condition</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Psychological condition</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Aircraft/equipment inadequate</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Institutional factors</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Procedure inadequate</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Material inadequate</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Facility inadequate</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Broad Causes/Factors by Accident Aircraft Type, 2005

- Aircraft: 25% Single-Engine Piston Airplane, 30% Multi-Engine Piston Airplane, 24% Turbine Airplane, 23% Helicopter
- Environment: 40% Single-Engine Piston Airplane, 28% Multi-Engine Piston Airplane, 42% Turbine Airplane, 37% Helicopter
- Personnel: 90%, 89%, 94%, 92%
Usability, when being interrupted often

- Attempting to perform longer programming tasks, such as adding waypoints or inserting approaches during busy segments of flight, can be problematic.
- It is not possible ... to reliably monitor ... the aircraft status during longer programming tasks, and
- it is difficult to suspend the programming in midstream without losing one's place.

This completely changes how you evaluate an application
- Can you make progress, one or two presses at a time?
- If you change apps, can you easily return to your place?
- Does the app warn you if you seem to be using it too much?
- How sure are you ... and how about some training first?
Solution: merge flying, UI workloads

Give the pilot, or application developer, both tasks at once!

- Simulated aircraft, so mistakes are only virtually lethal
- Simulated electronic device, so it competes fairly
- Combining simulated cockpit and physical electronic device is unrealistic because it helps pilots segregate the tasks

Then perform a conventional task driven evaluation ...
- Did the simulated pilot buy the farm, bust airspace, etc?
- Was it easier to ignore the device and do it the 'old' way?
- Would you have been scared if seated next to a real pilot, up in an airplane, watching it being flown around like that?
Two technology stacks, side by side

The pilot can only see via Xorg. The Android side stack is only accessible via the aircraft simulation.
Hardware cockpit, that's the easy case

• When the simulator is driving a full scale physical cockpit, you want the Android device to be real and full scale too!
• Android and the other instruments compete for your attention
• In this case, don't virtualize the device into the 3D model
• Tell your real device to get its GPS data from FlightGear
• This talk addresses getting everything simulated together Starting with the Android device ...
Android Emulation, kept out of sight

- Android is hosted as a QEMU session - lots of flexibility!
- Virtualizes user interface onto X11 display and its pointer
- Design a skin to fit everything into a 512 by 512 window
- Run Xvnc to move emulator off 3D immersive monitor(s)
- Use the same geometry as that skin, good for texturing
- Emulator accepts virtual GPS data from character sockets
- Use a udp network socket to deliver NMEA messages
Android 2.1 emulator in 512x512 window
FlightGear, adding an instrument

- Saved a screenshot of the emulator as texture: android.rgb
- Stuart Buchanan added a textured box on the front of the yoke.ac model file to be yoke-android.ac with that texture
- I replaced the .rgb name with the .vnc hostname for OSG (Had to vertically flip the screenshot texture and .ac file)
- Add stub hostname.vnc file as the .ac loader searches path
- Added animation to make android follow yoke movements
- Specify <vncaction> so SimGear forwards mouse events
- No source code changes were needed in FlightGear!
FlightGear yoke model, with added Android
SimGear, exposing OSG interaction

- Already associates mouse clicks with on-screen objects
- Uses `<action>` tag to change settings and trigger scripts (Those don't care exactly where the object got clicked)
- Add `<vncaction>` tag, maps 3D object coordinates to image
- New `VncCallback()` class accepts click and maps to 2D
- New `VncVisitor()` class searches object for special image (Not all textures within the object will be interactive)
- Hopefully it finds somewhere to `->sendPointerEvent()`
- Complicated because SimGear rewrites the OSG graph
- Replacing Geode with Effect wrapped by EffectGeode
- OSG doesn't understand that, so simple visiting fails
Open Scene Graph, nobody uses vnc?

- The plugin didn't work for me when built from source (Maybe nobody else has used it recently, always a risk)
- Added explicit locking of the image reference between the OSG and RFB rendering threads
- Kept the RFB protocol fully active until we see a callback to setFrameLastRendered(), for users that don't call!
- Allow the RFB thread to become active at the end once we're starting to deallocate the image
OpenSceneGraph, finally showing a VNC image
No, White wasn't intended

Seems like OSG and SimGear ... need to talk
Open Scene Graph, a better vnc image

• The plugin didn't work with SimGear even with those fixes; it didn't look enough like a RGBA image for ordinary code
• Have to fill in the top byte as 255 in the image buffer, so content is usable as RGBA (by RGB incapable tools)
• Need to ignore any directory path in the passed filename, just use the basename portion (for absolute forcing tools)
• As with a conventional file-loaded image, set the filename property of the image (in case we lose it)
• Wait until the RFB system knows our image size before returning its reference (for tools that rewrite texture data)
Open Scene Graph, securing session

• Default hostname.vnc implementation has no passwords (use firewall rules to protect client and server, maybe)
• Extended plugin source code for displays and passwords ignored/directories/hostname:59:secret.vnc
• Isn't actually necessary to get the system running but certainly makes demonstrations easier to set up
Final assembly of the pieces

- terrasync -S -p 5501 -d terrasync-cache
- vncserver :1 -depth 24 -geometry 512x512 -localhost
- DISPLAY=:1 emulator -avd fgfs
  -gps "tcp:localhost:5502,server,nowait"
- fgfs --fg-scenery terrasync-cache
  --nmea=socket,out,1,localhost,5502,tcp
  --atlas=socket,out,1,localhost,5501,tcp
- Starts the simulation, integrates with both apps
- Now choose an aircraft with a suitable cockpit that has an Android!
Some interesting detours in this work

- QEMU has native support for VNC, so omit Xvnc?
- On my system, the SDL-on-VNC configuration doesn't work
- FlightGear already has interactive textured instruments?
- They don't receive x-y mouse clicks, so no benefit there
- SimGear already delivers callbacks from mouse actions?
- Optional x,y parameters would be invasive everywhere
- OSG has a built in 3D interactive model as VNC object...?
- But the ac3d model loader doesn't allow sub-model files
Some lessons learned, or relearned

- Open protocols make it possible to combine applications
- If an implementation is rarely used, maybe it doesn't work
- Combining things in new ways is interesting and fun
- And may be difficult if the implementations disagree
- Open source means you can (eventually) fix that - at need
- How much need? Tracing C++ classes can be slow
Opportunities, Further Work

We can make the simulated integration code more efficient: eg: Emulator interfacing directly into OSG, without using Xvnc

There aren't that many apps for in-flight pilot use. Reducing pilot workload is desirable - so a market exists. This integration makes development and test much easier. We hope to see improvement with a lot more apps appearing.

Anyone wanting to build panel mounted android instruments will need to get these certified, which is a large up-front investment. Such instruments will be more desirable if there are more apps; developing them will help companies commit to such products.
Thank you

Questions?

^ Here, on the yoke

alex.perry@ieee.org
Homebuilt airplane opportunities

Many certification constraints are simpler for homebuilt airplanes.

Might mount an existing off the shelf android device on the panel.

Could wire the android device to (for example) a 2 axis autopilot so it receives off-course error data from the android application.

Before doing this, implement such wiring in the (much cheaper) simulated environment. FlightGear already has hooks for that. Then train how to use it safely, test the software algorithms, and examine failure modes with logging and diagnostics.

Experts will verify the wiring and manual disconnect, as well as check those logs to confirm you'll be able to disconnect 'in time'.