Video plugin using ffmpeg

New radR users: To view a movie showing radR being installed and used with the video plugin, install the ffmpeg package (see below) then use the ffplay.exe program to watch the flash movie attached below.

The basic target-finding algorithm in radR doesn't (yet) understand anything specific to radar, so it should be able to work with video camera data. The video plugin is a first cut at bringing such data into radR. It relies utterly on the wonderful open source ffmpeg audio/video processing package to take video input in just about any format, whether live or recorded in a file, and resample it to a sequence of grayscale images, with user-selected width, height, and frames per second. Data from these images is read directly into a radR sample matrix, and processed from there with radR's usual algorithms. See an example session in the movie attached to this page below (a lower quality flash version is here). The following screenshot comes from this processing:

ffmpeg: Installing this required third-party open source software

For radR Windows users, copies of the 32-bit and 64-bit static builds, repackaged into .ZIP files, are attached to the bottom of this page. Download the appropriate one for your version of Windows (if in doubt, use the 32-bit version), and extract it to `c:\ffmpeg`, so that the main ffmpeg program is called

```
c:\ffmpeg\bin\ffmpeg.exe
```

If you install ffmpeg to a different location, you will need to set the value of `ffmpeg.path` in the file radR/plugins/video/video.windows.conf.R.

The open source ffmpeg audio/video transcoder project is hosted here. Most linux distributions have ffmpeg available as a package. The Windows versions on this page are from http://ffmpeg.zeranoe.com/builds/.

Notes on this test video and the processing approach so far:

- the original video is in colour, but is converted on-the-fly to grayscale by ffmpeg so that there is just one data channel (i.e. gray, rather than red, green, and blue)
- colour in the image represents "blips" detected by radR
- radR picked up the duck in the foreground, and one of the more distant swallows(?), but missed at least two other birds; it is possible that a more careful choice of blip filtering parameters would do better, in particular, the choice of "int < 0.5" represents a strong filter that only allows through blips whose intensity (from black to white) is less than half the maximum (i.e. darker than 50% gray); this was chosen to remove spurious blips arising from the very low-variance upper sky region. Basically, radR's simple algorithm doesn't deal well with regions whose background "noise" distribution has fat tails.
- the blue bars are some kind of data encoded into the video, which change as you play it; they don't show up when the movie is viewed normally because they are only just slightly brighter than the black frame's background, but radR's target finding algorithm picks them up
- timestamps for this video are relative to the date/time at which the video was opened by radR.
- this video is an optimistic test case: the background is very stationary (on a windy day, it's possible the cat tails in the foreground might generate many spurious targets)
- this processing was done on a quad-core 2.2 GHz processor running Debian linux. The video was resampled on-the-fly by ffmpeg to 10 frames per second from its original 30, and at this frame rate (and resolution of 320x240 pixels), processing speed is higher than realtime, suggesting that even this crude approach is sufficient for some video processing. The bulk of the processing time is taken by radR, and not by ffmpeg, which means there is considerable room for improving processing speed by speeding up portions of radR and/or this plugin.

Changes to radR so far

- a new video plugin, modelled on the xir3000arch plugin: ~450 lines of R code, most of which is "boilerplate"; it grabs frames as 8-bit gray-scale, but remaps the values to a 15-bit integer with lower 7-bits equal to 0 (for improved resolution of stats cell mean and deviation parameters). Not super efficient (uses

R's readBin to read from the piped output of ffmpeg), but it did not require a single line of C or C++ code.

- repurposing of the "cold threshold" parameter in blip processing; it was obsolete, but now represents a low threshold, such that samples with scores below its value are treated as hot. This is required for finding targets in grayscale video, because the target may appear either brighter (i.e. large positive score) or darker (i.e. large negative score) than its background.
- a simple scan-converter (123 lines of C code), which visualizes a radR data matrix as if it represented rectangular (video) data, rather than the usual polar (radar) data; this allows zooming and panning but doesn't bother with rotation. This doesn't amount to much more than just copying the image from one location to another, but fits into radR's display paradigm.
- [Done: 02 Sept. 2011] the coordinate mappings between plot screen and sample data, and between sample data and physical (x, y, z, time) need to be modularized, so that different ones can be used for radar vs. video data; without this, tracking won't work and graphical examination of the plot window is also broken (i.e. the usual text window that tells you what the plot cursor is pointing at returns nonsense); part of this work had already been done in the phys branch of radR (to allow radR pulses pointing in arbitrary directions during a sweep), but has not been backported to the stable branch
- [Done: 02 Sept. 2011] fix apparent bug in seeking: for the *flying_birds_2.wmv* test video, at least, a manual seek prevents the first half or so of the archive from being seen (those frames are just gray)
- [Done: 02 Sept. 2011] don't wrap blip-finding around the top/bottom of rectangular data (this is currently done to allow blips in radar's polar data to cross the 0-degree cut-line)

TODO

- devise a better method to assign "scores" to samples when their underlying stats cell has zero variation. In a low resolution video with 8-bit grayscale, there can be regions of the screen that show no variation until an actual target appears. This makes the mean deviation for the corresponding stats cell equal to zero, preventing us from calculating a z-score. As a workaround, radR assigns the maximum or minimum possible score, depending on whether the newly-observed sample is brighter or dimmer than the (constant) stats cell mean, but this

causes sudden noise after a long period of constancy to also be treated as a potential target. Other, possibly better approaches:
    - replace 0 stats cell deviations with a mean of deviations from nearby cells (this just bumps the problem across the data matrix)
    - introduce fake quantization noise into the lower order 7 bits of the 15-bit video signal; because we're treating the data as 15-bits even though it really only contains 8, this is somewhat justifiable; there's probably a precendent for this in some video dithering algorithms
- eventually, we'd like to be able to use all video channels separately, as well as combined into grayscale. Whether that means as RGB or YUV or something else might depend on the application. ffmpeg seems to be able to spit out raw data in such formats, so this would mainly require changes to radR's processing stack of the same type as treating simultaneous data from multiple overlapping radars would require.
- we're using a slow pipe to get data out of ffmpeg and into radR; it would be better to have ffmpeg write individual frames into a memory segment shared by radR, to avoid repeated copying
- random access (seeking to arbitrary points in the video) is broken for some video formats: I'm not sure whether that's mpeg's fault or radR's, but playing for a few (tens of) frames usually fixes it, presumably by eventually reaching a key frame. Can we improve on this by modifying the ffplay program and use that instead of ffmpeg (ffplay is a sample client program included in the ffmpeg package) ?