Radar Video Range Correction Using a Plugin

How to write an SPx Server plugin process that can be used to apply range correction to radar video data

Summary

Cambridge Pixel HPx radar input hardware allows the user to apply a range offset correction (pre-trigger delay) to the radar video data that it captures.

However, sometimes it may be necessary to apply a similar correction to network radar video data or recorded radar video data. Furthermore, it is sometimes necessary to apply a scaling factor to the radar video data, as well as or instead of an offset.

Cambridge Pixel's SPx software framework supports the concept of user-written processing and the SPx Server application allows a compiled process to be loaded at runtime, from a dynamic-link library (Windows) or shared object file (Linux).

This application note describes how to use this facility to write a plugin process for SPx Server that is capable of adjusting the range of incoming radar video data, regardless of source, by applying a user-selectable offset and scaling.

This application note also serves to demonstrate the use of SPx Server plugins in a wider context, not just for range correction.

Introduction

Radar video data received via the network or replayed from a recording file may occasionally require a correction to the apparent range of the radar video samples. This correction may take the form of a constant offset to be added to the range of every sample or a scaling factor by which to multiply each sample's range.

Additionally, although radar video captured using an HPx card may have an adjustable range offset applied by the card itself, range scaling is not possible in the hardware. Some radar systems output an analogue radar video that is reconstituted from digital data and may be scaled as a result of the radar's internal processing. So range scaling may be may desirable even when using an HPx card.

The image below, in Figure 1, demonstrates the effect of range offset and scaling on a radar video. Radar returns all appear at some offset and multiple of their true value. The result is a radar video that is warped and doesn't align with the physical position of targets and features (e.g. coastlines).

Figure 1: Effect of range offset and scaling on radar video

The structure of radar video data within the SPx software framework makes it straightforward to correct for such range discrepancies. Radar video data in SPx comprises a header followed by an array of amplitude data samples for each return, as depicted in Figure 2. Each data sample within the return represents the radar video intensity at regular intervals across the full range extent. The return header specifies the start range and end range of the whole return and the number of samples into which this range extent is divided. This data structure is the same regardless of the original source of radar video (i.e. HPx card, network source or recording file).

In order to manipulate the range of the radar samples one needs only to edit the header for each radar data return, not the actual data samples.

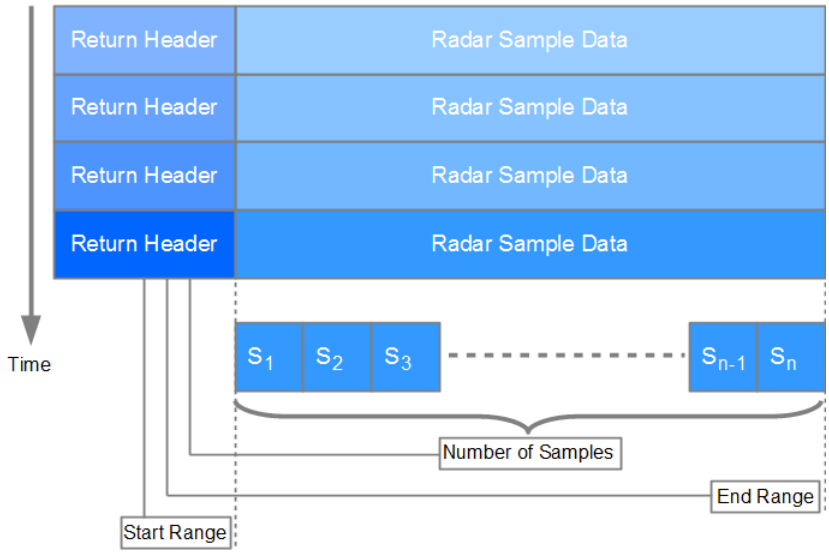

Figure 2: Radar video data structure

SPx Server Plugin

The SPx library is provided with example code for a plugin process (called "`SPxPluginProThresh`"). This document should be considered in conjunction with the "`SPxPluginProThresh`" example. The custom plugin process class is derived from the "`SPxPluginProcess`" class provided by the SPx library and the use of this class is fully described in the SPx API manual.

The custom plugin process is set up using the `SPX_PLUGIN_CREATE_OBJECT_FUNCTION` macro, and this must be used at the start of the plugin code. This macro introduces the name of the processing class that is being added. For example, to introduce a class called "`SPxPluginProRangeAdjust`":

```
/* Include this macro for each class defined in this plugin */
SPX_PLUGIN_CREATE_OBJECT_FUNCTION(SPxPluginProRangeAdjust)
```

The *`SPxPluginProcess`* class contains an initialisation function, `Init()`, that all derived classes must override. Within the initialisation function of our derived class we need to create the actual `SPxProcess` object that will perform the work:

```
m_process = new SPxProcess("Range Adjust", applyRangeAdjust);
```

Where `m_process` is an `SPxProcess` object, declared in the base *`SPxPluginProcess`* class. The constructor for the `SPxProcess` object is given two arguments here. The first one simply sets the name of the process and the second one is a function pointer to the function that actually performs the processing (see below).

Also within the initialisation function, we need to set up two process parameters: one to hold the additive offset and the other to hold the multiplicative scaling factor. For example:

```
REAL32 delta = SPxGetSafeReal("rangeDelta", 0.0, name);
REAL32 scale = SPxGetSafeReal("rangeScale", 1.0, name);
```

Where `name` is the name of the process and is automatically passed into the function call as an argument.

Next, still within the initialisation function, we need to make these new parameters available through the SPx Server GUI:

```
SPxProcessParameter *procParam;
m_process->AddParameterR("rangeScale", scale);
procParam = m_process->GetLastParameter();
procParam->SetCtrlDesc("Range scaling factor");
procParam->SetCtrlSpin();


m_process->AddParameterR("rangeDelta", delta);
procParam = m_process->GetLastParameter();
procParam->SetCtrlDesc("Range offset delta");
procParam->SetCtrlSpin();
```

Finally, the function that actually performs the processing needs to be written. The prototype of the actual processing function takes the form:

```
/* The actual function that implements our process. */
static void applyRangeAdjust(SPxRunProcess *rp, SPxReturn *r,
                             unsigned firstAzi, unsigned size);
```

This processing function is called automatically each time the process receives new radar video data from one or more azimuths. The `SPxRunProcess` pointer provides access to data relating to the process's execution, for example its parameter values, the source of data (input PIM) and the output destination (output PIM).

The `SPxReturn` pointer provides access to the first azimuth return that has been updated. "firstAzi" is then the azimuth index relating to this return and "size" is the number of azimuths that have been updated. The processing function should take into account the fact that it may be called not just for a single updated return (i.e. size == 1) but for multiple updated returns (i.e. size > 1).

The full listing for a range correction function is given below.

```
static void applyRangeAdjust(SPxRunProcess *rp, SPxReturn *r,
                             unsigned firstAzi, unsigned size)
{

    /* Get the parameter values */
    REAL32 scale = rp->GetParamValueR("rangeScale");
    REAL32 delta = rp->GetParamValueR("rangeDelta");

    /* Get the PIMs that we will use for input and output. */
    SPxPIM *inputPIM = rp->GetPIMA();
    SPxPIM *outputPIM = rp->GetPIMB();

    /* Set pointers to PIM buffers */
    SPxReturn **inputStore = inputPIM->GetStoreMemory();
    SPxReturn **outputStore = outputPIM->GetStoreMemory();

     /* Copy input to output */
    memcpy(outputStore, inputStore, sizeof(SPxReturn));

    /* Process each azimuth */
    for (unsigned int i = firstAzi; i < (firstAzi + size) ; i++)
    {
        /* Get azimuth number, allowing for crossings of North */
        unsigned int azimuth = i % inputPIM->GetAzimuthDim();

        /* Set pointers to azimuth to process
           and apply the scaling here.... */
        SPxReturn *rtn = outputStore[azimuth];
              rtn->header.startRange += delta;
              rtn->header.endRange += delta;
              rtn->header.startRange *= scale;
              rtn->header.endRange *= scale;
    }

}
```

The range correction is applied by adjusting the start and end ranges for each header. This is done within the "for" loop of the function above.

SPx Server Configuration

Once the plugin process has been written and compiled to a dynamic link library or shared object file, the final stage is simply to configure SPx Server to load and use the new plugin. The following settings should be made in the SPx Server configuration file:

```
ProUser0.PluginName = SPxPluginProRangeAdjust # Name of plugin file (no ext)
```

This parameter specifies the actual filename of the compiled plugin (without the file extension). If the filename is relative (as above) then the current working directory is checked first followed by the $(SPX)/SPxPlugins directory. If the filename is absolute (starts with either '/', '\' or '.') then the path to the filename is used as defined.

```
ProUser0.PluginClass = SPxPluginProRangeAdjust      # Name of the plugin class
```

This parameter tells SPx Server the name of the processing class to load from the plugin. It's possible for a plugin to contain multiple classes, so this parameter allows SPx Server to select the appropriate one.

```
ProUser0.InstanceName = ProRangeAdjust       # Alias assigned to this instance
```

This parameter sets the name of the plugin process object within SPx Server. The process is controlled by reference to this name. The process parameters defined within the `SPxPluginProcess` initialisation function are prefixed by this name, as is the "Enabled" parameter to turn the process on/off:

```
# Plugin parameters are prefixed with the instance name
ProRangeAdjust.Enabled = 1      # Initial state, 0 for off, 1 for on
ProRangeAdjust.Delta = 0        # Range offset
ProRangeAdjust.Scale = 1.0      # Scaling factor
```

The SPx Server GUI provides graphical controls for the parameters defined within a plugin process. This control is available under menu bar→"Preprocess"→<*InstanceName*>→"Configure…". An example is shown in Figure 3 below.

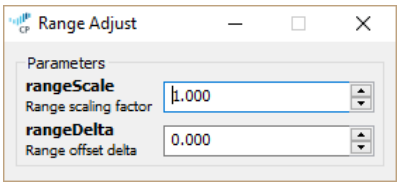

Figure 3: Example plugin process control in SPx Server

Now the range offset and scaling demonstrated in Figure 1 can be corrected for, as shown below in Figure 4, and the radar returns line up correctly with the physical features.

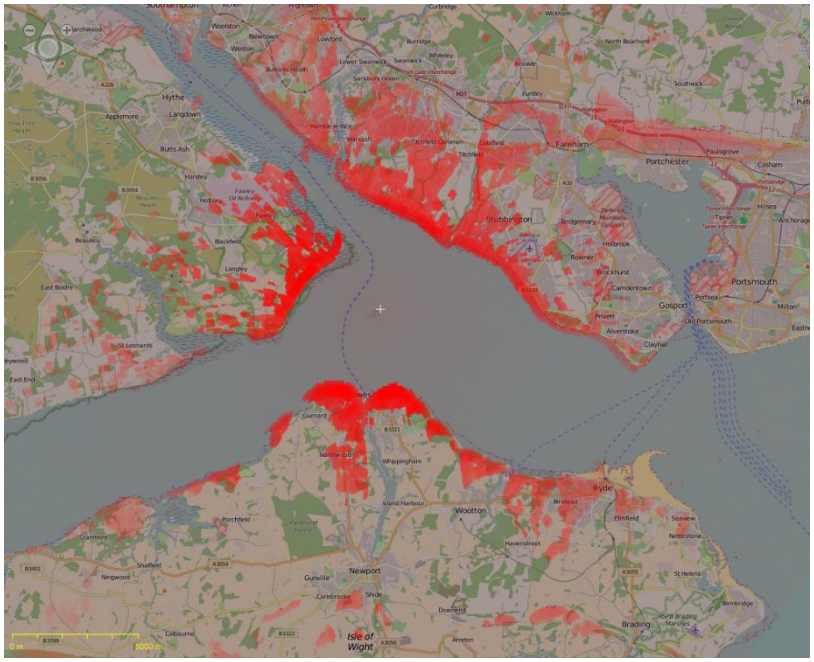

Figure 4: Radar video after range offset and scaling correction

A note on versions: the SPx library version number (major, minor and micro parts) used to build the plugin must correspond exactly to the SPx Server version. If there is any discrepancy between the two version numbers SPx Server will not load the plugin file and the process will not be available.

< End of document >