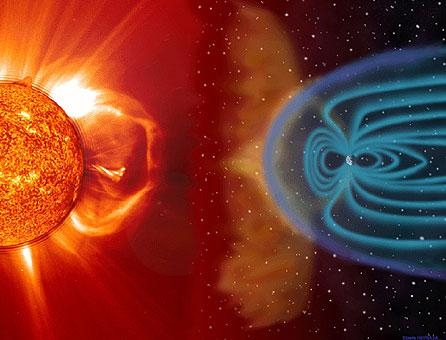
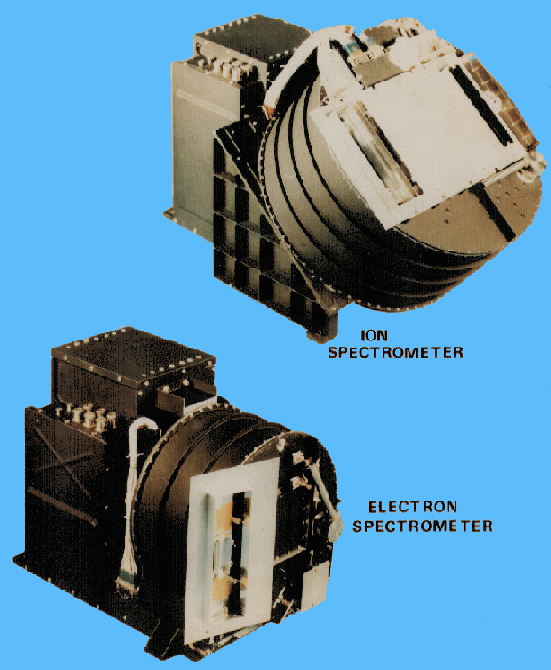
**Solar Wind Assignment**

The aim of this exercise is to build a graphical visualisation of live data feed that is streaming back from an orbiting satellite. The satellite in question is the Advanced Composition Explorer (ACE) which is constantly monitoring the behaviour of our Sun, watching for solar flares, solar storms and “mass ejections”. These powerful phenomena cause the Aurora Borealis on earth and have the potential to damage satellites and disrupt communication systems.

On board the ACE satellite is the Solar Wind Electron, Proton, and Alpha Monitor (SWEPAM) a special sensor which provides stream of data to us back on earth. The American National Oceanic and Atmospheric Administration (NOAA) provides a live data feed of the last 2 hours of solar activity:

<http://services.swpc.noaa.gov/text/ace-swepam.txt>

Our aim is to produce a graphical representation of this data

For this exercise, we will be using a Java-based programming language called Processing. It’s a powerful language, but has a gentle learning curve and is easy to get started for less experienced programmers.

**Task 1: Understand and Run Template Code**

A template program has been provided to help get you started. Take a look at it and try to understand what it does. Comments in the code have been included to explain some of the less obvious lines of code. When you feel that you have a reasonable grasp of the code, run it by clicking the “play” button.



**Task 2: Jump Back**

Currently, the basic program works its way through each line in the data set, printing out a single reading (the proton density). When it reaches the end of the data file, it just stops. Your first task is to change the code so that each time it reaches the end of the data, it jumps back to the start and begins again (hint: you’ll need to do something with the “lineNumber” variable)

Note: You may like to switch to using a local cache of SWEPAM data (as it can make testing faster)

Replace: *data = loadStrings(URL);*

With: *data = loadStrings("ace-swepam.txt");*

**Task 3: Drawing**

Once you are happy that your program cycles through the data continuously, it is time to start creating some graphics. The aim is to use the numerical value of density to change the colour, size and shape of graphical objects shown in the drawing window. To do this, you may find the following drawing functions useful:

* fill(): set the fill colour of subsequently drawn objects. Expects 3 parameters: Hue, Saturation and Brightness (take a look at the top-bar menu “Tools> Colour Selector…” for help)
* stroke(): set the outside edge colour of subsequently drawn objects (same parameters as fill)
* ellipse(): draw a circle with the current fill and stroke colours. Expects 4 parameters: xPosition, yPosition, width and height
* line(): draw a line with current stroke colour. Expects 4 parameters: fromX, fromY, toX, toY

Other shapes are available: https://processing.org/examples/shapeprimitives.html

We would suggest starting simply – just draw an ellipse (circle) who’s colour and size is set by the current value of proton density.

**Task 4: More than one data value**

Once you have a changing graphical representation of the proton density data, write some additional code to read in one of the other values from the data stream (temperature, or speed). Use this new value to change an aspect of your data visualisation (e.g. density affects the size of your drawn objects and temperature affects the colour).

**Task 5: Filtering Bad Data**

Occasionally there is a “bad” value in the data stream (a density of -9999.9). This is most likely due to a mis-reading being taken by the SWEPAM device. These values can really mess up a graphical visualisation, so add some code to filter them out.

**Task 6: Build as a Mobile App (optional)**

One of the nice features of Processing is that it is (relatively) easy to export a program as an Android App. This is mainly because Processing and Android are built around the same programming language. There are a couple of small changes that we need to make in order to build an app:

1. Switch Processing to “Android” mode (using the menu in the top right-hand corner)
2. Switch your phone into “USB Debugging” mode (process varies depending on the phone)
3. Connect phone to laptop via USB cable (accept any permission requests shown on phone)
4. Remove the *size(700,700);* line of code (the phone will just use a full screen resolution)
5. Press the “play” button in Processing
6. Wait (it can take a while to compile, transfer, install and run the app on the phone)

**Task 7: Gyroscope (totally optional)**

Now that you have a working App, it would be good to use some of the features of the phone…

Add the following line to the very start of your program: *import ketai.sensors.\*;*

Add the following to the end of your setup function: *KetaiSensor sensor = new KetaiSensor(this);*

*sensor.start();*

Add a new function at the end of your code:  *void onGyroscopeEvent(float x, float y, float z)*

The above code will provide your program with access to the gyroscope that is built into the phone. Every time the phone is moved, the *onGyroscopeEvent* will get called, with changes in orientation in 3 dimensions being passed in as the 3 parameters (x, y, z). You task is to work out which of the three represents motion in the horizontal plane and then use the values to shift your visualisation around the screen. The idea being that, as the user physically turns around, the visualisation always stays in the same place (Ask for a demo from the supervisor as this probably makes no sense on paper !).