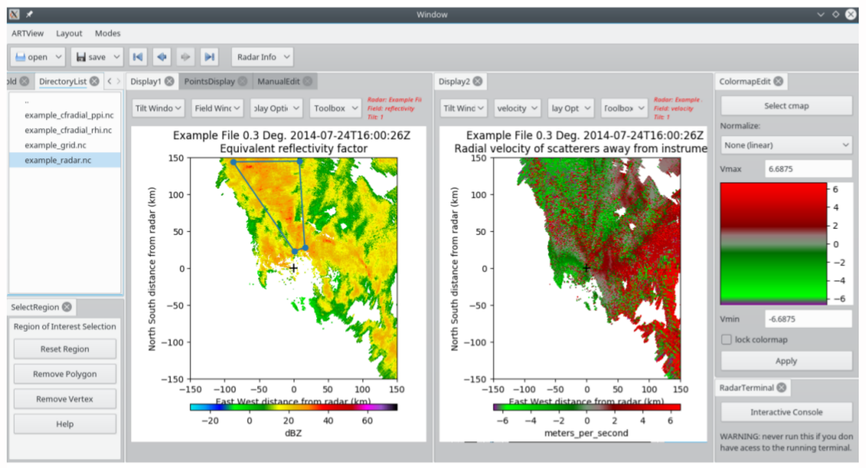
ARTview:

Towards a Open Source Radar Visualization Tool

A Work of Love from:
Anderson Luis Gama (University of Stuttgart)
Nick Guy (Climate Corporation)

CONTACT: artview-users@googlegroups.com
gamaanderson92@gmail.com
nick.guy@climate.com



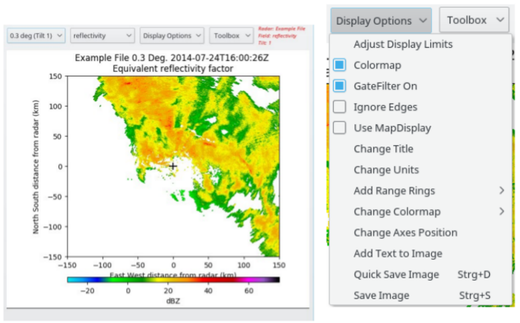
Abstract

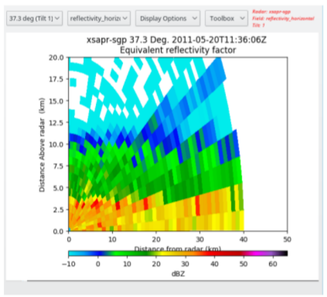
- ARTview is a open source tool to visualize, analyze and modify radar data. It is based upon Py-ART and build with PyQt.
- Its modular structure makes it extremely flexible and allows for different configuration and is easy to expand.
- As the visualization part is already robust, focus right now is in expanding the editing capacities and simplifying the use.
- Go through the pages to learn more about ARTview's current functionalities.
- The source code can be found at: http://nguy.github.io/artview

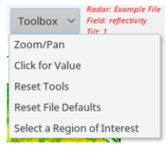
DOI 10.5281/zenodo.47224

Radar Display

- Display Py-ART Radar object.

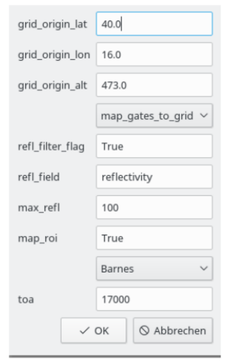


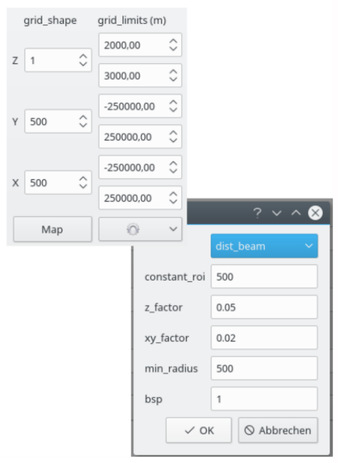




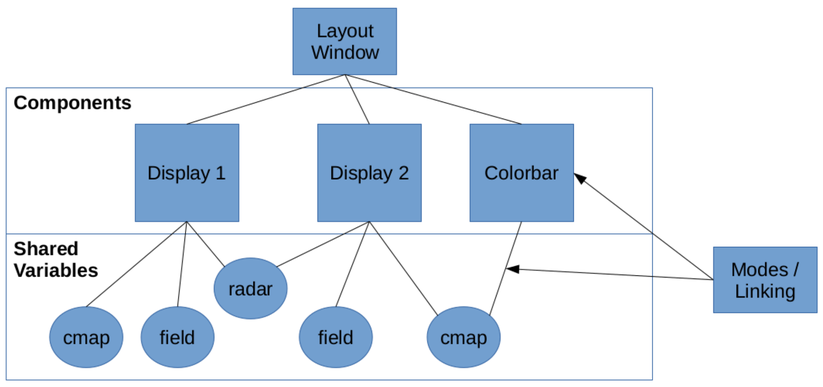
Map to Grid

- Use Py-ART interpolation tool to generate a Grid object from Radar data.
- Change the Interpolation parameters and see the results in real time.



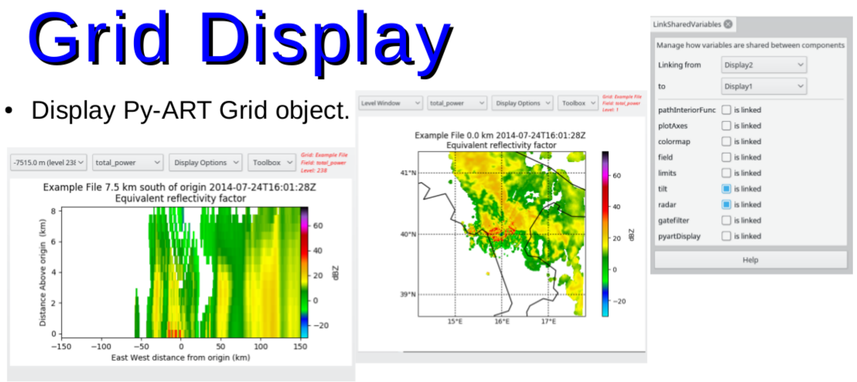


Modular Structure



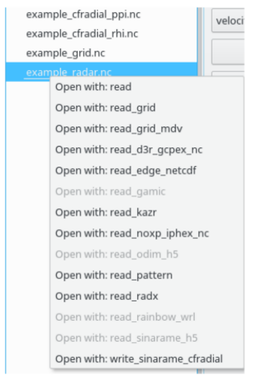
Grid Display

- Display Py-ART Grid object.



Open File

- There are two components use to open files:
- One to easily navigate within a directory and save alterations.
- And one to browser between directories



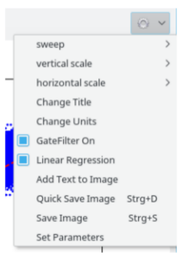
Scripting

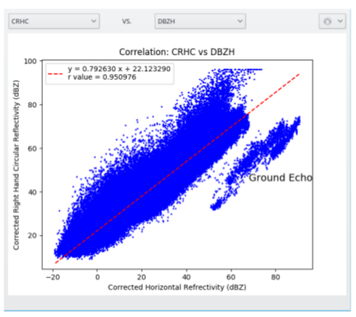
- ARTview can be call as a program or run from python. This allow customizing its design. Here is an example of such:

```
#import
import sys, pyart
from artview import core, components, plugins, view

# start pyqt
app = core.QtWidgets.QApplication(sys.argv)

#open navigator
navigator = components.FileNavigator(filename="/home/vagrant/artview/data/example_radar.nc")
radar = navigator.Vradar.value

#start RadarDisplay
plot1 = components.RadarDisplay(navigator.Vradar,name="Display 1")
plot2 = components.RadarDisplay(navigator.Vradar,name="Display 2")

window = components.Window()

window.splitHorizontal()
window.splitVertical()

window.layoutTree[(0,1,0)].addTab(linking,linking.name)
window.layoutTree[(0,1,0)].isClosable = False
window.layoutTree[(0,0)].addTab(navigator,navigator.name)
window.layoutTree[(0,0)].isClosable = False
window.layoutTree[(0,0)].tabBar().setVisible(False)
window.layoutTree[(0,1,1)].addTab(plot1,plot1.name)
window.layoutTree[(0,1,1)].addTab(plot2,plot2.name)

#start PyQt loop
app.exec_()
```

Correlation

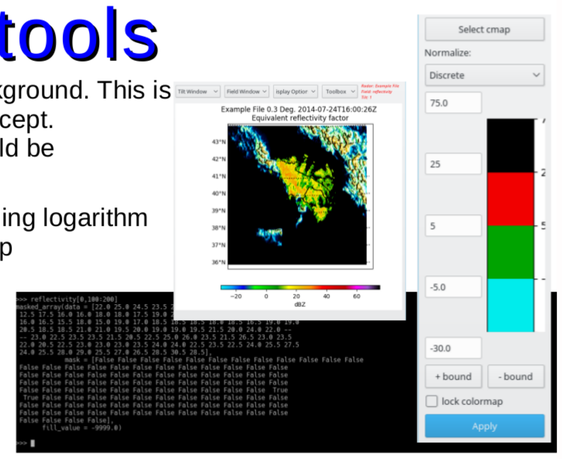
- Easily makes a correction plot of two fields to Analise the correlation.





Other tools

- Add topography background. This is mainly a proof of concept. Implementation should be application specific.
- Colormap edit, including logarithm and discrete colormap
- Direct real time access to the underling data. Modify it in python and see the result in ARTview



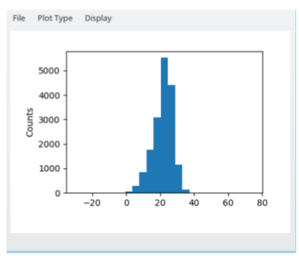
Plug-in

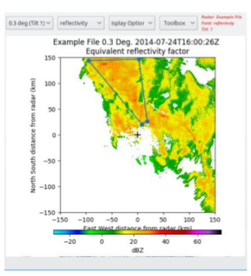
```
import artview

class RealTime(artview.core.Component):

    @classmethod
    def guiStart(self, parent=None):
        return self(), True

    def __init__(self, name="RealTime", parent=None):
        '''
        make test radar for animation
        '''
        super(RealTime, self).__init__(name=name, parent=parent)
        self.buildGui()
        self.defineSharedVariables()
        self.show()

    def defineSharedVariables(self):
        self.Vradar = artview.core.Variable(None)
        self.sharedVariables = {"Vradar": None}

    def buildGui(self):
        self.button = artview.core.QtWidgets.QPushButton("Start/Stop")
        self.setCentralWidget(self.button)
        self.button.clicked.connect(self.start_stop)
        self.running = False

        self.timer = artview.core.QtCore.QTimer()
        self.timer.timeout.connect(self.loop)
```

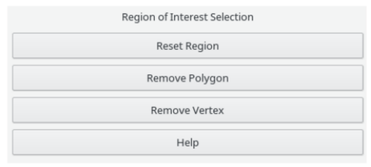
```
    def start_stop(self, *args):
        self.timer.start(5000)

    def loop(self):
        import subprocess
        # get file sorted by name
        a = subprocess.check_output(
            "ls -l plugin_data | awk '{print $9}'", shell=True)
        last_file = str(a).split('\\n')[-2]
        self.statusBar().showMessage("open plugin_data/"+last_file)
        self.openRadar("plugin_data/"+last_file)


    def openRadar(self, filename):
        import pyart
        radar = pyart.io.read(filename)
        self.Vradar.change(radar)

def realTime_mode():
    realTime = RealTime()
    display = artview.components.RadarDisplay(realTime.Vradar)

artview.modes.append(
    {'label': 'RealTime',
     'group': 'Personalised',
     'action': realTime_mode})

artview.scripts.layout()
artview.plugins._plugins["RealTime"] = RealTime
artview.run()
```

Data selection

- Use the mouse to select data to extract, analyze, filter and unfold







What is Next?

We develop for the Community, so we rather that you tell us:

- Auto-load multiple files?_____
- Python Console?_____
- Animation?_____
- Tutorials?_____

More Ideas?

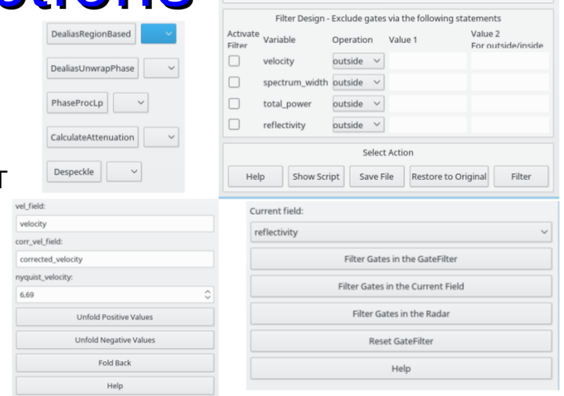
_____
_____
_____
_____

Layout Window

- The several components are organized as tabs in a layout window with a dynamic Panes. Panes can be divided and closed.
- Tabs can be dragged between panes or popped out to new windows.

Corrections

- ARTview allow several ways to correct and filter radar data.
- This include Py-ART algorithms and well as manual forms of data selection.



Acknowledgement

- We would like to thanks the Open Source Community for developing the tools that made this project possible.
- In special we thanks the developers of Py-Art: **JJ Helmus and SM Collis, JORS 2016, doi: 10.5334/jors.119**
- We thanks all that have collaborated in this project, in special:

Brenda Dolan

Timothy J Lang

Paul Hein