Current State of Zooniverse Jets

Charlie Kapsiak

September 1, 2020

Where is it?

All work can be found in the github repository: `https://github.com/CharKap/Solar_Zooniverse_Processor`.

Topics

- Preprocessor finished

Topics

- Preprocessor finished
- Documentation mostly finished

Topics

- Preprocessor finished
- Documentation mostly finished
- Zooniverse page mostly finished

Image Preprocessing

Image Preprocessing

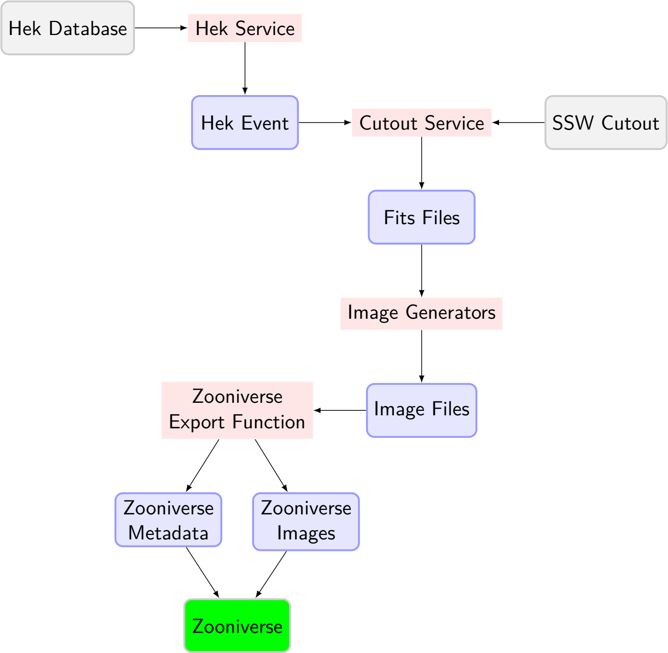
Program Features

More On Visuals

Aggregation

Package Capabilities

- Interface with external services like HEK
- Store persistent information to avoid redundant calls to the services
- Generate visuals based on information acquired from these services
- Export visuals and data in a format compatible with Zooniverse
- Import data from zooniverse as usable python objects
- Aggregate the imported data



Hek Database

Hek Service

Hek Event

Cutout Service

SSW Cutout

Fits Files

Image Generators

Image Files

Zooniverse Export Function

Zooniverse Metadata

Zooniverse Images

Zooniverse

Program Features

Image Preprocessing

Program Features

More On Visuals

Aggregation

Setting up the Database

In order to have persistence, we must first create the database. This is done by using the function create_tables().

```
import solar.database as db
db.create_tables()

```

Getting Events From HEK

Once the database has been setup we can begin to search the hek database for potential events. The interface for the HEK api is provided by the class Hek_Service.

```
import solar.service.hek as hserv
hek = hserv.Hek_Service(
    event_starttime="2015-10-01T00:00:00",
    event_endtime="2015-11-15T00:00:00",
    event_type=["cj"],
)
hek.submit_request()
found_events = hek.fetch_data()
hek.save_data()
```

Getting Fits Files from the Cutout Service

There are several ways to generate a new Cutout_Service. One may use an existing request or create one from attributes.

```
from solar.database import Hek_Event, Cutout_Service
from solar.service.attribute import Attribute
cutout = Cutout_Service(Attribute("param1", val1), \
        Attribute("param2", val2))
cutout = Cutout_Service(param1 = val1, param2 = val2)
event =  Hek_Event.get()
cutout = Cutout_Service._from_event(event)
old_cutout_request = Service_Request.select().where(
            Service_Request.service_type='ssw'
        ).get()
cutout = Cutout_Service._from_model(old_cutout_request)
cutout.subit_request()
cutout.fetch_data()
cutout.save_data()
cutout.save_request()

```

Generating Visuals

Visuals can be generated by using the image factories found in the solar.visual.img. Videos can be generated using the factories in solar.visual.vid.

There are two ways to generate visual. If persistence is not required, then the image can be generated using the factory itself.

```
import solar.visual.img as im
image_builder = im.Basic_Image("png")
f = Fits_File.get()
image_builder.create(f.file_path)
image_builder.save_visual(f,"savepath.png")
```

```
import solar.visual.img as im
from solar.database.tables.visual_file \
        import Visual_File
image_builder = im.Basic_Image("png")
f = Fits_File.get()
db_image = Visual_File.create_new_visual(f,
    image_builder)
```

Exporting To Zooniverse

Once we have a collection of images, we can export them. The function zooniverse_export() Takes a variable number of lists of lists of visual files and outputs them in a format readable by zooniverse.
The split function is used to break a list into manageable chunks, with overlap.

```python
import solar.zooniverse.export as ex
files_per_subject = 10
subject_overlap = 2
v = Visual_File.select().where() # Search is narrowed here
ex.zooniverse_export(split(split(v,files_per_subjet, subject_overlap)))
```

More On Visuals

Image Preprocessing
Program Features
More On Visuals
Aggregation

Annotations

The package includes tools to annotate images.

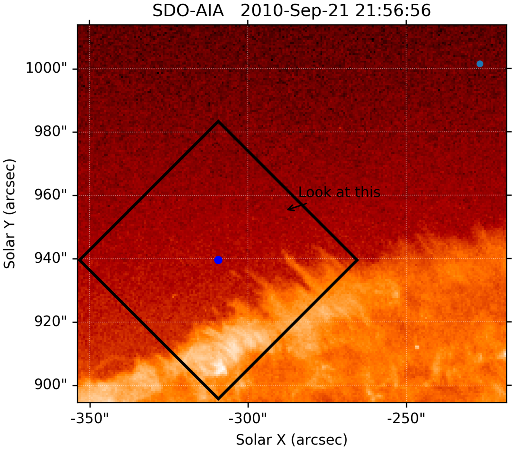
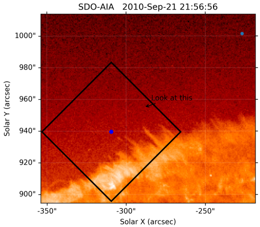


Figure: images/Annots

Annotations

```python
rect = Rect_Annot(x=0.4, y=0.4, w=0.3, h=0.4, a=45, lw=2)
circ = Circle_Annot(x=0.8, y=0.8)
rect_center = Circle_Annot(x=0.4, y=0.4, color="blue")
text = Text_Point(0.5, 0.5, "Look at this")
bim.add_annotation(rect, circ, rect_center, text)
fits_database_id = 1
f = Fits_File.get(Fits_File.id == fits_database_id)
bim.create(f.file_path)
```



Aggregation

Image Preprocessing

Program Features

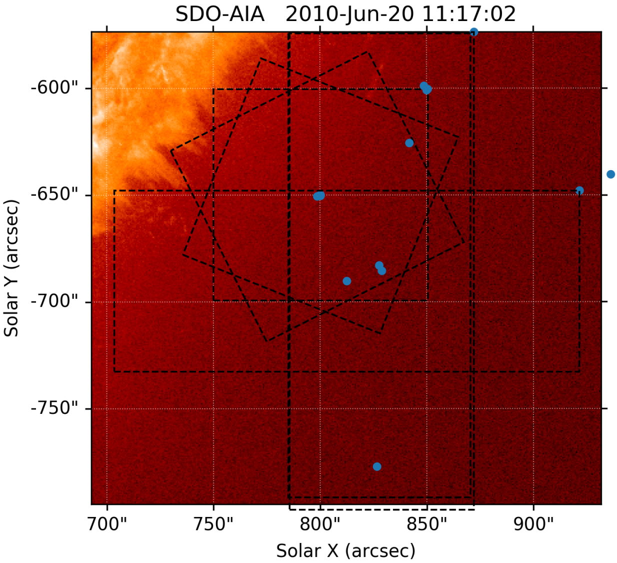
More On Visuals

Aggregation

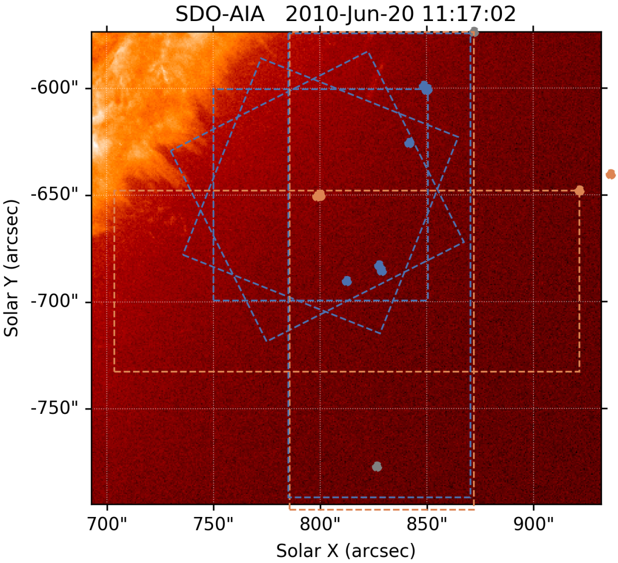
Aggregation

At present, this program contains only rudimentary methods for aggregating and processing the classified data.
The purpose of data aggregation is take the large number of different classifications made by zooniverse volunteers and attempt to extract a smaller amount of high quality data by doing some sort of “averaging.” Of course, because of the complexity of the data, simply averaging is insufficient. Instead we use a number of clustering algorithms.

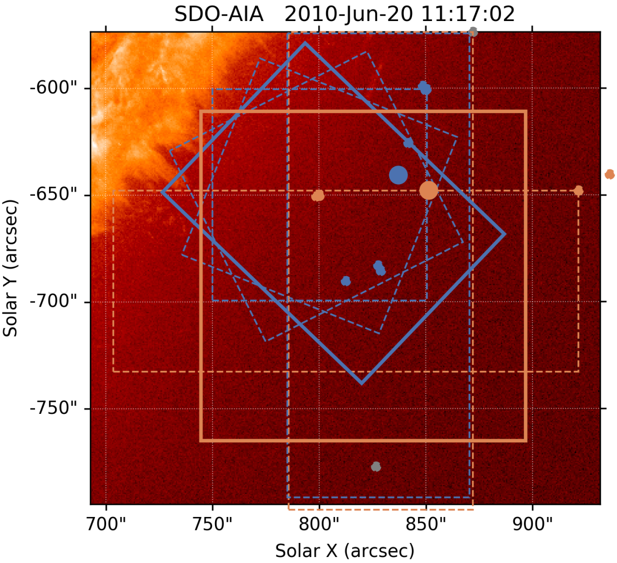
Example


SDO-AIA 2010-Jun-20 11:17:02
-600"
-650"
-700"
-750"
Solar Y (arcsec)
700"
750"
800"
850"
900"
Solar X (arcsec)

Example


SDO-AIA 2010-Jun-20 11:17:02
Solar Y (arcsec)
-600"
-650"
-700"
-750"
Solar X (arcsec)
700"
750"
800"
850"
900"

Example


SDO-AIA 2010-Jun-20 11:17:02
Solar Y (arcsec)
-600"
-650"
-700"
-750"
Solar X (arcsec)
700"
750"
800"
850"
900"

Thank You

Questions?