Note that in the programs there is a filepath function that directs to folder in which the images are kept, filename will pick out the specific image. The filepath is specific to the computer, so it will need to be updated for different computers

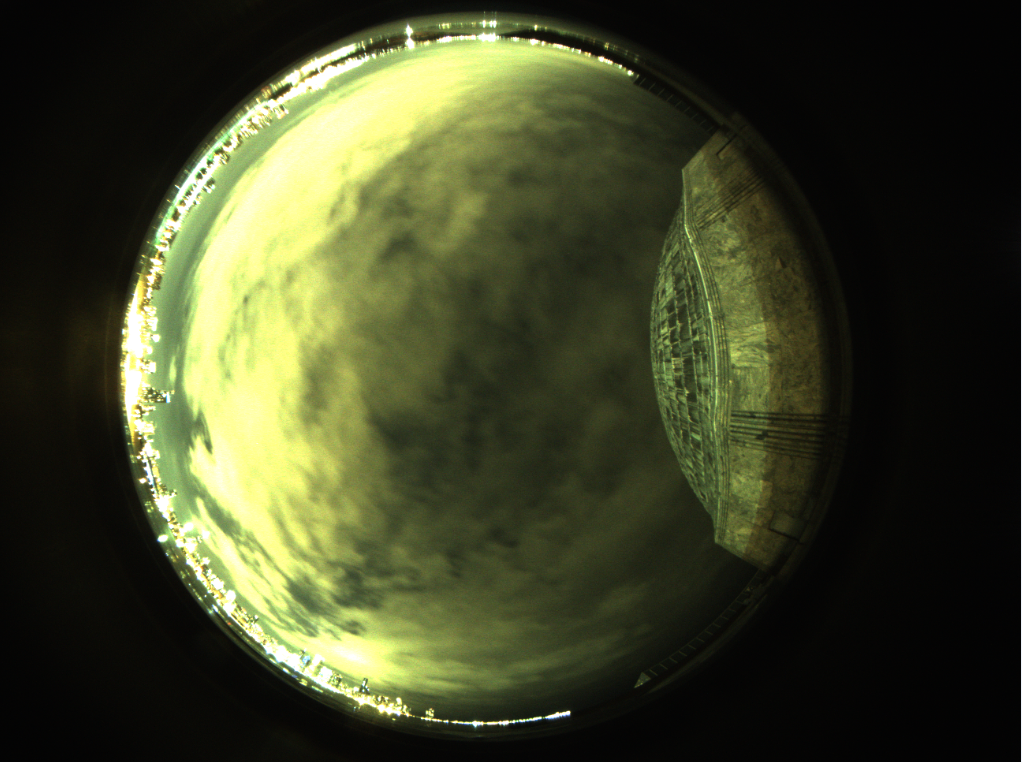
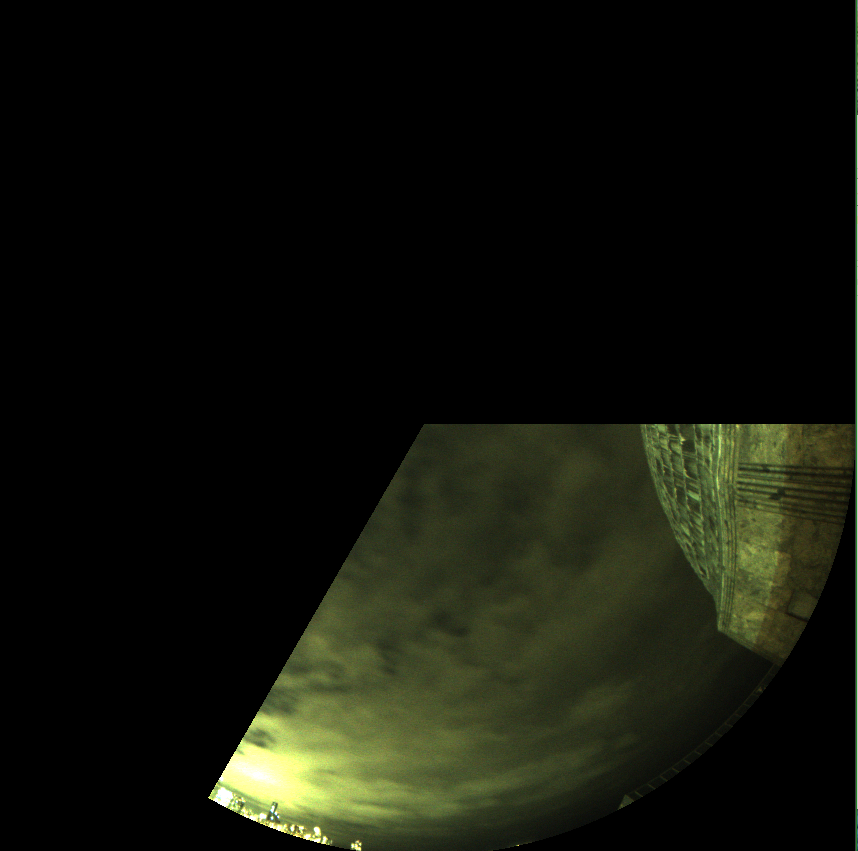
**Goals and hopes:**

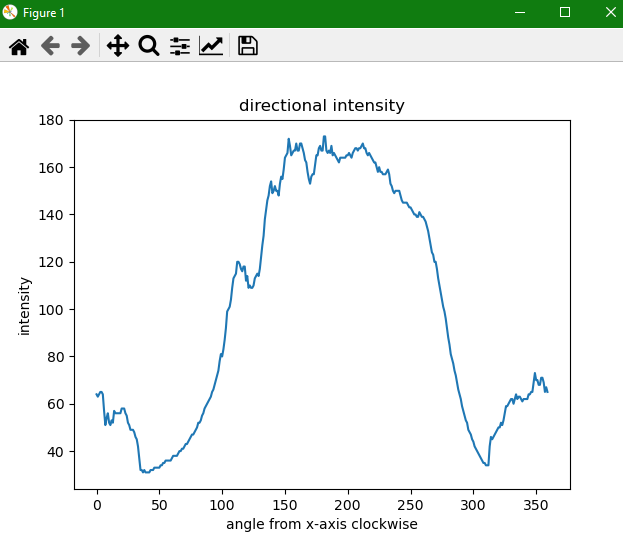
* Be able to crop or mask out stationary objects in all-sky image (trees, buildings, etc)
* To subtract out the stars from the image using starmaps of the given time
* 3D intensity plot connecting pie\_slice to radial intensity, should not be difficult
* Look into other ways of defining the center of the actual image(as opposed to Hough Circles which are inexact and most useful for rough analysis, such as the slices and radial intensity.

**Progress:**

As of 13 April 2021 the most significant Progress I have made is learning the ins and outs of Python, learning of different incredibly useful libraries, cv2, matplotlib, etc.

**Pizza Slice:**

The pizza\_slice.py file currently contains functions that will allow a tiff file (of specified size?) to be read and then split into “pie” slices from which an intensity vs angle graph is drawn. The graph starts at the positive x-axis going clockwise in degrees,.

Example tiff Image The first image of the tiff cut into 3 slices 

The pie\_plot intensity graph, you can see the large mound corresponding to the bright clouds.

**How I went about doing it**

There are 5 functions going into this procedure, each accomplishes one goal.

* find\_center(image)  
  A problem with the all-sky images I discovered is that they are not centered on the image, so I would need to find the center of the all\_sky circle. This is done with the cv2.HoughCircles function. This function highly depends on the image size, the parameters I have set work for the size of the images we get from the all-sky camera, I plan on specifying the image size, and if an incorrectly sized image is input, it should automatically resize it. The accumulator resolution, minDist, minRad, maxRad all affect how the function works with size. If improperly calibrated the HoughCIrcle will return multiple circles and their centers and mess up the process.   
   Note that this HoughCircles function is not an exact science, it likely does not return the exact center of the circle.  
   This Function returns the center and radius of the image in the form (x, y, r)
* crop(filename)  
  This function is there to simplify later analysis, while I could use the center given by find\_center() it would complicate matters. This function crops the tiff image into a square centered on the image. As it needs to know the center, this function calls find\_center() in order to do so.  
   This function returns the cropped image and the radius, that is half the width of the image.
* slices(radius, sectors)  
   This function is there to create the pie\_slice masks from which the tiff file will be subdivided. That is it creates a list of images, where each element of the list is in the form (pie\_slice, startangle) where pie\_slice is just a binary color pie slice image of appropriate size.  
   This function returns that list of (pie\_slice, startangle)
* pizza\_int(filename, sectors)  
   This function finds the angle dependant intensity of the tiff file. It calls the crop() function to center and square the all\_sky image, it also calls slices() to give it the appropriate sized masks in list form. This then compares the cropped tiff image to each mask and returns a list of points of the form (starting\_angle, average intensity of pie slice)
* pie\_plot(filename)  
   This function is very straightforward, it calls pizza\_int() and then plots its list of (angle, intensity). Note in this I have set the number of sectors (pie slices) to 360 to correlate deltax as one degree, I can have this lowered as 360 sectors increases the time to process one image

**Notes**:

This process depends heavily on the size of the tiff image, I will work on limiting that dependency.  
If the orientation of the all\_sky camera is known, we should be able to define directions as opposed to angle, that is North, South, East, and West. This way we would be able to describe which direction from the camera is the brightest or darkest.

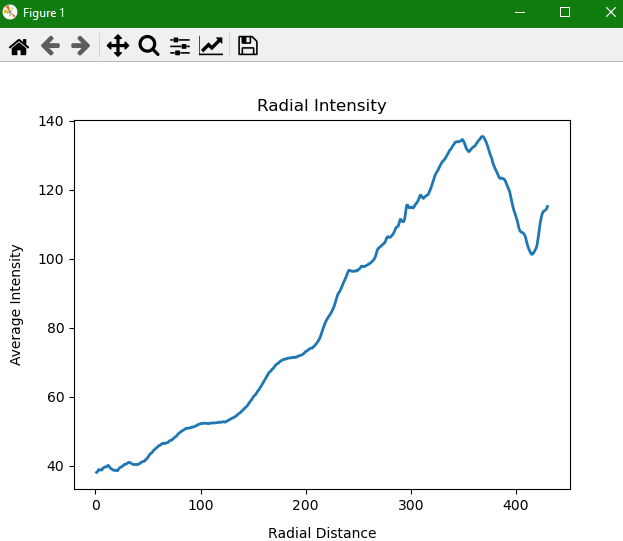
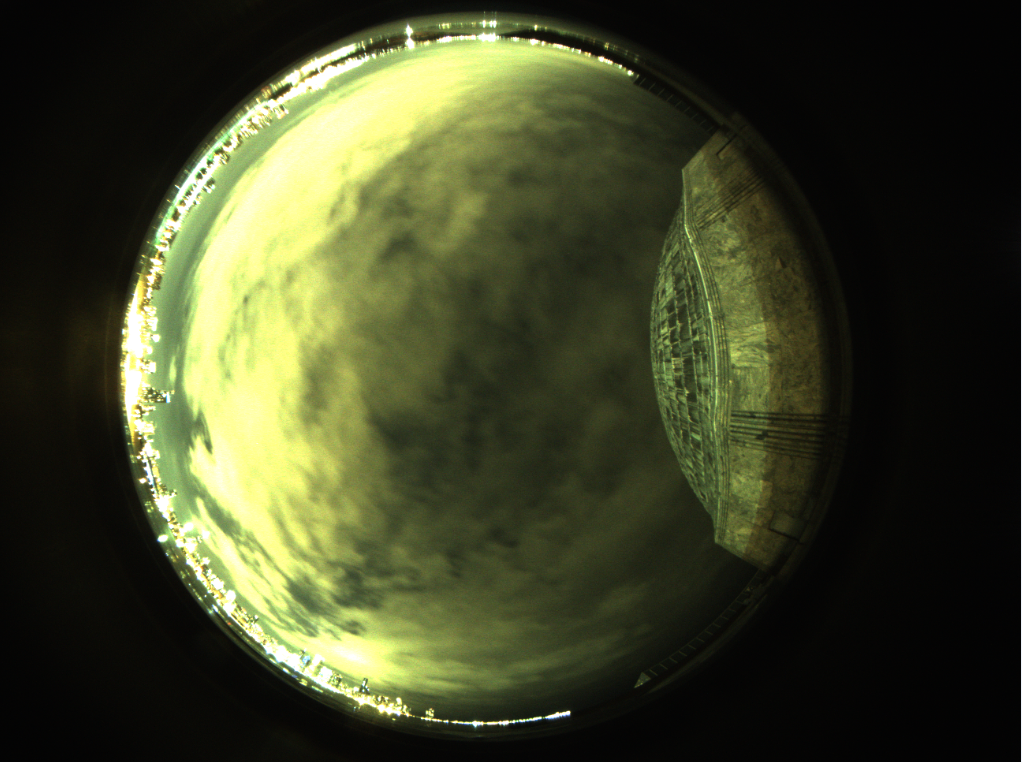
Currently all this does is return the average pixel intensity of the pie\_slices, ideally we would convert this into whatever units is most useful, lumens or dark\_sky units.

It is important however to know that this shows the variation of sky brightness depending on direction. In an ideal dark\_sky there would be no variation in direction, excepting for the presence of the moon.

One of the biggest issues in the process is it does not recognize non sky objects. That is if there are trees or buildings in the image it processes that all the same. Though such objects tend to be darker, because of shadows, so it will show up on the graph. If we can account for light pollution interference, this may be a way to map out the presence of stationary objects.

**Radial Intensity:**

The pizza\_slice.py file also has a function for radial intensity.



Original Tiff image The radial intensity graph

**How I did it:**

This process uses the same find\_center() and crop() as the pizza\_slice function.

The process uses meshgrids to organize the data into radial sets, I’m not sure how it does it beyond the fact that it gives results.

* rad\_int(filename)  
  This function calls the crop() function to square and center the image. Then by recording the center of the image and its “radius” the function constructs a meshgrid on which Radius is computed. Then bins are made in which intensity is averages for a given radial annular of width 1 pixel. It then plots using matplotlib’s internal functions.

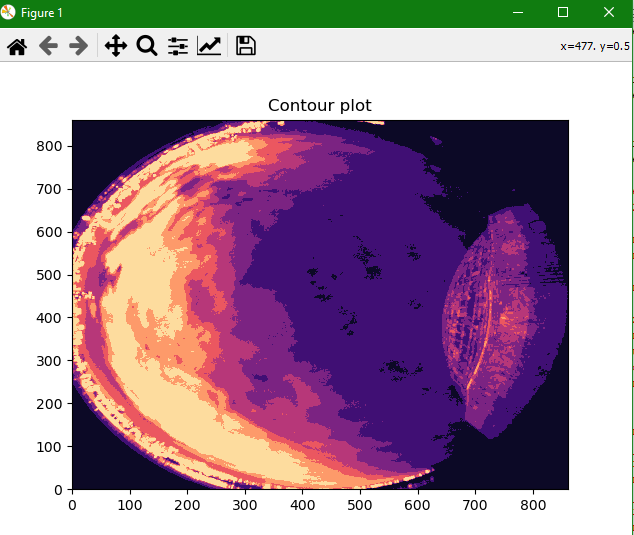
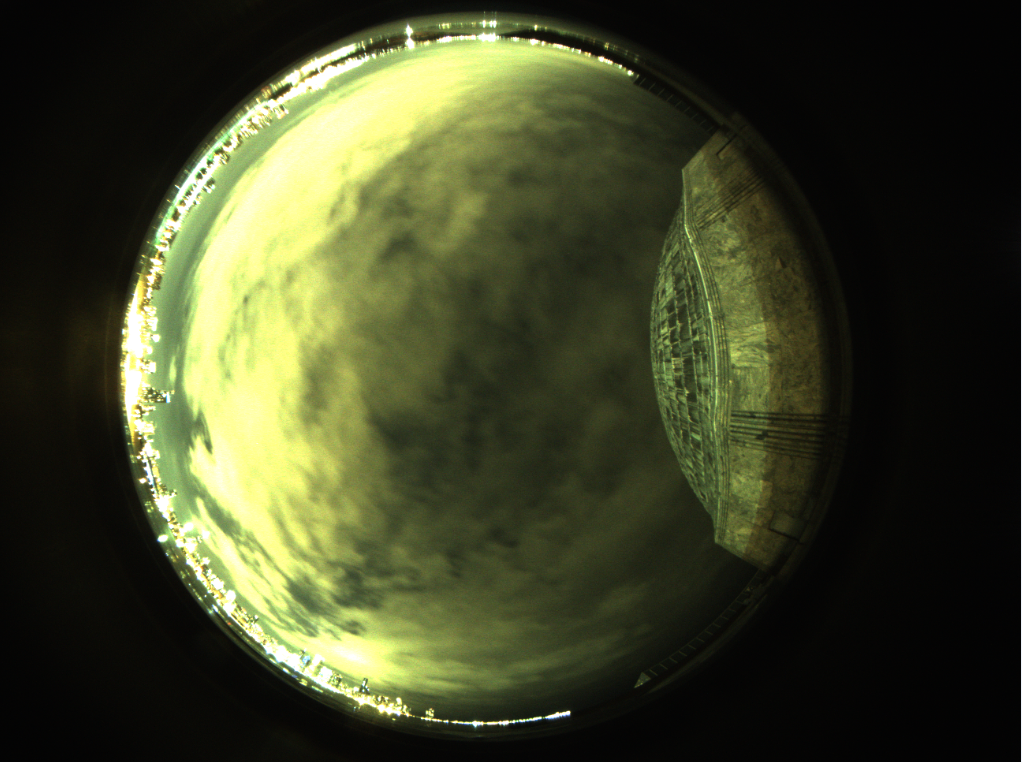
**Notes:**

I don’t really have many notes on this process except commenting that I do not truly understand how the meshgrid works, so adjusting it would be a trial and error sort of issue.

I think this process is very complete and shows great data. However, like with all the other processes it does not know how to deal with stationary objects, so that will affect the data.

Possible progression with this is connecting it to the pizza\_slice intensity to create a sort of 3d intensity plot.

**Contour of Image**

The contour.py file creates a contour image for a given tiff file.

Original tiff image The returned contour plot

**How I did it**

This is a fairly straightforward process. It uses the same two find\_center() and crop() functions as the pie slice. This is merely to center the image, the contour() function can still work even if the image is not centered.

* Contour()  
  This function calls the crop() to square and center the all sky image, note this is not necessary. It then grayscale the image, to allow matplot lab to use its internal contour functions. Then with just 2 lines matplotlib is able to create the contour image. I am not entirely sure how this works or where to get how bright each contour line is.

**Notes**:  
This uses matplotlib’s internal contour function, so I am not certain exactly how it works. As a result I currently do not know how to get how bright the image is. All we have is a visual representation of the difference in brightness between different areas of the image.

Eventually, I hope to be able to describe how bright each contour is, perhaps with a key saying this color corresponds to this brightness and will block out this many stars.

I am not sure of the data analysis use of this function, perhaps it is best for simply visual examples.