Computational Photography

Final Project

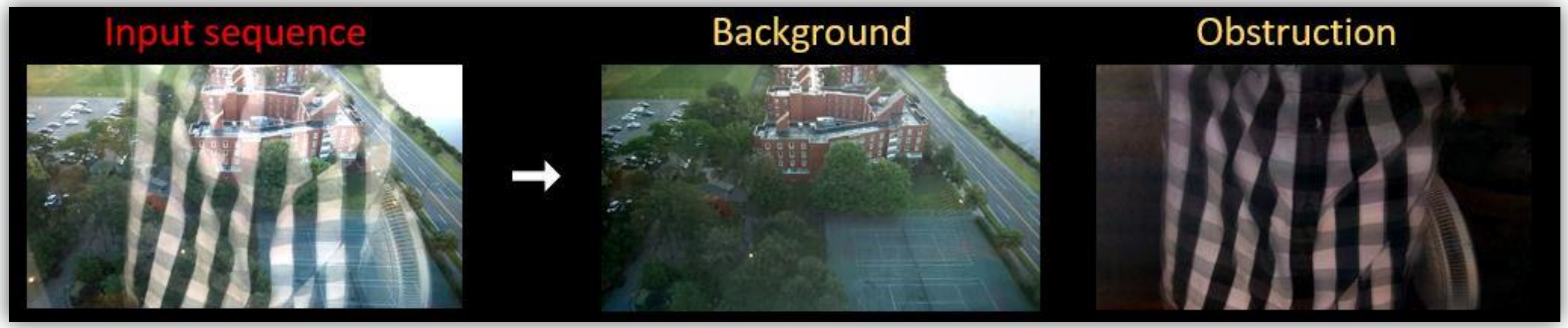
Josh Adams

CS6475 - Fall 2019

Obstruction Free Photography Replication

This project was to have obstructions removed from images. It is achieved by taking a set of images, typically five, while moving in a constant direction.

Many of the links in the report are to either gifs or image sets too large to include in the report.



Computational Approach to Obstruction-Free Photography - Image 1

Goal of Project

Original Project Scope:
The original scope of the project was to take a sequence of images which, contained an obstruction of some kind, and then process those images in a way which allowed for segmentation of both the background (the subject) and the obstruction. The goal was to do this to three sequences of images: a reflection, a complete obstruction (such as a fence), and semi-transparent obstruction such as rain drops

Motivation for Project
What motivated me to do this project was seeing how a sequence of images were combined to produce something different. All the images in the sequence contained an obstruction which produced less than ideal results. Taking the sequence and processing it to iteratively remove the obstruction and produce an image with just the subject remaining. Another aspect which I found to be extremely interesting was the reflection removal and how it would produce a relatively clear image of the reflection. This allowed for capture of an aspect of the image which typically is not considered. It gave a feeling of a secret message being captured in the sequences; this was able to be extracted with extensive processing of the image sequences.

Scope Changes

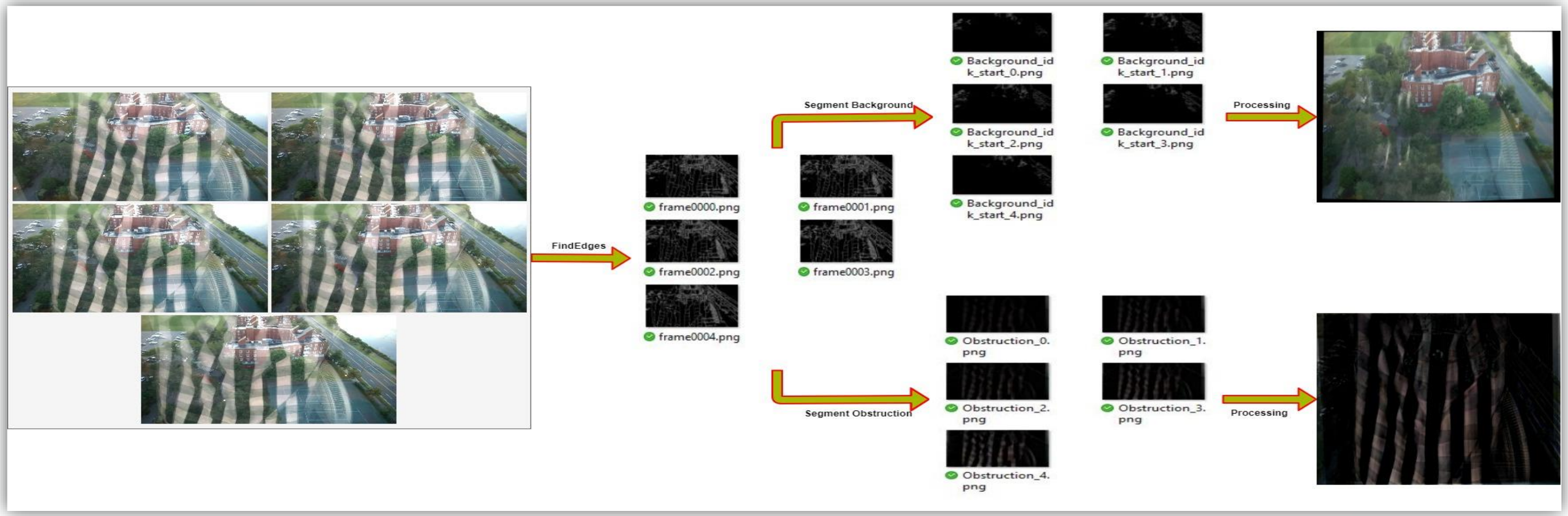
- **Did you run into issues that required you to change project scope from your proposal?**

I ran into many issue and it seemed to be one right after another. The scope changed from implementing three different types of obstruction removal to just reflection. The main reason for this was that many of the aspects of the paper are not explicitly defined. This resulted in an exorbitant amount of time devoted to trying to fill in the missing pieces. Much of this effort was wasted because the missing piece may have been filled, but it could have been filled incorrectly. This happened many times during the process of completing the assignment.

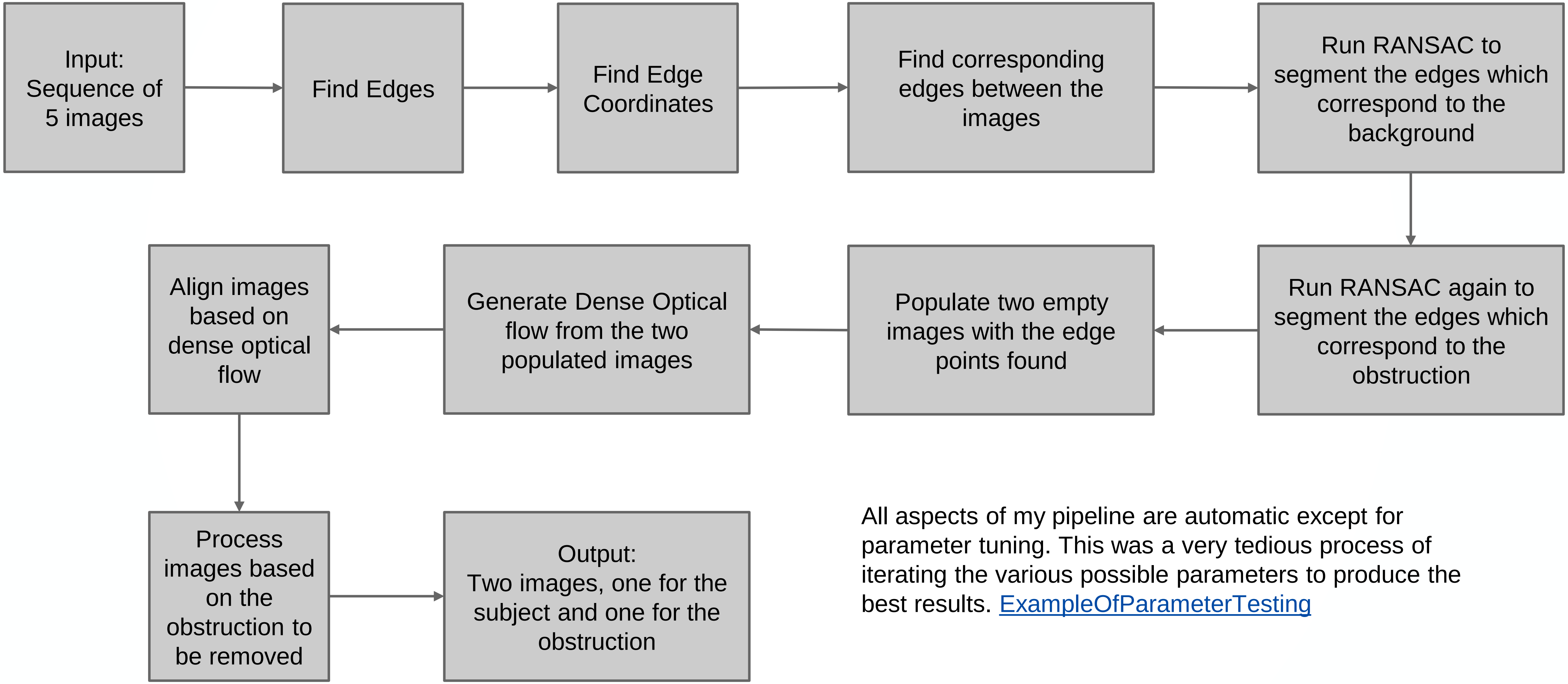
- **Give a detailed explanation of what changed**

The original scope of the project was to implement three different types of image obstruction removal techniques. One for reflections, one for opaque obstructions, and one for semi-transparent obstructions. The scope changed to only implementing the reflection removal. I was also not able to implement the use of dense optical flows to align images. I was able to calculate the dense optical fields but could not find a way to use the resulting flow to produce properly aligned images.

Showcase


FindEdges
frame0000.png
frame0001.png
frame0002.png
frame0003.png
frame0004.png
Segment Background
Segment Obstruction
Background_idk_start_0.png
Background_idk_start_1.png
Background_idk_start_2.png
Background_idk_start_3.png
Background_idk_start_4.png
Obstruction_0.png
Obstruction_1.png
Obstruction_2.png
Obstruction_3.png
Obstruction_4.png
Processing
Processing

Project Pipeline Overview


Input: Sequence of 5 images
Find Edges
Find Edge Coordinates
Find corresponding edges between the images
Run RANSAC to segment the edges which correspond to the background
Run RANSAC again to segment the edges which correspond to the obstruction
Populate two empty images with the edge points found
Generate Dense Optical flow from the two populated images
Align images based on dense optical flow
Process images based on the obstruction to be removed
Output: Two images, one for the subject and one for the obstruction

All aspects of my pipeline are automatic except for parameter tuning. This was a very tedious process of iterating the various possible parameters to produce the best results. ExampleOfParameterTesting

Project Pipeline

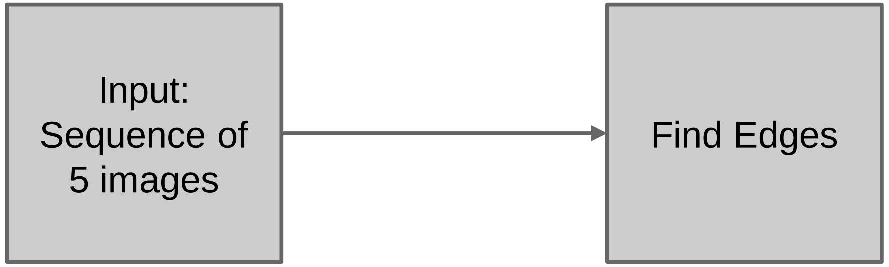
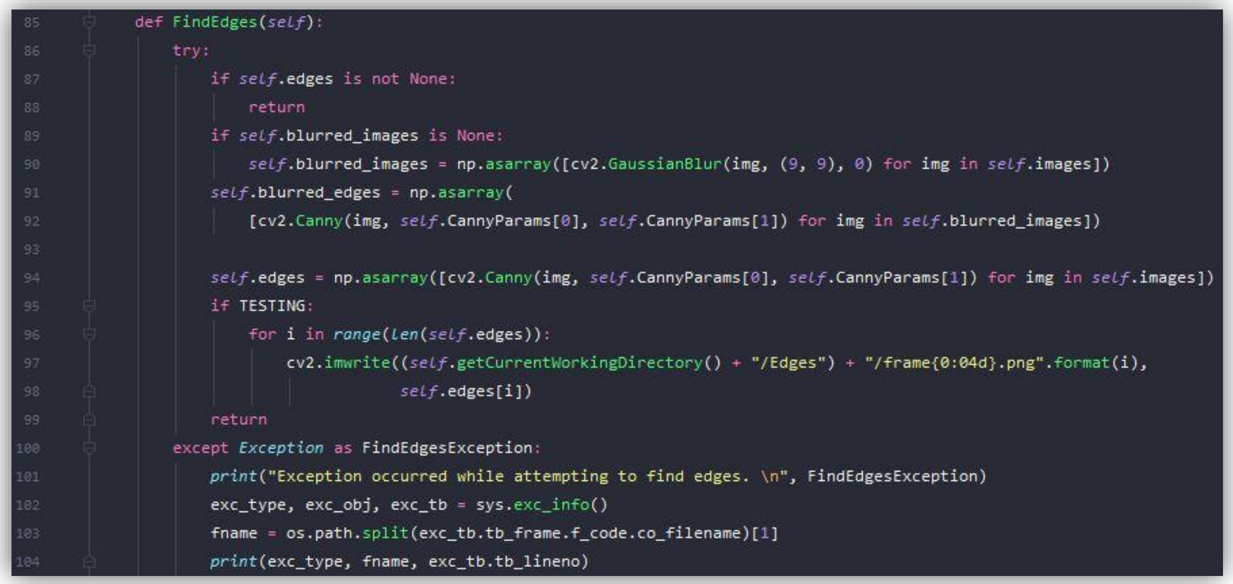
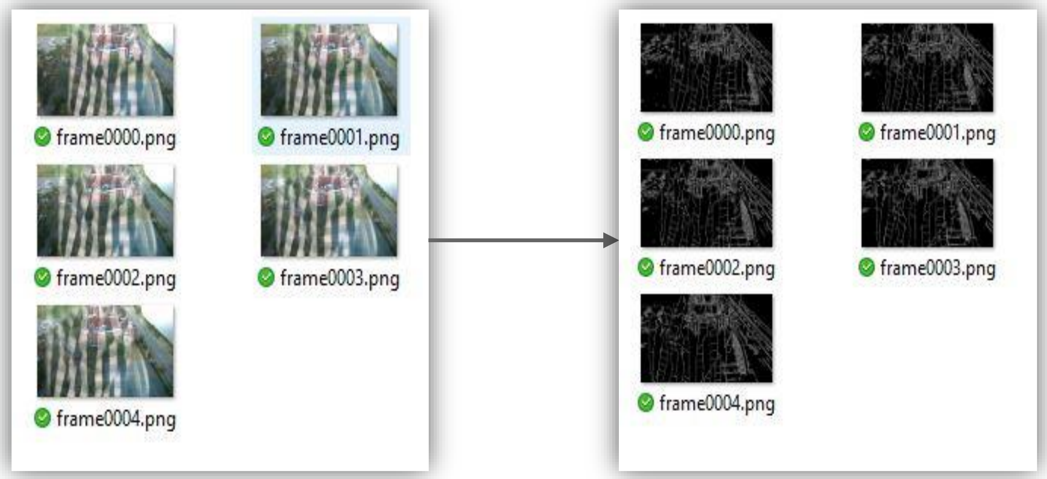


Image Sequence

Edge Sequence

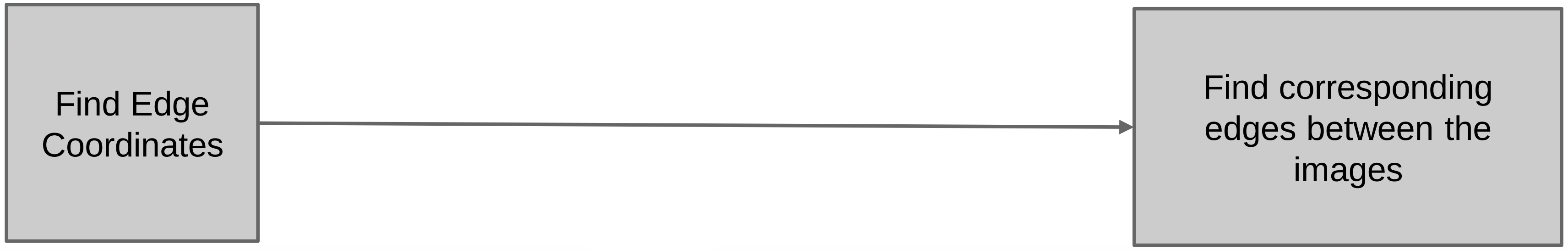


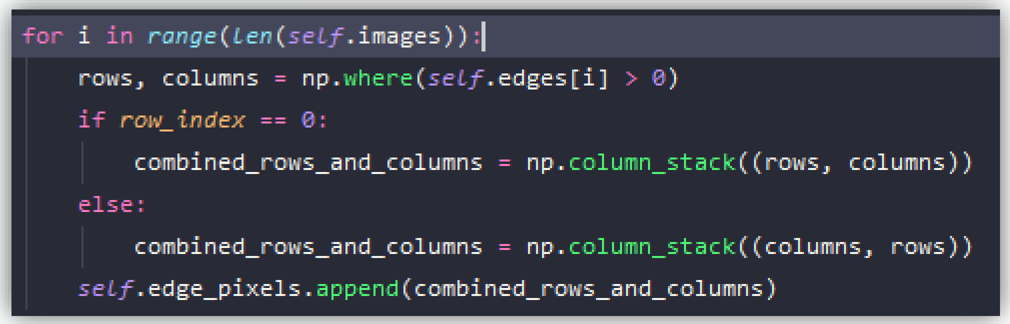
```
def FindEdges(self):
    try:
        if self.edges is not None:
            return
        if self.blurred_images is None:
            self.blurred_images = np.asarray([cv2.GaussianBlur(img, (9, 9), 0) for img in self.images])
        self.blurred_edges = np.asarray(
            [cv2.Canny(img, self.CannyParams[0], self.CannyParams[1]) for img in self.blurred_images])

        self.edges = np.asarray([cv2.Canny(img, self.CannyParams[0], self.CannyParams[1]) for img in self.images])
        if TESTING:
            for i in range(len(self.edges)):
                cv2.imwrite((self.getCurrentWorkingDirectory() + "/Edges") + "/frame{0:04d}.png".format(i),
                                  self.edges[i])
        return
    except Exception as FindEdgesException:
        print("Exception occurred while attempting to find edges. \n", FindEdgesException)
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
```



Finding the edges of all the images in the image sequence. This was an iterative process where I would change the passed in parameters to CannyEdge detection until the results converged on those used in the research paper. I know that canny edge detection implements a gaussian blur within its execution, but I added a prior blur to test those results. I found that I was able to almost produce the exact same results as those used in the research paper for the edges. While I was able to replicate their results for edges, I found that moving forward those edges would not produce reasonable results and, as such, alternate parameters were found and used.

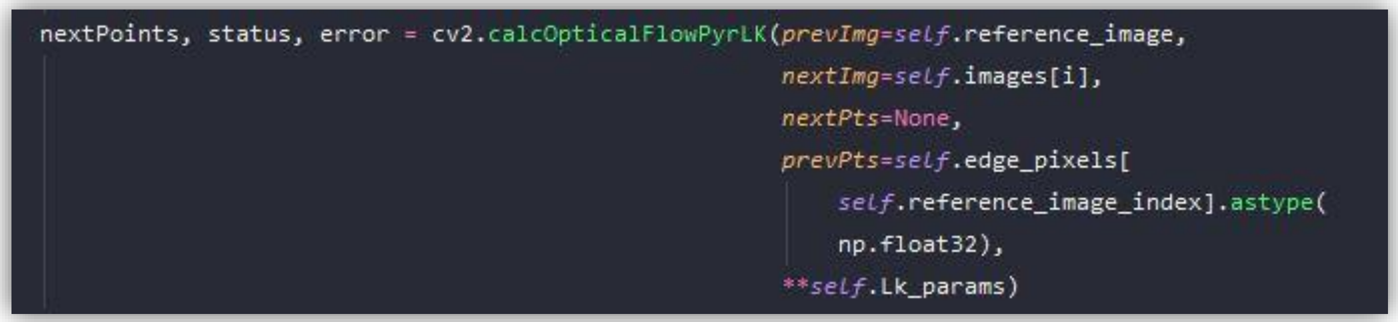
Project Pipeline





```
for i in range(len(self.images)):
    rows, columns = np.where(self.edges[i] > 0)
    if row_index == 0:
        combined_rows_and_columns = np.column_stack((rows, columns))
    else:
        combined_rows_and_columns = np.column_stack((columns, rows))
    self.edge_pixels.append(combined_rows_and_columns)
```

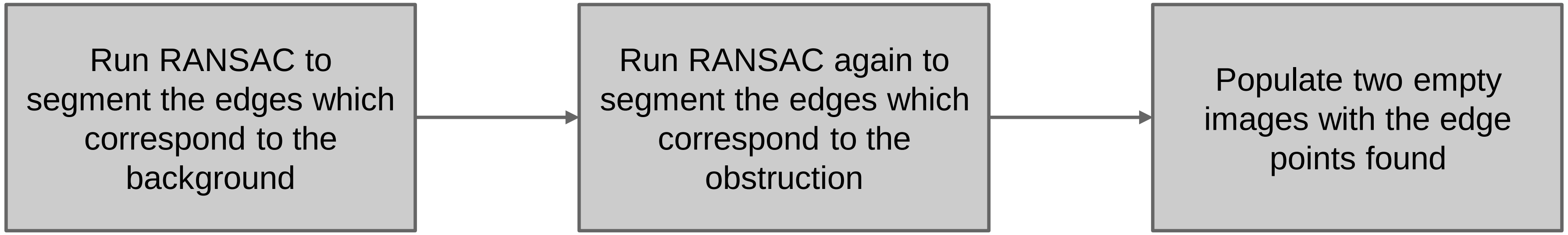
Canny edge detection produced an array where an edge is indicated by having an intensity of 255. To extract all the coordinates of the edges I used Numpy.where to find the coordinates where the value is greater than 0. Row_index is a passed in parameter and the reason for this was that later in the pipeline I discovered that other methods needed the rows/cols to be swapped. It was easy to implement here



```
nextPoints, status, error = cv2.calcOpticalFlowPyrLK(prevImg=self.reference_image,
                                                     nextImg=self.images[i],
                                                     nextPts=None,
                                                     prevPts=self.edge_pixels[
                                                         self.reference_image_index].astype(
                                                         np.float32),
                                                     **self.Lk_params)
```

To get the corresponding edge points between the images, we passed in two images into the ‘calcOpticalFlowPyrLK’ function. This produced an array of new points which correspond to the estimated location of those points in nextImg. The status array is essentially a Boolean mask of strongly correlated points. I filtered nextPoints and the previous edgePixels with the status array. This left me with only strongly correlated edge pixels between the two images.

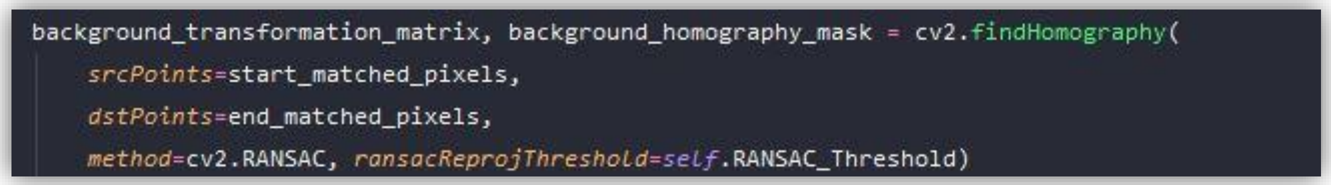
Project Pipeline



Run RANSAC to segment the edges which correspond to the background

Run RANSAC again to segment the edges which correspond to the obstruction

Populate two empty images with the edge points found

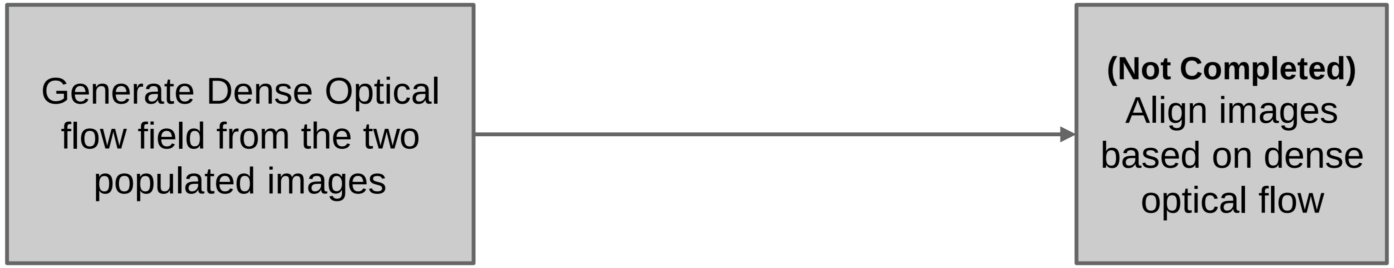


```
background_transformation_matrix, background_homography_mask = cv2.findHomography(
    srcPoints=start_matched_pixels,
    dstPoints=end_matched_pixels,
    method=cv2.RANSAC, ransacReprojThreshold=self.RANSAC_Threshold)
```

At this point in my pipeline I am trying to segment out the background and the obstruction. So we first pass in our first set of correlated edge pixels into findHomography. This produced a transformation matrix and a mask. The mask is then used to filter the passed in points into two groups. One that consists of the background pixels and one that contains the remaining pixels.

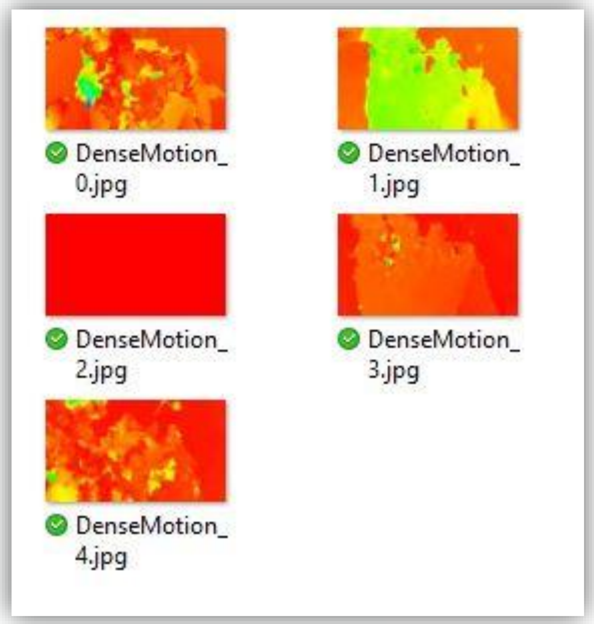
I then ran findHomography once more to try and segment out the obstruction from the remaining pixels. This, again, produced a mask which can be used to pull out those pixels which 'should' correspond to the obstruction.

Project Pipeline



Generate Dense Optical flow field from the two populated images

(Not Completed) Align images based on dense optical flow



The dense optical flow was calculated from the sparse motion flows which were previously calculated. The dense optical flow is pixel wise motion for the image, which is why ‘DenseMotion_2.jpg’ is completely red as it was the reference image.

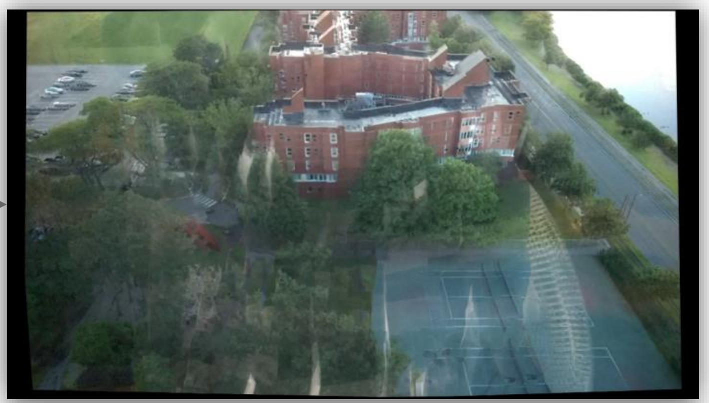
I was not able to align my images using the dense optical flows because I was not able to figure out how. I used sparse motion and the transformation matrix to warpPrerspective to align my images. This was not ideal, but I believe it produced acceptable results. Dense Optical Flow

Project Pipeline

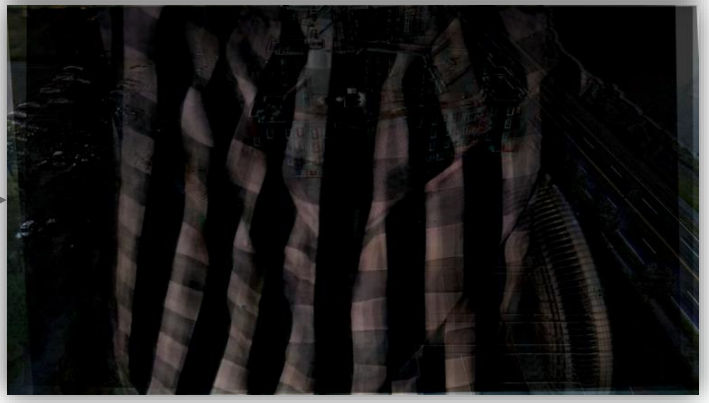
Process images based on the obstruction to be removed → Output: Two images, one for the subject and one for the obstruction

For each iteration through the images, I stored each generated background array and a weighted average of the generated obstruction.

To get the background (the subject), I took the minimum intensity for each pixel in the array of backgrounds

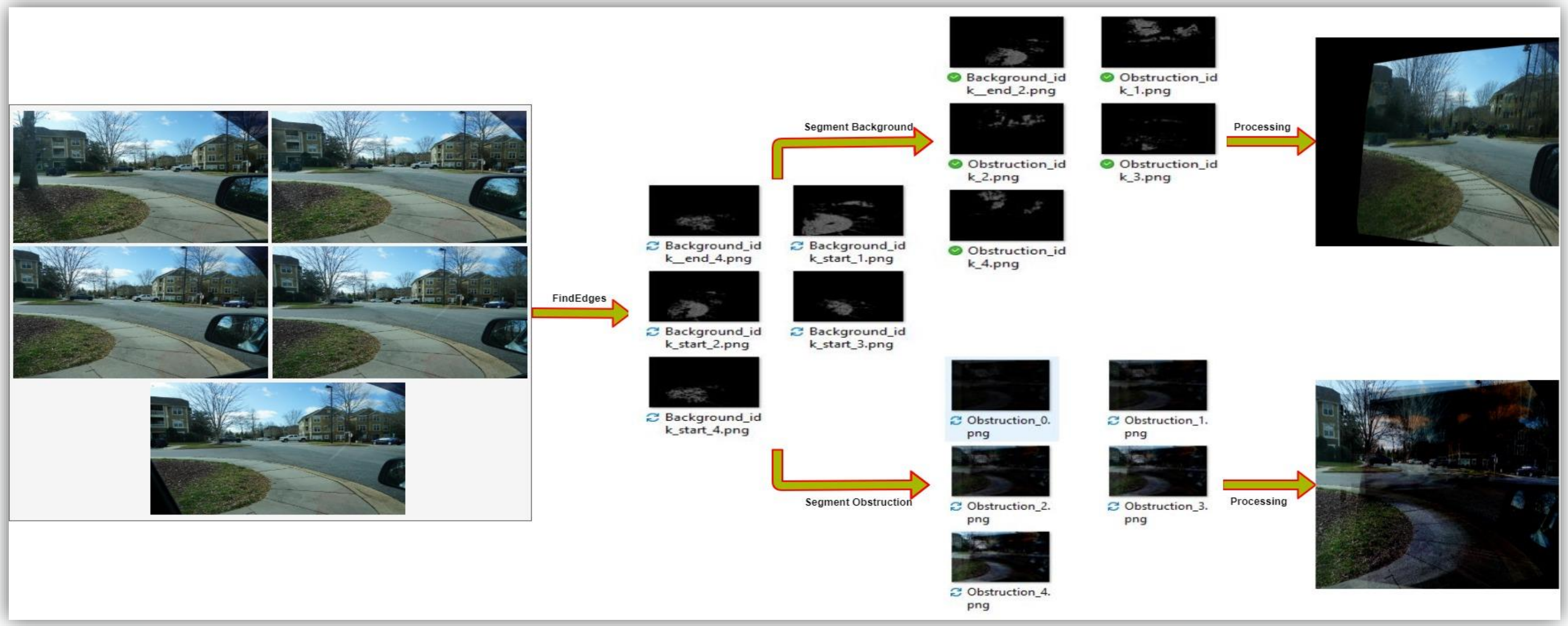


To get the obstruction I return the weighted average of the obstructions.



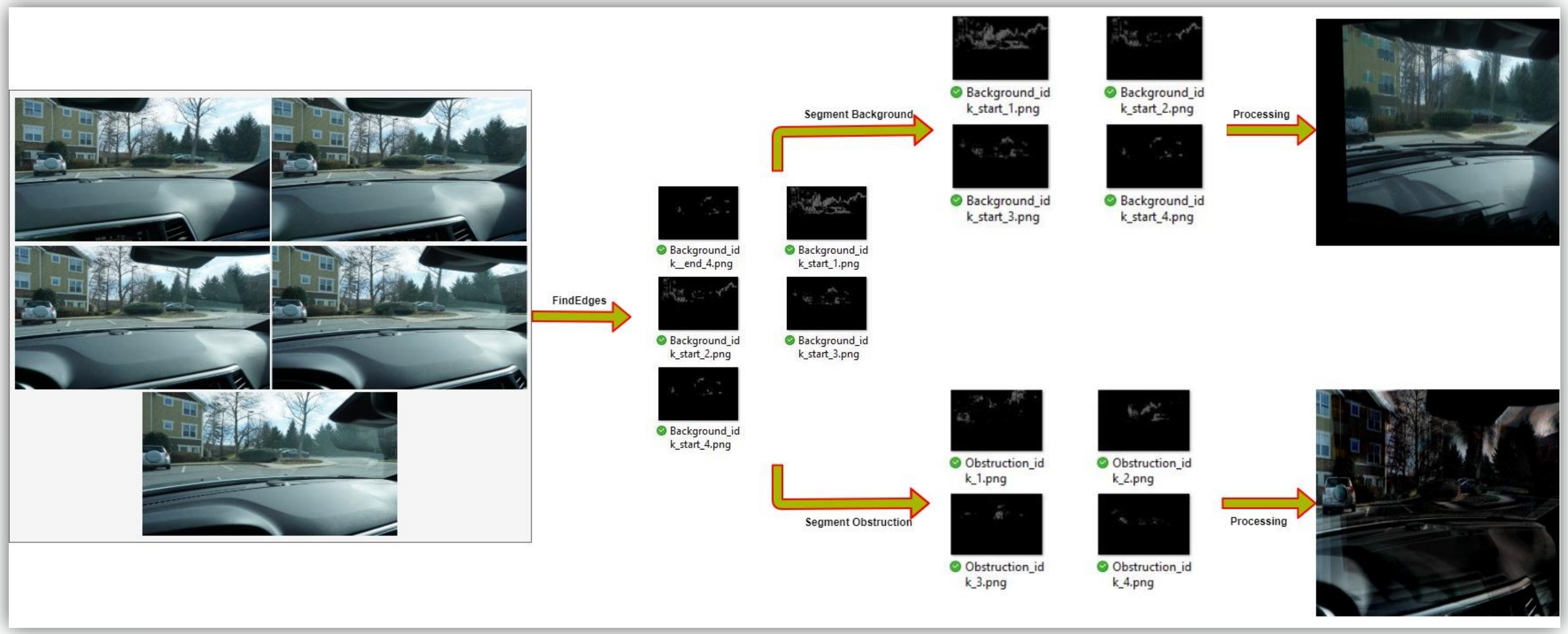
Demonstration: Result Set 1

- This image sequence I captured inside of my vehicle. I started from the far right and moved my camera toward the left while taking pictures of the parking lot.



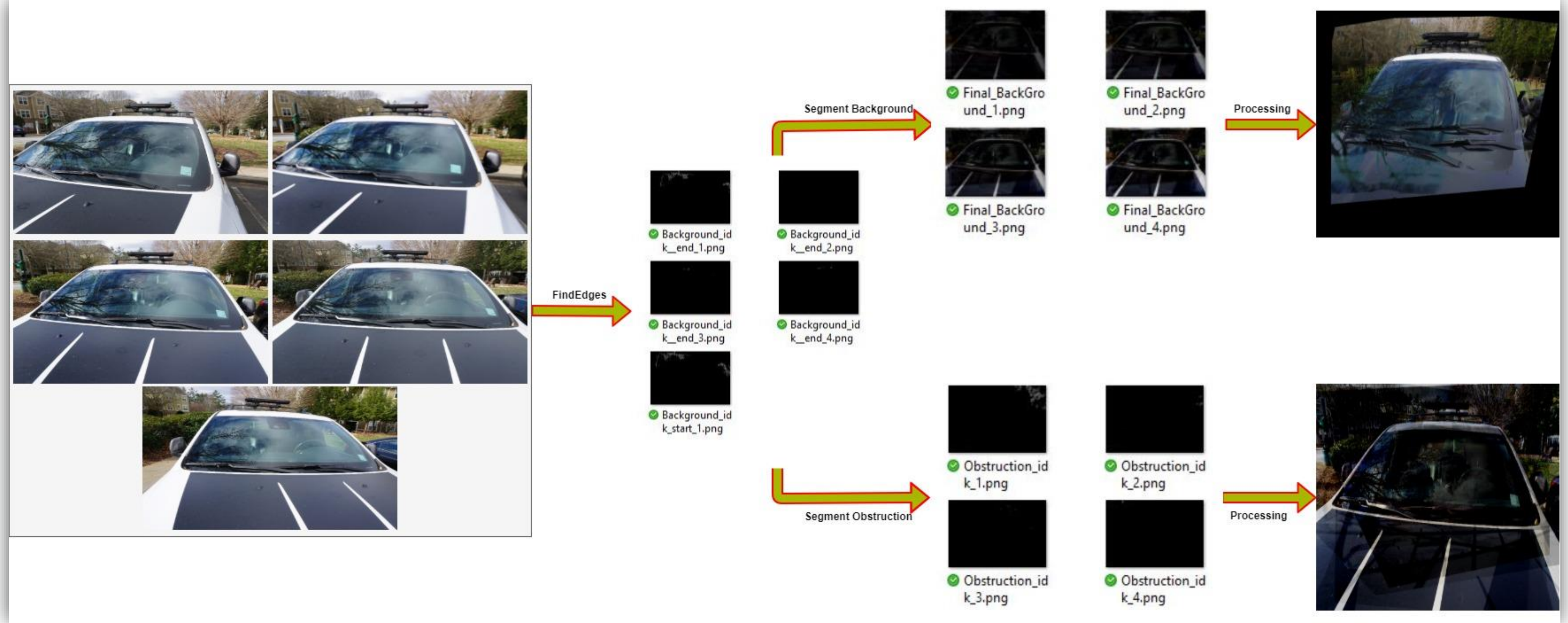
Demonstration: Result Set 2

- This was a second attempt at the reflection from within my vehicle. I the vehicle so it would face the sun.


FindEdges
Background_id k__end_4.png
Background_id k_start_2.png
Background_id k_start_4.png
Background_id k_start_1.png
Background_id k_start_3.png
Segment Background
Background_id k_start_1.png
Background_id k_start_2.png
Background_id k_start_3.png
Background_id k_start_4.png
Processing
Segment Obstruction
Obstruction_id k_1.png
Obstruction_id k_2.png
Obstruction_id k_3.png
Obstruction_id k_4.png
Processing

Demonstration: Result Set 3

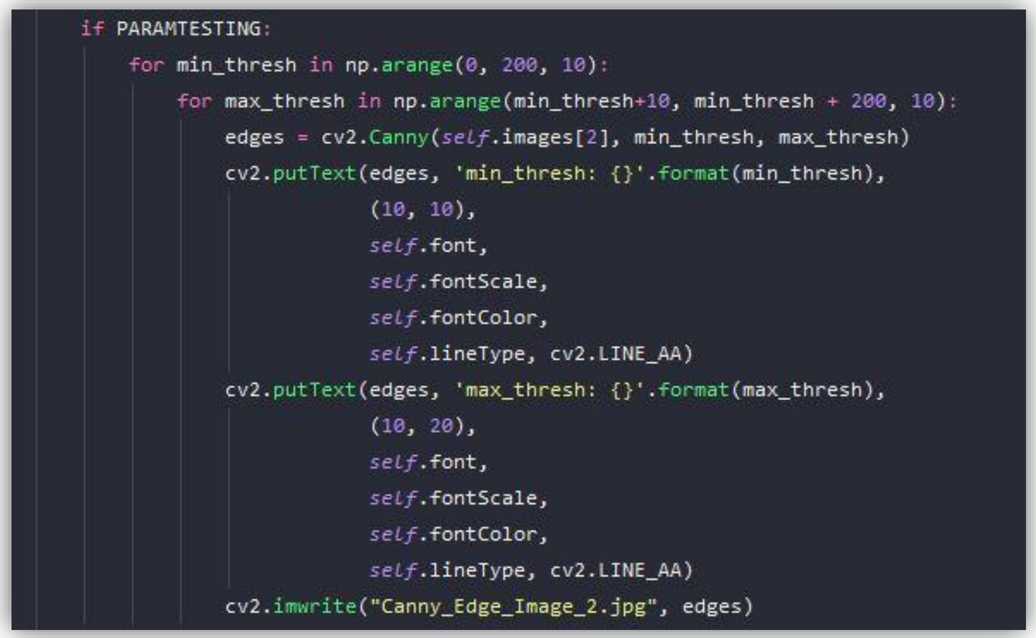
- This was the outside of my vehicle. The reflection was very pronounced in this example.



Project Development – FindEdges

I started off by finding the edges for all the images in my image sequence. This is straight-forward when using canny edge detection. An issue I came across during the usage of canny edge detection was finding valid parameters to pass to the function to produce results which resembled those of the research paper.

The first problem I encountered was determining the approximate parameters used with canny edge detection. To do this I created a loop to iterate over the various thresholds. I then compared my resulting edges to the edges found in the research paper.



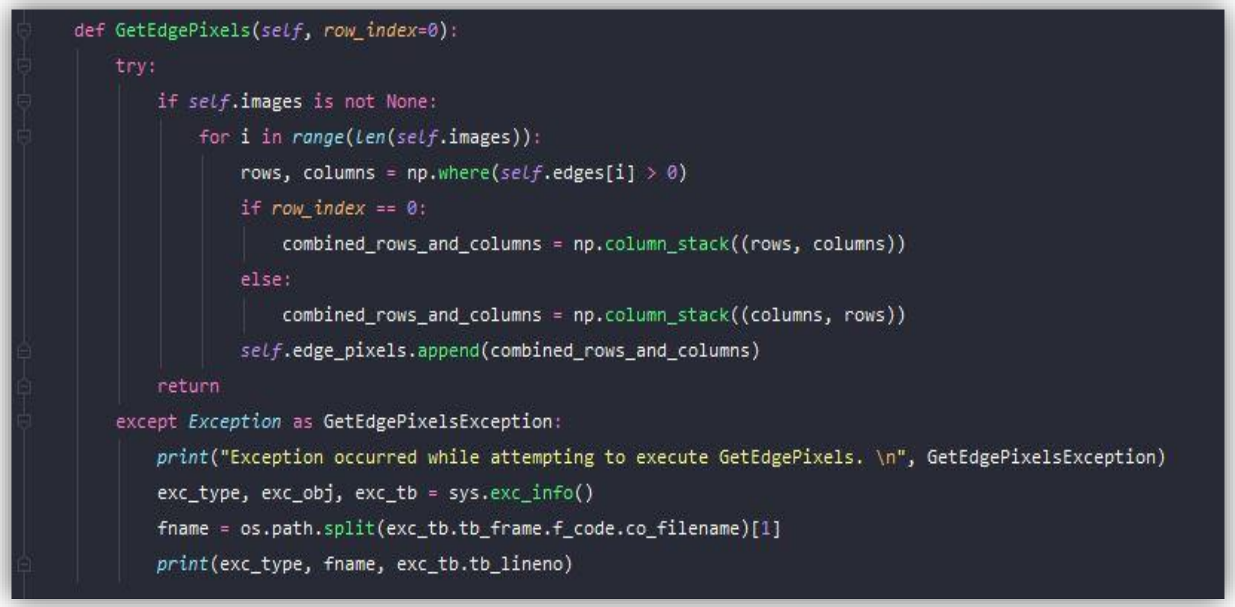
```
if PARAMTESTING:
    for min_thresh in np.arange(0, 200, 10):
        for max_thresh in np.arange(min_thresh+10, min_thresh + 200, 10):
            edges = cv2.Canny(self.images[2], min_thresh, max_thresh)
            cv2.putText(edges, 'min_thresh: {}'.format(min_thresh),
                        (10, 10),
                        self.font,
                        self.fontScale,
                        self.fontColor,
                        self.lineType, cv2.LINE_AA)
            cv2.putText(edges, 'max_thresh: {}'.format(max_thresh),
                        (10, 20),
                        self.font,
                        self.fontScale,
                        self.fontColor,
                        self.lineType, cv2.LINE_AA)
            cv2.imwrite("Canny_Edge_Image_2.jpg", edges)
```

Project Development - GetEdgePixels

The next function used is GetEdgePixels. I filtered out the edges for each image to create a 2D array of [X,Y] or [Y,X] coordinates. The reason I specified the order can be changed was due to coming across an issue where I consistently produced failing results. How this was handled is discussed in a later slide.

The next issue I encountered was obtaining the pixels which correspond to the edges found in the images. I later needed to add a parameter to establish the index at which rows should be located. This resolved a major issue I was having where I consistently produced inaccurate and many times failing results.

GetEdgePixels filters the edges of the image to create rows and columns. Then I used column_stack to manipulate the rows and columns into the expected format.



```
def GetEdgePixels(self, row_index=0):
    try:
        if self.images is not None:
            for i in range(len(self.images)):
                rows, columns = np.where(self.edges[i] > 0)
                if row_index == 0:
                    combined_rows_and_columns = np.column_stack((rows, columns))
                else:
                    combined_rows_and_columns = np.column_stack((columns, rows))
                self.edge_pixels.append(combined_rows_and_columns)
        return
    except Exception as GetEdgePixelsException:
        print("Exception occurred while attempting to execute GetEdgePixels. \n", GetEdgePixelsException)
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
```

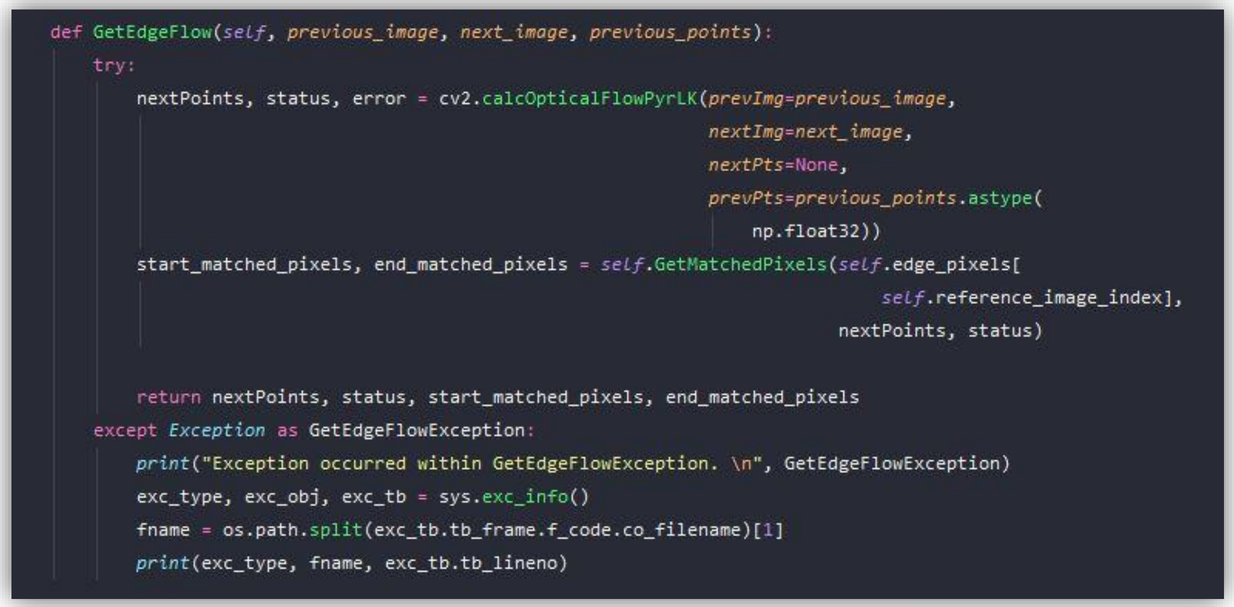
Project Development - GetEdgeFlow

The next function used is GetEdgeFlow. I used OpenCV’s ‘calcOpticalFlowPyrLK’ to find the corresponding edge pixels between the reference image and the image being processed. ‘calcOpticalFlowPyrLK’ works by exploiting the image gradients and eigenvalue analysis. This is similar to Harris Corners and allows us to generate our edge flow by finding corresponding edge points in the image sequence.

From GetEdgeFlow, I then call GetMatchedPixels which uses the mask produced by ‘calcOpticalFlowPyrLK’ to filter the points of the previous image's points and the newly generated points. This will return only the points which are strongly associated with one another in both sets of points.

Project Development - GetEdgeFlow

Within GetEdgeFlow I encountered so many issues, it was difficult to keep track of them all. The most impactful problem that I dealt with involved the parameters passed to ‘calcOpticalFlowPyrLK’. I spent about a week trying to resolve the parameters passed into this function. Ultimately, it was resolved via a Piazza post which said to not include the parameters I was trying to modify and to just use the defaults. This may seem trivial, but I had not considered not finding my own parameters to use. While the results are not perfect, they are much better than what I was getting.



```
def GetEdgeFlow(self, previous_image, next_image, previous_points):
    try:
        nextPoints, status, error = cv2.calcOpticalFlowPyrLK(prevImg=previous_image,
                                                              nextImg=next_image,
                                                              nextPts=None,
                                                              prevPts=previous_points.astype(
                                                                  np.float32))
        start_matched_pixels, end_matched_pixels = self.GetMatchedPixels(self.edge_pixels[
                                                                              self.reference_image_index],
                                                                          nextPoints, status)

        return nextPoints, status, start_matched_pixels, end_matched_pixels
    except Exception as GetEdgeFlowException:
        print("Exception occurred within GetEdgeFlowException. \n", GetEdgeFlowException)
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
```

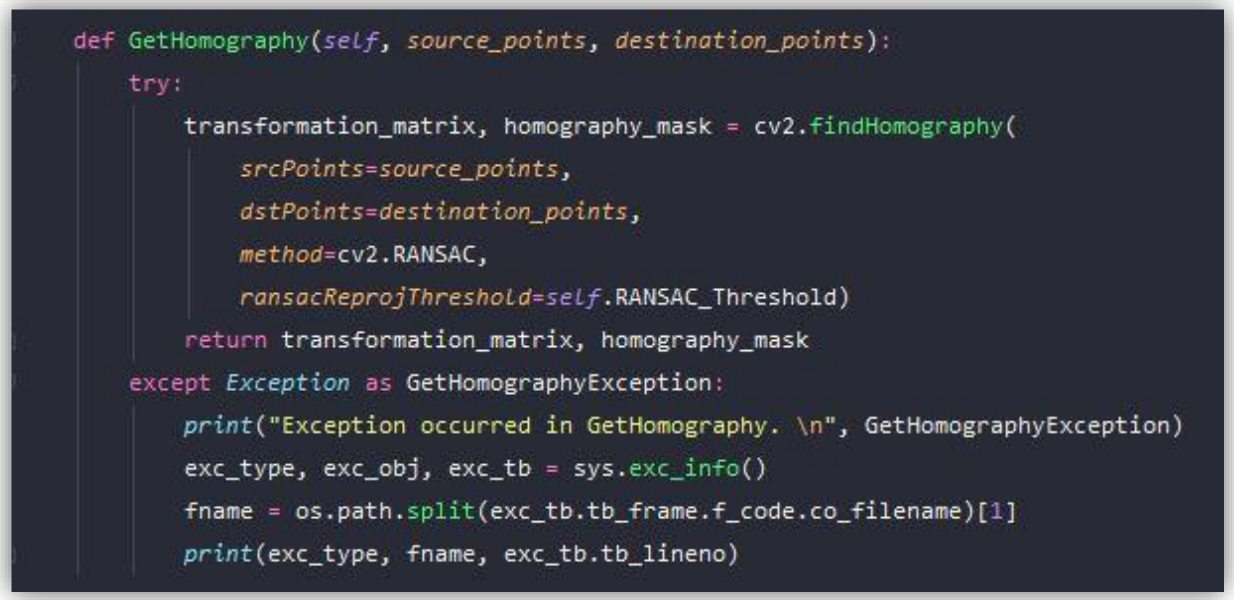
Project Development - GetHomography

Use several slides for a **DETAILED DISCUSSION** of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

You MUST include the following (though not limited to):

- Descriptions of problems, and how you handled them

I moved cv2.findHomography into its own function to help clean up the code. The main problem I encountered here was trying to determine if it was better to find the transformation matrix, going from the reference image to the next image, or to find the transformation matrix going from the next image to the reference image. I resolved this by making the function more generic and explicitly defined the way the function is called.



```
def GetHomography(self, source_points, destination_points):
    try:
        transformation_matrix, homography_mask = cv2.findHomography(
            srcPoints=source_points,
            dstPoints=destination_points,
            method=cv2.RANSAC,
            ransacReprojThreshold=self.RANSAC_Threshold)
        return transformation_matrix, homography_mask
    except Exception as GetHomographyException:
        print("Exception occurred in GetHomography. \n", GetHomographyException)
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
```

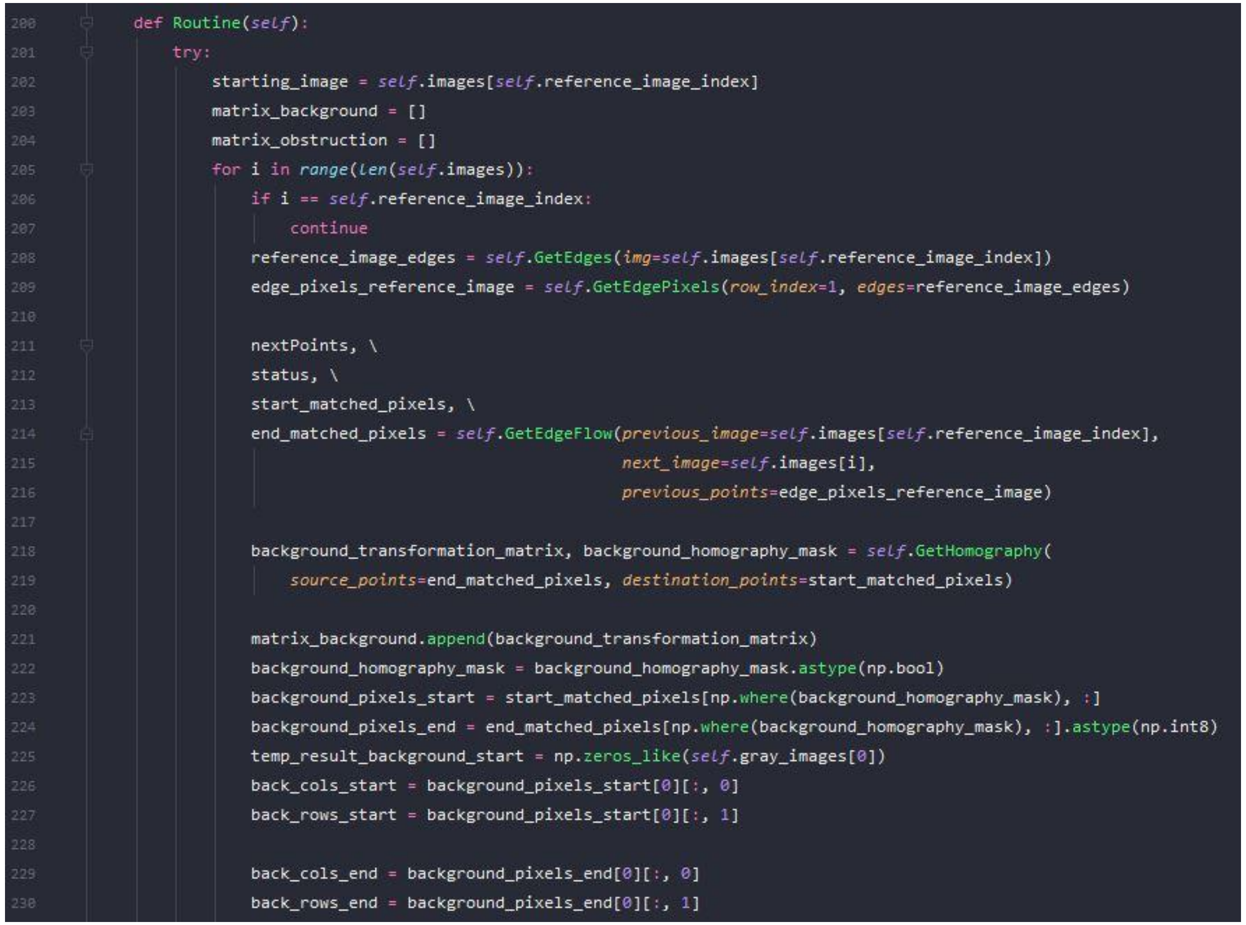
Project Development – General Code

Use several slides for a **DETAILED DISCUSSION** of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

You MUST include the following (though not limited to):

- Descriptions of problems, and how you handled them

I created a function that I called Routine. This is called automatically when you generate an instance of my ‘ImageObstructionClean’ class. This function will iterate over all images in the sequence and execute many different functions. The first would be to get the edges of the reference image. The reference image will be used later for all motion to be compared. Using ‘GetEdgePixels,’ I extract the pixel locations of all the edges that were found corresponding to the reference image. Next is to execute ‘GetEdgeFlow’ to generate a sparse optical flow for the first instance of edge pixels.



```python
def Routine(self):
    try:
        starting_image = self.images[self.reference_image_index]
        matrix_background = []
        matrix_obstruction = []
        for i in range(len(self.images)):
            if i == self.reference_image_index:
                continue
            reference_image_edges = self.GetEdges(img=self.images[self.reference_image_index])
            edge_pixels_reference_image = self.GetEdgePixels(row_index=1, edges=reference_image_edges)

            nextPoints, \
            status, \
            start_matched_pixels, \
            end_matched_pixels = self.GetEdgeFlow(previous_image=self.images[self.reference_image_index],
                                                  next_image=self.images[i],
                                                  previous_points=edge_pixels_reference_image)

            background_transformation_matrix, background_homography_mask = self.GetHomography(
                source_points=end_matched_pixels, destination_points=start_matched_pixels)

            matrix_background.append(background_transformation_matrix)
            background_homography_mask = background_homography_mask.astype(np.bool)
            background_pixels_start = start_matched_pixels[np.where(background_homography_mask), :]
            background_pixels_end = end_matched_pixels[np.where(background_homography_mask), :].astype(np.int8)
            temp_result_background_start = np.zeros_like(self.gray_images[0])
            back_cols_start = background_pixels_start[0][:, 0]
            back_rows_start = background_pixels_start[0][:, 1]

            back_cols_end = background_pixels_end[0][:, 0]
            back_rows_end = background_pixels_end[0][:, 1]
```

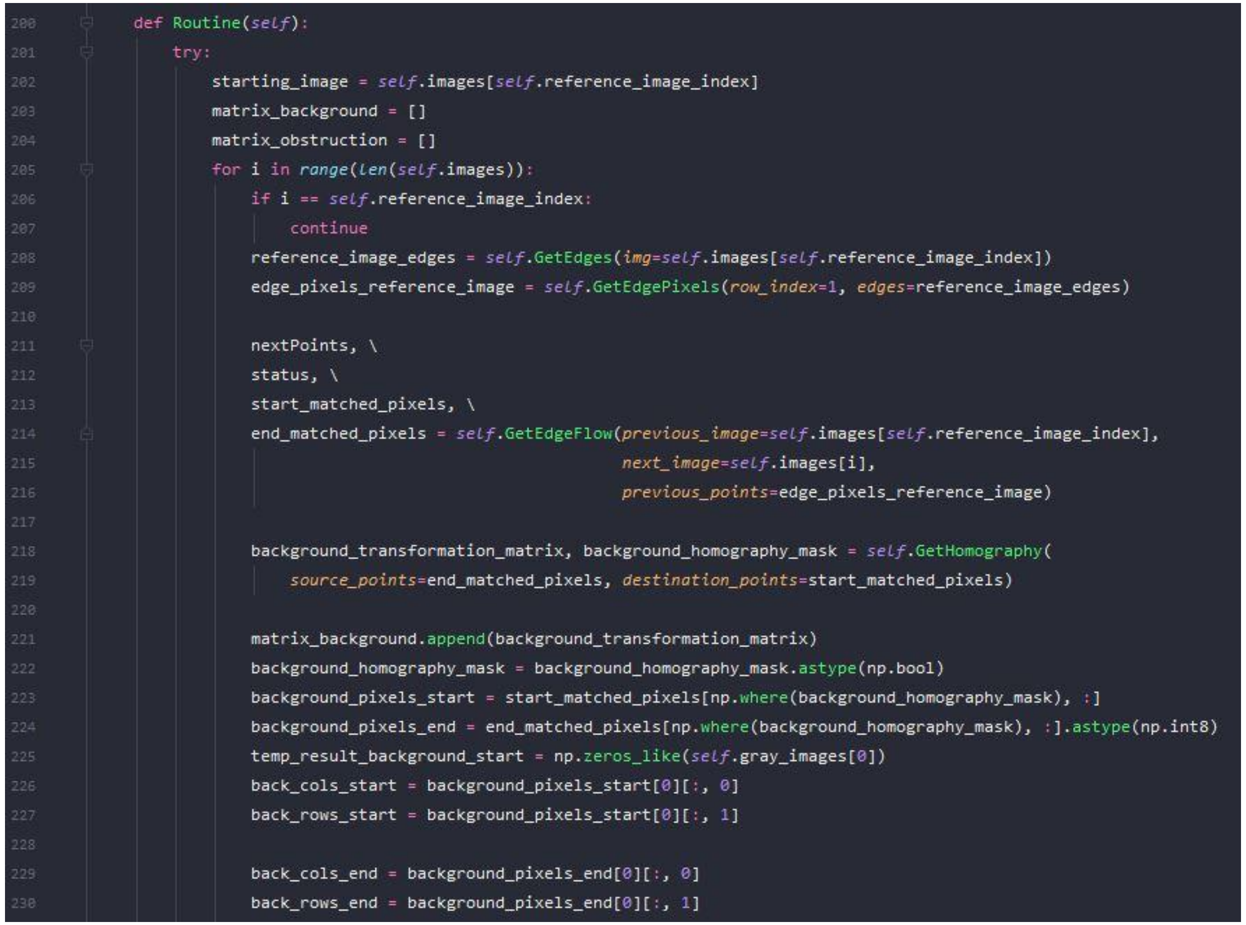
Project Development – General Code cont.

Use several slides for a **DETAILED DISCUSSION** of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

You MUST include the following (though not limited to):

- Descriptions of problems, and how you handled them

‘GetEdgeFlow’ will return an array that contains the calculated ‘nextPoints’, a status mask indicating strong correlating pixels between both starting and ending matched pixels. The starting and ending matched pixels were produced by filtering the arrays nextPoints and previous_points by the status array.
I use ‘findHomography’ with the method parameter set to RANSAC. This will produce a mask where a value of 1 indicates points which are able to be groups due to similar motion. This allows me to segment out this group of pixels which should correspond to the background of the image.



```python
def Routine(self):
    try:
        starting_image = self.images[self.reference_image_index]
        matrix_background = []
        matrix_obstruction = []
        for i in range(len(self.images)):
            if i == self.reference_image_index:
                continue
            reference_image_edges = self.GetEdges(img=self.images[self.reference_image_index])
            edge_pixels_reference_image = self.GetEdgePixels(row_index=1, edges=reference_image_edges)

            nextPoints, \
            status, \
            start_matched_pixels, \
            end_matched_pixels = self.GetEdgeFlow(previous_image=self.images[self.reference_image_index],
                                                  next_image=self.images[i],
                                                  previous_points=edge_pixels_reference_image)

            background_transformation_matrix, background_homography_mask = self.GetHomography(
                source_points=end_matched_pixels, destination_points=start_matched_pixels)

            matrix_background.append(background_transformation_matrix)
            background_homography_mask = background_homography_mask.astype(np.bool)
            background_pixels_start = start_matched_pixels[np.where(background_homography_mask), :]
            background_pixels_end = end_matched_pixels[np.where(background_homography_mask), :].astype(np.int8)
            temp_result_background_start = np.zeros_like(self.gray_images[0])
            back_cols_start = background_pixels_start[0][:, 0]
            back_rows_start = background_pixels_start[0][:, 1]

            back_cols_end = background_pixels_end[0][:, 0]
            back_rows_end = background_pixels_end[0][:, 1]
```

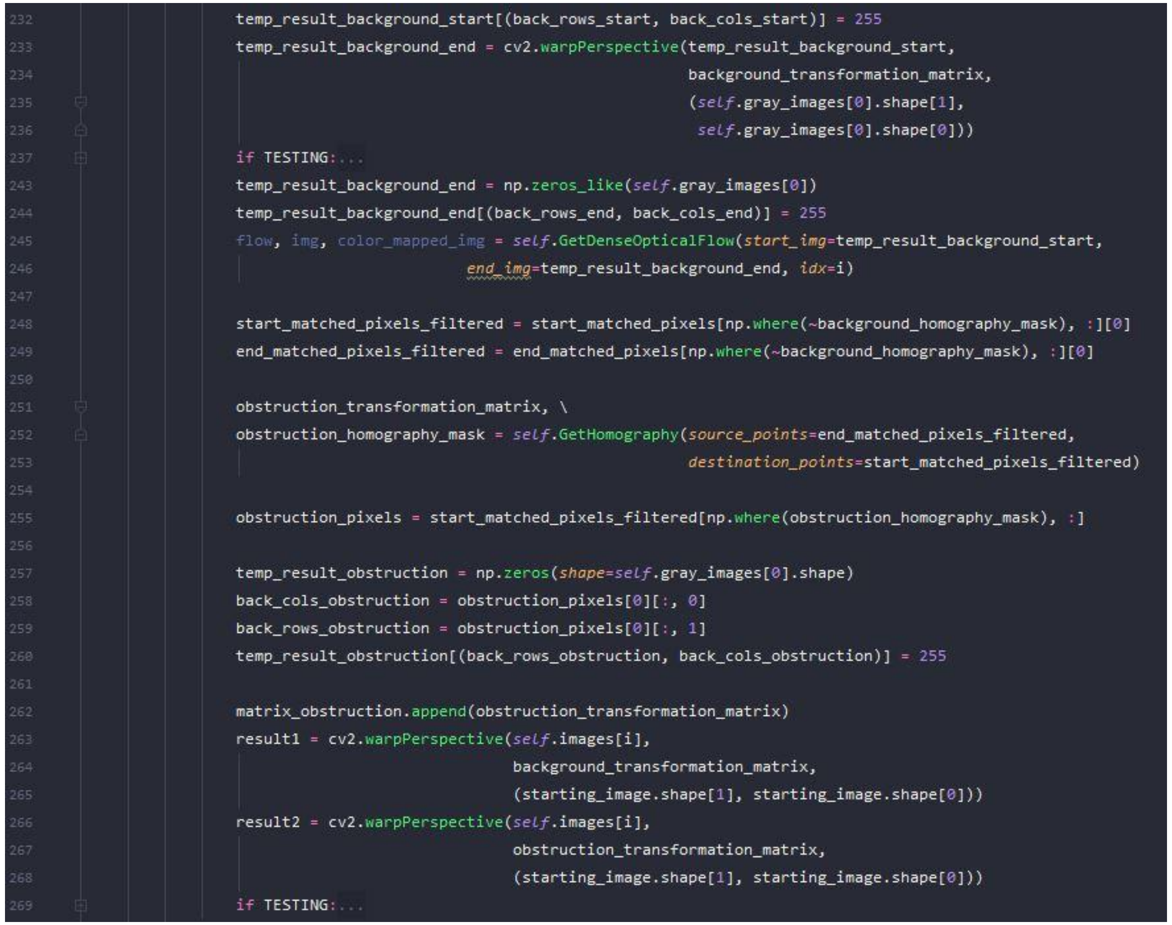
Project Development – General Code cont.

Use several slides for a **DETAILED DISCUSSION** of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

You MUST include the following (though not limited to):

- Descriptions of problems, and how you handled them

I used the points which correspond to the background to populate an empty image with the value of 255 at all point locations.
I then took my starting points and removed all points which are indicated in the homography_mask. I then ran ‘GetHomography’ on those filtered points. Again using RANSAC this will produce a mask where a value of 1 will indicate a grouping of points. I used that mask to extract the points which belonged to the obstruction group. I then generated an empty image and populate it with the value 255 for every point belonging to the obstruction.



```
temp_result_background_start[(back_rows_start, back_cols_start)] = 255
temp_result_background_end = cv2.warpPerspective(temp_result_background_start,
                                                 background_transformation_matrix,
                                                 (self.gray_images[0].shape[1],
                                                  self.gray_images[0].shape[0]))
if TESTING:...
temp_result_background_end = np.zeros_like(self.gray_images[0])
temp_result_background_end[(back_rows_end, back_cols_end)] = 255
flow, img, color_mapped_img = self.GetDenseOpticalFlow(start_img=temp_result_background_start,
                         end_img=temp_result_background_end, idx=i)

start_matched_pixels_filtered = start_matched_pixels[np.where(~background_homography_mask), :][0]
end_matched_pixels_filtered = end_matched_pixels[np.where(~background_homography_mask), :][0]

obstruction_transformation_matrix, \
obstruction_homography_mask = self.GetHomography(source_points=end_matched_pixels_filtered,
                                                 destination_points=start_matched_pixels_filtered)

obstruction_pixels = start_matched_pixels_filtered[np.where(obstruction_homography_mask), :]

temp_result_obstruction = np.zeros(shape=self.gray_images[0].shape)
back_cols_obstruction = obstruction_pixels[0][:, 0]
back_rows_obstruction = obstruction_pixels[0][:, 1]
temp_result_obstruction[(back_rows_obstruction, back_cols_obstruction)] = 255

matrix_obstruction.append(obstruction_transformation_matrix)
result1 = cv2.warpPerspective(self.images[i],
                              background_transformation_matrix,
                              (starting_image.shape[1], starting_image.shape[0]))
result2 = cv2.warpPerspective(self.images[i],
                              obstruction_transformation_matrix,
                              (starting_image.shape[1], starting_image.shape[0]))
if TESTING:...
```

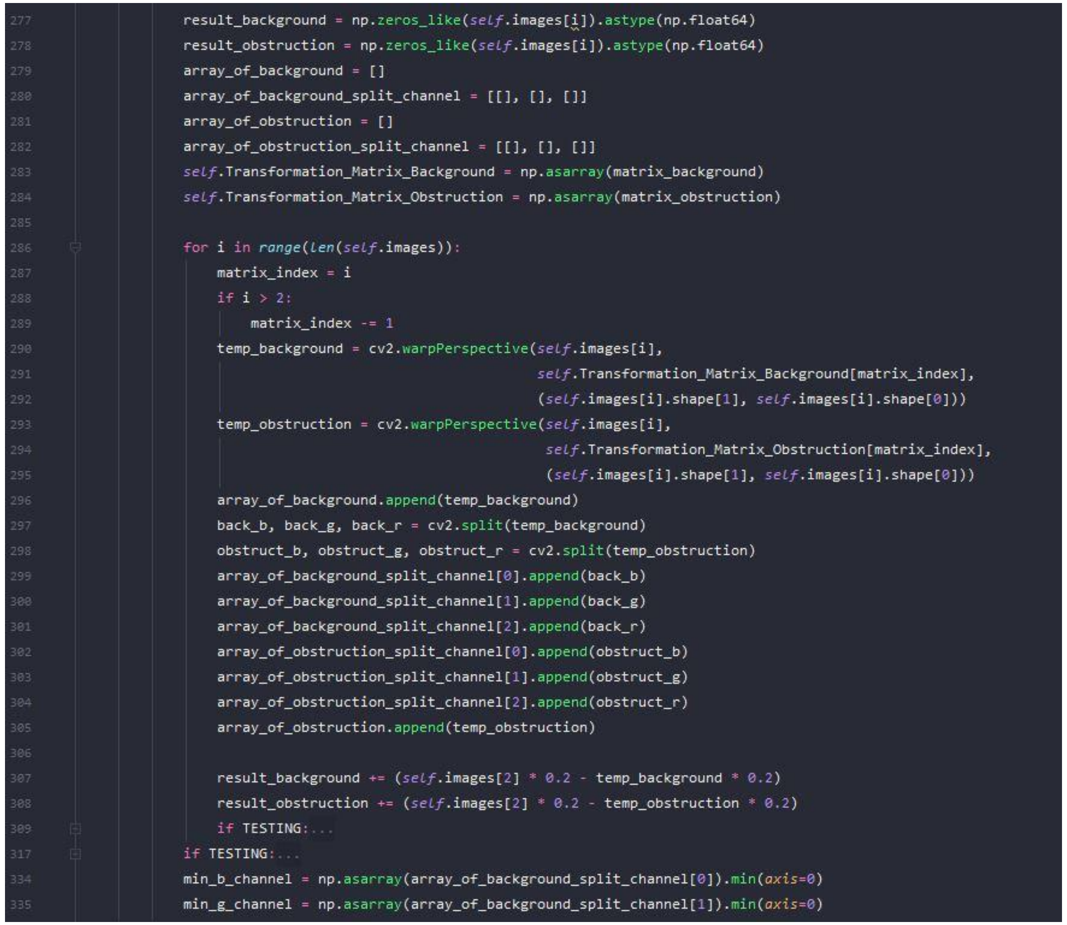
Project Development – General Code cont.

Use several slides for a **DETAILED DISCUSSION** of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

You MUST include the following (though not limited to):

- Descriptions of problems, and how you handled them

Now that I have a background and obstruction for each frame, I processed the frames and warp each frame by either the background or obstruction transformation matrix. I then added a 1/5 reference image with 1/5 transformed_background or 1/5 transformed_obstruction, respectively. If you look at those images at the end of this the obstruction result looks good and does not require any further processing. To extract the background we just subtract the obstruction result from the weighted image. This should result in a clear image with the reflection removed.



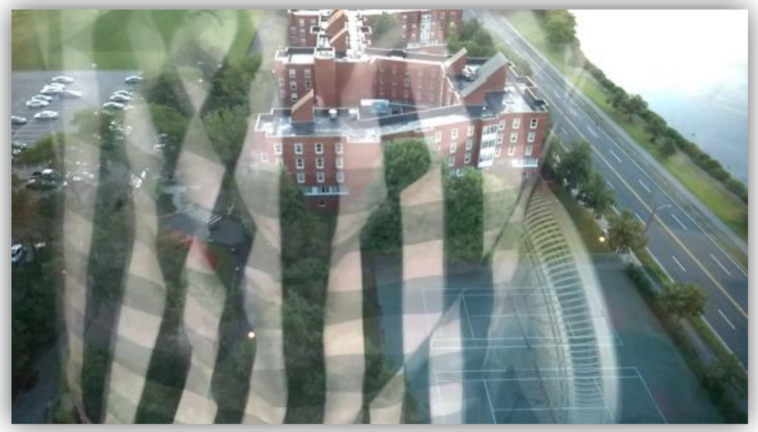
```
result_background = np.zeros_like(self.images[i]).astype(np.float64)
result_obstruction = np.zeros_like(self.images[i]).astype(np.float64)
array_of_background = []
array_of_background_split_channel = [[], [], []]
array_of_obstruction = []
array_of_obstruction_split_channel = [[], [], []]
self.Transformation_Matrix_Background = np.asarray(matrix_background)
self.Transformation_Matrix_Obstruction = np.asarray(matrix_obstruction)

for i in range(len(self.images)):
    matrix_index = i
    if i > 2:
        matrix_index -= 1
    temp_background = cv2.warpPerspective(self.images[i],
                                          self.Transformation_Matrix_Background[matrix_index],
                                          (self.images[i].shape[1], self.images[i].shape[0]))
    temp_obstruction = cv2.warpPerspective(self.images[i],
                                           self.Transformation_Matrix_Obstruction[matrix_index],
                                           (self.images[i].shape[1], self.images[i].shape[0]))
    array_of_background.append(temp_background)
    back_b, back_g, back_r = cv2.split(temp_background)
    obstruct_b, obstruct_g, obstruct_r = cv2.split(temp_obstruction)
    array_of_background_split_channel[0].append(back_b)
    array_of_background_split_channel[1].append(back_g)
    array_of_background_split_channel[2].append(back_r)
    array_of_obstruction_split_channel[0].append(obstruct_b)
    array_of_obstruction_split_channel[1].append(obstruct_g)
    array_of_obstruction_split_channel[2].append(obstruct_r)
    array_of_obstruction.append(temp_obstruction)

    result_background += (self.images[2] * 0.2 - temp_background * 0.2)
    result_obstruction += (self.images[2] * 0.2 - temp_obstruction * 0.2)
    if TESTING:...
if TESTING:...
min_b_channel = np.asarray(array_of_background_split_channel[0]).min(axis=0)
min_g_channel = np.asarray(array_of_background_split_channel[1]).min(axis=0)
```

Project Development

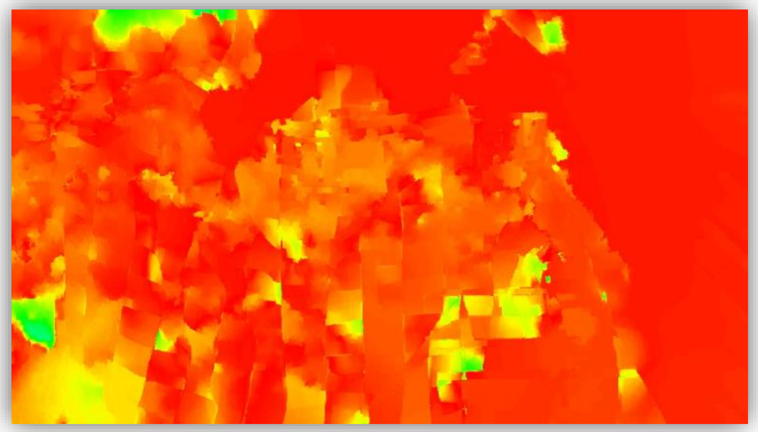
You MUST include the following (though not limited to):

- Good interim results

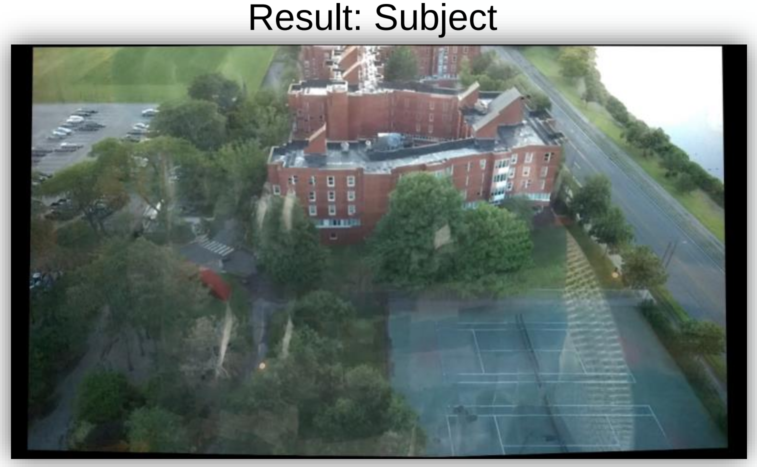
Reference Image



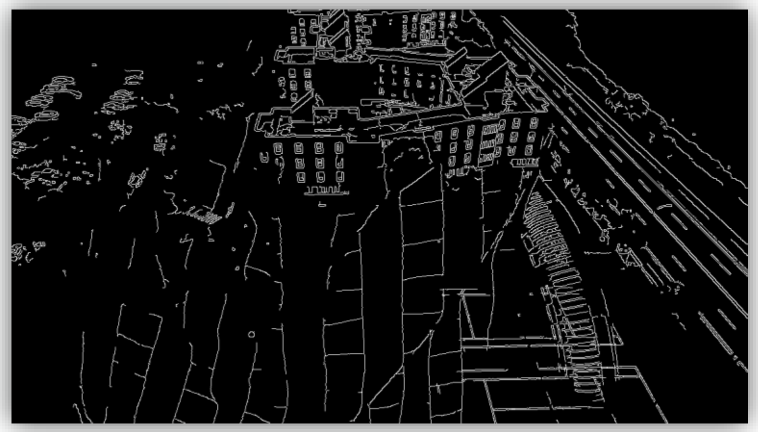
Dense Optical Flow



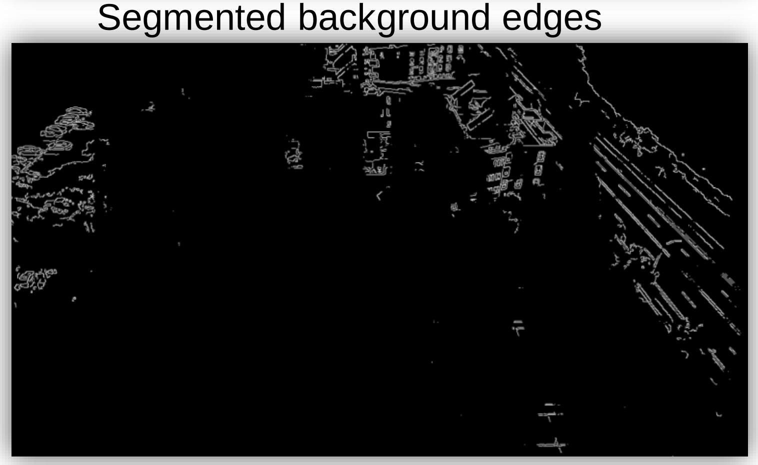
Result: Subject



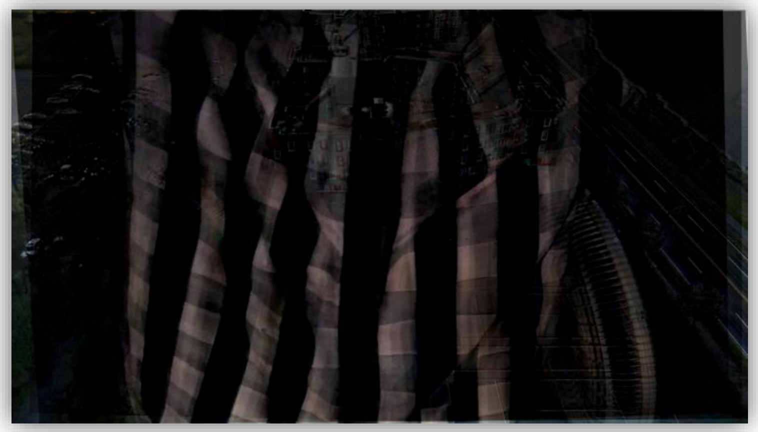
Edges



Segmented background edges



Result: Obstruction

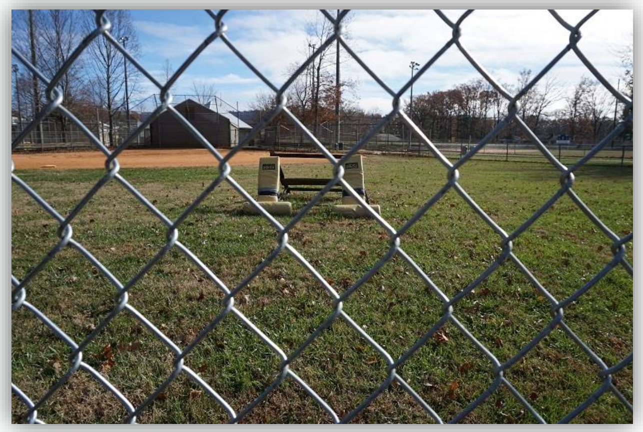


Project Development

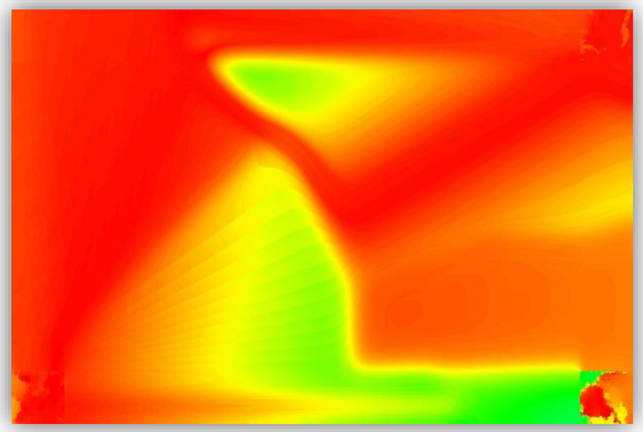
You MUST include the following (though not limited to):

- Failed interim results

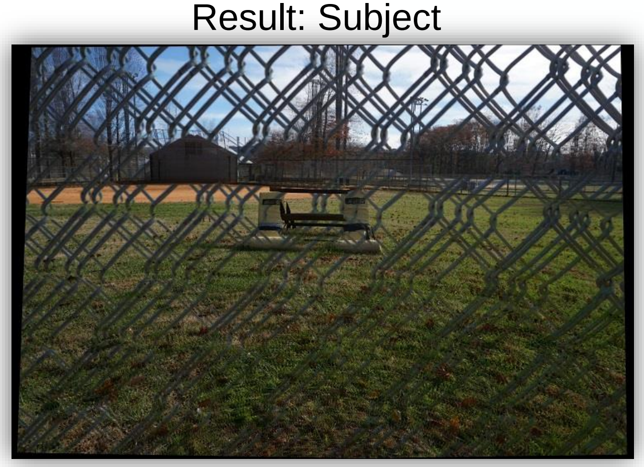
Reference Image



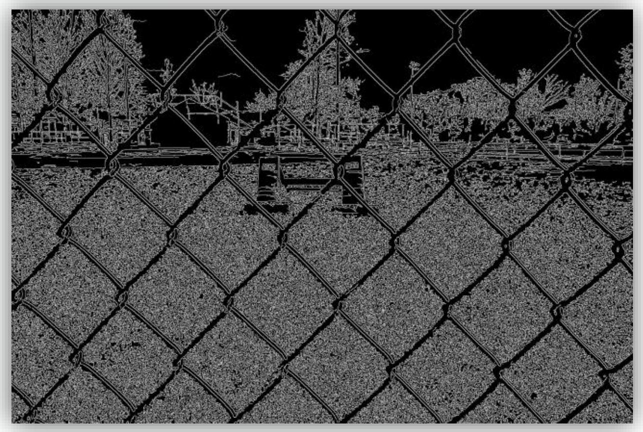
Dense Optical Flow



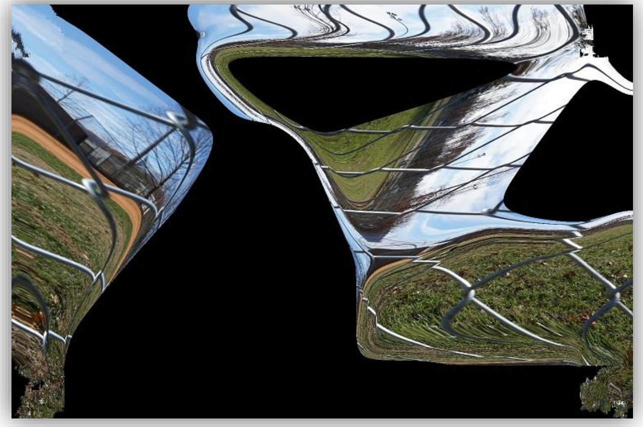
Result: Subject



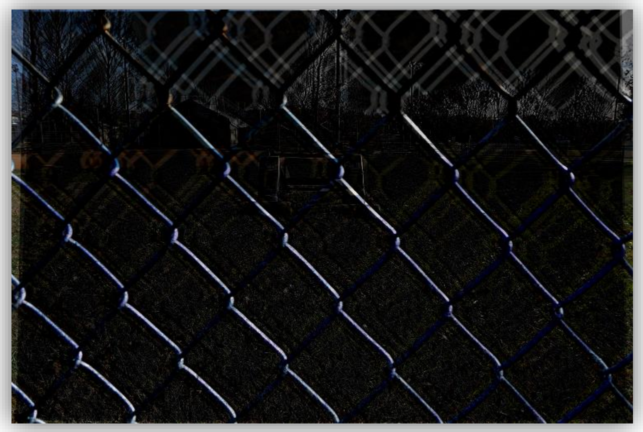
Edges



Segmented background edges



Result: Obstruction

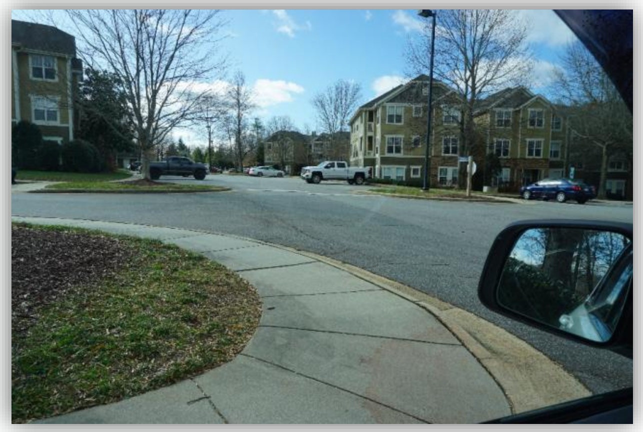


Project Development : Image Sequence 1

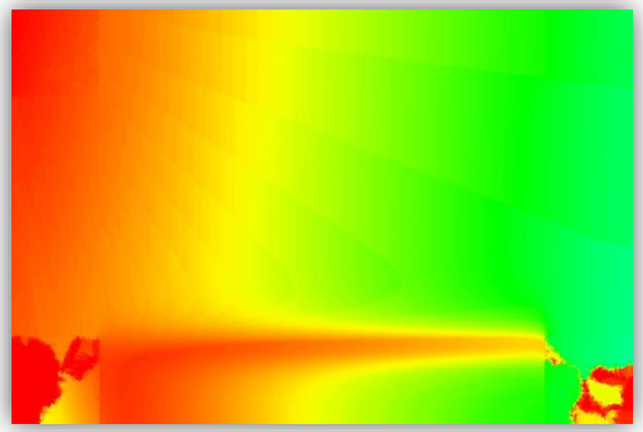
You MUST include the following (though not limited to):

- Good interim results

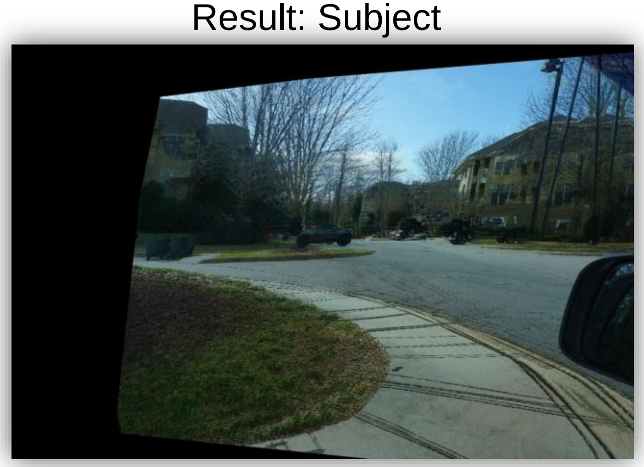
Reference Image



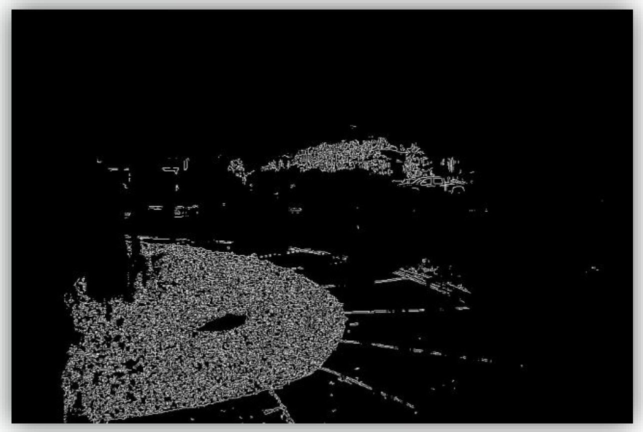
Dense Optical Flow



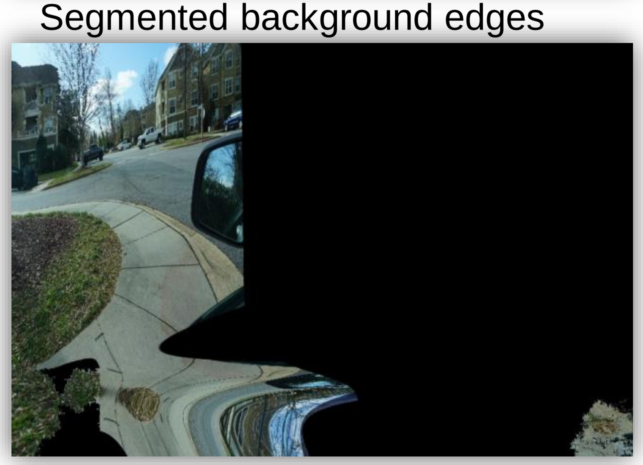
Result: Subject



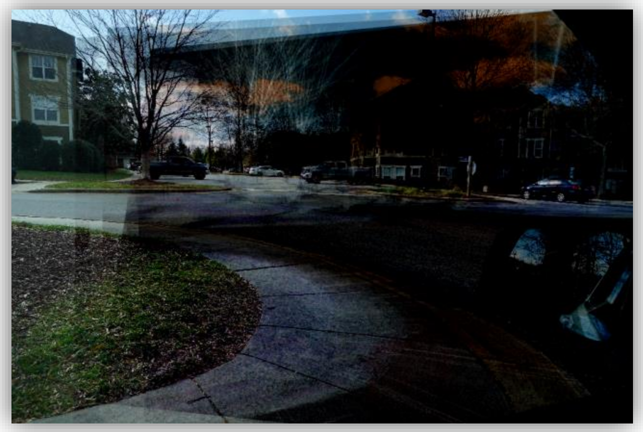
Edges



Segmented background edges



Result: Obstruction

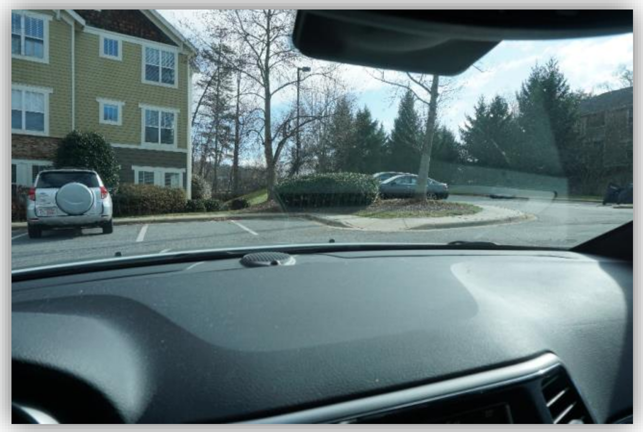


Project Development : Image Sequence 2

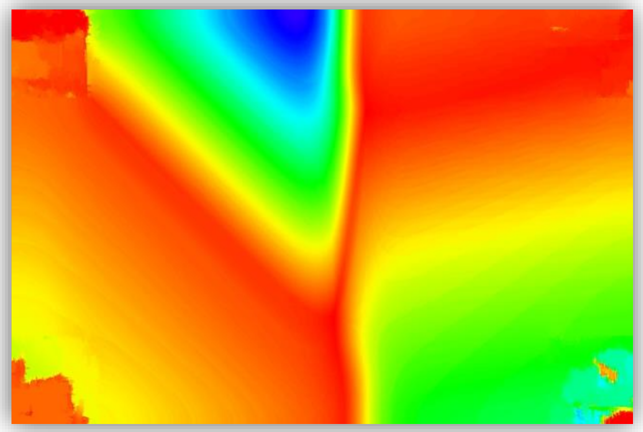
You MUST include the following (though not limited to):

- Failed interim results

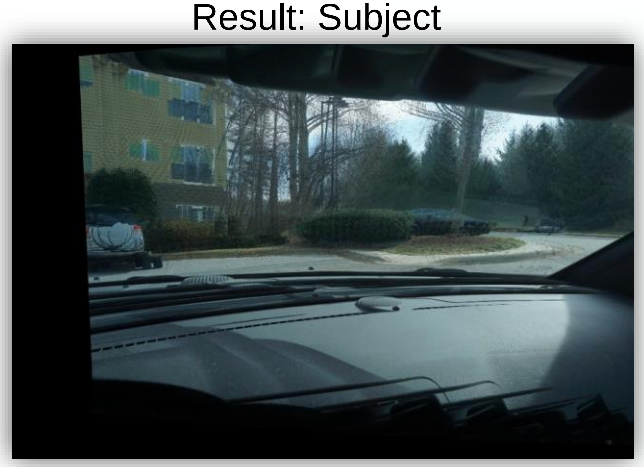
Reference Image



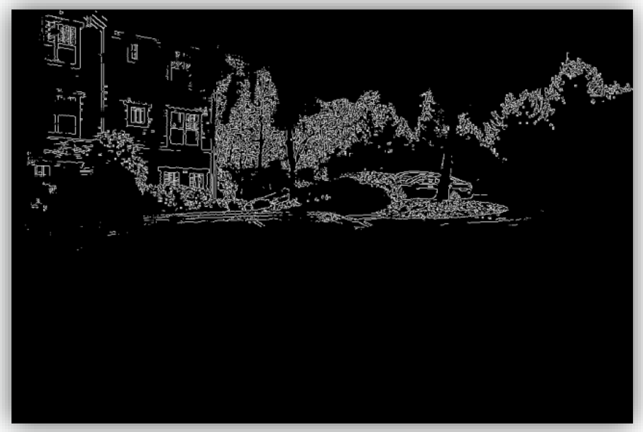
Dense Optical Flow



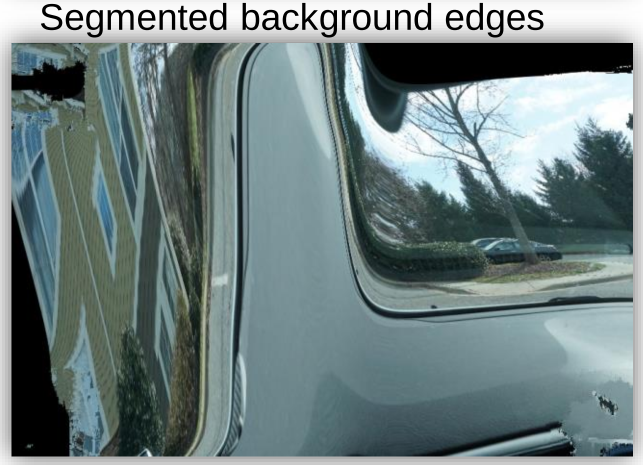
Result: Subject



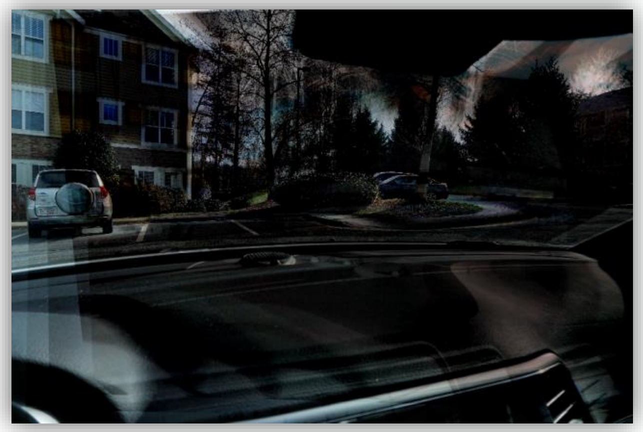
Edges



Segmented background edges



Result: Obstruction

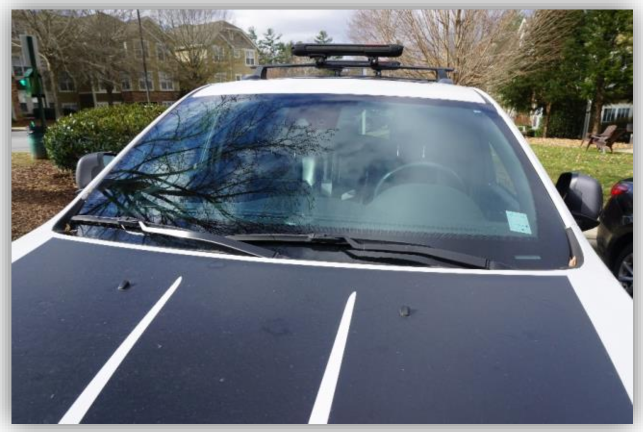


Project Development : Image Sequence 3

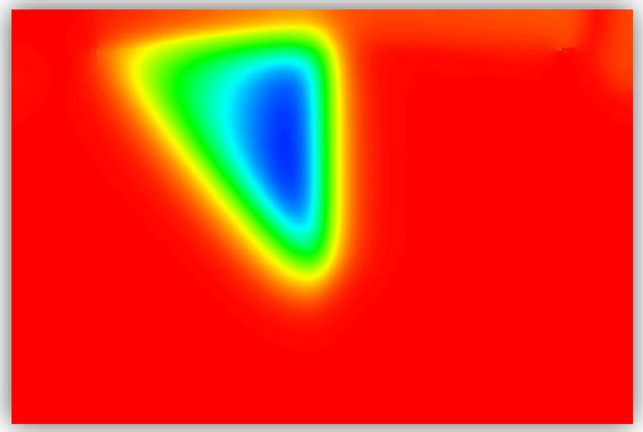
You MUST include the following (though not limited to):

- Good interim results

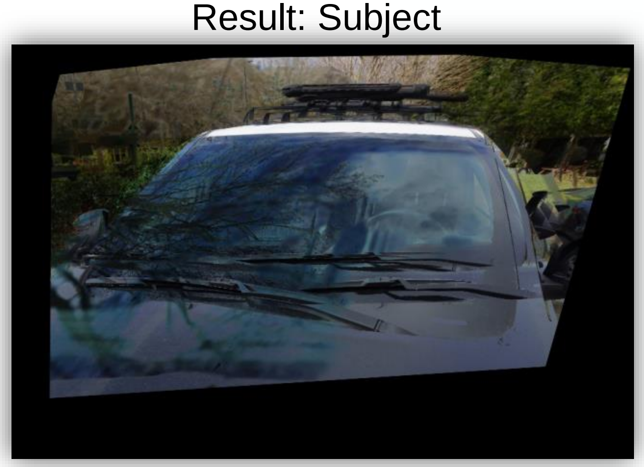
Reference Image



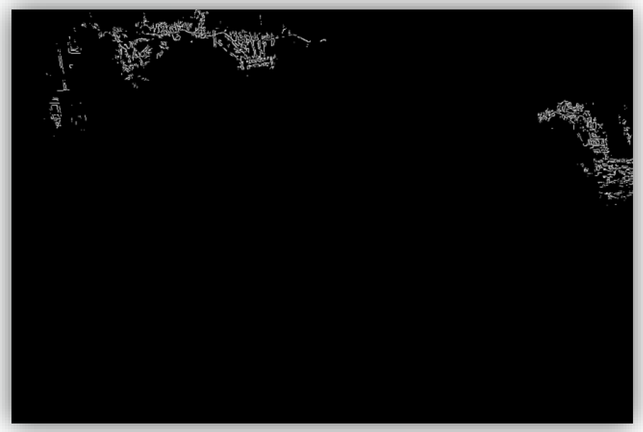
Dense Optical Flow



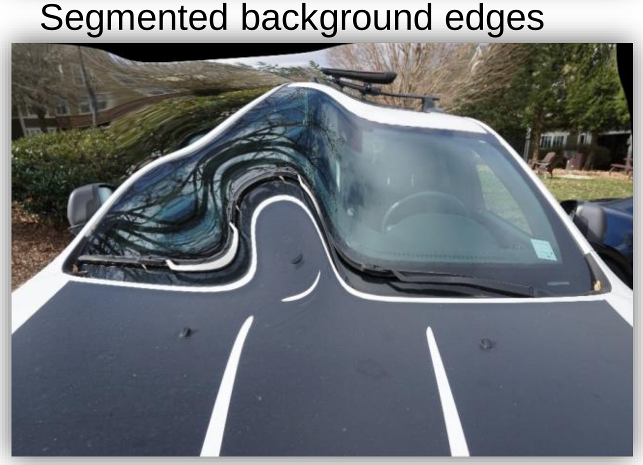
Result: Subject



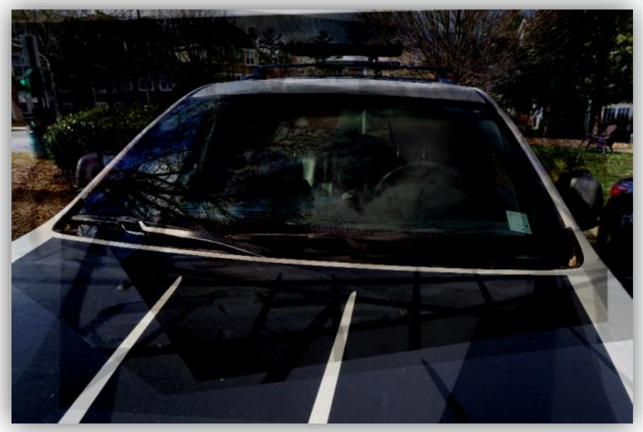
Edges



Segmented background edges



Result: Obstruction



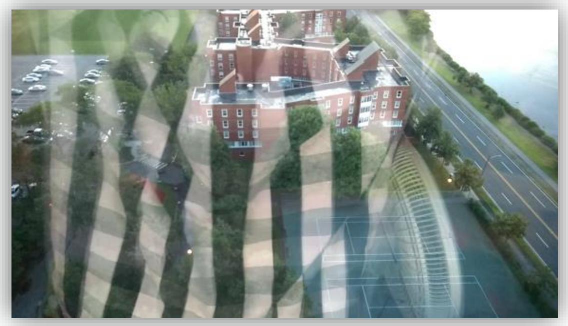
Project Development

You MUST include the following (though not limited to):

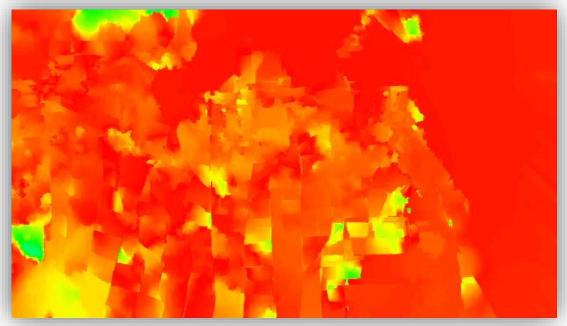
- What did you finish? What is not finished?

I was able to finish edge detection, edge pixel extraction, and edge pixel matching between images in sequence. I was able to finish running RANSAC to segment the background and the obstruction. I was able to generate the new images which correspond to the matching pixels between images for the edge pixels and generate a dense optical flow. I was also able to extract both the background and the obstruction from the image.

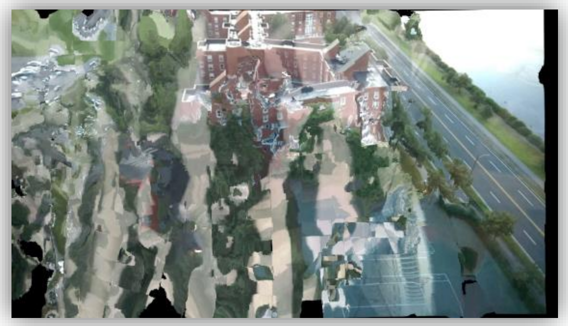
I was not able to properly align my images using the dense optical flow. I attempted using cv2.remap and I am not sure what went wrong. The results from using cv2.remap, rendered my images unrecognizable.



Frame0004



Dense Optical Flow



Result

Project Development

Use several slides for a **DETAILED DISCUSSION** of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

You MUST include the following (though not limited to):

- **What would you do differently?**

If I had to do this project over again, I would work with a teammate. This project by itself was very intensive and time consuming. Having a teammate to work with would have allowed more of this research paper to be implemented. I also would have posted even more questions to Piazza to try and get clarification on many of my unanswered question. I would have spent more time on figuring out how to apply the dense optical flow to better align my images. Having a teammate may have allowed for the missing information in the research paper to be gathered much faster.

Computation: Code Functional Description

Walk through your code functions:

- Code discussion may be incorporated in the Project Development section if that works better for you.

- - **Included functional description within the Project Development section of report**

Resources

- OpenCV
- Numpy
- MotionAndOpticalFlow.pdf (in resources) – Motion and Optical Flow by Ce Liu, Steve Seitz, Larry Zitnick, Ali Farhadi https://www.dropbox.com/s/lz87omryztwzpcu/MotionAndOpticalFlow.pdf?dl=0
- ImageProcessing.pdf (in resources) – Image Processing *Traitement d'images by Silvia Valero https://www.dropbox.com/s/l4qzrl8zydhbgxp/ImageProcessing.pdf?dl=0*
- OpticalFlow.pdf (in resources) – Image Processing: Optical Flow by Aleix M. Martinez https://www.dropbox.com/s/efn3gmu9fbe560a/OpticalFlow.pdf?dl=0
- OpticalFlow_1.pdf (in resources) – Optical Flow by Steve Seitz https://www.dropbox.com/s/8sa141qwi7nu014/OpticalFlow_1.pdf?dl=0
- SegmentationLecture.pdf (in resources) – Segmentation and Scene Understanding by Chris Choy https://www.dropbox.com/s/c522zxv0mkqc4zf/SegmentationLecture.pdf?dl=0
- Computational Approach to Obstruction-Free Photography. Image 1 – From the Research paper " A Computational Approach for Obstruction-Free Photography" SIGGRAPH2015

Resources

- Piazza posts - https://piazza.com/class/jzh9wxbybdo73g?cid=1008
- https://inf.u-szeged.hu/~ssip/2008/presentations2/Kato_ssip2008.pdf
- https://web.stanford.edu/class/cs231a/lectures/SegmentationLecture.pdf
- https://www.cc.gatech.edu/~afb/classes/CS4495-Fall2014/slides/CS4495-OpticFlow.pdf
- https://sites.google.com/site/obstructionfreephotography/
- https://ai.googleblog.com/2017/04/photoscan-taking-glare-free-pictures-of.html
- CS4495-OpticFlow.pdf (in resources) – CS 4495 Computer Vision Motion and Optic Flow https://www.dropbox.com/s/8xd1o7akr45uih7/CS4495-OpticFlow.pdf?dl=0
- Kato_ssip2008.pdf (in resources) – Markov Random Fields in Image Segmentation by Zoltan Kato https://www.dropbox.com/s/k96qa6s8uhrln9e/Kato_ssip2008.pdf?dl=0
- lecture_1015_motion.pdf (in resources) – Motion and Optical Flow by Jason Corso https://www.dropbox.com/s/amr9izwanley35h/lecture_1015_motion.pdf?dl=0
- https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/

Resources

- https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/

Appendix: Your Code

Code Language: *Python*

List of code files:

- *Main.py*
- *FinalHelper.py*