TOPICS:

* Artificial Intelligence
* Machine Learning
* Computer Vision
* Possible Techniques:
  + Computer Vision: Photogrammetry
* Neural Networks

How to Develop a 3D image:

* Methods:
  + Several images at different angles
  + 3D models
  + Position of the camera using triangulation method
  + Using video to render the objects on the ground

Questions?:

* What form does our data come in?
* What program is capable of doing this?
* What programming language does it use?
* How many things we want to recognize?
* Can we create a 3D image from a 2D image
* Why use 3D vs 2D for image recognition?
* Does 3D image immply Point Cloud
* Does that process require lidar?
* Can multiple images at different angles do the same as lidar?
* What are the biggest challenges you face?

Resources:

* N.E.O.N : free LiDAR data

Contacts:

* Steve Cassidy: (267)885-6835
* Nianqi Tian:(848)466-3422
* Mahmoud Shabana : (646)427-8840
* Gil Eckert (732) 687-3295

Sources:

* <https://github.com/openimages/dataset>
* <http://www.wolfib.com/Image-Recognition-Intro-Part-1/>
* <https://www.tensorflow.org/>
* <https://www.youtube.com/watch?v=EYbhNSUnIdU>
* <https://www.youtube.com/watch?v=lOwWNV9rpcc>

5/31

Conference call from AA

Discussed Photogrammetry

They see big promise

Topic would be more force on building 3D model from different angles pictures using C++.

Combining pictures, camera position, camera angles into 3D model.

How can we help them?

Conference call with AA:

They would like us to help them calculate camera placement using Vulkan platform. We need to use input they give us and use the Vulkan api to program the software to calculate camera positions from multiple images and use that information to create 3D point clouds from their 2D images.

Team needs to look at “Resource links” document provided by AA on the shared google drive..

Expecting more resources on Vulkan etc and our objectives from AA.

6/4

Aerial applications shared drive:

<https://drive.google.com/drive/folders/1IKS3kHu3kvaMy6ZmP-WCVkwYTorDMTS7?usp=sharing>

GPS coordinates are in the properties of each image.

We are in the process of peeling off picture properties to create some sort of a cluster diagram to see how the pictures are distributed around the cell tower.

One team member installed Vulkan but was unsuccessful in making anything work.

6/5

Our goal before conference call:

Drone takes pictures of an object.

We use pictures to generate 3D point cloud scene using external software

We use 3D point cloud as input to other software to identify what it is.

The output will be whether the object is correctly identified or not

Our success will be based on ability to correctly recognize objects in the scene

Conference Call #2:

* Concerns towards Photogrammetry:
  + visualSFM: less coding, easy method for generating point cloud
  + Creating their own 3D generation software (using vulkan)
  + The math behind the creation of point clouds from 2D imagery
  + Point cloud helps see it visually for large area processing.
  + Systematic flight pattern
  + 70% overlap!!!
  + Cell tower Photos:
    1. Computing the camera position using relevant to other photo positions
    2. Common pixel bits and points between pictures to create a plane of where the cameras are precisely
* Concerns towards TensorFlow:
* Recognizing the 3D object
* Narrowing our focus into recognition
* How to use TF to locate the common points in objects from multiple images then use those points to calculate actual camera position. (1 second = about 6 ft). So we would want to do better than that and do the calculations rapidly.

6/6/18

Image processing tutorials OpenCV Python

<https://www.youtube.com/watch?v=Z78zbnLlPUA&list=PLQVvvaa0QuDdttJXlLtAJxJetJcqmqlQq>

<https://www.youtube.com/watch?v=1pzk_DIL_wo>

<http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding>

We need some power

* AA can give us cloud usage

We still need to run the Visualsfm for helping us to understand making point cloud from 2d pictures.

For matching the pictures we can use OpenCV library in python.

6/11/18

Tian was able to compare one picture to another (python program: XXXXX?).

Using OpenCV (import cv2)

Pixel by pixel comparisons between 2 pictures for R, G, and B using Python.

Averaged 78% for all 3 colors between 2 similar pictures (R,G,B similarities were averaged).

To test again identical pictures yielded 100% similar as expected.

Different pictures of the cell tower yielded around 40% similarity.

Pictures were scaled down from 3000 by 4000 to 300 by 400 with about the same results.

The processing time on the scaled down pictures was significantly less.

Overall comparison of pictures for similarity should ideally divide each picture into puzzle pieces (chunks of pixels) that can be matched for similarity with puzzle pieces from other pictures. Puzzle pieces will be deemed similar if a decided upon threshold of similarity is met. The size of each puzzle piece is another decision that is open for experimentation and discussion.

Also, we are now able to access picture properties in excel (python: fileproperties.py)and are exploring how to calculate distances between two pictures taken of the same object using longitude/latitude and altitude. Haversine calculation can be used but may not be significant enough to use seconds data. The goal here would be to see which pictures were taken in the closest proximity to one another to help narrow our focus as we start looking for similarities between pictures. Distance between lat and long pictures: <https://www.movable-type.co.uk/scripts/latlong.html>

We will start uploading our code to Gitlab as soon as AA can provide us with access to that and to some cloud resources which should allow us to do more complex calculations since our hardware is limited.

6/12/18

Pillow : <https://www.blog.pythonlibrary.org/2017/10/24/enhancing-photos-with-python/>

How to determine an object in an image : <https://stackoverflow.com/questions/21338431/finding-the-position-of-an-object-in-an-image>

Conference Call:

What to do:

Important feedback:

* Finding which images are similar enough to find OVERLAP
* Pictures that face the same direction
* Determine if grayscale is more effective than colored photos
* Color plot histogram

6/13

Is differential gps being used by AA drones. If not, then why not?

Start of an algorithm:

* Scale down pics to a low density (say from 3k by 4k to 300 by 400)
* Use Steve’s openCV program to create pics with objects outlined <https://drive.google.com/open?id=1DG2ld6D3xcqbjSCOC_dplirI0X2uKA9v> (edges.py creates outline,Outlined photos are in /TrainTracks/outlines/

Uses Canny algorithm to detect edges and outline images in black and white by calculating certain min and max thresholds

See: <https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html>

<https://stackoverflow.com/questions/24862374/canny-edge-detector-threshold-values-gives-different-result>

<https://stackoverflow.com/questions/35586206/how-to-get-an-average-pixel-value-of-a-gray-scale-image-in-python-using-pil-nump>

* Use Tian’s open CV program to compare outlined pics to one another for similarity
* Use existing picture coordinates, apply points from outlined pictures to create a new point cloud image.
* Isolate the desired image (ie cell tower) from the point cloud. How?

How to calculate true camera position?

* Seems like we need some anchor points to measure from
* Do we use the point cloud to help us modify existing picture coordinates?
* Do we use other pictures to inform each other’s position?

Conference Meeting #4

Summary of Work:

1. have developed a program that matches key features from two photos using OpenCV as well as return the (x,y) pixel coordinate of each match.
2. have created outlined versions of the drone data that was provided to us by AA.
3. Steadily improved our similarity code, but still takes several hours to complete
4. Using panorama imaging to help find camera orientation?
5. Determine 3D distances between pictures to eventually create a point cloud by finding similar pictures.

Questions:

1. Where to go from feature matching?
2. What is the upside and downside of using differential GPS?
3. What might be a good way to sort our photos?
4. What data should we try to retrieve in order to create a 3D reconstruction?

6/20

Determined that latitute, longitude, and altitude picture properties were not reliable enough to help us quickly organize our pictures by similar orientation as opposed to using pixel level comparisons using python (as per Tian code).

Looked into the possibility of using color histograms to help us achieve this sorting by orientation.

6/23 Gil created some color histograms. Next step is to compare those histograms for similarity.

Steven was able to load and run VisualSFM on his mac to render a 3D point cloud. Progress.

6/25/18

Link for comparing color histograms (Tian will work on this):

<https://www.pyimagesearch.com/2014/07/14/3-ways-compare-histograms-using-opencv-python/>

With the first method of compares is using 4 OpenCV build-in methods. (Hellinger, Correlation, Intersection, Chi-Squared) With the picture of 1,3,4,6 compare to 1, 3 of the methods are saying that 1 and 6 are similar (the rest one is chi-squared). And after making the picture 4 brighter, the method turns out that picture 4 and 1 are more similar than it was.

Reference: <https://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html?highlight=comparehist#comparehist>

7/3/18

The histogram uses 2 mins to get the hist(color information) from the 170 pictures. The compare part uses less than half of a minute to go throw 170 by 170.

With the result of the histogram, Tian decided to group pictures with altitude and histogram.

7/10/18

Submitted completed code that uses a webcam to calibrate. Using Python, we were able to access the webcam and track key features on a checkerboard to help “correct” the image. Using equations similar to the camera matrix, distortion coefficient; the code is able to correct distortion and orientation while displaying points on the checkerboard.