Research – train of thought

## Looking at given data

* Scaling factors – don’t know anything about them, currently ignoring them.
* Training Data:
  + For each group of images:
    - Calibration:
      * Camera intrinsic and extrinsic values, for each image.
    - For each pair:
      * Covisibility
      * **Fundamental Matrix (TARGET)**
* Testing data:
  + Only PAIRS of IMAGES!

## Considering Approaches:

* Use all of the above data (beside scaling factor) for training. Which means:
  + Need another model to predict Calibration
    - F(I) -> K (I is image, F is mapping, K is Calibration Matrix)
  + Another model to predict R and T (Rotation and Translation vector)
    - F(I1, I2) -> R1, T1, R2, T2
  + And last model for everything:
    - F(I1, I2, R1, R2, T1, T2, K1, K2) -> F
* Ignore everything. Use only Images and F. Use the power of neural networks and their ability to naturally learn features. When imagining SIFT detector abstractly, a neural network can probably create a more reliable detector.
* Naturally, the latter looked easier for a 1-man team to implement. It would probably be easier to focus, as we focus on an somewhat “abstract network” instead of many “precise” features.
* Pros: probably easier to implement. Derived data suggest superior results over previous attempts.
* Cons: More “blackbox”. If things won’t work, it’ll be harder to debug, and understand why.

## Research over known approaches.

* After doing a BFS in the area of image matching, I’ve encountered the following paper:
* An End to End Network Architecture for Fundamental Matrix Estimation
* <https://arxiv.org/abs/2010.15528>
* Which introduce the idea of End to End Fundamental matrix extraction using a neural network
* This neural network / paper revised several ideas I will introduce soon.
* This gave me good direction and some building blocks to start my work.

## More research and approach decision.

* So, I kept reading and digging into works already done in the subject. And it looks like most of them have decided to extract matching key points and calculate the fundamental matrix from them. As we’ve learnt in class, it is possible and needs 7 or 8 key points.
* While 8 perfect key points can solve the problem, it is not feasible. But extracting many key points, using RANSAC algorithm to get the “really good ones”, and calculate the F matrix from that, will probably get a good approximation, most of the times.

## Evaluation

* We will soon start developing an algorithm and improve it over iterations. We need a way to measure it.
* As suggested by the code, We’ll extract Essential Matrix (E) from F. We use the cameras Kalibration for that.
* After we have E, we can calculate T and R using SVD. We will calculate the error over T and R
  + This is not fully true, as we calculate q from R which I did not fully delved into it.
  + This helps us over 1 example improvement, which is a course we’ll take farther down the line.
* We’ll do MAA over those results to get the final score.

## Baselines and improvements

* Started with the given SIFT code, which scores about 0.23~ on the MAA matrix. This code gave a lot of intuition on how to solve the problem
* Improving the baseline with using more modern detector (LoFTR) I could improve the baseline to about 0.5.
* After some fiddling with image scale and RANSAC parameters, I’ve got my score to about 0.55.
* This is the base line we’re starting with. 0.55

## Building a robust platform for research and evaluation

* In this phase I’ve build a notebook that will make it easy for me to take 2 images, and plot them.
* Run a matching algorithm, and plot the keypoints, as well as the inliers.
* Calculate score to see how good (low) q and T.
* This approach help me see the problems from evaluation in my “eyes” and get intuition to what I can do to improve. There will be a section down the line showing those experiments that will set my course of improving the baseline.
* Built a robust evaluation notebook / code to run over sample of the training data, and prints results – from that I can get the pair of images that worth to explore their errors.

## Experiments

* Scroll down to the end of the doc!

## Ideas to improve (based on experimenting)

* **Ensemble of models**. Different models yield different matching points. Concatenating all those matches will probably increase accuracy
* **Input Augmentation**. Some images are a bit rotated. There are probably additional warpings that might be interesting.
  + Rotation
  + Scaling
  + Try to find the Homogony Matrix/Transformation between images and use that as augmentation.

## Debugging algorithm mistakes:

* 95276376\_1580846323-54900410\_4709458745, err\_q=179.96 (deg), err\_t=1.22 (m)

This mistake is keep repeating where the rotation is 180 degree like wtf..

* 77972670\_214814210-64223960\_5316591967, err\_q=32.57 (deg), err\_t=30.73 (m)

Looking at this mistake, It looks like the images are rotated about 30 (deg) one from another.

Maybe aligning them and then extract matching points could solve the problem.

## Choosing RANSAC Algorithm:

* Choosing USAC\_MAGSAC – explain why
* Fiddling with parameters like:
  + Threshold – tried 0.5, 0.25, 0.15.
    - 0.15 improved results
  + Confidence – tried 0.99999, 0.99, 0.9999
    - 0.9999 seemed to work
  + Iterations – tried 10K and 20K
    - 20K improved results.

Experiments!

# Experiments

scene = 'sagrada\_familia'

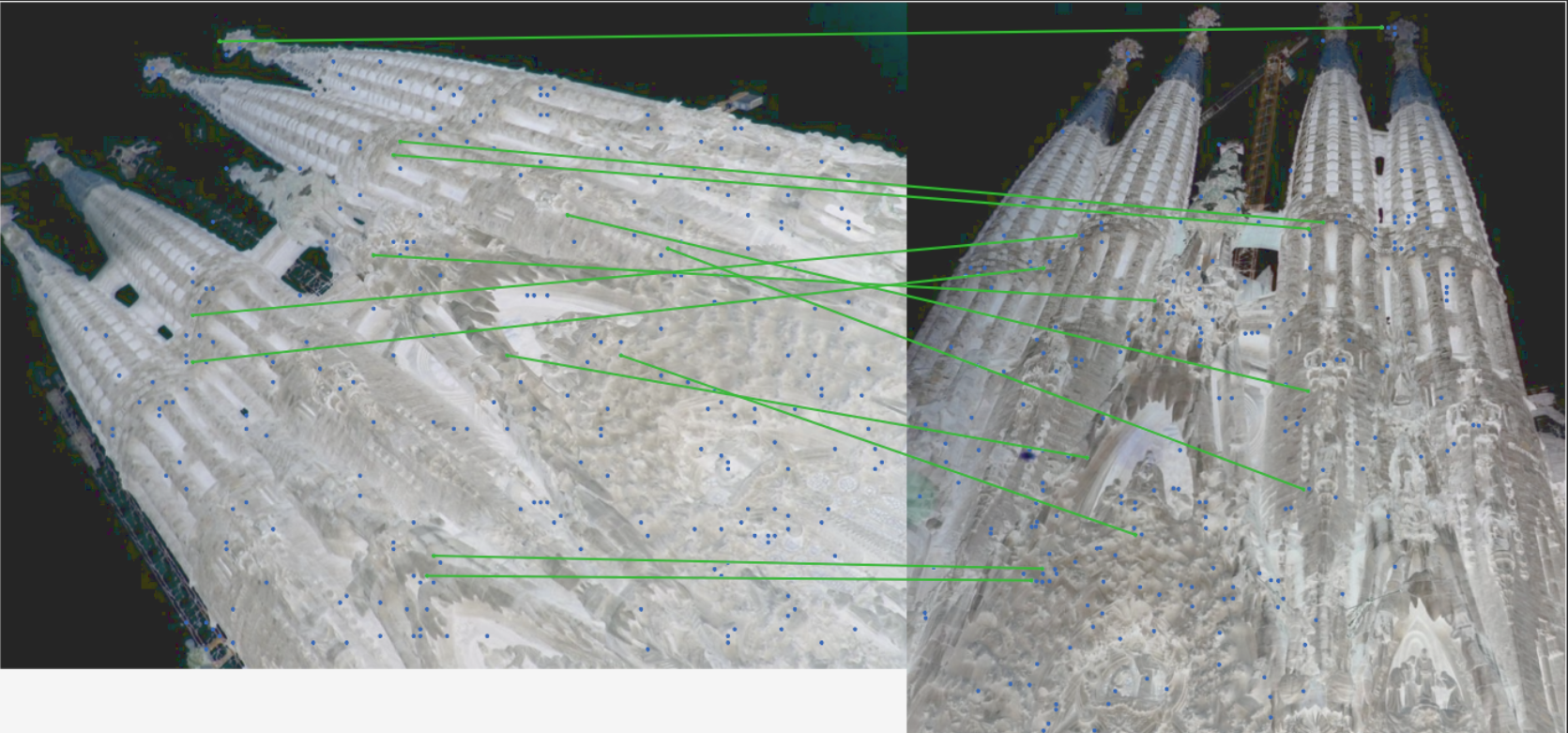
image\_1\_id = "77972670\_214814210"

image\_2\_id = "64223960\_5316591967"

A close-up of a building

Description automatically generated

Key points:



Error:

Pair "77972670\_214814210-64223960\_5316591967,

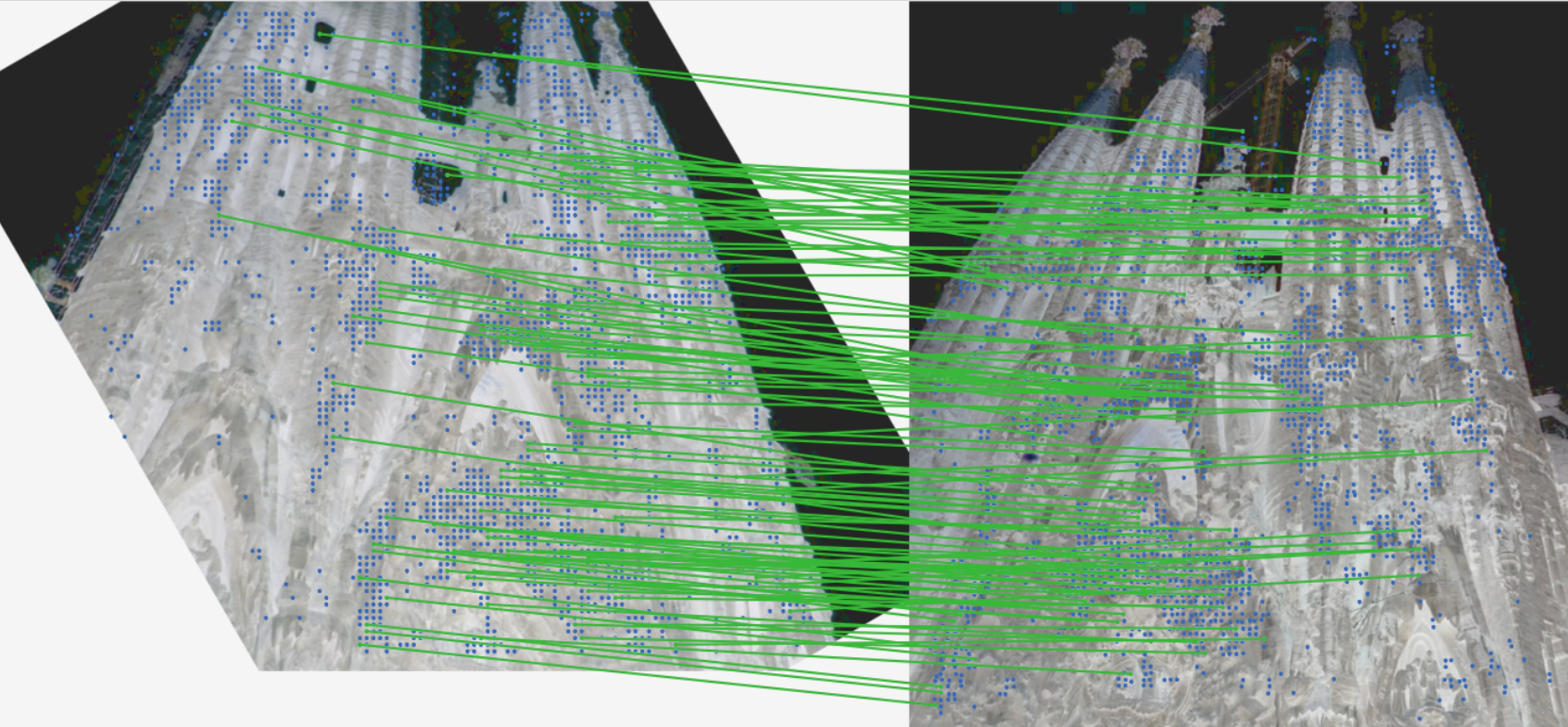
rotation\_error=15.45 (deg)

translation\_error=11.29 (m)

We can clearly see that we don’t have a lot of matching (inliners) key points

The images looks rotated.

We’ll rotate it to somehow align:



And holy cows! So many more keypoints.

Even though we’ve cutted parts of the first images. We have many more keypoints inliers!

And the error accordinally:

Pair "77972670\_214814210-64223960\_5316591967,

rotation\_error=0.63 (deg)

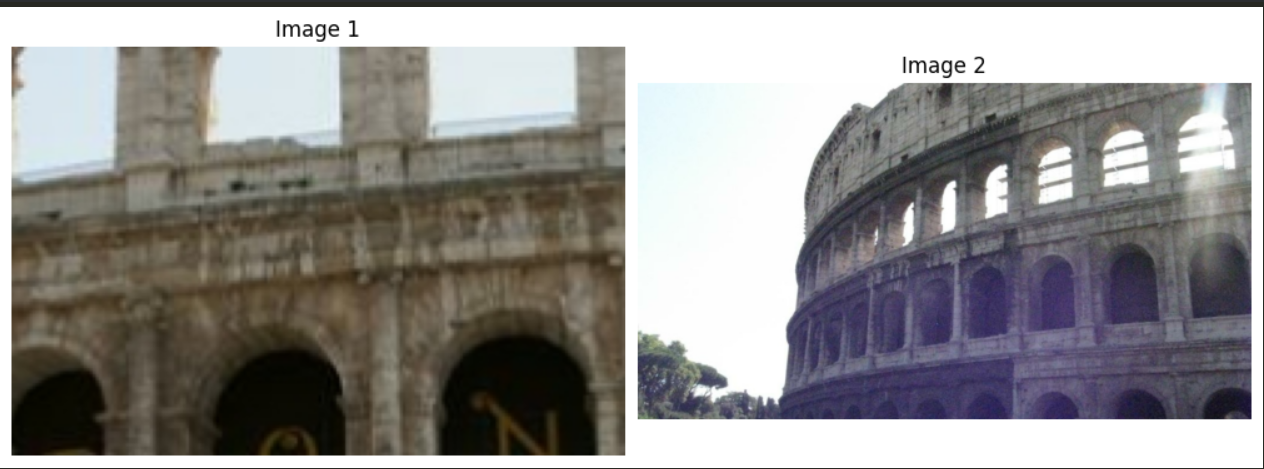
translation\_error=0.70 (m)

Decreased significantly!

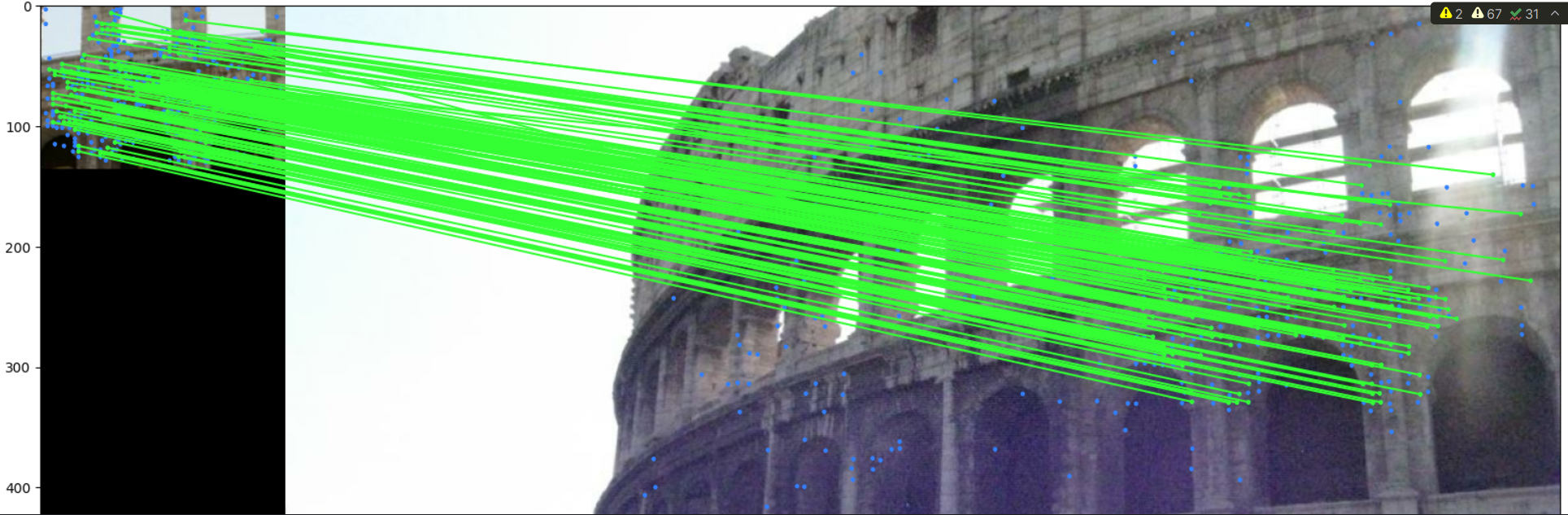
* We can create a better transformation where we cut less of the rotated image
* We can do some image augmentation with rotation and concat the matching keypoints
* Even if we do wrong with the image (rotate when not needed) ransac will remove outliers if needed, and concatenating keypoints will prove robust.

scene = "colosseum\_exterior"

image\_1\_id = "95149062\_6032288272"  
image\_2\_id = "26236359\_2890221439"



Key Points:



Error:

Pair "95149062\_6032288272-26236359\_2890221439,

rotation\_error=2.12 (deg)

translation\_error=9.90 (m)

I’ve tried to rotate the left image a bit (-6 degrees)

We got a few more points and decreased the translation error:

Pair "95149062\_6032288272-26236359\_2890221439,

rotation\_error=2.50 (deg)

translation\_error=4.32 (m)

This is too specific, and not good enough. That’s not the approach here.

What I would like, is to focus on the area of the key points, and crop that part of the image.

And then try to run a matcher again. I don’t need the whole image, It’s too much noise.

Approach:

* Let’s calculate the mean of all key points.
* Will work if only 1 center exists. What happens when there are several centers? Will fail.
* If we have several centers, a clustering algorithm will suit here.
* DBSCAN Vs K-means.
* I incline on using DBSCAN because:
  + We are looking by density of clusters
  + Clusters shape is unknown (we do suspect a **not-strict** sphere of some sort. Usually here our matcher influence this. With LoFTR we see forms of squares)
  + Number of clusters is unknown (we can overcome with k-means by guessing)
  + We know the eps (space between points) because we know our matching algorithm.
* Only thing worries me here is run-time. We do use asnycio and we’ll probably await over this operation. Currently the GPU and matching algorithm is the choke point.
* How about **Mean Shift Clustering?**
  + While mean shift clustering finds the center of cluster exactly the way we want (going “uphill” by the density of points around a radius) I am worried about the run time of this algorithm
  + If I’ll have time – I will try the latter.

Another thing I found here,

The first image is of 204 X 1xx pixels. When using the max pixels of (w, h) instead of my default pixels, which was 1080 at the time of evaluating this, I see results improved.

Therefor I will try to use original size of images when preprocessing into the matcher, which means taking the m = max(h, w) and resize to (m, m).