Final Report - Matching Trees Algorithm – Gili Gutfeld

Introduction

The *Matching Trees Algorithm* project aims to automate the process of matching tree data from an Excel file with corresponding photos obtained from Google Street View. By using geographic coordinates, the system seeks to facilitate the efficient identification and verification of trees across various urban environments. The practical application of this project provides a fast and automated approach to cross-reference real-world data with visual confirmation. The main objective is to develop a software tool to match trees listed in an Excel file with photos from a directory based on their geographic coordinates.

The objectives of the project in more specific:

- Extract geographic coordinates (x, y) and tree type/name from the Excel file.
- Analyze photos from the directory, each associated with location coordinates (x, y) and angle information.
- Implement an algorithm to compare tree locations with photo metadata to identify matches.
- Generate a report summarizing matched every tree detected in each photo for each tree in the Excel file.

Algorithm Design:

- Develop a matching algorithm that efficiently compares tree locations with photo metadata.
- Incorporate geographic distance calculations and angle alignment criteria as necessary.
- Ensure the algorithm is scalable to handle large datasets of photos and tree entries.

Deliverables:

- Software tool developed in Python for automated matching of trees with street view photos.
- Documentation including project specifications, installation guide, and user manual.
- Final report summarizing matched photos for each tree entry in the Excel file.

Implemetation

The running process is very simple, the code is implemented in a google colab notebook and we only to run the code there. You can find it in the next link:

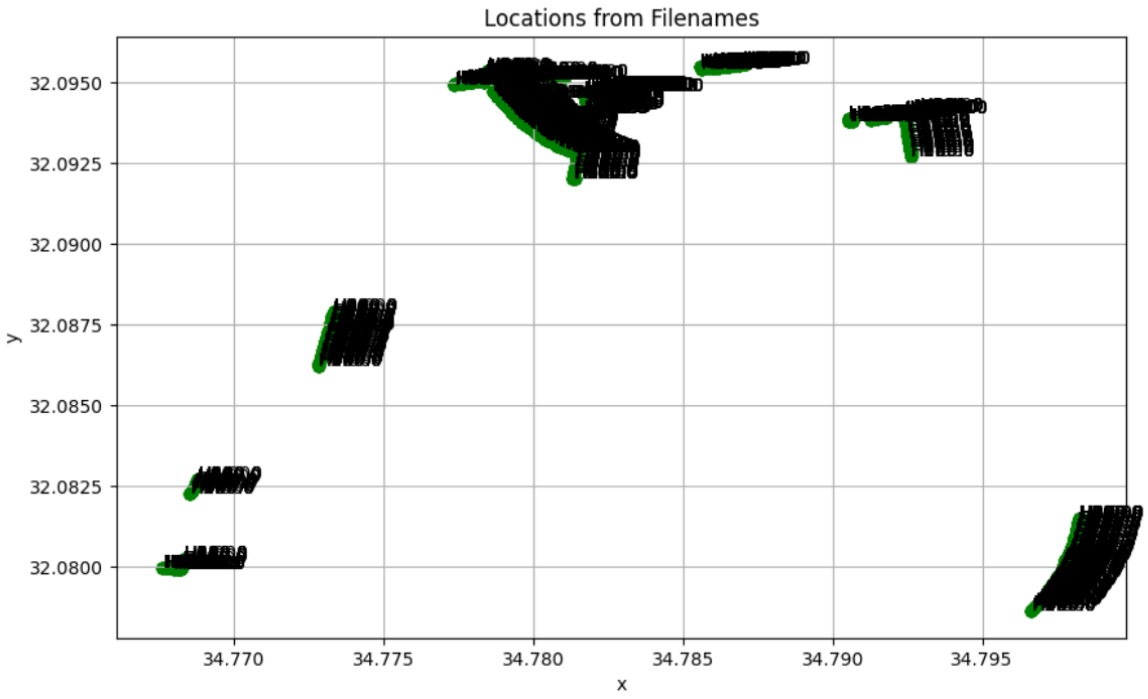
https://colab.research.google.com/drive/17I-RodS6TQYnOgUmKsnEvNnTYV185xsn?usp=sharing

The implementation leverages Python and its ecosystem of libraries:

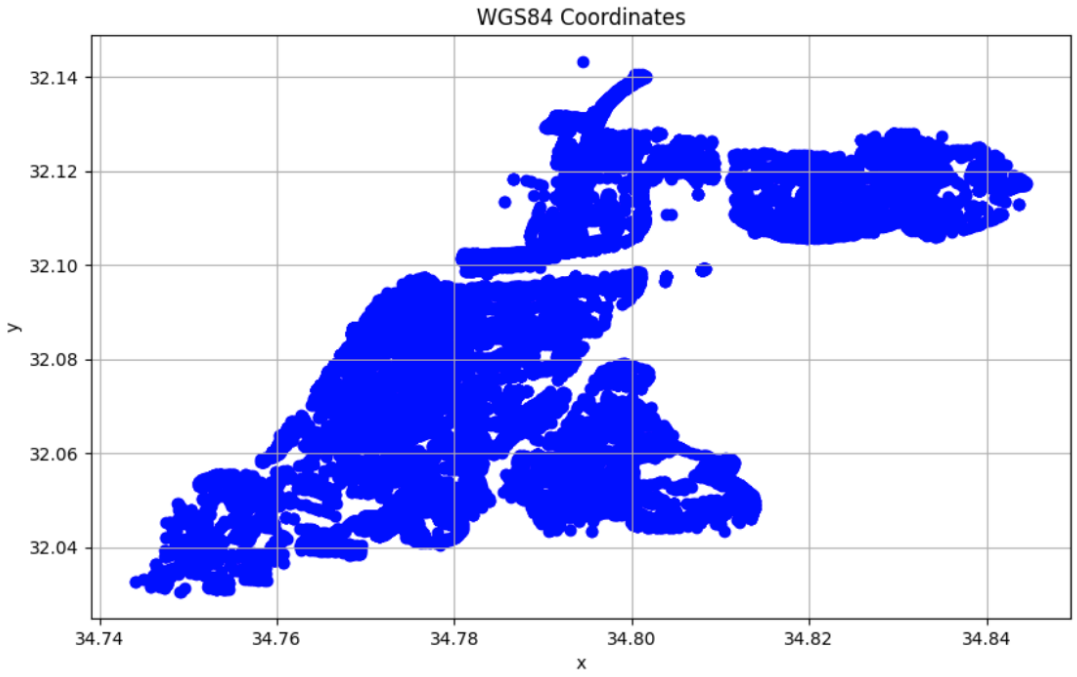
- Pandas is used to read and manipulate the Excel file containing the tree data.
- Geopy is employed for calculating the distance between geographic coordinates.
- The entire workflow is organized into a Python script, with clear modules for each task, allowing ease of modification or extension.

Table of contents

1. Matching Part - there is a url to a directory of the gsv photos (in Github). If you want to run the algorithm on new photos that is the place to select them.
2. Import the GSV images – list all the images and shows an example image.
3. Trees Coordinates – reads the coordinates file and plots all of them together. Saves in an array ‘points’ the id, x, y and type of each tree:



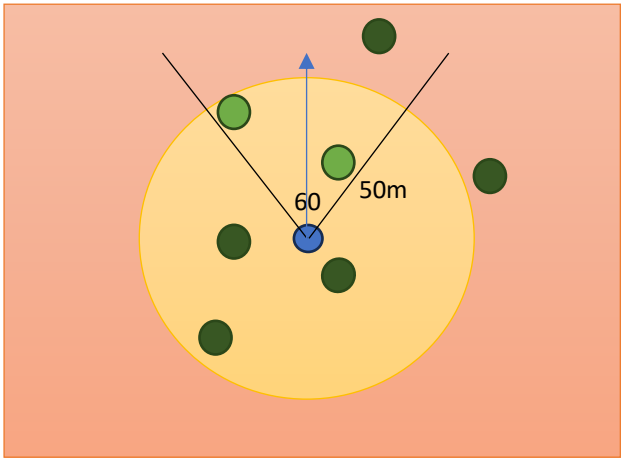
4. Images Details - saves in ‘data_image’ array all the files with their details (extracted from the file name). Then, plots the coordinates of the images:



From the plots, we observed that there are significantly fewer image points compared to the tree points in the Excel file, with the images being concentrated in specific areas.

To match the tree points to the images, we will develop an algorithm which takes an image and finds all the tree points from the excel file. Then, it puts in a list only the points that are in the right direction according to the angle of the image, and at a distance of about 50 meters from the location where the image was taken. In more than 50 meters the trees will look too small and probably a different image will include them in a closer point.

For example, if the image (blue circle) was taken in 0 degrees (north) we will consider only the two trees in the 50 meters range in the north direction which are light green in the next drawing:



Next, we take the image and find all the trees in the image with a detection model which is called Grounding Dino. Then, for each tree in the image we calculate the angle and the distance of the roots of the tree from the camera (the root is the real location of the tree). With that, we take the tree's list of the image and find the best match to each tree according to the specific location we detected.

5. Matching Algorithm – implementation of the algorithm.
6. Find matching with detection and without excel – runs the algorithm on the photos with detection, takes much more time because it's a heavy action (we need to use it only in the first time).
7. Find matching without detection and with excel – runs the algorithm on the photos without detection, it reads the excel file that was created when we used the detection before (we can use it only after running with detection). It allows us to tune the parameters of the algorithm and run it in much less time.

Methods

The project workflow consists of several key steps, as described below:

1. Data Extraction:
    - Extract tree data from the Excel file, including tree types, names, and geographic coordinates (x, y).
    - Extract geographic metadata (x, y coordinates and angle) from the photos in the specified directory.
    - We had to convert the coordinates of the excel file to the same coordinates of the photos. We did it with a python script bit it can be also done by using QGIS.
2. Matching Process:
    - Iterate through each tree listed in the Excel file.
    - For each tree, compare its geographic location with the metadata from the images, applying a distance threshold to identify potential matches.

Algorithm

The heart of the *Matching Trees Algorithm* is a geographic comparison algorithm designed to match trees in the Excel file with the corresponding photos. The steps of the algorithm are as follows:

- For each photo:
    - extract its details (name, x, y, heading).
    - Put the possible matching trees from the excel file in an array, we takes only the trees which are in a range of 50 meters (we can change this threshold) and in the right direction of the photo.
    - Run detection on the photo and get an array of detected trees.
    - For each detected tree:

- Extract the points of its borders.
- Calculate its distance from the image, which equals to the distance from bottom of the image to its roots.
- Calculate the angle of the tree in relation to the bottom of the image.
- Calculate the coordinates of the tree using the coordinates of the image, the distance of the tree from the image and the direction of the tree.
- Find the best match from the possible trees we saved before, using Euclidean distance.
- Save all the details of the match to excel file using pandas.

At the end of the algorithm we create an excel file which includes all the matching details we found, and its name has the parameters we chose, the count of distinct trees we found, the average and the median distance between the tree in the image to the matched tree from the survey.

The distance of tree step:

When we detect a tree in the image we need to estimate its distance from the image itself, then we will able to find its coordinates. For that, we find the middle bottom point of the tree which represents its root, that's the real location of the tree (its leaves or branches are not relevant). That's the easy part beacause we have the detected box's x, y, width and height. The tricy part is to estimate the distance, we know that there is a function that takes the location of the bottom point of the tree and outputs the distance, that's what we try to find:

The y range's of the roots point is between 0 to 1. We define two important parameters to the function. The first parameter is 'y_times' and it defines to how many "parts" we devides the image. The second parameter is 'y_exponent' and it defines the size (in meters) of each part.

Let's take for example y_times=8 and y_exponent=2. Then, we have 8 parts of the image (in y axis), the first part is $2^1 = 2$ meters from the image, the second part is $2^2 = 4$ meters from the image, the third part is $2^3 = 8$ meters from the image, because a tree in the end of the first part has a y value of 0.125, a tree in the end of the second part has a y value of 0.25 and so on..

The exponent effect is created because of the depth of the image, the first meter of the y distance from the image seems to be much more area in the image than the 10'th meter for example. Actually, we have infinity of meters in the image (if it is flat) and hence the size of each meter in the image strives to zero.

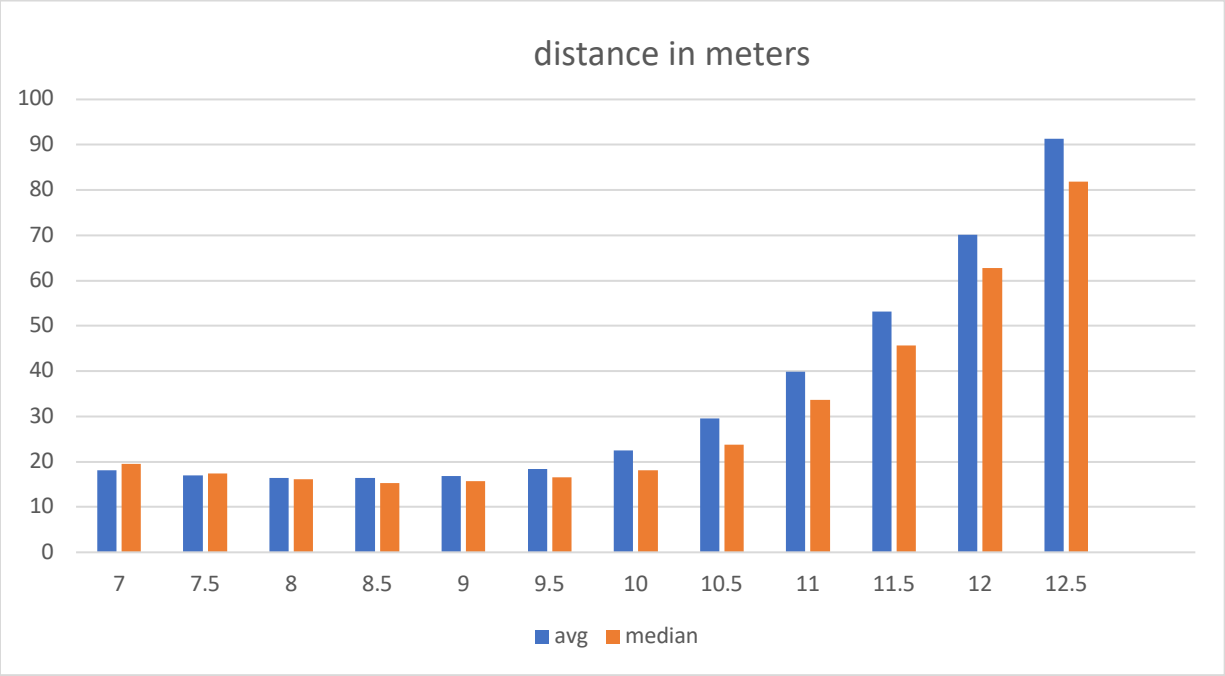
In addition, the angle is calculated by using x and y of the roots point in relation to the bottom point of the image, with tan function. After that, we divide the angle by 3 because the range of the photo is 60 degrees and not 180 (that's another parameter we can try to tune but this is more straightforward function).

Results

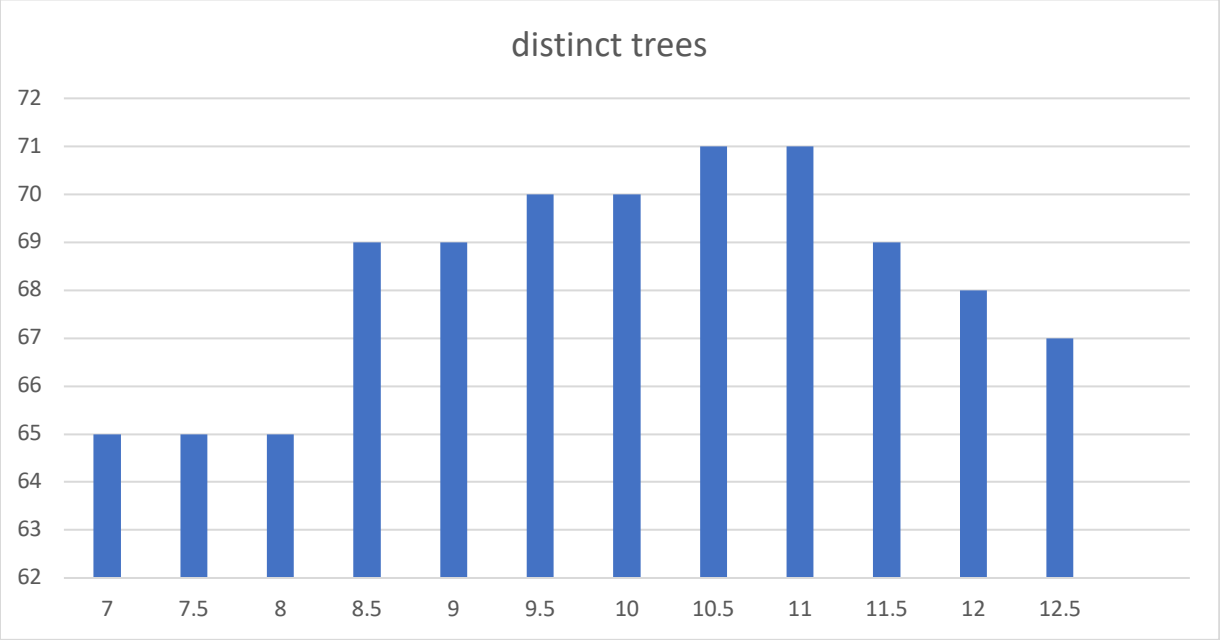
We define some ways to measure the quality of the algorithm. First, the average of the distance between the estimated location of the tree to the matched tree. It shows us how are the estimated location from the real location of the tree. Also, we look on the median distance which allows us to not be affected by outliers. Because we have an exponent here, we can have a few outliers or even one that can be very bad and change the average completely.

Second, we count the distinct matched trees in each photo and sum all of them together. The higher sum, the better result. That's because in a single image, each tree should get a unique tree from the excel file, and if we get the same match to two different trees, it's an error. We want as match unique trees as we can find in each photo. But in two different photos we can get the same matched tree (because the camera can photo the from more than one location or direction).

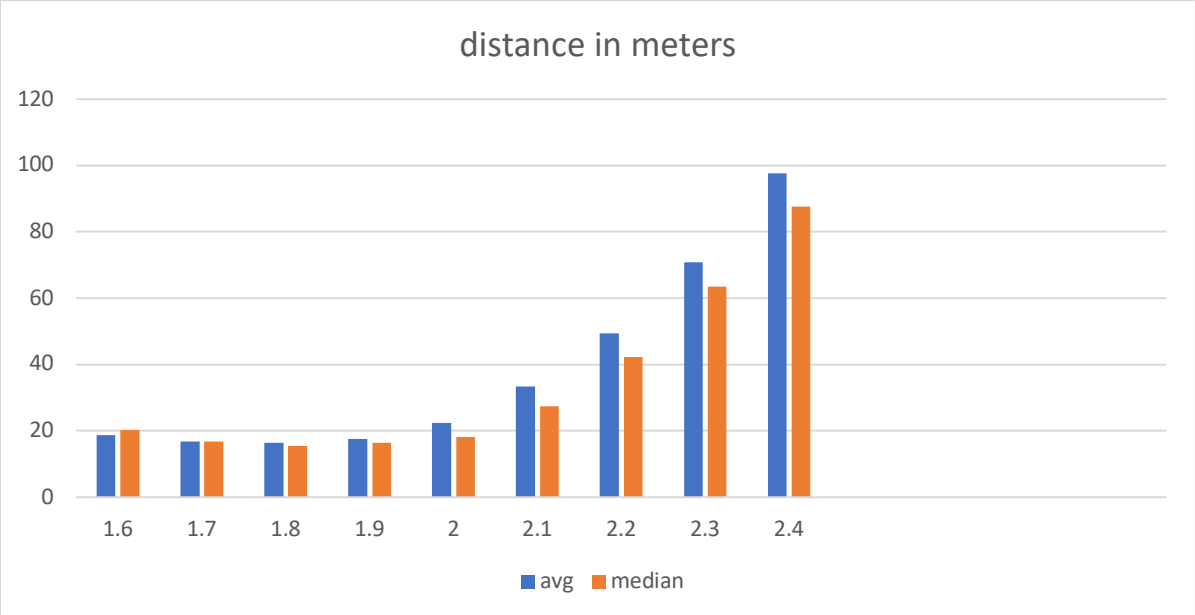
Now, let's tune each parameter. I take 200 different GSV photos for the experiments. First, we define the exponent to be 2 as constant and run the algorithm on different values of the parts parameter. The next graph shows the results that with small values of the parameter, the distance is smaller while with values which are bigger than 10, the distance becomes much larger because we have the exponent effect. The best average distance (the smallest) is in 8.5 and it is less than 17 meters. We can infer that values bigger than 10, are too big, while smaller values give small distance, but it's because the multiply is smaller. So 10 is a good value.



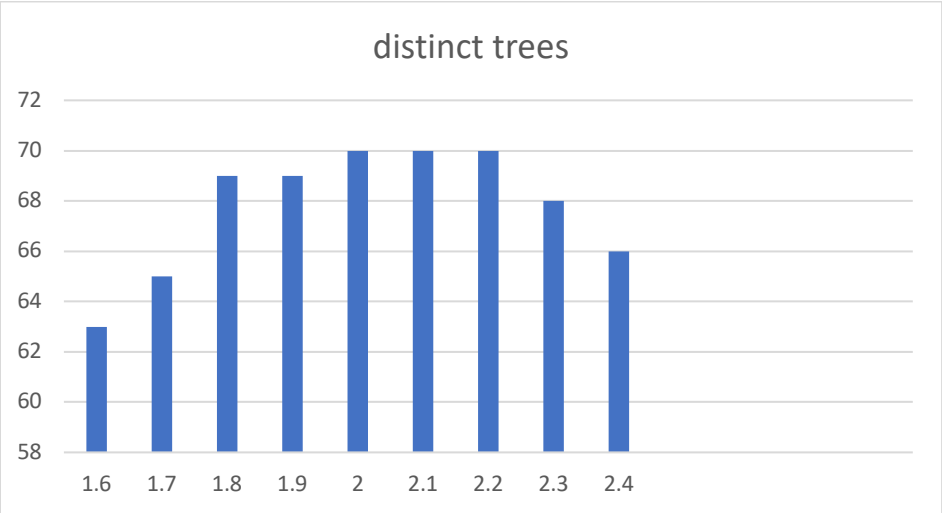
As for the number of distinct trees we receive the best when the parameter equals to 10.5 and 11, and it is 71 distinct trees. Again, around 10-11 is the best value.



Then, we define the parts paremeter to be 10 as constant and run the algorithm on different values of the exponent parameter. The next graph shows the results that with small values of the parameter, the distance is smaller while with values which are bigger than 2, the distance becomes much larger because we have the exponent effect. The best average distance (the smallest) is in 1.8 and it is less than 17 meters. We can infer that values bigger than 2, are too big, while smaller values give small distance, but it's because the multiply is smaller. So 1.8 is a good value.

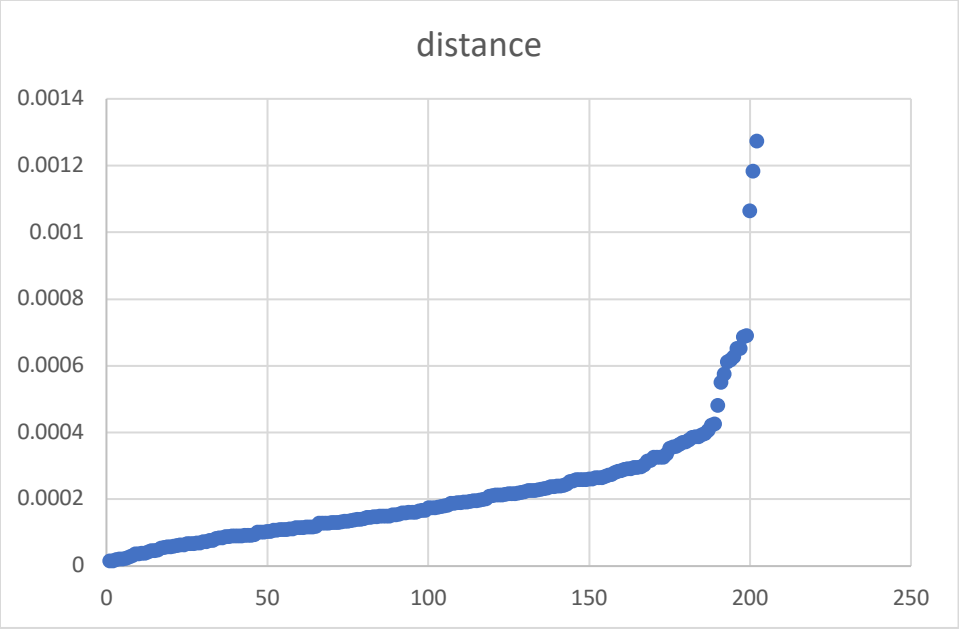


As for the number of distinct trees we receive the best value when the parameter is between 2 to 2.2, and it is 70 distinct trees.

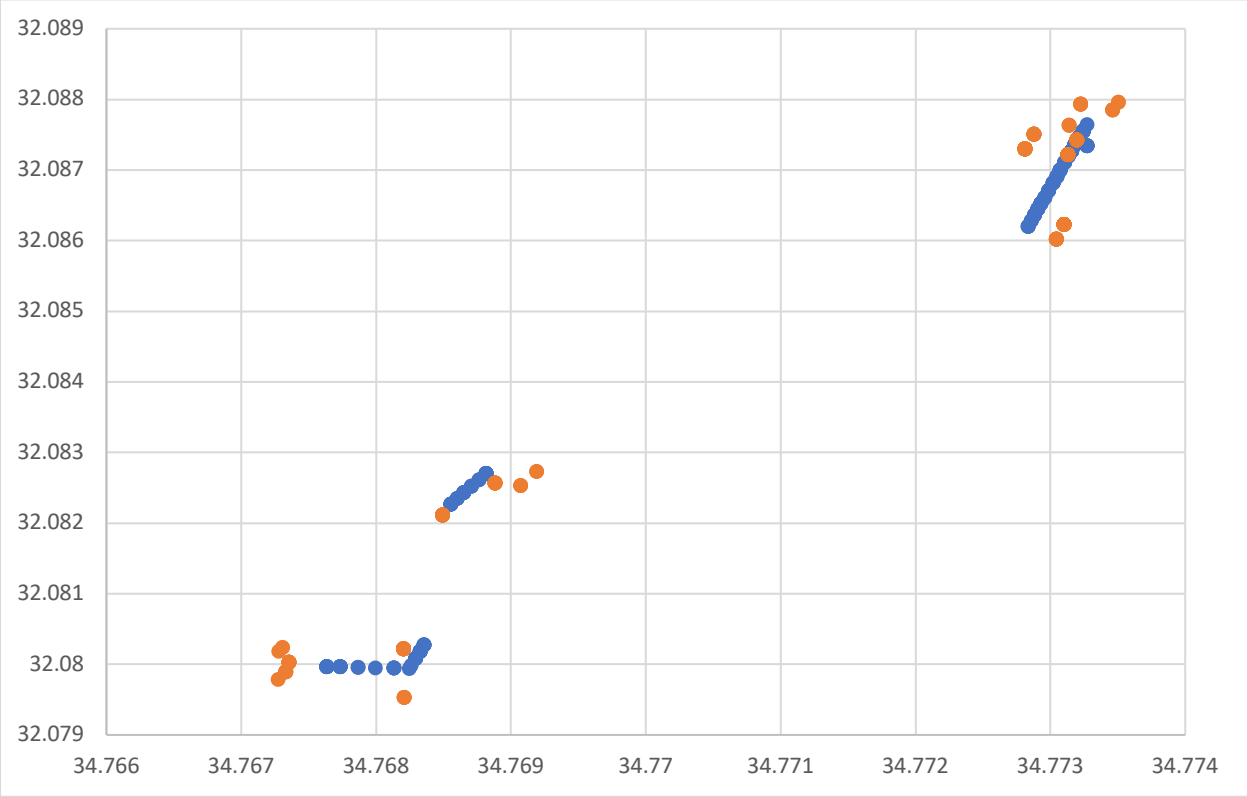


We want to combine the best values of the parameters, but we have a trade off between them. As we take a smaller exponent, we can take a bigger number of parts. I tried some different combinations of the parameters, and the best result was achieved with y_times=10.7 and y_exponent=1.9. We received 71 unique trees, average distance of 21 meters and median distance of 17 meters.

In the next graph we can see the distances between each estimated location to its matched tree for the last experiment. Note that the average are affected by a few big outliers:



The next graph shows the coordinates of some of the gsv images (the blue points), and beside them also the estimated coordinates of the trees we found in the images (the orange points):



The trees survey with the adjusted coordinates, the gsv directory, the cropped detected trees directory and the excel file with the matching results can be found here in my github repository:

https://github.com/giligutfeld/TreesProject

Here is an example of image with 5 trees, each one has its own location:

The algorithm found 6 different trees in the excel file which can match the image:

{'id': 8973, 'location_x': 34.76715127342473, 'location_y': 32.0798756525444, 'tree_name': 'וושינגטוניה חסונה'}

{'id': 8972, 'location_x': 34.76717427848561, 'location_y': 32.07994465547822, 'tree_name': 'וושינגטוניה חסונה'}

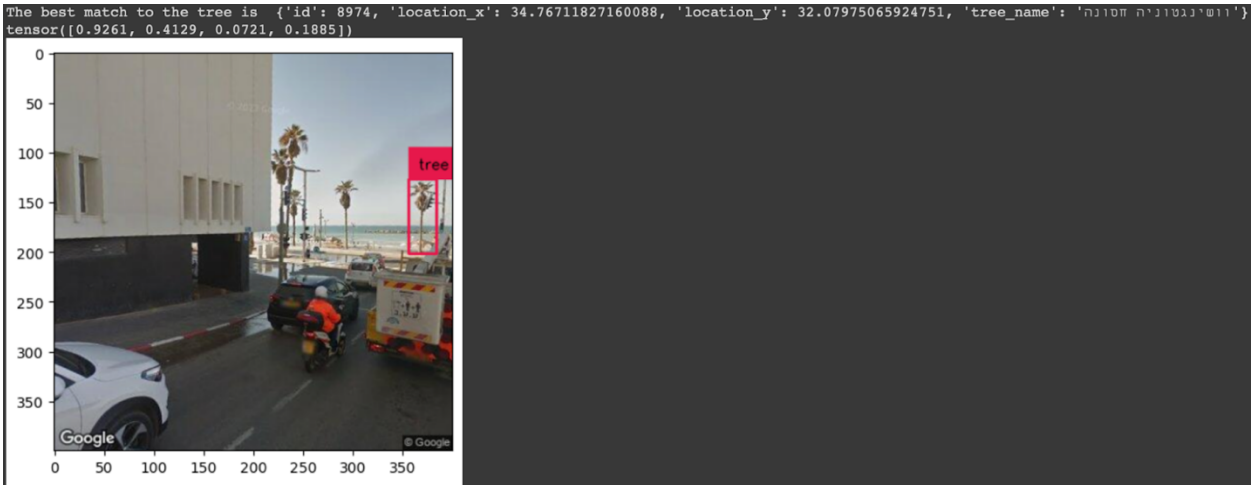
{'id': 9090, 'location_x': 34.76727227395289, 'location_y': 32.0796306578292, 'tree_name': 'וושינגטוניה חסונה'}

{'id': 9091, 'location_x': 34.76727227618476, 'location_y': 32.0797926547862, 'tree_name': 'וושינגטוניה חסונה'}

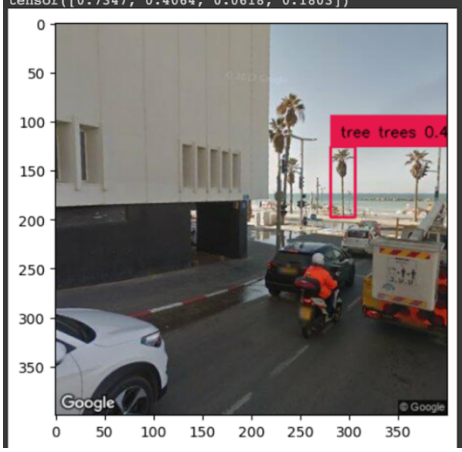
{'id': 9093, 'location_x': 34.7673302709502, 'location_y': 32.07992065604571, 'tree_name': 'וושינגטוניה חסונה'}

{'id': 9092, 'location_x': 34.76733127117031, 'location_y': 32.079899656886, 'tree_name': 'וושינגטוניה חסונה'}

Now, we run the trees detection model and for each tree we find the best match:

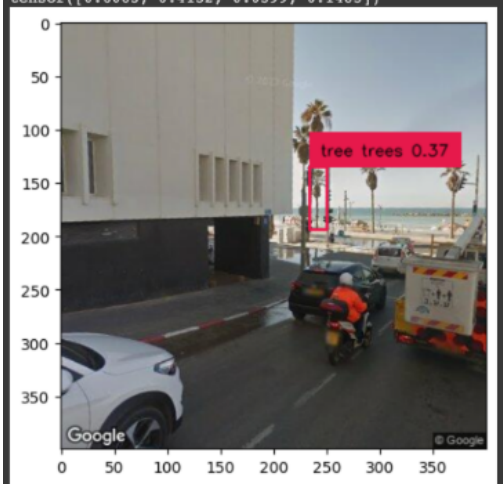


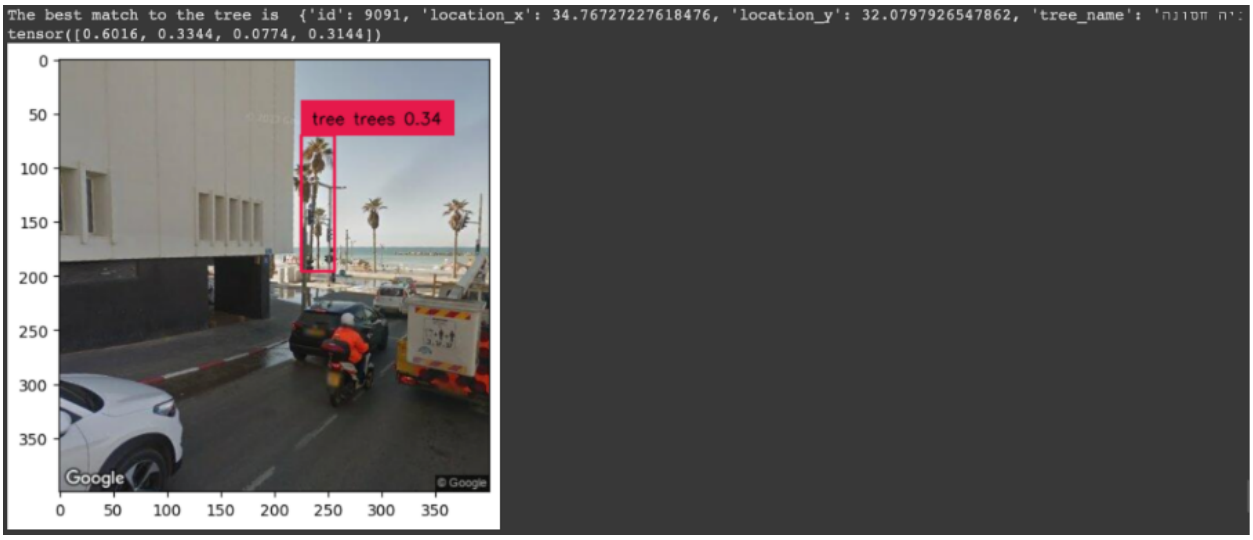
```
The best match to the tree is  {'id': 8975, 'location_x': 34.76707827864914, 'location_y': 32.0796836537445, 'tree_name': 'וושינגטוניה חסונה'}
tensor([0.7347, 0.4064, 0.0618, 0.1803])
```




```
The best match to the tree is  {'id': 9092, 'location_x': 34.76733127117031, 'location_y': 32.079899656886, 'tree_name': 'וניה חסונה
tensor([0.5678, 0.4081, 0.0479, 0.3410])
```

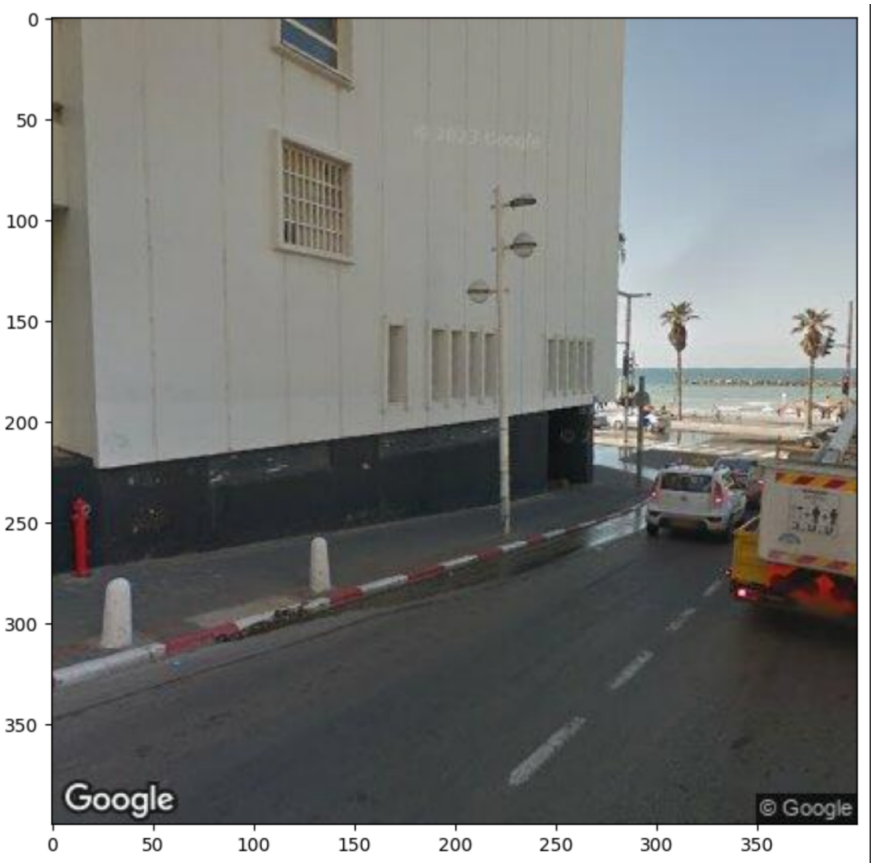

```
The best match to the tree is  {'id': 9091, 'location_x': 34.76727227618476, 'location_y': 32.0797926547862, 'tree_name': 'יה חסונה
tensor([0.6063, 0.4132, 0.0399, 0.1483])
```



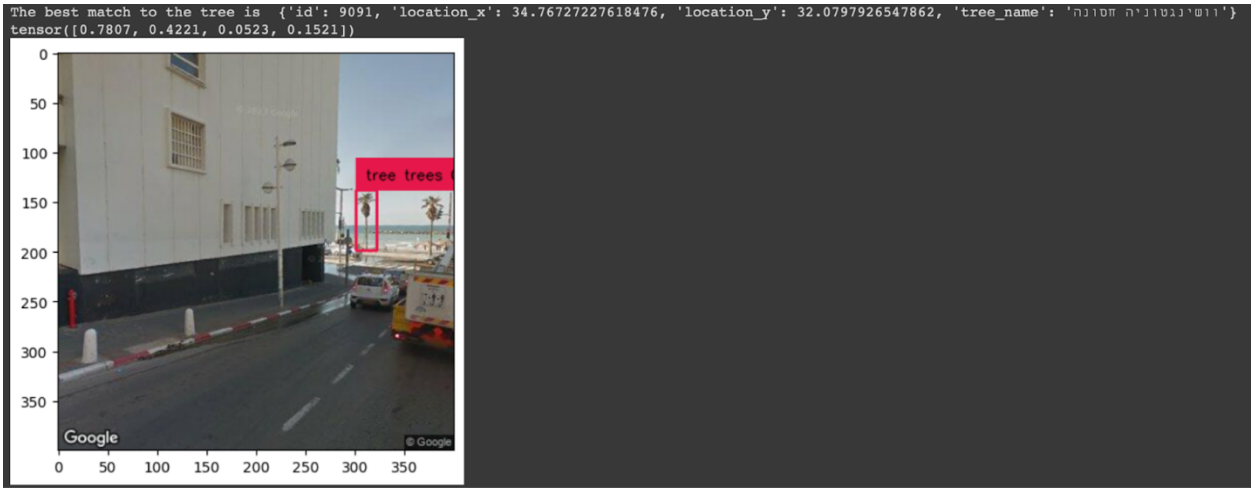


We can see that all the trees are from 'וושינגטוניה חסונה' type, and each tree got a unique matching tree from the file except of the last two. The reason is that both are almost in the same location, and it will be difficult to distinguish between them. Maybe with doing more fine tuning to the parameters we can do that.

Let's look at some more examples. We'll try to take a close image to the previous one with the same trees:

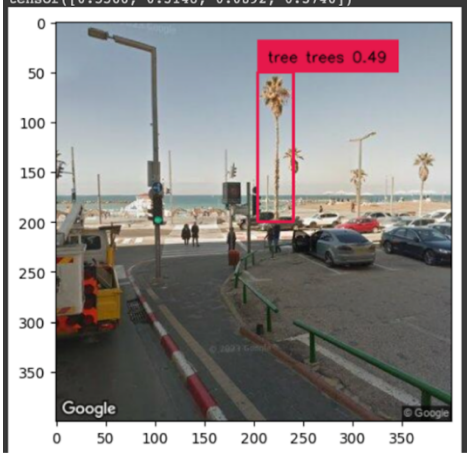


We are expecting to get the same tree ids, and for the one in the right we succeed. We got the same tree id for the same tree from two different angles of images! but that's not the case for the second tree, so we should continue to tune the parameters to get better results.

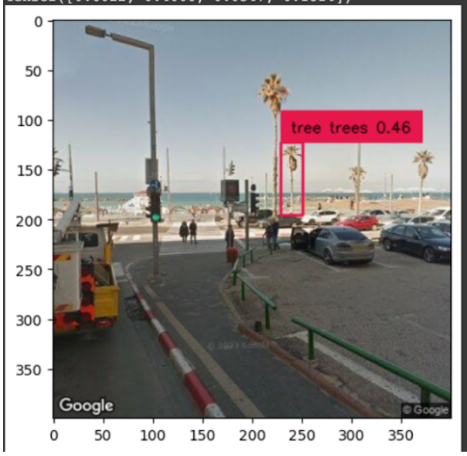


And here are some more examples:

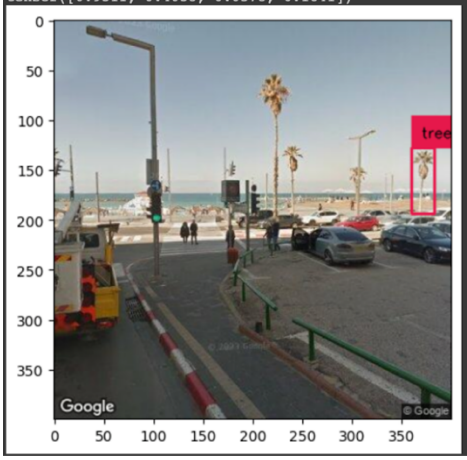
```
The best match to the tree is  {'id': 8971, 'location_x': 34.76727827761713, 'location_y': 32.08018965998098, 'tree_name': 'וושינגטוניה חוטית'}
tensor([0.5566, 0.3148, 0.0892, 0.3746])
```




```
The best match to the tree is  {'id': 8970, 'location_x': 34.76730427651198, 'location_y': 32.08024165650477, 'tree_name': 'וושינגטוניה חסונה'}
tensor([0.6022, 0.4006, 0.0547, 0.1814])
```

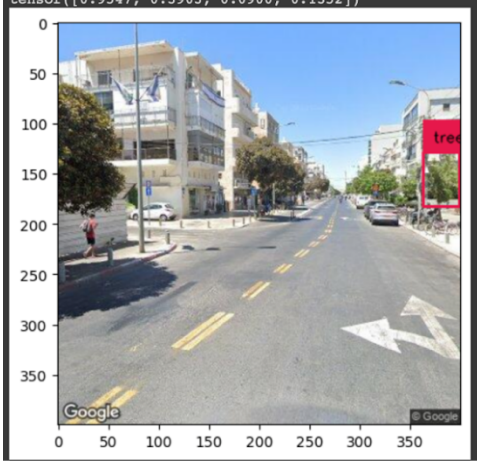



```
The best match to the tree is  {'id': 8969, 'location_x': 34.76733227189933, 'location_y': 32.08035265478529, 'tree_name': 'וושינגטוניה חסונה'}
tensor([0.9311, 0.4038, 0.0578, 0.1641])
```



```
The best match to the tree is  {'id': 24915, 'location_x': 34.77330504340988, 'location_y': 32.08814817386958, 'tree_name': 'פיקוס מעוקם'}
tensor([0.5287, 0.3808, 0.1816, 0.2028])
```


```
The best match to the tree is  {'id': 24896, 'location_x': 34.77336599775391, 'location_y': 32.08837970734307, 'tree_name': 'קופניון אנקרדי'}
tensor([0.9547, 0.3903, 0.0900, 0.1352])
```



The comparison was based on calculating the Euclidean distance between the tree in the image to the points in the excel file. Each image point, adjusted according to the angle (direction) of the image.

For an image point to be considered a match for a tree point, the distance had to be below a threshold that I selected.

Conclusions

The *Matching Trees Algorithm* provides a streamlined method for cross-referencing tree data with real-world images. By automating this process, it reduces the time and effort required for tasks like tree identification and monitoring. The flexibility of the algorithm, particularly its use of geographic distance and angle criteria, makes it adaptable to a variety of environments and use cases.

What's Can Be Next

To further enhance the project:

1. Run the algorithm on a larger dataset and remove outliear to avoid a wrong data, which consists of survey mistakes, low quality image or any more outliers.
2. Integration with Machine Learning: Incorporate machine learning models to automatically identify tree types from the photos. We will use our new tagged data to train a new model.
3. Real-Time Updates: Integrate live data from mapping APIs (e.g., Google Maps) for dynamic tree matching in real-time applications.
4. Enhanced Visualization: Develop a web interface or GIS system to visualize matched tree data and their respective photos.
5. Community Input: Allow users or city planners to manually review and confirm matched trees to improve accuracy.