PIV Project : Reverse Engineering Intersections and Crossings

(and everything related by homographies)

As the title suggests, the 2024-25 PIV project will unfold in an urban traffic setting and aims to observe pedestrian and driver's behavior. The data will be generated by processing videos and image sequences of intersections and applying basic geometric concepts that allow the extraction of metric information from videos (positions and velocities) or in a otherwise planar setting with objects moving on it.

This context is just … a context ! The techniques you will learn are general and can be applied seamlessly to almost any context with moving cameras or moving objects.

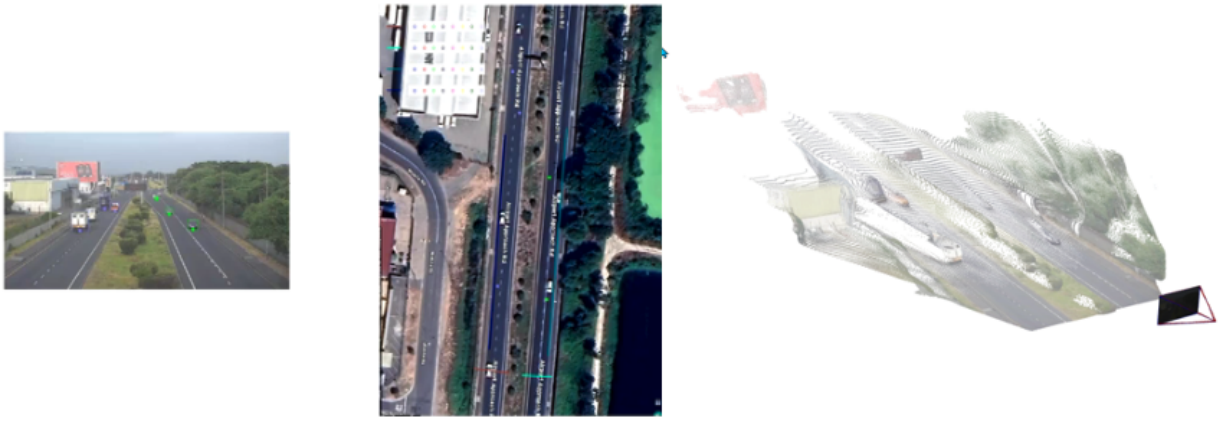
A project at two levels

The project will unfold in two main steps corresponding to two key topics: 2D and 3D processing.

Step 1 - Homographies and 2D processing

On the first and initial step, the goal is to map vehicles and pedestrians' positions to an image of Google Maps by estimating the projective coordinate transformations that map camera coordinates to Google Maps images (that are geo-referenced). This first step has in itself two intermediate steps with increasing degree of complexity :

1.1 - Processing one single video: The camera is static and the video contains moving objects that we want to have mapped to Google Maps.



With Image-to-image (perspective) transformations

and/or using the latest depth-from-images technology

For some sequences you'll have both the Google Maps image and/or a 3D map of the scene. Note that it can become quite challenging :

https://www.youtube.com/watch?v=UEIn8GJIg0E

Further details:

- You'll be given code that processes videos and outputs tracks (trajectories) of moving objects and its labels. The YOLO detector will be distributed in docker containers (you may install the package yourself, no need for containers) so that you just run code and obtain the relevant data (image tracks and object labels). The packages we will use are in the links below

- Installable from pip http://ultralytics.org or http://roboflow.org . You can also check https://github.com/jpcosteira/CIV5164Z/tree/main/0_INSTRUCTIONS and the following Github repositories:
    - YOLO (https://github.com/Toast5286/Yolo_grpc)
    - Feature matching (https://github.com/Toast5286/MatchingGRPC)
    - Dust3r
- With the YOLO tracks and a set of (given) matching points from one image (from the video) to an image of the area covered by the camera on Google Maps.

FINAL Specifications:

You will need to implement your code (in Python or Octave) in a file 'main.py' / 'main.m'. Your implementation must not require opencv or any other computer vision libraries or packages, since these will not be available in the evaluation environment. The inputs are files saved in the current directory. The following input files will be available:

- 'kp_gmaps.mat' containing a dictionary/struct with the keypoint matches between the first frame in the video ('img_00001.jpg') and the Google Maps image. Matches are stored in a Nx4 matrix containing N pairs of points in which the first two columns are pixel coordinates in the first image and the last two are pixel coordinates in the second image
- A video/sequence of images in the format 'img_0001.jpg', 'img_0002.jpg', etc.
- YOLO detections in the format 'yolo_0001.mat', etc each with a dictionary/struct that includes:
    - Bounding box: 'xyxy' with a Mx4 matrix in which each line corresponds to a detection and contains pixel coordinates, [x_blc, y_blc, x_trc, y_trc].

      (blc - bottom left corner, trc - top right corner)

    - Object ID: 'id', with a vector containing the id of all M objects.
    - Object class: 'class', with a vector containing the class of all M objects.

The output of your code must be saved in the current directory and must include:

- The transformation matrix in the format 'homography.mat', (using scipy.io.savemat in python), containing dictionary/struct {'H': H}, where H is the 3x3 transformation matrix.
- Transformed YOLO detections in the format 'yoloutput_0001.mat', etc, where each file contains the transformed dictionary/struct YOLO data with the new coordinates, exactly in the same format as the input:

    - Bounding box: ‘xyxy’ with a Mx4 matrix in which each line corresponds to a detection and contains pixel coordinates, [x_blc, y_blc, x_trc, y_trc].

      (blc - bottom left corner, trc - top right corner)

    - Object ID: ‘id’, with a vector containing the id of all M objects.
    - Object class: ‘class’, with a vector containing the class of all M objects.
- (OPTIONAL) Transformed images in the format ‘output_0001.jpg’, etc.

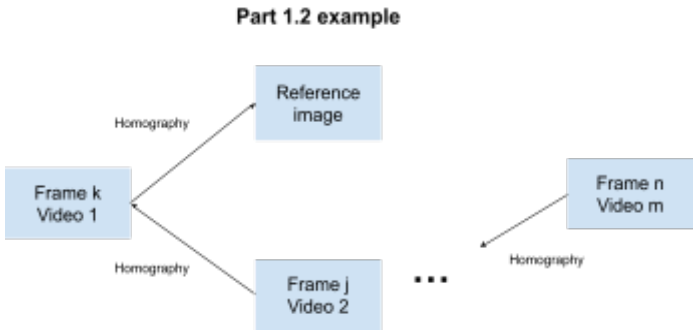
FINAL DEFINITIONS FOR 1.2

1.2 - Fusing multiple sources: Processing images from a moving camera or from two or more videos from different cameras viewing the same scene.

You will need to implement your code (in Python or Octave) in a file ‘main.py’ / ‘main.m’. Your implementation must not require opencv or any other computer vision libraries or packages, since these will not be available in the evaluation environment.

The task in part1.2 is to map the coordinates of all images in the videos to one 2D reference frame:

- For each video you'll be given the output of YOLO
- It is assumed that in each video, consecutive images overlap.
- You'll be given the keypoints of each frame and their respective descriptor
- The reference frame overlaps with at least one of the frames of at least one of the different videos and/or there is an overlap between some frames of different videos. In other words there is always a path between any frame of any video and the reference frame. Below one example where frame k of video 1 overlaps with the reference image and other frames of the other video overlap with each other



FINAL **Specifications for Part 1.2**:

Your code must include one file main.py and the project runs through the command

python main.py ref_dir input1_dir output1_dir input2_dir output2_dir … inputn_dir outputn_dir

- ref_dir **-** path for the directory of the data for the reference image
- inputK_dir - path to the directory of input image sequence K
- outputK_dir - path to the directory where your program will store output data

The data for the reference image and each of the videos is exactly the same format as in 1.1, that is:

INPUT DATA

In directory ref_dir the input data available is:

- One image (the reference image) in the format **'img_ref.jpg'**
- The keypoints and corresponding descriptors store in a file named **kp_ref.mat** where a dictionary/struct containing the following data:
    - 'kp': A Mx2 matrix where M represents the amount of keypoints detected for that image. Each row has the coordinates of 1 keypoint.
    - 'desc': A MxD matrix where M represents the amount of keypoints detected for that image and D is the length of the descriptor (ex. SIFT is 128, SURF is 64). Each row has the descriptor for each keypoint.

In each of the inputK_dir directories the input data available is:

- A video/sequence of images in the format 'img_0001.jpg', 'img_0002.jpg', etc.
- YOLO detections in the format 'yolo_0001.mat', etc each with a dictionary/struct that includes:
    - Bounding box: 'xyxy' with a Mx4 matrix in which each line corresponds to a detection and contains pixel coordinates, [x_blc, y_blc, x_trc, y_trc].

(blc - bottom left corner, trc - top right corner)

    - Object ID: ‘id’, with a vector containing the id of all M objects.
    - Object class: ‘class’, with a vector containing the class of all M objects.
- The keypoints and corresponding descriptors will be stored in a file for each frame (eg, ‘kp_0001.mat’ for), which has a dictionary/struct containing:
    - ‘kp’: A Mx2 matrix where M represents the amount of keypoints detected for that image. Each row has the coordinates of 1 keypoint.
    - ‘desc’: A MxD matrix where M represents the amount of keypoints detected for that image and D is the length of the descriptor (ex. SIFT is 128, SURF is 64). Each row has the descriptor for each keypoint.

NOTE: If no objects were detected in some frame there will be no YOLO file for that frame.

OUTPUT DATA

- In each directory of each video you should create a file **homographies.mat** with a dictionary {“H”:H} where H is a (3,3,Nv) array where Nv is the number of frames in that video and each element H(:,:,i) is the homography of frame i to the reference frame.
- Transformed YOLO detections in the format ‘yolooutput_0001.mat’, etc, where each file contains the transformed dictionary/struct YOLO data with the new coordinates, exactly in the same format as the input:
    - Bounding box: ‘xyxy’ with a Mx4 matrix in which each line corresponds to a detection and contains pixel coordinates, [x_blc, y_blc, x_trc, y_trc].

      (blc - bottom left corner, trc - top right corner)

    - Object ID: ‘id’, with a vector containing the id of all M objects.
    - Object class: ‘class’, with a vector containing the class of all M objects.

DATASETS:

You are encouraged to create your own datasets and use opencv or any other tool to detect feature points, track objects or visualization.

OUR DATASETS ARE AVAILABLE HERE:

https://drive.google.com/drive/u/0/folders/1_lWREinR5j9MvpCwMUoTHn4jeJwPitco

Step 2: Procrustes Analysis and Point Cloud Registration (preliminary)

The second step aims at reconstructing the whole scene using images from an instrumented vehicle. Using state-of-the-art 3D reconstruction and 3D point cloud registration techniques applied to this new set of images, the information gathered in the first step may be mapped on the virtual model of the intersection or city block. This way we cease to require the expensive yet trivial services provided by Google.

Datasets available at:

- https://drive.sipg.tecnico.ulisboa.pt/s/4bzxRSCLX6L2Zmw
- Google Drive

Further details:

- You'll be given a sequence of images and corresponding depth maps. Additionally, you will be given keypoints and the descriptors of each image that have the same format as ‘kp.mat’ from part 1.2 .
- You’ll need to estimate the (rigid / similarity / affine) transformation between point clouds and then merge all point clouds into a single one.

Specifications:

The input file will ‘cams_info.mat’, which has a dictionary/struct with N entries (one for each image), each containing:

    - ‘focal_lenght’: camera intrinsics (focal length ‘fx’, ‘fy’ with principal point ‘(cx,cy)’ at the center of the image plane)
    - ‘rgb’: RGB image
    - ‘depth’: depth map
    - ‘conf’: confidence map

The output of your code must be saved in the current directory and must include:

- Merged pointcloud in a file ‘output.mat’.
- Transformations for each pointcloud, in a ‘transforms.mat’, containing the dictionary/struct with N entries (one for each pointcloud), with ‘R’ 3x4 rotation matrix and ‘T’ 3x1 translation vector.

Submitting your Project

Your code must be submitted with a 4 page (max) report by Tuesday January 7th, 2025.

The report must be submitted in PDF to a folder in the pivhub.sipg.tecnico.ulisboa.pt account for your group.

The code must be submitted also to the same server in specific folders to be defined soon.