PIV Project 2024

Reverse Engineering Intersections and Crossings

As the title suggests, the 2024-25 PIV project will unfold in an urban traffic setting and the goal is to develop computer vision tools to observe pedestrians and driver's behavior. Specifically, we will look at intersections and collect data that will help analyze the features and merits of intersection design. Of course we will focus on the technology and the data produced by it.

The data will be generated by processing videos and image sequences and applying basic geometric concepts that allow the extraction of metric information from videos (positions and velocities).

Evidently, this traffic 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. We focus on a specific scenario to better illustrate the power of the techniques and because it will require the use of some tools (deep learning-based detectors and trackers) that students of previous PIV editions expressed they would like to have used.

Note:this project may include a voluntary collaboration with a team of students of the master´s in transportation form the University of Cape Town in South Africa (more on this later).

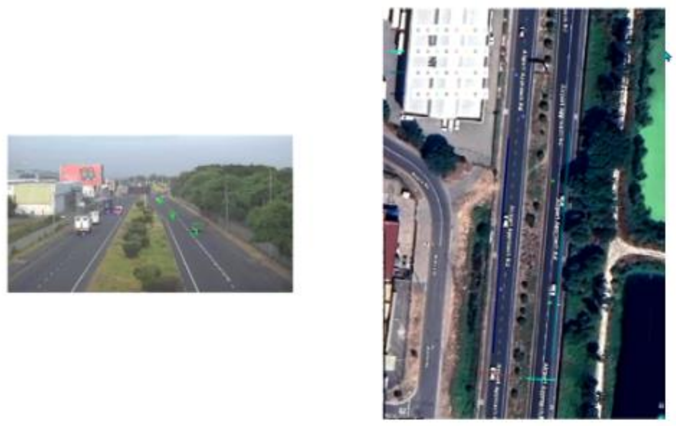
A project at two levels

The project will unfold in two main steps corresponding to two key topics: 2D and 3D processing. The first step comprises the subjects that we will address in the first 3 weeks and the second step the subsequent topics.

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

  Alternatively, we will serve you a running website where you can get the data (will be shared mid course).

- 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, you must produce the output shown here (the data not the graphics)

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_00001.jpg’, ‘img_00002.jpg’, etc.
- Depth maps for each image in the format ‘depth_00001.mat’, etc. You can assume the intrinsic parameter matrix:

  $$K = \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

  where $\begin{pmatrix} c_x & c_y \end{pmatrix}$ are the coordinates of the central pixel in the image.

- YOLO detections in the sequential format ‘yolo_00001.mat’…’yolo_*****.mat’ 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.

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

- Transformed images in the format ‘output_00001.jpg’, etc.
- Transformation matrices in the format ‘output_00001.mat’, etc, (using scipy.io.savemat in python), each file containing the dictionary/struct {‘H’: H}, where H is the 3x3 transformation matrix.
- Transformed YOLO detections in the format ‘yolooutput_00001.mat’, etc, where each file contains the updated dictionary/struct {‘bbox’: B} with the new coordinates.

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

Assuming you completed part 1.1, the next step is to fuse information of 2 videos where images overlap in certain regions. The task is to map the coordinates of both videos to

Google Maps.

Further details:

- You'll be given the same data as in Part 1.1, that is, the output of YOLO and at least 4 keypoint matches from one video frame to Google Maps.
- However, the video cameras may move (e.g. cellphones). In other words, while in part 1.1 you can assume the camera is fixed, here both cameras may move. So you must estimate the inter-frame motion and compensate for it. It is assumed that both videos overlap and at least one frame from one video has known points in the Google Maps image (as we say in 1.1).
- You'll be given the 3D position of some image points and optionally the intrinsic parameters of the cameras. With this information you must compute the camera pose (location and orientation of the cameras)

Download this video from https://drive.sipg.tecnico.ulisboa.pt/s/SGjaB99WXQ9cTRo
Additional datasets can be downloaded from: Google Drive

Specifications:

This part has the same specifications as part 1.1. The input information of the second video will be available in the formats ‘img2_00001.jpg’, ‘depth2_00001.mat’, ‘yolo2_00001.mat’, etc, and the same for the outputs.

The keypoint matches between the two videos will be stored in a ‘kp2.mat’ file containing a dictionary/struct with the keypoint matches between all the frames in the first video and all the frames in the second video. The dictionary/struct keys identify the pair of frames (eg, ‘img_00001_img2_00001’ for the first frame in each video) and each contains 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.

Step 2: Procrustes Analysis and Point Cloud Registration

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 matches between pairs of images points - in the regions that overlap.
- 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.

There is one project for everybody

As you may have perceived the project will require significant and continuous work. After all, “continuous work” was the key statement underlying the *new* IST *pedagogical* method (MEPP). Of course reality shows that this is easily written in (paper) rules but much harder to put in practice … particularly in 7 week with theory at full throttle!

That is why the project develops in “layers”:

-Part 1.1 can be done by any student, even without going to classes, discussing with colleagues and faculty, even with meager skills in python or matlab. With the correct prompts, ChatGPT will be happy to do the project for you! Investing a little bit will give you the thrill of accomplishing something very *cool*.

-**Part 1.2** should be the minimum for a good project and it can go up to the top grades

-**Part 2** If you complete the previous steps, this will complement your project with the last topics in 3D depth and shape estimation, 3D geometry and motion estimation.

About grades: The fact that you complete all steps does not assure you any grade level. **In PIV what you do counts as much as HOW you do it** ! We will clarify details later but you have to show that you really understood the concepts and not just mastered software libraries that do everything in a few lines of code… or that you did not learn anything from your colleague GPT-4.

In short, to pass in the project component (as in the exam) there is a minimum effort and minimum set of knowledge that you must show you acquired. The (minimum) project component (part 1.1) should be realizable by all students, you wouldn’t even need a group to do it so we recommend that all members of the group try to do it independently.

Part 1.2 is well specified but there are some options that you should thing about and decide, but part 2 is “underspecified”. Think of ideas and proposals on what to do with that data.

You’ll have 5 “top notch” Computer Vision people helping you!

Gustavo Vitor, Manuel Ferreira, Gabriel Moreira, Carlos Santiago, João P. Costeira ...
and Gil Beirão in the backoffice !