Project Final Description

Before you start writing code

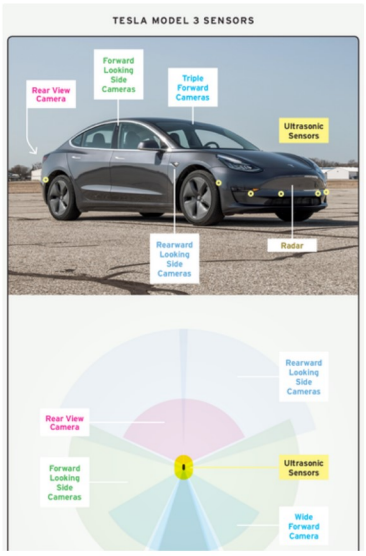
- Read carefully the document and make sure you understand how the project will unfold
- The project has one part fully specified and another part that require decisions and options from you and your group. In computer vision there are no "Maxwell Equations" , in other words, it is not a classic discipline with fundamental laws. Instead, usually it boils down to computing variables of interest from images... and an image is "many things".
- There is manifold ways of computing "stuff" from images so make sure you understand what is that you want to compute and what are the techniques. Don't just copy code from GitHub ! You must study and read the book...at least.
- Start working early in the project. Most tasks have open source available, you can process the images and get the data you need.
- Parts that are more difficult (geometry , estimation) will require basic knowledge on linear algebra...don't overlook or try to do it without diving into matrices and calculus... and understand the definitions (projective coordinates, homography, outliers, least-squares)

Let's roll !

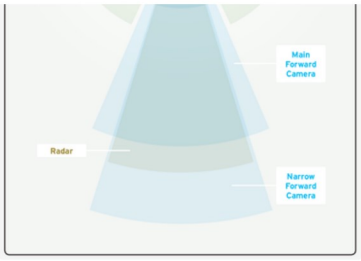
Motivation

You may be familiar with the "*computer vision platform*" also known as the **Tesla EV.** The Tesla vehicle is, basically, a glass, plastic and aluminum 4-wheel-frame, holding 2 tons of batteries to feed an electric motor and NVidia GPU's to process images from 8 cameras.

On the side figure we show one exemplar of the platform and the diagram of the cameras' viewing areas. Never mind the ultrasonic sensors and radar, all driving information and driver's assistance is obtained from the



cameras. A mistake as you will experience first hand !



Unfortunately such wonderful camera rig is not *open source* and the manufacturer just let us access 4 of the 8 cameras and in "batch/offline" mode. The video below shows one example of the output provided by the Tesla cameras.

0:00 / 4:2

As you can easily figure out, PIV2023's project will develop around the videos collected from this expensive gadget, which will demonstrate, and hopefully will convince you, that the computer vision foundations taught in class are useful to develop the powerful technologies used today and indispensable to understand them.

Some fundamental tasks in the project that map to specific topics from the syllabus :

- **Single camera calibration** - how to compute the camera model from data
- **Multiple camera calibration** - how to determine the relations between cameras (location and orientation)
- **Homographies -** How to map coordinates from sequences of images and

fuse all images into one single panoramic image with or without metric measurements. In other words, how google maps and street view work !

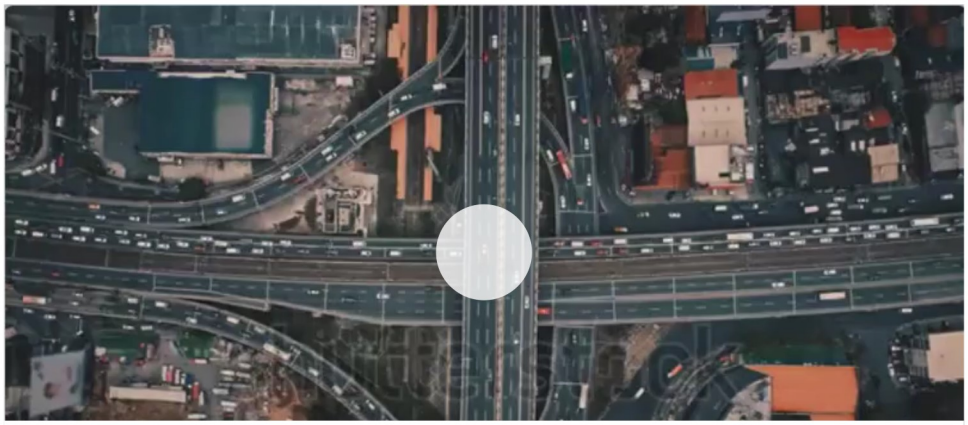
- **Stereo & SfM -** How to compute 3D structure from images combining either 2 images from different cameras or multiple images along time...or both. On top of this data the problem of point cloud registration emerges.
- Object **detection**, region **segmentation** (road, buildings,sky) and **optical flow**. Top performers are based in "Deep Learning" but mathematical models are fundamental to understand and interpret the images.

Three Levels of Complexity (work!)

This general description comprises tasks in three levels of complexity:

Given the stream of images you must compute the transformations between all images so that pixel coordinates are referred in a single reference frame. Imagine a drone with a camera facing down as illustrated below. Intuitively, there must be a way to reference these images relative to, say, the coordinates in google maps! In fact this will be the first processing you should do. In short, the three main steps are the following:

1. Uncalibrated Registration (Homography) :
    a. Given the stream of images from one camera you must compute the transformations between any image and a reference image (map). Your first job is to register each image to one specific image (given)...for example, images could be registered to google maps! Note that images may have strong perspective rather than this "almost only rotation and zoom" type of motion.



0:00 / 0:23

    b. Given two or more streams of images from two or more cameras

complete the same task as before. Essentially this adds extra data, implying extra transformations between each camera and between these and this way there are multiple ways of computing the desired transformations (detail ahead).

2. Calibrated Registration: Given the intrinsics of the cameras do the same process and extract important "metric" information relative camera orientation like position and orientation. It will be defined soon but in the Tesla data you must be able to compute the relative orientation/position of the 4 cameras.
3. Be creative with 3D processing (SLAM/SfM): The multiple cameras allow a more precise and robust estimation of motion or scene reconstruction from stereo or structure-from~motion. Using any available software (eg. Colmap , DeepXXX, Nerfs) either produce the 3D maps of the world or self-localization (or suggest an application). This part is conditioned on completion of the previous by the 5th week.

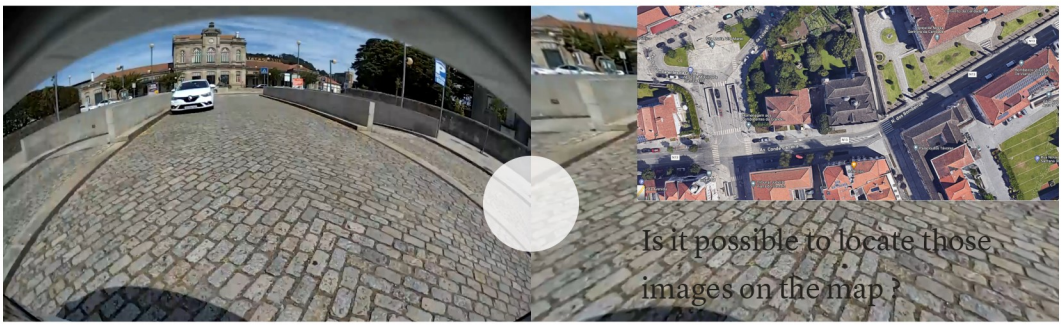
For each level, a protocol will be defined (function names and input/output formats), in other words, we will specify the inputs and outputs of python/Matlab functions so that we can run your code autonomously.

We will be evaluating what is reasonable to do in (3) since just for topic 1 you'll need 3 weeks of classes !

Sample data for some of the steps

Calibrate and correct distorted images (pinhole model). Below a matlab .mat file with the back camera parameters (readable with loadmat method from scipy). The right image is computed by the opencv function to remove radial distortion from images (to be explained in class).

back_camera_pinhole.mat 36.1KB



0:00 / 1:00

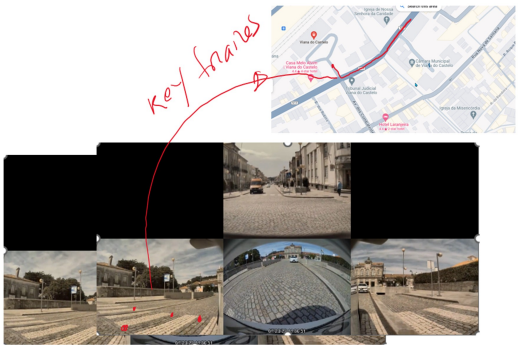
Bird's eye view of the several parts

Part 1

Input data : one/multiple video(s) of a moving vehicle

Output data: localization on a (given) map

Input parameters: coordinates of pairs of points between the map and a few keyframes



Part 2

Input data : one/multiple video(s) of a moving vehicle and calibration parameters

Output data: localization on a (given) map

Input parameters: coordinates of pairs of points between the map and a few keyframes. segmentation of the map (where you are supposed to be).

Part 3

Input data : one/multiple video(s) of a moving vehicle and calibration parameters

Output data: localization on a (given) map, "rig" calibration , possibly 3D locations ...something you came up with.

Input parameters: coordinates of pairs of points between the map and a few keyframes, segmentation of the map... anything else you require and is reasonably

Top grades (17-20) must have something in part 2 and/or part 3

In week 4 we will monitor project development and will decide if group is ok to do P2/P3

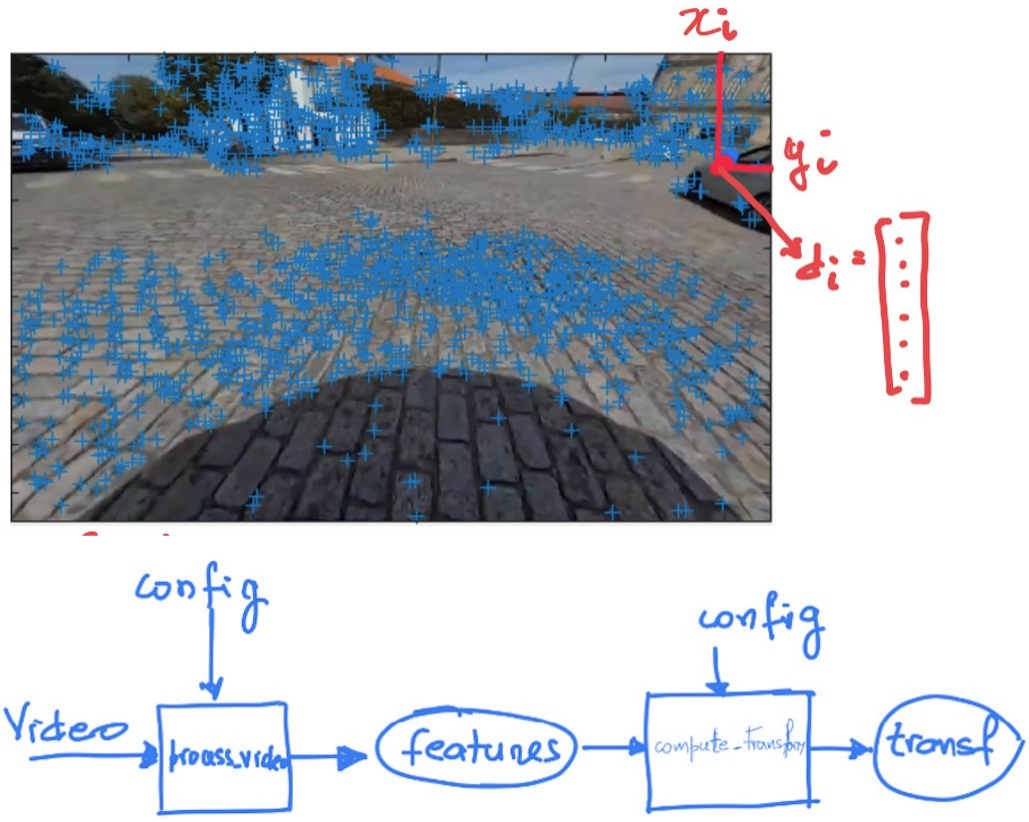
Detailed Specifications

Part 1 : Submit two python/matlab functions:

1. run command: python process_video.py conf_file.cfg
2. run command: python compute_transforms.py conf_file.cfg

The configuration file is the same and will contain all parameters for both programs (to be defined ahead).

Script process_video.py should do the video processing (image processing, feature extraction) to compute keypoints. The input is a configuration file and the output is a **.mat** file with **the keypoints** (x,y,descriptor).



process_video.py(.m) : As described in the figure the output file (its name is defined in config) containing a variable named "features". In matlab format is a cellarray with length "number of frames" and each element is a nxm matrix where each column is the concatenation of the x,y coordinates of each feature point and its descriptor d (128 if sift, 64 if surf ...).

In python you can check the format loading the example file surf_features.mat

```
f=loadmat('surf_features.mat') feat=f['features'] >>> type(feat)
<class 'numpy.ndarray'> >>> feat.shape (1, 41) >>> feat[0,2].shape
(66, 2949) 2949 keypoints each keypoint length=66 (xi,yi,di) di- 64
elements(SURF)
```

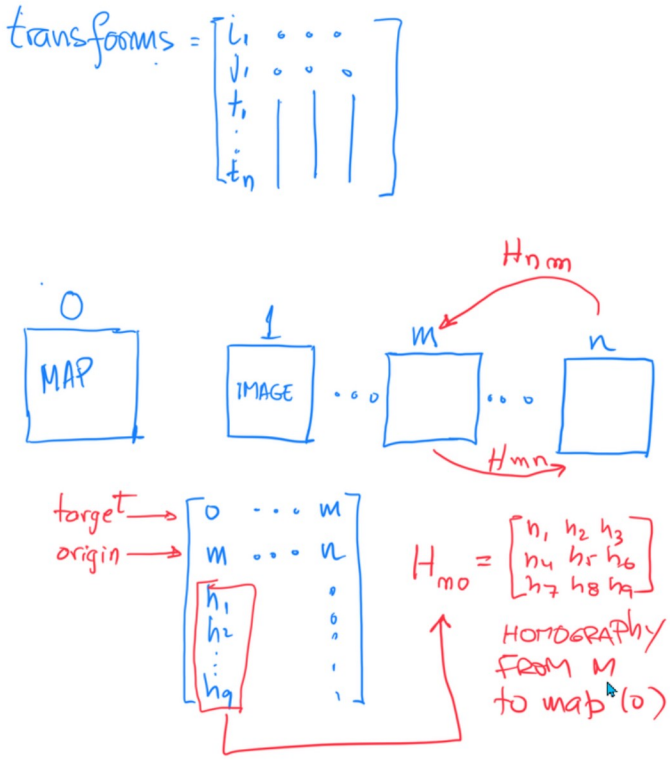
In matlab

```
>> whos features
  Name          Size                Bytes  Class    Attributes

  features      1x41             13458592  cell

>> d=features{1}
>> whos d
  Name        Size                 Bytes  Class     Attributes

  d         66x2623               692472  single
```

compute_transform.py(.m):

Input is the same configuration file and this program must process the keypoint file and generate as output a file with the transformations. In detail the output data has the form illustrated in the figure. The file which name is defined in the configuration file, should contain one variable called “transforms” that packs all transformations required (see configuration definition).

Note that index 0 refers to the map. Other indices (1...N) refer to the images in the video.



output file data format

In summary, for Part1 your main job is to compute the transformations from the data (keypoints).

Libraries and code guidelines for Part 1

process_video.py(.m) : You can import (python) or use (matlab) any toolbox or package you like to compute the keypoints as long as you follow the output guidelines.

compute_transform.py(.m): You can only import numpy, scipy libraries to implement the transformation computation. Exceptionally you can also use scikit-learn for the matching (nearest-neighbour) or any other toolbox required to implement the matching. For matlab, you can not use any toolbox (namely the computer vision toolbox).

Configuration file for PART 1

The format of the configuration file is the following:

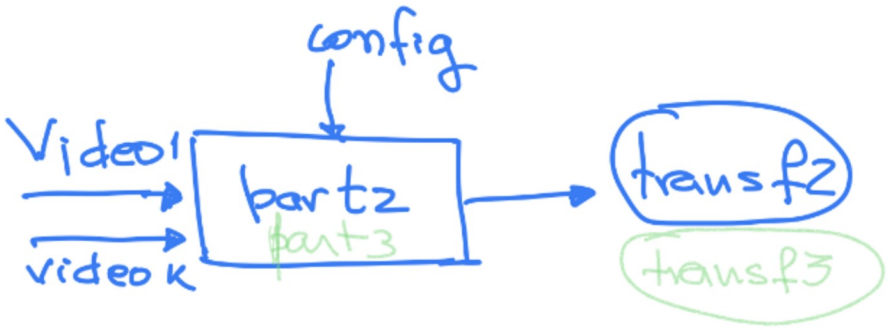
```
#IConfiguration file for PART1 #INPUT PARAMETERS #path to videofile
videos path/videoname.mp4 # Correspondences from points in the map
and points in a frame (first frame is 1) # there is a fixed label
(pixel) pts_in_map pixel xmap1 ymap1 xmap2 ymap2 xmap3 ymap3 xmap4
ymap4 … xmapN ymapN pts_in_frame framenumber x1 y1 x2 y2 x3 y3 x4 y4
… xN yN #another correspondence with another frame pts_in_map pixel
xmap1 ymap1 xmap2 ymap2 xmap3 ymap3 xmap4 ymap4 … xmapN ymapN
pts_in_frame frame2# x1 y1 x2 y2 x3 y3 x4 y4 … xN yN #optional -
image with map filename is a jpg or png file with the map. image_map
filename #This will be updated if need be. For example points in the
map may be expressed in geocode or meters.
#---------------------------------------------------------------------
#OUTPUT PARAMETERS #path of the file where you will save the
keypoints (features) process_video.py keypoints_out
path/file_for_keypoints.ext #required output. For part 1 (one video
only) type = homography and params = {all, map } transforms
homography all/map # if all must return all homographies between
images (Hij, i=1:N-1,j>i) # if map must return homographies from
image i (i=1:N) to the map (i=0) transforms_out
path/file_for_transforms.ext
```

Code and Configuration file for PART 2

For part 2 you are required to have only one script and you can use anything you wish, that is, opencv or any python/matlab packages or toolboxes. Your code should run as follows:

command to run: python part2.py config.

As in part1, the output will be a file with transformations. In this case the type of transformations will vary and will be added later on. For now consider the output are the homographies between images of all cameras and map or between them.



The configuration file has the following specification :

```
#IConfiguration file for PART2 #NPUT PRAMETERS #number of cameras
(input videos) cams integer #path to videofiles. There should be as
many files as numofcams videos path1/video1name.mp4
path2/video2name.mp4 … pathN/videoNname.mp4 #Intrinsics of cameras.
This is optional (ex: part 1 does not have intrinsics as input)
#Images have been corrected of radial distortion. If needed we will
change this to include #radial distortion #There should be as many
lines with Intrinsics as there are video files intrinsics Kx1 Ky1 Cx1
Cy1 intrinsics Kx2 Ky2 Cx2 Cy2 .... intrinsics KxN KyN CxN CyN #
Correspondences from points in the map and points in a frame (first
frame is 1) pts_in_map pixel xmap1 ymap1 xmap2 ymap2 xmap3 ymap3
xmap4 ymap4 … xmapN ymapN pts_in_frame frame1# x1 y1 x2 y2 x3 y3 x4
y4 … xN yN #another frame pts_in_map pixel xmap1 ymap1 xmap2 ymap2
xmap3 ymap3 xmap4 ymap4 … xmapN ymapN pts_in_frame frame2# x1 y1 x2
y2 x3 y3 x4 y4 … xN yN #optional - image with map filename is a jpg
or png file with the map. image_map filename #This will be updated if
need be. For example points in the map may be expressed in geocode or
meters.
#-------------------------------------------------------------------
#OUTPUT PARAMETERS #required output #for part 2,3 we may have type =
{homography, rigid,calibration} params={map,all,?} transforms type
params #Output file name (the output format is temporary). For now is
similar to part1 transforms_out path/file_for_transforms.ext
```

Parsing the configuration file

You do not have to use this suggestion but below we include one possible parsing of the configuration file into a dictionary with all parameters. All data is string but numeric parameters should be converted to float. Customize to your convenience.

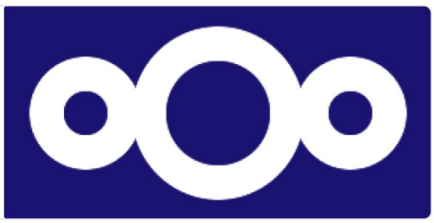
```
def parse_config_file(file_path): config_dict = {} with
open(file_path, 'r') as file: for line in file: line = line.strip()
# Ignore comments if line.startswith('#'): continue # Split the line
into tokens tokens = line.split() # Extract parameter names and
values param_name = tokens[0] param_values = [tokens[1:]] # Check if
the token already exists in the dictionary if param_name in
config_dict: # Add new values to the existing token
config_dict[param_name].extend(param_values) else: # Create a new
entry in the dictionary config_dict[param_name] = param_values
return config_dict
```

Datasets

Shared

Drive do SIPg - A alternativa possível à loucura mansa da UL

https://drive.sipg.tecnico.ulisboa.pt/s/piv?path=...



Check Specs Folder