Project: Small target tracking in satellite images

Due: Friday, 20th of May 2022, 4pm (NO EXTENSIONS)

Students are free to use either Matlab or Python and OpenCV, provided that all deliverables are provided

Grouping

Students have to work in groups of three to complete the project.

Groups should have been formed. The list can be found in the “Project Group Members 2022” forum.

Timeline

Week 7 to Week 12: Work on your project. You have 6 weeks. The tutor will be available in the labs or online during certain lab hours for questions (time to be confirmed on LMS under announcements).

Week 13 (23-27 May 2022): Demonstrations of your results. Times will be announced at a later stage.

Friday 20th of May 2022 4:00PM: Final deadline to submit your project on LMS.

The project

Small target detection in satellite images is of utter importance in many applications, including but not limited to urban planification, military and economic intelligence and land management. In this project, you will put yourselves in the shoes of a computer vision engineer team who has been tasked to develop an algorithm which to detect moving cars from satellite images.

Data preparation

For this project, we will use the VISO public dataset [https://github.com/The-Learning-And-Vision-Atelier-LAVA/VISO] to train and validate our model. Your first task is to download the data and write a parser to be able to use this data. The specifications required are:

- The parser should be able to read a sequences of images. Given a folder, a template for the image file names and frame range, frames can then be queried on demand.

- The parser should be able to read the definition of the regions where small objects have been labelled and return them as matlab or numpy arrays.

- The parser should not clog the ram of the computer, if the sequence is too large, each frame should be loaded only when queried.

Be aware that the 4 main folders in the VISO dataset are, in fine, the same data, which was organised in different ways. You need to use the “mot” (multiple objects tracking) folder for this project. The gt.txt files associated to each sequence contain an array in csv format with the first column representing the frame, the second column the track id, the third to sixth column the rectangle x, y top left coordinates, width and height in pixels. The remaining columns encode other arguments of the data and they should be ignored in this project.

We do recommend you use matlab object oriented syntax [https://au.mathworks.com/company/newsletters/articles/introduction-to-object-oriented-programming-in-matlab.html] to create your data loader. The class should be constructed with a given folder, and image name template and frame range. Ideally the frame range argument should be faculative, with the range automatically derived from the files in the folder. The class should then provide a method to get the actual frame range and a method to load a given frame from a frame index. You are allowed and encouraged to use convenience functions, like csvread [https://au.mathworks.com/help/matlab/ref/csvread.html] or pandas.read_csv if you are using python [https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html], to automatically parse the file containing the ground truth of small object tracks.

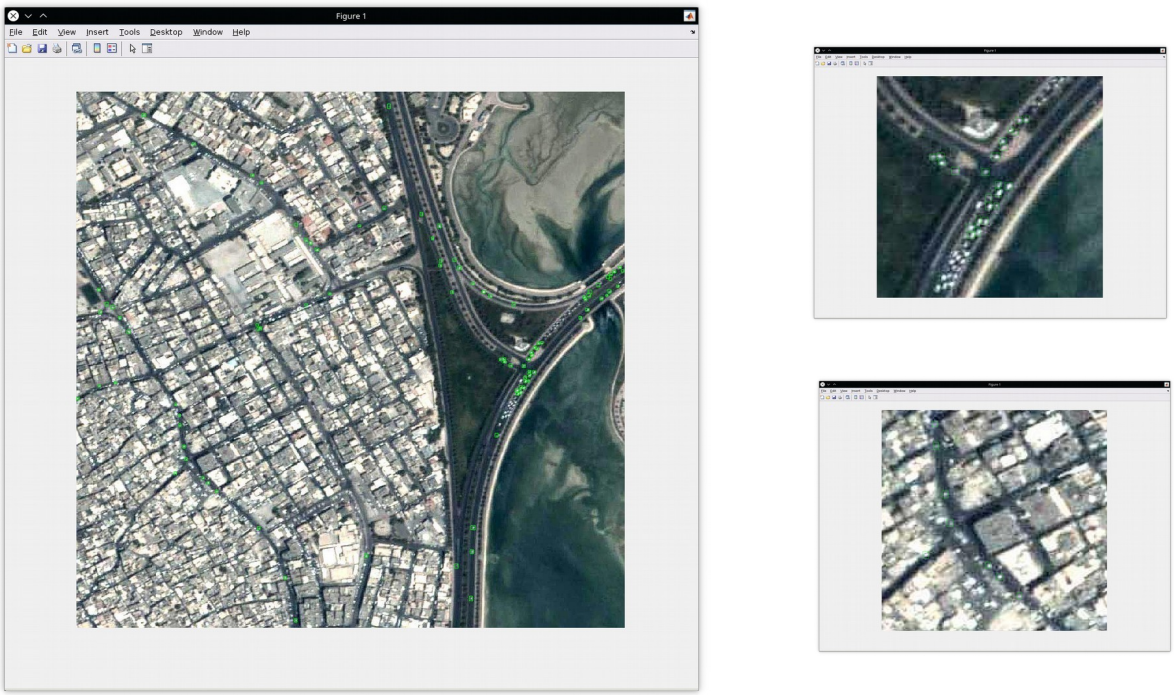

Figure 1: Output of the dataloader visualized using the imshow and rectangle plotting functions.

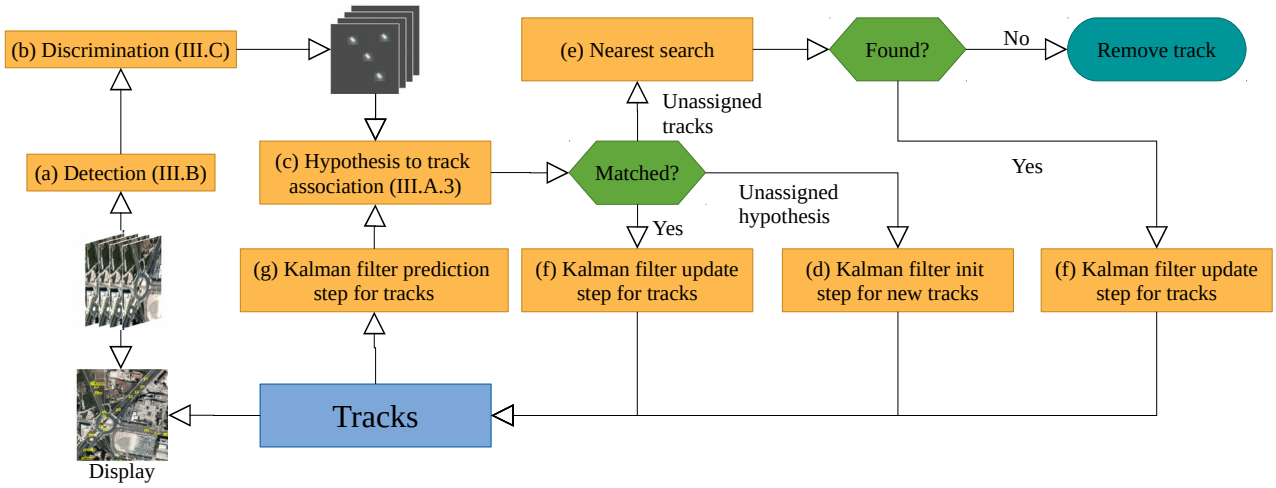

Figure 2: The workflow of the algorithm in the "Neelde in Haystack" paper. The relevant sections in the paper are indicated by the corresponding block. Please refer to the corresponding sections of this document for more details on how we expect you to implement those algorithms.

Listing directories and parsing filenames

To implement your dataloader, you will need to list files in a directory and parse their filenames. In Matlab, you can read the content of a directory by using the dir function [https://au.mathworks.com/help/matlab/ref/dir.html]. The function return a struct array containing information about each entry in the directory, including the filename. In Python, use the function os.listdir [https://docs.python.org/3/library/os.html#os.listdir], which return the list of files in the directory as a list of strings, or os.scandir [https://docs.python.org/3/library/os.html#os.scandir], which return an iterator of direntries objects.

The algorithm

Object tracking is a computer vision task where objects are automatically detected and then their trajectory is followed over multiple frames.

For this project, you will need to implement the method described in the paper “Needles in a Haystack: Tracking City-Scale Moving Vehicles From Continuously Moving Satellite” by Wei Ao et al. [https://ieeexplore.ieee.org/abstract/document/8861293] (a copy of the paper is available to on LMS under “Project”).

The algorithm is composed of two parts: the initial detection of candidate small objects, followed by their tracking over multiple frames.

Each step will be described in more details in the following sections.

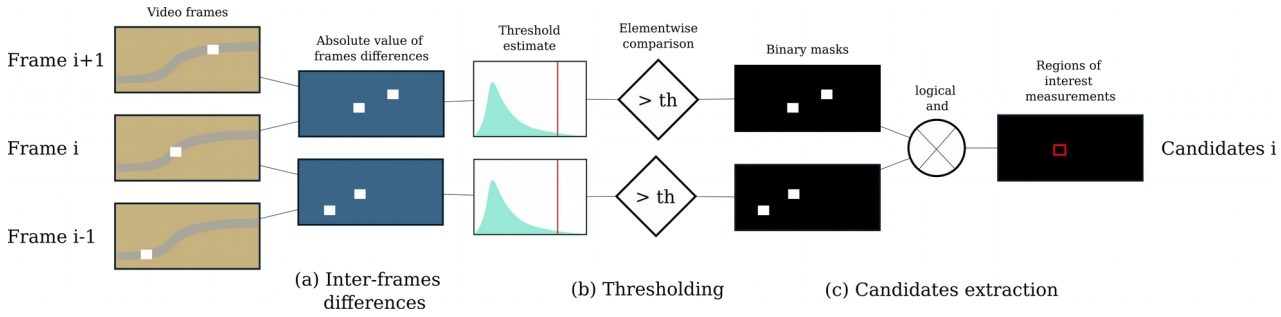

Figure 3: The detailled workflow for candidate objects detection.

Candidate small objects detection (Figure 2.a)

The first step of the algorithm is do detect small moving objects. To achieve this goal, you will haveto implement the Motion-Based Detection Using Local Noise Modeling algorithm described in section III.B of the “Needles in a Haystack” paper. Figure 3 details the procedure.

Inputs: for each frame index n from 1 to N-1, this step takes as input the frames at index n-1, n and n+1

Ouput: for each frame index n from 1 to N-1, this step outputs a binary image representing candidate small objects.

To make the algorithm faster, the image is split into regions of 30x30 pixels, where the algorithm is applied. The algorithm itself has three parts:

- First, **Inter-frames differences** (Figure 3 a) : difference images are computed for the frames n-1 and n, as well as n and n+1 to get two inter frame differences. To simplify the thresholding step, the absolute value of the differences are taken. (See Equations 7 and 8 in the “Needles in a Haystack” paper)

- Second, **thresholding** (Figure 3 b) : outlier difference pixels are detected by fitting an exponential distribution to the absolute differences. The probability distribution function of an exponential distribution is $\lambda e^{-\lambda x}$. The only parameter to estimate to fit the exponential distribution to the datais λ. Remember that the maximum likelihood estimator for λ given a series of absolute valued differences D is:

$$\lambda = \frac{1}{mean(D)} \quad .$$

The cumulative distribution function of the exponential distribution is then given by $1-e^{-\lambda x}$, which can be inverted to give us the threshold th to detect small moving objects based on the following equation:

$$th = \frac{-\ln(p_{fa})}{\lambda} \quad ,$$

where the limit probability for thresholding p(i.e., the probability point above the threshold is still an inlier) is set to 5%. All pixels with a difference above the threshold are considered as outliers andmarked as being part of a moving object. All other pixels are marked as being part of the background.

- The final step is the **candidates extractions** (Figure 3 c) : the thresholded binary mask for the frames n-1 and n is combined with the binary mask from the frames n and n+1 using an element-wise logical and operation. The candidates small moving objects are then represented by small pixel clusters which are white in the final binary image which can then be used to extract informations about the candidate, like the bounding box or morphological cues for filtering (see Candidate match discrimination).

Candidate match discrimination (Figure 2.b)

The next step is to clean up the candidate moving objects to remove incorrect matches caused by imaging noise or small motion of the satellite. To this end, you will use the method proposed by Wei Ao et al. in section III.C of the “Needles in a Haystack” paper.

Inputs: for each frame index n from 1 to N-1, this step takes as input a binary image representing the candidate small objects.

Ouput: for each frame index n from 1 to N-1, this step outputs the bounding box and centroid of each candidate small object.

Based on the similarity between detected and undetected pixels, the first step is to grow each detected region. The goal of this procedure is to recapture pixels that went undetected because of the overlap between objects between frame due to small motion. Once this is done, false positives can be detected using a series of morphological cues. The paper proposes 4 of them: area, extent, major axis length, and eccentricity (see below).

Region growing

The region growing procedure is described in section III.C.1 of the “Needles in a Haystack” paper. The algorithm first fits a 11x11 search windows centered on the centroid, i.e., the mean position of the pixels, of the current candidate cluster. The mean and standard deviation of pixel values within the 11x11 search windows are then computed and pixels that are in the (0.5%, 99.5%) quantile interval are then marked as being part of the candidate cluster as well. The formula to compute the quantile of the normal distribution in matlab is norminv [https://au.mathworks.com/help/stats/norminv.html]. In python, you can use the scipy.stats.norm.ppf function [https://docs.scipy.org/doc/scipy-1.8.0/html-scipyorg/reference/generated/scipy.stats.norm.html].

Morphological cues

The area cue is just the area of the cluster of pixels forming one candidate.

The extent is the ratio of the area, to the area of the candidate pixel cluster’s bounding box.

The major axis is the length of the major axis of the ellipse with the same normalized second centralmoments of the connected region.

The eccentricity is the eccentricity of the ellipse with the same normalized second central moments of the connected region. The eccentricity of an ellipse if given by $\sqrt{1-b^2/a^2}$ where b is the lengthof the minor axis and a the length of the major axis.

The major axis and eccentricity cues are defined based on “an ellipse with the same normalized second central moments of the connected region”. While this pedantic jargon can be intimidating at first, the underlying assumptions are really simple.

First, you need to create a matrix C with the x and y coordinates of all pixels in the candidate region:

$$C=\begin{bmatrix} x_0 & \dots & x_n \\ y_0 & \dots & y_n \end{bmatrix} .$$

The covariance matrix of the pixels coordinates, which is the normalized second central moments of the connected region, can be computed as:

$$\Sigma_{cc}=\frac{(C-\bar{C})(C-\bar{C})^T}{N-1} ,$$

where $\bar{C}$ is the mean coordinate of the pixels in the cluster and N is the number of pixels in the cluster. This covariance matrix can then be diagonalized:

$$\Sigma_{CC}=P^{-1}\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}P .$$

The major axis is then given by $max(\sqrt{\lambda_1},\sqrt{\lambda_2})$ and the eccentricity by $\sqrt{1-min(\lambda_1,\lambda_2)/max(\lambda_1,\lambda_2)}$. You can use the eig function of matlab to get the eigenvalues (λ_1 and λ_2) [https://au.mathworks.com/help/matlab/ref/eig.html]. In python, you can use numpy.linalg.eig for the same purpose [https://numpy.org/doc/1.18/reference/generated/numpy.linalg.eig.html].

You should then, for each morphological cue, use an interval with an upper and lower threshold.

Any shape outside of those intervals should be rejected as wrong detection.

Each upper and lower threshold for the morphological cues needs to be properly calibrated. To this end, you need to plot their distribution for both hypotheses that are matched to a ground truth object (see the evaluation section below to see how we see if a hypothesis is matched to an object), and hypotheses that are not matched, to choose your lower and upper thresholds.

For all the remaining regions, you then need to compute a bounding box and centroid for each pixel cluster to pass down to the next step in the algorithm, which is the tracking loop.

All the required measurements can be extracted from the binary images by using the blobanalysis class in the matlab vision toolbox [https://au.mathworks.com/help/vision/ref/vision.blobanalysis-system-object.html]. In python, you can use skimage.measure.label [https://scikit-image.org/docs/stable/api/skimage.measure.html#skimage.measure.label] and skimage.measure.regionprops from the skimage package

[https://scikit-image.org/docs/stable/api/skimage.measure.html#skimage.measure.regionprops] for the same purpose.

Tracking (Figure 2.c-g)

The tracking phase, which follows the moving objects is done using a Kalman filter. The Kalman filter is a tool used in computer vision, robotic, aeronautic, …, which smooths the output of a noisy sensor by combining the measurements of the sensor with a motion model. The approach is presented in Section III.A.(5) of the paper, but in little detail. We provide you, in the following section, more details on how you can implement the algorithm for your project. If you are interested you can also read the wikipedia article on the Kalman filter [https://en.wikipedia.org/wiki/Kalman_filter#Details]

Input: for each frame index n from 1 to N-1, this step takes as input a binary image representing candidate small objects, as well as the state of the tracker (the Kalman state vectors for each tracks, and the corresponding covariances estimates) from the previous frame.

Ouput: a series of tracks, each made up of a Kalman state vector representing the position, speed and acceleration of the tracked small objects.

The tracking algorithm is divided into multiple subparts, organized in a loop (see Figure 2 on page 3). First, the hypothesis from the previous steps are matched to the current track that is estimated by the algorithm. For the hypotheses without a track, new tracks should be created and initialized, tracks without an assigned hypothesis are stopped and discarded and track which are matched to a hypothesis undergo an iteration of the Kalman filter.

The Kalman filter is made up of two steps: The first step, which is performed before the track is assigned to a new hypothesis, is to use a motion model to predict the next position of the track. The second step, is to use a measure of the state of the track combined with the motion prediction to get the filtered state of the track.

Motion and observation model

At the core of the Kalman filter, are the motion and observation model. To each current track is attached a state vector x_i . Since the motion model used in the current paper assumes constant acceleration in a straight line, the state vector should store the x and y positions (which represent the centroid of the tracked object), speed and acceleration of the tracked object:

$$x_i = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix}$$

The next frame, assuming the time between frames is equal to τ, is given by:

$$x_{i+1} = F_k x_i + noise = \begin{bmatrix} 1 & 0 & \tau & 0 & \tau^2/2 & 0 \\ 0 & 1 & 0 & \tau & 0 & \tau^2/2 \\ 0 & 0 & 1 & 0 & \tau & 0 \\ 0 & 0 & 0 & 1 & 0 & \tau \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x_i + noise$$

To simplify your computation you can assume that the frames are equidistant in time, which means that we can set $\tau=1$ for all intended purposes.

The observation correspond to the position, which can be extracted from the state vector by a simple matrix multiplication:

$$y_i = H_k x_i + noise = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} x_i + noise$$

An important part of the Kalman filter is also the modelization of the uncertainty that is attached to the motion and observation processes.

We will use normal distribution with mean 0 and diagonal convariances to model the noise associated with the motion and observation models.

The covariance matrix of the motion model, Q_k , is given by:

$$Q_k = \begin{bmatrix} \sigma_p^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_p^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_v^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_v^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_a^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_a^2 \end{bmatrix},$$

and the covariance matrix of the observation model is given by:

$$R_k = \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_d^2 \end{bmatrix},$$

where σ_p is the standard deviation of the position estimate, σ_v of the speed estimate, σ_a of the acceleration estimate and σ_d of the moving object detection algorithm estimate. It is up to you to determine appropriate values for those parameters.

Kalman filter init step for new tracks (Figure 2.d)

Once a new object is detected in the image, a new track has to be initialized. By default its speed and acceleration will be set to 0, while its position will be set to the centroid of the detected pixels patch. The filter also do need to keep track of the covariance matrix of the estimated, written as P_k, which we initialize as equal to Q_k.

Kalman filter prediction step for tracks (Figure 2.f)

Between frames, the track position is updated using the motion model. Two values need to be computed, the predicted (a priori) state vector and the predicted (a priori) estimate covariance. The predicted state vector is given by:

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} \quad ,$$

and the predicted estimate covariance is given by:

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \quad .$$

Hypothesis to track association (Figure 2.c, Figure 2.e)

After the prediction step, because we have multiple tracks the algorithm is following and multiple hypotheses from the hypothesis detection and discrimination module, we need to match each track to the most plausible hypothesis and vice versa. To do this, we will use the euclidean distance as a cost, trying to match each track to the nearest hypothesis and vice versa. Yet, because we have multiple tracks and hypotheses this will not always be possible. To find the optimal match, there exist a dedicated algorithm called the Hungarian method [https://en.wikipedia.org/wiki/Hungarian_algorithm] . The method is quite complicated, especially when we factor in the fact that we will have to consider unmatched tracks and hypotheses as well. But fortunately, Matlab has a version of the algorithm that is already implemented and can be used directly for this step [https://au.mathworks.com/help/vision/ref/assigndetectionstotracks.html]. In python, you can obtain the same results with scipy.optimize.linear_sum_assignment [https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html]. Because scipy.optimize.linear_sum_assignment has no arguments to provide the cost of not assigning a track to an hypothesis and vice versa, you should add pseudo tracks and hypothesis to you cost matrix to represent actually assigning a track or hypothesis to nothing. The cost of assigning a track to such a pseudo hypothesis and a hypothesis to a pseudo track should be set to a positive constant of your choice, while assigning a pseudo track to a pseudo hypothesis should have a cost of zero.

Unassigned tracks, which are likely caused by the tracked small object stopping (i.e. the car is stopping at a traffic light) and thus avoiding detection, will then be forwarded to a local search algorithm that will correlate a square window around the object in the previous frame with a local neighborhood of the previous position in the current frame. The paper gives very little details on how they implemented this step, as such you are free to use any method that you have seen in the lectures (see especially lecture 10 – page 56). Unmatched tracks, due to insufficient matching the search region, will be discarded at that point.

Unassigned hypotheses will be assigned to new tracks that should be initialized at that point.

Kalman filter update step for tracks (Figure 2.g)

After the tracks and hypotheses have been associated with one another, the final step before moving to the next frame is to update the state estimate of the Kalman filter. Given the measured values z_k

(which are the centroids of each clusters in the hypothesis map), the following computations have to be done:

- Compute the error, called innovation, between the measurement and the prediction:

$$\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1} \quad .$$

- Compute the covariance of the innovation:

$$S_k = H_k P_{k|k-1} H_k^T + R_k \quad .$$

- Compute the optimal Kalman gain:

$$K_k = P_{k|k-1} H_k^t S_k^{-1} \quad .$$

- Compute the updated (a posteriori) state estimate:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k \quad .$$

- Finally, compute the updated (a posteriori) state covariance, which should be used in the next iteration of the algorithm:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \quad .$$

Evaluation

For each frame, you need to compare the bounding box of the small objects currently being tracked to the ones of the ground truth small moving objects in the dataset.

To evaluate the quality of the matching between the ground truth regions and the predicted regions, we will use the interesection over union metric [https://en.wikipedia.org/wiki/Jaccard_index]:

$$IOU(A,B) = \frac{A \cap B}{A \cup B} \quad .$$

Any region with an IOU greather than 0.7 are considered as matching. A pair of the proposed and the ground truth region matching together are considered a true positive, a proposed region without a matched ground truth region is a false positive and a ground truth region without a matched proposed region is a false negative. To measure the performance of your model, you need to report the precision, recall and F1 scores of your model, remember that:

$$precision = \frac{TP}{TP+FP} \qquad recall = \frac{TP}{TP+FN} \qquad F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Deliverables

The deliverable should be a matlab or python GUI (see https://www.qt.io/qt-for-python and https://www.pythonguis.com/tutorials/pyside-plotting-matplotlib/ if you are using python) which presents the user with all the hyperparameters which can be tweaked in your project. Once the model is calibrated, an image sequence can be provided to the GUI that will then load and show the image sequence with all tracked objects displayed. After playing the image sequence, your interface should display:

- The average number of moving object detected per frame, as a time series, as well as the proportion of ground-truth objects that have not been matched and the number of tracks which switched from one ground truth object to another.
- The average precision and recall score for each frames as a time series
- The average precision, recall and F1 score for the whole image sequence.

With the code, you should also provide a ReadMe file in markdown format detailing how the app should be launched and used.

FAQS

Do I have to write a report?

No. All information should be included in the ReadMe file (submitted with your code).

Presentations

In your final lab (week 13: 23-27 May 2022), one group member will present their system and demonstrate that their small object detection system is working properly.

Submission Requirements

Friday 20th of May 2022 by 4:00PM, you are required to upload (on LMS) a folder containing your Matlab code. Marks will be based on your presentation and the submitted code.