# Video synopsis

Samuel Kostadinov
*University of Trento*
Trento, Italy
samuel.kostadinov@studenti.unitn.it

## I. Introduction

Video synopsis is a method used in computer vision to summarize a video while keeping track of what happened in the field of view of the camera. This requires various different techniques, such as object recognition and motion tracking.
In this project, I implemented a video synopsis algorithm, provided some surveillance videos viewed from above of an indoor scenario.

## II. External components

This project uses some components to work. In particular, in this section, there is a list of all the external components used in this project.
Python standard library modules:

- *os*
- *sys*
- *subprocess*
- *shutil*
- *sqlite3*
- class *Counter* from *collections*
- *sqrt* and *floor* from *math*
- *time*

External modules:

- *wget*
- *gitpython*
- *numpy*
- *tqdm*
- *python-magic*
- *Send2Trash*
- *opencv-python*
- *scikit-image*

Moreover, this project uses YOLOv5 for object identification, in particular the YOLOv5s6 set of weights.
All the required modules are listed in the file requirements.txt inside the "code" folder, so it is enough to launch `python -m pip install -r requirements.txt`.
Note that the script requires git and ffmpeg being accessible through the system PATH, since it relies on git to get the YOLOv5 repo and on ffmpeg to divide the video into frames and create the video output.

## III. Usage

After cloning the repo from gitHub, it's important to place the video on which the synopsis should be performed in the folder named "data". It can be placed also in a subfolder. The requirements, listed on *requirements.txt*, should be installed with `python -m pip install -r requirements.txt` or a similar command, based on the system executable. The script gives the possibility to use the GPU for the object detection with YOLO. If the user decides to use the GPU, it's also important to put the right command in the file called "command.txt" inside the "utils" folder. The script will read this file and execute the command that it's contained inside it. To find the correct command for the CUDA support, it's suggested to visit https://pytorch.org/get-started/locally/ and copy paste the command found there after selecting the setup present on the machine.
When launching the script, it will ask the user some questions:

- The script will ask if the user wants to use the GPU support for pytorch
- The script will ask if the user wants to delete the database result of the process. If the user decides to do it, the file should go through the entire pipeline another time for regenerating the output.
- The script will ask if the user wants to delete all the frames result of the middle phases of the execution of the script itself. Note that any frame, except for the background, will be deleted permanently at the beginning of a new run.
- The script will ask if the user wants to see on the console the output of the Object detection process with YOLO
- The script will ask to the user to input the name of the file to be examined. Please make sure that if the file is in a subfolder of the "data" folder the input is in the form `relative/path/to/video/`. The script also assumes that the video is in .mp4 format, so please input an mp4 file.

The script, in its usage, will also need *git* and *ffmpeg* available in the system path since makes use of both of them. The script also will need the permission to install python modules since it will try to install the requirements of YOLO and, if the user wants the CUDA support, the pytorch version that uses the GPU.

## IV. Description of the script

The script can be divided into some parts:

- **Preliminary operations**:
  These operations are necessary to the script to work. First of all, it checks if the file is an mp4 video, then checks if it has already been processed. After that, it checks if

all the necessary files are present (such as YOLO and all the folders it will use). As last step of this part, the script gets informations about the video, such as the fps and the size.

- **Process phase**:
This phase consists mainly in three steps: dividing the video into single frames, rotate every frame by 90, 180 and 270 degrees, and tun YOLO on all the frames obtained after the rotation. This is the longest phase of the script's execution. Although the fact that YOLO analyses all the frames four times makes the script noticeably slower, this was necessary since the view angle from above made YOLO not very precise. Most likely due to the fact that most of the training data were from a different perspective, YOLO had some difficulties in identifying people when they moved from the bottom of the field of view to the top. In this case, in fact, the person would result in being rotated with the legs in a higher position than the head. Rotating the images, is the way adopted in this project to maximize the possibility to find every object in the scene. Some other things to notice are that the object detection happens through a subprocess using a modified version of the original script used by Ultralytics. This decision was taken because trying to integrate the model in opencv of in any external script causes some problems, especially if the weights must be converted to another format. I found various posts showing the same problem. Since the detection script provided by Ultralytics works fine, there was the possibility to use it. The way that was used to share the data found using YOLO is to write them on a database using *sqlite3*, a module that comes with the standard library.
- **Post process**:
This phase is one of the fastest to be executed, and its goal is to transform the coordinates of the detections with a rotation different from 0 and remove all bounding boxes that, probably, are not interesting. To do so, checks the boxes and if one of them is classified as an outlier is removed. Then, it interpolates boxes if there are no detections less then a certain number of frames.
- **Background search**:
This phase consists in a simple search. The script loops through the images in which there were no detections and computes, using scikit-image, the similarity between the examined image and the previous one. The background is the image with the highest similarity is chosen as background image and copied in the "utils" folder.
- **Tag association**:
The tag association mechanism can be divided into two parts. The first one assigns tags to all the boxes remaining on a proximity basis, while the second refines the results obtained by the first one. For example, if a tag appears less then a certain number of times, that tag is examined for substitution. This refining mechanism was implemented because even after post processing there is some noise, like for example there may be reflections on the elevator that this mechanism removes. Another possibility is that the same person is labelled with different tags due to some imprecision of YOLO in finding the bounding boxes as discussed in IV. Using this refining step, this problems are almost removed completely.
- **Output video creation**:
This step consists in generating the frames of the output video, which are obtained by overlaying the bounding box in the image with the person over the background. In case the people overlap during the process, a transparency effect is applied. The frames are saved, and the composition is pretty straight forward using ffmpeg.

## V. Critical aspects and possible improvements

Although the algorithm works and the results are quite good, it's not perfect. For example, when there is more than one person on the field of view, YOLO may not be very precise if the people are near each other, so the algorithm, since it relies on YOLO's object detection is not that precise. One possible way to improve the results could be using a fine-tuned version of YOLO to find people or to fine tune it directly with the dataset on which it should be used onto. Unfortunately, there were not enough data to fine tune it properly so in this project the base version of YOLOv5 was used.
Another problem regards the video themselves. If the video is not fluid and sometimes gets blocked, the algorithm may not work properly. Depending on where the video is blocked and what position did the people have when the video was blocked. Unfortunately, this happens in most of the videos. In the video named "2018-12-03 13-00-00 13-04-54.avi" there is a person, around 1:30 in the video, going from the top of the view to the bottom that stops near the elevator. In this person's case, the video stops some times and one of these is in a pose in which YOLO does not recognize him. Since the person stops for less than a certain number of frames, the interpolation mechanism comes into play and interpolates the positions of the person, but that results in an image in which the interpolated bounding box contains nothing since in the original video the person is blocked. Otherwise, if the person stops for a number of frames higher then a certain threshold, the algorithm will treat that bounding box as the bounding box of a person that exited the field of view, and will associate another tag to the next time the person appears in another position in the field of view.
It's also important to note that the script requires git and ffmpeg being installed and accessible through the system path. It also requires python installed to install the requirements of YOLOv5.

## References

[1] *YOLOv5 gitHub page*. Last accessed: 2022-08-16. URL: https://github.com/ultralytics/yolov5.