# ELEN0016: Computer Vision
# Task 1 Report (Group 5)

**Lionel LA ROCCA**,[1] **Sacha LEWIN**,[2] **Arthur LOUETTE**,[3] **Michaël MARÉCHAL**,[4] and **Adrien VINDERS**[5]

[1] *lionel.larocca@student.uliege.be (s161284)*
[2] *sacha.lewin@student.uliege.be (s181947)*
[3] *arthur.louette@student.uliege.be (s180981)*
[4] *michael.marechal@student.uliege.be (s191343)*
[5] *adrien.vinders@student.uliege.be (s194594)*

## I. INTRODUCTION

The first task of this project focuses first on video acquisition, either online through real-time camera feed, or offline with an already acquired video. Afterwards, we try and process these videos in order to obtain an estimation of the panning camera angle, and to produce a panoramic view of the background of the filmed scene, thus substracting at best the moving elements in the foreground of the scene.

## II. WORK DISTRIBUTION

This project being done in a large group of 5 students, we tried to separate the work in an efficient way.

The first subtask that was done was building the image database, described below, and generating our reference panoramas for later comparing with our panoramas generated with the scenes with moving objects and people in the foreground. We did this all together, in order to all get our hands on the Jetson camera and OpenCV.

For the other subtasks, we tried to equally distribute between us the work load. Note however that even if a subtask is done by a part of the group, the rest of the group still worked on it, though less than the students responsible for it.

Acquisition and motion estimation modules were done mainly by Sacha Lewin and Arthur Louette, whereas the generation of the panorama without the foreground was mainly to be done by the rest of the team. However, we struggled a lot with this task and all the team worked on it.

## III. IMAGE DATABASE

We decided to use our own image database, for which we filmed two scenes, one inside, and one outside.

The inside scene consists of a small classroom in which the camera is panning from left to right and vice-versa in front of a wall and tables. The reference scene was as required shot without any moving element, for performance assessment and for trying methods first. Afterwards, we shot the same scene but with us moving and throwing objects in the camera field.

FIG. 1: Inside Scene, Reference, 1st frame.

FIG. 2: Inside Scene, 1st frame.

The outside scene was simply done in the garden of the Montefiore institute. We did a similar panning movement. The view however does not has a wall in front of it anymore, but trees, cars and other things at a further distance. We also shot a scene with us moving and throwing objects in a similar way to the inside scene.

FIG. 3: Outside Scene, Reference, 1st frame.

FIG. 4: Outside Scene, 1st frame.

## IV. ACQUISITION

This first module is smaller and consists of building some interface for easily retrieving the frames for processing in other modules.

For performance purpose, we chose to use grayscale images.

As needed, the module allows for two modes of operation, which we will call online and offline in this report. Online operation consists of reading the live feed from the webcam (and in particular the Jetson camera when running on the board), frame by frame. The offline mode is different as it reads existing recorded frames from a folder, using a path prefix, and iterating over indexed frame files, as for the format used in building the image database.

Our interface thus easily allows to switch between these modes, while having a single `read` function to be used in other modules.

Note that even though we call these modes *online* and *offline*, all other modules strictly work in an online manner, which means they work frame by frame, they do not have access to the whole feed. This means even when using the so-called offline mode where frames are read from the files, the algorithms we put in place read the frames as if they were being shot live.

## V. MOTION ESTIMATION

### A. Problem Formulation

In this module, we try to estimate the motion of the camera during the video sequence. The camera has a single degree of freedom, which is panning (i.e. rotating around its vertical y axis).

We formulate the problem as the following. Given a sequence of frames, we try to return some angle (in degrees) for each time step, which is an estimation of the current panning angle of the camera at the given time step during the sequence, considering the initial angle to be 0, and this angle is 0 at the first frame.

### B. Implementation

We implemented this using feature tracking and matching with ORB[1] and FLANN[2]. Using a comparative analysis[3], we decided to use ORB as the best feature detector in terms of both accuracy and performance. Indeed, our first choice was using SIFT[4], but performances were not great and we also had license issues with the old version of OpenCV installed on the Jetson. Using other free-to-use algorithms like KAZE led to very slow performance so ORB was chosen.

The idea is to track features between two frames, and obtain a matrix which maps points from the first frame to the second one, the so-called homography matrix. It is then possible to decompose this matrix in order to estimate the motion of the camera by obtaining a rotation matrix, from which we extract the second angle (around y) to get the panning rotation movement.

The decomposition of the homography matrix is performed using a function[5] included in OpenCV, which requires giving a camera matrix, that contains the parameters of our camera. We thus first perform a calibration[6] of our camera using a checkerboard pattern, and taking more than 9 pictures of it. This method returns a camera matrix we can use for decomposing the homography matrix.

From there, we thus have methods to extract an angle between two frames. Using our acquisition module, we can easily build an algorithm that reads frames one by one, and each time compares the new one with the previous one. However, this is not suitable for performance reasons. We decided to perform the angle update not between every frame, but every $n$ frames. This parameter can be adjusted when running the module, but is set as a default to every 15 frames.

Updating too often is not needed considering the camera is not moving at a high speed in our case, but this will lead to worse results on a fast moving camera as it will get confused and ignore some movement, or struggle if the two images share nothing in common.

Our implementation then has options for displaying the angle live (or printing it), and also for storing as a video the result when reading frames from a folder.

The angle is initially set at 0, and is updated as every time, our function gives the difference of angle between the two given frames.

### C. Results and Validation

A big assumption with this method is the presence of a planar surface during the movement in the camera field, which we can assume for the inside scene, but is less true for the outside scene where there is no wall. We should thus observe worse results when there is no wall in front of the camera to track points. When testing on various videos, and environments, we indeed observe better results inside, yet still not perfect. The results outside are still pretty good, considering the assumption is not fulfilled.

For validation, we measured the true angle we panned when filming, which was between 90 and 100 degrees inside, and about 100 to 110 degrees outside. We also compared during a same sequence, the angle at two different time steps when the camera has the same angle. Indeed, as we pan left and right, we go twice through the same direction, and therefore the angle should be about the same at the these two different instants.

We did our testings on all of our 4 sequences, but also with some other sequences with our online mode to try it over a longer time sequence. For validation, we will however here mostly consider our 4 sequences.

For the comparison with the ground truth angle, we observe pretty good values, as the angle inside reaches 94° and 91° outside.

We also observe nice results when comparing the same camera direction at different time steps. For example, our scenes with moving objects and people go back and forth more often, but over a more narrow angle, and angles stay consistent as desired over time for same directions.

For the reference scenes, the camera starts from the left, goes to the right, and comes back once, so we can compare the angles when it comes back to the angles when it goes to the right, and check for consistency. Figures 5 and 6 show very similar angles when the camera is pointed in the same direction, at two very different times. The same goes for outside, as shown on Figures 7 and 8.

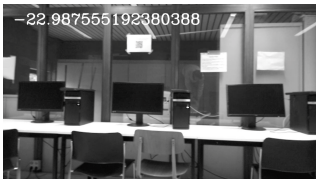


FIG. 5: Angle on 1st panning (⟶): 23°


FIG. 6: Angle on 2nd panning (⟵): 27°


FIG. 7: Angle on 1st (⟶): 38.5°


FIG. 8: Angle on 2nd panning (⟵): 36.5°

Note that as mentioned, the angle is updated about every 0.5 seconds (every 15 frames), and more precise results are obtained when skipping no frame and calculating the angle between every frame, but performance is highly impacted and makes real time angle estimation impossible. However, we tried, and we indeed get better results. For the same given moments than on previous figures, we obtain angles of 22.9° and 22.30 for inside, which are closer as expected.

A video is provided along with the report to demonstrate the angle during a whole video.

### D. Possible Improvements

We considered different options for this sub-task, especially for the case of being outside where we can't assume there is a planar surface to follow. One option we considered was making use of the optical flow.

We have therefore tried to apply a method using optical flow to estimate camera motion [7] on our dataset.
Nevertheless, first the algorithm struggled to detect the type of motion of the camera. Indeed, trucking was detected instead of panning. The procedure to detect the difference is based on the fact that when trucking, all objects in the scene move at the same speed. Whereas, while panning objects that are closer to the camera move faster than those that are further. Therefore, the procedure consist in calculating first optical flow and translation values for each point on a pair of frames. Then clustering them in 3 clusters and decide between trucking and panning by comparing the diffence between the center of the two most distant clusters and a threshold. Therefore, we think that we could adapt this threshold to detect properly the motion but when we looked at the estimation of the angle between the frames we see that the angle estimated between them is nearly always zero so there is probably a mistake in our implementation of the algorithm but we were not able to catch it.

Moreover, this method offers no guarantee of better results. It is also mentioned that the accuracy suffers from moving objects in the video and should be tackled by removing moving object of the image first which we have not done.

## VI. FOREGROUND REMOVAL

### A. Problem Formulation

Here, what we want is to detect and remove the foreground from a series of frames given by the same rotating camera of the previous point. The problem here come from the moving camera because not only the foreground but the background also move which make it very different from ordinary foreground substractor with only a comparison from a reference frame.

So, the problem can be formulated as the following, Given a sequence of frames (each arriving online), we try to return the same frames without foreground.

### B. Implementation

In our case we use the function SelfieSegmentation of the library MediaPipe to create a mask differentiating background and foreground. This function uses machine learning and allows to create a foreground mask that is accurate and robust to the camera movements. Once the foreground mask is generated we apply the previous images frame to the pixel of the current frame for which the mask denotes foreground. It allows to make the foreground disappear but create some artefact at the edge of the foreground if the camera moves too quickly. We chose this library solution over BackgroundSubtractorMOG2 from openCV for its higher robustness to the camera movements

we also have tried a different method with success :
we use the function called BackgroundSubtractorMOG2 from the openCV library.This function is a Gaussian Mixture-based Background Segmentation Algorithm. What it does is to take a picture as reference and assigns a Gaussian Distribution to each pixel for modeling it (with the weight equal the number of times a color is present for that pixel). Thus the longer a color rests in the same position, the higher the chance of it being in the background. Of course this algorithm is not the best when the camera is rotating because the same object from the background will move and thus create some blur in the resulting image. So what we do is to apply this function for each range of 25 frames where we can

considerate the background almost fixed and the foreground a lot more quickly in his shifting. Unfortunately, this solution has some drawbacks, if the foreground moves too slowly it can appear on the background or if the camera moves to quickly the resulting image will be more blurry and of course if the foreground remains in the same position it will appear on the background. We also tried different algorithms without success :

1. edge detection based algorithm :
   The idea of the edge detection based algorithm is to detect edges, contours and foreground objects area from a frame. Then a mask can be made to categorize background and foreground. Applying this mask to the frame allows to remove the foreground from the frame in order to stitch the output frame to the panorama. This method problem is that when the camera pans, it detects edges from the background and too few foreground object areas are detected with respect to background object area. As a result this method does not remove foreground from the video feed.
2. vpi background substracor based algorithm It's a background substractor provided by Nvidia to allows the usage of CUDA as backend to speed up the computations. Unfortunately it is well suited for fixed camera but not for moving cameras. Then this method was unsuccessful
3. feature detection based algorithm (SURF/SIFT/FAST) This method aims to extract features and then identify foreground features. With all features annotated with foreground or background it would have been possible to remove the features of the foreground
4. running average :
   The idea was to take the average of the color of each pixel form a pack of frames to create a background. This is pretty similar to the MOG background substracor but it's simpler.

### C. Results and Validation

Our code (here used in panorama.py with the README for the explanation of how to use it) works online and offline. After testing our code, if the foreground don't move too fast, some shapes of foreground remain after the subtractions

## VII. PANORAMA GENERATION

### A. Problem Formulation

After obtaining some frames with only the background from the camera, what we want is to make a panorama and to estimate the position of the camera in the panorama at the same time.

### B. Implementation

We implemented this using the function stitching and the frames given by our background substractor using MediaPipe (function explained previously). The function stitching has 4 main parts:

Firstly it will detect the key points of the frames (similar to surf). Then it will find matched features/descriptors, estimate camera parameters and calculate the homography matrix using the RANSAC algorithm. Finally it will wrap the images and merge them together to produce our panorama.

The RANSAC method is a iterative method to estimate mathematical model parameter.

### C. Results and Validation

In the end, the stitching function makes a good panorama but the disadvantage of this one is that it's pretty slow.

Here is a example of a panorama with background subtraction

FIG. 9: Panorama generated on 750 frames

## VIII. TECHNOLOGY AND USAGE

The code was done using OpenCV 4.6.0 with Python 3.8.15. The `README.md` file explains how to run each part.

[1] Kurt Konolige Ethan Rublee, Vincent Rabaud and Gary Bradski. Orb: An efficient alternative to sift or surf. 2011.
[2] Arul Suju D and Hancy Jose. Flann: Fast approximate nearest neighbour search algorithm for elucidating human-wildlife conflicts in forest areas. 2017.
[3] Shaharyar Ahmed Khan Tareen and Zahra Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. 2018.
[4] David Lowe. Distinctive image features from scale-invariant keypoints. 2004.
[5] OpenCV Homography Matrix Decomposition.
[6] Camera calibration with OpenCV.
[7] Camera motion estimation using optical flow, 2020.