Intrusion Detection System - Project Work

Marco Cucè, Alessandro Stockman

March 14, 2021

1 Introduction

Given the following tasks: - For each frame of an input video, the system needs to detect and display objects not belonging to the background scene and produce a textual output listing the found blobs and their meaningful features. - The system is required to discriminate between a present blob and a false one originated by the removal of an object from the background reference.

The proposed solution implements the developed library `intrusiondetection`, this notebook shows the step-by-step operations computed to achieve the final outputs displaying the computations on some key frames alongside considerations over the made choices.

```
[1]: import numpy as np
     import cv2
     from matplotlib import pyplot as plt

     from intrusiondetection.parameters import ParameterList
     from intrusiondetection.utility import distance_euclidean
     from intrusiondetection.morphology import MorphOp, MorphOpsSet
     from intrusiondetection.video import Video
     from intrusiondetection.displayable import Background
     from intrusiondetection.enum import BackgroundMethod

     %matplotlib inline

     input_video_path = "rilevamento-intrusioni-video.avi"
     video = Video(input_video_path)

     key_frame_indexes = [124, 150, 325]
     key_frames = [video.frames[frame_index] for frame_index in key_frame_indexes]
```

1.1 Considerations

Video characteristics: - 12 fps - ~41s - 320x240 pixels - 8 bit/pixel (256 gray levels)

Observed details: - There is a meaningful amount of noise present. - The lighting condition in the scene changes slightly. - The person changes its moving velocity in the course of the video.

2 Background Estimation

2.1 Initial Background

To estimate a good background for the scene, it's necessary to perform an interpolation between some chosen frames of the video, the parameters of the operation are:

- The interpolation function used, the methods took in consideration are `np.mean` and `np.median`.
- The amount of initial frames, which is tuned based on a tradeoff between the smallest and most stable number. Therefore, the considered values are 40, 80, 120

```
[2]:  from intrusiondetection.utility import subplot_images

      bg_medn_40 = Background(input_video_path, np.median, 40)
      bg_medn_80 = Background(input_video_path, np.median, 80)
      bg_medn_120 = Background(input_video_path, np.median, 120)
      bg_mean_40 = Background(input_video_path, np.mean, 40)
      bg_mean_80 = Background(input_video_path, np.mean, 80)
      bg_mean_120 = Background(input_video_path, np.mean, 120)

      subplot_images([
          {
              'object': bg_medn_40,
              'key': 'image',
              'title': 'Median of 40 pixels'
          },
          {
              'object': bg_medn_80,
              'key': 'image',
              'title': 'Median of 80 pixels'
          },
          {
              'object': bg_medn_120,
              'key': 'image',
              'title': 'Median of 120 pixels'
          }
      ])

      subplot_images([
          {
              'object': bg_mean_40,
              'key': 'image',
              'title': 'Mean of 40 pixels'
          },
          {
              'object': bg_mean_80,
              'key': 'image',
```

```
        'title': 'Mean of 80 pixels'
    },
    {
        'object': bg_mean_120,
        'key': 'image',
        'title': 'Mean of 120 pixels'
    }
])
```

| Median of 40 pixels | Median of 80 pixels | Median of 120 pixels |

| Mean of 40 pixels | Mean of 80 pixels | Mean of 120 pixels |

By performing various tests, it has been noticed that the output doesn't result in a significant change of quality by increasing the value over 80 frames, on the other hand, by decreasing the value under 80, the resulting image looks unclear using both functions.

It has also been observed that the `np.median` function achieves more stable solutions.

The chosen values are 80 frames using the `np.median` interpolation.

```
[3]: initial_background = bg_medn_80
```

2.2 Background update

The two main approaches to obtain a dynamic background are `blind` and `selective`.

The following section shows an example of a blind background computed using an adaption rate of `0.3`.

```
[4]: blind_backgrounds = video.process_backgrounds(BackgroundMethod.BLIND,␣
     ↪initial_background, 0.3)
     key_frame_backgrounds = [blind_backgrounds[key_frame_index] for key_frame_index␣
     ↪in key_frame_indexes]

     subplot_images([
         {
             'object': key_frame_backgrounds[0],
             'key': 'image',
             'title': ''
         },
         {
             'object': key_frame_backgrounds[1],
             'key': 'image',
             'title': ''
         }
     ])
```
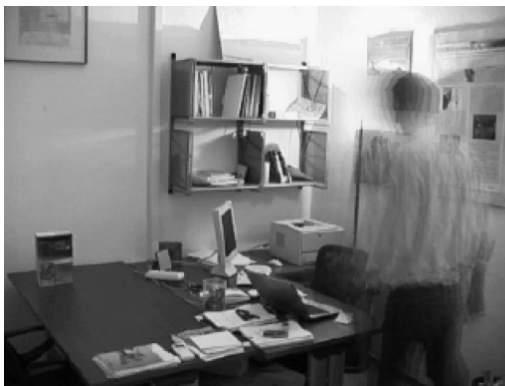

For the selective approach, a threshold and a distance function have to be used to compute the subtraction between the frame and the background.

As a further improvement, some binary morphology operators are applied to obtain a more meaningful background mask: - 3x3 Opening: Denoising of the subtraction - 50x50 Closing: Filling of any potential holes - 15x15 Dilation: Achievement of a mask big enough to contain the detected objects and their position after any movement

```
[5]: selective_backgrounds = video.process_backgrounds(
         BackgroundMethod.SELECTIVE, initial_background,
         0.3,
         30,
         distance_euclidean,
         MorphOpsSet(
             MorphOp(cv2.MORPH_OPEN, (3,3)),
             MorphOp(cv2.MORPH_CLOSE, (50,50), cv2.MORPH_ELLIPSE),
```

```
        MorphOp(cv2.MORPH_DILATE, (15,15), cv2.MORPH_ELLIPSE)
    )
)
key_frame_backgrounds = [selective_backgrounds[key_frame_index] for␣
 ↪key_frame_index in key_frame_indexes]

for key_frame_background in key_frame_backgrounds:
    key_frame_background.display_row([
        {
            'key': 'subtraction',
            'title': 'Subtraction'
        },
        {
            'key': 'mask_raw',
            'title': 'Mask'
        },
        {
            'key': 'mask_refined',
            'title': 'Mask after binary morphology'
        },
        {
            'key': 'image',
            'title': 'Final Result'
        },
    ])
```

| Subtraction | Mask | Mask after binary morphology | Final Result |
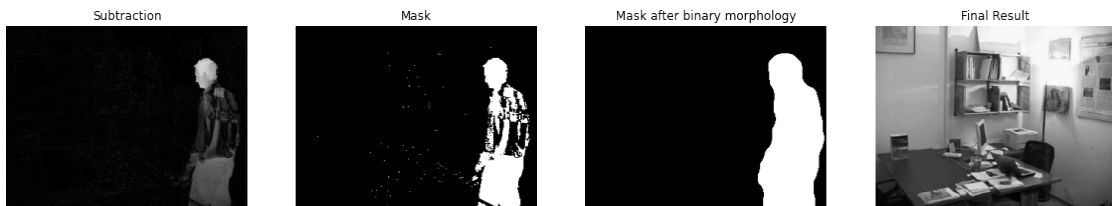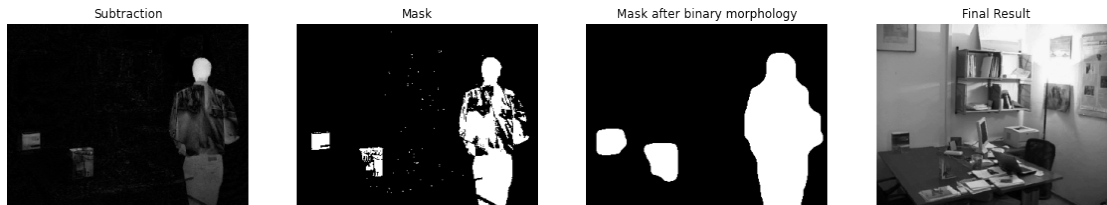
A selective background clearly obtains better results, it is therefore the chosen method.

3 Change Detection

3.1 Background Subtraction

To perform the actual change detection the first step is to subtract the current frame with respect to the corresponding background:

```
[6]: threshold = 15
distance_function = distance_euclidean


for key_frame_index, key_frame in zip(key_frame_indexes, key_frames):
    key_frame.apply_change_detection(selective_backgrounds[key_frame_index],␣
 ↪threshold, distance_function)
    key_frame.display_row([
        {
            'key': 'subtraction',
            'title': 'Subtraction'
        },
        {
            'key': 'mask_raw',
            'title': 'Thresholded Mask'
        }
    ])
```

Subtraction

Thresholded Mask

Subtraction

Thresholded Mask

Subtraction

Thresholded Mask

3.2 Binary Morphology

Thanks to the selective background update, the resulting mask doesn't show a big amount of noise, a series of binary morphology operators is applied to improve furthermore the result: - 3x3 Rectangular Opening: Removal of the majority of the noise - 50x50 Closing: Execution of holes filling - 10x10 Opening: Deletion of small blobs due to the remaining noise

```
[7]: ms1 = MorphOpsSet(
         MorphOp(cv2.MORPH_OPEN, (3,3))
     )

     ms2 = MorphOpsSet(
         MorphOp(cv2.MORPH_OPEN, (3,3)),
         MorphOp(cv2.MORPH_CLOSE, (50, 50), cv2.MORPH_ELLIPSE),
     )

     ms3 = MorphOpsSet(
         MorphOp(cv2.MORPH_OPEN, (3,3)),
         MorphOp(cv2.MORPH_CLOSE, (50, 50), cv2.MORPH_ELLIPSE),
         MorphOp(cv2.MORPH_OPEN, (10,10), cv2.MORPH_ELLIPSE),
     )

     for key_frame in key_frames:
         frame_op1, frame_op2 = key_frame.copy(), key_frame.copy()
         frame_op1.apply_morphology_operators(ms1)
         frame_op2.apply_morphology_operators(ms2)
         key_frame.apply_morphology_operators(ms3)
         subplot_images([
             {
                 'object': frame_op1,
                 'key': 'mask_refined',
                 'title': 'Operation'
             },
             {
                 'object': frame_op2,
                 'key': 'mask_refined',
                 'title': 'Operation'
             },
             {
                 'object': key_frame,
                 'key': 'mask_refined',
                 'title': 'Operation'
             }
         ])
```
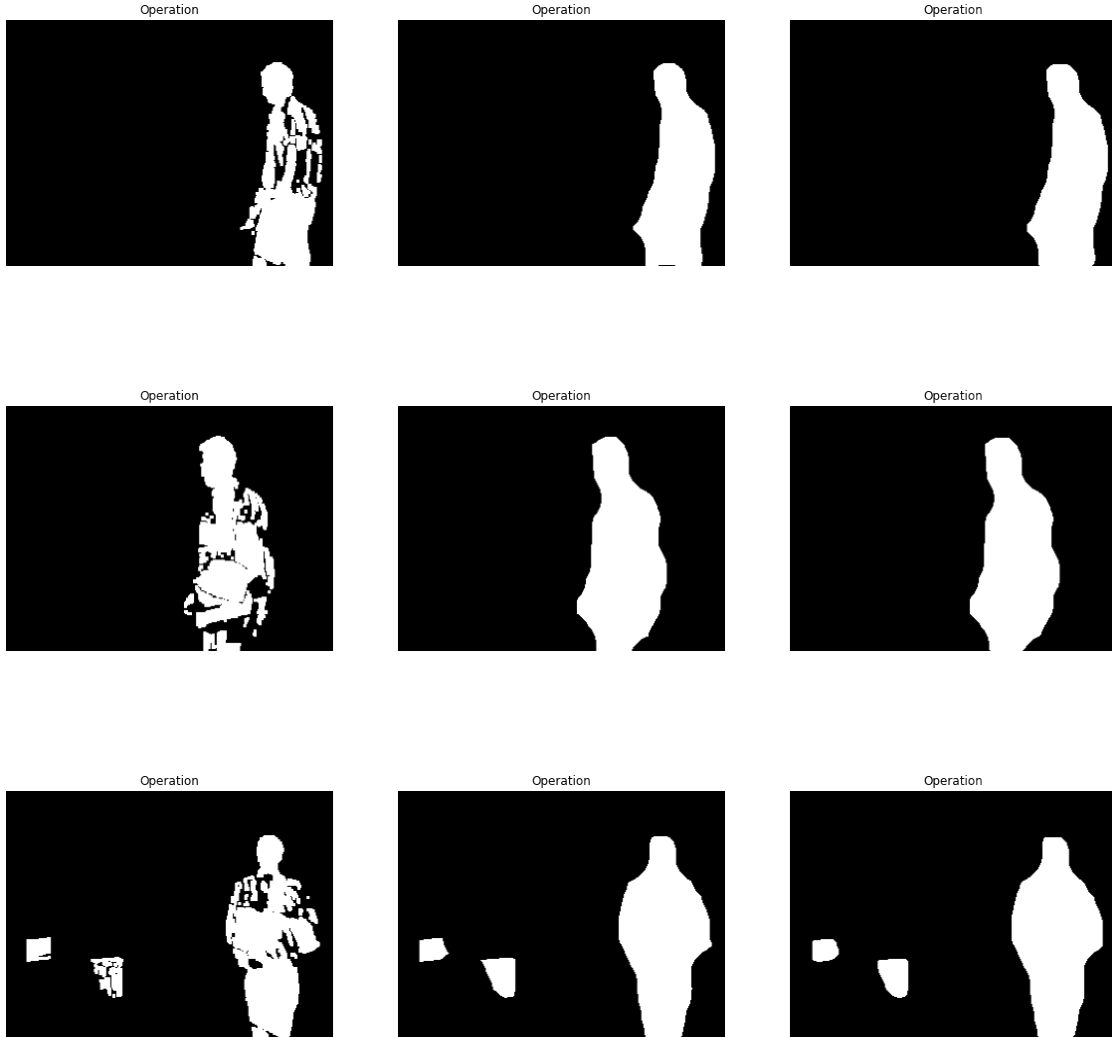
4 Blob Analysis

4.1 Blob Labeling

The labeling of the obtained image is performed using the `cv2.connectedComponents` function.

```
[8]: for key_frame in key_frames:
         key_frame.apply_blob_labeling(create_output=True)

     subplot_images([
         {
             'object': key_frames[0],
             'key': 'blobs_labeled',
```

```
        'title': "",
    },
    {
        'object': key_frames[1],
        'key': 'blobs_labeled',
        'title': "",
    },
    {
        'object': key_frames[2],
        'key': 'blobs_labeled',
        'title': "",
    },
])
```


4.2 Selected Features

The considered features are: - Area - Perimeter - Barycentre

4.3 Label Remapping

It is necessary to keep continuity between the blob labeling in subsequential frames, therefore, a similarity function is computed to seek for correspondances between the current and previous blobs.

The label are then remapped to match the ones in the previous frame. The blobs are displayed with their label printed on the barycentre.

Given the area of the two objects A_1, A_2, their barycentres $(x_1, y_1), (x_2, y_2)$, the total area of the frame A_F and its diagonal d_F the similarity function is defined as:

$$\frac{\frac{A_1 - A_2}{A_F} + \frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{d_F}}{2}$$

A threshold parameter is then applied to define a lower bound for a minimum similarity below which two blobs are always considered different.

```
[9]: from intrusiondetection.utility import default_parameters

     for key_frame, key_frame_index in zip(key_frames, key_frame_indexes):
         key_frame_prev = video.frames[key_frame_index - 1]
         key_frame_prev.intrusion_detection(default_parameters(),␣
     →selective_backgrounds[key_frame_index - 1], [])
         key_frame.apply_blob_remapping(key_frame_prev.blobs, 80, create_output=True)

         subplot_images([
             {
                 'object': key_frame_prev,
                 'key': 'blobs_remapped',
                 'title': 'Previous Frame Labeling'
             },
             {
                 'object': key_frame,
                 'key': 'blobs_remapped',
                 'title': 'Current Frame Labeling'
             }
         ])
```


Previous Frame Labeling — Current Frame Labeling

Previous Frame Labeling — Current Frame Labeling

Previous Frame Labeling   Current Frame Labeling

4.4 Classification

To classify the blobs as `person` or `object`, a `classification_score` is computed. The value is based on their area by normalizing it with respect to the total number of pixels in the frame.

After some analysis of the video, the chosen threshold for the classification is `2.6`.

```python
for key_frame in key_frames:
    key_frame.apply_classification(2.6, create_output=True)

subplot_images([
    {
        'object': key_frames[0],
        'key': 'blobs_classified',
        'title': ''
    },
    {
        'object': key_frames[1],
        'key': 'blobs_classified',
        'title': ''
    },
    {
        'object': key_frames[2],
        'key': 'blobs_classified',
        'title': ''
    }
])
```

5 True/False Object Recognition

To detect whether an object is effectively present or not, the selected approach is the evaluation of the edge strength, such that, when the contours of a found blob presents sharp edges in the original image, it is considered to be present, otherwise it is labeled as fake.

The computation is performed by using a Sobel operator returning a smoothed edge score, not taking into account the large amount of noise present in the original frame.

Given the fact that in the majority of the situations, the presence/absence of the object is continuous between contiguous frames, it has been chosen to use a value obtained by computing a weighted sum between the current edge score and the score of the correspondent object in the precedent frame.

```
[11]: for key_frame in key_frames:
          key_frame.apply_object_recognition(92, 0.1)

      subplot_images([
          {
              'object': key_frames[0],
              'key': 'blobs_detected',
              'title': ''
          },
          {
              'object': key_frames[1],
              'key': 'blobs_detected',
              'title': ''
          },
          {
              'object': key_frames[2],
              'key': 'blobs_detected',
              'title': ''
          }
      ])
```

6 Output Generation

6.1 Text Output

A CSV file is then generated, for each frame the following informations are stored: - Frame Index - Number of Detected Objects - A row for each detected object containing: - Object Identifier - Area - Perimeter - Barycentre (x coordinate) - Barycentre (y coordinate) - Classification Score - Edge Score - Object Presence [True / False] - Classification

6.2 Video Output

The graphical output shows the contours of the found objects, the color of the contour depends on the object classification: - Person: Blue - True Object: Green - Fake Object: Red

```python
[12]: for key_frame in key_frames:
          key_frame.generate_graphical_output()

      subplot_images([
          {
              'object': key_frames[0],
              'key': 'image_output',
              'title': ''
          },
          {
              'object': key_frames[1],
              'key': 'image_output',
              'title': ''
          },
          {
              'object': key_frames[2],
              'key': 'image_output',
              'title': ''
          }
      ])
```