Lab 12

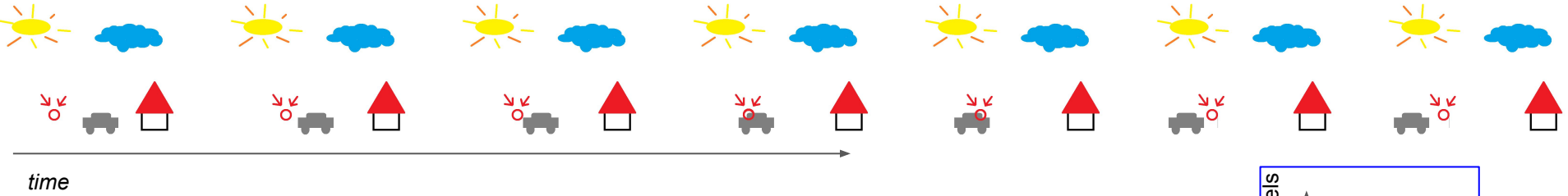
Basic Image Processing
Fall 2019

Video segmentation with temporal histogram

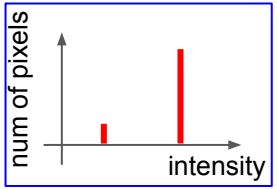
Video - 3D / 4D image sequence, the dimensions are:

- spatial coordinates (row, column),
- color channel,
- frame number.

Statistical background model - **temporal histogram**: the temporal histogram of each (row, column) pixel location is used to generate the background image.



The value with the highest peak is the value of the background pixel at that location:



Steps today

1. load a color video file in MATLAB
2. convert a color frame array (4D) to a grayscale frame array (3D)
3. calculate the background image with the help of the temporal histogram
4. create a moving object detector: the difference of the actual image and the background image is binarized and enhanced with some morphology
5. solve a high-level surveillance exercise - counting people walking into and out of a building

Today’s work

The results of this lab are not going to be uploaded.

The goal is to observe a simple but useful application of video processing.

Now please

download the ‘Lab 12’ code package

from the

submission system

Exercise 1

Implement the function `video_loader` in which you have to load an avi file to a 4D array.

The function has 2 parameters:

- **Input1:** name of the video file
- **Output1:** 4D `uint8` array (*height* x *width* x *color_channel* x *frame_number*) of the color frame sequence

Please use the `VideoReader` object, and the `hasFrame` and `readFrame` operations on it.

- create the your VideoReader entity as `vr = VideoReader(`filename`)`
- allocate space to one `uint8` output array with the help of `vr.Height` and `vr.Width`
- the `hasFrame` helps you to repeatedly check whether any unread frame is still present in the `vr` object, you can use the `readFrame` operation to read the next frame

Please test your function with the script `test1_loader`.

Exercise 2

Implement the function `rgb_video_to_gray_video` in which you have to convert your 4D color frame array to a 3D grayscale frame array.

The function has 2 parameters:

- **Input1:** `rgb_array` - 4D `uint8` array, *height* x *width* x *color_channel* x *frame_number*
- **Output1:** `gray_array` - 3D `uint8` array, *height* x *width* x *frame_number*

Please use the built-in function `rgb2gray` frame-wise (and `squeeze` if necessary).

Please test your function with the script `test2_rgb2gray`.

Exercise 3

Implement the function `calculate_background` in which you have to calculate the *mode* of the intensities pixel-wise, inside a sliding window range.

The function has 2 parameters:

- **Input1:** `gray_video` - 3D `uint8` array, *height* x *width* x *frame_number*
- **Output1:** `background` - 3D `uint8` array, *height* x *width* x *frame_number*

The *mode value* can be calculated with the built-in function `mode`, please apply it along the 3rd dimension (second parameter).

The sliding window should be interpreted as the past 100 frames - or less, if you are at the beginning of the video. Roughly formalizing:

```
start_idx = max(frame_idx-100, 1);
```

… `gray_video(:, :, start_idx:frame_idx)` …

where `frame_idx` refers to the loop variable iterating on your input array, frame-wise.

Please test your function with the script `test3_bg`. Please be patient, the processing of the frames takes a while.

actual input, frame135/393

stat. background, from frames35-135

Exercise 4

Implement the function `detect_blobs` in which you have to mask those regions which contain moving objects in the video.

The function has 5 parameters:

- **Input1:** `gray_video` - 3D `uint8` array, *height* x *width* x *frame_number*
- **Input2:** `background` - 3D `uint8` array, *height* x *width* x *frame_number*
- **Output1:** `diff_th` - the thresholded difference of the two inputs, `uint8` and same size
- **Output2:** `morph1` - eroded version of `diff_th`, also `uint8` and same size
- **Output3:** `morph2` - dilated version of `morph1`, also `uint8` and same size

Tips and tricks on the next slide!

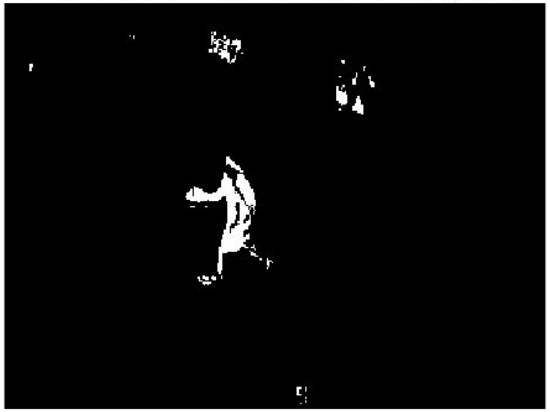
Please test your function with the script `test4_blobdetector`. Please be patient, the processing of the frames takes a while.

Exercise 4 - continued

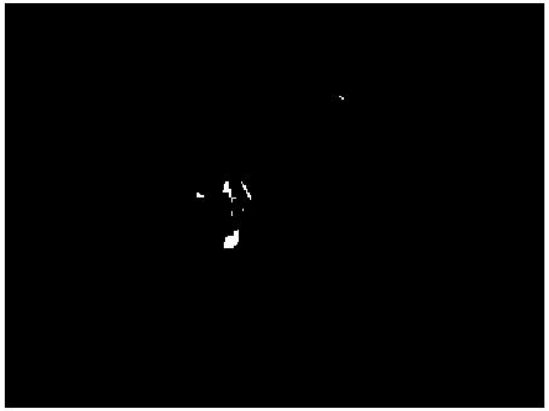
- Define a threshold for the differences (`d_th`), 50 is a good choice.
- Convert the two input arrays to have type `double` (use `foo = double(foo);`).
- Allocate `uint8` type arrays (filled with zeros) for the output args, their size should match the size of the input arrays.
- With a `for` loop iterate through all of the frames and compute the followings:
    - Calculate `frame_diff` as the `abs` value of the difference between the actual frame of `gray_array` and `background`.
    - Update `frame_diff`, now it should contain its thresholded version (255 if above threshold, 0 below (or equal)). Use logical operations.
    - Save `frame_diff` as the appropriate frame of output array `diff_th`.
    - Apply morphological *erosion* on this `frame_diff` array with `imerode,` with *structuring element* `'disk'` and *radius* value `3`, this should be saved in `morph_1`,
    - Apply morphological *dilation* on the result of the *erosion* with `imdilate`, with *structuring element* `'disk'` and *radius* value `7`., this should be saved in `morph_2`.

actual input, frame47/393

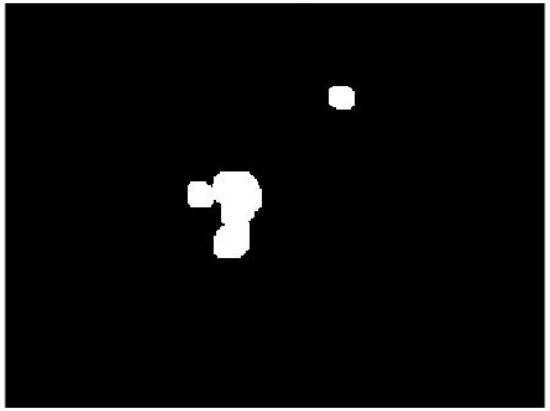
thresholded diff. between act. and background



result of erosion



result of dilation, final mask



Exercise 5

Implement the function `walk_in_walk_out_counter` in which you have to count the number of entering/leaving persons (moving objects) with the help of *sensitive zones*. The function has 5 parameters:

- **Input1:** `morph_2` - 3D `uint8` array, *height* x *width* x *frame_number* (the morphologically enhanced object-blobs)
- **Output1:** `active_pixels_zoneA` - a *1* x *frame_number* sized vector, containing the number of moving objects' pixels inside *zone A* for every frame separately
- **Output2:** `active_pixels_zoneB` - same as previous, but for *zone B*
- **Output3:** `active_pixels_zoneC` - same as previous, but for zone C
- **Output4:** `persons_inside` - a *1* x *frame_number* sized vector, containing the number of persons inside the building at every frame-iteration

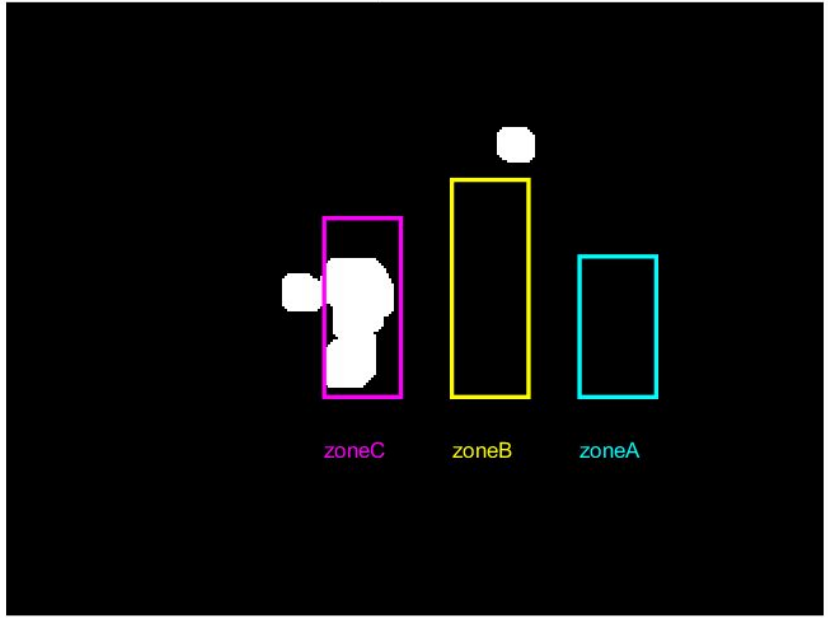
Tips and tricks on the upcoming slides!

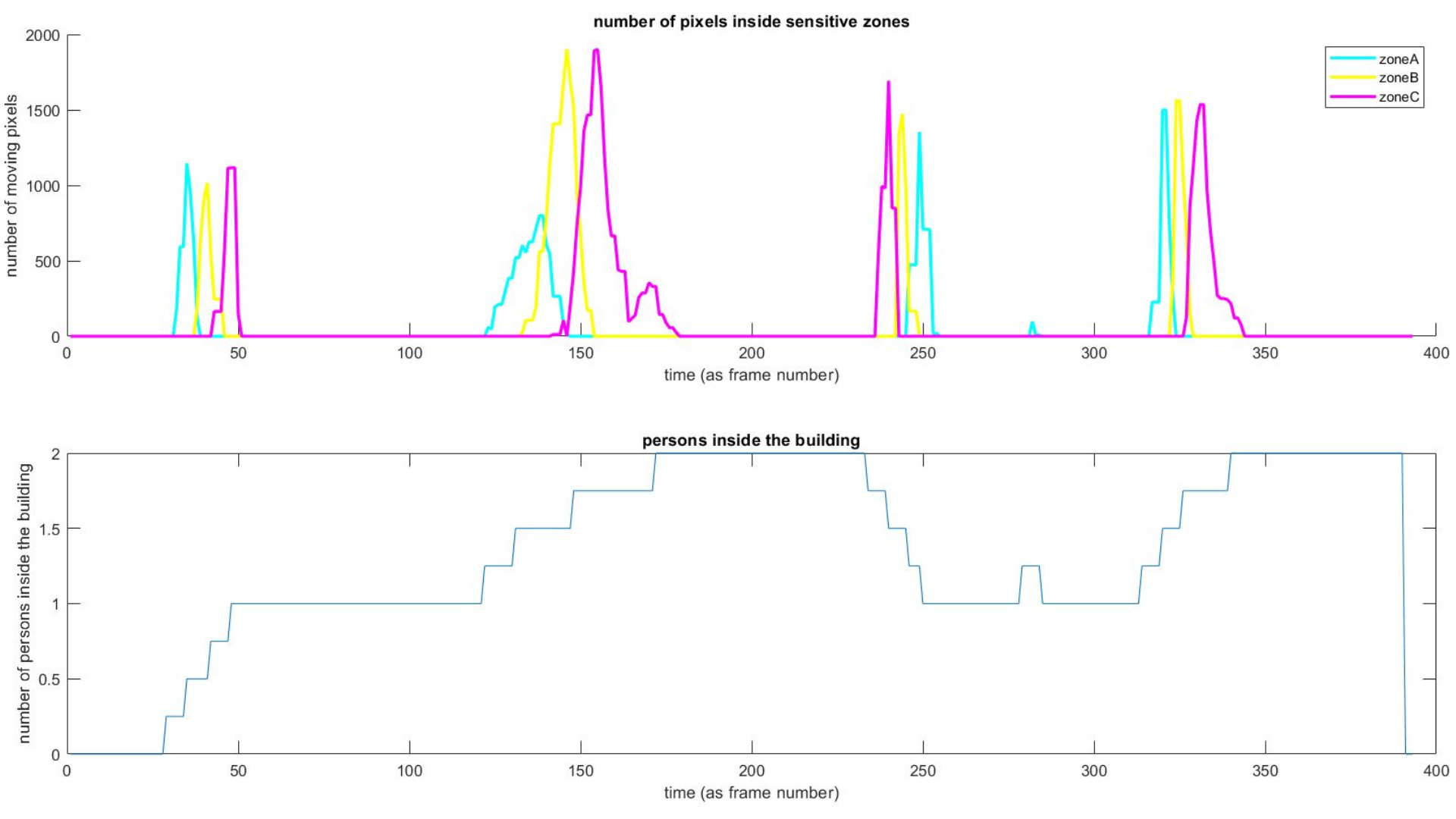
Please test your function with the script `test5_counter`.

(This tester works with the help of a matlab archive, which has been already created during `test4_blobdetector`.)

actual input, frame47/393

result of dilation, final mask with zones




number of pixels inside sensitive zones
number of moving pixels
time (as frame number)
zoneA
zoneB
zoneC
persons inside the building
number of persons inside the building
time (as frame number)

Exercise 5 - continued

- Convert the `morph_2` array to double and scale it to `[0, 1]`.
- Allocate spaces as zero vectors for the first three return values.
- With a `for` loop iterate through the `morph_2` array, along the *frame_number* dimension, and at each frame summarize the number of pixels inside the *sensitive zones* -- coordinates:
  - `zoneA: (100:155, 225:255)`
  - `zoneB: (70:155, 175:205)`
  - `zoneC: (85:155, 125:155)`
  - Save the zone-sums into the `active_pixels_zoneX` arrays at the appropriate index (given by the frame index).
- After the loop, indicate in logical index-vectors, which frames contained more active pixels, than a predefined threshold (`min_patch_size=90;`), like `somebody_in_zoneA = active_pixels_zoneA > min_patch_size;`
- Set `persons_inside` to an empty array, then at this point, please test your function with **script** `test5_counter`.

Exercise 5 - continued

- We will define region-transitions as follows:
  - `0` - `zoneA` transition → + 0.25 person
  - `zoneA` - `zoneB` transition → + 0.25 person
  - `zoneB` - `zoneC` transition → + 0.25 person
  - `zoneC` - `0` transition → + 0.25 person
  - `0` - `zoneC` transition → - 0.25 person
  - `zoneC` - `zoneB` transition → - 0.25 person
  - `zoneB` - `zoneA` transition → - 0.25 person
  - `zoneA` - `0` transition → - 0.25 person
- These transitions will be examined with a 7 unit-wide sliding window, with the help of the logical vectors `somebody_in_zone`*`X`* and with a special logical vector `nobody_in_zones`:
  `nobody_in_zones = ~somebody_in_zoneA & ~somebody_in_zoneB & ~somebody_in_zoneC;`
- Let's initialize the vector `persons_inside`: a *zero* vector with the same size as our logical vectors.
- Let's introduce two helper variables, in order to keep records of *current* and *previous transition states* (`current_state`, `previous_state`, respectively). The state-transitions can be abbreviated as `'0A'`, `'AB'`, `'BC'`, … please store these abbreviations in the helper variables. Initially, both of them should have the value of `'00'`.

Exercise 5 - continued

- Let's create a huge `for` loop (iterating between `4` and *`Length`*-3, due to the 7 unit wide sliding window), in which:
  - identify in which transition type are you currently in (*8* `if`/`elseif` *branches!*)

```
if nobody_in_zones(frame_idx-3) && somebody_in_zoneA(frame_idx+3)
    current_state = '0A';
elseif ...
```

  - after you have identified the `current_state`, please compare it (`strcmp`) with the `previous_state`:
    - *if they are equal*, the current element of `persons_inside` should remain the same as in the previous iteration,
    - *if they differ*:
      - update `previous_state` to `current_state`,
      - give the appropriate value to the current element of `persons_inside`:
        - if `current_state` is one of `0A`, `AB`, `BC`, `C0`, then increment `persons_inside` current element with `+0.25`, compared to the previous iteration,
        - if `current_state` is not in the above list, then decrement the current element of `persons_inside` by `-0.25`.
- Test your code again with **script** `test5_counter`.

THE END