University of Ljubljana, Faculty of Electrical Engineering
Robot Vision (RV)

May 24/25, 2016

Exercise 10: Tracking and motion analysis

Created by: Žiga Špiclin | http://lit.fe.uni-lj.si/RV | Homework deadline: June 1, 2016

Instructions

Based on an image sequence acquired with one or more cameras we can extract and analyze the motion of objects in 3D space and further employ this information to devise a closed-loop control system, object manipulation and tracking or even 3D object reconstruction and object detection. Process of motion analysis is generally composed of several basic methods, such as robust detection of object(s) of interest, correspondence detection between images in sequence and image registration. During this exercise you will devise a method for object or *target* tracking based on **Lucas-Kanade image registration**.

To perform fast registration this method employs a parametric model of image transformation $\mathcal{T}: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ between sequential 2D images $I_k(\boldsymbol{x})$ and $I_{k-1}(\boldsymbol{x})$ ($\boldsymbol{x} = [x, y]^{\mathrm{T}}$), where the first image was acquired at time $t(k)$ and the second at $t(k-1)$, and $t(k) > t(k-1)$. Parametric model $\mathcal{T}(\boldsymbol{x}; \boldsymbol{p})$ can be a simple translation, similarity, affine or projective transformation ($\boldsymbol{p}$ has 2, 3, 6 or 8 parameters). We assume that the two images in sequence, I_1 and I_2, have consistent intensity of corresponding regions $I_1(\boldsymbol{x}) = I_2(\mathcal{T}(\boldsymbol{x}; \boldsymbol{p})) + n(\boldsymbol{x})$[1], thus a simple image similarity measure like sum of squared intensity differences is used. We also assume small target motion between images I_1 and I_2 ($\boldsymbol{p} \rightarrow 0$) and therefore search for an optimal small step $\Delta\boldsymbol{p}^\star$ in transformation parameters

$$\Delta\boldsymbol{p}^\star = \arg\min_{\Delta\boldsymbol{p}} \sum_{\boldsymbol{x}} \left[I_2(\mathcal{T}(\boldsymbol{x}; \boldsymbol{p} + \Delta\boldsymbol{p})) - I_1(\boldsymbol{x})\right]^2 ,$$

while the transformation parameters are obtained as $\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta\boldsymbol{p}^\star$. The optimal step $\Delta\boldsymbol{p}^\star$ can be efficiently computed without optimization, such that the expression $I_2(\mathcal{T}(\boldsymbol{x}; \boldsymbol{p} + \Delta\boldsymbol{p}))$ is approximated by a Taylor series

$$I_2(\mathcal{T}(\boldsymbol{x}; \boldsymbol{p} + \Delta\boldsymbol{p})) \approx I_2(\mathcal{T}(\boldsymbol{x}; \boldsymbol{p})) + \nabla I_2(\mathcal{T}(\boldsymbol{x}; \boldsymbol{p})) \frac{\partial \mathcal{T}}{\partial \boldsymbol{p}} \Delta\boldsymbol{p} ,$$

where $\nabla I_2(\mathcal{T}(\boldsymbol{x}; \boldsymbol{p})) = [I_x, I_y]$ is the image gradient I_2[2] of current transformation $\mathcal{T}(\boldsymbol{x}; \boldsymbol{p})$. Expression $\partial\mathcal{T}/\partial\boldsymbol{p}$ represent the Jacobian matrix of first order derivatives and depends on the employed parametric transformation $\mathcal{T}$. The difference between the images in the first equation can be written in matrix form as $\boldsymbol{I}_t + \boldsymbol{B}\Delta\boldsymbol{p}$, where $\boldsymbol{I}_t = \boldsymbol{I}_2(\mathcal{T}(\boldsymbol{p})) - \boldsymbol{I}_1$; $\boldsymbol{I}_k$ is a $N \times 1$ vector of image intensity values. Expression for $\boldsymbol{B}$ when transformation model is translation or affine are given below.

Translation:

$$\mathcal{T}(\boldsymbol{x}; \boldsymbol{p}) = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

$$\frac{\partial \mathcal{T}}{\partial \boldsymbol{p}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\nabla I_2 \frac{\partial \mathcal{T}}{\partial \boldsymbol{p}} = [I_x, I_y]$$

$$\boldsymbol{B} = [\boldsymbol{I}_x, \boldsymbol{I}_y] \in \mathbb{R}^{N \times 2}$$

Affine:

$$\mathcal{T}(\boldsymbol{x}; \boldsymbol{p}) = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\frac{\partial \mathcal{T}}{\partial \boldsymbol{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

$$\nabla I_2 \frac{\partial \mathcal{T}}{\partial \boldsymbol{p}} = [I_x x, I_y x, I_x y, I_y y, I_x, I_y]$$

$$\boldsymbol{B} = [\boldsymbol{I}_x \boldsymbol{X}, \boldsymbol{I}_y \boldsymbol{X}, \boldsymbol{I}_x \boldsymbol{Y}, \boldsymbol{I}_y \boldsymbol{Y}, \boldsymbol{I}_x, \boldsymbol{I}_y] \in \mathbb{R}^{N \times 6}$$

Optimal step $\Delta\boldsymbol{p}^\star$ of transformation parameters is expressed as

$$\Delta\boldsymbol{p}^\star = \arg\min_{\Delta\boldsymbol{p}} (\boldsymbol{I}_t + \boldsymbol{B}\Delta\boldsymbol{p})^{\mathrm{T}} (\boldsymbol{I}_t + \boldsymbol{B}\Delta\boldsymbol{p}) ,$$

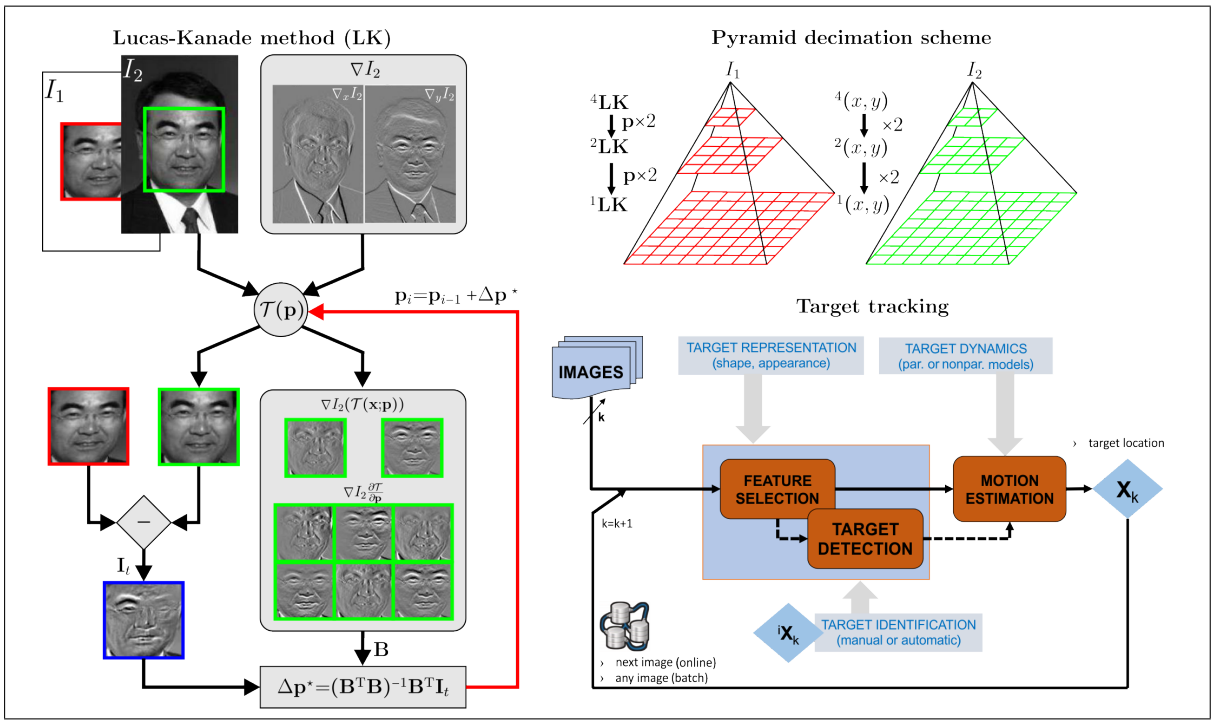
where its value is obtained by deriving the above equation with respect to $\Delta\boldsymbol{p}$ and equalizing to 0. This gives a linear system of equations for $\Delta\boldsymbol{p}^\star$, which can be solved as

$$\Delta\boldsymbol{p}^\star = -\left(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}\right)^{-1} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{I}_t .$$

[1] $n(\boldsymbol{x}) \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise.

[2] $\nabla I_2 = \left[\frac{\partial I_2}{\partial x}, \frac{\partial I_2}{\partial x}\right] = [I_x, I_y]$

Lucas-Kanade is an iterative method, where in each iteration an optimal step is computed $\Delta \boldsymbol{p}^{\star}$ and the transformation parameters are updated $\boldsymbol{p}_i = \boldsymbol{p}_{i-1} + \Delta \boldsymbol{p}^{\star}$ until some predefined number of iterations $i_{\max}$ or until $\|\boldsymbol{p}_i - \boldsymbol{p}_{i-1}\| < \epsilon$. The method is illustrated in the figure below.



Lucas-Kanade method generally converges into the nearest local minimum, therefore when target motion is larger between two consecutive images a good initial guess of the transformation parameters $\boldsymbol{p}$ is required. The initial guess can be obtained by registration based on pyramid decimation scheme, such that the images are first aligned when image sampling is sparse and then refined at denser image sampling. The sampling density is usually changes for a factor of $\times 2$ and the parameters $\boldsymbol{p}$ should also be updated according to this factor when going from one sampling density to another.

Target tracking in a video can be carried out by defining either manually or automatically the target center (x_c, y_c) and a rectangular bounding box (w, h) in the first image. If target center is known in the first image I_1, then Lucas-Kanade method can be used to align I_1 onto the next acquired image I_2 and then use the computed $\mathcal{T}(\boldsymbol{x}; \boldsymbol{p})$ to update the target center in I_2. This procedure is repeated for all consecutive image pairs $\{I_k, I_{k+1}\}, k = 1, \ldots, T-1$.

Materials for this exercise are two video files `video1.avi` and `video2.avi` acquired using a security camera overlooking a sidewalk. In Python script `funkcije_vaja10.py` you will find functions required to carry out the assignments, which will involve the design of motion tracking method to analyse movement of pedestrians on the sidewalk.

Besides these materials you will need an external library `ffmpeg`, which is required to decode the given video files and which is available online at https://www.ffmpeg.org/download.html. You should copy the executable `ffmpeg.exe` into your working folder.

1. Load the executable `ffmpeg.exe` into your working folder and then load the frames in `video1.avi` into a Python variable `numpy.array` using function `loadVideo()` and display using `showVideo()` as demonstrated by the following script:

```
# load the library
from funkcije_vaja10 import *
# load video
oVideo = loadVideo( 'video1.avi' )
# display video
showVideo( oVideo )
```

2. Write a function that aligns two grayscale images `iImgFix` and `iImgMov` using Lucas-Kanade method:

```
def regLucasKanade( iImgFix, iImgMov, iMaxIter ):
        return oPar, oImgReg
```

where `iImgMov` is a moving image, which we want to align into a fixed image `iImgFix`. Parameter `iMaxIter` represents the number of steps of the Lucas-Kanade method. Function returns a two-row vector `oPar` with translations $\boldsymbol{p} = [t_x, t_y]^{\mathrm{T}}$ and the transformed moving image in variable `oImgReg`. Input images `iImgFix` and `iImgMov` and the output image `oImgReg` have the same dimensions.

Verify the registration method using the first image in video `video1.avi`, such that the image is first transformed using parameters $\boldsymbol{p} = [-5, 3]^{\mathrm{T}}$ and then registered using Lucas-Kanade method. The computed motion parameters `oPar` should have approximately the same values as $\boldsymbol{p}$.

3. Write a function that aligns two grayscale images `iImgFix` in `iImgMov` using a pyramid-based Lucas-Kanade method:

```
def regPyramidLK( iImgFix, iImgMov, iMaxIter, iNumScales ):
        return oPar, oImgReg
```

where `iImgMov` is a moving image, which we want to align into a fixed image `iImgFix`. Input parameter `iNumScales` (s) gives the number of consecutive decimations of input images `iImgMov` and `iImgFix` with a step 1:2, which then form a pyramid of images with progressively smaller sampling density. Lucas-Kanade method is perfomed consecutively on input images in the pyramid starting at the smallest and progressing to the largest sampling density, while each time a sampling density is changed the initial parameters are updated as $\boldsymbol{p}^s = 2 \times \boldsymbol{p}^{s-1}$. Parameter `iMaxIter` denotes the number of iterations of the Lucas-Kanade method in each level of the pyramid.

Verify the function on the first image in the video `video1.avi`, such that then image is transformed using parameters $\boldsymbol{p} = [50, -30]^{\mathrm{T}}$ and then registered to the original image using the pyramid-based Lucas-Kanade method. The computed motion parameters `oPar` should have approximately the same values as $\boldsymbol{p}$.

4. Write a function for tracking a target in a video using a pyramid-based Lucas-Kanade method:

```
def trackTargetLK( iVideoMat, iCenterXY, iFrameXY ):
        return oPathXY
```

where `iVideoMat` is a video in an array of dimensions $X \times Y \times T$; X, Y are spatial axes and T the time axis. Input parameters `iCenterXY` and `iFrameXY` are two-row vectors, which correspond to the target center (x_c, y_c) and to the size of rectangular bounding box (w, h). Function should return a path of the target in an array `oPathXY` of dimensions $2 \times T$, i.e. the target center in each of the video frames.

Verify the method for target tracking on a video `video1.avi`.

In the first frame the target has a center of $(x_c, y_c) = [34, 371]^{\mathrm{T}}$ and resides in a rectangular window $(w, h) = [40, 40]^{\mathrm{T}}$. Display the video and insert the target path as shown on the figure below.

Target tracking using Lucas-Kanade method (`video1.avi`)

Homework Assignments

Homework report in the form of a Python script entitled `NameSurname_Exercise9.py` should execute the requested computations and function calls and display requested figures and/or graphs. It is your responsibility to load library packages and provide supporting scripts such that the script is fully functional and that your results are reproducible. The code should execute in a block-wise manner (e.g. `#%% Assignment 1`), one block per each assignment, while the answers to questions should be written in the corresponding block in the form of a comment (e.g. `# Answer: ...`).

Your assignment is to devise a method for target tracking in video `video2.avi`. In the first frame the target that we want to track has center coordinates $(x_c, y_c) = [287, 415]^{\mathrm{T}}$ and resides in rectangular window $(w, h) = [45, 30]^{\mathrm{T}}$.

In order to track this target you will need to improve the Lucas-Kanade method or devise a new method, which is more robust to perform tracking of this target. The ideas to improve the method of tracking can be sourced from previous lab work and/or the lectures.

1. The basic Lucas-Kanade method used to track a target in `video1.avi` cannot track the target chosen in `video2.avi`. Display the video frame, where the method fails. What is the reason for failure of tracking?

2. Devise and improved method for tracking the target in `video2.avi`. Briefly describe your approach to target tracking and the changes/improvements made to the original pyramid-based Lucas-Kanade method.

3. Run your improved tracking method on `video2.avi` and determine the target path `oPathXY`. Display the video and the target path as shown on the figure below.

Target tracking (`video2.avi`)

The grading of this exercise will depend on the level of innovation you will integrate into the tracking method and based on how far the method will be able to track the target.