Faculty of Mechanical Engineering and Robotics

Department of Robotics and Mechatronics

Basics of AI and Deep Learning

Course for Mechatronic Engineering with English as instruction language

Instruction 9:

Video Processing
and
Object/Feature Tracking

You will learn: how to load and process video in MATLAB, how to perform object tracing based on binary image object features, how to apply correlation measure for object tracking, what are invariant features (SIFT, SURF) and how to use them for object tracking, what is optical flow and how to calculate optical flow field from frames of video sequence

Additional materials:

- Course lecture

Course supervisor:
Ziemowit Dworakowski, zdw@agh.edu.pl

Instruction author:
Krzysztof Holak, holak@agh.edu.pl

Object tracking

Motion detection, estimation and object tracking are the most important application of video processing. In this exercise, you will learn the basics of the motion analysis using MATLAB. You'll start with the task of object tracking on the video frame sequence.

Object tracking on binary video sequence

To do this exercise, you will apply the methods that you have already learned in the previous exercises. In the case of the video, the image processing and analysis have to applied to each frame of the movie sequence. In this laboratory you will learn how to process video files in MATLAB.

In MATLAB, we store video frames in an object called **`VideoReader.`** Let's create this object by calling the function

```
video = VideoReader(filename);
```

Instead of filename, provide the path to the video file as in the case of loading single images. **`VideoReader`** type video object contains various information about the video sequence. For example,
`video.FrameRate` contains information about the number of frames per second, **`video.NumFrames`** has information about the number of frames in a sequence,
`video.Height, video.Width` has information about the resolution of each frame of the video.

You can play the video in MATLAB window with a simple code, for example

```
currAxes = axes;

while hasFrame (video)

    vidFrame = readFrame(video);
    image(vidFrame,"Parent",currAxes)
    currAxes.Visible = "off";
    pause(1/v.FrameRate)

end
```

`hasFrame()` function is used to check if there are frames left in the sequence and the **`v.FrameRate`** variable determines the appropriate frame display time.

`readFrame()` function is used to retrieve one frame from the video at a time .

Task 4.1 Write a simple tracker that estimates a trajectory of an object on a video sequence. Find the centroid of the object in each of the frame of the video. Having data, draw the object's trajectory in 2D and show its displacement along the x and y axis as a function of time on graphs. Try to determine the object's velocity and acceleration - through numerical differentiation.

An example of simple tracking

```
clear all
clc
close all

video = VideoReader('bouncing_ball.mp4'); % reading video file

% % info on video
% video.FrameRate;
% video.NumFrames;
% video.Height;
% video.Width;


% Finding a drawing trajectory of the centroid
index = 1;

figure,
h = imshow(rgb2gray(read(video,1))); hold on
h1 = plot(0,0,'*'); hold on


while hasFrame (video) % look for video frame processing

    vidFrame = readFrame(video);

    im = rgb2gray(vidFrame);
    imBW = im > 10;   % image thresholding, threshold set to 10 in this example

    imL = bwlabel(imBW); % labelling and object analysis - we need centroid
    features = regionprops(imL, 'centroid');
    Centroids(index,:) = features.Centroid;

    set(h, 'CData', im);  % Update the frame % how to update data in figures
    set(h1,'XData',Centroids(index,1),'YData',Centroids(index,2));
    title(['Time ' num2str(video.CurrentTime)]);
    drawnow;
    index = index + 1;

end
```

Tracking using image correlation

As you know from previous exercises, one can find a pattern in the image using a similarity measure, for example normalized cross correlation coefficient. When applied to the video sequence, this approach may be used to track a pattern across frames of the sequence.
To start the exercise, you have to load a video sequence using **`VideoReader()`** function as in the previous exercise. Next, let's read a first frame of the sequence and covert it to a grayscale.

```
% Read the first frame and convert it to grayscale
frame1 = read(video, 1);
frame1_gray = rgb2gray(frame1);
```

Now, you can display the frame and chose an object to be tracked across the sequence. To do this, use **`imcrop()`** function discussed in the first lab exercises on computer vision and image processing. Let's assume that the cropped pattern in stored in a variable **`template`** and its cropping **`rectangle`** is a variable rect.

You can initialize the tracking. Prepare the variables for tracking a pattern.

```
objectPosition = rect;  % Store initial position
trackedPositions = objectPosition;  % To store the positions
```

Tracking can be visualized in the figure.

```
figure;
h = imshow(frame1_gray); hold on;
rect_handle = rectangle('Position', objectPosition, 'EdgeColor',
'r', 'LineWidth', 2);
```

Proceed with all frames in the sequence.

```
clear all
clc
close all

video01 = VideoReader('dvdscreensaver.mp4');

% Read the first frame and convert it to grayscale
frame1 = read(video01, 1);
frame1_gray = rgb2gray(frame1);

[template objectPosition] = imcrop(frame1_gray);

 % Store initial position
trackedPositions = objectPosition;

% visualization of tracking process

figure;
h = imshow(frame1_gray); %hold on;
rect_handle = rectangle('Position', objectPosition, 'EdgeColor', 'r', 'LineWidth', 2);

% Process each subsequent frame in the video
while hasFrame(video01)

    % Read the next frame and convert to grayscale
    frame2 = readFrame(video01);
    frame2_gray = rgb2gray(frame2);

    % Perform normalized cross-correlation (NCC) for template matching
    correlationMap = normxcorr2(template, frame2_gray);

    % Find the peak of the correlation (most similar region)
    [maxCorr, imax] = max(abs(correlationMap(:)));
    [ypeak, xpeak] = ind2sub(size(correlationMap), imax(1));

    % Compute the offset (translation) from the correlation peak
    corr_offset = [xpeak - size(template, 2), ypeak - size(template, 1)];

    % Update the object's position
    objectPosition = [corr_offset, size(template, 2), size(template, 1)];
    trackedPositions = [trackedPositions; objectPosition];  % Append new position

    % Update the displayed rectangle for the new object position
    set(rect_handle, 'Position', objectPosition);
    set(h, 'CData', frame2_gray);  % Update the frame
    title(['Time ' num2str(video01.CurrentTime)]);
    drawnow;

end
```

Task 4.2 Write a function that tracks a template across frames of a video sequence. Visualize the tracking using the code provided in the lab instruction. A user should be able to chose any rectangular pattern in the first frame of the sequence manually. Modify the function in such a way that it draws a trajectory of the tracked object in one of the frames of the sequence.

Tracking objects using optical flow

In the case of working with grayscale videos with complex scenes it may be very hard or impossible to generate binary image frames with objects using thresholding or other segmentation methods. Also if the object appearance change during the motion it may lead to erroneous tracking when image correlation is applied. Moreover, all of the tracing discussed so far need detection by object either through means of binary image analysis or by manually choosing a template to be track. It is possible to automatically analyze the video to detect interesting objects based on the motion of their pixels alone. You can apply a motion estimation method and in the next step threshold the image into binary image into objects based on the values of their displacements.

One of the most commonly used motion estimation method is called optical flow. It uses a consecutive frames of a video sequence to estimate a displacement of pixels. In the basic version, it assumes that the brightness of the moving objects do not change between frames (the brightness constancy assumption). Mathematically it uses the similar approach as in the computation of the Harris corner detector. One of the challenges of the optical flow is that a single pixel gives one optical flow equation and there are two parameters of motion to estimate for each pixel (u, v coordinates). Therefore one assumes that the pixels in a close neighborhood have the same optical (motion) and its value is found by combining an optical flow equation for each of them in single equation. The solution is obtained in a least squares sense.

Let's see how image segmentation and object tracking can be computed using Lukas Kanade optical flow method.

Again, first you need to load a video file into **videoReader** object and read a first frame converting it into grayscale image.
Next, you have to initialize the optical flow object and prepare visualization figure. In the example we use Farneback method. You can also test other optical flow algorithms available in MATLAB (classic Lukas-Kanade, Horn-Shunck)

```
% Initialize optical flow object
opticFlow = opticalFlowFarneback;

figure;
h = imshow(frame1_gray);
hold on;
```

Next, let's estimate the motion for each frame of the video sequence, and find the regions which undergo the largest motion. These will be our objects to be tracked in the video sequence. Here is a code fragment that performs tracking based on estimated motion computed by optical flow.

```
clear all
clc
close all

% Load video
videoFile = 'dvdscreensaver.mp4'; % path to video file
videoObj = VideoReader(videoFile);

% Display video
figure;
```

```
hold on;

% Farneback optical flow initialization
opticFlow = opticalFlowFarneback;

% A variable to store centroids
centroidsPrev = [];

while hasFrame(videoObj)
    % Read a frame
    frame = readFrame(videoObj);

    % Change into grayscale
    grayFrame = rgb2gray(frame);

    % Compute Optical flow
    flow = estimateFlow(opticFlow, grayFrame);

    % Compute motion mask based on a magnitude of optical flow
    motionMask = sqrt(flow.Vx.^2 + flow.Vy.^2) > 2; % Próg ruchu

    % Find object contours
    stats = regionprops(motionMask, 'Centroid', 'Area','BoundingBox');

    % Remove small objects - noise removal operation - may use morphology
    % before regionprops
    stats = stats([stats.Area] > 100); % Set area threshold

    % Show objects
    imshow(frame);
    hold on;

    % If there are objects, track them
    if ~isempty(stats)
        % Draw centroids of objects
        for i = 1:length(stats)
            centroid = stats(i).Centroid;
            plot(centroid(1), centroid(2), 'ro'); % Centroid
        end
    end

    % Display video
    pause(1/videoObj.FrameRate);
end
```

Modify the code to store the centroid and bounding boxes of each tracked object for all frames of the sequence.

Task 4.3 Write a program for tracking objects in a video sequence using chosen optical flow method. Use a modified version that stores centroids and bounding boxes of the tracked objects. Draw trajectory of your tracked object.

It is possible to visualize the motion of the objects without thresholding and segmenting the scene into objects. You can show the motion of each pixels in the image sequence by using MATLAB **`quiver()`** tool for field flows visualization. You can also use heatmap to show the magnitude of the motion – using **`imshow()`** or other visualization function you have learned in the previous exercises.
Therefore, for each frame, First visualize the motion vectors

```
flow_x = flow.Vx;
flow_y = flow.Vy;

[rows, cols] = size(frame2_gray); hold on;
quiver(repmat((1:cols), rows, 1), repmat((1:rows)', 1, cols),
flow_x, flow_y, 'r', 'MaxHeadSize', 1.5, 'LineWidth', 1.5);
```

And see the magnitude of the optical flow using heatmap approach

```
imshow(flow_magnitude, []); %Display flow magnitude as a heatmap
```

A simple program to visualize results of optical flow tracking – to compare optical flow methods available in MATLAB. Please note which parts of objects are considered as moving in each of the available optical flow methods.

```
clear all
clc
close all

video = VideoReader('dvdscreensaver.mp4');
% bouncing_ball.mp4

frame1 = read(video, 1);
frame1_gray = rgb2gray(frame1);

% Initialize optical flow object
% opticFlow = opticalFlowHS;
opticFlow = opticalFlowFarneback;
% opticFlow = opticalFlowLK;

figure;
h = imshow(frame1_gray); hold on;
h1 = quiver(0,0,0,0,'r', 'MaxHeadSize', 1.5, 'LineWidth', 1.5);

while hasFrame(video)

 % Read the next frame and convert to grayscale
    frame2 = readFrame(video);
    frame2_gray = rgb2gray(frame2);

     [rows, cols] = size(frame2_gray); hold on;

    % Compute optical flow between the current and previous frames
    flow = estimateFlow(opticFlow, frame2_gray);

    flow_x = flow.Vx;
    flow_y = flow.Vy;

    % Compute the magnitude of the flow vectors to detect moving objects
    flow_magnitude = sqrt(flow.Vx.^2 + flow.Vy.^2);

    % Update the figure with the current frame
    set(h, 'CData', frame2_gray);  % Update the frame
    set(h1,'UData',flow_x,'VData',flow_y,'XData',repmat((1:cols), rows,
1),'YData',repmat((1:rows)', 1, cols));
    %quiver(repmat((1:cols), rows, 1), repmat((1:rows)', 1, cols), flow_x, flow_y, 'r',
'MaxHeadSize', 1.5, 'LineWidth', 1.5);
    title(['Frame ' num2str(video.CurrentTime)]);
    drawnow;
end
```

Task 4.4 Modify the code obtained in Task 4.3 to visualize the optical flow in each frame using discussed method. Create three video sequences that show object tracking using thresholding approach, and visualize motion using vector field.

Tracking using Harris corner features

The tracking can be carried out using local features, like the ones computed by Harris corner detector. Each of the corners are tracked independently, therefore this tracker can be applied to tracking rigid as well as deformable objects. In the lab, we will use Harris

corner detector provided in MATLAB – a function **detectHarrisFeatures()**.

To track only a selected set of feature points, first initialize the point tracker in your program. Here is an example of initialization, please remember that the settings may be different for your video example.

```
tracker = vision.PointTracker('MaxBidirectionalError', 2, ...
'NumPyramidLevels', 3, 'BlockSize', [51 51]);
```

Next, you have to read first frame, convert it into a grayscale image and detect corner points using one of the methods you know after previous exercise, e.g. Harris corner detector.

```
prevframe = read(video,1);
prevFrame = rgb2gray(frame);

corners = detectHarrisFeatures(prevFrame, 'MinQuality',0.001);
initialCorners = corners.Location; % Get the corners coordinates
```

You can chose the best features to be tracked as shown in the previous laboratory exercise. Next you have to set the initial tracking point of the tracker to the ones, you found using corner detector.

```
initialize(tracker, points.Location, prevFrame);
```

Next, in the loop you process frames of the video sequence one by one as in the case of all previous tracking methods. To track the corners detected in the previous step, use the following code:

```
[points, isFound] = step(tracker, currFrame);
```

The results of tracking can be displayed using a simple code:

```
imshow(currFrame);
hold on;
plot(points(:, 1), points(:, 2), 'go');
hold off;
drawnow;
```

Here, is the example of the code of tracking using feature points only. Note that there is a piece of code that checks if detected features are not outside the boundaries of the image. It prevents the points to be lost when object it the boundary. However, if in your video the object does not touch the boundary as it moves, this part of the code is not necessary.

```
clc
clear all
close all
% Load the video frames
vid = VideoReader('bouncing_ball.mp4');
numFrames = vid.NumberOfFrames;
height = vid.Height;
width = vid.Width;

% Initialize the point tracker
tracker = vision.PointTracker('MaxBidirectionalError', 2, ...
'NumPyramidLevels', 3, 'BlockSize', [51 51]);
```

```
% Read the first frame
prevFrame = rgb2gray(readFrame(vid));

% Detect good features to track in the first frame
points = detectHarrisFeatures(prevFrame, 'MinQuality', 0.001);

% Initialize the point tracker with the detected points
initialize(tracker, points.Location, prevFrame);

% Create a figure to display the results
figure;

% Loop through the frames
for i = 2:numFrames
    % Read the current frame
    currFrame = rgb2gray(readFrame(vid));

    % Track the points using the point tracker
    [points, isFound] = step(tracker, currFrame);

    % Boundary checking
    for j = 1:size(points, 1)
        if points(j, 1) < 1 || points(j, 1) > width || points(j, 2) < 1 || points(j,
2) > height
            isFound(j) = false;
        end
    end

    % Remove points that are outside the image boundaries
    points = points(isFound, :);

    % Display the results
    imshow(currFrame);
    hold on;
    plot(points(:, 1), points(:, 2), 'go');
    hold off;
    drawnow;

    % Update the previous frame
    prevFrame = currFrame;
end
```

Task 4.5 Write a program that performs a sparse optical flow motion estimation for a video. The program should work with one as well as many objects represented by their local features. Write a helper function to visualize trajectories and velocities of points. Check the function on special examples, e.g. the object which undergoes translation should have the same velocity for all its chosen local features, rotating objects' features should behave according to rotational motion kinematics etc.

Another approach to feature tracking is a use of feature descriptors that have been briefly discussed in the previous lab exercise. This approach uses invariant feature descriptors to track a patches of image which may undergo linear transformations. As an example, we will implement the SURF feature descriptor.

Again, the task begins with initializing a videoReader object and reading first frame of the sequence, changing it into grayscale.

Next, you have to detect SURF features and descriptors on the first frame using proper parameters as discussed in the previous lab instruction.

```
% Detect SURF features in the first frame
surfPoints = detectSURFFeatures(grayFrame);

% Extract SURF descriptors
```

```
[features, validPoints] = extractFeatures(grayFrame, surfPoints);
```

You can visualize the features as follows

```
figure;
imshow(grayFrame);
hold on;
plot(validPoints.selectStrongest(100), 'showOrientation', true);
title('SURF Key Points in First Frame');
hold off;
```

Next, in the loop, you have to detect features in the following frames of the sequence. Because the features are detected independently on each frame, they have to be matched between two consecutive frames. The code for this approach to feature tracking:

```
clear all
clc
close all

video = VideoReader('dvdscreensaver.mp4');

frame = read(video,1);
grayFrame = rgb2gray(frame); % Convert to grayscale

% Detect SURF features in the first frame
surfPoints = detectSURFFeatures(grayFrame);

% Extract SURF descriptors
[features, validPoints] = extractFeatures(grayFrame, surfPoints);

figure;
imshow(grayFrame);
hold on;
plot(validPoints.selectStrongest(100), 'showOrientation', true); hold on
title('SURF Key Points in First Frame');

index = 1;

% Track the features in subsequent frames

figure,
 h = imshow(grayFrame); hold on
 h1 = plot(0,0,'*');
 title('Tracking objects');

while hasFrame(video)
    % Read the next frame
    frame = readFrame(video);
    grayFrame = rgb2gray(frame); % Convert to grayscale

    % Detect SURF features in the current frame
    surfPointsCurr = detectSURFFeatures(grayFrame);

    % Extract SURF descriptors from the current frame
    [featuresCurr, validPointsCurr] = extractFeatures(grayFrame, surfPointsCurr);

    % Match the features from the previous and current frames
    indexPairs = matchFeatures(features, featuresCurr);

    % Get the matched points
    matchedPointsPrev = validPoints(indexPairs(:, 1), :);
    matchedPointsCurr = validPointsCurr(indexPairs(:, 2), :);

    matches{index} = matchedPointsCurr.Location;

    % Display the matched features
   set(h,'CData',grayFrame); hold on

set(h1,'XData',matchedPointsCurr.Location(:,1),'YData',matchedPointsCurr.Location(:,2));
```

```
    drawnow;

     % Update the previous points and features for the next frame
     features = featuresCurr;
     validPoints = validPointsCurr;

     index = index + 1;

end
```

Task 4.6 Write the program that tracks object features based on its local feature points. Check the invariance of tracked points to the different types of motion that object may undergo – translation, rotation, change of distance to the camera, change of viewing angle. Test how well the method tracks local feature in the case of this types of object's motion.

Additional Tasks

Task 4.7 Write a tracker program for tracking a number of object in the binary image. The objects and background should be such that it is possible to extract objects using thresholding operation. For each object you should plot its trajectory of its centroid. Also compute velocity vector for each tracker object.

Task 4.8 Modify your correlation based tracker that it may track a number of pattern simultaneously in the same video sequence. Draw trajectory of each tracked object in the first frame of the sequence. Compute velocity vectors for each of the objects.

Task 4.9 Analyze the shape of a deformable body in the video sequence. Divide an image of the body into template windows and track each of the windows using normalized cross correlation tracker. In order to simplify the computation you may restrict the search region for each template. Plot the shape of the body in a given frame in a graph.

Task 4.10 Write a program that carries out a tracking of local features of deformable object. Based on the results make visualization of object's change of shape across the frame of the sequence. How such approach can be applied for classification problem, for example, in a gesture recognition algorithm?

Task 4.11 Write a program that classifies objects based on their behavior in the video sequence. The feature vector may contain features associated with geometry of objects as well as features describing their kinematics e.g. direction of motion or velocity, if such approach is suitable for analyzed video sequence.