IHA-4206 Mechatronics & Robot Programming
Tracking and Detection Using Computer vision

Learning Outcomes

In this lab, you will learn

- Sending and receiving image messages through a ROS network
- Color Blob detection using HSV color model
- Face detection and tracking using KLT algorithm

ROS Image messages

Boot up Matlab (currently only available on the windows machines). Matlab should have Robotics System, Image Processing, Support Package for USB Webcams and Computer Vision System toolboxes pre-installed. Install the toolboxes using the add-on explorer if this isn't the case.

Images are transmitted through a ROS network using Image messages. There are two types of Image messages:

```
'sensor\_msgs/Image'
'sensor\_msgs/CompressedImage'
```

Both of theses can be read using img = readImage(msg) function in Matlab.

Create a new main script, and start the script by setting up a global ROS node using the command:

```
rosinit
```

Next, create a ROS publisher which will be publishing the webcam feed into our ROS-network.

```
camPub = rossubscriber('/camera', 'sensor_msgs/Image');
```

Here *'/camera'* is the topic we are publishing to and *'sensor_msgs/Image'* is the message type.

We also need to create a subscriber for the camera:

```
camSub = rossubscriber('/camera', @color_blob_tracking);
```

Here *@color_blob_tracking* is our callback function (described in the next part).

For visualization, we'll also create a global VideoPlayer object.

```
global videoPlayer
videoPlayer = vision.VideoPlayer();
```

Use the pre-made function *webcam_pub.m* to publish the webcam feed

```
webcam_pub(camPub);
```

Finally, finish the script by shutting down the global node using the command

```
rosshutdown
```

Color blob detection using HSV color model

HSV color model

HSV (Hue, Saturation, Value) is an alternative representation of the RGB color model, where colors of each hue are arranged in a radial slice. It was designed to more closely align with the way human vision perceives color-making attributes. See:
`https://en.wikipedia.org/wiki/HSL_and_HSV`

Color blob detection

Start by creating a function file and name it *color_blob_tracking.m*. When finished, this function can be tested with the included *test_colorBlobTracking.m* file. After testing, we'll be using this as a callback function for our subscriber, i.e. the subscriber will execute the function every time it receives a message.

The function will take 2 input arguments (and return no outputs) by default:

```
function MyCallbackFunction(src,msg)
```

src is the associated subscriber object, and *msg* is the received message object. In this case we are only interested in the received message, so you can replace the first input argument with ~(i.e. ”tilde” or ”matomerkki in Finnish”).

The first thing the function should do, is to convert the Image message into a readable RGB image. Do this using readImage(msg) function.

Next we'll define some parameters for extracting the color blob from our image. We need to set the minimum and maximum Hue (color region in the “Hue radial slice”), minimum saturation (low saturation values are close to white) and minimum value of color (low values are close to black).

```
hueMin = 6/12;
hueMax = 10/12;
satMin = 0.4;
valMin = 0.1;
```

The values for *hueMin* and *hueMax* above are for our test video in *test_colorBlobTracking.m*. You will need to change these when you implement the final blob detector.

Next, we need to convert our RGB image into a HSV image. This can be done using the rgb2hsv(RGBimage) function. The three channels (3rd dimension) in the output image are the HSV channels respectively.

Your task: Threshold each channel to find pixels with HSV values within the limits defined in the ”hueMin, hueMax, satMin, and valMin” parameters above. As a results, you will obtain three different binary images, which you need to combine into a single binary image using the logical and (& in Matlab), ie. to find out pixels that meet all three requirements. Note that for the Hue channel you have to compare the values so that they are between the minimum and maximum threshold, for others, only minimum is set.

Now that we have our filtered binary image (where ideally our color blob is the only visible region), we can find contiguous regions using the regionprops(BW,properties) function. You can use the code below.

```
blobs = regionprops(filtered_img, {'Centroid','Area','EquivDiameter'});
```

Next, we will choose the region with the largest area (which is presumably our tracked blob). A single if-else statement is used to check if any contiguous regions were detected. Go ahead and copy the code below.

```
if isempty(blobs)
    % No blobs were detected
    c = [];
    mag = [];
else
    % Choose the region with the largest area
    % (presumably the tracked object)
end
% Visualize
```

Here *c* is the centroid and *mag* is the magnitude (radius) of the region with the largest area.

Your task: Find the index of the blob with the largest area, and store the Centroid and EquivDiameter/2 (radius) of the blob into c and mag (Hint: Check the Matlab documentation for max() function). Do all this under the else-part of the if-else-statement.

Use the code below to draw a circle around the largest detected blob. Again, do this under the else-statement. Here the input argument img is the original image.

```
img = insertShape(img, 'Circle',[floor(c),mag], 'LineWidth',5, 'Color','red');
```

Finally, visualize the frame with the previously created VideoPlayer object (If you're interested, you can create another VideoPlayer object and also display the binary threshold image, as shown below).

```
global videoPlayer
step(videoPlayer, img)
if ~videoPlayer.isOpen
    rosshutdown
end
```

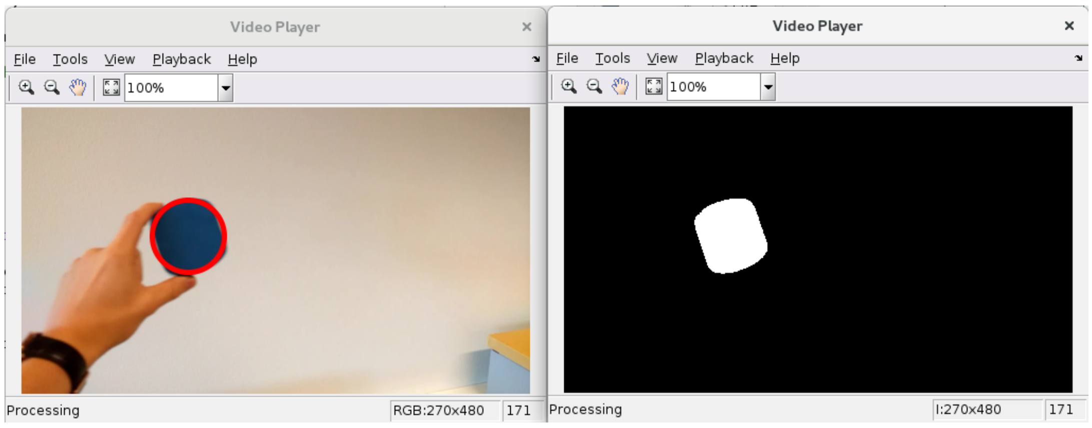

Figure 1: Detected blue blob and binary threshold image

Finally, run *test_colorBlobTracking.m* to test your tracking function. note: it is useful to study the content of *test_colorBlobTracking.m* function for future tasks.

Blob detection from camera feed

After you have tested and confirmed that your function works properly with the pre-recorded video, the next task is to run it with live camera feed.

Your task: Revisit the instructions given in Section "ROS Image messages" and implement a system that receives frames from the camera as they arrive, detects the blobs, and displays them in the video player. Note that that hueMin and hueMax parameter might not be correct for your camera setup and need tuning. Start testing with similar simple setup as in the "test video" above. Then move gradually to harder backgrounds to explore the limits of the color blob tracking method.

Face detection and tracking using the KLT algorithm

Kanade-Lucas-Tomasi (KLT) algorithm

The KLT algorithm is a feature tracker, which first detects a set of feature points and then tracks these using optical flow. In this exercise, we will use Viola-Jones detection algorithm to find a face in an image and then forward to the KLT algorithm that follows it. See `https://en.wikipedia.org/wiki/Kanade\OT1\textendashLucas\OT1\textendashTomasi_feature_tracker`
`https://en.wikipedia.org/wiki/Viola\OT1\textendashJones_object_detection_framework`.

In this work, the detection and tracking functions are implemented using Matlab Computer Vision toolbox. The exercise material contains functions *face_detector.m* and *face_tracker.m* that implement the face detection and tracking parts, respectively. Additionally, the function *draw_results.m* is provided for drawing the tracking results into the rgb frame.
Your task: Open these functions and study how they work.

Face detection and tracking

Your task: Create a new function file called face_finder.m, which will be used as a callback function for the subscriber (created similarly in the previous part of the exercise). The function *face_finder.m* takes ROS-image messages as an input.

Your task: Start the function by adding a persistent structure called data (persistent variables are local to the function, but their values are retained in memory between each call to the function). The exact format of the structure should be as follows.

```
persistent data
% Create a face detector object (Matlab Computer Vision toolbox)
data.faceDetector = vision.CascadeObjectDetector();
% Create a point tracker object (Matlab Computer Vision toolbox)
data.pointTracker = vision.PointTracker('MaxBidirectionalError', 3);
% Initialize
data.nPts = 0; % Number of currently tracked points
data.bboxPoints = 0; % Location of face bounding box corners
data.oldPoints = 0; % Locations of the feature points in previous frame (needed
    for tracking)
```

After that, convert the image message into a readable RGB image using the readImage(msg) function, and then create a grayscale copy of the image using rgb2gray(RGBimage).

Inside the face finder function, we will run either detector or tracker depending on the conditions. In our case, we set that if less than 10 feature points inside the face are successfully tracked we run the detector (you can try changing this limit and see what happens). Otherwise, we will update the face position using tracker.

Your task: Insert the following if-else statement to your *face_finder* function.

```
% Read frame
if data.nPts < 10
    % Run face detector
else
    % Run face tracker
end
% Display the annotated video frame
```

Complete the statement by utilising the given *face_detector.m* and **face_tracker.m** functions.

Finally, draw the detected face and feature points to the RGB frame using **draw_results.m** function. Then display the annotated video frame using our videoPlayer:

```
global videoPlayer
step(videoPlayer, videoFrame)
if ~videoPlayer.isOpen
    rosshutdown
end
```

Use *test_faceTracking.m* to test your function with pre-recorded video.

Detecting and tracking faces from camera feed

Your task: Once you have successfully tested your methods with *test_faceTracking.m* function, use your *face_finder.m* as a callback function for our subscriber and detect faces from real-time camera feed (similarly as in color blob tracking). See if you can track your own or your friends face.

Bonus task: The face detector might detect a face in a place where no face is actually present, and in the worst case the point tracker tracks stationary points somewhere in background. This means that the tracker never "releases" the points, and the actual face is left unnoticed. To prevent this, one way is to perform a new face detection regularly (e.g. every 100 frames). Implement this periodic face detection.