AI & Robotics: Lab Course
Weekly Exercise 3

Joaquim Ortiz de Haro & Jung-Su Ha
Learning & Intelligent Systems Lab, TU Berlin
Marchstr. 23, 10587 Berlin, Germany

Summer 2022

The goal of this session is to enable you to use OpenCV within your code.
The first example in **04-opencv/main.ipynb** subscribes to a webcam and starts a little loop that displays the image using opencv. The webcam is only to have more fun during coding – later your code has to run on the simulated camera. The second example in **04-opencv/main.ipynb** does our standard simulation loop, but in each iteration grabs the simulated RGB and depth image, displays them using opencv, converts them to a point cloud, and displays this point cloud in your model configuration. So this is your first example of bringing camera signals into your 3D model world.
Here you can find OpenCV documentation:

- In C++, see https://docs.opencv.org/master/, in particular the OpenCV Tutorials - Image Processing section
- For python, see https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

1 Color-based tracking from real images

The goal of this exercise is to implement a 'red object tracker'. First develop it within the webcam example.

a) Implement a color filter to find all pixels that are red-ish.

b) Display the binary mask image that indicates red pixels.

c) Find contours to this binary image, which returns the segments.

d) Display the contours by drawing them into the original RGB image.

e) Test if you can also track the falling red object in the second example of **04-opencv/main.ipynb**.

(Hint: you can google it with "opencv hsv filter".)

2 Color-based tracking in simulation

Apply the above tracker to the dropping red object in the simulated world (See the skeleton code in **e03-redBall.ipynb** in ISIS):

a) Identify the "center of red" in camera coordinates, identify the "mean red depth", and combine this to yield the 3D position estimate. Create a simple sphere in your model world that moves with this tracking estimate.

b) Is the "mean red depth" value the best estimation of the red ball's position? If not, discuss (& implement) a better method.

3 Foreground segmentation in simulation

Foreground/background is a useful technique to track movable objects. Using the code from the last exercise (that, is your solution based on the skeleton in **e03-redBall.ipynb**), first add two fixed red spheres in the scene. In this setting, a color-based object tracker would fail to detect the movable ball.

In this exercise, you have to track the ball using foreground segmentation.

a) Store the first image as background image. (Or average over several first images.)

b) For every new image filter those pixels, that are significantly different to the background.

c) Display the binary image that indicates change pixels.

d) Estimate the 3D Position of the moving ball.

Think about how one can do the same segmentation for depth. In particular, assume that the "background" has always the largest pixel depth – so when the depth of a pixel is less than before, then it must be foreground. Realize this within the simulation loop, where you have access to depth.