Using OpenCV for 🐀Science🧪

https://github.com/Aramist/opencv_tutorial

2024-09-17

Table of Contents

What is OpenCV?

- A very expansive C++ imaging processing library with a Python wrapper

- Main modules:
  - core. **Core functionality**
  - imgproc. **Image Processing**
  - imgcodecs. **Image file reading and writing**
  - videoio. **Video I/O**
  - highgui. **High-level GUI**
  - video. **Video Analysis**
  - calib3d. **Camera Calibration and 3D Reconstr**
  - features2d. **2D Features Framework**
  - objdetect. **Object Detection**
  - dnn. **Deep Neural Network module**
  - ml. **Machine Learning**
  - flann. **Clustering and Search in Multi-Dimensi**
  - photo. **Computational Photography**
  - stitching. **Images stitching**
  - gapi. **Graph API**
- Extra modules:
  - alphamat. **Alpha Matting**
  - aruco. **Aruco markers, module functionality was moved to objdetect module**
  - bgsegm. **Improved Background-Foreground Segmentation Methods**
  - bioinspired. **Biologically inspired vision models and derivated tools**
  - cannops. **Ascend-accelerated Computer Vision**
  - ccalib. **Custom Calibration Pattern for 3D reconstruction**
  - cudaarithm. **Operations on Matrices**
  - cudabgsegm. **Background Segmentation**
  - cudacodec. **Video Encoding/Decoding**
  - cudafeatures2d. **Feature Detection and Description**
  - cudafilters. **Image Filtering**
  - cudaimgproc. **Image Processing**
  - cudalegacy. **Legacy support**
  - cudaobjdetect. **Object Detection**
  - cudaoptflow. **Optical Flow**
  - cudastereo. **Stereo Correspondence**
  - cudawarping. **Image Warping**
  - cudev. **Device layer**
  - cvv. **GUI for Interactive Visual Debugging of Computer Vision Programs**
  - datasets. **Framework for working with different datasets**
  - dnn_objdetect. **DNN used for object detection**
  - dnn_superres. **DNN used for super resolution**
  - dpm. **Deformable Part-based Models**
  - face. **Face Analysis**
  - freetype. **Drawing UTF-8 strings with freetype/harfbuzz**
  - fuzzy. **Image processing based on fuzzy mathematics**
  - hdf. **Hierarchical Data Format I/O routines**
  - hfs. **Hierarchical Feature Selection for Efficient Image Segmentation**
  - img_hash. **The module brings implementations of different image hashing algorithms.**
  - intensity_transform. **The module brings implementations of intensity transformation alg**
  - julia. **Julia bindings for OpenCV**
  - line_descriptor. **Binary descriptors for lines extracted from an image**
  - mcc. **Macbeth Chart module**
  - optflow. **Optical Flow Algorithms**
  - ovis. **OGRE 3D Visualiser**
  - phase_unwrapping. **Phase Unwrapping API**
  - plot. **Plot function for Mat data**
  - quality. **Image Quality Analysis (IQA) API**
  - rapid. **silhouette based 3D object tracking**
  - reg. **Image Registration**
  - rgbd. **RGB-Depth Processing**
  - saliency. **Saliency API**
  - sfm. **Structure From Motion**
  - shape. **Shape Distance and Matching**
  - signal. **Signal Processing**
  - stereo. **Stereo Correspondance Algorithms**
  - structured_light. **Structured Light API**
  - superres. **Super Resolution**
  - surface_matching. **Surface Matching**
  - text. **Scene Text Detection and Recognition**
  - tracking. **Tracking API**
  - videostab. **Video Stabilization**
  - viz. **3D Visualizer**
  - wechat_qrcode. **WeChat QR code detector for detecting and parsing QR code.**
  - xfeatures2d. **Extra 2D Features Framework**
  - ximgproc. **Extended Image Processing**
  - xobjdetect. **Extended object detection**
  - xphoto. **Additional photo processing algorithms**

- A very expansive C++ imaging processing library with a Python wrapper
  - Error messages are usually aren't very informative

```
>>> H = cv2.getPerspectiveTransform(src_points[..., None, None], dst_points)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
cv2.error: OpenCV(4.8.0) /Users/xperience/GHA-OpenCV-Python/_work/opencv-python/
opencv-python/opencv/modules/imgproc/src/imgwarp.cpp:3528: error: (-215:Assertio
n failed) src.checkVector(2, CV_32F) == 4 && dst.checkVector(2, CV_32F) == 4 in
function 'getPerspectiveTransform'
```

  - Documentation is C++ oriented with Python on the sidelines

◆ getPerspectiveTransform() [2/2]

```
Mat cv::getPerspectiveTransform ( InputArray src,
                                  InputArray dst,
                                  int        solveMethod = DECOMP_LU
                                )
```

Python:

cv.getPerspectiveTransform(src, dst[, solveMethod]) -> retval

Installing OpenCV

`pip install opencv-python`

Data Structures

Data Structures

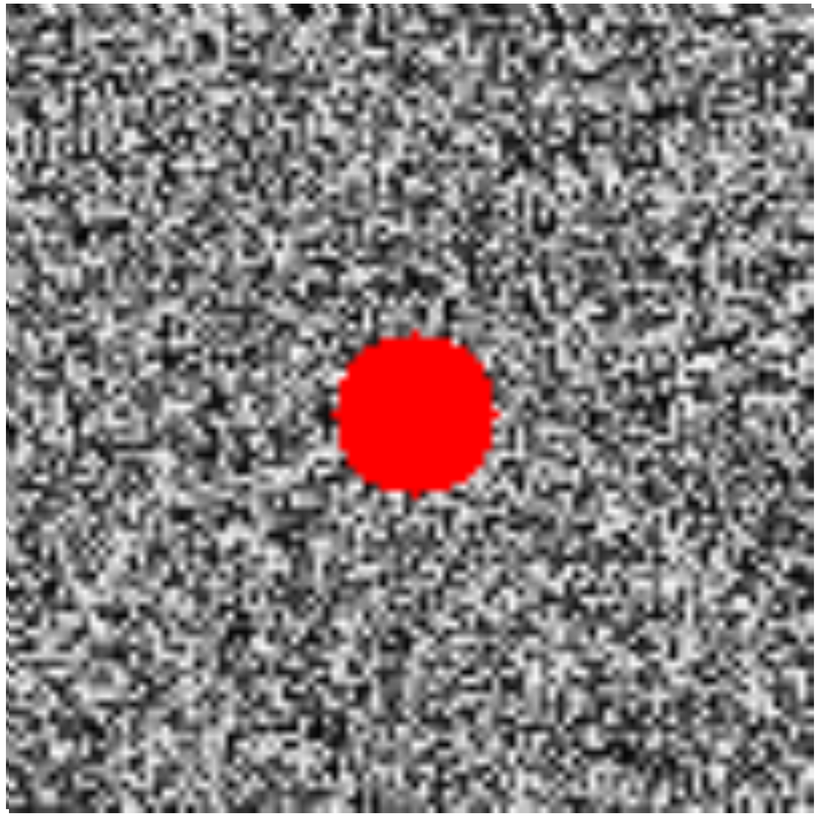
- One of the more confusing things to OpenCV beginners
  - Images are NumPy arrays
    - C-ordering (row index first, then column)
      - Img[y, x] -> pixel at (x, y)
    - Y=0 at top of image (like a matrix in mathematics)
  - When manipulating images directly, use (y, x) order (e.g. img[y, x, :] = (0, 0, 255))
  - When calling OpenCV functions, use (x, y) order (e.g. cv2.circle(img, (x, y), ...))

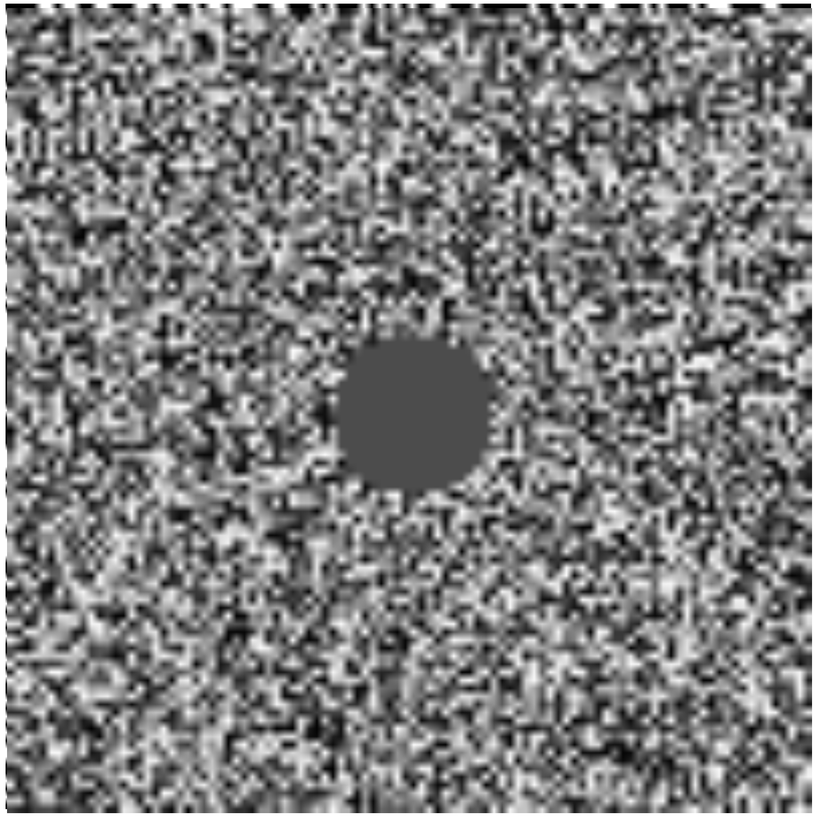
Data Structures

- Grayscale images have shape (height, width) and dtype np.uint8
- Colored images have shape (height, width, num_channels) and (usually) dtype np.uint8. Some functions support greater bit depths
- Non-image matrices are usually CV_32F (OpenCV's internal name for float32 arrays)
  - Will show up if you are fitting rectangles, finding contours, or transforming images

Common Image Operations

Common Image Operations: Color Space Conversion





```
graycircle = cv2.cvtColor(circle, cv2.COLOR_BGR2GRAY)
```

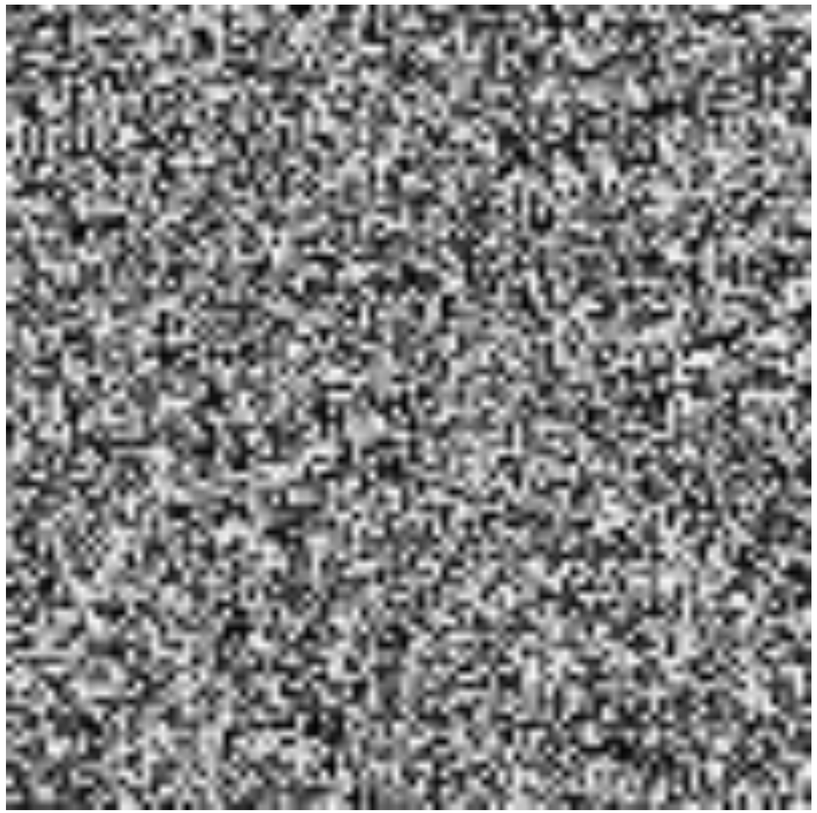
Common Image Operations: Color Space Conversion

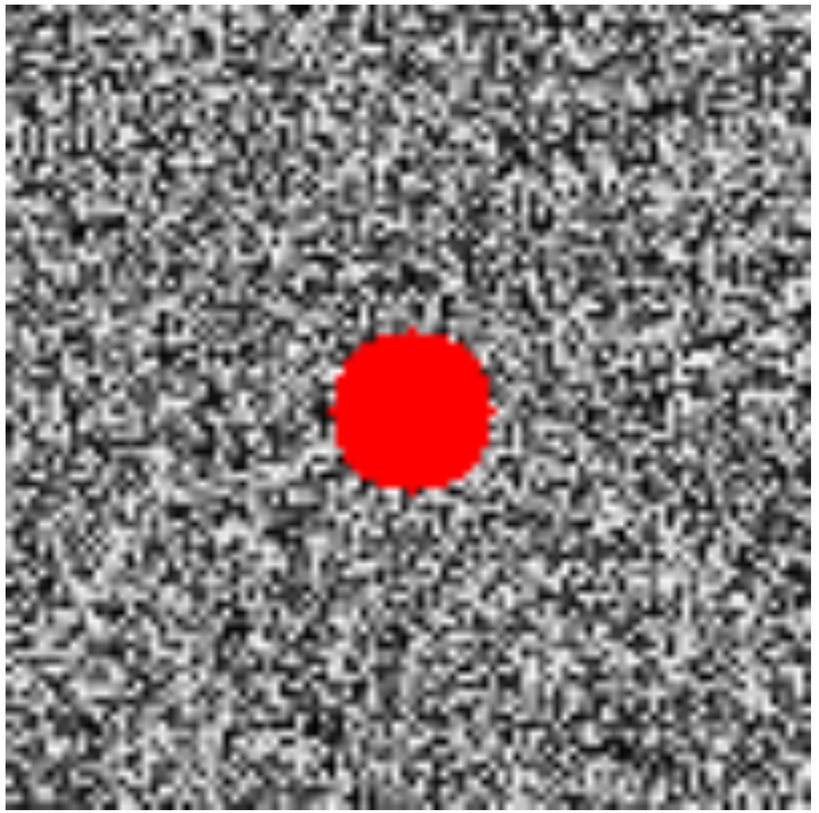
- Arguably the most useful function in object detection
  - "A lot of problems are easily solved if you choose the right basis" —Eero, probably
- Supports every color space you know, and then some
- The only (multichannel) OpenCV functions which make any assumptions about the image's color space are imread and imwrite, which assume BGR or grayscale

Common Image Operations: Drawing

<- 100px ->

<- 100px ->



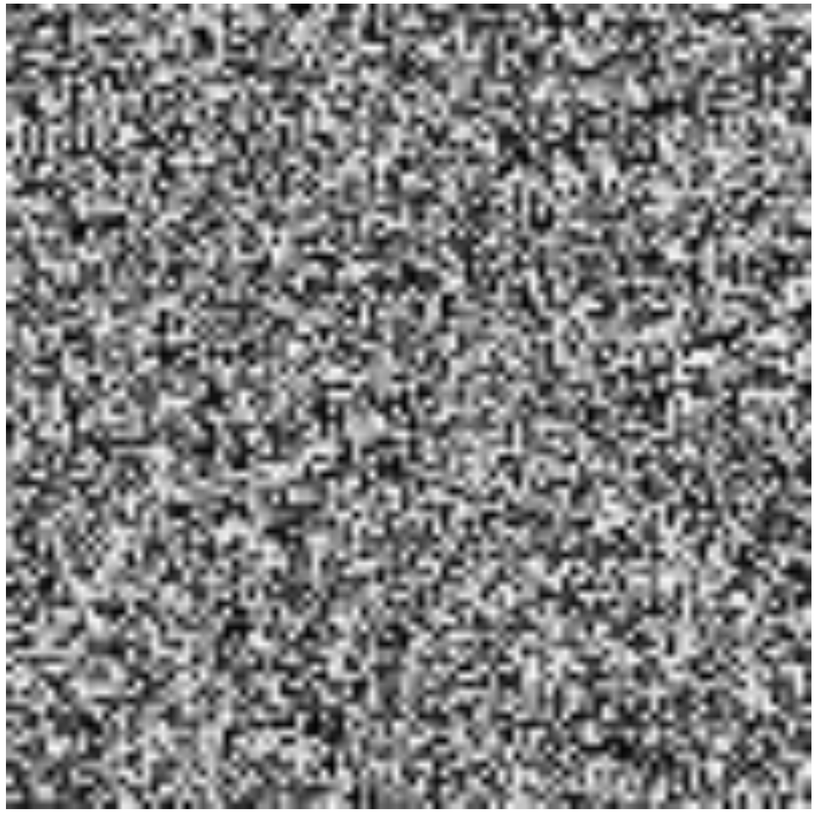


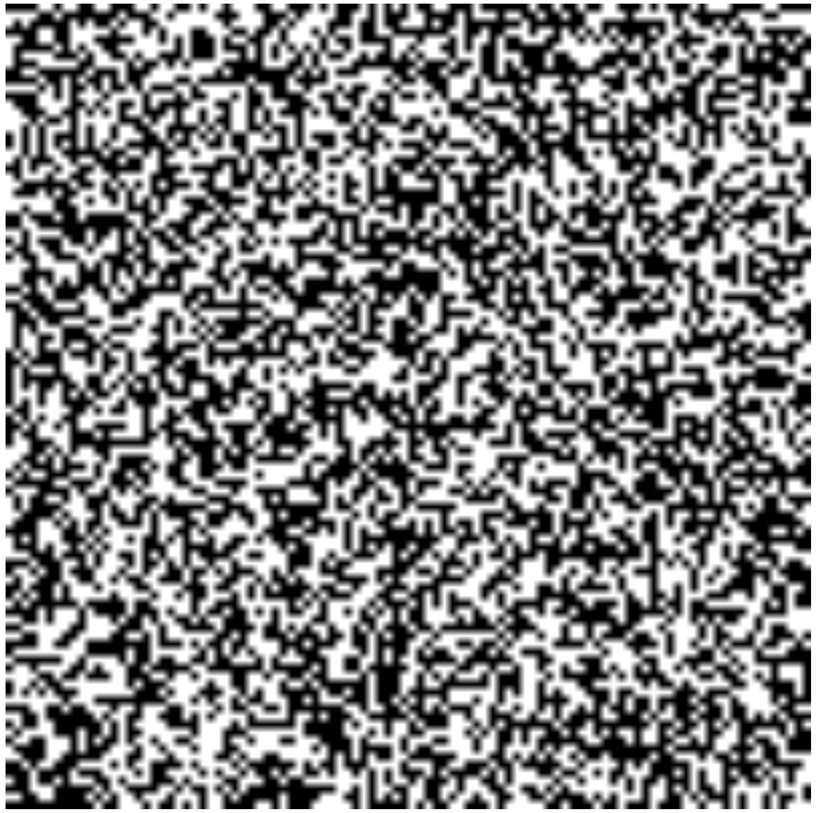
```
im_w_circle = cv2.circle(noise, (50, 50), 10, (0, 0, 255), -1)
```

Common Image Operations: Drawing

- Also see: cv2.drawContours, cv2.putText, cv2.ellipse, cv2.line, cv2.rectangle

Common Image Operations: Thresholding



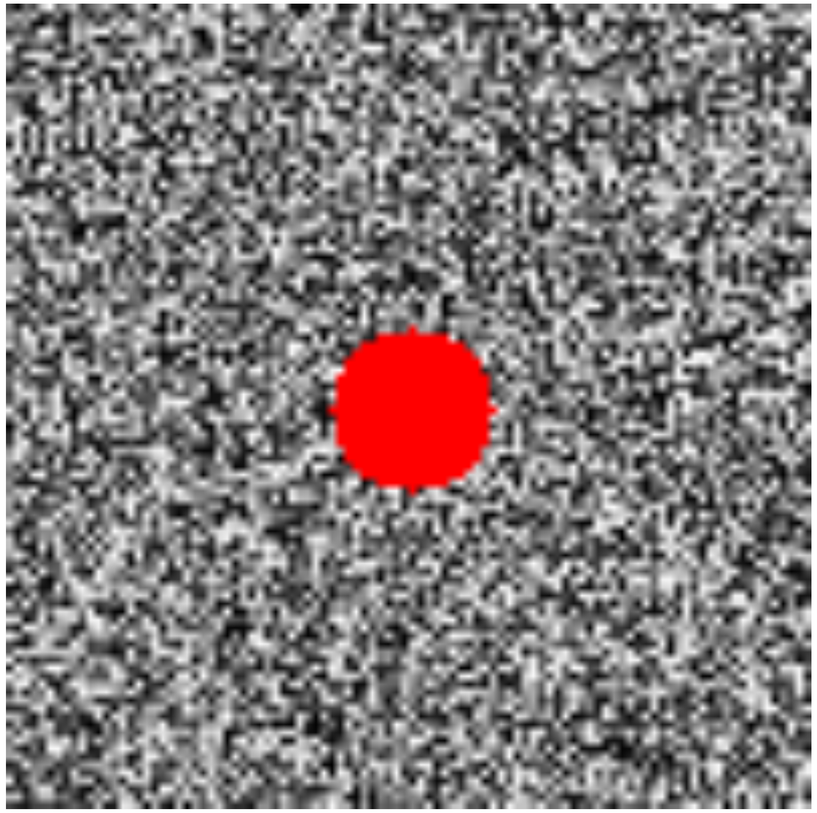


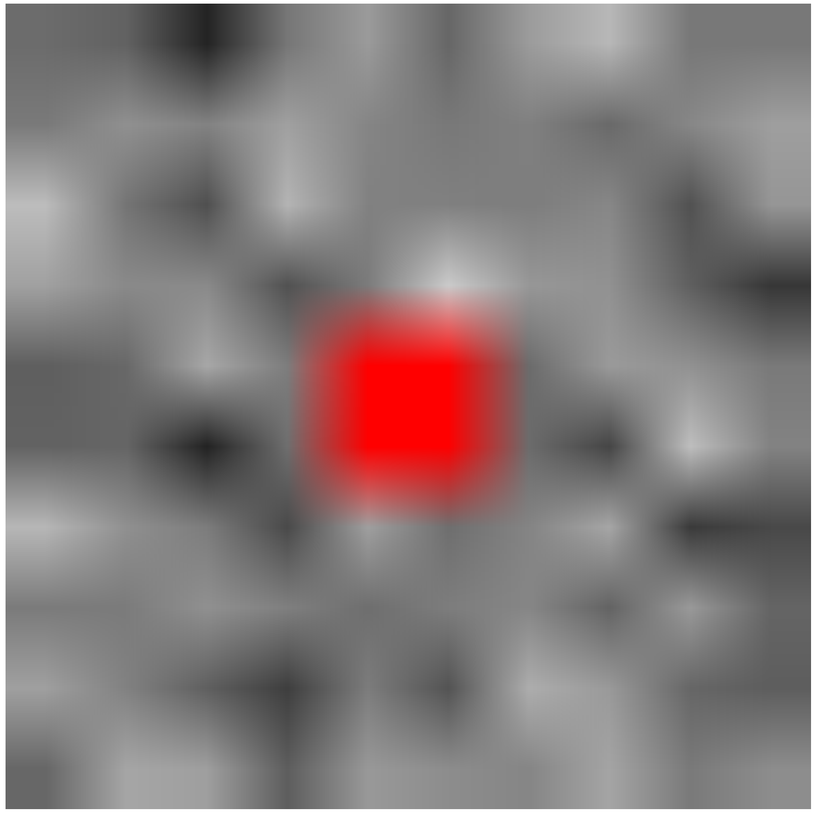
```
_, thresh = cv2.threshold(im, 127, 255, cv2.THRESH_BINARY)
```

Common Image Operations: Thresholding

- Good for masking images
- Has many modes (see: cv::ThresholdTypes)
- In practice I usually just use NumPy unless I'm doing adaptive thresholding (see: cv2.adaptiveThreshold)
  - The previous operation can be rewritten in many ways:
    - Im = np.where(im > 127, 255, 0).astype(np.uint8)
    - Im = (im > 127).astype(np.uint8) * 255
    - ...

Common Image Operations: Resizing





```
circle_small = cv2.resize(circle, (10, 10))
```

Common Image Operations: Resizing

- Not much to say
- Supports several interpolation methods, default is linear

Common Image Operations: Erode & Dilate


```
kernel = np.ones((5, 5), dtype=np.uint8)
dilation = cv2.dilate(im_with_hole, kernel)
eroded = cv2.erode(dilation, kernel)
```

Common Image Operations: Erode & Dilate

- Used all the time in object detection to:
  - Patch holes within solid objects (dilate then erode)
  - Remove small artifacts while preserving large objects (erode then dilate)
- The structuring element (improperly labeled kernel in the last slide) determines the "rule" for which pixels get turned on or off

Honorable Mentions:

- Blur: cv2.GaussianBlur, cv2.blur
- Save your arrays: np.ascontiguousarray

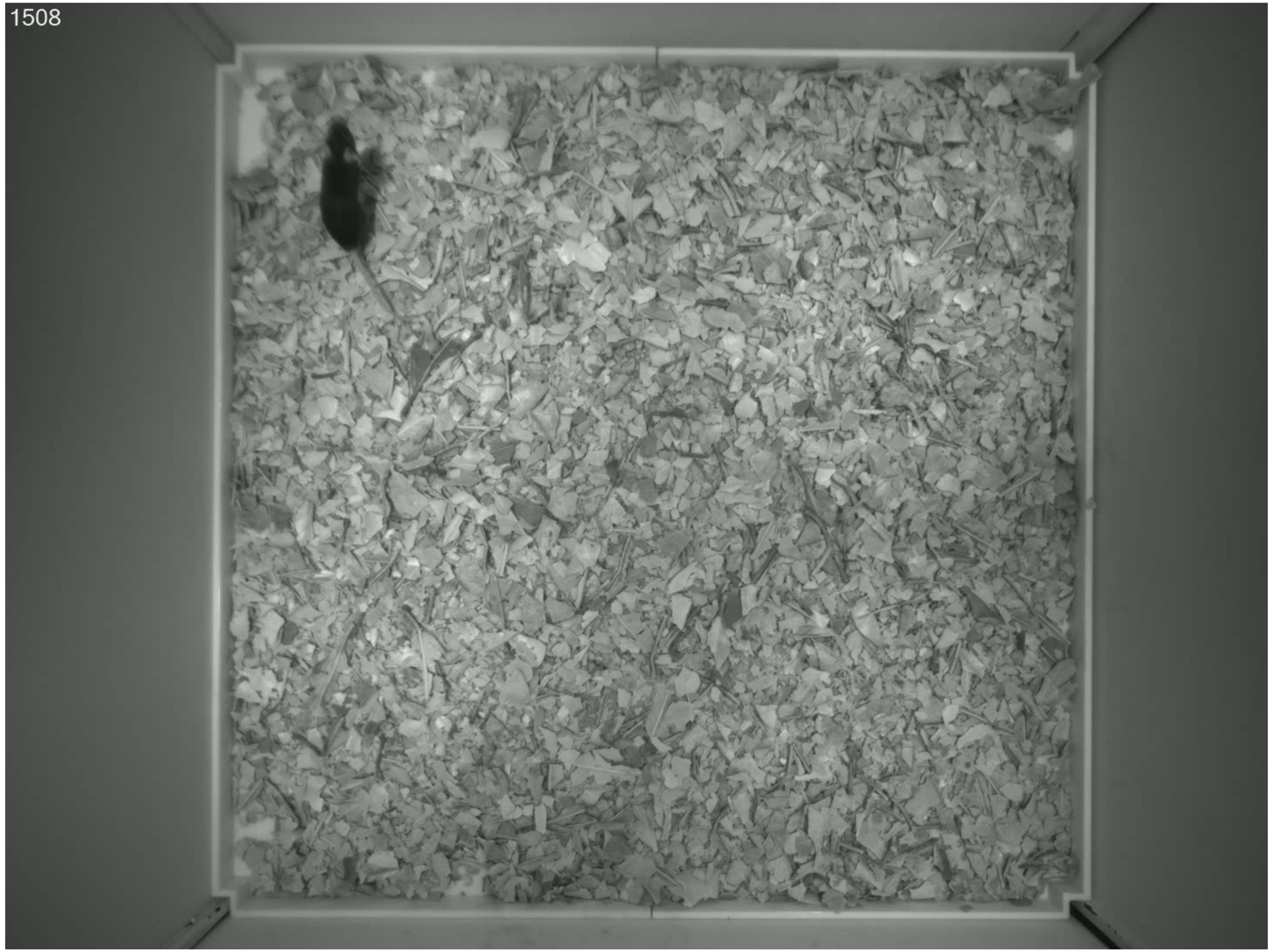
Hands on: Basic Object Detection

Hands on: Contours & Basic Object Detection

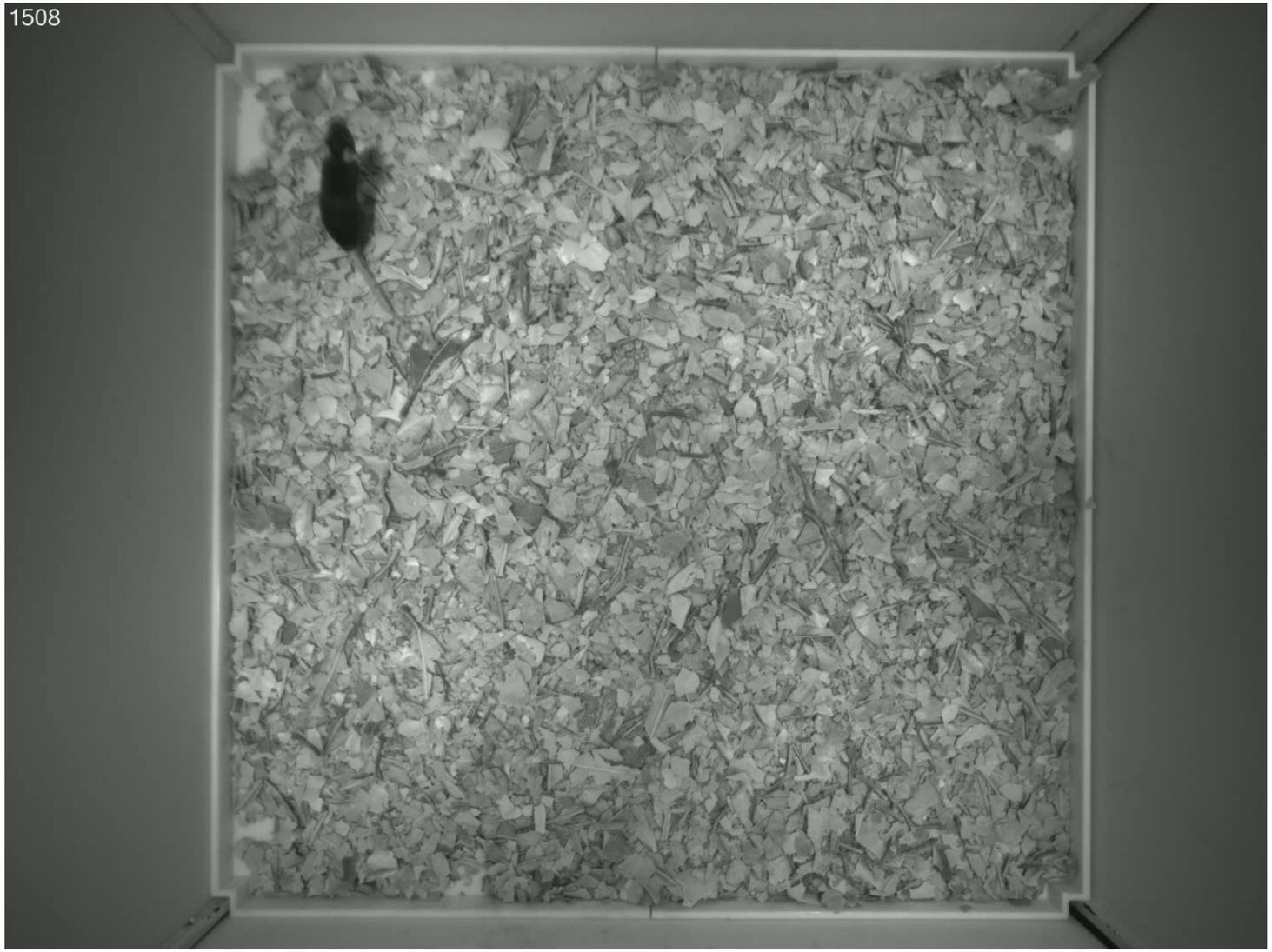
- Detect contours in binary images: cv2.findContours(image, mode, approx_method)
  - Mode determines how OpenCV handles the contour hierarchy. For basic object detection, cv2.RETR_LIST or cv2.RETR_EXTERNAL are fine. If you care about whether a contour lies within another contour, consider reading the wiki (see: cv::RetrievalModes)
  - approx_method can usually be ignored, determines how contour edges are approximated

Hands on: Contours & Basic Object Detection

How can we find the mouse?



Hands on: Contours & Basic Object Detection



The mouse is very dark, relative to the background

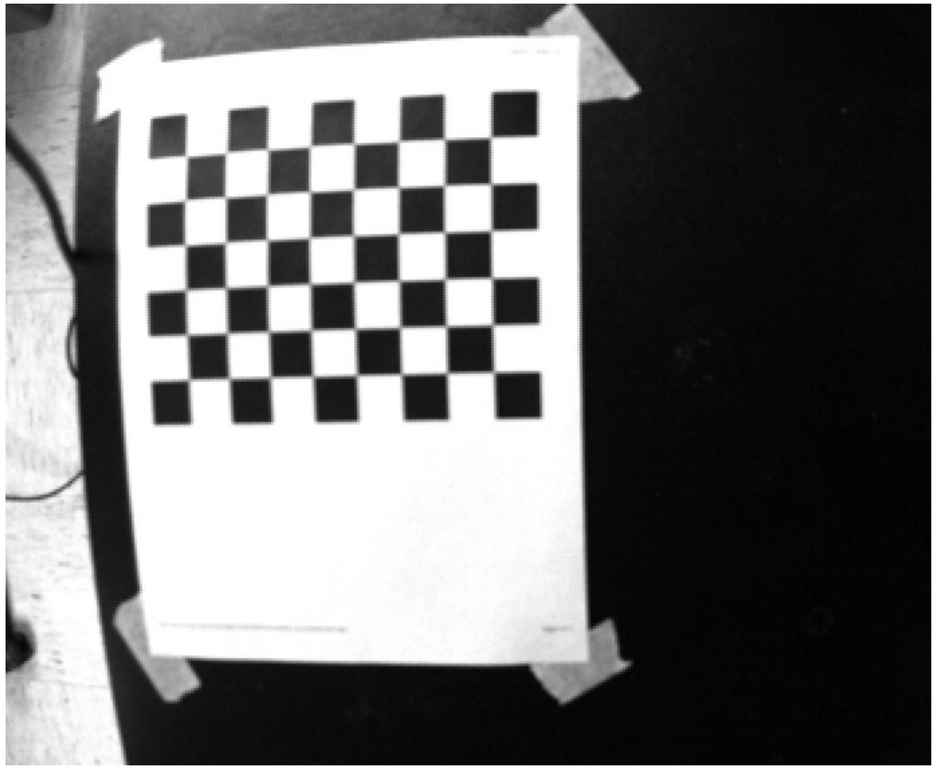
Thresholding in the value channel might be sufficient

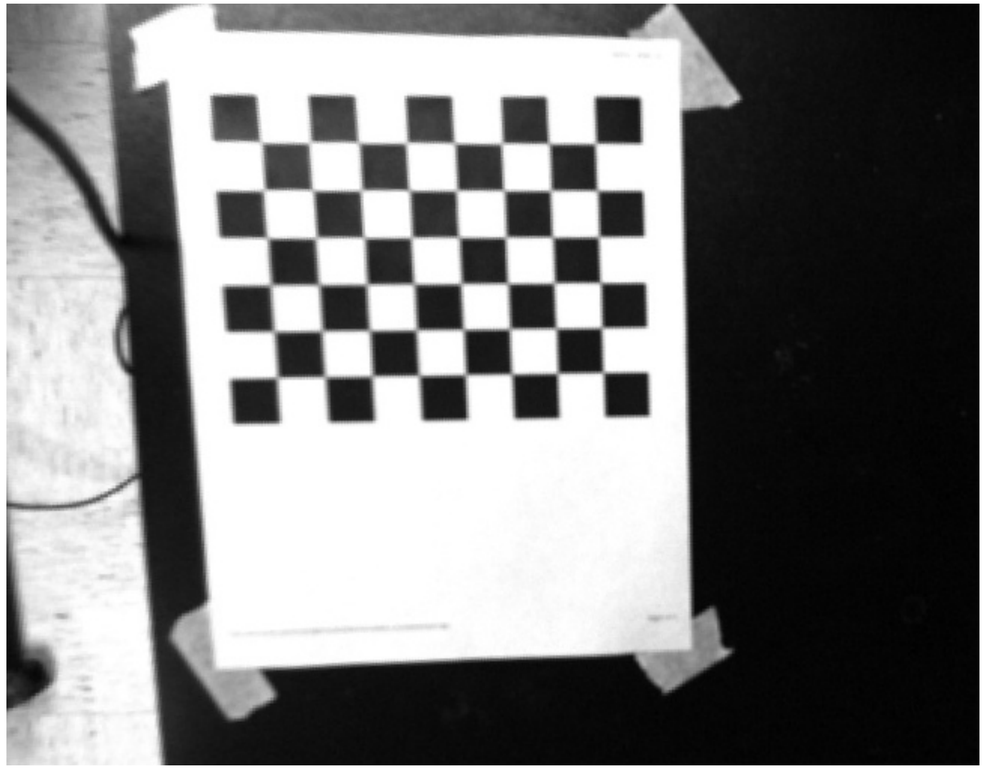
Hands on: Contours & Basic Object Detection

- Useful contour filtering strategies:
  - Compare to known shape:
    - Circles can be detected by the ratio between perimeter (cv2.arcLength) and equivalent diameter 2sqrt(contourArea / pi)
    - Rectangles and ellipses can be detected by seeing how well a best-fit shape matches the contour (cv2.fitEllipse, cv2.minAreaRect)
- Straight lines are more easily located through a Hough transform of the binary image than contour filtering

Uncommon Image Operations

Uncommon Image Operations: Distortion

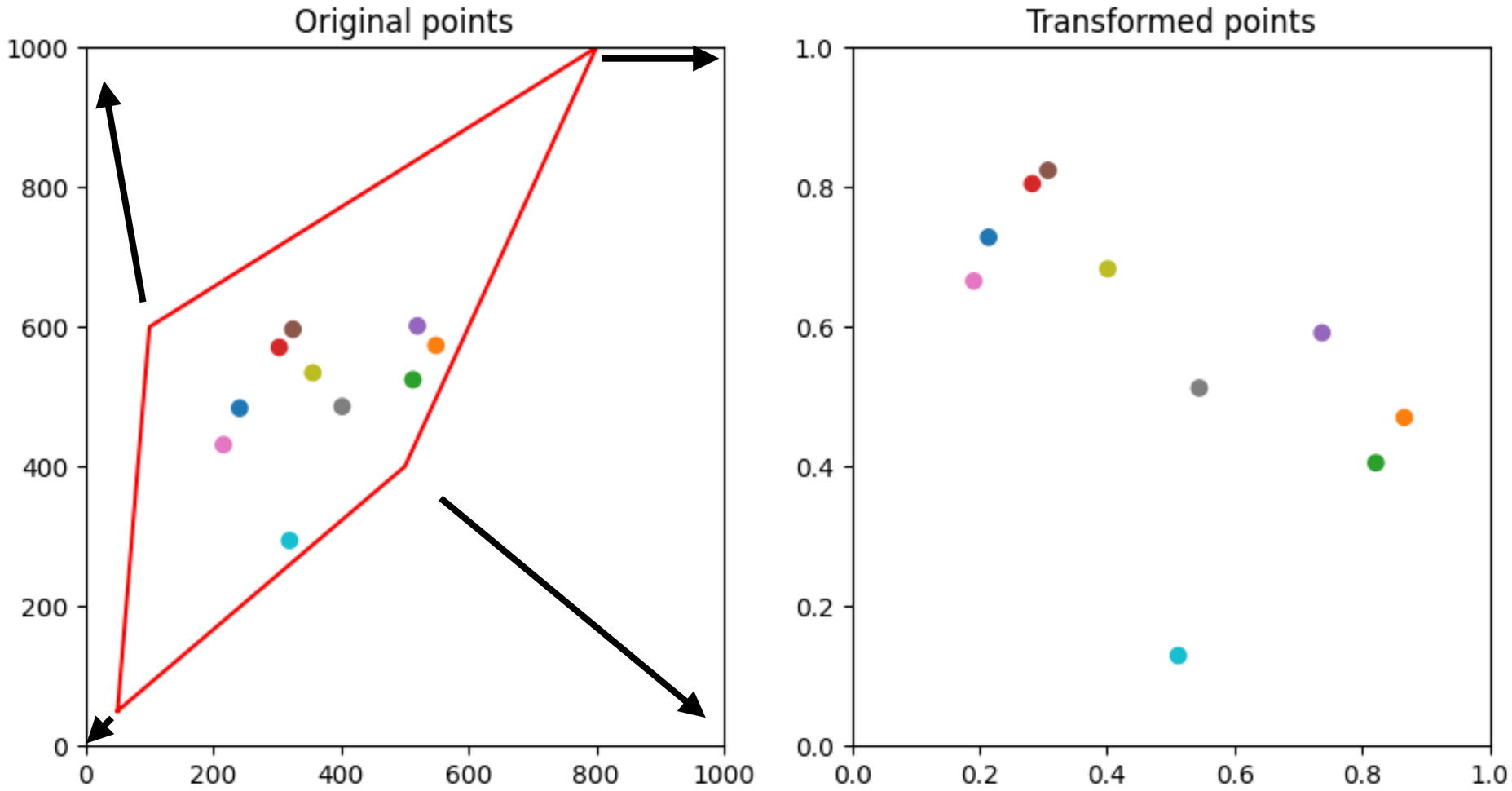




Uncommon Image Operations: Distortion

- Very situational
- https://github.com/Aramist/camera-calibration

Uncommon Image Operations: Perspective



Uncommon Image Operations: Perspective

- Solves for the affine transformation which best maps one quadrilateral to another
- Useful for 2D coordinate frame conversions
- Can warp entire images!
- See cv2.warpPerspective

```
src_points = np.array([
    [50, 50],
    [500, 400],
    [800, 1000],
    [100, 600],
]).astype(np.float32)

dst_points = np.array([
    [0.0, 0.0],
    [1.0, 0.0],
    [1.0, 1.0],
    [0.0, 1.0],
]).astype(np.float32)

H = cv2.getPerspectiveTransform(src_points, dst_points)

test_points = np.random.uniform(np.array([200, 200]), np.array([700, 700]), (10, 2))
test_points = np.concatenate([test_points, np.ones((10, 1))], axis=1)  # add 1 to account for the bias term
transformed_points = np.einsum('ij,bj->bi', H, test_points)
transformed_points = transformed_points[:, :2] / transformed_points[:, 2:3]
```

Working with GUIs

Working with GUIs

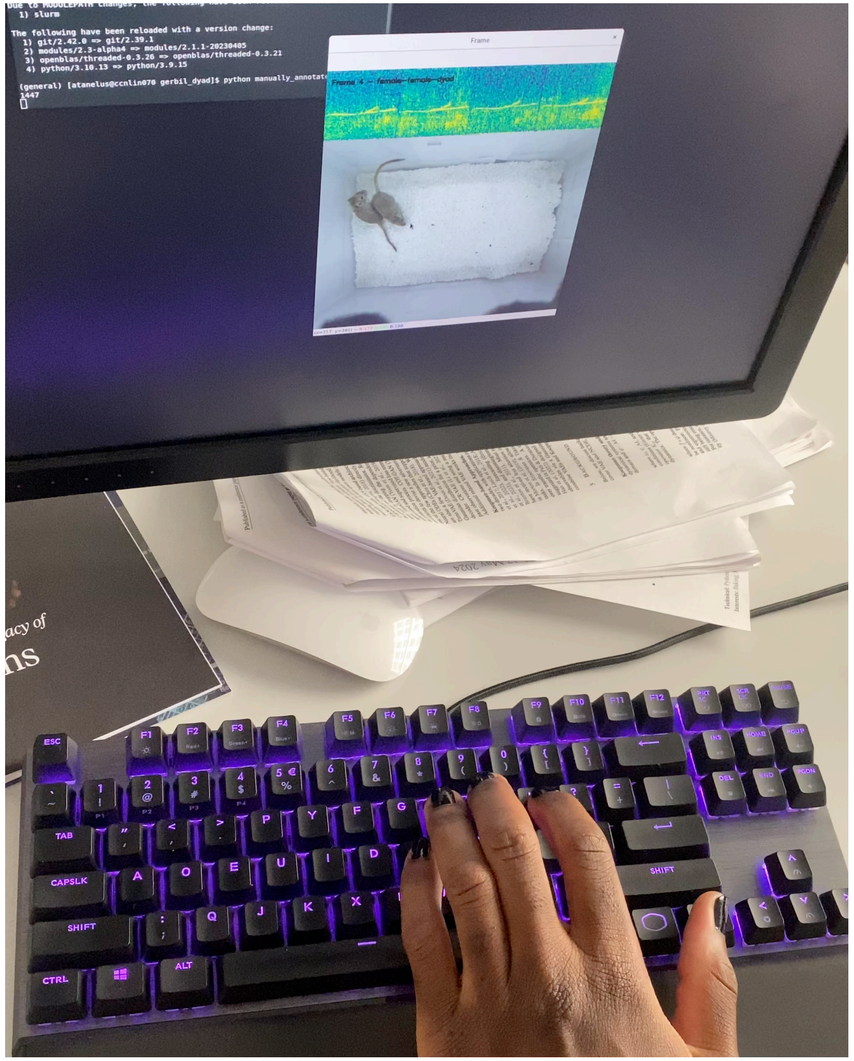
- Create a window: cv2.namedWindow(name, mode)
  - I usually use cv2.WINDOW_NORMAL (resizable, ratio locked)
- Show an image: cv2.imshow(windowName, image)
- IMPORTANT receive input: cv2.waitKey(max_wait_ms)
  - Receives keyboard input from the active window
  - Important because calling this prompts opencv to update the window and show any queued images
- Destroy windows: cv2.destroyAllWindows()

Working with GUIs: Advanced

- Mouse callbacks: cv2.setMouseCallback(win_name, callback_fn)
  - def callback_fn(event, x, y, flags, param)
  - Event is an enum for the event type(l click down, r click up, etc.)
  - (x,y) is the position of the mouse in the image displayed on the window
  - I've never touched the other two args

Working with GUIs: Case Study

Gerbil dyadic interaction data annotation in <300loc : https://gist.github.com



Working with Videos

Working with Videos: Reading Videos

- Videos are loaded through the cv2.VideoCapture class
- Initialize the class with a string (path to a video) or an integer (index of the webcam to use)
- Grab frames with read: returns a boolean (whether a frame was successfully grabbed) and an image
- ✨live demo✨ (what could go wrong?)

Working with Videos: Writing Videos

- Videos are written with the cv2.VideoWriter class. This is one of the more inconveniently designed features of OpenCV
- Initialize with cv2.VideoWriter(filename: str, fourcc: int, fps: float, size: tuple[int, int], isColor: bool)
  - Filename does not accept path-like objects, only str
  - If you end up with 512-byte empty videos, double check your size arg. OpenCV does not throw errors if you feed in an image of the wrong size, rather it silently fails to write.
  - If you fail to call release() at the end of your script, you will likewise end up with an empty video (on my machine)

Working with Videos: Writing Videos

- What is a fourcc code?
  - Describes the codec used to encode / compress the video. The codes available to you depend on your system and the extension you wish to save the video as. Passing in a value of -1 into the constructor will prompt OpenCV to list all that are available on your system for a given filetype.
  - I commonly use 'DIVX' for .avi videos and 'mp4v' for .mp4 (capitalization matters)
  - To get the fourcc integer code from a string, use cv2.VideoWriter_fourcc(*string_code)
- ✨live demo✨

Working with Videos: Writing Videos

- Matplotlib figures can be converted to images in OpenCV without much overhead:

```
def get_frame_buffer(figure: mpl.figure.Figure) -> np.ndarray:
    """Returns a BGR image for opencv from a matplotlib figure"""
    figure.canvas.draw()
    buf = figure.canvas.buffer_rgba()
    width, height = figure.canvas.get_width_height()
    return np.frombuffer(buf, np.uint8).reshape(height, width, 4)[..., 2::-1]
```

Challenge: Tracking Football Players