Computer vision

Ilia Nechaev

31 October 2024

What will be today

General topics:

1. Math camera model and camera calibration
2. Color spaces and their applications in computer vision
3. Inference of neural networks without TensorFlow and PyTorch

Pinhole camera model

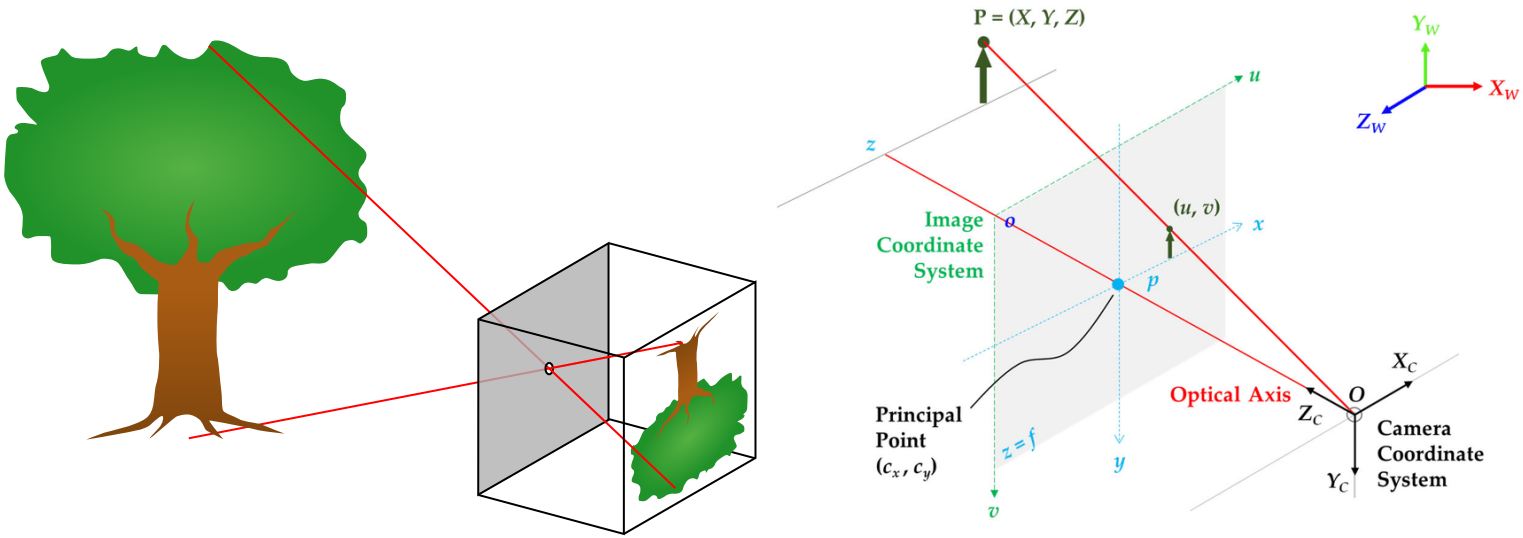

Figure: Pinhole camera

In this model we need to know the following parameters:

1. Focal length f or improved focal length (f_x, f_y) (because of the pixel aspect ratio)
2. Principal point (c_x, c_y)

In this scenario we can calculate the projection of 3D point (X, Y, Z) to 2D point (x, y) using the following formula [1]:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \cdot \frac{X}{Z} + c_x \\ f_y \cdot \frac{Y}{Z} + c_y \end{bmatrix}$$

[1] In this formula we assume that the camera is located at the origin of the coordinate system

Frame camera model

Frame camera model is a pinhole camera model with distortions.

Figure: Distortions in frame camera model

Types of distortions:

1. Radial distortion: k_1, k_2, k_3. The formula for radial distortion is:

$$x'' = x' \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y'' = y' \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

2. Tangential distortion: p_1, p_2. The formula for tangential distortion is:

$$x'' = x' + (2p_1 xy + p_2 \cdot (r^2 + 2x^2))$$
$$y'' = y' + (p_1 \cdot (r^2 + 2y^2) + 2p_2 xy)$$

Full projection formula

Assume we have:

1. Focus length (f_x, f_y)
2. Principal point (c_x, c_y)
3. Distortion coefficients $(k_1, k_2, k_3, p_1, p_2)$
4. 3D point (X, Y, Z)

Then the full projection formula is:

1. $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{bmatrix}$
2. $r = \sqrt{x'^2 + y'^2}$
3. $\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x' \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2 \cdot (r^2 + 2x'^2) \\ y' \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1 \cdot (r^2 + 2y'^2) + 2p_2 x' y' \end{bmatrix}$
4. $\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \cdot x'' + c_x \\ f_y \cdot y'' + c_y \end{bmatrix}$

Camera calibration

Camera calibration is the process of finding these coefficients:
$f_x, f_y, c_x, c_y, k_1, k_2, k_3, p_1, p_2$
The most common way to calibrate the camera is to use a chessboard pattern. During practice, you will use OpenCV to calibrate the camera.

1. Make several photos of the chessboard pattern from different angles
2. Find corners of the chessboard pattern
3. Optimize reprojection error function (for both intrinsic and extrinsic parameters)
4. Get the coefficients

Camera calibration result

Figure: Comparison of distorted and undistorted images

Color spaces

Color spaces are the way to represent colors in the computer. Just a few of them:

1. RGB
2. HSV
3. HSL
4. YUV
5. YCbCr

YUV

YUV is a color space that separates the luminance and chrominance components of the color.

Figure: YUV components

Figure: YUV blure example

Canny edge detector




Contour detector

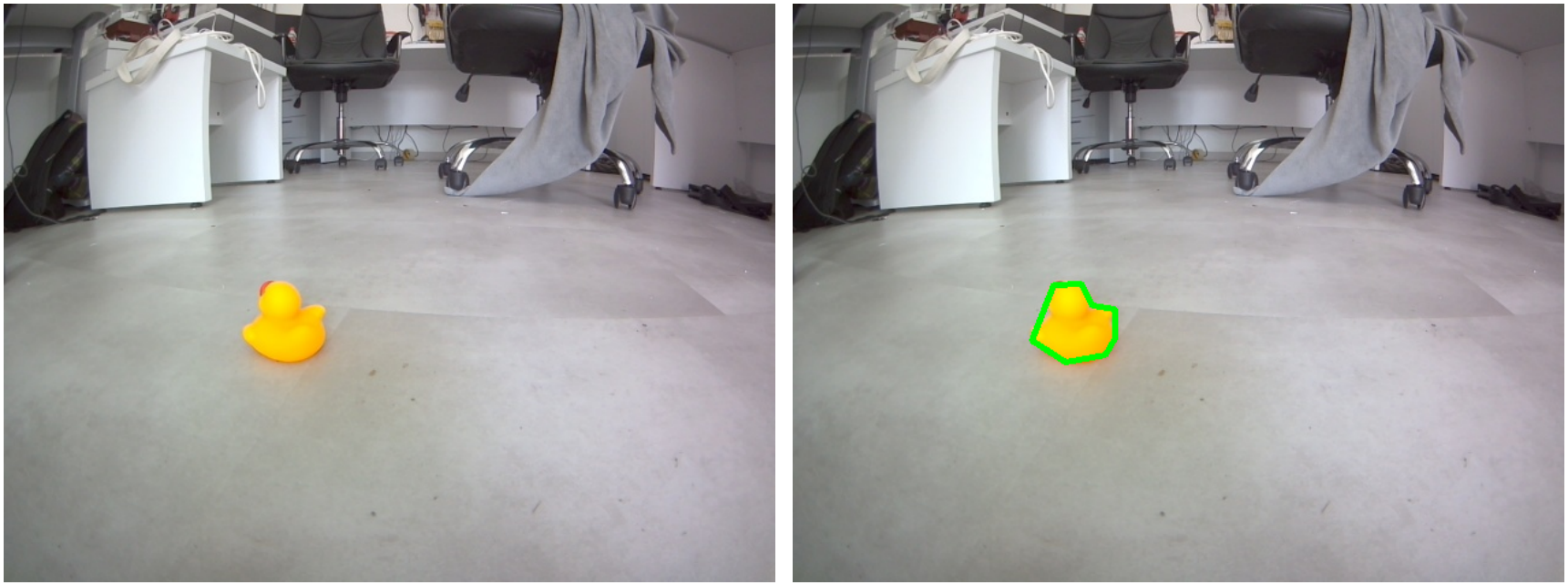

Figure: Contour detector example

OpenCV capabilities

What else can OpenCV do?

1. Camera calibration (as we discussed)
2. Color spaces transformations (as we discussed)
3. Camera extrinsic estimation (will be discussed in the next lecture)
4. Image processing (bluring, sharpening, etc.)
5. Morfological operations (dilation, erosion, etc.)
6. Video processing
7. Text recognition
8. Object tracking
9. Drawing shapes
10. Machine learning inference
11. A lot of else

Do not reinvent the wheel or "Simpsons already did it!"

If you want to make neural network algorithm for specific task, firstly check if there is a pre-trained model for this task. Possible sources of pre-trained models:

1. kaggle.com
2. Model zoo of the framework you use (torchvision, tensorflow-hub, etc.)
3. huggingface.co
4. universe.roboflow.com
5. YOLO

Simple inference example

```python
import cv2
import matplotlib.pyplot as plt
import os
from inference.models.utils import get_roboflow_model
API_KEY = os.getenv("ROBOFLOW_API_KEY")
duckietown_model = get_roboflow_model(
    model_id="{}/{}".format("duckietown-euvxp", 1), api_key=API_KEY
)
frame = cv2.imread("./duckiebot.jpg")
predictions = duckietown_model.infer(image=frame, confidence=0.5,
     iou_threshold=0.5)
pred = predictions[0].predictions[0]
bbox = cv2.rectangle(frame, (int(pred.x - pred.width / 2), int(pred.y -
     pred.height / 2)), (int(pred.x + pred.width / 2), int(pred.y + pred.
     height / 2)), (255,0,0), 5)
plt.imshow(bbox)
```

Figure: Duckiebot detection

Neural networks inference without training frameworks

1. ONNX (Open Neural Network Exchange)
2. OpenCV DNN module
3. TensorRT (NVIDIA)
4. OpenVINO (Intel)
5. CoreML (Apple for iOS)
6. ARM Compute Library (ARM)
7. other platform specific solutions

OpenCV DNN module

1. Supported frameworks: TensorFlow, Torch7, Caffe, Darknet, ONNX
2. A lot of supported layers
3. Both CPU and GPU support
4. Support of OpenCL and CUDA

```
1  import cv2
2  MODEL_PATH = "model.onnx"
3  IMAGE_PATH = "image.jpg"
4  SIZE=(256, 256)
5  opencv_net = cv2.dnn.readNetFromONNX(MODEL_PATH)
6  image = cv2.imread(IMAGE_PATH)
7  blob = cv2.dnn.blobFromImage(image, scalefactor=1.0 / 255, size=SIZE,
       crop=False, swapRB=True)
8  opencv_net.setInput(blob)
9  output = opencv_net.forward() # np.array
```