Video Processing

Lab 1: Harris Corner Detector + VP Intro

Administration - General Information

- Contact:
    - vptau2022@gmail.com
    - Course Forum
- Office Hours (by appointment): Sunday 11:00.
- 4-5 labs = 1 for every exercise + 1-2 for the project.
- 3 HW Exercises:
    - Harris Corner Detector + VP Basic Ops
    - Video Stabilization
    - Tracking
- 1 Project:
    - Input: Unstable video of a person walking.
    - Output: Stabilized Video with a different background + Tracking the person with a rectangle.

Administration - Homework Guidelines

```python
def create_grad_x_and_grad_y(input_image):
    """Calculate the gradients across the x and y-axes.

    Args:
        input_image: np.ndarray. Image array.
    Returns:
        tuple (Ix, Iy): The first is the gradient across the x-axis and the
        second is the gradient across the y-axis.

    Recipe:
    If the image is an RGB image, convert it to grayscale using OpenCV's
    cvtColor. Otherwise, the input image is already in grayscale.
    Then, create a one pixel shift (to the right) image and fill the first
    column with zeros.
    Ix will be the difference between the grayscale image and the shifted
    image.
    Iy will be obtained in a similar manner, this time you're requested to
    shift the image from top to bottom by 1 row. Fill the first row with zeros.
    Finally, in order to ignore edge pixels, remove the first column from Ix
    and the first row from Iy.
    Return (Ix, Iy).
    """
```

3 exercises + 1 project

Python3.9, conda, linux

Insert your code here

Report: PDF only. First line= your IDs.

Some items have changed from previous years. Do not copy.

Administration - How do we run your code?

```python
def create_grad_x_and_grad_y(input_image):
    """Calculate the gradients across the x and y-axes.

    Args:
        input_image: np.ndarray. Image array.
    Returns:
        tuple (Ix, Iy): The first is the gradient across the x-axis and the
        second is the gradient across the y-axis.

    Recipe:
    If the image is an RGB image, convert it to grayscale using OpenCV's
    cvtColor. Otherwise, the input image is already in grayscale.
    Then, create a one pixel shift (to the right) image and fill the first
    column with zeros.
    Ix will be the difference between the grayscale image and the shifted
    image.
    Iy will be obtained in a similar manner, this time you're requested to
    shift the image from top to bottom by 1 row. Fill the first row with zeros.
    Finally, in order to ignore edge pixels, remove the first column from Ix
    and the first row from Iy.
    Return (Ix, Iy).
    """
```

We install the conda virtual environment with the environment.yml which we supply.

We run the file or files containing:

```python
if __name__ == "__main__":
```

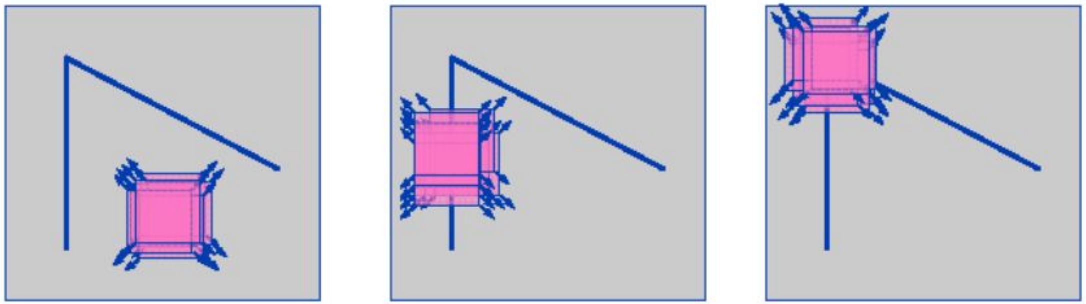
from the command line.

Administration - any other questions?

Harris Corner Detector

Why Corners?

Their very different than their neighbourhood - that’s why they’re distinctive.



“flat” region:
no change in
all directions

“edge”:
no change along
the edge direction

“corner”:
significant change
in all directions

Harris corner detector gives a mathematical approach for determining which case holds.

How Do We Compute Corners?

For each window in the image, we compute:

$$\sum [I(x+u,y+v) - I(x,y)]^2$$

$$\approx \sum [I(x,y) + uI_x + vI_y - I(x,y)]^2 \qquad \text{First order approx}$$

$$= \sum u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2$$

$$= \sum \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \qquad \text{Rewrite as matrix equation}$$

$$= \begin{bmatrix} u & v \end{bmatrix} \left(\sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

How Do We Compute Corners?

For each window in the image, we compute:

$$\sum [I(x+u, y+v) - I(x,y)]^2$$

$$\approx \sum [I(x,y) + uI_x + vI_y - I(x,y)]^2 \quad \text{First order approx}$$

$$= \sum u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2$$

$$= \sum \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \quad \text{Rewrite as matrix equation}$$

$$= \begin{bmatrix} u & v \end{bmatrix} \left(\sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

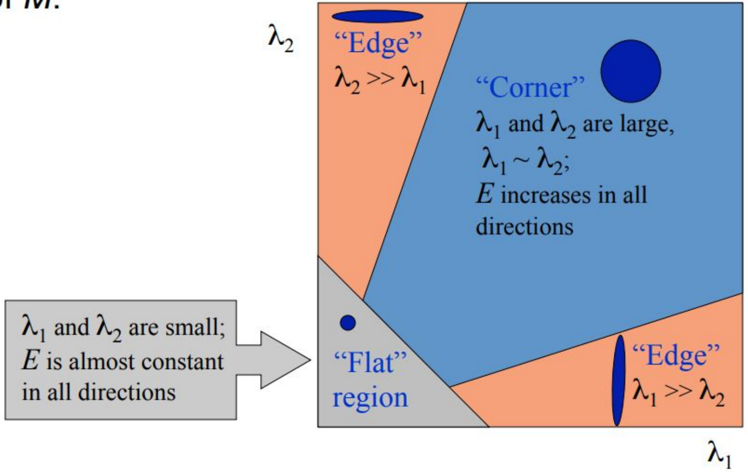
M

How Do We Compute Corners?

$$E(u,v) = \sum_{(x,y)\in W}[I(x+u,y+v) - I(x,y)]^2 \approx [u \;\; v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

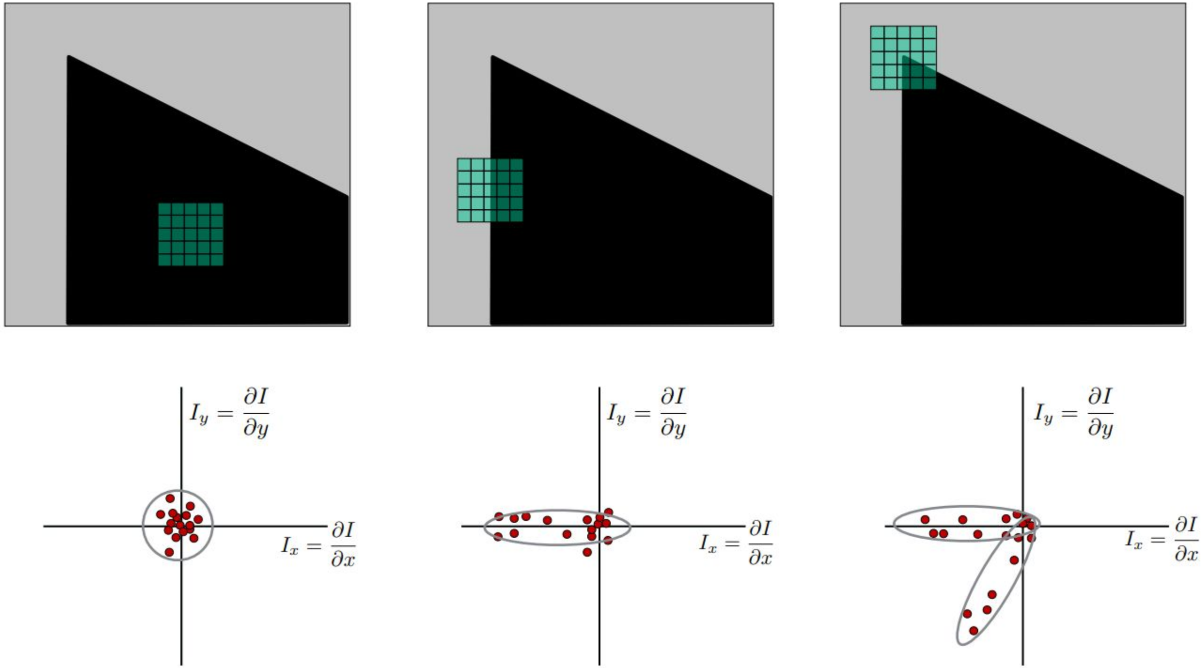
Interpreting the eigenvalues

Classification of image points using eigenvalues of *M*:



$$M = \sum_{x,y} \begin{bmatrix} I_x^2 & I_x * I_y \\ I_x * I_y & I_y^2 \end{bmatrix} = A^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} A$$

Intuition



$I_y = \frac{\partial I}{\partial y}$
$I_x = \frac{\partial I}{\partial x}$
$I_y = \frac{\partial I}{\partial y}$
$I_x = \frac{\partial I}{\partial x}$
$I_y = \frac{\partial I}{\partial y}$
$I_x = \frac{\partial I}{\partial x}$

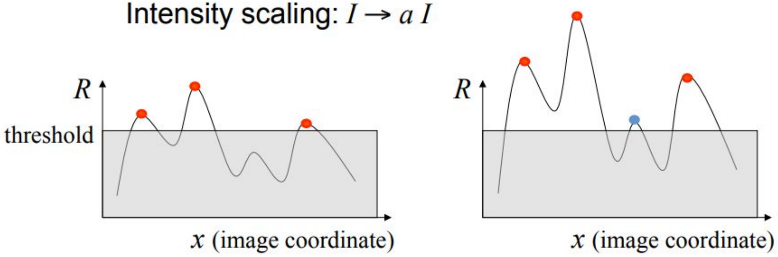
Invariance

Affine intensity change

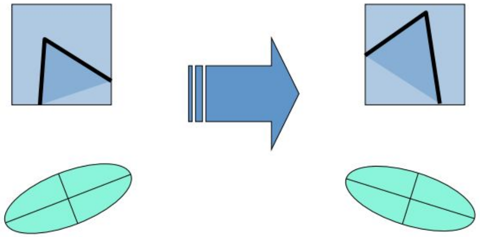
$I \to a\,I + b$

Only derivatives => invariance to intensity shift $I \to I + b$

Intensity scaling: $I \to a\,I$

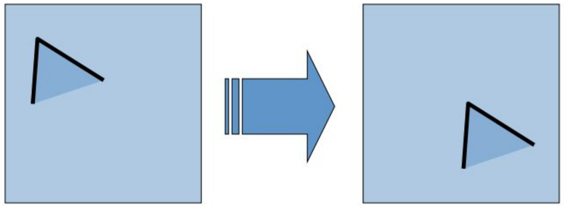


Harris: image rotation



Second moment ellipse rotates but its shape
(i.e. eigenvalues) remains the same

Harris: image translation



Harris Corner Detector - Algorithm

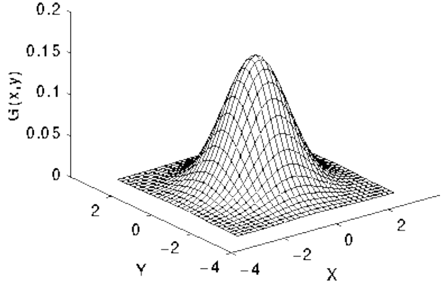
Input: Image I.

Output: Image with the same size indicating where corners are.

Alg:

- $[I_x, I_y] = gradient(I)$
  - I_x^2 - pixel-wise multiplication of I_x
- Define filter g – usually box filter (5X5 ones), or gaussian.
- $S_{xx} = conv(I_x^2, \text{g})$, $S_{yy} = conv(I_y^2, \text{g})$, $S_{xy} = conv(I_x \cdot I_y, \text{g})$

Gaussian filter g



$$Det(M) = \lambda_- \cdot \lambda_+$$
$$Trace(M) = \lambda_- + \lambda_+$$
$$0.04 < k < 0.06$$

Harris Corner Detector - Algorithm

Input: Image I.

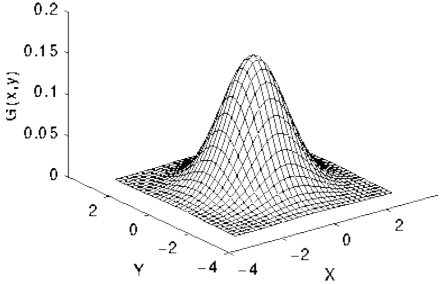
Output: Image with the same size indicating where corners are.

Alg:

- $[I_x, I_y] = gradient(I)$
  - I_x^2 - pixel-wise multiplication of I_x
- Define filter g – usually box filter (5X5 ones), or gaussian.
- $S_{xx} = conv(I_x^2, g)$, $S_{yy} = conv(I_y^2, g)$, $S_{xy} = conv(I_x \cdot I_y, g)$

Window Sum Trick

Gaussian filter g



$$Det(M) = \lambda_- \cdot \lambda_+$$
$$Trace(M) = \lambda_- + \lambda_+$$
$$0.04 < k < 0.06$$

Harris Corner Detector - Algorithm

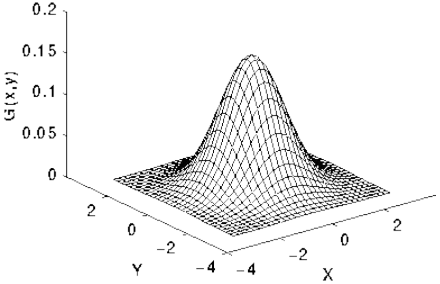
Input: Image I.

Output: Image with the same size indicating where corners are.

Alg:

- $[I_x, I_y] = gradient(I)$
  - I_x^2 - pixel-wise multiplication of I_x
- Define filter g – usually box filter (5X5 ones), or gaussian.
- $S_{xx} = conv(I_x^2, \text{g})$, $S_{yy} = conv(I_y^2, \text{g})$, $S_{xy} = conv(I_x \cdot I_y, \text{g})$
- R = $\frac{\lambda_- \cdot \lambda_+}{\lambda_- + \lambda_+} \approx \det(M) - k \cdot [trace(M)]^2 = S_{xx} \cdot S_{yy} - S_{xy}^2 - k\left(S_{xx} + S_{yy}\right)^2$
- $R(R < \theta) = 0$, $where\ \theta$ is a user defined threshold, R – Response image
- Optional: Non-maximum suppression of R in each tile

Gaussian filter g



$$Det(M) = \lambda_- \cdot \lambda_+$$
$$Trace(M) = \lambda_- + \lambda_+$$
$$0.04 < k < 0.06$$

Is this result good for us?