

CSE185

Introduction to Computer Vision

Lab 02: For Loop Operation

Instructor: Prof. Ming-Hsuan Yang

TA: Chun-Han Yao & Yi-Wen Chen

If statement

- If statement

```
if EXPRESSION  
    ...  
end
```

- If-else statement

```
if EXPRESSION  
    ...  
else  
    ...  
end
```

```
if EXPRESSION  
    ...  
elseif EXPRESSION  
    ...  
else  
    ...  
end
```

Loop

- For loop

```
for i = 1:10  
    ...  
end
```

- While loop

```
while EXPRESSION  
    ...  
end
```

Translation

- Shift image by 50 pixels:

$$\begin{aligned} I_2(y, x) &= 0 && \text{if } x \leq 50 \\ I_2(y, x) &= I_1(y, x - 50) && \text{if } x > 50 \end{aligned}$$



Translation

- Shift image by 50 pixels
- Use For loop

```
I2 = zeros(300, 400 + 50, 3, 'uint8');  
  
for y1 = 1 : 300  
    for x1 = 1 : 400  
  
        y2 = ???  
        x2 = ???  
  
        I2(y2, x2, :) = I1(y1, x1, :);  
  
    end  
end
```

- Use submatrix indexing

```
I2 = zeros(300, 400 + 50, 3, 'uint8');  
I2(1:300, 51:500+50, :) = I1;
```

Rotation

- Rotate image 45 degree: (do NOT use `imrotate()`)

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$



Rotation

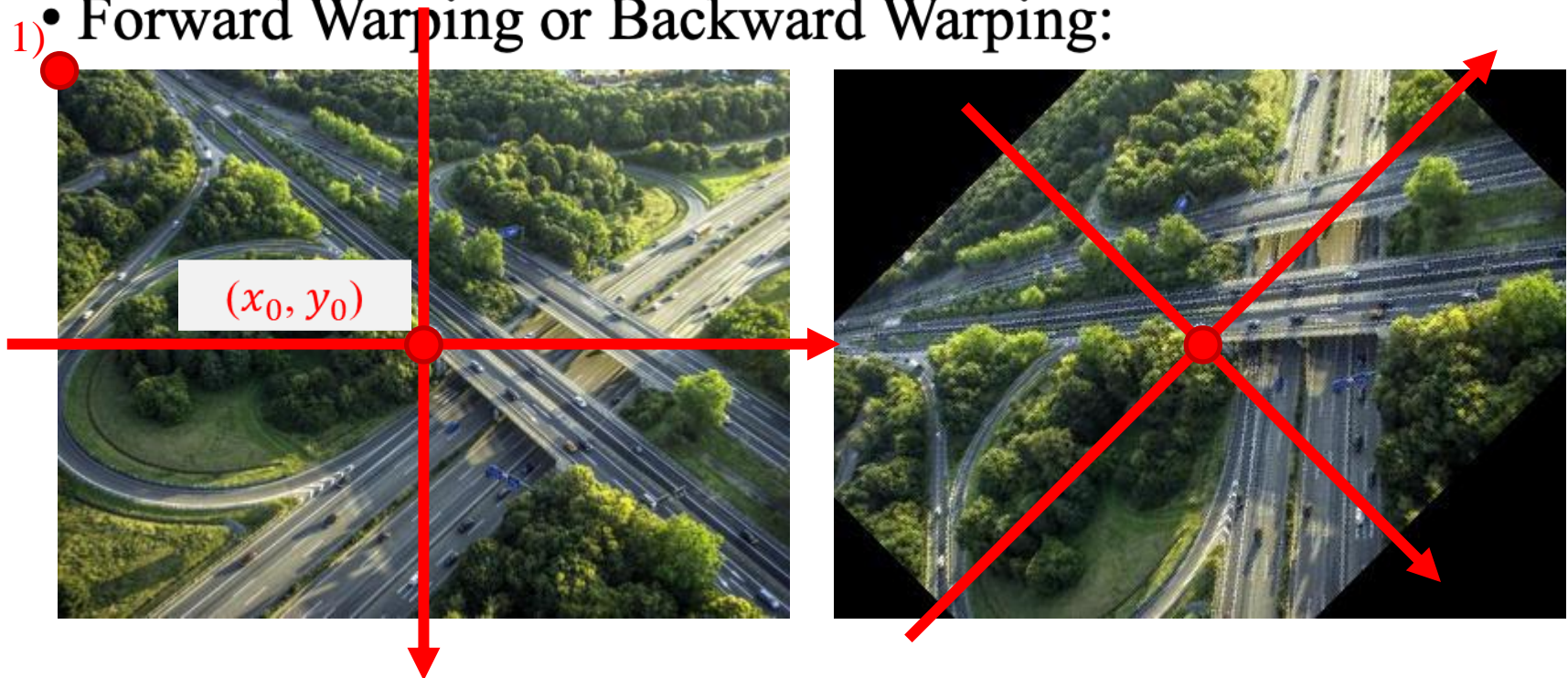
- Rotate image 45 degree:

- shift origin (x_0, y_0) to the center of the input image

$$x_2 = \cos(\theta) \cdot (x_1 - x_0) + \sin(\theta) \cdot (y_1 - y_0) + x_0$$

$$y_2 = -\sin(\theta) \cdot (x_1 - x_0) + \cos(\theta) \cdot (y_1 - y_0) + y_0$$

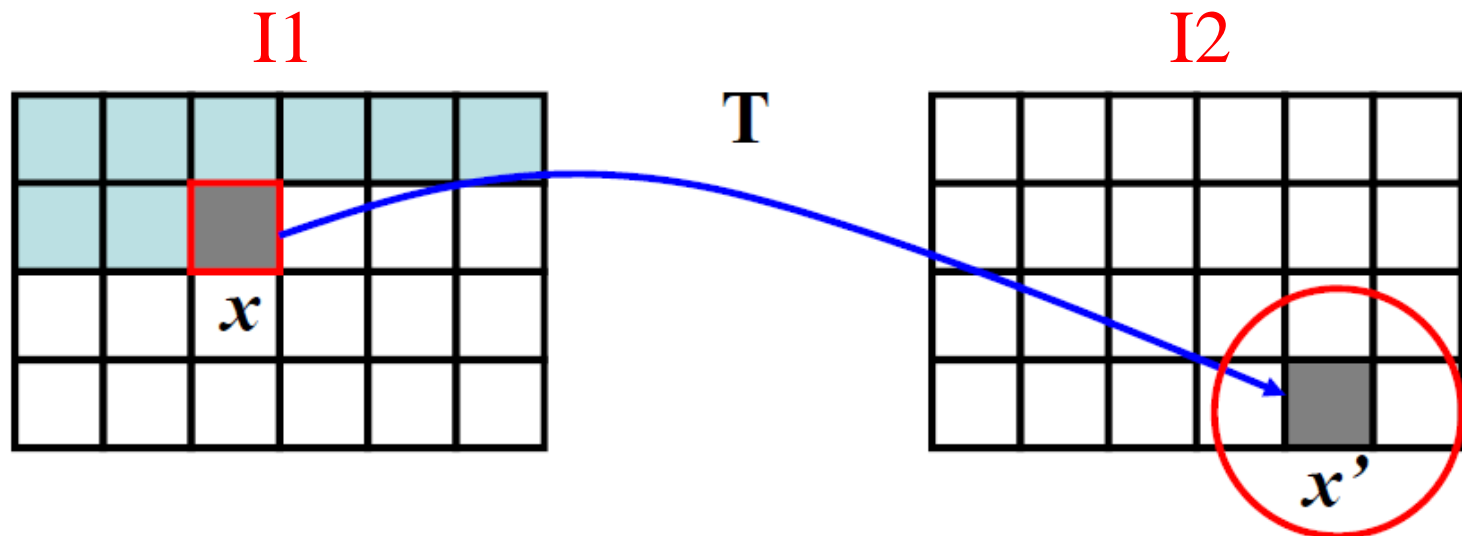
- Forward Warping or Backward Warping:



Forward Warping

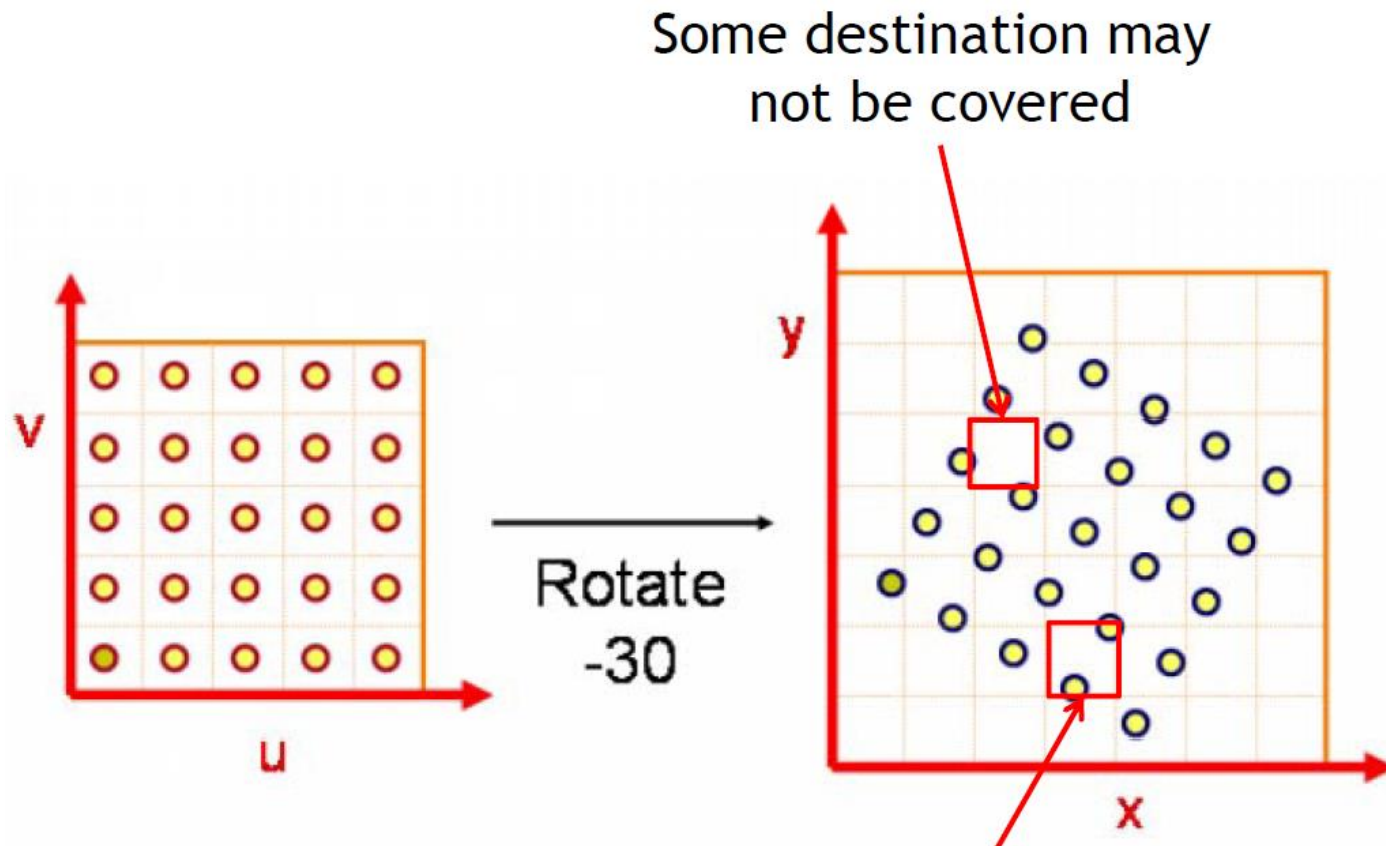
- Suppose I_1 is input image, I_2 is warped image
- Pseudocode:

```
for each pixel  $(y_1, x_1)$  in  $I_1$ :  
     $(y_2, x_2) = \text{Rotate}(y_1, x_1)$   
    if  $(y_2, x_2)$  is inside  $I_2$ :  
         $I_2(y_2, x_2) = I_1(y_1, x_1)$   
    end  
end
```



Forward Warping

- Forward warping will produce “holes”:



Forward Warping

- Forward warping will produce “holes”:

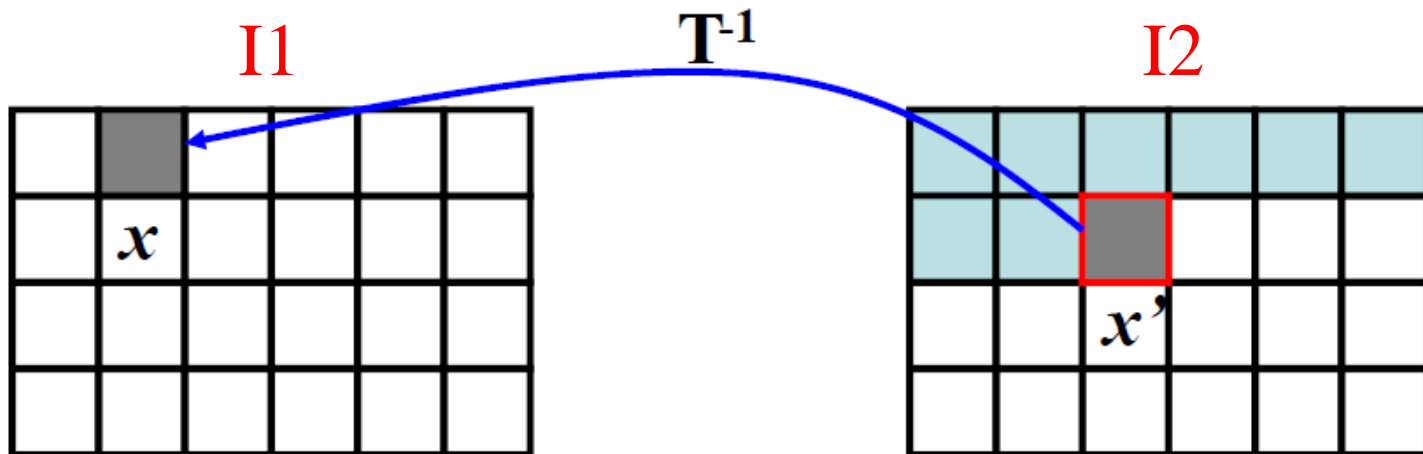


Backward/Inverse Warping

- Suppose I_1 is input image, I_2 is warped image:

```
for each pixel  $(y_2, x_2)$  in  $I_2$ :  
   $(y_1, x_1) = \text{Rotate}^{-1}(y_2, x_2)$   
  if  $(y_1, x_1)$  is inside  $I_1$ :  
     $I_2(y_2, x_2) = I_1(y_1, x_1)$   
  end  
end
```

$\text{Rotate}^{-1} = \text{Inverse warping}$



Backward/Inverse Warping



Backward/Inverse Warping: Hints

- The inverse of a rotation matrix is still a rotation matrix:

$$\begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}^{-1} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

- Use `cosd()` and `sind()` if your angle is in degree, use `cos()` and `sin()` if your angle is in radian.
- Use nearest neighbor sampling:

```
(y1, x1) = Rotate-1(y2, x2)  
y1 = round(y1);  
x1 = round(x1);  
If( 1 <= y1 && y1 <= H && 1 <= x1 && x1 <= W )  
...
```

Implemented by sin and cos

- Your rotated image will look similar to:

```
imrotate(I1, 45, 'nearest', 'crop')
```

Image Processing in MATLAB

- Horizontally flip image: do NOT use `flip()`



- Use For loop, or submatrix indexing:

```
>> vec = [10, 20, 30, 40, 50];
```

```
>> vec(5:-1:1)
```

Specify the step size to -1

```
ans =
```

```
    50    40    30    20    10
```


Scaling

- Down-sample image by a factor of 2: do NOT use `imresize()`



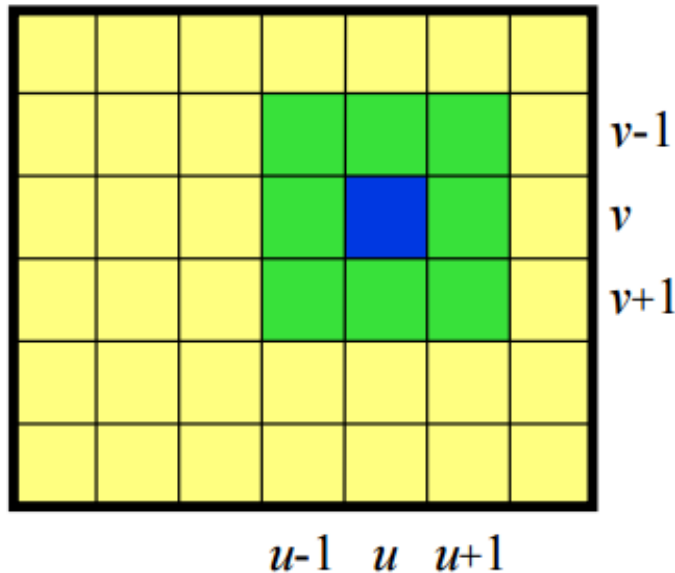
- Your result should look like

`imresize(I1, 0.5, 'nearest')`

- You should use For loop or submatrix indexing

Spatial Filter

- A **spatial filter** is an image operation where each pixel value $I(u, v)$ is changed by a function of the intensities of pixels in a neighborhood of (u, v) .



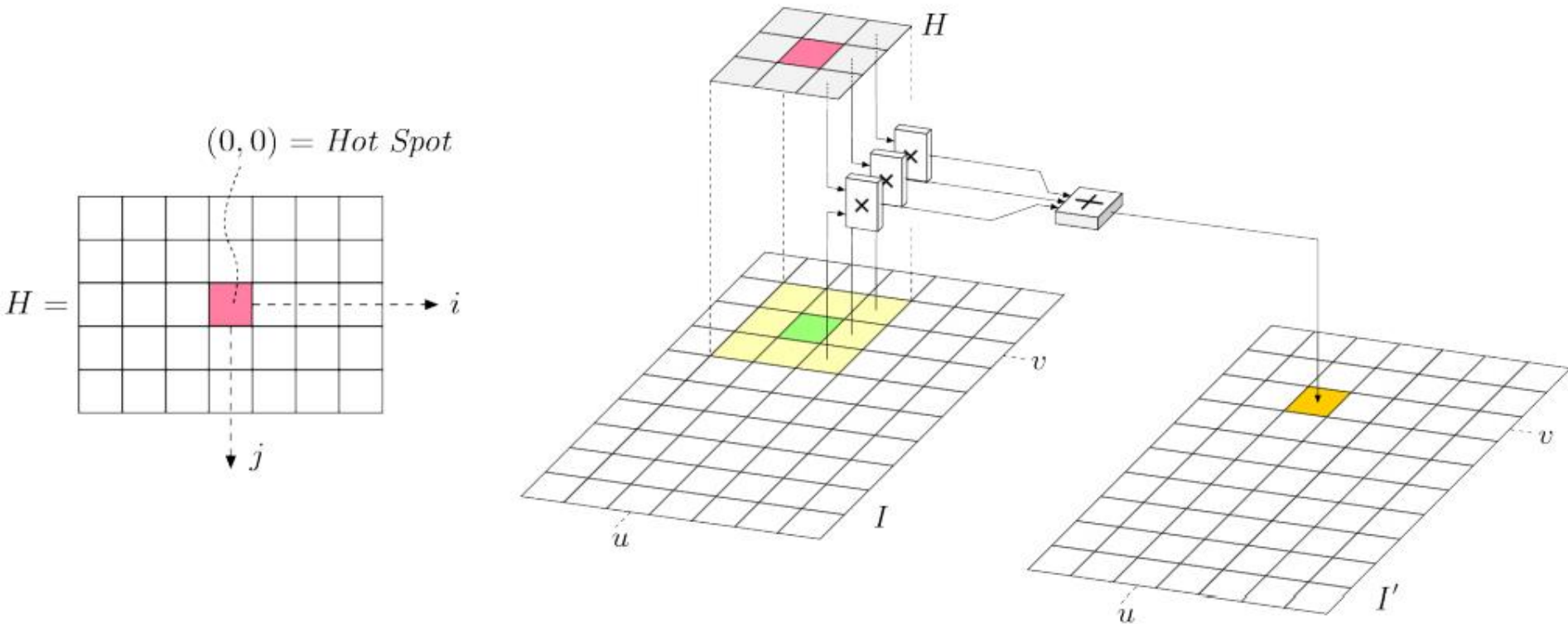
- Mean filter:

$$I'(u, v) = \frac{1}{9} \sum_{i=-1}^1 \sum_{j=-1}^1 I(u + i, v + j)$$

Linear Filter

- H is the filter kernel/matrix:

$$I'(u, v) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(u+i, v+j) \cdot H(i, j)$$



Linear Filter

- Assume H is a 3×3 mean filter: $H = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

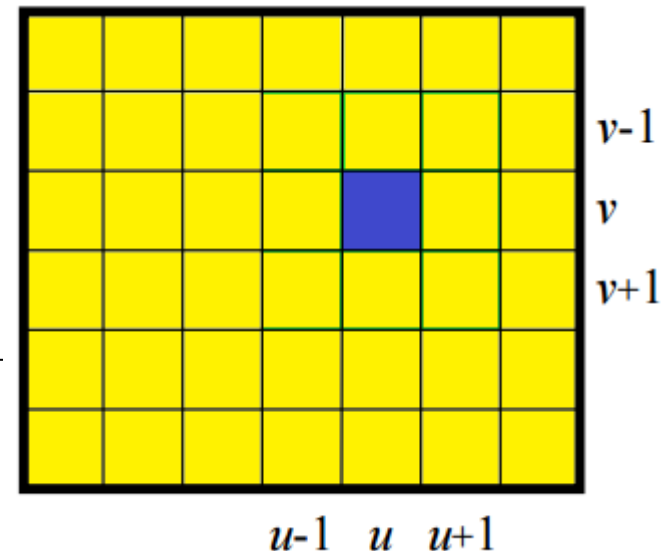
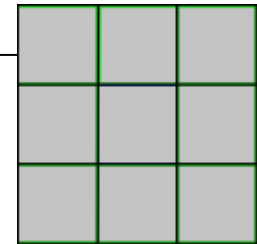
```
I1 = im2double(imread('lena.jpg'));  
I2 = zeros(size(I1));
```

```
for u = 1 : size(I1, 2)  
    for v = 1 : size(I1, 1)
```

```
        value = ???;  
        I2(v, u) = value;
```

```
    end
```

```
end
```



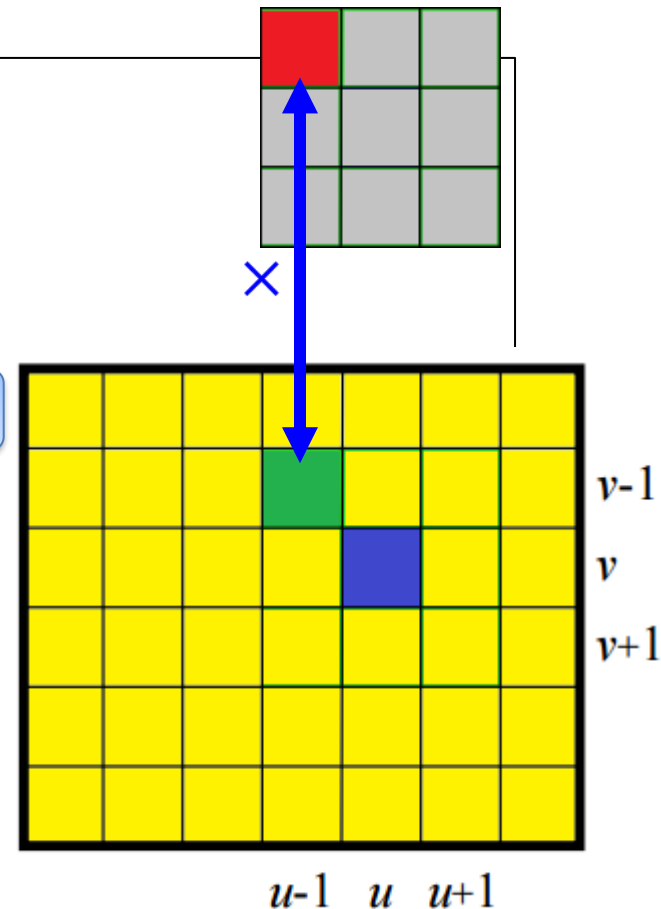
Linear Filter

$$I'(u, v) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(u+i, v+j) \cdot H(i, j)$$

```
I1 = im2double(imread('lena.jpg'));  
I2 = zeros(size(I1));
```

```
for u = 1 : size(I1, 2)  
    for v = 1 : size(I1, 1)  
  
        value = 0;  
        for i = -1:1  
            for j = -1:1  
                value = value + ???  
            end  
        end  
        I2(v, u) = value;  
    end  
end
```

Select neighborhood



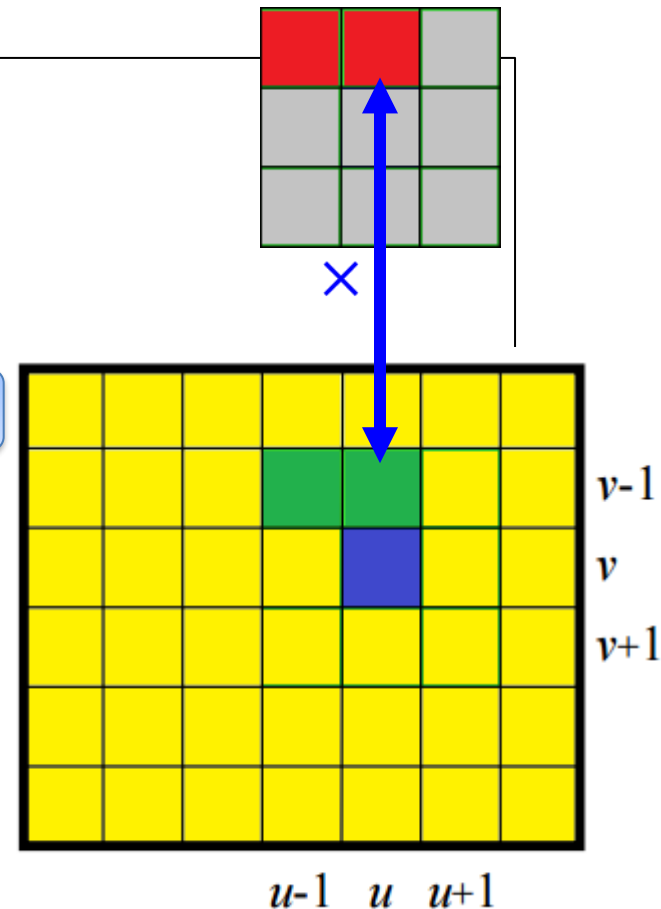
Linear Filter

$$I'(u, v) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(u+i, v+j) \cdot H(i, j)$$

```
I1 = im2double(imread('lena.jpg'));  
I2 = zeros(size(I1));
```

```
for u = 1 : size(I1, 2)  
    for v = 1 : size(I1, 1)  
  
        value = 0;  
        for i = -1:1  
            for j = -1:1  
                value = value + ???  
            end  
        end  
        I2(v, u) = value;  
    end  
end
```

Select neighborhood



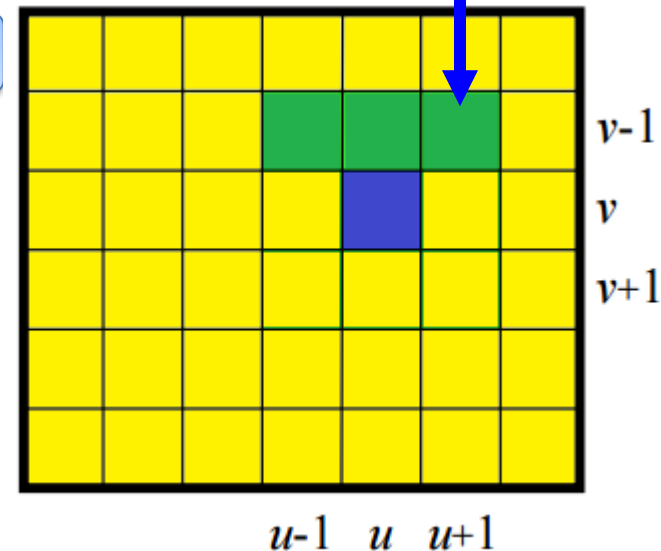
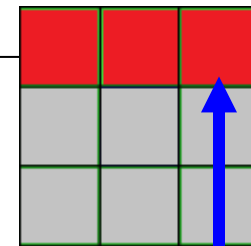
Linear Filter

$$I'(u, v) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(u+i, v+j) \cdot H(i, j)$$

```
I1 = im2double(imread('lena.jpg'));  
I2 = zeros(size(I1));
```

```
for u = 1 : size(I1, 2)  
    for v = 1 : size(I1, 1)  
  
        value = 0;  
        for i = -1:1  
            for j = -1:1  
                value = value + ???  
            end  
        end  
        I2(v, u) = value;  
    end  
end
```

Select neighborhood



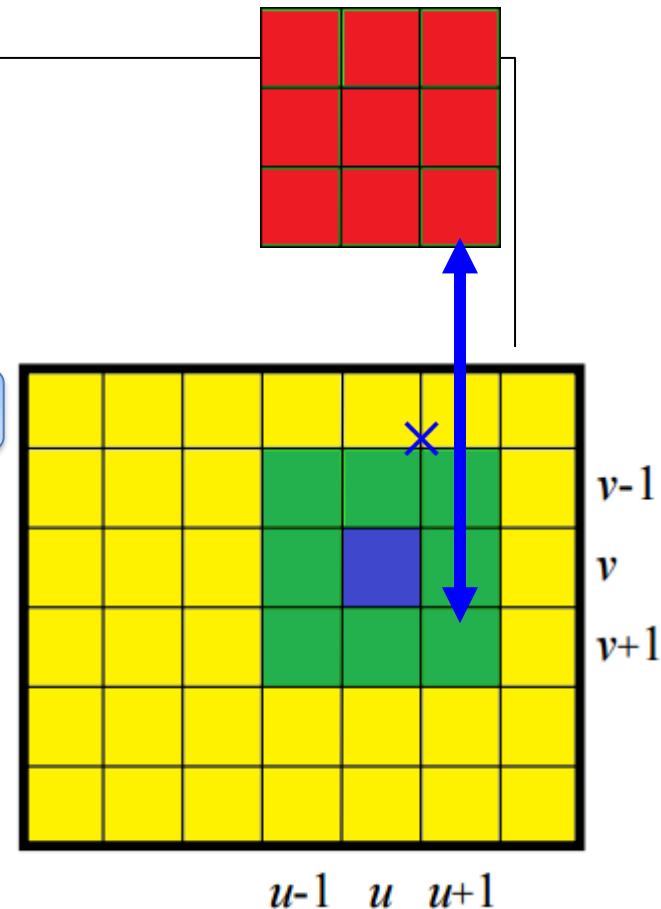
Linear Filter

$$I'(u, v) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(u+i, v+j) \cdot H(i, j)$$

```
I1 = im2double(imread('lena.jpg'));  
I2 = zeros(size(I1));
```

```
for u = 1 : size(I1, 2)  
    for v = 1 : size(I1, 1)  
  
        value = 0;  
        for i = -1:1  
            for j = -1:1  
                value = value + ???  
            end  
        end  
        I2(v, u) = value;  
    end  
end
```

Select neighborhood

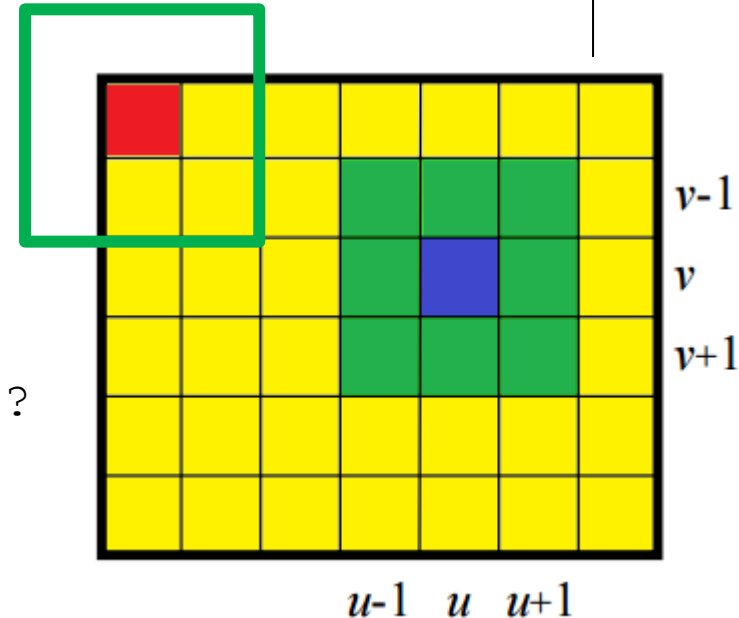


Linear Filter

- Be careful around boundaries
 - ignore the pixels on boundaries
 - padding/extend boundaries [Optional]

```
I1 = im2double(imread('lena.jpg'));  
I2 = zeros(size(I1));
```

```
for u = 1 : size(I1, 2)  
    for v = 1 : size(I1, 1)  
  
        value = 0;  
        for i = -1:1  
            for j = -1:1  
                value = value + ???  
            end  
        end  
        I2(v, u) = value;  
  
    end  
end
```



Be careful about the index

Linear Filter

- Note: For Loop in MATLAB is very slow!
- Tips: use `tic` and `toc` to measure the elapsed time

```
%% Gaussian filter
hsize = 11;
sigma = 4;
tic
I = gaussian_filter_slow(img, hsize, sigma);
toc
tic
I = gaussian_filter_fast(img, hsize, sigma);
toc
```

```
Elapsed time is 3.522677 seconds.
Elapsed time is 1.608707 seconds.
```

```
fx >>
```

Linear Filter

- Extract patch/sub-matrix, and do matrix/vector operation.
- Again, be careful about the index around image boundaries

```
I1 = im2double(imread('lena.jpg'));  
I2 = zeros(size(I1));
```

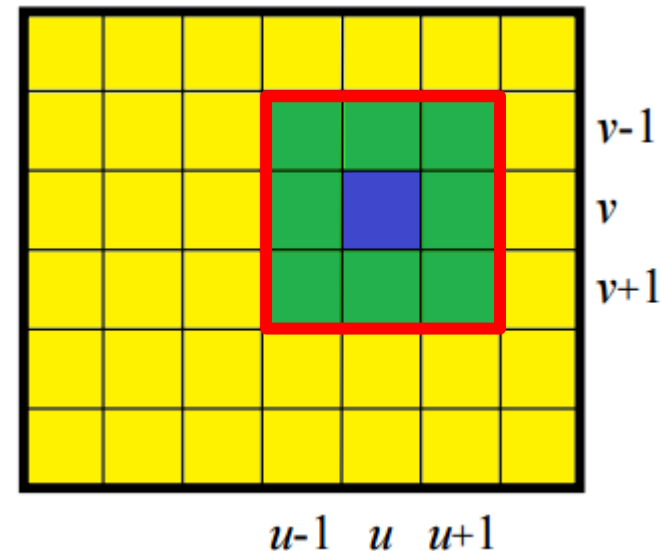
```
for u = 1 : size(I1, 2)  
    for v = 1 : size(I1, 1)
```

```
        x1 = ???; x2 = ???;  
        y1 = ???; y2 = ???;  
        patch = I1(y1:y2, x1:x2);  
        % convert matrix to vector  
        % matrix/vector operations  
        value = ???;  
        I2(v, u) = value;
```

```
    end
```

```
end
```

Check index range



Linear Filter

- Skip boundary pixels
 - How many pixels to shift?

```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1+shift_u : size(I1, 2)-shift_u
    for v = 1+ shift_v : size(I1, 1)- shift_v

        x1 = ???; x2 = ???;
        y1 = ???; y2 = ???;
        patch = I1(y1:y2, x1:x2);
        % convert matrix to vector
        % matrix/vector operations
        value = ???;
        I2(v, u) = value;

    end
end
```

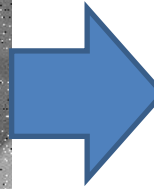
Shift depends on the filter size

Median Filter

- Extract patch with the same size as the filter
- Calculate the median value of the patch (Use `median()`)
- Fill in the median value to the output pixel



Noisy input image



Median filter output

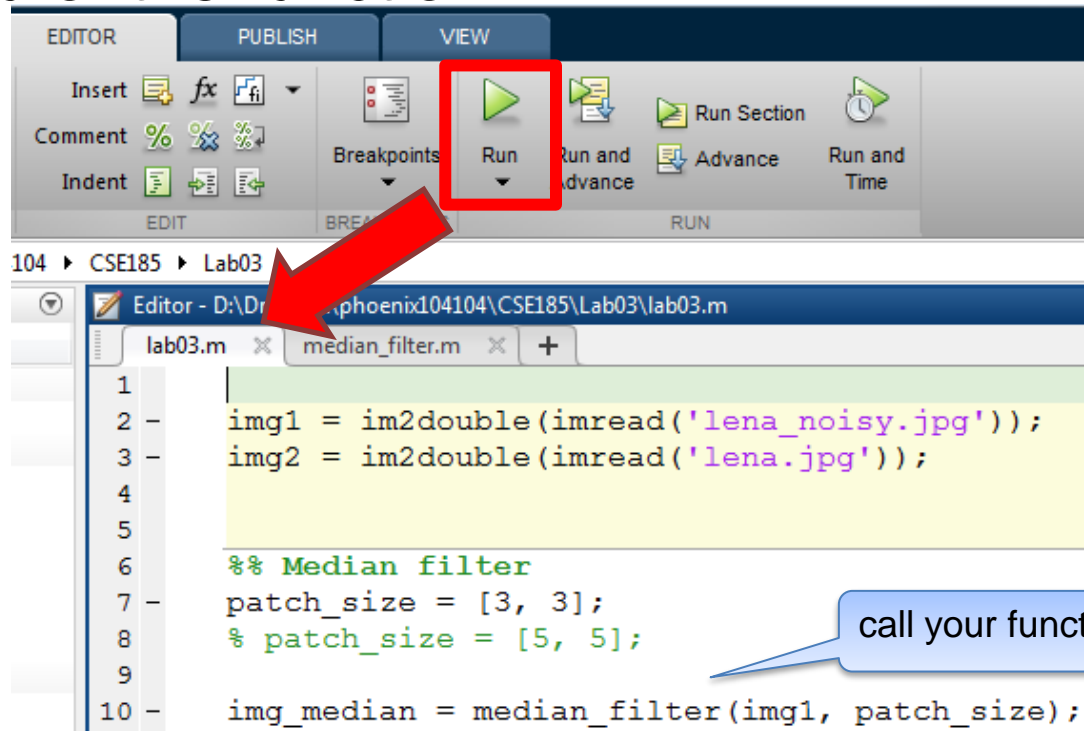
MATLAB Function

- Function template:

```
function output = median_filter(input, patch_size)
    output = input;
end
```

Assign output value, it will be automatically returned at the end of the function

- lab02.m is your main function, run this script instead of the function



call your function

Median Filter

- median_filter.m

```
function output = median_filter(img, patch_size)
    % YOUR CODE HERE
end
```

- In lab02.m:

```
img = im2double(imread('lena_noisy.jpg'));

%% Median filter
patch_size = [3, 3];
% patch_size = [5, 5];

img_median = median_filter(img, patch_size);
imwrite(img_median, 'median.jpg');
```

- Compare your result with built-in function:

```
I = medfilt2(img, patch_size);
```

Median Filter: Hints

- MATLAB function `median()`:

```
>> A = [1, 2, 3;  
        4, 5, 6;  
        7, 8, 9];  
>> median(A)  
ans =  
     4     5     6
```

- But we need a single value
 - convert matrix to vector first
 - or apply function twice
 - Question: are the results the same?

Lab Assignment 02: image warping & filter

1. Rotate 01.jpg by 45 degrees using forward warping, and save as **rotate_0.jpg**
 2. Rotate 01.jpg by 45 degrees using backward warping, and save as **rotate_1.jpg**
 3. Implement median_filter.m for lena_noisy.jpg, use patch size = 3 and save the image as **median_0.jpg**
 4. Use patch size = 5, and save the image as **median_1.jpg**
- * Save your main code as **lab02.m**, and function for Q3, 4 as **median_filter.m**
- * Upload your output images, lab02.m, median_filter.m (one .zip file) to CatCourses
- * Do **NOT** use any built-in function (e.g., `imrotate`)