

Machine Vision Pipeline & Real Applications

Machine Vision

Dr. Mostapha Kalami Heris

 m.k.heris@shu.ac.uk

School of Engineering and Built Environment

**Sheffield
Hallam
University**

Learning Outcomes

- Explain the main stages of a machine vision pipeline.
- Connect preprocessing, segmentation, and morphology to real applications.
- Analyze how each stage influences downstream measurement and decisions.
- Interpret several machine vision examples (illumination correction, circular-object detection, OCR, deblurring).
- Evaluate strengths and limitations of different workflows.

Road Map for Today

- 1** The Machine Vision Pipeline: Scene → Decision
- 2** Why Pipelines Matter in Real Applications
- 3** Application Walkthroughs
- 4** Strengths, Weaknesses, and Practical Insights
- 5** Summary & Reflection

Motivation

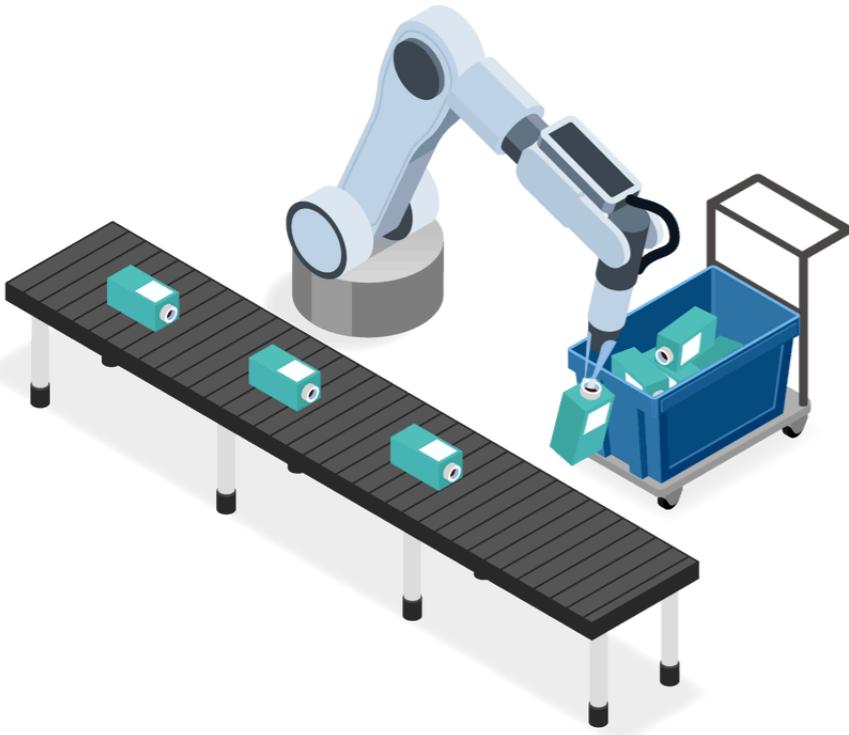
What Happens After You Learn the Basics?

We have covered and discussed filters, segmentation, and morphology.

But how do these tools come together in a real machine vision system?

This lecture connects the individual techniques to complete application workflows.

Machines Must Turn Images into Decisions



Every system must decide something:

- ✓ Is this part defective?
- ✓ Where is the object located?
- ✓ What does this text say?
- ✓ How many objects are present?

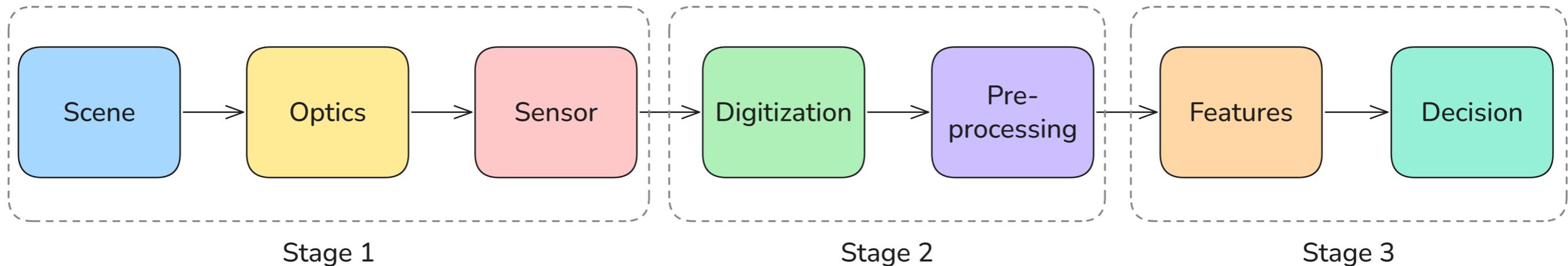
Why Think in Pipelines?

- Real images have **illumination issues, noise, blur, and shadows**.
- No single operation works alone; success depends on **carefully combining steps**.
- Pipelines organize processing from **raw image → meaningful decision**.

You have already learned the tools. Now we map them into a complete workflow.

The Machine Vision Pipeline

From Scene to Decision



Some typical operations:

1 Scene & Image Acquisition

2 Preprocessing

3 Segmentation

4 Cleaning & Morphology

5 Measurement & Feature Extraction

6 Decision or Classification

Stage 1: Scene → Sensor

- Illumination
- Optics (lens, aperture, focus)
- Sensor quality
- Noise sources

Poor lighting or blur here cannot be fully fixed later.



Stage 2: Preprocessing

- Denoising (linear & nonlinear filters)
- Contrast adjustment
- Background correction
- Color-to-grayscale conversion

Goal: prepare the image so segmentation has the best chance to succeed.

Stage 3: Segmentation

- Global thresholding
- Adaptive thresholding
- Otsu's method
- Multilevel thresholding



Segmentation converts pixels → meaningful regions. Everything after this stage depends on its quality.

Stage 4: Cleaning & Morphology

- Opening: remove small bright noise
- Closing: fill small dark holes
- Erosion: separate objects
- Dilation: connect parts
- Top-hat: correct illumination

Morphology prepares segmented regions for accurate measurement.

Stage 5: Measurement & Decision

Common measurements (`regionprops`):

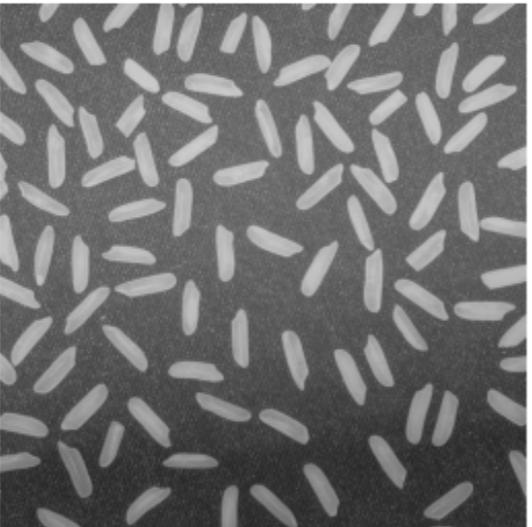
- Area, perimeter, centroid
- Orientation, roundness
- Bounding box
- Mean intensity

The final decision is only as good as the previous steps.

1 Correcting Nonuniform Illumination

Real Example: Uneven Lighting

Real images often suffer from uneven illumination. A simple global threshold can fail because the background is brighter on one side.



Why Illumination Correction Matters

- Thresholding assumes a **uniform background**.
- Shadows, gradients, or reflections break this assumption.
- Background correction produces more **reliable segmentation**.
- Essential for inspection, microscopy, food quality control, etc.

Strategy: Estimate the Background

Use a **large structuring element** to approximate the background.

Then **subtract** it from the original image.

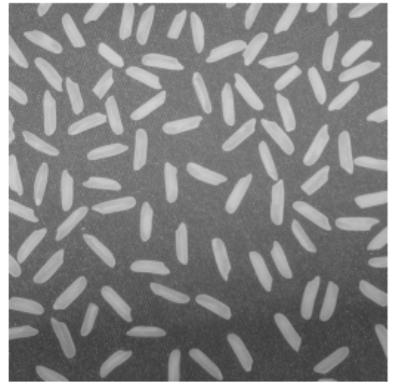
For bright objects:

$$I_{\text{bg}} = I \circ S, \quad I_{\text{corr}} = I - I_{\text{bg}}$$

Where S is a large disk SE.

Visualizing Background Modeling

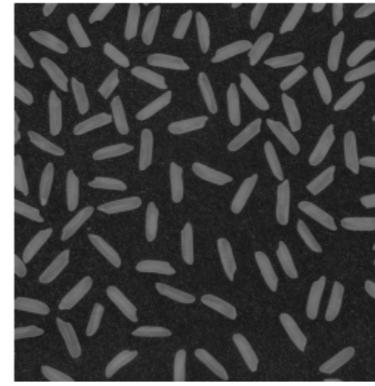
Original



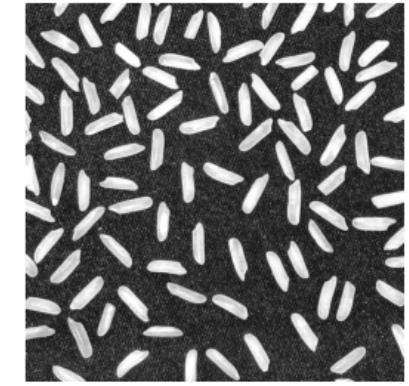
Estimated Background



Corrected Image



Improved Image



A large SE follows illumination changes but ignores small objects (e.g., rice grains).

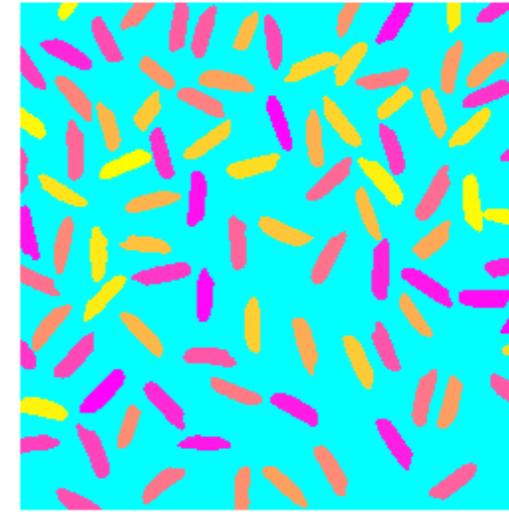
MATLAB Demo: Background Correction

```
I = imread('rice.png'); % Example with uneven illumination  
  
% 1. Create a large structuring element  
se_bg = strel('disk', 15);  
  
% 2. Estimate background by opening  
I_bg = imopen(I, se_bg);  
  
% 3. Subtract to correct illumination  
I_corr = imsubtract(I, I_bg);  
  
% 4. Segment the corrected image  
BW = imbinarize(I_corr);
```

MATLAB

 SE must be larger than the objects so it tracks illumination, not object shape.

After Correction: Reliable Segmentation



Clean segmentation enables accurate object counting and measurement.

Extracting Objects and Measurements

Using `regionprops`, we can measure:

- Area
- Centroid
- Perimeter
- Bounding box

MATLAB

```
[L, num] = bwlabel(BW);  
stats = regionprops(L, 'Area', 'Centroid', 'BoundingBox');
```

This transforms *raw pixels* to *meaningful data* for inspection systems.

2 Detecting Circular Objects

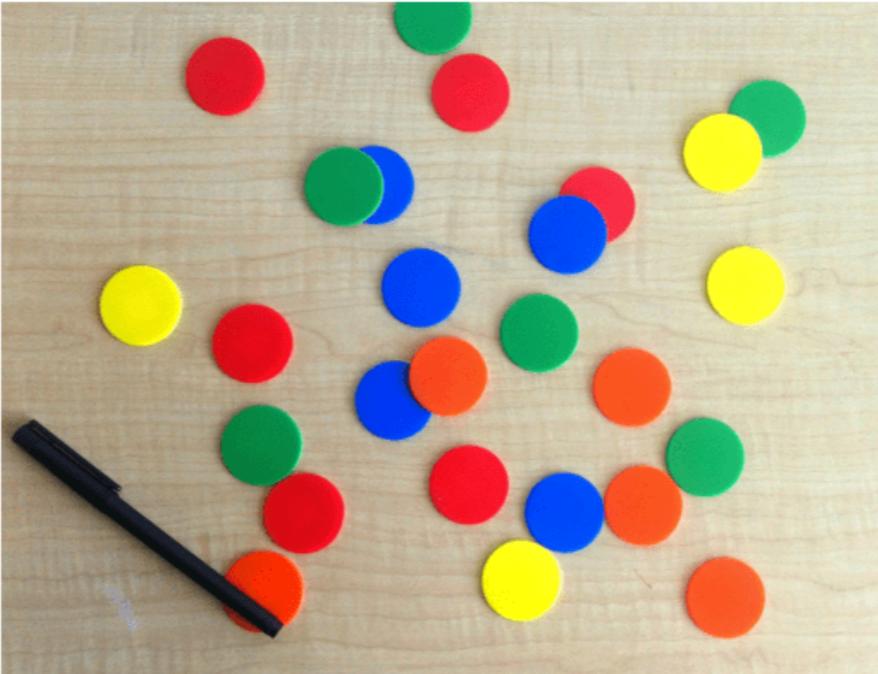
Why Circular Objects?

Circular objects appear in many domains:

- Industrial parts (washers, bearings)
- Pharmaceutical tablets
- Fruits and agricultural produce
- Coins and tokens

Each application needs **detection + measurement** for automation and quality control.

Example Image



Mixed-size circular objects, ideal for demonstrating the `imfindcircles` workflow.

Pipeline for Circular Object Detection

- 1 Read and prepare the image
- 2 Convert to grayscale (if needed)
- 3 Optionally enhance contrast
- 4 Use Hough-based circle detection with `imfindcircles`
- 5 Visualize results with `viscircles`
- 6 Use centers and radii for measurement and decisions

Method: Hough Circle Transform

The Hough Transform searches for **circular patterns** in an edge map.

Key parameters in `imfindcircles` :

- **Radius range** (e.g. `[20 60]`)
- **Sensitivity** (detection aggressiveness)
- **ObjectPolarity** (bright or dark circles)

How Hough Circle Detection Works

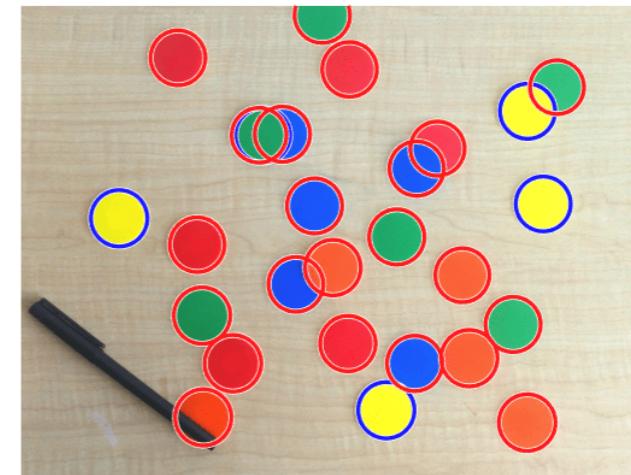
- 1 Edge detection identifies potential circle boundaries.
- 2 For each edge point, votes are cast in parameter space for possible circle centers and radii.
- 3 Peaks in the accumulator space indicate likely circle locations and sizes.

Visualizing Hough Detection

Input Image



Detected Circles



Hough-based detection works well when edges are clear and radii fall within a defined range.

MATLAB Demo: Hough-Based Detection

```
I = imread('coloredChips.png'); % Example image  
G = rgb2gray(I); % Convert to grayscale  
  
% Detect circles in a given radius range  
[centers, radii] = imfindcircles(G, [20 60], ...  
    'Sensitivity', 0.92, ...  
    'ObjectPolarity', 'bright');  
  
imshow(I);  
viscircles(centers, radii, 'EdgeColor', 'g');
```

MATLAB

💡 Try adjusting Sensitivity and radius range to see which circles appear or disappear.

Measuring the Detected Circles

Once we have **centers** and **radii**, we can:

- Count the number of circular objects
- Estimate diameters or areas
- Filter circles by size range
- Use locations for pick-and-place or alignment tasks

MATLAB

% Count objects

```
numCircles = numel(radii);
```

% Pixel diameters

```
diameters = 2 * radii;
```

% Approximate areas

```
areas = pi * (radii.^2);
```

Limitations of Hough Detection

- Requires **reasonably strong edges**
- Sensitive to **radius range** selection
- May miss **low-contrast circles**
- Computationally heavier than simple thresholding

In practice, good **preprocessing** and sensible parameter choices are essential.

Key Insight

Hough-based circle detection provides a **direct and robust way** to detect and measure circular objects, as long as we supply a reasonable **radius range** and tune **sensitivity** for the image at hand.

3 Optical Character Recognition (OCR)

OCR: Why It Matters

OCR converts **images of text** into machine-readable characters.

It is used in:

- Industrial labels and serial numbers
- Keypads and instrumentation panels
- Packaging and medical devices
- Licence plates and signage

OCR only works well when the **preprocessing pipeline** is correct.

The OCR Pipeline

- 1 Image acquisition
- 2 Preprocessing (illumination correction, noise removal)
- 3 Segmentation (extract text from background)
- 4 Character extraction
- 5 OCR inference
- 6 Using constraints (CharacterSet, ROI, Layout mode)

OCR Example: Keypad Image

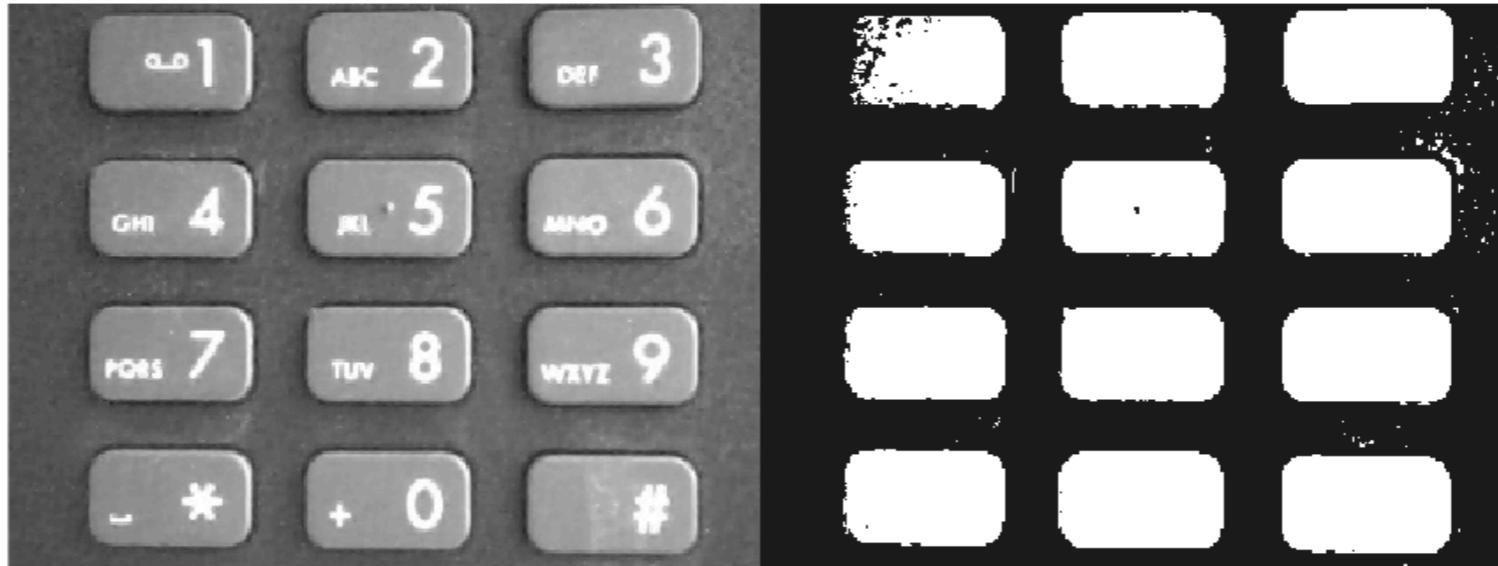


The digits are printed on a dark, uneven background → segmentation is difficult.

What Goes Wrong Initially?

- Background is nonuniform.
- Digits are small and low contrast.
- Default binarization in `ocr` fails.
- The OCR engine expects paragraphs and lines, not isolated buttons.
- Automatic layout analysis assumes “document structure,” which does not apply here.

Checking the Initial Binarization



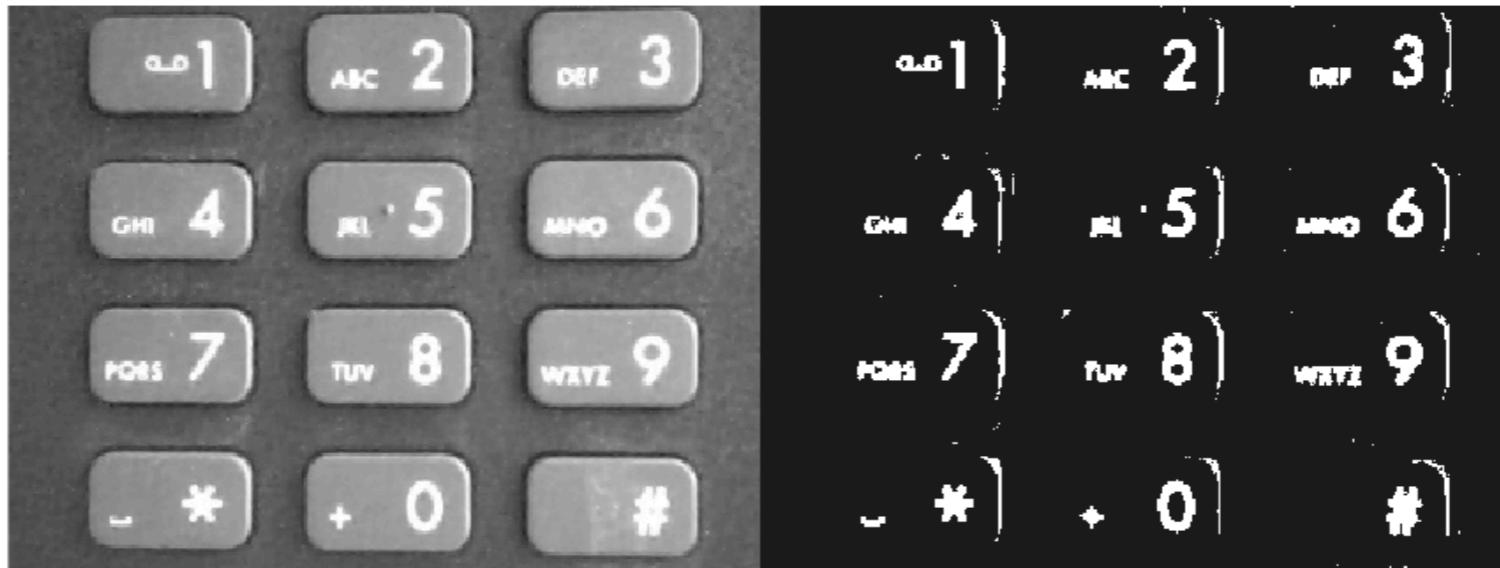
The binary image contains almost no text, so OCR returns nothing.

Strategy: Correct the Background

Use morphological operations to remove background variations:

- **Top-hat filtering** to suppress the dark key surfaces
- **Morphological reconstruction** to clean artifacts
- Binarization + inversion for OCR

Visualizing the Preprocessing



After correction, digits appear clearly in the binary image.

MATLAB Demo: Preprocessing for OCR

```
I = imread("keypad.jpg");
I = im2gray(I);

% Remove background variation
Icorrected = imtophat(I, strel("disk", 15));

% Further clean using morphological reconstruction
marker = imerode(Icorrected, strel("line", 10, 0));
Iclean = imreconstruct(marker, Icorrected);

% Binarize and invert (dark text on light background)
BW = imbinarize(Iclean);
BW = imcomplement(BW);
```

MATLAB

Try adjusting the SE size.

Too small → poor correction.

Too large → halos or missing digits.

Guiding the OCR Engine

Use constraints to improve accuracy:

- *CharacterSet* = "0123456789#"^{*}
- **LayoutAnalysis** = "none" or "block"
- Use **ROIs** around the buttons
- Ignore tiny artifacts and small labels

MATLAB Demo: Constrained OCR

```
results = ocr(BW, ...
    CharacterSet="0123456789*#", ...
    LayoutAnalysis="none");  
  
disp(results.Text);
```

MATLAB

Constraining the character set dramatically improves recognition.

Key Insight

OCR is not a single-step solution.

Success depends on:

- Correcting illumination
- Clean segmentation
- Using character constraints
- Selecting appropriate ROIs

This mirrors the keypad example exactly.

Summary & Reflection

Today's Key Messages

- A machine vision system is best understood as a **pipeline**.
- Each stage (preprocessing, segmentation, morphology) **affects what comes next**.
- Real applications require **careful combinations** of classical tools.
- Illumination correction, circular-object detection, and OCR, all demonstrate how small design choices affect outcomes.
- Robust systems rely on **understanding limitations and choosing the right tools**.

Connecting Back to the Pipeline

- 1** Scene → Sensor → Image
- 2** Preprocessing
- 3** Segmentation
- 4** Morphology & Cleaning
- 5** Measurement
- 6** Application-specific decisions

Each application we saw today followed this structure, even though the techniques within each stage differed.

Reflection Questions

- 1 Which application (illumination correction, circle detection, OCR) felt most intuitive to you?
- 2 Which stage of the pipeline do you think is **most fragile** in real-world images?
- 3 How would you build a pipeline for your own project or research task?

You may discuss in pairs or think individually before the lab session.

Exit Ticket



Before you leave, please complete the online form and provide your feedback.

1. One clear concept from today
2. One confusing concept
3. One thing to try in MATLAB this week

Scan the QR code or visit the link below.



 forms.office.com/e/YvWnAr6yJ

Questions & Support

- You are encouraged to ask questions anytime💡
 - During the lecture
 - In lab sessions
 - By email → m.k.heris@shu.ac.uk
- No question is too simple — asking questions:
 - Helps you learn faster
 - Builds confidence
 - Improves understanding for the whole class

