bhw02

December 1, 2025

Hoang Anh Nguyen

[1]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

[2]:

```
def rectify_document(img_bgr, corners):
    """
    Perform document rectification using perspective transformation.

    Args:
        img_bgr: Input image in BGR format
        corners: List of 4 corner points [top-left, top-right, bottom-right,␣
 ↪bottom-left]

    Returns:
        warped_img: Rectified document image
    """
    # Convert corners to numpy array
    src_points = np.array(corners, dtype=np.float32)

    # Width
    width_top = np.sqrt(((src_points[1][0] - src_points[0][0]) ** 2) +
                        ((src_points[1][1] - src_points[0][1]) ** 2))
    width_bottom = np.sqrt(((src_points[2][0] - src_points[3][0]) ** 2) +
                           ((src_points[2][1] - src_points[3][1]) ** 2))
    max_width = int(max(width_top, width_bottom))

    # Height
    height_left = np.sqrt(((src_points[3][0] - src_points[0][0]) ** 2) +
                          ((src_points[3][1] - src_points[0][1]) ** 2))
    height_right = np.sqrt(((src_points[2][0] - src_points[1][0]) ** 2) +
                           ((src_points[2][1] - src_points[1][1]) ** 2))
    max_height = int(max(height_left, height_right))

    # Define destination points for the rectified document (rectangle)
    dst_points = np.array([
        [0, 0],                          # Top-left
        [max_width - 1, 0],              # Top-right
        [max_width - 1, max_height - 1], # Bottom-right
        [0, max_height - 1]              # Bottom-left
    ], dtype=np.float32)

    # Compute the perspective transformation matrix
    homography_matrix = cv2.getPerspectiveTransform(src_points, dst_points)

    # Apply the perspective transformation to warp the image
    warped_img = cv2.warpPerspective(img_bgr, homography_matrix, (max_width,␣
 ↪max_height))

    return warped_img, homography_matrix
```

[3]:

```
# Define the image corner points
corners = [
    [1100., 300.],  # 1. Top-Left
    [2700., 1080.], # 2. Top-Right
    [2450., 3400.], # 3. Bottom-Right
    [300., 3150.]   # 4. Bottom-Left
]

image_path="data/input.jpg"
img_bgr = cv2.imread(image_path) # Load the image

# Document Rectification using Fixed Coordinates
warped_img, homography_matrix = rectify_document(img_bgr, corners)
```

[4]:

```
print("Homography Matrix (3x3):")
print(homography_matrix)
```

```
Homography Matrix (3x3):
[[ 8.23398274e-01  2.31129340e-01 -9.75076903e+02]
 [-4.99271624e-01  1.02414692e+00  2.41954710e+02]
 [-1.34454712e-04  5.13463331e-05  1.00000000e+00]]
```

[5]:

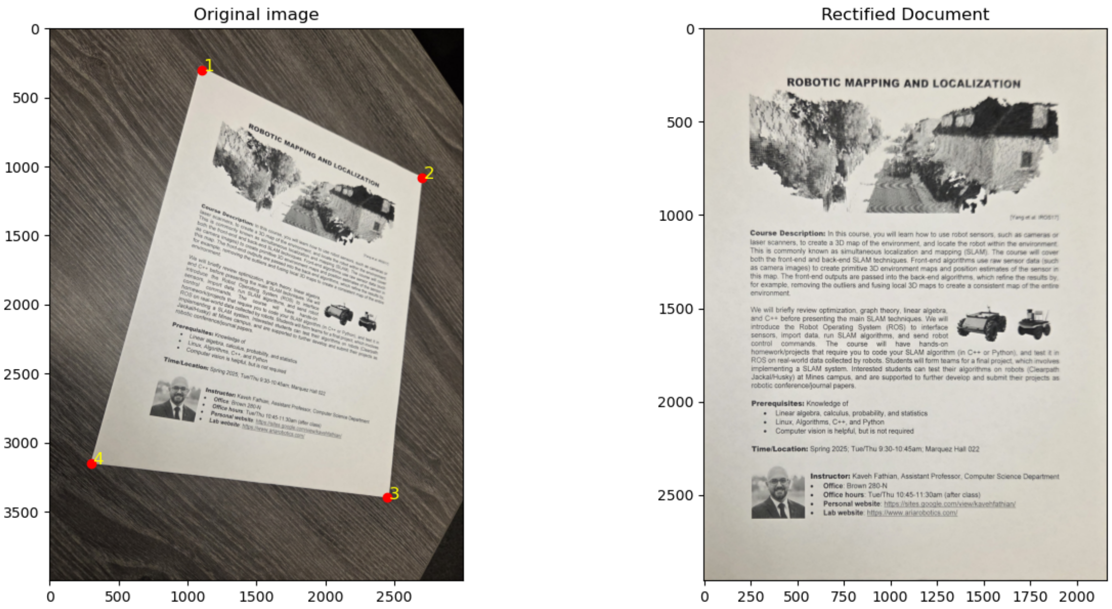
```
# Display Results
fig, axes = plt.subplots(1, 2, figsize=(15, 7))

img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB) # Convert to RGB for display␣
 ↪with matplotlib
axes[0].imshow(img_rgb) # Original Image
axes[0].set_title('Original image')

# Mark the selected points on the original image
for i, (x, y) in enumerate(corners):
    axes[0].plot(x, y, 'ro') # Red circle
```

```python
    axes[0].text(x + 10, y + 10, str(i + 1), color='yellow', fontsize=12)

warped_img_rgb = cv2.cvtColor(warped_img, cv2.COLOR_BGR2RGB) # Convert the warped image to RGB for display
axes[1].imshow(warped_img_rgb) # Warped/Rectified Image
axes[1].set_title('Rectified Document')
plt.show()
```



ROBOTIC MAPPING AND LOCALIZATION

Course Description: In this course, you will learn how to use robot sensors, such as cameras or laser scanners, to create a 3D map of the environment, and locate the robot within the environment. This is commonly known as simultaneous localization and mapping (SLAM). The course will cover both the front-end and back-end SLAM techniques. Front-end algorithms use raw sensor data (such as camera images) to create primitive 3D environment maps and position estimates of the sensor in this map. The front-end outputs are passed into the back-end algorithms, which refine the results by, for example, removing the outliers and fusing local 3D maps to create a consistent map of the entire environment.

We will briefly review optimization, graph theory, linear algebra, and C++ before presenting the main SLAM techniques. We will introduce the Robot Operating System (ROS) to interface sensors, import data, run SLAM algorithms, and send robot control commands. The course will have hands-on homework/projects that require you to code your SLAM algorithm (in C++ or Python), and test it in ROS on real-world data collected by robots. Students will form teams for a final project, which involves implementing a SLAM system. Interested students can test their algorithms on robots (Clearpath Jackal/Husky) at Mines campus, and are supported to further develop and submit their projects as robotic conference/journal papers.

Prerequisites: Knowledge of
- Linear algebra, calculus, probability, and statistics
- Linux, Algorithms, C++, and Python
- Computer vision is helpful, but is not required

Time/Location: Spring 2025; Tue/Thu 9:30-10:45am; Marquez Hall 022

Instructor: Kaveh Fathian, Assistant Professor, Computer Science Department
- **Office**: Brown 280-N
- **Office hours**: Tue/Thu 10:45-11:30am (after class)
- **Personal website**: https://sites.google.com/view/kavehfathian/
- **Lab website**: https://www.ariarobotics.com/