Visión Artificial VIAR25

Lab 1: PyTorch Setup, Camera Calibration & Image Transformations

Prof. David Olivieri

Uvigo

8 de septiembre de 2025

Resumen del Laboratorio 1

Objetivos del Laboratorio

Transformar la teoría matemática de la Clase 1 en código funcional. Al final de este laboratorio, habrás implementado los algoritmos básicos que impulsan los sistemas de visión por computador.

Qué Implementaremos Hoy

1. **Configuración del Entorno**: PyTorch + OpenCV + herramientas de visualización
2. **Entrada/Salida y Manipulación de Imágenes**: Carga, transformación y visualización de imágenes
3. **Calibración de Cámara**: Método de Zhang con detección de tablero de ajedrez
4. **Estimación de Homografía**: Algoritmo DLT con RANSAC
5. **Corrección de Distorsión de Lente**: Implementación del modelo Brown-Conrady
6. **Transformaciones de Espacios de Color**: Conversión RGB $\leftrightarrow$ HSV

Estructura del Laboratorio

- **Código de Demostración**: Implementación paso a paso con explicaciones
- **Ejercicios**: Retos de programación prácticos para comprobar la comprensión
- **Preguntas**: Conexiones teóricas para profundizar en el aprendizaje
- **Desafíos Avanzados**: Implementaciones adicionales para obtener créditos extra

Configuración del Entorno & Fundamentos de PyTorch

Librerías Necesarias

```python
import torch
import torchvision
import torchvision.transforms as transforms
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import RANSACRegressor
import glob

```

Ejercicio 1.1: Operaciones con Tensores en PyTorch

Tarea: Crear una función que convierta entre arrays de NumPy y tensores de PyTorch

```python
def numpy_to_torch(img_np):
    """Convertir un array numpy HxWxC en un tensor torch CxHxW"""
    # TODO: Implementar conversi n
    pass

def torch_to_numpy(img_torch):
    """Convertir un tensor torch CxHxW en un array numpy HxWxC"""
    # TODO: Implementar conversi n
    pass

```

Preguntas:

- ¿Por qué PyTorch y OpenCV utilizan diferentes órdenes de canales?
- ¿Cuándo usarías tensores en GPU frente a tensores en CPU para procesamiento de imágenes?

Entrada/Salida de Imágenes y Transformaciones Básicas

Demo: Carga y Visualización de Imágenes

```python
def load_image(filepath):
    """Cargar imagen usando OpenCV y convertir a RGB"""
    img_bgr = cv2.imread(filepath)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    return img_rgb

def visualize_images(images, titles, figsize=(15, 5)):
    """Mostrar m ltiples im genes en una fila"""
    fig, axes = plt.subplots(1, len(images), figsize=figsize)
    for i, (img, title) in enumerate(zip(images, titles)):
        axes[i].imshow(img, cmap='gray' if len(img.shape)==2 else None)
        axes[i].set_title(title)
        axes[i].axis('off')
    plt.tight_layout()
    plt.show()

```

Ejercicio 1.2: Transformaciones Geométricas

Tarea: Implementar la jerarquía de transformaciones de la Clase 1

```python
def apply_transformation(img, transform_matrix):
    """Aplicar una matriz de transformaci n 3x3 a una imagen"""
    # TODO: Usar cv2.warpPerspective
    pass

def create_similarity_transform(scale, rotation, translation):
    """Crear matriz de transformaci n de similitud"""
    # TODO: Implementar usando las ecuaciones de la clase
    pass

```

Detección de Esquinas para Calibración

Demo: Detección de Esquinas de un Tablero de Ajedrez

```python
def detect_checkerboard_corners(img, pattern_size):
    """Detectar esquinas de un tablero de ajedrez con precisi n sub-p xel"""
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    # Encontrar esquinas
    ret, corners = cv2.findChessboardCorners(gray, pattern_size, None)

    if ret:
        # Refinar esquinas a precisi n sub-p xel
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
                    30, 0.001)
        corners = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)

    return ret, corners

def visualize_corners(img, corners, pattern_size):
    """Dibujar las esquinas detectadas en la imagen"""
    img_with_corners = img.copy()
    cv2.drawChessboardCorners(img_with_corners, pattern_size, corners, True)
    return img_with_corners

```

Ejercicio 1.3: Análisis de la Detección de Esquinas

Preguntas:

- ¿Qué ocurre si el tablero de ajedrez está desenfocado? Prueba con diferentes calidades de imagen.
- ¿Cómo afecta la precisión en la detección de esquinas a los resultados de la calibración?
- Implementa una función para medir la repetibilidad de la detección de esquinas en múltiples vistas.

Zhang's Camera Calibration Algorithm

Demo: Complete Calibration Pipeline

```python
def calibrate_camera(calibration_images, pattern_size, square_size):
    """Implement Zhang's camera calibration method"""

    # Prepare object points (3D points in real world space)
    objp = np.zeros((pattern_size[0] * pattern_size[1], 3), np.float32)
    objp[:,:2] = np.mgrid[0:pattern_size[0], 0:pattern_size[1]].T.reshape(-1,2)
    objp *= square_size

    # Arrays to store object points and image points
    objpoints = []  # 3D points in real world space
    imgpoints = []  # 2D points in image plane

    for img_path in calibration_images:
        img = load_image(img_path)
        ret, corners = detect_checkerboard_corners(img, pattern_size)

        if ret:
            objpoints.append(objp)
            imgpoints.append(corners)

    # Perform calibration
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
        objpoints, imgpoints, img.shape[:2][::-1], None, None)

    return ret, mtx, dist, rvecs, tvecs

```

Algoritmo de Calibración de Cámara de Zhang

Demo: Flujo Completo de Calibración

```
def calibrate_camera(calibration_images, pattern_size, square_size):
    """Implementar el m todo de calibraci n de c mara de Zhang"""

    # Preparar puntos objeto (puntos 3D en el espacio del mundo real)
    objp = np.zeros((pattern_size[0] * pattern_size[1], 3), np.float32)
    objp[:,:2] = np.mgrid[0:pattern_size[0], 0:pattern_size[1]].T.reshape(-1,2)
    objp *= square_size

    # Listas para almacenar puntos objeto y puntos de imagen
    objpoints = []  # Puntos 3D en el espacio del mundo real
    imgpoints = []  # Puntos 2D en el plano de la imagen

    for img_path in calibration_images:
        img = load_image(img_path)
        ret, corners = detect_checkerboard_corners(img, pattern_size)

        if ret:
            objpoints.append(objp)
            imgpoints.append(corners)

    # Realizar la calibraci n
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
        objpoints, imgpoints, img.shape[:2][::-1], None, None)

    return ret, mtx, dist, rvecs, tvecs

```

Estimación de Homografía con DLT

Demo: Implementación de la Transformada Lineal Directa

```python
def estimate_homography_dlt(src_pts, dst_pts):
    """Estimate homography using Direct Linear Transform"""
    assert len(src_pts) >= 4, "Need at least 4 point correspondences"

    # Normalize points for numerical stability
    src_norm, T1 = normalize_points(src_pts)
    dst_norm, T2 = normalize_points(dst_pts)

    # Set up linear system Ah = 0
    A = []
    for i in range(len(src_norm)):
        x, y = src_norm[i]
        u, v = dst_norm[i]

        A.append([0, 0, 0, -x, -y, -1, v*x, v*y, v])
        A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])

    A = np.array(A)

    # Solve using SVD
    U, S, Vt = np.linalg.svd(A)
    h = Vt[-1, :] / Vt[-1, -1]  # Normalize by last element
    H_norm = h.reshape(3, 3)

    # Denormalize
    H = np.linalg.inv(T2) @ H_norm @ T1

    return H / H[2, 2]  # Normalize by bottom-right element

```

Normalización de Puntos para Estabilidad Numérica

Demo: Normalización de Hartley

```python
def normalize_points(points):
    """Normalize points for DLT numerical stability"""
    points = np.array(points, dtype=np.float32)

    # Compute centroid
    centroid = np.mean(points, axis=0)

    # Compute mean distance to centroid
    distances = np.linalg.norm(points - centroid, axis=1)
    scale = np.sqrt(2) / np.mean(distances)

    # Create transformation matrix
    T = np.array([
        [scale, 0, -scale * centroid[0]],
        [0, scale, -scale * centroid[1]],
        [0, 0, 1]
    ])

    # Apply transformation
    points_hom = np.column_stack([points, np.ones(len(points))])
    points_norm = (T @ points_hom.T).T
    points_norm = points_norm[:, :2] / points_norm[:, [2]]

    return points_norm, T

```

Ejercicio 1.5: Robustez de la Homografía

Tarea: Comparar DLT normalizado vs. no normalizado

- Implementar DLT sin normalización
- Probar con puntos en diferentes escalas (ej. [0,1] vs [0,1000])
- Medir el condicionamiento de la matriz $A^T A$

Pregunta: ¿Por qué la normalización mejora la estabilidad numérica?

RANSAC para Estimación Robusta de Homografía

Demo: Implementación de RANSAC

```python
def ransac_homography(src_pts, dst_pts, threshold=3.0, max_iters=1000):
    """Robust homography estimation using RANSAC"""
    best_H = None
    best_inliers = []
    best_score = 0

    n_points = len(src_pts)

    for _ in range(max_iters):
        # Randomly sample 4 points
        sample_idx = np.random.choice(n_points, 4, replace=False)
        sample_src = src_pts[sample_idx]
        sample_dst = dst_pts[sample_idx]

        # Estimate homography from sample
        H = estimate_homography_dlt(sample_src, sample_dst)

        # Count inliers
        inliers = []
        for i in range(n_points):
            # Transform source point
            src_hom = np.append(src_pts[i], 1)
            dst_pred_hom = H @ src_hom
            dst_pred = dst_pred_hom[:2] / dst_pred_hom[2]

            # Compute reprojection error
            error = np.linalg.norm(dst_pred - dst_pts[i])
            if error < threshold:
                inliers.append(i)

        # Update best model if better
        if len(inliers) > best_score:
            best_score = len(inliers)
            best_inliers = inliers
            best_H = H

    return best_H, best_inliers

```

Corrección de la Distorsión de Lente

Demo: Modelo de Distorsión Brown-Conrady

```python
def undistort_points(points, mtx, dist):
    """Undistort points using camera parameters"""
    # Convert to homogeneous coordinates
    points_hom = np.column_stack([points, np.ones(len(points))])

    # Apply inverse camera matrix to get normalized coordinates
    points_norm = np.linalg.inv(mtx) @ points_hom.T
    points_norm = points_norm[:2, :].T

    # Apply distortion correction
    k1, k2, p1, p2, k3 = dist.ravel()

    points_corrected = []
    for x, y in points_norm:
        r2 = x*x + y*y

        # Radial distortion correction
        radial_factor = 1 + k1*r2 + k2*r2*r2 + k3*r2*r2*r2

        # Tangential distortion correction
        x_corrected = x * radial_factor + 2*p1*x*y + p2*(r2 + 2*x*x)
        y_corrected = y * radial_factor + p1*(r2 + 2*y*y) + 2*p2*x*y

        points_corrected.append([x_corrected, y_corrected])

    # Convert back to pixel coordinates
    points_corrected = np.array(points_corrected)
    points_corrected_hom = np.column_stack([points_corrected, np.ones(len(points_corrected))])
    points_pixel = (mtx @ points_corrected_hom.T).T

    return points_pixel[:, :2]

```

Transformaciones de Espacios de Color

Demo: Conversión de RGB a HSV

```python
def rgb_to_hsv_manual(rgb_img):
    """Manual RGB to HSV conversion following lecture algorithm"""
    rgb_normalized = rgb_img.astype(np.float32) / 255.0
    h, w, c = rgb_img.shape
    hsv_img = np.zeros_like(rgb_normalized)

    for i in range(h):
        for j in range(w):
            r, g, b = rgb_normalized[i, j]

            # Value (brightness)
            v = max(r, g, b)
            delta = v - min(r, g, b)

            # Saturation
            s = 0 if v == 0 else delta / v

            # Hue
            if delta == 0:
                h_val = 0
            elif v == r:
                h_val = 60 * ((g - b) / delta % 6)
            elif v == g:
                h_val = 60 * ((b - r) / delta + 2)
            else:  # v == b
                h_val = 60 * ((r - g) / delta + 4)

            hsv_img[i, j] = [h_val, s, v]

    return hsv_img

```

Ejercicio 1.6: Análisis de Espacios de Color

Tarea: Comparar la conversión manual con la de OpenCV

- Implementar la conversión de HSV a RGB
- Medir las diferencias numéricas entre ambas implementaciones
- Visualizar distribuciones de color en distintos tipos de imágenes

Implementación de Ecualización de Histograma

Ejercicio 1.7: Mejora Avanzada de Imágenes

Tarea: Implementar ecualización adaptativa de histograma

```
def histogram_equalization(img):
    """Global histogram equalization from scratch"""
    # TODO: Implementar el algoritmo explicado en clase
    pass

def adaptive_histogram_equalization(img, tile_size=(8, 8)):
    """CLAHE (Contrast Limited Adaptive Histogram Equalization)"""
    # TODO: Implementar ecualizaci n basada en bloques (tiles)
    # TODO: Aplicar limitaci n de contraste
    # TODO: Interpolaci n bilineal entre bloques
    pass

```

Preguntas:

- ¿Cuándo falla la ecualización global de histograma?
- ¿Cómo afecta el tamaño de bloque a los resultados de la ecualización adaptativa?
- ¿Cuáles son los compromisos entre mejora de contraste y amplificación del ruido?

Ejercicios Avanzados & Retos de Extensión

Problemas Desafío (Créditos Extra)

Para estudiantes que busquen una comprensión más profunda y experiencia en implementación.

Reto 1: Calibración Multi-Cámara

- Implementar la calibración de una cámara estéreo
- Estimar la pose relativa entre dos cámaras
- Validar las restricciones de la geometría epipolar
- **Conexión Teórica**: ¿Cómo se relaciona esto con la estimación de la matriz fundamental?

Reto 2: Calibración en Tiempo Real

- Implementar la calibración en vivo usando una cámara web
- Añadir detección automática de tablero de ajedrez y evaluación de calidad
- Visualizar el progreso de la calibración en tiempo real
- **Conexión Teórica**: ¿Cuál es el número mínimo de vistas necesarias?

Reto 3: Patrones de Calibración Personalizados

- Implementar calibración usando puntos circulares en lugar de tableros de ajedrez
- Comparar precisión y robustez frente a los patrones de tablero de ajedrez
- Manejar la visibilidad parcial del patrón
- **Conexión Teórica**: ¿Cómo afecta la elección del patrón a la precisión en la detección de esquinas?

Preguntas Teóricas para una Comprensión Más Profunda

Preguntas Conceptuales

1. **Modelo de Cámara**: ¿Por qué el modelo de cámara estenopeica es insuficiente para cámaras reales? ¿Cuándo usarías modelos ojo de pez frente a modelos de perspectiva?
2. **Teoría de la Calibración**: Demuestra que se necesitan al menos 6 correspondencias de puntos para estimar todos los parámetros de la cámara. ¿Qué ocurre con exactamente 6 puntos?
3. **Restricciones de la Homografía**: ¿Bajo qué condiciones una homografía no está definida? ¿Cuántos grados de libertad tiene una homografía y por qué?
4. **Modelos de Distorsión**: ¿Por qué los coeficientes de distorsión de la lente tienen efectos diferentes? ¿Cuándo serían necesarios términos de orden superior?
5. **Estabilidad Numérica**: Explica por qué el algoritmo DLT se vuelve inestable sin normalización. ¿Qué mide el número de condición?

Preguntas para Discusión

- ¿Cómo modificarías el método de Zhang para un escenario con cámara en movimiento?
- ¿Qué nivel de precisión de calibración se requiere para diferentes aplicaciones (RA, robótica, metrología)?
- ¿Cómo validarías la calidad de la calibración sin disponer de *ground truth*?

Evaluación del Laboratorio & Entregables

Qué Entregar

1. **Funciones Implementadas**: Todas las funciones de los ejercicios con la documentación adecuada
2. **Resultados de Calibración**: Parámetros de cámara obtenidos de tu propio conjunto de datos de calibración
3. **Informe de Análisis**:
   - Análisis del error de reproyección
   - Comparación de diferentes algoritmos (DLT vs RANSAC)
   - Discusión de casos de fallo y limitaciones
4. **Visualización**: Gráficas claras que muestren la calidad de la calibración y el comportamiento de los algoritmos

Criterios de Calificación

- **Corrección (40 %)**: Los algoritmos producen resultados matemáticamente correctos
- **Calidad del Código (25 %)**: Implementación limpia, bien documentada y eficiente
- **Análisis (25 %)**: Discusión reflexiva de los resultados y conexiones teóricas
- **Creatividad (10 %)**: Retos de extensión e ideas novedosas

Avance del Próximo Laboratorio

Lab 2: Operaciones de convolución personalizadas, aumento de datos, canalizaciones de preprocesamiento — construyendo la base para los enfoques de aprendizaje profundo.