LABORATORIO PRÁCTICO

Procesamiento de Imágenes RGB-D en Tiempo Real

Pipeline Completo de Visión Artificial

Dr. Oscar Loyola Valenzuela
Procesamiento de Imágenes

Noviembre 2025

Resumen

Este laboratorio presenta un enfoque práctico e integrador para el procesamiento de imágenes en tiempo real, combinando técnicas fundamentales de visión artificial (filtrado, detección de bordes, morfología matemática, transformadas de Hough) aplicadas a flujos de video capturados con cámaras Intel RealSense D435i o webcams convencionales. Los estudiantes desarrollarán sistemas completos que clasifican formas geométricas e inspeccionan calidad en tiempo real, aplicando conceptos teóricos a problemas del mundo real.

Índice

1. Introducción

El procesamiento de imágenes en tiempo real representa uno de los desafíos más importantes en visión artificial moderna. A diferencia del análisis de imágenes estáticas, trabajar con video requiere considerar aspectos temporales, eficiencia computacional y robustez ante variaciones dinámicas del entorno.

1.1. Contexto

Las aplicaciones industriales modernas demandan sistemas de visión artificial capaces de:

- Procesar video a 30+ FPS manteniendo precisión
- Adaptarse a condiciones variables de iluminación
- Clasificar y medir objetos automáticamente
- Generar reportes y estadísticas en tiempo real

Este laboratorio aborda estos requisitos mediante la implementación de dos sistemas completos que integran múltiples técnicas de procesamiento de imágenes.

2. Objetivos

2.1. Objetivos Generales

- Integrar técnicas de procesamiento de imágenes en sistemas de tiempo real
- Desarrollar habilidades prácticas de implementación en Python/OpenCV
- Aplicar conocimientos teóricos a problemas industriales concretos

2.2. Objetivos Específicos

1. Implementar y comparar diferentes técnicas de filtrado espacial
2. Aplicar detección de bordes y morfología matemática para segmentación
3. Utilizar transformadas de Hough para detección de formas
4. Extraer y analizar características geométricas de objetos
5. Desarrollar clasificadores de formas basados en análisis de contornos
6. Integrar mediciones 3D usando información de profundidad (RealSense)
7. Optimizar pipelines para procesamiento en tiempo real (30 FPS)

3. Requisitos

3.1. Hardware

- **Opción A**: Intel RealSense D435i (2 disponibles - grupos en rotación)
- **Opción B**: Webcam USB o laptop integrada
- Procesador: Intel Core i5 o superior (recomendado)
- RAM: 8GB mínimo

3.2. Software

- Python 3.8+
- OpenCV 4.5+ (`opencv-python`)
- NumPy
- Matplotlib
- pyrealsense2 (solo para RealSense)
- Spyder o Jupyter Notebook

3.3. Instalación

```
# Entorno Conda (recomendado)
conda create -n vision python=3.10
conda activate vision

# Paquetes principales
pip install opencv-python numpy matplotlib scipy

# Solo para RealSense
pip install pyrealsense2

# Verificacion
python -c "import cv2; print(cv2.__version__)"
```

Listing 1: Instalación de dependencias

Importante - Intel RealSense SDK

Para usar la cámara RealSense, debe estar instalado el Intel RealSense SDK 2.0:

- Descargar de: `https://github.com/IntelRealSense/librealsense/releases`
- Instalar: `Intel.RealSense.SDK-WIN10-[version].exe`
- Verificar con Intel RealSense Viewer

4. Conceptos Fundamentales Integrados

Este laboratorio integra los siguientes conceptos vistos en clase:

4.1. Filtrado Espacial

- **Gaussian Blur**: Reducción de ruido gaussiano
- **Median Filter**: Eliminación de ruido sal y pimienta
- **Bilateral Filter**: Suavizado preservando bordes

4.2. Detección de Bordes

- **Canny**: Detector óptimo de bordes
- **Sobel/Scharr**: Gradientes direccionales
- **Laplaciano**: Segunda derivada

4.3. Morfología Matemática

- **Erosión/Dilatación**: Operaciones básicas
- **Opening/Closing**: Limpieza de ruido y relleno
- **Gradient**: Detección de contornos

4.4. Análisis de Contornos

- **Aproximación poligonal**: Douglas-Peucker
- **Momentos**: Centroide, área, orientación
- **Características**: Circularidad, compacidad, relación de aspecto

4.5. Transformadas

- **Hough Transform**: Detección de líneas y círculos
- **Características HOG**: Histograma de gradientes orientados

5. Ejercicio 1: Clasificador de Formas Geométricas

5.1. Descripción

Desarrollo de un sistema que clasifica en tiempo real formas geométricas (círculos, cuadrados, triángulos, rectángulos) capturadas por video. El sistema debe:

- Detectar y segmentar objetos
- Clasificar formas usando análisis de contornos
- Medir dimensiones (con RealSense)
- Mantener estadísticas de detección
- Procesar a velocidad interactiva (¿15 FPS)

5.2. Pipeline de Procesamiento

El sistema sigue el siguiente flujo:

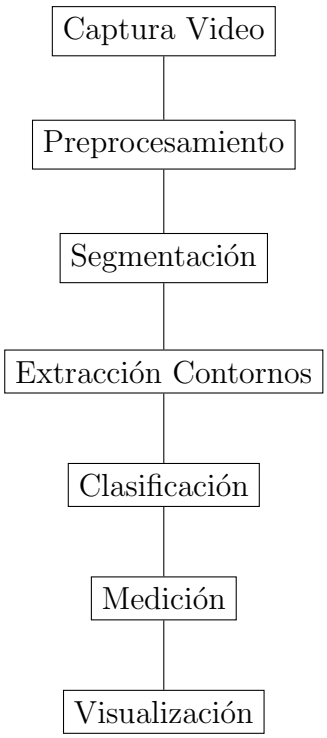

Figura 1: Pipeline del clasificador de formas

5.3. Implementación

5.3.1. Estructura de la Clase Principal

```python
class ShapeClassifier:
    def __init__(self, use_realsense=True):
        """
        Inicializa el clasificador

        Parameters:
        -----------
        use_realsense : bool
            True para RealSense, False para webcam
        """
        self.use_realsense = use_realsense
        self.shape_counts = {
            'circulo': 0,
            'cuadrado': 0,
            'triangulo': 0,
            'rectangulo': 0,
            'otro': 0
        }

        # Inicializacion de camara...
```

Listing 2: Estructura del clasificador (parcial)

5.3.2. Preprocesamiento

El método de preprocesamiento combina filtrado y segmentación:

```python
def preprocess_frame(self, color_image, depth_image=None):
    """
    PASO 1: PREPROCESAMIENTO
    Aplica filtros y segmentacion
    """
    # 1. Convertir a escala de grises
    gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)

    # 2. FILTRADO: Reducir ruido con Gaussian Blur
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # 3. SEGMENTACION
    if depth_image is not None:
        # Con RealSense: usar profundidad
        mask_depth = np.logical_and(
            depth_image > 400,
            depth_image < 1200
        ).astype(np.uint8) * 255

        _, mask_intensity = cv2.threshold(blurred, 60, 255,
                                          cv2.THRESH_BINARY)
        mask = cv2.bitwise_and(mask_intensity, mask_depth)
    else:
        # Con webcam: solo intensidad
        _, mask = cv2.threshold(blurred, 60, 255,
                                cv2.THRESH_BINARY)

    # 4. MORFOLOGIA: Limpiar mascara
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
```

```
                                        (5, 5))
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel,
                            iterations=2)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel,
                            iterations=1)

    return blurred, mask
```

Listing 3: Método de preprocesamiento

Concepto Clave: Segmentación Dual

El uso combinado de profundidad e intensidad permite segmentación más robusta:

- **Profundidad**: Elimina fondo y objetos lejanos
- **Intensidad**: Separa objetos con contraste
- **AND lógico**: Solo píxeles que cumplen ambas condiciones

5.3.3. Clasificación de Formas

La clasificación se basa en el análisis del contorno:

```
def classify_shape(self, contour):
    """
    PASO 3: CLASIFICACION DE FORMA
    Usa analisis de contorno y aproximacion poligonal
    """
    # Calcular perimetro
    perimeter = cv2.arcLength(contour, True)

    # Aproximacion poligonal (Douglas-Peucker)
    epsilon = 0.04 * perimeter
    approx = cv2.approxPolyDP(contour, epsilon, True)

    # Numero de vertices
    vertices = len(approx)

    # Calcular area y circularidad
    area = cv2.contourArea(contour)
    circularity = (4 * np.pi * area / (perimeter * perimeter)
                   if perimeter > 0 else 0)

    # CLASIFICACION
    shape_name = "otro"

    if vertices == 3:
        shape_name = "triangulo"

    elif vertices == 4:
        # Distinguir cuadrado de rectangulo
        x, y, w, h = cv2.boundingRect(approx)
        aspect_ratio = float(w) / h if h > 0 else 0

```

```
32          if 0.95 <= aspect_ratio <= 1.05:
33              shape_name = "cuadrado"
34          else:
35              shape_name = "rectangulo"
36
37      elif vertices > 8:
38          # Muchos vertices -> probablemente circulo
39          if circularity > 0.7:
40              shape_name = "circulo"
41
42      return {
43          'name': shape_name,
44          'vertices': vertices,
45          'area': area,
46          'perimeter': perimeter,
47          'circularity': circularity,
48          'approx': approx
49      }
```

Listing 4: Método de clasificación

Algoritmo de Douglas-Peucker

La aproximación poligonal reduce el número de puntos del contorno manteniendo su forma:

$$\epsilon = k \cdot P \tag{1}$$

donde P es el perímetro y k es un factor (típicamente 0.02-0.05).
Un ϵ pequeño preserva más detalles, uno grande simplifica más.

5.3.4. Cálculo de Circularidad

La circularidad es una medida de qué tan parecido es un contorno a un círculo perfecto:

$$\text{Circularidad} = \frac{4\pi A}{P^2} \tag{2}$$

donde:

- A = área del contorno
- P = perímetro del contorno

Valores:

- Círculo perfecto: $C = 1{,}0$
- Cuadrado: $C \approx 0{,}785$
- Formas irregulares: $C < 0{,}5$

5.3.5. Medición con Profundidad

Solo disponible con RealSense:

```python
def calculate_dimensions(self, contour, depth_frame=None):
    """
    PASO 4: MEDICION (si hay profundidad)
    Calcula dimensiones reales usando profundidad
    """
    x, y, w, h = cv2.boundingRect(contour)
    cx, cy = x + w//2, y + h//2

    dimensions = {
        'width_px': w,
        'height_px': h,
        'center': (cx, cy)
    }

    if depth_frame is not None:
        # Obtener distancia
        distance = depth_frame.get_distance(cx, cy)

        if distance > 0:
            # Obtener intr nsecos
            intrinsics = (depth_frame.profile
                          .as_video_stream_profile()
                          .get_intrinsics())

            # Factor de conversion pixeles a mm
            pixels_to_meters = distance / intrinsics.fx

            dimensions['width_m'] = w * pixels_to_meters
            dimensions['height_m'] = h * pixels_to_meters
            dimensions['distance_m'] = distance

    return dimensions
```

Listing 5: Medición de dimensiones reales

Conversión Píxel-Métrico

La conversión de píxeles a unidades métricas depende de:

$$d_{\text{real}} = d_{\text{píxeles}} \times \frac{Z}{f_x} \tag{3}$$

donde:

- d_{real} = dimensión en metros
- $d_{\text{píxeles}}$ = dimensión en píxeles
- Z = distancia al objeto (profundidad)
- f_x = distancia focal en píxeles (de intrínsecos)

5.4. Código Completo

El código completo está disponible en el archivo adjunto: `shape_classifier.py`
Aspectos destacados del código:

- Funciona con RealSense o webcam (parámetro `use_realsense`)
- Parámetros ajustables en tiempo real (trackbars)
- Visualización multi-panel (original, máscara, bordes, resultado)
- Estadísticas en tiempo real
- Historial de detecciones

5.5. Ejecución

```
# Con RealSense
python shape_classifier.py --camera realsense

# Con webcam
python shape_classifier.py --camera webcam

# O directamente en Python/Spyder
classifier = ShapeClassifier(use_realsense=True)
classifier.run()
```

Listing 6: Ejecutar clasificador

5.6. Instrucciones de Uso

1. Ejecutar el programa
2. Colocar objetos con formas geométricas claras frente a la cámara
3. Ajustar parámetros usando trackbars si es necesario:
   - **Min Area**: Área mínima para detectar (filtrar ruido)
   - **Threshold**: Umbral de intensidad para segmentación
4. Observar clasificación en tiempo real
5. Presionar 's' para guardar frames interesantes
6. Presionar 'r' para resetear estadísticas
7. Presionar 'q' para salir

6. Tareas Específicas para Estudiantes

Tarea 1: Análisis de Filtros (30 minutos)

Objetivo: Comparar efectos de diferentes filtros en la clasificación
Procedimiento:

1. Modificar el método `preprocess_frame()` para usar:
   - Gaussian Blur con kernel sizes: 3, 5, 7, 9
   - Median Blur
   - Bilateral Filter
2. Para cada filtro, medir durante 100 frames:
   - Número promedio de formas detectadas
   - Precisión de clasificación (usar objetos conocidos)
   - FPS (frames por segundo)
3. Crear tabla comparativa:

Filtro	# Formas	Precisión	FPS
Gaussian 3x3			
Gaussian 5x5			
Gaussian 7x7			
Gaussian 9x9			
Median 5			
Bilateral			

Cuadro 1: Resultados comparativos de filtros

Preguntas de análisis:

- ¿Qué filtro proporciona mejor balance precisión/velocidad?
- ¿Cómo afecta el tamaño del kernel al rendimiento?
- ¿En qué casos el Bilateral Filter es superior?

Entregable: Tabla completada + gráficas + análisis (1 página)

Tarea 2: Mejora de Clasificación (30 minutos)

Objetivo: Mejorar la precisión del clasificador
Implementar en `classify_shape()`:

1. **Relación de aspecto mejorada**:

```python
# Distinguir triangulos equilateros vs isosceles
if shape_name == 'triangulo':
    # Calcular longitudes de lados
    sides = []
    for i in range(len(approx)):
        p1 = approx[i][0]
        p2 = approx[(i+1) % len(approx)][0]
        length = np.linalg.norm(p1 - p2)
        sides.append(length)

    # Clasificar por uniformidad de lados
    sides_std = np.std(sides)
    if sides_std < 5:  # Umbral a ajustar
        subtype = "equilatero"
    else:
        subtype = "isosceles"

```

2. **Compacidad**:

$$\text{Compacidad} = \frac{P^2}{4\pi A} \tag{4}$$

   Valores típicos:

   - Círculo: $\approx 1{,}0$
   - Cuadrado: $\approx 1{,}27$
   - Triángulo equilátero: $\approx 1{,}65$

3. **Momentos de Hu**:

```python
moments = cv2.moments(contour)
hu_moments = cv2.HuMoments(moments)

# Los momentos de Hu son invariantes a:
# - Traslacion
# - Rotacion
# - Escala

```

Validación:

- Crear dataset de 5 objetos de cada forma
- Medir accuracy de clasificación
- **Meta**: Accuracy $> 85\,\%$

Entregable: Código modificado + tabla de accuracy + análisis

Tarea 3: Aplicación Práctica - Contador Automático (45 minutos)

Objetivo: Desarrollar sistema de conteo para simulación de línea de producción
Especificaciones:

- Dibujar línea virtual de conteo en pantalla (horizontal o vertical)
- Detectar cuando un objeto cruza la línea
- Clasificar el objeto al cruzar
- Mantener contadores separados por tipo de forma
- Evitar contar el mismo objeto múltiples veces
- Calcular throughput (objetos/minuto)

Algoritmo de detección de cruce:

```
def detect_line_crossing(self, centroid, obj_id):
    """
    Detecta si un objeto cruzo la linea
    """
    cx, cy = centroid

    # Verificar posicion anterior
    if obj_id in self.last_positions:
        last_cy = self.last_positions[obj_id]

        # Deteccion de cruce hacia abajo
        if (last_cy < self.line_position and
            cy >= self.line_position):
            return True, 'down'

        # Deteccion de cruce hacia arriba
        elif (last_cy > self.line_position and
              cy <= self.line_position):
            return True, 'up'

    # Actualizar posicion
    self.last_positions[obj_id] = cy

    return False, None
```

Funcionalidades requeridas:

1. Tracking simple de objetos entre frames
2. Detección de cruce bidireccional
3. Visualización de línea y contadores
4. Gráfica de throughput en tiempo real
5. Reporte final con estadísticas

Entregable:

- Código completo funcional
- Video de demostración (30 segundos)
- Reporte con throughput medido

7. Ejercicio 2: Sistema de Inspección de Calidad

7.1. Descripción

Sistema avanzado que inspecciona objetos en tiempo real verificando:

- **Forma correcta**: Verificación geométrica precisa
- **Dimensiones**: Dentro de tolerancias especificadas
- **Defectos**: Detección de agujeros, grietas, irregularidades
- **Color**: Uniformidad (opcional)

7.2. Especificaciones de Calidad

Forma	Parámetro	Especificación
Círculo	Diámetro Circularidad	30-50 mm $> 0{,}8$
Cuadrado	Lado Aspect Ratio Ángulos	30-50 mm $0{,}9 - 1{,}1$ $85° - 95°$

Cuadro 2: Especificaciones de inspección

7.3. Pipeline de Inspección

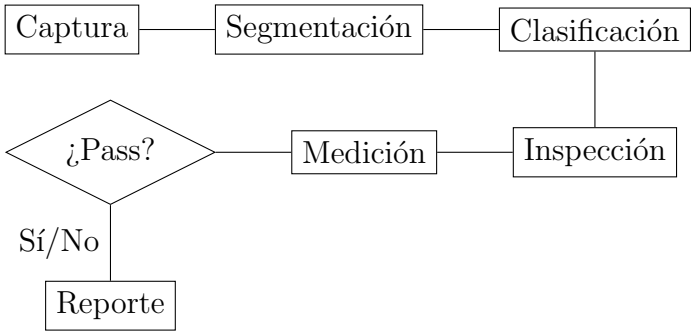

Figura 2: Pipeline de inspección de calidad

7.4. Implementación de Inspecciones

7.4.1. Inspección de Forma

```
def inspect_shape(self, contour, shape_name):
    """
    Inspecciona si la forma cumple especificaciones
    """
    defects = []
```

```
6
7       perimeter = cv2.arcLength(contour, True)
8       area = cv2.contourArea(contour)
9
10      if perimeter == 0:
11          return False, ["Perimetro invalido"]
12
13      circularity = 4 * np.pi * area / (perimeter ** 2)
14
15      if shape_name == 'circulo':
16          # Verificar circularidad
17          min_circ = self.specs['circulo']['min_circularity']
18          if circularity < min_circ:
19              defects.append(
20                  f"Baja circularidad: {circularity:.2f}"
21              )
22
23      elif shape_name == 'cuadrado':
24          # Verificar rectangularidad
25          x, y, w, h = cv2.boundingRect(contour)
26          aspect_ratio = float(w) / h if h > 0 else 0
27
28          max_diff = self.specs['cuadrado']['max_aspect_ratio_diff']
29          if abs(aspect_ratio - 1.0) > max_diff:
30              defects.append(
31                  f"No es cuadrado: AR={aspect_ratio:.2f}"
32              )
33
34      # Verificar agujeros internos
35      if self.has_holes(contour):
36          defects.append("Contiene agujeros")
37
38      passed = len(defects) == 0
39      return passed, defects
```

Listing 7: Verificación geométrica

7.4.2. Detección de Agujeros

```
1  def has_holes(self, contour):
2      """
3      Detecta si el contorno tiene agujeros internos
4      """
5      # Crear mascara del contorno
6      mask = np.zeros((480, 640), dtype=np.uint8)
7      cv2.drawContours(mask, [contour], -1, 255, -1)
8
9      # Rellenar desde el exterior
10     filled = mask.copy()
11     h, w = filled.shape
12     flood_mask = np.zeros((h+2, w+2), dtype=np.uint8)
13     cv2.floodFill(filled, flood_mask, (0, 0), 255)
14
15     # Invertir y buscar regiones internas
16     filled_inv = cv2.bitwise_not(filled)
```

```
    holes = cv2.bitwise_and(mask, filled_inv)

    # Threshold para ruido
    return np.sum(holes) > 100
```

Listing 8: Detección de defectos internos

Algoritmo de Detección de Agujeros

1. Crear máscara binaria del contorno (relleno)
2. Aplicar flood-fill desde exterior
3. Invertir resultado
4. Intersección con máscara original = agujeros

Este método detecta cualquier región desconectada dentro del contorno.

7.4.3. Verificación de Dimensiones

```
def check_dimensions(self, dimensions, shape_name):
    """
    Verifica si las dimensiones estan en especificacion
    """
    if dimensions is None:
        return False, ["No se pudo medir"]

    defects = []

    if shape_name == 'circulo':
        # Calcular diametro promedio
        diameter = (dimensions['width_mm'] +
                    dimensions['height_mm']) / 2

        min_d = self.specs['circulo']['min_diameter_mm']
        max_d = self.specs['circulo']['max_diameter_mm']

        if diameter < min_d:
            defects.append(
                f"Muy pequeno: {diameter:.1f}mm < {min_d}mm"
            )
        elif diameter > max_d:
            defects.append(
                f"Muy grande: {diameter:.1f}mm > {max_d}mm"
            )

    elif shape_name == 'cuadrado':
        side = (dimensions['width_mm'] +
                dimensions['height_mm']) / 2

        min_s = self.specs['cuadrado']['min_side_mm']
        max_s = self.specs['cuadrado']['max_side_mm']

```

```
        if side < min_s:
            defects.append(
                f"Muy pequeno: {side:.1f}mm < {min_s}mm"
            )
        elif side > max_s:
            defects.append(
                f"Muy grande: {side:.1f}mm > {max_s}mm"
            )

    passed = len(defects) == 0
    return passed, defects
```

Listing 9: Verificación dimensional

7.5. Generación de Reportes

```
def print_final_report(self):
    """
    Imprime reporte final de inspeccion
    """
    print("\n" + "="*60)
    print("REPORTE FINAL DE INSPECCION")
    print("="*60)
    print(f"Total inspeccionado: {self.inspected_count}")
    print(f"Aprobados: {self.passed_count}")
    print(f"Rechazados: {self.failed_count}")

    if self.inspected_count > 0:
        yield_rate = (self.passed_count /
                      self.inspected_count) * 100
        print(f"Yield Rate: {yield_rate:.2f}%")

        if self.defect_types:
            print("\nDefectos mas comunes:")
            from collections import Counter
            defect_counts = Counter(self.defect_types)

            for defect, count in defect_counts.most_common(5):
                print(f"  - {defect}: {count} veces")
```

Listing 10: Reporte final de inspección

7.6. Métricas de Desempeño

7.6.1. Yield Rate

El *yield rate* es la métrica principal de calidad:

$$\text{Yield Rate} = \frac{N_{\text{aprobados}}}{N_{\text{total}}} \times 100\,\% \tag{5}$$

Interpretación:

- Yield $> 95\,\%$: Proceso excelente

- Yield $85 - 95\,\%$: Proceso bueno
- Yield $< 85\,\%$: Proceso requiere mejoras

7.6.2. Análisis de Pareto

Identificar los defectos más frecuentes usando principio 80/20:

```python
from collections import Counter
import matplotlib.pyplot as plt

def pareto_analysis(defect_types):
    """
    Genera grafica de Pareto de defectos
    """
    # Contar defectos
    counter = Counter(defect_types)

    # Ordenar por frecuencia
    defects = [d for d, _ in counter.most_common()]
    counts = [c for _, c in counter.most_common()]

    # Calcular porcentaje acumulado
    total = sum(counts)
    cumulative = np.cumsum(counts) / total * 100

    # Graficar
    fig, ax1 = plt.subplots(figsize=(10, 6))

    # Barras
    ax1.bar(defects, counts, color='steelblue')
    ax1.set_xlabel('Tipo de Defecto')
    ax1.set_ylabel('Frecuencia', color='steelblue')
    ax1.tick_params(axis='y', labelcolor='steelblue')
    plt.xticks(rotation=45, ha='right')

    # Linea acumulada
    ax2 = ax1.twinx()
    ax2.plot(defects, cumulative, color='red',
             marker='o', linewidth=2)
    ax2.set_ylabel('% Acumulado', color='red')
    ax2.tick_params(axis='y', labelcolor='red')
    ax2.axhline(y=80, color='gray', linestyle='--')

    plt.title('Analisis de Pareto - Defectos')
    plt.tight_layout()
    plt.show()
```

Listing 11: Análisis de Pareto de defectos

7.7. Código Completo

El código completo del inspector está disponible en: `quality_inspector.py`

7.8. Ejercicio Práctico

1. Ejecutar el inspector con objetos de prueba
2. Intentar alcanzar Yield Rate $> 90\,\%$
3. Generar reporte de Pareto
4. Identificar defecto más común
5. Proponer mejora en el proceso de fabricación

8. Comparación: RealSense vs Webcam

8.1. Ventajas de RealSense

Aspecto	RealSense D435i	Webcam
Segmentación	Robusta por profundidad (independiente de iluminación)	Depende de contraste/iluminación
Medición	Dimensiones reales en mm	Solo píxeles (relativa)
Fondo variable	Fácil eliminar por distancia	Requiere background subtraction
Información 3D	Completa (X, Y, Z)	Solo 2D (X, Y)
Costo	Alto ($200+)	Bajo ($10-50)
Portabilidad	Requiere USB 3.0	Universal

Cuadro 3: Comparación de características

8.2. Cuándo Usar Cada Una

Usar RealSense cuando:

- Se necesitan mediciones métricas precisas
- El fondo es complejo o variable
- Se requiere información 3D
- La iluminación no es controlable

Usar Webcam cuando:

- Solo se necesita clasificación cualitativa
- El fondo es uniforme y controlado
- La iluminación es constante
- El costo es limitante

8.3. Adaptación del Código

Todos los ejercicios funcionan con ambas cámaras:

```
# Cambiar facilmente entre camaras
use_realsense = True  # False para webcam

# El resto del codigo es identico
classifier = ShapeClassifier(use_realsense=use_realsense)
classifier.run()
```

Listing 12: Código adaptable

La única diferencia:

- Con RealSense: mediciones en metros
- Con webcam: mediciones en píxeles

9. Optimización y Mejores Prácticas

9.1. Optimización de Velocidad

9.1.1. Profiling de Código

Identificar cuellos de botella:

```python
import time

def timed_function(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"{func.__name__}: {(end-start)*1000:.2f}ms")
        return result
    return wrapper

# Uso
@timed_function
def preprocess_frame(self, image):
    # ... codigo ...
    pass
```

Listing 13: Profiling básico

9.1.2. Técnicas de Optimización

1. **Reducir resolución**:

```python
# Procesar a menor resolucion
small = cv2.resize(frame, None, fx=0.5, fy=0.5)
# Procesar...
# Escalar resultados de vuelta si necesario

```

2. **ROI (Region of Interest)**:

```python
# Procesar solo region central
h, w = frame.shape[:2]
roi = frame[h//4:3*h//4, w//4:3*w//4]

```

3. **Saltar frames**:

```python
# Procesar cada N frames
if frame_count % 3 == 0:
    # Procesamiento pesado
    pass
else:
    # Solo visualizacion
    pass

```

4. **Vectorización con NumPy**:

```python
# Mal (loop Python)
for i in range(len(array)):
    array[i] = array[i] * 2

# Bien (vectorizado)
array = array * 2

```

9.2. Manejo de Errores

```python
def get_frame(self):
    """
    Obtiene frame con manejo de errores
    """
    try:
        if self.use_realsense:
            frames = self.pipeline.wait_for_frames(timeout_ms=5000)

            depth_frame = frames.get_depth_frame()
            color_frame = frames.get_color_frame()

            if not depth_frame or not color_frame:
                raise RuntimeError("Frame invalido")

            # ... procesar ...

        else:
            ret, frame = self.cap.read()
            if not ret:
                raise RuntimeError("No se pudo leer de webcam")

            # ... procesar ...

    except Exception as e:
        print(f"Error capturando frame: {e}")
        return None, None, None

    return color_image, depth_image, depth_frame
```

Listing 14: Manejo robusto de errores

9.3. Logging

```python
import logging

# Configurar logger
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('vision_system.log'),
        logging.StreamHandler()
    ]
```

```
)

logger = logging.getLogger(__name__)

# Uso
logger.info("Sistema iniciado")
logger.warning(f"FPS bajo: {fps}")
logger.error(f"Error en clasificacion: {error}")
```

Listing 15: Sistema de logging

10. Preguntas de Reflexión

1. **Filtrado**: ¿Por qué el filtro bilateral preserva mejor los bordes que el Gaussiano? Explique el mecanismo matemático.
2. **Morfología**: ¿En qué orden se deben aplicar las operaciones morfológicas (opening vs closing) para eliminar ruido sal-y-pimienta? Justifique.
3. **Aproximación poligonal**: ¿Cómo afecta el parámetro ϵ en Douglas-Peucker a la clasificación de formas? ¿Hay un valor óptimo?
4. **Circularidad vs Compacidad**: ¿Cuál métrica es más robusta ante ruido en el contorno? ¿Por qué?
5. **Profundidad**: ¿Cómo varía la precisión de medición con la distancia en RealSense? ¿Existe una distancia óptima?
6. **Tiempo real**: ¿Cuál es el trade-off entre precisión y velocidad? ¿Cómo priorizarías según la aplicación?
7. **Iluminación**: ¿Cómo se podría hacer el sistema robusto a cambios de iluminación sin usar profundidad?
8. **Escalabilidad**: ¿Cómo modificarías el sistema para inspeccionar 100 objetos/minuto?

11. Conclusiones

Este laboratorio ha presentado un enfoque integral al procesamiento de imágenes en tiempo real, integrando múltiples técnicas fundamentales:

- **Filtrado espacial** para reducción de ruido
- **Detección de bordes** para localización de objetos
- **Morfología matemática** para limpieza y refinamiento
- **Análisis de contornos** para extracción de características
- **Clasificación geométrica** basada en propiedades invariantes

- **Medición 3D** usando información de profundidad

Los sistemas desarrollados (clasificador de formas e inspector de calidad) demuestran cómo estas técnicas se combinan para resolver problemas reales de visión artificial industrial.

11.1. Aplicaciones en la Industria

Las técnicas presentadas son directamente aplicables a:

- Control de calidad automatizado
- Sistemas de sorting y clasificación
- Inspección de manufactura
- Visión robótica
- Automatización industrial
- Metrología óptica

A. Instalación de Intel RealSense SDK

A.1. Windows

1. Descargar desde: `https://github.com/IntelRealSense/librealsense/releases`
2. Buscar: `Intel.RealSense.SDK-WIN10-[version].exe`
3. Ejecutar instalador con permisos de administrador
4. Conectar cámara RealSense
5. Verificar con Intel RealSense Viewer

A.2. Linux (Ubuntu/Debian)

```
# Agregar repositorio
sudo apt-key adv --keyserver keys.gnupg.net \
    --recv-key F6E65AC044F831AC80A06380C8B3A55A6F3EFCDE

sudo add-apt-repository \
    "deb https://librealsense.intel.com/Debian/apt-repo \
    $(lsb_release -cs) main"

# Instalar
sudo apt-get update
sudo apt-get install librealsense2-dkms
sudo apt-get install librealsense2-utils
sudo apt-get install librealsense2-dev

# Verificar
realsense-viewer
```

B. Códigos Completos

Los códigos completos están disponibles en el repositorio:
`https://github.com/DrOscarL/Artificial-Vision-Labs/`
Archivos incluidos:

- `shape_classifier.py` - Clasificador de formas
- `quality_inspector.py` - Inspector de calidad

C. Solución de Problemas Comunes

C.1. Error: “No device connected”

- Verificar conexión USB 3.0
- Reinstalar drivers de RealSense
- Probar en otro puerto USB

C.2. Error: "bilateralFilter unsupported format"

- Convertir imagen a uint8 o float32
- Usar: `depth.astype(np.float32)`

C.3. FPS muy bajo (<15)

- Reducir resolución de captura
- Optimizar código (ver Sección 8.1)
- Verificar uso de CPU/GPU