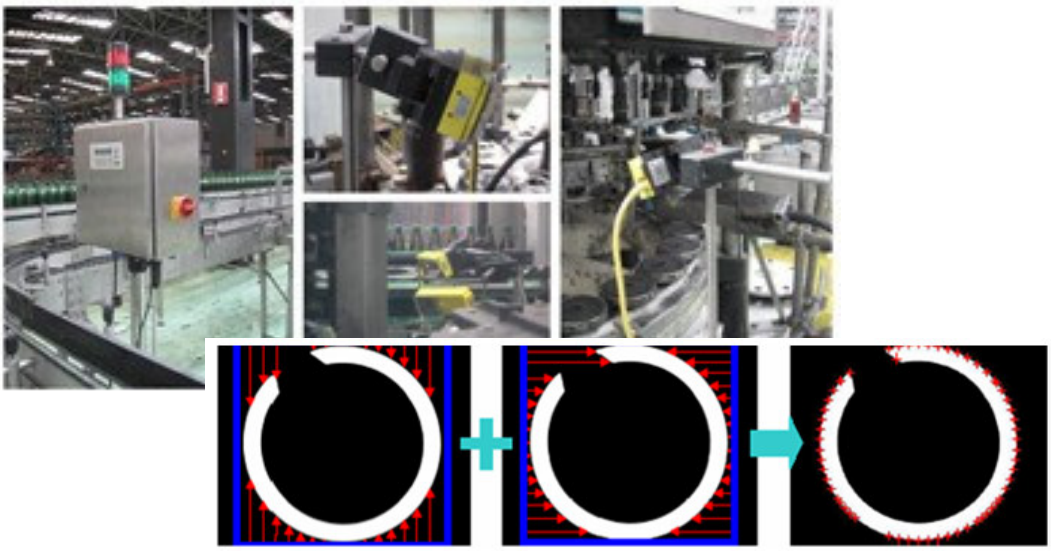
Práctica libre:

Control de calidad de piezas mecánicas mediante visión artificial.



Sistemas de Percepción

Tabla de contenidos

1. Introducción.

Es una realidad, en el entorno industrial, los métodos de estampado y formación de hojas metálicas para la fabricación de piezas mecánicas de pequeño tamaño son uno de los principales factores que limitan la precisión y la productividad alcanzables. Estos métodos son bastante inestables, un pequeño cambio de los parámetros de producción o de la calidad del material por una parte, y un desgaste excesivo o la fractura de la herramienta por otra parte, pueden conducir a la fabricación de piezas mecánicas no conformes. Este tipo de problemas es especialmente difícil de detectar cuando se presentan pequeñas piezas mecánicas, a menudo fabricadas en gran cantidad y a muy alto ritmo. Las tecnologías de producción en el actual entorno económico globalizado y, por tanto, fuertemente competitivo, se basa en la producción de piezas mecánicas no conformes que sigue hasta que un control por muestreo revela el problema y permita decidir la producción [Pro2Control 07].

Este documento corresponde a la práctica libre de la asignatura que tiene como título **Control de calidad de piezas mecánicas mediante visión artificial**.

Este documento se estructura de la siguiente forma:

- En el capítulo 2, se introduce al lector en los objetivos del trabajo a realizar. En el capítulo 3, se presenta una descripción del trabajo desarrollado. El capítulo 4, muestra los resultados obtenidos para una serie de piezas mecánicas obtenidas a la salida de la cadena de producción. El capítulo 5, realiza una recopilación de la bibliografía utilizada. Todos los enlaces URL están disponibles a fecha de Junio de 2008. Por último, se ha dedicado un anexo para justificar así la elección de las herramientas utilizadas (Matlab).

2. Objetivos del trabajo.

Dentro de este ámbito, se pretende abordar la aplicación de un control por visión artificial, y de un sistema flexible de tratamiento de imágenes para la detección instantánea de tres piezas mecánicas. Así, el objetivo final sería desarrollar un sistema de control de producción que se pudiese integrar sobre una herramienta de fabricación de las piezas mecánicas, para su posible utilidad práctica. Las referencias de estas tres piezas mecánicas a analizar mediante este sistema de visión artificial son: IA04, 0863-012, y 5828-001; que se han tomado como imágenes sobre un fondo claro con el fin de diferenciar bien la propia pieza mecánica del fondo[1]. Del mismo modo, en la salida de la cadena de producción estas piezas mecánicas deberán satisfacer una serie de tolerancias:

La referencia IA04 de la pieza mecánica –véase la figura 1– corresponde a un anillo de retención más comúnmente llamado *circlips* en el entorno industrial. Estos anillos son componentes de ensamblado mecánico que se usa generalmente para realizar paradas axiales. La pieza mecánica se puede colocar en un canal dentro de un tubo o de un exterior un eje. Las tolerancias exigidas para esta pieza son:

- A. Radio del contorno interior, entre 21,56 y 22,11 milímetros.
- B. Ranura, entre 4 y 6,5 milímetros.
- C. Valor medio del contorno interior/exterior, entre 2,25 y 2,5 milímetros.

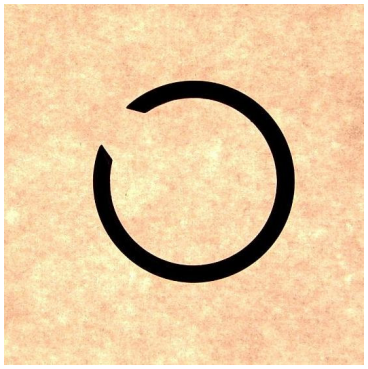


figura 1. Pieza mecánica de referencia IA04.

La referencia 0863-012 tiene una forma externa circular –véase la figura 2–. El aspecto interior es más complejo con cuatro porciones salientes en el contorno interno. Además, contiene dos agujeros. Las tolerancias exigidas para esta pieza son:

[1] La resolución espacial de 550x550 píxeles de cada imagen resulta razonable en esta aplicación. La precisión que ofrece tal tamaño es satisfactoria para las vistas tomadas de cada pieza mecánica.

- A. Eje de simetría, entre 3,25 y 3,55 milímetros.
- B. Radio del contorno exterior, entre 37.55 y 38.30 milímetros.
- C1. Ranura, borde izquierdo, mínimo 5,45 milímetros.
- C2. Ranura, borde derecho, mínimo 5,45 milímetros.
- D1. Agujero, izquierdo, mínimo 2,55 milímetros.
- D2. Agujero, derecho, mínimo 2,55 milímetros.

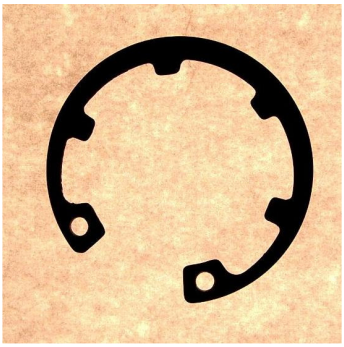


figura 2. Pieza mecánica de referencia 0863-012.

Finalmente, la referencia 5828-001 es similar a 0863-012 –véase la figura 3–. La diferencia principal es que en vez de cuatro porciones salientes, tiene tres porciones salientes en el contorno interno. Las tolerancias exigidas para esta pieza son:

- A. Radio del contorno exterior, entre 33.75 y 34.50 milímetros.
- B. Radio del contorno interior, entre 21.60 y 22 milímetros.
- C1. Agujero, izquierdo, mínimo 2,55 milímetros.
- C2. Agujero, derecho, mínimo 2,55 milímetros.

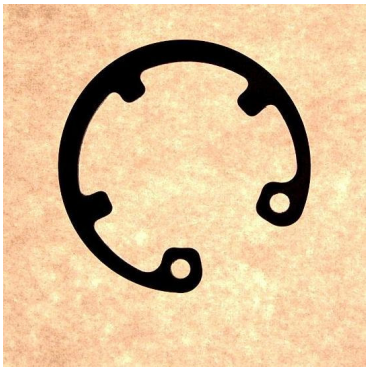


figura 3. Pieza mecánica de referencia 5828-001.

Como hemos dicho anteriormente, el alto ritmo de producción puede provocar imperfecciones añadidas: residuos metálicos de la prensa adheridos a las piezas mecánicas, rebabas, defectos de circularidad, etc. Con el fin de medir el

grosor de cada una de las piezas mecánicas, una nueva cámara sería necesaria para tomar las vistas de lado.

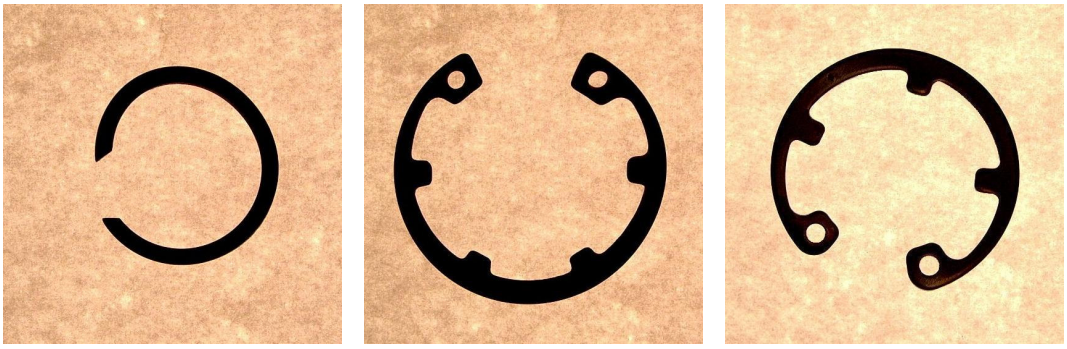


figura 4, 5 y 6. Imperfecciones de las piezas mecánicas.

3. Trabajo desarrollado.

Es de sabido en la literatura que la mayoría del procesamiento de una imagen se puede realizar en *software*. De hecho, el aumento de la velocidad de computación de los PCs actuales y el abaratamiento de la memoria para los mismos, hace posible la utilización de este tipo de sistemas para la implementación de sistemas de visión artificial. Una de las configuraciones típicas, es utilizar una tarjeta de digitalización de vídeo instalada en el bus de expansión del PC y una cámara que se conecta a la misma. En este tipo de configuraciones, la tarjeta digitalizadora suele tener memoria suficiente para almacenar varios *frames* y en algunos casos tiene un procesador específico para el tratamiento de la imagen. En otros casos es el microprocesador propio del PC el que se encarga de realizar el procesamiento de la imagen.

Atendiendo a esto, el resto de este capítulo se estructura de la siguiente forma:

- La sección 3.1., surge con el propósito de buscar en primera instancia las características de las piezas mecánicas: básicas, comunes entre las tres; específicas, que reúne aquellas particularidades que las diferencia. La sección 3.2., atiende al algoritmo resultante para tratar las piezas mecánicas según la información recopilada en la sección anterior. La sección 3.3., responde a la implementación de los métodos (funciones) realizados para el entorno de Matlab.

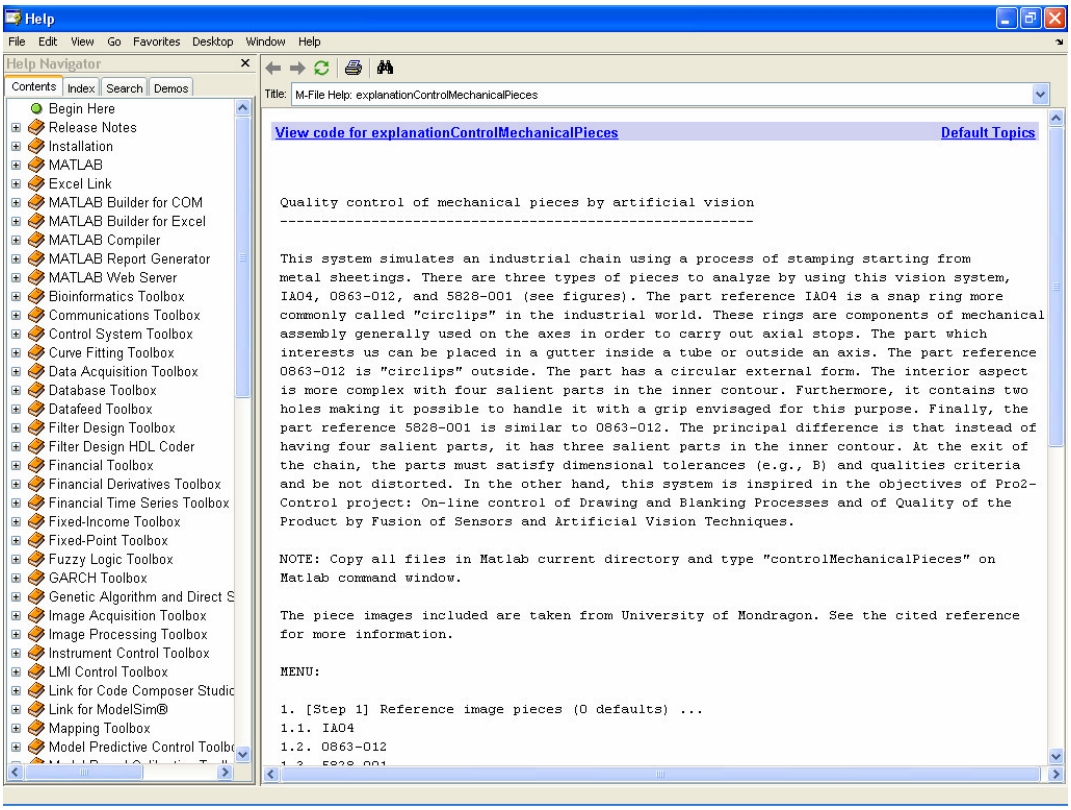


figura 7. `explanationControlMechanicalPieces.m`.

3.1. Búsqueda de las características de las piezas mecánicas.

Cada método se ha sometido a varios criterios antes de utilizarse. Un criterio es la precisión que el método debe alcanzar. En efecto, algunas tolerancias exigidas son muy reducidas. Es necesario encontrar métodos que nos permiten dilucidar su validez. Además, cuanto más precisión se quiera, el tiempo de cálculo aumentará y viceversa. El otro criterio es contemplar un tratamiento genérico. Esto es, resulta de interés encontrar métodos reutilizables y en consecuencia evitar los tratamientos específicos. Un ejemplo es la función `processGenericPiece` implementada para Matlab –véase la figura 8– que se aplica a los tres modelos de piezas mecánicas.

3.1.1. Básicas.

3.1.1.1. El pretratamiento.

En primer lugar, no podemos utilizar directamente la imagen de entrada (presencia de ruido, etc.). Un pretratamiento se requiere. Por otro lado, Matlab dispone de funciones que detectan todos los contornos cerrados en una imagen binaria, como es `regionprops`, que se utiliza conjuntamente con la función `edge` y `bwboundaries`.

```
function structProcessGenericPiece=processGenericPiece(pImagePiece)

    tic;

    structProcessGenericPiece={};

    %grayscale image

    imagePieceGrayscale=rgb2gray(pImagePiece);

    %binarisation

    imagePieceBinary=imagePieceGrayscale<120;
    imagePieceBW=bwareaopen(imagePieceBinary,50);
    dimension=size(imagePieceBinary);
    [imagePieceBW, thresh]=edge(imagePieceBinary, 'sobel');

    clear imagePiece imagePieceGrayscale imagePieceBinary;

    %determine borders and to show the different detected objects.

    [imagePieceBWBoundaries,L]=bwboundaries(imagePieceBW,'noholes');
    clear imagePieceBWBoundaries imagePieceBW;

    %

    statsObjects=regionprops(L, 'Area', 'Centroid', 'EquivDiameter', 'PixelList');
    imagePieceBWOneObject=ismember(L,1);
```

figura 8. Extracto de código Matlab: *processGenericPiece.m*.

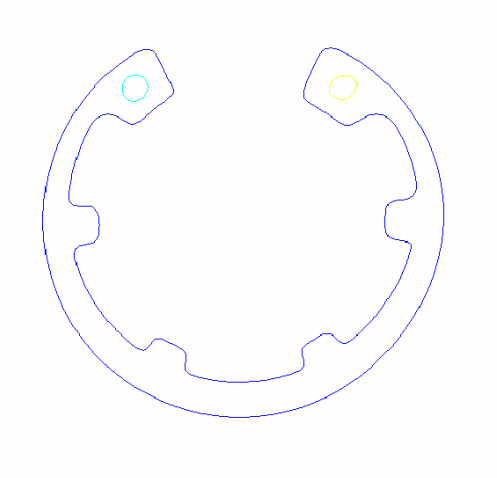


figura 9. Contornos cerrados de una pieza mecánica 0863-012
(defectuosa: circularidad agujero 2).

Esto permite caracterizar la región correspondiente a los agujeros de las piezas mecánicas 0863-012 y 5828-001.

3.1.1.2. El perfil de la pieza mecánica.

Las tres piezas mecánicas tienen formas circulares. Con el fin de tener la mayor precisión posible manteniendo al mismo tiempo una rapidez de cálculo correcta, optamos por la detección del contorno exterior partiendo de los lados de la imagen (barrido horizontal y vertical). Este método da resultados satisfactorios (*detectExternalContourImageBinary.m*). Los puntos resultantes se utilizan en el modelo de los menores cuadrados para encontrar el círculo que modela el contorno exterior. Este círculo sirve para encontrar el centro de la pieza mecánica (*obtainParametersCircle.m*).

El método del rayo giratorio (*revolvingRay.m*) que se ha seleccionado utiliza un segmento cuyo origen está en el centro de la pieza mecánica. Del mismo modo, se cambia la dirección sobre 360º en torno a la pieza mecánica con el fin de generar un perfil (*profile*) de ésta, como se muestra en la siguiente figura 11. Además, el perfil contiene toda la información necesaria para los cálculos de todas las medidas: grosores de la pieza mecánica, posición de los contornos exterior e interior, etc.

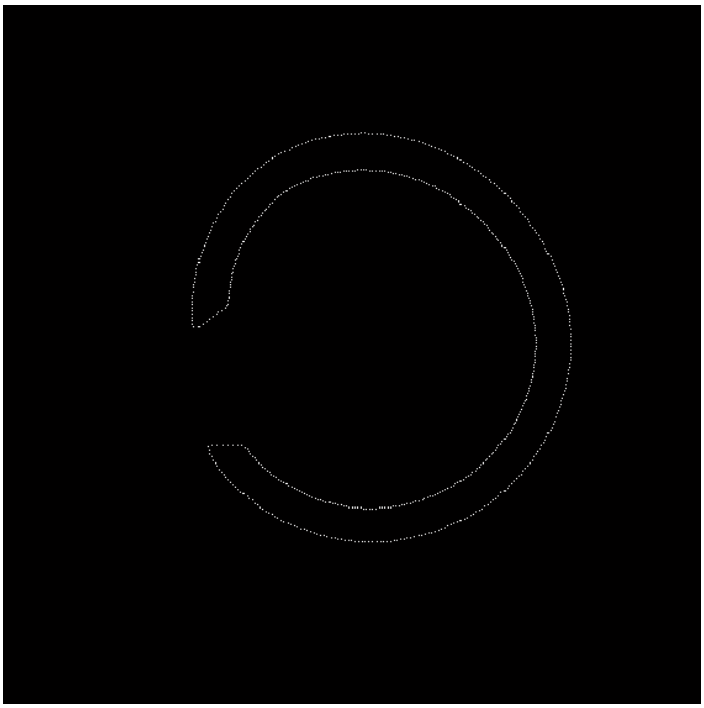


figura 10. Posición de los contornos exterior e interior de una pieza mecánica IA04 (defectuosa: circularidad).

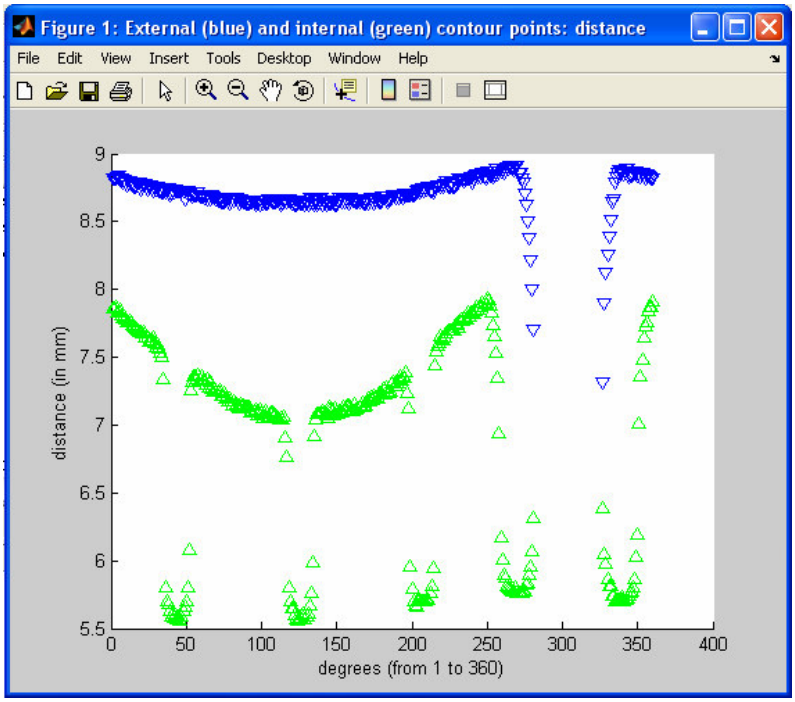


figura 11. Perfil de una pieza mecánica 0863-012.

Un inconveniente principal subsiste. Para que el método funcione, es necesario determinar previamente el centro de la pieza mecánica. Mayormente, es suficiente con utilizar la información contenida en el perfil para determinar la mayoría de las tolerancias y detectar las posibles deformaciones (rebabas, etc.). Los puntos del contorno interior permiten modelar el círculo interior de la parte y determinar su diámetro. Los grosores recuperados sobre el perímetro de la pieza mecánica permiten definir el grosor medio de la pieza y detectar rebabas. La parte del perfil correspondiente a la ranura, permite calcular el tamaño (ángulo) de apertura (*extractAngleContour.m*).

3.1.2. Específicas.

El análisis del perfil determina todas las especificaciones de la pieza mecánica IA04, pero no para las piezas mecánicas 0863-012 y 5828-001. Estas piezas mecánicas disponen de dos agujeros en sus extremos cuyo diámetro es necesario medir. Estos agujeros no aparecen en el perfil de la pieza mecánica sino que se detectan en la fase de pretratamiento –sección 3.1.1.1–.

Por ejemplo, la tolerancia A de la pieza mecánica 0863-012, viene dada por su eje de simetría. Para la medición de esta tolerancia, se obtienen los ángulos que conforman las coordenadas de los puntos centroides de las regiones asociadas a los dos agujeros, con respecto al eje de abscisas (*plannerDegrees.m*). Se calcula la media de estos dos valores y, posteriormente, se determina su ángulo opuesto. Como los puntos del contorno

exterior e interior se almacenan (su índice) por este ángulo, la tolerancia, que viene dada por su distancia Euclídea, resulta trivial –véase la figura 12–.

```
%verify tolerances

infoPointsBWOneObjectExternalContour=structProcessGenericPiece.InfoPointsExternalContour;
infoPointsBWOneObjectInternalContour=structProcessGenericPiece.InfoPointsInternalContour;
infoAngleSlot=structProcessGenericPiece.InfoAngleSlot;
maxDistancePointBWOneObjectExternalContourByAngle=structProcessGenericPiece.MaxDistancePointExterns
statsObjects=structProcessGenericPiece.StatsObjects;

if (length(statsObjects)~=3)
    warndlg('Piece image does not have 2 holes.', ' Warning ');
    return;
end

%A (axis of symmetry of the part)

x0=structProcessGenericPiece.CenterCoordinates(1);
y0=structProcessGenericPiece.CenterCoordinates(2);

centroidTwoObject=statsObjects(2).Centroid;
centroidThreeObject=statsObjects(3).Centroid;

angleCentroidTwoObject=plannerDegrees([x0 y0], [centroidTwoObject(2) centroidTwoObject(1)]);
angleCentroidThreeObject=plannerDegrees([x0 y0], [centroidThreeObject(2) centroidThreeObject(1)]);

angleSlotMean=mean([angleCentroidTwoObject angleCentroidThreeObject]);
angleSlotMeanComplementary=round(mod(angleSlotMean+180,360));

distanceAngleSlotMeanComplementaryExternalContour=infoPointsBWOneObjectExternalContour(angleSlotMe
distanceAngleSlotMeanComplementaryInternalContour=infoPointsBWOneObjectInternalContour(angleSlotMe
distanceAngleSlotMeanComplementaryPartWidth=distanceAngleSlotMeanComplementaryExternalContour-dist
```

figura 12. Extracto de código Matlab: `demandedTolerances0863012.m`.

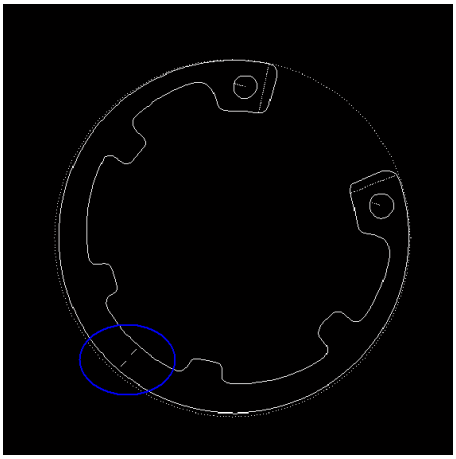


figura 13. Eje de simetría de pieza mecánica 0863-012.

3.2. El algoritmo para tratar cada pieza mecánica.

La etapa siguiente consiste en describir todos los pasos juntos. La figura 14 describe el desarrollo del tratamiento para una pieza mecánica.

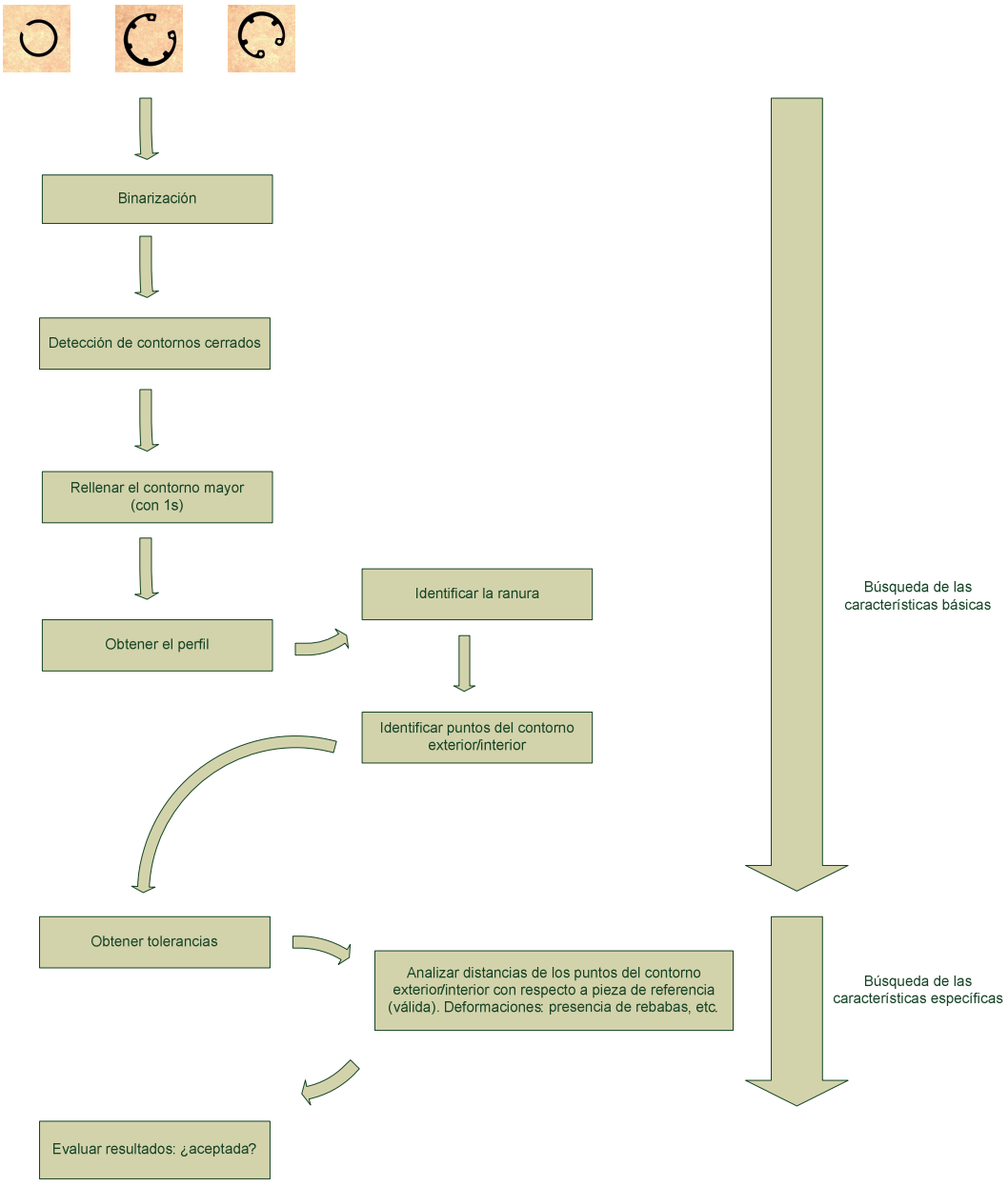


figura 14. El algoritmo para tratar cada pieza mecánica.

3.3. Implementación en Matlab.

Antes de empezar la implementación del código, se definió una estructura de ficheros .m, propios de Matlab que resultasen los más legibles posibles y reutilizables:

`\tutorials`:
Vídeos .avi para una comprensión rápida del uso de la aplicación.

`\source code\images`:
imágenes (estáticas) de piezas mecánicas.

`\source code\video`:
imágenes en movimiento de piezas mecánicas (*frames*).

`\source code`
ficheros .m

Las principales funciones se detallan a continuación:

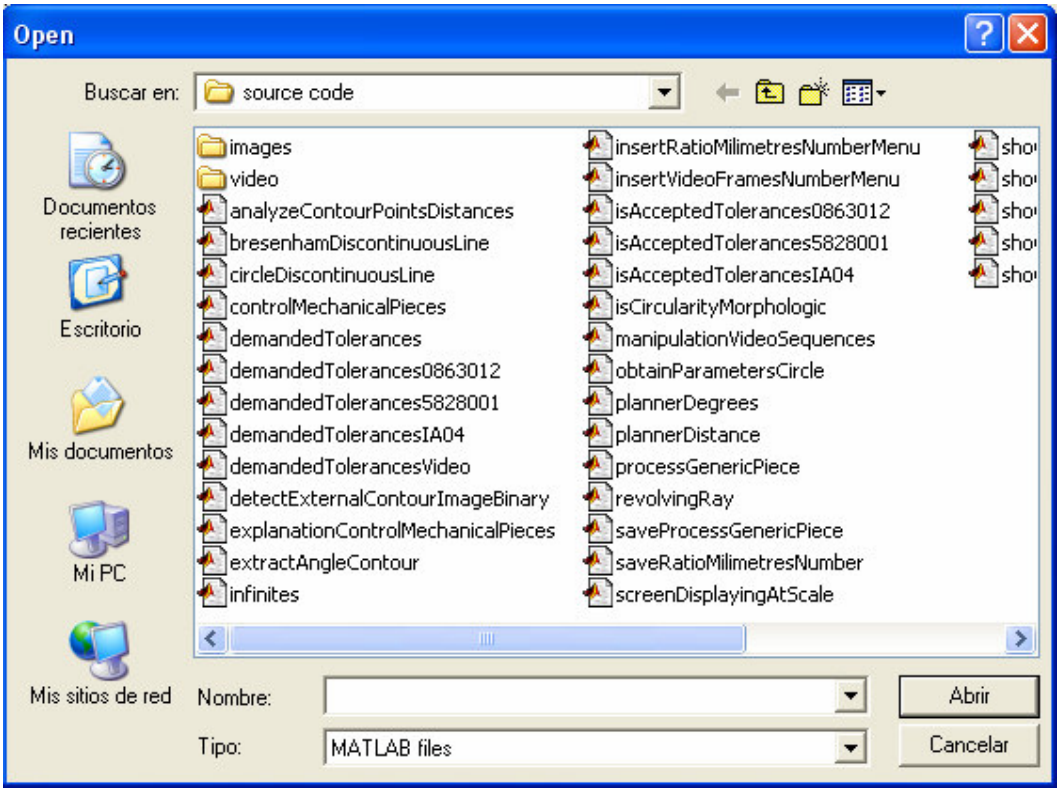


figura 15. Organigrama de ficheros .m.

3.3.1. processGenericPiece.m.

```
structProcessGenericPiece
=processGenericPiece
(pImagePiece)
```

Realiza un pretratamiento de la imagen asociada a una pieza mecánica cualquiera, de entre las tres. El argumento de entrada `pImagePiece` corresponde a la imagen RGB leída. La función `showGenericPiece` se asocia a ésta, que muestra el resultado por pantalla, bien a través de la línea de comandos o mediante figuras representando así las características obtenidas. Este resultado es una estructura de datos como la siguiente figura 16.

Field	Value
DimensionDescription	'Dimension of the piece'
Dimension	[550 550]
ImageDescription	'Image piece'
Image	*<550x550x3 uint8>*
CenterDescription	'Center point'
CenterCoordinates	[269 277]
InfoPointsContourDescription	'Information about contour points (external and internal) of the piece'
InfoPointsExternalContour	*<360x3 double>*
InfoPointsInternalContour	*<360x3 double>*
InfoAngleSlotDescription	'Slot (start and end angle)'
InfoAngleSlot	[231 278]
MaxDistancePointExternalContourByAngleDescription	'Circle representing the external contour'
MaxDistancePointExternalContourByAngle	288
MinDistancePointInternalContourByAngleDescription	'Least squares circle of the internal contour'
MinDistancePointInternalContourByAngle	75
StatsDescription	'Information about the detected objects in the piece'
StatsObjects	*<3x1 struct>*
StatsImages	*<550x550 double>*

figura 16. Estructura de datos: `structProcessGenericPiece`.

3.3.2. detectExternalContourImageBinary.m

```
infoPointsBinaryExternalContour=
detectExternalContourImageBinary
(pImageBinary)
```

Realiza la detección del contorno exterior partiendo de los lados de la imagen (barrido horizontal y vertical). Los puntos resultantes se utilizan en el modelo de los menores cuadrados para encontrar el círculo que modela el contorno exterior.

3.3.3. obtainParametersCircle.m

```
[x0, y0]=
```

```
obtainParametersCircle
(points2D, showDetailsFlag)
```

El argumento de entrada `points2D` contiene puntos que modelan un contorno exterior. Este círculo (contorno) sirve para encontrar el centro de la pieza mecánica (`obtainParametersCircle.m`). Éste viene dado a través de las coordenadas: `x0` e `y0`.

3.3.4. demandedTolerances.m

```
[structTolerancesObtained, structProcessGenericPiece]
=demandedTolerances
(pNamePieceReference, pStructProcessGenericPieceReference)
```

Permite buscar un fichero de nombre que empiece por `pNamePieceReference` y tenga de extensión `jpg`. Posteriormente, realiza un pretratamiento de la imagen, obteniéndose por último las tolerancias demandas para esa pieza mecánica. El argumento de entrada `pStructProcessGenericPieceReference` permite comparar los perfiles de esta pieza mecánica con su pieza homóloga válida, diferencia de distancias del contorno exterior e interior. Esto es útil para contemplar la posible existencia de rebabas o protuberancias del metal.

La función `showTolerances`<Número de referencia de la pieza> se asocia a ésta, que muestra el resultado por pantalla, bien a través de la línea de comandos o mediante figuras representando así las tolerancias obtenidas.

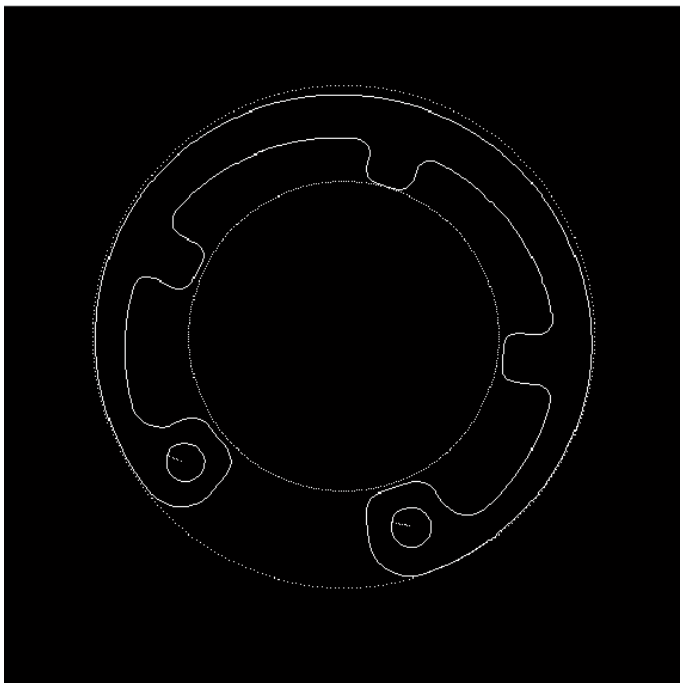


figura 17. Tolerancias obtenidas de una pieza mecánica 5828-001.

El resultado de la función `demandedTolerances` es una estructura de datos como la siguiente figura 18.

Field	Value
NamePieceReference	'5828-001'
AToleranceDescription	'Circle representing the external contour'
AToleranceDiameter	390.25
AToleranceCoordinates	[439 267]
BToleranceDescription	'Compute the least squares circle of the internal contour'
BToleranceDiameter	242.98
BToleranceCoordinates	[142 208]
C1ToleranceDescription	'Hole 1'
C1ToleranceDiameter	30.914
C1ToleranceCentroid	[402.01 269.62]
C2ToleranceDescription	'Hole 2'
C2ToleranceDiameter	30.507
C2ToleranceCentroid	[302.16 419.67]
BurrsDescription	'Analyze contour (external and internal) points distances (to reference piece)'
BurrsDistancePointsExternalContourPieces	*<360x1 double>*
BurrsDistancePointsInternalContourPieces	*<360x1 double>*
PieceTolerancesDescription	'Image piece with its tolerances'
PieceTolerancesImage	*<550x550 double>*

figura 18. Estructura de datos: `structTolerancesObtained`.

3.3.5. analyzeContourPointsDistances.m

```
[distancePointsExternalContourPieces,
distancePointsInternalContourPieces]
=
analyzeContourPointsDistances
(pStructProcessGenericPieceReference,
pStructProcessGenericPieceOne)
```

Analiza las distancias de los puntos del contorno exterior e interior de dos piezas mecánicas. Una de ellas se considera de referencia y válida (`pStructProcessGenericPieceReference`). El algoritmo de esta función busca el eje de simetría de las piezas mecánicas, y recorre ambas desde ese posición hacia la izquierda (hasta el extremo de la ranura), y hacia la derecha (hasta el extremo de la ranura[2]). Se utiliza para contemplar la posible existencia de rebabas o protuberancias del metal. Otra deformación que detecta inherentemente podría corresponder a la falta de un saliente en el contorno interior de una pieza mecánica 0863-012 ó 5828-001.

[2] De este modo, no se rotan las imágenes.

3.3.6. showAnalyzeContourPointsDistances.m

```
showAnalyzeContourPointsDistances
(pStructTolerancesObtained, pMilimetresNumber)
```

El argumento de entrada `pStructTolerancesObtained` contiene el análisis de las distancias de los puntos del contorno exterior e interior de dos piezas mecánicas. Una de ellas se considera de referencia y se considera válida. Éste está en los campos `BurrsDistancePointsExternalContourPieces` y `BurrsDistancePointsInternalContourPieces` –véase la figura anterior 18–. La línea discontinua en rojo representa el umbral (*threshold*) en milímetros. Por ejemplo, 0.8 milímetros. Esta función está relacionada con la anterior, `analyzeContourPointsDistances`.

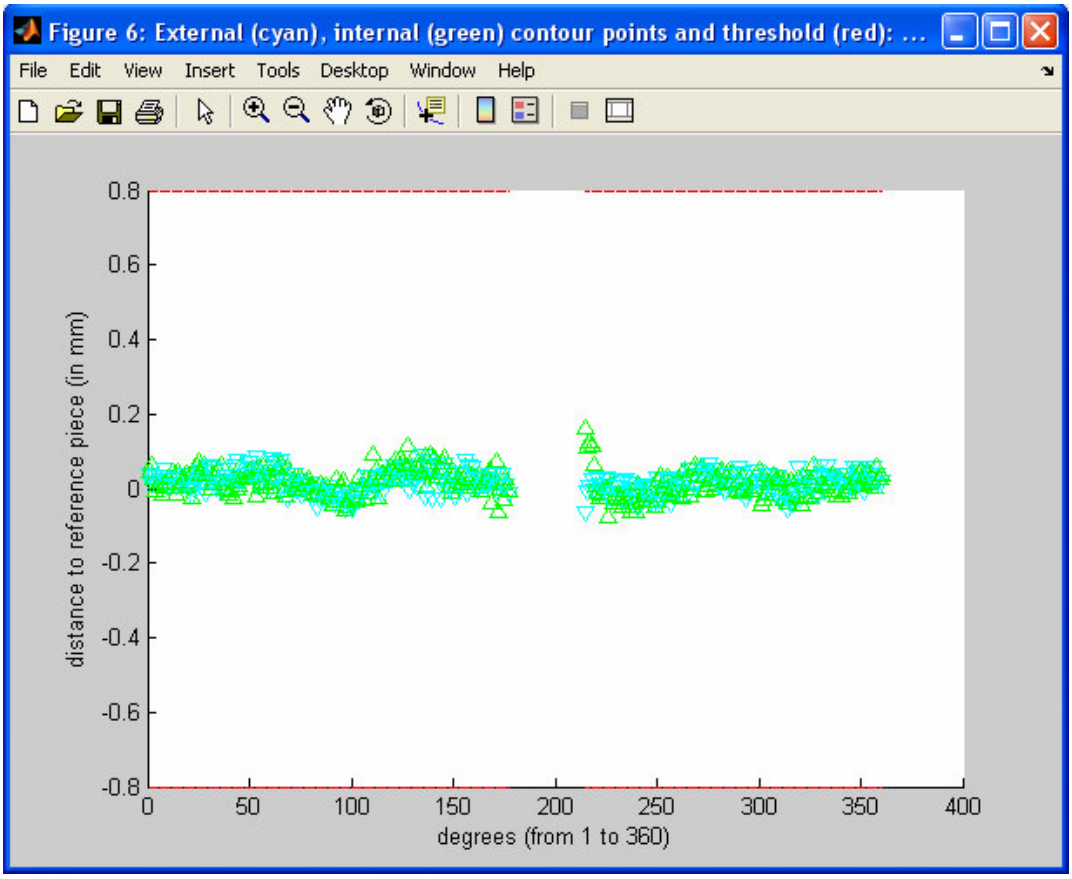


figura 19. Diferencia de distancias del contorno exterior e interior entre dos piezas mecánicas.

3.3.7. extractAngleContour.m

```
infoAngleSlot
=
extractAngleContour
```

```
(infoPointsExternalContour)
```

La parte del perfil correspondiente a la ranura de una pieza mecánica, que permite calcular el tamaño (ángulo) de apertura de ésta (*extractAngleContour.m*), en el intervalo de (0, 360] grados y en sentido antihorario. Por ejemplo, [133 160].

3.3.8. revolvingRay.m

```
[infoPointsExternalContour, infoPointsInternalContour]
=
revolvingRay
(pImageBinary, pCenter)
```

Implementa el método del rayo giratorio. Además, el perfil resultante (`infoPointsExternalContour,infoPointsInternalContour`) contiene buena parte de la información necesaria para los cálculos de las medidas (grosores de la pieza mecánica, posición de los contornos exterior e interior, etc.).

3.3.9. isAcceptedTolerances<Número de referencia de la pieza>.m

```
isAccepted
=
isAcceptedTolerances<Número de referencia de la pieza>
(pStructTolerancesObtained<Número de referencia de pieza>,
pRatioPixelInMilimetres)
```

Devuelve `true` si la pieza mecánica <Número de referencia de la pieza> es válida atendiendo a sus respectivas tolerancias exigidas. Devuelve `false` en caso contrario. Del mismo modo, muestra por pantalla resultados y valores de las tolerancias obtenidas.

3.3.10. isCircularityMorphologic.m

```
[isAccepted, varianceDistanceRegionPoints]
=
isCircularityMorphologic
(pCoordinatesRegionPoints,pRegionCenter,pCircularityValueTh
reshold)
```

Implementa un descriptor para caracterizar la circularidad de la región de una imagen. La circularidad es una medida que expresa la similaritud de un objeto a un círculo. Se puede expresar como la varianza de la distancia de los píxeles (`pCoordinatesRegionPoints)` del borde al centroide (`pRegionCenter`) del objeto. Un objeto es perfectamente circular si la varianza resultante es cero. `pCircularityValueThreshold` establece un umbral. Por ejemplo, 0,9. Si se rebasa, se devuelve `false`, no circular.

3.3.11. screenDisplayingAtScale.m

```
outputImage
=
screenDisplayingAtScale(pImage, pScale)
```

Redimensiona la imagen proporcionada a través del argumento de entrada `pImage`. `pScale` es un vector que define la escala (en el espacio horizontal y vertical). Por ejemplo, si su valor es [1.0 1.0] devuelve la imagen original con las misma dimensiones. Se utiliza para mostrar figuras de piezas mecánicas procesadas acorde con la resolución de la pantalla.

3.3.13. Funciones con el prefijo save.

Guardan las distintas estructuras de datos de la aplicación.

3.3.14. Funciones con el prefijo bresenham.

Implementan el algoritmo bresenham[3] para trazar una línea recta a partir de las coordenadas de dos puntos de una imagen.

3.3.15. controlMechanicalPieces.m

Corresponde al menú de la propia aplicación de control por visión artificial, y de un sistema flexible de tratamiento de imágenes para la detección instantánea de tres piezas mecánicas. De este modo, se facilita la entrada (*input*) de valores. Estos valores serán mayormente imágenes de estas tres piezas mecánicas donde la fuente de origen vendrá dada por: imágenes (estáticas), vídeo (imágenes en movimiento, *frames*) o webcam (posible futura línea de trabajo). O el ratio milímetros/píxel de las imágenes a tratar.

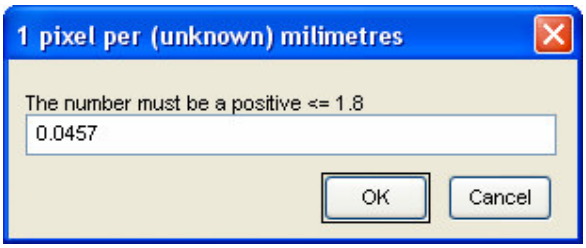


figura 20.Ratio milímetros/píxel.

La siguiente figura muestra el menú principal de la aplicación. El primer paso básico es introducir las imágenes de las piezas de referencia válidas (*[Step 1] Referente image pieces (0 defaults)*). El siguiente paso es capturar las tolerancias de la pieza mecánica (*[Step 2] Capture tolerances …*).

[3] http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

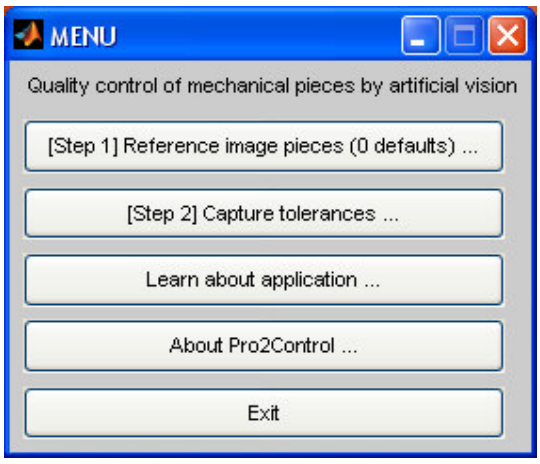


figura 21. Interfaz gráfica (GUI):
Quality control of mechanical pieces by artificial vision.

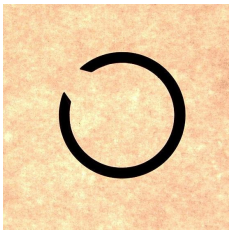
4. Resultados obtenidos.

El trabajo se realizó con vistas frontales de las piezas mecánicas, que se obtendrían a través de una cámara colocada en esa posición. Sin embargo, faltarían aquellas asociadas a la vista lateral. Para ello, se necesitaría una segunda cámara. Los métodos han dado resultados satisfactorios para un total de nueve imágenes frontales[4].

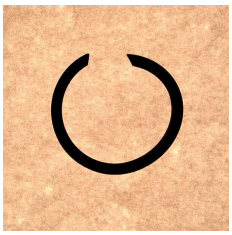
4.1. IA04

El ratio milímetros/píxel de las imágenes a tratar es: 0.0923.

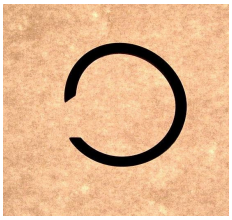
A. Radio del contorno interior: 11.6565 mm.
B. Ranura: 6.6558 mm (desde 133º a 160º).
C. Valor medio del contorno interior/exterior: 2.4114 mm



A. Radio del contorno interior: 10.7817 mm.
B. Ranura: 8.769 mm (desde 80º a 117º).
C. Valor medio del contorno interior/exterior: 2.4883 mm



A. Radio del contorno interior: 10.8349 mm.
B. Ranura: 8.7354 mm (desde 117º a 214º).
C. Valor medio del contorno interior/exterior: 2.4878 mm



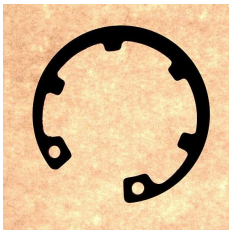
4.2. 0863-012

El ratio milímetros/píxel de las imágenes a tratar es: 0.0873.

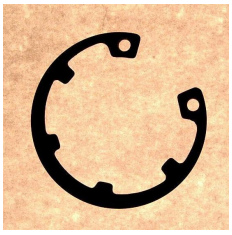
---

[4] Es de interés ahora validarlos con un mayor número de muestras procedentes directamente de la cadena de producción.

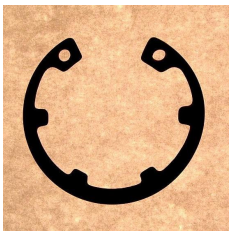
- A. Eje de simetría: 3.324 mm (67º)
- B. Radio del contorno exterior: 38.2928 mm
- C1. Ranura, borde izquierdo: 5.2219 mm
- C2. Ranura, borde derecho: 5.43 mm
- D1. Agujero, izquierdo: 2.4958 mm
- D2. Agujero, derecho: 2.678 mm



- A. Eje de simetría: 3.2194 mm (230º)
- B. Radio del contorno exterior: 37.9786 mm
- C1. Ranura, borde izquierdo: 5.3339 mm
- C2. Ranura, borde derecho: 5.274 mm
- D1. Agujero, izquierdo: 2.4905 mm
- D2. Agujero, derecho: 2.5603 mm



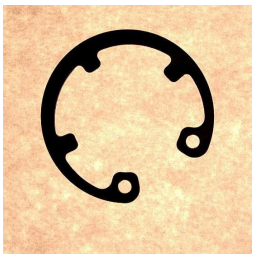
- A. Eje de simetría: 3.4057 mm (272º)
- B. Radio del contorno exterior: 39.1388 mm
- C1. Ranura, borde izquierdo: 5.6232 mm
- C2. Ranura, borde derecho: 5.285 mm
- D1. Agujero, izquierdo: 2.4632 mm (no circular)
- D2. Agujero, derecho: 2.6629 mm



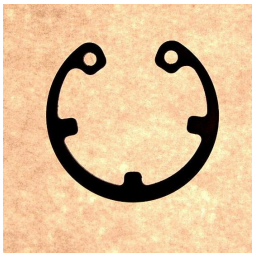
4.3. 5828-001

El ratio milímetros/píxel de las imágenes a tratar es: 0.0892.

- A. Radio del contorno exterior: 34.8104 mm
- B. Radio del contorno interior: 21.6739 mm
- C1. Agujero, izquierdo: 2.7576 mm
- C2. Agujero, derecho: 2.7212 mm

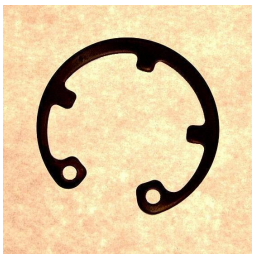


- A. Radio del contorno exterior: 34.883 mm
- B. Radio del contorno interior: 21.5864 mm
- C1. Agujero, izquierdo: 2.6668 mm



- C2. Agujero, derecho: 2.771 mm

- A. Radio del contorno exterior: 36.3975 mm
- B. Radio del contorno interior: 22.4713 mm
- C1. Agujero, izquierdo: 2.8476 mm
- C2. Agujero, derecho: 2.8951 mm



Por último, la prueba de rendimiento ha consistido en calcular el número de piezas mecánicas por segundo que la aplicación puede tratar. Para ello, se utilizó las funciones propias de Matlab, `tic` y `toc`[5]. La cifra está por encima de las expectativas, aunque se sabe que existen PCs de 4 ó 5 veces más potencia en el mercado que el que se ha utilizado para el desarrollo de esta práctica libre. Estos resultados, tiempo de ejecución, se muestran a través de la línea de comandos del entorno de Matlab durante la aplicación. Si bien una prueba de rendimiento más fiable que el tiempo de ejecución es el número de operaciones necesarias (independiente de la máquina PC).

[5] La resolución que devuelven es mayor que el segundo.

5. Bibliografía

Las siguientes referencias bibliográficas se han utilizado:

[Agam 06]	Agam, G. *"Introduction to programming with OpenCV"*. Department of Computer Science, Illinois Institute of Technology (EE.UU.), 2006. Disponible en: http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/index.html

[Izagirre et al. 08]	Izagirre, A., eta Mendikute, M. *"Pertzepzio sistemak"*. Elektronika saila, Goi Eskola Politeknikoa, Mondragon Unibertsitatea. Klase magistralak, 2008.

[McAndrew 04]	Mc Andrew, A. *"An Introduction to Digital Image Processing with Matlab"*. School of Computer Science and Mathematics, Victoria University of Technology (Australia). Disponible en: https://netfiles.uiuc.edu/rmalik2/shared/DigitalImageProcessing-Matlab.pdf

[MATLAB 08]	MATLAB – The Language of Technical Computing. *"MATLAB Help"*. The MathWorks, Inc., 2008.

[Pro2Control 07]	Pro2Control - On-line Control of Drawing and Blanking Processes and of Quality of the Product by Fusion of Sensors and Artificial Vision Techniques, CORDIS FP6. Disponible en: http://www.pro2control.net

[Stavens 08]	Stavens, D. *"Introduction to OpenCV"*. Stanford Artificial Intelligence Lab. Standford University (EE.UU.), 2008. Disponible en: http://ai.stanford.edu/~dstavens/cs223b/stavens_opencv_optical_flow.pdf

[Visual C++ 08]	Visual C++ OpenCV Library Wiki. *"OpenCV with Visual C++ 6.0, 2005 Express, and 2008 Express"*. Sourceforge, 2008. Disponible en: http://opencvlibrary.sourceforge.net

[Visual C++ 08b]	Visual C++ 2008 Express. Microsoft, 2008. Disponible en: http://www.microsoft.com/express/vc

6. Tabla de figuras.

7. Anexo

Con el propósito de elegir la herramienta más conveniente, vamos a implementar un ejemplo básico como es la detección de bordes de una imagen:

- Utilizando librerías de OpenCV llamadas desde Visual C++.
- Utilizando Matlab.

Además, se realizará los procedimientos desde el inicio para valorar el grado de complejidad y el tiempo requerido para realizar la misma aplicación.

7.1. Implementación con librerías OpenCV llamadas desde Visual C++.

7.1.1. Instalación.

A continuación, explicaremos todos los pasos para instalar las librerías de OpenCV sobre Visual C++ [Visual C++ 08].

Si no tenemos instalado las librerías de OpenCV, deberemos bajarnos el fichero ejecutable para Windows en la siguiente dirección:

`http://opencvlibrary.sourceforge.net`

7.1.1.1. Enlazar DLLs.

Tras obtener éste procedemos a su instalación en Windows. Ésta no tiene mayor dificultad ya que se debe realizar los pasos que se indican en su instalación. A partir de este punto, consideramos que se ha instalado en `C:\Archivos de programa\OpenCV`. Después de la instalación, debemos añadir el siguiente directorio en el `Path` del sistema:

`OpenCV\bin`

Éste se encuentra dentro del `Panel de control`, en `Sistema` y, dentro de éste, en la pestaña de `Opciones avanzadas`, seleccionamos `Variables de entorno`, y ahí encontramos el fichero `Path`. Ahora es necesario configurar en Visual C++ todas las librerías de OpenCV para que podamos realizar nuestras aplicaciones.

7.1.1.2. Personalizar opciones globales.

Para llamar a las librerías de OpenCV se debe primeramente en Visual C++ y dentro del menú `Herramientas` seleccionar la alternativa de `Opciones` y en la pestaña de `Directorios de VC++` añadir las siguientes direcciones:

Dentro de `Archivos de biblioteca` añadimos la siguiente dirección –véase la figura 22–:

```
C:\Archivos de programa\OpenCV\lib
```

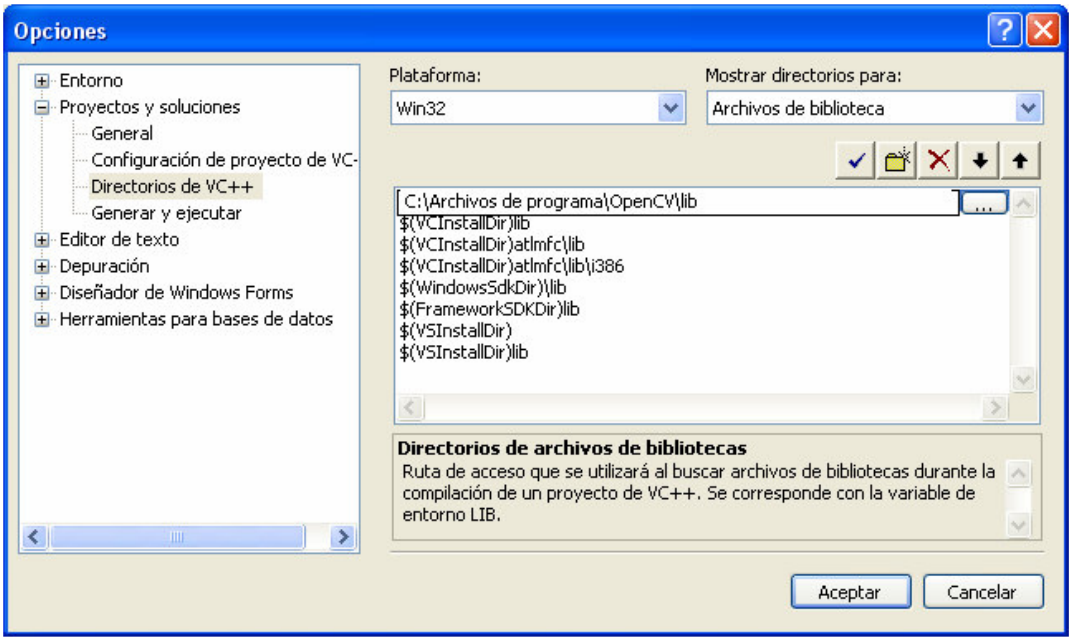


figura 22. Directorios de VC++. Archivos de biblioteca.

Ahora dentro de `Archivos de inclusión` debemos añadir las siguientes direcciones –véase la figura 23–:

```
C:\Archivos de programa\OpenCV\cv\include
C:\Archivos de programa\OpenCV\cxcore\include
C:\Archivos de programa\OpenCV\otherlibs\highgui
C:\Archivos de programa\OpenCV\cvaux\include
C:\Archivos de programa\OpenCV\otherlibs\cvcam\include
```

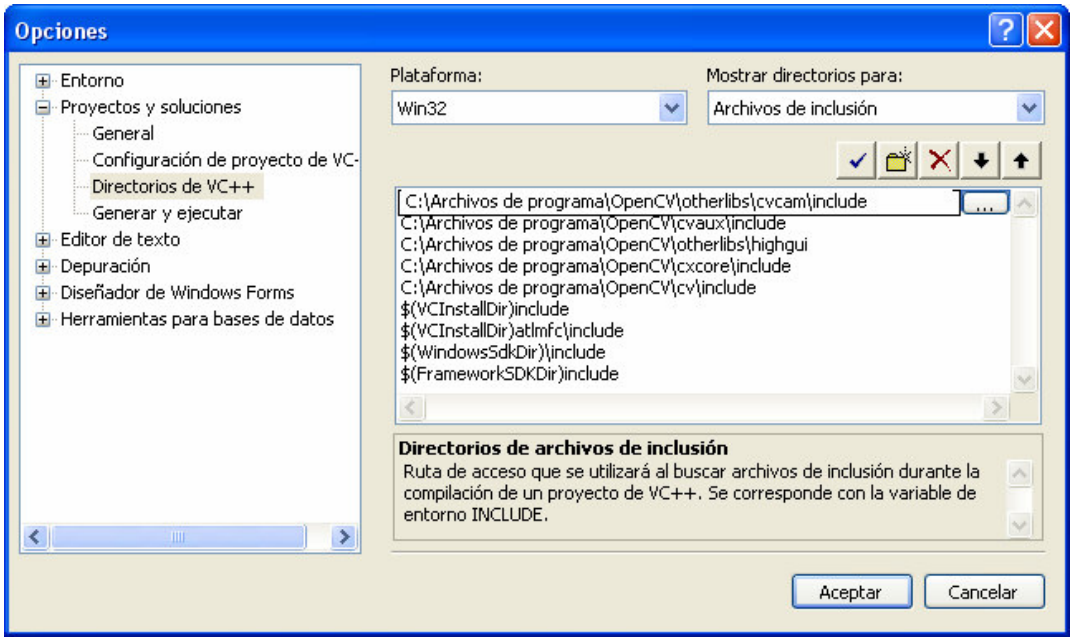


figura 23. Directorios de VC++. Archivos de inclusión.

Dentro de `Archivos de código fuente` añadiremos las siguientes direcciones –véase la figura 24–:

```
C:\Archivos de programa\OpenCV\cv\src
C:\Archivos de programa\OpenCV\cxcore\src
C:\Archivos de programa\OpenCV\cvaux\src
C:\Archivos de programa\OpenCV\otherlibs\highgui
C:\Archivos de programa\OpenCV\otherlibs\cvcam\src\windows
```

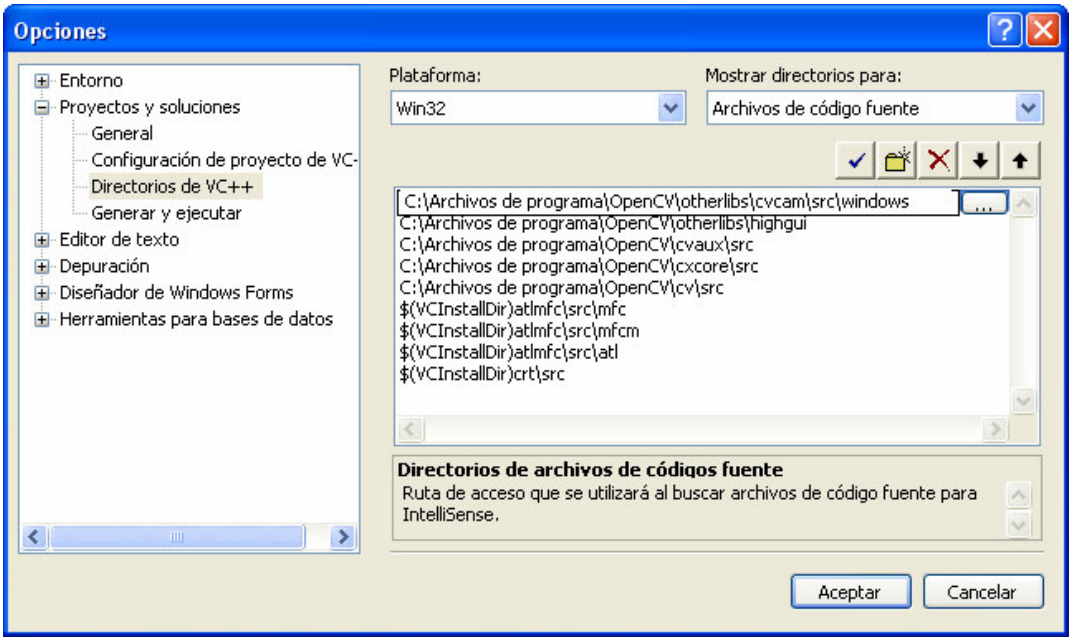


figura 24. Directorios de VC++. Archivos de código fuente.

Luego pulsa sobre el botón `Aceptar` de la ventana de diálogo `Opciones`.

7.1.1.3. Crear un nuevo proyecto `PieceImageDetection`.

En Visual C++ creamos una nueva aplicación[6]. Para ello, seleccionamos del menú: `Archivo` → `Nuevo` → `Proyecto`. Elige `Aplicación de consola Win32` y escribe el nombre del proyecto. Por ejemplo: `PieceImageDetection` y elige la carpeta destino. En la nueva pantalla, pulsa sobre finalizar.

Después de realizar los anteriores pasos Visual C++ creará la carpeta del proyecto. Por defecto, tiene el mismo nombre del proyecto:

```
PieceImageDetection.vcproj
```

Y tres ficheros fuente:

```
PieceImageDetection.cpp
stdafx.cpp
stdafx.h
```

`stdafx` son ficheros de cabecera precompilados, que se pueden usar si se desea reducir el tiempo de compilación.

[6] Para este ejemplo, se hace uso de la versión 2008 Express Edition [Visual C++ 08b].

Por ejemplo, considera que nosotros hemos creado un nuevo proyecto que se llama `PieceImageDetection`. Abre el fichero `PieceImageDetection.cpp`, e inserta las directivas `#include` asociadas a OpenCV:

```
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
```

Ten en cuenta que se deberían incluir después de `stdafx.h`. Ahora, escribe algo de código propio de OpenCV, y compila pulsando F7. Saldrán errores. Añade dependencias del proyecto al entorno de trabajo (*workspace*). Elige desde el menú: Proyecto → Propiedades.

Elige: Vinculador → Categoría Entrada → Dependencias adicionales. Añade las rutas a todas las librerías necesarias (`cxcore.lib cv.lib highgui.lib cvaux.lib cvcam.lib`). Ahora si en el momento de compilación denota la falta de un fichero `windows.h`, entonces necesitarás instalar la última versión de Microsoft SDK.

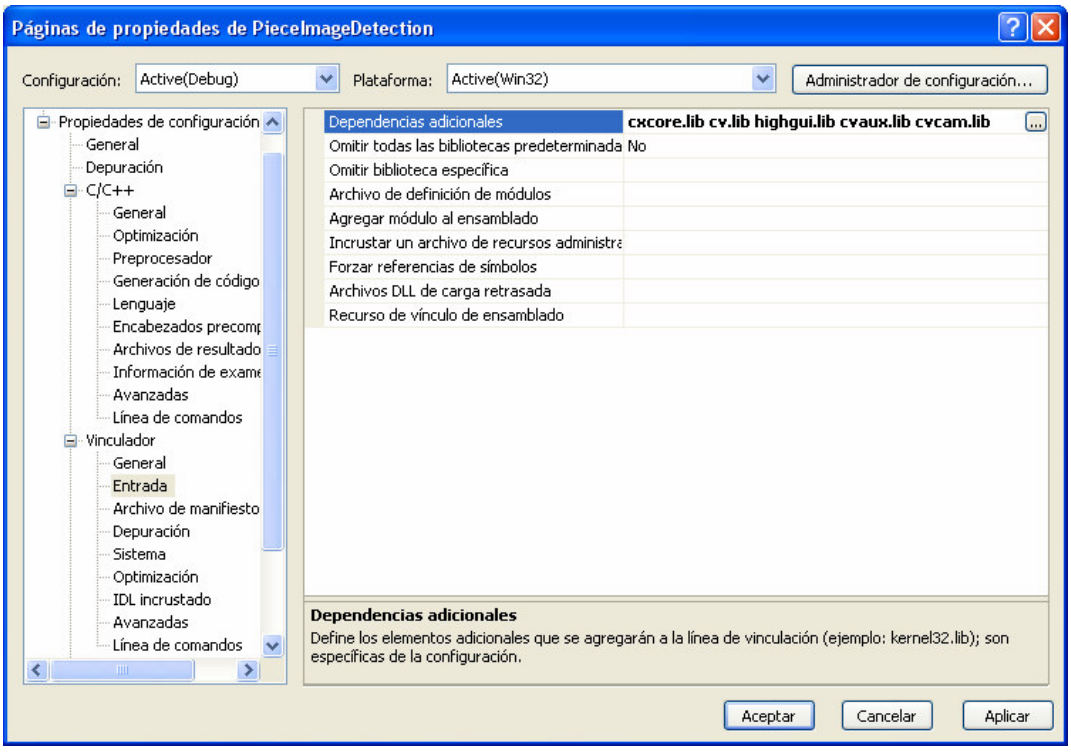


figura 25. Propiedades del proyecto. Dependencias adicionales.

Ahora el entorno de trabajo está listo para trabajar en Visual C++ con librerías de OpenCV.

7.1.2. Ejemplo de detección de bordes realizado en Visual C utilizando librerías de OpenCV.

A continuación, presentamos un ejemplo –véase las figuras 26 y 27– que se ha realizado en Visual C++ y utiliza librerías de OpenCV. Para que estos programas trabajen conjuntamente, Visual C++ debe llamar a las librerías de OpenCV, instalado previamente, y así se podrá compilar.

```
#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "stdafx.h"

#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#endif

char wndname[]='Edge';
char tbarname[]='Threshold';
int edge_thresh=1;

IplImage *image=0, *cedge=0, *gray=0, *edge=0;

//define a trackbar callback

void on_trackbar(int h)
{
    cvSmooth(gray, edge, CV_BLUR, 3, 3, 0, 0);
    cvNot(gray, edge);

    //run the edge detector on grayscale
    cvCanny(gray, edge, (float)edge_thresh, (float)edge_thresh*3, 3);

    cvZero(cedge);
    //copy edge points
    cvCopy(image, cedge, edge);
```

figura 26. Extracto de código para detección de bordes (OpenCV)-1.

En la primera parte de este programa tenemos las directivas `#include` que nos permite añadir librerías o funciones que se encuentran en otros ficheros a nuestro programa.

- Las variables `char` es un tipo especial de entero designado para ir guardando un tipo especial de caracteres.
- La biblioteca de OpenCV representa imágenes en el formato `IplImage` que viene de Intel Imagen que Procesa Biblioteca (IPL).
- Implementamos el algoritmo de Canny para detección de bordes, para esto utilizamos la función `CvCanny` la cual me permite la detección de bordes de imágenes en escala de grises (*grayscale*).
- La función `CvCopy` permite copiar los arreglos en otro.
- La función `CvShowImage` permite desplegar la imagen a la cuál se le ha realizado la detección de bordes.

En la siguiente figura 27 no se hace mayor aclaración de las líneas ya que se encuentra comentado en el mismo.

```
    cvShowImage(wndname, cedge);
}


int _tmain(int argc, _TCHAR* argv[])
{
    char* filename = argc == 2 ? argv[1] : (char*)"0863-012-3-distorted.jpg";

    if ( (image=cvLoadImage(filename,1))==0 )
        return -1;

    //create the output image
    cedge=cvCreateImage(cvSize(image->width, image->height), IPL_DEPTH_8U, 3);

    //convert to grayscale
    gray=cvCreateImage(cvSize(image->width, image->height), IPL_DEPTH_8U, 1);
    edge=cvCreateImage(cvSize(image->width, image->height), IPL_DEPTH_8U, 1);
    cvCvtColor(image, gray, CV_BGR2GRAY);

    //create a window
    cvNamedWindow(wndname, 1);

    //create a toolbar
    cvCreateTrackbar(tbarname, wndname, &edge_thresh, 100, on_trackbar);

    //show image
    on_trackbar(0);

    //wait for a key stroke; the same function arranges events processing
    cvWaitKey(0);
    cvReleaseImage(&image);
    cvReleaseImage(&gray);
    cvReleaseImage(&edge);
    cvDestroyWindow(wndname);

    return 0;
}
```

figura 27. Extracto de código para detección de bordes (OpenCV)-2.

7.2. Implementación en Matlab.

7.2.1. Instalación.

A continuación, indicaremos brevemente los pasos para la instalación de Matlab sobre el entorno de Windows. La instalación de Matlab no es nada complicada ya que una vez que se tienen los CDs de instalación procedemos a insertarlos en el CD-ROM y solamente seguimos los pasos que nos indican. Matlab viene con sus *toolbox* en el momento de su instalación (*Image Processing*) y, por tanto, no se necesita hacer ninguna configuración adicional [MATLAB 08].

7.2.2. Ejemplo de detección de bordes realizado en Matlab.

Ahora, con objeto de comparar ambas alternativas, veremos los pasos a seguir para implementar el mismo programa de detección de bordes realizado con librerías de OpenCV llamadas desde Visual C++, pero ahora en Matlab.

```
function pieceImageDetection(pImagePiece)

    figure('Name', 'Image Piece: Original');
    imshow(pImagePiece);

    %grayscale image

    imagePieceGrayscale=rgb2gray(pImagePiece);

    %binarisation

    imagePieceBinary=imagePieceGrayscale<120;
    imagePieceBW=bwareaopen(imagePieceBinary,50);
    dimension=size(imagePieceBinary);
    [imagePieceBW, thresh]=edge(imagePieceBinary, 'sobel');

    clear imagePiece imagePieceGrayscale imagePieceBinary;

    %determine borders and to show the different detected objects.

    [imagePieceBWBoundaries,L]=bwboundaries(imagePieceBW,'noholes');

    figure('Name', 'Image Piece: Boundaries');
    imshow(label2rgb(L, @jet));

    clear imagePieceBWBoundaries imagePieceBW;

    %
```

figura 28. Extracto de código para detección de bordes (Matlab).

- La función `imread` permite leer las imágenes de un fichero, su contenido viene dado ya en el argumento de entrada `pImagePiece`.
- La función `rgb2gray` me permite transformar una imagen a escala de grises.
- La función `edge` permite realizar la detección de bordes de una imagen binaria mediante los operadores de: Prewitt, Sobel, Canny o Laplaciana del Gaussiano, entre otros.
- La función `bwareaopen` borra de una imagen binaria, por defecto, los componentes conectados (objetos) que tienen menos de un número de píxeles (50), produciendo como salida otra imagen binaria.
- La función `bwboundaries` permite extraer los bordes de cada uno de los objetos de una imagen binaria y etiquetarlos. Si se quiere desplegar

las imágenes de `bwboundaries` se debe utilizar la función `label2rgb` que permite convertir una matriz etiquetada en una imagen RGB.

- La función `figure` permite crear ventanas independientes para desplegar las imágenes en cuadros diferentes.
- La función `imshow` despliega tanto la imagen original como los bordes en dos ventanas `figure`.

7.3. Conclusiones.

Podemos darnos cuenta que en Matlab la programación es más sencilla y más comprensible que en Visual C++. De hecho, en Matlab conseguimos los mismos resultados que en Visual C++ sin tener que realizar aplicaciones tan extensas. Además, el estilo de programación de Matlab es más sencillo, y con una alta capacidad de cómputo para procesar datos matriciales como son las imágenes. Por último, OpenCV integra una función de visualización, pero es muy básica. Para suplir esta deficiencia, existe la biblioteca GTK+[7] que integra funciones de visualización y dispone de un generador gráfico de interfaces.

[7] http://www.gtk.org