Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ASISTENTE A LA REALIZACIÓN DE PUZLES

Alumno: Patricia Martínez Padilla

Tutores: Alejandro Sanchez García
Silvia María Satorres Martínez
Dpto: Ingeniería Electrónica y Automática

Septiembre, 2019

RESUMEN

Este proyecto tiene como objetivo la creación de una interfaz gráfica para la resolución de rompecabezas automático que se basa en conceptos múltiples de procesamiento de imagen así como el algoritmo de ensamblaje de piezas desmontadas.

Para llevar a cabo este proyecto se han requerido varias herramientas software como son: el software matemático MATLAB que ofrece un lenguaje de programación propio (lenguaje M) y la extensión Image Processing Toolbox para el procesamiento, el análisis y la visualización de imágenes, así como para el desarrollo de algoritmos.

ABSTRACT

This project aims to create an automatic puzzle solving program that is based on multiple concepts of image processing as well as the assembling of disassembled parts algorithm.

To carry out this project, several software tools have been required, such as the MATLAB mathematical software that offers its own programming language (M language) and the Image Processing Toolbox extension for image processing, analysis and visualization, as well as for the development of algorithms.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El presente capítulo recoge las razones del trabajo realizado, cómo ha surgido, cuáles son los objetivos y las diferentes variantes que se pueden seguir para abordar proyectos de características similares.

1.1. Contexto

Un rompecabezas o puzle es un juego popular que desafía al jugador a formar una imagen o figura uniendo entre sí piezas que encajan a la perfección. Un rompecabezas estándar para adultos está formado por 50 o más piezas. Mientras que los rompecabezas para niños implican de 15 a 4 piezas. Las piezas que conforman un puzle son variadas. La figura 1 da algunos ejemplos de sus variaciones. Un rompecabezas canónico es aquel en el que las piezas son cuadradas y se unen en forma cuadriculada a través de pestañas y bolsillos a lo largo de sus bordes. La Figura 1 (a) muestra un rompecabezas canónico, difiere del rompecabezas estándar en que este es apictorial, lo que significa que no tiene una imagen guía. La Figura 1 (b) también demuestra un rompecabezas apictorial, sin embargo, a diferencia del primer rompecabezas, no es canónico: las piezas encajan en un esquema diferente al esquema de la red canónica. La Figura 1 (c) muestra un rompecabezas en 3D.

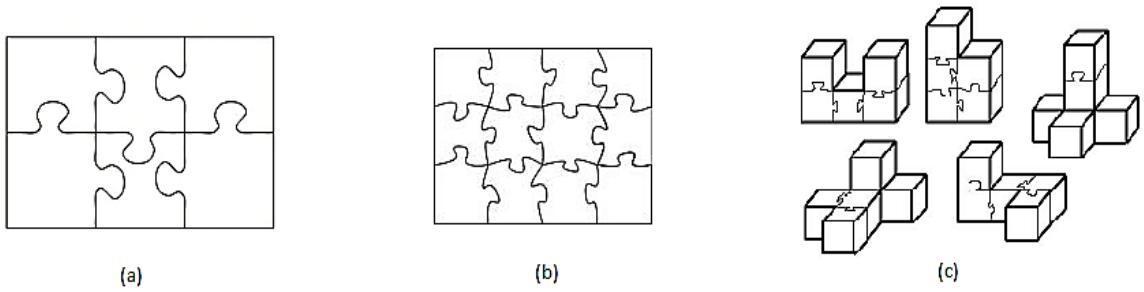


Figura 1. Tipos de puzle.

En la resolución de un rompecabezas entran en juego dos factores importantes: la compatibilidad de las piezas a encajar y la imagen de las piezas ensambladas que debe seguir el patrón de la imagen de referencia. Sin

embargo, existe la posibilidad de que el puzle pueda resolverse incluso sin la presencia de la imagen, pues la solución es la unión de piezas que son compatibles entre sí.

En éste trabajo la tarea es ensamblar piezas de un puzle utilizando técnicas de procesamiento de imágenes sin tener información sobre la imagen de referencia. En este proyecto se estudiarán rompecabezas canónicos de 6 piezas.

1.2. Motivación

Desde que somos pequeños, los puzles han sido utilizados para estimular el desarrollo cognitivo y sensorial con el fin de enseñarnos a percibir los elementos que nos rodean y comenzar así un proceso de interrelación. A su vez, el campo de la visión por computador busca producir el mismo efecto para que los sistemas robóticos puedan extraer información a partir de imágenes ofreciendo soluciones a un problema real. Es por ello, que la resolución de un puzle utilizando técnicas de visión por computador resulta ser un problema atractivo y un desafío intelectual. Ésta disciplina incluye métodos para adquirir, procesar y analizar imágenes del mundo real con el fin de que éstas puedan ser tratadas por un ordenador. Estas técnicas permiten automatizar tareas y aportar a las máquinas la información necesaria para una correcta toma de decisiones.

Debido a la importancia que tiene la visión artificial dentro de la ingeniería en general y en la industria en particular, asignaturas que abarcan estos conceptos han sido incluidas en el Plan de Estudios de Grado. En éste caso, es la asignatura de Sistemas de percepción industrial la que proporciona un amplio estudio de la estructura, componentes y conocimientos de las técnicas y algoritmos utilizados en las etapas que conforman un sistema de visión por computador.

Es por tanto en este punto, cuando surge el interés de la realización de un proyecto implicado en el estudio y la puesta en marcha de múltiples conceptos

de visión por computador que a su vez, son altamente aplicables a proyectos futuros.

1.3. Objetivos

El objetivo principal de este proyecto es la creación de una interfaz gráfica en la cual se podrá cargar una imagen de las piezas desordenadas que conforman un puzle para que finalmente el algoritmo programado nos muestre la solución de dicho puzle y las piezas en su correcta orientación. Para alcanzar el objetivo principal se deben de seguir varios objetivos específicos mencionados a continuación:

- Adquisición de la imagen que contiene las piezas del puzle.
- Procesamiento de imagen.
- Extracción de características de las piezas para extraer información sobre su forma.
- Proceso de emparejamiento para el ensamblaje.
- Visualización en una interfaz gráfica.

Con el fin de superar los objetivos mencionados se tomaron varias suposiciones:

- Las piezas de la imagen de origen no se superponen ni se tocan.
- La imagen original es exactamente vertical y no hay efecto de perspectiva.
- Todas las piezas son de forma rectangular (las diagonales entre esquinas opuestas deben cortar a los 90 grados) formadas por solo 4 lados que pueden ser: recta, cabeza u orificio.
- Para un puzle de 6 piezas: Las piezas se deben organizar en una matriz de 2x3.
- Las piezas pueden tomar cualquier orientación siempre y cuando uno de los lados se mantenga lo más paralelo posible al eje horizontal de la imagen.

1.4. Estado de la técnica

La búsqueda de un algoritmo eficaz capaz de resolver rompecabezas lleva décadas de estudios. Durante ese tiempo se han publicado artículos en los cuales se nombra el extenso potencial que se obtiene de este trabajo y que es aplicable a otros esfuerzos más serios como reconstruir artefactos arqueológicos, encajar objetos escaneados en impresoras 3D o incluso ajustar una secuencia de aminoácidos de una proteína conocida a un mapa de densidad electrónica.

El estudio de los rompecabezas en ciencias de la computación comenzó en 1964 con Freeman y Gardner [1], quienes abordaron el problema en términos de si las técnicas de visión artificial son capaces de determinar si dos bordes coinciden. Su modelo proponía una forma de resolver rompecabezas utilizando la división de curvas y encontrando esquinas agudas para hacer correspondencias según las distancias entre esos puntos. Dichas curvas fueron identificadas utilizando códigos de cadena, este método permite codificar configuraciones geométricas arbitrarias para su posterior análisis y manipulación en entornos digitales. El código de cadena más intuitivo fue propuesto por Freeman en 1961 dónde una curva es representada por una secuencia de pequeños vectores de longitud unitaria y un conjunto limitado de posibles direcciones. En la práctica, el borde del objeto es superpuesto en una cuadrícula y los puntos de borde son aproximaciones a dicha cuadrícula. El movimiento a través de los píxeles del límite está codificado con un alfabeto $\sum(F8) = \{0,1,2,3,4,5,6,7,8\}$ donde cada elemento representa 45º desde la dirección positiva del eje de coordenadas x. Éste método es conocido como Código de Cadena de Freeman F8, Freeman también determinó que los límites de las formas pueden describirse con conectividad de 4 píxeles siendo en este caso representados por 90º desde la dirección positiva del eje x [2]. Ya que el emparejamiento entre dos piezas se hace entre una pequeña sección de los límites de sus límites respectivos, Freeman y Gardner separaron las cadenas en varias secciones llamadas ‘chainlets’ cada uno de los cuales debe ser emparejada con uno, y solo uno, de los chainlets de otra figura. Características tales como: la longitud de la cadena, longitud de la línea recta entre dos puntos finales, número de veces que la cadena cruza el vector distancia y la relación

entre las áreas encerradas por la cadena fueron extraídas de los chainlets con el fin de buscar la similitud entre características como condición necesaria de emparejamiento.

Un conjunto de m características en un chainlets puede ser expresado como un vector en el espacio de características m-dimensionales. Dado que la similitud ente características es una condición de emparejamiento, ambos vectores deben encontrarse muy cerca o de lo contrario, ser un límite exterior. Siendo el vector de características i denotado por F_i, y sus componentes por f_{ik} (k=1, 2, . . , m) donde m es el número total de características. Dos cadenas i y j son similares si:

$$\left|f_{ik} - f_{jk}\right| \leq T_k \qquad (1)$$

se cumple para todo k. Aquí T_k es un umbral positivo asignado a la característica k en función de la precisión con la que se quiera determinar la característica.

Este proceso fue basado en un crecimiento en espiral hacia afuera, en sentido anti horario y centrado alrededor de una pieza elegida inicialmente. Este recorrido es repetido hasta llegar a una conclusión final donde todas las cadenas son emparejadas o clasificadas como límites exteriores. Es en este punto donde fueron descubiertos los errores de ensamblaje que surgen al no conseguir cerrar ciertas uniones debido a la presencia de ruido de cuantificación que puede ocultar algunas diferencias de contorno. El flow chart de este proceso es mostrado en la Figura 2.

Con este trabajo Freeman y Gardner buscaban explorar las capacidades de una computadora digital para resolver problemas de datos gráficos, particularmente problemas en los que los datos gráficos eran altamente manipulados y reconocidos.

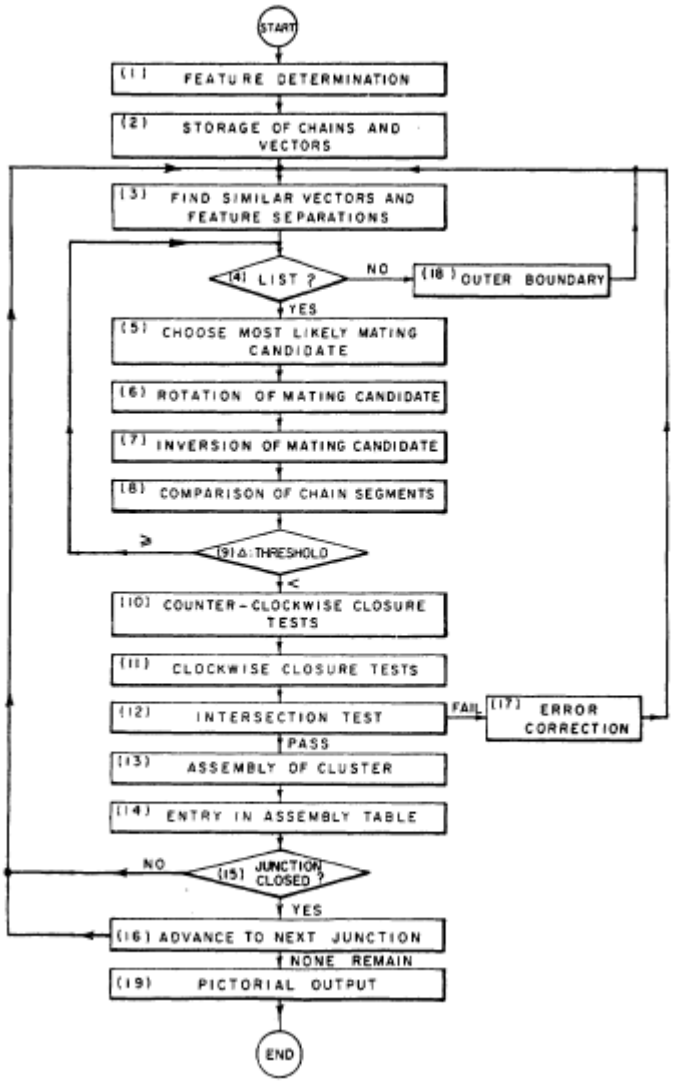


Figura 2. Flow chart propuesto por H. Freeman y L. Garder.

Más tarde, Wolfson [4] replanteó el problema siguiendo un enfoque el cual se asemeja mucho al utilizado en este trabajo. Wolfson utilizó aproximaciones poligonales para delimitar los objetos, de esta manera suavizaría los bordes y obtendría curvas más suavizadas y con reducción de ruido. Estas curvas serían posteriormente divididas en cuatro subcurvas pertenecientes a los lados de la pieza de puzle que serán utilizadas para el proceso de emparejamiento.

Fueron aplicados dos métodos para encontrar las cuatro esquinas de cada pieza. El primer método está basado en encontrar los puntos de interrupción, estos puntos son reconocidos por tener un cambio brusco de pendiente en la dirección de la tangente y, por lo tanto, obtener en la segunda derivada un pico. La segunda técnica de búsqueda explota el hecho de que las

esquinas de una pieza estén entre sí a 90º. Ésta característica es usada en este trabajo como primera premisa de búsqueda de esquinas. Las piezas son identificadas según pertenezcan al marco o no, para ello se evaluó las posibles secciones rectas que pertenecían al límite del objeto. Por lo tanto, el proceso de ensamblaje comienza por crear el marco de manera que las distancias entre lados sean las menores posibles e iguales entre lados opuestos. El problema combinatorio mencionado es conocido como 'el problema del vendedor ambulante' cuyos algoritmos ya creados fueron de utilidad a Wolfson en su estudio y que, seguidamente, utilizaría Goldberg para proponer su algoritmo y resolver 204 piezas de puzle, el rompecabezas más grande resulto automáticamente hasta la fecha. [4]

Al igual que el trabajo de Wolfson, Golberg ensambla primero las piezas del marco usando la heurística para el problema del vendedor ambulante pero difiere de Wolfson al contemplar la posibilidad de obtener piezas que no necesariamente tengan cuatro lados bien definidos, además de usar en todo momento la geometría global, por ejemplo, manteniendo la geometría de las piezas iniciales. La colocación de las piezas interiores se lleva a cabo en tres etapas: una calificación que mide qué tan bien encaja una pieza, una estrategia para el orden de colocación de las piezas y un paso de optimización para reajustar la incrustación después de colocar cada nueva pieza.

La calificación que mide qué tan bien encaja una pieza P se lleva a cabo en dos pasos: primero calcula la posición que tendría P si realmente perteneciera a ese hueco y luego se calcula una puntuación usando esta posición. La puntuación para P es calculada caminando por el límite y buscando en cada vértice el límite más cercano en cualquiera de las piezas vecinas. La puntuación es el promedio de las distancias entre los vértices de P y los puntos más cercanos. Siguiendo la calificación obtenida se ensambla el rompecabezas.

Desde un punto de vista no tan geométrico, Yao y Shao [5] decidieron resolver el acertijo usando la coincidencia de rompecabezas por combinación de formas. Existen cuatro esquinas en una pieza de un puzle canónico, y su curva de límite se puede separar por tanto en cuatro bordes entre los cuatro

puntos de esquina. Cada uno de los bordes es clasificado y denotado por una 'L' si pertenece a una línea recta, un cóncavo o borde curvo denotado por 'C' y un convexo denotado por 'V'. Cualquier pieza de puzle canónico puede ser descrita por medio de estos tres tipos de borde. Esqueletizando los cuatro bordes todas las piezas del rompecabezas se pueden clasificar en: esquina, borde o interior. Siguiendo la nomenclatura anterior, las piezas de las esquinas se clasifican en 4 tipos que se denominan R0, R1, R2 y R3, respectivamente. Las configuraciones son 'CCLL', 'VCLL', 'CVLL' y 'VVLL' como se puede en la tabla de la Figura 3. Los patrones están descritos empezando por la esquina inferior izquierda. Las piezas de borde, denotadas por E, son clasificadas en 8 tipos mientras que las piezas denotadas por 'I', piezas del área interior, tienen 6 configuraciones. Por lo tanto, las piezas de cualquier puzle canónico pueden ser clasificadas en 18 categorías, cuyas relaciones de conexión se resumen en las columnas 4 a 7 de la Figura 3.

	Piece type	Boundary description	Edge 0	Edge 1	Edge 2	Edge 3
R	R_0	*CCLL*	***E***.*V*	***E***.*V*	None	None
	R_1	*VCLL*	***E***.*C*	***E***.*V*	None	None
	R_2	*CVLL*	***E***.*V*	***E***.*C*	None	None
	R_3	*VVLL*	***E***.*C*	***E***.*C*	None	None
E	E_0	*CCCL*	***E***.*V*, ***R***.*V*	***I***.*V*	***E***.*V*, ***R***.*V*	None
	E_1	*CVCL*	***E***.*V*, ***R***.*V*	***I***.*C*	***E***.*V*, ***R***.*V*	None
	E_2	*VCCL*	***E***.*C*, ***R***.*C*	***I***.*V*	***E***.*V*, ***R***.*V*	None
	E_3	*CCVL*	***E***.*V*, ***R***.*V*	***I***.*V*	***E***.*C*, ***R***.*C*	None
	E_4	*VVCL*	***E***.*C*, ***R***.*C*	***I***.*C*	***E***.*V*, ***R***.*V*	None
	E_5	*CVVL*	***E***.*V*, ***R***.*V*	***I***.*C*	***E***.*C*, ***R***.*C*	None
	E_6	*VCVL*	***E***.*C*, ***R***.*C*	***I***.*V*	***E***.*C*, ***R***.*C*	None
	E_7	*VVVL*	***E***.*C*, ***R***.*C*	***I***.*C*	***E***.*C*, ***R***.*C*	None
I	I_0	*CCCC*	***E***.*V*, ***I***.*V*	***E***.*V*, ***I***.*V*	***E***.*V*, ***I***.*V*	***E***.*V*, ***I***.*V*
	I_1	*VCCC*	***E***.*C*, ***I***.*C*	***E***.*V*, ***I***.*V*	***E***.*V*, ***I***.*V*	***E***.*V*, ***I***.*V*
	I_2	*VCVC*	***E***.*C*, ***I***.*C*	***E***.*V*, ***I***.*V*	***E***.*C*, ***I***.*C*	***E***.*V*, ***I***.*V*
	I_3	*CVVC*	***E***.*V*, ***I***.*V*	***E***.*C*, ***I***.*C*	***E***.*C*, ***I***.*C*	***E***.*V*, ***I***.*V*
	I_4	*VVVC*	***E***.*C*, ***I***.*C*	***E***.*C*, ***I***.*C*	***E***.*C*, ***I***.*C*	***E***.*V*, ***I***.*V*
	I_5	*VVVV*	***E***.*C*, ***I***.*C*	***E***.*C*, ***I***.*C*	***E***.*C*, ***I***.*C*	***E***.*C*, ***I***.*C*

Tabla 1. Clasificación de los bordes de las piezas canónicas según Yao y S Shao

El contenido de esta tabla juega un papel importante en el algoritmo propuesto en este trabajo para resolver el problema del rompecabezas.

2. DISEÑO DEL PROYECTO

El presente capítulo se detalla los métodos y tecnologías utilizados para llevar a cabo el proyecto así como el software empleado para la resolución del problema. A continuación, en el capítulo 3 se detallará el enfoque técnico del problema con las etapas de procesamiento de imagen utilizadas, enfatizando en las técnicas que más influyeron para alcanzar los objetivos. El capítulo 4 recoge el funcionamiento que sigue el solucionador del puzle cuyos resultados obtenidos serán mostrados en el capítulo 5 junto con las conclusiones alcanzadas. Las líneas futuras serán abordadas en el capítulo 6 y, finalmente, las referencias bibliográficas y anexos en los capítulos 7 y 8.

2.1. Estructura del Proyecto

Como se ha podido comprobar en el capítulo anterior, para abordar este trabajo tenemos un inmenso campo de posibilidades. En este caso, para intentar resolver el problema seguiremos con la estructura mostrada a continuación.

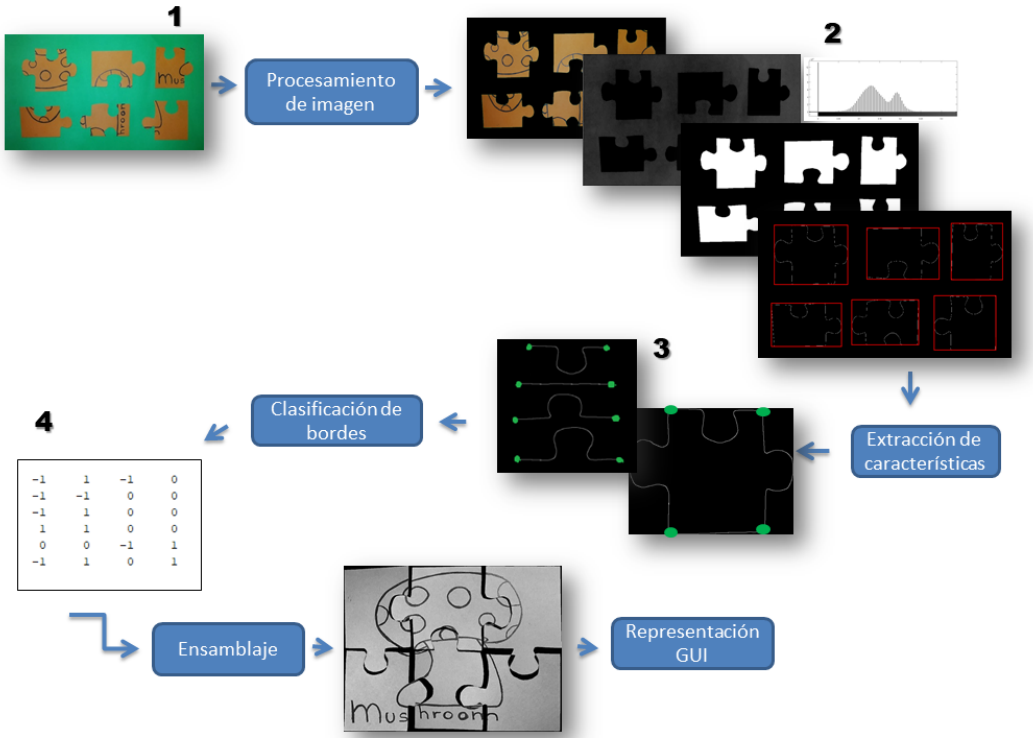


Figura 3. Estructura del proyecto.

En primer lugar el proceso de Adquisición (1) se encargará de tomar la imagen origen de las piezas del puzle que se quiere ensamblar, la imagen será tomada con un teléfono móvil de modo que en trabajos futuros pueda enfocarse a una aplicación para móvil inteligente. Un fondo verde será utilizado como fondo de las piezas para una fácil segmentación. Una vez digitalizada la imagen es guardada en el ordenador en el que correrá el software encargado de la resolución.

La imagen origen será procesada (2) con el fin de eliminar el fondo y obtener solo la imagen de las piezas que son objeto de interés. Una serie de técnicas serán aplicadas para solucionar los posibles problemas que pueden producirse en el proceso de adquisición y para facilitar la búsqueda de información que se llevará a cabo en las etapas de extracción de características.

Las piezas individualmente son identificadas para extraer los píxeles del contorno, los cuales serán transformados a coordenadas polares y recorridos en el sentido de las agujas del reloj para identificar las esquinas que determinan las piezas y diferenciar así los 4 lados (3). Estos lados serán clasificados según sean lado, cabeza u orificio representados numéricamente por 0, 1 y -1 respectivamente. De este modo cada pieza de puzle queda definida en función de sus lados. Los datos son almacenados en una matriz (4).

El ensamblaje del puzle se lleva a cabo con los datos de la matriz almacenada que contiene los datos de las piezas. El algoritmo solucionador utilizará los datos de los lados de las piezas para enlazarlos con los lados vecinos. La solución final permitirá fusionar las imágenes de las piezas individuales para formar la resolución del puzle (5).

El resultado se mostrará en una interfaz gráfica creada por el software detallado en el apartado siguiente y que permitirá al usuario cargar la imagen origen desde pantalla para visualizar los resultados.

2.2. Software utilizado

En este apartado hablaremos sobre la herramienta software matemático MATLAB utilizada para la creación de la estructura anterior.

2.2.1. Generalidades

MATLAB es un software matemático cuyo nombre viene de sus siglas en inglés Matrix Laboratory (laboratorio de matrices), esto indica su característica principal para la que fue diseñado, el cálculo matricial. Sin embargo, se ha ido extendiendo y adaptándose a las nuevas modalidades de modo que abarca la mayoría de ámbitos en ingeniería.

Este software ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M) que permite operaciones vectoriales y matriciales, funciones, cálculo lambda y programación orientada a objetos.

Entre sus prestaciones principales se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete MATLAB dispone, entre otras muchas, de dos herramientas adicionales que expanden sus prestaciones: Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI).

Otra de las opciones que incluye Matlab y que van a ser de gran utilidad serán las cajas de herramientas (toolboxes), en concreto la Toolbox de Procesamiento de Imágenes (Image Processing Toolbox). Como su nombre indica, dicha caja de herramientas proporciona un conjunto completo de algoritmos estándar necesarios para el procesamiento, análisis y visualización de imágenes, así como para el desarrollo de algoritmos. Dichos algoritmos y funciones serán detallados en el Capítulo 3 de Tratado de Imagen.

En este trabajo utilizaremos la gran mayoría de sus funciones principales y además haremos uso de herramientas adicionales como GUIDE para el desarrollo de la interfaz gráfica que detallaremos en el siguiente apartado.

2.2.2. Desarrollo de interfaz gráfica (GUI)

Una interfaz gráfica de usuario (GUI), es una interfaz construida a través de objetos gráficos, tales como menús, botones, listas y barras de desplazamiento que representan información y acciones disponibles. De este modo, permitirá al usuario una comunicación fácil con el sistema operativo de una máquina u ordenador. [6]

A continuación describirá cómo se estructuran y desarrollan las GUIs usando el entorno GUIDE de Matlab con la finalidad de transmitir al lector una serie de conceptos básicos que le permitan entender el funcionamiento del programa que se desarrolla en este trabajo fin de grado, tratando de no profundizar más de lo necesario.

Matlab permite desarrollar y definir un conjunto de elementos (botones, menús, ventanas...) que permiten utilizar de manera fácil e intuitiva, programas realizados en este entorno. Los gráficos de Matlab tienen una estructura jerárquica, formada por objetos de distintos tipos.

En la figura 4 se refleja la forma en la que se dispone esta jerarquía:

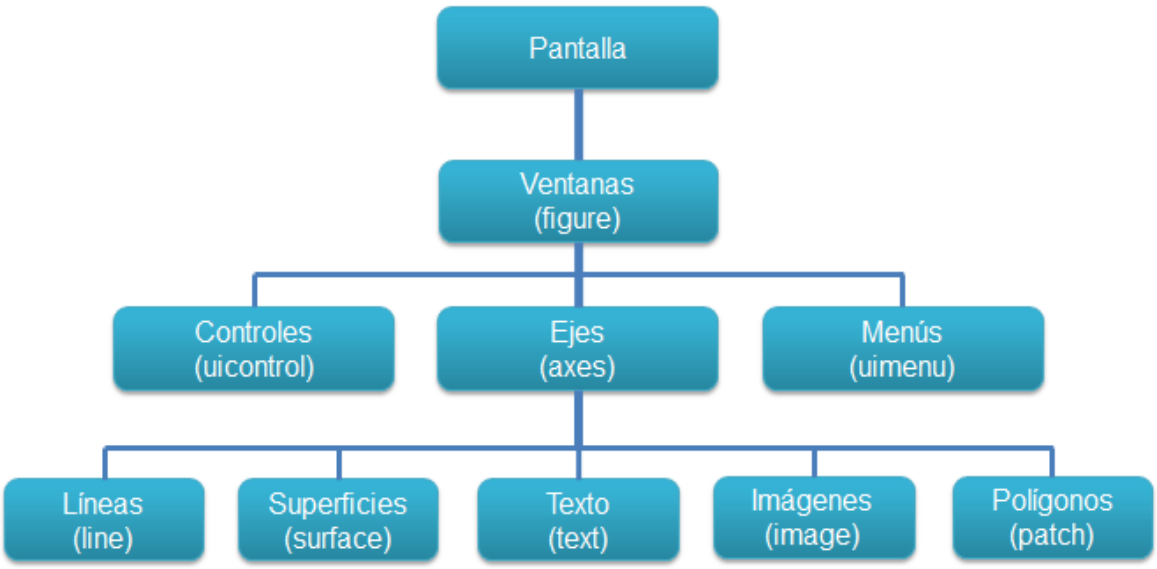


Figura 4. Jerarquía de la GUI en Matlab.

Como se muestra en el esquema anterior, el objeto más general es la pantalla y de él dependen todos los demás. Una pantalla puede contener una o más ventanas (figures) y a su vez cada ventana puede contener uno o varios ejes de coordenadas (axes) con los que se representan objetos de más bajo nivel. Dentro de ellas se pueden encontrar controles (uicontrols) y menús (uimenus).

Finalmente, los ejes pueden contener los cinco elementos que se indican en el gráfico (*line, surface, text, image* y *patch*).

2.2.2.1 Entorno GUIDE

Se pueden tomar varios caminos para acceder al entorno GUIDE de Matlab. Una de las formas de acceder al entorno es escribiendo la palabra «guide» en la «Command Windows» en la ventana de comandos de Matlab y pulsando la tecla intro. Hecho esto, aparecerá una ventana emergente como la que se muestra en la Figura 5.

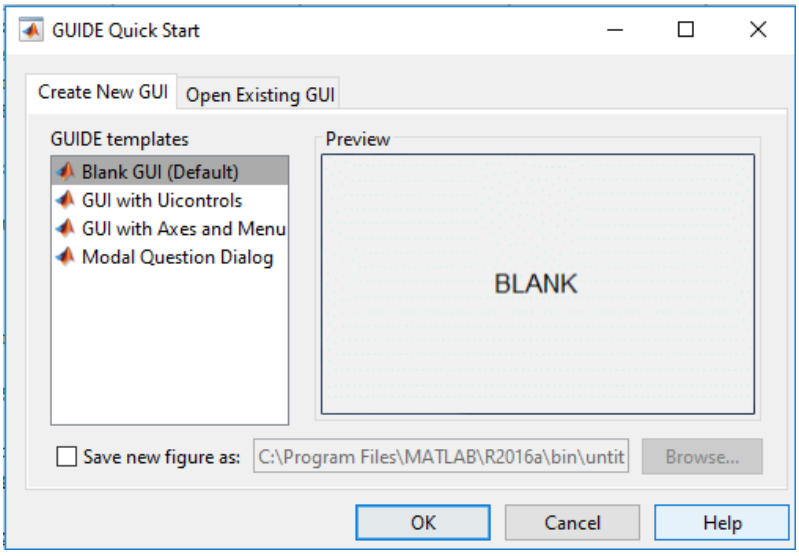


Figura 5. Ventana inicial para crear una GUI en Matlab.

Seleccionando la opción marcada por defecto estaremos accediendo a una GUI totalmente en blanco que será el espacio de trabajo en el cual dispondremos nuestros objetos (Figura 6).

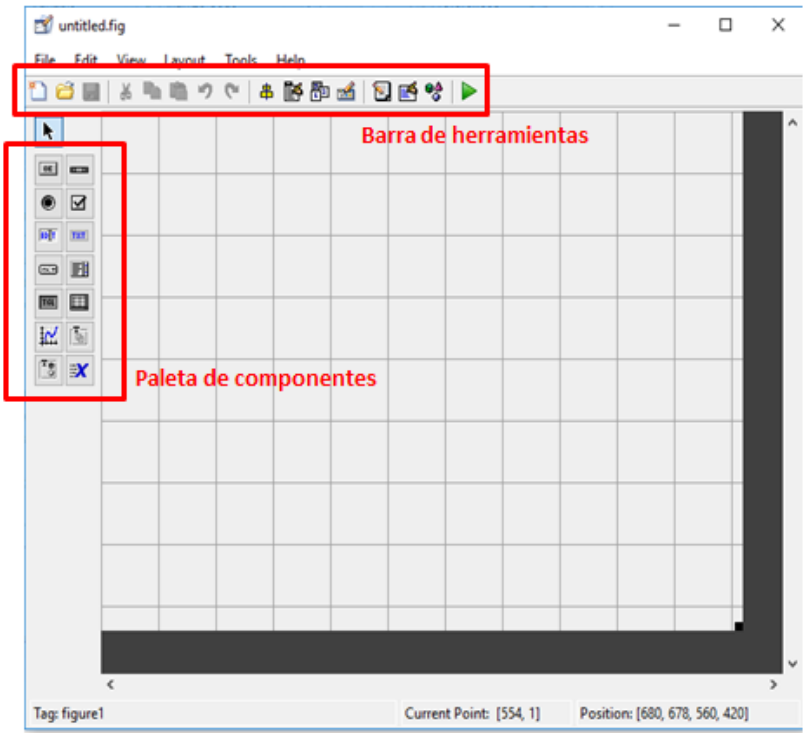


Figura 6. GUI de Matlab en blanco.

Una vez que se accede al entorno GUIDE, se procede a crear el diseñar la organización de los objetos gráficos que compondrán nuestra interfaz. A continuación de muestran las opciones de edición:

I. Paleta de componentes

Muestra los diferentes controles que Matlab permite añadir en el área de trabajo. Para añadirlos solo basta con arrastrarlos. En la tabla 2 se especifican cada uno de los botones que aparecen en la figura anterior junto al tipo de botón al que pertenecen y una breve descripción de las funcionalidades que tiene.

Botón	Tipo	Descripción
	**Push Button**	Botón de presión.
	**Slider**	Barra de deslizamiento.
	**Radio Button**	Botón circular, como un botón de presión con enclavamiento.

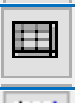
	**Check Box**	Casilla de verificación, similar a botón circular.
	**Edit**	Texto editable, permite al usuario de la GUI introducir valores.
	**Static Text**	Texto estático, proporciona información o resultados.
	**Pop-up Menu**	Menú desplegable, permite elegir opción de entre varias.
	**Listbox**	Lista, muestra al usuario una lista de variables.
	**Toggle Button**	Botón de presión con enclavamiento.
	**Table**	Tabla, con datos que el usuario puede ver y modificar.
	**Axes**	Ejes en los que se representan las imágenes y gráficas.
	**Panel**	Sirve para organizar un grupo de botones.
	**Button Group**	Grupo de botones circulares, mutuamente sucesivos.
	**ActiveX Control**	Utilizado para introducir videos.

Tabla 2. Paleta de componentes de la GUI.

II. Barra de herramientas

En la siguiente tabla (Tabla 3) se muestran las opciones que se encuentran dentro de la barra de herramientas señalada en la Figura 6.

Botón	Tipo	Descripción
	**New Figure**	Crea una nueva interfaz gráfica
	**Open Figure**	Abre una GUI ya existente
	**Save Figure**	Guarda la GUI con la que se está trabajando
	**Cut**	Corta el objeto seleccionado
	**Copy**	Copia el objeto seleccionado
	**Paste**	Pega el objeto que se halle en el portapapeles
	**Undo**	Deshace la última acción realizada
	**Redo**	Rehace la última acción deshecha
	**Align Objects**	Sirve para alinear los objetos introducidos entre si

	**Menu Editor**	Abre una ventana para crear una barra de menús
	**Tab Order Editor**	Ordena los objetos que se superponen entre si
	**Toolbal Editor**	Abre una ventana para crear la barra de herramientas
	**Editor**	Abre una ventana que permite editar el código de la GUI
	**Property Inspector**	Permite modificar las propiedades de cada objeto
	**Object Browser**	Muestra la vista en árbol de todos los objetos de la GUI
	**Run Figure**	Ejecuta la GUI

Tabla 3. Barra de herramientas de la GUI.

Con las herramientas anteriormente citadas se puede diseñar la ventana de la interfaz gráfica propuesta en este proyecto, que como se ha comentado, debe ser lo más fácil e intuitiva posible para el usuario.

2.2.2.2 Estructura de la GUI

Cuando se salva la aplicación GUI que se crea en el entorno anterior se generan automáticamente dos ficheros con distintas extensiones: *archivo.FIG* y *archivo.M.*

El archivo con extensión .fig, es un fichero binario que contiene la información referente a la GUI que se ha diseñado. En él se guarda la ventana con los objetos gráficos insertados en ella junto con sus propiedades para su posterior modificación o uso. Este fichero se actualiza automáticamente al realizar cambios desde el entorno GUIDE, por lo que el programador no debe manipularlo.

Archivo.m, es un fichero de texto que contiene el código de la GUI. En él se crean las funciones asociadas a cada objeto gráfico creado en la GUI, dichas funciones se las conoce como callback y permiten definir la acción que llevará a cabo la aplicación cuando se actúe sobre dicho objeto. Por lo tanto, el código generado está compuesto de funciones (Figura 7) que se ejecutan al ser llamadas por el usuario mediante la interfaz.

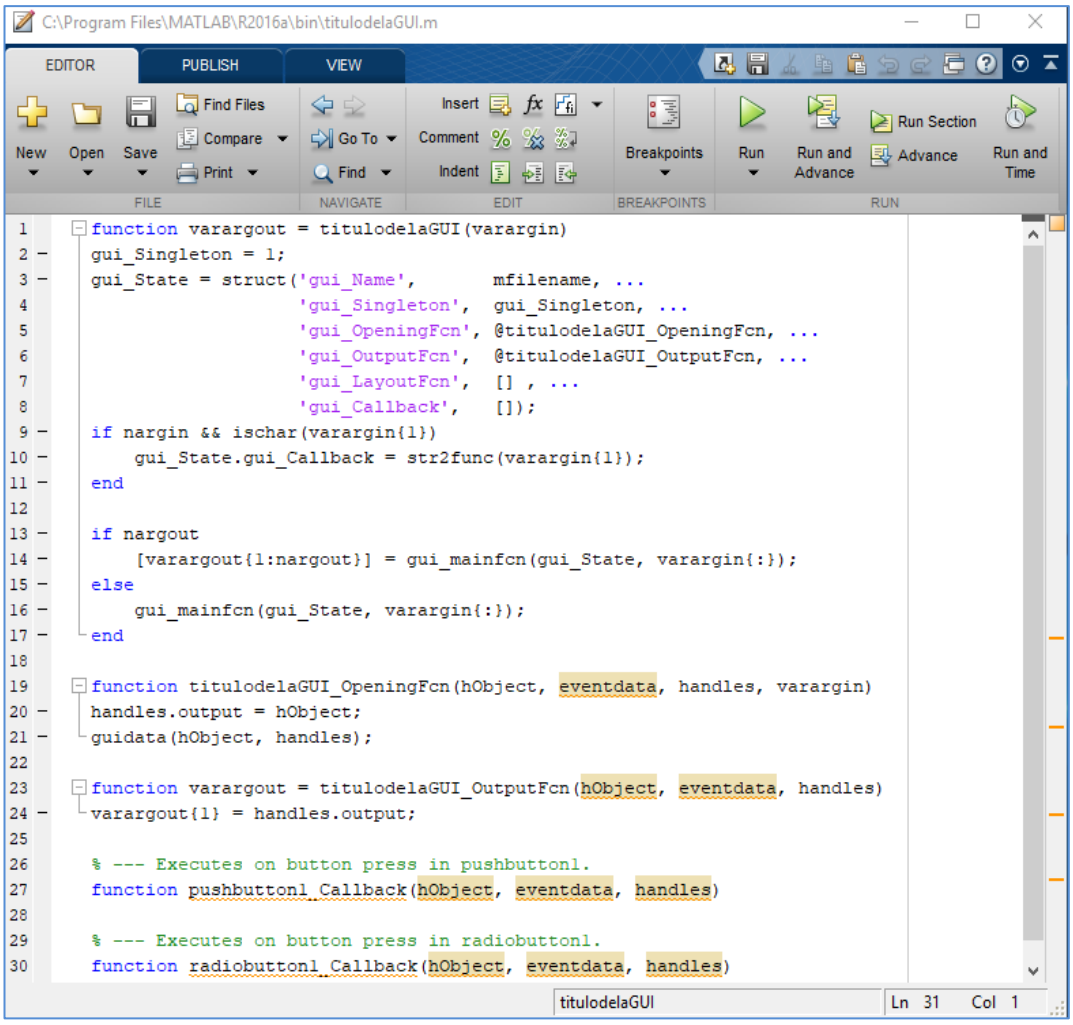


Figura 7. Captura del fichero con las funciones que contiene la GUI.

- **Función titulodelaGUI:** Esta función es la primera en ser llamada, cuando la GUI se abre. Contiene toda la información del entorno GUIDE creado y sus características.
- **Función titulodelaGUI_OpeningFcn:** Aquí se deberá programar lo que se quiera ejecutar en primera estancia, justo al abrir la GUI.
- **Función titulodelaGUI_OutputFcn:** Esta función es llamada cuando el entorno gráfico se cierra. Si queremos que nos devuelva un valor o valores cuando es cerrada, es aquí donde habrá que asignarlo.
- **Función tag_del_objeto_Callback:** Este tipo de funciones son creadas a la misma vez que el objeto gráfico al que corresponden. Así, como lo explicado anteriormente, se ejecutan cuando el usuario interactúa con las figuras.

2.2.2.3 Los handles y las funciones get y set

Como hemos visto en el apartado anterior, la estructura de la GUI está formada por funciones, en este apartado se explicará cómo funciona el flujo de información entre dichas funciones.

Las variables declaradas dentro de una función se las conoce como variables locales, es decir, esas variables solo van a poder ser manipuladas en dicha función y desde fuera no se podrá acceder a ellas. Para que eso pueda ser posible la variable debe ser de tipo global, es ahí donde entran en juego la estructura handles.

La estructura handles controla el flujo de información entre funciones, de modo que almacena variables globales creadas por el usuario y otras creadas automáticamente. Para crear una variable global se accede a handles y se salva la estructura con guidata como se muestra a continuación:

```
>>handles.Nombre_de_la_variable= valor;

>>guidata(hObject, handles);
```

De este modo, en cualquier función se puede acceder a la información de la variable con el comando:

```
>>X=handles.Nombre_de_la_variable;
```

La utilidad de la estructura handles en una GUI va más allá de definir variables, sino que también guarda información sobre cada objeto gráfico creado y permite modificarlos.

Por ejemplo, si necesitamos consultar el estado de un botón de enclavamiento usaremos la función get seguida del nombre del botón.

```
>>get(handles.togglebutton1,'value')
```

Esta función devolverá un uno en caso de que el botón togglebutton1 se encuentre pulsado, y un cero en caso contrario.

Para modificar propiedades de un objeto como el color de un botón, el contenido de un texto, el tamaño de letra de un botón…etc., implementaremos el siguiente comando:

>>set(handles.tag,'propiedad','valor')

3. TRATAMIENTO DE IMAGEN

Toda la información necesaria para la realización de este trabajo se encuentra en la imagen que tomamos del puzle, es por ello la importancia de un correcto tratado de imagen. La calidad de la información que extraigamos de la imagen inicial dependerá de las etapas de procesamiento por las que haya pasado.

Éste capítulo detalla las etapas de procesamiento que se han llevado a cabo para extraer toda la información relevante.

3.1. Adquisición de imagen

Con el fin de capturar la imagen con las piezas que van a ser ensambladas en el puzle se utilizó la cámara principal de un móvil Samsung S5 de 16 megapíxeles (ISOCELL /1/2.6") con enfoque selectivo. Tras varias pruebas de imagen se observó que utilizando imágenes tomadas con la máxima resolución que nos aporta la cámara (16 MP), el procesamiento de ellas se hacía muy lento y se conseguían los mismos resultados que con imágenes de menor resolución. Por lo tanto, se tomó la decisión de reducir significativamente la resolución de imagen a 2.4 megapíxeles para así reducir la carga de memoria y hacer el procesamiento de imagen menos complejo.

Las piezas se colocaron boca arriba sobre un fondo fácilmente segmentable, formado por una plancha verde, con cuidado de no superponer unas con otras y evitando inclinaciones. Las imágenes fueron tomadas desde una posición superior mirando directamente hacia abajo para reducir la distorsión de la perspectiva. Se hizo uso del flash que tiene la cámara para reducir al máximo los sombreados producidos por la iluminación exterior.

Los puzles utilizados en el proyecto fueron creados con cartulina a mano alzada, por lo que no tienen patrones similares entre ellos. Aunque la imagen de los puzles no es de importancia para el ensamblaje, se decidió dibujar sobre en algunos de ellos para facilitar la comprobación de los resultados.

3.2. Escala de grises

La imagen de entrada es una imagen en formato RGB, por lo que cada color aparece descompuesto en sus tres componentes espectrales primarias rojo, verde y azul. Estos componentes aparecen en tres matrices bidimensionales superpuestas en escala de grises. En la escala de grises cada pixel es representado por un único valor del 0 al 255, el cual representa cómo de oscura es la componente espectral, siendo el 255 el nivel de intensidad máximo. Por lo tanto cada pixel de la imagen RGB está formado por la suma, por síntesis aditiva, de las tres componentes espectrales primarias. Si cada pixel tiene 3 Bytes (rojo, verde y azul) y cada uno de ellos 256 tonos, la imagen en RGB tendrá 16 millones de colores posibles.

En este proyecto se ha elegido utilizar una lámina verde como fondo de imagen para facilitar la descomposición de colores (Figura 8 imagen izquierda). Para acceder a las componentes de una imagen RGB se debe especificar la dimensión en la que se encuentra siendo 1 para la componente roja, 2 verde y 3 azul como se muestra en el código siguiente.

```
>>imd=im2double(im); %Doble precisión para restar imágenes
>>imR=imd(:,:,1); %Componente roja
>>imG=imd(:,:,2); %Componente verde
>>imB=imd(:,:,3); %Componente azul
>>imagenverde=imG-imR-imB;
```

Restando la componente roja y azul a la componente verde se obtienen las zonas predominantemente verdes [7] (Figura 8 imagen derecha). Así obtendremos el fondo de imagen y podremos diferenciar las piezas objeto de estudio y el fondo a descartar.

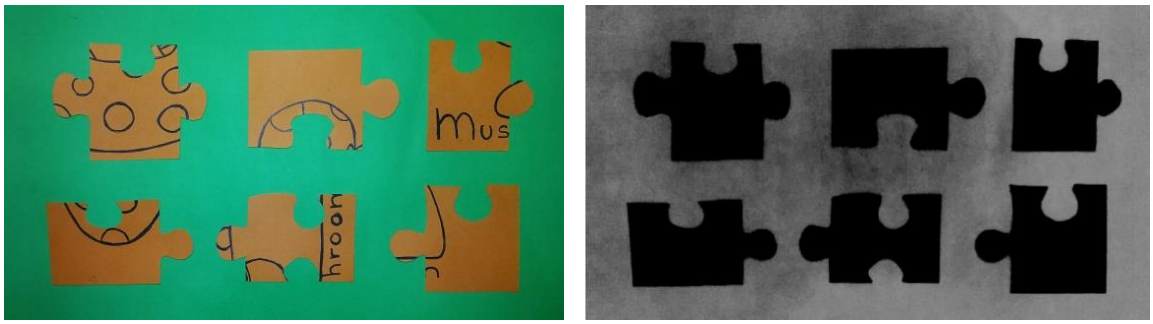


Figura 8. Imagen origen (izquierda) y el resultado de la descomposición (derecha).

3.3. Filtrado

Un proceso necesario después de procesos de captura, digitalización o transmisión y antes de la aplicación de detectores de bordes es la reducción de ruido de una imagen digital. Este proceso tiene como objetivo modificar aquellos píxeles cuyo nivel de intensidad es distinto al de sus vecinos suavizando así las variaciones de intensidad y eliminando efectos espurios que han podido dejar la adquisición. Existen distintos algoritmos capaces de reducir el ruido de una imagen digital. En este proyecto se eligió utilizar un filtro lineal.

Los filtros lineales realizan una operación de convolución entre la imagen dada y una máscara predefinida. El mayor inconveniente de éstas técnicas es el difuminado que se produce en los bordes, por lo que este tipo de filtros debe ser utilizado con precaución.

Podemos encontrar varios tipos de filtros lineales en función del ruido que presente la imagen. El proceso de adquisición de la imagen origen con la que trabajamos puede producir pequeñas variaciones en la imagen debido a las diferencias de ganancias del sensor o el ruido propio de la digitalización. Por este motivo, se decidió utilizar un filtro gaussiano. Utilizando este filtro, el valor de cada punto de la imagen filtrada es el resultado de haber promediado con distintos pesos los valores vecinos a ambos lados del punto. De esta forma las variaciones rápidas de intensidad son suavizadas y reemplazadas por una transición más gradual. La función de convolución que realiza este filtro se aproxima a la discretización de una gaussiana de media cero y varianza sigma (2).

$$h(u,v) = \frac{1}{\sqrt{2\pi}\cdot\sigma} \cdot e^{\frac{u^2+v^2}{2\sigma^2}} \qquad (2)$$

Donde x=Distancia desde el origen en el eje horizontal
y=Distancia desde el origen en el eje vertical
σ= Desviación estandar de la distribución Gaussiana

La imagen siguiente es el resultado de un filtrado gaussiano, y se observa que respecto a la etapa anterior muestra una mejora suavizando los bordes y reduciendo las manchas.

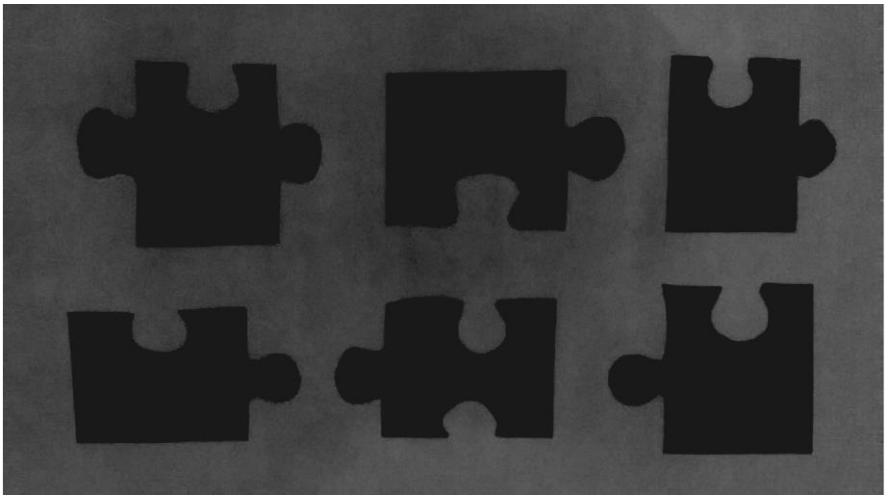


Figura 9. Resultado tras aplicar filtro gaussiano.

3.4. Segmentación

El proceso de segmentación divide la imagen en zonas o regiones homogéneas a partir de su contorno, conectividad o características de los píxeles. La umbralización es una de los métodos más utilizados en segmentación de imágenes digitales. Las técnicas de umbralización persiguen obtener un valor representativo llamado umbral que permita separar una imagen diferenciando el fondo (background) y el objeto (foreground). El resultado de esta separación es una imagen binaria en la que solo persisten dos valores: verdadero y falso. En una imagen digital estos valores están representados por 0 y 1, o negro y blanco.

La mayoría de las técnicas de umbralización están basadas en la información que se obtiene del histograma de la imagen digital. El histograma muestra los píxeles y niveles de gris que se encuentran en la imagen, por lo que en un histograma con dos niveles de gris bien marcados, referentes al objeto y fondo, el umbral óptico sería aquel valor que separa ambas regiones.

Tras las etapas anteriores la imagen obtenida es una imagen en escala de grises con el fondo y el objeto bien diferenciado, pues los objetos aparecen

con niveles de intensidadades negativos (computacionalmente considerados 0) como resultado de la resta de las componentes de rolor rojo y azul. En este caso, es facil determinar que el umbral se encuentra cerca del valor de intensdiad 0.

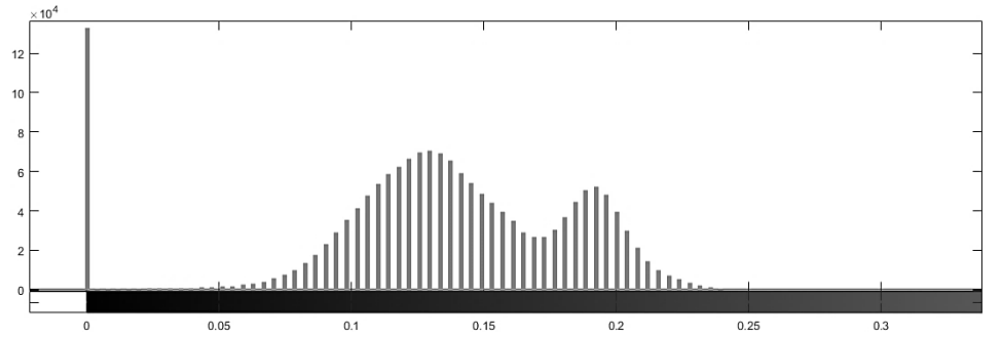


Figura 10. Histograma de la imagen puzle en escala de grises.

Comos se observa en el histograma de la Figura 10, la mayoría de los píxeles tienen valor 0, es decir, son parte del objeto. Por otro lado, a distintos niveles de gris se encuentra el fondo. El umbral de separación entre los dos, conforme a lo previsto, se encuentra cerca de 0.

Si aplicamos la función graythresh en matlab a la imagen en escala de grises obtenemos el valor del umbral más optimo según el método de Otsu. Este método utiliza técnicas estadisticas basadas en la varianza de la dispersión de niveles de gris.

Tras obtener el valor del umbral, este se compara pixel a pixel con el nivel de la imagen en escala de grises, siendo 1 si el valor del pixel supera al umbral y 0 en caso contrario.

$$\text{imagen\_binaria}(x,y) = \begin{cases} 1, & \text{imagen\_gris}(x,y) > U \\ 0, & \text{imagen\_gris}(x,y) \leq U \end{cases} \qquad (3)$$

Donde x=Distancia desde el origen en el eje horizontal
y=Distancia desde el origen en el eje vertical
U= Valor del umbral
imagen_gris=Imagen en escala de grises

El resultado es una imagen binaria con el fondo blanco y las piezas del puzle en negro.

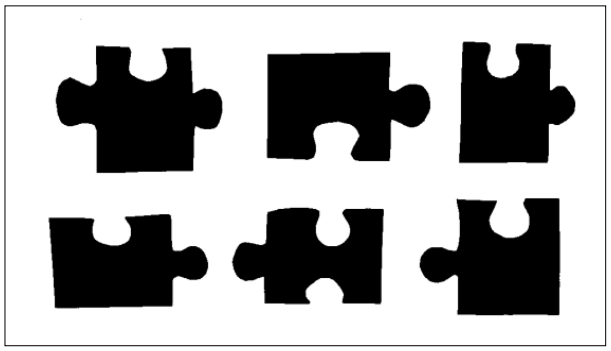


Figura 11. Resultado de la segmentación

3.5. Procesamiento morfológico.

La segmentación puede dar lugar a falsos pixeles en la delineación de los objetos debido a una mala clasificación, bordes imprecisos o regiones solapadas. Es por ello que se requiere de un procesamiento posterior para corregir los posibles pixeles erróneos. En esta fase se emplea un procesamiento morfológico formado por operaciones de erosión y dilatación.

Las transformaciones morfológicas tienen como objeto la extracción de formas geométricas en los conjuntos sobre los que se aplica, mediante la utilización de otro conjunto denominado elemento estructurante. El elemento estructurante es elegido en función de la morfología sobre la que va a ser aplicado. En la siguiente imagen aparecen algunos de los elementos estructurantes disponibles en Matlab.

Figura 12. Distintos tipos de elementos estructurantes

El proceso de erosión es el resultado de comprobar si dicho elemento estructurante B está completamente incluido dentro del conjunto X. Si todos los

pixeles vecinos al pixel de estudio pertenecen al objeto, entonces el pixel de estudio también pertenece al objeto. Si no ocurre, el resultado de la erosión es un conjunto vacío.

En las siguientes líneas de código muestra cómo aplicar a la imagen binarizada del proceso anterior una operación de erosión con un cuadrado de 25 píxeles de lado como elemento estructurante. Estas funciones trabajan sobre objetos blancos y fondos negros, por lo que la imagen binaria calculada anteriormente debe ser negada.

```
>>SE = strel('square',25);
>>bw=imerode(not(bw),SE);
```

El resultado es una imagen en la que los objetos menores al objeto estructurante desaparecen (Figura 13).

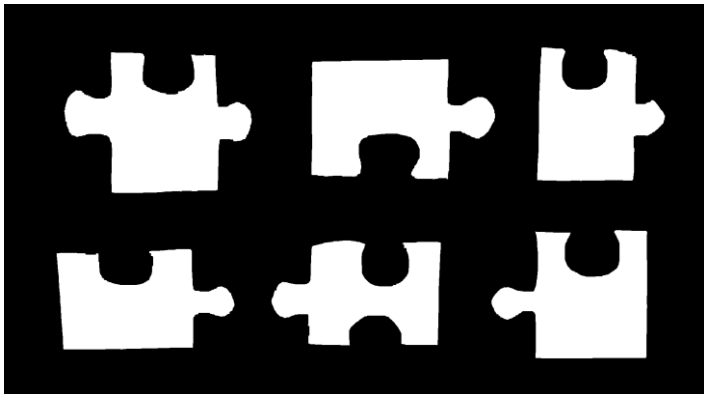


Figura 13. Resultado del proceso de erosión.

El proceso de dilatación es una transformación dual a la de erosión. El resultado de este es el conjunto de elementos tal que algún elemento del conjunto estructurante está contenido en el conjunto X. Si alguno de los pixeles de entorno del pixel en estudio pertenece al objeto, entontes del pixel de estudio también lo hará.

La implementación en Matlab aplicando el mismo elemento estructurante para realizar la dilatación es el siguiente.

```
>>SE = strel('square',25);
>>bw=imdilate(not(bw),SE);
```

Cuyo resultado es una imagen siguiente (Figura 14) donde se observan los objetos dilatados y las grietas cerradas.

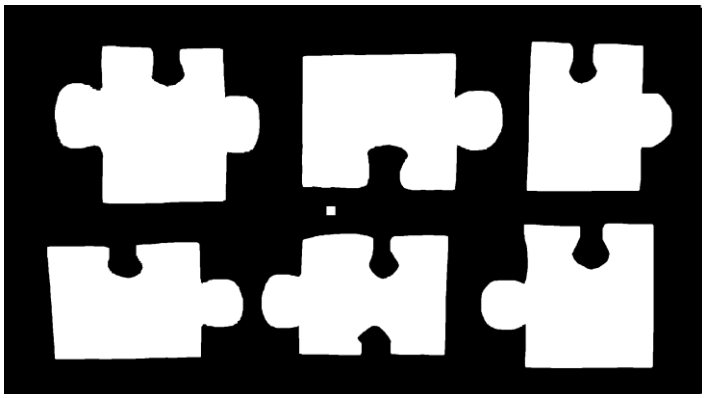


Figura 14. Resultado del proceso de dilatación.

Ambos procesos morfológicos modifican el tamaño de los objetos. La erosión puede eliminar pequeños objetos indeseados, sin embargo tiene el inconveniente de disminuir el tamaño del resto. Sin embargo, este efecto puede ser arreglado si se aplicaban sucesivamente operaciones de erosión y cierre, a esto se le conoce como opening o apertura binaria. De esta manera, se eliminan todos los objetos que no están contenidos en la estructura y, además, no modifica el tamaño de los que la superen.

Por otro lado, los resultados de la dilatación muestran objetos engrandados a los que se les han cerrado agujeros y grietas. Como ocurre en la operación anterior, el ensanchamiento de los objetos puede ser solucionado con la combinación de operaciones de dilatación seguidas de erosión, closing o cierre binario. El cierre binario corrige las gritas que la erosión no puede rellenar sin modificar el tamaño de los objetos.

La implementación de estas funciones en Matlab se muestra a continuación.

```
>>bw   = bwmorph(not(bw),'open');
>>bw   = bwmorph(bw,'close');
>>bw   = imfill(bw,'holes');
```

La función imfill rellena los huecos que hayan podido quedar tras el procesamiento morfológico. Finalmente, el resultado muestra los objetos sin grietas, con bordes más uniformes y conservando su tamaño original.

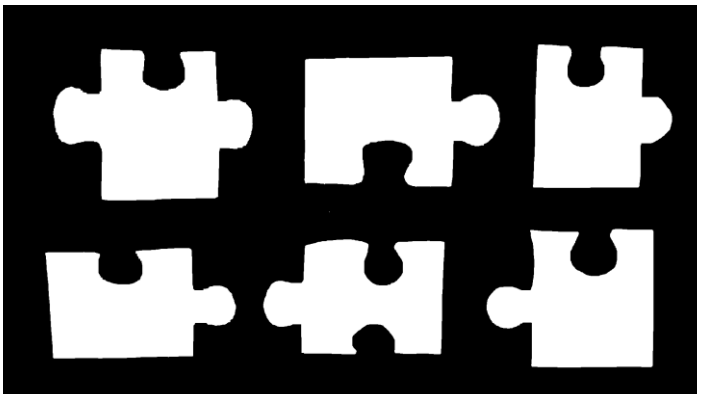


Figura 15. Resultado del proceso de apertura y cierre.

Multiplicando la máscara creada con la imagen origen se obtiene una imagen de las piezas a color y el fondo eliminado que será de utilidad en el proceso de acoplamiento de piezas. El resultado puede verse en la Figura 16.

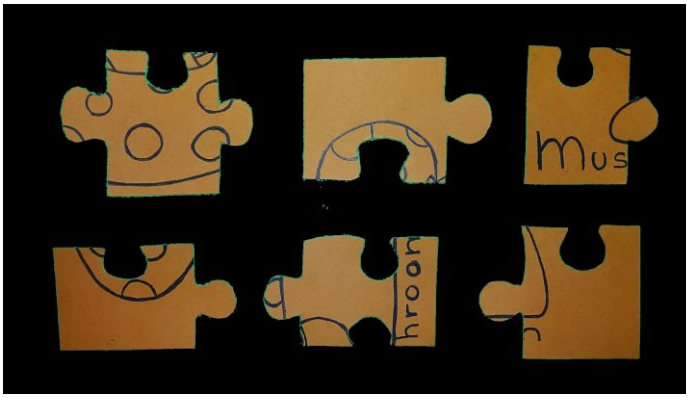


Figura 16. Eliminación del fondo.

3.6. Detección de bordes

Un paso previo a la extracción de características es la detección de bordes. En este proceso se buscan las transiciones entre dos niveles de gris significativamente distintos. La detección de bordes es una técnica muy útil para obtener información de la frontera de los objetos.

Uno de los algoritmos más utilizados para la detección de bordes es el algoritmo de Canny, este algoritmo utiliza la primera derivada debido a que esta toma valores de cero en todas las regiones donde no varía la intensidad y tiene un valor constante en toda la transición de intensidad. De esta manera, un cambio de intensidad se manifiesta como cambio brusco en la primera

derivada. Esta característica es utilizada para detectar bordes y es por lo tanto en la que se basa el algoritmo de Canny.

El algoritmo de Canny consiste en cuatro etapas:

1. Obtención del gradiente: Para la obtención del gradiente se aplica un filtro gaussiano a la imagen original con el objetivo de suavizar la imagen y eliminar posibles ruidos. Una vez se suaviza la imagen se calcula la magnitud y módulo del gradiente de cada pixel.
2. Supresión no máxima al resultado del gradiente: En este paso se logra adelgazar los bordes obtenidos con el gradiente, hasta alcanzar un píxel de anchura.
3. Histéresis del umbral a la supresión no máxima: En esta etapa se aplica una función de histéresis con dos umbrales. De este modo se pretende reducir los posibles contornos falsos.
4. Cerrar contornos: El objetivo de la última etapa consiste en cerrar los contornos que pudiesen haber quedado abiertos. Para cada pixel se busca un posible patrón de los ocho posibles que delimitan el entorno, si alguno de los píxeles es ya un pixel de borde, se entiende que el borde se ha cerrado.

El algoritmo de Canny está implementado en Matlab utilizando la función edge como muestra el siguiente código.

```
>>thresh = umbral;
>>sigma = 5;
>>E = edge(bw,'canny',thresh,sigma);
```

La función edge binariza la imagen antes de ser procesada en función al valor del parámetro thresh. En nuestro caso, la imagen ya ha sido umbralizada por lo que el parámetro es el calculado con anterioridad. El valor de sigma indica la desviación estándar que seguirá el suavizado Gaussiano. Se determinó que con sigma=5 se obtenían resultados satisfactorios para este proyecto.

El resultado tras aplicar la detección de bordes se muestra en la Figura 17.

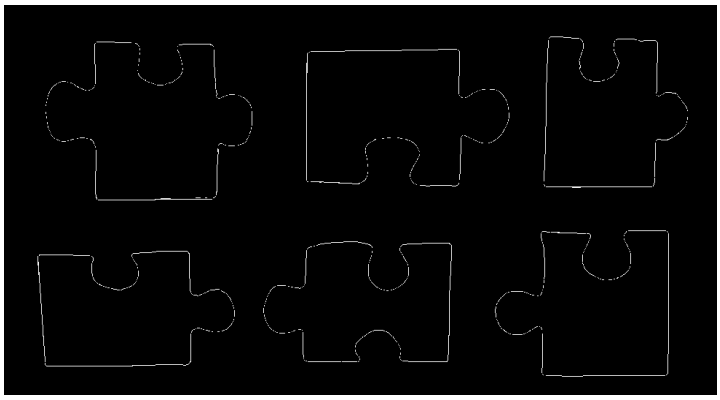


Figura 17. Detección de bordes Canny.

3.7. Extracción de características

En los apartados siguientes se explican los pasos que se han seguido para extraer y codificar en contorno de cada pieza. La obtención de los contornos dará lugar a la búsqueda de esquinas con el fin de diferenciar los lados de las piezas. Las esquinas serán localizadas mediante las coordenadas polares definidas por el contorno, de esta manera se podrán diferenciar los cuatro lados y posteriormente clasificarlos. Las etapas principales para conseguir el objetivo son las siguientes:

- Localización y separación de piezas
- Detección de contorno y conversión a coordenadas polares
- Detección de máximos locales y esquinas
- Detección de orientación y rectificación
- Separación y clasificación de lados

3.7.1. Localización y separación de piezas

Para la extracción del contorno de las piezas se deben procesar individualmente. Para conocer la cantidad de objetos que se encuentran en la imagen se utiliza la función contenida en Matlab bwconncomp, esta función devuelve el número de componentes conectados en una imagen binaria. Además, aporta información de la conectividad y los índices lineales de los píxeles de cada objeto (Figura 18).

```
K>> CC=bwconncomp(E)

CC =

    Connectivity: 8
       ImageSize: [1152 2048]
      NumObjects: 6
    PixelIdxList: {[1950x1 double]  [2218x1 double]  [2231x1 double]
    [2261x1 double]  [2321x1 double]  [1928x1 double]}
```

Figura 18. Información obtenida con bwconncomp.

Las propiedades de los objetos se obtienen con regionprops, esta función devuelve las propiedades que se especifiquen de cada uno de los elementos. En este proyecto, se han considerado necesarias propiedades como el área, centroide y boundingbox. Esta última propiedad devuelve los lados de un rectángulo que envuelve la región del objeto y será útil para aislar cada objeto individualmente. Los datos de la función son devueltos en una estructura de tres campos, uno por cada propiedad especificada, que incluyen los datos de cada uno de los elementos encontrados como se observa en la Tabla 4.

Fields	Area	Centroid	BoundingBox
1	1950	[403.3913 863.7318]	[152.5000 709.5000 542 317]
2	2218	[462.3305 340.8161]	[174.5000 137.5000 568 435]
3	2231	[1.0659e+03 851.7669]	[773.5000 684.5000 517 330]
4	2261	[1.1425e+03 348.3485]	[890.5000 158.5000 558 375]
5	2321	[1.6910e+03 848.9345]	[1.4105e+03 654.5000 477 398]
6	1928	[1.7170e+03 325.5654]	[1.5425e+03 124.5000 397 413]

Tabla 4. Propiedades de los objetos encontrados.

La implementación en Matlab queda como muestra el siguiente código.

```
>>CC=bwconncomp(E);
>>r  = regionprops(E,'Area','Centroid','BoundingBox');
>>s=find([r.Area]<500); %Áreas menores de 500 eliminadas
```

Las áreas menores de 500 píxeles son eliminadas de los objetos de interés para desechar posibles defectos que puedan no haberse solucionado con las etapas anteriores. Las distintas piezas son separadas recortándolas por los límites del rectángulo obtenido con el algoritmo mencionado como se muestra en la Figura 19.

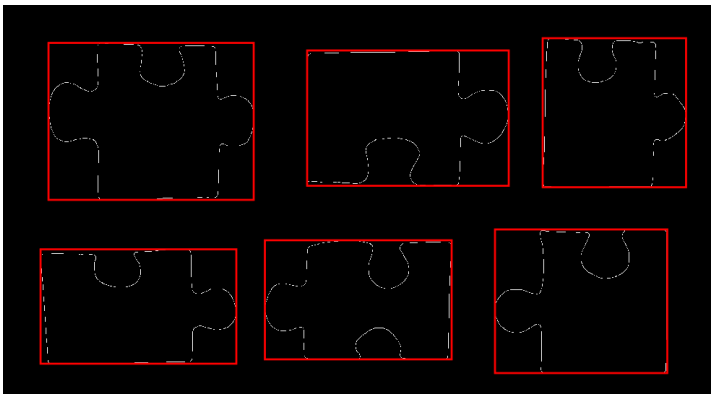


Figura 19. Rectángulo para separar cada pieza.

3.7.2. Detección de contorno y conversión a coordenadas polares

Una vez desglosadas las piezas del puzle, el primer paso es localizar las esquinas para separar los lados. Conforme a los supuestos sobre las piezas de puzle utilizadas, es asumido que cada pieza de puzle es rectangular y por lo tanto, forman un alguno entre esquinas de 90 grados. Esta propiedad se puede explotar para localizar los puntos que son esquina y descartar los que no lo son. Por lo tanto, el método utilizado para localizar las esquinas en función de su fase consiste en la conversión de coordenadas cartesianas a coordenadas polares del contorno de las piezas.

En primer lugar se obtienen las coordenadas cartesianas del contorno. Dichas coordenadas están formadas por los píxeles que pertenecen a la frontera de las piezas. Para hallar el conjunto de píxeles se utilizó en Matlab la función bwboundaries. Esta función devuelve una matriz de dos columnas para las coordenadas 'X' e 'Y', y tantas filas como píxeles tenga el contorno. Para una correcta descripción de los límites se utilizó la versión de conectividad 8 que siguen los códigos de cadena de Freeman. Esta conectividad persigue que dos píxeles contiguos forman parte del mismo objeto si ambos están conectados a lo largo de la dirección horizontal, vertical o diagonal. A diferencia de la conectividad 4 que solo considera píxeles conectados los que se encuentran en dirección horizontal o vertical. (Figura 20)

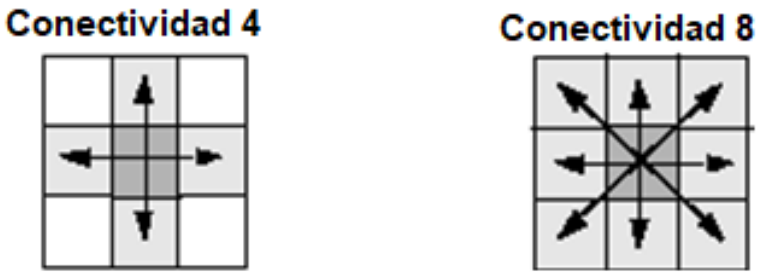


Figura 20. Tipos de conectividad.

La conversión de coordenadas cartesianas a coordenadas polares se lleva a cabo mediante la función cart2pol(x,y) implementada en Matlab. El resultado son dos vectores theta y rho. Theta corresponde al ángulo formado en el sentido contrario a las agujas del reloj en el plano x-y medido en radianes y rho contiene los valores de las distancias desde el origen hasta el punto en concreto. Todos los puntos quedan representados por su distancia al origen y ángulo con el plano x-y.

La implementación correcta en Matlab para el trazo del contorno y su conversión a coordenadas polares:

```
%Trazo del contorno del objeto solo por fuera (noholes)
>>B=boundaries(im,8,'noholes');
>>B=B{1};
>>BW1 = bound2im(B, max(B(:,1)), max(B(:,2)),
min(B(:,1)), >>min(B(:,2)));
%A cada valor de B se le resta la media
>>X = bsxfun(@minus, B, mean(B));
%Cartesianas a polares
>>[theta,rho] = cart2pol(X(:,1),X(:,2));
```

El resultado para una pieza del puzle es el siguiente:

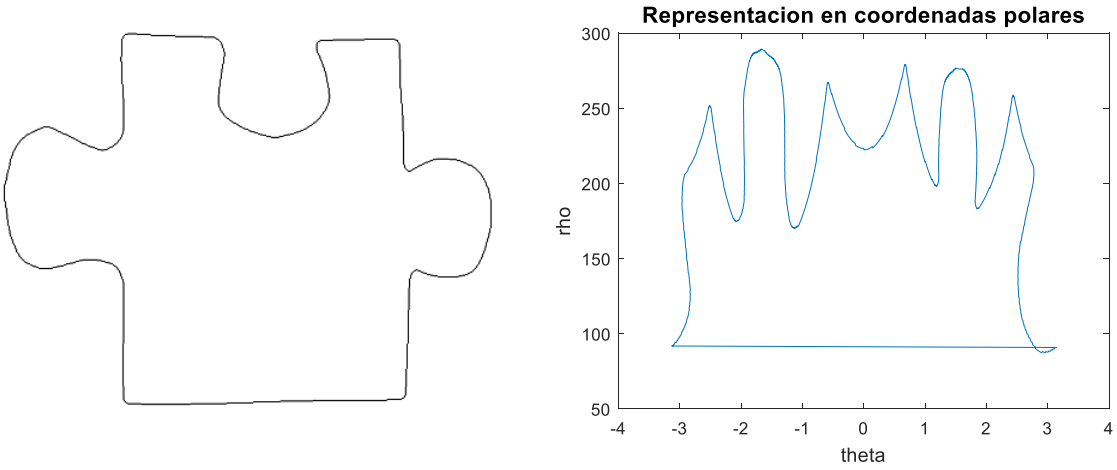


Figura 21. Representación del contorno en coordenadas polares.

Como se muestra en la Figura 21, existen puntos representados en coordenadas polares que representan máximos locales.

3.7.3. Detección de máximos locales y esquinas.

Conforme a lo avanzado en el apartado anterior, existe un ángulo de 90 grados aproximadamente entre las esquinas de las piezas. Es ésta característica la que determinará qué puntos son considerados esquinas.

Para que la labor de búsqueda de esquinas sea lo más rápida posible, se consideran solo los puntos más dominantes, siendo estos los máximos locales de las coordenadas que forman el contorno. Los máximos locales están formados por los puntos que sean mayores que sus dos puntos vecinos. Antes de la búsqueda de picos, será necesario realizar una convolución para suavizar algunos de los picos que se encuentran muy próximos.

En Matlab, existe una función para la búsqueda de máximos locales. Dicha función admite como entrada un valor de separación mínima entre puntos. Cuando se especifica un valor entero de muestras mínimo el algoritmo busca el pico más alto de las coordenadas e ignora todos los picos que se encuentren dentro del valor especificado para la distancia mínima. Se observó, que una media de 80 puntos unían las esquinas. Por lo que se utilizó ese valor como dato para eliminar máximos que no se encuentren en ese rango.

La implementación correcta en Matlab para encontrar máximos locales con una distancia mínima es la siguiente:

```
>>[pks,pos] = findpeaks(datos,'MinPeakDistance',80);
```

Devuelve los picos donde se dan los máximos locales (pks), y adicionalmente, los índices a los que corresponden (pos). Buscando los máximos encontraremos puntos que pertenecen tanto a las esquinas como a los bolsillos u orificios de las piezas, por lo que obtendremos picos erróneos que no forman parte de las esquinas.

Para desechar los resultados erróneos, los máximos encontrados son analizados con el fin de encontrar los cuatro picos que formen entre si un ángulo de 90º. A continuación se muestra el contorno de una pieza (Figura 22) y su diagrama de coordenadas polares con los puntos considerados máximos locales marcados con una cruz roja y los puntos reconocidos como esquinas con un círculo en verde.

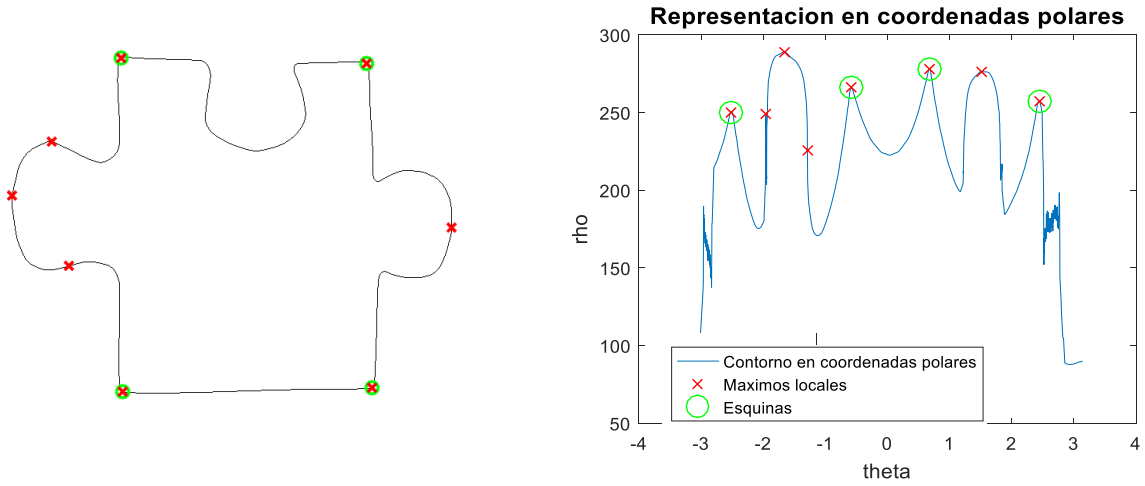


Figura 22. Detección de esquinas y máximos locales.

3.7.4. Separación y clasificación de lados

La búsqueda de esquinas es un paso importante para poder separar los píxeles en función del lado al que pertenecen. Una vez determinados los puntos que son esquina podemos separar el vector que contiene los pixeles del contorno para clasificar el tipo de lado.

El método utilizado para la clasificación de lados está basado en tomar las distancias y ángulos que existen entre las esquinas y los píxeles que se

encuentran entre ellas, así como las distancias de los píxeles interiores al centro de gravedad.

Como en el proceso anterior, no todos los píxeles que forman el lado son tomados para la clasificación. Se realiza una segunda búsqueda de máximos locales, esta vez sin especificar una distancia mínima. De esta manera aparecen todos los puntos que forman parte de los lados, que son máximos locales y por lo tanto, picos de interés. Estos serán los puntos que serán de utilidad para extraer información válida sobre el tipo de lado.

La Figura 23 muestra todos los máximos locales encontrados sin especificaciones de distancia.

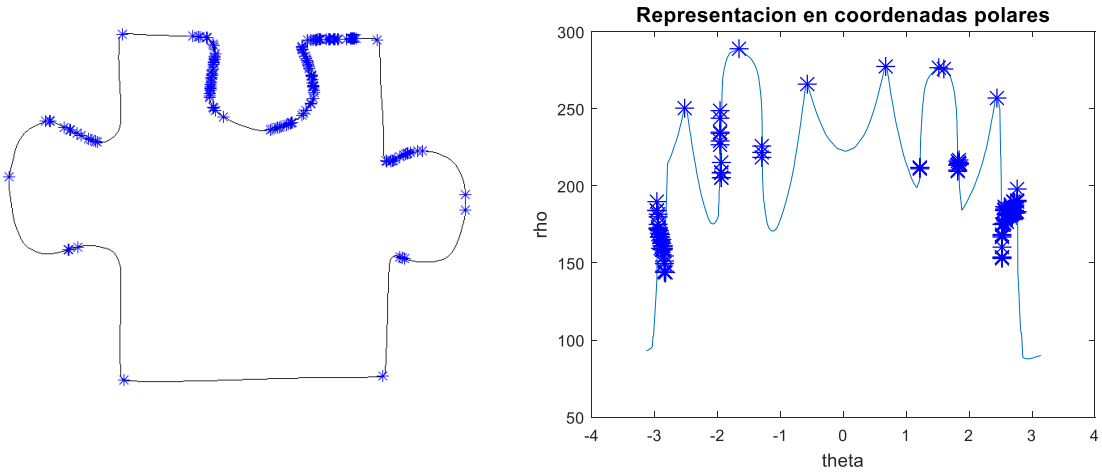


Figura 23. Detección de todos los máximos locales.

Como se puede comprobar en la figura anterior, entre esquina y esquina existen máximos locales que forman parte de la cabeza u orificio de la pieza, y que no existen máximos en el lado que es recto. Esa será una de las principales características para clasificar los lados rectos.

Conforme a lo observado, los lados rectos quedan caracterizados porque entre sus esquinas no tienen máximos locales, y si los hubiera, éstos están alineados con alguna de las esquinas. Para calcular la alineación se toma el ángulo que forma con la horizontal la recta que une el punto y la esquina. Si este ángulo es de 90º o 180º, en función del lado que estemos estudiando, serán puntos alineados, y por lo tanto el lado es recto.

Para la clasificación de cabeza u orificio, se recorren los máximos locales calculados en sentido contrario a las agujas del reloj empezando por la primera esquina que será la esquina superior izquierda. A cada punto se le calcularán dos distancias que serán de utilidad y permitirán obtener qué tipo de lado es. Las distancias que se consideran son las siguientes:

1. Distancia al centro del lado

Esta distancia es calculada para elegir el punto que más centrado se encuentre entre las dos esquinas, que será el punto más representativo. El primer paso es calcular la distancia media entre ambas esquinas, y finalmente, la distancia del punto a una de las esquinas. Si la diferencia entre ambas distancias está dentro de un rango de aceptación, el punto será considerado representativo, de lo contrario, será rechazado. Las distancias serán tomadas en el eje paralelo al lado en el que se encuentre.

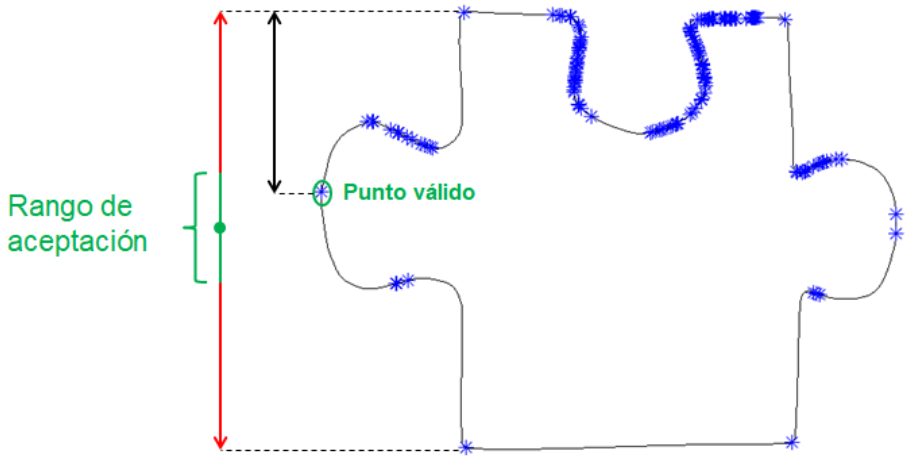


Figura 24. Distancia al centro del lado.

2. Distancia al centroide

La distancia al centroide indicará el tipo de lado al que corresponde. Para calcularla, tomaremos el punto del centroide obtenido anteriormente y calcularemos la distancia que hay hasta él, en el eje perpendicular al lado en el que se encuentra. A su vez tomaremos la misma distancia pero desde la esquina. Si la distancia del punto (d1) es mucho mayor a la de la esquina (d2) sabremos que en ese lado existe una cabeza, por el contrario habrá un orificio. La figura 25 muestra un ejemplo de detección de cabeza.

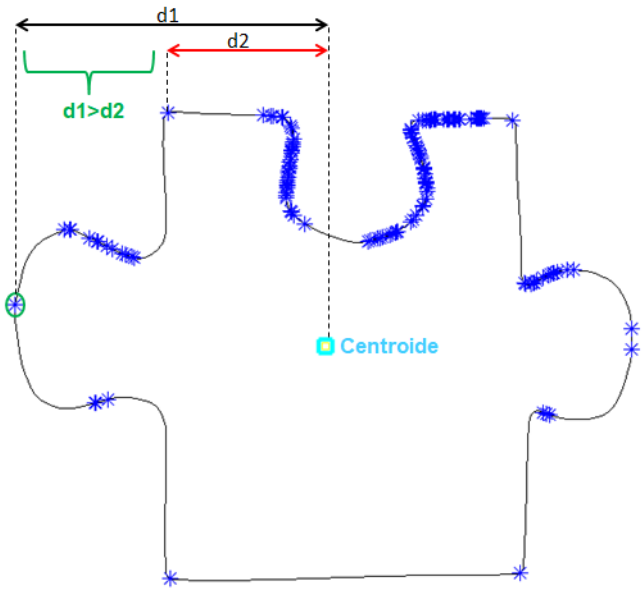


Figura 25. Distancia al centroide.

El siguiente diagrama de flujo de la Figura 26 muestra el proceso que se sigue cuando se toman las coordenadas de un lado para su clasificación.

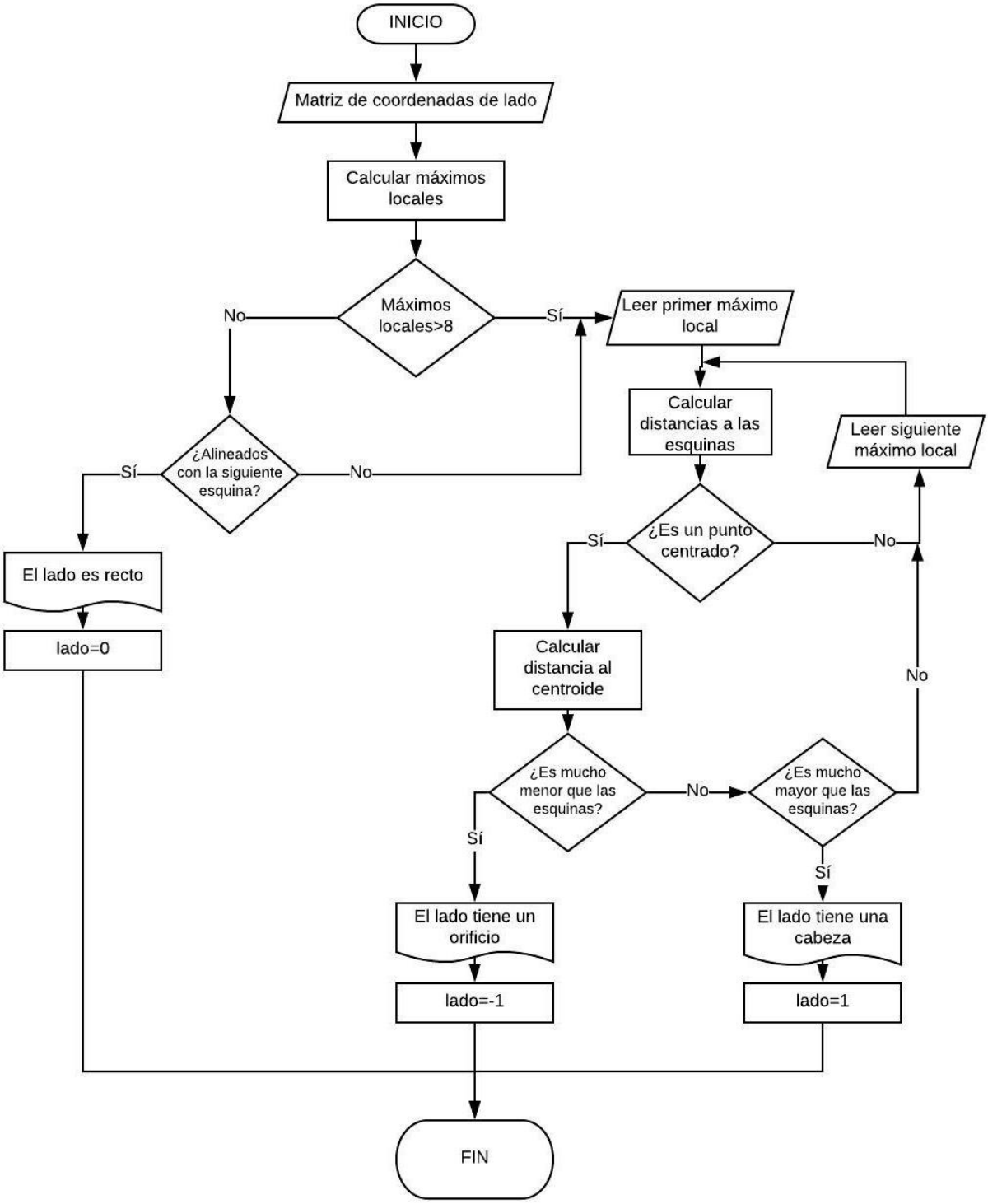


Figura 26. Diagrama de flujo para clasificar los lados.

Una vez clasificados los lados, obtendremos un vector de 4 números por cada pieza correspondientes a los lados recorridos en sentido contrario a las agujas del reloj. Estos números serán 0, si el lado es recto, 1 si tiene una cabeza y -1, si el lado tiene un orificio como en la siguiente Figura 27.

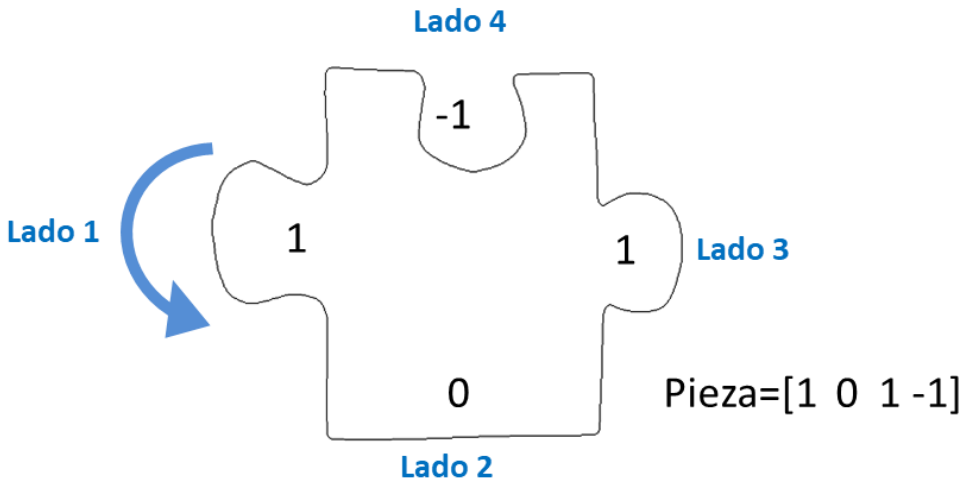


Figura 27. Clasificación de lados.

En total obtendremos una matriz de 6 filas (una por cada pieza de puzle) y 4 columnas (una por cada lado). Siendo la primera fila la correspondiente a la pieza procesada en primer lugar, esta es la que más cerca del eje vertical izquierdo se encuentre en la imagen origen. De esta manera, la fila 6 corresponderá a la pieza más alejada de dicho eje. Como ejemplo de esto, la Figura 28 muestra la imagen origen tratada en los procesos anteriores y la matriz generada finalmente. Dicha matriz será utilizada en el siguiente capítulo para el ensamblaje del puzle.

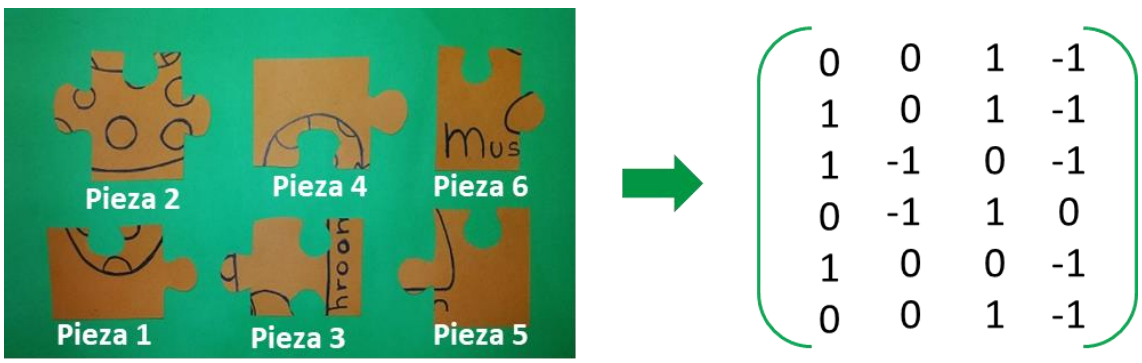


Figura 28. Matriz de piezas y lados generada.

4. ENSAMBLAJE DE PUZLE

El ensamblaje del puzle es el proceso final por el que pasan las piezas ya clasificadas en función de sus lados (recto, cabeza u orificio) para dar una única solución al puzle. La solución viene dada por la unión de las piezas que encajan a la perfección.

En el presente capítulo se detallará el proceso de ensamblaje propuesto para dar solución al puzle y cómo las imágenes de cada pieza son fusionadas para poder mostrar una imagen final con el resultado.

4.1 Algoritmo de ensamblaje

Una vez obtenida la matriz de piezas clasificadas por sus lados, el proceso de ensamblaje es llevado a cabo por un algoritmo propuesto que dará solución a dicha matriz.

En primer lugar, el algoritmo recibe la matriz de piezas y lados como el de la Figura 28 y lo divide en piezas que formen parte de las esquinas o de los lados. Al ser un puzle de 6 piezas, no tiene piezas centrales. Las piezas de las esquinas serán aquellas que tienen 2 lados rectos, por lo tanto la fila que tenga dos 0 será una pieza de esquina. Por otro lado, los lados serán aquellos que tengan un lado recto entre sus lados. Se obtienen por tanto, dos matrices, una de esquinas y otra, de lados.

El ensamblaje del puzle es llevado a cabo mediante tres divisiones a las que se le denominarán segmentos. Así, la resolución del puzle empezará por ensamblar cada segmento por separado para unir los tres segmentos finales. Los segmentos pertenecen a las columnas formadas en una matriz de 2x3 cuyas filas pertenecen a las piezas de puzle que se muestran en la siguiente figura (Figura 29).

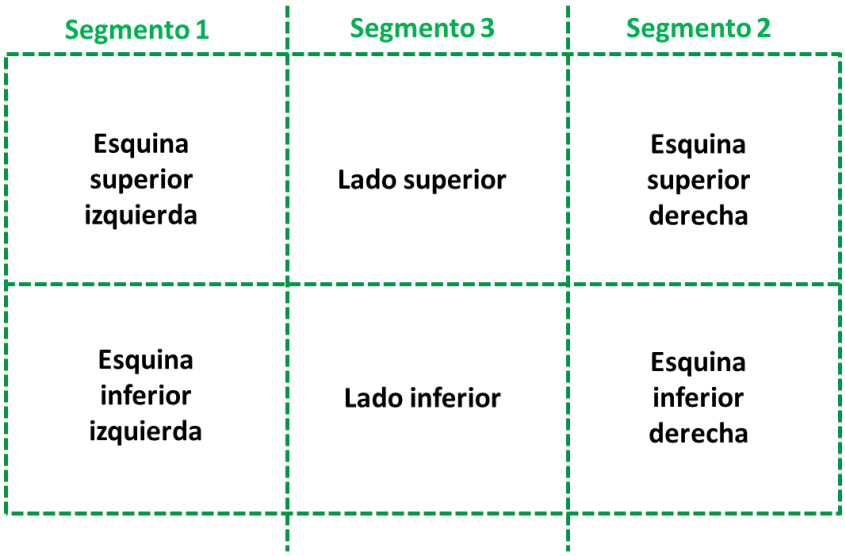


Figura 29. Segmentos de ensamblaje

En primer lugar, se resuelven los segmentos 1 y 2, y teniendo en cuenta los resultados obtenidos para ambos segmentos se resuelve el segmento 3.

I. Segmento 1.

La resolución del segmento 1 empieza por buscar la esquina superior izquierda, como las piezas que tenemos en la imagen no están en su orientación correcta, esta esquina a priori, puede ser cualquiera. Por lo que se optará por un proceso iterativo hasta que la esquina escogida de paso a la solución final.

La esquina superior izquierda debe cumplir que los lados que se encuentran en la posición 1 y 4 sean lados rectos. Los otros dos lados se guardarán en dos variables, el lado 2 será de enlace para el segundo elemento del segmento 1 (lado1_2). Por lo tanto, la primera pieza correspondiente a una esquina será girada hasta hacerla coincidir con el vector clave [0 x x 0]. Donde x es cualquier valor de lado (1 o -1).

Del mismo modo, la esquina inferior izquierda cumplirá que los lados rectos se encuentren en las posiciones 1 y 2. El vector clave para esta esquina es el que enlace con la esquina calculada anteriormente. Este enlace será correcto si ambos lados de unión tienen valores opuestos, cabeza (1) con orificio (-1). De manera que el vector clave [0 0 x opuesto(lado1_2)]. El resto

de esquinas deben ser giradas para comprobar cuál de ellas cumple dicha cadena.

La Figura 30 muestra el resultado del segmento 1. En rojo aparecen los requisitos impuestos por el tipo de esquina.

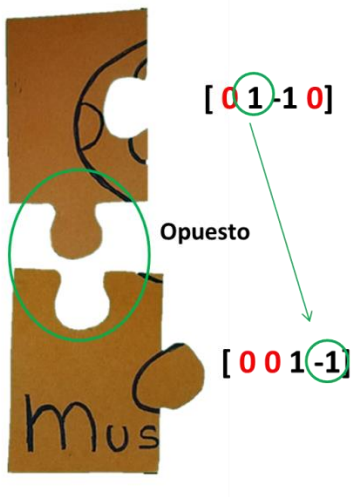


Figura 30. Requisitos segmento 1.

II. Segmento 2

La resolución del segmento 2 seguirá el mismo método propuesto para el segmento 1. En este caso el vector clave que se debe perseguir para la esquina superior derecha debe tener sus lados rectos en las posiciones 3 y 4 esto es [x x 0 0]. El nexo de unión con el segundo elemento del segmento será el lado 2 (lado2_2) como se muestra en la Figura 31.

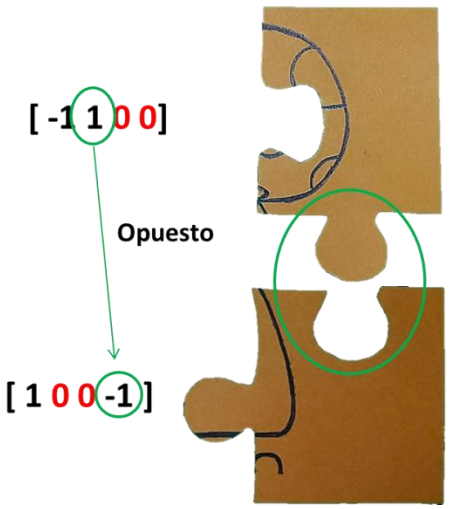


Figura 31. Requisitos segmento 2.

Como se observa en la figura, la esquina inferior derecha, debe cumplir que el lado 4 enlace con la esquina superior y que sus lados rectos estén en las posiciones 2 y 3, quedando el vector clave [x 0 0 opuesto(lado2_2)]. Si la esquina restante no encaja con la descripción, se vuelve a crear el segmento 1 pero esta vez se partirá de la siguiente esquina encontrada.

III. Segmento 3

Finalmente, el segmento 3 al ser el último, debe cumplir más especificaciones y no tiene tantos grados de libertad. Así pues, el lado superior debe cumplir los dos enlaces con los segmentos 1 y 2 y la condición de lado. Mientras que el lado inferior queda determinado por las condiciones impuestas. Si no las cumple, se vuelve a las etapas anteriores para partir de soluciones distintas.

Por lo tanto, la relación entre los segmentos queda como muestra la siguiente imagen.

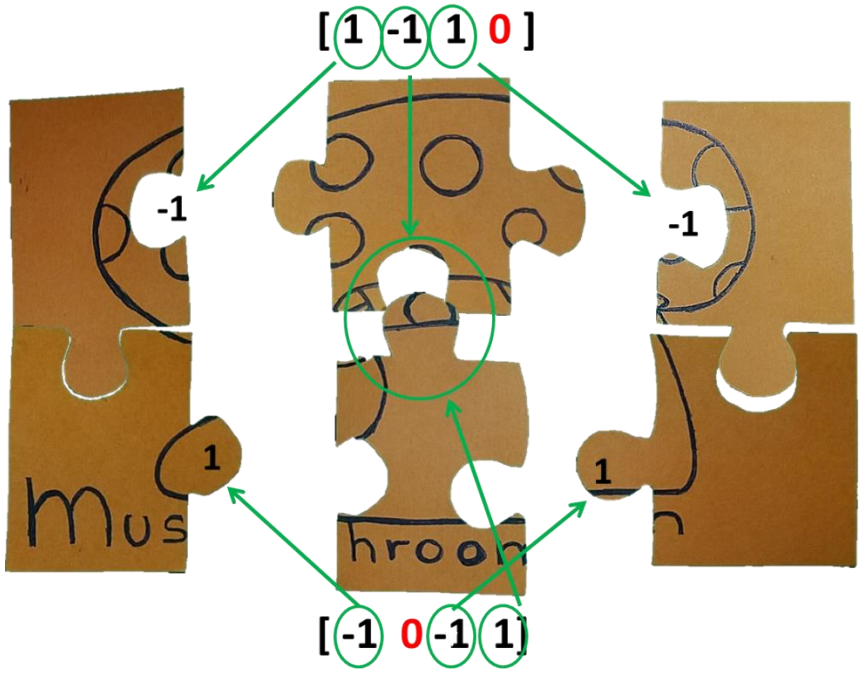


Figura 32. Relación de segmentos 1, 2 y 3.

El resultado del proceso de ensamblaje son dos matrices de 2x3. Una de ellas corresponde a las posiciones que ocupan las piezas en el puzle, de modo que en la posición (1,1) de la matriz se encuentre el número de la pieza que

corresponde a la esquina superior izquierda. La segunda matriz contiene los giros de 90º que tienen que dar dichas piezas en sentido contrario a las agujas del reloj para ser colocadas correctamente.

Un ejemplo de resultado final sería el siguiente:

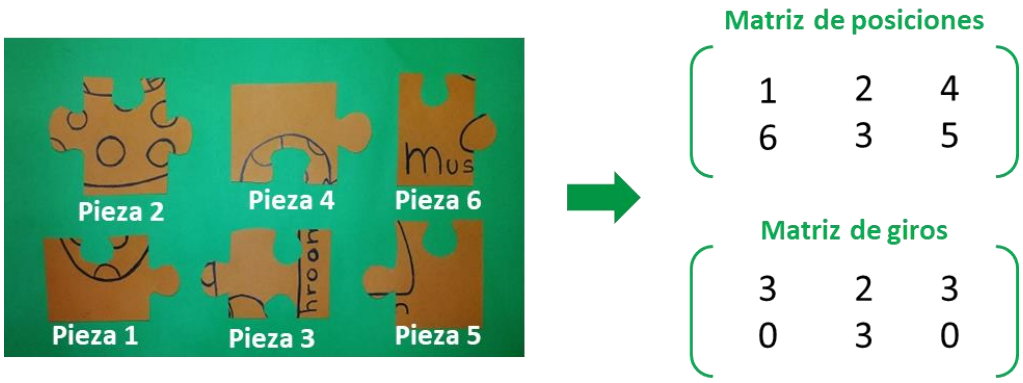


Figura 33. Matrices de solución.

La interpretación de ambas matrices indica que la pieza 1 tiene que girar 270º hacia la izquierda para armar el puzle y que la pieza 6 tiene una orientación correcta porque no tiene que ser girada. De modo, que compaginando la información aportada en ambas matrices, podemos obtener la correcta colocación y orientación de las piezas (Figura 34).

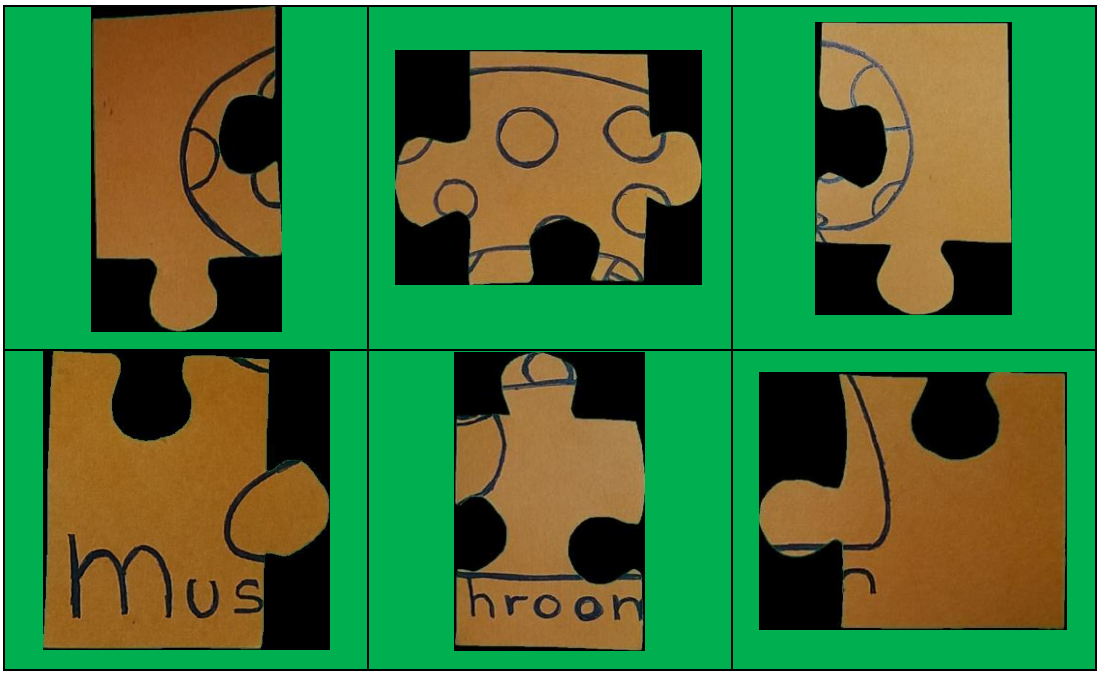


Figura 34. Solución del puzle.

4.2 Fusión de piezas

El resultado del puzle ensamblado se muestra en una imagen final en la cual tomas las piezas están fusionadas entre sí. La fusión se lleva a cabo en Matlab pero se necesita de un proceso previo en el cual la imagen a fusionar es desplazada para dejar un espacio extra a la imagen con la que será fusionada de manera que ambas encajen perfectamente. La fusión se hará en primer lugar para la unión de cada uno de los segmentos y, finalmente, se unirán entre ellos.

Para formar un segmento, se deben unir dos piezas entre sí. Para ello, se crea una matriz de ceros del tamaño del espacio necesario para encajar la pieza que se fusiona. La máscara creada es concatenada con la pieza de modo que quede una imagen como la siguiente:

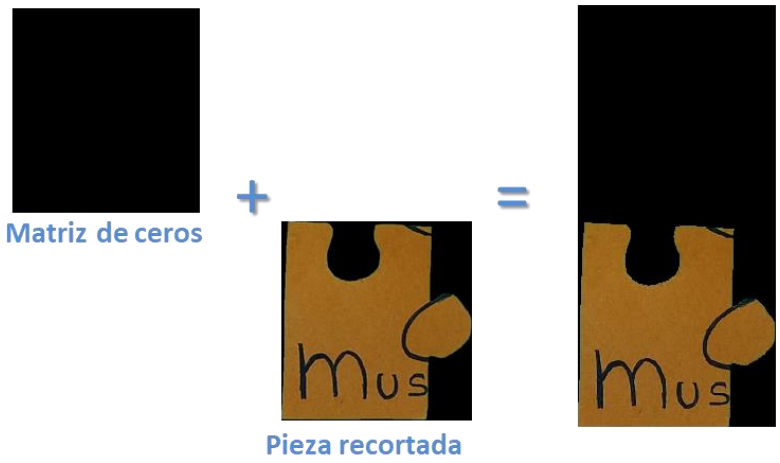


Figura 35. Resultado de la concatenación.

El resultado de la concatenación es una imagen de la pieza original con un espacio sobrante para el proceso de fusión. La función que permite la fusión de dos imágenes en Matlab es imfuse.

```
>>imfusionada = imfuse(p1,b,'diff');
```

Donde p1 es la imagen de la pieza y b la matriz de unos. El resultado es diferencia ‘diff’ de ambas (Figura 36).

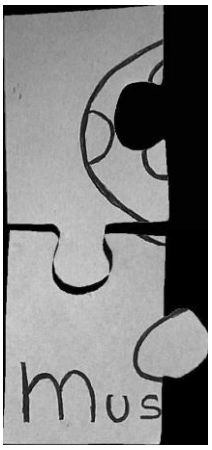


Figura 36. Fusión de piezas del segmento.

Finalmente, para fusionar los 3 segmentos entre sí, se crea una matriz de unos de distinto tamaño y se concatenan con el segmento 2 dejando espacio para la fusión (Figura 37).

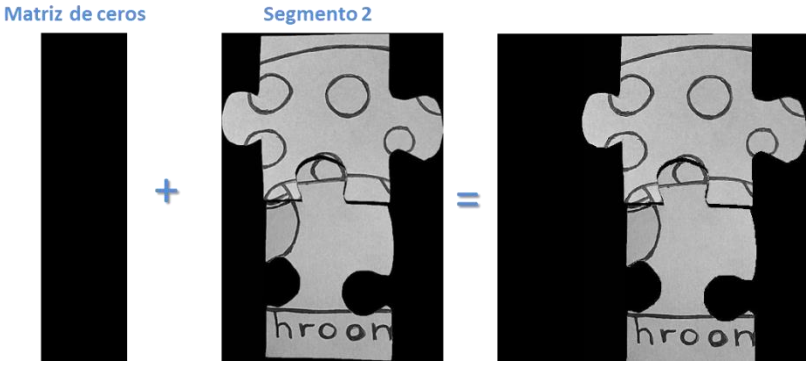


Figura 37. Concatenación de segmentos.

Finalmente, la imagen resultante se fusiona con el segmento 1 para volver a repetir la concatenación anterior y fusionarla con el segmento 3. Quedando como resultado, la imagen completa del puzle.

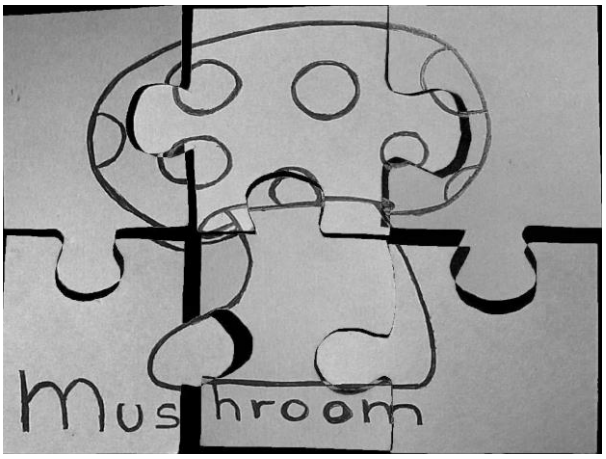


Figura 38. Puzle final fusionado.

5. RESULTADOS

En este capítulo se mostrarán los resultados obtenidos para cada uno de los métodos en estudio. Primero, explicaremos las pruebas realizadas con las distintas orientaciones de las piezas del puzle y el motivo por el cual no se ha seguido desarrollando y se ha optado por cambiar de método.

Posteriormente, veremos los resultados obtenidos para distintos puzles propuestos así como el tiempo de ejecución y el porcentaje de acierto.

5.1 Resultados en la búsqueda de esquinas con piezas giradas.

Para la búsqueda de esquinas ha sido fundamental la representación en coordenadas polares. Las coordenadas polares de las esquinas se encuentran a 90º entre sí, por lo que la búsqueda se reduce si filtramos por estas características. Las primeras pruebas realizadas mostraron que existen puntos entre sí que no pertenecen a las esquinas y que también cumplen dicha condición. En este caso podemos encontrar las piezas de puzle con 3 cabezas, o en puzles futuros, con 4 cabezas.

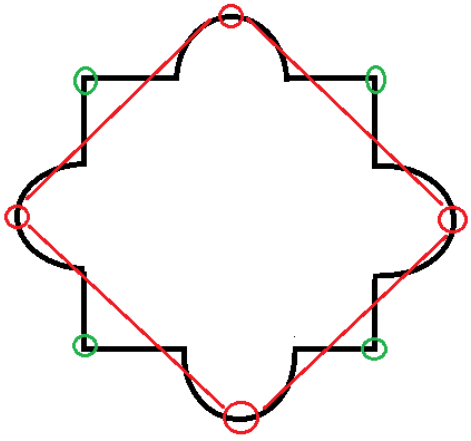


Figura 39. Máximos a 90º entre sí.

Para evitar la errónea toma de esquinas se decidió tener en cuenta en ángulo en forma la recta que une ambos puntos considerados como esquina y la horizontal de la imagen, esto puede verse en la Figura 40. Para que dicho ángulo fuera de interés, la imagen de la pieza debería de estar lo más recta

posible. Así el ángulo que forman entre esquina es conocido y podemos descartar los puntos que no cumplan las características.

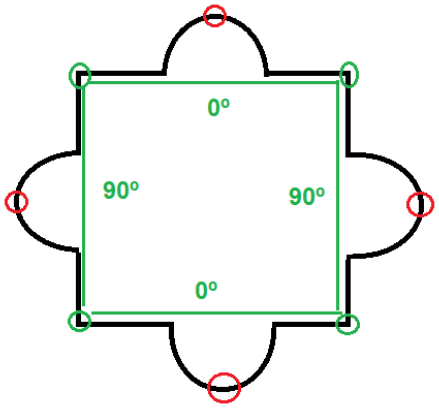


Figura 40. Ángulos resultantes con la horizontal tras la unión de esquinas.

Debido a esto, se descartó la posibilidad de que las piezas se encontraran rotadas en la imagen adquirida a más de 30º, ya que esa es la tolerancia aceptada. Teniendo en cuenta que la imagen es tomada por una cámara que no está fija y la colocación de las piezas sobre la superficie es manual, se tomó una tolerancia de ±30º de orientación permitida.

Por otro lado, esta condición mostro en un principio falsas esquinas, ya que aparecen picos de interés que cumplen los requisitos anteriormente mencionados, la Figura 41 muestra un ejemplo de falsas esquinas.

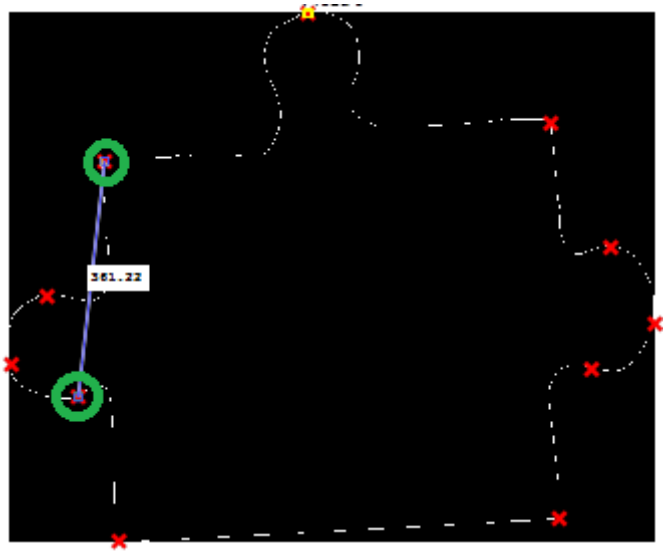


Figura 41. Falsa esquina.

El punto evaluado en la Figura 41 cumple las condiciones mencionadas anteriormente, sin embargo, no pertenece a ningún punto de las esquinas. Como medida para evitar estos falsos picos se tomaron los datos del perímetro de cada figura. Con los datos del perímetro podemos obtener el valor de los lados en los casos más extremos como pueden ser el un cuadrado o un hexágono. Ambos resultados fueron tomados como límites para considerar la distancia entre posibles esquinas, de esta manera se redujeron los errores.

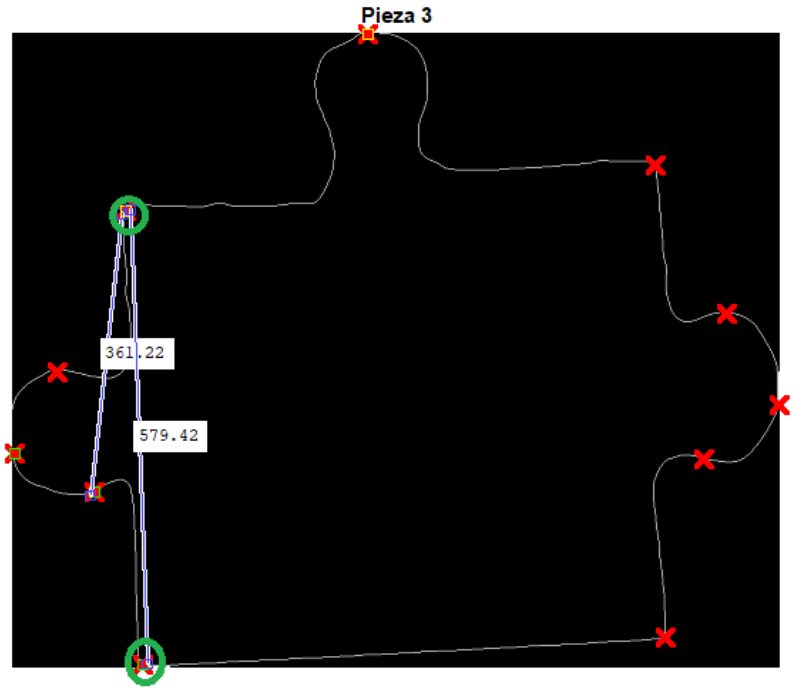


Figura 42. Corrección de esquinas mediante el perímetro.

El perímetro de la figura anterior muestra que la distancia entre las esquinas de la figura debe encontrarse entre 559 y 840 por lo que el punto considerado anteriormente como esquina es descartado, mientras que los puntos que corresponden a las esquinas cumplen dicha condición perfectamente como se muestra en la Figura 43.

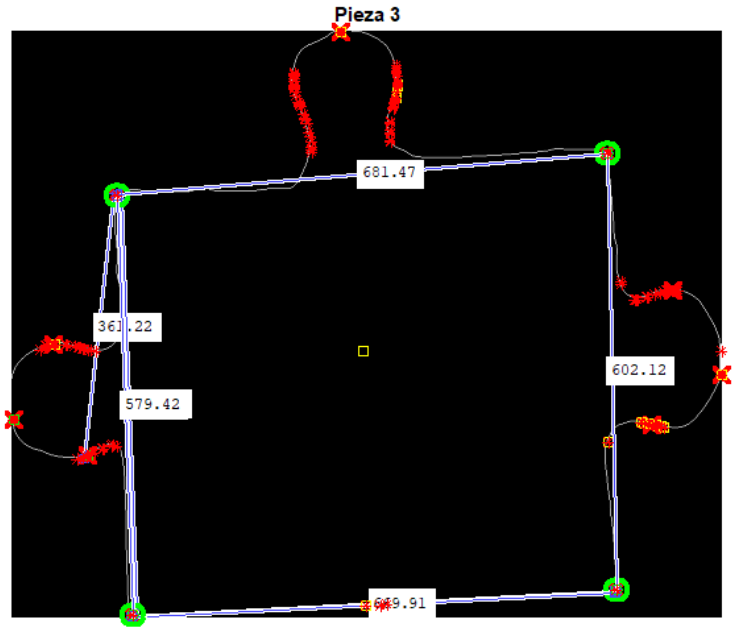


Figura 43. Distancias entre esquinas.

Estos criterios fueron puestos a prueba en 102 piezas diferentes, de las cuales, solo una obtubo una esquina erronea.

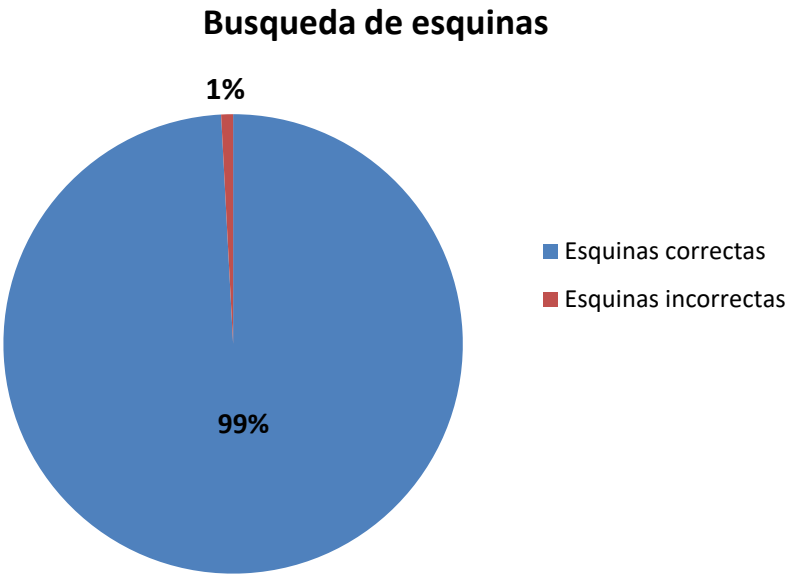


Figura 44. Resultados en la búsqueda de esquinas

5.2 Resultados para la resolución de puzles

Las pruebas se realizaron con 14 tipos de puzles diferentes, todos ellos difieren en formas, tamaño y color.

Los resultados obtenidos en cada imagen se muestran a continuación:

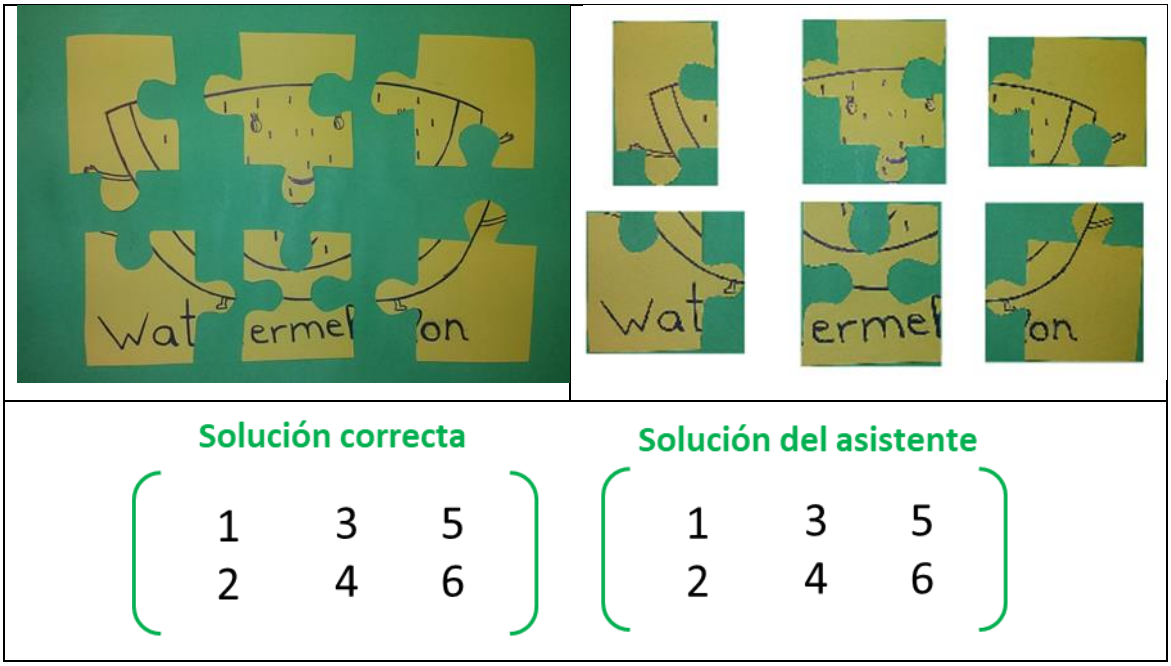


Figura 45. Resultados Puzle 1.

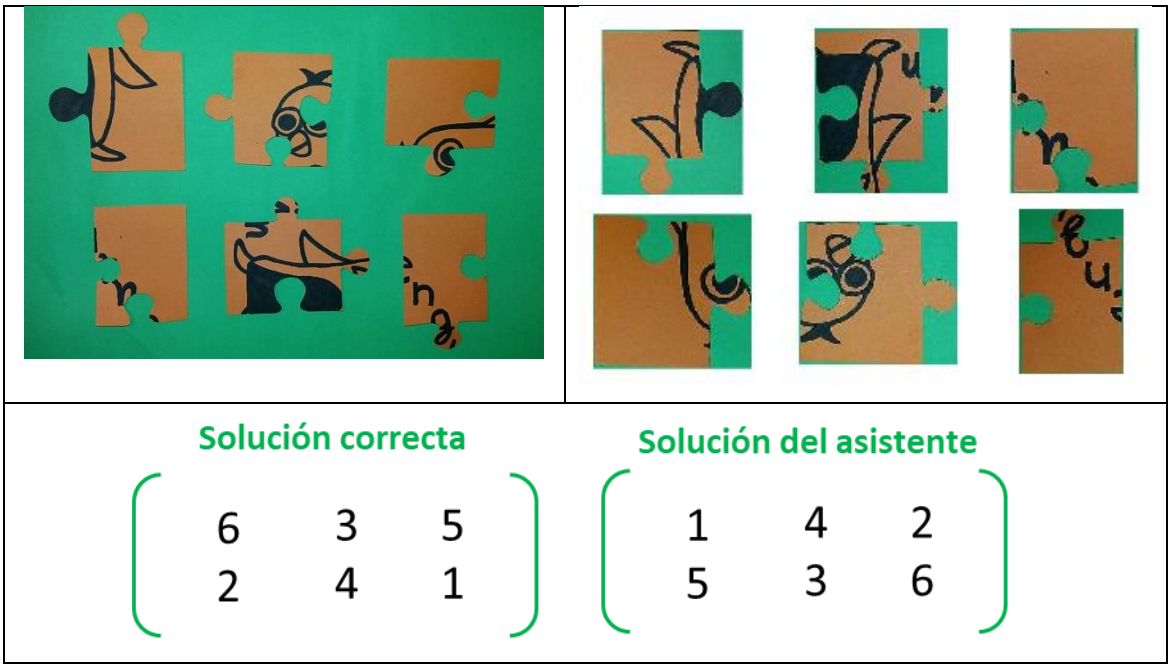


Figura 46. Resultados Puzle 2.

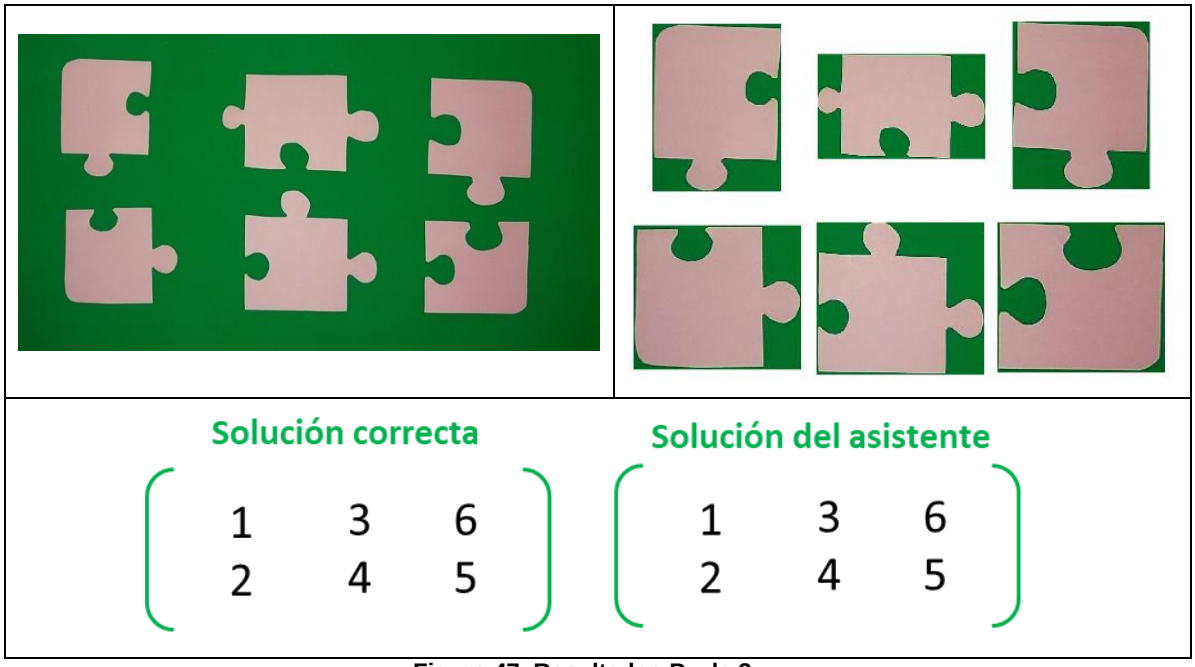


Figura 47. Resultados Puzle 3.

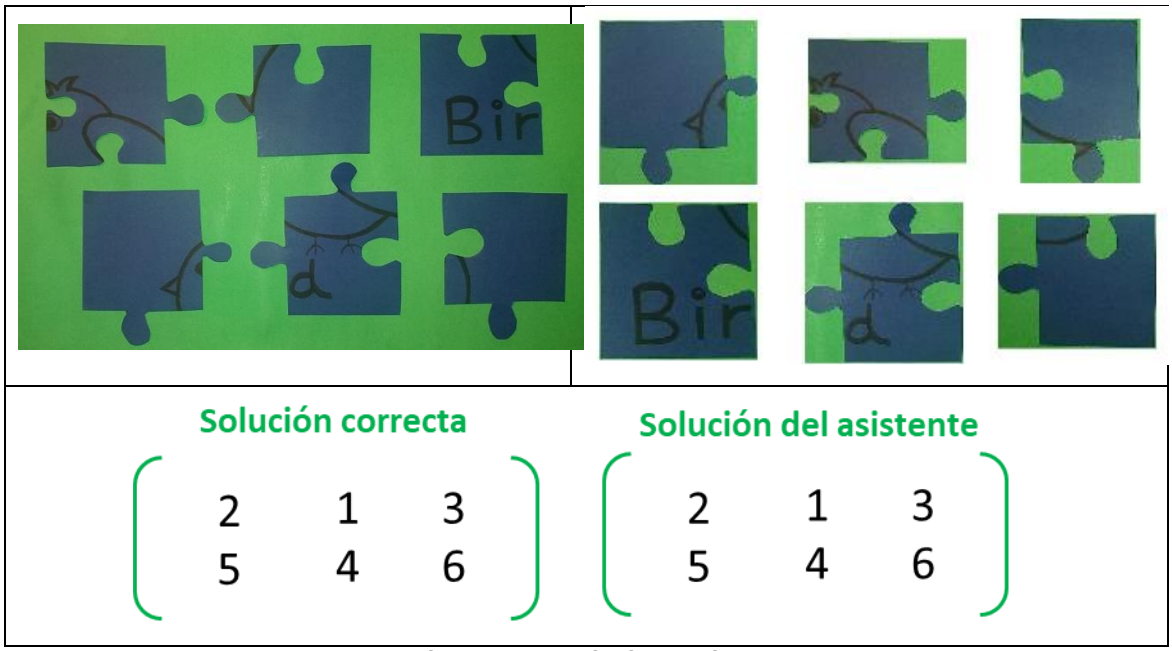


Figura 48. Resultados Puzle 4.

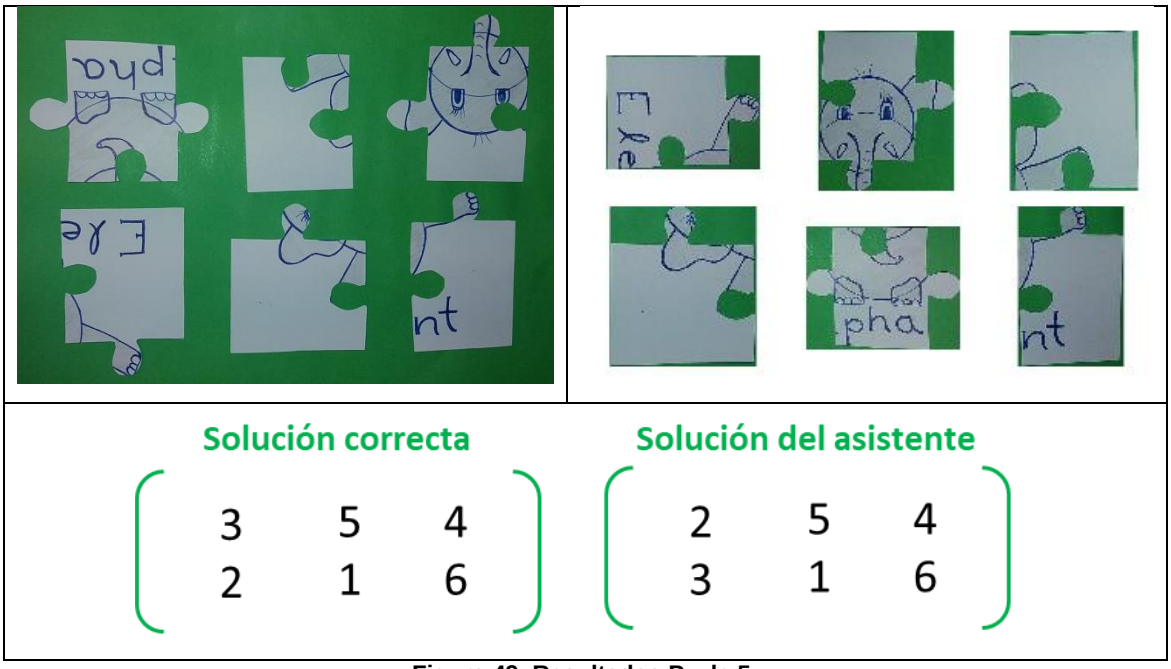


Figura 49. Resultados Puzle 5.

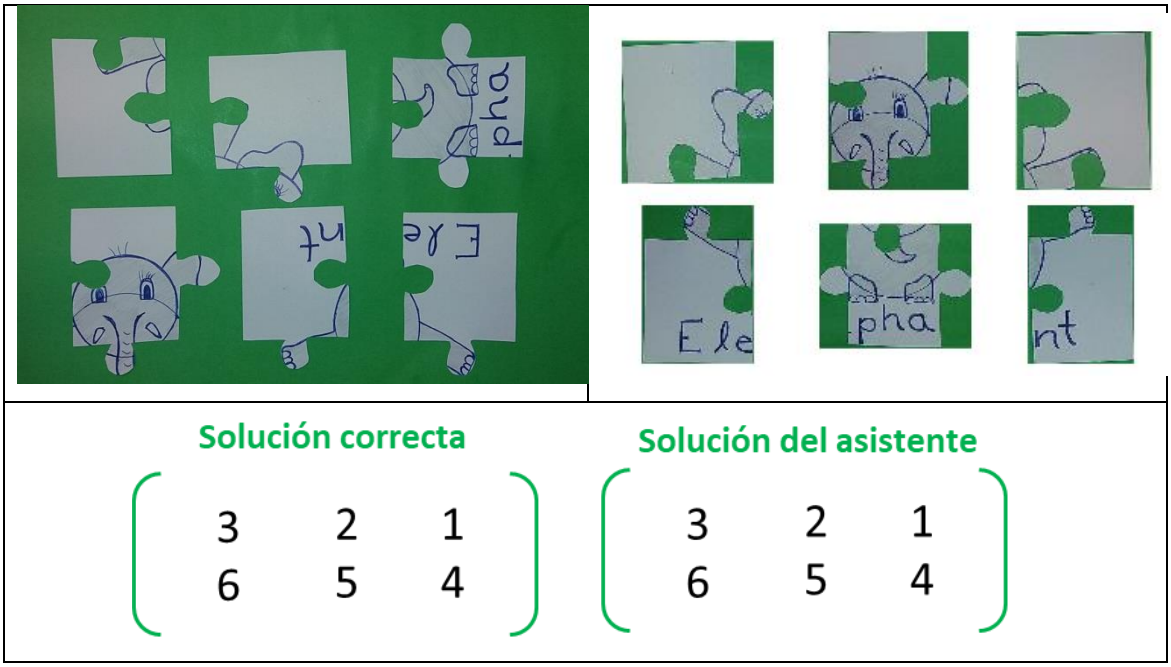


Figura 50. Resultados Puzle 6.

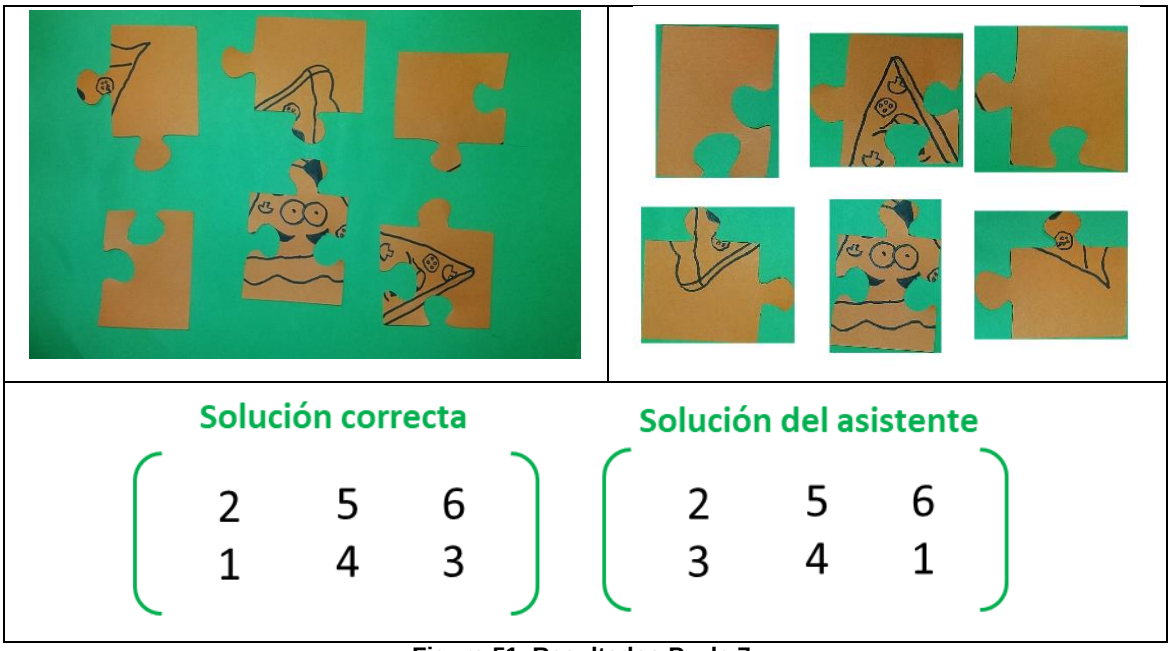


Figura 51. Resultados Puzle 7.

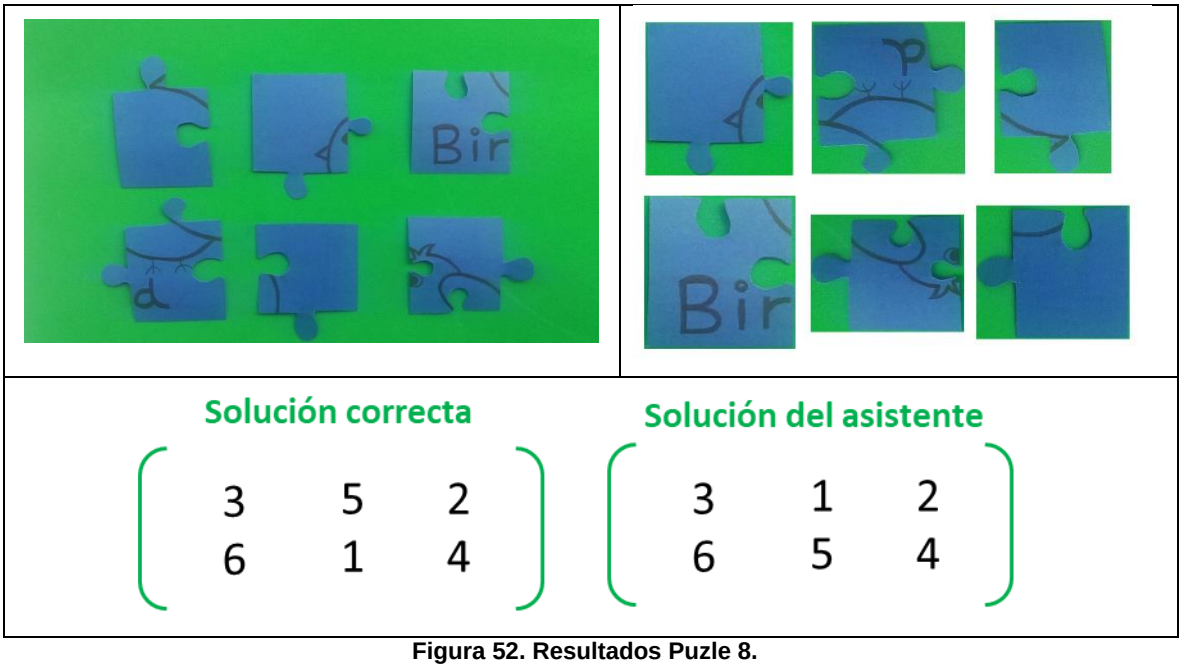


Figura 52. Resultados Puzle 8.

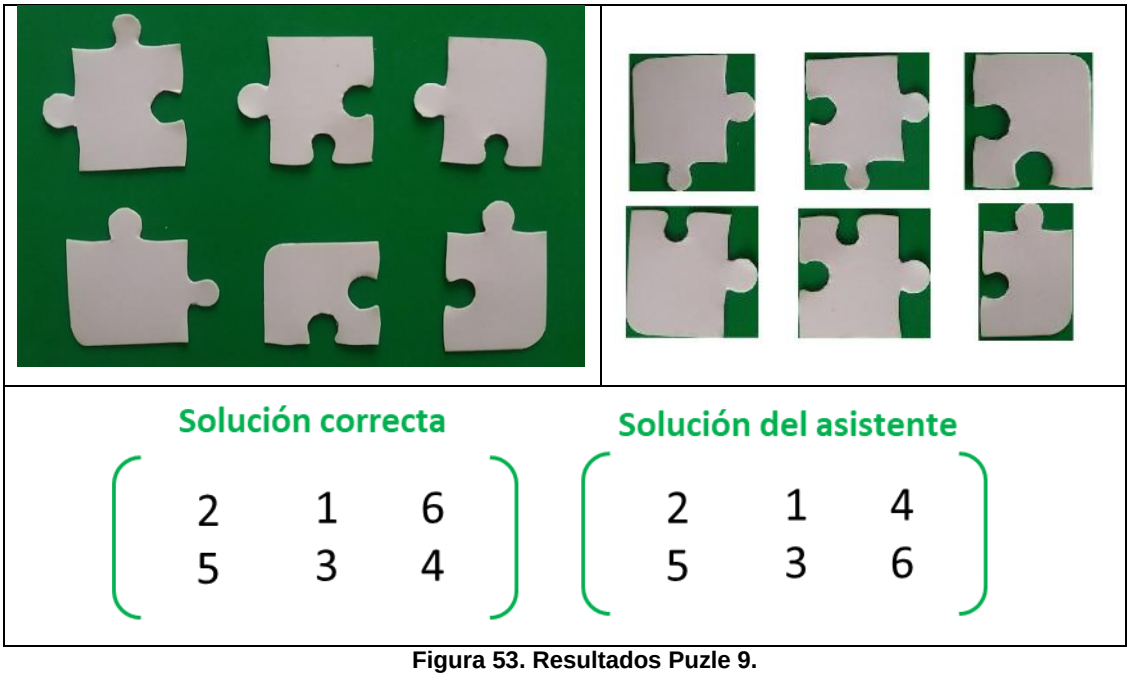


Figura 53. Resultados Puzle 9.

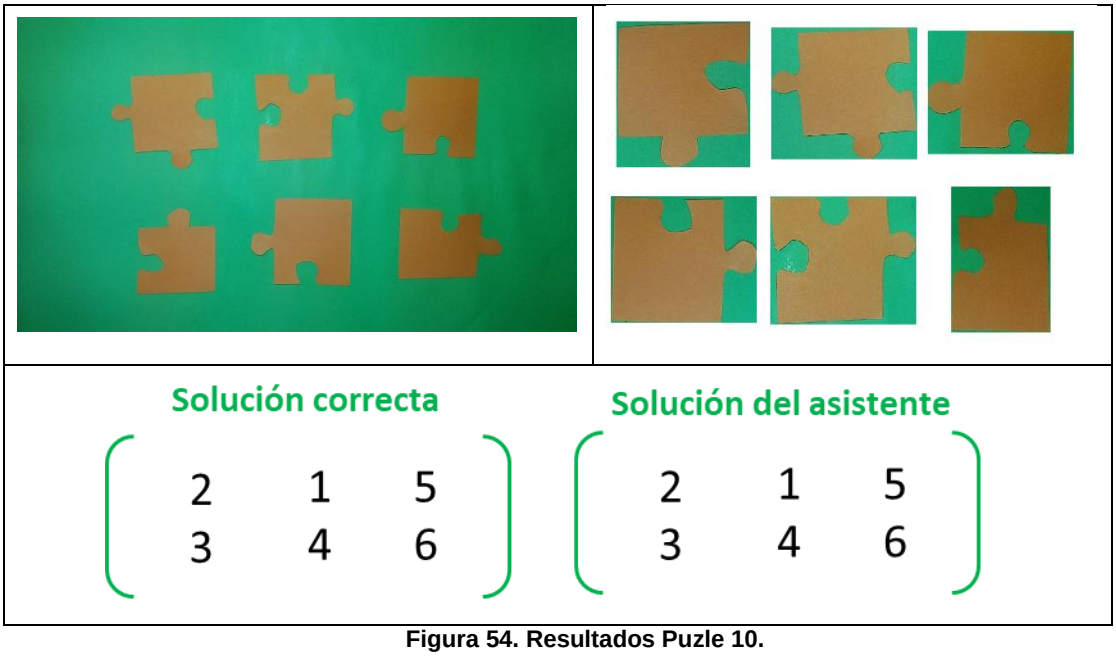


Figura 54. Resultados Puzle 10.

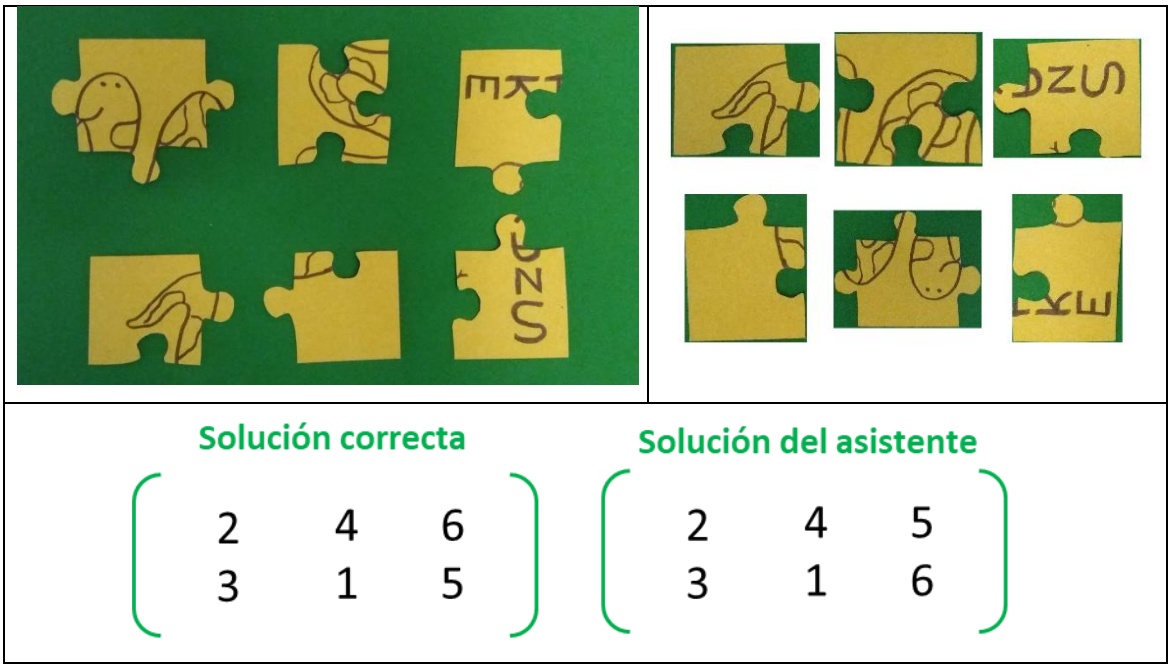


Figura 55. Resultados Puzle 11

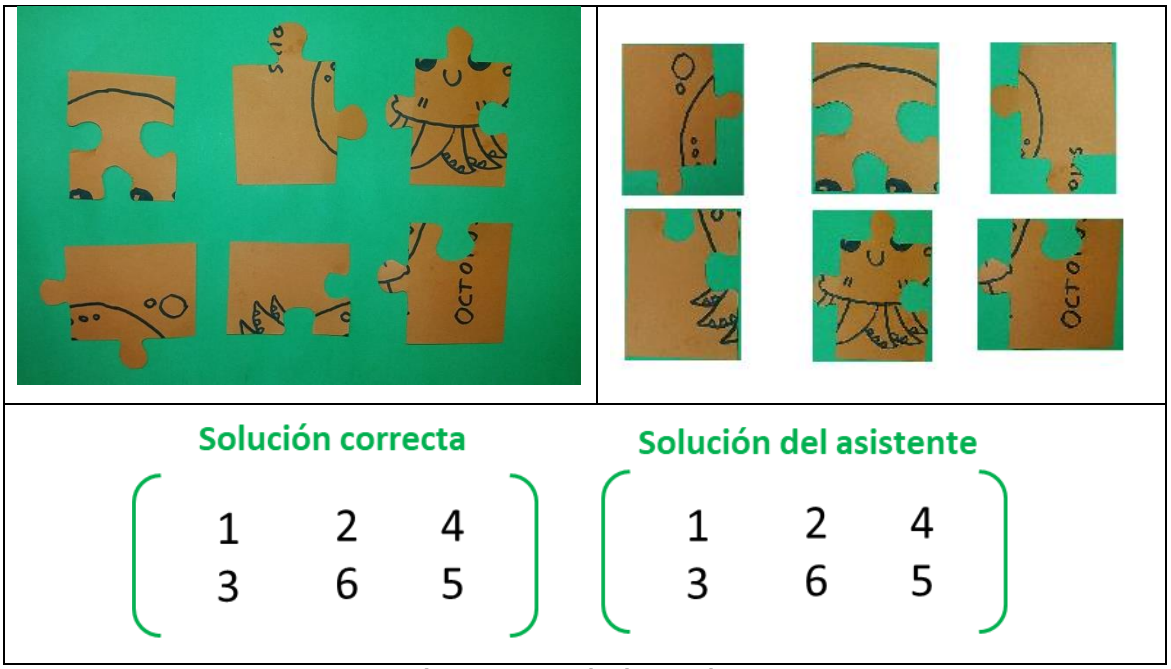


Figura 56. Resultados Puzle 12

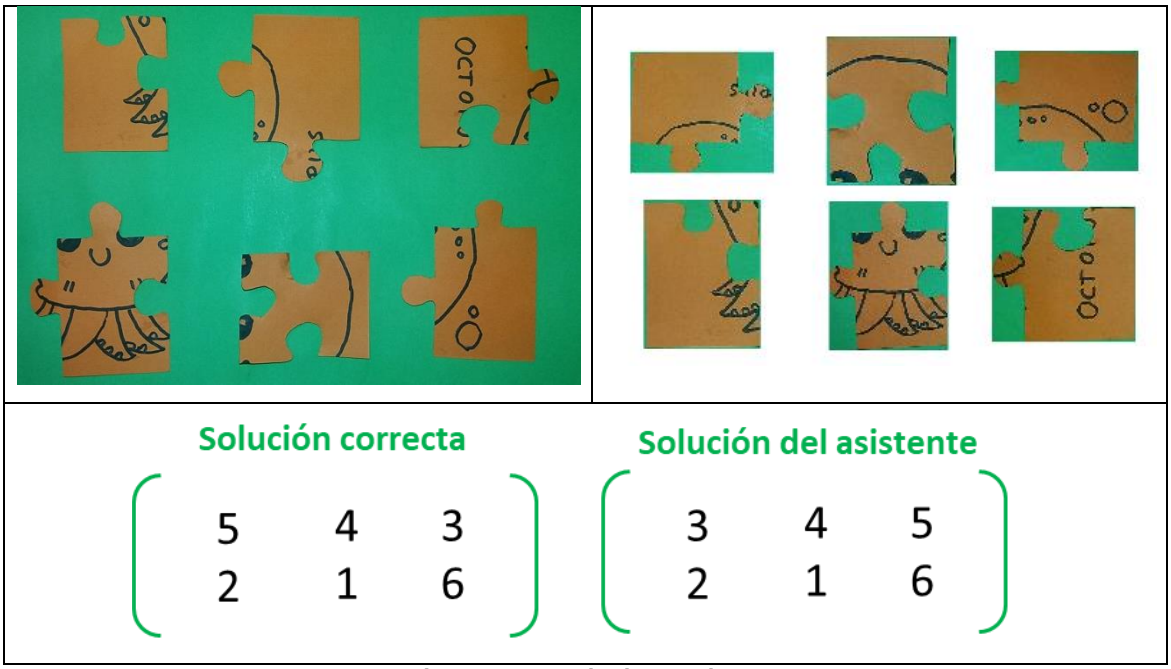


Figura 57. Resultados Puzle 13

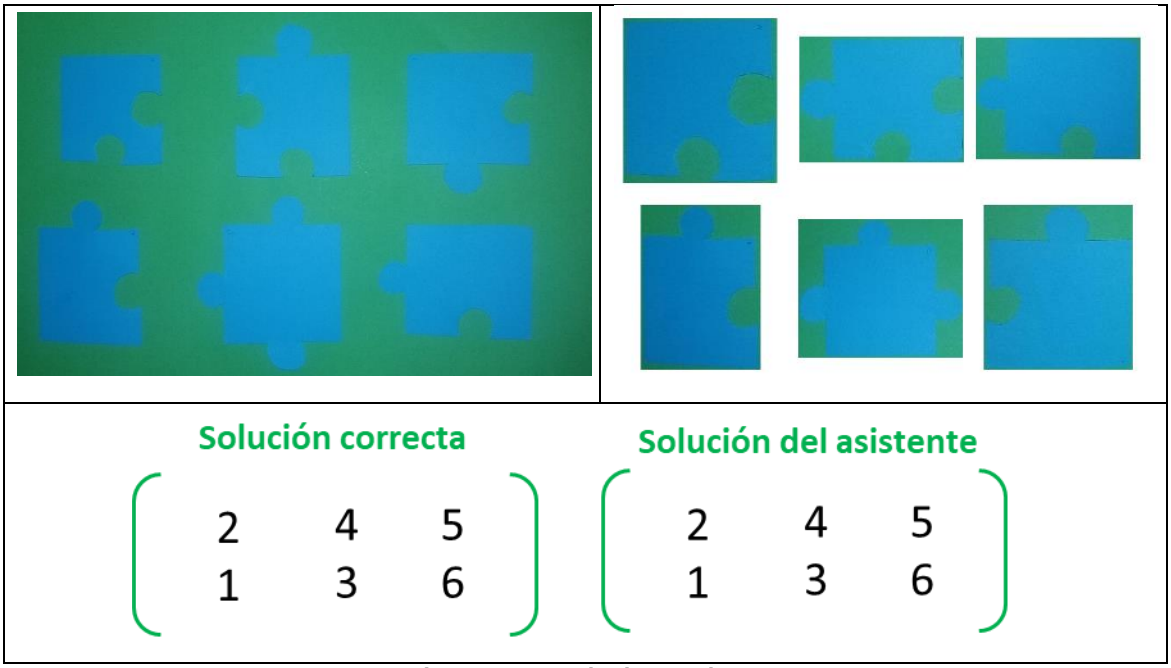


Figura 58. Resultados Puzle 14

Llegados a este punto son considerados como puzles ensamblados correctamente aquellos que aunque la imagen se presente volteada, la combinación de piezas es correcta. Por lo tanto, la siguiente tabla resume el porcentaje de acierto y el tiempo transcurrido para dar la solución de cada uno de los puzles utilizados.

Puzle	Tiempo de procesamiento (s)	Acierto
1	0.080908	OK
2	0.036594	OK
3	0.007123	OK
4	0.005199	OK
5	0.002780	NOK
6	0.010160	OK
7	0.039774	NOK
8	0.047605	NOK
9	0.012500	OK
10	0.001964	OK
11	0.010714	NOK
12	0.015102	OK
13	0.007982	NOK
14	0.036071	OK

Tabla 5. Tiempo de procesamiento y solución de los puzles ensayados.

Los resultados obtenidos muestran que el porcentaje de acierto para el ensamblaje del puzle es del 65%.

5.3 Interfaz gráfica

La ejecución del código se lleva a cabo mediante una interfaz gráfica que permite al usuario interactuar con el ordenador de una manera sencilla e intuitiva (Figura 59).

Ilustración 59. Interfaz gráfica

La interfaz principal consta de un botón principal, un conjunto de botones llamados ‘Opciones’ y un cajetín de solución donde recoge la información obtenida tras el análisis.

El botón ‘Cargar Imagen’ nos permite elegir una imagen desde nuestro directorio que haya sido previamente guardada en el ordenador. (Figura 60)

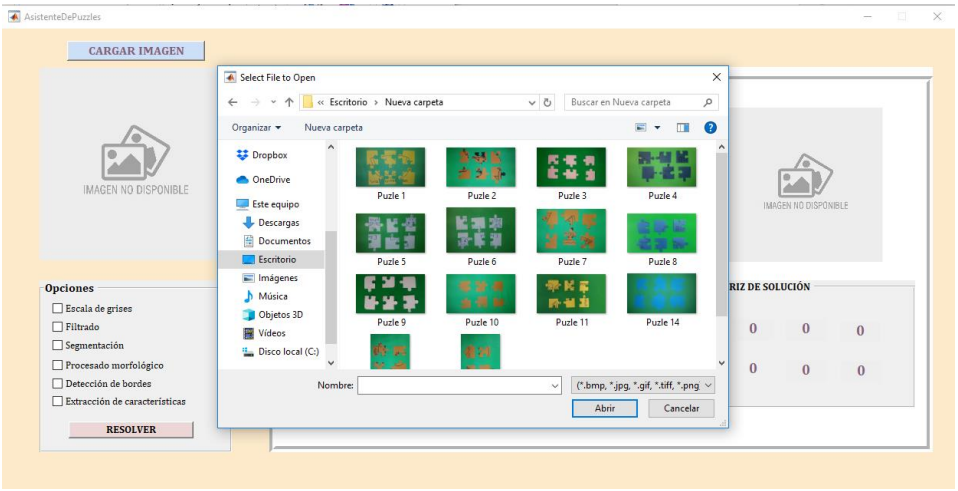


Figura 60. Seleccionar imagen.

Entrando en el directorio y seleccionando la imagen del puzle que queramos ensamblar aparecerá cargada en la ventana superior izquierda (Figura 61)

Figura 61. Interfaz con imagen cargada.

Una vez cargada la imagen tenemos un conjunto de botones en el apartado 'Opciones' que podemos seleccionar. Estos botones corresponden a los distintos pasos que el programa realiza en el tratado de imágenes. Activando dichas casillas, el programa abrirá una ventana mostrando los resultados obtenidos por cada paso seleccionado de modo que tendremos la posibilidad de obtener los resultados de: la escala de grises, el proceso de filtrado, segmentación, resultados del procesamiento morfológico, detección de bordes y las imágenes de cada pieza en la extracción de características. Para activarlos basta con hacer click en el cuadro blanco.

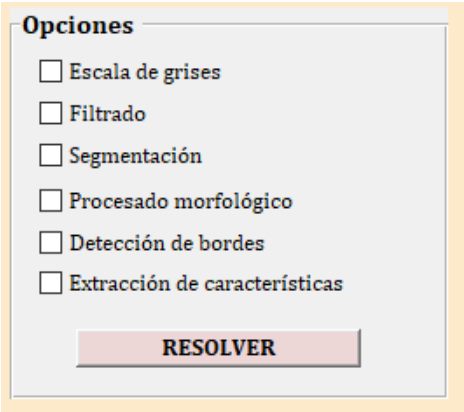


Figura 62. Cuadro de opciones.

Una vez seleccionadas las opciones que desee el usuario se debe pulsar el botón de resolver para mostrar el resultado final. Al pulsar el botón aparecerá una barra de proceso que indica en qué etapa se encuentra nuestro asistente.

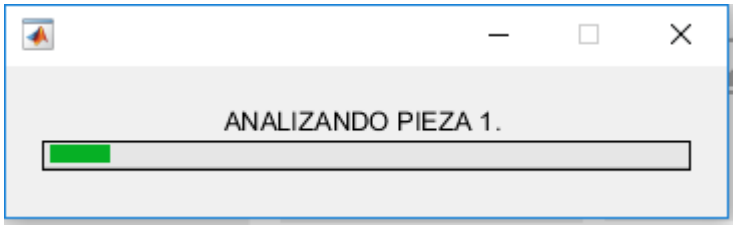


Figura 63. Barra de proceso.

Tras finalizar el análisis, el asistente mostrará las imágenes del puzle ordenadas en la posición dada como correcta junto con la matriz de solución donde indica la posición de cada pieza respecto a la imagen principal que vendrá enumerada pieza a pieza. (Figura 64).

Figura 64. Solución mostrada por el Asistente de Puzle

6. CONCLUSIONES Y TRABAJOS FUTUROS

La realización de este trabajo fue motivada por el deseo de profundizar en las capacidades de un ordenador digital para resolver problemas cuyos datos principales son gráficos. La solución de un rompecabezas requiere aplicar técnicas de tratado de imágenes así como de reconocimiento de patrones y formas para obtener información necesaria y procesarla en búsqueda de un objetivo concreto. Durante este trabajo, muchas técnicas han sido puestas en juego y muchas de ellas han sido descartadas. Este estudio me ha retado a investigar, analizar y probar, las distintas técnicas aplicadas que finalmente me han llevado a elegir el mejor camino para llevar a cabo la resolución de un puzle.

Las principales dificultades han sido encontradas en el momento de procesar la información extraída de las figuras. Algo que a simple vista para el ojo humano es fácilmente reconocible, como pueden ser las esquinas de una pieza de rompecabezas, es una tarea compleja para un ordenador digital. El ruido presente en las imágenes que adquirimos o las irregularidades captadas por la propia pieza en sí nos llevan a obtener datos erróneos. Cómo distinguir dichos datos de aquellos que nos son de interés fue un proceso en el cuál me llevó a descartar la posibilidad de obtener imágenes que contengan piezas giradas. Ya que el proceso de clasificación de lados según sean rectos, cóncavos o convexos es basada principalmente en el ángulo formado entre los puntos medios de borde y las esquinas.

Muchos de los parámetros utilizados en este algoritmo son generados automáticamente, tales como umbrales y límites. Otros probablemente deberán ser ajustados ligeramente para funcionar correctamente en otros rompecabezas de prueba que no estén enfocados en los utilizados en este proyecto.

En cuanto a la clasificación de piezas en función de si sus lados son rectos, cóncavos o convexos y utilizar como imágenes de entrada puzles de

seis piezas nos permitió ensamblar un puzle sin necesidad de utilizar características como el color, grosor o textura.

Ya que en este proyecto solo se han tenido en cuenta la forma de las piezas, los puzles que han sido ensamblados incorrectamente son resultado de la coincidencia en forma de las piezas que lo forman. Como línea futura, un criterio de unión entre piezas debiera ser la comparación con una imagen modelo o la concordancia entre líneas de unión de ambos dibujos. De esta manera, se evitaría el intercambio de piezas en un mismo puzle por su idéntica geometría.

Durante el desarrollo de este trabajo se han tenido en cuenta puzles de 6 piezas divididos por 3 segmentos para facilitar el emparejamiento, para aumentar el número de piezas, basta con añadir un 4 segmento perpendicular a los 3 creados anteriormente como se muestra en la Figura 54.

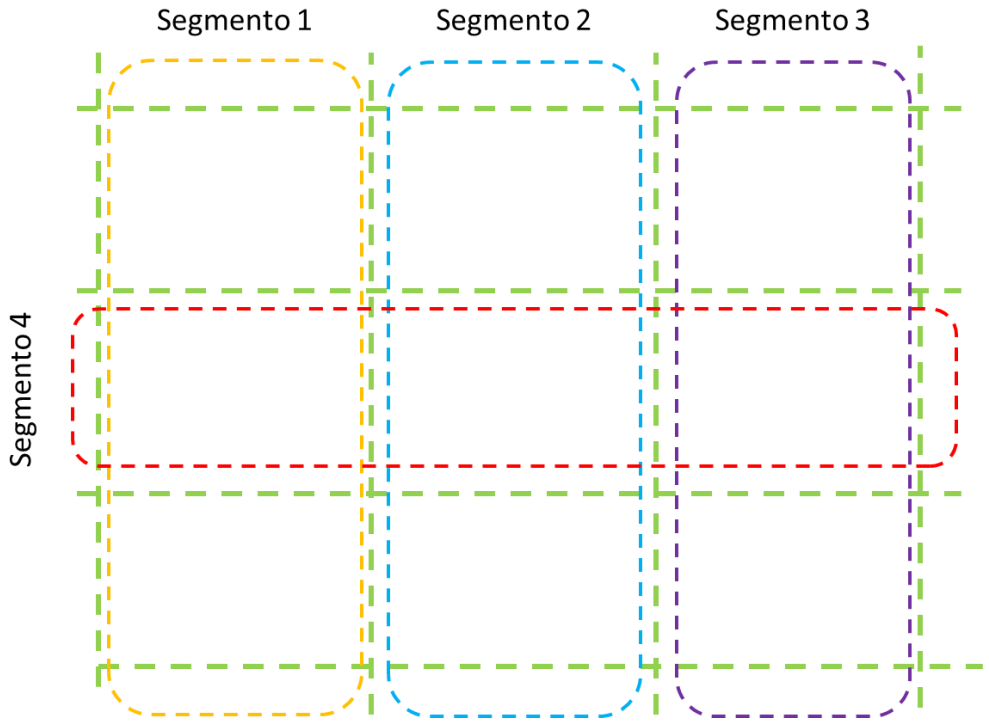


Figura 65. Segmento 4 para el aumento de piezas.

7. BIBLIOGRAFÍA

[1]. H. Freeman and L. Garder, Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition, IEEE Trans. on Electronic Comp. EC-13, (1964) pp.118–127.

[2] Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Transactions on Electronic Computers 2, (1961) pp. 260–268

[3] Wolfson H.E., Schonberg, A., Kalvin, A and Lamdan, "Solving Jigsaw Puzzles by Computer," Annals of Operation Research, 12 (1988) 51-64.

[4] Goldberg, D., Malon, C and Bern "A Global Approach to Automatic Solution of Jigsaw Puzzles," SoCG'02, June, (2002) pp. 5–7

[5] Yao, F.H., Shao, G.F ., "A Shape and Image Merging Technique to Solve Jigsaw Puzzles," Pattern Recognition Letters, 24, (2003) pp. 1819-1835.

[6] *GUI de MATLAB* [en línea] [Fecha de consulta: 10 de Septiembre de 2018]. Disponible en: https://es.mathworks.com/discovery/matlab-gui.html

[7] VENKATACHALAPATHI Vigneshwar, "Automated Solver for Jigsaw Puzzles" [en línea] [Fecha de consulta: 9 de Octubre de 2018]Disponible en: https://github.com/Kawaboongawa/Zolver

[8] Javier Enebral González, "Detección y asociación automática de puntos característicos para diferentes aplicaciones", Tesis doctoral. Universidad de Cataluña, (2009)