UNIVERSIDAD SIGLO 21

INTELIGENCIA ARTIFICIAL

Licenciatura en Informática

Trabajo Práctico 4

Prof. Titular Disciplinar: María Paula González

Prof. Titular Experto: Pablo Alejandro Virgolini

Alumno: Pablo Alejandro Hamann

Legajo: VINF010782

Año: 2025

Tabla de contenido

1. Objetivo

El objetivo para esta actividad es:

1.

2. Situación problemática

La situación problemática se centra en la línea de montaje de blocks (de motores), donde se dan pequeños desplazamientos sobre la cinta transportadora que ocasionan errores en el posicionamiento de las piezas, interrumpiendo el proceso, y potencialmente llegando a detener toda la línea de producción. Estos problemas surgen porque los robots ensambladores carecen de autonomía para ajustar sus movimientos ante estas variaciones. La empresa busca implementar inteligencia artificial para otorgar a los robots un nivel suficiente de autonomía, permitiendo que puedan identificar y corregir automáticamente estos desplazamientos menores, ahora haciendo uso de la **transformada de Hough**, que nos permitiría detectar rápidamente y de forma robusta el centro del anillo de referencia en el block del motor. Esto es esencial para corregir automáticamente el desplazamiento y permitir al robot realizar correctamente la operación de posicionamiento y montaje.

3. Desarrollo de las consignas

3.1. Formulación y características de la transformación de Hough

La transformación de Hough es una técnica muy utilizada en el procesamiento de imágenes para identificar patrones geométricos, como **rectas**, **círculos** y otras formas geométricas definidas mediante parámetros. Su principio básico consiste en convertir los puntos detectados en una imagen desde su espacio original a un espacio de parámetros, donde el patrón buscado se representa como un punto o una región fácilmente reconocible.

Esta técnica permite superar limitaciones como la discontinuidad en los patrones y la presencia de ruido.

3.1.1. Transformada de Hough para rectas

En el caso de rectas, la transformación estándar se basa en la representación:

Espacio original (imagen):

$$y = mx + b$$

donde:

- m es la pendiente de la recta
- b la ordenada al origen.

Espacio transformado:

Cada punto (x, y) en el espacio original se transforma en una línea en el espacio de parámetros (m, b), representando todas las rectas que podrían pasar por dicho punto.

Por practicidad y robustez, se utilizan coordenadas polares (ρ, θ), por lo cual al formulación queda definida como:

$$\rho = x \cos\theta + y \sin\theta$$

En este espacio, un punto (x, y) corresponde a una curva sinusoidal, y la acumulación de intersecciones entre estas curvas permite identificar parámetros (ρ, θ) de las rectas presentes en la imagen.

Características principales:

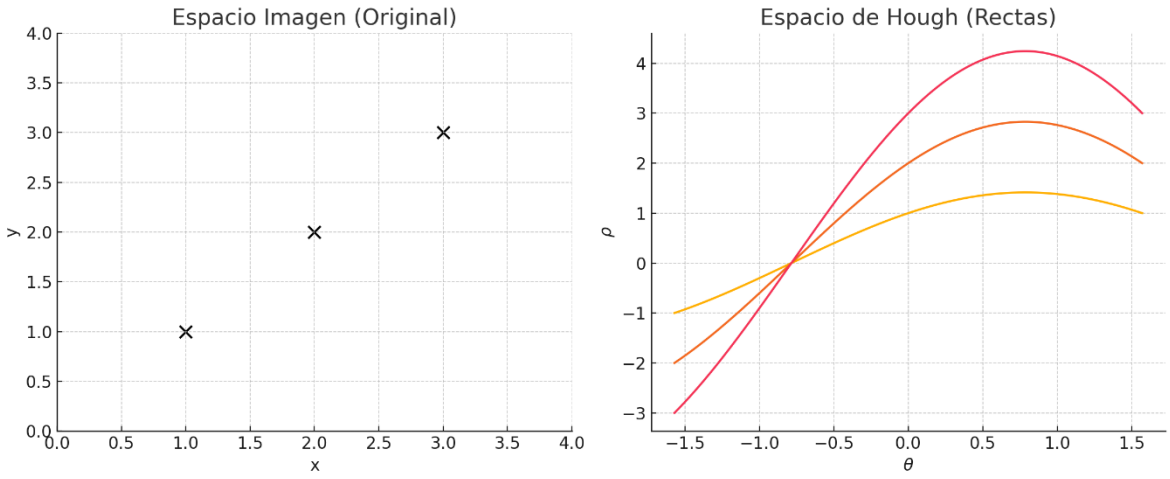
- Robustez frente al ruido.
- Capacidad de detectar rectas discontinuas.
- Facilidad de implementación.
- Eficiencia en imágenes con pocos patrones rectilíneos claros.

Ejemplo:

Una imagen con tres puntos alineados en el espacio original produce curvas sinusoidales que se cruzan en un único punto en el espacio transformado, indicando la presencia de una recta específica.

- **Espacio original:** Se presentan puntos que podrían alinearse formando una recta.

- **Espacio de Hough:** Cada punto genera una curva sinusoidal. La intersección de estas curvas indica la presencia de la recta y determina sus parámetros (ρ, θ).



3.1.2. Transformada de Hough para circunferencias

La ecuación general de una circunferencia en el plano cartesiano es:

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

donde:

- $(x - x_0)$ son las coordenadas del centro
- R es el radio de la circunferencia.

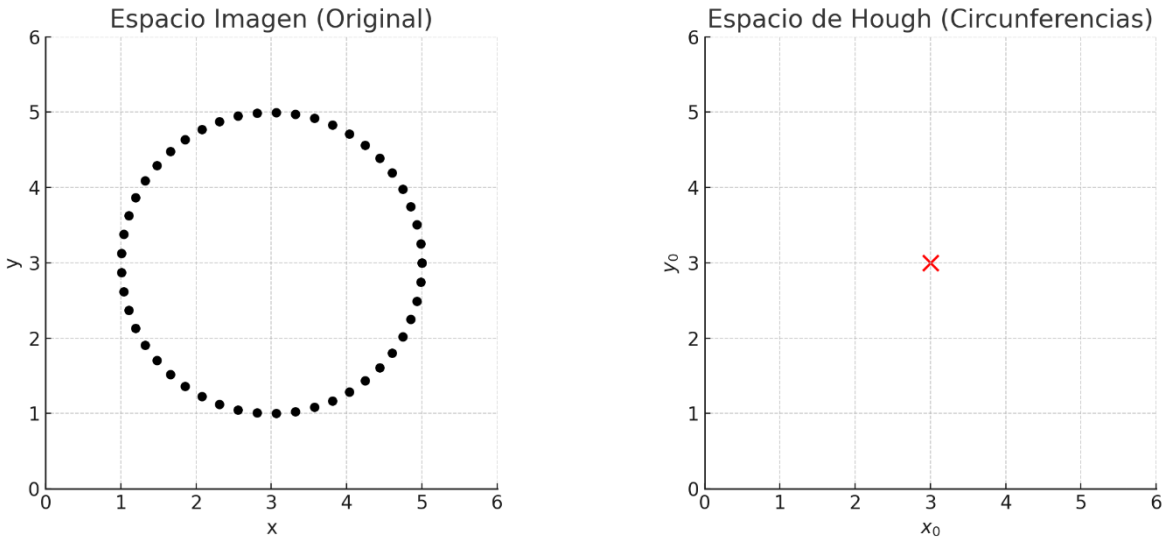
En su versión más general, el espacio de parámetros es tridimensional (x_0, y_0, R). Sin embargo, cuando el radio RRR es conocido (como en el caso del problema presentado), el espacio de parámetros se simplifica a dos dimensiones (x_0, y_0), representando únicamente los posibles centros. Entonces ahora cada punto en el espacio original se convierte en una circunferencia en el espacio de parámetros (nuestros potenciales centros). La acumulación de circunferencias en el espacio transformado permite identificar claramente el centro del círculo original.

Características principales:

- Permite detectar figuras curvas precisas.
- Robusta a la presencia de ruido moderado.
- Eficaz cuando el radio es conocido o limitado a ciertos valores predefinidos.

Ejemplo:

Una imagen con puntos pertenecientes a una circunferencia definida con radio conocido generará un conjunto de círculos en el espacio transformado que se intersectan claramente en el centro real de la circunferencia original.



- **Espacio original:** Los puntos forman una circunferencia claramente definida.
- **Espacio de Hough:** Los puntos se transforman en un único punto que representa el centro de la circunferencia original, dado que el radio es conocido.

3.1.3. Tabla comparativa

Resumen de características generales de la transformación de Hough

Ventajas	Desventajas
Muy efectiva en la detección de patrones geométricos bien definidos.	Requiere una discretización adecuada del espacio de parámetros.
Alta tolerancia a puntos faltantes o ruido.	Puede ser computacionalmente intensiva en problemas con muchos parámetros o con alta resolución.
Implementación algorítmica sencilla y rápida (particularmente en problemas bien definidos).	La elección inadecuada de parámetros puede producir falsos positivos.

3.2. Desarrollo e implementación de un prototipo destinado a la transformada de Hough de rectas

A continuación, una versión sencilla y clara del algoritmo básico de la *Transformada de Hough* para detectar rectas:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TransfHoughRectas {

    private final int rhoMax;
    private final int[][] accumulator;
    private final double[] sinCache;
    private final double[] cosCache;
    private final int thetaMax = 180;

    private List<double[]> points = new ArrayList<>();

    public TransfHoughRectas(List<double[]> points, int width, int height) {
        this.points = points;

        rhoMax = (int) Math.hypot(width, height);
        accumulator = new int[2 * rhoMax][thetaMax];

        sinCache = new double[thetaMax];
        cosCache = new double[thetaMax];

        for (int theta = 0; theta < thetaMax; theta++) {
            double thetaRad = Math.toRadians(theta);
            sinCache[theta] = Math.sin(thetaRad);
            cosCache[theta] = Math.cos(thetaRad);
        }
    }

    // Realiza la acumulación de votos en el espacio de Hough
    public void performTransform() {
        for (double[] p : points) {
            double x = p[0];
            double y = p[1];
            for (int theta = 0; theta < thetaMax; theta++) {
                int rho = (int) Math.round(x * cosCache[theta] + y * sinCache[theta]) +
rhoMax;
```

```
                if (rho >= 0 && rho < 2 * rhoMax)
                    accumulator[rho][theta]++;
            }
        }
    }

    // Devuelve el máximo de la acumulación (parámetros de la recta)
    public int[] getMaxAccumulator() {
        int max = 0;
        int rhoMaxFound = 0;
        int thetaMaxFound = 0;

        for (int rho = 0; rho < 2 * rhoMax; rho++) {
            for (int theta = 0; theta < thetaMax; theta++) {
                if (accumulator[rho][theta] > max) {
                    max = accumulator[rho][theta];
                    rhoMaxFound = rho - rhoMax;
                    thetaMaxFound = theta;
                }
            }
        }

        return new int[]{rhoMaxFound, thetaMaxFound, max};
    }

    // Método principal de ejemplo
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<double[]> points = new ArrayList<>();

        System.out.println("Deberá ingresar coordenadas para (x ; y)...\nIngrese dos
números (enteros o decimales), de a pares, separados por un espacio.\nEjemplo: 0
1.5\nPuede dejar de introducir valores en cualquier momento, introduciendo la letra
'q'.");

        int contador = 1;
        while (true) {
            System.out.print("(x[" + contador + "] ; y[" + contador + "]) = ");
            String input = scanner.nextLine().trim();

            if (input.equalsIgnoreCase("q")) {
                break;
            }

            String[] parts = input.split("\\s+");
```

```
            if (parts.length != 2) {
                System.out.println("¡Error! Deben ingresarse dos valores numéricos
separados por espacio.");
                continue;
            }

            try {
                double x = Double.parseDouble(parts[0]);
                double y = Double.parseDouble(parts[1]);
                points.add(new double[]{x, y});
                contador++;
            } catch (NumberFormatException e) {
                System.out.println("¡Error! Ingresar números válidos.");
            }
        }

        if (points.isEmpty()) {
            System.out.println("No se ingresaron coordenadas de puntos.");
            scanner.close();
            return;
        }

        // Mostrar resumen de puntos ingresados
        System.out.println("\nValores ingresados:");
        for (int i = 0; i < points.size(); i++) {
            double[] point = points.get(i);
            System.out.printf("[%d] (x ; y) = (%.1f ; %.1f)\n", i+1, point[0],
point[1]);
        }
        System.out.println();

        // Determinar límites aproximados para espacio Hough
        int width = (int) points.stream().mapToDouble(p -> p[0]).max().getAsDouble() +
10;
        int height = (int) points.stream().mapToDouble(p -> p[1]).max().getAsDouble() +
10;

        TransfHoughRectas hough = new TransfHoughRectas(points, width, height);
        hough.performTransform();

        int[] lineParameters = hough.getMaxAccumulator();

        if (lineParameters[2] > 1) { // al menos dos puntos formando línea
            System.out.print("Recta detectada con parámetros: ");
```

```
            System.out.println("(ρ ; θ) = (" + lineParameters[0] +" ; "+
lineParameters[1]+"°)");
            System.out.println("Puntos alineados: " + lineParameters[2]);
        } else {
            System.out.println("No se detectó una recta clara.");
        }

        scanner.close();
    }

}
```

Cómo funciona

1. El programa solicita el ingreso de pares (x, y). El ingreso finaliza cuando el usuario introduce **q**.
2. Con los puntos ingresados, se ejecuta la transformada de Hough acumulando votos en el espacio Hough.
3. Finalmente, el prototipo informa si detecta claramente una recta, mostrando parámetros detectados (ρ, θ) y la cantidad de puntos alineados detectados.

Salida (muestra de ejemplo)

```
Deberá ingresar coordenadas para (x ; y)...
Ingrese dos números (enteros o decimales), de a pares, separados por un espacio.
Ejemplo: 0 1.5
Puede dejar de introducir valores en cualquier momento, introduciendo la letra 'q'.
(x[1] ; y[1]) = 1 1
(x[2] ; y[2]) =  2 2.1
(x[3] ; y[3]) = 3.3 2.9
(x[4] ; y[4]) =
¡Error! Deben ingresarse dos valores numéricos separados por espacio.
(x[4] ; y[4]) = q

Valores ingresados:
[1] (x ; y) = (1,0 ; 1,0)
[2] (x ; y) = (2,0 ; 2,1)
[3] (x ; y) = (3,3 ; 2,9)

Recta detectada con parámetros: (ρ ; θ) = (0 ; 127°)
Puntos alineados: 3

Process finished with exit code 0
```

Características

1. **Interactivo**: el prototipo solicita al usuario, el ingreso de pares numéricos para las coordenadas (x, y).
2. **Flexible**: maneja tipos de datos enteros y decimales (*double*).
3. Programa de consola (**CLI**).

Ventajas

1. El código es sencillo y fácil y posibilita una fácil comprensión del método de *Transformada de Hough* para rectas.
2. Eficiente para pruebas pequeñas (como las de entornos educativos y demostrativos).

Limitaciones

1. La matriz acumuladora (`accumulator`) puede crecer significativamente en función del tamaño máximo esperado de las coordenadas, lo cual puede implicar un uso intensivo de memoria en situaciones reales con imágenes o puntos con rangos muy grandes.
2. Sensible a ruido (puntos alejados de la línea), pudiendo detectar falsos positivos si no se implementan técnicas adicionales de umbralización o filtrado.
3. Elegir adecuadamente la resolución del espacio es crítico para una detección efectiva.

Dificultades

1. Dado que en Java prescindí de un entorno gráfico y en su lugar, decidí trabajar con CLI, se hicieron necesarias validaciones, excepciones y validaciones para asegurar que las entradas de los usuarios fueran adecuadas.
2. Determinar límites óptimos y adecuados (`rhoMax`, `width` y `height`) para la matriz acumuladora, requirió antes, planificación (es decir, anotar previamente, en papel y lápiz, por ejemplo, de forma tal de ver con qué valores se alimenta al programa.

3.3. Desarrollo e implementación de un prototipo destinado a la transformada de Hough de circunferencias

Se presenta un prototipo que permite explorar el funcionamiento de la Transformada de Hough en el caso particular y práctico de la detección de circunferencias con Radio conocido.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TransfHoughCircunferencias {

    private final int[][] accumulator;
    private final int width, height, radius;

    private List<double[]> points = new ArrayList<>();

    public TransfHoughCircunferencias(List<double[]> points, int width, int height, int
radius) {
        this.points = points;
        this.width = width;
        this.height = height;
        this.radius = radius;
        accumulator = new int[width][height];
    }

    public void performTransform() {
        for (double[] p : points) {
            double x = p[0];
            double y = p[1];

            for (int theta = 0; theta < 360; theta++) {
                int a = (int) Math.round(x - radius * Math.cos(Math.toRadians(theta)));
                int b = (int) Math.round(y - radius * Math.sin(Math.toRadians(theta)));

                if (a >= 0 && a < width && b >= 0 && b < height) {
                    accumulator[a][b]++;
                }
            }
        }
    }

    public int[] getMaxAccumulator() {
```

```
        int max = 0;
        int aMaxFound = 0;
        int bMaxFound = 0;

        for (int a = 0; a < width; a++) {
            for (int b = 0; b < height; b++) {
                if (accumulator[a][b] > max) {
                    max = accumulator[a][b];
                    aMaxFound = a;
                    bMaxFound = b;
                }
            }
        }

        return new int[]{aMaxFound, bMaxFound, max};
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<double[]> points = new ArrayList<>();

        System.out.print("\nIngrese el radio conocido para la circunferencia: ");
        int radius = Integer.parseInt(scanner.nextLine().trim());

        System.out.println("Ingrese pares de coordenadas (x ; y).\nPuede dejar de
introducir valores en cualquier momento, introduciendo la letra 'q'.");

        int contador = 1;
        while (true) {
            System.out.print("(x[" + contador + "] ; y[" + contador + "]) = ");
            String input = scanner.nextLine().trim();

            if (input.equalsIgnoreCase("q")) {
                break;
            }

            String[] parts = input.split("\\s+");
            if (parts.length != 2) {
                System.out.println("¡Error! Deben ingresarse dos valores numéricos
separados por espacio.");
                continue;
            }

            try {
                double x = Double.parseDouble(parts[0]);
```

```
                double y = Double.parseDouble(parts[1]);
                points.add(new double[]{x, y});
                contador++;
            } catch (NumberFormatException e) {
                System.out.println("¡Error! Ingresar números válidos.");
            }
        }

        if (points.isEmpty()) {
            System.out.println("No se ingresaron coordenadas.");
            scanner.close();
            return;
        }

        System.out.println("\nValores ingresados:");
        for (int i = 0; i < points.size(); i++) {
            double[] point = points.get(i);
            System.out.printf("[%d] (x ; y) = (%.1f ; %.1f)\n", i + 1, point[0],
point[1]);
        }
        System.out.println();

        int width = (int) points.stream().mapToDouble(p -> p[0]).max().getAsDouble() +
radius + 10;
        int height = (int) points.stream().mapToDouble(p -> p[1]).max().getAsDouble() +
radius + 10;

        TransfHoughCircunferencias hough = new TransfHoughCircunferencias(points, width,
height, radius);
        hough.performTransform();

        int[] circleParams = hough.getMaxAccumulator();

        if (circleParams[2] > 1) {
            System.out.print("Centro detectado de la circunferencia: ");
            System.out.println("(a ; b) = (" + circleParams[0] + " ; " + circleParams[1]
+ ")");
            System.out.println("Puntos alineados con este centro: " + circleParams[2]);
        } else {
            System.out.println("No se detectó claramente una circunferencia.");
        }

        scanner.close();
    }
}
```

Cómo funciona

1. El programa solicita el ingreso de un radio previamente conocido, y un conjunto de pares de coordenadas (x, y) El ingreso finaliza cuando el usuario introduce **q**.
2. A partir de estas entradas, cada punto ingresado vota en un espacio acumulador por los posibles centros que darían lugar a una circunferencia con dicho radio.
3. Esto se realiza recorriendo todos los ángulos (de 0° a 359°) alrededor de cada punto y determinando qué posiciones podrían corresponder al centro.
4. Finalmente, el punto con la mayor acumulación de votos indica el centro más probable de la circunferencia que se ajusta mejor a los puntos ingresados.

Salida (de ejemplo)

```
Ingrese el radio conocido para la circunferencia: 4
Ingrese pares de coordenadas (x ; y).
Puede dejar de introducir valores en cualquier momento, introduciendo la letra 'q'.
(x[1] ; y[1]) = 0 4
(x[2] ; y[2]) = 4 0
(x[3] ; y[3]) = -4 0
(x[4] ; y[4]) = 0 -4
(x[5] ; y[5]) = q

Valores ingresados:
[1] (x ; y) = (0,0 ; 4,0)
[2] (x ; y) = (4,0 ; 0,0)
[3] (x ; y) = (-4,0 ; 0,0)
[4] (x ; y) = (0,0 ; -4,0)

Centro detectado de la circunferencia: (a ; b) = (0 ; 0)
Puntos alineados con este centro: 60

Process finished with exit code 0
```

En el ejemplo, ingresamos puntos que se corresponden a una circunferencia de radio 4 centrada en el origen. Vemos que el centro $(a; b)$ es detectado correctamente, y lo que (a primera vista) podría llamar la atención es el hecho de que se detectan 60 puntos, pero ingresamos solo 4 puntos. Bien, esto es así porque el acumulador en la posición $(0; 0)$ recibió 60 votos. ¿Y por qué? Porque en cada punto ingresado, el algoritmo recorre todos

los ángulos de $0°$ a $359°$, y para cada ángulo calcula un posible centro $(a; b)$ por el cual se vota. Si el centro calculado cae en $(0; 0)$, suma un voto al acumulador en esa celda.
Entonces, si ingresamos 4 puntos, tenemos que cada uno recorrió 360 ángulos, así:
$4 * 360 = 1440$ votos posibles, de los cuales 60 coincidieron exactamente con el punto $(0; 0)$.

Características

1. **Interactivo**: el prototipo solicita al usuario, el ingreso de pares numéricos para las coordenadas (x, y).
2. **Flexible**: maneja tipos de datos enteros y decimales (*double*).
3. **Radio conocido**: el usuario especifica un radio conocido, simplificando la representación de la circunferencia en un espacio 2D (coordenadas del centro).
4. Programa de consola (**CLI**).

Ventajas

1. Como el radio se conoce previamente, se simplifica el resto de los datos a dos dimensiones, y esto se traduce en eficiencia computacional.
2. El código es sencillo y fácil y posibilita una fácil comprensión del método de Transformada de Hough para circunferencias.
3. Facilidad para pruebas y validación: La interacción con el usuario facilita la realización de pruebas rápidas.
4. Claridad en los resultados: La detección clara del centro es inmediata, proporcionando el máximo número de puntos que se alinean con dicho centro.
5. Eficiente para pruebas pequeñas (como las de entornos educativos y demostrativos).

Limitaciones

1. Dependencia del radio conocido: No sirve cuando el radio varía o es desconocido.
2. Sensible al ruido: los puntos aislados o mal colocados podrían generar centros incorrectos, requiriendo estrategias adicionales de filtrado.

3. Sobre el uso de memoria (y rendimiento): la matriz acumuladora crece en tamaño conforme aumenta la resolución o tamaño del área considerada.

Dificultades

1. Dado que en Java prescindí de un entorno gráfico y en su lugar, decidí trabajar con CLI, se hicieron necesarias validaciones, excepciones y validaciones para asegurar que las entradas de los usuarios fueran adecuadas.
2. Se debe planificar previamente (en papel y lápiz, como se decía), los valores a ingresar, para evitar desbordes o indetecciones.
3. El manejo de errores y detecciones de entradas del usuario para garantizar estabilidad (propia de las interfaces CLI), así como la implementación de su lógica, toman su tiempo.

3.4. Balance de los tres enfoques tratados

En cuadro comparativo (en página siguiente):

Desde el punto de vista de las ventajas vs. los inconvenientes:

Enfoque	Ventajas	Inconvenientes
Transformada de Hough para rectas	**Implementación sencilla y rápida:** método directo, robusto y eficaz.	**Limitado a patrones rectilíneos:** no resuelve detección de figuras curvas directamente.
	Alta precisión en detección de líneas rectas: ideal para patrones lineales claros.	**Sensibilidad a la resolución y discretización:** depende de la elección correcta del espacio acumulador.
	Robustez ante discontinuidades y puntos faltantes: puede manejar errores leves en la posición.	**Escalabilidad en imágenes complejas:** consume muchos recursos en entornos grandes o con alta resolución.
Transformada de Hough para circunferencias (radio conocido)	**Adecuada para figuras curvas:** especialmente útil para localizar centros con precisión.	**Dependencia del radio previamente conocido:** no aplicable si el radio varía o es desconocido.
	Eficiencia en casos de radio conocido: simplificación importante en complejidad computacional.	**Alta sensibilidad al ruido significativo:** puntos aislados pueden generar detecciones erróneas.
	Precisión potencialmente alta: método determinista con resultados claros y confiables.	**Escalabilidad en problemas complejos:** similar al caso anterior, requiere consideraciones adicionales en imágenes grandes o complicadas.
Modelo de Hopfield (redes neuronales)	**Capacidad de corrección automática de imágenes:** recuperación efectiva ante imágenes parcialmente dañadas o distorsionadas.	**Limitada capacidad de almacenamiento:** directamente proporcional al número de neuronas disponibles.
	Robustez ante ruido leve: muy útil para situaciones donde se requieren pequeñas correcciones automáticas.	**Limitación en precisión para entornos industriales exigentes:** requiere alta complejidad para ofrecer precisión suficiente.
	Adaptabilidad y aprendizaje: puede entrenarse con distintos patrones y adaptarse a nuevas situaciones.	**Complejidad de escalabilidad real:** mayor complejidad en modelos avanzados, necesidad de múltiples capas para manejar imágenes industriales reales.

Desde el punto de vista del criterio:

Criterio	Hough (Rectas)	Hough (Circunferencias)	Hopfield (redes neuronales)
Implementación	Sencilla y directa	Relativamente simple	Compleja y necesita entrenamiento
Precisión	Alta para patrones lineales	Alta para centros circulares	Limitada en alta precisión

Criterio	Hough (Rectas)	Hough (Circunferencias)	Hopfield (redes neuronales)
Robustez al ruido	Moderada-Alta	Moderada	Alta en correcciones pequeñas
Flexibilidad	Limitada a rectas	Limitada a círculos (de radio conocido)	Flexible por aprendizaje
Escalabilidad	Moderada	Moderada	Baja (alta complejidad al escalar)
Aplicabilidad industrial	Moderada (problemas específicos)	Buena para casos definidos	Baja en versión básica

Recomendación

Teniendo en cuenta el contexto industrial descrito, en el cual se busca precisión absoluta en la detección automática de la posición del block sobre la cinta transportadora, y considerando especialmente la restricción clave asociada a los costos del proyecto (que nos imposibilita recomendar “todos” o “un mix de todo”), la solución recomendada es la **Transformada de Hough para circunferencias (de radio conocido)**.

Justificación

El problema específico consiste en identificar con precisión el centro de una circunferencia conocida (aro "**C**") en la pieza a ensamblar. Y la Transformada de Hough para circunferencias, especialmente con radio conocido, encaja como anillo al dedo en nuestro caso de estudio, ya que podemos obtener resultados deterministas, rápidos y fiables. Es decir, esto sería nuestra adecuación técnica.

En principio, de tener que elegir entre los tres enfoques, tal vez de forma intuitiva optaríamos por el modelo de Hopfield, o bien por un híbrido que combine que combine métodos neuronales adaptativos y las transformadas de Hough(tanto para rectas como para circunferencias). Pero las restricciones presupuestarias de un entorno industrial local (que no es tan grande ni desarrollado como en otros lugares del mundo), buscan y agradecen un balance ideal entre rendimiento, precisión y costo, y en este sentido, la Transformada de Hough para circunferencias es la solución recomendada.

4. Repositorio en Github

Accesos directos:

- **Repositorio de la materia:** https://github.com/linkstat/ia
- **Este documento (PDF):** https://github.com/linkstat/ia/raw/refs/heads/master/docs/HAMANN-PABLO-ALEJANDRO-TP4.pdf