Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto Terminal

---

# Implementación de tres algoritmos para desplazar un robot en el plano, en presencia de obstáculos

---

Tapia de la Rosa Alfonso
208303632

Trimestre 2013-Invierno

Fecha de Entrega 15 de Abril de 2013

---

Asesor: Dra. Dolores Lara Cuevas,
Profesor Asociado C, Departamento de Sistemas

Asesor: Dr. Francisco Javier Zaragoza Martínez,
Profesor Titular B, Departamento de Sistemas

Tabla de contenido

## Resumen

En este proyecto se aplicaron distintos algoritmos para planificación de un robot, basada en mapas trapezoidales y Dijkstra para la construcción de las rutas de navegación del robot. El propósito de la planificación, consiste en que los algoritmos construyan una ruta de navegación libre de colisiones, que lo lleve desde un punto inicial hasta un punto final.

Llegando a la simplificación de resolver como debe moverse un robot por una escena con obstáculos. Este objetivo se consigue con la gráfica de visibilidad, de esta manera identifica el área en donde se moverá y en dónde están los obstáculos después la serie de algoritmos que son base para llegar a esta solución del problema.

En los grafos de visibilidad, cada vértice está conectado con todos sus vértices visibles pudiendo elegir el más conveniente, es decir, el camino más corto usando el algoritmo de Dijkstra.


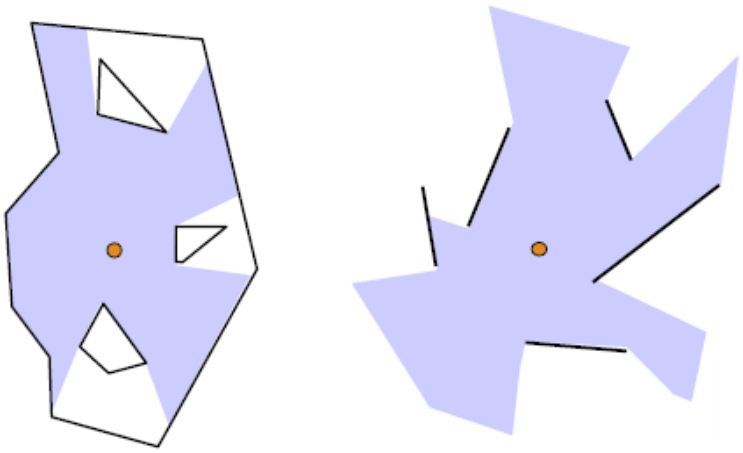

*Ilustración 1. El área de visibilidad después de ejecutar los grafos de visibilidad*

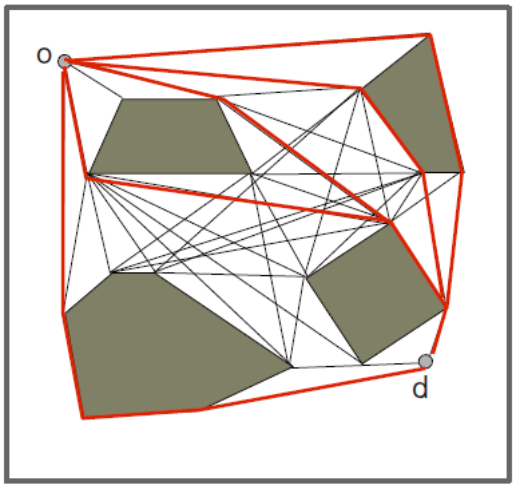


*Ilustración 2. Robot y meta con las posibles rutas*

# 1. Objetivos

Implementar software para resolver el problema de desplazar un robot en el plano. En este tipo de problemas, dado un escenario con obstáculos, se desea encontrar la ruta más corta para desplazar un robot desde un punto de inicio S, hasta un punto meta T; de tal forma que el robot no colisione con ningún obstáculo.

## 1.1. Objetivos específicos

1. Implementar un módulo tal que, cuando el robot es un punto, obtenga la ruta más corta por la cual _este se debe desplazar en el plano, en presencia de un escenario con obstáculos. De tal forma que se minimice la distancia euclidiana recorrida por el robot, y que este no intersecte ningún obstáculo.

2. Implementar un módulo tal que, cuando el robot sea un polígono, obtenga la ruta más corta por la cual _este se debe desplazar en el plano, en presencia de un escenario con obstáculos. De tal forma que se minimice la distancia euclidiana recorrida por el robot, y que este no intersecte ningún obstáculo. En este caso nuestro robot presentara una nueva desventaja, la cual es tener área, con lo que podría llegar a darse el caso de que no exista una ruta hacia el punto destino. El polígono solo puede trasladarse.

3. Implementar un módulo que grafiqué la solución para cada uno de los casos explicados anteriormente.

# 2. Introducción

La meta a alcanzar de este proyecto es implementar métodos algorítmicos para desplazar un robot en el plano (robot motion planning). Es decir, realizar el trazado del camino óptimo de un robot dentro de un mapa de obstáculos. Una ruta es óptima, si es la que garantiza que el robot llegue a su destino recorriendo la menor distancia euclidiana posible. Es decir, que no hay otra ruta que lleve al robot a su destino y que recorra menos distancia. La aplicación permitirá construir un escenario sobre el cual se movera nuestro robot, para que posteriormente se realice la optimización y trazado de la trayectoria del mismo. Se permitira observar la solución gráficamente.

Los algoritmos que implementaremos pertenecen al área de investigación conocida como Geometría Computacional. No pretendemos en esta propuesta hablar en extenso del área, pero es importante mencionar que los algoritmos geométricos serán la parte vertebral del proyecto terminal. En este proyecto trabajaremos entonces únicamente con objetos geométricos, como son puntos y polígonos.

## 3. Desarrollo del Proyecto

Este proyecto esta basado en la Planificación de Trayectorias (Path Planning) que es un apartado de la planificación de movimientos(motion Planning) el cual estudia el desplazamiento autónomo de un objeto móvil (un robot) en una escena con obstáculos desde un punto origen a un punto destino sin colisionar con ninguno de ellos.

El problema se vuelve complejo con los grados de libertad del robot y el tipo de escenario (2D/3D), por esta razón simplificaremos a decir que el robot se mueve en un espacio libre en 2D y no puede rotar. El primer paso es crear la estructura de los mapas trapezoidales, pues estos construyen las particiones del espacio libre en una escena.

Cada trapecio corresponde a una región libre de obstáculos de la cual sabemos sus dimensiones y las regiones adyacentes que están o no están libres de obstáculos.

Se crea un camino libre de obstáculos uniendo puntos en el interior de cada trapecio. El objeto móvil solo debe seguir estos caminos pre-calculados.

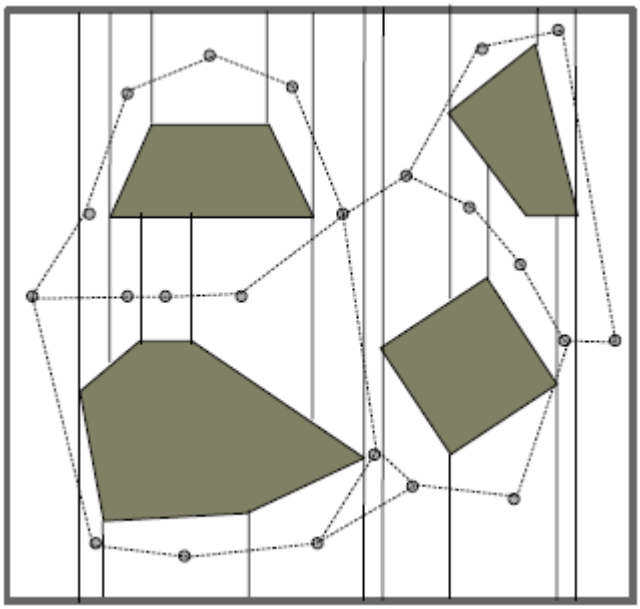

*Ilustración 3 Ejemplo del mapa trapezoidal*

Siguiendo el marco teórico de esta estructura de datos se crea la primera aplicación de mapas trapezoidales que posteriormente se replicaría en las versiones finales del proyecto.

Al ejecutar el código, el primer paso es insertar todos los segmentos que conformaran el escenario de nuestro robot.

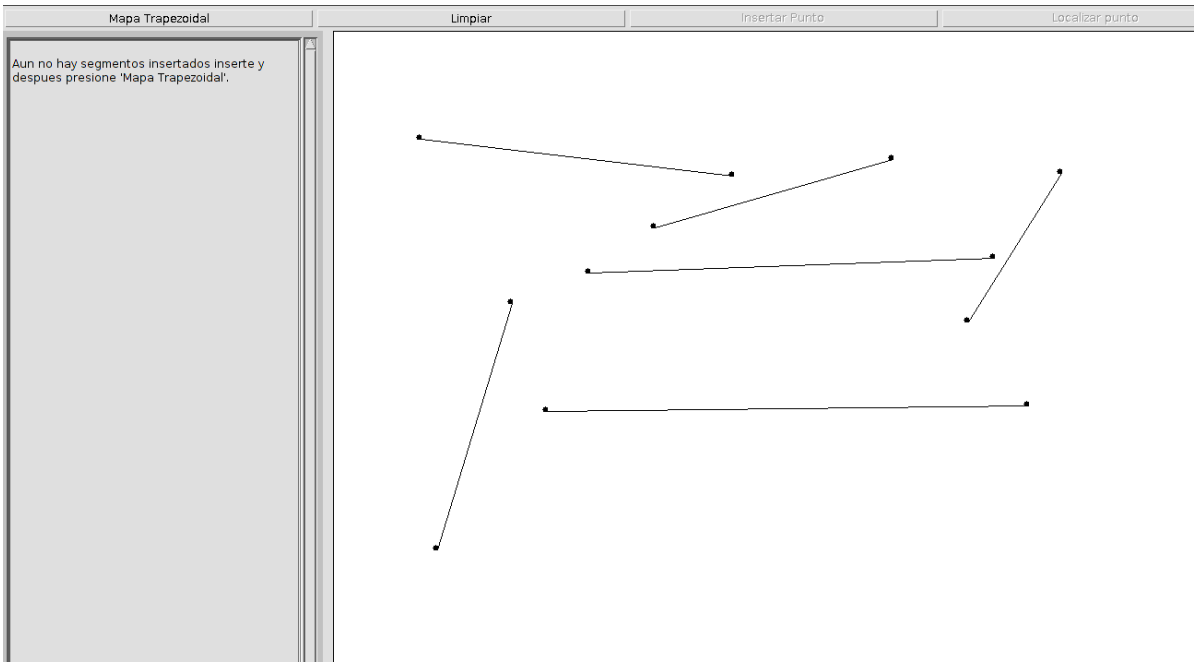


*Ilustración 4. Inserción de los segmentos del escenario para obtener los mapas trapezoidales*

El código que regula esta parte es el siguiente:

```
class Segment
{
   public Point left;    public Point right;

   public Segment(Point p1, Point p2)
   {
      if(p1.x < p2.x)
      {
         left = p1;     right = p2;
         return;
      }
      if(p1.x > p2.x)
      {
         left = p2;    right = p1;
         return;
      }
      if(p1.y < p2.y)
      {
         left = p1;    right = p2;
         return;
      } else
      {
         left = p2;   right = p1;
         return;
      }
   }
```

Posteriormente, Ejecutamos el algoritmo de los mapas trapezoidales para construir la estructura e identificar los puntos, segmentos y trapecios que se forman, esto permitirá conocer el escenario y saber cada región como será nombrada.

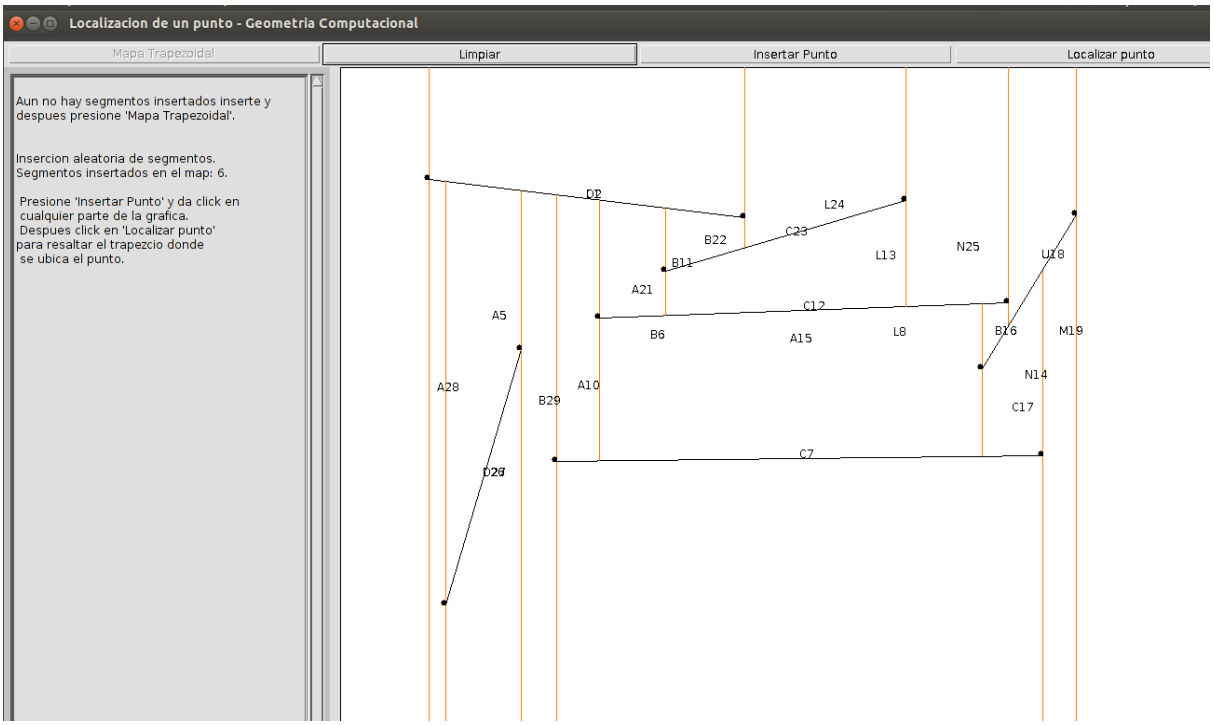


*Ilustración 5. Construcción del mapa trapezoidal, después de la inserción de segmentos.*

Esta búsqueda en la estructura es basada en las funciones del código que coloco a continuación.

```
public boolean createMap(Segment segments[], int i)
  {
     clear();
     parent.repaint();
     if(i > 0)
     {
        numSegments = i;
        computeBoundingBox(segments, numSegments);
        Point p1 = new Point(minX - over, minY - over, -1);
        Point p2 = new Point(maxX + over, minY - over, -1);
        Point p3 = new Point(minX - over, maxY + over, -1);
        Point p4 = new Point(maxX + over, maxY + over, -1);
        Segment lsegment = new Segment(p1, p2);
        Segment lsegment1 = new Segment(p3, p4);
        T = new Trapezoid();
        T.name = "root";
        T.top = lsegment;
        T.bottom = lsegment1;
        T.left = p3;
        T.right = p2;
```

```
        T.lRight = null;
        T.uRight = null;
        T.lLeft = null;
        T.uLeft = null;
        D = new Node();
        D.node = "t";
        D.t = T;
        T.node = D;
        numTrapezoids++;

        infoArea.addItem("Insercion aleatoria de segmentos.");
        S = randPermutation(segments, numSegments);

        insertSegments(S, numSegments);

        infoArea.addItem("Segmentos insertados en el map: " + numSegments+".");
    }
    return true;
}
```

Ahora si teniendo lo anterior es fácil localizar un punto en el escenario del robot, para esto hay un apartado de prueba, en este insertamos un punto de prueba, el cual es indiferente si es el robot o el punto destino pues solo es para poner a prueba la estructura de datos.

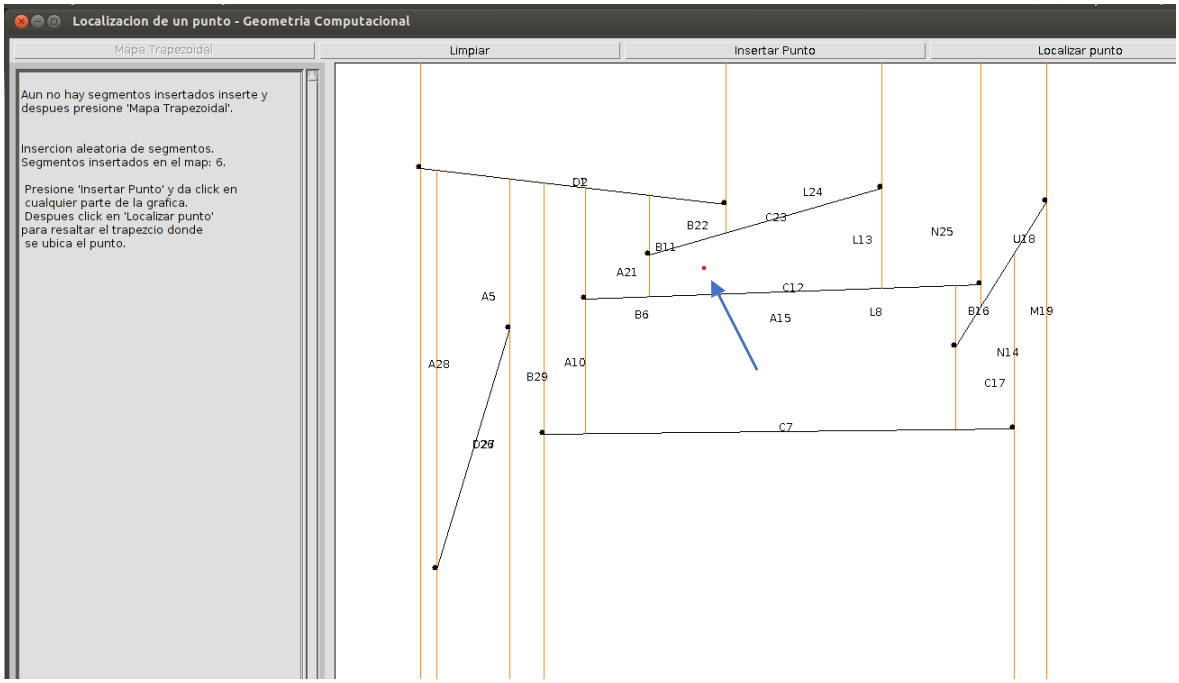


*Ilustración 6. La Flecha señala el punto insertado para la prueba de búsqueda.*

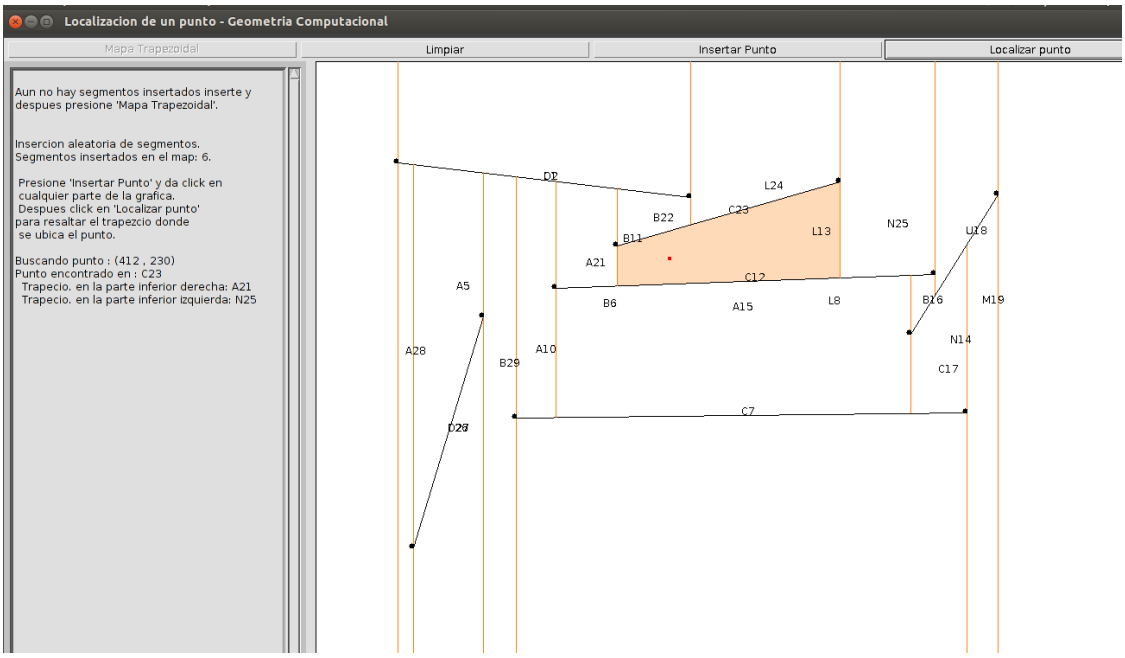


*Ilustración 7. Ejecutamos la búsqueda y el trapecio donde está el punto cambia de color para identificarlo.*

```
private Node find(Node node, Point p, Point q)
   {
Node nodeFound;
 nodeFound = node;
       while(true)
            {
            while(nodeFound.node == "x")
                    {
                if(nodeFound.p != null && nodeFound.isLeft(p))
                            {
               nodeFound = nodeFound.left;
                                    } else {
               nodeFound = nodeFound.right;
                                            }
                                                    }
           if(nodeFound.node == "y")
                    {
if(Computation.counterClockWise(nodeFound.segment.left, nodeFound.segment.right, p) == -1 ||
Computation.counterClockWise(nodeFound.segment.left,  nodeFound.segment.right,  p)  ==  0  &&
Computation.counterClockWise(nodeFound.segment.left, nodeFound.segment.right, q) == -1)
           nodeFound = nodeFound.left;
                    else
                                nodeFound = nodeFound.right;
                                        } else {
                                        return nodeFound;
                                                }
                                                   }
                                                        }
```

```
public boolean findPoint(Point point)
  {
     infoArea.addItem("\nBuscando punto : ("+point.x+" , "+point.y+")");
     long timerStart = System.currentTimeMillis();
        Node node = D;
        while(true)
        {
        if(node.node == "x")
        {
           if(node.isLeft(point))
                    {
              node = node.left;
                    } else
                              {
                    node = node.right;
                                }
           continue;
                              }
        if(node.node != "y")
           break;
        if(Computation.isAbove(node.segment.left, node.segment.right, point))
            {
           node = node.left;
            } else
                    {
                    node = node.right;
                    }
                                }
     long timerEnd = System.currentTimeMillis();
     long runningTime = Math.abs(timerEnd - timerStart);
     System.out.println("----->>>Tiempo en ejecutar busqueda un punto : "+runningTime+" ms");
     infoArea.addItem("Punto encontrado en : " + node.t.name);
     if(node.t.uLeft != null)
         infoArea.addItem("  Trapecio. en la parte alta derecha: " + node.t.uLeft.name);
     if(node.t.lLeft != null)
         infoArea.addItem("  Trapecio. en la parte inferior derecha: " + node.t.lLeft.name);
     if(node.t.uRight != null)
          infoArea.addItem("  Trapecio. en la parte alta izquierda: " + node.t.uRight.name);
     if(node.t.lRight != null)
         infoArea.addItem("  Trapecio. en la parte inferior izquierda: " + node.t.lRight.name);
         drawTRFound(node.t);
         return true;
                                    }
```

Esta es la parte fundamental del problema, y los mapas trapezoidales son los que construirán la estructura de datos base para resolverlo.
Lo siguiente es crear el grafo de visibilidad para obtener las rutas y elegir el camino mínimo.

El Teorema dice: el camino más corto uniendo los puntos origen y destino es una poli línea del grafo de visibilidad.

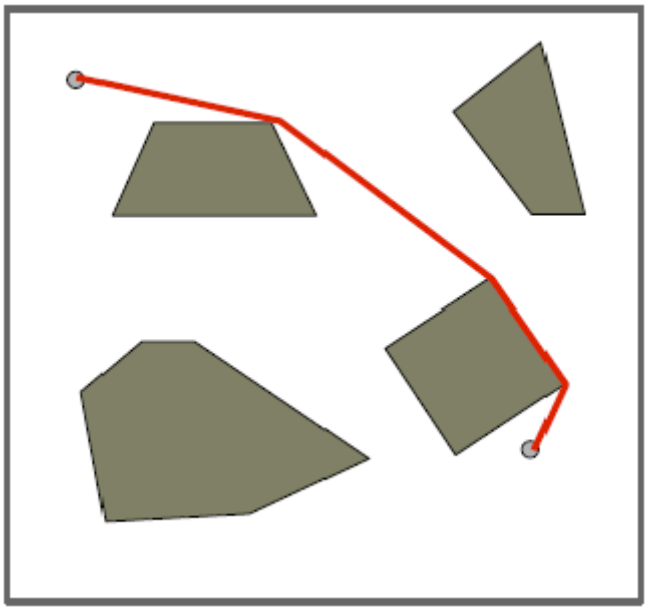

*Ilustración 8. Ejemplo del Grafo de visibilidad*

De esta manera  detallamos el Algoritmo para encontrar el camino mínimo:

1. Construir el grafo de visibilidad del conjunto S de objetos junto con los puntos o y d, siendo de o y d los puntos de inicio y destino.

2. El peso entre dos vértices en el grafo de visibilidad, VG (Sv,Sw), se inicializa con la longitud de la diagonal entre los vértices Sv y Sw

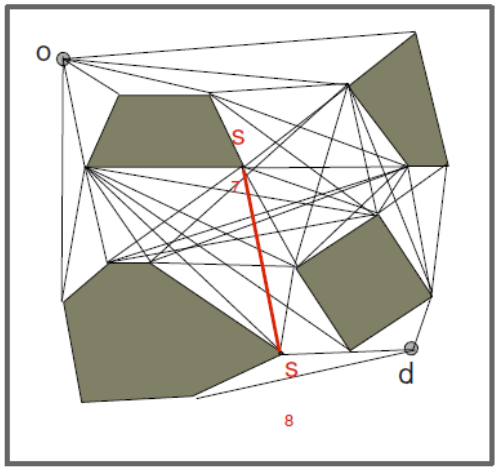


*Ilustración 9. Primer paso del Grafo de Visibilidad, este será ejecutado sobre todos los segmentos.*

3. Utilizar el algoritmo de Dijkstra para encontrar el camino mínimo entre los puntos o y d.

   El algoritmo de Dijkstra encuentra todos los caminos posibles desde el punto origen al destino, sumando las aristas y quedándose finalmente con el camino más corto.

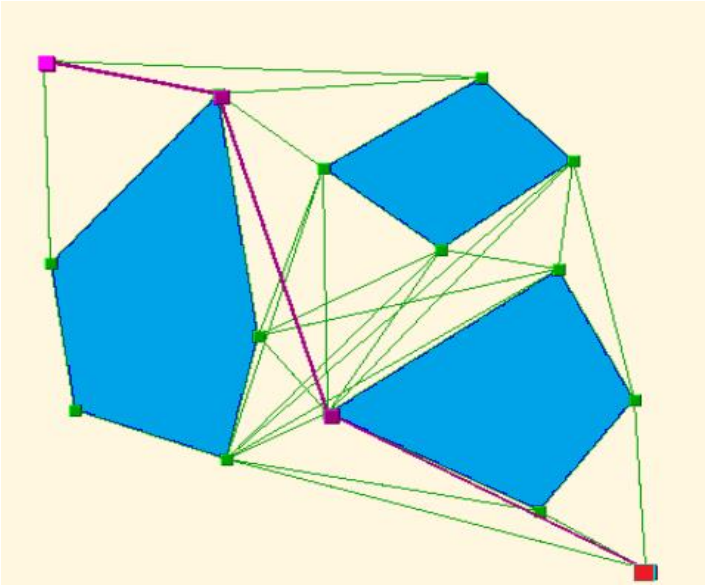

*Ilustración 10. Elección del camino más corto.*

Es importante remarcar que cuando esto aplica cuando nuestro robot es considerado un punto, pero si este fuera un polígono tendremos que involucrar un algoritmo más.

Los robots son considerados objetos puntuales en vez de polígonos y los obstáculos crecen el tamaño del objeto móvil (sumas de Minkowski). Al tener esto en cuenta aplicamos los algoritmos anteriores de mapas trapezoidales y grafo de visibilidad, pues las sumas de Minkowski hacen ver al robot que es un polígono como un punto.

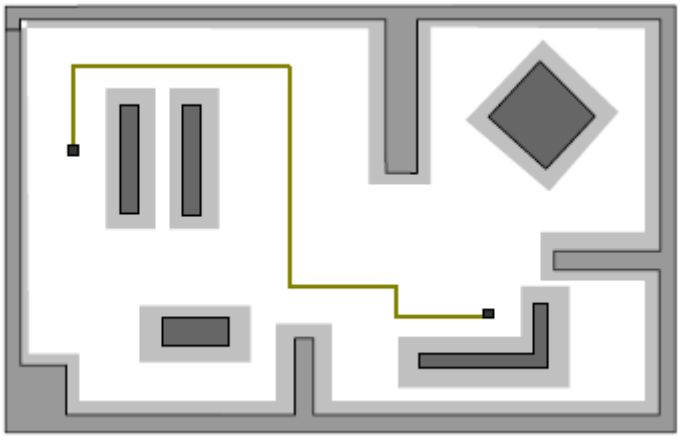

*Ilustración 11. L a línea Gris claro alrededor de los obstáculos representa las sumas de Minkowski.*

Una vez definida la utilización del Grafo de Visibilidad es posible llevarlo a la práctica en nuestra aplicación.

Utilizando el código de mapas trapezoidales, se crea un escenario y la estructura para navegar y obtener las rutas posibles, para que finalmente se trace la ruta final que deberá seguir el robot.

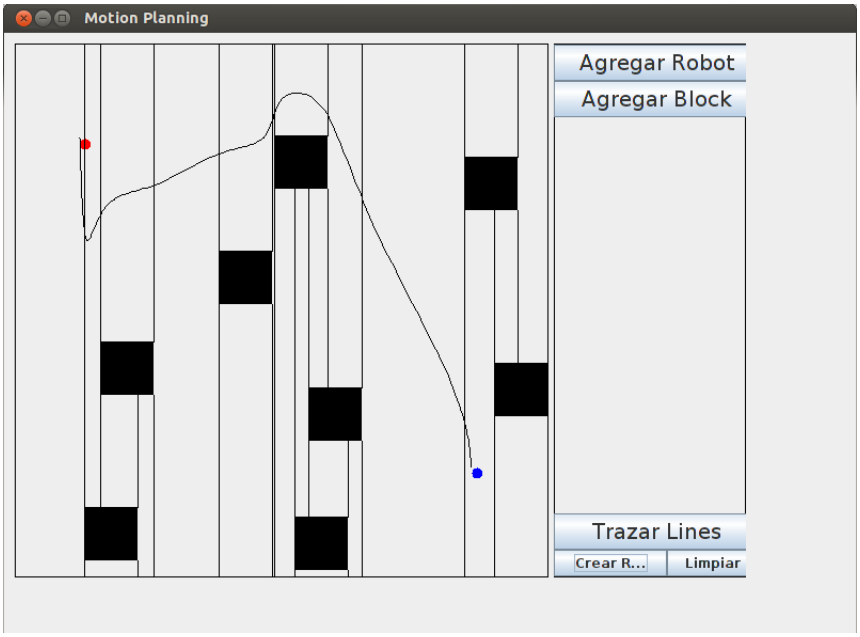


*Ilustración 12. Aplicación después de agregar las técnicas y algoritmos definidos.*

La clase que conjunta las bases del desarrollo final del proyecto es la siguiente:

```
public class RobotPanel extends JPanel{

        ArrayList<Robot> robots;
        ArrayList<Block> blocks;
        ArrayList<Line> lines;
        ArrayList<Cell> cells;
        ArrayList<Cell> paths;
        ArrayList<double[]> splines;

        //temp for debug
        Cell start_global=null; Cell end_global=null;
        double[] c = new double[15];
        double[] spline1;
         double[] spline2;
        double[] spline3;
        boolean draw_lines;
```

```
public RobotPanel(){

            this.setLayout(new GridLayout(10,1));

            this.setBorder(BorderFactory.createLineBorder(Color.blac
            k));
            this.setSize(new Dimension(500,500));
            this.setPreferredSize(new Dimension(500,500));
            this.setVisible(true);

            robots = new ArrayList<Robot>();
            blocks = new ArrayList<Block>();
            lines = new ArrayList<Line>();
            cells = new ArrayList<Cell>();
            splines = new ArrayList<double[]>();
            paths = new ArrayList<Cell>();
            draw_lines = false;
                        }

public double[] spline (ArrayList<Cell> shortestPath)
                    {
            //c contains the points which will be interpolated
            c = new double[shortestPath.size()*3];
            int y = 0;
            for(Cell cell : shortestPath){
            c[y] = cell.getX();
            c[y+1] = cell.getY();
            c[y+2] = 0.0;
            y+=3;
                        }

            spline1 = SplineFactory.createBezier (c,20);
            //Different Interpolation Techniques
            spline2 = SplineFactory.createCubic (c,      20);
            System.out.println ("-- Cubic");
            for (int i = 0; i < spline2.length; i+=3)
            System.out.println(spline2[i] + "," + spline2[i+1] + ","
            + spline2[i+2]);
            return spline2;
                              }

public void paintComponent(Graphics g)
            {
      super.paintComponent(g);
      Graphics2D g2d = (Graphics2D) g;
      for(Robot rob: robots){
      g2d.setColor(Color.red);
      g2d.fillOval((int)rob.getStartX(),  (int)rob.getStartY(),  10,
      10);
      g2d.setColor(Color.blue);
```

```
            if(rob.getEndX()>0 && rob.getEndY()>0)
      g2d.fillOval((int)rob.getEndX(), (int)rob.getEndY(), 10, 10);
                  }
      g2d.setColor(Color.black);
            for(Block block: blocks)
                  {
g2d.fillRect((int)block.getPosX(),(int)block.getPosY(),block.getSi
ze(), block.getSize());
                  }
      if(draw_lines)
                  {
                  g2d.setColor(Color.black);

      for(Line line: lines)
                        {
g2d.drawLine((int)line.start_x, (int)line.start_y, (int)line.end_x,
(int)line.end_y);
                        }
                              }

      if(splines!=null && splines.size()!=0)
                  {
            System.out.println("Splines is not NULL");
      for(int x = 0; x < splines.size(); x++)
                  {
            System.out.println("There is at least 1 Spline");

      for(int y = 0; y < splines.get(x).length - 3; y+=3)
                  {
            System.out.println("Drawing a line in the Spline");

            g2d.drawLine((int)splines.get(x)[y],
            (int)splines.get(x)[y+1],
            (int)splines.get(x)[y+3],
            (int)splines.get(x)[y+4]);
                  }
                        }
                              }
                                    }
public void addRobot(Robot rob)
        {
           robots.add(rob);
        }

public void addBlock(Block block)
        {
           boolean add = true;
      for(int x = 0; x < blocks.size(); x++)
                  {
      if(block.getPosX()>= blocks.get(x).getPosX()-block.getSize()
      && block.getPosX()<= blocks.get(x).getPosX() +
      blocks.get(x).getSize())   {
```

```
      if(block.getPosY()>= blocks.get(x).getPosY()-block.getSize()
      && block.getPosY() <= blocks.get(x).getPosY() +
      blocks.get(x).getSize())   {
                  add = false;
                  System.out.println("Block cannot be drawn here!");
                  x = blocks.size()+1;
                  return;
                              }
                                    }
                                          }
                        if(add){
                           blocks.add(block);
                                }
                              lines.clear();
                                                }

public Line upLine(double x, double y, Block block)
{
      Line new_line = new Line();
      new_line.start_x = x;
      new_line.end_x = x;
      new_line.start_y = 0;
      new_line.end_y = y;
            //checar insercion
            for(Block test_block:blocks)
                  {
            if(!block.equals(test_block))
                  {
            if(x>=test_block.getPosX()&&
            x<=test_block.getPosX()+test_block.getSize() &&
            test_block.getPosY()+test_block.getSize()<y)
                  {
            if(test_block.getPosY()+test_block.getSize()>new_line.st
            art_y)
                  {
      new_line.start_y = test_block.getPosY()+test_block.getSize();
                  }
                        }
                              }
                                    }
            if(new_line.end_y<new_line.start_y)
                  {
            double temp = new_line.start_y;
            new_line.start_y = new_line.end_y;
            new_line.end_y = temp;
                  }
                  return new_line;
                   }
```

```
public Line downLine(double x, double y, Block block)
        {
           Line new_line = new Line();
           new_line.start_x = x;
           new_line.end_x = x;
           new_line.start_y = y;
           new_line.end_y = 500;

      for(Block test_block:blocks)
                  {
      if(!block.equals(test_block))
                  {
      if(x>=test_block.getPosX()&&
      x<=test_block.getPosX()+test_block.getSize() &&
      test_block.getPosY()>y)
                  {
      if(test_block.getPosY()<new_line.end_y)
                  {
             new_line.end_y = test_block.getPosY();
                   }
                        }
                              }
                                  }
      if(new_line.end_y<new_line.start_y)
                  {
                           double temp = new_line.start_y;
                           new_line.start_y = new_line.end_y;
                           new_line.end_y = temp;
                  }
                              return new_line;
                                        }
public void compute_lines()
           {
      for(Block block:blocks)
                  {
      Line topLeft = upLine(block.getPosX(),block.getPosY(),block);
      Line bottomLeft = downLine(block.getPosX(),
      block.getPosY()+block.getSize(), block);
      Line topRight = upLine(block.getPosX()+block.getSize(),
      block.getPosY(),block);
      Line bottomRight = downLine(block.getPosX()+block.getSize(),
      block.getPosY()+block.getSize(),block);
      lines.add(topLeft);
      lines.add(bottomLeft);
      lines.add(topRight);
      lines.add(bottomRight);
                  }
                        }
```

```
public boolean lines_intersect(double x1, double y1,double x2,
double y2, double x3, double y3, double x4, double y4)
        {
      if((x1==x2 && x2==x3 && x3==x4) || (y1==y2 && y2==y3 && y3==y4))
            return true;
            double   ua   =   ((x4-x3)*(y1-y3)-(y4-y3)*(x1-x3))/((y4-
            y3)*(x2-x1)-(x4-x3)*(y2-y1));
             double   ub   =   ((x2-x1)*(y1-y3)-(y2-y1)*(x1-x3))/((y4-
            y3)*(x2-x1)-(x4-x3)*(y2-y1));

      if((ua >0 && ua<1) && (ub>0 && ub<1))
            return true;
      else
            return false;
                              }
public void create_cells()
      {
            cells.clear();
      for(int i = 0; i<lines.size()-1; i++)
                  {
      for(int j=i+1; j<lines.size(); j++)
                  {
            Line line1 = lines.get(i);
            Line line2 = lines.get(j);
      if(line2.start_x < line1.start_x)
                  {
            lines.set(i,line2);
            lines.set(j,line1);
                  }
                        }
                              }
      for(Line line:lines)
                  {
            cells.add(new Cell(line));
                  }
      for(Cell cell: cells)
                  {
      for(Cell cell2: cells)
                   {
      if(!cell.equals(cell2) && cell.getX()<cell2.getX()){
            if(!cell.getAdjacent().isEmpty()   &&   cell2.getX()   >
            cell.getAdjacent().get(0).getX())
                  break;
                  boolean collision = false;
     for(Block block: blocks){

      if(lines_intersect(cell.getX(),cell.getY(),cell2.getX(),cell2
      .getY(),block.getPosX(),block.getPosY(),block.getPosX()+block
      .getSize(),block.getPosY()) ||
      lines_intersect(cell.getX(),cell.getY(),cell2.getX(),cell2.ge
      tY(),block.getPosX(),block.getPosY(),block.getPosX(),block.ge
      tPosY()+block.getSize()) ||
```

```
      lines_intersect(cell.getX(),cell.getY(),cell2.getX(),cell2.ge
      tY(),block.getPosX()+block.getSize(),block.getPosY(),block.ge
      tPosX()+block.getSize(),block.getPosY()+block.getSize()) ||
      lines_intersect(cell.getX(),cell.getY(),cell2.getX(),cell2.ge
      tY(),block.getPosX(),block.getPosY()+block.getSize(),block.ge
      tPosX()+block.getSize(),block.getPosY()+block.getSize()))
            {
                  collision = true;
                  break;
            }
                  }
      if(!collision)
            cell.addAdj(cell2);
                  }
                        }
                              }
                                    }
public double distance(Cell L1, Cell L2)
      {
      double dist = Math.sqrt((Math.pow(L1.getX()- L2.getX(),2) +
      Math.pow(L1.getY()-L2.getY(),2)));
      return dist;
       }

public Cell Dijkstra(Robot robot)
       {
            ArrayList<Cell> shortestPath = new ArrayList<Cell>();
            ArrayList<Cell> cells_copy = new ArrayList<Cell>();
      for(int i = 0; i<cells.size(); i++)
            {
            Cell cell = cells.get(i);
            cells_copy.add(cell);
            }
            Cell start = new Cell(robot.getStartX(),
            robot.getStartY(),0);
            start_global = start;
      for(Cell cell: cells_copy){
      if(!start.getAdjacent().isEmpty() && cell.getX() >
      start.getAdjacent().get(0).getX())
                  break;
            else
                  {
                  if(cell.getX() > start.getX() && cell.line.start_y
                  < start.getY())
                              start.addAdj(cell);
                  }
                        }
      cells_copy.add(start);
      Cell end = new
      Cell(robot.getEndX(),robot.getEndY(),100000000);
      end_global=end;
      Cell found = null;
```

```
      for(int i=cells.size()-1; i>-1; i--)
                {
                      Cell cell = cells.get(i);
      if(cell.getX()<end.getX()){
      if(found!=null  && cell.getX()< found.getX())
                      break;
                      boolean collision = false;
      for(Block block: blocks)
           {
      if(lines_intersect(end.getX(),end.getY(),cell.getX(),cell.get
      Y(),block.getPosX(),block.getPosY(),block.getPosX()+block.get
      Size(),block.getPosY())   ||
      lines_intersect(end.getX(),end.getY(),cell.getX(),cell.getY()
      ,block.getPosX(),block.getPosY(),block.getPosX(),block.getPos
      Y()+block.getSize())      ||
      lines_intersect(end.getX(),end.getY(),cell.getX(),cell.getY()
      ,block.getPosX()+block.getSize(),block.getPosY(),block.getPos
      X()+block.getSize(),block.getPosY()+block.getSize())  ||
      lines_intersect(end.getX(),end.getY(),cell.getX(),cell.getY()
      ,block.getPosX(),block.getPosY()+block.getSize(),block.getPos
      X()+block.getSize(),block.getPosY()+block.getSize()))
            {
                collision = true;
            }
                }
      //if(cell.getX()<end.getX() && cell.line.end_y > end.getY())
      if(!collision)
           {
                cells_copy.get(i).addAdj(end);
                found = cell;
           }
                }
                      }
      cells_copy.add(end);
      while(!cells_copy.isEmpty()){
                //Find minimum
                Cell min = null;
      for(Cell line: cells_copy)
           {
      if(min == null)
                min=line;
           else
                if(line.path<min.path)
                          min=line;
                                     }
cells_copy.remove(min);
shortestPath.add(min);
      for(Cell adj: min.getAdjacent())
           {
      if(adj.path > min.path + distance(adj,min))
           {
                adj.path = min.path + distance(adj,min);
```

```
                adj.parent = min;
                                }
                                        }
                                                }
      if(end.parent == null)
            end.parent = start;
            //return shortestPath;
            return end;
                                                                }

public void CreatePaths()
        {
           splines.clear();
           paths.clear();
           create_cells();
           splines = new ArrayList<double[]>();
      for(Robot robot: robots)
           {
      if(robot.getEndX()>0 && robot.getEndY()>0 &&
      robot.getStartX()>0 && robot.getEndY()>0){
           Cell shortestPath = Dijkstra(robot);
           //Use these cells to draw interpolated shortest path
           paths.add(shortestPath);
           ArrayList<Cell> full_path = new ArrayList<Cell>();
           Cell parent = shortestPath;
      while(parent!=null)
           {
           full_path.add(parent);
           parent = parent.parent;
           }
     if(!full_path.isEmpty())
           splines.add(spline(full_path));
           }
                }
                        }
public void clear()
        {
           robots.clear();
           blocks.clear();
           lines.clear();
           paths.clear();
           cells.clear();
           splines.clear();
           repaint();
        }
                }   //Fin de la Clase
```

## 4. Conclusiones

Se logró realizar una aplicación con una interfaz gráfica amigable capaz de realizar la creación de escenarios, creación de la estructura de datos con mapas trapezoidales y el grafo de visibilidad que junto con Dijkstra arroja la ruta mínima que deberá seguir el robot.

Se implementó la aplicación que engloba el marco teórico descrito en el desarrollo, con ello se consigue tener los módulos propuestos, principalmente el mapa trapezoidal, para el cual se tiene una aplicación, la cual hace una partición en regiones del escenario, de modo que la unión de ellas es el plano (polígono, espacio, etc.) y la intersección es el conjunto de ejes o semiejes que les sirven de frontera entre las regiones.

Además se obtuvo una interfaz en donde se fue mostrando gráficamente la aplicación de los algoritmos descritos, lo cual también fue un modulo principal a implementar.

Con lo anterior se logró cubrir los objetivos específicos propuestos y se vio alcanzado el objetivo general.

## Referencias


J. C. Latombe, *Robot Motion Planning.* Springer, 1991.

M. Sharir, *Handbook of Discrete and Computational Geometry*, ch. Algorithmic motion planning. Chapman & Hall/CRC, 2004.

T. Lozano-Pérez, “Automatic planning of manipulator transfer movements," *IEEE Transactions in: Systems, Man and Cybernetics*, vol. 11, pp. 681-698, 1981.

T. Lozano-Pérez, “Spatial planning: A conguration space approach," *IEEE Transactions on Computers*, vol. 32, pp. 108-120, 1983.

T. Lozano-Pérez and M. A.Wesley, “An algorithm for planning collisionfree paths among polyhedral obstables," *Communications of the ACM*, vol. 22, pp. 560-570, 1979.

K. Kedem, R. Livne, J. Pach, and M. Sharir, “On the union of jordan regios and collision-free translational motion amidst polygonal obstacles," *Discrete and Computational Geometry*, vol. 1, pp. 59-71, 1986.

K. Kedem and M. Sharir, “An ecient algorithm for planning collisionfree translational motion of a convex polygonal object in 2-dimensional polygonal space," *Discrete and Computational Geometry*, vol. 5, pp. 43-75, 1990.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry Algorithms and Applications*. Springer, 2008.

Universidad de Jaén, http://wwwdi.ujaen.es/asignaturas/gc/

CGAL Computational Geometry Algorithms Library, http://www.cgal.org/.