CAPÍTULO 4

Implementación.

En el Capítulo 1, mencionamos que nuestro algoritmo de optimización de trayectorias sería probado por medio de un simulador en dos dimensiones, con el fin de mostrar su ejecución.

En este capítulo se hablará acerca de este proceso de implementación. Brevemente se describirá el simulador que estamos utilizando; la programación de la Cinemática Inversa cuyo problema se resuelve dentro del mismo algoritmo de Planificación de trayectorias (ACA), siguiendo la propuesta de Ahuactzin y Gupta (1999) y finalmente el desarrollo de un algoritmo que da una solución al Problema del Agente Viajero Multidimensional.

4.1 Desarrollo del simulador.

Para visualizar el resultado de nuestro algoritmo de optimización de trayectorias fue necesario el desarrollo de un simulador.

Este simulador es simple, está proyectado en tres dimensiones aunque la ejecución de sus movimientos se realice únicamente en un plano. Es decir, la rotación de sus articulaciones es siempre sobre el eje z, los obstáculos son colocados asignándoles coordenadas en *x, y*, de igual forma para las marcas que representan los puntos que deseamos alcanzar con el elemento terminal del robot simulador. El valor de z es de cero y permanece constante (Figura 4.1).

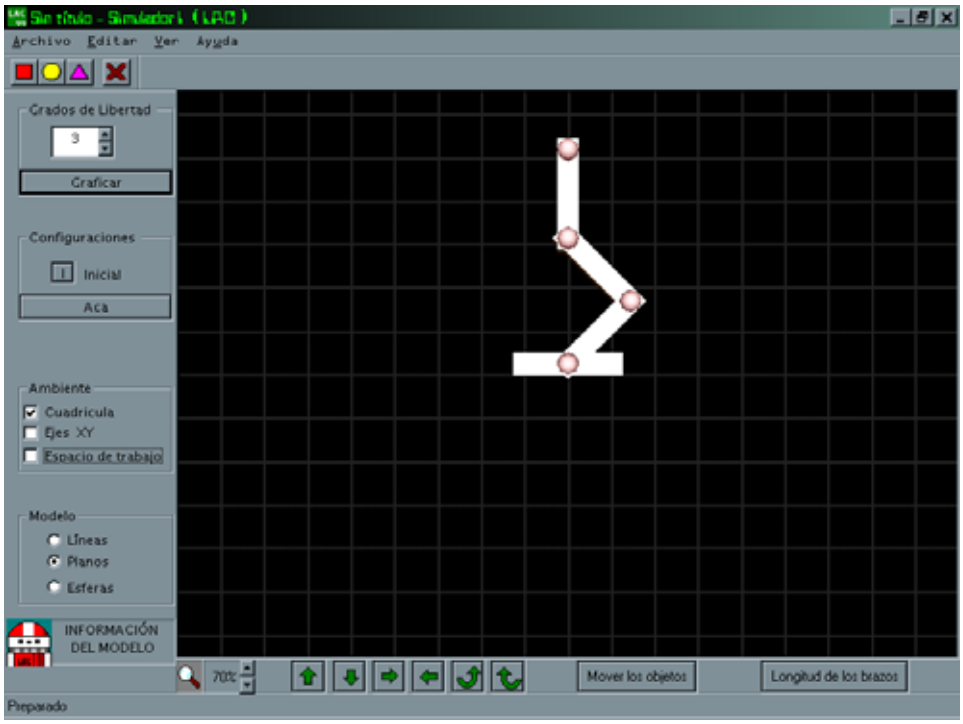

Figura 4.1 El Simulador.

La razón de esta proyección en tres dimensiones tiene que ver más bien con la visualización de los gráficos y además para dejar abierta la posibilidad de desarrollar la ejecución de nuestro algoritmo con rotaciones en los ejes *x, y, z;* con obstáculos y puntos también en este espacio de tres dimensiones.

Para el robot manipulador, el número de grados de libertad es variable. Sin embargo, sí tiene un valor constante como máximo.

El simulador puede construir tres vistas distintas: mediante planos, líneas y esferas. Estas últimas, se utilizan para visualizar el método empleado para la detección de colisiones y que se explicará más adelante.

4.1.1 Objetos en el espacio.

En el simulador pueden agregarse tres tipos de obstáculos que corresponden a las figuras: cubo, cono y esfera.

El sistema utiliza un cuadro de diálogo que permite crear objetos con dimensiones especificadas por el usuario (Figura 4.2). Al colocar un objeto en el espacio, el sistema detecta si este nuevo obstáculo está o no en colisión con el robot (Figura 4.3).

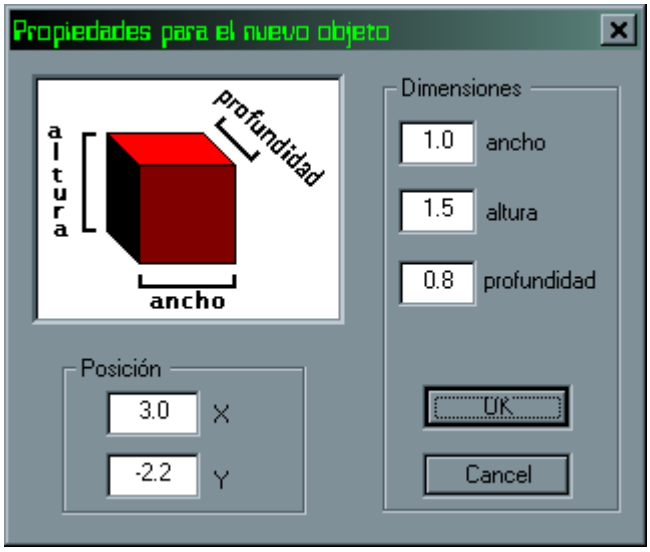

Figura 4.2 Posición y dimensión de los obstáculos.

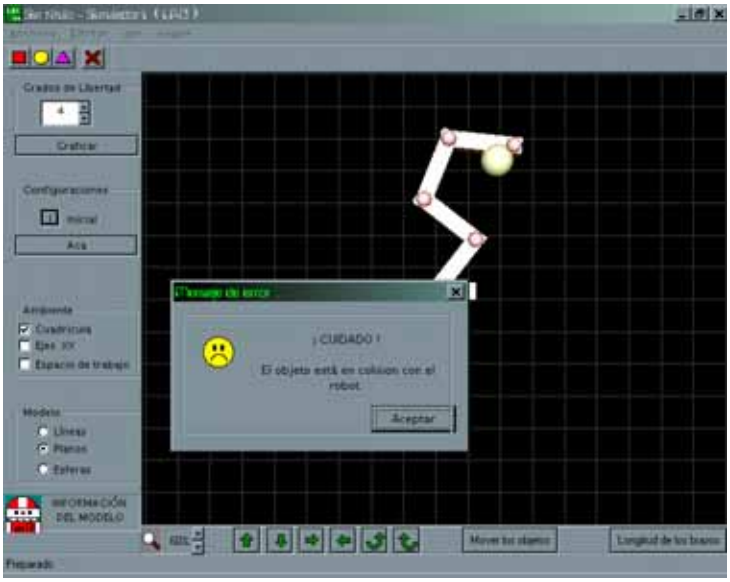

Figura 4.3 Detección de colisiones.

Si deseamos modificar la posición de un objeto especifico ya creado con anterioridad, la interfaz contiene los componentes necesarios que permiten la movilidad de dicho objeto dentro del espacio donde está situado el modelo (Figura 4.4)

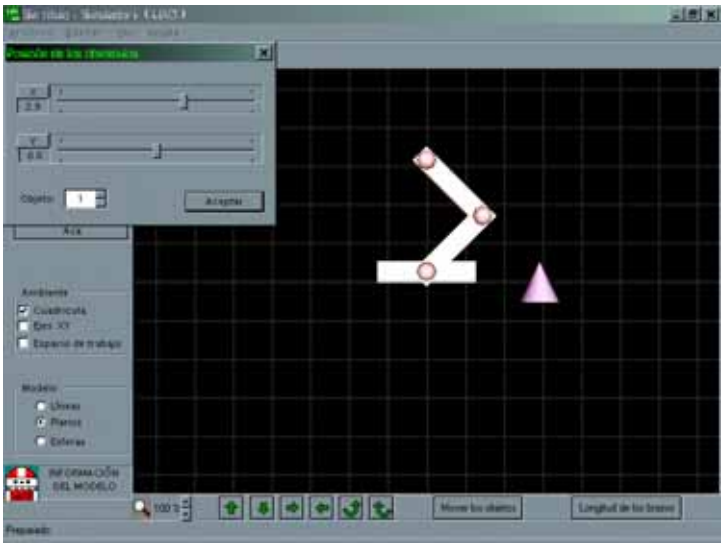

Figura 4.4 Modificación de la posición de los obstáculos.

4.1.2 Desarrollo del sistema.

El desarrollo del sistema es orientado a objetos. Siguiendo las bases de análisis, diseño y modelado de los objetos.

4.1.3 Lenguaje de programación y equipo.

El algoritmo de optimización de trayectorias, el algoritmo del Agente Viajero y el simulador se desarrollaron en el lenguaje de programación Visual C++ 5. La programación de gráficos fue realizada en OpenGL. El equipo empleado fue una PC con Windows 98.

4.2 Implementación del ACA y de la Cinemática Inversa.

Como se mencionó en el Capítulo 2, estamos utilizando la propuesta de Ahuactzin y Gupta (1999), un algoritmo de planificación de trayectorias (ACA) basado en una aproximación global de la Cinemática Inversa que es llamado: la Trayectoria Cinemática.

Recapitulando, este algoritmo consiste en que dada una configuración inicial del robot, el problema radica en encontrar un camino a una configuración accesible que corresponda a la posición y orientación del elemento terminal del manipulador.

Enseguida se explicarán detalles de la implementación de este algoritmo.

4.2.1 Notación.

La figura 4.5 corresponde a la notación utilizada en la formulación de la Trayectoria Cinemática.

d	métrica propuesta para la Trayectoria Cinemática.
$\|\bullet\|$	denota la normal Euclidiana en $\mathbb{R}^n$
dist	denota la distancia entre un punto y un conjunto en $\mathbb{R}^n$
$\hat{q} = (q_1, q_2, \ldots q_n)$	configuración del robot
A	denota el brazo del robot
G	el elemento terminal de A
$\hat{x} = (x, y, z, \alpha, \beta, \gamma)$	configuración del elemento terminal
C_A	espacio de configuración de A
C_G	espacio de configuración de G
$C_{A_{free}}$	subconjunto libre de colisiones de C_A
$C_{A_{free}}(\hat{q})$	denota el componente entero accesible de $C_{A_{free}}$ que contiene el punto $\hat{q}$
L_i	landamark i
L	conjunto de landmarks

Figura 4.5 Notación de la Trayectoria Cinemática
(Ahuactzin y Gupta, 1999)

4.2.2 La métrica.

La métrica *d* es utilizada para medir la distancia entre dos sistemas de coordenadas de referencia, F_a y F_b del robot que se representan por medio de las matrices de transformación ${}^cT_a$ y ${}^cT_a$.

Para un sistema de coordenadas de referencia tenemos tres puntos físicos o vértices dentro del espacio de trabajo del robot. Cada vértice corresponde a cada uno de los vectores unitarios. La distancia *d* se define como la raíz cuadrada de la suma de los cuadrados de las distancias Euclidianas entre los vértices correspondientes a los dos sistemas de coordenadas de referencia:

$$d({}^{c}T_{a}, {}^{c}T_{b}) = \sqrt{d_x^2 + d_y^2 + d_z^2}$$

donde d_x, d_y, d_z son las distancias entre los vértices x, y, z de los vectores unitarios respectivamente.

(Ahuactzin y Gupta, 1999)

4.2.3 Formulación del problema.

El problema de la Cinématica Inversa es planteado como un problema de optimización, donde la función de costo c está definida como:

$$c(\hat{q}, \hat{x}_g) = d(T(f(\hat{q})), T(\hat{x}_g))$$

distancia entre: configuración actual del elemento terminal; configuración deseada del elemento terminal

donde T se refiere a la matriz de transformación homogénea

Podemos plantear el problema de la Cinemática Inversa como sigue:

$$\hat{q}_{\bullet} \in \left\{ q: \min_{\hat{q} \in C_{A\,free}(q_o)} c(\hat{q}, \hat{x}_g) \right\}$$

dada una configuración $\hat{q}_o$
y una configuración deseada para el elemento terminal $\hat{x}_g$
determinar una configuración $\hat{q}_{\bullet}$ accesible para el robot

(Ahuactzin y Gupta, 1999)

4.2.4 Implementación de *SEARCH*.

Como se mencionó, la propuesta de Ahuactzin y Gupta (1999), explota la estructura cinemática serial del manipulador al construir un algoritmo local eficiente que resuelve el problema de optimización planteado.

Formalmente, denotamos un intervalo accesible para el eje i:

$$C^i_{\hat{q}} = \{ (q_1, q_2, \ldots, q_{i-1}, q_i + \theta_i, q_{i+1}, \ldots, q_n) \}$$
$$\theta \in [\Delta_i^{min}, \Delta_i^{max}]$$

donde:

$(q_1, q_2, \ldots, q_n)$ → es una configuración libre del robot

$[\Delta_i^{min}, \Delta_i^{max}]$ → es el intervalo que representa los límites de la articulación i

$C^i_{\hat{q}} \subset C_{A_{free}}$ → su solución está dentro del subconjunto libre de colisiones del espacio de configuraciones del brazo del robot

Vamos a decir que $\hat{x}_g \in C_G$ es una configuración de G. Para un $\hat{x}_g \in C_{A_{free}}$, la función *g* es definida como:

$$g(\hat{q}, i) = \hat{q}^{min}: \min_{\hat{q}' \in C_{\hat{q}}^i} c(\hat{q}', \hat{x}_g)$$

Si tenemos una configuración del robot:

$$\hat{q} = (q_1, q_2, \ldots, q_i, \ldots, q_{n-1}, q_n)$$

entonces: $g(\hat{q}, i) = (q_1, q_2, \ldots, q_i^{min}, \ldots, q_{n-1}, q_n)$

Dada una configuración inicial $\hat{q}_o = (q_1^0, q_2^0, \ldots, q_n^0) \in C_{A_{free}}$ $g(\hat{q}, i)$ se aplica repetidamente de una manera iterativa.

(Ahuactzin y Gupta, 1999)

Algorítmicamente, podemos definir el algoritmo del SEARCH de la siguiente forma:

```
SEARCH ( q̂_o, x̂_g )
begin
q̂ = g^n(q̂_o)
    if ( q̂ == q̂_o  or  f(q̂) == x̂_g )
          return (q̂);
    else
          return (SEARCH(q̂, x̂_g))
end.
```

Empezamos con $\hat{q} = \hat{q}_o$, *SEARCH* aplica repetidamente la función $g^n(\hat{q}_o)$ usando el resultado de la iteración anterior como punto inicial de la nueva iteración.

Se puede apreciar que $\hat{q}_{loc} = SEARCH(\hat{q}_0, \hat{x}_g)$ es un mínimo global (una solución para el problema de la Cinemática Inversa) si se cumple que $c(\hat{q}', \hat{x}_g) = 0$, en caso contrario se habrá encontrado un mínimo local.

Para la implementación de $g(\hat{q}, i)$ se requiere calcular el valor de θ. Si consideramos que P(x, y, z) es un vector que representa los vectores unitarios (i, j, k) para la localización deseada del elemento terminal con respecto al sistema de coordenadas de referencia, y P'(x', y', z') es el vector que representa los vectores unitarios del elemento terminal con respecto al sistema de coordenadas de referencia, podemos calcular θ de la siguiente forma:

$$\theta = \arctan\left[\frac{-x_1y_1' + y_1x_1' - y_2'x_2 + x_2'y_2 - y_3'x_3 + x_3'y_3}{x_1x_1' + y_1y_1' + x_2'x_2 + y_2'y_2 + x_3'x_3 + y_3'y_3}\right]$$

En nuestro proyecto sólo consideramos la posición deseada en x, y del elemento terminal, así tenemos que el cálculo de θ se simplifica, quedando:

$$\theta = \arctan\left[\frac{-x_1y_1' + y_1x_1'}{x_1x_1' + y_1y_1'}\right]$$

(Ahuactzin y Gupta, 1999)

4.2.5 Implementación de *EXPLORE*.

El propósito de *EXPLORE*, es capturar la conectividad de un espacio de configuración accesible para una configuración inicial $\hat{q}_0 \in C_{A_{free}}$ Podemos obtener una representación del espacio mediante la generación de trayectorias libres de colisiones y colocando *landmarks* sobre el espacio accesible $C_{A_{free}}(\hat{q}_0)$, comenzando desde un *landmark* inicial,

donde $L_1 = \hat{q}_0$. La estructura obtenida por *EXPLORE* es esencialmente un árbol, donde el conjunto de nodos representa un conjunto de configuraciones en $C_{A_{free}}(\hat{q}_0)$ y el conjunto de ligas representa las trayectorias libres que conectan estas configuraciones. Esta es la llamada Trayectoria Cinemática (Ahuactzin y Gupta, 1999).

La versión ideal de *EXPLORE* es el *EXPLORE*$_\infty$, el cual requiere de una optimización global, el cual es un problema complejo. La implementación usada es una versión menos ideal y que está basada en un método q*uasirandom* (o *quasi* Monte Carlo) para una optimización global.

En resumen, lo que hace *EXPLORE* es generar trayectorias *random* para cada uno de los *landmarks*. Cada una de estas trayectorias está situada dentro de $C_{A_{free}}(\hat{q}_0)$. Los puntos finales de estas trayectorias *random* son llamados embriones. El embrión más lejano al conjunto de *landmarks* es escogido como nuevo *landmark* y el proceso se repite. El pseudocódigo tiene la siguiente forma:

```
EXPLORE (m)
begin
    ρ_m = 0
    for i = 1 to m
        min_distancia = ∞
        for j = 1 to m
            distancia = ‖ ê_i - L_j ‖
            if ( distancia < min_distancia )
                minima_distancia = distancia
        if ( minima_distancia > ρ_m )
            k = i
            ρ_m = min_distancia

    L_m+1 = ê_k
    δ̂_m+1 = γ̂_k
    γ̂_m+1 = random_path_from( L_m+1 )
    γ̂_k   = random_path_from( L_k )
    ê_m+1 = γ̂_m+1(1)
    ê_k   = γ̂_k(1)

    return(ρ_m)
end
```

El conjunto de *landmarks* es inicializado con $L = \{ L_1 = \hat{q}_0 \}$ *EXPLORE* genera una trayectoria factible random $\hat{\gamma}_1$ y el extremo de esta es inicializado como el primer embrión, $\mathcal{E}_1 = \{\hat{e}_1 = \hat{\gamma}_1(1)\}$. El conjunto de extremo del árbol se inicializa a cero. En el paso *m*, se dice que el embrión $\hat{e}_k \in \mathcal{E}_m$ es el más lejano de L_m. La trayectoria $\hat{\gamma}_k$ correspondiente es agregada al conjunto $\hat{\delta}_{m+1} = \hat{\gamma}_k$

Dos nuevos embriones son generados, uno que reemplaza al embrión que se convirtió en *landmark* y el que está asociado a este nuevo *landmark*.

La función *random_path_from*($\hat{q}$)s quien genera una trayectoria *random* iniciando en la configuración $\hat{q}$.

4.3 Implementación del ACA en nuestro proyecto.

Ahora, vamos a describir la utilización de los algoritmos *SEARCH* y *EXPLORE* en nuestro sistema.

El primer paso es construir un árbol de tamaño *m*. De manera iterativa se hace el llamado al algoritmo de *EXPLORE* mediante un ciclo de uno a *m.* Notemos que *m* corresponde en realidad al número de *landmarks* en el árbol.

Una vez creado el árbol, el segundo paso es la utilización del algoritmo *SEARCH* para resolver el problema de la Cinemática Inversa en cada punto. El pseucódigo se puede ver como:

```
for( i=1 to MAX_LANDMARKS)
        for( j=1 to NO_PUNTOS)
                if ( q = SEARCH(i) )
                  Agregar_solucion( j, q )
```

Lo que hacemos es buscar para cada punto, el conjunto de *landmarks* que alcanzan la configuración encontrada por la Cinemática Inversa resuelta en el algoritmo de *SEARCH*. Esta relación de landmarks y configuración de un punto se almacena en una estructura llamada Camino_Cinemático y puede verse como:

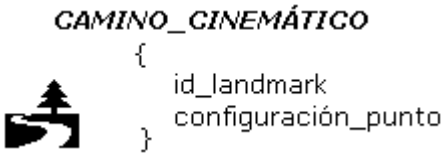

Figura 4.6 Camino Cinemático

Esta estructura es parte de otra llamada Ciudad_Cinemática:

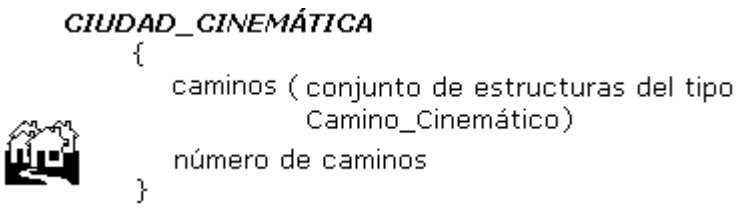

Figura 4.7 Ciudad Cinemática.

Cuando llamamos a Agregar_solución, lo que estamos haciendo es almacenar en un arreglo de tipo Ciudad_Cinemática, un nuevo camino, es decir una nueva configuración para el punto j.

Estas estructuras nos facilitan el manejo de las relaciones que existen entre el conjunto de puntos y las soluciones que hay para cada uno de ellos. Los nombres utilizados están relacionados además con el planteamiento del problema de optimización de este trabajo, el Problema del Agente Viajero.

Como se mencionó en el Capítulo 3, el Problema del Agente Viajero Multidimensional puede verse gráficamente de la siguiente forma (Figura 4.8):

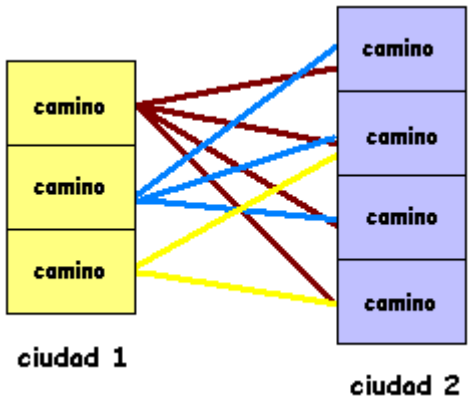

Figura 4.8 Problema del Agente Viajero Multidimensional.

Ahora podemos entender que los caminos son en realidad las relaciones entre un punto y un *landmark*, y que las ciudades representan cada uno de los puntos.

Es importante mencionar que la configuración para alcanzar un punto, que es la solución que nos devuelve la Cinemática Inversa, en muchas ocasiones es casi la misma (varía con valores cercanos al cero). Esto ocurre principalmente en los casos donde el simulador tiene pocos grados de libertad. A pesar de que aparentan ser soluciones semejantes, no es así. Lo importante no está únicamente en los valores de la configuración, sino en esta relación de la que hablamos de *landmark* y configuración del punto.

Un desplazamiento del *landmark i* a una configuración *s,* puede ser menos costosa que del *landmark j* a una configuración *s'* cercana a la configuración *s* (Figura 4.9). Por esta razón, no son eliminadas las configuraciones a pesar de ser parecidas, pues lo que nos interesa es el costo que llevó a colocar al robot en un punto del espacio.

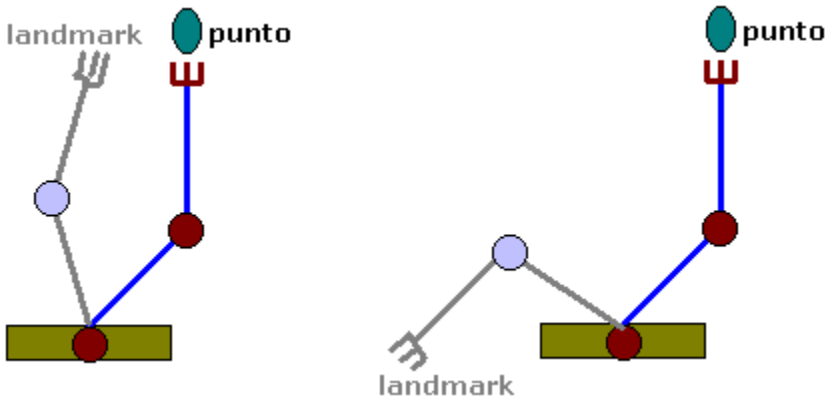

Figura 4.9 Relación de un *landmark* con la configuración de un punto.

4.4 Solución al Agente Viajero Multidimenticional.

Planteadas las relaciones entre configuraciones de puntos y *landmarks*, lo siguiente fue crear una matriz de costos que almacenara la distancia de un punto a otro.

Los valores que contiene se obtienen de la siguiente manera: se calcula la distancia de un punto al *landmark* con el que está relacionado, luego se obtienen la lista de *landmarks* que permiten alcanzar el *landmark* del siguiente punto, esta lista se obtiene recorriendo el árbol del EXPLORE. Finalmente, se obtiene la distancia que hay entre el segundo punto y su *landmark* relacionado. La suma de todas estas distancias es un valor almacenado en la matriz de costos.

La matriz de costos, representa todas las posibilidades que surgen al relacionar cada punto con cada una soluciones posibles del mismo, y almacena el gasto requerido para viajar de una "ciudad" a otra.

El algoritmo de solución es un tipo de algoritmo genético. Tenemos dos tipos de estructuras:

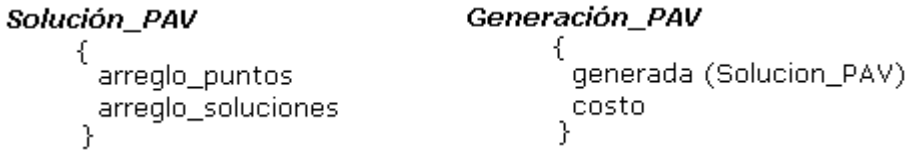

Figura 4.10 Estructuras del PAV.

La estructura Solución_PAV almacena una posible solución de recorrido. Por lo tanto, Generación_PAV representa una solución factible al problema de optimización y su costo asociado.

Nosotros tenemos un arreglo de estructuras del tipo Generación_PAV, es decir, un conjunto de posibles soluciones con sus respectivos costos. Lo que hace el algoritmo es recorrer dicho arreglo un total de *m* veces. Toma su primer elemento y toma uno al azar y los combina, después elige cuál de las tres posibles soluciones tiene el menor costo y lo almacena en un arreglo auxiliar del mismo tamaño. Siempre se está almacenando la mejor solución de todo el arreglo, es decir el elemento de Generación_PAV con el menor costo. En la siguiente iteración, el arreglo auxiliar vacía su contenido en el arreglo inicial. Este procedimiento se repite *m* veces y la solución almacenada es la solución que se presenta en el simulador.