Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Diagramas de Voronoi aplicados a futbol de Robots

por
Emilio Francisco Oca

Director de Tesis
Dr. Juan M. Santos

Tesis para optar al grado de
Licenciado en Ciencias de la Computación

15 de Diciembre de 2005

A mi familia

Gabriela, Mariano, Ana y Sebastían.

...por acompañarme y sostenerme.

Índice general

Capítulo 1

Introducción

1.1. Consideraciones generales

Los diagramas de Voronoi permiten resolver computacionalmente una famila de problemas geométrico-espaciales conocidos como 'de la oficina postal'. El problema consiste en determinar la pertenencia de un punto a distintas regiones en la que se halla particionado el espacio, en nuestra analogía las oficinas postales y su *área de influencia* geográfica.

Como primer aproximación podemos decir que dado un conjunto S de sitios (por ejemplo, puntos) en el plano, su diagrama de Voronoi es la partición de ese plano en regiones, una para cada sitio, tal que la región del sitio p contiene todos los puntos del plano que están más cerca de p que de cualquier otro sitio en S.

El diagrama de Voronoi genera una partición, o tacelación, del espacio que es función de los sitios. Cada segmento, o tacel, del diagrama se ubica entre dos sitios y es la línea en donde la *influencia* de los mismos se iguala. En el punto donde tres o más taceles se tocan es donde la misma cantidad de sitios ejercen una misma influencia.

El espacio al que nos referimos en el párrafo anterior puede ser de cualquier dimensión. Entonces, las áreas de Voronoi serían espacios y los taceles, hiperplanos. En el mismo sentido, la *influencia* puede ser representada por cualquier métrica. Los diagramas de Voronoi se dividen en clases según la función de distancia utilizada.

Es inmediato asociar un diagrama de Voronoi sobre un plano con una función de influencia Euclidiana como una serie de senderos que se ubican entre los obstáculos a evitar, con sumo cuidado como en la Figura 1.1.

Determinar el recorrido a seguir por un robot es una tarea compleja en la que priman la necesidad de llegar de un punto a otro según algún criterio y sin colisiones. Los diagramas de Voronoi, por definición, constituyen una red de senderos a seguir para evitar todos los obstáculos que han sido considerados en el diagrama. Por este motivo son muy utilizados a la hora de asistir en el planeamiento y navegación de robots.

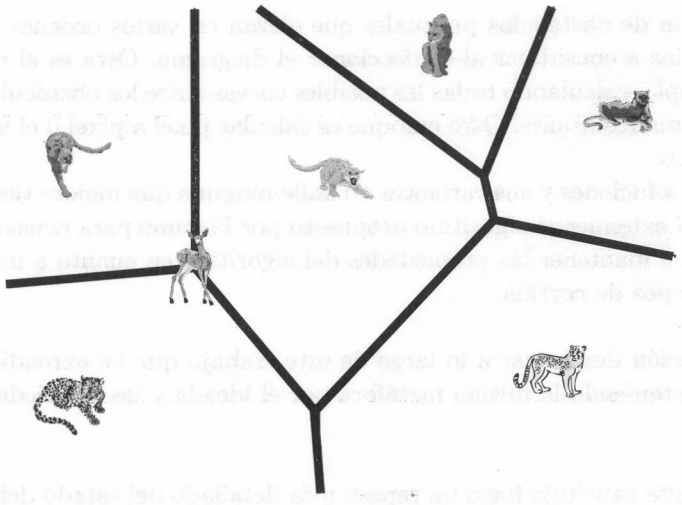


Figura 1.1: Peligros a evitar

¿Qué tan difícil es obtener un diagrama de Voronoi? Existen distintos algoritmos de construcción con complejidades que van desde el $O(n^3)$ al $O(n\ log(n))$ con n la cantidad de sitios. El presente trabajo parte del algoritmo de barrido, o de Fortune [For/87] que presenta un orden de $O(n\ log(n))$.

Es frecuente que un robot tenga que moverse en un espacio poblado por objetos no puntuales como podrían ser una mesa, una pared, otro robot, etc. A los fines prácticos estos obstáculos pueden modelarse como polígonos, pero introducir estos polígonos en un diagrama de Voronoi no siempre es trivial. Intuitivamente es fácil extender la idea y lograr un diagrama que refleje las *áreas de influencia* de cada uno de estos nuevos obstáculos. Lo que no resulta tan fácil o intuitivo es cómo extender los algoritmos existentes conservando sus propiedades.

La motivación particular de este trabajo es obtener en tiempos razonables un camino posible para que un robot se mueva en medio de un entorno dinámico con una cantidad apreciable de obstáculos moviéndose (entre 6 y 22 objetos).

Por ejemplo, consideremos un robot moviéndose en el entorno de un partido de fútbol de robots y el espacio especial que conforma el campo de juego. Los robots amigos, y de los otros, serán obstáculos a esquivar, pero también lo serán las paredes. Algo importante de notar es que un camino posible será no sólo el moverse entre los robots, sino también noverse entre los robots y las paredes. Las paredes se transforman, entonces, en obstáculos no puntuales.

Al hacer un relevamiento sobre el estado del arte de las técnicas de confección de diagramas de Voronoi que contemplen la inclusión de obstáculos no puntuales he encontrado soluciones *no razonables* en cuanto a los objetivos del presente trabajo.

Una de las soluciones frecuentes es el reemplazo de los osbtáculos no puntuales

por una sucesión de obstáculos puntuales que elevan en varios ordenes de magnitud la cantidad de sitios a considerar al confeccionar el diagrama. Otra es el uso de la fuerza bruta, por ejemplo, calculando todas las posibles curvas entre los obstáculos para después podar el diagrama resultante. Otro enfoque es calcular pixel a pixel o el uso de hardware gráfico específico.

Entre estas soluciones y sus variantes no hallé ninguna que mejore tiempos de $O(n^2)$. Entonces decidí extender el algoritmo propuesto por Fortune para considerar segmentos de manera tal de mantener las propiedades del algoritmo en cuanto a uso de espacio en memoria y tiempos de corrida.

Es mi intención demostrar a lo largo de este trabajo que he extendido el algoritmo de Fortune manteniendo la misma metáfora por él ideada y las propiedades antes mencionadas.

En el siguiente captítulo hago un repaso más detallado del estado del arte, doy algunas definiciones y comento las principales formas de construir un diagrama de Voronoi. En el Capítulo 3 incorporo un análisis de las complicaciones que aportan los obstáculos no puntuales y los distintos enfoques que se encuentran al intentar construir diagramas de Voronoi que los incluyan. En el Capítulo 4 detallo cómo funciona el algoritmo de Fortune original y cuales son las extensiones del algorítmo que propongo. En el Capítulo 5 exploro una prueba de concepto guiando un robot en medio de un tablero dispuesto como un partido de fútbol de robots. Finalmente en el Capítulo 6 resumo las conclusiones del presente trabajo y comento distintas líneas de trabajo futuras.

Capítulo 2

Estado del arte y construcción de diagramas de Voronoi

Mucho del material incluido en este capítulo es una recopilación y resumen del material encontrado principalmente en [AK/96] y [dBvKOS/94]. Se incluye para conveniencia del lector.

2.1. Algunas definiciones

A lo largo de este trabajo denotaré como S al conjunto no vacío de puntos o sitios p, q, r, ... que generan un diagrama de Voronoi. A n como el cardinal de S. La 'influencia' que los sitios ejercen es especificado mediante una métrica que aquí siempre será la distancia Euclidiana o norma en L_2, que denotaré como $d(p,r)$ con p y r puntos en el plano. El segmento entre los puntos p a q será $\overline{pq}$. Y $\overline{A}$ la clausura del conjunto convexo A de puntos en el plano, es decir A y su perímetro.

Sea $B(p,q)$ el *bisector* de p y q, con p, $q \in S$, la línea perpendicular al segmento $\overline{pq}$, que lo cruza por su centro, definido por la siguiente expresión

$$B(p,q) = \{\ x \mid d(p,x) = d(q,x)\ \} \tag{2.1}$$

que a su vez genera los semiplanos abiertos

$$D(p,q) = \{\ x \mid d(p,x) < d(q,x)\ \} \tag{2.2}$$

que contiene a p y $D(q,p)$ que contiene a q. Llamaremos

$$RV(p,S) = \bigcap_{q \in S, q \neq p} D(p,q) \tag{2.3}$$

la región abierta[1] de Voronoi de p con respecto de S. Finalmente el diagrama de Voronoi de S queda definido como

$$V(S) = \bigcup_{p,q \in S, q \neq p} \overline{RV(p,S)} \cap \overline{RV(q,S)}. \tag{2.4}$$

[1] pues no incluye su frontera

La frontera común entre la clausura de dos regiones pertenece a $V(S)$ y se la denomina eje de Voronoi o tacel. Si el tacel e delimita las regiones de los sitios p y q se cumple que $e \subset B(p,q)$. Los extremos de los taceles se llaman vértices y pertenecen a las fronteras de tres o mas regiones de Voronoi.

Lema 2.1.1. *Un diagrama de Voronoi con tres o mas sitios no colineales $V(S)$ tiene $O(n)$ vértices y ejes, y el promedio de taceles en la frontera de una región de Voronoi es menor a 3.*

Demostración. Por el teorema de Euler [Gib/85] para grafos planares es cierta la siguiente relación para v, t, n y c, vértices, taceles, sitios y componentes conexos respectivamente

$$v - t + n = 1 + c. \tag{2.5}$$

Un diagrama de Voronoi no es un grafo propio, pues tiene arcos que no son segmentos, sino semirectas. Para salvar esta *dificultad* podemos agregar un nuevo vértice v_∞ *en el infinito* al que todas las semirectas confluyen.

Sabemos que por cada vértice hay al menos 3 taceles que llegan a él pero a su vez cada tacel llega a dos vértices, entonces vale la desigualdad

$$t \geqslant 3v/2 \tag{2.6}$$

siendo que $c = 1$, si reemplazamos en eq. 2.5 obtenemos

$$v - t + n = 2 \models v = 2 - n + t \tag{2.7}$$

Si reemplazamos usando eq. 2.6 queda

$$\begin{aligned} v \geqslant 2 - n + 3v/2 &\models v - 3v/2 \geqslant 2 - n \\ &\models v(-1/2) \geqslant 2 - n \\ &\models v \leqslant (2-n)/(-1/2) \\ &\models v \leqslant 2n - 4 \end{aligned} \tag{2.8}$$

reemplazando ahora la eq. 2.8 en la eq. 2.7

$$2 - n + t \leqslant 2n - 4 \models t \leqslant 3n - 6$$

Ahora bien, si sumáramos los taceles contenidos en las n regiones de Voronoi obtendríamos 2t ya que cada tacel se cuenta dos veces. Entonces

$$2t \leqslant 6n - 12$$

Si tomáramos el promedio de taceles por región obtendríamos

$$2t/n \leqslant (6n - 12)/n \tag{2.9}$$

donde

$$\begin{aligned} (6n-12)/n &\models 6(n-2)/n \\ 6(n-2)/n &< 6 \end{aligned} \tag{2.10}$$

Finalmente, combinando eq. 2.9 y eq. 2.10 obtenemos que el promedio de taceles por región queda acotado de la siguiente manera

$$2t/n < 6$$

La misma cota aplica para $V(S)$. □

Corolario 2.1.1. *En el transcurso de la demostración anterior hemos despejado en las eq. 2.8 y 2.8 dos valores que también son de importancia. Que el número de taceles en un diagrama de Voronoi es a lo sumo* $3n - 6$ *y que el número de vértices es a lo sumo* $2n - 4$.

2.2. Construcción incremental

Green y Sibson[GS/78] estudiaron la idea de construir diagrama de Voronoi incrementalmente, o sea, obtener $V(S)$ a partir de $V(S\backslash\{p\})$ insertando p. La región de p puede tener hasta $n - 1$ taceles, con $n = |S|$, lo cual conduce a una complejidad de $O(n^2)$, pues por cada una de las n inserciónes de un p hay que revisar su relación con sus probables $n - 1$ vecinos (uno por cada tacel). Varios autores realizaron mejoras a este algoritmo y lograron tiempos esperados de corrida de $O(n\ log(n))$, pero que en la práctica son de $O(n)$ cuando los sitios están distribuidos especialmente, ver Ohya et al. [OIM/84] y Sugihara y Iri [SI/92].

2.3. Divide & conquer

Shanos y Hoey [SH/75] presentaron por primera vez un algoritmo que determinísticamente corría en tiempo $O(n\ log(n))$. El algoritmo consiste en dividir el conjunto de sitios S a través de una línea obteniendo los subconjuntos L y R de aproximadamente la misma cantidad de elementos. Luego se construyen los diagramas de Voronoi $V(L)$ y $V(R)$ recursivamente. La partes esenciales del algoritmo son encontrar la línea divisoria y el trabajo de unir $V(L)$ y $V(R)$ para obtener $V(S)$. Shanos y Hoey muestran que estas tareas pueden llevarse a cabo en tiempo $O(n)$, obteniendo un tiempo total de $O(n\ log(n))$.

2.4. Sweepline

La estrategia del algoritmo de Fortune es barrer con una línea horizontal desde un lado del plano hacia el otro. Mientras el barrido es realizado se va obteniendo y registrando información sobre las intersecciones de la estructura del diagrama con la línea de barrido. Mientras la línea de barrido se mueve la información obtenida no cambia, excepto por ciertos casos, que es cuando suceden eventos.

El algoritmo requiere mantener la intersección del diagrama de Voronoi con la línea de barrido, y esto no es inmediato, pues parte del diagrama recorrido depende de aquel por recorrer. Por ejemplo cuando la línea de barrido alcanza al primer vértice de una

región Voronoi en su recorrido, todavía no ha alcanzado al sitio que genera esa región. Claramente no poseemos la información necesaria para calcular ese vértice.

Entonces, en lugar de intentar mantener la información sobre la intersección de la línea con el diagrama, mantendremos la información de la parte del diagrama que ya no es afectada, y por lo tanto no será modificada, por ningún sitio que pueda aparecer más allá de la línea de barrido.

¿Pero qué parte del diagrama ya conformado no cambiará? Es lo mismo que preguntar: ¿Para qué parte de los puntos del semiplano ya revisado sabemos cual es su sitio más cercano, y en consecuencia, la región a la que pertenece?

Sea l^+ el semiplano ya recorrido por la línea de barrido l. Para cualquier punto $p \in l^+$ la distancia de ese punto hacia cualquier sitio a descubrir por la línea l es mayor que la distancia a la línea en sí. Se deduce entonces que el sitio más cercano a p no puede estar más allá de l si p ya está al menos tan cerca de otro sitio $s \in l^+$ como lo está de l.

Los puntos que están tan cerca o más de s que lo que están de l se hallan contenidos por una parábola. Se genera entonces, una sucseción de parábolas detrás de l llamadas waveFront o línea de playa.

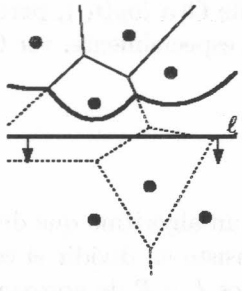


Figura 2.1: Diagrama de Voronoi a medio barrer. Tomado de [dBvKOS/94].

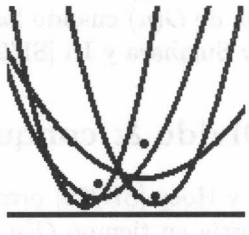


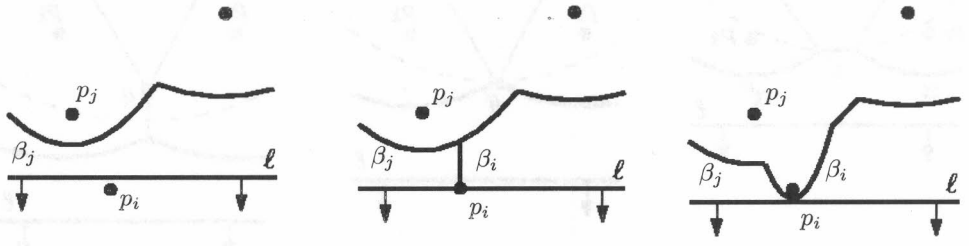
Figura 2.2: La línea de playa y las parábolas que la conforman.

Es fácil notar que los cruces entre parábolas consecutivas, o quiebres de la línea de playa, se dan sobre los taceles que están conformándose. Esto no es casualidad, pues esos cruces trazan los taceles mientras la línea de barrido se mueve por el plano.

Entonces será la línea de playa el límite entre el diagrama inamovible y del que está en exploración. La línea de playa varía continuamente mientras se realiza el barrido. Pero, no es necesario que el barrido sea continuo, pues la estructura de la línea de playa (o sea las parábolas que las constituyen como arcos o componentes de la onda) irá cambiando en momentos precisos. Cada arco aparece y desaparece en eventos predecibles, salvo el primero que simplemente aparece y el o los últimos que prevalecen.

El evento en el cual un nuevo arco aparece (secuencia en la Figura 2.3) es cuando un sitio es alcanzado por la línea de barrido. La parábola definida por este nuevo sitio en l^+ nace como una parábola degenerada con cero amplitud conectando la línea de barrido con la línea de playa. Mientras la línea de barrido avanza la nueva parábola se hace cada

vez más ancha. Llamaremos a este evento un evento de sitio.



(a) (b) (c)

Figura 2.3: Secuencia de avance y evolución de la línea de playa. a) Un nuevo sitio está por ser alcanzado por la línea de barrido. b) El sitio está sobre la línea de barrido y se genera una parábola degenerada de ancho cero. c) La línea de barrido sobrepasó al sitio y recupera un aspecto normal. Tomado de [dBvKOS/94].

¿Qué implica la ocurrencia de un evento de sitio? Sea β_j la parábola generada entre l y el sitio p_j, y p_i el sitio que genera el evento de sitio sobre la parábola β_j (Figura 2.3(b)). Por definicion de las parábolas, la distancia entre l y los sitios p_i y p_j es la misma, y al hallarse por sobre la línea de playa ya no puede aparecer otro sitio con una distancia menor. Por lo tanto, el punto en donde la vieja línea de playa se cruza con la nueva parábola (degenerada en un principio) será parte de un tacel. Lo mismo aplica a cada quiebre en la línea de playa y explica por que estos quiebres delinean los taceles.

Lema 2.4.1. *Sólo mediante un evento de sitio aparece un nuevo arco en la línea de playa.*

Demostración. Si el nuevo arco no se genera como lo hemos descripto mediante una parábola extremadamente angosta, sólo queda la posibilidad de que emerja por detrás de la línea de playa. Pero esto es opuesto al comportamiento de una parábola que se extiende y aplana en la medida que su foco se aleja de la linea sobre la cual se genera. □

Como consecuencia del lema 2.4.1 podemos afirmar que $2n - 1$ es una cota a la cantidad de componentes que puede tener la línea de playa. Se obtiene rápidamente al considerar que cada sitio que se encuentra mediante el barrido da lugar a un nuevo arco que a su vez parte en dos aquel arco sobre el que se genera en la línea de playa.

El segundo tipo de evento en el algorítmo de línea de barrido es el evento en el cual desaparece un arco de la línea de playa como se ve en la secuencia de la Figura 2.4. Llamaremos a este evento un evento de círculo.

Los arcos vecinos al que se extingue no pueden pertenecer a la misma parábola. Usando el mismo argumento del lema anterior, si fueran de la misma parábola significaría que el arco que desaparece se generó sobre esa parábola dando lugar a ese par de vecinos, y en ese caso debiera seguir extendiéndose y no estar desapareciendo.

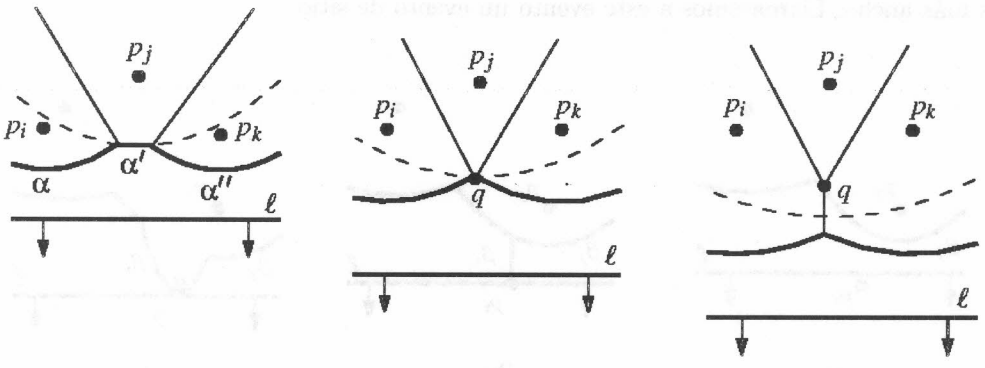


Figura 2.4: Secuencia de desaparición de una parábola en un cruce de taceles. Tomado de [dBvKOS/94].

En consecuencia los tres arcos pertenecen a parábolas generadas por sus distintos respectivos sitios. El arco que se encogía fue trazando dos taceles, uno en cada cruce con sus vecinos.

Finalmente esos taceles se unirán generando un vértice y a partir de él, un nuevo tacel que será trazado por la línea de playa en la medida que siga su avance.

Geométricamente significa que la línea de barrido alcanzó el punto más bajo del círculo cuyo perímetro pasa por los puntos involucrados y cuyo centro es el vértice generado.

Recordemos que si la línea de barrido alcanza el extremo de este círculo, significa que todos los puntos involucrados se hallan a la misma distancia del nuevo vértice y que, dada la posición de la línea de barrido, nada que se halle más allá podrá alterar esta situación.

En otras palabras, en el punto en el que un arco se hace de longitud cero y otros dos confluyen sobre él, siendo que se trata de parábolas con foco en cada sitio, podemos afirmar que las distancias entre ese punto, los sitios y la línea de barrido es la misma.

Lema 2.4.2. *Los arcos sólo desaparecen mediante un evento de círculo.*

Demostración. Es claro que un arco se expande o se reduce. El que se reduce lo hace entre dos arcos originados en distintas parábolas. Siendo distintas parábolas los sitios involucrados son al menos tres (pueden ser más si son más de uno los arcos que desaparecen, incluso simultaneamente). Éstos serán cocirculares; es decir, pertenecen a la misma circunferencia; pues, sus respectivas parábolas se tocan en el centro del círculo que también es tangente a la línea de barrido. □

Corolario 2.4.1. *La estructura de la línea de playa sólo se ve modificada en alguno de los dos eventos mencionados*[2].

[2]Los eventos posibles en un diagrama de Voronoi pueden ser hasta cuatro si usamos métricas más

Es directo, ya que los eventos de sitio dan lugar a nuevos arcos y los eventos de círculo los consumen.

Entonces queda definido el ciclo de vida de la línea de playa a través de la sucesión de eventos de sitio y eventos de círculo. El producto emergente de la evolución de la línea de playa es el diagrama de Voronoi.

En [DK/94] los autores dan una generalización de este algoritmo y demuestran que, con una adecuada elección de las estructuras de datos a utilizar, su complejidad temporal y espacial es $O(n\ log(n))$ y $O(n)$ para el peor caso, respectivamente.

Otra interpretación

Es de utilidad a la hora de imaginarse o visualizar por qué funciona el algorítmo darle otra interpretación geométrica. Considerémos que cada sitio proyecta en el espacio sobre sí un cono que se abre a 45°. El diagrama de Voronoi coincide con la proyección de las intersecciones de los conos vecinos sobre el plano en el que están ubicados los sitios.

En la Figura 2.5 se observan tres sitios y sus conos cortados a cierta altura vistos desde arriba. Si consideramos las ecuaciones de los conos y calculamos sus intersecciones, obtenemos una parábolala en cada caso. Si a cada parábola la proyectamos sobre el plano, es decir eliminamos su coordenada Z, obtenemos rectas como las que se observan en la figura.

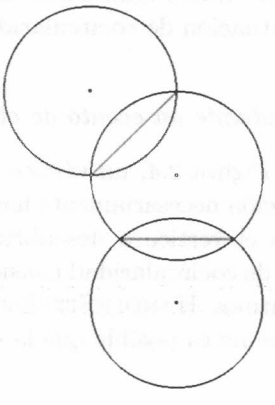


Figura 2.5: Proyección de tres conos sobre el plano y la sombra de sus intersecciones

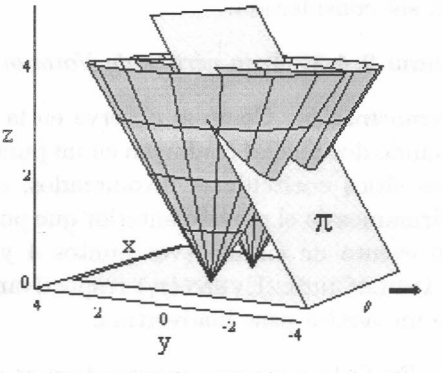


Figura 2.6: Representación espacial de la línea de barrido sobre dos sitios

Teniendo este modelo en mente, la línea de barrido puede verse como la intersección de un plano que, también a 45° en el espacio, pasa por sobre los sitios ya explorados y por debajo de aquellos aún sin explorar. En la Figura 2.6 se ve la parte superior del plano π cruzando por sobre dos sitios. La intersección de π con los conos sobre los sitios son parábolas, que proyectadas sobre el plano que alberga a los sitios generan nuevamente

generales que la Euclideana [DK/94]

parábolas. En la Figura 2.2 se pueden ver las proyecciones de cada sitio y que parte de estas proyecciones conforman el línea de playa.

En la Figura 2.3 se observa como la línea de barrido alcanza un sitio (se produce un evento de sitio). En el momento que la línea de barrido se ubica sobre el sitio (Figura 2.3(b)) se puede ver como la intersección entre el plano de la línea de barrido y el cono del sitio conforman la una parábola *degenerada*, una recta ya que en ese punto, el plano y el cono son tangentes por llevar la misma inclinación.

También resulta fácil *ver* el Lema 2.4.1. Una parábola nunca atravesará por detrás a otra con la que no tiene contacto. Las parábolas generadas por π y los respectivos conos crecerán al mismo paso y generarán parábolas que también crecerán al mismo paso.

Cada evento de círculo puede ser previsto ni bien tenemos dos taceles en formación que tiendan a cruzarse antes del infinito. La previsión no es infalible, pues, al no saber que vendrá desde la parte inexplorada, no sabemos si el evento llegará a suceder. Puede ocurrir que la aparición de un evento de sitio agregue un nuevo tacel y que las posibles intersecciones de este con los existentes alteren las previsiones ya realizadas como muestran las Figuras 2.7 y 2.8.

Preveeremos un evento de círculo cuando la línea de barrido alcance el extremo del círculo (sean tangentes) definido por cada terna de arcos que proyecten la intersección de sus respectivos taceles por delante de la línea de playa. Como tendremos uno de estos eventos para cada terna válida no es posible que una situación de cocircularidad suceda sin ser considerada.

Lema 2.4.3. *Todo vértice de Voronoi es detectado mediante un evento de círculo.*

Demostración. Como se observa en la secuencia de la Figura 2.4, un vértice se obtiene cuando dos taceles confluyen en un punto. En esta situación necesariamente hay al menos tres sitios cocirculares involucrados, cuyo centro será el vertice a descubrir. Pero ya afirmamos en el párrafo anterior que por cada situación de cocircularidad consideraremos un evento de círculo (ver puntos 4 y 5 de los algoritmos HANDLESITEEVENT(p_i) y HANDLECIRCLEEVENT(p_l) respectivamente) por lo que no es posible que la ocurrencia de un vértice pase inadvertida. □

En todo momento mantendremos una cola de prioridad de eventos ordenada según la ocurrencia próxima de cada evento. Los eventos de sitio se saben de antemano y están dados por sus propias coordenadas. Los eventos de círculo serán agregados en la medida que se vayan descubriendo y eliminados en la medida que se vayan efectuando o descartando.

Como primer aproximación, el detalle del algoritmo sería el siguiente:

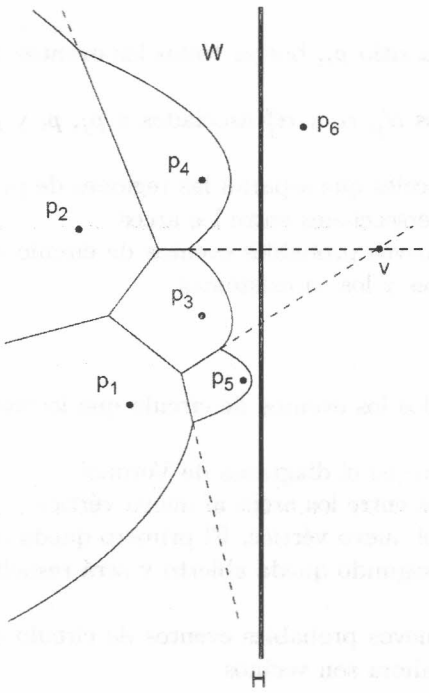


Figura 2.7: En el punto v el arco correspondiente a p_3 desaparecerá. p_4, p_3 y p_5 son cocirculares con centro en v. Entonces esperamos un evento de círculo sobre v. Tomada de [AK/96].

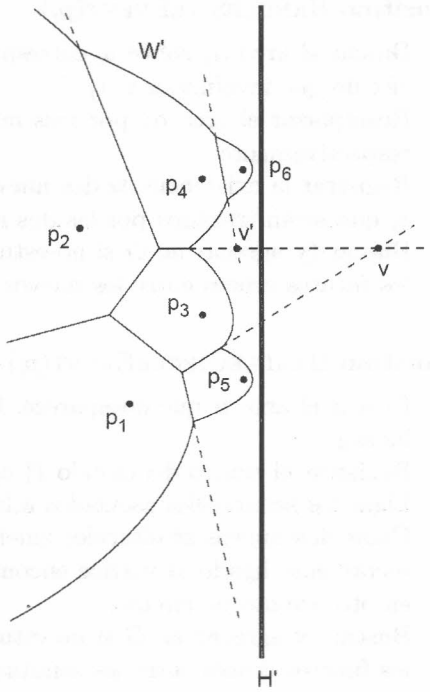


Figura 2.8: En una variante de la Figura 2.7, cuando la línea de barrido sobrepasa a p_6 surge otro tacel. El arco correspondiente a p_4 es el que desaparecerá en v'. Descartamos el evento sobre v y lo esperamos sobre v'. Tomada de [AK/96].

Algoritmo DIAGRAMADEVORONOI(P)
Input: Conjunto $P = \{p_i\}$
Output: El diagrama $Vor(P)$

1. Inicializar la lista ordenada de eventos Q con todos los eventos de sitio.
2. **mientras** Q es no vacía
3. **hacer** considerar el primer evento e de Q
4. **si** e es evento de sitio
5. **entonces** HANDLESITEEVENT(p_i).
6. **si no** HANDLECIRCLEEVENT(p_l) con p_l el punto en la intersección de la línea de barrido y el círculo asociado al evento e.
7. Remover e de Q

Los procedimientos para administrar los eventos se definen así,

Algoritmo HANDLESITEEVENT(p_i)

1. Buscar el arco α_j sobre p_i correspondiente a sitio p_j, borrar todos los eventos de círculo que involucran a α_j
2. Reemplazar el arco α_j por tres nuevos arcos α'_j, α_i y α''_j asociados a p_j, p_i y p_j respectivamente.
3. Registrar la existencia de dos nuevos semitaceles que separan las regiones de p_j y p_i que serán trazados por las dos nuevas intersecciones entre los arcos.
4. Buscar (y agregar en Q si no estuvieren) nuevos probables eventos de círculo en los futuros cruces entre los nuevos semitaceles y los ya existentes.

Algoritmo HANDLECIRCLEEVENT(p_l)

1. Buscar el arco α que desaparece, borrar todos los eventos de círculo que lo involucran.
2. Registrar el centro del círculo P_l como vértice en el diagrama de Voronoi.
3. Ligar los semitaceles asociados a los quiebres entre los arcos al nuevo vértice.
4. Crear dos nuevos semitaceles emergiendo del nuevo vértice. El primero queda directamente ligado al vértice encontrado, el segundo queda abierto y será resuelto en otro evento de círculo.
5. Buscar (y agregar en Q si no estuvieren) nuevos probables eventos de círculo en los futuros cruces entre los semitaceles que ahora son vecinos.

Al presentar estos algorítmos hago introducción de un término nuevo, semitacel. Por razones prácticas, durante la construcción del algoritmo definiremos un par de semitaceles por cada tacel.

Veamos por qué. Siempre que atendemos un evento de sitio descubrimos un nuevo tacel que separará al nuevo sitio del sitio en cuyo arco de línea de playa ocurrió el evento. En ese momento el nuevo tacel tiene dimensión cero, pero inmediatemente, junto con el avance de la línea de barrido, el tacel comenzará a crecer hacia ambos lados siguiendo los quiebres en la línea de playa. Un semitacel representa la mitad del tacel que crece desde el punto en que fue descubierto y se proyecta posiblemente hacia el infinito.

Por cada tacel tendremos un semitacel en cada lado libre del tacel. Aclaro esto atento al siguiente escenario: cuando atendemos un evento de círculo dos taceles se encuentran y surge uno nuevo que sólo crece en uno de sus extremos, pues el punto donde se lo descubre es uno de los extremos del futuro tacel y ya no se moverá.

A los fines prácticos del algoritmo, contar con una entidad específica que denote cada extremo libre de un tacel es de suma utilidad. Un tacel aporta cierta información relacionada a los sitios que lo generaron y la curva que lo representa. Mientras que es el rol de semitacel aportar la información sobre el sentido de avance de un extremo. En conjunto permiten identificar un extremo de un tacel y, por ejemplo, asociarlo a un quiebre de la línea de playa o resolver un cruce entre taceles adecuadamente.

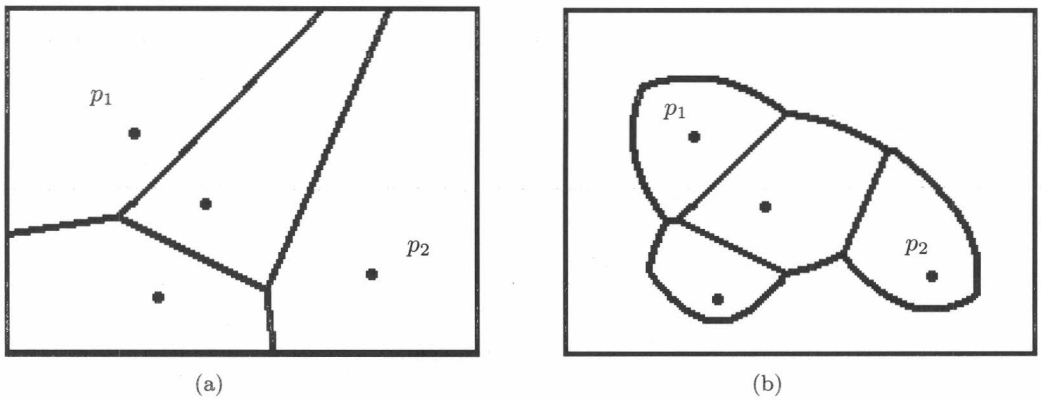
Tener semitaceles es una desición de diseño. La información que aporta un tacel es necesaria. También lo es pensar en sus extremos libres. La decisíon consiste en separar

estos conceptos y darles entidad propia, y la libertad de manejarlos por separado.

Capítulo 3

Complicaciones de los obstáculos no puntuales

3.1. La pared, un jugador más



(a) (b)

Figura 3.1: Diagrama de Voronoi adaptado para considerar los límites como obstáculos

En la Figura 3.1 se puede observar un mismo conjunto S de sitios para los que se ha obtenido el diagrama de Voronoi asociado. A la izquierda está representado el diagrama tradicional que luego ha sido enmarcado. A la derecha se ve el mismo diagrama pero al obtenerlo se ha tenido en cuenta el marco como un único obstáculo más.

En 3.1(a) existe una única manera de cruzar a través de los taceles desde la región de p_1 a p_2. También existen muchos extremos de taceles que no llevan a ningún lado. En contraparte, en 3.1(b) hay varios posibles caminos para llegar de la región de p_1 a p_2 al ser considerada también la posibilidad de rodear por fuera el conjunto S pasando entre los sitios y el marco. Claramente 3.1(b) es una mejor opción a la hora de armar un camino mediante taceles de Voronoi.

3.1(b) es un diagrama de Voronoi válido, en particular es un diagrama general de

Voronoi[1] por que posee generadores de Voronoi, también llamados atractores, no puntuales, en este caso, el marco como un todo, aunque también podrían ser cada lado por separado.

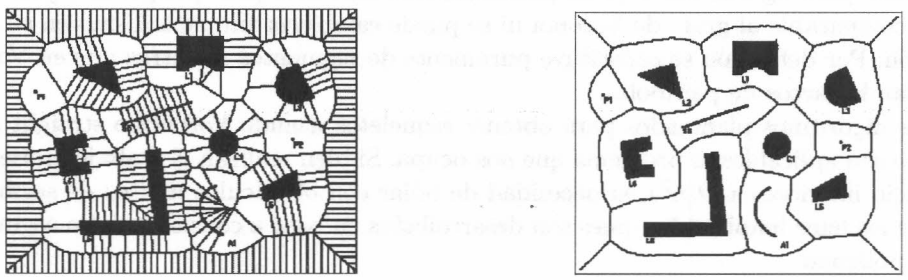
3.2. Enfoques

Como ya mencionamos, es difcil encontrar bibliografa que se ocupe del tema de diagramas generales de Voronoi que incluyan polígonos, o incluso sitios con radio superior a cero que a además intenten resolverlos en $O(n\ log(n))$.

Pendragon and While [PW/99] presentan un algoritmo intuitivo y luego lo extienden para polígonos. Definen el tipo de relación entre distintos obstáculos y las curvas que sus fronteras generan. Calculan cada curva y luego descartan las que no corresponden. El algoritmo presenta un orden promedio de $O(n^2)$y un peor caso de $O(n^3)$.

Papadopoulou [PL/99] genera diagramas sin parábolas en tiempo $O(n^2)$ utlizando métricas L_1 o L_∞. Pero estos diagramas no son los más adecuados para la navegación de robots; sí lo son, en cambio para distribución de componentes en obleas VLSI(very large system integration), para el cálculo de la disipación de calor en VLSI o demás distribuciones donde predominen las formas paralelas.

Vleugels y Overmars[VO/95] calculan una estructura que aproxima un diagrama de Voronoi utilizando grillas de tamaño y precisión variables para ir descubriendo la ubicación de los segmentos que componen el diagrama. El resultado es una colección de áreas que incluyen los taceles del diagrama de Voronoi con presición tan alta como se desee. Los autores ofrecen resultados prácticos razonables, incluso en 3 dimensiones, pero el algoritmo sigue teniendo un orden de $O(n^3)$.



(a) (b)

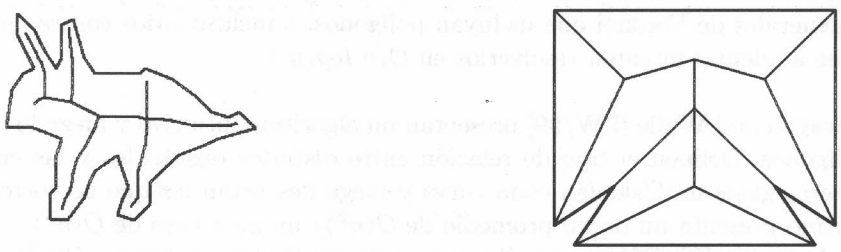
Figura 3.2: a) Diagrama general de Voronoi a partir de la partición de segmentos en una serie de puntos. b) Diagrama obtenido luego de podar los taceles artificiales. Tomadas de [RD/03]

Roque y Doering[RD/03] definen polígonos a partir de los perfiles obtenidos de los obstáculos en una imagen digitalizada. Cada segmento de estos polígonos se particiona en varios generadores de Voronoi dependiendo de la precisión buscada en el diagrama de Voronoi a obtener. Como se ve en la Figura 3.2 la cantidad de generadores de Voronoi

[1]del inglés, Generalized Voronoi diagram.

pasa a ser proporcional a la precisión buscada y no de la cantidad original de obstáculos.

Otro uso de los diagramas de Voronoi es para obtener alguna estructura interna o esqueleto a partir de un polígono o figura en el plano. Varios autores estudian el tema con distintos enfoques. Mayya y Rajan[MR/94] toman los segmentos como obstáculos para generar la estructura, mientras que Klein y Lingas [KL/93] toman los vértices.



(a) (b)

Figura 3.3: a) una variante de esqueleto tomada de [MR/94] construido usando un diagrama de Voronoi que luego fue podado. b) otra variante de esqueleto tomada de [EE/99] construida mediante el algoritmo Stratight Skeletons

Para la finalidad particular de construir esqueletos, autores en temas relacionados con los diagramas de Voronoi como Aichholzer y Aurenhammer[AA/96], han desarrollado otra clase de diagramas llamados Straight Skeletons que son una alternativa a los diagramas de Voronoi y pueden resolverse en tiempo esperado de $O(n\ log(n))$. No es apropiado para figuras con objetos puntuales sino más bien para polígonos o polilíneas. No es comparable al grafo de Voronoi ni se puede establecer un paralelismo en su construcción. Por definición se constituye puramente de segmentos mientras que en Voronoi abundan los arcos de parábolas.

Los algoritmos planteados para obtener esqueletos usando Voronoi o straight skeletons no son aplicables al problema que nos ocupa. Si bien realizan el trabajo en tiempos y espacio inferiores a $O(n^2)$, la necesidad de lidiar con obstáculos mixtos de segmentos y sitios los hace inaplicables, pues son desarrollados en base a considerar sólo segmentos de un polígono.

Capítulo 4

Fortune en detalle y su extensión

4.1. Extendiendo Fortune

Recordemos que la motivación particular de este trabajo es obtener en tiempos razonables un camino posible para que un robot se mueva en medio de un entorno dinámico con una cantidad apreciable de obstáculos moviéndose.

No he podido encontrar bibliografía o referencias a trabajos que resuelvan este problema particular con los requerimientos de tiempo buscados. Por lo tanto, mediante este trabajo presento un algoritmo que en no más de $O(n\ log(n))$ resuelve diagramas con las características citadas, similares a 3.1(b).

Propongo una extensión del algoritmo de barrido de Fortune, utilizando las mismas propiedades geométricas y su paralelismo con la representacion tridimensional de conos proyectados en el espacio sobre los sitios. Esto permitirá ir descubriendo el diagrama mediante un barrido, nuevamente con un costo $O(n\ s(n))$ siendo n la cantidad de sitios y $s(n)$ el costo de mantener la línea de playa.

4.2. Extensión del modelo

Tomando como partida el modelo planteado en la Figura 2.6, tomemos uno de los puntos y extendámoslo para que tome el aspecto de un segmento. Si hacemos lo mismo con el cono que se proyectaba sobre él obtendremos un espacio similar a una carpa canadiense invertida pero terminada en sus extremos en dos semiconos. Los laterales de este volumen son dos semiplanos que surgen del segmento generado a 45° y por lo tanto son tangentes a los dos semiconos que surgen ahora de cada extremo del mismo segmento.

Si tomamos un rectángulo y transformamos cada uno de sus segmentos en obstáculos realizando las correspondientes proyecciones; generamos sobre el rectángulo, entre el plano que lo contiene y las proyecciones, un volúmen similar a un techo a cuatro aguas, todas a 45°. En la Figura 4.1 se ve una vista superior de las proyecciones generadas.

Si colocamos un sitio, y su cono, dentro del rectángulo y proyectamos la intersección del techo a cuatro aguas con el cono generado por el sitio sobre el plano dentro del

rectángulo obtenemos algo similar a lo que se observa en la Figura 4.2.

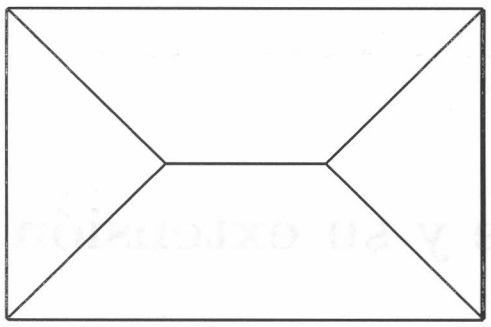


Figura 4.1: Proyecciones de los lados hacia el interior del rectángulo

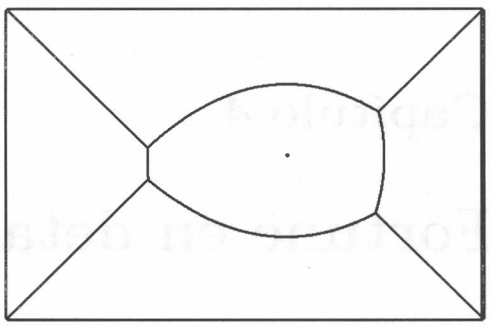


Figura 4.2: Proyecciones de los lados hacia el interior del rectángulo y la intersección con el cono del sitio en su interior

En la Figura 4.3 se observan las intersecciones entre el cono generado por el sitio y el plano que emana de cada uno de los lados. Estas intersecciones generan parábolas en el espacio que se cruzan sobre las líneas que son intersección entre los planos. Una apropiada combinación de intersecciones entre los planos y entre planos y parábolas genera la Figura 4.2.

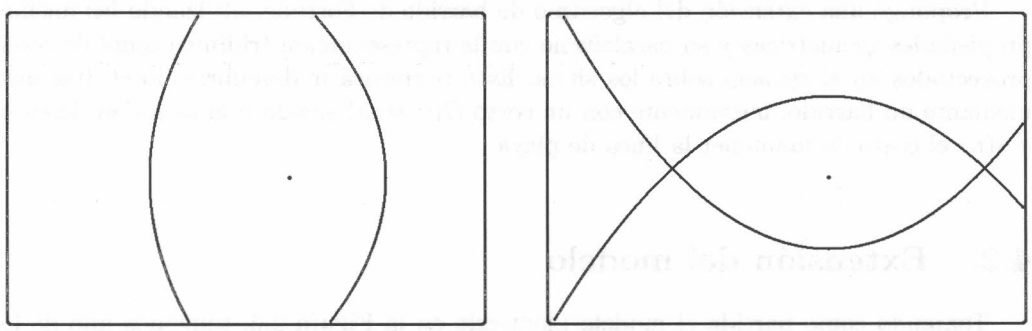


Figura 4.3: Intersecciones de los planos generados por cada lado con el cono generado por el sitio

Otra forma de ver las parábolas en la Figura 4.3 es la siguiente. Consideremos a cada segmento del perímetro como una línea de barrido que se aleja del centro del rectángulo. Esta línea de barrido genera entre el sitio y sí misma una parábola de la misma manera que ya hemos discutido en la sección 2.4. Cada parábola es un arco de la línea de playa que generaría el segmento si fuera una línea de barrido.

Es decir, seguimos trabajando con parábolas, tanto en el espacio como sobre el plano, y siguen valiendo las propiedades y distancias que enunciamos sobre la interacción línea de barrido-línea de playa. Por lo tanto la Figura 4.2 es un diagrama de Voronoi considerando como generadores de Voronoi al sitio y los lados del perímetro.

4.3. Extensión del algoritmo

El corazón del algoritmo sigue siendo el mismo, seguimos considerando una lista de prioridad con eventos de sitio y eventos de círculo.

4.3.1. Bases del algoritmo original

En el algoritmo original propuesto por Fortune la línea de playa modificaba su topología mediante dos eventos, el de sitio, que incorporaba un arco a la línea de playa, y el de círculo que lo eliminaba. Los eventos de sitios vienen dados, uno por cada sitio. Los eventos de círculo los buscabamos exhaustivamente cada vez que agregábamos un tacel al grafo intentando detectar futuros cruces.

A lo largo de las sección 2.4 se sostiene que el algoritmo funciona básicamente por lo siguiente:

1. Los quiebres en la línea de playa trazan los taceles de Voronoi.
2. Un arco en la línea de playa sólo aparece mediante un evento de sitio.
3. Un arco en la línea de playa sólo desaparece mediante un evento de círculo.
4. No hay otra forma de alterar la topología de la línea de playa.
5. No se deja de lado ningún posible evento de círculo.

En la secuencia de imágenes de la Figura 2.4 se observa que cuando dos taceles se cruzan en el punto q, más allá de la línea de playa, significa que al menos tres sitios son cocirculares constituyendo el círculo C_q con centro q. En este caso definimos un posible evento de círculo e_q para ser considerado en la lista de eventos.

Puede ocurrir que, entre el instante de definir el evento y el momento de atenderlo, ocurra otro evento que afecte la configuración de e_q. Por ejemplo, un tacel vecino podría generar otro evento de círculo e'_q posterior a la línea de playa pero anterior a e_q. Cuando llegue el momento de atender e'_q, e_q se descarta. Esta misma situación se da en la secuencia de Figuras 2.7 y 2.8.

Si nada de esto ocurre, atendemos finalmente el evento e_q cuando simultáneamente: a) la línea de barrido alcance el extremo de C_q haciéndose tangente, y b) la línea de playa alcance el punto q. Como en la imagen central de la Figura 2.4

La topología de la línea de playa se modifica, pues un arco desaparece. Los extremos de los taceles que se cruzan en p_q se definen y acotan en ese punto. Surge un nuevo tacel con uno de sus extremos también definido en p_q y el otro indefinido proyectándose al infinito. Y finalmente se revisa si el extremo libre del nuevo tacel se cruza con otro alguno de sus posibles dos vecinos en busca de otros eventos de círculo.

4.3.2. Nuevas definiciones

Antes de revisar las expresiones 2.1–2.4 redefinamos algunos términos. Sea

$$S^+ = \{\, st \mid st : Sitio(x), x \in S\} \cup \{\, sg \mid sg : Segmento(x), x \in \{Sur,\ Oeste,\ Norte,\ Este\}\}$$

el conjunto sobre el que obtendremos $V(S^+)$ y

$$d^+(s,x) = \text{mín}(\{\ d(p,x) \mid p \in s\})$$

donde x es un punto en el plano y $s \in S^+$ y la pertenencia de puntos en *Sitios* o *Segmentos* es la esperable.

Luego, podemos revisar la expresión 2.1 redefiniéndola como

$$B(p,q) = \{\ x \mid d^+(p,x) = d^+(q,x)\ \}$$

pero ahora, con p y $q \in S^+$. $B(p,q)$ define la *curva* que separa las regiones de los generadores p y q que puede ser una recta o una parábola.

Esta *curva* definida por $B(p,q)$ genera los espacios abiertos

$$D(p,q) = \{\ x \mid d^+(p,x) < d^+(q,x)\ \}$$

que contiene a p y $D(q,p)$ que contiene a q. Nuevamente, llamaremos

$$RV(p,S^+) = \bigcap_{q \in S^+, q \neq p} D(p,q)$$

la región de Voronoi de p con respecto de S^+. Finalmente el diagrama de Voronoi de S queda definido como

$$V(S^+) = \bigcup_{p,q \in S^+, q \neq p} \overline{RV(p,S^+)} \cap \overline{RV(q,S^+)}.$$

Como la diferencia entre S y S^+, y entre $d()$ y $d^+()$ puede deducirsce del contexto, de ahora en más las notaremos como S y d.

4.3.3. Nuevos taceles

Pero antes un tacel siempre era una recta, ahora puede ser también una parábola. ¿Qué características tienen?

Recordemos que un tacel en un diagrama de Voronoi es la intersección de dos regiones de Voronoi vecinas. Los puntos de una parábola se hallan equidistantes de la línea (en nuestro caso los segmentos perimetrales) y del foco que la genera.

Conforman un tacel los puntos a lo largo del tramo de la parábola que se hallan más cerca del sitio y del segmento que genera la parábola que de cualquier otro sitio o segmento.

Si un punto de la parábola se halla equidistante del sitio, del segmento en cuestión y de cualquier otro generador, ese punto es un extremo del tacel.

En la Figura 4.4 se aprecia un segmento horizontal y tres sitios p_i, p_j y p_k. El tacel es el sector de la parábola que se halla sobre la zona grisada. Nótese que un tacel contra un segmento es exactamente lo mismo que una componente de la línea de playa sobre la línea de barrido.

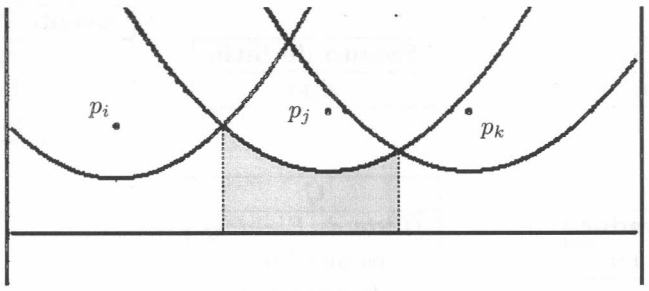


Figura 4.4: Tres puntos p_i, p_j y p_k con su respectivas parábolas. La porción de la parábola sobre la zona gris conforma un tacel.

Estos taceles parabólicos también estarán definidos o indefinidos en sus extremos y contarán con un semitacel parabólico por cada extremo libre. Los extremos definidos serán aquellos que tengan un punto concreto de inicio por tocarse con otros taceles. Los extremos indefinidos serán proyectados al infinito.

Los cruces de las proyecciones de estos nuevos taceles serán considerados exactamente de la misma manera a la hora de preveer eventos de círculo. Más allá de su forma y de algunas consideraciones a la hora del cálculo geométrico, no haremos diferencia alguna entre taceles rectos o parabólicos.

4.4. El algoritmo propuesto

Habiendo dado los pasos y definiciones necesarios podemos avanzar y observar en detalle el algoritmo propuesto.

4.4.1. Estructura de datos

Para facilitar el seguimiento de los algoritmos y el flujo de información primero presento los tipos de datos utilizados. En la Figura 4.5 observamos los principales Tipos que componen el modelo de datos desarrollado para el algoritmo. Se omiten algunos componentes internos por que son propios de la implementación y no aportan a la hora de esclarecer cómo se relacionan entre sí los distintos Tipos. Cuando sea conveniente algunos de estos atributos serán detallados.

4.4.2. Pseudocódigo

Según el algoritmo de Fortune deben considerarse los generadores de Voronoi siguiendo su orden.

En el algoritmo original, el primer sitio se agregaba sin más trámite. Los siguientes siempre se ubicarían en algún punto sobre alguna parábola generada por los sitios

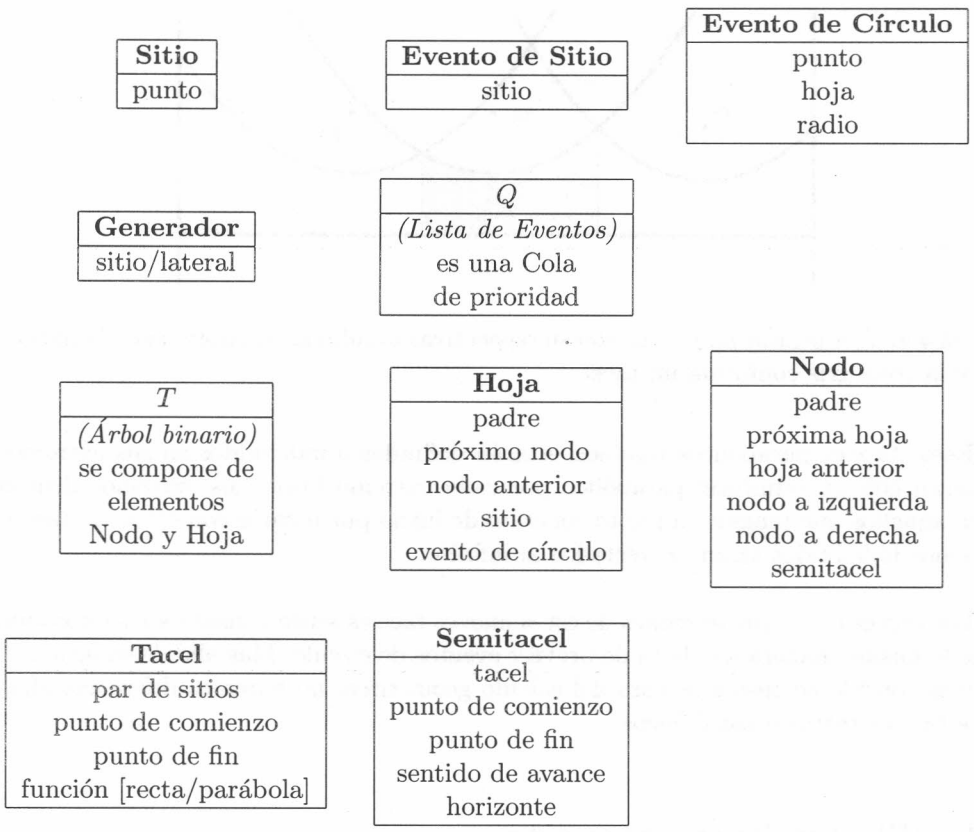


Figura 4.5: Principales Tipos y sus atributos que componen el modelo de datos desarrollado para el algoritmo.

anteriores.

Pero en el algoritmo que presento, siempre el primer obstáculo es el marco. Si tomamos el orden de izquierda a derecha, el primer atractor a considerar es el (segmento) lateral izquierdo, luego el inferior y superior en un orden determinado pero arbitrario. Sólo entonces podemos considerar los sitios. Así, la única diferencia con el algoritmo original es que los primeros tres atractores a considerar son los segmentos del perímetro.

La función INICIALIZAR$T()$ inicializa T considerando los tres laterales mencionados generando un árbol como el de la Figura 4.7.

Algoritmo INICIALIZAR$T()$
Output: El árbol de estado T

1. Sean hojaInf, hojaCent, HojaSup hojas asociadas a los segmentos atractores inferior, izquierdo y superior respectivamente.
2. Sean nodoSup y nodoInf nodos.
3. Asociar a cada nodo su tacel y semitacel.
4. Definir nodoInf como raiz de T.
5. Definir hojaInf como hijo izquierdo de nodoInf.
6. Definir nodoSup como hijo derecho de nodoInf.
7. Definir hojaCent como hijo izquierdo de nodoSup.
8. Definir hojaSup como hijo derecho de nodoSup.

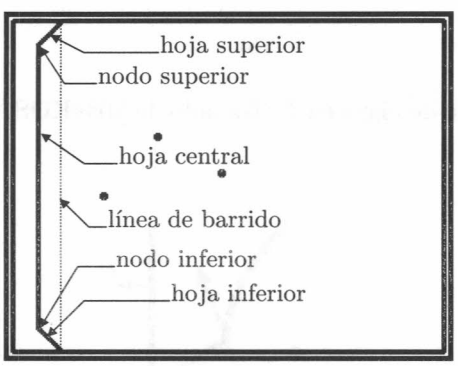


Figura 4.6: Línea de playa avanzada hacia la derecha.

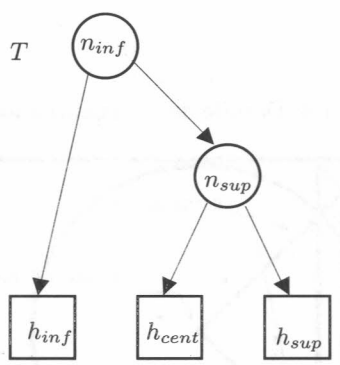


Figura 4.7: Árbol T correspondiente a la línea de playa de la Figura 4.6.

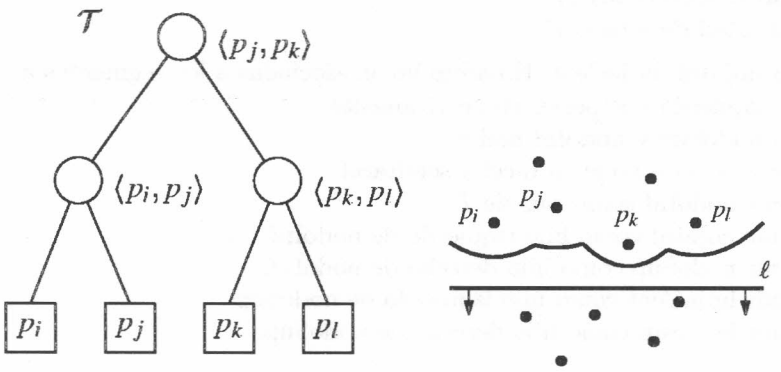
En las Figuras 4.6 y 4.7 se observa como los componentes parabólicos o rectos de la línea de playa se organizan en las hojas del árbol binario T y como cada quiebre se asocia al nodo interno correspondiente.

La línea de playa de la Figura 4.6 se muestra ya avanzada hacia la derecha para que puedan apreciarce sus tres componentes, sino, los componentes diagonales tendrían longitud cero. También vale recordar que avanzar la línea de playa no modifica el árbol mientras no ocurra ningún evento.

Un caso más general puede observarse en la Figura 4.8.

Como ya dijimos, en cada quiebre de la línea de playa hay asociado un tacel que se compone de dos semitaceles. Estos tienen origen sobre el punto en que el tacel es descubierto y, sobre la misma dirección, parten en sentidos distintos. En la Figura 4.10 se ve como un tacel es trazado por la línea de playa. El punto a es la proyección del sitio p_i sobre el tacel descubierto y a su vez el origen de los semitacles que parten en los sentidos que marcan las flechas. El punto a fue también el punto donde la parábola que genera p_i tocó por primera vez (en su estado inicial degenerada) a la otra parábola.

Para llegar a la Figura 4.9 y observar el resultado completo de inicializar T debemos sumar a la Figura 4.6 los dos taceles ya descubiertos y el evento de círculo que provoca



(a) (b)

Figura 4.8: Detalle de la organización de la línea de playa en T. Tomado de [dBvKOS/94].

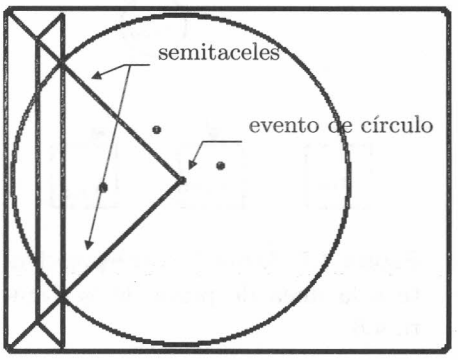


Figura 4.9: Línea de playa instantes después de comenzar incluyendo semitaceles y evento de círculo.

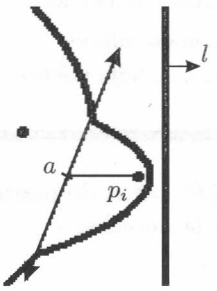


Figura 4.10: Detalle de un tacel siendo trazado por la línea de playa.

su futura unión. Cada tacel es descubierto en su esquina respectiva y ese punto es el origen de los dos semitaceles correspondientes. Uno de esos dos semitaceles tiene sentido hacia afuera y entra en contacto con el cruce de los segmentos laterales. En consecuencia su extención es cero, pues se extiende desde el punto en el que es descubierto el tacel hasta el cruce de los atractores que lo generan. El semitacel opuesto se extiende, a priori, hasta el infinito. En este caso particular, habiendo dos taceles descubiertos, los dos semitaceles que se proyectan hacia el infinito se cruzan generando el evento de círculo.

Entonces, teniendo en cuenta la inicialización de T, el algoritmo original queda modificado de la siguiente manera:

Algoritmo DIAGRAMADEVORONOI(P)
Input: Conjunto $P = \{p_i\}$
Output: El diagrama $Vor(P)$

1. Por cada elemento $p \in P$, agregar a la lista ordenada de eventos Q un sitio en p.
2. INICIALIZAR$T()$.
3. Mientras Q no vacía:
4. primer evento e en Q:
5. si e evento de sitio: HANDLESITEEVENT(e)
6. si e evento de círculo: HANDLECIRCLEEVENT(e)
7. Llevar la línea de barrido al lateral derecho.
8. Hacer taceles de los componentes de la línea de playa.
9. Cerrar todos los semitaceles que queden abiertos.

Algoritmo HANDLESITEEVENT(e)

1. Obtener hoja h y sitio s que están sobre e.
2. Eliminar de la lista Q y de h el evento de círculo de h, si existiera.
3. Obtener una nueva hoja h_e que crecerá dentro de h, actualizar T.
4. Buscar nuevos eventos de círculo en los vecinos de h_e.

Algoritmo HANDLECIRCLEEVENT(e)

1. Tomar h la hoja que desaparecerá asociada a e.
2. Borrar todos los eventos de círculo asociados a h y a sus hojas vecinas, de tenerlos.
3. Si se desea, registrar el centro del evento de círculo como v, el nuevo vértice del diagrama de Voronoi.
4. Reducir T eliminando h y sus dos nodos vecinos, generar un nuevo nodo que relacione las dos hojas que ahora son vecinas.
5. Ligar al nuevo vértice el punto de fin de los semitaceles asociados a los quiebres entre los arcos que confluyen entre sí.
6. Crear nuevos tacel y semitacel emergiendo de v. Ligar en ambos el punto de comienzo con v.
7. Buscar nuevos eventos de círculo en los probables cruces del nuevo semitacel con sus semitaceles vecinos.

4.4.3. Ejemplo de corrida

En la siguiente serie de figuras veremos como evoluciona el algoritmo. Estas figuras y varias de las ya mostradas son parte del software desarrollado para este trabajo que se comenta en detalle en el anexo A.

En la Figura 4.11 vemos el área ya dispuesta.

En la Figura 4.12 observamos que se han cumplido los pasos 1 y 2 de DIAGRAMADEVORONOI(P). Pero, a diferencia de la explicación dada sobre las Figuras 4.9 y 4.10 la línea de barrido no se ha movido. Resumiendo, ya se han considerado los tres primeros laterales; hay dos semitaceles que parten de las esquinas y se cruzan

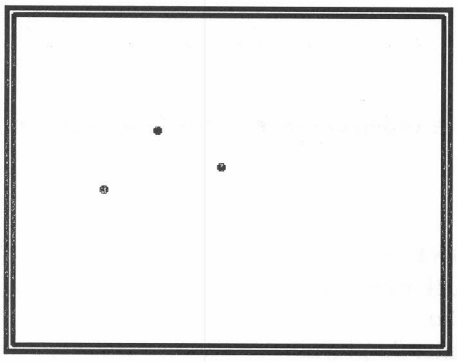


Figura 4.11: Los sitios en el tablero, listos para comenzar.

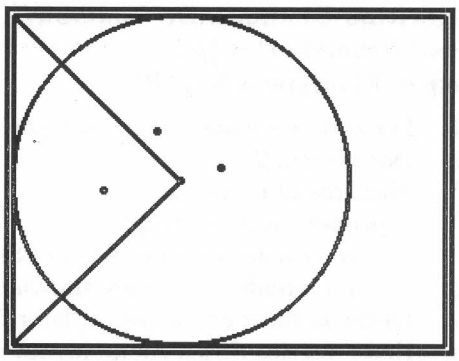


Figura 4.12: Inicialización

mas allá de la línea de barrido dando lugar al primer evento de círculo; la línea de playa consta ya de tres componentes, dos diagonales de longitud cero que surgen del los bordes de la línea de barrido (en contacto con los laterales) y se dirigen hacia atrás al centro, y un tercer componente situado entre los otros dos que va de lado a lado, paralelo al lateral izquierdo y a la línea de barrido; y el árbol T posee tres hojas, de las cuales la central tiene asociado el evento de círculo encontrado.

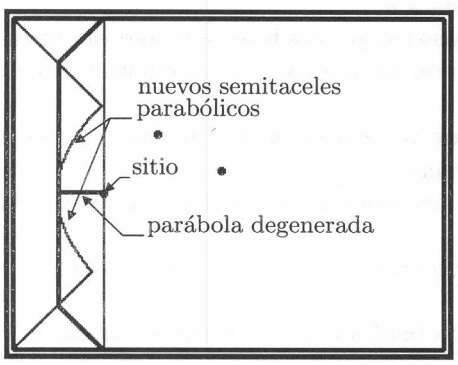


Figura 4.13: Se atiende el primer evento de sitio.

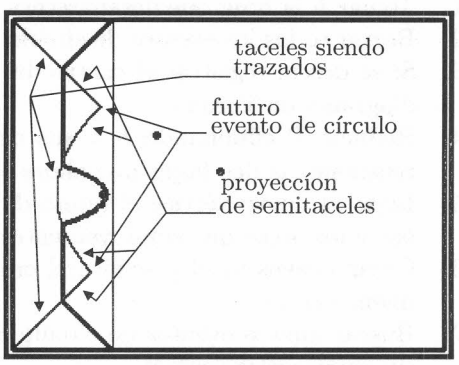


Figura 4.14: Detalle de línea de playa y taceles.

En la Figura 4.13 podemos observar el resultado de atender el primer evento de sitio. La línea de barrido está sobre el sitio encontrado. La línea de playa tiene cinco componentes, la central es la parábola degenerada indicada en la figura. A sus flancos están las componentes que separan el lateral izquierdo de la línea de barrido. Antes de la ocurrecnia del evento de sitio estas dos componentes eran una sola. En los extremos siguen estando las componentes diagonales que parten/llegan hacia la línea de barrido.

Para facilitar la tarea del lector en la Figura 4.14 se ha avanzado un poco la línea de barrido de manera que las cinco componentes de la línea de playa se vean con facilidad. Se aprecian tambien los tres taceles que están siendo trazados por los quiebres de la línea de playa. En particular el tacel parabólico tiene extensión cero al momento de la

Figura 4.13. También se aprecian las proyecciones de los semitaceles y los dos puntos en los que posiblemente se crucen generando un evento de círculo.

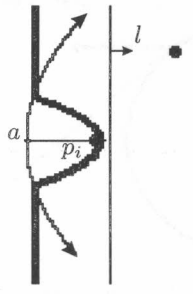


Figura 4.15: Detalle de la generación de un tacel parabólico y sus dos semitaceles

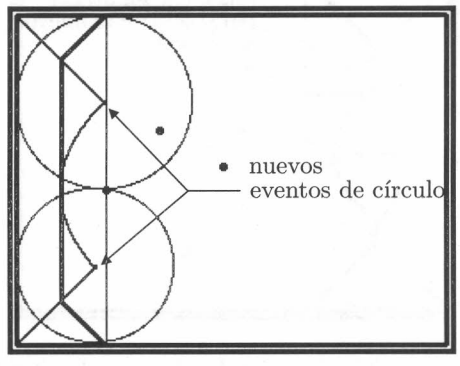


Figura 4.16: Evento de sitio atendido y cruces de taceles revisados

En la Figura 4.15 vemos un detalle de como se genera un tacel parabólico y sus dos semitaceles. El origen de los semitaceles es el punto a, aquel en donde la parábola degenerada de p_i tocó inicalmente la componente central de la línea de playa en la Figura 4.12. En ese instante el tacel tenía longitud cero pero sus semitaceles ya se extendían hacia el infinito.

Por último, en la Figura 4.16 vemos los dos eventos de círculo descubierto en el posible cruce de los semitaceles. Notar que siempre que se encuentre un posible cruce, no tiene sentido seguir dibujando (y considerando) ese semitacel desde ese punto en adelante. Llamo a este punto *horizonte* del semitacel. En la Figura 4.12 se ve que los semitaceles diagonales se dibujan sólo hasta su cruce. De la misma manera en la Figura 4.16 se dibujan sólo hasta su primer cruce con otro semitacel.

Recordemos que cuando se detecta un cruce entre taceles significa que, si nada interfiere, cuando la línea de playa alcance ese punto tendremos tres generadores de Voronoi cocirculares. Esto implica que esos dos taceles a lo sumo llegan hasta ese punto. ¿Cuándo algo interfiere? Cuando la línea de barrido descubre otro generador y éste genera nuevos semitaceles que se cruzan con los taceles anteriormente considerados, interfiriendo en sus planes de cruzarse. Ante esta situación es claro que aquellos semitaceles llegarán a lo sumo al nuevo cruce. ¿Por qué vale cuando se crucen antes y no después? Por los mismos motivos. Consideremos nuevamente dos semitaceles que tienen previsto cruzarse en un futuro. Si el descubrimiento de un nuevo generador diera lugar a un nuevo semitacel que se cruzara con la proyección más allá del horizonte de uno estos semitaceles, no tiene sentido considerar en este cruce un evento de círculo, pues el otro evento de círculo sucederá primero y, para entonces, no habrá tacel con quien cruzarse.

Esto es precisamente es lo que ha ocurrido en la Figura 4.16, los dos primeros semitaceles que esperaban cruzarce en la Figura 4.12 en el evento de círculo han sido interferidos por la ocurrencia de dos nuevos semitaceles que los cruzarán antes. Estos nuevos cruces generan sus respectivos eventos de círculo y descartan el evento de círculo anterior que ya no se aprecia en la Figura 4.16.

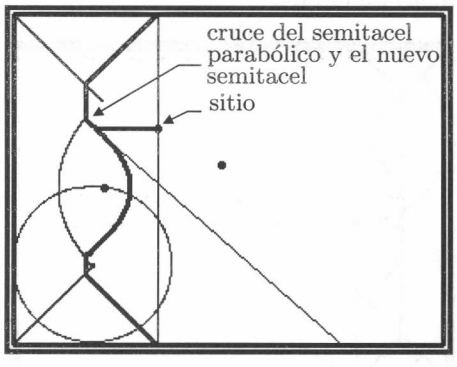


Figura 4.17: Se atiende el segundo evento de sitio.

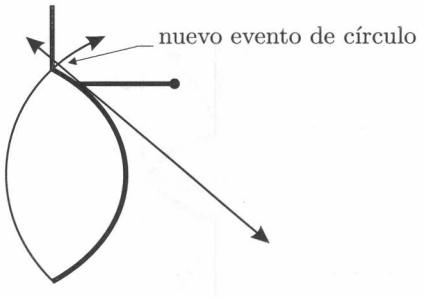


Figura 4.18: Detalle del nuevo evento de círculo.

En la Figura 4.17 la línea de barrido llega hasta el segundo sitio y observamos el resultado de atender un evento de sitio. La proyeción del sitio da sobre un costado de la componente central de la línea de playa. Se generan dos semitaceles rectos que trazarán el tacel que separará al nuevo sitio del sitio que genera la componente central. El nuevo semitacel que avanza hacia la derecha tiene su horizonte en el marco mismo. Mientras que el otro semitacel tiene su horizonte en el semitacel parabólico. Este nuevo cruce, ampliado en Figura 4.18, provoca el descarte del evento de círculo entre el tacel parabólico y aquel que avanza desde la esquina superior izquierda que ya no se incluye en la Figura 4.17. Nótese que el tacel proveniente de la esquina sigue dibujándose hasta su horizonte anterior. Finalmente, el nuevo cruce provoca la aparición de un nuevo evento de círculo. La Figura 4.19 resume esta situación.

Notemos que cuando definimos un evento de círculo sobre una hoja, se descartan los eventos de círculo que tengan asociadas las hojas vecinas. Porque, si logramos definir uno de éstos eventos, dos semitaceles se cruzan en su recorrido entre su punto de inicio y su horizonte, el punto de cruce pasa a ser el horizonte de cada semitacel. Si el horizonte anterior se debía a otro evento de círculo es claro que este último está desactualizado pues antes que éste semitacel se cruce con un tercer semitacel, ocurrirá el nuevo cruce encontrado que está más cerca de la línea de playa.

El evento de círculo agregado en el paso anterior es el próximo evento en ser atendido. Volvamos al detalle de la Figura 4.18 y proyectemos la situación de la línea de playa y los taceles tal como se ve en la Figura 4.20 al momento de atender el nuevo evento de círculo.

Antes de refrescar algunos de los conceptos ya vertidos detallemos el contenido de la Figura 4.20. En trazo grueso observamos la línea de playa en el momento de detectar el evento de círculo que estamos por atender (Figura 4.18). En trazo punteado observamos la línea de playa avanzada (de manera exagerada para facilitar la comprensión) al momento de atender el evento de círculo. La componente asociada al sitio p_i ya se aprecia como una parábola; en cuyo despliege ha ido trazado el tacel recto t_r sobre las componentes parabólicas c_{pa} y c_{pb} .

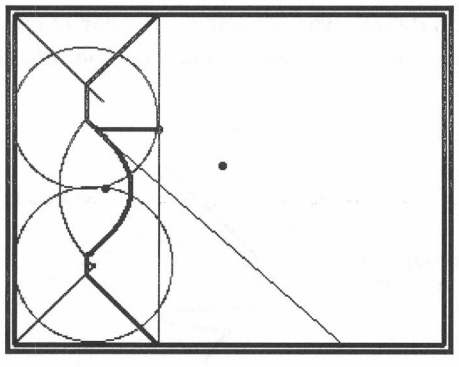


Figura 4.19: Nuevo evento de círculo.

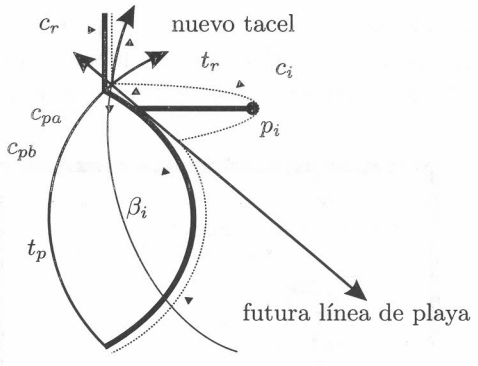


Figura 4.20: Detalle al atender el evento de círculo definido en la Figura 4.19.

Desde el instante de detectar el evento de círculo y hasta el instante de atenderlo la línea de playa se compone, comenzando desde su parte superior, de una componente recta c_r, seguida de la componente parabólica c_{pa}, que está seguida por la componente parabólica c_i (en un principio degenerada) asociada al sitio pi, que está seguida de la componente parabólica c_{pb}. En el instante de detectar el evento de círculo la parábola se observa degenerada en una recta, en el instante de atender el evento de círculo la componente c_{pa} se observa de longitud cero.

Antes de la ocurrencia del sitio p_i las componentes c_{pa} y c_{pb} eran una sola, precisamente c_i las dividió. El quiebre entre c_r y c_{pa} traza uno de los extremos del tacel parabólico t_p, mientras que el quiebre entre c_{pa} y c_i traza el extremo que retrocede del tacel recto t_r. Ahora bien, en el paso anterior encontramos que t_p y t_r son trazados por los distintos extremos de una componente y se cruzan por delante de la línea de playa, motivos necesarios y suficientes para considerar un futuro evento de círculo (Figura 4.18). Si la línea de barrido llega hasta el punto de cruce, necesariamente la componente asociada al evento de círculo desaparece. En este caso c_{pa} desaparece fruto del avance de c_r y la expansión de c_i.

Al desaparecer una componente las componentes que ahora son vecinas comienzan a trazar un nuevo tacel, que no requiere de dos semitaceles pues uno de sus extremos ya está resuelto. ¿Qué función tiene asociado el nuevo tacel? Si los atractores son dos sitios o dos segmentos la función es una recta. Mientras que en el caso mixto la función es una parábola. El nuevo tacel marcará el límite de proximidad entre el sitio p_i y el segmento izquierdo siguiendo la curva de la parábola β_i.

Note el lector que estamos simplemente ante a una variante de la situación descripta en la secuencia de la Figura 2.4.

Finalmente, como es de rigor, hay que buscar nuevos cruces entre el nuevo semitacel y sus vecinos inmediatos. En particular, el nuevo semitacel parabólico se cruza con el semitacel que desciende desde la esquina superior izquierda dando a lugar a otro evento de círculo que puede ser observado en la Figura 4.21.

Vale notar que nunca un evento de círculo se sucederá por detrás de la línea de barrido. Pues los semitaceles que se cruzan, lo harán a futuro, cercano o lejano, pero futuro;

lo que implica que las línea de playa y de barrido deben avanzar para provocar el cruce. Es por esto mismo, que no debemos preocuparnos de que algún evento inesperado nos obligue a hacer retroceder la línea de playa.

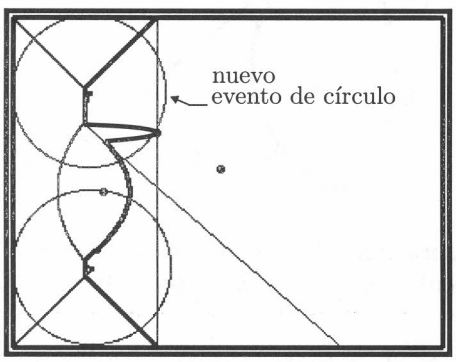


Figura 4.21: Nuevo evento de círculo (y próximo en ser atendido).

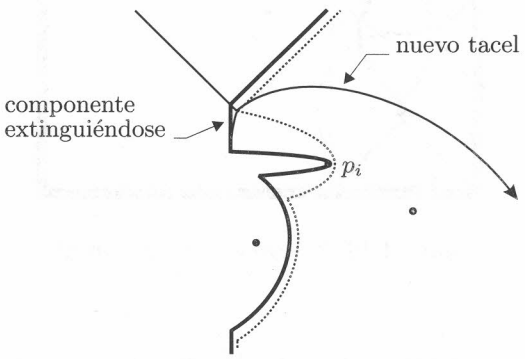


Figura 4.22: Detalle de del resultado de atender el segundo evento de círculo.

En el detalle de la Figura 4.22 podemos observar como (fruto del avance de la línea de barrido) la componente parabólica asociada al sitio p_i se expande a la vez que los componentes rectos contiguos avanzan. La componente recta vertical se extingue entre sus componentes vecinas. En el instante en que su longitud se hace cero atendemos un evento de círculo. De atender el evento surge un nuevo tacel parabólico que marcará el límite de proximidad entre los atractores p_i y lateral superior.

En la Figura 4.23 observamos la situación final de este paso. Hemos revisado si el nuevo tacel se cruza con algún otro tacel en busca de nuevos eventos de círculo pero tal cruce no existe.

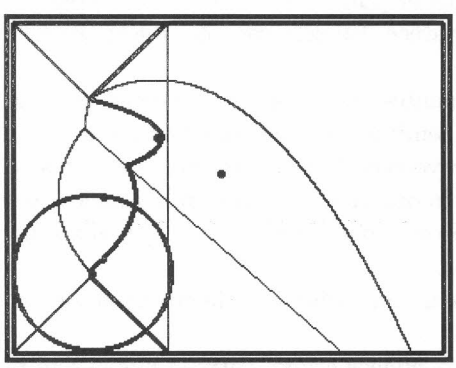


Figura 4.23: Listos para atender el próximo evento de círculo.

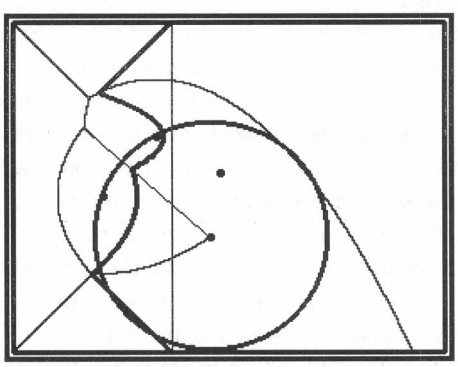


Figura 4.24: El evento de círculo ha sido atendido y generó uno nuevo.

En la Figura 4.24 observamos el resultado de atender el evento de círculo señalado en la Figura 4.23. De la misma manera, una componente se extinguió y un nuevo tacel surgió. Pero esta vez encontramos que se cruzaba con otro tacel y generamos el

correspondiente evento de círculo (en trazo grueso).

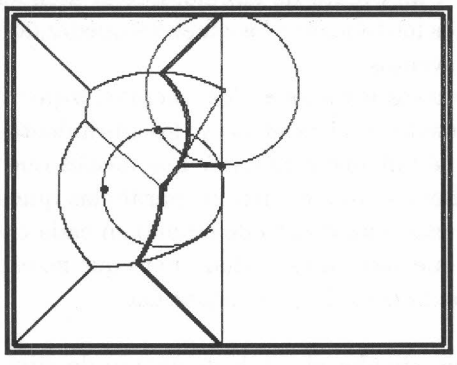


Figura 4.25: Atendemos el tercer evento de círculo.

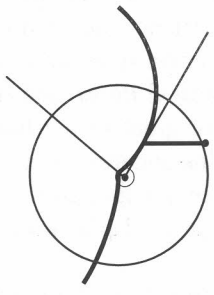


Figura 4.26: Detalle de los taceles que generan el próximo evento de círculo a atender.

En la Figura 4.25 observamos el resultado de atender el último evento de sitio. Nuevamente aparece un tacel tangencial a la componente sobre la que se proyecta el sitio correspondiente al evento atendido. En este caso cada semitacel se cruza con otro semitacel proveniente de un quiebre inmediato anterior o posterior, provocando la aparición de dos nuevos eventos de círculo.

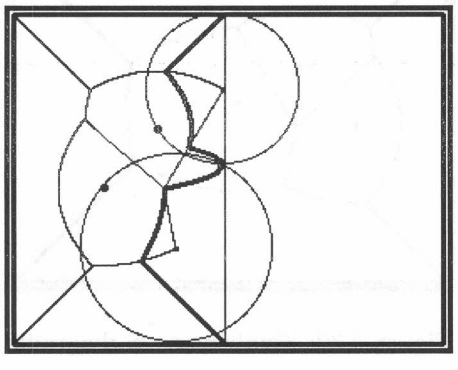


Figura 4.27: Atendemos el cuarto evento de círculo.

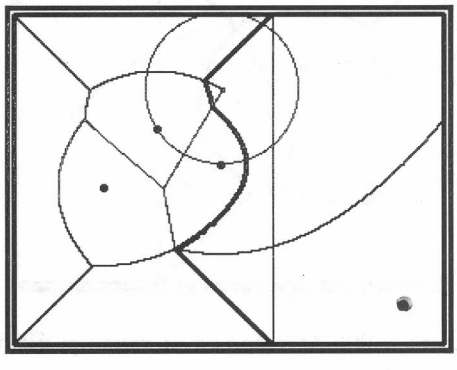


Figura 4.28: Atendemos el quinto evento de círculo.

En la Figura 4.27 observamos, nuevamente, el resultado de atender un evento de círculo. Una componente de la línea de playa desaparece, dos taceles se unen y surge uno nuevo. Eventualmente, como en este caso, el nuevo tacel se cruza con otro preexistente y se genera otro evento de círculo.

A estas alturas, es probable que el lector este haciendo cuentas mentales tratando de inferir cuántos eventos de círculo pueden ocurrir.

Veamos, volviendo al algoritmo original, el primer sitio genera la primera parábola. Sobre ésta surgirá la próxima parábola generada por el próximo sitio provocando dos quiebres. En total durante el desarrollo del algoritmo obtendremos dos quiebres por

cada sitio, es decir $2n$ quiebres. Ahora bien, en el peor caso, cada par de semitaceles proyectándose apartir de sendos quiebres genera un evento de círculo, que al atenderlo aporta otro quiebre, y otro semitacel. Si repetimos hasta agotar los quiebres quedándonos con uno solo, hemos consumido a los sumo $2n$ eventos.

En el algoritmo que planteo tenemos tres nuevos atractores, los laterales izquierdo, superior e inferior, que van a influir particularmente en el tacelado y descubrimiento de eventos durante el desarrollo del algoritmo. ¿Qué influencia ejercen? Los taceles que en el algoritmo original se perdían en el infinito ahora se dan contra las parábolas que sus sitios asociados generan contra los laterales, forzando un evento de círculo en cada caso. El resultado es que ganamos eventos de círculo que antes no sucedían, pero que no están fuera del cálculo del párrafo anterior, por lo que la cota de $2n$ se mantiene.

En la Figura 4.28 observamos el resultado de atender el siguiente evento de círculo. Esta vez obtenemos un semitacel acotado sólo por el lateral derecho. Como en la atención de los eventos anteriores, aunque no lo mencionara hasta este momento, cada vez que un semitacel se cruza contra alguno de los segmentos que hacen de marco recibe ese punto de cruce como horizonte. Cuando ese semitacel se cruze contra algo, y es seguro que lo hará, a lo sumo será con el marco. En consecuencia no tiene sentido considerar cualquier semitacel mas allá del marco y se fija su cruce con éste como horizonte.

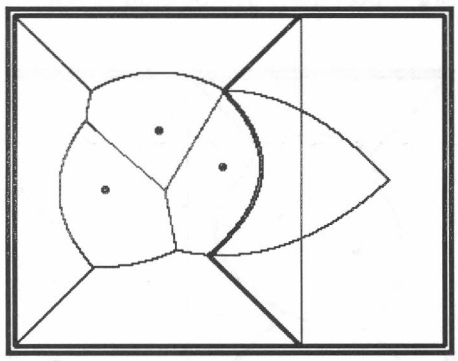


Figura 4.29: Atendemos el sexto y último evento de círculo.

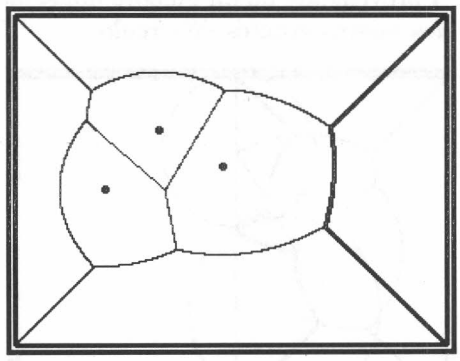


Figura 4.30: Finalizamos el diagrama

En la Figura 4.29 observamos el resultado de atender el último evento de círculo. ¿Esta vez dos semitaceles se cruzan pero no definimos un evento de círculo? Si estubiéramos construyendo un diagrama de Voronoi en un plano infinito, consideraríamos cada evento que surgiera. Pero como sólo nos interesa lo que ocurre dentro de los segmentos que hacen de marco descartamos aquellos eventos que sucederán por fuera. Notemos que en este cruce los tres atractores cocirculares son los laterales superior e inferior, más el último de los sitios, lo que nos da un radio para el evento de círculo igual al ancho del marco. Para atender a este evento la línea de barrido debería llegar más allá del último lateral.

Finalmente en la Figura 4.30 observamos el diagrama final con la última posición de la línea de playa en trazo grueso correspondiente a llevar la línea de barrido hasta el

lateral derecho, el ultimo de los atractores. En este momento, por su definición geométrica, las componentes de la línea de playa coinciden con los taceles que separan las áreas de influencia de cada atractor en contacto con la línea de playa. Entonces, damos por terminado el diagrama fijando como punto final de los semitaceles libres el punto en el que éstos tocan la línea de playa, e incorporando al diagrama los componentes de la línea de playa traducidos en taceles.

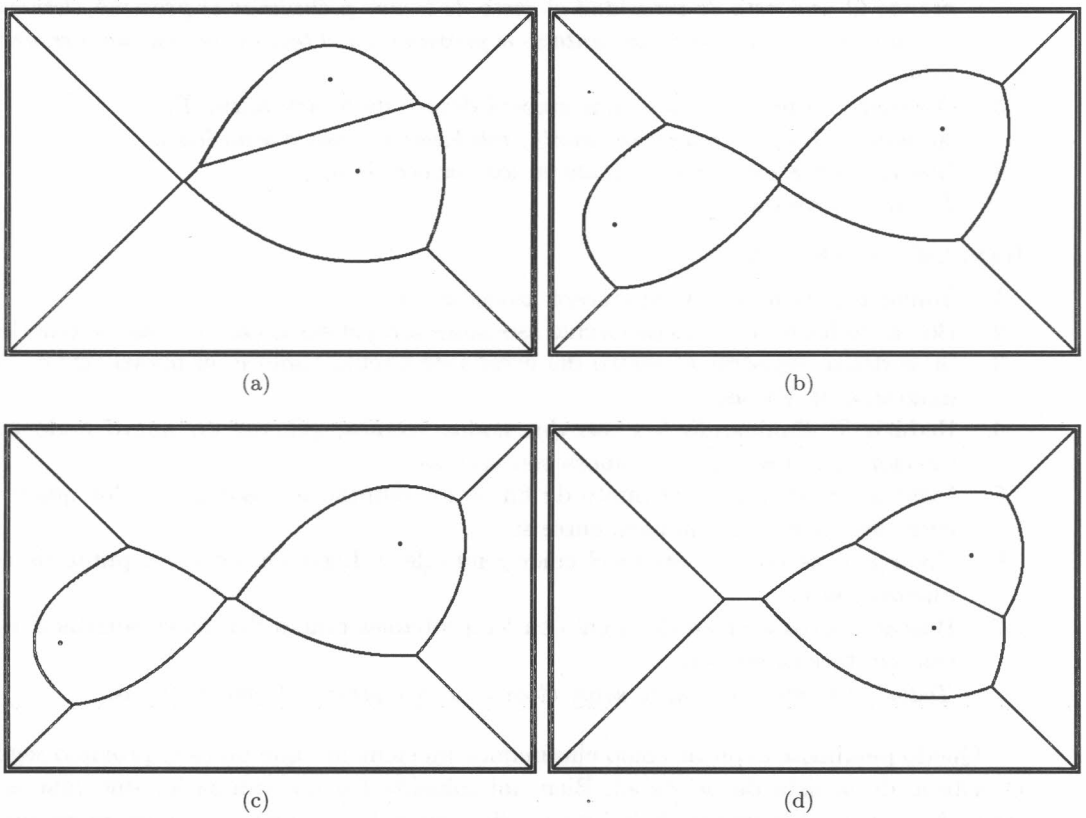


Figura 4.31: Ejemplos de diagramas de Voronoi.

En la Figura 4.31 podemos observar otras variantes de tacelados resultantes para diagramas con pares de sitios.

4.4.4. Análisis de complejidad temporal

Ya he mencionado que el algoritmo de línea de barrido propuesto por Fortune tiene un tiempo de corrida de $O(n\ log(n))$, pero todavía no he detallado por qué.

El procedimiento principal DIAGRAMADEVORONOI(P) tiene un ciclo que consume los eventos. Hay dos tipos de eventos, los eventos de sitio, uno por cada sitio; y los eventos de círculo, acotados por $2n$. Por cada evento se invoca a HANDLESITEEVENT(

e) o a HANDLECIRCLEEVENT(*e*).

Veamos el costo de estos dos precedimientos:
HANDLESITEEVENT(e)

1. Obtener hoja h y sitio s que están sobre e.
   Implica revisar el árbol T de profundidad log n en busca de la hoja.
2. Eliminar de la lista Q y de h el evento de círculo de h, si lo tuviera.
   Siendo Q una lista de prioridad el costo de tomar y eliminar el próximo elemento es log n, pero e no necesariamente es el próximo, en el texto veremos como resuelvo esto.
3. Obtener una nueva hoja h_e que crecerá dentro de h, actualizar T.
   Actualizar T lleva tiempo constante, rebalancearlo nos cuesta log n.
4. Buscar nuevos eventos de círculo en los vecinos de h_e.
   De tiempo constante.

HANDLECIRCLEEVENT(e)

1. Tomar h la hoja que desaparecerá asociada a e.
2. Borrar todos los eventos de círculo asociados a h y a sus hojas vecinas, de tenerlos.
3. Si se desea, registrar el centro del evento de círculo como v, el nuevo vértice del diagrama de Voronoi.
4. Reducir T eliminando h y sus dos nodos vecinos, generar un nuevo nodo que relacione las dos hojas que ahora son vecinas.
5. Ligar al nuevo vértice el punto de fin de los semitaceles asociados a los quiebres entre los arcos que confluyen entre sí.
6. Crear nuevos tacel y semitacel emergiendo de v. Ligar en ambos el punto de comienzo con v.
7. Buscar nuevos eventos de círculo en los probables cruces del nuevo semitacel con sus semitaceles vecinos.
   Todos de tiempo constante salvo el caso de reorganizar T que es log n.

Quedó pendiente explicar como eliminamos un elemento que no es el próximo según el criterio de la lista de prioridad. Bien, mi solución fue no eliminarlo, sino marcarlo como descartado. Eliminarlo de la lista implica buscarlo con costo n y luego reorganizar el árbol con el que se implementa la lista con costo *log* n. En contraste consumir el próximo elemento tiene costo *log* n. Marcar el evento como descartado provoca que cuando llegue el tope de la lista sea consumido inmediatemente.

¿Cuál es el peor caso de esta estrategia? Si eliminámaramos todos los eventos salvo el primero en la lista, cuando lo consumimos tendríamos un costo de reorganizar la lista de la totalidad de los eventos en ella que es $O(n\ log(n))$. Cierto, pero no es tan grave. Al marcar un evento como descartado sólo estamos difiriendo su eliminación de la lista a futuro. El costo temporal del algoritmo no depende de los eventos efectivamente atendidos sino de todos los posibles. Y cada evento generado ya está considerado, sea que su costo se sume paso a paso, o que ocurra todo en conjunto en un momento dado. En definitva no hay peor caso, cada elemento que incorporemos al la lista tendrá un costo *log* n y cada eliminación también lo tendrá.

Entonces nos queda que tenemos $O(n)$ eventos con un costo de atención $O(log(n))$, tanto para los eventos de círculo como los de sitio. En conjunto obtenemos un costo temporal de $O(n\ log(n))$.

4.4.5. Análisis de complejidad espacial

Para analizar la complejidad espacial del algoritmo repasemos las estructuras utilizadas.

Sabemos que el Arbol T es de profundidad $O(log(n))$, lo que implica una complejidad espacial $O(n)$. La lista de prioridad de eventos Q arranca con n eventos de sitio y se le pueden agregar hasta $2n$ eventos de círculo (por lo visto en la explicación de la Figura 4.27), lo que nos deja nuevamente con $O(n)$ de complejidad espacial. Finalmente, por el Lema 2.1.1 sabemos que la cantidad de taceles es de $O(n)$ que implica una complejidad espacial de $O(n)$ para taceles y semitaceles.

Capítulo 5

Caso de estudio

5.1. Introducción

En el Capítulo 1 afirmaba que determinar el recorrido a seguir por un robot puede ser una tarea compleja que puede ser atacada y resuelta satisfactoriamente con diagramas de Voronoi, que por definición constituyen una red o mapa de senderos válidos a la hora de guiar al robot a través de los obstáculos considerados.

En el Capítulo 3 notamos que si los obstáculos a considerar son no puntuales, el costo de obtener el diagrama de Voronoi puede incrementarse sensiblemente de acuerdo a la técnica usada para construirlo. Esto dio lugar a este trabajo de tesis en donde desarrollo un algoritmo que resuelve la problemática de los diagramas de Voronoi con obstáculos no puntuales con un costo temporal de $O(n\ log(n))$.

Ahora bien, ¿dónde, o cuándo es necesario un algoritmo de estas características? Supongamos un campo minado entre construcciones que debe ser explorado y desactivado. En este caso bien podemos esperar lo necesario para construir el mapa y no tenemos un objetivo específico al que arribar. Pero si lo que nos urge es cruzarlo, entonces es conveniente usar un algoritmo con tiempos de corrida eficientes que nos permitan avanzar evitando las minas. Mas aún, necesitamos saber que efectivamente lograremos cruzar el campo minado. Es decir, hay un objetivo en algún punto del campo, posiblemente en el otro extremo, y necesitamos un plan para llegar a él o determinar su inexistencia.

Esto último pone una exigencia más sobre la forma de obtener una solución. Cruzar el campo implica considerarlo en su totalidad. No es posible tomar decisiones locales, pues estas podrían llevarnos a soluciones no óptimas o callejones sin salida, en el peor de los casos. Por otro lado, explorar todas las posibles soluciones puede ser demasiado costoso en tiempo y demorar demasiado la misión. Bajo estos conceptos el algoritmo que aporto es de suma utilidad porque permite obtener una solución basada en información global en tiempos razonables.

Afortunadamente no contamos con campos minados en nuestras instalaciones. Pero ya he planteado otra situación de características similares, el fútbol de robots. En este caso no hay construcciones pero si hay un límite no puntual a respetar y obstáculos a evitar, que si bien no harán volar al robot por los aires (en la mayoría de los casos),

seguro nos aportarán una penalidad si son embestidos.

El hecho de que los obstáculos tienen posiciones dinámicas aportan un grado más de complejidad y la necesidad de la mayor eficiencia posible [1]. Es inaceptable demorarse demasiado en obtener una solución que explore cada posibilidad para llegar a una conclusión basada en información global. Por otro lado, como mencionamos más arriba, obtener un plan rápidamente basado sólo en información local puede no ser suficiente.

En virtud de esta situación de compromiso, un diagrama de Voronoi presenta una solución razonable. Una vez obtenido, se puede lograr un plan utilizando cualquiera de los algoritmos disponibles para obtener caminos mínimos. Nuevamente el dinamismo exige un tiempo de resolución de bajo orden de magnitud en tiempo de cómputo, pues cualquier plan se vuelve obsoleto en milisegundos.

Si un plan de acción se vuelve obsoleto en cuestión de milisegundos, ¿qué sentido tiene obtener todo un plan con un origen y un destino como el que provee un diagrama de Voronoi si rara vez vamos a poder recorrerlo en su totalidad debido a lo dinámico? Precisamente por la existencia de un destino y un plan para llegar a él. En esta situación, lo valioso es tener una dirección hacia la cual partir y saber que es parte de una solución que nos lleva a destino.

Si bien el contexto es dinámico también es continuo. En general, es de esperarse que la solución actual será muy similar a la solución que podemos obtener en el próximo instante. De esta manera una sucesión de soluciones nos permitirán acercarnos a nuestro objetivo.

5.2. Condiciones experimentales

En la Figura 5.1 podemos ver como es un posible entorno para el fútbol de robots. Consiste de un tablero de color negro, con las áreas delimitadas en blanco y cuyos laterales hacen de caja y delimitan el campo de juego tanto para los jugadores como para la pelota. Los arcos en donde debe entrar la pelota se encuentra delimitados de la misma manera.

La imagen es tomada desde una cámara dispuesta a una altura de 2 metros y centrada sobre el tablero. La seal de video de la cámara es digitalizada y un módulo de visión en un host, luego de identificar los objetos de interés, extrae sus coordenadas en el campo de juego cada 33ms.

En el mismo host, otro módulo, que define la estrategia del equipo, decide cual va a ser el destino de cada robot e interroga al módulo de navegación dándole las coordenadas del robot a mover, su destino, la dirección para aproximarse al objetivo, y las coordenadas del resto de los robots que se transforman en obstáculos. El módulo de navegación entrega como respuesta las velocidades para las ruedas del robot de modo que el mismo se dirija al destino indicado sorteando los obstáculos. Este diálogo ocurre 30 veces por segundo.

La estrategia para guiar al robot fue desarrollada por el equipo UBASot [SS/03] y está basada en el método de la velocidad de obstáculos propuesto por Fiorini [Fio/95]. En esta solución particular, cuando se detecta un obstáculo entre el robot dirigido y su

[1] [AK/96] se demuestra que $O(n\ log(n))$ es la complejidad mínima posible.

destino se busca un objetivo secundario que permita sortear el obstáculo. Si resultara que, a partir de este destino secundario, existe un nuevo obstáculo que se interpone hacia el destino original, se busca por segunda vez otro destino secundario y se procede sin más consideraciones. Por lo tanto, si el resultado de este proceder fuese enfrentarse a un tercer obstáculo se vuelve a correr, oportunamente, el mismo algoritmo.

El principal inconveniente de esta estrategia reside en que, si al resolver el tercer obstáculo quedamos en una situación similar a la original entramos en deadlock.

Esta estrategia es fruto de un compromiso entre la necesidad de dar una respuesta rápida, o reactiva, haciendo un análisis meramente local; y la búsqueda de una solución global, o deliverativa, mucho más costosa en tiempo que tenga en cuenta el estado de situación general. Evitar un obstáculo y volver a considerar una segunda vez evitar un segundo obstáculo es una solución de compromiso que colapsa cuando nos topamos con tres o mas obstáculos distribuidos apropiadamente.

5.3. Objetivos de los experimentos

La situación de juego en un partido de fútbol de robot es muy compleja. Si dejáramos simplemente correr el juego no tendríamos garantías de llegar a alguna configuración que presente los problemas descriptos en la sección anterior, y además serían muy difíciles de reproducir sistemáticamente.

Dado que es relativamente fácil distribuir los robots de manera de forzar al módulo de navegación a entrar en deadlock, decidí considerar situaciones estáticas representativas a modo de prueba de concepto. Esto es, casos donde un robot dado debe llegar hasta la pelota ubicada en algún punto del campo de juego, aproximándose con la dirección adecuada de manera tal que la pelota salga despedida hacia el arco mientras que en su camino aparecen otros robots dispuestos de manera tal que el módulo de navegación falle.

Así, en este ambiente controlado, podré comparar la estrategia actual con el algoritmo que propongo proveyendo al robot de un plan de acción para llegar desde donde esté hasta la pelota y comparar sus desempeos.

Para poner en funcionamiento el algoritmo desarrollado partí de la aplicación en uso para controlar los robots insertando la posibilidad de usar diagramas de Voronoi en el código que calcula la trayectoria de un robot hacia un destino dado.

La inserción se realizó mediante unas pocas líneas que comunican, vía un socket, la información necesaria a la aplicación que desarrollé basada en el algoritmo que propongo.

Mi aplicación, que corre sobre el mismo host, recibe las coordenadas de los robots y la pelota, y retorna una coordenada de destino intermedia más un ángulo deseado de arribo. Los datos de entrada son acondicionados por la interfaz y pasado al algoritmo que construye el diagrama de Voronoi. Con la información del diagrama, el algoritmo de cálculo de caminos mínimos obtiene el camino hacia el destino indicado y provee, mediante la interfaz de salida una coordenada de destino intermedio, *el milestone*, junto con una dirección de llegada a este.

En términos de Voronoi, el robot siempre se va a mover siendo el centro de su región

de Voronoi. El milestone es el punto en el que el camino, partiendo desde el robot, toca al primero de los vértices de Voronoi y su orientación, o dirección, es la pendiente de ese primer tacel. Así me aseguro que el robot se va alineando en la dirección del próximo tacel, aunque nunca llegue a tocar ninguno de ellos.

De esta manera proveo siempre un destino libre de obstáculos desactivando esa responsabilidad del módulo de navegación (basado en Fiorini) y encolumnando el derrotero del robot sobre los taceles del diagrama de Voronoi. Lo que no se desactiva del módulo de navegación es la responsabilidad de alinear al robot con la dirección de arribo al objetivo buscada.

Cuando las regiones de Voronoi del robot y de la pelota son adyacentes y sus coordenadas son visibles entre sí a través del tacel que sapara las regiones la interfaz deja de intervenir y ofrece tanto el destino y dirección original como respuesta.

Lo que sigue son 6 escenarios que logré preparar para contrastar el comportamiento actual de la navegación con el nuevo comportamiento obtenido gracias al modificación que introduje usando los diagramas de Voronoi. Hay situaciones en la que el robot no logra pasar, en la que colisiona contra los laterales y/o en la que embiste a los demás jugadores. En otro de los escenarios elegidos el robot logra pasar hasta su destino pero observaremos las consecuencias de una solución demasiado localizada.

5.4. Resultados

5.4.1. Escenario A

Veamos un primer ejemplo, en la Figura 5.1 observamos una imagen tomada de la cámara utilizada para sensar a los robots. Se encuentra distinguido con un círculo en el extremo superior izquierdo el robot que se desea controlar para que se mueva hasta el extremo inferior derecho donde está la pelota. Los robots *amigos* son aquellos que poseen algún cuadrado color azul claro. Los *oponentes* son quienes lucen algún cuadrado amarillo.

En la Figura 5.2 observamos una esquematización de la Figura 5.1 en donde los robots *amigos* se representan con orientación y a los *oponentes* con círculos sin orientación conocida. Por último, la pelota se observa como un círculo de un radio menor.

En la Figura 5.3 observamos la trayectoria del robot utilizando el algoritmo original de la aplicación del equipo UBASot. Vemos que intenta lograr un claro hacia la pelota desplazándose sucesivamente hacia su derecha hasta encontrarse contra el borde del tablero.

En la Figura 5.4 observamos el diagrama de Voronoi generado dada la posición de los robots y la pelota en el campo de juego (el centro de cada uno de estos objetos representa un generados en el diagrama). En trazo grueso se observa el camino mínimo para llegar a la pelota. La flecha indica el primer milestone de las piernas del camino.

En la medida en que el robot avance, tanto el diagrama como el milestone van a ir variando de manera continua. En la Figura 5.5 observamos como evolucionan la posición y dirección deseadas en el milestone.

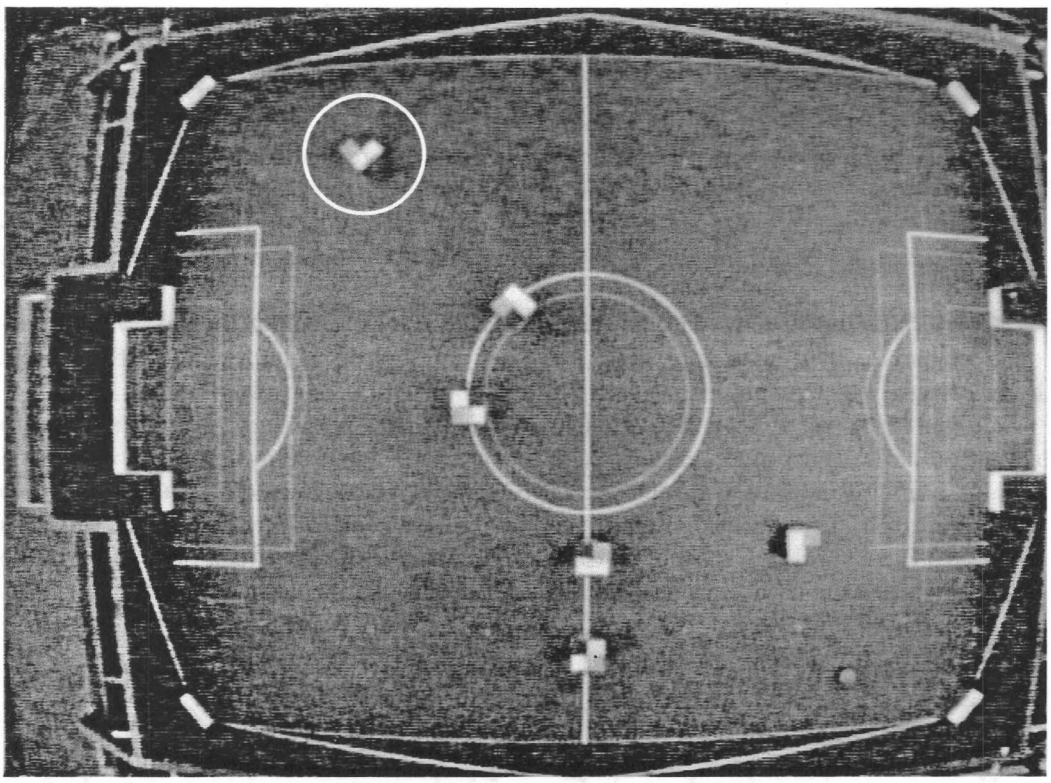


Figura 5.1: Escenario A

La Figura 5.6 da un detalle de como se va modificando la región de Voronoi del robot a controlar de acuerdo se acerca a la pelota. En las últimas dos instantáneas de la secuencia el milestone se ubica directamente sobre la pelota en coincidencia con el *salto* que se observa en la secuencia de milestones de la Figura 5.5.

Este *salto* es en donde se deja correr a la aplicación sin necesidad de usar el diagrama de Voronoi para establecer un nuevo milestone. Esto se debe a que las regiones de Voronoi del robot y la pelota son adyacentes y, simultáneamente, su línea visual pasa a través del tacel compartido. La interfaz deja de intervenir y el algoritmo original trabaja con la dirección y coordenada de destino original.

Finalmente, en la Figura 5.7 observamos la trayectoria del robot y cómo en el último tramo da un rodeo para llegar a la pelota respetando la dirección provista por la estrategia original consecuencia de haber tomado nuevamente el control.

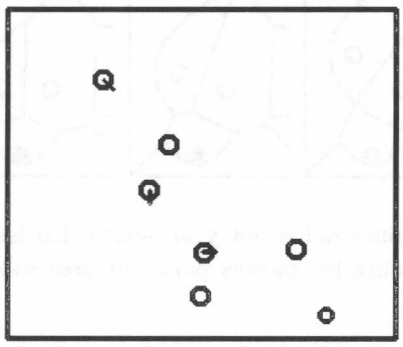


Figura 5.2: Esquema del escenario A.

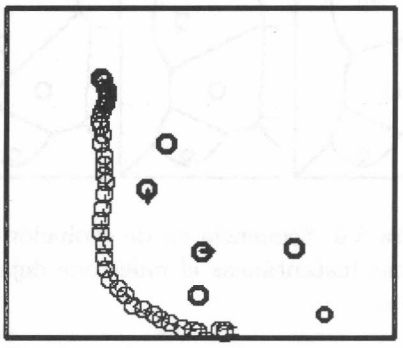


Figura 5.3: Recorrido original sin usar diagramas de Voronoi.

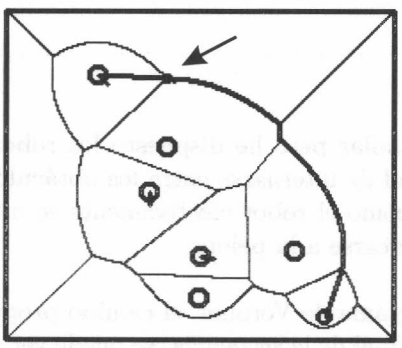


Figura 5.4: Diagrama de Voronoi sobre el escenario A. Camino a la pelota y milestone.

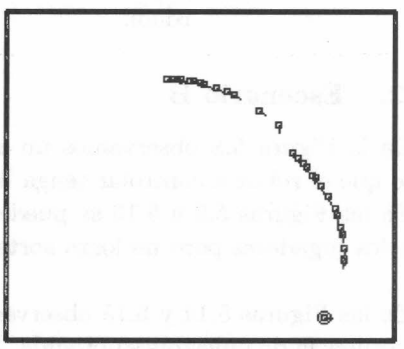


Figura 5.5: Evolución del milestone.

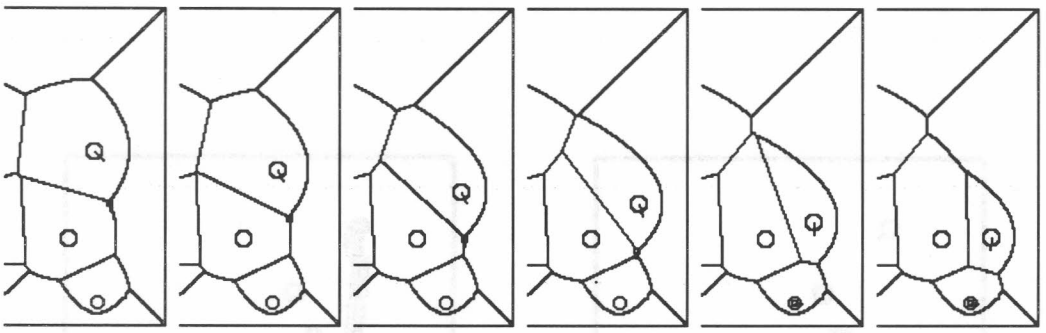


Figura 5.6: Secuencia en de evolución de las regiones del robot y la pelota. En las dos últimas instantáneas el milestone deja de estar sobre los taceles para ubicarse sobre la pelota.

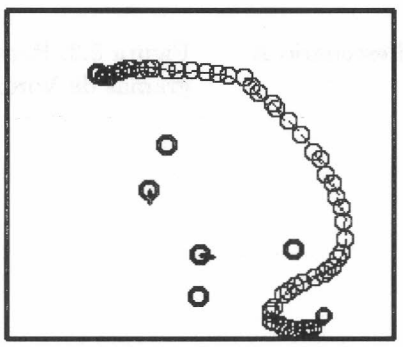


Figura 5.7: Derrotero del robot segun Voronoi.

5.4.2. Escenario B

En la Figura 5.8 observamos un escenario similar pero he dispuesto los robots de modo que el robot a controlar tenga la posibilidad de internarse entre los obtáculos.

En las Figuras 5.9 a 5.13 se puede observar como el robot efectivamente se interna entre los jugadores pero no logra sortearlos y acercarse a la pelota.

En las Figuras 5.14 y 5.15 observamos el diagrama de Voronoi, el camino propuesto y la secuencia de milestones obtenida. En el paso final de la secuencia, se puede ver como el algoritmo de navegación original se independiza del uso del diagrama de Voronoi para llegar al objetivo.

La Figura 5.16 muestra la trayectoria completa del robot que logra sortear los obstáculos y acceder a la pelota con la dirección deseada.

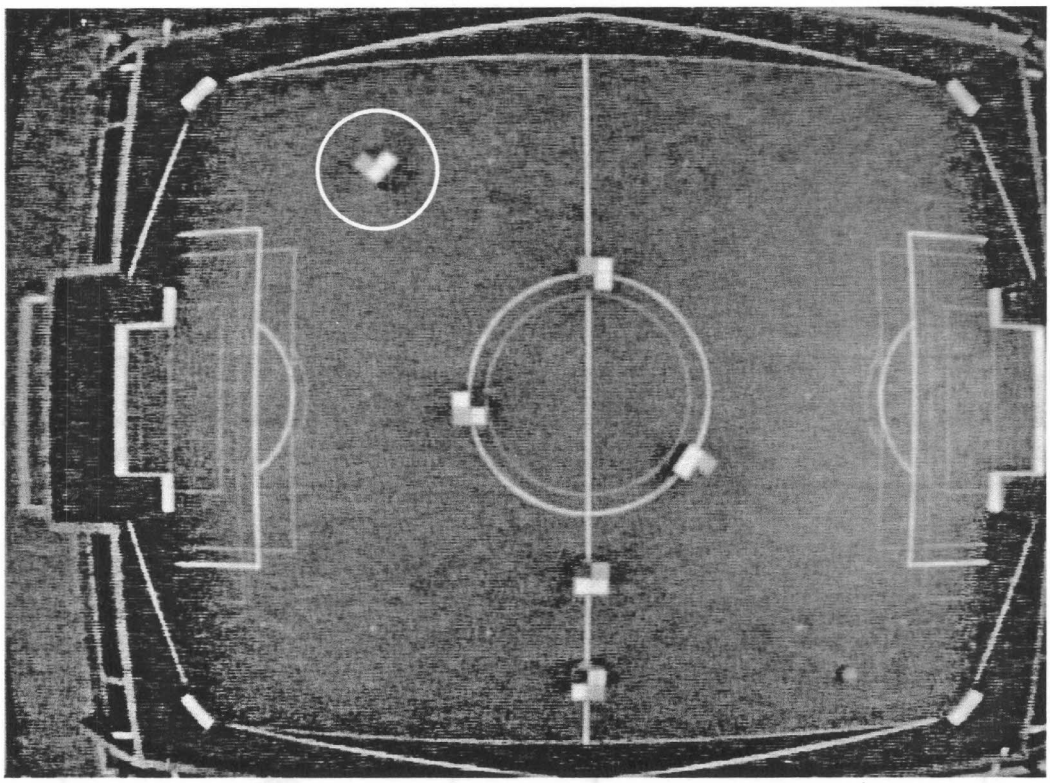


Figura 5.8: Escenario B

5.4.3. Escenario C

En el ejemplo de la Figura 5.18 observamos cómo el robot, guiado por el algoritmo original, avanza trabajosamente entre los obstáculos dispuestos según se observa en el escenario de la Figura 5.17. En su trayectoria,el robot controlado se aproxima demasiado a los obstáculos en lugar de mantener distancia y termina impactando con el lateral inferior.

No menos trabajoso es el avance observado en la Figura 5.20 guiado por el diagrama de Voronoi aunque, sin embargo, se ve que se respetan las distancias hacia los obstáculos.

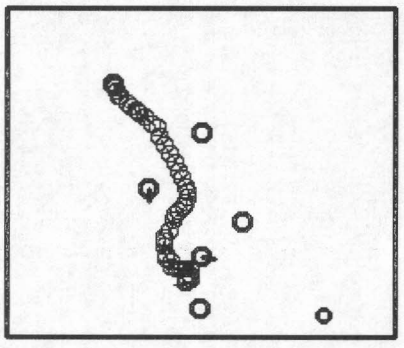


Figura 5.9: El robot avanza entre los obstáculos. Intenta sortear a un compañero prácticamente rozándolo. Gira hacia abajo.

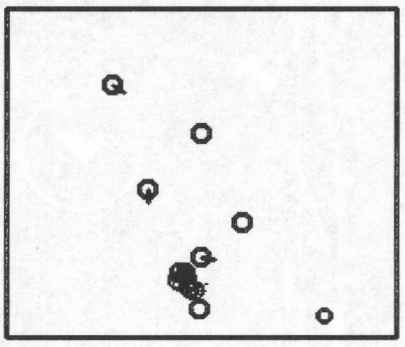


Figura 5.10: Gira sobre sí mismo, vuelve a sortear a su compañero y desciende.

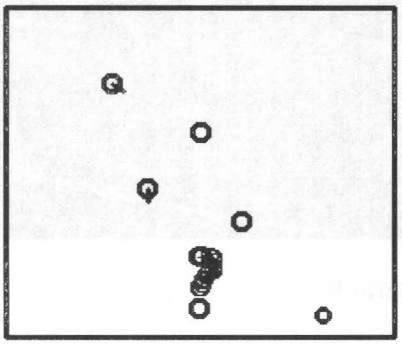


Figura 5.11: Sube y finalmente embiste a su compañero en sentido opuesto a la pelota.

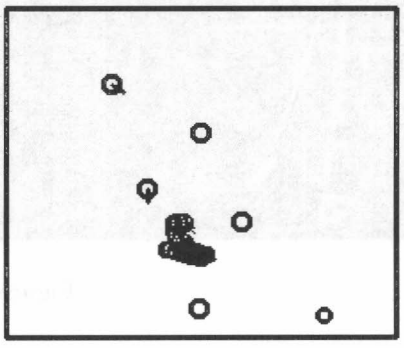


Figura 5.12: Empuja a su compañero hacia atrás, sube y se vuelve hacia la pelota.

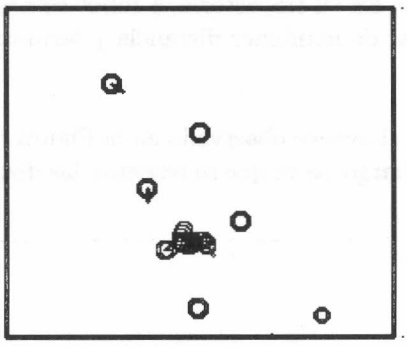


Figura 5.13: Pero sigue girando sobre sí mismo hasta volver a embestir a su compañero. Luego avanza (posiblemente en busca del referí).

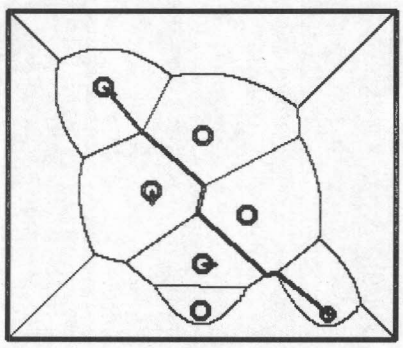


Figura 5.14: Propuesta de Voronoi para el Escenario B.

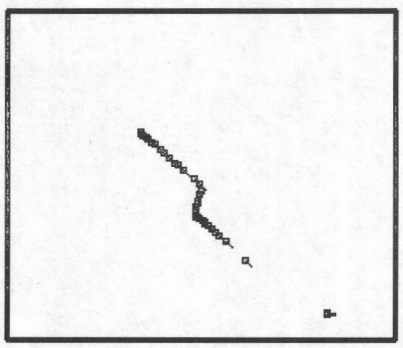


Figura 5.15: Evolución del Milestone.

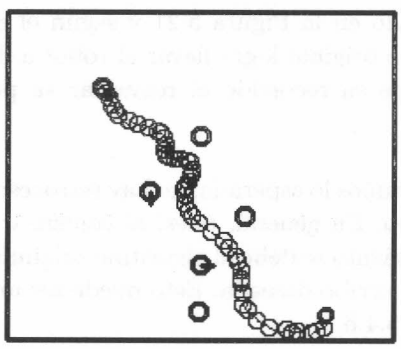


Figura 5.16: Derrotero del robot guiado por Voronoi.

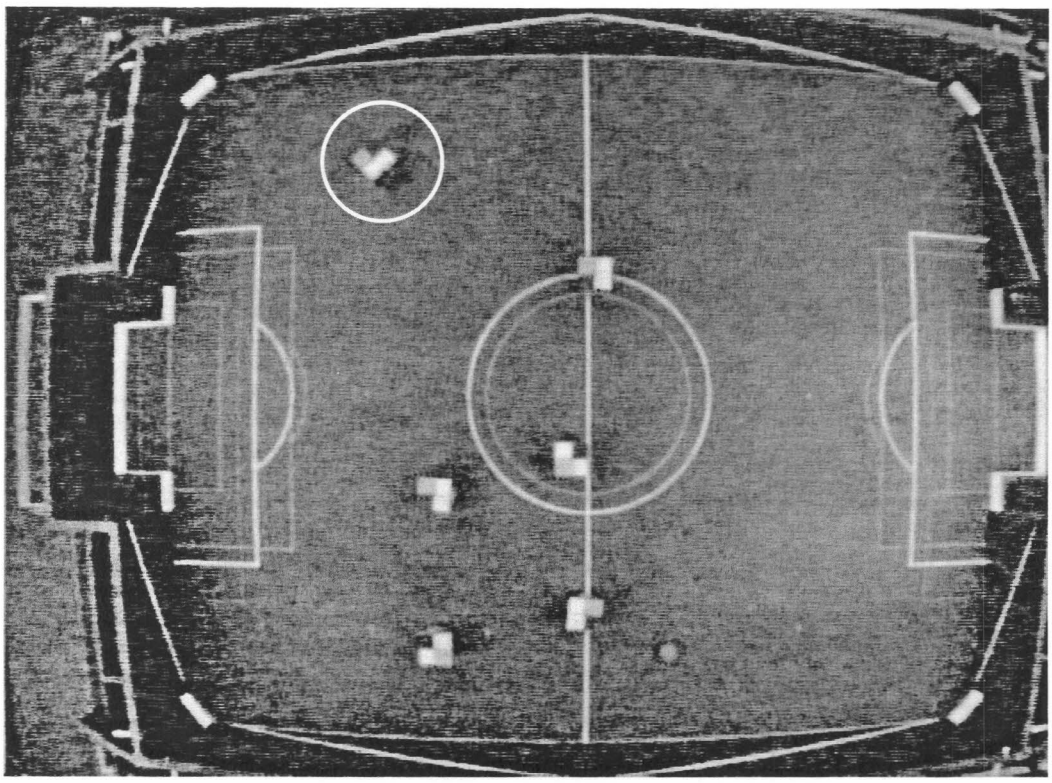


Figura 5.17: Escenario C

5.4.4. Escenario D

En el escenario planteado en la Figura 5.21 y según el esquema de la Figura 5.22, observamos que el algoritmo original logra llevar al robot a destino. Sin embargo vemos que debe remontar parte de su recorrido al reevaluar su posición luego de sortear el primer obstáculo.

En la Figura 5.24 observamos lo esperado, no hay retrocesos y el derrotero, en general, es correcto hacia el objetivo. En general, pues, al comenzar vemos que se ejecuta una pequeña S. Este comportamiento se debe al algoritmo original en su intención de orientar al robot con la dirección de arribo deseada. Esto puede ser un problema y lo analizo con un poco mas de detalle en 5.4.6.

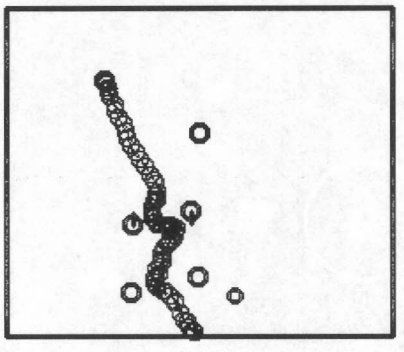


Figura 5.18: El robot avanza con dificultad entre entre los obstáculos pero finalmente impacta contra el lateral inferior.

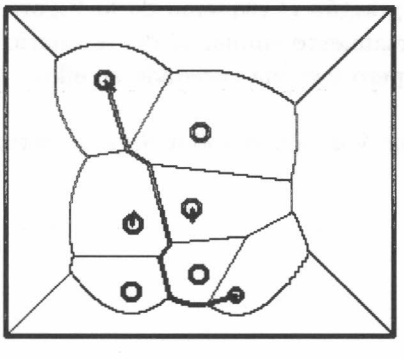


Figura 5.19: Solución Voronoi para el Escenario C.

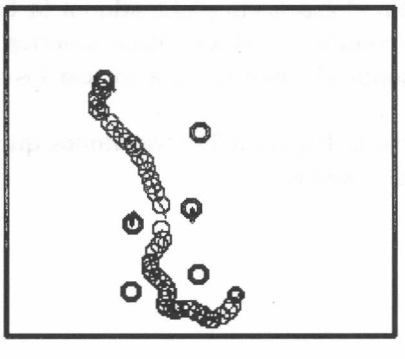


Figura 5.20: Derrotero del robot guiado por Voronoi.

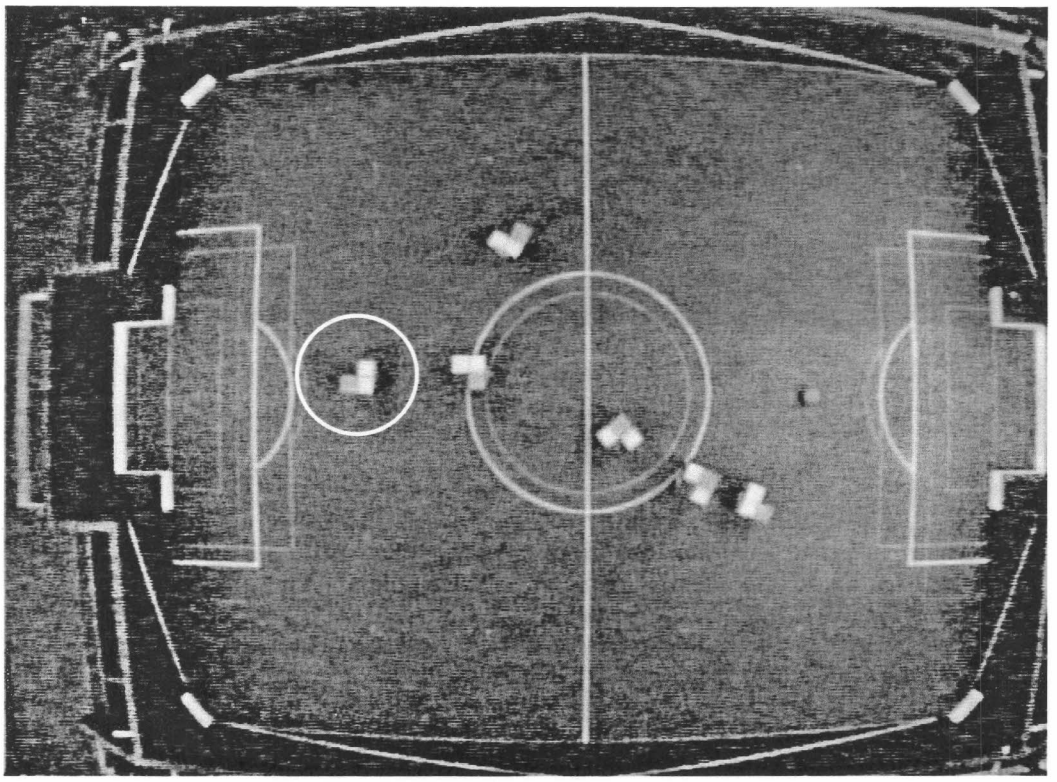


Figura 5.21: Escenario D

5.4.5. Escenario E

En el escenario planteado en la Figura 5.25 y según el esquema de la Figura 5.26, observamos que el algoritmo muestra un comportamiento similar al del escenario 5.4.2 en donde el robot intenta sortear los obstáculos pero termina enredado en ellos.

En la Figura 5.28 observamos que, guiado por Voronoi, el robot llega a destino sin inconvenientes.

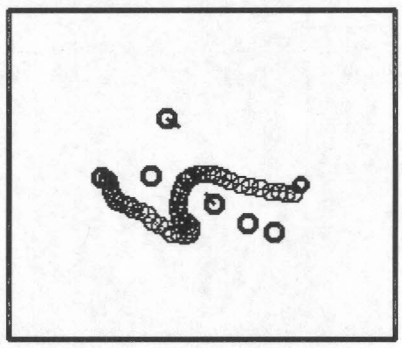


Figura 5.22: El robot avanza y retrocede para llegar a destino.

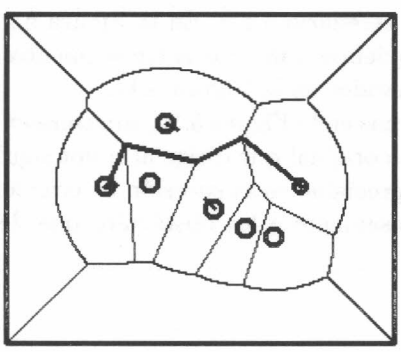


Figura 5.23: Solución Voronoi para el Escenario D.

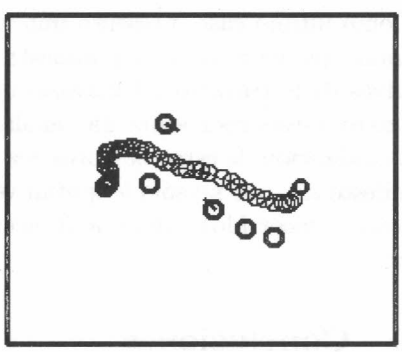


Figura 5.24: Derrotero del robot guiado por Voronoi.

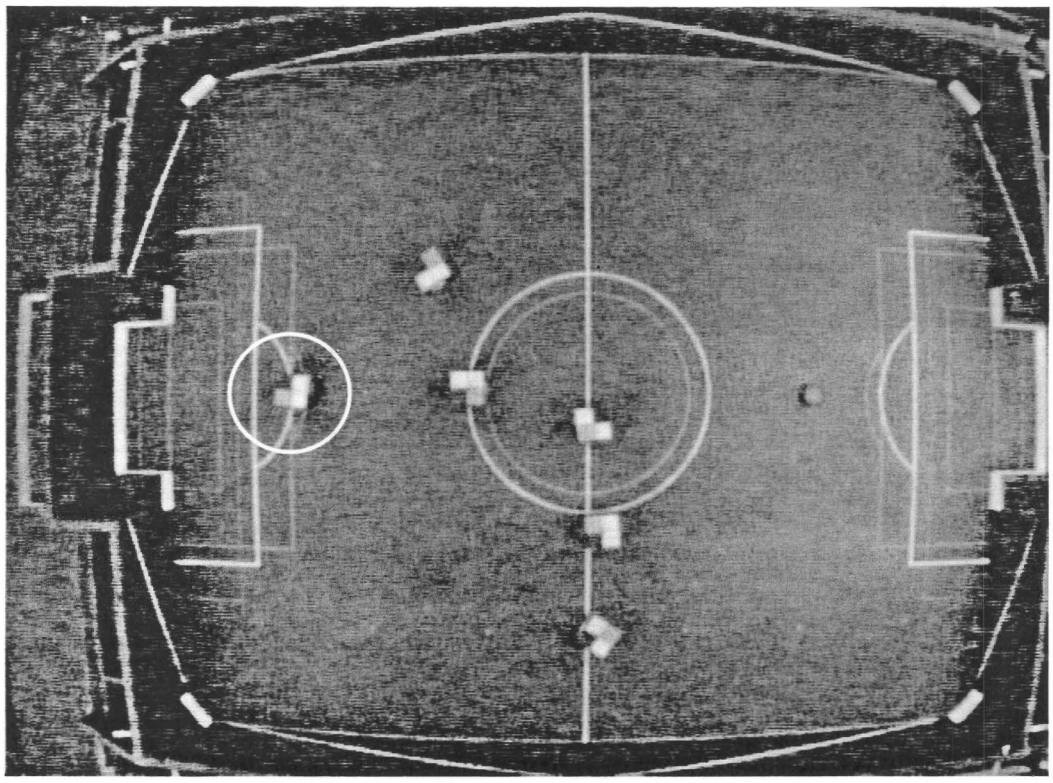


Figura 5.25: Escenario E

5.4.6. Escenario F

Como último caso, presento una variante del escenario 5.4.2. En la Figura 5.29 observamos que en el camino propuesto aparecen quiebres (ángulos entre segmentos consecutivos de la trayectoria) mayores que los observados en la Figura 5.14.

Los excesivos rodeos que da, señalados con flechas en la Figura 5.30, son consecuencia de la conjunción de estos quiebres con el algoritmo original que dirige al robot siguiendo los milestones, que no sólo le quitan velocidad (apreciable en la sucesión de círculos que grafican el recorrido), sino que lo acercan peligrosamente a los obstáculos que debiera evitar.

5.5. Conclusiones

A lo largo de este capítulo documento una prueba de concepto al utilizar diagramas de Voronoi para dirigir a un robot a través de un terreno con obstáculos y lo comparo con un algoritmo de navegación que no usa la modificación propuesta en este trabajo.

Para esto, se evaluaron una serie de casos particulares (escenarios 5.4.1 a 5.4.6) donde el algortimo de navegación original no cumple satisfactoriamente su misión. En los escenarios 5.4.1 a 5.4.5 se puede observar como, la modificación introducida valiéndose

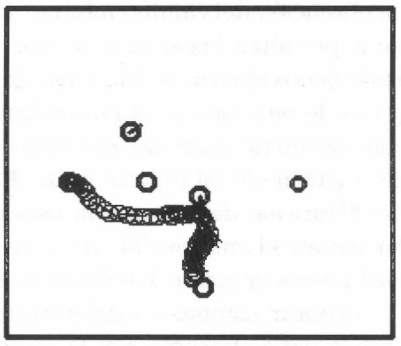


Figura 5.26: El robot intenta sortear los obstáculos pero, al contrario, termina avalanzándose sobre ellos.

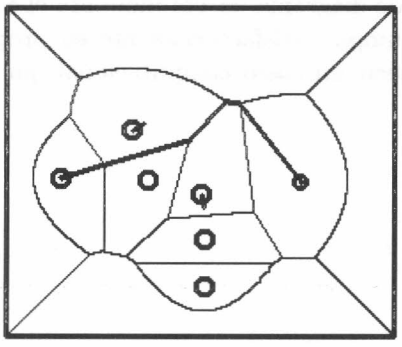


Figura 5.27: Solución Voronoi para el Escenario E.

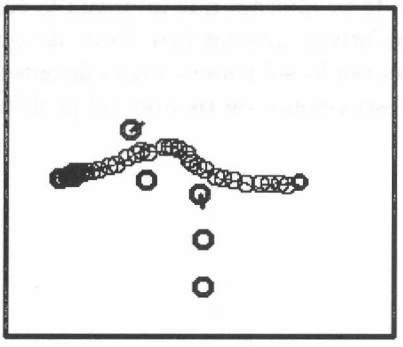


Figura 5.28: Derrotero del robot guiado por Voronoi.

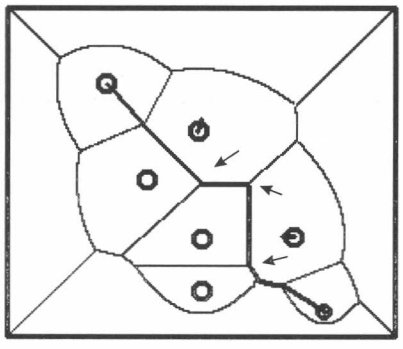


Figura 5.29: Solución Voronoi para el Escenario F. Las flechas indican los quiebres importantes en el camino propuesto.

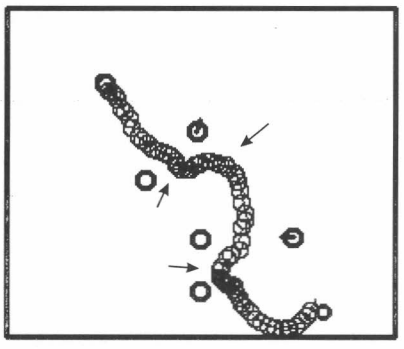


Figura 5.30: Derrotero del robot guiado por Voronoi. Las flechas indican donde el recorrido sufre un rodeo para alinearse con la inclinación esperada del siguiente segmento del camino.

del diagrama de Voronoi y la obtención del camino mínimo, le permite al robot encontrar trayectorias alternativas que le permiten llegar al objetivo, cuando en la original no era posible o se realizaba en forma menos eficientes. Dado que la complejidad computacional de la modificación propuesta es de bajo orden, la intercalación del cálculo del diagrama de Voronoi es compatible con las restricciones del problema con la ventaja fundamental que permite administrar información global para la toma de decisiones locales mientras que en la versión original del algoritmo de navegación esto no era posible.

Un comentario particular merece el análisis del último escenario (5.4.6). En esa experiencia se puede observar que la mera intercalación del método propuesto no es suficiente sino que se hace necesario incorporar algunas consideraciones extra en el último tramo de la trayectoria donde el robot debe alinearse con la dirección deseada utilizando solamente el algoritmo de navegación original (dado que ya no tiene sentido el uso del diagrama de Voronoi).

Como conclusión general de estas experiencias podemos notar que efectivamente el método propuesto, que intercala la construcción de diagrama de Voronoi para el cálculo de objetivos intermedios libres de obstáculos, cumple satisfactoriamente su propósito presentando soluciones mas eficientes sin introducir un costo computacional que viole las restricciones de tiempo del problema.

Capítulo 6

Conclusiones y Futuros Trabajos

6.1. Conclusiones

Al comienzo del presente trabajo analizamos las virtudes de los diagramas de Voronoi cuando son considerados para la navegación de robots. En ese mismo contexto comentamos que con frecuencia, un escenario real contiene objetos que no pueden ser considerados puntuales, y en este sentido también observamos que no es trivial extender los diagramas de Voronoi para que los incluyan. Las soluciones existentes que atacan el problema alcanzan complejidades temporales que las hacen inaplicables en tiempo real.

En particular, este trabajo se gestó ante la necesidad de un algoritmo rápido para la navegación de robots para el problema de fútbol de robots.

Del estudio de los algoritmos existentes para obtener un diagrama de Voronoi y sus complejidades analizadas en los capítulos 2 y 3, centré mi atención en el algoritmo de Fortune y en la metáfora espacial que acepta como una de sus interpretaciones.

Basado en esta metáfora, a lo largo del Capítulo 4 me permití extender el dominio del algoritmo de Fortune para que considere tanto puntos como segmentos, respetando la intención original del autor y las propiedades de uso de espacio y tiempo, como quedó demostrado.

Luego brindé una implementación que acepta cuatro segmentos que conforman el perímetro de un rectángulo y obstáculos puntuales en su interior. Esta implementación fue desarrollada en Squeak, un flavor de Smalltalk, por los motivos que expreso en el Apéndice A. Junto con el algoritmo he desarrollado distintas herramientas gráficas que me asistieron al momento de entender y modelar el problema, testearlo, extraer imágenes para este documento e, incluso, generar código LaTeX para documentarlo e incluirlo en este trabajo. La interfaz del módulo de visualización y testeo se documenta en detalle en el Apéndice A.

Todo el proyecto está disponible en el Swiki accesible desde www.squeak.org.

En el Capítulo 5, a modo de prueba de concepto, documento un trabajo de campo en el que la aplicación desarrollada para este trabajo guía efectivamente a un robot através de los obstáculos en una cancha de fútbol de robots.

Para esto implementé una interfaz que se conecta con mi aplicación y le permite ser interrogada por el software de aplicación desarrollado por UBASot para controlar su equipo de robots en un partido. Junto con esta interfaz implementé un simulador que permite tomar los logs de cada experiencia y reproducir los pasos dados por los robots o recrear un partido completo, prescindiendo de la aplicación original, a efectos de estudio y análisis de los resultados.

En el Capítulo 5 se reproducen varios escenarios en el que queda expuesta la eficiencia del algoritmo que propongo para asistir en la navegación de un robot en contraste con el uso de solo el módulo de navegación de UBASot. En estos escenarios vemos desde situaciones que el algoritmo original no puede sortear a situaciones en la que el recorrido obtenido dista mucho de ser óptimo, pero que el algoritmo que propongo resuelve adecuadamente.

Finalmente expongo algunas consideraciones sobre futuros trabajos que podrían guiar la continuación de la actividad realizada hasta ahora y que he documentado en este manuscrito.

6.2. Futuros Trabajos

6.2.1. Robustez computacional

Tanto en los algoritmos analizados como en el propuesto se realiza la computación de valores numéricos por dos razones principales: para tomar decisiones (test) o para construir valores [BP/97, LPT/96].

Los segundos constituyen la respuesta del algoritmo. Los problemas de precisión son perfectamente tolerables en el contexto del problema estudiado. Mientras que los primeros influyen en la lógica misma del algoritmo y por lo tanto, los errores de cálculo pueden provocar la completa invalidez del resultado final.

Un trabajo posible, para extender el uso del método propuesto, es medir y acotar los posibles errores de precisión en la toma de decisiones. En el presente trabajo basta con disponer un grupo no muy importante de sitios concéntricos para empezar a tener problemas de este tipo.

6.2.2. Extención a segmentos y polígonos

El algoritmo presentado hace un fuerte uso de que los segmentos sean isotéticos[1] y periféricos. Sin embargo la teoría geométrica que lo sustenta no depende exclusivamente de ello.

De este modo, una extensión natural del método propuesto es poder poner a consideración del algoritmo no sólo un perímetro rectangular sino segmentos y/o polígonos en general.

[1] traducción libre del inglés isothetic, cuando los segmentos siempre se encuentran o a 0° ó a 90°

6.2.3. Traducción a lenguajes de bajo nivel

Por los motivos que se expresan en el Apéndice A he trabajado en Smalltalk. Habiendo terminado la etapa exploratoria del problema y teniendo ya un modelo sólido del algoritmo y los tipos de datos involucrados, resulta pertinente su traducción a C.

La aplicación utilizada por el equipo UBASot de esta facultad para controlar los robots durante el partido está escrito en C por los requerimientos temporales y de hardware extremos propios de esta temática.

Para lograr una comunicación entre mi aplicación y la del equipo fue necesario realizarla vía sockets, pues no fue posible transformar de manera trivial código Smalltalk a algo que pudiera comunicarse con comodidad a C.

Algo de suma utilidad para la aplicación de fútbol de robots sería contar con mi desarrollo en una dll. Esto es, el algoritmo de obtención del diagrama de Voronoi más la obtención del camino mínimo entre origen y destino.

6.2.4. Distancias Sesgadas

El algoritmo de Fortune no considera que los obstáculos estén en movimiento. Sin embargo, en general, la información que describe la actividad obstáculos (otros robots) está disponible. La respuesta del módulo de navegación puede variar significativamente si se considera, por ejemplo, pasar por detrás y no por delante de un obstáculo que podría embestir al robot controlado.

En el trabajo de Aichholzer et al. [AAC+/96] se consideran diagramas de Voronoi construidos con métricas sesgadas en donde la influencia de un obstáculo no es pareja en todos los sentidos.

Un tema interesante a ser estudiado es la posibilidad de introducir métricas alternativas y distintas de acuerdo al obstáculo (i.e. considerando la dirección y magnitud de su vector velocidad) considerado realizando las extensiones que correspondan preservando complejidad temporal y espacial.

Índice de figuras

Apéndice A

Implementación

A.1. Introducción

El diagrama de Voronoi es un diagrama en donde se destacan las propiedades geométricas de sus segmentos componentes en relación a los atractores que los generan. Nuevamente, la geometría, tanto en dos como en tres dimensiones, es la metáfora que sustenta el desarrollo del algoritmo de Fortune.

Cada línea que se traza durante el desarrollo del diagrama representa una relación geométrica entre otros componentes, que a su vez son la expresión de otras relaciones. En muchos casos, estas líneas tambíen son dinámicas. Es difícil mantener en la mente o en el papel tantas propiedades relacionadas con tanta geometría. Es mucho mas cómodo jugar como si fueran ladrillos de *Lego* pudiendo combinar siempre de una manera nueva e imprevista hasta ese momento. Tal como cuando jugamos con esos ladrillos es bueno poder ver y manipular al instante.

Pero más aún, tratándose de software podemos apuntar más alto, aspirar a más. Como si fueran ladrillos, potencia la exploración y descubrimeinto poder alterar el comportamiento de los elementos computacionales con los que estamos trabajando, cambiar estructuras, estratégias, versiones, etc. sin que esto afecte los casos concretos que se estan evaluando ni su continuidad. Tener estos ladrillos permiten descubrir propiedades o confirmar las intimamente intuídas, poder ver una y otra vez, hacia adelante, hacia atrás.

En definitiva, y llevando las expectativas al máximo, con estos ladrillos se podría jugar con lo que se tiene entre manos, para que fuera arcilla que se comportara como un modelo vivo que pudiera ser reificado sin que implique volver a empezar en cada intento. Ladrillos que minimicen los tiempos entre teorizar, implementar y palpar los resultados. Finalmente, también es deseable un entorno donde estos ladrillos convivan, algo que sirva de marco, de mesa de trabajo. Algo que permita el 'Gestalt' de todos esos ladrillos individuales para convertir el todo en más que la suma de sus partes.

En pocas palabras, personalmente espero para trabajar en un entorno que sea una herramienta que potencie la exploración como forma de descubrimiento.

Vale destacar que también es deseable que el lenguaje elejido, sus características y

el entorno de trabajo no sean un obstáculo que complique inecesariamente la tarea de exploración distrayendo la atención a cuestiones de Tipos, interfaces, manejo de memoria, compiladores, linqueo, etc. Toda la energia debe poder enfocarce en la exploración del modelo y no del paradigma, el lenguaje, la ide y/o el sistema operativo.

Construir una herramiemta con estas propiedades sería material para otra tesis pero afortunadamente ese trabajo ya fue realizado durante los 70's. Smalltalk es un entorno de desarrollo y una herramienta que maximiza todas las propiedades mencionadas.

De los distintos sabores de Smalltalk actuales, he decido por utilizar Squeak por que presenta una serie de ventajas gracias a su activa comunidad de usuarios (entre ellos quienes inventaron la Tecnología de Objetos y la primera de sus implementaciones, Smalltalk-80), sus herramientas gráficas y su riquísimo entorno.

Por los párrafos que preceden a estas líneas es claro que priorizo en la elección del entorno de trabajo y el lenguaje a utilizar la facilidad para explorar el problema, atacarlo y resolverlo. No obstante, una vez resuelto, el algoritmo resultante puede ser re-escrito en cualquier otro lenguaje, pues ya no hay nada que descubrir solo hay que lidiar con el lenguaje elegido y con las complejidades inherentes del mismo.

A.2. La aplicación

Sigue a continuación una descripción de la aplicación desarrollada en Squeak. No es una aplicación tradicional contenida en un ejecutable, sino mas bien una aplicación desarrollada dentro de un entorno y así es como se expone. Si bien se puede escindir de Squeak todas sus herramientas y reducir todo el entorno a lo indispensable para soportar el trabajo que se presenta, prefiero ofrecerlo completo y brindar a quien se anime todas las posibilidades que Smalltalk brinda para explorar y palpar lo realizado.

La aplicacón no sólo es la implementación del algoritmo sino la suma de todas las herramientas desarrolladas para descubrirlo, pulirlo y presentarlo. El resultado permite al lector explorar y comprobar visualmente todo lo descripto en este trabajo de la misma manera en que el autor lo ha realizado.

Para empezar debemos arrancar el entorno Squeak. Esto se logra ejecutando `Squeak.exe`. Para comenzar a trabajar busquemos la solapa derecha denominada `Tools`. Extendámosla arrástrandola con el mouse hacia dentro, busquemos el icono de un `Workspace` y arrastrémoslo hacia adentro fuera de la solapa. Dentro del workspace escribimos `VoronoiCommander new openInHand` y lo evaluamos tipeando `ALT+D` o con ayuda del mouse como se ve en la Figura A.1. Luego de un par de segundos, dependiendo de la cantidad de casos de test que haya definidos observamos la consola que se ve en la Figura A.2.

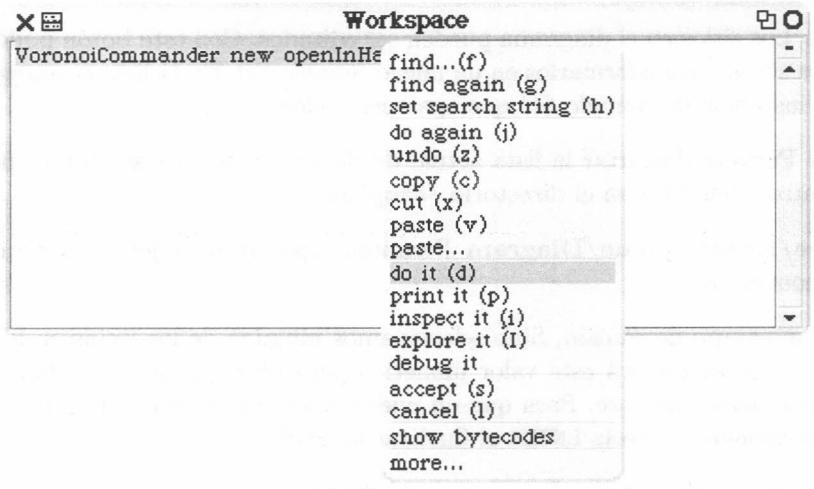


Figura A.1: En un workspace evaluamos...

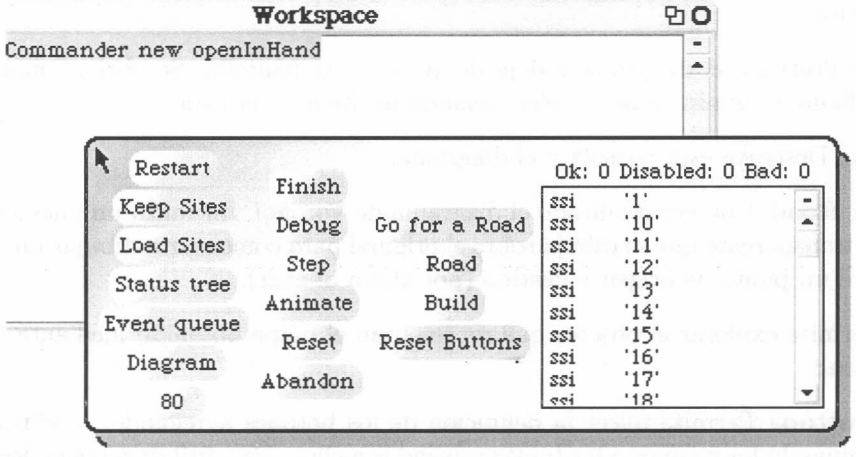


Figura A.2: Consola de comando.

La consola aparece sujeta al mouse y debemos elejir un lugar para clickear y depositarla. Observamos tres hileras de botones y una lista. Veamos uno por uno...

Restart Reinicia el diagrama. Si hay un *Sample Set* seleccionado en la lista se abre un submenú con las opciones `restart`, `reversed`, `inversed` e `inversed & reversed` que permiten espejar y/oo reflejar el conjunto de sitios selecionado. Sino hay un *Sample Set* seleccionado se tomará el valor numérico del ultimo botón para obtener

esa cantidad de sitios ubicados aleatoriamente.

Keep Sites Los sitios en el diagrama pueden ser editados. Con este botón podemos tomar los sitios y transformarlos en un nuevo *Sample Set*. En la lista se agregará una nueva instancia de *Sample Set* que aparecerá seleccionada.

Load Sites Permite descartar la lista actual de *Sample Sets* y cargar todos los que se encuentren definidos en el directorio `/samples`.

Status Tree/Event Queue/Diagram Permiten explorar los objetos que representan estos conceptos.

80 Esto es un campo de edición, Sino selecionamos ninguno de los items en la lista de *Sample Sets* se tomará este valor numérico para obtener esta cantidad de sitios ubicados aleatoriamente. Para que un nuevo valor sea considerado por el sistema debe presionarse la tecla `ENTER` al finalizar la edición.

Finish Permite consumir todos los eventos del diagrama.

Debug Permite consumir el próximo evento abriendo un debugger del entorno.

Step Permite consumir el próximo evento.

Animate/Stop Animation Permite animar, o detener la animación, del consumo de eventos

Reset Se destruye el diagrama y deja de verse en la pantalla. Se obtiene uno nuevo mediante el botón `Reset` o seleccionando un ítem en la lista.

Abandon Destruye esta consola y el diagrama.

Go for a Road Una vez finalizado el diagrama de Voronoi. Inicializa un nuevo diagrama transparente que se dibuja sobre el original para comenzar a dibujar un camino entre un punto de origen y destino (por ahora al azar).

Road Permite explorar el objetos que representan el mapa obtenido mediante `Go for a Road`.

Reset Buttons Permite releer la definición de los botones agregando, modificando o eliminando los mismos o las tareas asociadas a ellos. Muy útil durante el desarrollo y exploración.

Para cada ítem en la lista de *Sample Set Item* podemos obtener, mediante el botón derecho del mouse, las opciones que se ven en la Figura A.3.

update Habiendo finalizado un diagrama, indica al ítem que tome la actual configuración de taceles como patrón.

test Toma los sitios del ítem y obtiene su diagrama, luego compara los taceles obtenidos con el patrón que tiene definido (de tenerlo).

save Salva el ítem en disco.

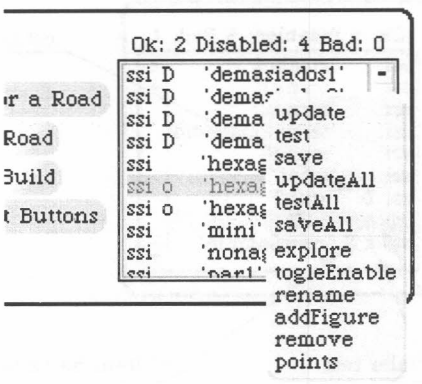


Figura A.3: Opciones para los items en la lista.

updateAll/testAll/saveAll Realiza un la operación con todos los items en la lista.

testAll Realiza un `Update` con todos los items en la lista.

explore Permite explorar el objeto qeu representa al ítem.

togleEnable Habilita o deshabilita el ítem. Evaluar `Test` en un ítem deshabilitado no tiene ningún efecto.

rename Permite cambiar el nomre del ítem.

addFigure Permite asociar una imagen al ítem seleccionado (por ejemplo una toma de la cámara en un partido).

remove Elimina el ítem de la lista (no del disco).

points Permite explorar una objeto colección con las coordenadas de los sitios que conforman el ítem.

La descripción que vemos para cada ítem es la misma que vemos en un debugger del entorno y por convención indica la clase de la instancia (en este caso su acronismo, `SSI`) más información pertinente que nos permita distinguir una instancia de otra.

En la Figura A.4 observamos que un ítem presenta la letra `D` indicando que está deshabilitado. También que el ítem denominado 'uno3' ha pasado su último test correctamente y que los taceles que tiene definido en su patrón coinciden con los taceles que se obtienen de realizar su diagrama. No es el caso del ítem denominado 'newSample' que indica con la letra `S` que no ha sido salvado aún o que ha cambiado su patrón asociado; y que la última comparación de los taceles obtenidos de su diagrama no coincidió con su patrón (de tenerlo).

También podemos observar sobre el cuadro de la lista la leyenda `'Ok: 4 Disabled: 5 Bad: 1'` que indica que en las evaluaciones de test hasta el momento hay 4 resultados exitosos, 5 items deshabilitados que no se consideran, y un item cuyo

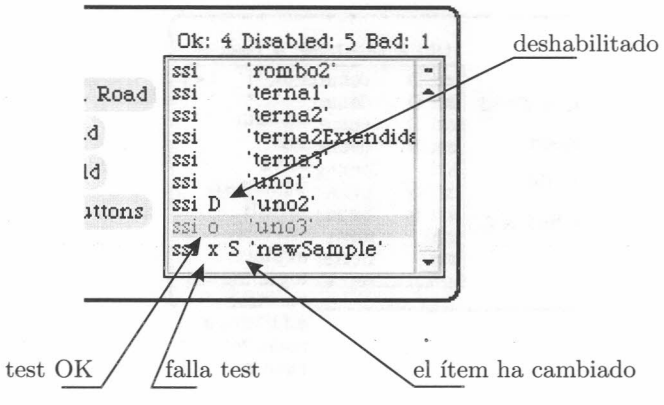


Figura A.4: Detalle de los ítems.

patrón no coincide con el diagrama que se optiene a partir de él.

Nos resta ver la componente gráfica y principal de esta herramienta, la pizarra. Ya mencioné que la pizarra se destruye mediante el botón `Reset`. Se activa mediante el botón `Restart` o selecionando algún ítem.

Al activar la pizarra o cambiar el ítem selecionado el conjunto de sitios se dibuja y se inicializa el árbol de estado T, por lo que se aprecian los dos semitaceles que surgen de las esquinas y el primer evento de círculo en el punto donde se cruzan.

En la Figura A.5 observamos la pizarra luego de seleccionar el ítem *'mini'*, el mismo que ha servido de ejemplo páginas atrás. Pero también observamos en la esquina inferior izquierda de la pizarra dos elementnos más que no habíamos mencionado hasta ahora. El botón *elementos* y la última coordenada del cursor sobre la pizarra con respecto a su esquina superior izquierda.

En la medida que el cursor se mueva sobre la pizarra su cordenada relativa se irá mostrando y actualizando. Esta funcionalidad es de mucha utilidad para corroborar las coordenadas de aquello que se dibuje en la pizarrà.

Al presionar el botón *elementos* accedemos al menú que se observa en la Figura A.6. Este menú contiene un checkbox por cada elemento dibujable en la pizarra que permite activarlo o desactivarlo con un click, o cambiar su color mediante ALT+RightClick como en la Figura A.7.

El último ítem del menú abre un submenú que permite obtener un archivo .bmp con la imagen de la pizarra en colores tal como se la ve, en dos colores (útil sólo cuando se ha clieckeado el ítem `B & W`), en dos colores resaltado (como el caso anterior pero engrosando un pixel el negro) y en varias capas generando un archivo por cada elemento seleccionado en las dos primeras secciones.

El menú desaparecerá luego de realizar cualquier acción en él, si queremos conservarlo es preciso clickear sobre el pin en su esquina superior derecha.

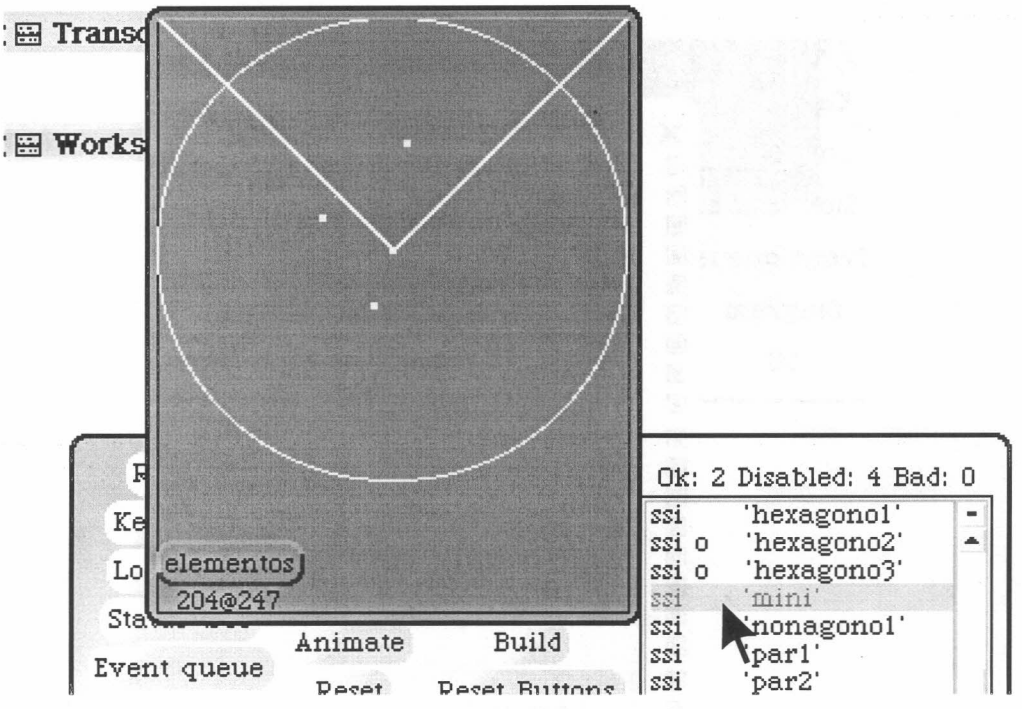


Figura A.5: *'mini'* seleccionado.

Si clickeamos o arrastramos el cursor sobre la pizarra una línea nos seguirá. Es la *Cursor Line* y va de lado a lado de la pizarra en forma paralela a la línea de barrido. Esta línea nos permite dibujar elementos de referencia que han sido de mucha utilidad durante el desarrollo del presente trabajo. Estos elelemtnos constituyen la primera sección del menú. La segunda sección contiene los elementos del diagrama de Voronoi.

Como es de suponerse, a la hora de utilizar estas referencias, es necesario ser mesurado; arrastrar el cursor con todos los elementos activos puede conducirnos a una imagen como en la Figura A.8.

Por ejemplo, cuando el próximo evento de círculo es el señalado en la Figura A.9, adelantando la *Cursor Line* podemos observar la parábola sobre la que se dibujará el tacel que surgirá de atender el evento. En la misma figura podemos observar también otra de los juguetes de Squeak, el `Magnifier` que se toma del `Widget` tab de la misma forma que tomamos el `Workspace` del tab `Tools`.

Para teminar, cuando deseamos cerrar el entorno clickeamos sobre el fondo y obtenemos un menú que en sus últimas posiciones nos ofrece `save and quit` y `quit`. La primera de estas opciones persiste el entorno de manera que cuando volvamos a arrancarlo lo encontramos en exactamente la misma situación en que lo dejamos, la segunda, luego de pedirnos una confirmación, cierra el entorno y descarta cualquier cambio o actividad que hallamos desarrollado en él.

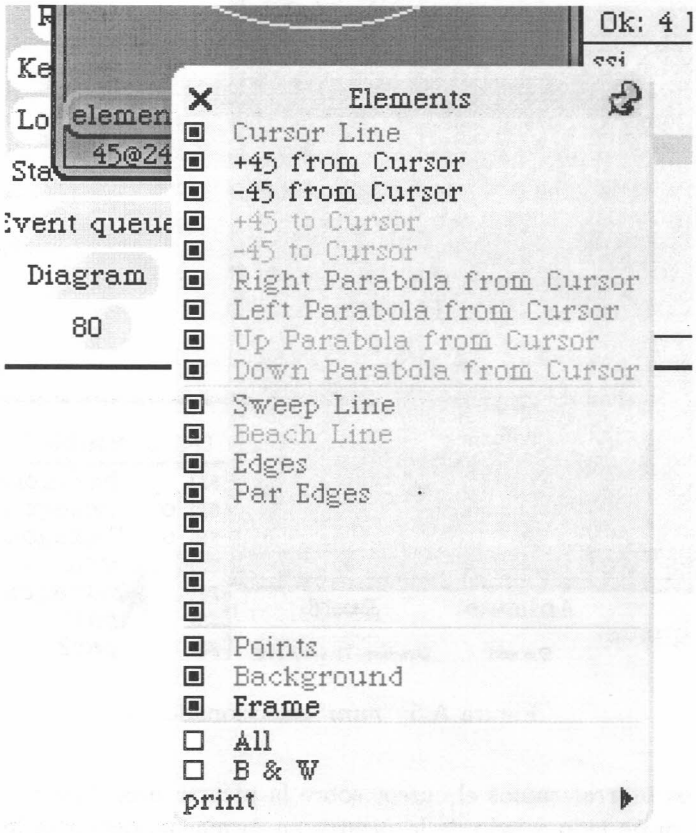


Figura A.6: Elementos

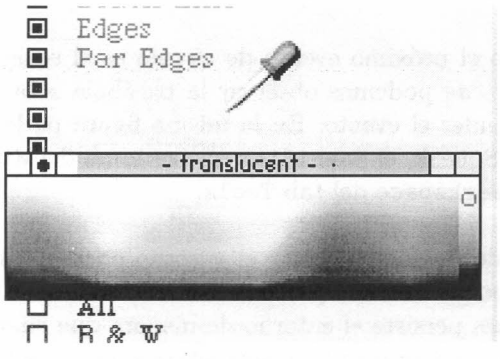


Figura A.7: Selector de color

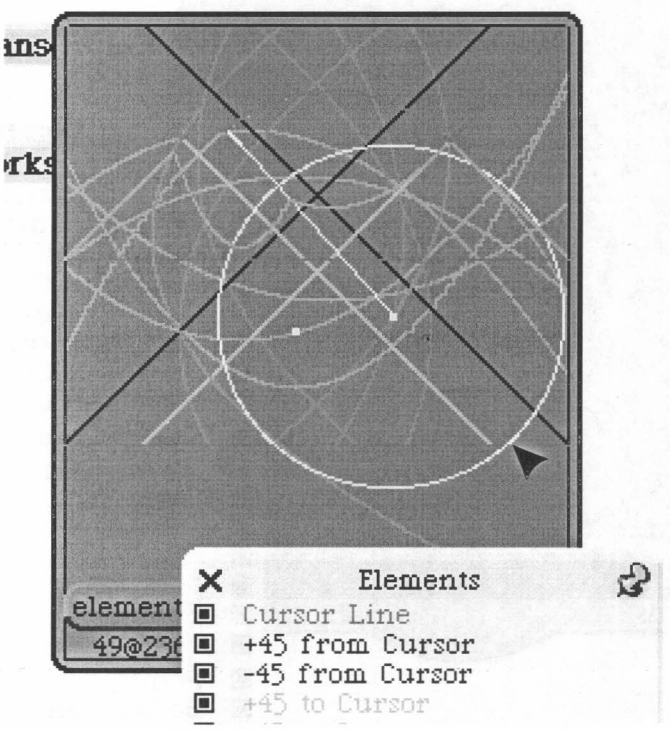


Figura A.8: Todos los elementos de referencia juntos!

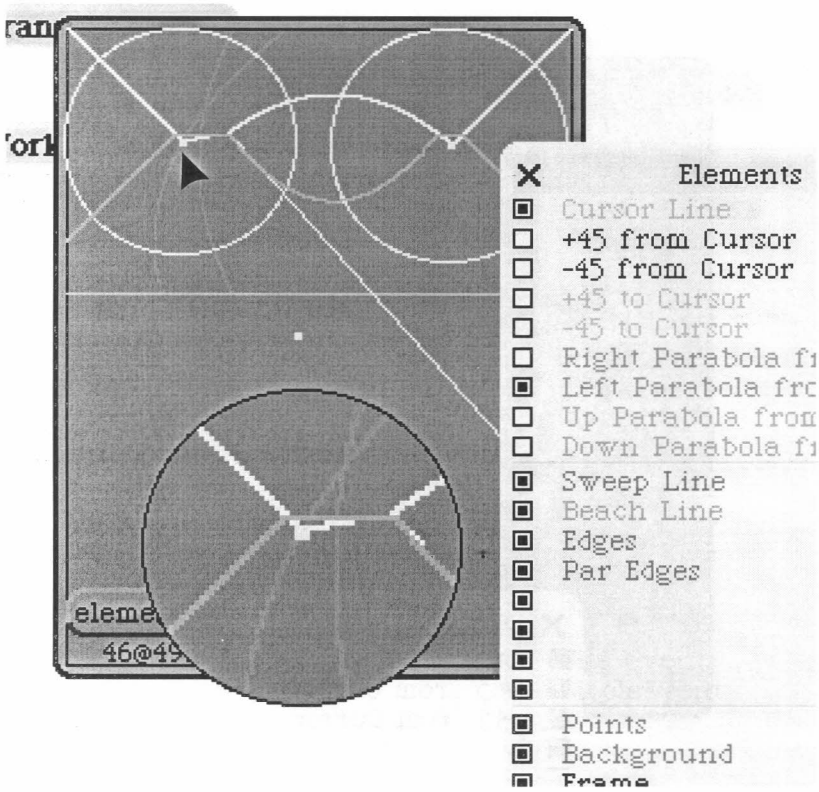


Figura A.9: Mediante la referencia *Left Parabola* podemos prever el recorrido del tacel que surgirá de atender el evento señalado

Apéndice B

Código del algoritmo, comentado

En este apéndice ofrezco una breve introducción de las clases principales que componen el modelo de objetos con el que construí la aplicación y mi adaptación del algoritmo.

Comienzo exponiendo el corazón de la aplicación en las secciones Objetos del Modelo, Elementos del Diagrama e Interfaz. Posteriormente presento las tres subaplicaciones que me ayudaron completar este trabajo. Por último presento algunas clases utilitarias.

B.1. Objetos del Modelo

B.1.1. Recta

Object subclass: **Recta**
instanceVarablesNames: ' pendiente base infinita vertical '

Curva, representa a una Recta.
Posee contructores para obtener rectas de pendiente 0 o infinita, indicando el punto de cruce del eje Y o X respectivamente. También se puede obtener la recta mediatriz entre dos puntos.
A sus instancias se les puede:
* interrogar por su valor en una ordenada o coordenada.
* transformar en otra recta rotandola 90.
* consultar la intersección con otra curva.
* consultar la intersección un rectángulo.

B.1.2. Parabola

Object subclass: **Parabola**
instanceVarablesNames: ' a b c positiva '

Curva, representa una Parábola.
Se pueden obtener instancias de Parabola en función de un punto (que será el foco) y una recta.
A sus instancias se les puede:

* interrogar por su valor en una ordenada o coordenada.
* transformar en otra recta rotandola 90.
* consultar la intersección con otra curva.
* consultar la intersección un rectángulo.

B.1.3. VoronoiEvent

Object subclass: **VoronoiEvent**

Cabeza de la jerarquía de eventos.
Representa un evento en el algoritmo de Fortune.
Define un orden total entre eventos.

B.1.4. CircleEvent

VoronoiEvent subclass: **CircleEvent**
instanceVarablesNames: ' point leaf radi descarted '

Evento de Círculo.
Tiene asociada la hoja del árbol de estado T relacionada a la componente del la línea de playa que tiende a desaparecer con este evento.
Las coordenadas del evento se calculan en función del punto centro del círculo y del radio.

B.1.5. SiteEvent

VoronoiEvent subclass: **SiteEvent**
instanceVarablesNames: ' site '

Evento de Sitio,
Tiene asociado un sitio.

B.1.6. Site

Object subclass: **Site**
instanceVarablesNames: ' point start stop edges '

Sitio puntual en un diagrama de Voronoi.
Tiene un punto asociado.
start y stop indican si el sitio es el origen de un camino o el destino.
Durante el desarrollo del algoritmo va almacenando los taceles (edges) de su perímetro.
Sus instancias saben dibujarse en una pizarra.
Posee cuatro subclases, Ball, Foe, Friend y Milestone que son usadas el las simulaciones

y permiten darle un aspecto gráfico individual a cada clase.

B.1.7. FrameSide

Object subclass: **FrameSide**
instanceVarablesNames: ' recta '

Sitio no puntual en el algoritmo propuesto.
Tiene una recta asociada.
Sus instancias saben dibujarse en una pizarra.

B.2. Elementos del Diagrama

B.2.1. VoronoiVertice

Object subclass: **VoronoiVertice**
instanceVarablesNames: ' point edges unmarkedEdges '

Un vértice en el diagrama de Voronoi.
Posee su coordenada y los taceles que llegan a él.
Permite ir marcando sus taceles para colaborar en el armado de un camino mínimo.
Sabe ofrecer el tacel no marcado con costo mínimo.

B.2.2. VoronoiEdge

Object subclass: **VoronoiEdge**
instanceVarablesNames: ' sitePair startPoint endPoint function marked cost '

Un tacel en un diagrama de Voronoi.
Tiene asociados los sitios que separa y una curva que lo representa.
Tiene asociadas dos coordenadas que delimitan el inicio y fin del tacel en el diagrama.
También se lo utiliza al obtener el camino a través del diagrama.
Sabe transformarse en un EdgeRecord.

B.2.3. VoronoiParabolaEdge

VoronoiEdge subclass: **VoronoiParabolaEdge**
instanceVarablesNames: ' rotated '

Especialización de VoronoiEdge para representar a los taceles parabólicos.

B.2.4. VoronoiHalfEdge

Object subclass: **VoronoiHalfEdge**
instanceVarablesNames: ' edge startPoint endPoint xStep horizontPoint '

Semitacel a ser utilizado por el tacel para descubrir sus extremos.
Tiene un tacel asociado en el que delega toda la información redundante.

B.2.5. VoronoiHalfParabolaEdge

VoronoiHalfEdge subclass: **VoronoiHalfParabolaEdge**

Especialización de VoronoiHalfEdge para representar a los semitaceles parabólicos.

B.2.6. VoronoiDiagram

Object subclass: **VoronoiDiagram**
instanceVarablesNames: ' edges halfEdges faces vertices start stop '

El diagrama de Voronoi.
Aloja los taceles, semitaceles y demás objetos que van apareciendo a medida que se va descubriendo el diagrama.

B.2.7. VoronoiStatus

Object subclass: **VoronoiStatus**
instanceVarablesNames: ' root builder firstLeaf '

La única instancia de esta clase tiene como responsablidad mantener el árbol de estado T agregando o quitando hojas y nodos según el evento atendido.

B.2.8. VoronoiInternalNode

Object subclass: **VoronoiInternalNode**
instanceVarablesNames: ' leftNode rightNode halfEdgePointer padre prev next '

Nodo interno en el árbol de estado T.
Tiene un padre, hijos, links a sus vecinos en la línea de playa y el semitacel que esta trazando.

B.2.9. VoronoiLeafNode

Object subclass: **VoronoiLeafNode**
instanceVarablesNames: ' site circleEventPointer padre prev next '

Hoja que compone el árbol de estado T.
Tiene un padre, links a sus vecinos en la línea de playa, un sitio asociado y posiblemente un evento de círculo.

B.2.10. VoronoiBuilder

Object subclass: **VoronoiBuilder**
instanceVarablesNames: ' points events diagram status line drawBox '

El algoritmo.
Se encarga de ir consumiendo los eventos, armar el arbol T, e interactuar con la consola y la pizarra.

B.3. Interfaz del diagrama de Voronoi

B.3.1. VoronoiCommander

AlignmentMorph subclass: **VoronoiCommander**
instanceVarablesNames: ' voronoi builder dashBoard animateButton setsList setsList-Morph sitesAmount selectionIndex testResultLabel roadMap '

Implementa el tablero de comando para explorar el la evolución de la construcción de un diagrama de Voronoi.

B.3.2. VoronoiDashBoard

DashBoard subclass: **VoronoiDashBoard**
instanceVarablesNames: ' voronoi cursorLine cursorPlace lastLine background elements '

Pizarra sobre la que se dibujan los componentes de un diagrama de Voronoi

B.3.3. VoronoiRoadMap

Object subclass: **VoronoiRoadMap**
instanceVarablesNames: ' dashBoard diagram exploredEdges pathTree commonEdge box '

Posee la inteligencia para obtener un caminio en base a un diagrama de Voronoi.
Se encarga de ir dibujando en el RaodMapDashBoard.

B.3.4. RoadMapDashBoard

DashBoard subclass: **RoadMapDashBoard**
instanceVarablesNames: ' roadMap '

Pizarra para dibujar el desarrollo del algorítmo para obtener el camino mínimo entre dos puntos sobre el diagrama de Voronoi.

B.4. Elementos de la Subaplicación de Testeo

B.4.1. SampleSet

Object subclass: **SampleSet**
instanceVarablesNames: ' points records sampleName enabled boardSize imageName '

Representa un caso de test, incluye un conjunto de puntos y el conjunto de taceles esperado de obtener Voronoi en base al conjunto de puntos.

B.4.2. SampleSetItem

Object subclass: **SampleSetItem**
instanceVarablesNames: ' sample saved match '

Representa al caso de test SampleSet en la lista de casos contenida en VoronoiCommander.

B.4.3. EdgeRecord

Object subclass: **EdgeRecord**
instanceVarablesNames: ' points '

Representa un tacel que puede ser guardado en disco para ser utilizado en un caso de test.
Tiene dos subclases que representan taceles rectos o parabólicos respectivamente.

B.5. Elementos de la Subaplicación de Servicio Voronoi

B.5.1. VoronoiService

Object subclass: **VoronoiService**
instanceVarablesNames: ' dashBoard '

Posee un solo método 'milestone: aCollection on: box'que recibe una colección de puntos y el tamaño de la cancha.
El primer elemento de la colección de puntos es el origen del camino a buscar, el último es el destino.
Retorna la coordenada final del primer segmento que conforma el camino encontrado.
Delega y organiza su trabajo en VoronoiBuilder y VoronoiRoadMap.

B.5.2. ClientManager

Object subclass: **ClientManager**
instanceVarablesNames: ' quiet shouldStop '

Se encarga de recibir conexiones tcp e invocar a VoronoiService con lo recibido.
Puede trabajar de manera silenciosa o mostrar lo que va ocurriendo gráficamente.

B.5.3. ServerDashBoard

DashBoard subclass: **ServerDashBoard**
instanceVarablesNames: ' diagram points target '

Permite visualizar graficamente el servicio que brinda VoronoiService.

B.6. Elementos de la Subaplicación de Simulación

B.6.1. VoronoiPlayer

AlignmentMorph subclass: **VoronoiPlayer**
instanceVarablesNames: ' voronoi builder dashBoard traceButton playButton tcpButton roadMap reader positionSlider mySocket '

Permite tomar un log de corrida de la aplicacón del equipo de UBASot y mostrarlo gráficamente supeorponiendo el gráfico del diagrama de Voronoi en cada paso.

B.6.2. ScriptReader

Object subclass: **ScriptReader**
instanceVarablesNames: ' lines map players idx '

Permite leer un log para simular su corrida.

B.6.3. PlayerDashBoard

VoronoiDashBoard subclass: **PlayerDashBoard**
instanceVarablesNames: ' target pathTree trace '

Pizarra sobre la que se dibujan los componentes de un diagrama de Voronoi.
Se utiliza en la simulación de archivos de log.

B.7. Utilitarios

B.7.1. TextTranslator

Object subclass: **TextTranslator**

Utilitario, transforma las clases Smalltalk y los comentarios en código LaTeX.
Permite extraer sólo la clase y su definición o incluir también un detalle de los métodos.

B.7.2. TestLogger

Object subclass: **TestLogger**
instanceVarablesNames: ' context '

Loggea la jecución de toda la bateria de test como casos de test en C++ u otros dialectos de Smalltalk en la forma de código ejecutable.
De esta manera se puede controlar que la traducción a otro lenguaje se comporta de manera similar a la versión original Squeak.

B.7.3. DashBoard

RectangleMorph subclass: **DashBoard**
instanceVarablesNames: ' cachedCanvas drawBox update '

Pizarra genérica para dibujar componentes.

B.7.4. PriorityQueue

Object subclass: **PriorityQueue**
instanceVarablesNames: ' list '

Implementa una cola de proridad, sus elementos son instancias de VoronoiEvent

Apéndice C

Glosario

Principalemente basado en [PL/99].

straight skeletons[AA/96]: es una la alternativa a los diagrama de Voronoi que puede resolverse en tiempo esperado de $O(n\ log(n))$ y es utilizado para obtener los esqueletos de figuras. No es apropiado para figuras con objetos puntuales sino más bien para polígonos o polilíneas. No es comparable al grafo de Voronoi ni se puede establecer un paralelismo en su construcción. Por definición se constituye puramente de segmentos mientras que en Voronoi abundan los arcos de parábolas.

incircle test: determinar si un punto está dentro del círculo conformado por otros tres o más puntos.

región de Voronoi, celda, cara, área de Voronoi, Voronoi cell: en un espacio salpicado de sitios, una región de Voronoi contiene uno de los sitios generadores del diagrama de Voronoi y abarca todos los puntos que se hallan más cerca del ese sitio que de cualquier otro.

diagrama de Voronoi, Voronoi diagram: partición del espacio en regiones llamadas celdas o regiones de Voronoi.

sitio, owner, site, generador de Voronoi, atractor: curva (frecuentemente un punto) sobre la que se tomarán las distancias para delimitar una Voronoi cell.

línea de barrido, sweepline: línea que delimita el área barrida y los eventos analizados del resto del área por barrer en el algoritmo de Fortune.

línea de playa, wavefront, beach line: serie de arcos de parábola que tienen un sitio como foco y separa los puntos del espacio en aquellos que están más cerca del sitio que de la línea de barrido en el algoritmo de Fortune.

arcos, wavefront component: uno de los arcos que componen la línea de playa en el algoritmo de Fortune.

wavefront element: sitio que genera y es foco de un arco en la línea de playa en el algoritmo de Fortune.

evento de sitio, site event: evento en el cual un sitio es alcanzado por la línea de barrido en el algoritmo de Fortune.

evento de círculo, circle event, spike event: evento que anuncia que la línea de barrido ha provocado por su avance que en la línea de playa un tacel se encuentre con otro tacel en el algoritmo de Fortune.

tacel, edge: es la curva que separa una región de Voronoi de otra. Cada uno de sus extremos, o es infinito, o es un vértice del diagrama.

vértice, vertex: punto en común a tres o más regiones de Voronoi. Sobre los vértices confluyen tantos taceles como regiones tenga en común.

semitacel, spike, semiedge: mitad de un tacel que apunta al infinito o está señido a algún quiebre de la línea de playa y es trazado por este en el algoritmo de Fortune.

Apéndice D

Análisis de la bibliografía

Originalmente esta Sección simplemente un apunte de las impresiones del autor sobre los escritos que se ivan analizando en las etapas tempranas de este trabajo. Tomó forma de documento con la intención de servir de referencia al lector.

Approximating Generalized Voronoi Diagrams in Any Dimension - Vleugels, Overmars - 95[VO/95]

Computan el grafo utilizando grillas de tamaño y precisión variables para ir descubriendo la ubicación de los segmentos que componen el diagrama. El resultado es una coleccion de áreas que incluyen los taceles del diagrama de Voronoi con presición tan alta como se desee.

Path-planning by Tessellation of Obstacles - Tane Pendragon, Lyndon While - 03[PW/99]

Presentan un algoritmo intuitivo y luego lo extienden para polígonos. Definen el tipo de relación entre distintos obstáculos y las curvas que sus fronteras generan. Calculan cada curva y luego descartan las que no corresponden. El algoritmo presenta un orden promedio de $O(n^2)$y un peor caso de $O(n^3)$.

Robust plane sweep for intersecting segments - Boissonnat, Preparata - 97[BP/97]

Referente al cómputo de algoritmos geométricos. Analiza el problema de tener errores de precisión en la toma de decisiones del algoritmo.

Robust Proximity Queries, an Illustration of Degree-driven Algorithm Design - Liotta, Preparata, Tamassia - 96[LPT/96]

Referente al cómputo de algoritmos geométricos. Proveen un marco y una metodología para el cómputo de algoritmos geométricos garantizando correctitud topológica.

The Big Sweep, On the Power of the Wavefront Approach to Voronoi Diagrams - Dehne, Klein - 92[DK/94]

Da una definición simple del diagrama de Voronoi.

"Dado un set S de n sitios en el plano. su diagrama de Voronoi es la partición del plano en regiones, una para cada sitio, tal que la región del sitio p contiene todos los puntos del plano que están más cerca de p que de cualquier otro sitio en S."

Definición simple de métrica:

"Sea d la métrica en el plano, o sea, la función que asigna a cada par de puntos a, b una distancia no negativa $d(a,b)$, tal que $d(a,b) = 0$ ssi $a = b$, $d(a,b) = d(b,a)$, y $d(a,c) \leq d(a,b) + d(b,c)$."

El bisector compuesto por los puntos equidistantes entre dos sitios no necesariamente es una curva, pues en la métricas L_2 y L_∞ pueden ser areas.

Terminación del algoritmo:

"Cuando todos los puntos han sido detectados y no quedan pendientes más eventos de círculo el diagrama de Voronoi se obtiene removiendo la línea de playa.

Define un orden en el que considerar los eventos cuando se solapan, evento de círculo y luego evento de sitio.

Explica por que es natural la extensión a vértices de grado mayor que 3. Al resolver el evento de círculo se buscan más eventos de círculo, que pueden conincidir en su punto de origen. Como la línea de barrido no avanza sino hasta el próximo evento, y el próximo será el evento de círculo recién detectado y la línea de barrido (y la de playa) no se moverá. El nuevo evento es resuelto y así sucesivamente.

Voronoi Diagrams - Aurenhammer, Klein - 96[AK/96]

En 4.5.2 definen y desarrollan funciones de distancia convexas (Convex distance functions).

pag 5. Definition 2.1: define Voronoi definiendo primero el bisector, luego el semiplano y luego la región de Voronoi como la intersección de todos los semiplanos generado por el punto de esa región.

pag 43. Describe Generalized Sites. En particular sitios poligonales.

pag 46. Segunda mitad de hoja, discute quienes y como implementaron Voronoi.

pag 57. Define y desarrolla Convex Distance Functions. También habla sobre la familia de normas $L_p, 1 \leqslant p < \infty$ (o de Linkowski). Concluye que L_∞ son equivalentes pero rotadas 45 grados.

pag 72. Aplicaciones geométricas. Post Office.

pag 87. Motion Planning. Diagramas de Voronoi con segmentos como obstáculos dentro de un marco que también es obstáculo.

N. Mayya and V. Rajan. Voronoi diagrams of polygons: A framework for shape representation. In CVPR94, pages 638643, 1994. URL citeseer.nj.nec.com/mayya94voronoi.html.[MR/94]

Usan Voronoi para representar el esqueleto de formas en el plano.

A Linear-time Randomized Algorithm for the Bounded Voronoi Diagram of a Simple Polygon - Klein, Lingas.[KL/93]

Usan Voronoi para representar el esqueleto de formas en el plano.

O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. Proc. 2nd Annu. Internat. Conf. Computing and Combinatorics, pp. 117–126. Lecture Notes in Computer Science 1090, Springer-Verlag, 1996[AA/96]

Analisis sobre los Straight Skeletons.

Raising Roofs, Crashing Cycles, and Playing Pool - Eppstein, Erickson - 99[EE/99]

Analisis sobre los Straight Skeletons.

Constructing Approximate Voronoi Diagrams from Digital Images of Generalized Polygons and Circular Objects - Roque, Doering - 03[RD/03]

Extrae el perfil de los objetos de una imagen digitalizada, los poligoniza y arma un diagrama de Voronoi con eso, pero cada segmento lo parte en n generadores según la precisión que desee obtener.

Bibliografía

[AA/96] Oswin Aichholzer y Franz Aurenhammer. Straight skeletons for general polygonal figures in the plane. En Springer-Verlag, editor, *Proc. 2nd Annu. Internat. Conf. Computing and Combinatorics*, Lecture Notes in Computer Science 1090, páginas 117–126, 1996.

[AAC+/96] Oswin Aichholzer, Franz Aurenhammer, Chen, D. T. Lee, y Evanthia Papadopoulou. Skew voronoi diagrams, 1996.

[AK/96] Franz Aurenhammer y Rolf Klein. *Chapter XY, Voronoi Diagrams.* Mathematics Series. Elseiver Science B. V., 1996.

[BP/97] Jean-Daniel Boissonnat y Franco P. Preparata. Robust plane sweep for intersecting segments. Informe Técnico 3270, Institut national de recherche en informatique et en automatique, Sophia Antipolis Cedex, France, Septiembre 1997.

[dBvKOS/94] M. de Berg, M. van Kreveld, M. Overmars, y O. Schwarzkopf. *Computational Geometry Algorithms and Applications*, capítulo 7, Voronoi Diagrams, The Post Office Problem, páginas 296–305. Springer, 1994.

[DK/94] Frank Dehne y Rolf Klein. The big sweep, on the power of the wavefront approach to voronoi diagrams. En *Springer Lecture Notes in Computer Science*, volumen 841, páginas 296–305, Adelaide, Australia, Noviembre 1994. 19th International Symposium on Mathematical Foundations of Computer Science (MFCS '94).

[EE/99] David Eppstein y Jeff Erickson. Raising roofs, crashing cycles, and playing pool. *Discrette & Computational Geometry*, 1999. URL http://www.uiuc.edu/ph/www/jeffe/pubs/cycles.html.

[Fio/95] P. Fiorini. *Robot motion planning among moving obstacles.* PhD thesis, University of California, Los Angeles, Enero 1995.

[For/87] S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(2):153–174, 1987.

[Gib/85] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, Cambridge, 1985.

[GS/78] P. J. Green y R. R. Sibson. Computing dirichlet tessellation in the plane. *Computer Journal*, 21:168–173, 1978.

[KL/93] Rolf Klein y Andrzej Lingas. A linear-time randomized algorithm for the bounded voronoi diagram of a simple polygon. *International Journal of Computational Geometry & Applications*, 1993.

[LPT/96] Giuseppe Liotta, Franco P. Preparata, y Roberto Tamassia. Robust proximity queries, an illustration of degree-driven algorithm design. Informe técnico, Nationall Science Fundation, USA, 1996.

[MR/94] N. Mayya y V. Rajan. Voronoi diagrams of polygons: A framework for shape representation. *CVPR94*, páginas 638–643, 1994. URL citeseer.nj.nec.com/mayya94voronoi.html.

[OIM/84] T. Ohya, M. Iri, y K. Murota. Improvements of the incremental methos for the voronoi diagrams with computational comparison of various algorithms. *J. Opr. Res. Soc. Japan*, 27(4):306–336, 1984.

[PL/99] Evanthia Papadopoulou y D. T. Lee. The l_∞ voronoy diagram of segments and vlsi applications. *International Journal of Computational Geometry & Applications*, 1999.

[PW/99] Tane Pendragon y Lyndon While. Path-planning by tessellation of obstacles. En *Conferences in Research and Practice in Information Technology*, volumen 16, Adelaide, Australia, 1999. Twenty-Sixth Australasian Computer Science Conference (ACSC2003).

[RD/03] Roque y Doering. Constructing approximate voronoi diagrams from digital images of generalized polygons and circular objects. En Science Press, editor, *WSCG'2003*, Febrero 2003.

[SH/75] M. I. Shamos y D. Hoey. Clossest-point problems. *In proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, páginas 151–162, 1975.

[SI/92] K. Sugihara y M. Iri. Construction of the voronoi diagrams for 'one million' generators in single-precision arithmetic. *Proc. IEEE*, 80(9):1471–1484, Septiembre 1992.

[SS/03] Sergio Soria y Juan M. Santos. Velocity and trayectory computing using the velocity obstacle approach in robot soccer. *FIRA Robot World Congress*, Octubre 2003.

[VO/95] Jules Vleugels y Mark Overmars. Approximating generalized voronoi diagrams in any dimension. Informe técnico, Department of Computer Science, Utrech University, The Netherlands, Mayo 1995.