UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Algoritmos de búsqueda 3D

Presentado por Víctor Pérez Esteban
en Universidad de Burgos — 19 de junio de 2017
Tutor: Dr. José Francisco Diez Pastor, Dr. César García Osorio

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. Dr. José Francisco Diez Pastor, profesor del departamento de nombre departamento, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Víctor Pérez Esteban, con DNI 71279579A, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 19 de junio de 2017

Vº. Bº. del Tutor:	Vº. Bº. del Tutor:
D. Dr. José Francisco Diez Pastor	D. Dr. César García Osorio

Resumen

En los últimos años hemos visto que la evolución de los sistemas inteligentes se hace cada vez más presente en la sociedad. Uno de los sistemas inteligentes en que más se han observado esfuerzo en el desarrollo, inversiones por las grandes compañías y relevancia social ha sido la aparición de los primeros prototipos de los vehículos autónomos. Coches que se conducen solos.

Para la realización de estos sistemas inteligentes se tienen que combinar muchos procedimientos que doten del conocimiento necesario al sistema inteligente sobre su entorno, y que permitan una vez adquirido su utilización para desplazarse a su destino. Dos de estos procedimientos principales son el cómo conocer la ruta a seguir, y una vez conocida la ruta cómo seguirla hasta alcanzar su final.

En este trabajo simularemos estos procesos en un entorno tridimensional, para poder simular algunos de los problemas que se encuentran los vehículos autónomos en el mundo real y tener una visión de como solventar los problemas que plantea. Se calcularán rutas que se puedan seguir por parte del vehículo y se implementarán métodos para seguir esas rutas de forma autónoma, viendo en cada paso los problemas que se presentan.

Descriptores

Búsqueda de rutas, vehículos autónomos, seguimiento de rutas

Abstract

In recent years we have seen the evolution of intelligent systems that are beginning to be present in society. One of these intelligent systems in what has seen the most effort in development, investments by large companies and social relevance has been the appearance of the first prototypes of autonomous vehicles. Cars that are driven by itselves.

For the realization of these intelligent systems many procedures have to be combined to provide the intelligent system with the necessary knowledge about its environment and to allow it to be used once it has reached its destination. Two of these main procedures are how to know the route to follow, and once known the route how to follow it until reaching its end.

In this work, we will simulate these processes in a three-dimensional environment, in order to simulate some of the problems that autonomous vehicles find in the real world and to have a vision of how to solve the problems that it poses. Routes will be calculated that can be followed by the vehicle and will be implemented methods to follow these routes autonomously, seeing at each step the problems that arise.

Keywords

Routes searching, autonomous vehicles, tracking routes.

Índice general

Índice de figuras

Índice de tablas

Introducción

El vehículo autónomo es uno de los avances tecnológicos que más impacto va a tener en la sociedad en los próximos años. Tal es así, que las mayores compañías ya están trabajando en el cambio que se avecina.

Brian Krzanich, director de Intel declaró [1] que el cambió será tan explosivo que las compañías que no se preparen se enfrentarán al fracaso y a la extinción. Se esperá que las cantidades de dinero que mueva este nuevo sector sean gigantescas, 800 mil millones de dolares para el año 2035 y hasta 7 billones de dolares en lo siguientes 15 años [1].

Este nuevo sector no solo involucra a la industria automovilística y sus empresas tradicionales. Empresas del sector de la informática, como Intel, Nvidia, Google, Apple están invirtiendo en el desarrollo de las tecnologías necesarias. Y muchas empresas mas pequeñas que trabajan en tecnologías necesarias para el coche autónomo han sido compradas por gigantes tecnológicos, como Mobileye comprada por Intel especializada en visión computarizada, Primesense comprada por Apple especializada en los sensores de tres dimensiones, o Waze comprada por Google especializada en los mapas para la navegación GPS [2].

Pero no solo es una revolución en la economía. Supondrá también un cambio social. Los vehículos autónomos tienen menos accidentes que los humanos [3], son más ecológicos el reducir las emisiones de CO_2 [4], y cambiarán la manera de la sociedad de desplazarse. Los coches autónomos no solo permiten usar el tiempo que las personas pasan al volante en otras actividades, también permiten más independencia en sus desplazamientos a personas mayores o con discapacidad. Cambia la forma en que la gente puede compartir vehículos y desplazamientos como lo ha hecho Uber. El espacio de las ciudades destinado aparcamiento podría cambiar drásticamente con vehículos que se aparcan solos. También se verá una reducción del tráfico y la congestión en las carreteras debido a su conducción más eficiente.

Como hemos mencionado, el número de tecnologías necesarias para construir un coche autónomo es enorme. Desde el propio vehículo, todos los sis-

temas para la detección del entorno y la visión computarizada, hasta todo el software necesario que hay que desarrollar. Algoritmos que permitan la percepción, conocer el entorno y las variables imprevistas que surjan. Algoritmos que permiten conocer la localización exacta en tiempo real. Algoritmos que permitan planificar las acciones que debe llevar a acabo el vehículo. Y algoritmos que permitan la locomoción del vehículo y desplazarlo siguiendo la planificación.

De todas estás tecnologías, en este proyecto nos hemos fijado en las dos últimas. Al usar un entorno tridimensional para realizar la simulación, podemos conocer el entorno en el que se encuentra el vehículo así como su localización. De esta forma podemos centrarnos en el desarrollo de algoritmos permitan planificar una ruta y que le indiquen a un vehículo como desplazarse de forma autónoma en un entorno continuo en un espacio tridimensional, como ocurre en la realidad. También nos permite ver las nuevas dificultades que plantea y adaptar los algoritmos que hemos ido viendo en la carrera en un entorno más realista, así como otros nuevos que permiten entender y solventar los problemas a los que se enfrentan los coches autónomos que circulan por las calles.

Objetivos del proyecto

Los objetivos del proyectos son:

1. Crear un espacio tridimensional dónde poder representar y probar el desarrollo de un vehículo autónomo.
2. Desarrollar algoritmos que permitan la búsqueda de rutas desde la posición del vehículo hasta una meta.
3. Dotar el vehículo de autonomía para poder seguir las rutas calculadas por si mismo, mediante mecanismos que permitan calcular el giro del volante y la fuerza que el motor debe aplicar a las ruedas para ajustarse a la ruta planificada.

En el proyecto se implementarán los algoritmos que hemos considerado más interesantes, y se plantea de tal forma que sea posible añadir en el futuro nuevos algoritmos para poder mejorar la autonomía del vehículo.

Conceptos teóricos

En esta sección vamos a explicar los algoritmos utilizados en el desarrollo del proyecto. Primero veremos los algoritmos relativos a la planificación de la ruta, y a continuación aquellos relativos al seguimiento de la misma de una forma autónoma por parte del vehículo.

3.1. Algoritmo A*

A* es un algoritmo de búsqueda informada del tipo primero el mejor, que usa una función de evaluación para elegir hacia que nodo expandirse desde un nodo inicial hacia un nodo final.

Para representar el espacio de búsqueda del algoritmo, se usa una cuadrícula, es por lo tanto un algoritmo que opera en un espacio de estados discreto. Cada nodo de la cuadricula puede representar un espacio donde es posible desplazarse o un obstáculo que es inalcanzable.

La función F() de evaluación calcula el coste de cada nodo, con lo que se eligen los nodos con menor coste para llegar al destino a través del camino óptimo. Esta función está formada por dos funciones a su vez: una función G() que calcula el coste del camino seguido desde el nodo inicial a ese nodo concreto, y una función H() o función heurística que hace un estimación del coste del camino desde ese nodo al nodo final o meta.

Al algoritmo comienza desde el nodo inicial explorando los nodos adyacentes o sucesores, cual es de menor coste. Un nodo ya explorado, es decir del que ya se ha buscado sus sucesores, se manda a la lista de nodos cerrados. Con los sucesores se forma una lista de nodos abiertos o por explorar, de la cual se elige el de menor coste para ser el siguiente en ser explorado, hasta que se alcance el nodo meta o no queden más nodos por ser explorados (si no quedasen más nodos significaría que no hay un camino entre el nodo inicial y la meta). Si al explorar un nodo esta ya se encontraba en alguna de las listas

de nodos abiertos o cerrados, se actualizarán los valores de los nodos al de menor coste encontrado.

Heurística

Al algoritmo A* es completo, lo que significa que encontrará un camino hasta la meta siempre que este exista. Además, para que sea admisible, que significa que siempre encontrará un camino óptimo, su función H() también debe ser admisible.

Una función H() es admisible siempre y cuando no sobrestime el coste del camino desde un nodo hasta la meta. Por ejemplo se puede considerar que el camino desde un nodo hasta la meta será la línea recta.

La admisibilidad del A* trae consigo un gran coste computacional debido al gran número de nodos explorados. Para mejorar la eficiencia podemos dar pesos a las funciones G() y H(), de tal forma que si damos mas valor a G() la búsqueda se expandirá en anchura buscando el camino, mientras que si damos más valor a H() se expandirá más rápido acercándose a la meta.

Pseudocódigo A*

Además de secciones tenemos subsecciones.

3.2. *Path Smoothing* o Suavizado

Poner imagenes comparativas

Un problema del A* tradicional es que las rutas que encuentra, aunque sean las más cortas no tienen una apariencia realista. Esto es debido a que el A* discretiza el espacio de búsqueda, en nuestro caso el espacio en tres dimensiones, para poder buscar la ruta. Dependiendo de la representación que elijamos, se acercará más a o menos a la realidad, pero en cualquier caso se producirá una diferencia que suele estar alejada de la representación ideal de esa misma ruta.

Por ejemplo, en el caso de una búsqueda en un parrilla, si seguimos el camino a través de cada casilla, cuando el camino siga una ruta en diagonal, el A* seguirá un camino en zigzag. Aunque este camino es perfectamente válido, en la realidad no se sigue un camino en zigzag sino que se sigue la línea recta que representa. En el caso de las curvas el problema es parecido. El camino encontrado estará formado por segmentos rectos en vez de seguir una ruta redondeada.

Por este motivo una vez realiza una vez obtenida la ruta, un proceso de suavizado de tal forma que se acerque la ruta obtenida a través del A*, a la representación ideal de esa ruta.

Eliminar el zigzag

Poner imagenes

El primer método que hemos usado y que suele dar buenos resultados, es eliminar el zigzag. Este método es básicamente lo mismo que realiza el theta* que explicaremos más adelante en 3.3.

El zigzag es el problema más habitual que nos hemos encontrado. Al representar un espacio de tres dimensiones a través de casillas, se produce habitualmente porque aunque se use la representación octal permitiendo el movimiento diagonal, esto al pasarlo a un espacio en tres dimensiones quiere decir que sólo tenemos ocho posibles ángulos de movimiento: 0^o, 45^o, 90^o, 135^o, 180^o, 225^o, 270^o y 315^o

En realidad las posibilidades reales para movernos son cualquier angulo de los 360^o. Por tanto, si nuestro objetivo está en un angulo diferente a esos ocho, se produce un zigzag combinándoles hasta que se consigue llegar.

La forma para eliminar el zigzag, es comprobar si es posible eliminar estos pasos intermedios. Es decir, si para llegar al objetivo el A* ha usado varias casillas en el espacio discreto, es posible que en un espacio no discreto pudiéramos llegar directamente con un movimiento en un ángulo distinto a los ocho que permite el A*.

Para ello, usamos la línea de visión. Si existe una línea de visión entre una casilla del A* y otra quiere decir que podemos desplazarnos siguiendo una línea recta formada entre esas dos casillas. Con las casillas devueltas por el A*, comprobamos si existe línea de visión entre ellas. Entre casillas consecutivas siempre habrá línea de visión, porque si no, no sería un camino válido, pero puede también haber línea de visión entre las siguientes de forma que haya casillas sobrantes debido al zigzag y a la forma que tiene el A* de buscar el camino.

Así que lo que comprobamos es que existe una línea de visión entre una casilla del A* y la siguiente más lejana posible, y eliminamos a las casillas intermedias. De esta forma obtenemos una línea recta entre esas dos casillas y eliminamos el zigzag producido por las casillas intermedias.

Descenso gradiente

poner imágenes con resultados de distintos alphas y betas

El descenso gradiente es un algoritmo iterativo de optimización que a partir de N valores iniciales $(x_1, x_2, x_3, ..., x_n)$, itera una función modificando esos valores gradualmente hacía un mínimo de una función $f(x_1, x_2, x_3, ..., x_n)$. El algoritmo termina cuando el cambio que se producen en los valores es menor que la tolerancia indicada.

Para optimizar la ruta, queremos obtener los valores que minimizan las distancia entre los puntos sucesivos de la ruta, y a su vez que minimicen las distancia a la ruta original.

Para minimizar la distancia entre los puntos con respecto a la ruta original usaremos las función:

$$y_i = y_i + \alpha(x_i - y_i)$$

Donde α es el peso que le damos a cuanto de cerca queremos que la ruta suavizada esté de la ruta original, x es el punto de la ruta original e y es el nuevo punto suavizado.

Para minimizar la distancia entre los puntos sucesivos de la ruta, usamos la función:

$$y_i = y_i + \beta(y_{i+1} + y_{i-1} - 2 * y_i)$$

Donde β es el peso que le damos al suavizado de la ruta, y tenemos en cuenta la distancia tanto con el punto anterior como con el puto siguiente.

Para calcular el error que se produce, o el cambio hacia el mínimo, usamos la función:

$$error = error + (z_i - y_i)$$

Donde z es el valor de y antes de calcular el nuevo mínimo.

Curvas Bézier

Las curvas Bézier [5, 6] son una forma de representar curvas a través de una función que recibe un parámetro. Hemos usado la función de las curvas de Bézier para suavizar la ruta y así obtener giros y curvas más cercanas a una representación real

Una función de una curva Bézier tiene varios puntos de control (al menos dos). De estos puntos, el primero y el último representan el inicio y el final de la curva. Esta función además recibe un parámetro que puede tomar los valores comprendidos entre cero y uno. Si el parámetro toma el valor cero, entonces la función devuelve el punto de inicio, mientras que si toma el valor uno devuelve el punto final. De esta forma, dando valores entre cero y uno, la función devuelve los valores que se encuentran entre los puntos de inicio y de fin.

Si la función toma dos puntos de referencia, lo que obtendremos sera una recta y sus puntos intermedios. Si la función toma tres puntos entonces tendremos una curva donde el vértice será cercano al punto intermedio. Podemos

usar más puntos para representar curvas más complejas o dar curvas con más variedad de formas.

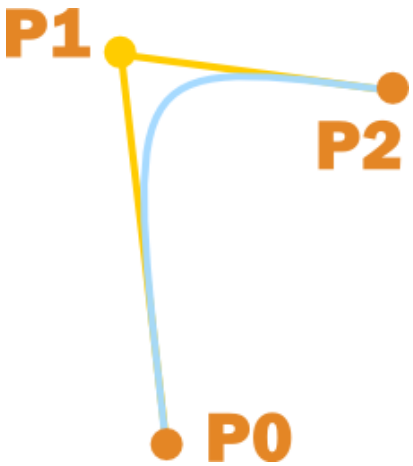

Figura 3.1: Esquema de una curva Bézier cuadrática de Herman Tulleken, Bézier Curves for your Games: A Tutorial [7].

Hemos usado la función Bézier con tres puntos después de eliminar el zigzag, así que cada tres puntos del A* sin zigzag se usan como puntos de control que servirán para crear la curva. Si los puntos más o menos están alineados, el resultado será una recta suavizada, y si forman una curva, se eliminan los segmentos rectos y se reemplazan con puntos que forman una curva.

La función Bézier cuadrática para tres puntos de control y parámetro t es:

$$[x, y, z] = (1–t)^2 P_0 + 2(1–t)tP_1 + t^2 P_2$$

En formato expandido:

$$\begin{aligned} x &= (1–t)^2 x_0 + 2(1–t)tx_1 + t^2 x_2 \\ y &= (1–t)^2 y_0 + 2(1–t)ty_1 + t^2 y_2 \\ z &= (1–t)^2 z_0 + 2(1–t)tz_1 + t^2 z_2 \end{aligned}$$

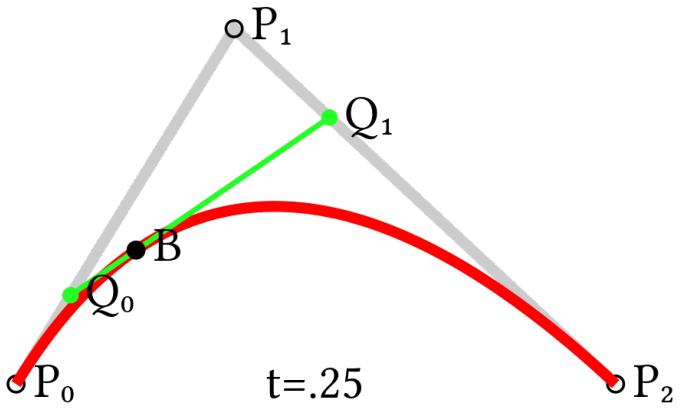

Figura 3.2: Esquema de una curva Bézier cuadrática de Bézier curve — Wikipedia [8].

3.3. Theta*

Theta A* es un algoritmo desarrollado por A. Nash, K. Daniel, S. Koenig y A. Felner [9, 10] en 2010 que modifica el A* original para obtener rutas más realistas.

Como explicamos anteriormente, el A* por la forma en la que representa el espacio de búsqueda da resultados poco realistas, con lo que hay que realizar un postprocesado de la ruta obtenida. El theta* incorpora este proceso al propio A* para obtener de forma directa rutas que no requieran del proceso de suavizado o que lo minimice.

Para ello, el cambio que hace el Theta* respecto al A* es que un sucesor a una casilla puede ser cualquier casilla, en vez de ser unicamente las que tiene alrededor. El algoritmo es el mismo que el A* original, pero cuando se calculan los sucesores, además de calcular si es un sucesor válido y el coste de ese sucesor, comprueba si hay linea de visión con la casilla del nodo padre del nodo que lo generó. Si no hay línea de visión, sigue el algoritmo del A* y le asigna al sucesor como padre la casilla que lo originó. Pero si hay línea de visión, le asigna como padre no el nodo que lo generó, sino el padre de este, calculando el coste del sucesor teniendo en cuenta esta otra casilla. De esta forma elimina las casillas intermedias que origina el A* que no son necesarias de forma similar a como se realizaría en el post procesado para eliminar el zigzag.

Podemos observar el funcionamiento del algoritmo en la figura 3.3. La línea discontinua roja muestra el camino encontrado por el A*. En cambio, el Theta*, al haber una línea de visión entre el nodo S_{start} y el nodo S', le asigna como padre al nodo S' el nodo S_{start} sin pasar por el nodo intermedio S como hace el A*.

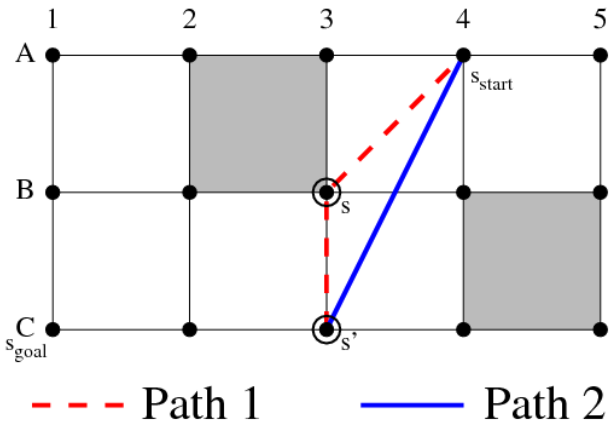

Figura 3.3: Comparación de los caminos elegidos por el A* (path1) y el theta* (path2) de [11].

Pseudocódigo Theta*

3.4. A* con vértices

El algoritmo A* como hemos visto, divide el espacio de búsqueda en casillas de un espacio discreto. Esto presenta los problemas de rutas poco realistas, y además al ser discreto, no podemos representar una línea recta entre dos puntos distantes, si no que debemos recorrer todas las casillas intermedias.

Podemos mejorar ambos problemas usando un *Nav Mesh* 5.1. Un *Nav Mesh* usa polígonos para representar el espacio de búsqueda de tal forma que los lados de estos polígonos y sus vértices se encuentran en los bordes del espacio recorrible. De esta forma, podemos usar esos vértices como las casillas del espacio discreto, lo que produce tres mejoras importantes:

1. La representación del espacio recorrible se ajusta más a un espacio continuo al poder ir de un vértice a otro sin pasar por estados intermedios.
2. Permite que las rutas obtenidas sean más realistas que las obtenidas con la representación discreta.
3. Al ser un espacio continuo dentro de los polígonos del *Nav Mesh*, evitamos pasar por todos los estados intermedios de una representación discreta, reduciendo enormemente las casillas a explorar y reduciendo el tiempo de ejecución del algoritmo.

NOTA: no se que mas comentar

3.5. *Hybrid A**

Los algoritmos que hemos visto anteriormente, tanto los que usan una representación totalmente discreta hasta los que buscan aproximarse a rutas más realistas en un espacio continúo, tienen el problema de que no tienen en cuenta las características del vehículo que va a realizar la ruta.

El *Hybrid A** se basa en el A* tradicional, pero teniendo en cuenta estos dos factores: que el vehículo se mueve por un espacio continuo, y que lo hace siguiendo sus limitaciones físicas.

Para ello, utiliza una aproximación continua al A*. En nuestro caso, la aproximación usada ha sido el modelo de la bicicleta. Este modelo es válido porque los ejes de un vehículo son simétricos, por lo que el resultado de un lado del vehículo será equivalente al del otro lado, pudiéndolo simplificar como si fuera un solo eje formado por la rueda delantera y la rueda trasera.

Cual es la referencia correcta a Sebastian Thrun? insertar imagen del modelo de la bicicleta

Los elementos de este modelo son:

1. La rotación del vehículo θ.
2. El ángulo de giro de las ruedas α
3. La distancia entre los ejes l
4. La distancia recorrida por el vehículo d

Con estos elementos podemos calcular en que posición se encontrará el vehículo en un espacio de tiempo, teniendo en cuenta su capacidad de maniobra.

Las formulas para el calculo de la nueva posición del vehículo son:

$$x' = x + v\Delta t sin\theta$$
$$z' = z + v\Delta t cos\theta$$
$$\theta' = \theta + \Delta t\omega$$

No se cual sería la formula para la y

Para $v\Delta t$ hemos realizado una aproximación, tal que hemos considerado que la velocidad del vehículo v va a ser constante, y que Δt es el tiempo que tarda en recorrer una unidad de nuestra representación. Esto es válido en nuestro caso y nos permite simplificar las ecuaciones porque la velocidad de nuestro vehículo a través del *PID Control* va a ser aproximadamente constante. En un entorno real v sería la velocidad del vehículo entre el lapso de tiempo recorrido entre las mediciones representado por ΔT

Para calcular ω, que es el radio de giro que realiza el vehículo en ese espacio de tiempo continuo, usamos la siguiente fórmula: referencia de https://www.youtube.com/watch?v=5lV https://www.udacity.com/course/intro-to-artificial-intelligence–cs271 Norvig y thrun

$$\omega = \frac{d}{l} * tan\alpha$$

Con estos elementos ya podemos calcular cual es el siguiente estado del vehículo en un espacio continuo. Los sucesores del estado actual serán esos estados continuos, variando el angulo de giro, y podemos tener en cuenta el sentido permitiendo al vehículo ir marcha atrás añadiendo los sucesores correspondientes.

Esto nos permite además de obtener rutas más realistas, rutas que se adecuan a la maniobrabilidad del vehículo, rutas que permiten maniobrar y cambiar de sentido si fuera necesario.

Una de las desventajas del *Hybrid A** es que no es un algoritmo completo al contrario que el A*. Cuando calculamos los sucesores de un estado se asigna el mejor a una casilla discreta correspondiente del espacio discreto. Además, de forma tradicional el *Hybrid A** si vuelve a una casilla ya explorada, descarta ese camino. De esta forma, aunque en la mayoría de los casos encontrará un camino, no siempre lo hará aunque exista.

3.6. *PID Control*

Algoritmo	Discreto	Continuo
A*	X	
Theta*	X	
A* vértices		X
Hybrid A*		X

Tabla 3.1: Comparación de los algoritmos según su representación

Algoritmo	Segundos	Porcentaje
A*	10.0	100 %
Theta*	9.0	90 %
A* vértices	1.0	1 %
Hybrid A*	15	150 %

Tabla 3.2: Comparación de los algoritmos según su rendimiento

3.7. Comparativa de los algoritmos

En esta sección vamos a comparar los algoritmos dependiendo de si utilizan una representación del espacio de búsqueda continua o discreta. A continuación también compararemos el rendimiento de los distintos algoritmo tomando como referencia al A*.

coloca las tablas donde no es no se porque

poner valores reales

3.8. Imágenes

Se pueden incluir imágenes con los comandos standard de LaTeX, pero esta plantilla dispone de comandos propios como por ejemplo el siguiente:

Figura 3.4: Autómata para una expresión vacía

3.9. Listas de items

Existen tres posibilidades:

- primer item.
- segundo item.

1. primer item.
2. segundo item.

Herramientas	App AngularJS	API REST	BD	Memoria
HTML5	X			
CSS3	X			
BOOTSTRAP	X			
JavaScript	X			
AngularJS	X			
Bower	X			
PHP		X		
Karma + Jasmine	X			
Slim framework		X		
Idiorm		X		
Composer		X		
JSON	X	X		
PhpStorm	X	X		
MySQL			X	
PhpMyAdmin			X	
Git + BitBucket	X	X	X	X
MikTEX				X
TEXMaker				X
Astah				X
Balsamiq Mockups	X			
VersionOne	X	X	X	X

Tabla 3.3: Herramientas y tecnologías utilizadas en cada parte del proyecto

Primer item más información sobre el primer item.

Segundo item más información sobre el segundo item.

-

3.10. Tablas

Igualmente se pueden usar los comandos específicos de LATEXo bien usar alguno de los comandos de la plantilla.

Herramientas	App AngularJS	API REST	BD	Memoria
HTML5	X			
CSS3	X			
BOOTSTRAP	X			
JavaScript	X			
AngularJS	X			
Bower	X			
PHP		X		
Karma + Jasmine	X			
Slim framework		X		
Idiorm		X		
Composer		X		
JSON	X	X		
PhpStorm	X	X		
MySQL			X	
PhpMyAdmin			X	
Git + BitBucket	X	X	X	X
MikTEX				X
TEXMaker				X
Astah				X
Balsamiq Mockups	X			
VersionOne	X	X	X	X

Tabla 3.4: Herramientas 0

Técnicas y herramientas

4.1. Programas usados

Unity

Unity un motor para juegos usado principalmente para desarrollar videojuegos y simulaciones. Que sea un motor para juegos quiere decir que está formado por varios motores a su vez, como el motor gráfico y el motor de físicas.

Unity proporciona un editor donde se puede crear escenas junto con un gran número de herramientas, ejemplos y modelos para crear escenarios tanto 2D y principalmente 3D.

Tiene una licencia de uso "por pasos" de ingresos obtenidos por el contenido creado con el. Existen distintas versiones de la licencia según el uso que se vaya hacer así como de la cantidad de ingresos que se generen con su uso. En la versión Personal, su puede usar libremente siempre que los ingresos generados por el contenido usado no supere los 100 mil dólares anuales, y es la versión de la licencia usada. Las siguientes versiones de la licencia requieren un pago de una cuota anual por cada usuario y el límite de ingresos por el uso comercial del contenido creado va aumentando.

Para el proyecto probamos de forma descendiente las distintas versiones de Unity para linux en un xUbuntu 16.04, debido a que las últimas versiones producían errores que cerraban el editor al iniciarlo o al poco tiempo después de abrirse. Creemos que esto se debía a un problema de compatibilidad entre los *drivers* de la tarjeta gráfica que no tenían soporte para algunas de las nuevas características de Unity. La primera versión estable que funcionaba en el equipo de desarrollo fue la versión 5.3.6f1, y ha sido la usada para su realización.

Texmaker

Para la realización de la memoria se ha usado latex con el editor Texmaker. Texmaker es un editor libre con licencia GPL.

Texmaker incluye varias herramientas como el corrector ortográfico o el visor de pdf que facilitan la creación de los documentos.

La versión usada ha sido la 4.4.1.

SmartGit

SmartGit es un gestor gráfico para git que soporta github y que facilita la realización de commits, push y pull de los repositorios del proyecto.

Tiene una licencia no comercial que permite el uso de forma gratuita a desarrolladores de código abierto, profesores, estudiantes, desarrollos no comerciales y también organizaciones sin ánimo de lucro.

La versión que hemos usado ha sido la 17.0.3

GitHub y ZenHub

La plataforma elegida para los repositorios ha sido github, principalmente por su compatibilidad con Zenhub que es la herramienta de gestión de proyectos que hemos usado.

Zenhub es una herramienta que permite la planificación y control del proyecto. Es un complemento de firefox que se integra con github y añade funciones como las boards para manejar el control de las issues y las milestones del proyecto, y que permite ver los gráficos del progreso y burnouts que se ha realizado.

La versión utilizada ha sido la 2.34.2

Aspectos relevantes del desarrollo del proyecto

5.1. *Navigation Mesh*

Una navigation mesh o nav mesh es un conjunto de polígonos, triángulos normalmente, que se usan en los motores gráficos 3D para representar la zona o camino recorrible de un agente. Se crean a partir de los objetos estáticos de una escena, que son los obstáculos, donde no se puede desplazar. El resto de la zona de la escena será zona recorrible por donde nos podemos desplazar.

En relación por ejemplo al A*, sería el equivalente a la matriz que representa el mapa donde se buscará los nodos a explorar.

Está formada por los vértices de los polígonos, y si un punto esta dentro del área que forman entonces ese punto es recorrible. Esto es mas realista en un escenario en tres dimensiones debido a que permite un movimiento menos discretizado, al contrario que las parrillas de casillas que se adecuan más a una representación en dos dimensiones. Y si consideramos los vértices los nodos sucesores, resulta más óptimo computacionalmente, ya que el número de nodos (vértices) a explorar es menor, debido a que no tenemos que tener en cuenta todos los puntos intermedios hasta llegar al vértice, si no unicamente si el vértice es visible o alcanzable desde el nodo actual.

Hemos usado una nav mesh para generar de forma automática el mapa para el A*. En Unity esto se puede hacer de dos formas. En versiones anteriores a la versión 5.6 solo es posible generarlo desde el editor antes del tiempo de ejecución. A partir de la versión 5.6, que permite acceder a las herramientas del editor en tiempo de ejecución, se puede construir a través de NavMeshBuilder, dando mucha más libertad a la hora de crear mapas de forma dinámica o interactiva. Lamentablemnte no se pudo usar la versión 5.6 y la versión más actual que funcionaba en el equipo de desarrollo fue la 5.3.6.

La clase NavMesh de Unity tiene métodos que podemos usar como RayTrace, que lanza un rayo desde un vector a otro y devuelve si son visibles dentro de la NavMesh, es decir, si existe una línea recta entre ellos que los una dentro del camino recorrible. Este método lo hemos usado para decidir si un nodo sucesor es válido o no. Al usarlo encontramos un bug que producía que a veces no se encontrará el camino o bucles infinitos a recorrer las listas. Esto es debido a que puede ocurrir que no devuelva lo mismo dependiendo del orden o sentido en el que se le indiquen los vectores entre los que lanzar el rayo. Por ejemplo RayTrace(vector1, vector2) puede ser visible pero RayTrace(vector2, vector1) no. Para solucionarlo decidimos que para que fuera un sucesor válido debe ser visible en los dos sentidos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] A. Tovey, "Self-driving cars will add $7 trillion a year to global economy, says intel," 2017, [Online; accessed 8-June-2017]. [Online]. Available: http://www.telegraph.co.uk/business/2017/06/02/self-driving-cars-will-add-7-trillion-year-global-economy-says/

[2] I. Lunden, "Intel buys mobileye in $15.3b deal, moves its automotive unit to israel," 2017, [Online; accessed 8-June-2017]. [Online]. Available: https://techcrunch.com/2017/03/13/reports-intel-buying-mobileye-for-up-to-16b-to-expand-in-self-driving-tech/

[3] C. Farr, "The first study of self-driving car crash rates suggests they are safer," 2017, [Online; accessed 8-June-2017]. [Online]. Available: https://www.fastcompany.com/3055356/the-first-study-of-self-driving-car-crash-rates-suggests-they-are-safer/

[4] C. Thompson, "8 ways self-driving cars will drastically improve our lives," 2016, [Online; accessed 8-June-2017]. [Online]. Available: http://www.businessinsider.com/how-driverless-cars-will-change-lives-2016-12/

[5] H. Tulleken, "Bézier curves for your games: A tutorial," [Online; accessed 6-June-2017]. [Online]. Available: http://devmag.org.za/2011/04/05/bzier-curves-a-tutorial/

[6] Wikipedia, "Bézier curve — wikipedia, the free encyclopedia," 2017, [Online; accessed 6-June-2017]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=B%C3%A9zier_curve&oldid=783711812

[7] H. Tulleken, "Bézier curves for your games: A tutorial. quadratic curve," [Online; accessed 6-June-2017]. [Online]. Available: http://devmag.org.za/blog/wp-content/uploads/2011/04/bezier_2.png

[8] Wikipedia, "Construction of a quadratic bézier curve," 2017, [Online; accessed 6-June-2017]. [Online]. Available: https://en.wikipedia.org/wiki/B%C3%A9zier_curve#/media/File:B%C3%A9zier_2_big.svg

[9] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," pp. 533–579, 2010.

[10] A. Nash, "Theta*: Any-angle path planning for smoother trajectories in continuous environments," [Online; accessed 6-June-2017]. [Online]. Available: http://aigamedev.com/open/tutorials/theta-star-any-angle-paths/

[11] ——, "Theta*: Any-angle path planning for smoother trajectories in continuous environments. figure 6 comparison theta* and a*," [Online; accessed 6-June-2017]. [Online]. Available: http://aigamedev.com/static/tutorials/aap-path12.png