UNIVERSITAT POLITÈCNICA DE CATALUNYA

Máster:

AUTOMÀTICA I ROBÒTICA

Tesi de Master

SISTEMA DE GUIADO ASISTIDO PARA LA EJECUCION DE TAREASVIRTUALES CON DISPOSITIVOS HAPTICOS.

Luis Alejandro Chacón Encalada

Director: Jan Rosell Gratacòs

Curs Academic 2009/2010

Febrero 2010

RESUMEN

El empleo de dispositivos hápticos en sistemas de teleoperación, permite al operador ejecutar las tareas con mayor efectividad, esto junto al desarrollo de nuevas tecnologías logra que la eficiencia en las actividades de teleoperación sea cada día mayor.

En este proyecto se establece una arquitectura para el desarrollo de aplicaciones de guiado háptico para ayuda a la teleoperación o ejecución de tareas virtuales, basada en una sola librería (H3D) con la cual se pueden realizar las tareas de representación gráfica en tres dimensiones, manejo de colisiones, representación del dispositivo háptico y creación de la interfaz gráfica para el usuario. Esta arquitectura es de software libre y multiplataforma.

Empleando esta arquitectura se ha implementado una aplicación, para realizar la planificación de una ruta a seguir desde un punto inicial hacia un punto final, dentro de un espacio virtual tridimensional. La ruta es seguida mediante el dispositivo háptico y haciendo uso de las funciones de fuerza de éste, se logra hacer que el usuario se mantenga dentro de la ruta sugerida. Esta aplicación constituye un módulo de asistencia al operador en las tareas de teleoperación o tareas virtuales, que le permitirá simular el recorrido de una ruta para la ejecución de una tarea.

En la planificación de la ruta se emplea el método de los árboles de exploración rápida, y se implementan básicamente dos de sus algoritmos, la aplicación presenta de forma gráfica los arboles empleados y muestra el camino a seguir.

Para la interacción entre el usuario y la aplicación a través del dispositivo háptico, se crea un nodo adicional que permite usar el dispositivo háptico para mover el móvil en el espacio virtual.

Se propone una variación al algoritmo de planificación basado en los árboles de exploración rápida, para permitir al usuario establecer a través del dispositivo háptico un conjunto de puntos por los cuales se pretende pase la ruta calculada.

En el final de proyecto se sugieren las mejoras posibles, y las recomendaciones para crear nuevas aplicaciones más eficientes y útiles para el operador

MOTIVACIÓN

El uso de dispositivos hápticos, en las tareas de teleoperación es un campo en desarrollo constante, día a día aparecen nuevas opciones, necesidades y nuevas tecnologías para permitir al operador tener una mayor participación sensorial en las actividades de teleoperación. Esto motiva a seguir explorando posibles soluciones a diferentes problemas asociados a la teleoperación.

La arquitectura actual de desarrollo de aplicaciones en el Instituto de Organización y Control, (IOC) requiere de integrar diferentes librerías de distintos fabricantes, lo cual demanda esfuerzo y tiempo de desarrollo para la integración; esto crea la necesidad de buscar una arquitectura donde se puedan integrar todas las librerías en un solo sistema, sin recurrir a soluciones particulares de terceros, tener un código más sencillo, y con menos tiempo de desarrollo, la figura 1.1 muestra la propuesta de un entorno de desarrollo integrado.

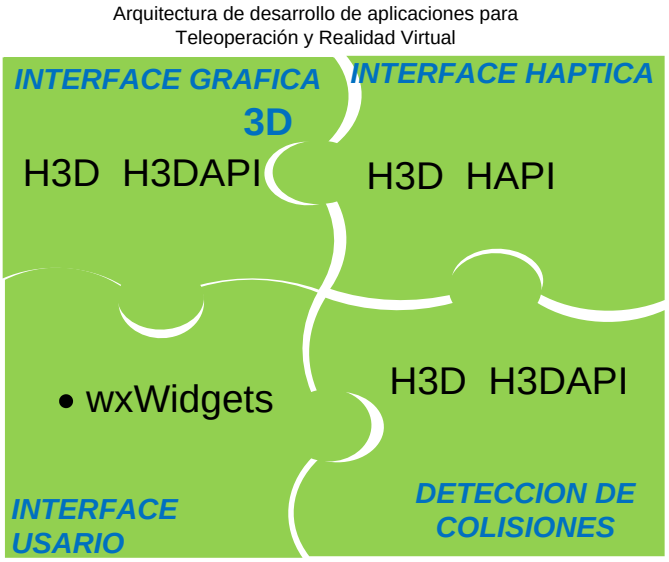


Figura 1. 1. Entorno de desarrollo integrado usando H3D

Por otro lado en el sistema de teleoperación del IOC se han desarrollado módulos de ayuda para la teleoperación, dentro de estos falta un módulo de asistencia de guiado en tareas virtuales, este módulo permitiría al operador simular y entrenarse al realizar las tareas de teleoperación. Y también ayudaría en el guiado de la tarea, con lo cual el operario podrá realizar la tarea con mayor rapidez, en menor tiempo y con precisión.

Este módulo tiene como objetivo permitir al operador seguir una trayectoria de movimiento calculada por el planificador, y ser asistido en el movimiento para evitar al operador salirse de las restricciones del movimiento.

Consta básicamente:

- **Módulo de simulación:** Este módulo es el responsable de mantener el modelo de la estación y usarlo para detectar posibles colisiones entre el entorno, el robot y las piezas manipuladas y retroalimentar la fuerza correspondiente. Esto permite al operador reaccionar rápidamente ante posibles situaciones de colisión.
- **Módulo de planificación:** Este módulo de planificación es el encargado de calcular la trayectoria a seguir por el operador, desde una posición inicial hasta una posición meta, manteniendo las restricciones especificadas por el operador. El planificador incluye un lazo de realimentación de posición y fuerza para permitir la ejecución segura de movimientos con el guiado por parte del sistema al operador.

OBJETIVOS

Implementar un módulo de guiado asistido para la ejecución de tareas virtuales con dispositivo hápticos, como un módulo dentro de los módulos de ayuda para el sistema de Teleoperación del IOC. Este permitirá guiar al operador en el seguimiento de una trayectoria planificada por el sistema entre dos ubicaciones distintas en el espacio libre de colisiones.

Objetivos Específicos.

- Probar y usar el conjunto de librerías de la biblioteca H3D, para tener un sistema de desarrollo de aplicaciones, manteniendo la filosofía del uso de software libre, multiplataforma, simplificando el desarrollo de aplicaciones de teleoperación.
- Implementar un planificador incremental, basado en el algoritmo RRT, para generar trayectorias en el espacio 3D, a ser seguidas por un robot.
- Desarrollar un interface de usuario, para crear espacios tridimensionales, aplicable a espacios virtuales, empleando un estándar portable y genérico.
- Mantener para el desarrollo de este módulo, la programación orientada a objetos y el lenguaje de programación C++.

1 INTRODUCCION

Hoy en día vivimos la época donde se avanza rápidamente con nuevos desarrollos tecnológicos para mejorar las condiciones del ser humano, extendiendo sus capacidades o desarrollando las ya existentes.

Las exigencias día a día crecen más, se puede hablar de actividades donde se requiere realizar operaciones con el operador ubicado a cierta distancia, por ejemplo en la desactivación de bombas, o en la exploración de otros planetas, las aplicaciones crecen en función de los avances tecnológicos.

Gracias a estos desarrollos se ha llegado finalmente a lo que se conoce como sistemas de teleoperación maestro-esclavo, en los cuales un manipulador denominado esclavo reproduce fielmente los movimientos de un dispositivo o manipulador maestro, controlado a su vez manualmente por un operador humano. Es entonces cuando la teleoperación cobra importancia como tecnología básicamente para describir la capacidad del ser humano de hacer actividades a distancia.

La Teleoperación significa "hacer el trabajo a distancia", aunque "trabajo" puede significar casi cualquier cosa. Además, el término "distancia" es vago: puede referirse a una distancia física, donde el operador está separado del robot por una larga distancia, pero puede también referirse a un cambio de escala, por ejemplo en cirugía robótica un cirujano puede usar tecnología de micro-manipulación para dirigir cirugías a nivel microscópico.

En los sistemas de teleoperación de robots, actualmente llamados sistemas tele-robóticos, la intervención del operador humano muchas veces es imprescindible, especialmente en entornos no estructurados y dinámicos en los cuales los problemas de percepción y planificación automática son muy complejos.

La telerobótica puede considerarse como una forma evolucionada de la teleoperación, caracterizada por el aumento de la autonomía en el sistema remoto manteniendo una intervención significativa del operador humano para supervisión o teleoperación directa [1].

Por la misma esencia de la telerobótica en la que no se tiene acceso al entorno donde se encuentra el robot, se requiere que el sistema que funciona como controlador proporcione la mayor cantidad de retroalimentación sensorial al operador. Para cumplir con este objetivo, se tiene en la actualidad tecnologías para implementar módulos de ayuda en las tareas de la teleoperación.

Los módulos de ayuda [2] permiten implementar una teleoperación bilateral, donde se puede obtener información desde el robot hacia el operador, esta información puede ser visual, o de realimentación de fuerzas. Estos pueden ser:

- Módulos de posicionamiento relacional

- Módulos de realidad aumentada.
- Módulos de simulación.
- Módulos de Supervisión.

Este proyecto se enmarca dentro de los sistemas telerobóticos, y pretende desarrollar un módulo de ayuda, para asistir al usuario en la teleoperación.

1.1 Marco Teórico.

El objetivo de este marco es presentar una descripción básica de algunos conceptos empleados en este proyecto, revisando temas de teleoperación, realidad virtual, dispositivos hápticos, y arquitecturas existentes para poder realizar la implementación de aplicaciones que integran estas tecnologías.

1.1.1 Teleoperación.

Conjunto de tecnologías que comprenden la operación o gobierno a distancia de un dispositivo por un ser humano. Por tanto, teleoperar es la acción que realiza un ser humano de operar o gobernar a distancia un dispositivo; mientras que un sistema de teleoperación será aquel que permita teleoperar un dispositivo, que se denominará dispositivo teleoperado.

Dentro de la teleoperación se han creado actualmente nuevas disciplinas, tales como:

- ***Telemanipulación****:* conjunto de tecnologías que comprenden la operación o gobierno a distancia por un ser humano de un manipulador. Por tanto, telemanipular es la acción que realiza un ser humano de operar o gobernar a distancia un manipulador, mientras que un sistema de telemanipulación será aquel que permita teleoperar un manipulador, que se denominará manipulador teleoperado.
- ***Telerobótica:*** conjunto de tecnologías que comprenden la monitorización y reprogramación a distancia de un robot por un ser humano. Se hablará entonces de la teleoperación de un robot, que se denominará telerobot o robot teleoperado.
- ***Telepresencia:*** situación o circunstancia que se da cuando un ser humano tiene la sensación de encontrarse físicamente en el lugar remoto. La telepresencia se consigue realimentando coherentemente al ser humano suficiente cantidad de información sobre el entorno remoto.

1.1.2 Realidad virtual.

Situación o circunstancia que se da cuando un ser humano tiene la sensación de encontrarse en un lugar distinto de donde físicamente está gracias a la información generada exclusivamente por un computador. El entorno que se genera, y en el que el operador se encuentra inmerso se denomina entorno virtual, y la situación de estar en él, también se conoce como presencia virtual.

La realidad virtual puede ser de dos tipos: inmersiva y no inmersiva. Los métodos inmersivos de realidad virtual con frecuencia se ligan a un ambiente tridimensional creado por computadora el cual se manipula a través de cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano. La realidad virtual no inmersiva utiliza medios como los ofrecidos por el Internet en el cual se puede interactuar en tiempo real con diferentes personas en espacios y ambientes no reales sin la necesidad de dispositivos adicionales a la computadora

Los ambientes virtuales inmersivos son espacios tridimensionales, reales o imaginarios, generados por computadora, con los cuales el usuario puede interactuar y le producen la sensación de estar dentro de un ambiente o lugar. La sensación de presencia se genera cuando se integran varios elementos, como son una rápida generación de varias imágenes de alta calidad por segundo, desplegadas en un área que cubra un amplio grado de campo de visión del usuario, y que resultan cuando el usuario interactúa al moverse o modificar el espacio y sonido espacial relacionado con el ambiente al que se da vida. Para que la interacción en estos ambientes sea de la forma más natural posible, se recurre al uso de dispositivos especiales que nos permiten una manipulación natural con el ambiente, como pueden ser el uso de guantes, sistemas de rastreo de movimiento, o interfaces de entrada muy específicos, estrechamente vinculados con el ambiente en que se trabaja. Por ejemplo, en simuladores de manejo se utilizan volantes y palancas de velocidades, o en el caso de simulación para cirugía se utilizan simuladores de los instrumentos, según la cirugía en cuestión.

Una de las ventajas de poder utilizar ambientes virtuales inmersivos es acceder a espacios inaccesibles o con riesgo, y poder modificar los eventos que ahí ocurren. Recrear ambientes para entrenamiento que serían muy costosos o no son posibles,

Los ambientes virtuales inmersivos, dependiendo de la aplicación a la que estén dirigidas, requieren de alguno o varios de los siguientes elementos:

- **Cargadores de escenas 3D** Estos son componentes de software que permiten leer desde uno, o varios archivos, en diversos formatos la escena 3D o las partes que forman la escena 3D, como son: geometrías, imágenes, personajes, sonidos, etc.
- **Diversas formas de navegación** Las formas de navegación son las opciones con las que el usuario cuenta para mover los objetos virtuales, inspeccionarlos o moverse a través de la escena. Algunas de estas formas de navegación son, por ejemplo: el moverse alrededor del objeto, ubicado en el centro de su visión, "volar libremente" en una escena 3D o simular que el usuario camina en la escena 3D.
- **Manejo de colisiones** Dependiendo de las formas de navegación, se puede requerir atravesar cualquier objeto para la navegación libre o que se detectan las colisiones con los objetos para simular que el usuario camina por la escena 3D.

- **Animación de objetos** Las escenas 3D pueden contener elementos no estáticos, como objetos animados desde dentro de la aplicación o animaciones creadas con algún software de animación.
- **Integración de personajes** Se pueden agregar personajes a las escenas 3D, ya sea para tener un avatar que nos guíe a través del mundo virtual o para incrementar el realismo de la escena, al tener muchos personajes que pasean en ella.
- **Programación de despliegue en espacios envolventes** El despliegue de la aplicación de realidad virtual inmersiva se puede realizar en cascos, caves (seis pantallas que forman un cubo dentro del cual está el usuario), pantallas curvas, etc.; los cuales dan la sensación de que la escena 3D envuelve al usuario.
- **Integración de interfaces de interacción** Para interactuar con las escenas 3D, existen diversos dispositivos que permiten manipular los objetos virtuales. Por ejemplo, guantes electrónicos para "tomar" moléculas virtuales o los complejos brazos electro-mecánicos (llamados dispositivos hápticos) para la simulación de cirugías.

1.1.3 Interface háptico.

Con el término "interface háptico" se conoce a aquellos dispositivos mediante los cuales se permite al usuario tocar, sentir o manipular objetos simulados en entornos virtuales y sistemas teleoperados.

La importancia de los interfaces hápticos es determinante en la realización de tareas típicamente "hápticas", o en las que se requiera un alto grado de entrenamiento, como pueden ser: administración de anestesia epidural, palpado de bultos cancerígenos, ensamblaje de conjuntos complejos antes de ser fabricados, etc. Ayudan a su vez, a incrementar la sensación de presencia o inmersión del usuario dentro de un entorno simulado, proporcionando restricciones naturales al movimiento de objetos.

Según [3] los interfaces hápticos pueden clasificarse en tres grandes grupos, según proporcionen:

1. Realimentación de fuerza.
2. Realimentación táctil.
3. Realimentación propioreceptivo.

Cada uno de ellos aportará al usuario información referente a un determinado campo, siendo clave la selección del tipo de interface que necesitaremos, en función de las características que deseamos controlar en nuestra aplicación.

Los interfaces con realimentación de fuerza aportan datos relacionados con la dureza, peso e inercia del objeto virtual.

Los interfaces con realimentación táctil permiten adquirir datos tales como la geometría del objeto virtual, su rugosidad y temperatura, entre otros.

Por último, los interfaces con realimentación propioreceptivo dan información acerca de la posición del cuerpo del usuario o su postura.

A la hora de seleccionar un interface háptico para una determinada aplicación, se debe distinguir, en una primera etapa, el tipo de realimentación deseado, pudiendo elegir entre:

- Interfaces que proporcionen una realimentación de fuerza,
  - ✓ **Dispositivos escritorio.-** Permiten la interacción puntual con el objeto virtual a través de un terminal, materializado como un lápiz, dedo virtual o un joystick.
  - ✓ **Guantes.-** Permiten la manipulación diestra (en múltiples puntos de contacto) de objetos virtuales con retorno de fuerza.
- Interfaces con realimentación táctil.
  - ✓ **Dispositivos de escritorio.-** Permiten al usuario sentir o percibir la dureza de una superficie, su rugosidad, seguir contornos lisos, o materiales elásticos en 2 dimensiones.
  - ✓ **Guantes.-** que permiten simular con libertad de movimiento el contacto en múltiples puntos con el objeto virtual, individuando su textura, pero no características tales como su peso.

En una segunda fase, se debe analizar las características técnicas de cada uno de los interfaces comerciales disponibles en el mercado, para seleccionar el más apropiado para el objetivo. Las principales especificaciones técnicas a considerar son:

- **Número de grados de libertad** del dispositivo que requiera la aplicación.
- **Espacio de trabajo** es decir, la extensión del volumen dentro del cual el manipulador puede posicionar el elemento terminal.
- **Rango de control de fuerza** o lo que es lo mismo, los niveles máximo y sostenido de fuerza que puede ejercer el dispositivo. Según [3] los dedos de un humano pueden ejercer de 30 a 50 N de fuerza en periodos breves de tiempo y de 4 a 7 N en periodos sostenidos. Para que el confort del usuario u operador esté dentro de unos niveles de seguridad admisibles, las fuerzas ejercidas por el interface deben ser inferiores en un 15% a los valores máximos anteriormente citados.
- **Fricción aparente**: las pérdidas por fricción en un interface háptico deben ser inferiores a la mínima fuerza o par que podamos percibir mientras interactuamos con el entorno virtual, ya que en caso contrario nuestro interface dejaría de ser "transparente", pues no podríamos diferenciar si las fuerzas percibidas por el usuario provienen de la realimentación deseada o de las pérdidas mecánicas del dispositivo en sí. Los valores de fricción aparente deben mantenerse en valores por debajo del

7% de las fuerzas y del 12,7 % de los pares de fuerza aplicados en la interacción con el entorno virtual.

- **Rigidez** de un interface háptico se relaciona íntimamente con la habilidad del mismo para generar restricciones al movimiento del operador dentro del entorno virtual, impidiendo que se penetre dentro de los distintos sólidos virtuales, y permitiendo, de este modo, su inspección y manipulación.

  La rigidez máxima de un interface depende de la fuerza máxima que pueda desarrollar y del mínimo desplazamiento que detecte (N/m). Según [4], en la práctica, un interface debe proporcionar una rigidez mínima de 20 N/m. para que el operador pueda recorrer adecuadamente una superficie virtual.

- **Inercia aparente.** Es la masa mínima percibida por el operador cuando mueve el interface háptico a través del espacio libre. En el caso de que la inercia aparente del dispositivo fuese demasiado alta, el operador podría fatigarse en exceso. Un estudio piloto [Ellis entre ot. 1996] indicaba que eran aceptables masas de 50g para operaciones que duraran media hora o menos, si bien estudios recientes recomiendan valores cercanos a 100g.
- **Back-driveability** con este término se alude a la transparencia del interface, en tanto que no debe ejercerse ninguna fuerza sobre la mano del usuario mientras no exista interacción física con el entorno virtual. Esta capacidad del sistema para seguir el movimiento de la mano del usuario rápidamente y sin oposición recibe el nombre de *back-driveability*.
- **Rango dinámico**: es el ratio entre el máximo valor de salida del actuador frente a la fricción del mecanismo.
- **Ancho de banda**. Puede considerarse como una medida de su calidad, cuanto mayor sea éste, menores serán los retardos en la transmisión de información, aumentando la estabilidad general del sistema.

La interacción háptica es un área de investigación relativamente nueva, dentro de la que varios grupos están desarrollando dispositivos que permiten al usuario la posibilidad de interacción física con un medio virtual o remoto.

En este proyecto se emplea el háptico Phantom Premiun de Sensable (http://www.sensable.com/haptic-phantom-premium.htm), un dispositivo háptico de 1.5/6.0 DOF(grados de libertad), permite explorar aplicaciones en áreas que requieran realimentación de fuerza en 6 grados de libertad. Aplicaciones incluyen ensamble virtual, prototipado virtual, mantenimiento en planificación de caminos, teleoperación y modelamiento molecular, este háptico se conecta al computador vía puerto paralelo, y es compatible con plataformas Windows® 2000/XP, RedHat, Linux, RedHat Fedora ™ y SUSE 9.0.

	PHANTOM Premium 1.5/6DOF		PHANTOM Premium 1.5 High Force/6DOF	
Workspace	Translational	15 W x 10.5 H x 7.5 D inches 381 W x 267 H x 191 D mm	Translational	15 W x 10.5 H x 7.5 D inches 381 W x 267 H x 191 D mm
	Rotational		Rotational	
	Yaw	297 degrees / 5.18 radians	Yaw	297 degrees / 5.18 radians
	Pitch	260 degrees / 4.54 radians	Pitch	260 degrees / 4.54 radians
	Roll	335 degrees / 5.85 radians	Roll	335 degrees / 5.85 radians
Footprint	13 W x 10 D inches / 330 W x 254 D mm		13 W x 10 D inches / 330 W x 254 D mm	
Range of motion	Lower arm movement pivoting at elbow		Lower arm movement pivoting at elbow	
Nominal position resolution	Translational	860 dpi / 0.03mm	Translational	3784 dpi / 0.007 mm
	Rotational		Rotational	
	Yaw & Pitch	0.0023 degrees 0.00004 radians	Yaw & Pitch	0.0023 degrees 0.00004 radians
	Roll	0.0080 degrees 0.00014 radians	Roll	0.0080 degrees 0.00014 radians
Backdrive friction (x, y, z)	0.15 oz / 0.04 N		0.75 oz / 0.2 N	
Maximum exertable force and torque at nominal position (orthogonal arms)	Translational	1.9 lbf / 8.5 N	Translational	8.4lbf / 37.5N
	Rotational		Rotational	
	Yaw & Pitch	73 oz-in / 515 mNm	Yaw & Pitch	73 oz-in / 515 mNm
	Roll	24 oz-in / 170 mNm	Roll	24 oz-in / 170 mNm
Continuous exertable force and torque at nominal position (orthogonal arms)	Translational	0.3 lbf / 1.4 N	Translational	1.4 lbf / 6.2N
	Rotational		Rotational	
	Yaw & Pitch	27 oz-in / 188 mNm	Yaw & Pitch	27 oz-in / 188 mNm
	Roll	7 oz-in / 48 mNm	Roll	7 oz-in / 48 mNm
Stiffness	20 lbf in^{-1} 3.5 N mm^{-1}		20 lbf in^{-1} 3.5 N mm^{-1}	
Inertia (apparent mass at tip)	< 0.30 lbm < 136 g		< 0.46 lbm < 210 g	
Weight (device only)	~ 20 lb ~ 9 Kg		~ 20 lb ~ 9 Kg	
Force feedback	x, y, z, Tx, Ty, Tz		x, y, z, Tx, Ty, Tz	
Position sensing	x, y, z, roll, pitch, yaw		x, y, z, roll, pitch, yaw	
Interface	Parallel port		Parallel port	
Supported Platforms	Intel-based PCs		Intel-based PCs	
GHOST® SDK Compatibility	Yes		Upon special request	
OpenHaptics™ Toolkit Compatibility	Yes		Yes	

Figura 1. 2. Hoja de especificaciones de los dispositivos hápticos en el IOC.

1.1.4 Sistema de teleoperación del Instituto de Organización y Control de Sistemas Industriales.

En el Instituto de Organización y Control de Sistemas Industriales (IOC) de la Universidad Politécnica de Cataluña, cuenta con un sistema de teleoperación, constituye prácticamente un sistema telerobot dentro del cual se desarrollan proyectos de investigación.

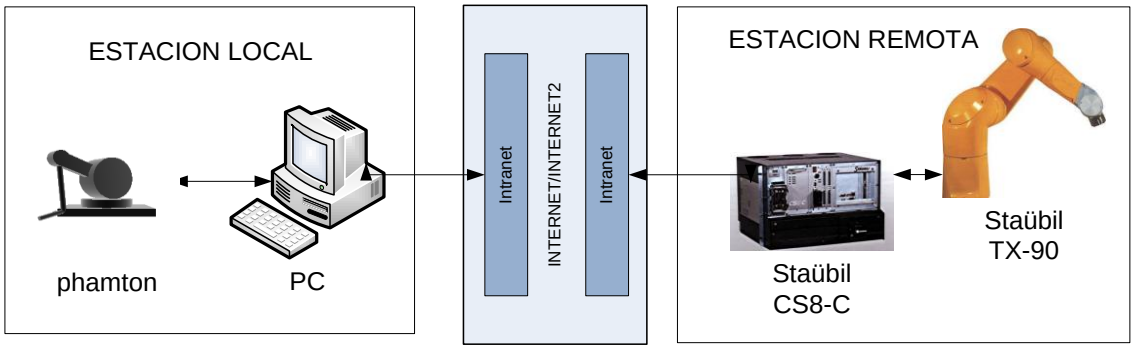


Figura 1. 3. Componentes físicos del sistema del IOC [2]

En la figura 1.3 se muestran los componentes físicos del sistema en el IOC; se puede encontrar:

- Una estación local, compuesta por un computador personal, un dispositivo háptico y un canal de comunicación.
- Una estación remota, constituida por un robot, el controlador y el canal de comunicación.

La estación local y remota se comunican vía internet.

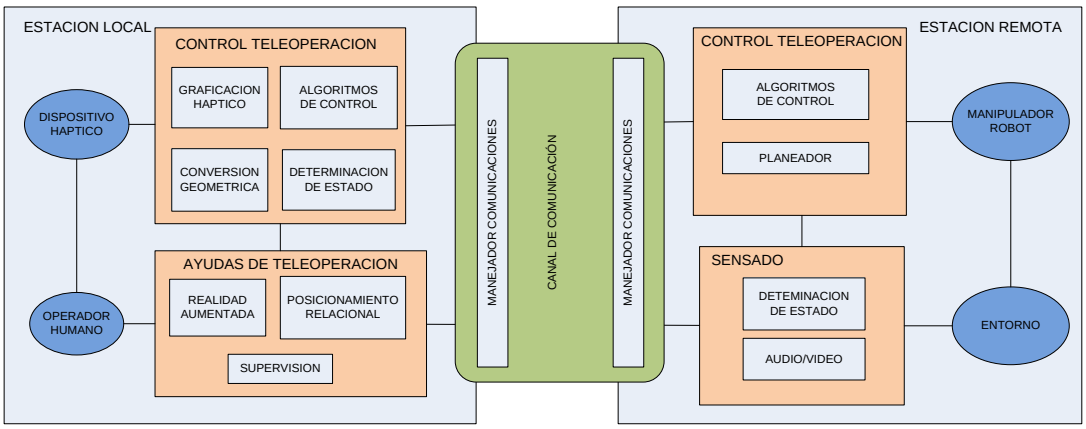


Figura 1. 4. Diagrama Lógico del sistema de teleoperación IOC [2]

En la figura 1.4 se tiene una representación del diagrama lógico del sistema del IOC, a nivel lógico, se tienen principalmente sistemas como:

- Algoritmos de control para estabilizar y acoplar el dispositivo háptico con el robot remoto, a través de un canal de comunicación.
- Módulo de posicionamiento relacional para localizar y generar restricciones en el dispositivo háptico.
- Un módulo de realidad aumentada para integrar un entorno real y virtual de información visual.

En la figura 1.4 se puede identificar el conjunto de herramientas "Ayudas de Teleoperación", actualmente se tiene un módulo de realidad aumentada, uno de posicionamiento relacional, y uno supervisión. Es aquí donde interesa añadir un nuevo módulo, como se muestra en la figura 1.5; el objetivo de este es dotar al operador de un herramienta de asistencia en el seguimiento de una trayectoria planificada, de esta forma el operador primero podrá simular en un entorno virtual la trayectoria a seguir, y la herramienta tratara de hacer sus movimientos guiados al camino a recorrer.

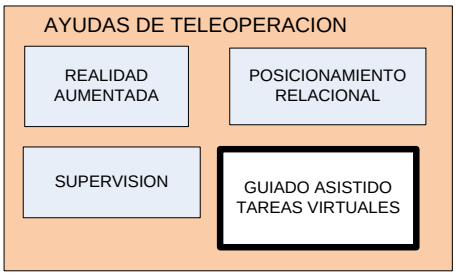


Figura 1. 5. Módulo de Ayudas de Teleoperación modificado.

1.1.5 Planificador de Caminos.

El problema de planificación de caminos consiste en llevar un objeto, desde una configuración inicial hasta otra final, dentro del espacio libre de colisiones.
Sobre este problema se encuentra amplia literatura, encontrando un gran número de métodos efectivos para resolver el problema de la planificación en tiempo real, tales como campos de potenciales, grafos de visibilidad, diagramas de Voronoi, etc. Sin embargo cuanto más complejo es el entorno, mayor número de grados de libertad, o más restricciones cinemáticas presenta el robot, mayores son las limitaciones de los métodos y el tiempo necesario para encontrar la solución.

Los métodos basados en muestreo son los más comúnmente utilizados en la resolución de problemas de planificación de movimientos. [5]
Estos métodos se basan en la generación de muestras libres de colisión en el espacio de configuraciones (C_{space}) y que en conjunto con las interconexiones, también libres, forman los mapas de carreteras o los árboles. Varios de los tópicos necesarios para el desarrollo de un planificador basado en el muestreo son del campo de la geometría computacional:

- la ubicación del objeto.
- la generación de muestras
- la determinación del estado de colisión o no de las muestras;
- la búsqueda de vecinos
- la representación en grafos y sus algoritmos de búsqueda.

Este conjunto de algoritmos captura la conectividad del espacio de configuración libre del robot en una estructura especial finita, un grafo, como en los métodos roadmap probabilístico (PRM) y en los árboles aleatorios de exploración rápida (RRT).

1.1.5.1 Arboles de exploración Rápida (RRT)

Este método es llamado "Rapidly Exploring Random Tress" RRT. El algoritmo del RRT es sencillo, rápido y de fácil extensión a escenarios complejos, asegura una exploración equiprobable de todo el espacio de configuraciones [6].

Un RRT es una estructura de datos y un algoritmo diseñado para buscar eficientemente en espacios no convexos de alta dimensión. Los RRTs se construyen gradualmente en forma tal, que explora uniformemente el espacio.
En la figura 1.6 se observa una representación del RRT, el árbol explora todo el espacio libre para determinar un posible camino.

Los RRTs son especialmente adecuados para problemas de planificación de rutas que implican obstáculos y limitaciones diferenciales (no holonómicas). Se pueden considerar como una técnica para generar trayectorias de bucle abierto para sistemas no lineales con restricciones de estado. Un RRT por intuición puede ser considerado como un modo de Monte Carlo de influir la búsqueda en las regiones Voronoi más grandes. Algunas variaciones pueden ser consideradas como fractales estocásticos.

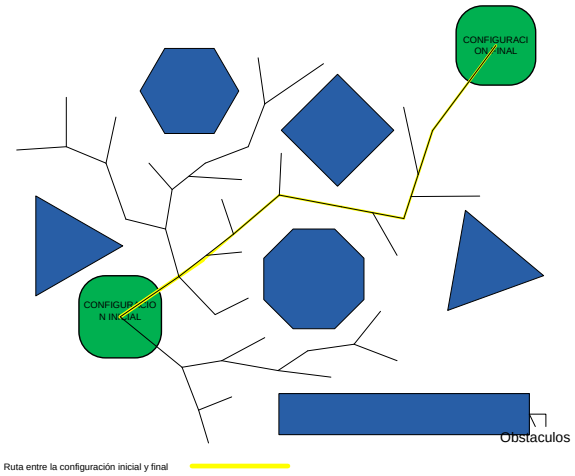


Figura 1. 6. RRT, crecimiento del árbol.

1.1.5.2 Método probabilístico de carreteras (PRM).

Método desarrollado en [7] cuyo objetivo es encontrar rutas libres de colisión para robots de cualquier tipo en espacios de trabajo estáticos, es decir ambientes donde los obstáculos no se muevan. El método consta de dos fases:

1. fase de construcción
2. fase de consulta.

En la fase de construcción se genera un mapa probabilístico de carreteras conocido como "roadmap". Primero se generan aleatoriamente *n* configuraciones libres para el robot y posteriormente se intentan conectar a través de un simple y rápido planeador local. El roadmap construido se encuentra en el espacio libre y se representa como un

grafo no dirigido. Las configuraciones generadas constituyen los nodos del grafo y las rutas calculadas por el planeador local sus aristas.

Una vez construido el roadmap se pueden realizar consultas, una consulta pregunta si existe una ruta entre dos configuraciones cualesquiera del robot. Para ejecutar una consulta el método intenta primero conectar las configuraciones inicial y final a nodos con los cuales hubo conexión par encontrar la secuencia de aristas que conectan a estos nodos.

Finalmente la concatenación de los segmentos sucesivos se transforma en una ruta factible para el robot como se muestra en la figura 1.7. Es decir la ruta está formada por tres subrutas, la subruta que conecta la configuración inicial con un nodo del grafo, la subruta presente en el grafo entre los dos nodos de conexión y la subruta que conecta un nodo del grafo con la configuración final.

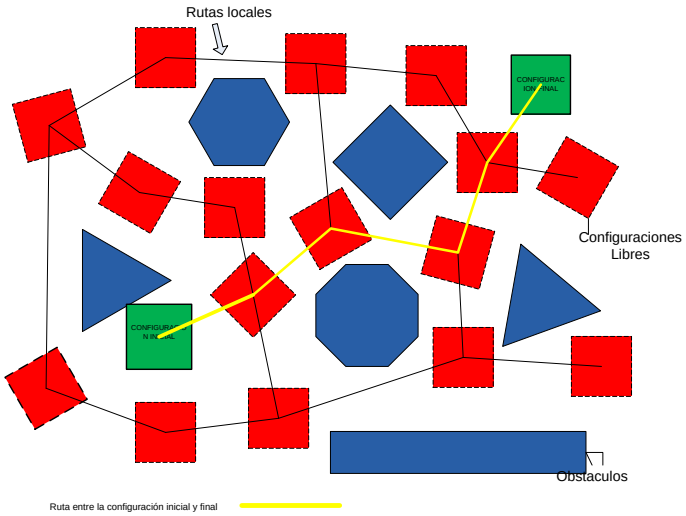


Figura 1. 7. Roadmap probabilístico, las configuraciones libres y las rutas locales se generan en la fase de construcción, y la ruta entre la configuración inicial y final, en la fase de consulta.

1.2 Arquitectura para desarrollo de aplicaciones de software (framework)

Para las aplicaciones implementadas en el IOC, en teleoperación y realidad virtual se requiere sean multiplataforma. Para desarrollar una aplicación, se establece usar un entorno de desarrollo integrado por diferentes librerías, con las cuales se puedan implementar los diferentes elementos de la aplicación, en la figura 1.8 se observa una arquitectura de desarrollo, y algunas opciones para implementarla.

Básicamente consta de 4 tipos de aplicación:

1. Interface gráfica.
2. Interface háptica.

3. Interface de usuario.
4. Detección de colisiones.

La filosofía de esta arquitectura es usar librerías de software libre, y la programación orientada a objetos, la herramienta de programación es C++.

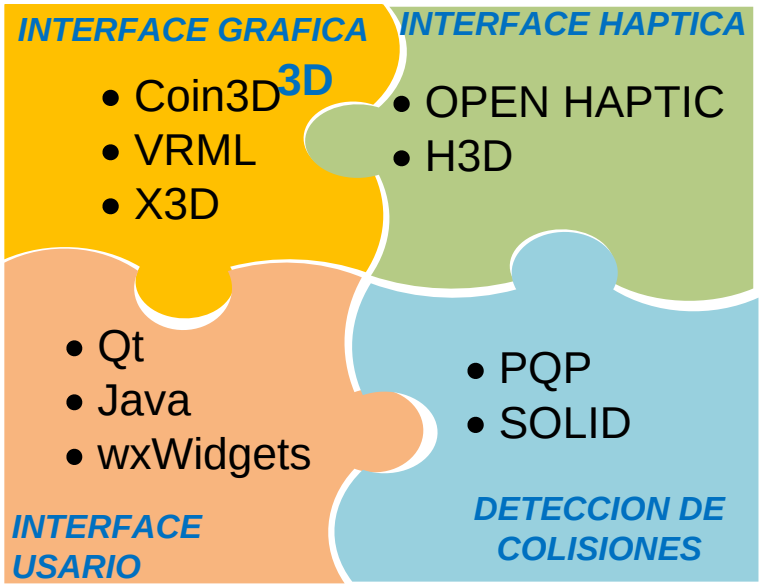


Figura 1. 8. Arquitectura para el Desarrollo de aplicaciones de Entornos Virtuales.

1.2.1 Interface Gráfica.

Este componente de la aplicación tiene el objetivo de realizar una representación de los objetos reales, en un escenario virtual. Su principal característica es permitir trabajar con las representaciones en un ambiente 3D, es decir tener las herramientas para desarrollar una aplicación en tres dimensiones.

En la figura 1.8 se aprecia algunas opciones para poder implementar este componente de la aplicación, las cuales se explican brevemente a continuación.

1.2.1.1 Lenguaje de Modelamiento de Realidad Virtual VRML (Virtual Reality Modelling Language).

VRML es un formato de archivo que permite la creación de objetos y mundos tridimensionales interactivos. El estándar VRML fue creado y desarrollado por el VRML Consortium [8].

VRML apareció en escena en 1994 y llegó a ser la primera tecnología reconocida oficialmente por la ISO (International Organization for Standarization) como estándar para la creación, distribución y representación de elementos 3D a través de Internet.

VRML fue diseñado para cumplir con los siguientes requerimientos básicos:

- Habilitar la posibilidad del desarrollo de programas para crear, editar y mantener archivos VRML, además de programas para la importación y exportación del formato
- VRML a otros formatos gráficos tridimensionales (Authorability).
- Aportar la capacidad de utilizar, combinar y reutilizar objetos dinámicos tridimensionales dentro de un mismo mundo VRML (Composability).
- Incorporar la capacidad de crear nuevos tipos de objetos no definidos específicamente como parte de VRML (Extensibility).
- Abrir la posibilidad de que sea implementado en una amplia variedad de sistemas presentes en el mercado (Implementable).
- Resaltar la importancia del funcionamiento interactivo en una amplia variedad de plataformas existentes (Performance).
- Permitir la creación de mundos tridimensionales de cualquier tamaño (Scalability).

VRML es un lenguaje jerárquico de marcas que usa Nodos, Eventos y Campos para modelar realidades virtuales tanto estáticas, como dinámicas. Los Nodos, que se usan para instanciar alguna de las 54 primitivas del lenguaje, no son más que una colección de Campos que contienen los atributos básicos de la primitiva.

Los Campos (Fields) son los atributos que definen el comportamiento de la primitiva, con la excepción los campos especiales (eventIn y eventOut) que permiten enviar y recibir eventos a otros Campos. Mediante estos Campos especiales y el comando ROUTE, se puede controlar el flujo de Eventos, encaminando el efecto de una acción entre múltiples objetos para animar una escena o simplemente pasar información a esos objetos.

Las primitivas del VRML se agrupan según función, en nueve colecciones distintas:

1. Agrupación de nodos
2. Grupos especiales
3. Sensores
4. Geometría
5. Propiedades de geometría
6. Apariencia de la geometría
7. Interpoladores
8. Nodos excluyentes
9. Nodos comunes

1.2.1.2 Extensible 3D (X3D)

X3D es un estándar abierto XML, un formato de archivo 3D que permite la creación y transmisión de datos 3D entre distintas aplicaciones y, especialmente, aplicaciones en red. Sus características son [9]:

- X3D está integrado en el lenguaje de marcas extensible *XML*; esto representa un paso fundamental a la hora de conseguir una correcta integración en:
  - ✓ Servicios Web.
  - ✓ Redes Distribuidas.
  - ✓ Sistemas multiplataforma y transferencia de archivos y datos entre aplicaciones.
- X3D es Modular (tiene componentes): esto permite la creación de un núcleo 3D más ligero ajustado a las necesidades de los desarrolladores.
- X3D es Extensible: permite añadir componentes para ampliar las funcionalidades según las necesidades del mercado.
- X3D es Perfilado: se pueden escoger distintos grupos de extensiones apropiadas según las necesidades específicas de la aplicación.
- X3D es Compatible con VRML: se mantiene el desarrollo, el contenido y la base de VRML97.

X3D emplea una arquitectura modular para dar una mayor extensibilidad y flexibilidad. La gran mayoría de las aplicaciones no necesitan de todo el poder de X3D, como tampoco el soporte para todas las plataformas y funcionalidades definidas en la especificación. Una ventaja de X3D es que se agrupa por componentes que pueden ser usados por las implementaciones para una plataforma definida o un mercado concreto.

1.2.1.3 Coin3D

Es un conjunto de herramientas de gráficos 3D de alto nivel, desarrollado en C++ y multiplataforma, ideado para ser usado en programas de visualización 3D en tiempo real. Coin3D se basa en OpenGL. Es considerada una capa de alto nivel para programación gráfica y render 3D. Así, código escrito mediante Coin3D puede coexistir con código OpenGL [10].

Las características de la librería son:

- Texturas 3D, multitextura.
- Librerías de importación/exportación para diversos formatos de archivo.
- Soporte de datos geográficos.
- Conjunto de primitivas (cajas, conos, cilindros, etc.).
- Render en tiempo real.
- Render de sombras.
- Soporte para multi-hilo.
- Render paralelo en múltiples procesadores.

1.2.2 Interfaces Hápticos.

Las librerías para hápticos nos permiten desarrollar aplicaciones de software para el manejo de los dispositivos hápticos.

1.2.2.1 OpenHaptics

El OpenHaptics® SDK [11] permite desarrollar aplicaciones de software para el manejo de los dispositivo hápticos Phantom® de SensAble Technologies, sobre el lenguaje de programación C++.

En la figura 1.9, se presenta la arquitectura de este toolkit; incluye el conjunto de librerías HDAPI (Haptic Device API), HLAPI (Haptic Library API), Utilidades, Controladores del Dispositivo (PHANTOM Device Drivers PDD) y ejemplos de códigos fuente.

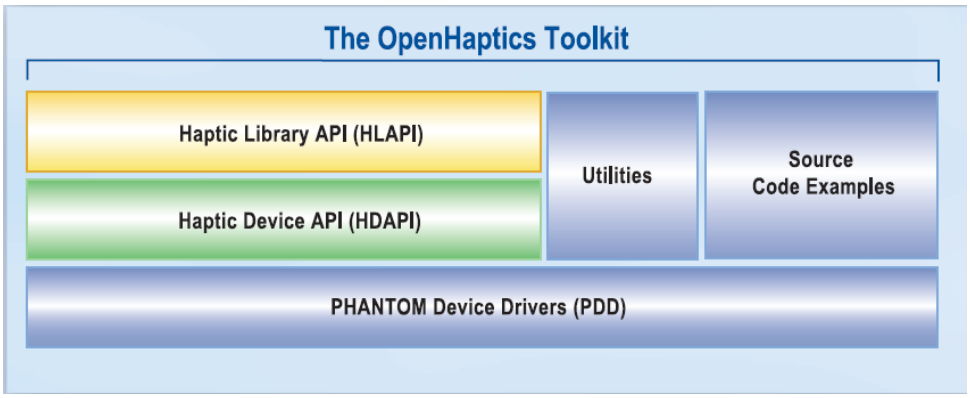


Figura 1. 9. Arquitectura del OpenHaptics ToolKit (www.sensable.com)

La librería HDAPI.- Permite además de otras utilidades, el acceso a bajo nivel al Phantom ofreciendo al programador la posibilidad de renderizar fuerzas directamente; configurar los controladores en tiempo de ejecución; leer el estado (posición, velocidad, fuerza aplicada), entre otras .A través de la HDAPI se puede integrar el dispositivo háptico en diferentes aplicaciones desarrolladas en C o C++, para la manipulación de sólidos generados a partir de imágenes de otros paquetes gráficos o de librerías especializadas que incluyan propiedades de deformación o colisión (por ejemplo VTK, Open Tissue, VCollide ,PQP, Solid, etc.).

Algunas de las posibilidades de programación de la arquitectura de HDAPI; ofrecidas con esta librería son:

- Leer el estado del dispositivo (Get Identification, Safety, Last Values Parameters):
- Configurar el estado del dispositivo (Set Forces, Safety Parameters):
- Planificador (Scheduler Priority Codes):

- Opciones de habilitación / deshabilitación (hdEnable, hdDisable Parameters):
- Manejador de errores (Device Error Codes)
- Funciones de calibración (Calibration Return Codes, Styles):

La librería HLAPI.- Haptic Library API proporciona una programación a alto nivel del Phantom. Incluye funciones de render háptico para aplicaciones gráficas desarrolladas con OpenGL, permitiendo especificar sobre primitivas geométricas tales como triángulos, líneas y puntos, con propiedades como rigidez y fricción, facilitando además la integración con entornos virtuales cuyo código ya ha sido implementado [12].

La arquitectura de HLAPI, a través de la cual se oculta al desarrollador de operaciones control del entorno: la sincronización entre de hilos háptico y grafico, detección de colisiones, cálculo y generación de fuerzas sobre un punto de contacto virtual (proxy).
Algunas de las posibilidades de programación ofrecidas con esta librería son:

- Render de fuerzas dependientes del movimiento; constante, modelo masa muelle, amortiguador, fricción, inercia, etc.
- Especificación de las propiedades físicas sobre los materiales superficiales: fricción, dureza, amortiguamiento, sensibilidad de todas las caras, etc.
- Definición y atención a gráficos y hápticos complementarios a la aplicación: para Windows incluye la posibilidad que el dispositivo se comporte como un ratón 2d; entradas digitales (interruptores del phamton), contactos virtuales (touch, untouch), etc.

1.2.2.2 H3D

La H3D [13]es una librería abierta para manejo de dispositivos hápticos, la plataforma de desarrollo de software usa los estándares abiertos OpenGL y X3D, con hápticos en un gráfico de escena unificado, para manejarse tanto en la parte gráfica como háptica. H3D es multiplataforma e independiente del dispositivo háptico. Esto permite la integración de audio así como gráficos estéreo.

Combina la parte gráfica y del háptico en una sola plataforma.

Añade dispositivos hápticos, a modelos 3D existentes.

- Permite un rápido desarrollo de aplicaciones con hápticos, usando X3D y Python.
- Fácilmente extensible en la parte gráfica y del háptico con características particulares usando C++.
- Soporta dispositivos hápticos SensAble, ForceDimension Novint, y MOOG FCS
- Soporta la mayoría de los sistemas de video estéreo 3D.
- Corre en todas las plataformas incluyendo Linux, Windows y Mac.

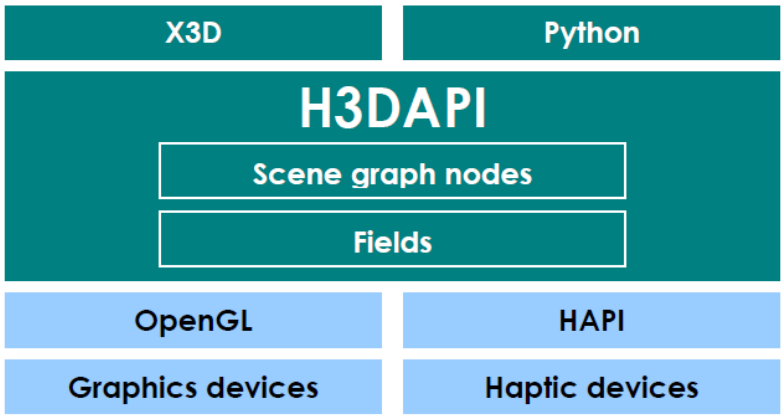


Figura 1. 10. Arquitectura del H3D

La figura 1.10 muestra la arquitectura de la librería H3D; Permite la entrada de información a través de archivos X3D, o código a través de Python. La H3DAPI, maneja la escena gráfica, y se comunica con el OpenGL para la visualización de la escena, en el dispositivo gráfico, La HAPI, se comunica con los drivers del dispositivo háptico.

La HAPI, comprende cuatro tipos de funciones:

1. Funciones para manejar el dispositivo, lo cual permite manejar diferentes hápticos.
2. Funciones para manejar geometrías basadas en háptico, consistente de tres subpartes, manejo de colisiones, algoritmo de render háptico, y algoritmo de interacción.
3. Funciones para manejar el espacio libre háptico, con diferentes efectos de fuerza.
4. Funciones para manejo del lazo háptico.

Todas las escenas H3DAPI, y la interacción con archivos X3D y scripts Python, pueden ser recreadas usando nodos H3D y librerías de campos. Gracias al poder del lenguaje de programación C++ permite personalizar y extender las habilidades del H3DAPI.

1.2.3 Interface gráfica de usuario

La interfaz gráfica de usuario, conocida también como **GUI** (*graphical user interface*) es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz [14].

La implementación de este componente tiene diferentes opciones, manteniendo la filosofía de usar software libre y multiplataforma, se tienen varias opciones.

1.2.3.1 Java

El API de Java 3D es un conjunto de clases para crear aplicaciones con elementos 3D. Ofrece a los desarrolladores la posibilidad de manipular geometrías complejas en tres dimensiones [15].

Ventajas

- La principal ventaja que presenta este API 3D, frente a otros entornos 3D, es que permite crear aplicaciones gráficas 3D independientes del tipo de sistema.
- Las librerías de Java permiten aprovechar la aceleración gráfica por hardware que incorporan muchas tarjetas gráficas.

Desventajas.

- Aunque es gratuito, el Kit de desarrollo de la plataforma Java (JDK) no es software libre.
- Java no está diseñado para construir aplicaciones rápidas, el hecho de correr bajo una máquina virtual, y utilizar en exceso herramientas como polimorfismo dinámico hace que las aplicaciones Java tiendan a ser lentas.

1.2.3.2 Qt

Otra alternativa a considerar es el Framework Qt, una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. En el cual se basa el escritorio de GNU/Linux KDE [16] .

Ventajas:

- Qt simplifica el desarrollo de aplicaciones para equipos desktop para ambos programadores, C++ y Java
- El hecho de tener una buena velocidad, debido a que está escrito en C++ y tener un muy buen diseño orientado a objetos es una ventaja Qt.
- El código Qt es más legible.

Desventajas.

- La aplicación estática más pequeña ocupa casi 4 MB y dinámica 10MB, esto lo hace consumir muchos recursos en aplicaciones pequeñas.
- Por el método de "*signal-slot*" usa un pre procesado MOC, lo cual lo hace no tener código C++ puro.

1.2.3.3 GTK

GTK es una biblioteca del equipo GTK+, la cual contiene los objetos y funciones para crear la interfaz gráfica de usuario [17].

Ventajas:

- Está disponible para muchísimos sistemas operativos y es completamente libre, con licencia LGPL.

Desventajas:

- En primer lugar, GTK+ solo posee funcionalidad para desarrollar interfaces gráficas de usuario y deja un lugar vacio a aspectos como bases de datos, redes, etc.
- El API de GTK+ está escrito de forma estructurada en lenguaje C, y la intención es trabajar de forma orientada a objetos. Aunque existe GTKmm con el mismo diseño estructurado pero usando clases en C++.

1.2.3.4 wxWidgets

wxWidgets [18], una herramienta para aplicaciones gráficas antes conocido como wxWindows, especializado en el desarrollo de aplicaciones multiplataforma en lenguaje C++.

Ventajas:

- Soporta varios sistemas operativos, pero con una pequeña particularidad: de acuerdo a la plataforma en que se ejecute la aplicación, esta tomara el "look and feel del SO" donde se esté ejecutando. Por ejemplo, si se ejecuta en MS-Windows, la aplicación tendrá la apariencia de una aplicación hecha con el API de Windows, pero si se ejecuta en GNU/Linux, tendrá la apariencia de una aplicación GTK2. Es decir que la parte de GUI es solo una capa para el API nativa de cada SO.
- wxWidgets cuenta con una versión embebida para dispositivos como PDAs y teléfonos celulares.
- Es cien por ciento libre en todas las plataformas que soporta, con una licencia LGPL modificada
- Tiene un API orientada a objetos que es, en extremo, fácil de aprender y utilizar.
- Posee funcionalidades para muchos aspectos aparte de la construcción de interfaces gráficas. Como gráficos 2D, 3D con OpenGL, Bases de Datos (ODBC), Redes, Impresión, Hilos, entre muchas otras.

Desventajas:

- Su diseño orientado a objetos no es el mejor que se pueda ver.
- Se abusa de la utilización de macros para realizar ciertas operaciones (como las tablas de eventos), aunque esto hace que codificar sea más fácil, también complica la labor de depuración, especialmente para los mismos desarrolladores de wxWidgets.

- Por otro lado, debido a que la herramienta se comenzó a desarrollar hace bastante tiempo, no cuenta con soporte para algunas características relativamente modernas del lenguaje C++ como por ejemplo el manejo de excepciones, y la STL.

1.2.4 Detector de colisiones.

La detección de colisiones es un conjunto de algoritmos matemáticos que permiten simular el comportamiento de los objetos en la realidad. Por ejemplo, que dos objetos no se intersequen, o que un personaje camine sobre el piso y luego pueda subir gradas. La detección de colisiones es un tema bastante complicado pero es el núcleo de cualquier aplicación de teleoperación o realidad virtual 3D.

En el mercado encontramos muchas librerías para poder detectar colisiones en 3D, tales como PQP, SOLID, OPCODE, ODE, I-COLLIDE; aquí revisamos dos de ellas.

1.2.4.1 Proximity Query Package (PQP).

PQP [19] este paquete fue desarrollado por la universidad de Carolina del Norte, y es libre para uso no comercial. Ejecuta una eficiente detección de colisión y cálculos de distancia para una colección de triángulos en un mundo 3D.

Características:

- Interface con C++
- Compatible con plataformas de Linux y Windows.
- PQP_REAL tipo definible por usuario para operaciones de punto flotante o para operaciones dobles.
- Soporta tres tipos de consultas de proximidad - la detección de colisión, el cómputo de distancia, y la verificación de tolerancia usando la misma estructura modela para todas las consultas.
- Fácil incorporarse a usos del cliente. Los programas del cliente deben incluir el archivo de cabecera "PQP.h" y vincularse a la biblioteca PQP.
- Ninguna restricción especial topológica o de la información de adyacencia son requeridas para modelos de entrada, solo son necesarios una lista de los triángulos del modelo.

1.2.4.2 SOLID.

Es una librería para la detección de colisión de objetos tridimensionales con movimiento cuerpo rígido y la deformación. SÓLID [20] es diseñado para ser usado en aplicaciones interactivas de gráficos 3D, y sobre todo eficiente para la detección de colisión de objetos en mundos descritos en VRML.

Características:

- Las formas de los objetos son representadas por formas primitivas (caja, cono, cilindro, esfera), y los complejos politopos (segmentos de recta, polígonos convexos, poliedros convexos). Una sola forma puede ser usada para instanciar múltiples objetos.
- El movimiento es especificado por traslaciones, rotaciones, y escalamientos no uniformes del sistema de coordenadas local de cada objeto en movimiento. Estos cambios pueden ser absolutos o relativos con respecto al marco anterior. El sistema de coordenada local también puede ser puesto según una serie de dieciséis valores float o dobles a representar en la matriz 4x4 principal, de la columna de una transformación afín, como es usado en OpenGL.
- Las deformaciones de formas complejas pueden ser especificadas usando arreglos de vértices definidos por cliente.
- La respuesta de colisión es definida por el cliente mediante funciones de llamada automática. La respuesta puede ser definida por par de objetos, para todos los pares que contienen un objeto específico, y como la falta para todos los pares de objetos.
- Respuesta de Call-backs pueden utilizar los datos que describen la configuración de la colisión de un par de objetos que colisionan.
- El marco de la coherencia es explotada por el mantenimiento de un conjunto de pares de objetos próximos (barrido progresivo y poda del eje ABB), y el almacenamiento en caché que separa los ejes de estos pares. Esta característica es opcional y puede activarse y desactivarse en cualquier momento durante la simulación.

1.2.5 Arquitectura de desarrollo en el IOC.

Para la realización de los módulos de programación en el IOC, se usan herramientas de software libre, estas son un conjunto de librerías de programación multiplataforma lo cual constituye el marco de programación de los sistemas como se muestra en la figura 1.11, este marco está conformado por [21].

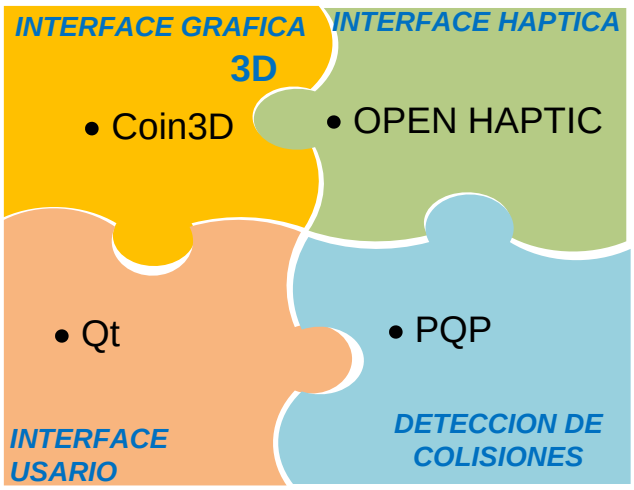


Figura 1. 11. Esquema de la Arquitectura usada en el IOC.

- **Coin3D**.-Para la representación gráfica de los modelos 3D
- **Qt .-** Para diseñar de forma gráfica la interfaz a través de "*drag and drop*".
- **PQP .-** Para la detección de colisiones,
- **OpenHaptics**.- Para manejo del dispositivo háptico.

Adicionalmente se usan:

- **Boost Graph** (http://www.boost.org/) Para manejar estructuras de datos tipo grafo.
- **XML** Para la descripción completa de una escena robótica. El archivo XML puede contener información tanto del robot como de los obstáculos (ubicación del archivo VRML o Inventor, nombres, escalas, posiciones y orientaciones), y cualquier otro aspecto relacionado con la definición del problema de planificación como las configuraciones inicial y final.

De acuerdo a lo establecido en los objetivos de este proyecto, se establece un arquitectura de desarrollado diferente a la usada hasta la fecha en el IOC, y está basada principalmente en el uso de la API H3D, como lo muestra la figura 1.12.

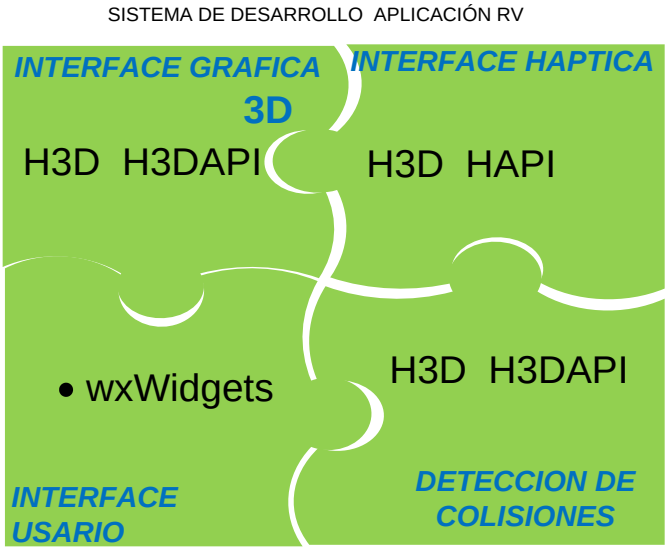


Figura 1. 12. Arquitectura de desarrollo basada en la API H3D.

2 PLANEACION DE MOVIMIENTOS.

La geometría y el movimiento son dos fenómenos omnipresentes en el mundo físico. Para operar en este mundo o simularlo necesitamos un modelo apropiado del ambiente, estructuras de datos compactas para representar los modelos y algoritmos eficientes para generar los movimientos de los diferentes tipos de objetos presentes en el mundo.

En muchas aplicaciones se requieren robots con la habilidad, de sensar y moverse sin colisionar con obstáculos existentes en un ambiente dado. La planificación de movimientos de robots, es una disciplina que trata con tales problemas; en un sentido limitado, el problema básico consiste en encontrar movimientos libres de colisión para los robots en un ambiente conocido poblado con obstáculos [22]. La planificación de movimientos tiene su origen en la robótica, considerada además una característica fundamental de los robots autónomos. Sin embargo, encontramos aplicaciones fuera de la robótica tradicional, en dominios tan dispersos como la animación por computadora, simulación de cirugías médicas y biología computacional, entre otros.

En su enfoque más simple, la planificación de movimientos es un problema esencialmente geométrico: dada una descripción geométrica de un objeto y un ambiente estático, nuestro objetivo es encontrar un camino sin colisión para que el objeto se mueva desde una configuración inicial hasta una configuración final. A esto se le llama el problema básico de la planificación de movimientos [22]. Una dificultad importante al desarrollar un algoritmo general para la planificación de movimientos (o un planificador de movimientos como frecuentemente se le conoce) es el gran número de grados de libertad (gdl) que puede tener un objeto. Desafortunadamente el algoritmo más rápido que cumple con la propiedad de completitud es exponencial con respecto al número de grados de libertad de un objeto en movimiento [23]. Además, el movimiento de los objetos a menudo está sujeto a restricciones físicas. Por ejemplo, un carro no puede moverse lateralmente.

Lozano-Pérez [24] introduce el concepto de espacio de configuraciones, en el cual el robot es tratado como un simple punto y el problema de planificación de movimientos se transforma, para encontrar un camino uni-dimensional en el espacio de configuraciones libre. El espacio de configuración (C-space) consiste en el conjunto de todas las posiciones y orientaciones que el robot puede tomar. Obtener el espacio de configuraciones es equivalente a que el robot recorra y se posicione en todos los puntos posibles del espacio de trabajo.

El robot se representa como un punto y los obstáculos son construidos dentro de él tal y como se muestra en la figura 2.1. El problema de planeación de movimientos para un objeto se convierte en un problema de planeación de movimientos para un punto.

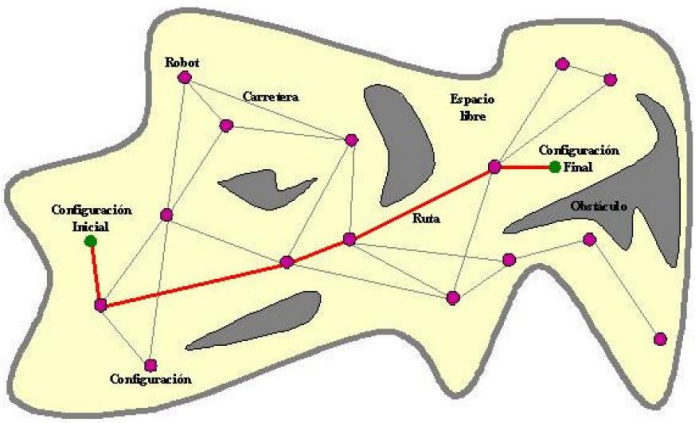


Figura 2. 1. Espacio de configuraciones

La ventaja principal que ofrece este espacio es que cualquier par de puntos que no pertenezcan a la región de los obstáculos representan dos configuraciones libres para el robot. Si además estos dos puntos pueden ser unidos por una línea (recta o curva) que pase por las regiones libres significa que existe un camino libre de colisión entre estas dos configuraciones.

Una limitación de los algoritmos de planificación de movimientos es la necesidad de adquirir con anterioridad el modelo completo del ambiente para que el planificador pueda actuar. Una solución a este problema es la planificación de movimientos usando sensores (PMCS) [25]. La PMCS permite a los robots trabajar de forma autónoma en ambientes desconocidos. Específicamente, el robot es capaz de moverse a una configuración dada sin un conocimiento anticipado del ambiente. La falta de este conocimiento previo es la diferencia principal entre la planificación de movimientos que usa sensores y la planificación de movimientos que usa modelos. La planificación de movimientos usando sensores no es, por consiguiente, un proceso fuera de línea, ya que, el movimiento se genera paso a paso mientras se acumula más y más información acerca del ambiente.

Debido a la dependencia exponencial de los algoritmos de planificación de movimientos con el número de grados de libertad se ha considerado al muestreo aleatorio como una técnica fundamental para atacar este problema. Sacrificando una cantidad limitada de completitud, el muestreo aleatorio mejora significativamente la eficiencia de los planificadores de movimientos, haciéndolos prácticos para un amplio rango de aplicaciones.

La planificación de caminos para espacios de configuración de altas dimensiones ha sido muy exitosa con métodos aleatorios [26] [27] [7] [28] [29]. Este conjunto de algoritmos captura la conectividad del espacio de configuración libre del robot en una estructura especial finita conocida como grafo, tal como en el algoritmo de hilo de Ariadna (ACA) [30], en los métodos roadmap probabilístico (PRM) ([7] [29]) y en los árboles aleatorios de exploración rápida (RRT) [31]

2.1 Árboles aleatorios de exploración rápida

En esta sección, se presenta el diseño y análisis de algoritmos de planificación de caminos basados en árboles aleatorios de exploración rápida (RRT), técnica desarrollada [6] en la universidad de Illinois, EUA. La base de estos métodos es la construcción incremental de árboles de búsqueda que intentan explorar rápida y uniformemente el espacio de estados, ofreciendo beneficios similares a los obtenidos por otros métodos exitosos de planificación aleatoria ([7], [29]).

Una descripción grafica del algoritmo para la creación de arboles aleatorios de exploración rápida (RRT) se muestra en la figura 2.2, este captura la conectividad del espacio de configuración libre del robot en una estructura grafo (árbol) finita [28]. El RRT original se basa en la construcción de un árbol de configuraciones que crece buscando a partir de un punto de origen, el objetivo es construir un árbol de exploración que cubriera uniformemente el espacio libre de colisión.

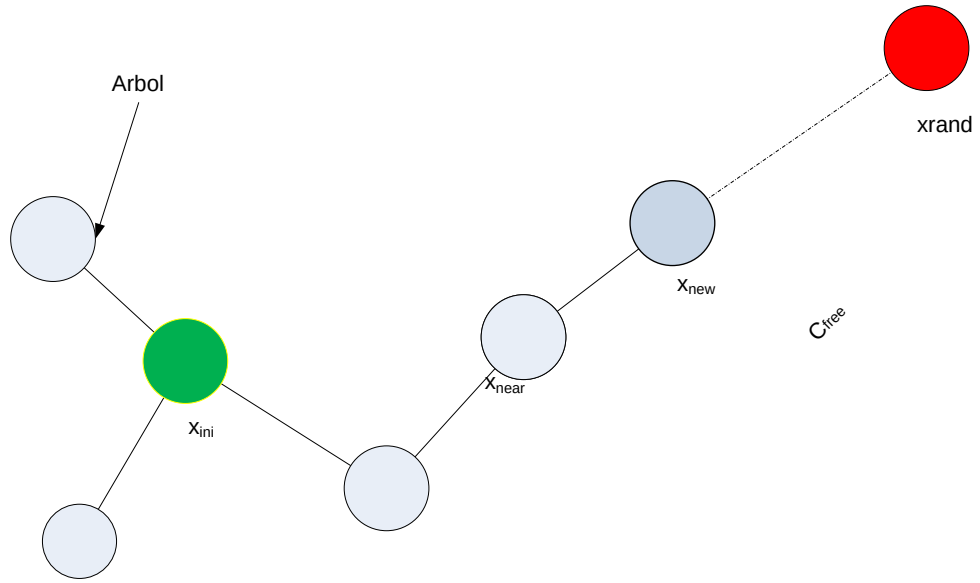


Figura 2. 2. Árbol de exploración rápida RRT

2.2 Formulación del problema

El tipo de problemas considerados por el enfoque RRT están formulados en términos de seis componentes:

1. **Espacio de estados**: Un espacio topológico, X.
2. **Valores límite**: $x_{init} \in X$ y $x_{goal} \in X$
3. **Detector de colisión:** Una función, D: X → {verdadero, falso}, que determina sí las restricciones globales son satisfechas desde el estado x. Esta puede ser una función binaria o real.
4. **Entradas**: Un conjunto, U, que especifica el conjunto de controles o acciones que pueden afectar al estado.

5. **Simulador incremental:** Dado el actual estado, x (t) y las entradas aplicadas sobre un intervalo de tiempo, {u (t′) t ≤ t′ ≤ t + Δt} , calcular x (t + Δt).
6. **Métrica**: Una función real, ρ: X × X →[0,∞) , la cual especifica la distancia entre pares de puntos en X.

La planificación de caminos generalmente es vista como una búsqueda en el espacio de estados, X, para un camino continúo desde una configuración inicial, $\mathbf{x}_{ini}$ hacia una configuración final $\mathbf{x}_{meta} \in$ X. Se asume que se tiene un conjunto de restricciones diferenciales sobre X y cualquier camino solución debe mantener al estado dentro de este conjunto. Un detector de colisión reporta si para una configuración $\mathbf{x}_i$, entra en colisión o no. Generalmente, se utiliza la notación, $\mathbf{x}_{free}$ para referirse al conjunto de configuraciones libres de colisión.

2.2.1 Algoritmo de Árboles aleatorios de exploración rápida

```
CONSTRUIR_RRT(x_ini)
1 T.ini(x_ini);
2 para k=1 a K
3       xrand←ESTADO_ALEATORIO();
4       EXTENDER(T,x_rand);
5 Regresa T
```

```
EXTENDER(T,x)
1 x_near ←VECINO_MAS_PROXIMO(x,T);
2 si NUEVO_ESTADO(x,x_near,x_new,u_new) entonces
3       T.agrega.vertice(x_new)
4       T.agrega.arista(x_near,x_new,u_new)
5   si x_new= x  entonces
6       Regresa Alcanzado;
7   sino
8       Regresa Avanzado;
9 Regresa Atrapado;
```

Figura 2. 3. Algoritmo básico de construcción del RRT

El algoritmo básico de construcción de RRT, se muestra en la figura 2.3. En cada iteración se intenta extender el grafo agregando un nuevo vértice en dirección a un estado seleccionado aleatoriamente. La función *EXTENDER*, ilustrada en la figura 2.3, selecciona del grafo el vértice más cercano a un estado dado. Este vértice se elige de acuerdo a una métrica ρ. La función *NUEVO_ESTADO* hace un movimiento hacia x aplicando una entrada u ∈ U para algún incremento Δt. Esta entrada puede seleccionarse aleatoriamente o probando todas las posibles entradas eligiendo aquella que produzca un nuevo estado tan próximo como sea posible a x (si U es infinito puede usarse una aproximación o una técnica analítica). *NUEVO_ESTADO* utiliza implícitamente una función de detección de colisiones para determinar si el nuevo estado (y todos los estados intermedios) entra en colisión o no con los objetos del espacio. Si

NUEVO_ESTADO se cumple, el nuevo estado junto con la entrada, se representan por medio de x_{new} y u_{new}, respectivamente. Pueden ocurrir tres situaciones:

1. Alcanzado, el nuevo vértice alcanza al estado muestreado x, (tendríamos un umbral, $\|x_{new} - x\| < \varepsilon$ para un ε pequeño, ε > 0);
2. Avanzado, un nuevo vértice $x_{new} \neq x$ es agregado al RRT.
3. Atrapado, *NUEVO_ESTADO* falla en producir un nuevo estado que se encuentre en X_{free}.

2.3 Diseño de planificadores de caminos

Generalmente consideramos al RRT como un bloque en la construcción de un planificador eficiente. Por ejemplo, se puede usar un RRT para escapar de un mínimo local en un planificador de caminos aleatorio con campos de potencial. Se presenta varias alternativas de planificadores que utilizan un RRT. La elección recomendada depende de varios factores, tal como, sí existieran restricciones diferenciales, el tipo de algoritmo para la detección de colisiones, o la eficiencia en el cómputo del vecino más próximo.

2.3.1 Planificadores RRT simples.

En principio, el RRT básico puede usarse como un planificador de caminos por que sus vértices, eventualmente, cubrirán un componente conectado de $\mathbf{x}_{free}$, llegando arbitrariamente cerca de cualquier $\mathbf{x}_{goal}$ especificado. El problema es que sin ningún sesgo hacia el objetivo, la convergencia es lenta. Un planificador mejorado, llamado RRT_GoalBias [32], se obtiene reemplazando la función *ESTADO_ALEATORIO* en la figura 2.3 con otra que elija, con cierta probabilidad, entre x_{goal} o un estado tomado de cualquier parte del espacio de estados. Incluso con una probabilidad pequeña (p.ej. 0.05) de elegir x_{goal}, RRT_GoalBias usualmente converge al objetivo mucho más rápido que el RRT básico. Pero, sí se designa un sesgo muy grande hacia el objetivo entonces el planificador empieza a comportarse como un planificador aleatorio de campos de potencial, ya que pueden presentarse mínimos locales. Una mejora es, RRT_GoalZoom [32] el cual reemplaza la función *ESTADO_ALEATORIO* con una decisión, basada en cierta probabilidad, de optar por un estado aleatorio dentro de una región circunvecina al objetivo o por uno elegido del espacio de estados completo. El tamaño de la región alrededor del objetivo lo controla el vértice del árbol más cercano al objetivo. El efecto es que el foco de muestreo converge al objetivo como el RRT se acerca a él. Este planificador funciona bastante bien en la práctica, sin embargo, su rendimiento aún puede degradarse debido a mínimos locales. En general, parece mejor sustituir la función *ESTADO_ALEATORIO* con un esquema de muestreo que extraiga los estados desde una función de densidad de probabilidad no uniforme con un sesgo gradual hacia el objetivo. La figura 2.4 muestra un ejemplo de un RRT construido al muestrear los estados.

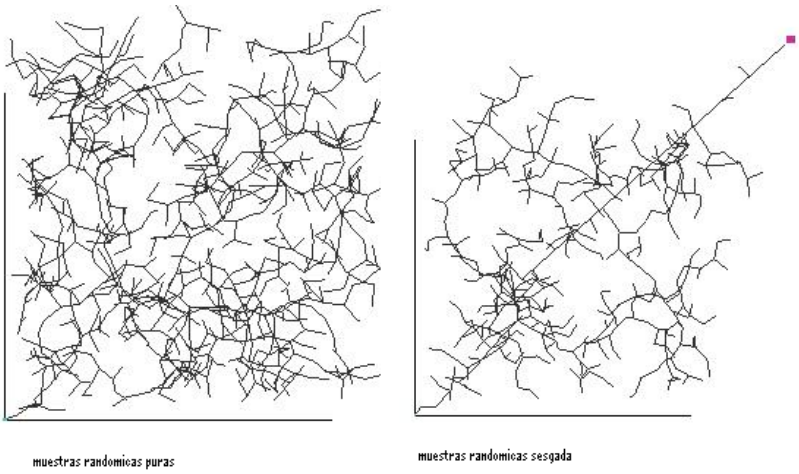


Figura 2. 4. Un RRT en 2D construido usando un muestreo sesgado

Un problema más a considerarse, es el tamaño del paso usado en la construcción del RRT. Este podría elegirse dinámicamente durante la ejecución, acorde con la función que calcula la distancia en la detección de colisión. Si los cuerpos están lejos de colisionar, se puede tomar un paso grande. Además de seguir esta idea, ¿qué tan lejos debe estar $\boldsymbol{x}_{new}$ de $\boldsymbol{x}_{near}$?, ¿Trataríamos de conectar $\boldsymbol{x}_{near}$ con $\boldsymbol{x}_{rand}$? En lugar de intentar extender un RRT por un paso incremental, *EXTENDER* puede ejecutarse iterativamente hasta alcanzar al estado aleatorio o un obstáculo, como se muestra en descripción del algoritmo *CONECTAR* de la figura 2.5. Al sustituir *EXTENDER* por *CONECTAR* el árbol crece muy rápido, si lo permiten las restricciones de detección de colisión y las restricciones diferenciales. Una de las ventajas clave de la función *CONECTAR* es que construimos un camino tan largo como sea posible con una sola llamada al algoritmo del vecino más próximo. Esta ventaja motiva la elección de un algoritmo voraz.

```
CONECTAR(T,x)
1  repetir
2  S ←EXTENDER(T,x)
3  hasta que no (S=Avanzado)
4  Regresa S
```

Figura 2. 5. Función CONECTAR

La experiencia ha demostrado que la función *CONECTAR* produce mejores resultados en problemas holonómicos y la función *EXTENDER* es sobresaliente en problemas noholonómicos. Una razón para esta diferencia es que *CONECTAR* confía más en la métrica, y los problemas no-holonómicos requieren del diseño de buenas métricas.

2.3.2 Planificadores bidireccionales

Inspirados en las técnicas clásicas de búsqueda bidireccional, parece razonable esperar un mejor desempeño al crecer dos árboles de exploración, uno desde $\boldsymbol{x}_{init}$ y el otro a partir de $\boldsymbol{x}_{goal}$; se obtiene una solución cuando los dos árboles se encuentran. Para una búsqueda simple, la implementación es directa, sin embargo, la construcción RRT debe guiarse para asegurar que ambos árboles se encuentren antes de cubrir el espacio entero y permitir una unión eficaz.

```
RRT_BIDIRECCIONAL(x_ini,x_goal)
1  T_a.inic(x_ini); T_b.ini(x_goal);
2  para k=1 a K
3    x_rand ←ESTADO_ALEATORIO();
4        si (EXTENDER(T_a,x_rand)<>Atrapado) entonces
5        si (EXTENDER(T_b,x_new)=Alcanzado) entonces
6            Regresa CAMINO(T_a,T_b)
7    INTERCAMBIAR(T_a,T_b)
8  Regresa Fallo
```

Figura 2. 6. Planificador bidireccional usando un RRT

La figura 2.6 muestra el algoritmo ***RRT_BIDIRECCIONAL*** [32], puede compararse con el algoritmo ***CONSTRUIR_RRT*** de la figura 2.1. El RRT_BIDIRECCIONAL divide el tiempo de cómputo entre dos procesos:

1. explorar el espacio de estados;
2. intentar crecer los árboles uno hacia el otro.

Siempre existen dos grafos T_a y T_b, hasta que se enlazan y se encuentra una solución. En cada iteración un grafo crece, y se intenta conectar el nuevo vértice con aquel más cercano en el otro grafo. Entonces, se invierten los roles intercambiando los árboles. El crecimiento de dos RRTs también fue propuesto en [28] para planificación kinodynamic, en ese enfoque, en cada iteración ambos árboles crecían hacia un estado aleatorio. El algoritmo actual intenta que los árboles crezcan uno hacia el otro en la mitad de tiempo, con lo cual se obtiene un buen rendimiento.

Se consideran algunas variaciones del planificador anterior. Puede reemplazarse cualquier ocurrencia de *EXTENDER* con *CONECTAR* en el RRT_BIDIRECCIONAL. Cada reemplazo hace a la operación más agresiva. Si sustituimos *EXTENDER* por *CONECTAR* en la línea 4, entonces el planificador explora agresivamente el espacio de estados, con la misma compensación que el RRT básico. Si reemplazamos *EXTENDER* con *CONECTAR* en la línea 5, el planificador intenta unir los árboles agresivamente en cada iteración. Esta variante sería muy exitosa para resolver problemas de planificación holonómica, por comodidad la denominamos ***RRT_ExtCon*** [32]y al algoritmo original como ***RRT_ExtExt*** [32]. Entre las variantes discutidas hasta el momento, encontramos a RRT_ExtCon más eficiente para problemas holonómicos [33] y a RRT_ExtExt ideal para problemas noholonómicos. El planificador más agresivo que se puede construir es

sustituyendo las dos ocurrencias de EXTENDER por CONECTAR, líneas 4 y 5, originando RRT_ConCon [32].

2.3.3 Planteamientos adicionales con RRT

Si un enfoque dual ofrece ventajas sobre un árbol simple, entonces es natural preguntar sí el crecimiento de tres o más árboles ofrecería mayores ventajas. Estos árboles adicionales, iniciarían en estados aleatorios. Por supuesto, los problemas de conexión en el caso de la planificación no-holonómica serían aún más difíciles de resolver. El tiempo de cómputo debería dividirse entre intentar extender los árboles y tratar de conectarlos unos a otros. Muchas cuestiones quedan por investigar en este y otros planificadores que usen un RRT.

Es interesante considerar el caso límite, en el cual se construye un nuevo RRT para cada estado aleatorio $\boldsymbol{x}_{rand}$. Una vez generado un nuevo vértice, puede aplicarse la función CONECTAR, de la figura 2.5, a cada RRT. Para mejorar el rendimiento, podemos considerar los vértices que están a una distancia fija de $\boldsymbol{x}_{rand}$, de acuerdo con la métrica. Si una conexión tiene éxito entonces los dos árboles se unen en un único grafo. El algoritmo resultante, simula el comportamiento del roadmap probabilístico. Por lo que, el roadmap probabilístico puede considerarse como una versión extrema de un algoritmo que usa RRTs, donde se construye un número máximo de RRTs independientes.

2.4 Resumen de las propiedades de los RRTs

Esta última sección presenta un resumen de las propiedades representativas de los RRTs así como algunas desventajas.

- La expansión de los RRT es fuertemente sesgada hacia las porciones no exploradas del espacio de estados.
- La distribución de vértices en un RRT se aproxima a la distribución de muestreo, permitiendo un desarrollo consistente.
- Un RRT es probabilísticamente completo bajo condiciones muy generales.
- El algoritmo RRT es relativamente simple, lo cual facilita su análisis (esta es también una característica de los roadmaps probabilísticos).
- Un RRT siempre permanece conectado, aún cuando el número de aristas sea mínimo.
- Un RRT puede considerarse como un módulo de planificación de caminos, el cual puede ser adaptado a una gran variedad de sistemas genéricos de planificación.
- Se pueden construir algoritmos completos de planificación de caminos sin requerir de la habilidad para dirigir el sistema entre dos estados prescritos, lo cual amplia grandemente la aplicación de los RRT.

Los principales inconvenientes de los métodos RRTs es la fuerte dependencia en la métrica y que la razón de convergencia se expresa en virtud de parámetros que no pueden medirse fácilmente.

2.5 Propuesta de Variación RRT_GoalBias.

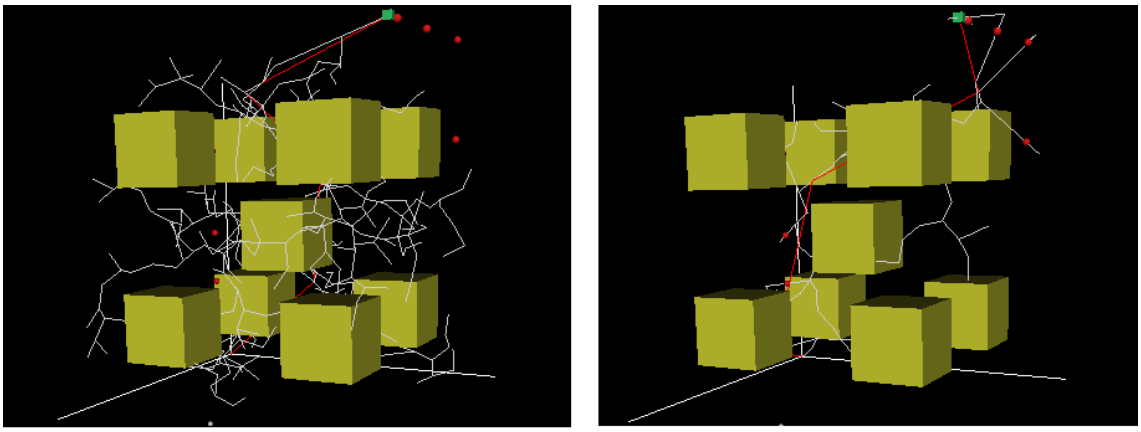
Como se describe en la sección 2.3.2 el algoritmo RRT_GoalBias introduce la posición $\mathbf{x}_{goal}$ con cierta probabilidad en la generación de las muestras aleatorias para de esta manera hacer que el árbol crezca en esa dirección. Se propone como una variación a este algoritmo el introducir varios puntos de paso, estos serán introducidos de acuerdo a cierta probabilidad en las muestras aleatorias generadas de tal forma que guíen el camino hacia la meta con lo cual el árbol crecerá con la tendencia a pasar por los puntos seleccionados, el objetivo es alcanzar la posición $\mathbf{x}_{goal}$ en menor tiempo y con un número menor de nodos empleados.

```
ESTADO_ALEATORIO()
1. R<<rv   //obtener una valor de probabilidad.
2. puntos[i]
3. GoalProb= xx
4. Si (rv >xx y rv <2*xx) entonces Regresa x_rand.
5. Si (rv >2*xx y rv <3*xx) entonces Regresa puntos[1]
6. ….
7. Si(rv>(i-1)*xxx y rv<i*xx)entonces Regresa puntos[i]
8. OtroCaso Regresa x_goal
```

Figura 2. 7. Algoritmo para generar muestras aleatorias con sesgo a los puntos de paso

En la figura 2.7 se muestra el algoritmo implementado, para poder obtener las muestras aleatorias con un sesgo hacia los puntos de paso, de acuerdo a un intervalo entre probabilidades se genera un punto perteneciente a los puntos de paso.

En la aplicación implementada, el usuario fija los puntos de paso mediante el dispositivo háptico, la figura 2.8a muestra el resultado al ejecutar el algoritmo RRT_GoalBias, empleando como sesgo la meta $\mathbf{x}_{goal}$, y en la figura 2.8b se usa la variación, los *n* puntos (enrojo) se han marcado como camino sugerido de paso.



a) RRT con muestro randómico GoalBias

b) RRT con muestro randomico y puntos de paso Goal Bias

Figura 2. 8. Resultado de usar el RRT_GoalBias modificado.

2.6 Adaptación de la librería MSL-RRT

La MSL (*Motion Strategy Library*) es una librería para un fácil desarrollo y prueba de algoritmos de planificación de movimientos para una amplia variedad de aplicaciones. La arquitectura del software es orientada a objetos y el diseño general es altamente modular [35].

Como parte de este proyecto, se adapto la librería MSL para poder utilizarla en el sistema operativo Windows XP (la versión original se desarrollo en Linux). Se agregaron además otras funcionalidades que no estaban presentes en la versión original de MSL. Entre estas funcionalidades tenemos:

- Una detección de colisiones basada en la librería H3D,
- Re-planteamiento de las clases dedicadas a la visualización 3D,
- Una presentación visual de los caminos encontrados en cada problema de planificación.

Esta librería incluye planificadores que usan árboles de exploración rápida (RRTs).

2.6.1 Descripción general

La librería MSL consiste de clases jerárquicas en C++, cada una de ellas sirve para un propósito independiente.

Cada una de las clases jerárquicas se explica brevemente a continuación:

- **Model:** Los métodos permiten que los algoritmos de planificación calculen el siguiente estado del sistema, dado el estado actual, un intervalo de tiempo, y una entrada de control aplicada sobre ese intervalo.
- **Geom**: Estas clases definen las representaciones geométricas de todos los obstáculos del ambiente, y cada parte del robot. Los métodos permiten a los algoritmos de planificación determinar si cualquier parte del robot esta en colisión con otra parte o con los obstáculos del ambiente.

- **Problem:** Esta es una clase interface para un planificador, la cual abstrae al diseñador del algoritmo de planificación lejos de detalles específicos tales como la detección de colisión, y la simulación dinámica. Cada instancia del problema incluye tanto una instancia de la clase *Model* como una de *Geom*. También se incluye un estado inicial y uno final, lo cual permite que un problema sea resuelto, típicamente por un algoritmo de planificación.
- **Solver:** Hay actualmente un tipo de solucionador, el cual es una jerarquía de planificadores. Un objeto *Solver* es inicializado con una instancia de la clase **Problem**, y un método busca una estrategia de movimiento que resuelva el problema.
- **Planner**: Esta clase contiene los miembros y métodos correspondientes a la creación del planificador, se definen los arboles y las restricciones que tendrá el problema de planeación.
- **RRT**: La clase base para todos los algoritmos derivados del RRT básico.
- **RRTGoalBias**: La clase para la implementación del algoritmo RRTGoalBias, derivada del RRT.
- **RRTDual**: La clase para la implementación del algoritmo RRTDual, derivada del RRT.

2.6.2 Clases para el RRT.

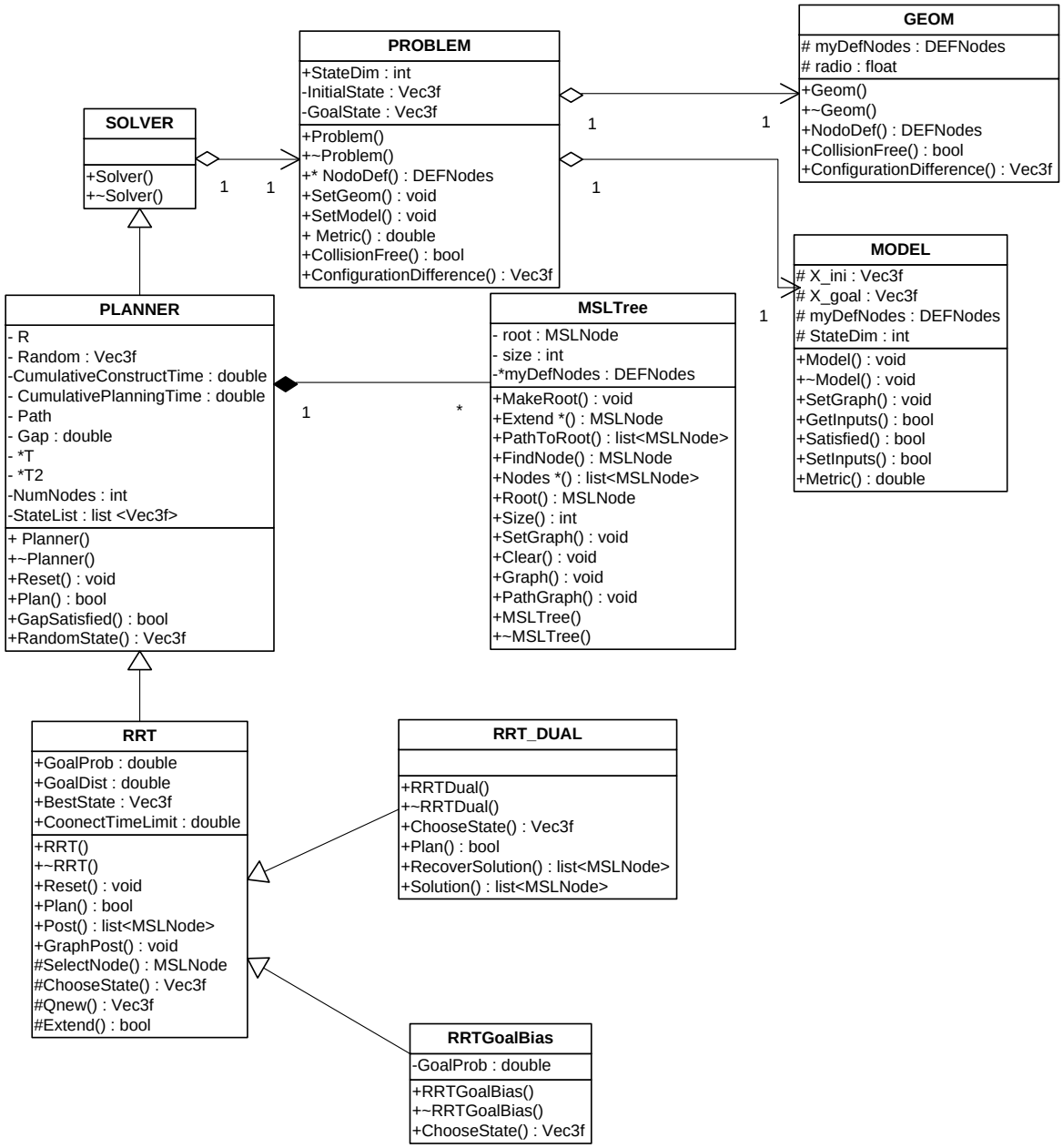


Figura 2. 9. Implementación de clases para los algoritmos RRT

La figura 2.9 muestra el diagrama UML de las clases correspondientes a la implementación de los algoritmos RRT.

En la implementación de los algoritmos RRT se han implementado clases adicionales para crear las estructuras y los procedimientos, necesarios para manejar la información.

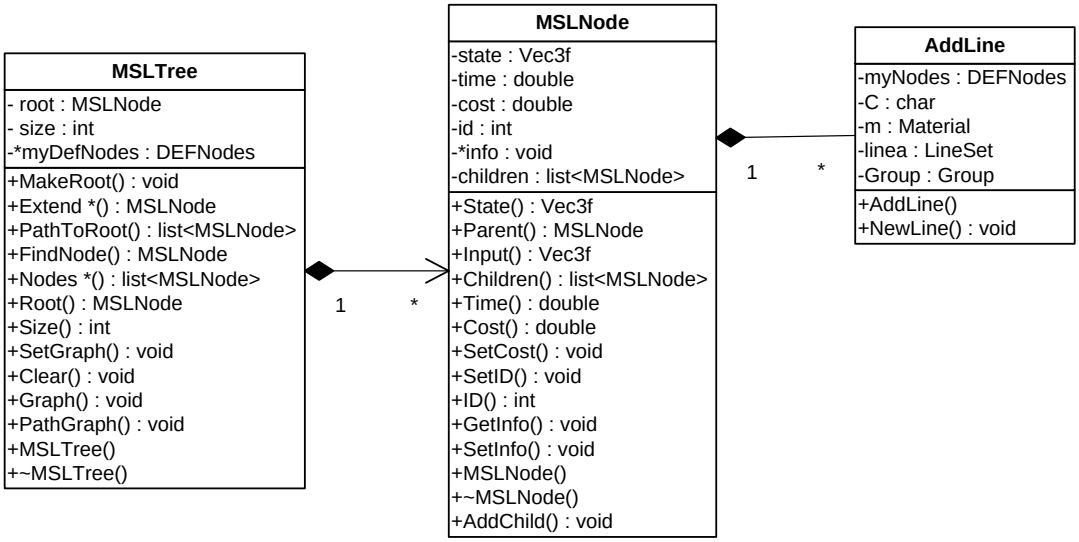


Figura 2. 10. Clase MSLNode y MSLTree.- crean la estructura de datos para el árbol RRT

Para el manejo del grafo del RRT, las clases MSLNode y la MSLTree mostradas en la figura 2.10 fueron modificadas cambiando la librería matemática para manejar vectores de la MSL por la librería matemática de la API H3D, se añaden métodos gráficos, para poder representarlos en la escena.

La estrategia empleada para la representación grafica, es crear una clase AddLine, la cual representar "una rama" del árbol, desde un nodo padre hacia un nodo hijo.

La tabla 2.1 Muestra los métodos creados para el manejo de la representación gráfica del árbol.

Método	Descripción.
`MSLTree::SetGraph(H3D::X3D::DEFNodes *dn=NULL)`	Establece la conexión entre el árbol y la escena gráfica.
`void MSLTree::Graph()`	Crea una representación gráfica del árbol.
`void MSLTree::PathGraph(list<MSLNode*> nl)`	Crea una representación gráfica del camino desde un nodo hacia la raíz.

Tabla 2.1 Métodos para mostrar el árbol en la escena.

2.6.3 Trayectoria final.

Una vez obtenido el árbol del RRT, para seleccionar un camino entre la posición inicial $\mathbf{x}_{ini}$ y final $\mathbf{x}_{goal}$, se realiza un post procesado del árbol, para obtener un conjuntos de puntos, para crear una trayectoria final.

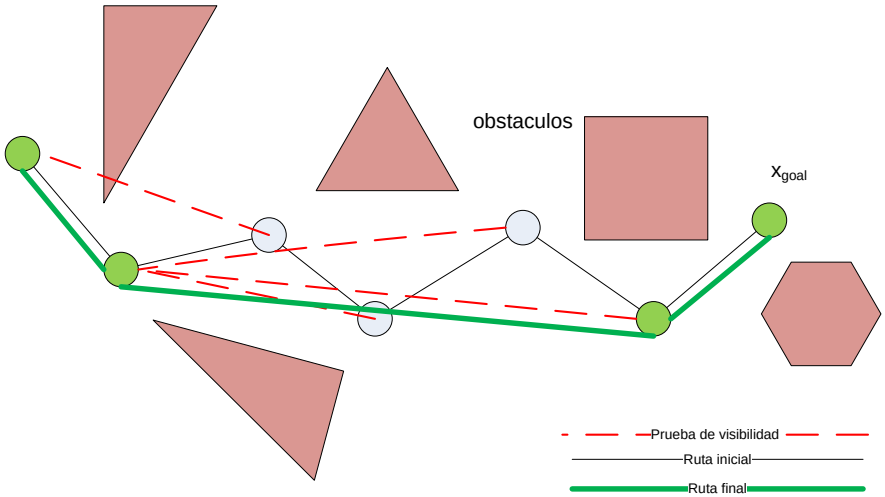


Figura 2. 11. Procesado de los nodos del árbol RRT, para obtener la Ruta final

El algoritmo de la figura 2.11 y 2.12 básicamente prueba la posibilidad de alcanzar el nodo siguiente a través de una línea recta, desde el punto de inicio $\mathbf{x}_{ini}$ hacia adelante hasta llegar al punto final $\mathbf{x}_{goal}$

```
PostProcesado(List<Nodes> Path)
1 y_ini=path.inicio
2 Almacenar(camino<-y_ini)
3 Para x=path.inicio a path.fin
4     movil=x+1
5     Si(colision(y_ini,movil))
6         y_ini=x
7         Almacenar(camino<-y_ini)
8 Retorno camino
```

Figura 2. 12. Algoritmo para obtener un camino final.

En la figura 2.13 se muestra el árbol generado por el algoritmo del RRT, representado por segmentos de línea que unen los puntos explorados en el espacio libre de colisiones, y un camino escogido para llegar a la meta.

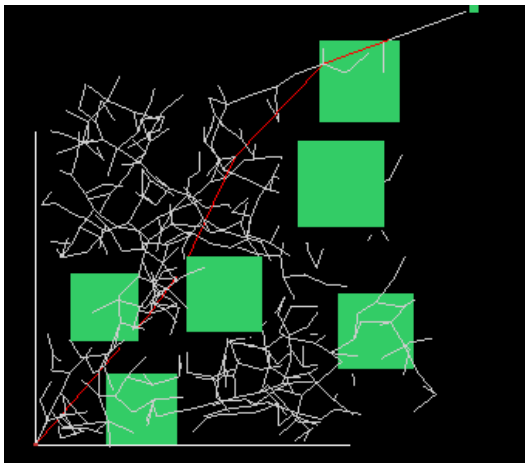


Figura 2. 13. Ejemplo de un camino usando un árbol RRT y procesando el camino final.

3 ARQUITECTURA DE DESARROLLO.

En el capítulo de introducción se indicó la arquitectura establecida para la implementación de aplicaciones de teleoperación mostrada en la figura 3.1.

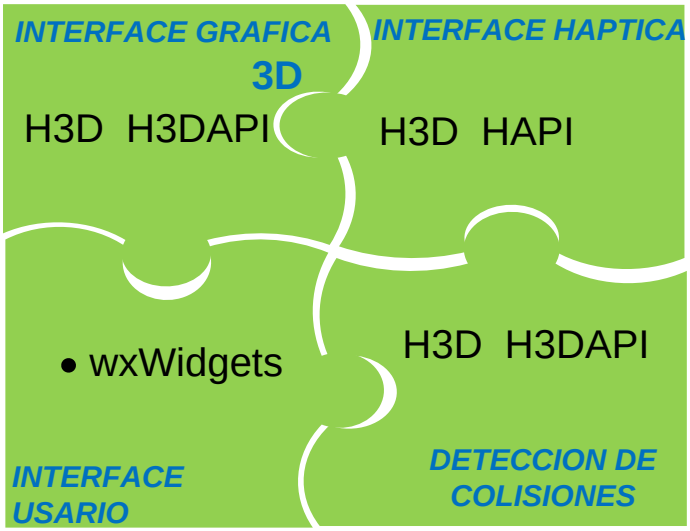


Figura 3. 1. Arquitectura de desarrollo para aplicaciones de Teleoperación.

La implementación de esta arquitectura se desarrolla en C++ y la programación orientada a objetos, se utiliza como interfaz de programación de aplicaciones (API) la librería H3D.

La principal característica por la cual se emplea el API H3D es por estar construida en base a los siguientes estándares.

- **X3D.-** Extensible 3D es un formato de archivo, sucesor del estándar VRML. Sin embargo X3D es más que un formato de archivo, es un estándar ISO para el grafo de la escena, diseño que es fácilmente extensible para ofrecer nuevas funcionalidades en manera modular.
- **XML**.- Extensible Lenguaje de Etiquetas, es un lenguaje de etiquetas estándar usado en una gran variedad de aplicaciones. El formato de los archivos X3D está basado en XML, y H3D viene con un completo procesador XML para cargar las definiciones del grafo de la escena.
- **OpenGL**.- Librería de gráficos multi plataforma, un estándar para gráficos en 3D. Hoy en día todos los procesadores gráficos soportan aceleradores OpenGL, está disponible en todos los sistemas operativos hasta ahora conocidos.
- **STL**- La librería de plantillas estándar de C++, una larga colección de plantillas que soportan un rápido desarrollo de aplicaciones C++ altamente eficientes.

Otra característica del API H3D es que está diseñada para un proceso de desarrollo rápido. Mediante la combinación X3D, C++ y el lenguaje de script Python, empleando

una combinación de X3D, C++ y Python se puede reducir en más de la mitad el tiempo de desarrollo comparado con hacerlo solo con C++.

La API H3D tiene tres componentes.

1. H3DAPI para el manejo de la escena.
2. HAPI para la comunicación con el dispositivo háptico.
3. UTIL un conjunto de nodos para aplicaciones extras.

La API H3D ofrece una extensión del X3D para aplicaciones de toque, usando dispositivos hápticos, con lo cual se convierte en una herramienta ideal para construir aplicaciones hapto-visuales y para aplicaciones de teleoperación.

3.1 La interface grafica.

El conjunto de clases H3DAPI de la API H3D, han sido implementadas para manejar el grafo de la escena, usan la sintaxis del X3D y así es como maneja todo el grafo. Por lo tanto para programar usando la API H3D, manejaremos los nodos y los campos del grafo de la escena.

El X3D es un lenguaje jerárquico de marcas que usa Nodos, Eventos y Campos para modelar realidades virtuales tanto estáticas, como dinámicas (sección 1.2.1.2).

- **Los Campos** (*Fields*) son los atributos que definen el comportamiento del objeto gráfico representado. Las funciones principales de los campos son:
    - Almacenamiento de Datos.-El campo almacena un dato de algún tipo.
    - Relaciones de dependencia.- Los campos establecen la relación entre un campo y otro, de manera que un cambio de valor de uno, dispara y actualiza otro.
    - Relaciones de comportamiento.- Un campo calcula su valor, dependiendo de los campos conectados a él.

  Las relaciones de dependencia en el X3D son especificadas a través de conectar los campos a través de un enrutamiento, esto establece una ruta entre un campo y otro, de manera tal, que un cambio en un campo envía un evento a otro u otros campos conectados a él.

- **Un Nodo** *(Node)* es un contenedor que agrupa campos, para crear entidades reutilizables. Estos son usados para construir el grafo de la escena X3D. Un nodo reúne todos los campos que son necesarios para controlar el comportamiento de lo que representan. El nodo especifica una interface para el mismo, determinando los campos a los cuales el usuario tiene acceso y ocultando las funcionalidades internas al usuario. En un nodo X3D, los campos pueden ser de 4 tipos.
  1. Entrada (*inputOnly*).- Solo admite entradas y su valor puede ser cambiado, no se puede obtener su valor.

2. Salida (*outputOnly*).- Son salidas del nodo, su valor no puede ser cambiado y su valor puede ser obtenido.
3. Inicialización (*inititalizeOnly*).- El valor solo puede ser iniciado una vez, y después trabaja como un nodo de salida.
4. Entrada Salida (*InputOutput*) .- No tiene restricciones.

- **Eventos** son los cambios en los valores de los campos, realizados por el usuario o por algún tipo de generador de eventos interno.

En la grafica 3.2 se muestra una representación del grafo de una escena, los nodos *Shape* y *Appearance* se definen a través de los diferentes campos, la escena final será una red de nodos.

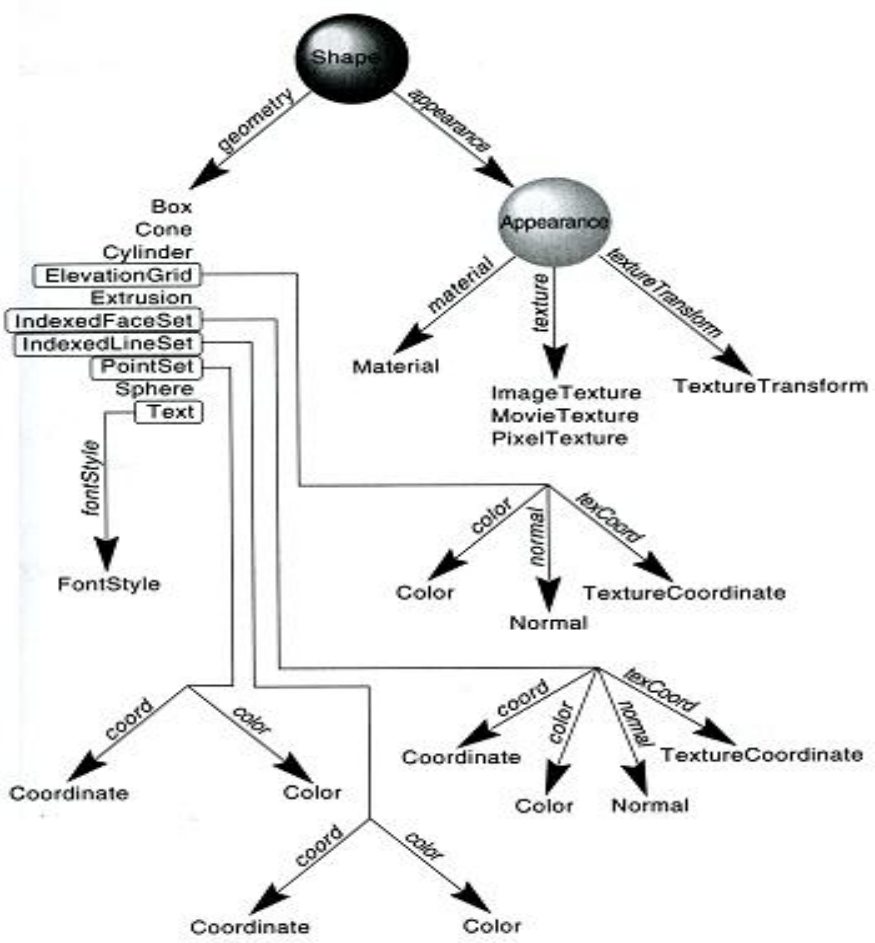


Figura 3. 2. Grafo de la escena.

Los Nodos se usan para instanciar alguna de las 54 formas básicas del lenguaje, llamadas primitivas, cada nodo entonces es una colección de Campos que contienen los atributos básicos de la primitiva.

Las primitivas del X3D se agrupan según función, en nueve colecciones distintas:

- **Agrupación de nodos**
- **Grupos especiales**
- **Sensores**
- **Geometría**
- **Propiedades de geometría**.
- **Apariencia de la geometría.**
- **Interpoladores**.
- **Nodos excluyentes**
- **Nodos comunes**

La descripción de un archivo X3D usa la sintaxis XML para describir sus componentes.

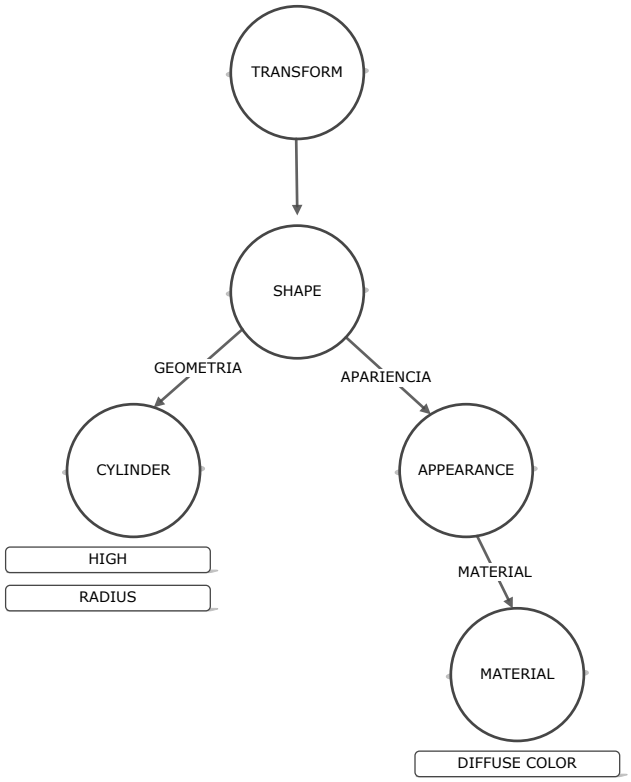


Figura 3. 3. Grafo de una escena simple.

La figura 3.3 indica el grafo de una escena simple, su descripción en X3D, empleando la sintaxis XML, se muestra en la figura 3.4 pueden apreciarse algunos nodos básicos, y los campos necesarios para definirlos.

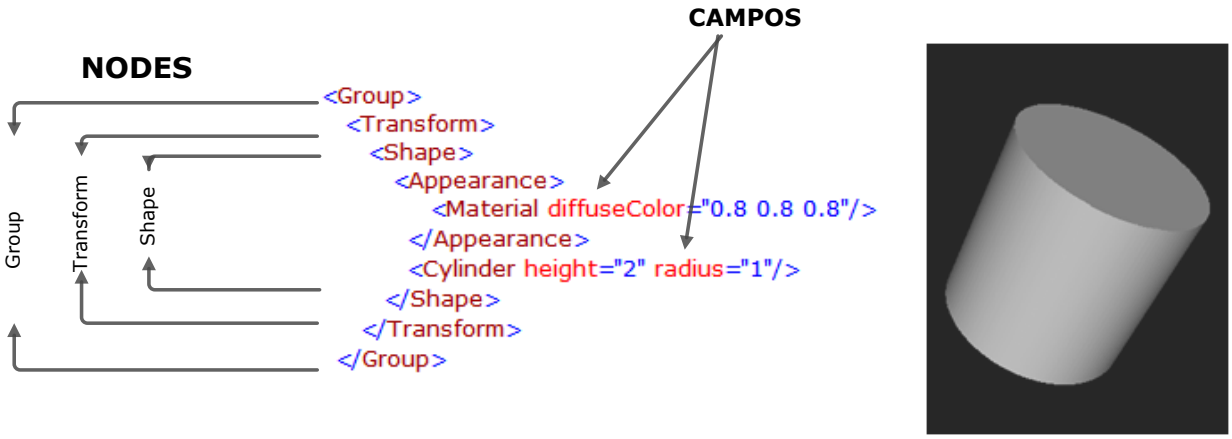


Figura 3. 4. Descripción XML de la escena y la salida resultante en el visor de X3D

Para creación de archivos tipo X3D, se puede usar un editor de texto cualquiera, y escribir empleando la sintaxis XML.

Para este proyecto se usa el editor XMLNotepad 2007 como se muestra en la figura 3.5; este editor usa para la descripción de los nodos el archivo de descripción x3d-3.2.xsd. Este archivo de su puede obtener de http://www.web3d.org/specifications/x3d-3.2.xsd

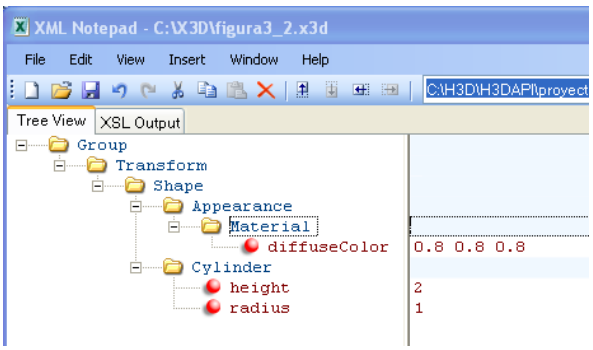


Figura 3. 5. Creación del archivo X3D usando el XML Notepad.

El editor XMLNotepad usa una representación de directorios para describir la escena, su uso es totalmente intuitivo y se obtiene libremente de la página de Microsoft.

3.1.1 La API H3D: H3DAPI.

La API H3DAPI es un conjunto de clases definidas como NODOS lo que nos permite manejar el grafo de la escena y extender las capacidades del X3D [13].

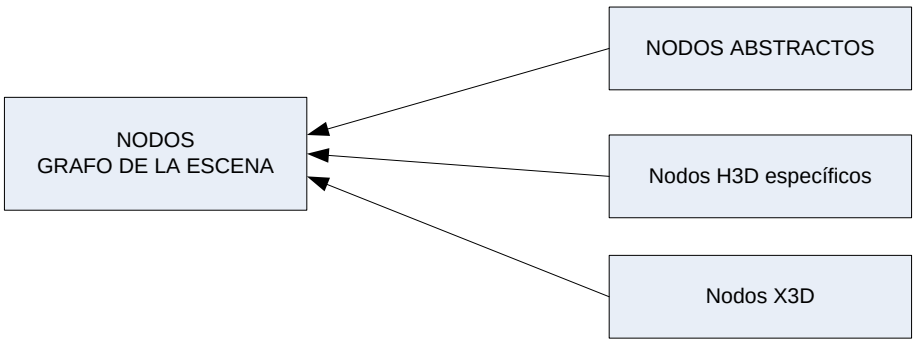


Figura 3. 6. Clases (nodos) API H3DAPI

En la figura 3.6 se observa la clasificación básica de los diferentes nodos (clases) con las cuales se trabaja en la H3DAPI.

La programación con H3DAPI sigue los mismos conceptos que el X3D, maneja el grafo de la escena a través de campos y nodos, para esto define clases y procesos los cuales manejan elementos del X3D, y se han extendido sus capacidades a través de nuevos nodos.

Una de las ventajas del API H3D al usar X3D, es que al ser un estándar ISO de diseño la mayoría de los software para gráficos permiten grabar en formato X3D, y por lo tanto se pueden importar y usar en H3D. Esto permite al usuario añadir características hápticas a gráficos ya existentes.

3.1.1.1 Campos H3D.

Una característica importante en la H3D es la forma de actualización de los campos, el grafo de la escena por defecto actualiza los campos usando evaluación "perezosa" es decir el valor del campo se actualiza en el instante que se consulta por él. Si se requiere una evaluación inmediata en la H3D se puede usar una función de actualización especial.

La API H3D tiene básicamente dos tipos de clases para actualización inmediata estos son:

AutoUpdate.- Se deriva una clase de este tipo, si se requiere evaluar un campo, cuando un tipo de cambio específico ocurre y los campos son del mismo tipo.

TypedField.- Se deriva una clase de este tipo cuando se requiere evaluar un cambio, en campos de diferente tipo.

3.1.1.2 Nodos H3D.

Una característica del API H3D al programar en C++, es permitir la creación de nuevos nodos, para extender las capacidades del H3D. Para crear un nuevo nodo se debe definir que campos estarán disponibles en la interface del nodo, y determinar las dependencias entre estos campos, entonces se crea una red interna para hacer lo que se requiere del nodo.

Los nodos pueden ser heredados de las clases abstractas H3D o X3D, y deberán iniciar su nombre de acuerdo al tipo al que pertenezcan.

Los procesos a definir cuando creamos un nuevo nodo son principalmente:

- void initialize().
- void render().
- void traverseSG(TraverseInfo &ti).
- bool lineIntersect(const Vec3f &from, const Vec3f &to, LineIntersectResult &result).
- void closestPoint(const Vec3f &p, NodeIntersectResult &result).
- bool movingSphereIntersect(H3DFloat radius, const Vec3f &from, const Vec3f &to,NodeIntersectResult &result).
- string defaultXMLContainerField().

Una plantilla para crear nuevos nodos podría ser la siguiente:

```
#ifndef   NODETEMPLATE H
#define __NODETEMPLATE_H__
 // Archivos include necesarios
 namespace H3D {
  class NodeTemplate: public BaseClass{
  public:
    NodeTemplate( Inst< FieldType >  _myFieldName = 0);
     // Funciones virtuales publicas.
     // Campos y tipos de campos.
    auto_ptr< FieldType >  myFieldName;
     /// El H3DNodeDatabase para este nodo.
    static H3DNodeDatabase database;
   protected:
    // funciones virtuales protegidas.
  };
```

Para utilizar un nuevo nodo se puede hacerlo de dos maneras.

1. Incluyendo el código del nuevo nodo dentro del ejecutable y correrlo en un solo programa.

2. Para que sea compatible con cualquier X3D entonces compilamos el nodo en una librería de enlace dinámico DLL, y la importamos a través del nodo.

```
<ImportLibrary library=”c:\H3D\bin\UI.dll/>
```

3.2 El Interface Háptico.

La API H3D maneja el dispositivo háptico a través de la librería HAPI, está ha sido diseñada de manera modular para permitir al usuario añadir, sustituir o modificar algún componente del proceso de render háptico.

La HAPI básicamente consiste de funciones para:

- Manejo del dispositivo.
- Geometría del dispositivo háptico
  - Manejo de colisiones.
  - Algoritmo de render del háptico.
  - Algoritmo de interacción con la superficie.
- Manejo del espacio libre.
  - Efectos de fuerza.
- Manejo del lazo háptico.

Manejo del dispositivo.

Provee de funciones para manejar el dispositivo a través de una interface para varios dispositivos hápticos. Maneja la inicialización, y apagado, accede al estado del dispositivo tales como la posición y orientación, y a las salidas tales como fuerza y torque. HAPI puede extenderse fácilmente implementando las funciones abstractas para tales tareas.

Geometría del dispositivo Háptico.

Manejo de Colisiones.

HAPI contiene clases para el manejo de colisiones usadas en los algoritmos de render del háptico. Un usuario crea estas instancias y alimenta estas al dispositivo háptico. También se cuenta con clases para arboles binarios, tales como ejes alineados, (AAB) y cajas envolventes (OBB).

Algoritmo de render háptico.

El algoritmo de render háptico es necesario para calcular las fuerzas y momentos de rotación de la interacción del dispositivo háptico con objetos en la escena, Proporciona una interfaz para dar al usuario la oportunidad de poner en práctica su propio algoritmo. Hay cuatro algoritmos ya implementados.

1. Algoritmo GodObject.- basado en un punto proxy.

2. Algoritmo Ruspini.- Basado en una esfera proxy.
3. Chai3D.- usa la librería Chai3D para render.
4. OpenHaptics.- Usa la librería OpenHaptics para render.

Algunos de estos algoritmos tienen restricciones con el tipo de características a poder ser usadas.

Manejo de Superficies.

Cuando tocamos la superficie de una geometría, el algoritmo de render tiene que conocer las fuerzas generadas dependiendo de la penetración de la superficie, esto es manejado por las clases definidas para esto. El usuario puede definir funciones arbitrarias para fuerzas y movimientos del proxy, esto no trabaja para todos los algoritmos de render.

Efectos de Fuerza.

Se pueden trabajar con efectos de fuerza sin estar basados en el toque con geometrías, estas dependen solamente de la posición y la orientación del dispositivo. Por ejemplo campos de fuerzas, gravedad, resortes, viscosidad. Para estos casos se usan efectos de fuerza.

El manejo del lazo del háptico

HAPI maneja lazos de alta prioridad ejecutándose a 1000Hz para el render háptico y provee mecanismos para comunicarse con diferentes hilos.

3.2.1 Especificando el dispositivo Háptico.

Para especificar el dispositivo háptico se tiene básicamente dos opciones.

1. Usar la utilidad H3DLoadSettings con la cual se puede especificar las propiedades del dispositivo háptico, con esta interface se fijan los parámetros del dispositivo.
2. Especificar las características del dispositivo háptico, a través de un nodo agrupante *DeviceInfo* en el archivo X3D y fijar allí sus propiedades.

```
<DeviceInfo>
  <PhantomDevice positionCalibration="2 0 0 0
                                      0 2 0 0
                                      0 0 2 0
                                      0 0 0 1" >
    <OpenHapticsRenderer/>
    <Shape containerField="stylus">
      <Appearance> <Material/> </Appearance>
      <Sphere radius="0.005" />
    </Shape>
  </PhantomDevice>
</DeviceInfo>
```

Figura 3. 7. Ejemplo de configuración del dispositivo háptico.

En la figura 3.7 se tiene el nodo *DeviceInfo*, y los campos *PhamtonDevice*, usado para el dispositivo háptico de SensAble , en el ejemplo se especifica también el estilo del dispositivo, esto es la representación grafica del dispositivo, para el ejemplo se observa una esfera.

3.2.2 Configuración del dispositivo Háptico.

La API H3D permite acceder a los campos del dispositivo háptico, por ejemplo para conectar desde alguno de sus campos, dependiendo del nivel de programación en el que nos encontremos se puede hacer de diferentes maneras

X3D.

Si se está en X3D se accede al dispositivo háptico a través de definir el *H3D_EXPORTS*. Este nodo contiene el campo *HDEV* como el nombre del dispositivo, por ejemplo el dispositivo háptico es definido como *HDEV*.

```
<IMPORT inline DEF=´H3D_EXPORTS' exportedDEF='HDEV'
AS='HDEV'/>
```

A través del *HDEV* podremos acceder a los demás campos del háptico, el nombre se puede cambiar a HDEV1, HDEV2 si se tiene más de un dispositivo háptico.

C++

Si estamos en este nivel, recurrimos al uso de punteros a las clases *DeviceInfo* y *H3DHapticsDevice*.

```
DeviceInfo *di =DeviceInfo::getActive();

H3DHapticsDevice *hdev=NULL;

Hdev=di →device→getValueByIndex(0)
```

3.2.3 Configuración de las propiedades de los objetos

Para hacer una figura "tocable", es necesario definir las propiedades hápticas de la figura. Para esto definimos las propiedades en el campo *Appearance* del nodo *Shape.*

Efectos de superficie.

Actualmente hay 5 tipos de superficies que pueden ser definidas.

- **SmoothSurface**.- una superficie sin fricción.
- **FrictionalSurface.**- una superficie con fricción.
- **OpenHapticsSurface**.- Especifica las propiedades cuando usamos OpenHaptics, solo trabaja con OpenHapticsRenderer.

- **DepthMapSurface.-** Una superficie con fricción en la cual una textura decide como se siente la superficie.
- **HapticTexturesSurface**.- Una superficie con fricción en la cual una textura modifica los valores de algunos de los parámetros dependiendo de donde la figura es tocada y como las coordenadas son especificadas para la figura.

Efectos de fuerza

Adicionalmente a tener figuras tangibles, se pueden agregar efectos de fuerza en el espacio libre. Un efecto de fuerza es más o menos una función de la posición y la orientación del dispositivo háptico con la fuerza y el torque. Los efectos de fuerza disponibles son:

- **ForceField.-** Especifica una fuerza constante para enviar al dispositivo háptico.
- **MagneticGeometryEffect**.- Es similar al comportamiento de una superficie magnética, es un dispositivo háptico con un resorte el cual es usado para mantener el dispositivo ligado al a superficie.
- **PositionFunctionEffect**.- Tres funciones deciden la fuerza de salida con una entrada de posición del dispositivo háptico.
- **SpringEffect.-** Un resorte ligado al háptico, que puede ser usado para mantener el dispositivo háptico a una posición dada.
- **TimeFunctionEffect.-** Tres funciones deciden la fuerza de salida correspondiente a una entrada de tiempo.
- **ViscosityEffect** – Especifica una fuerza en dirección opuesta al movimiento del háptico

3.3 Interface de Usuario (GUI).

Para la creación de la interface con el usuario se usan los wxWidgets. En la API H3API, se ha definido una librería la cual nos permite conectar estos a la aplicación.

Los wxWidgets proporcionan una interfaz gráfica basada en las bibliotecas ya existentes en el sistema (nativas), con lo que se integran de forma óptima y resultan muy portables entre distintos sistemas operativos. Están disponibles para Windows, MacOS, GTK+, Motif, OpenVMS y OS/2.

Para desarrollar la interfaz gráfica se emplea:

- Un compilador de C++, en este proyecto se usa VISUAL STUDIO2008.
- Los wxWidgets para Windows, aquí la versión 2.8.10
- wxFormBuilder. Constructor de interfaces para wxWidgets, versión 3.0.

El wxFormBuilder es un editor WYSIWYG (*What You See Is What You Get*, lo que ves es lo que obtienes) que ayuda a crear proyectos en C++ utilizando las librerías WxWidgets cómodamente desde un entorno gráfico.

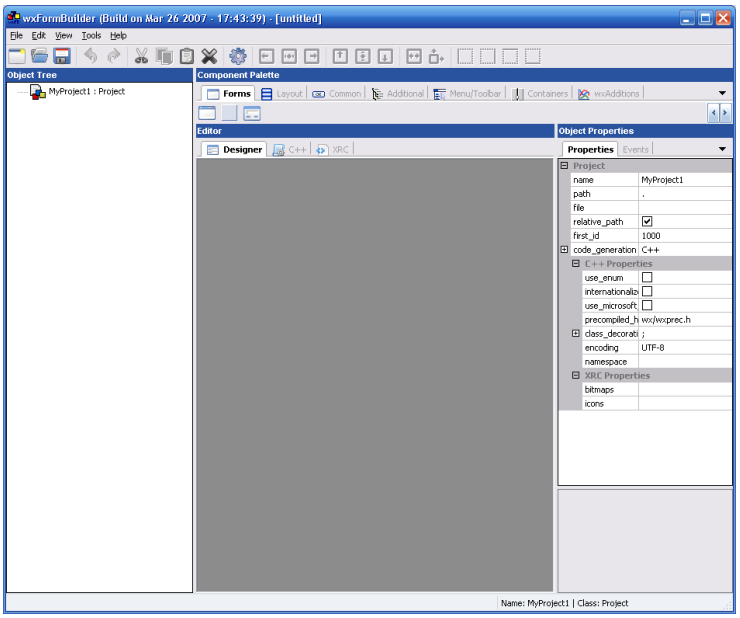


Figura 3. 8. Interface wxFormBuilder

El objetivo de wxFormBuilder es ofrecer los controles incluidos en cualquier aplicación para ahorrar tiempo en programar ese control y su orden correspondiente. Ahora, para incluir las ventanas, botones y demás elementos sólo hay que acudir a las múltiples opciones que ofrece wxFormBuilder y adaptarlas a las necesidades del proyecto.

3.3.1 Programación con wxFormBuilder.

La creación de la interface para el usuario en el wxFormBuilder es bastante intuitiva, mediante las técnicas comunes de arrastrar y pegar, se crean los elementos que constituirán la interface para el usuario, una vez creada la organización de los elementos, se definen las características, propiedades y los eventos a través de los cuales interactuara con el usuario final.

El proceso se indica en la figura 3.9.

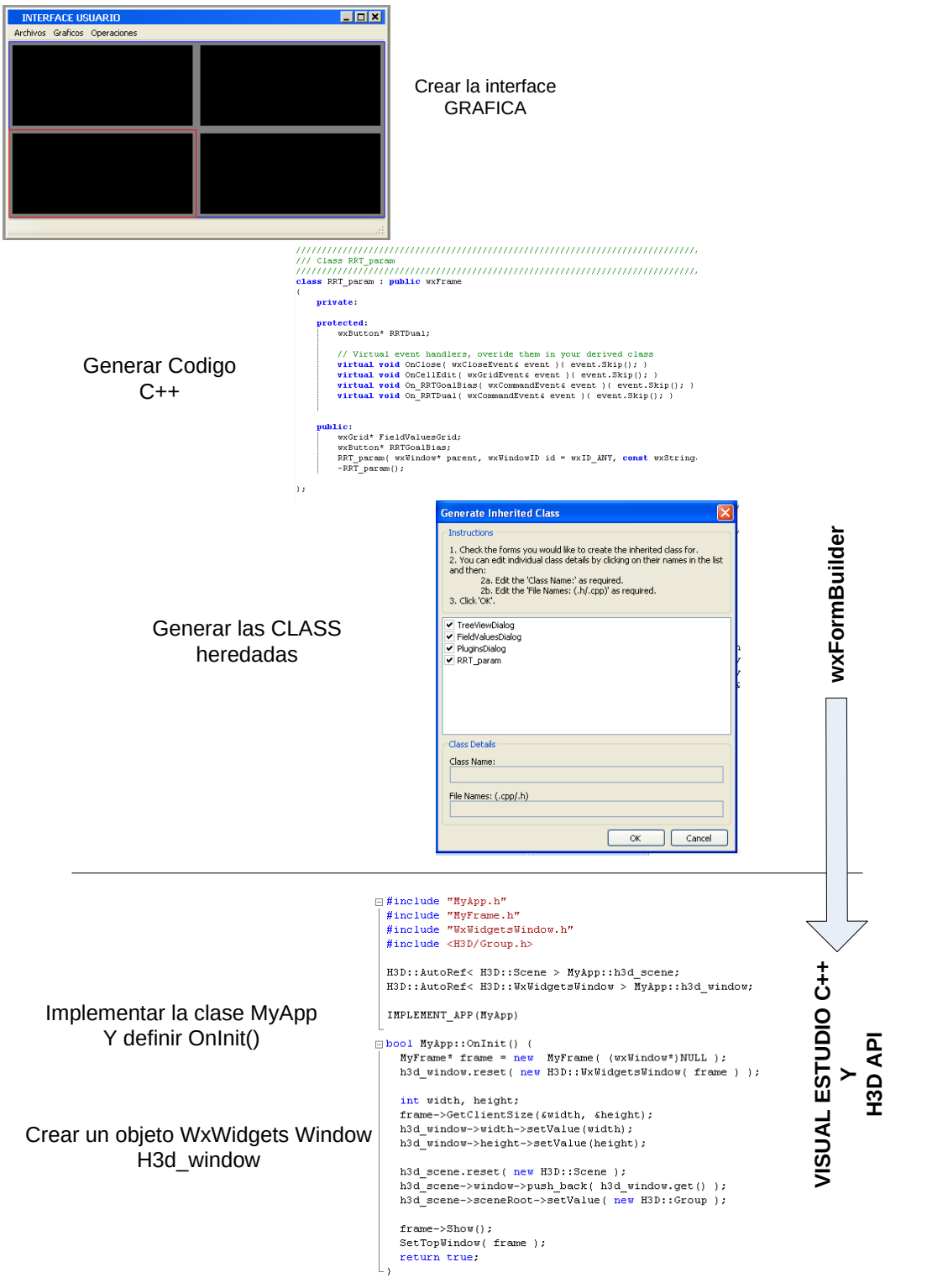


Figura 3. 9. Proceso de creación de la aplicación.

3.3.2 H3D con wxWidgets

Para conectar la interface del usuario con el H3D, se usa una librería del H3D wxWidgets, que ha sido definida para hacer este enlace.

Figura 3. 10. Conexión entre wxWidget y el H3DAPI

En la instalación de la API H3D se incluye la librería wxWidgets, un canvas que permite usar los wxWidgets con H3D, el tipo de librería a usar para la creación de la GUI en H3D es decisión del programador, se puede usar wxWidgets o la librería GLUT.

El grafo de la escena esta contenido por un objeto *Scene*. El objeto *Scene* es el nodo más alto en la H3D, este mantiene el grafo, y la ventana en la cual se graficara el grafo de la escena.

Para mostrar la escena en un ventana creada con una aplicación GUI como wxWidgets, es necesario implementar la clase abstracta *H3DWindowNode*, para esto H3D ha pre-implementado esta clase.

El proceso para crear una aplicación será:

- Crear un marco derivado del wxFrame.
- Crear una ventana del tipo WxWidgetsWindow
- Contener la ventana creada en el marco, y use la ventana como la ventana a la escena H3D
- Implementar los manejadores de eventos de la aplicación

3.4 Detección de Colisiones.

Para la detección de colisiones se emplea la detección de colisiones del H3DAPI, esta detección de colisiones se hace en modo offline. La prueba de colisión se realiza sobre los elementos definidos dentro del nodo de agrupamiento "*Group*".

Por defecto todas las geometrías de la escena son colisionables, excepto las líneas y los puntos, la API H3DAPI se define la capacidad de detección de colisión, solo para objetos esféricos, esto es una limitación en la actualidad dentro de esta arquitectura.

Los procesos de detección de colisión disponibles son:

Bool **lineIntersect** (const Vec3f &from, const Vec3f &to, **LineIntersectResult** &result)	Detecta la colisión entre un segmento de línea y un Nodo.

closestPoint (const Vec3f &p, **NodeIntersectResult** &result)	Encuentra el punto más cercano en el Nodo.
Bool **movingSphereIntersect** (H3DFloat radius, const Vec3f &from, const Vec3f &to, **NodeIntersectResult** &result)	Encuentras si existe colisión entre una esfera de radio r, moviéndose desde un punto inicial a uno final

4 EL SISTEMA DE GUIADO.

El sistema de guiado tiene como objetivo el implementar un módulo de ayuda para el usuario, este módulo permite seguir un camino calculado por el planificador, dentro de un entorno virtual tridimensional, y mantenerse unido a él mediante una fuerza de adhesión impuesta por el dispositivo háptico.

El problema a resolver se plantea de la siguiente forma.

En un archivo de entrada de tipo X3D se modela el entorno virtual, en este se definen: los objetos para modelar la escena, y el objeto móvil, en este caso una esfera de radio "r", se especifican la posición inicial del móvil, por defecto un vector de posición $\mathbf{x}_{ini}[0,0,0]$, la posición final, por defecto $\mathbf{x}_{goal}[1,1,1]$. La aplicación carga la escena y se calcula un camino a seguir a través de uno de los algoritmos RRT implementados. Una vez obtenido el camino, se sigue la ruta con el dispositivo háptico. La aplicación genera las fuerzas necesarias para mantener al háptico pegado a la ruta a seguir.

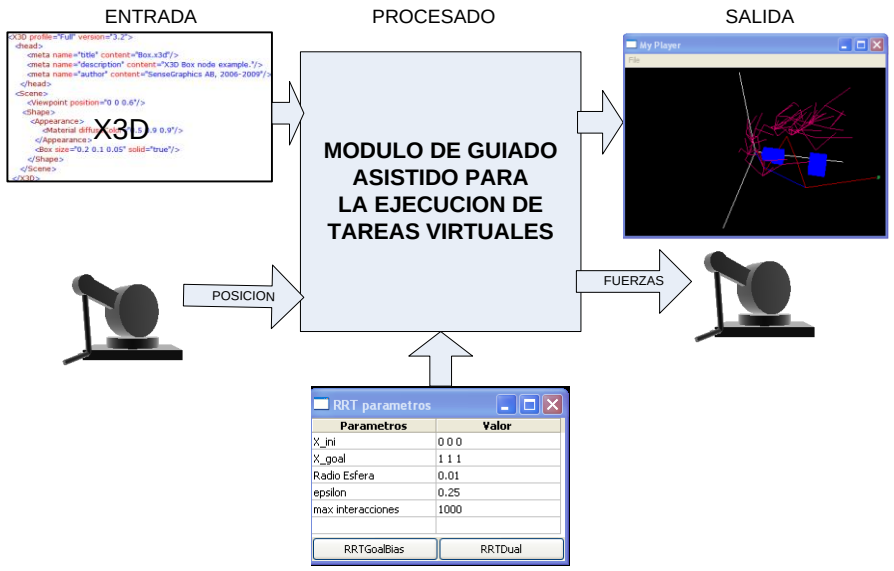


Figura 4. 1. Descripción esquemática del sistema de guiado asistido

La figura 4.1 indica los componentes básicos de la aplicación desarrollada. Se ha dividido en 4 áreas de operación.

1. Las entradas.- Se compone de un archivo de entrada en forma X3D para describir la escena virtual, el móvil y el ingreso de la posición del háptico.
2. El procesado donde se implementa el planificador, y las rutinas necesarias para recibir y enviar los datos al dispositivo háptico, y los comandos del interface de usuario.
3. La escena de salida.- La interface visual con el usuario, para permitir la interacción con el módulo.
4. Las fuerzas de salida para guiar el dispositivo háptico.

A continuación se describe cada uno de los componentes, tanto a nivel operativo, como su implementación.

4.1 Archivo de Entrada.

El archivo de entrada es de tipo X3D, describe la escena con sus objetos y el móvil, en este proyecto el móvil corresponde a una esfera.

El archivo de entrada X3D es estándar, esto implica que podemos emplear todas las opciones de construcción del estándar.

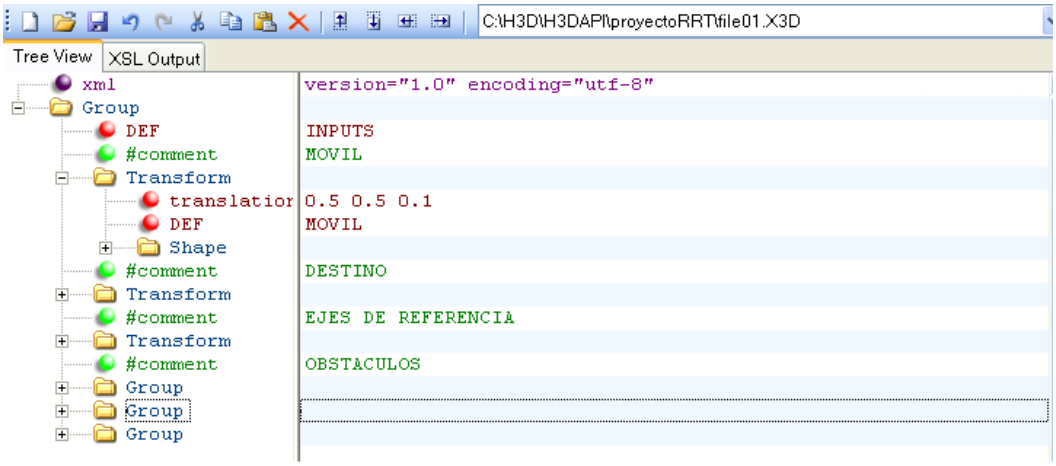


Figura 4. 2. Archivo de entrada típico X3D.

La figura 4.2 muestra un archivo típico de entrada en formato X3D, es importante identificar los nodos a través de la etiqueta *DEF*, el archivo debe tener la estructura mostrada en la figura 4.3. Sus componentes principales son:

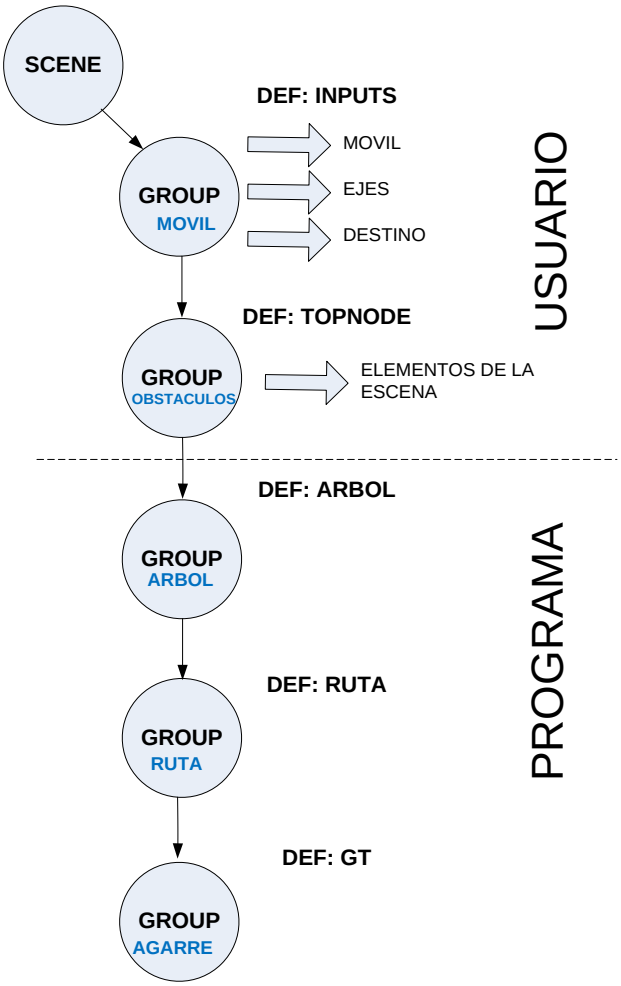


Figura 4. 3. Estructura del archivo de entrada, descrita en nodos

1. **El móvil**.- Definido como una esfera, se especifica su radio y la posición inicial, la posición del móvil se puede alterar a través del interfaz GUI del módulo. Es obligatorio definir los siguientes parámetros.

ELEMENTO: MODELO		Descripción.
Transform	Translation	*Posición inicial del modelo.*
	DEF	MOVIL
Shape	Sphere	
	Radius	*Especificar radio de la esfera*

2. **Los ejes**.-Definido a través de líneas 3D un sistema de ejes de referencia, no es obligatorio para la escena definirlos, estos ejes no participan en la detección de colisiones.
3. **El destino.-** Definido como una caja centrada en la configuración objetivo, la posición del destino se puede alterar a través del interfaz GUI del módulo. Esta geometría no participa en la detección de colisiones. Es obligatorio definir los siguientes parámetros.

ELEMENTO: DESTINO		Descripción.
Transform	Translation	*Posición final del modelo.*
	DEF	DESTINO
Shape	Box	
	size	*Especificar tamaño de la referencia*

4. **Los obstáculos.-** Definido a través del nodo *Group*, contiene todos los elementos de la escena, sin restricciones, es obligatorio identificar el nodo; todos los elementos de este nodo participan en la detección de colisiones.

ELEMENTO: OBSTACULOS		Descripción.
Group	DEF	TOPNODE (***Si no se especifica este DEF, el programa no podrá manejar la escena)***

5. **El árbol**.- Definido para agregar los elementos visuales (ramas) del árbol del RRT en la escena.

ELEMENTO: ARBOL		Descripción.
Group	DEF	ARBOL ***(Si no se especifica este DEF, el programa no podrá presentar el árbol en la escena)***

6. **La ruta**.- Definido para agregar la ruta a seguir, este nodo tiene la característica de tener la propiedad "***magnetic***" activada para obligar a seguir el camino.

ELEMENTO: RUTA		Descripción.
Group	DEF	RUTA
Shape	Appearance	
	• Material	**MagneticSurface**
	LineSet	DEF: MAGNETIC
	• vertexCount	2
	• Coordinate	Point : 0 0 0,0 0 0
	• HapticsOptions	maxDistance:0.15

7. **El agarre**.- Este es un nodo nuevo, no está definido en el estándar del X3D, ni en la API H3D, fue necesario crear este nodo porque en el algoritmo de render *Ruspini*, se puede definir una esfera, como puntero del háptico, pero no se mantiene la propiedad *"magnetic"* necesaria para seguir el camino planificado. Se fija entonces el render a *OpenHaptics* y se creo este nodo denominado *GraspTransform* para poder "agarrar" el objeto móvil con el dispositivo háptico y seguir el camino planificado a través de la escena.

Para poder "agarrar" un objeto en la escena es necesario crear un nodo GraspTransform que permite interactuar con el móvil a través del botón del dispositivo háptico.

Cuando se activa el botón del háptico, se obtienen los valores de la posición y orientación actual, una matriz de transformación es actualizada para mover el objeto con los cambios de la posición y la orientación. Al liberar el botón, la matriz de transformación deja de actualizarse.

Los campos de este nodo son:

- grasp-. Obtiene el estado del botón del dispositivo háptico.
- graspPosition.- Obtiene la posición del háptico.
- graspOrientation.- Obtiene la orientación del háptico.
- Matrix.- Matriz de transformación

Los métodos para operar con este nodo son:

- Update( ).- Detecta inmediatamente el estado del botón del háptico.
- graspObject( ).- Obtiene el estado de la posición y orientación del háptico; llama a la actualización de la matriz de transformación.
- releaseObject( ).- Almacena el estado final de la matriz de transformación y actualiza los valores del objeto a la última posición de cambio.

Una vez compilado el código, se usa una dll para cargarlo en el archivo X3D.

ELEMENTO: AGARRE		Descripción.
Group	DEF	AGARRE
ImportLibrary	url	**UsefulNodes**
IMPORT	Inline	DEF: H3D_EXPORTS
	Exported	DEF: HDEV
	AS	HDEV
GraspTransform	DEF	GT
	Shape	Sphere
	ROUTE	fromNode HDEV fromField mainButton toNode GT toField grasp
	ROUTE	FromNode HDEV FromField trackerPosition ToNode GT ToField graspPosition
	ROUTE	FromNode HDEV FromField trackerOrientation ToNode GT ToField graspOrientation

4.2 El Programa del Módulo de Guiado.

El módulo de guiado tiene 4 componentes

1. El interface de usuario GUI.
2. El planificador RRT.
3. El manejador de escena X3D.
4. El manejo online

4.2.1 El interface de Usuario.

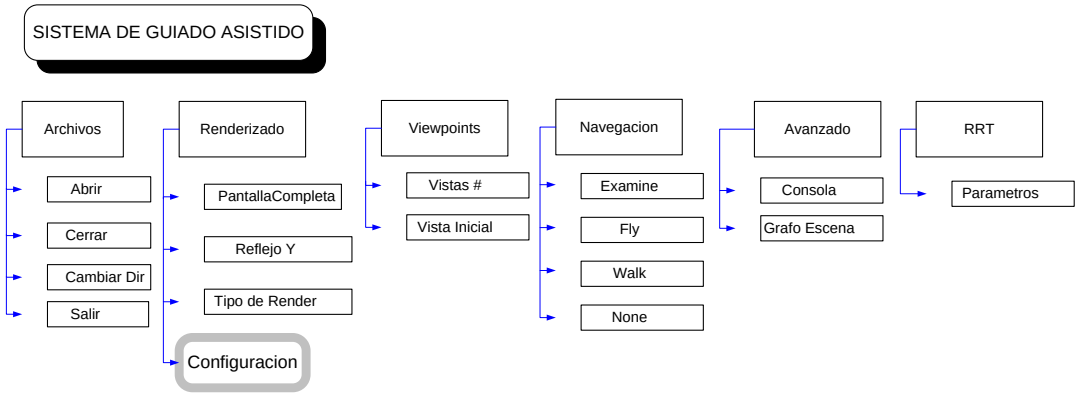


Figura 4. 4. Esquema de la interface de usuario del SISTEMA DE GUIADO ASISTIDO.

La figura 4.4 muestra la descripción del esquema implementado para la interface de usuario.

Se han definido 6 menús para las operaciones a realizarse con la interface.

1. **Archivos**.- Corresponde a las opciones para manejar el archivo de entrada de tipo X3D.
2. **Renderizado**.- Se maneja las opciones de render de la escena, permitiendo obtener una vista en pantalla completa, la opción de ver la imagen reflejo alrededor del eje Y, y los diferentes tipos de render del dispositivo háptico.
3. **Puntos de Vista**.- Se accede a los puntos de vista definidos en la escena, para las cuatro imágenes vistas de la escena. Y una opción para la vista inicial.
4. **Navegación**.- Las opciones compatibles con las diferentes formas de navegar establecidas en el X3D.
5. **Avanzado**.- Dos herramientas que permiten al usuario recibir información de la aplicación.
    - La consola.- Presenta mensajes de texto al usuario sobre las acciones realizadas.
    - El Grafo de la Escena.- Un visor del grafico del X3D, permite modificar los elementos de la escena, para realizar pruebas.

- Háptico Online.- Esta opción avanzada, muestra la posición del dispositivo háptico, y permite recalcular el camino a seguir cada vez que se presiona el botón del dispositivo háptico, el algoritmo empleado será establecido en la ventana de Parámetros del RRT.

6. RRT.- Recibe los parámetros de entrada para el planificado RRT.

La figura 4.5 muestra una vista de la interface de usuario de la aplicación implementada. A continuación se describen sus componentes.

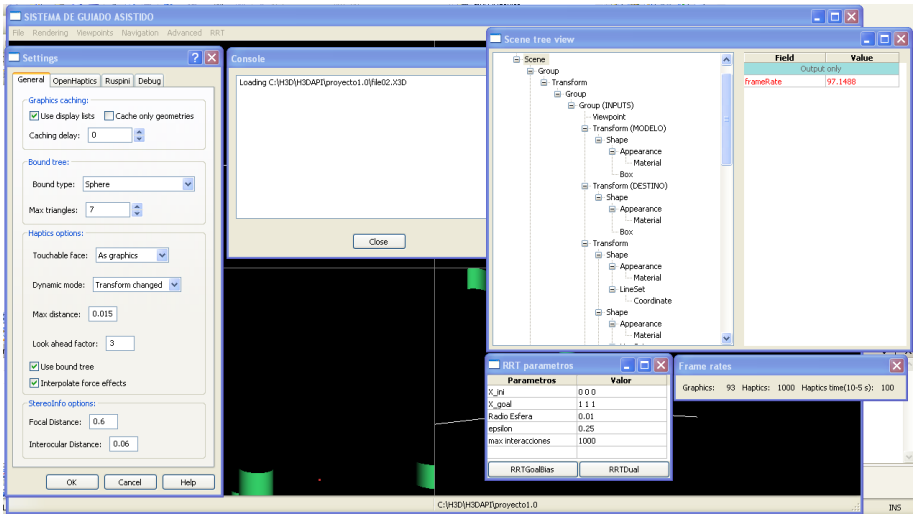


Figura 4. 5. Interface GUI de la aplicación.

4.2.1.1 Configuración de la aplicación.

Este es un conjunto de paneles tipo "*notebook*" mostrados en la figura 4.6, para configurar las características visuales usadas con el dispositivo háptico.

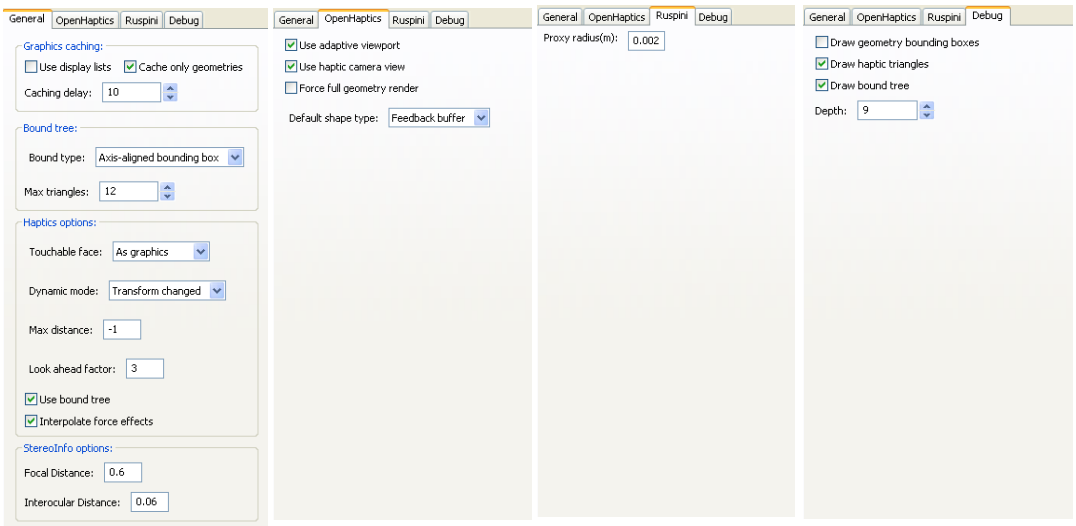


Figura 4. 6. Opciones de configuración de la aplicación

General

Define las condiciones generales del trabajo con el dispositivo háptico.

OpenHaptics.

Para este proyecto se usa el render OpenHaptics, por ser compatible con la opción "*magnetic*" del archivo X3D. Se tienen las siguientes opciones.

- Use adaptive viewport.- Optimiza el espacio del envolvente usado para graficar el dispositivo háptico.
- Use haptic camera view.- Modifica el OpenGL, para regraficar solo un espacio del entorno del dispositivo háptico.
- Force full geometry render.

Ruspini.

Esta opción permite definir el radio de la esfera usado para representar el dispositivo háptico, esta esfera interactúa con la escena, en esta opción de render, no es compatible con la opción "magnetic".

Debug.

La opción *debug* mostrara las geometrías envolventes usadas para detección rápida de colisiones, de acuerdo al tipo de envolvente seleccionado en las opciones *General.*

4.2.2 El planificador RRT.

Los algoritmos implementados en el RRT, son

- RRT Goal_Bias
- RRT Dual.

Figura 4. 7. Parámetros del RRT.

La figura 4.7 muestra el interface para el manejo del RRT, mediante esté se establece los parámetros estándar para ambos algoritmos.

$\mathbf{x}_{\_ini}$	Posición inicial del móvil.
$\mathbf{x}_{\_goal}$	Posición final del móvil.
Radio Esfera	Radio del móvil.
epsilon	Distancia de crecimiento de cada rama del árbol.

Max interacciones	Número máximo de iteraciones para la búsqueda de la solución.

Los botones seleccionan el tipo de algoritmo a usar, una vez activado este algoritmo se mantendrá establecido, hasta cambiar de opción.

4.2.3 El manejador de la escena X3D.

La API H3D permite unir complementos a las aplicaciones, estos son extensiones del H3D para manejar características de los nodos X3D, en esta aplicación se usa el visor de los componentes del grafo mostrado en la figura 4.8, a través del árbol, esto permite al usuario interactuar con los diferentes componentes de la escena.

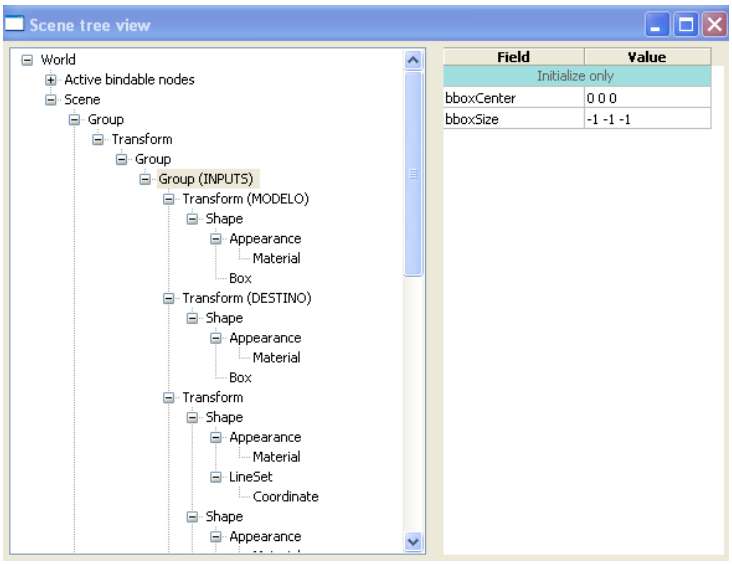


Figura 4. 8. El Tree view para ver y manipular el grafo de la escena.

4.2.4 Manejo Online

Esta opción avanzada, muestra la posición del dispositivo háptico, y permite recalcular el camino a seguir cada vez que se presiona el botón del dispositivo háptico.

El comportamiento dependerá del algoritmo establecido en la ventana de Parámetros del RRT.

RRT_GoalBias: Se activa la opción de fijar "n" puntos al presionar del botón del dispositivo háptico, una vez fijado el último punto, automáticamente se calcula el camino

RRT_Dual: Se activa la opción de calcular el camino desde la posición actual del dispositivo háptico, al presionar el botón del háptico se fija el punto y se calcula inmediatamente el camino.

5 PRUEBAS Y RESULTADOS.

Para la prueba del sistema de guiado se definen 8 archivos X3D, con diferentes configuraciones y diferente número de obstáculos.

Las pruebas a realizar consisten en:

- ✓ Fijar los puntos inicial $\mathbf{x}_{ini}$ en (0,0,0) y $\mathbf{x}_{goal}$ en (1,1,1) para todas las pruebas.
- ✓ Variar el valor del épsilon entre 0.1 ,0.25 ,0.5 y hacer 10 simulaciones para cada variación.
- ✓ Registrar los valores de tiempo y número de nodos empleado, obtener el promedio y registrar este valor en las tablas de pruebas.
- ✓ El número de interacciones es fijo en 1000.

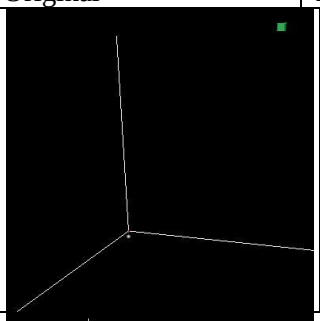
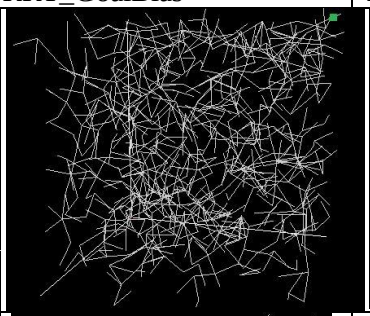
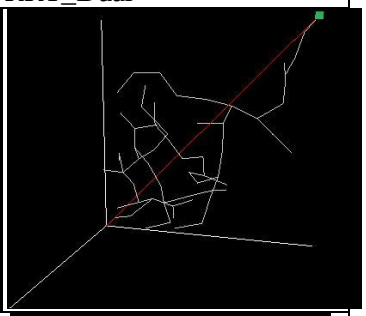
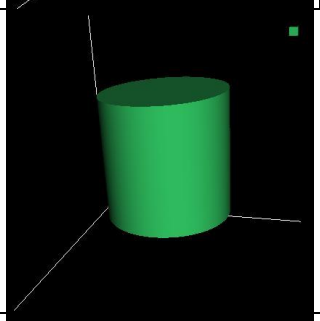
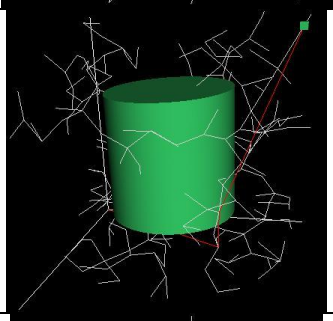
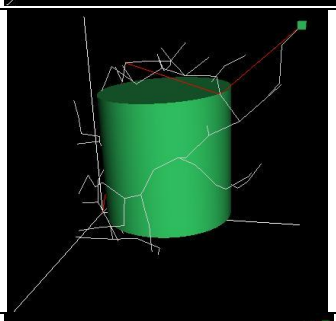
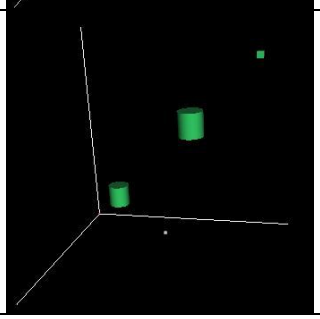
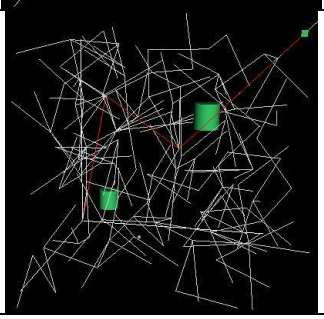
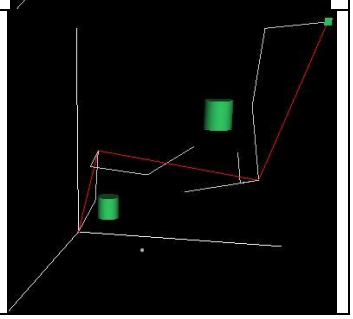
5.1 Prueba de los algoritmos RRT_GoalBias y RRTDual.

ARCHIVO File 00	**RRT_GOALBIAS**			**RRT_DUAL**		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	2.70	1001	100	0.015	50	0
epsilon(0.1)	1,97	1001	100	0.015	50	0
epsilon(0.5)	2,02	1001	100	< 0.001	6	0
ARCHIVO File 01	**RRT_GOALBIAS**			**RRT_DUAL**		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	0,080	73	0	< 0.001	8	0
epsilon(0.1)	0,097	226	10	0,0016	43	0
epsilon(0.5)	0,040	293	0	< 0.001	10	0
ARCHIVO File 02	RRT_GOALBIAS			RRT_DUAL		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	0,440	307	0	< 0.001	18	0
epsilon(0.1)	0,064	192	60	0,0016	56	0
epsilon(0.5)	0,007	112	0	< 0.001	7	0
ARCHIVO File 03	RRT_GOALBIAS			RRT_DUAL		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	0.030	83	20	0.051	175	0
epsilon(0.1)	0.030	90	20	0.056	165	0
epsilon(0.5)	0.003	11	0	0.004	13	0
ARCHIVO File 04	RRT_GOALBIAS			RRT_DUAL		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	0.040	30	0	0.047	42	0
epsilon(0.1)	0.130	86	0	0.210	58	0
epsilon(0.5)	0.014	13	0	0.012	12	0
ARCHIVO File 05	RRT_GOALBIAS			RRT_DUAL		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	0.054	101	0	0.090	27	0

epsilon(0.1)	0.460	376	0	0.018	61	0
epsilon(0.5)	0.010	35	0	0.002	8	0
ARCHIVO File 06	RRT_GOALBIAS			RRT_DUAL		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	0.113	177	0	0.016	58	0
epsilon(0.1)	0.119	178	0	0.013	55	0
epsilon(0.5)	0.007	20	0	0.002	9	0
ARCHIVO File 07	RRT_GOALBIAS			RRT_DUAL		
	Tiempo (s)	Num nodos	%falla	Tiempo(s)	Num nodos	%falla
epsilon(0.25)	0.025	62	20	0.051	174	0
epsilon(0.1)	0.034	90	20	0.056	175	0
epsilon(0.5)	0.003	10	0	0.004	13	0

Tabla 5.1. Comparación de tiempos y nodos entre los algoritmos RRT

En las tabla5.1 se observa que el algoritmo RRT_DUAL, usa menos tiempo para calcular el árbol, emplea menor cantidad de nodos, y tiene un porcentaje de falla bajo.

Archivo	Original	RRT_GoalBias	RRT_Dual
FILE00			
FILE01			
FILE02			

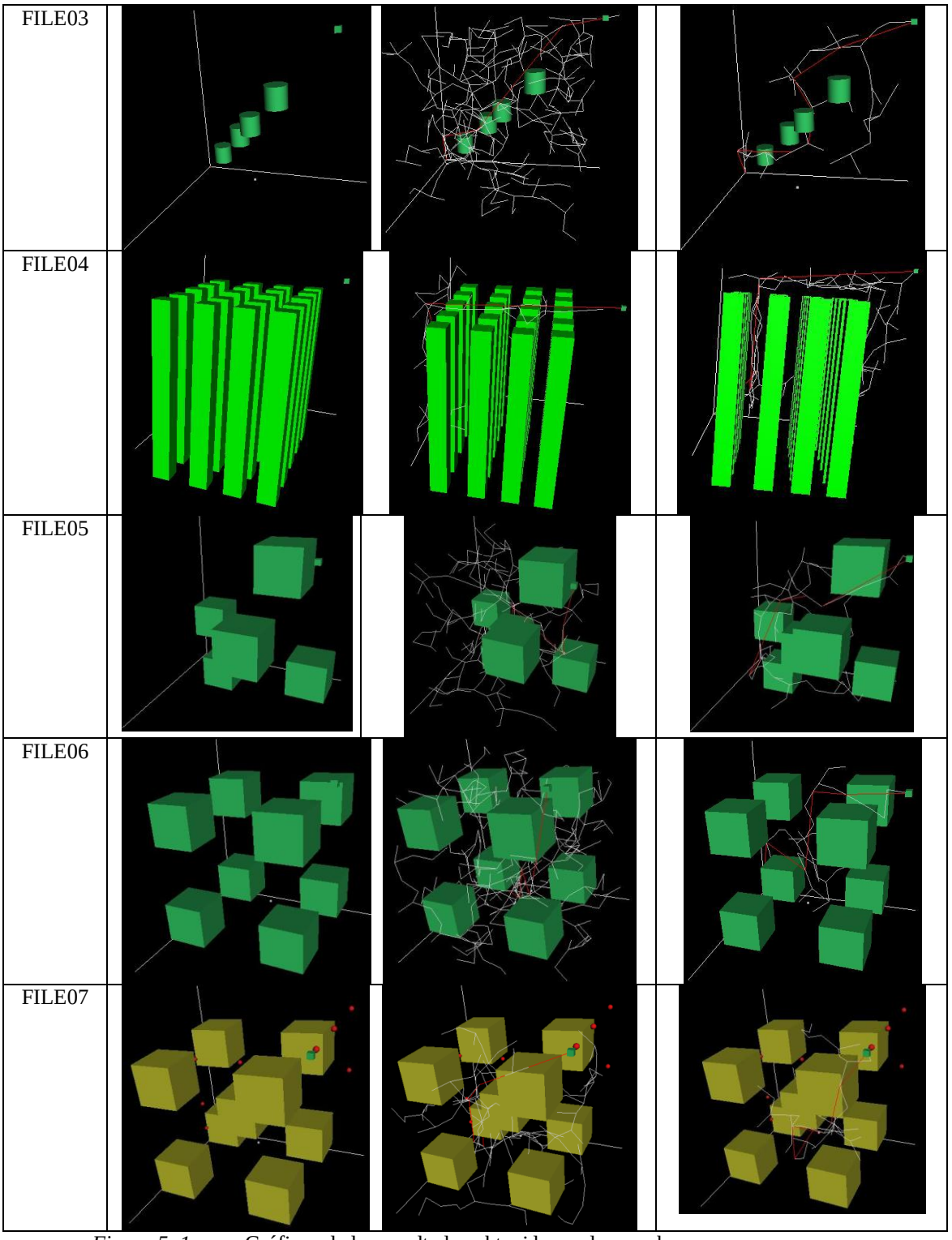


Figura 5. 1. Gráficos de los resultados obtenidos en las pruebas.

En la figura 5.1 se muestra los resultados obtenidos al aplicar el algoritmo a diferentes escenarios, se aprecia siempre la menor cantidad de nodos usados por el algoritmo

RRT_Dual. En la aplicación el camino final obtenido no es eficiente, se deberá mejorar el algoritmo de post procesado para poder optimizar el camino resultante.

5.2 Pruebas entre RRT_GoalBias y el algoritmo modificado.

Se usan 10 puntos de paso para esta prueba**.**

ARCHIVO File 07	**RRT_GOALBIAS**		RRT GOALBIAS modificado	
	Tiempo (s)	Num nodos	Tiempo(s)	Num nodos
x_{ini}(0,0,0) epsilon(0.25)	0.025	62	0,013	35
x_{ini}(0.0,0.0,0.0) epsilon(0.1)	0.034	90	0,097	88
x_{ini}(0,0,0) epsilon(0.5)	0.003	10	0.001	8

Tabla 5.2. Comparación de los algoritmos RRT_GoalBias

En la tabla 5.2 se muestra el resultado del algoritmo RRT_GoalBias modificado, el algoritmo mejora en un 50% el tiempo empleado y en un 20% el número de nodos obtenidos para el camino.

PUNTOS	**Tiempo (s)**	**Num Nodos**
1	0.068	93
2	0.170	93
3	0.066	57
4	0.048	56
5	0.033	66
6	0.036	62
7	0.150	168
8	0.069	93
9	0.031	44
10	0.097	88

Tabla 5.3. Resultado variando el número de puntos de paso. (épsilon 0.10)

En la tabla 5.3 se ha probado el algoritmo RRT_GoalBias modificado, variando el número de puntos usados como paso.

Resultado de la modificación del Algoritmo RRT modificado.

RRT_GoalBias original	RRT_GoalBias modificado

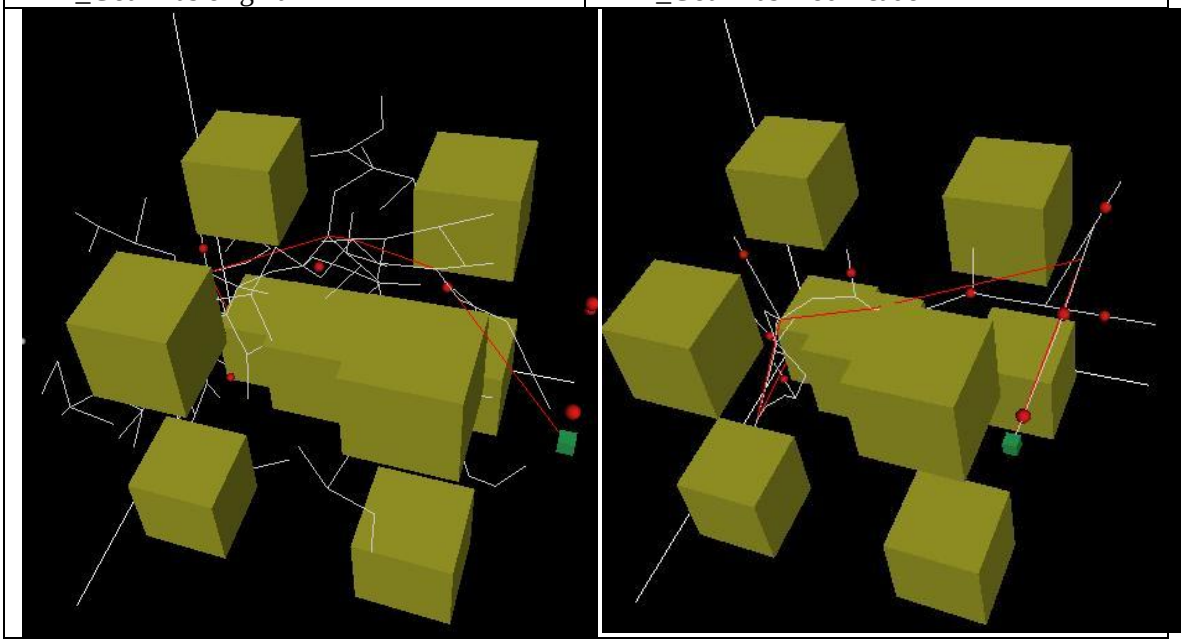


Figura 5. 2. Comparación entre el algoritmo original del RRT y el modificado.

El algoritmo RRT_modificado permite sugerir un recorrido a ser seguido por el planificador, en la figura 5.2 se puede ver, cuando se usan el algoritmo modificado los puntos (en rojo) forman parte del árbol, pero no precisamente del camino final a seguir.

5.3 Pruebas de seguimiento: camino sin asistencia y con asistencia.

ARCHIVO	SIN ASISTENCIA t(s)	CON ASISTENCIA t(s)
FILE00	2	1
FILE01	7.9	2.3
FILE02	5	3.0
FILE03	20	7.5
FILE05	23	5
FILE06	35	6.5

Tabla 1.1. Tiempos empleados en el seguimiento.

En la tabla 5.6 se muestra una estimación de tiempos empleados para seguir el camino planificado por el RRT, la característica “magnetic” empleada en el sistema reduce en un 30% el tiempo requerido para recorrer el trayecto; se debe tomar en cuenta que la ruta planificada no siempre es la mejor.

6 CONCLUSIONES Y TRABAJOS FUTUROS.

Se ha conseguido unificar bajo una sola librería las tareas de la representación gráfica, la representación háptica, y la detección de colisiones.

No obstante actualmente la librería de colisiones esta limitada a usar como objeto móvil una esfera.

Se ha establecido una arquitectura basada en el API H3D, para desarrollar aplicaciones de teleoperación y realidad virtual, en espacios tridimensionales, con la característica de ser multiplataforma y de software libre.

Una de las ventajas del API H3D al usar X3D, es que al ser un estándar ISO de diseño la mayoría de los software para gráficos permiten grabar en formato X3D, y por lo tanto se pueden importar y usar en H3D. Esto permite al usuario añadir características hápticas a gráficos ya existentes.

En base a los resultados obtenidos con los algoritmos implementados, se concluye que el algoritmo RRT_DUAL, es el de mejor desempeño para obtener caminos en menos tiempo y con menor número de nodos.

El algoritmo RRT_GoalBias modificado permite fijar puntos para "sugerir" al algoritmo el camino por donde deberá pasar, esto no implica precisamente pasar por los puntos, sino una tendencia del camino a seguirlos.

La aplicación implementada constituye el módulo de asistencia de guiado asistido para tareas virtuales con dispositivos hápticos, permitirá al usuario la simulación en un entorno virtual tridimensional del camino a seguir.

TRABAJOS FUTUROS.

El problema con el manejo de colisiones no es trivial, requiere la programación de las clases y los procesos para poder trabajar con geometrías diferentes a una esfera. La forma como está diseñada la API H3D permitiría desarrollar en un futuro un nuevo conjunto de clases para manejar las colisiones con geometrías diferentes.

Actualmente la API H3D ha añadido un conjunto de nodos para realizar la simulación de cuerpo rígido, basado en la librería ODE (Open Dinamics Engine), con estos nodos, se podría simular completamente todo el escenario virtual, si se añade a los nodos la capacidad de "enganchar" el objeto al dispositivo háptico.

En el futuro se puede modificar el algoritmo RRT_Dual, para usarlo con puntos de paso.

7 Bibliografia

1. **Gomez de Gabriel, Jésus Manuel, Ollero Baturone, Anibal y Garcia Cerezo, Alfonso Jose.** *Teleoperación y Telerrobótica.* Madrid : Prentice Hall, 2006. pp 4-6.

2. **Rodriguez, Adolfo, y otros.** *A Multimodal Teleoperation Framework:Implementation and Experiments.* Barcelona : UPC, 2008. pp 177-182.

3. **Burdea, Gregory C.** *Force and touch feedback for virtual reality.* New York USA : John Wiley , 1996.

4. **Salisbury, T.H. Massie J.K.** *The Phantom haptic interface: a device for probing virtual objects.* Chicago : ASME, 1994. pp 2.

5. **Choset, Howie, y otros.** *Principles of Robot Motion Theory, Algorithms and Implementations.* London : MIT Press, 2005. pp 197-208.

6. **La Valle, Steven M.** *Rapidly-Exploring Random Trees: A new toll for path planning.* Iowa : Department Computer Science Iowa State University, 1998. pp 1-4.

7. **Kavraki, Lydia E, y otros.** *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* s.l. : IEEE Transactions on Robotics and Automation, 1996. Vols. Vol. 12, No. 4. pp 576-580.

8. **web3D.org.** The Virtual Reality Modelin Language. *VRML97 ISO/IEC 14772.* [En línea] web3d.org, 1997. http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/#International Standard ISO/IEC 14772-1:1997.

9. **Consortium, Web3D.** Web3D Consortium-Royalty Free Opens Standars for Real Time 3D Comunications . *Web3D Consortium.* [En línea] Web3D.org, 2005. http://www.web3d.org/.

10. **SIM, Kongsberg.** sim Coin3D graphics. *3D Graphics Development Tools.* [En línea] Kongsberg SIM AS, 2008. http://www.coin3d.org/.

11. **Technologies, Sensable.** SensAble. [En línea] SensAble Technologies, 2009. http://www.sensable.com/.

12. **Salamanca, María Luisa Pinto.** *Tutorial de uso de OHAE sobre windows.* Colombia : OpenSurg, 2009. pp 1-5.

13. **H3D.org.** http://www.h3dapi.org/. [En línea] H3D. http://www.h3dapi.org/.

14. **Expósito, Carlos Marrero.** *Interfaz Gráfica de Usuario Aproximación semiótica y cognitiva.* España : Universidad de la Laguna, 2006. pp 7-13.

15. **Garcia de Jalon, Javier, y otros.** *Aprenda Java como si estubiera en primero.* Navarra : Tecnun, 2000. pp 165-167.

16. **Corporation, Nokia.** Qt- A cross-platform application and UI framework. [En línea] Nokia , June de 2008. http://qt.nokia.com/.

17. **GTK.org.** The GTK+Project. [En línea] gtk.org. http://library.gnome.org/devel/gtk/stable/.

18. **Smart, Julian, Hock, Kevin y Csomor, Stefan.** *Cross-Platform GUI Programming with wxWidgets.* Illinois : Prentice Hall, 2005.

19. **Wilkie, David.** *Analysis of Collision Detection Library PQP.* Department of Computer Science : North Caroline, 2009. pp 2-5.

20. **Bergen, Gino Van Den.** *Collision Detection in Interactive 3D enviroment.* San Francisco : ELSEVIER, 2004.

21. **Perez, A. y Rosell, J.** *Planificación de Movimientos en Robotic:,curso práctico para implementar un PRM.* Barcelona : CEA-IFAC, 2008. pp 1-8.

22. **Latombe, Jean Claude.** *Robot Motion Planning.* Massachussetts : Kluwer Academics , 1991. pp 1-10.

23. **Canny, Jhon F.** *Complexity of Robot Motion Planning.* California : MIT Press, 1988.

24. **Perez, T Lozano.** *Spatial planning: A configuration approach.* s.l. : IEEE Transactions on, 1983. Vol. 35. pp 108-120.

25. **Freda, Luigi, Oriolo, Giuseppe y Vecchioli, Francesco.** *Sensor-based Exploration for General Robotic Systems.* Nice-Francia : IEEE International Conference on Intelligent Robots and Systems, 2008. pp 22-26.

26. **Amato, Nancy, Bazayit, Burchan y Dale, Lucia.** *OBPRM: An obstacle based prm for 3d workspaces.* Texas : Workshop on the Algorithmic foundations, 1998. pp 155-168.

27. **Saha, Mitul y Latombe, Jean Clause.** *On finding narrow passages with probabilistic roadmap planners.* Standford : Workshop on the Algorithmic Foundations of robotics, 1998. pp 1-24.

28. **Kuffner., S. M. LaValle and J. J.** *Randomized kinodynamic planning.* s.l. : Algorithmic and Computational Robotics:New Directions, 2201. Vol. 20. pp 378-400.

29. **Svetska., M. H. Overmars and P.** *A probabilistic learning approach to motion planning.* Netherlands : Utrecht University, 1995. pp 19-37.

30. **Bessiére, Pierre y Ahuactzin, Juan Manuel.** *The ariadne's clew algorithm: Global planning with local methods.* Yokohama : IEEE, 1993. pp 1373-1380.

31. **La Valle, Steven J y Kuffner, James.** *Rapidly-exploring random trees:Progress and prospects.* s.l. : Algorithmic and Computational Robotics: New Directions, 2000. pp 1-19.

32. **LaValle, Steven M.** *Planning Algorithms.* Illinois : Cambridge, 2006.

33. **Kuffner., S. M. LaValle and J. J.** *RRT-connect: An efficient approach to single-query path planning.* s.l. : IEEE:Robotics and Automation, 2000. pp 1-7.

34. **Geraerts, Roland y Overmars, Mark.** *Sampling and node adding in probabilistic roadmap planners.* Utrecht : ElSevier, 2005. pp 165-167.

35. **LaValle, Steve.** Motion Strategy Library. [En línea] University of Illinois., 2006. http://msl.cs.uiuc.edu/msl/.