Diseño de un Robot Tipo PA-10 virtual para Aplicaciones Quirúrgicas

Carlos Eduardo Fernández Riomalo
Héctor Andrés Guastar Morillo

Director: PhD Oscar Andrés Vivas Albán

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial
Popayán, 2014

Diseño de un Robot Tipo PA-10 virtual para Aplicaciones Quirúrgicas

Carlos Eduardo Fernández Riomalo
Héctor Andrés Guastar Morillo

Tesis presentada a la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca para la obtención del
Título de

Ingeniero en Automática Industrial

Director: PhD Oscar Andrés Vivas Albán

Popayán, 2014

Hoja de Aprobación

_____

_____

_____

Director _____
Phd. Oscar Andrés Vivas

Jurado _____

Jurado _____

Fecha de sustentación: Popayán, 2014

Agradecimientos.

Primero queremos agradecer a Dios, quien nos ha iluminado en el trayecto de este camino. A nuestros padres, hermanos y demás familiares que con su apoyo incondicional y sus consejos nos ayudaron a formarnos como personas de bien. A nuestro director de tesis Oscar Andrés Vivas Albán, quien nos colaboró intelectualmente durante el desarrollo de este proyecto. A nuestros amigos y compañeros de estudio con quienes pasamos buenos momentos, aprendimos y adquirimos habilidades para el ejercicio de la ingeniería. A la Universidad del Cauca, en especial a la facultad de Ingeniería Electrónica y Telecomunicaciones y a sus profesores que con su profesionalismo y su responsabilidad aportaron del tiempo de ellos para brindarnos su conocimiento y hacernos crecer como profesionales. Y a los evaluadores de este proyecto que con sus puntos de vista fomentan el crecimiento y la mejora continua.

Resumen

En este proyecto se aborda el diseño y simulación del robot PA-10 con el fin de ser implementado en aplicaciones quirúrgicas. Para ello, el robot se modela matemáticamente a nivel geométrico y dinámico con el objetivo de comprobar, si este puede utilizarse en procedimientos laparoscópicos, manteniendo un punto fijo en el espacio (trocar), que sirve como entrada a la cavidad abdominal.

Para demostrar que la estructura cinemática del brazo robótico respeta el paso por el trocar, se diseña un control por par calculado (CTC), el cual, permite obtener un error menor al orden de los milímetros para trayectorias definidas. Posteriormente, se desarrolla una interfaz gráfica que muestra la interacción de tres robots; un porta endoscopio (Hibou) y dos quirúrgicos (LapBot y PA-10). Los robots Hibou y LapBot han sido implementados en tesis anteriores.

Además de esto, en la aplicación se implementa un algoritmo de detección de colisiones y deformación, con el propósito de darle una mayor realidad a la aplicación, debido a que se puede observar la interacción del efector final del robot PA-10 con los órganos de la cavidad abdominal.

Finalmente se obtuvo el software SVRQ (Simulador Virtual de Robótica Quirúrgica) llamado así por sus desarrolladores, ya que este ambiente tridimensional, permite verificar el comportamiento de un robot ante situaciones a las que tendrá que enfrentarse en la realidad o para observar posibles fallas o posibles mejoras en la construcción de las piezas del prototipo físico.

Abstract

This project focuses on PA-10 robot design and simulator, in order to be applied in surgical procedures. To do so, robot is modeled mathematically at a geometric and dynamic level to prove if it can be used in laparoscopic procedures, keeping a fixed point on the space (trocar), that allows to get into the abdominal cavity.

To demonstrate that kinematic structure of the robot arm respects the pass through of trocar, it is designed a calculated torque controller (CTC), which lets to get a minor mistake towards the order of millimeters for predefined trajectories. Subsequently, a graphic interface is developed that shows interaction among three robots; an endoscopic holder (Hibou) and two surgical (LapBot and PA-10). Hibou and LapBot robots have been developed in previous thesis.

In addition, in the application a collision and deformation detection algorithm is implemented, with the purpose to make the application more real, due to the fact that the interaction between end effector PA-10 and abdominal cavity organs can be noted.

Finally got SVRQ software called by its developers, because this 3D environment, allows checking robot behavior in real situations it will have to face later, or noticing possible failures or improvements in the construction of the physical prototype's parts.

Tabla de contenido

1. Introducción 1
2. Robótica y su aplicación en la medicina. 2
   2.1. Cirugía Laparoscópica. 2
   2.2. Robots quirúrgicos. 3
      2.2.1. Robot PA-10. 5
   2.3. Simuladores quirúrgicos. 6
3. Simulación del robot PA-10. 7
   3.1. Diseño CAD del robot PA-10. 8
      3.1.1. Cálculo de los parámetros inerciales. 9
   3.2. Simulación del robot PA-10 en MATLAB – Simulink. 14
      3.2.1. Proceso de optimización. 15
      3.2.2. MGI del robot PA-10. 17
      3.2.3. Prueba del MGI y MGD. 19
      3.2.4. Simulación del robot PA-10 en *Virtual Reality*. 23
4. Software para manipulación del robot PA-10. 27
   4.1. Alcances y restricciones del software. 27
      4.1.1. Alcances 28
      4.1.2. Restricciones. 28
      4.1.3. Blender. 28
      4.1.4. Librería para interfaz gráfica VTK. 29
      4.1.5. Plataforma de desarrollo Visual Studio. 29
      4.1.6. QT Designer. 30
      4.1.7. Librerias V-Collide y RAPID. 30
   4.2. Modelado *UML*. 31
      4.2.1. Diagrama de casos de usos. 32
      4.2.2. Diagrama de clases. 33
   4.3. Clases desarrolladas para la implementación del robot PA-10. 38
      4.3.1. Clase “ObjetoActor”. 38
      4.3.2. Clase “Robot PA-10”. 39
      4.3.3. Clase “m2c”. 42

5. Algoritmo de detección de colisiones y deformación. 43
5.1. Acondicionamiento de los objetos a colisionar. 44
5.2. Posición de los objetos a colisionar. 44
5.3. Detección de colisiones y deformación. 51
5.3.1. Componente de colisión. 52
5.3.2. Componente de deformación. 53
6. Pruebas de la aplicación. 58
6.1. Manipulación del robot. 59
6.2. Deformación de órganos. 63
7. Conclusiones y trabajos futuros. 67
7.1. Conclusiones. 67
7.2. Trabajos futuros. 68
8. Referencias bibliográficas. 69
A. Anexo A: Instalación de herramientas software. 72
1. Instalación de Visual Studio 2008. 72
2. Instalación de QT 73
3. Instalación de CMake. 77
4. Instalación de la librería VTK. 78
5. Instalación del SDL, V-Collide y RAPID. 81
B. Anexo B: Uso de Solid Edge, Blender y VTK para el ensamblado del robot. 84
C. Anexo C: Manual de Usuario. 91

Listado de tablas.

Tabla 3-1. Parámetros geométricos del robot PA-10. 7
Tabla 3-2. Masas de cada articulación. 12
Tabla 3-3. Distancia a los centros de masas de cada articulación. 12
Tabla 3-4. Primer momento de inercia de cada articulación en los ejes X, Y y Z. 13
Tabla 3-5. Segundo momento de inercia de cada articulación en los ejes X, Y y Z. 14
Tabla B-1. Comandos de Blender. 87

Tabla de ilustraciones.

Figura 2-1. Cirugía laparoscópica. 3
Figura 2-2. Robot ZEUS. 4
Figura 2-3. Robot Da Vinci. 4
Figura 2-4. Robots Hibou y LapBot. 5
Figura 2-5. Robot PA-10. 6
Figura 2-6. Simulador Lap Mentor. 6
Figura 3-1. Robot PA-10 - Articulaciones y Ejes. 7
Figura 3-2. Actuador diseñado en Solid Edge. 8
Figura 3-3. Brazo robótico ensamblado. 9
Figura 3-4. Opción sistema de coordenadas. 10
Figura 3-5. Orientación mediante “Teclado”. 10
Figura 3-6. Robot PA-10 con sus ejes de rotación. 11
Figura 3-7. Herramienta verificar y propiedades físicas. 11
Figura 3-8. Propiedades físicas del conjunto. 12
Figura 3-9. Segundo momento de inercia. 13
Figura 3-10. Esquema de optimización. 17
Figura 3-11. Comportamiento del Robot Tipo PA-10 con el proceso de optimización. 17
Figura 3-12. Prueba del MGI y MGD del Robot Tipo PA-10. 19
Figura 3-13. Subsistema Visual PA-10. 20
Figura 3-14. Comportamiento del robot PA-10 con MGI, MGD y consigna circular. 21
Figura 3-15. Error cartesiano del robot PA-10 con MGI y MGD. 21
Figura 3-16. Esquema de control CTC. 22
Figura 3-17. Consigna circular. 22
Figura 3-18. Error cartesiano del robot PA-10 con control CTC. 23
Figura 3-19. Importando objetos en formato *.wrl. 24
Figura 3-20. Traslación del hombro en el nodo *Transform*. 25
Figura 3-21. Robot PA-10 ensamblado en herramienta *3D World Editor* de *Simulink*. 25
Figura 3-22. Simulación robot PA-10 utilizando consigna circular. 26
Figura 4-1. Software base para este proyecto. 27
Figura 4-2. Uso de Blender para el modelado del Robot PA-10. 29
Figura 4-3. Uso de QT para el desarrollo de las *GUI*. 30
Figura 4-4. Diagrama de casos de usos al inicializar la aplicación. 32
Figura 4-5. Diagrama de casos de usos en escenario quirúrgico. 32
Figura 4-6. Diagrama de casos de usos en escenario de colisión y deformación. 33
Figura 4-7. Diagrama de clases general para la aplicación SVRQ. 33
Figura 4-8. Diagrama detallado de clases parte 1. 34
Figura 4-9. Diagrama detallado de clases parte 2. 35
Figura 4-10. Robot PA-10 ensamblado en VTK. 41
Figura 5-1. Visualización y deformación de objetos virtuales 3D. 43
Figura 5-2. Área de trabajo del órgano terminal. 45

Figura 5-3. Visualización en SVRQ y Blender. 46
Figura 5-4. Planos cartesianos en SVRQ y Blender. 46
Figura 5-5. Desplazamiento en Ymax (SVRQ) y en Xbmin (Blender). 47
Figura 5-6. Desplazamiento en Ymin (SVRQ) y en Xbmax (Blender). 47
Figura 5-7. Desplazamiento en Ymax (SVRQ) y en Ybmax (Blender). 48
Figura 5-8. Desplazamiento en Ymin (SVRQ) y en Ybmin (Blender). 48
Figura 5-9. Desplazamiento en Zmax (SVRQ) y en Zbmax (Blender). 49
Figura 5-10. Desplazamiento en Zmin (SVRQ) y en Zbmin (Blender) 49
Figura 5-11. Inversión del eje Y de SVRQ. 50
Figura 5-12. Análisis de los planos cartesianos de Blender y de vtkMatrix4x4. 51
Figura 5-13.Propagación de movimiento en un mallado triangular. 54
Figura 5-14. Unión de resortes en serie y paralelo. 55
Figura 6-1. El efector final y el *joystick* no realizan ningún movimiento. 59
Figura 6-2. El efector final y el *joystick* se mueven hacia la derecha 60
Figura 6-3. El efector final y el *joystick* se mueven hacia la izquierda. 60
Figura 6-4. El efector final y el *joystick* se mueven hacia atrás. 61
Figura 6-5. El efector final y el *joystick* se mueven hacia adelante. 61
Figura 6-6. El efector final y el *joystick* se mueven hacia abajo. 62
Figura 6-7. El efector final y el *joystick* se mueven arriba. 62
Figura 6-8. Deformación del higado con el instrumento quirúrgico "Tijera". 63
Figura 6-9. Deformación del estómago con el instrumento quirúrgico "Tijera". 64
Figura 6-10. Deformación de la vesícula con el instrumento quirúrgico "Tijera". 64
Figura 6-11. Deformación de la cística con el instrumento quirúrgico "Retractor". 65
Figura 6-12. Deformación del estómago con el instrumento quirúrgico "Pistola Endoclip". 65
Figura 6-13. Ubicación de la "Pistola Endoclip" para colocar los endoclip. 66
Figura 6-14. Endoclips ubicados. 66
Figura A-1. Instalación de Visual Studio 2008 (Paso 1). 72
Figura A-2. Instalación de Visual Studio 2008 (Paso 2). 72
Figura A-3. Instalación de Visual Studio 2008 (Paso 3). 72
Figura A-4. Instalación de Visual Studio 2008 (Paso 4). 72
Figura A-5. Instalación de Visual Studio 2008 (Paso 5). 73
Figura A-6. Instalación de Visual Studio 2008 (Paso 6). 73
Figura A-7. Instalación de Visual Studio 2008 (Paso 7). 73
Figura A-8. Instalación de QT (Paso 1). 74
Figura A-9. Instalación de QT (Paso 2). 74
Figura A-10. Instalación de QT (Paso 3). 74
Figura A-11. Instalación de QT (Paso 4). 74
Figura A-12. Instalación de QT (Paso 5). 74
Figura A-13. Instalación de QT (Paso 6). 74
Figura A-14. Instalación de QT (Paso 7). 75
Figura A-15. Instalación de QT (Paso 8). 75
Figura A-16. Instalación de QT (Paso 9). 75

Figura A-17. Instalación de QT (Paso 10). 75
Figura A-18. Instalación de QT (Paso 11). 75
Figura A-19. Instalación de QT (Paso 12). 76
Figura A-20. Instalación de CMake (Paso 1). 77
Figura A-21. Instalación de CMake (Paso 2). 77
Figura A-22. Instalación de CMake (Paso 3). 77
Figura A-23. Instalación de CMake (Paso 4). 77
Figura A-24. Instalación de CMake (Paso 5). 77
Figura A-25. Instalación de CMake (Paso 6). 77
Figura A-26. Instalación de la librería VTK (Paso 1). 78
Figura A-27. Instalación de la librería VTK (Paso 2). 78
Figura A-28. Instalación de la librería VTK (Paso 3). 79
Figura A-29. Instalación de la librería VTK (Paso 4). 79
Figura A-30. Instalación de la librería VTK (Paso 5). 80
Figura A-31. Instalación de la librería VTK (Paso 6). 80
Figura A-32. Instalación de la librería VTK (Paso 7). 80
Figura A-33. Instalación de la librería VTK (Paso 8). 80
Figura A-34. Instalación de la librería VTK (Paso 9). 81
Figura A-35. Instalación de la librería VTK (Paso 10). 81
Figura A-36. Instalación de la librería VTK (Paso 11). 81
Figura A-37. Instalación de la librería VTK (Paso 12). 81
Figura A-38. Instalación de la librería V-Collide y RAPID (Paso 4). 82
Figura A-39. Instalación de V-Collide y RAPID (Paso 5) 82
Figura A-40. Instalación de V-Collide y RAPID (Paso 6). 83
Figura B-1. Guardar pieza en formato *.wrl (Paso 1). 84
Figura B-2. Guardar pieza en formato *.wrl (Paso 2). 84
Figura B-3. Importando objetos en Blender (Paso 1). 85
Figura B-4. Importando objetos en Blender (Paso 2). 85
Figura B-5. Pieza unida en Blender. 86
Figura B-6. Exportar objeto en Blender. 86
Figura B-7. Desplazamiento del objeto mediante el comando G. 87
Figura B-8. Ensamblando robot en Blender. 88
Figura B-9. Pieza brazo. 88
Figura B-10. Robot ensamblado. 89
Figura B-11. Robot ensamblado y método Mover_Robot activo. 90
Figura C-1. Inicio de la aplicación. 91
Figura C-2. Mensaje de atención. 91
Figura C-3. Interfaz gráfica de SVRQ. 92
Figura C-4. Barra de herramientas. 92
Figura C-5. Opción Dispositivo. 92
Figura C-6. Opción Ayuda. 92
Figura C-7. Opción Acerca De. 93

Figura C-8. Selección de Escenario. 93
Figura C-9. Escenario Quirúrgico. 94
Figura C-10. Instrumentos quirúrgicos. 94
Figura C-11. Cámara del robot porta endoscopio HIBOU. 94
Figura C-12. Escenario Colisión y Deformación. 95
Figura C-13. Deformación sobre el estómago. 95
Figura C-14. Posicionamiento pistola endoclip. 96
Figura C-15. Endoclip colocados. 97
Figura C-16. Opción salir. 97

1. Introducción

Con el paso de los años el uso de la robótica en el campo de la medicina se ha incrementado, la precisión y exactitud que otorga el uso de esta tecnología ha brindado grandes ventajas en cirugía cardiaca, gastrointestinal, pediátrica, neurocirugía y la cirugía mínimamente invasiva. Este tipo de tecnología es particularmente importante en un ambiente quirúrgico, donde es primordial contar con bajas fuerzas de interacción, a la hora de hacer contacto con los tejidos y órganos, evitando perjuicios a los pacientes y el personal quirúrgico.

Actualmente se realizan varias operaciones con robots, el robot quirúrgico más ampliamente utilizado hasta el día de hoy es el Da Vinci, pero su alto costo lo convierte en un robot de difícil acceso para ser utilizado con fines de entrenamiento quirúrgico.

Por otro lado, para la práctica de cirugías laparoscopias se han desarrollado diferentes simuladores quirúrgicos, los cuales buscan ayudar en la formación de estudiantes de medicina llevándolos a realizar procedimientos de cirugía laparoscopia con pacientes virtuales en diferentes escenarios.

El propósito de este trabajo es diseñar un prototipo del robot PA-10 para ser utilizado en aplicaciones quirúrgicas, considerando todas sus características físicas y su comportamiento bajo un ambiente quirúrgico. La prueba del robot PA-10 se lleva a cabo dentro de un software desarrollado el cual permite al usuario interactuar con el robot, y observar todos los movimientos de este, respetando la incisión en la cavidad abdominal.

El trabajo está organizado en siete secciones, en la segunda sección, se hace una breve descripción sobre los robots utilizados en cirugías laparoscópicas y el software más usado para entrenamiento quirúrgico. En la tercera sección, se muestra el diseño y modelado del robot PA-10. En la cuarta sección se presenta el software desarrollado para ejecutar los movimientos del robot. En la quinta sección se encuentra el algoritmo implementado para que el efector final del robot PA-10 pueda interactuar con los órganos de la cavidad abdominal. En la sexta sección se presentan las pruebas que se llevaron a cabo para verificar el comportamiento del robot bajo un ambiente quirúrgico. Y finalmente en la séptima sección se presentan las conclusiones del trabajo realizado.

2. Robótica y su aplicación en la medicina.

La robótica surge debido a la necesidad de facilitar y mejorar las operaciones realizadas en el campo industrial, transformando la forma de trabajar y producir, sin embargo los robots industriales eran solo conocidos por quienes los operaban. Debido a las grandes ventajas de la utilización de robots para tareas complejas y de alta precisión se desarrollaron robots dedicados a participar como asistentes en procedimientos quirúrgicos, entre estos procedimientos se destacan la tele-cirugía, la cirugía mínimamente invasiva (CIS), la cirugía cardiaca, la gastrointestinal, la pediátrica y la neurocirugía [1].

Esta rama de la robótica quirúrgica junto con la robótica de rehabilitación, forman la rama de la robótica médica que incluye; las prótesis robóticas, los exoesqueletos, los robots asistentes en rehabilitación física, entre otros.

La robótica médica se empieza a desarrollar cuando los médicos e investigadores detectan los diferentes problemas que se presentan en los procedimientos quirúrgicos y de rehabilitación, poniendo en riesgo la vida del paciente y demorando el proceso de recuperación. Para solucionar estos problemas se hace uso de la tecnología desarrollando prótesis, sistemas de recuperación robóticos y robots quirúrgicos que han mejorado notablemente la realización de tratamientos y operaciones, capacitando a la vez el personal médico para poder manejar estos instrumentos tecnológicos.

2.1. Cirugía Laparoscópica.

Para tener un mayor conocimiento de la cirugía laparoscópica se asistió a una intervención quirúrgica de colecistectomía[1]. En ella se pudo observar los procedimientos que se efectúan para llevar a cabo esta cirugía. Primero se realizó una pequeña incisión en el abdomen para inflar la cavidad abdominal, con el fin de separar la pared abdominal de los órganos, elevando el abdomen para que el cirujano tenga acceso al interior del paciente y pueda manipular los instrumentos con relativa facilidad. La elevación se realiza mediante insuflación empleando CO_2[2], dado que suprime la combustión y el cuerpo lo absorbe con más facilidad en comparación a la insuflación con aire. Posteriormente se insertan unos tubos cortos y delgados (trocares) en el

[1] Colecistectomía: Intervención que se realiza para la extracción de la vesícula biliar que presenta inflamación o se encuentra obstruida por cálculos biliares.
[2] CO_2: Dióxido de Carbono.

abdomen. A través de estos trocares se introducen instrumentos largos y angostos, que el cirujano utiliza para manipular, cortar y coser tejidos (Figura 2-1).

Este tipo de cirugía beneficia en grandes proporciones al paciente, ya que después de someterse a una operación de estas su recuperación va hacer más rápida y su estancia en el hospital más corta, debido a que los órganos y tejidos operados no se lesionan tanto, gracias a la precisión en la operación que permite la cirugía laparoscópica.

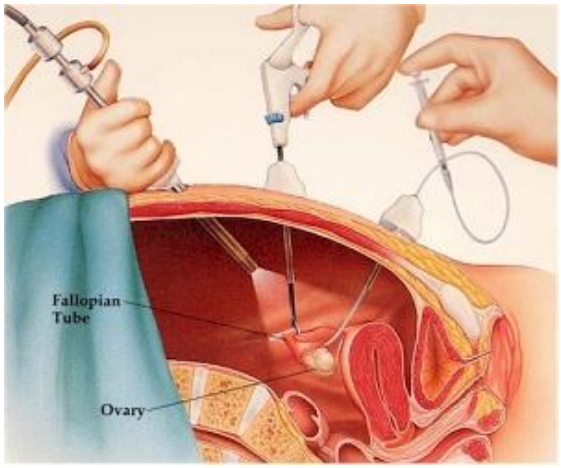

Figura 2-1. Cirugía laparoscópica.

Con el análisis realizado anteriormente, se tiene una perspectiva general del procedimiento quirúrgico, donde se destacan ciertos factores: el acondicionamiento del paciente, la precisión del cirujano para llevar a cabo la cirugía, el manejo del endoscopio que permite la visualización al interior del abdomen y los instrumentos quirúrgicos utilizados. Con base a esto, se pretende implementar un software de robótica quirúrgica que simule el comportamiento de los robots frente a situaciones que deberán enfrentar en la realidad.

2.2. Robots quirúrgicos.

En los últimos años el desarrollo de robots ha superado las expectativas en el campo médico. Los robots actuales presentan una mayor complejidad en su construcción y mayor eficiencia en su funcionamiento. Los robots más importantes en el campo de la robótica quirúrgica son: el robot ZEUS y el Da Vinci.

El robot ZEUS (Figura 2-2), fue creado en 1998 por la compañía *Computer Motion* de California con el propósito de realizar cirugías a distancia con

instrumentos laparoscópicos, su plataforma se monta sobre la mesa de operaciones y es modular, donde uno de sus módulos es el robot AESOP que opera la cámara. Tiene 5 grados de libertad y su bajo peso lo hace portable [2].

Figura 2-2. Robot ZEUS.

El Da Vinci *Surgical System* (Figura 2-3), fue creado en 1998 pocos meses después del ZEUS, por la compañía *Intuitive Surgical*, obtuvo la licencia de la *FDA*[3] en 1999 y es el robot quirúrgico más sofisticado que se tiene hasta el momento [3]. Desde que se autorizó su funcionamiento se han realizado miles de operaciones pues hay más de 1000 equipos instalados en los hospitales del mundo. Este robot es un sistema *on-line* del tipo amo-esclavo que consta de una consola quirúrgica computarizada conectada a un manipulador endoscópico con tres o cuatro brazos robóticos para manejar los instrumentos. Esta manipulación a través de una consola permite filtrar el temblor del cirujano y ampliar o reducir sus movimientos [4].

Figura 2-3. Robot Da Vinci.

[3] *FDA: Food and Drugs Administrator.*

La Universidad del Cauca a la vanguardia del desarrollo tecnológico ha desarrollado dos robots que satisfacen esta idea (Figura 2-4). Uno de ellos es el robot porta endoscopio Hibou, diseñado con siete grados de libertad, modelado a nivel geométrico, dinámico, y simulado en un ambiente tridimensional, que permite seguir con eficiencia las trayectorias que se necesitan en procedimientos laparoscópicos, manteniendo un punto fijo en el espacio para la rotación del efector final que se introduce al paciente, respetando el paso por la incisión [5].

Por otra parte, en el año 2009 se diseñó el robot LapBot, especialmente orientado al proceso quirúrgico de una colecistectomía. Este robot posee una arquitectura de nueve grados de libertad, con seis articulaciones activas que permiten posicionar y orientar el instrumento quirúrgico en un espacio tridimensional; y tres pasivas, que permiten mantener el punto fijo de movimiento sobre el punto de incisión donde se encuentra el trocar. Con estos tres grados de libertad se garantiza que el robot es intrínsecamente seguro y que no dañará al paciente por un esfuerzo agresivo en la incisión [6].

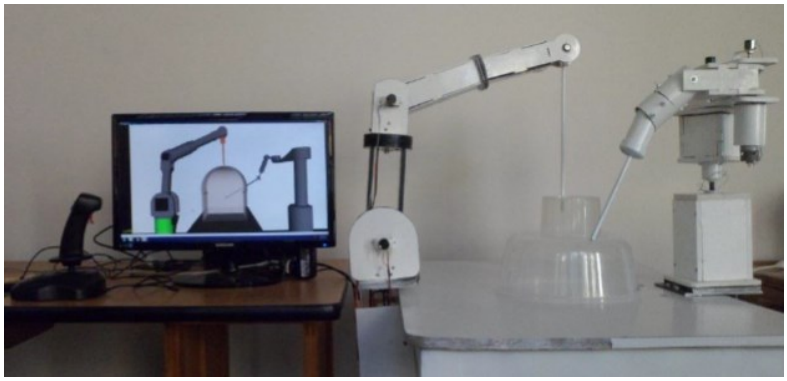

Figura 2-4. Robots Hibou y LapBot.

2.2.1. Robot PA-10.

En un trabajo de grado realizado en la Universidad del Cauca “Estudio del desempeño de un robot PUMA en operaciones de laparoscopia”, desarrollado por los ingenieros Oscar Mora y Bernardo Garcés, se demostró que el robot PUMA no es conveniente utilizarlo en este tipo de cirugías, debido a que sus características geométricas no permiten que el robot respete el paso por el trocar. En este mismo trabajo se plantea como solución la implementación del robot PA-10, ya que este tiene dos articulaciones que permiten posicionar la

muñeca en los tres ejes, el robot PUMA tan solo tiene una que posiciona a la muñeca solo en un plano [7].

El robot PA-10 (Figura 2-5), es un brazo robótico de siete grados de libertad, con siete articulaciones rotoides, cuyos accionamientos son motores *AC*[4]. Es un manipulador con muñeca de ejes concurrentes fabricado por *Mitsubishi Heavy Industries,* que sirve a la investigación en el ámbito de la robótica industrial, centrándose en la logística y los escenarios de producción [8].

Figura 2-5. Robot PA-10.

2.3. Simuladores quirúrgicos.

En cuanto a los simuladores quirúrgicos utilizados para cirugía laparoscópica, se encuentran gran variedad de soluciones, entre las que se destaca el Lap Mentor (Figura 2-6), el cual proporciona el más alto nivel de formación quirúrgica, pues proporciona una solución completa para la formación de los alumnos de todos los niveles y en todas las disciplinas [9]. El sistema incluye diferentes procedimientos de laparoscopia y pacientes en diferentes escenarios. Además incluye múltiples procedimientos quirúrgicos (incluyendo ginecología, urología y cirugía general).

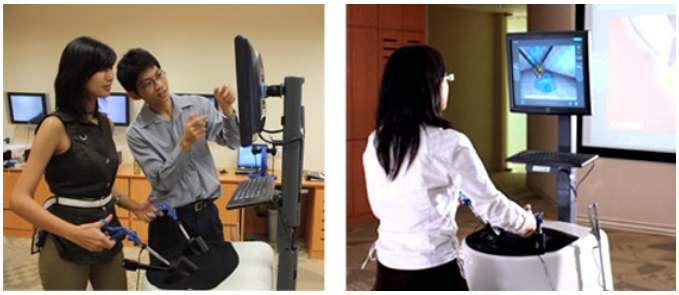

Figura 2-6. Simulador Lap Mentor.

[4] *AC: Alternating current.*

3. Simulación del robot PA-10.

Para realizar la simulación del robot PA-10, es necesario hallar los parámetros geométricos del robot (Figura 3-1). Al analizar las relaciones entre ejes y articulaciones se obtiene la tabla de parámetros (Tabla 3-1).

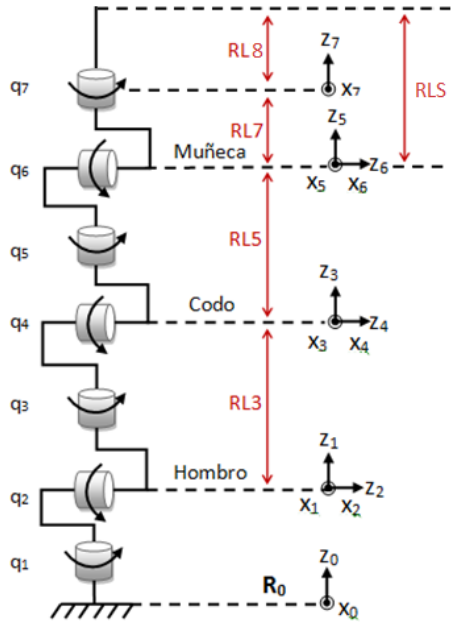

Figura 3-1. Robot PA-10 - Articulaciones y Ejes.

j	σ_j	α_j	d_j	θ_j	r_j
1	0	0	0	q_1	0
2	0	-90	0	q_2	0
3	0	90	0	q_3	RL3
4	0	-90	0	q_4	0
5	0	90	0	q_5	RL5
6	0	-90	0	q_6	0
7	0	90	0	q_7	RL7
8	0	0	0	0	RL8

Tabla 3-1. Parámetros geométricos del robot PA-10.

Dónde:

j: representa el número de la articulación o cuerpo.
σj: indica si la articulación es de traslación (1) o de rotación (0).
αj, ***θj***: son ángulos que dependen de los ejes de las articulaciones o de la rotación de una articulación rotoide.
dj, ***rj***: son distancias entre los ejes de las articulaciones o el desplazamiento de una articulación prismática.

Posteriormente con la utilización del software *Symoro* [10] se calculan los modelos dinámicos y geométricos que simulan el comportamiento del robot. Antes de realizar la simulación del robot en MATLAB – Simulink, se procede a diseñar el robot PA-10 mediante un software CAD[5] (*Solid Edge*). Este diseño se realiza para obtener los valores del primer y segundo momento de inercia que luego serán introducidos en los modelos dinámicos.

3.1. Diseño CAD del robot PA-10.

El diseño del robot PA-10 se realiza utilizando el software de diseño CAD Solid Edge, este permite la representación de piezas y conjuntos en 3D así como la generación de planos en 2D. Solid Edge admite trabajar sobre los siguientes tipos de entorno: pieza, conjunto, chapa, soldadura y plano [11]. Para este trabajo se empieza el diseño en el entorno pieza dándole la forma deseada a cada uno de los componentes del robot. Posteriormente se ensamblan las piezas en el entorno conjunto obteniendo un diseño preliminar del PA-10 que se utilizará en el simulador virtual de cirugía laparoscópica.

Se debe considerar el material con el cual se va a diseñar el prototipo, dado que este es determinante a la hora de calcular los parámetros inerciales pertenecientes al modelo dinámico del robot, por esta razón se diseñan los motores con un peso y unas dimensiones aproximadas a las características brindadas por un fabricante (Figura 3-2).

Figura 3-2. Actuador diseñado en Solid Edge.

[5] *CAD: Computer-Aided Design.*

En el entorno pieza, se diseñan cada uno de los componentes que conforman la estructura del robot y se les adiciona el material respectivo, para una futura construcción. Así, para las piezas de las articulaciones se selecciona acrílico como el material que soportará toda la estructura. La construcción de la estructura completa del robot mediante Solid Edge se realiza de forma secuencial en el entorno conjunto, empezando desde la primera articulación hasta el efector final (Figura 3-3), el cual atraviesa el trocar.

Figura 3-3. Brazo robótico ensamblado.

3.1.1. Cálculo de los parámetros inerciales.

Para simular el comportamiento del robot en MATLAB - Simulink, se deben identificar los parámetros inerciales pertenecientes al modelo dinámico del brazo robótico, por lo cual se hace uso del programa CAD Solid Edge. Esta herramienta CAD permite agregar tipos de materiales y ejes de rotación a la estructura realizada obteniendo así medidas como la masa, el volumen, el centro de masa y los parámetros necesarios en el modelo dinámico, como son los dos momentos de inercia o de área [12].

Después de haber realizado el ensamble de todos los elementos que conforman el brazo robótico en el entorno conjunto de Solid Edge, y haber asignado a cada una de las piezas y motores su respectivo material dándole cierta densidad, se procede a asignar a la estructura los ejes de rotación de cada articulación. Los ejes de rotación son asignados dependiendo del

movimiento que realizará el robot, donde el eje Z será el eje de giro de cada articulación.

En la barra de operaciones del software Solid Edge, se utiliza la opción de sistema de coordenadas (Figura 3-4), para crear un nuevo sistema personalizado. Se puede orientar y trasladar un sistema de coordenadas en los ejes X, Y y Z, que no estén referidos a un sistema de coordenadas base, esto se hace mediante la opción “Teclado” (Figura 3-5).

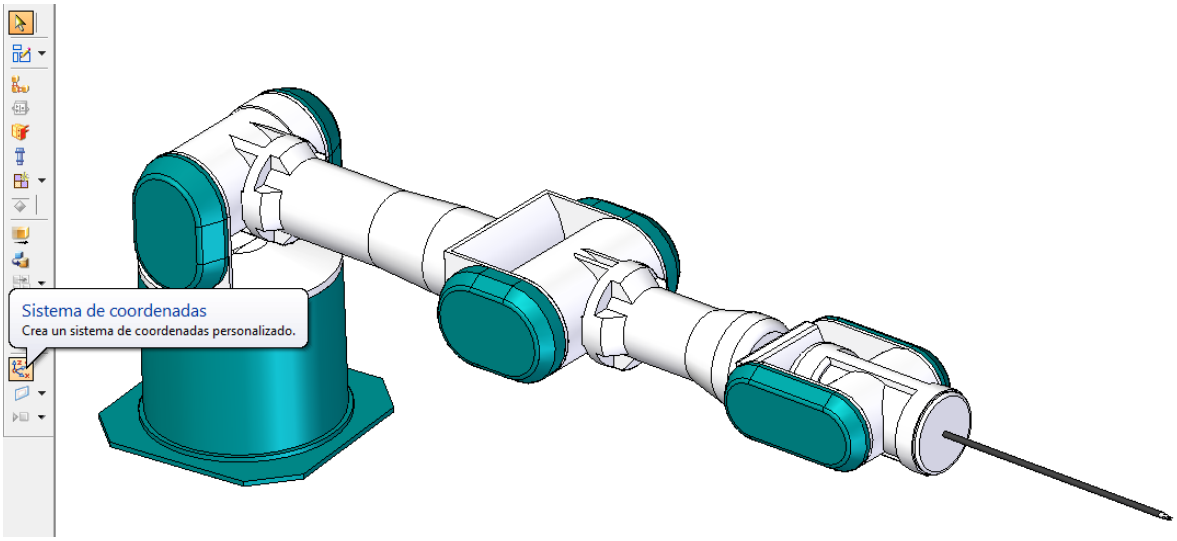

Figura 3-4. Opción sistema de coordenadas.

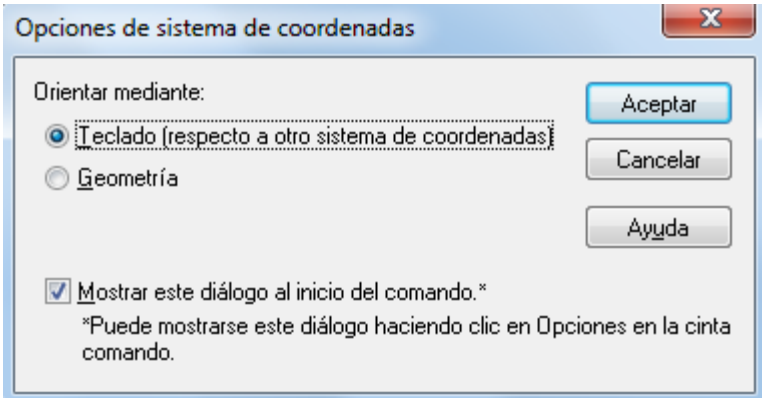

Figura 3-5. Orientación mediante “Teclado”.

Una vez asignados los ejes de rotación (Figura 3-6), se procede a obtener los parámetros de cada articulación, para esto se seleccionan únicamente las piezas que mueve cada actuador.

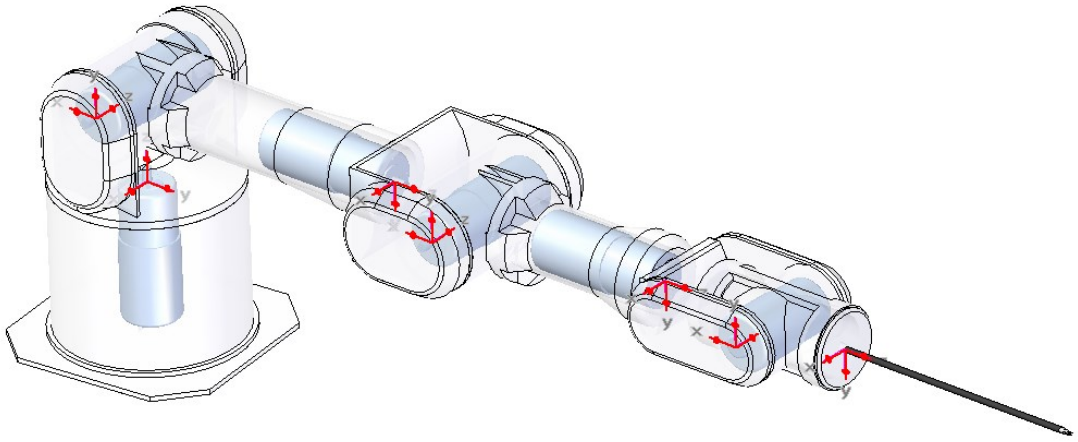

Figura 3-6. Robot PA-10 con sus ejes de rotación.

Por medio de la opción Verificar >> Propiedades Físicas (Figura 3-7), se pueden obtener los valores de masa que soporta cada articulación en la estructura, el centro de masa y los cálculos de los momentos de inercia (primer y segundo momento de inercia) (Figura 3-8).

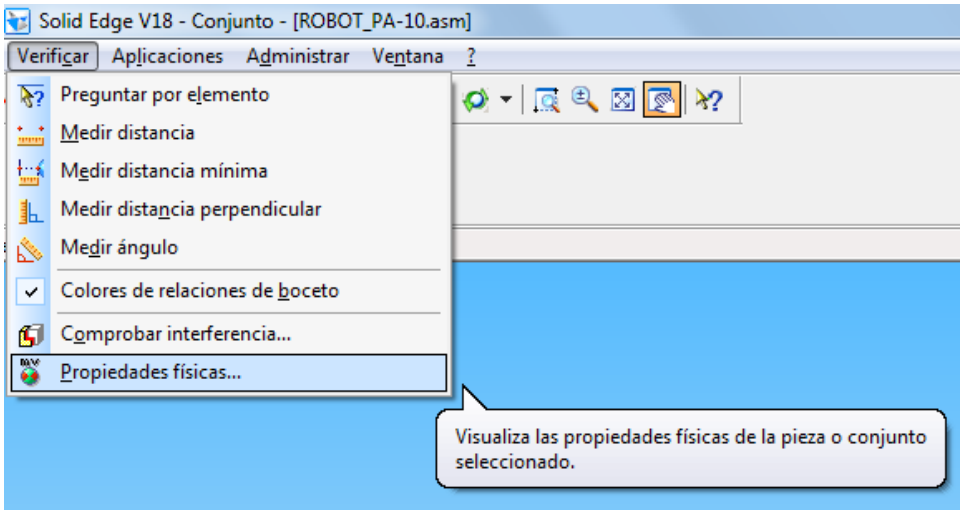

Figura 3-7. Herramienta verificar y propiedades físicas.

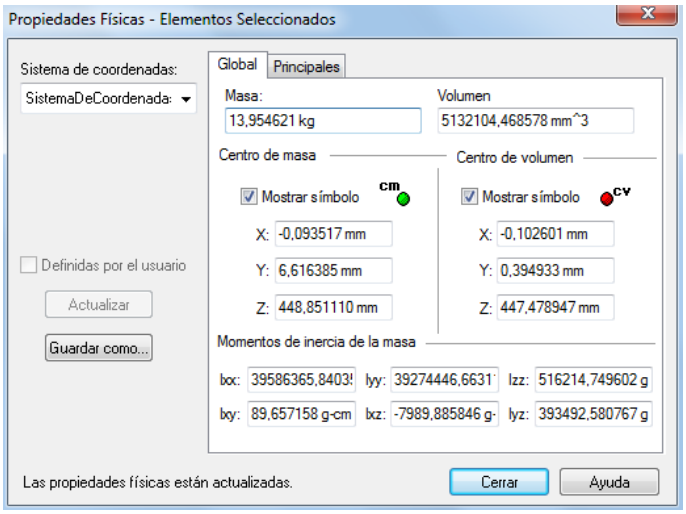

Figura 3-8. Propiedades físicas del conjunto.

Al realizar el procedimiento anterior para cada articulación se obtiene la tabla 3-2.

ARTICULACIONES	MASA SOPORTADA POR CADA ARTICULACION (Kg)	
Articulación 1	Masa 1	13,954621
Articulación 2	Masa 2	11,113362
Articulación 3	Masa 3	8,077282
Articulación 4	Masa 4	5,668207
Articulación 5	Masa 5	3,029131
Articulación 6	Masa 6	0,620056
Articulación 7	Masa 7	0,115751

Tabla 3-2. Masas de cada articulación.

Los valores que proporciona Solid Edge para el centro de masa de cada una de las articulaciones están expresados en milímetros, por lo que es necesario convertirlos a un sistema de unidades estándar (MKS) (Tabla 3-3).

ARTICULACIONES	Centro de masa en X (m)	Centro de masa en Y (m)	Centro de masa en Z (m)
Articulación 1	-0,000093517	0,006616385	0,44885111
Articulación 2	-0,000116556	-0,455118732	0,099645802
Articulación 3	-0,000054452	0,006683184	0,264450135
Articulación 4	-0,000029712	-0,243481426	0,089392548
Articulación 5	-0,000018241	0,008910466	0,091889279
Articulación 6	0,000123998	-0,02762377	0,081624261
Articulación 7	-0,000001229	0	0,004

Tabla 3-3. Distancia a los centros de masas de cada articulación.

Teniendo en cuenta las propiedades físicas que entrega Solid Edge, se puede calcular el primer momento de inercia o momento estático, multiplicando la masa soportada por cada articulación por la distancia al centro de masa de

cada una de sus coordenadas X, Y y Z. Este procedimiento se realiza para la primera articulación en las ecuaciones (3.1), (3.2) y (3.3). Igualmente se debe hacer para las articulaciones restantes concluyendo con la tabla 3-4.

Cálculo del momento estático para la primera articulación:

$$MX1 = Masa\ 1 * \Delta cm_x \quad (3.1)$$

$$MX1 = 13.954621\ Kg * (-0.000093517\ m) = -0.001304994\ Kg * m$$

$$MY1 = Masa\ 1 * \Delta cm_y \quad (3.2)$$

$$MX1 = 13.954621\ Kg * (0.006616385\ m) = 0.092329145\ Kg * m$$

$$MZ1 = Masa\ 1 * \Delta cm_z \quad (3.3)$$

$$MX1 = 13.954621\ Kg * (0.44885111\ m) = 6.263547125\ Kg * m$$

Articulaciones	Momento de inercia en X (Kg.m)		Momento de inercia en Y (Kg.m)		Momento de inercia en Z (Kg.m)	
Articulación 1	MX1	-0,001304994	MY1	0,092329145	MZ1	6,263547125
Articulación 2	MX2	-0,001295329	MY2	-5,057899222	MZ2	1,107399869
Articulación 3	MX3	-0,000439824	MY3	0,053981962	MZ3	2,136038315
Articulación 4	MX4	-0,000168414	MY4	-1,380103123	MZ4	0,506695466
Articulación 5	MX5	-5,52544E-05	MY5	0,026990969	MZ5	0,278344664
Articulación 6	MX6	7,68857E-05	MY6	-0,017128284	MZ6	0,050611613
Articulación 7	MX7	-1,42258E-07	MY7	0	MZ7	0,000463004

Tabla 3-4. Primer momento de inercia de cada articulación en los ejes X, Y y Z.

El segundo momento de inercia es obtenido mediante Solid Edge (Figura 3-9).

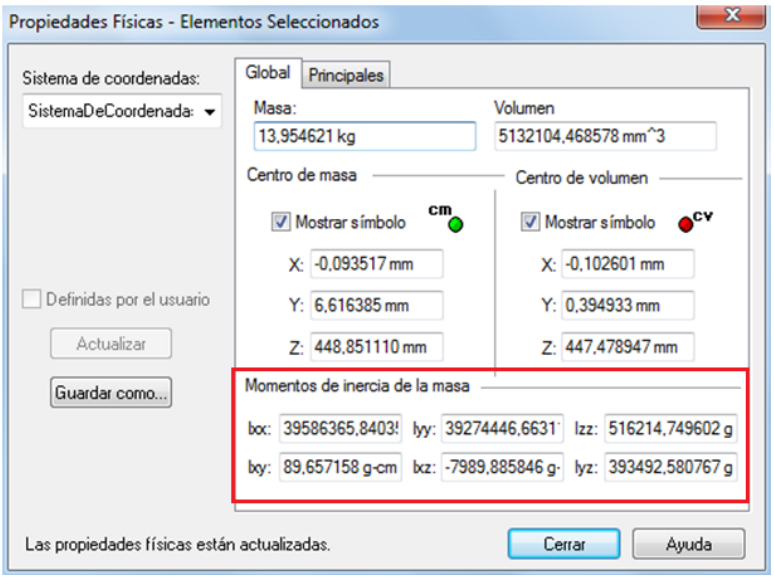

Figura 3-9. Segundo momento de inercia.

Con el fin de tener un mismo sistema de unidades es necesario convertir estos valores de g.cm^2 a Kg.m^2 (3.4). Este procedimiento se realiza para todos los valores de la tabla 3-5.

$$Ixx = Valor\ g.cm^2 * \frac{1\ Kg}{1000\ g} * \frac{1\ m^2}{10000\ cm^2} \quad (3.4)$$

$$Ixx = 39586365.84\ g.cm^2 * \frac{1\ Kg}{1000\ g} * \frac{1\ m^2}{10000\ cm^2} = 3.958636584\ Kg.m^2$$

Los valores del segundo momento de inercia se presentan en la tabla 3-5.

Articulaciones	Momento de inercia en X, Y y Z (Kg.m^2)					
Articulación 1	Ixx	3,958636584	Iyy	3,927444666	Izz	0,051621475
	Ixy	8,96572E-06	Ixz	-0,000798989	Iyz	0,039349258
Articulación 2	Ixx	3,034902562	Iyy	0,14484614	Izz	2,904476538
	Ixy	0,000676965	Ixz	-0,000114148	Iyz	-0,511540072
Articulación 3	Ixx	0,792444985	Iyy	0,773889573	Izz	0,0294865
	Ixy	1,24651E-05	Ixz	-0,000182586	Iyz	0,01473708
Articulación 4	Ixx	0,482682499	Iyy	0,063293651	Izz	0,427046066
	Ixy	8,28407E-05	Ixz	-2,04521E-06	Iyz	-0,126755607
Articulación 5	Ixx	0,04282409	Iyy	0,033653767	Izz	0,013760459
	Ixy	1,3935E-05	Ixz	-5,99829E-06	Iyz	0,00264512
Articulación 6	Ixx	0,007625747	Iyy	0,006506279	Izz	0,002435758
	Ixy	-2,01906E-06	Ixz	7,97667E-06	Iyz	-0,001455902
Articulación 7	Ixx	0,00011655	Iyy	0,000116549	Izz	0,00022816
	Ixy	0	Ixz	3,4305E-08	Iyz	5,6E-12

Tabla 3-5. Segundo momento de inercia de cada articulación en los ejes X, Y y Z.

3.2. Simulación del robot PA-10 en MATLAB – Simulink.

Luego de obtener los valores de los momentos de inercia, se procede a realizar un proceso de optimización de codo y muñeca, de esta forma se garantiza que el robot PA-10 al realizar la trayectoria deseada respete el paso por el trocar. Este proceso de optimización, se basa en el proceso de optimización desarrollado en el trabajo de grado "Estudio del desempeño de un robot PUMA en operaciones de laparoscopia" [7]. Cabe aclarar, que el proceso de optimización desarrollado en el trabajo de grado anterior a este, se efectúa para un trocar que se encuentra ubicado entre el codo y la muñeca, es decir en la longitud RL5 (Figura 3-1), mientras que para este trabajo, se realiza para un trocar que se encuentra ubicado entre la muñeca y el efector final (RLS) (Figura 3-1), por lo tanto las ecuaciones cambian.

3.2.1. Proceso de optimización.

Para realizar el proceso de optimización se debe tener en cuenta la posición del trocar (Ptr), que se encuentra entre la muñeca (Pm) y el efector final (Pd). Se requiere entonces que entre los tres puntos exista la condición de colinealidad, es decir, que estos puntos se encuentren alineados en cada instante de tiempo. Además de esto, hay que tener en cuenta la posición del codo (Pc), el cual ayuda a posicionar la muñeca para cumplir con la condición de colinealidad.

La alineación de los vectores de posición nombrados anteriormente, se cumple cuando el producto vectorial $\overrightarrow{PdPtr} \wedge \overrightarrow{PdPm}$ es igual a cero.

Sean (x_d, y_d, z_d) las coordenadas de Pd, (x_m, y_m, z_m) las de Pm, (x_c, y_c, z_c) las de Pc y (x_{tr}, y_{tr}, z_{tr}) las de Ptr:

$$\overrightarrow{PdPtr} = \begin{bmatrix} x_{tr} - x_d \\ y_{tr} - y_d \\ z_{tr} - z_d \end{bmatrix} \quad y \quad \overrightarrow{PdPm} = \begin{bmatrix} x_m - x_d \\ y_m - y_d \\ z_m - z_d \end{bmatrix} \quad (3.5)$$

El producto vectorial $\overrightarrow{PdPtr} \wedge \overrightarrow{PdPm}$ es:

$$\overrightarrow{PdPtr} \wedge \overrightarrow{PdPm} = \begin{bmatrix} (y_{tr} - y_d)(z_m - z_d) - (z_{tr} - z_d)(y_m - y_d) \\ (z_{tr} - z_d)(x_m - x_d) - (x_{tr} - x_d)(z_m - z_d) \\ (x_{tr} - x_d)(y_m - y_d) - (y_{tr} - y_d)(x_m - x_d) \end{bmatrix} \quad (3.6)$$

Igualando a cero (3.6):

$$\begin{aligned} (y_{tr} - y_d)(z_m - z_d) - (z_{tr} - z_d)(y_m - y_d) &= 0 \\ (z_{tr} - z_d)(x_m - x_d) - (x_{tr} - x_d)(z_m - z_d) &= 0 \quad (3.7) \\ (x_{tr} - x_d)(y_m - y_d) - (y_{tr} - y_d)(x_m - x_d) &= 0 \end{aligned}$$

En la figura 3-1 se puede ver que RL3 va desde el origen R_0 hasta el codo (x_c, y_c, z_c). La distancia entre estos dos puntos esta definida por:

$$\sqrt{(x_c)^2 + (y_c)^2 + (z_c)^2} = RL3 \quad (3.8)$$

RL5 es la distancia del codo (x_c, y_c, z_c) a la muñeca (x_m, y_m, z_m):

$$\sqrt{(x_m - x_c)^2 + (y_m - y_c)^2 + (z_m - z_c)^2} = RL5 \quad (3.9)$$

Y RL7 más RL8 (RLS), va desde la muñeca (x_m, y_m, z_m) hasta el órgano terminal (x_d, y_d, z_d):

$$\sqrt{(x_d - x_m)^2 + (y_d - y_m)^2 + (z_d - z_m)^2} = RLS \quad (3.10)$$

Además RLS es la suma de la distancia de la muñeca (x_m, y_m, z_m) al trocar (x_{tr}, y_{tr}, z_{tr}), más la del trocar al efector final (x_d, y_d, z_d):

$$\sqrt{(x_m - x_{tr})^2 + (y_m - y_{tr})^2 + (z_m - z_{tr})^2} +$$

$$\sqrt{(x_{tr} - x_d)^2 + (y_{tr} - y_d)^2 + (z_{tr} - z_d)^2} = RLS \quad (3.11)$$

Se tiene un sistema de siete ecuaciones con seis incógnitas, que son las posiciones del codo y de la muñeca, que se representan así [7]:

$$X = \begin{bmatrix} X1 = x_c \\ X2 = y_c \\ X3 = z_c \\ X4 = x_m \\ X5 = y_m \\ X6 = z_m \end{bmatrix} (3.12)$$

Sea $F(X)$ un vector que agrupe las siete ecuaciones ((3.7), (3.8), (3.9), (3.10), (3.11)), así:

$$F(X) = \begin{bmatrix} F_1(X) = (y_{tr} - y_d)\,(X4 - z_d) - (z_{tr} - z_d)\,(X5 - y_d) \\ F_2(X) = (z_{tr} - z_d)\,(X4 - x_d) - (x_{tr} - x_d)\,(X6 - z_d) \\ F_3(X) = (x_{tr} - x_d)\,(X5 - y_d) - (y_{tr} - y_d)\,(X4 - x_d) \\ F_4(X) = (X1)^2 + (X2)^2 + (X3)^2 - RL3^2 \\ F_5(X) = (X4 - X1)^2 + (X5 - X2)^2 + (X6 - X3)^2 - RL5^2 \\ F_6(X) = (x_d - X4)^2 + (y_d - X5)^2 + (z_d - X6)^2 - RLS^2 \\ F_7(X) = \sqrt{(X4 - x_{tr})^2 + (X5 - y_{tr})^2 + (X6 - z_{tr})^2} + \\ \qquad \sqrt{(x_{tr} - x_d)^2 + (y_{tr} - y_d)^2 + (z_{tr} - z_d)^2} - RLS \end{bmatrix} (3.13)$$

Se utiliza el algoritmo de Levenberg Mardquart, el cual es un algoritmo numérico iterativo de optimización, que resuelve la función $F(X)$(3.13) obteniendo las seis incógnitas [7]. La solución del método depende de las derivadas parciales de la función $F(X)$(3.13) para hallar la solución a partir de unas condiciones iniciales. Cuando el resultado de remplazar los valores de cada variable en las ecuaciones dadas para la solución del modelo es cercano a cero, (aproximadamente 1e-15), entonces el proceso se detiene y entrega los valores de cada variable para los cuales se cumplió el mínimo resultado posible.

Siguiendo el esquema de la figura 3-10, se introducen las posiciones en x, y, z de la consigna deseada al algoritmo de Levenberg Mardquart, del cual se obtienen las tres posiciones del codo (x_c, y_c, z_c) y las tres de la muñeca (x_m, y_m, z_m) que garantizan el paso por el trocar.

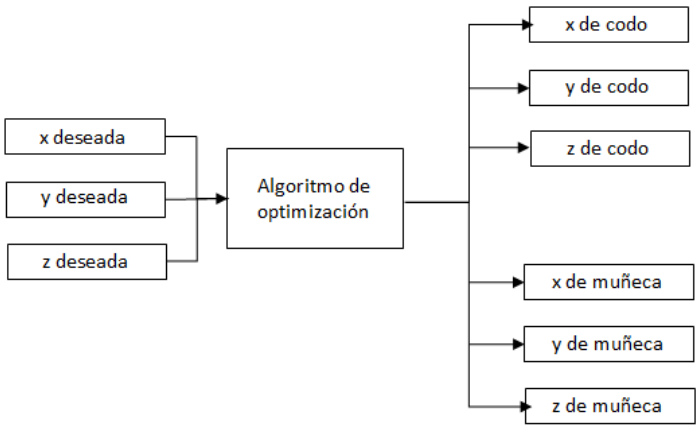

Figura 3-10. Esquema de optimización.

Una vez realizado el proceso de optimización se obtiene el resultado que se muestra en la figura 3-11, donde el robot realiza la consigna deseada, la cual, es un circulo de r = 0.04m y origen en x = 0.32 m, y = 0.48 m. Además de esto, se puede verificar que el robot respeta el paso por el trocar, ubicado en las posiciones x = 0.32 m, y = 0.48 m, z = 0.23 m.

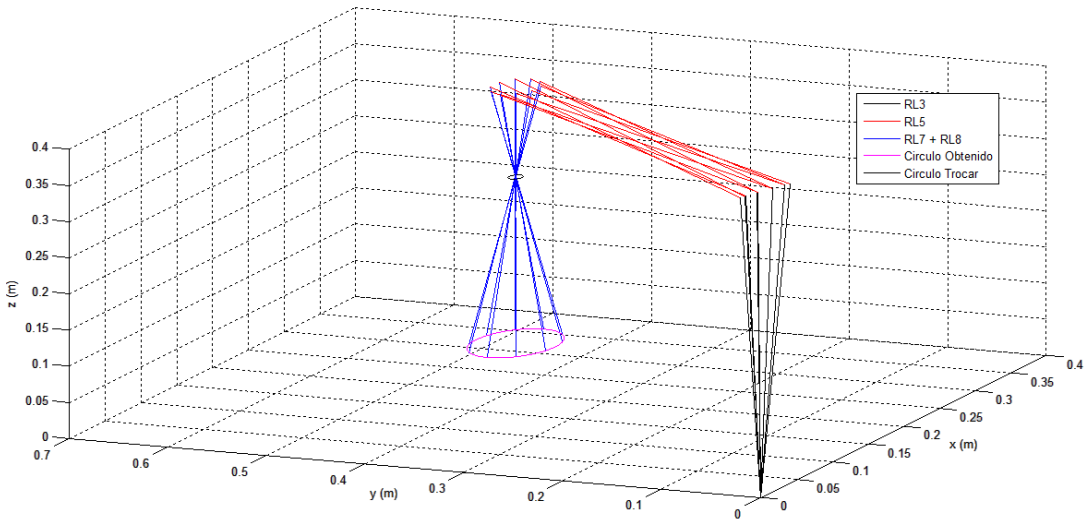

Figura 3-11. Comportamiento del Robot Tipo PA-10 con el proceso de optimización.

3.2.2. MGI del robot PA-10.

El MGI[6] del robot PA-10 fue calculado en el trabajo de grado "Estudio del desempeño de un robot PUMA en operaciones de laparoscopia" [7]. Aquí se muestra las ecuaciones obtenidas que determinan el posicionamiento de cada una de las articulaciones.

[6] MGI: Modelo Geométrico Inverso.

- Articulación 1:

$$q_1 = \pm \arctan\left(\frac{{}^0y_c}{{}^0x_c}\right) (3.14)$$

- Articulación 2:

$$q_2 = \pm \arccos\left(\frac{{}^0z_c}{RL3}\right) (3.15)$$

- Articulación 3:

$$q_3 = \pm \arctan\left(\frac{{}^2z_m}{{}^2x_m}\right) (3.16)$$

Dónde Pmx, Pmy y Pmz son las posiciones de la muñeca en cada eje.

$${}^2x_m = C2C1Pmx + C2S1Pmy - S2Pmz$$
$${}^2z_m = -S1Pmx + C1Pmy$$

- Articulación 4:

$$q_4 = \pm \arccos\left(\frac{-{}^2y_m - RL3}{RL5}\right) (3.17)$$

Dónde:

$${}^2y_m = -S2C1Pmx - S2S1Pmy - C2Pmz$$

- Articulación 5:

$$q_5 = \pm \arctan\left(\frac{{}^4z_d}{{}^4x_d}\right) (3.18)$$

Dónde Pdx, Pdy, Pdz son las posiciones de la consigna deseada.

$${}^4z_d = (-S3C2C1 - C3S1)Pdx + (-S3C2S1 + C3C1)Pdy + S3S2Pdz$$

$${}^4x_d = (C4(C3C2C1 - S3S1) - S4S2C1)Pdx + (C4(C3C2S1 + S3C1) \cdots$$
$$-S4S2S1)Pdy + (-C4C3S2 - S4C2)Pdz + S4RL3$$

- Articulación 6:

$$q_6 = \pm \arccos\left(\frac{Pdx - M2}{M1}\right) (3.21)$$

Dónde Pdx y Pdz son las posiciones en los ejes X y Z de la consigna deseada respectivamente:

$$M1 = K2 - \left(\frac{K1K5}{K4}\right)$$

$$M2 = K3 + \left(\frac{K1(Pdz - K6)}{K4}\right)$$

$$K1 = -C1C2C3C4C5RLS + C1C2S3S5RLS + C1S2S4C5RLS + S1S3C4C5RLS + S1C3S5RLS$$

$$K2 = -C1C2C3S4RLS - C1S2C4RLS + S1S3S4RLS$$

$$K3 = C1C2C3S4RL5 + C1S2C4RL5 + C1S2RL3 - S1S3S4RL5$$

$$K4 = S2C3C4C5RLS - S2S3S5RLS + C2S4C5RLS$$

$$K5 = S2C3S4RLS - C2C4RLS$$

$$K6 = C2C4RL5 + C2RL3 - S2C3S4RL5$$

3.2.3. Prueba del MGI y MGD.

Después de obtener el MGI, se procede a realizar la prueba de los modelos geométricos, para ello se monta el diagrama ilustrado en la figura 3-12, en MATLAB – Simulink.

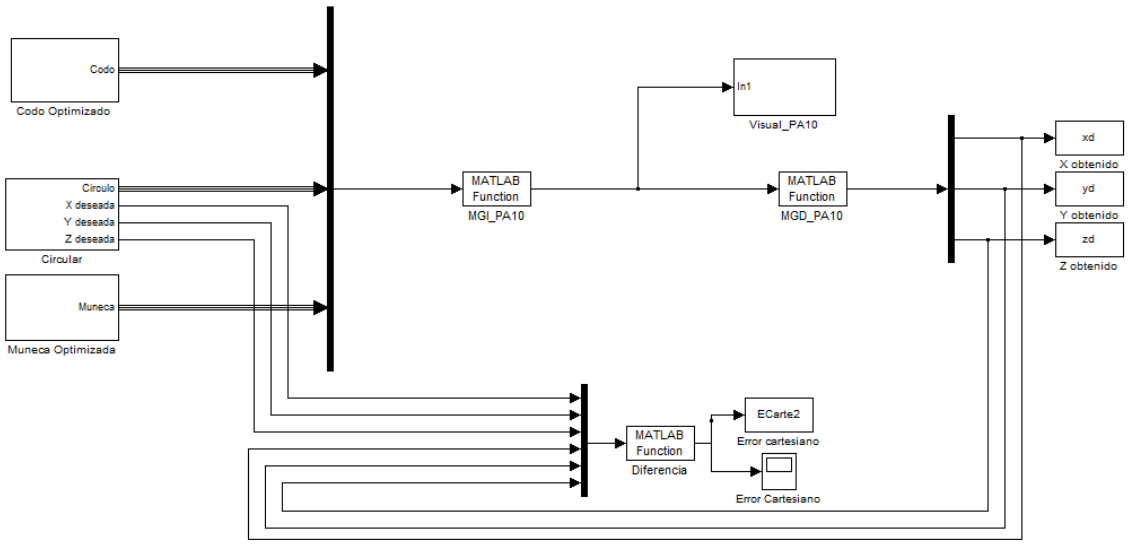

Figura 3-12. Prueba del MGI y MGD del Robot Tipo PA-10.

El subsistema visual PA-10 (Figura 3-13), se encuentra compuesto por los modelos geométricos directos del hombro, codo, muñeca y efector final, los cuales permiten visualizar el comportamiento del robot al llevar a cabo la simulación con los modelos MGI y MGD[7].

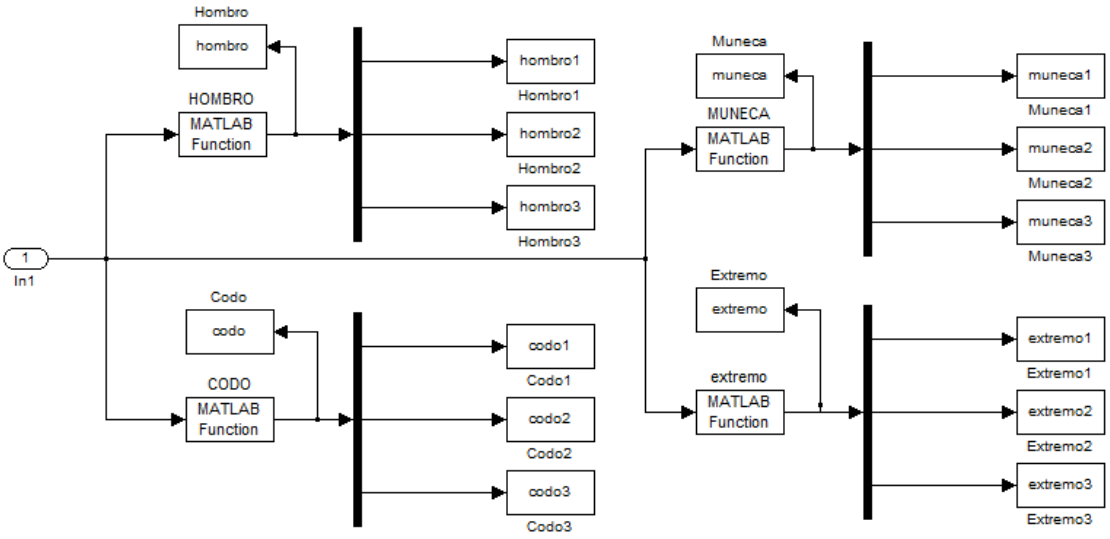

Figura 3-13. Subsistema Visual PA-10.

Al ejecutar la simulación se obtienen las figuras 3-14 y 3-15, donde se puede observar que el robot respeta nuevamente el paso por el trocar y realiza la consigna deseada con un error cartesiano de $7.26 * 10^{-4}$ m. Estos modelos también se probaron con la consigna lineal y con la consigna espiral obteniendo errores cartesianos por debajo del orden de los milímetros.

[7] MGD: Modelo Geométrico Directo.

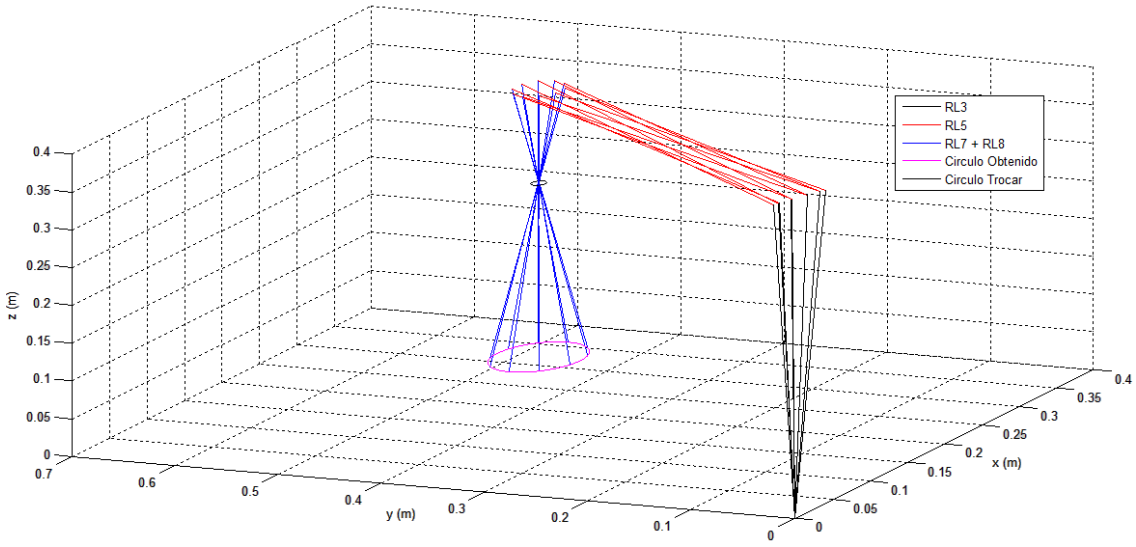

Figura 3-14. Comportamiento del robot PA-10 con MGI, MGD y consigna circular.

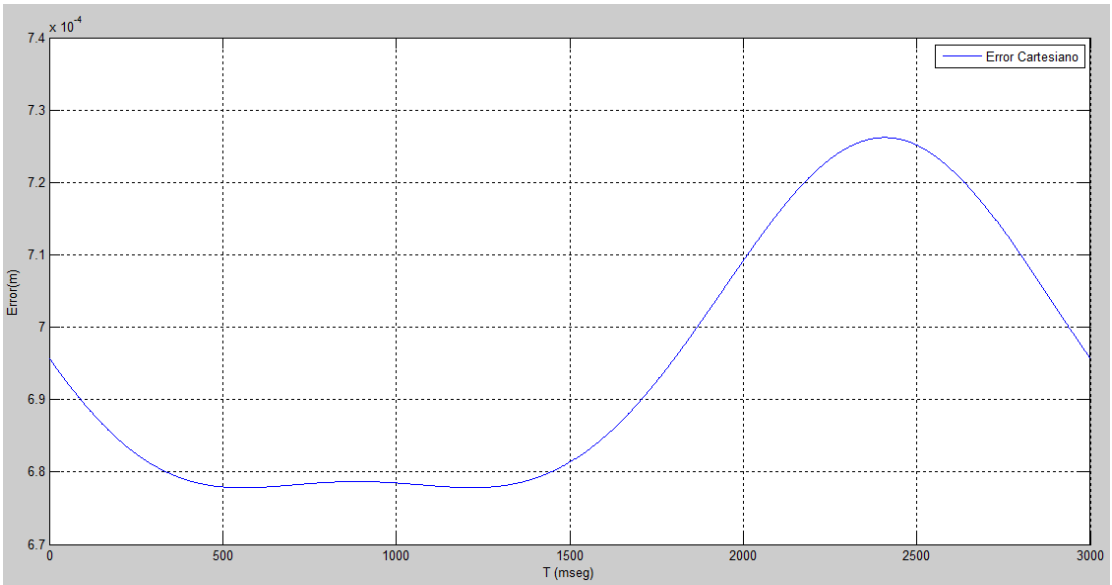

Figura 3-15. Error cartesiano del robot PA-10 con MGI y MGD.

Posteriormente, para verificar que la estructura cinemática del brazo robótico respeta el paso por el trocar, se diseña un controlador *CTC*[8] cartesiano en el entorno MATLAB – Simulink (Figura 3-16). En este segmento se utilizan los momentos de inercia calculados anteriormente, ya que estos son necesarios en los modelos dinámicos.

[8] *CTC: Computed Torque Control.*

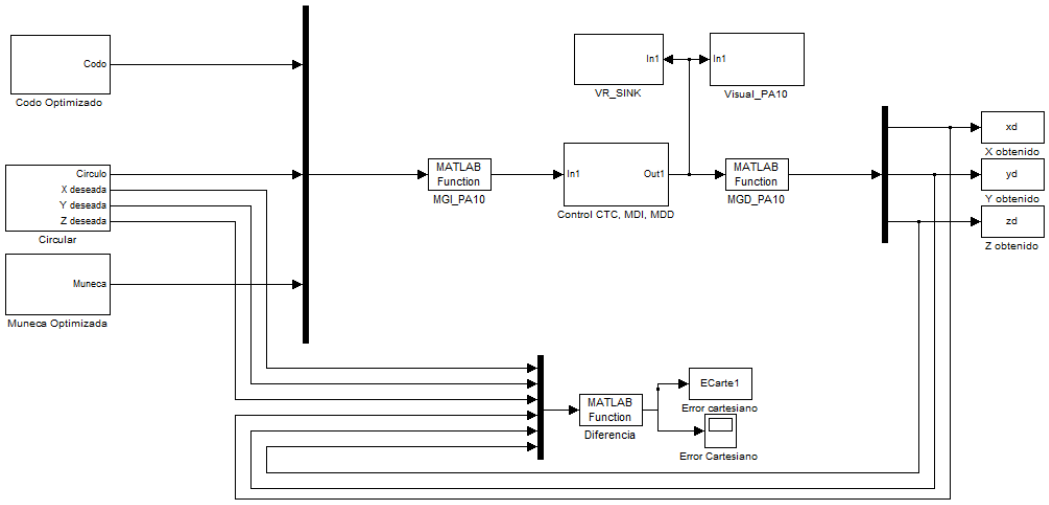

Figura 3-16. Esquema de control CTC.

La consigna que se siguió para verificar el comportamiento del robot PA-10, es la consigna circular. En la figura 3-17, se puede observar el seguimiento de esta consigna, donde la traza azul es el círculo deseado y el rojo es el obtenido. Además esto, se consiguió un error cartesiano de $9.96 * 10^{-4}$ m (Figura 3-18), el cual es un buen error para un robot que va a ser utilizado en una cirugía de este tipo.

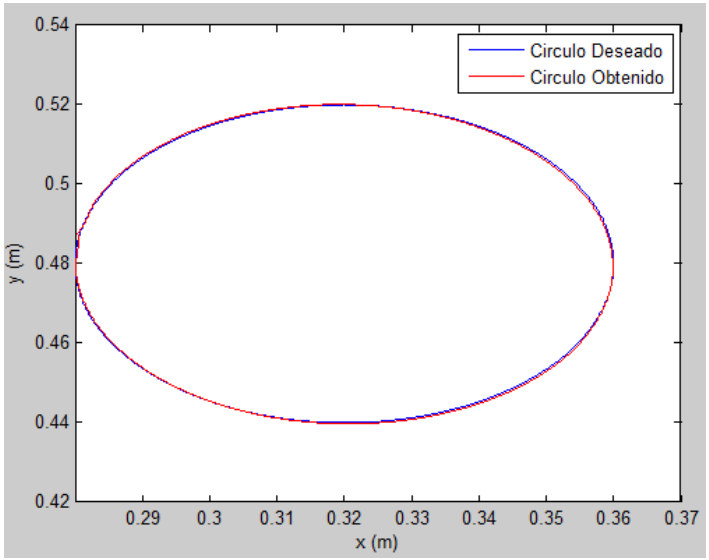

Figura 3-17. Consigna circular.

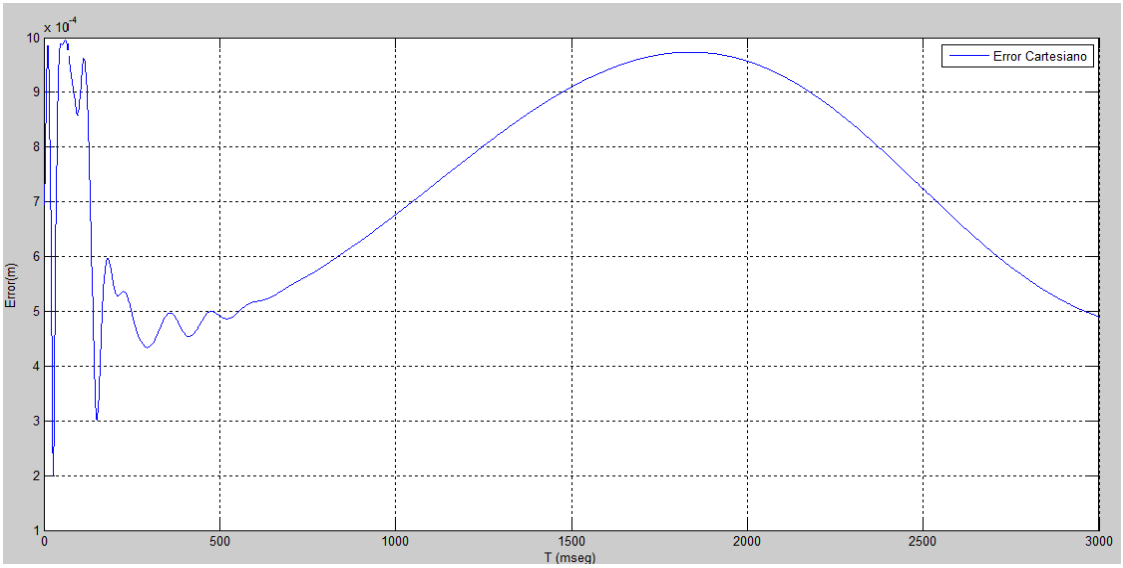

Figura 3-18. Error cartesiano del robot PA-10 con control CTC.

A partir de esta simulación se pueden calcular los torques que permiten el dimensionamiento de los actuadores de cada articulación, para una futura construcción.

3.2.4. Simulación del robot PA-10 en *Virtual Reality*.

Virtual Reality Toolbox es un paquete de herramientas de MATLAB que permite visualizar simulaciones de sistemas dinámicos en una escena 3D de realidad virtual creada en el lenguaje estándar *VRML*[9] [13]. En *VRML* las escenas que se construyen se conocen comúnmente como *worlds* (mundos) y por esta razón tienen la extensión *.wrl.

Todos los archivos de *VRML* son archivos de texto (*ASCII*[10]) y por lo tanto pueden ser editados con cualquier editor de textos. Sin embargo existen editores como *3D World Editor*, que proporcionan una interfaz gráfica de usuario con la que se puede crear escenas detalladas ensambladas a partir de modelos 3D exportados por fuentes *CAD* [14].

Antes de empezar a trabajar en el mundo virtual se procede a cambiar la extensión de las piezas del robot que se encuentran en formato *.par del entorno pieza de Solid Edge al formato *.wrl, extensión que utiliza la herramienta *3D World Editor* de MATLAB-2013. Este cambio de formato se

[9] *VRML: Virtual Reality Modeling Language.*
[10] *ASCII: American Standard Code for Information Interchange.*

realiza en Solid Edge guardando la pieza como imagen y estableciendo el tipo de archivo “Documento VRML”, de extensión *.wrl.

Con el fin de visualizar el comportamiento del robot en *Virtual Reality*, es necesario crear antes un mundo virtual para poder conectar el modelo realizado en *Simulink*. Una vez abierto este entorno de diagramas de bloques, se busca la librería de *Simulink 3D Animation* donde se encuentra el bloque *VR Sink*. Después de agregar el bloque *VR Sink* al modelo es posible crear el mundo virtual usando *3D World Editor*. La interfaz *GUI*[11] del editor, no sólo permite la representación 3D de objetos, sino que también los presenta en forma de árbol para mayor facilidad en el tratamiento de los nodos (elementos del mundo virtual) que tienen propiedades del objeto.

Una vez abierto el editor, el primer nodo que se añade es el *Background* que determina el color del fondo del entorno, para ello se selecciona en el menú del editor *nodes>>Add>> Bindable>> Background*. Luego se añade un nodo *transform* seleccionando *nodes>>Add>> Group>> Transform*. En este nodo se encuentra un campo *children* donde se carga la habitación en formato *.wrl (Figura 3-19). Al importar este objeto se crea un nodo *Group*, que agrupa otros nodos como nodos *Shapes*, los cuales definen las características de la pieza importada.

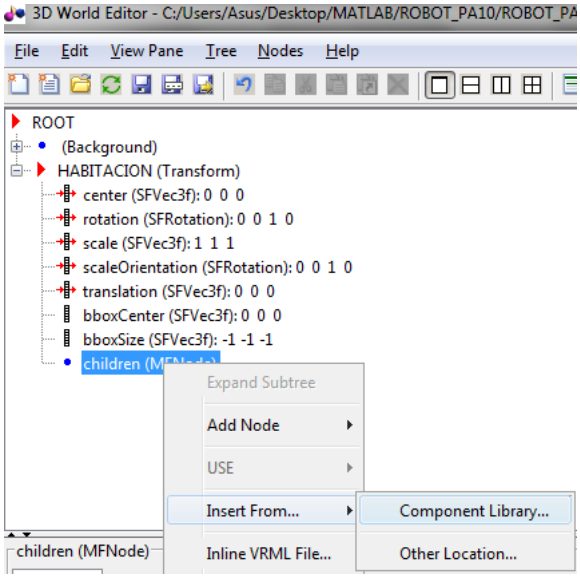

Figura 3-19. Importando objetos en formato *.wrl.

Una vez cargada la habitación se procede a ensamblar las piezas del robot. Cada componente que se desee agregar se debe hacer de la misma forma como se agregó la habitación. Es de tener en cuenta que para añadir un

[11] *GUI: Graphical User Interface*.

nuevo nodo se debe agregar en el campo *children* del nodo anterior, formado así el árbol de nodos.

Los ejes de coordenadas que se definieron en la creación de cada una de las piezas en Solid Edge coinciden con los ejes de coordenadas X, Y y Z, en el mundo virtual de la herramienta de MATLAB. A medida que se va acoplando cada una de las piezas del robot de extensión *.wrl, estas se deben colocar según la posición que corresponda dentro del conjunto global de la estructura física en el *3D World Editor*. Para mover las piezas a partir del origen de coordenadas, se usa el nodo *Transform* (Figura 3-20), con el cual se puede trasladar (*translation*), rotar (*rotation*) o escalar (*scale*) los elementos del robot.

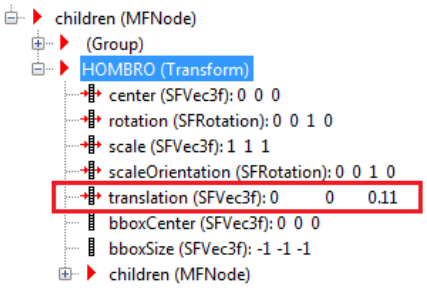

Figura 3-20. Traslación del hombro en el nodo *Transform*.

En la figura 3-21, se muestra el resultado de pasar la habitación y las piezas del robot diseñado en Solid Edge al programa *3D World Editor* de *Simulink*. En el panel izquierdo se puede ver el árbol del robot, donde se incluyen los nodos como *Background*, *Transform, children, Group y Shapes*.

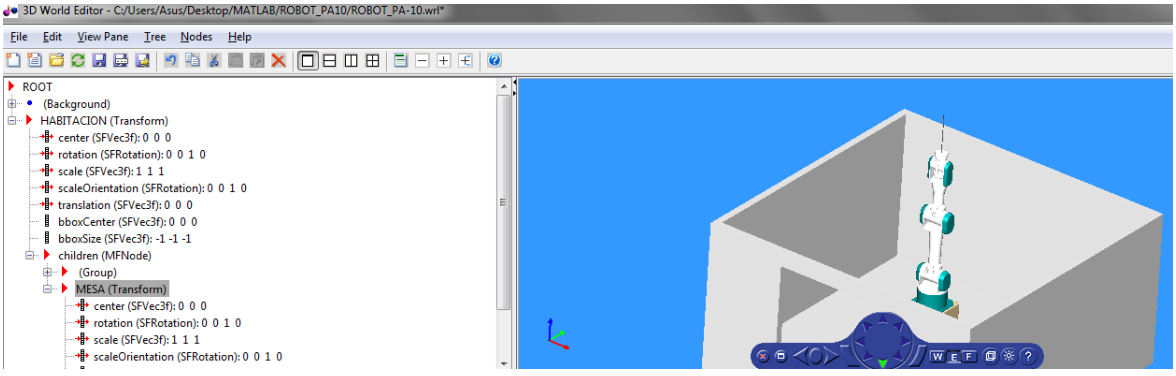

Figura 3-21. Robot PA-10 ensamblado en herramienta *3D World Editor* de *Simulink*.

Luego de haber creado el mundo virtual junto con el modelo en *Simulink*, se realiza la simulación con la consigna circular para observar en el ambiente 3D, cómo el robot PA-10 respeta claramente el paso por el trocar (Figura 3-22).

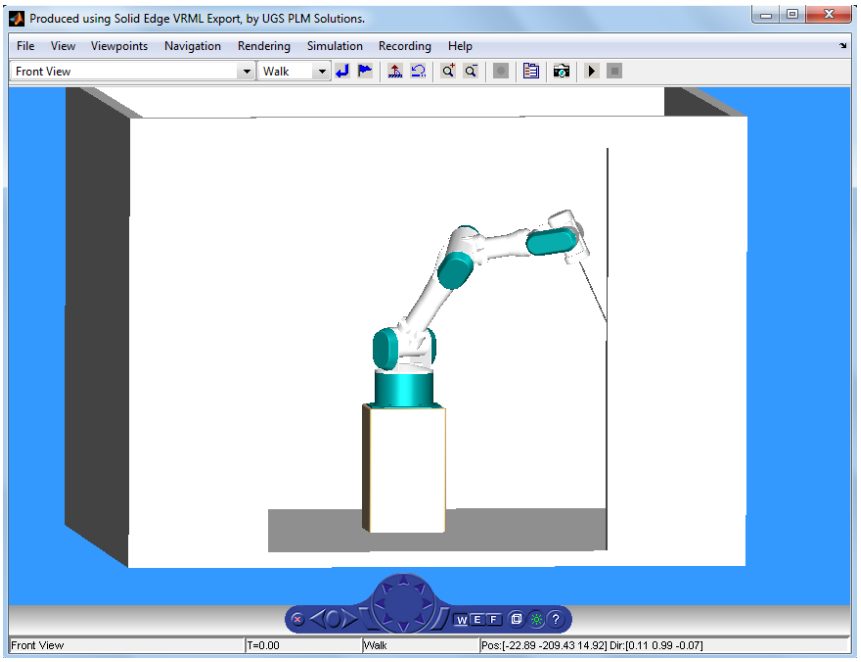

Figura 3-22. Simulación robot PA-10 utilizando consigna circular.

4. Software para manipulación del robot PA-10.

Para el desarrollo de la interfaz gráfica en la que se puedan manipular los movimientos del robot, se hace uso de un software base (Figura 4-1), creado por los ingenieros Luis Daladier Guerrero y Cristian Méndez, desarrollado en una asignatura de la Maestría en Automática de la Universidad del Cauca.

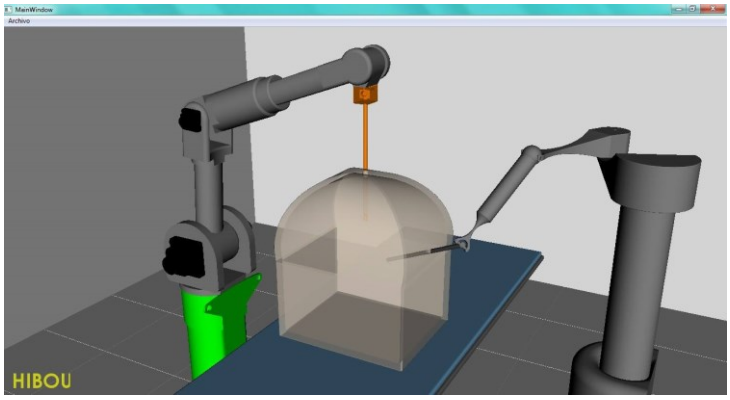

Figura 4-1. Software base para este proyecto.

El software base consta de un espacio de trabajo donde están los robot Hibou y Lapbot en una habitación. En este entorno se encuentra una camilla, una colchoneta y una caja que simula la cavidad abdominal, con el fin de verificar si los robots respetan el paso por el trocar. El movimiento de los robots se realiza por medio de un dispositivo externo (*joystick*) que permite al usuario interactuar con ellos.

4.1. Alcances y restricciones del software.

Para desarrollar la aplicación SVRQ[12], se utilizaron las siguientes librerías y plataformas:

- Blender y Librería VTK (motor gráfico).
- Visual Studio y QT (plataforma de desarrollo).
- V-Collide y RAPID (librerías de colisión y deformación)

[12] SVRQ: Simulador Virtual de Robótica Quirúrgica.

4.1.1. Alcances

- La aplicación permite la interacción entre los tres robots (Hibou, Lapbot y PA-10) dentro de la cavidad abdominal, a través del dispositivo de mando (*joystick* o *gamepad*).

- El software cuenta con una interfaz gráfica de usuario amigable permitiendo que la aplicación sea de fácil uso.

- Seleccionando el escenario de colisión y deformación de la aplicación, el usuario podrá interactuar con algunos órganos de la cavidad abdominal.

4.1.2. Restricciones.

- La deformación se realiza mediante el contacto del efector final del PA-10 con un órgano a la vez. Es decir si el efector final se encuentra deformando el estómago y colisiona con la arteria cística[13] esta no se va a deformar ya que el software solo permite deformar un órgano a la vez.

4.1.3. Blender.

Blender es una plataforma de animación de código abierto que facilita relacionar y posicionar los objetos en un entorno 3D. Esta herramienta se emplea con el fin de distribuir adecuadamente los objetos 3D en la escena y cambiar el formato *.wrl de las piezas, que conforman la estructura del robot diseñada en Solid Edge, al formato *obj que admite la biblioteca VTK (Figura 4-2) (Ver anexo B).

[13] Arteria cística: arteria que irriga la vesícula biliar y el conducto cístico

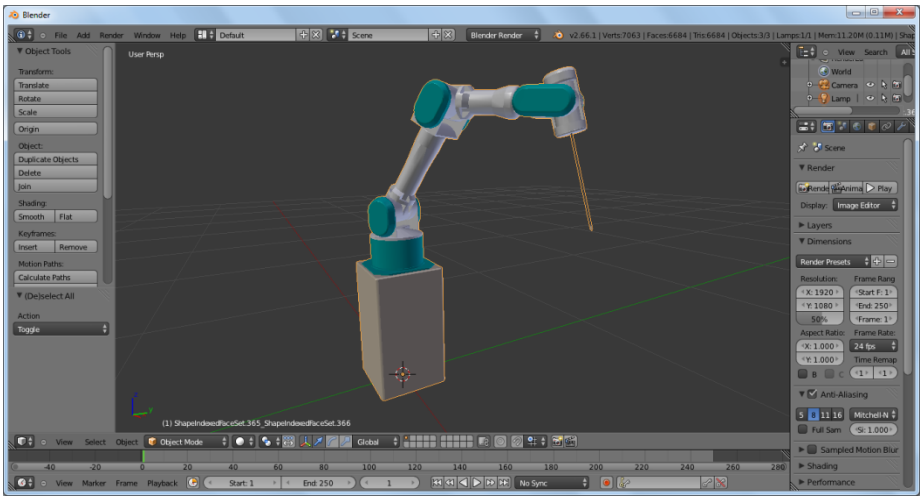

Figura 4-2. Uso de Blender para el modelado del Robot PA-10.

4.1.4. Librería para interfaz gráfica VTK.

VTK es un conjunto de librerías de código abierto orientada a objetos y disponible para visualización y tratamiento de imágenes. VTK está implementado en C++. Utiliza la aplicación CMake para el proceso de compilación y además puede integrarse en herramientas como Qt.

VTK cuenta con una amplia funcionalidad para el procesamiento de imágenes y visualización. No sólo permite visualizar figuras geométricas, sino que además soporta una amplia variedad de algoritmos de visualización incluyendo: escalar, vector, tensor, textura, métodos volumétricos; y técnicas de modelado avanzadas como: modelado implícito, reducción de polígonos, el suavizado de malla, corte y contorno. Debido a su gran potencia, se hacen necesarios amplios recursos de memoria en el ordenador para poder aprovechar la totalidad de sus funcionalidades [15], (para instalar ver Anexo A).

4.1.5. Plataforma de desarrollo Visual Studio.

Microsoft Visual Studio es un entorno de desarrollo integrado para sistemas operativos Windows, soporta varios lenguajes de programación como Visual C++, Visual C#, Visual J#. ASP: NET y Visual Basic .NET, aunque en la actualidad existe compatibilidad y extensiones para otros [16].

Microsoft Visual Studio permite crear aplicaciones que integran librerías y motores gráficos para la implementación de simuladores y al mismo tiempo otras librerías para conexión a hardware externo, lo que permite realizar aplicaciones externas al computador mediante un protocolo de comunicación USB [12], (para instalar ver Anexo A).

4.1.6. QT Designer.

QT es un *framework* de desarrollo de aplicaciones multiplataforma escrito en el lenguaje de programación C++, se utiliza para el desarrollo de software de aplicación con una interfaz gráfica de usuario (*GUI*) y para el desarrollo de programas no gráficos como herramientas de línea de comandos y consolas para servidores, que pueden ejecutarse en múltiples plataformas, incluyendo Microsoft Windows, Apple Mac OS X, Linux y muchas plataformas móviles como Symbian y Windows Phone.

QT Designer es la herramienta de QT que se emplea para el diseño y creación de la interfaz gráfica de usuario (*GUI*), con el fin de proveer un entorno visual sencillo que gestione la interacción del sistema con el usuario basándose en relaciones visuales como iconos, botones, ventas, menús, etc [17], (Figura 4-3) (para instalar ver Anexo A).

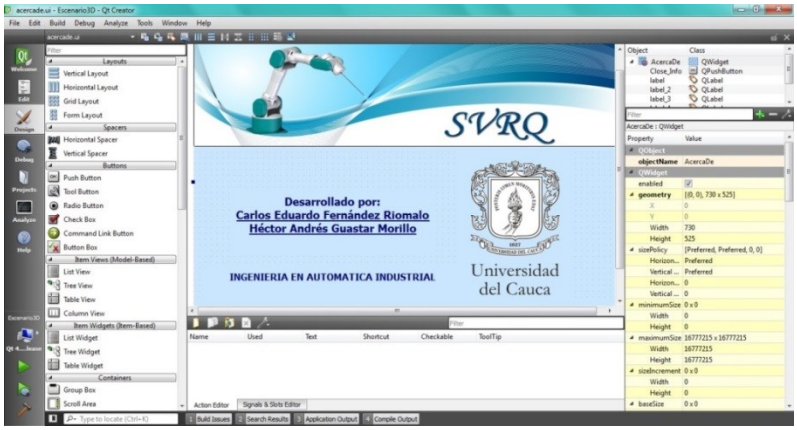

Figura 4-3. Uso de QT para el desarrollo de las *GUI*.

4.1.7. Librerias V-Collide y RAPID.

V-Collide es una librería de detección de colisiones desarrollada por el grupo *GAMMA*[14] de la Universidad de Carolina del Norte [18].

[14] GAMMA: Geometric Algorithm for Modeling, Motion and Animation.

La librería está diseñada para operar en ambientes que contienen un gran número de objetos geométricos formados con mallas de triángulos, realizando la detección de colisiones entre parejas de objetos a través de dos etapas:

- Construir *OBB*[15] para cada objeto, con el fin de encontrar parejas de triángulos que posiblemente intersecten los *OBB*. Esta técnica se basa en encerrar los objetos a colisionar dentro de cajas que se orientan según la alineación de los objetos [19].

- Verificar si las parejas de triángulos detectadas en la etapa anterior realmente se intersectan.

Estas etapas se encuentran en un componente de V-Collide llamado RAPID, el cual también es una librería de detección de colisiones independiente. Las diferencias entre ambas librerías son las siguientes:

- V-Collide conserva información acerca de dónde se encuentran los objetos en el ambiente, de tal manera que si estos no se mueven sus locaciones no tienen que ser recalculadas, como en RAPID.

- V-Collide permite la verificación de muchos objetos de forma simultánea y RAPID solamente permite la verificación de dos.

- Rapid reporta qué parejas de triángulos exactamente colisionaron, mientras que V-Collide solo reporta colisión entre los objetos.

Para el proceso de instalación de estas librerías ver Anexo A.

4.2. Modelado *UML*.

Para el desarrollo de una aplicación, es de vital importancia llevar a cabo un proceso de modelado, ya que permite conocer con claridad las relaciones de los elementos que conforman la aplicación. Por tal razón, se utilizan los diagramas *UML*[16] que permiten visualizar, especificar, construir y documentar un software por medio de una representación gráfica.

[15] *OBB: Oriented Bounding Box*.
[16] *UML: Unified Modeling Language*.

4.2.1. Diagrama de casos de usos.

En el diagrama mostrado en la figura 4-4, se indican los casos de uso necesarios para que el usuario pueda acceder a la aplicación y elija el escenario a trabajar.

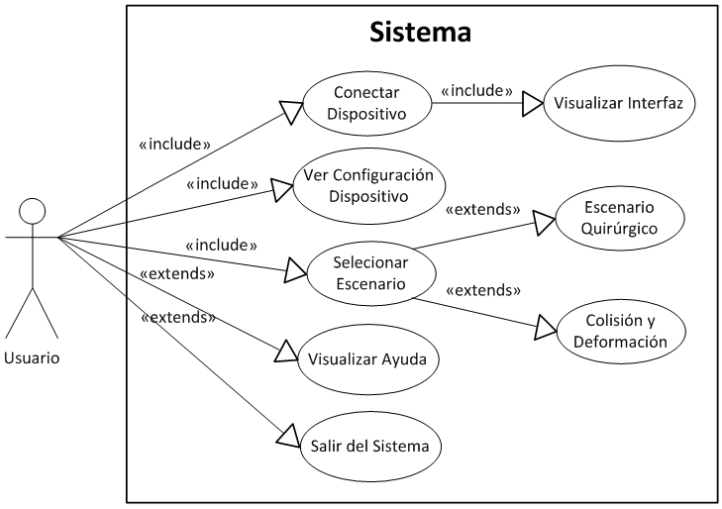

Figura 4-4. Diagrama de casos de usos al inicializar la aplicación.

Una vez elegido el escenario quirúrgico como ambiente a trabajar, el usuario podrá interactuar con los casos de usos implementados en la figura 4-5. Si por el contrario, el usuario decide acceder al escenario de colisión y deformación deberá interactuar con los casos relacionados a este entorno (Figura 4-6).

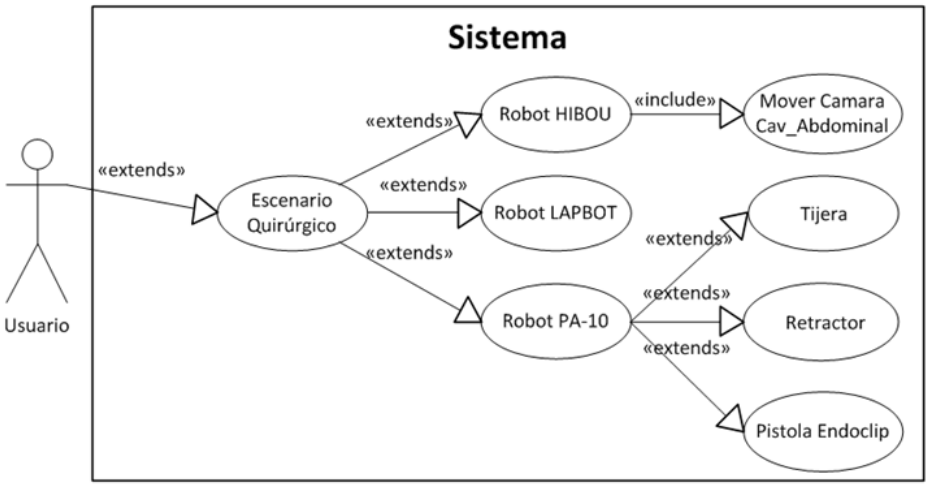

Figura 4-5. Diagrama de casos de usos en escenario quirúrgico.

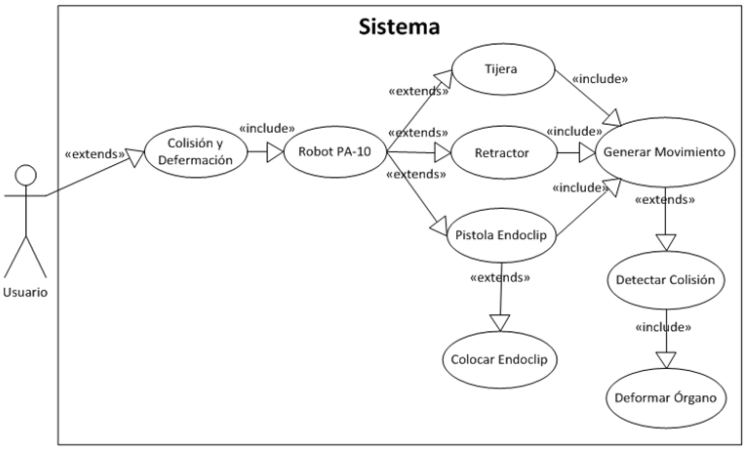

Figura 4-6. Diagrama de casos de usos en escenario de colisión y deformación.

4.2.2. Diagrama de clases.

En la figura 4-7 se puede observar que la clase Escenario 3D, es la clase base para llevar a cabo la ejecución del programa. Las clases Ayuda, Acerca De y Barra Progreso, se encuentran relacionadas con la clase Escenario 3D mediante una agregación, ya que estas clases pueden coexistir independientemente del programa principal. La clase Objeto Actor es una clase heredada por las clases Quirófano, Cavidad Abdominal, Robot Hibou, Robot LapBot y Robot PA-10, esto se debe a que estas clases pueden acceder a los atributos y métodos de la clase Objeto Actor. Las clases restantes tienen una relación por composición, debido a que su nexo es muy fuerte y estas clases no pueden existir sin las otras clases. En las figuras 4-8 y 4-9 se muestra el diagrama de clases más detallado.

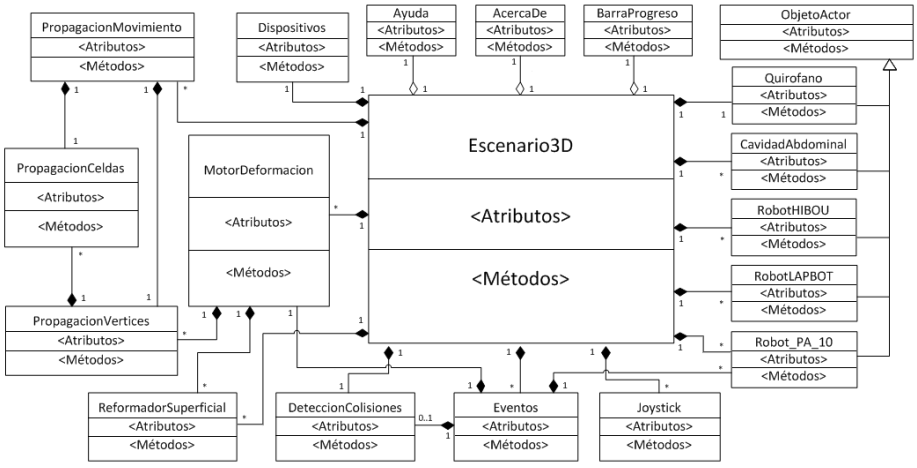

Figura 4-7. Diagrama de clases general para la aplicación SVRQ.

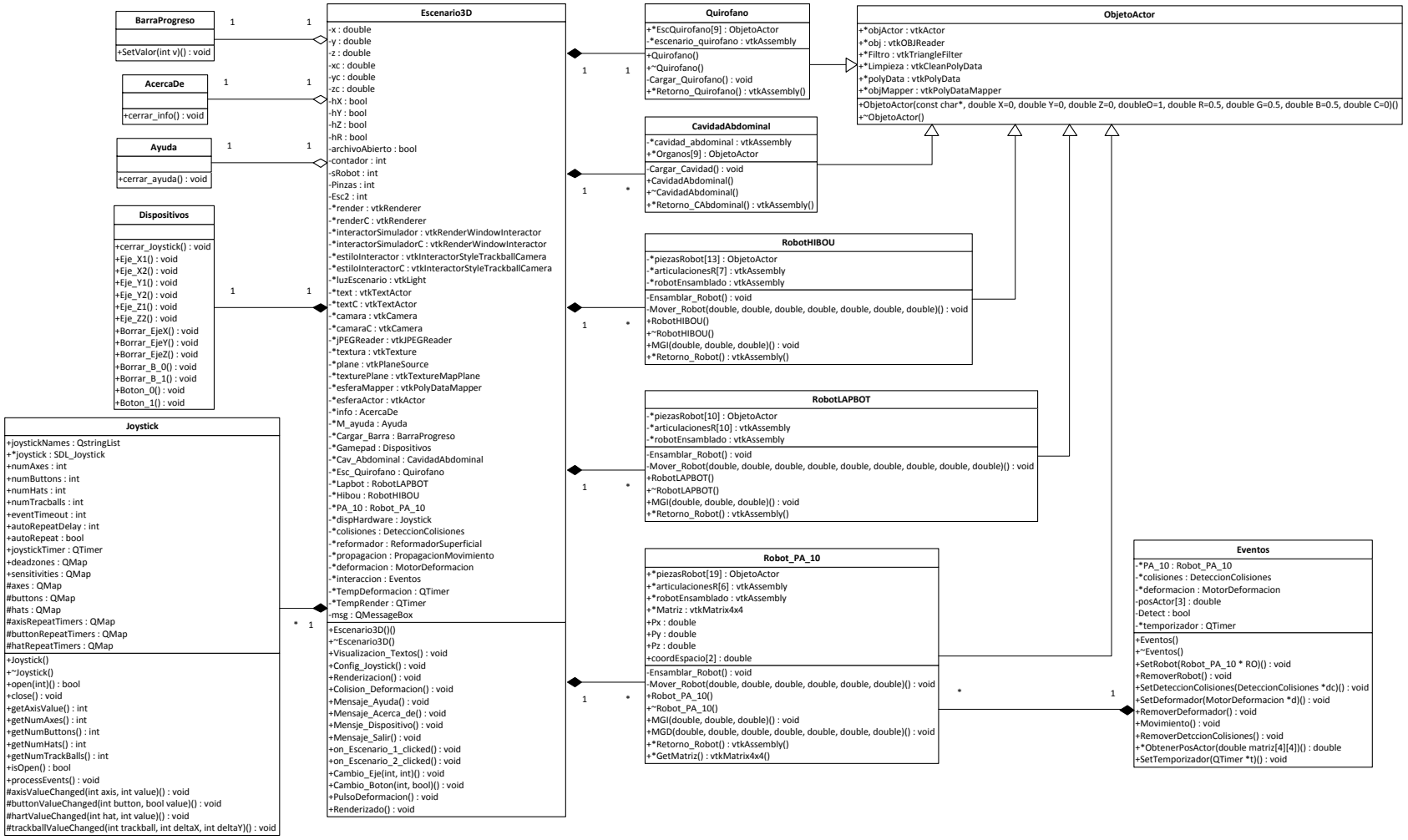

Figura 4-8. Diagrama detallado de clases parte 1.

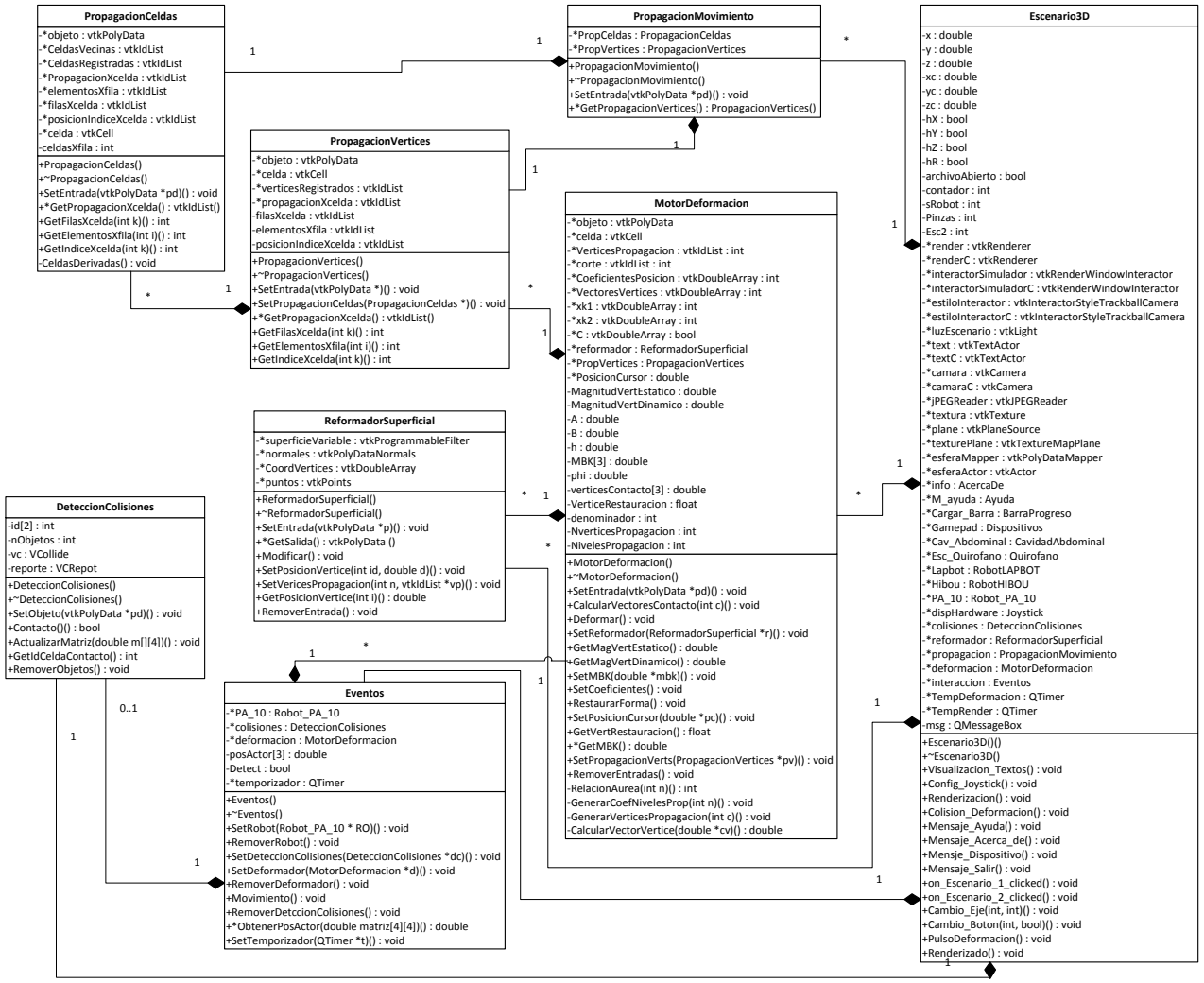

Figura 4-9. Diagrama detallado de clases parte 2.

A continuación se especifican las clases de VTK utilizadas para el desarrollo de la aplicación [20]:

- **vtkActor:** Permite renderizar un objeto en la escena Sus propiedades y posición están dados en coordenadas cartesianas.

- **vtkAssembly:** Permite ensamblar actores dentro de una jerarquía padre-hijo.

- **vtkCamera:** Es una cámara virtual para el renderizado 3D. Proporciona métodos para posicionar y orientar el punto de vista y el punto focal.

- **vtkCell:** Es una clase abstracta que especifica las interfaces para las celdas de datos (puntos, líneas, polígonos y tetraedros de los cuales se componen los conjuntos de datos de visualización).

- **vtkCleanPolyData:** Es un filtro que toma los datos poligonales como entrada y genera datos poligonales como salida. Además puede combinar puntos duplicados, eliminar los puntos que no se utilizan, y transformar celdas degeneradas en formas apropiadas.

- **vtkDoubleArray:** Es una matriz de valores de tipo *double*. Proporciona métodos para la inserción y recuperación de valores y automáticamente se redimensionará para guardar los nuevos datos.

- **vtkIdList:** Lista de identificadores de puntos o celdas.

- **vtkInteractorStyleTrackballCamera**: Permite al usuario manipular de forma interactiva (rotar, zoom, etc.) la cámara, para cambiar la visual de la escena.

- **vtkLight**: Es una luz virtual para renderizado 3D. Proporciona métodos para localizar y apuntar la luz, establecer brillo y color.

- **vtkMatrix4x4:** Representa y manipula las matrices 4x4 de transformación. En concreto trabaja con matrices que se encuentran en 3D usando coordenadas homogéneas.

- **vtkOBJReader:** Sirve para leer archivos obj.

- **vtkPoints:** Representa y manipula puntos 3D.

- **vtkPolyData:** Representa un conjunto de datos como líneas, polígonos y triángulos.

- **vtkPolyDataMapper:** Genera primitivas gráficas a partir de datos poligonales.

- **vtkPolyDataNormals:** Calcula la normal del punto para una malla poligonal, este filtro puede reordenar los polígonos para asegurar la consistencia de orientación a través de los polígonos vecinos. Quita los bordes afilados y duplica puntos con normales separadas para obtener una superficie mejor definida.

- **vtkProgrammableFilter:** Es un filtro que puede ser programado por el usuario. Para utilizar el filtro se define una función que recupera la entrada del tipo correcto, crea los datos, y luego manipula la salida del filtro. El uso de este filtro evita la necesidad de creación de subclases.

- **vtkProperty:** Librería utilizada para la representación y modificación de propiedades de la superficie de un objeto geométrico, tales como iluminación, color, etc.

- **vtkRenderer**: Coordina la renderización de luces, cámara y actores.

- **vtkRenderWindow** : Es un objeto abstracto para especificar el comportamiento de una ventana de representación. Una ventana de renderizado es una ventana en una interfaz gráfica de usuario que permite mostrar imágenes.

- **vtkRenderWindowInteractor:** Provee una plataforma independiente de mecanismos de interacción para eventos del mouse, teclado y tiempo.

- **vtkTextActor** Se utiliza para colocar anotación de texto en una ventana.

- **vtkTextProperty:** Es un objeto que representa las propiedades del texto. Las principales propiedades que se pueden establecer son el color, la opacidad, el tamaño de fuente, familia de fuentes justificación horizontal y vertical, estilos negrita / cursiva / sombra.

4.3. Clases desarrolladas para la implementación del robot PA-10.

Desde el punto de vista de la POO[17], existen dos maneras de codificar una clase, uno es el método en línea, donde en el archivo ".h" se declaran y codifican los métodos. El otro método es el fuera de línea, en el cual se usa el archivo ".h" para declarar los métodos y se crea un archivo ".cpp" para codificar los métodos como tal [21]. En este trabajo se utiliza el método fuera de línea. A continuación se explican las clases más relevantes de la aplicación.

4.3.1. Clase "ObjetoActor".

Esta clase se utiliza para crear los objetos que forman parte del renderizado (escenario).

```
public:
ObjetoActor(const char*,double X = 0,double Y = 0, double Z = 0,double
O = 1,double R = 0.5,double G = 0.5,double B = 0.5, double C = 0);
```

La función más importante dentro de esta clase es el constructor "ObjetoActor", el cual recibe los parámetros necesarios para renderizar el objeto. Estos parámetros son:

- Char *: Entrega la dirección del archivo ".obj" a cargar en el render.

- X, Y y Z: Son las variables que definen la posición en el espacio de la escena donde se debe cargar el objeto.

- O: Este parámetro define qué tan opaco es el objeto cargado en la escena, varía entre 0 y 1, donde 0 es trasparente y 1 es totalmente opaco.

[17] POO: Programación Orientada a Objetos.

- R, G y B: Son los parámetros para determinar el color de las mallas que componen al objeto. Estos van de 0 a 1, equivalente al formato de color que va de 0 a 255.

- C: Este parámetro especifica los objetos a colisionar. Puede tomar valores entre 0 - 2. El valor 0 significa que el objeto no va a colisionar y está definido por defecto. El valor 1 declara el objeto a deformar, mientras que el valor 2 especifica el objeto rígido a colisionar con el objeto deformable.

En la clase donde se cree un objeto del tipo "ObjetoActor", se crea en el archivo ".h" un objeto de la siguiente forma: `ObjetoActor *piezasRobot[22]`. Cabe aclarar que el número 22 indica el número de objetos pertenecientes a la instancia `piezasRobot` de la clase "ObjetoActor". Una vez hecho esto, en el lugar donde se llama el constructor se usa el siguiente comando:

```
piezasRobot[0] = new ObjetoActor("../../SVRQ/Piezas/PA-10/BASE.obj",0,0,0,1,1,1,1);
```

El comando anterior entrega a la clase "ObjetoActor" los parámetros necesarios para cargar un archivo ".obj" en la escena (render).

4.3.2. Clase "Robot PA-10".

4.3.2.1. Archivo .h de la clase Robot PA-10.

En este archivo se declaran los atributos y métodos de las que consta el módulo "Robot PA-10.cpp", así como también se declaran otros archivos de encabezado necesarios para su funcionamiento.

```
#ifndef ROBOT_PA_10_H
#define ROBOT_PA_10_H

#include <qmath.h>
#include "ObjetoActor.h"
#include "vtkAssembly.h"
#include "m2c.h"
#include <vtkMatrix4x4.h>

using namespace std;

class Robot_PA_10
{
private:
  void Ensamblar_Robot();
  void Mover_Robot(double, double, double, double ,double,double);
```

```
public:
   ObjetoActor *piezasRobot[22]; //Número de piezas que conforman el robot
   vtkAssembly *articulacionesR[6]; // Número de articulaciones del robot
   vtkAssembly *robotEnsamblado;
   vtkMatrix4x4 *Matriz;

   Robot_PA_10();
   ~Robot_PA_10();
   double Px,Py,Pz;
   double coordEspacio[2]; //Vector que almacena las posiciones(X,Y,Z) del MGD
   vtkMatrix4x4 *GetMatriz();

   void MGI(double, double, double);
   void MGD(double, double, double, double, double, double, double);
   inline vtkAssembly *Retorno_Robot(){ return (robotEnsamblado);};
};
#endif
```

4.3.2.2. Archivo .cpp de la clase Robot PA-10.

En este archivo, se implementa el código para las funciones que fueron declaradas anteriormente en el archivo .h.

En esta clase se muestra cómo cargar un objeto en la escena.

```
void Robot_PA_10::Ensamblar_Robot()
{
   piezasRobot[0] = new ObjetoActor("../../SVRQ/Piezas/PA-
10/BASE.obj",0,0,0,1,1,1,1);
   piezasRobot[1]     =     new     ObjetoActor("../../SVRQ/Piezas/PA-
10/HOMBRO.obj",0,0.11,0,1,1,1,1);

...
```

Cada uno de los objetos llamados en esta función, carga cada una de las piezas correspondientes del robot, partiendo desde la base hasta el efector final. El vector `piezasRobot[n]` carga todos los archivos “.obj” que forman la estructura del Robot PA-10, cada vez que se crea el objeto de la clase “ObjetoActor” este recibe los parámetros necesarios.

Para conectar las estructuras que componen el robot, se utiliza la lógica padre-hijo, por lo que el robot se empieza a ensamblar desde la última articulación hasta la base. Esta forma de ensamblado implica que al ordenar el posicionamiento de alguna articulación específica, se verán afectados los respectivos hijos de la pieza.

```
articulacionesR[6] = vtkAssembly::New();//En la articulación 6 se carga la
muñeca 3.
articulacionesR[6]->AddPart(piezasRobot[7]->objActor);
articulacionesR[6]->SetOrigin(0,0.868,0);
```

```
articulacionesR[0] = vtkAssembly::New();//En la articulación 0 se carga el
hombro.
articulacionesR[0]->AddPart(piezasRobot[1]->objActor);
articulacionesR[0]->AddPart(piezasRobot[9]->objActor);
articulacionesR[0]->AddPart(piezasRobot[10]->objActor);
articulacionesR[0]->AddPart(articulacionesR[1]);

robotEnsamblado = vtkAssembly::New();//Se añade la base del robot.
robotEnsamblado->AddPart(articulacionesR[0]);
robotEnsamblado->AddPart(piezasRobot[0]->objActor);
robotEnsamblado->AddPart(piezasRobot[8]->objActor);
robotEnsamblado->AddPosition(-0.075,-0.23,-0.45); //Origen del robot.
```

El objeto "`robotEnsamblado`", contiene la estructura completa del robot que se observa en la escena del render (Figura 4-10). La combinación de las funciones "`vtkAssembly::New`", "`AddPart`", "`AddPosition`", y el vector "`articulacionesR[n]`" permiten simular el comportamiento de las articulaciones del robot, ya que convierten cada articulación en la referencia necesaria para que el objeto haga el movimiento en el espacio [21].

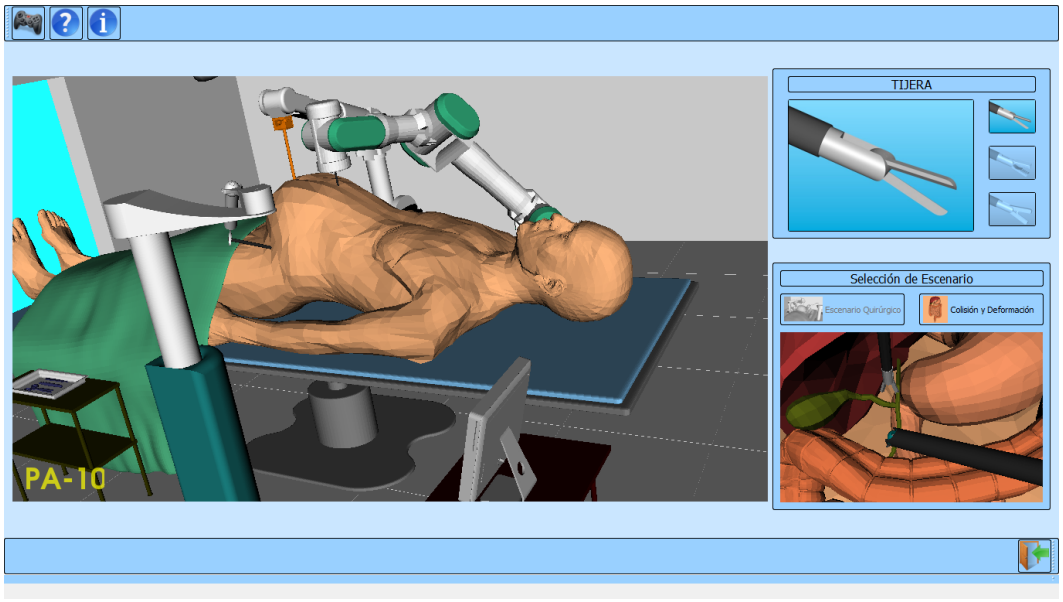

Figura 4-10. Robot PA-10 ensamblado en VTK.

Por otro lado, esta clase posee un método llamado MGI, el cual tiene codificado las ecuaciones necesarias para calcular según los parámetros recibidos desde el programa principal, la posición (en longitud) y orientación (en grados) para cada articulación y así poder alcanzar la posición requerida.

Finalmente este método entrega a un método llamado "Mover_Robot" los parámetros necesarios para posicionar las articulaciones con el fin de alcanzar la posición deseada.

```
void Robot_PA_10::Mover_Robot(double t1, double t2, double t3,double
t4,double t5,double t6)
{
   articulacionesR[0]->SetOrientation(0,t1,0); //Orientación en Y
   articulacionesR[1]->SetOrientation(t2,0,0); //Orientación en X
   articulacionesR[2]->SetOrientation(0,t3,0); //Orientación en Y
   articulacionesR[3]->SetOrientation(t4,0,0); //Orientación en X
   articulacionesR[4]->SetOrientation(0,t5,0); //Orientación en Y
   articulacionesR[5]->SetOrientation(t6,0,0); //Orientación en X
 }
```

Donde t1, t2, t3, t4, t5, t6, son los ángulos calculados por el MGI, para orientar cada una de las articulaciones y llevar el efector final del robot a la posición deseada.

4.3.3. Clase “m2c”.

Esta clase se utiliza para encontrar las posiciones de optimización de codo y muñeca, que son entregados al MGI para orientar cada una de las articulaciones, con el fin de que el robot respete el paso por el trocar.

4.3.3.1. Archivo .h de la clase m2c.

```
#define PI 3.141592
#define RL3 0.40
#define RL5 0.35
#define RL7 0.08
#define RL8 0.28
#define RLS 0.36

using namespace std;

double  f_solve_pa10(double t[5],double x,double y, double z);
void nelmin ( double fn ( double x[], double x1, double y, double z),
int n, double start[], double xmin[], double *ynewlo, double reqmin,
double step[], int konvge, int kcount,   int *icount, int *numres, int
*ifault , double COR[]);
```

Los métodos implementados para calcular las soluciones de las ecuaciones presentadas en el MGI del robot son “f_solve_pa10“ y “nelmin“.

El método “f_solve_pa10“ recibe cuatro parámetros de entrada. El primer parámetro contiene las posiciones iniciales de cada articulación para empezar el proceso de optimización de codo y muñeca. Los tres parámetros restantes son las posiciones cartesianas X, Y y Z enviadas desde el *joystick*. Este método retorna el mínimo valor posible que puede tomar los valores del método “nelmin“.

El método de optimización “nelmin” tiene 13 parámetros. De los cuales siete son de entrada y seis de salida. Para que el método funcione hay que especificar la función y el número de variables a optimizar, las condiciones iniciales y la posición deseada del órgano terminal. Este método retorna las posiciones optimizadas de codo y muñeca para que el robot PA-10 respete el paso por el trocar.

Se realiza el cambio de optimización de Levenberg Mardquart a Nelder y Mead [22], ya que esta técnica no depende de las derivadas parciales del sistema de ecuaciones como lo utiliza el método numérico de Levenberg Mardquart explicado en el capítulo anterior. Por tal razón se escoge este algoritmo que agiliza la solución del problema.

5. Algoritmo de detección de colisiones y deformación.

Para lograr que el efector final del robot PA-10 interactúe con los órganos de la cavidad abdominal, se hace uso del algoritmo “Visualización y deformación de objetos virtuales 3d” (Figura 5-1), [23], desarrollado por los ingenieros Raúl Gómez y Faber Montero.

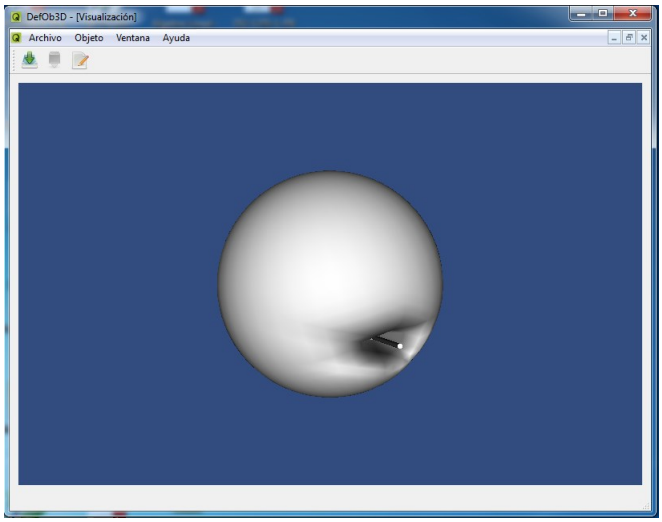

Figura 5-1. Visualización y deformación de objetos virtuales 3D.

Esta aplicación permite al usuario deformar un objeto mediante acciones de presión y estiramiento realizadas por medio del mouse y el teclado sobre un punto de la superficie del objeto. Al eliminar la presión ejercida sobre el objeto deformable este puede recuperar su forma original.

Partiendo de esta aplicación se realizan los cambios pertinentes para que el software desarrollado (SVRQ), presente la colisión y deformación con algunos órganos del abdomen y se haga por medio del efector final del PA-10 el cual es movido a través de un *joystick*.

5.1. Acondicionamiento de los objetos a colisionar.

El primer paso que se realiza en el algoritmo de detección de colisiones y deformación es seleccionar los objetos a colisionar (órganos y efector final del PA-10). Para ello se definen en las clases "CavidadAbdominal" y "Robot_PA_10" los objetos correspondientes a la clase "ObjetoActor", con su respectivo indicador de colisión, C = 1 para los órganos y C = 2 para el efector final, como se explicó en el capítulo anterior en la sección 4.3.1.

- **Clase "CavidadAbdominal"**

```
Organos[9]= new ObjetoActor (" ../ ../ SVRQ/ Piezas/ SISTEMA
_DIGESTIVO/ ORGANOS.obj" ,0.4 ,0.25,-0.1325,1,0.976,0.584,0.392,C=1);
```

- **Clase "Robot_PA_10"**

```
piezasRobot[15] = new ObjetoActor (" ../ ../ SVRQ/ Piezas/ PA-10/
PINZA_T.obj",0,0.868,0,1,0.156863,0.156863,0.156863,C=2);
```

Al efectuar este procedimiento, al objeto "`Organos[9]`" se le realiza un proceso de filtrado y limpieza, por medio de las instancias "`Filtro`" de la clase "vtkTriangleFilter" y "`Limpieza`" de la clase "vtkCleanPolyData". Este proceso es importante, ya que de esta forma se garantiza que el objeto está compuesto por polígonos triangulares a los que se les ha eliminado los vertices redundantes para evitar inconvenientes con el reporte de colisiones, el reformador superficial y propagación de movimiento. Por el contrario al objeto "`piezasRobot[15]`", el procedimiento que se le realiza es la lectura de polígonos triangulares que lo conforman para luego ser enviado a la clase "DetecciónColisiones".

5.2. Posición de los objetos a colisionar.

Por otro lado en la clase "Robot_PA_10", se define un método que retorne la "`Matriz`" instancia de la clase "vtkMatrix4x4". Esta matriz almacena constantemente la posición del órgano terminal del robot, el cual es calculado por medio del método MGD.

Antes de almacenar los valores de posición del efector final en la matriz, se hace necesario efectuar un proceso de escalización, con el fin de acoplar los movimientos del órgano terminal a las posiciones cartesianas, en las que se encuentra el órgano a deformar. Para ello hay que tener en cuenta la posición en que se ubican los objetos en el entorno Blender, ya que estas posiciones afectan a los órganos y al actor a la hora de colisionar. Por esta razón, se realiza el siguiente procedimiento:

Primero se calcula el área de trabajo que recorre el órgano terminal, observando todos los movimientos que realiza la pinza efectuados desde el dispositivo de mando. Para ello se imprime en consola las posiciones del órgano terminal calculadas en el MGD y se lleva al joystick a sus valores máximos en los ejes X, Y y Z. Se obtienen los valores mostrados en la figura 5-2, donde la distancia recorrida en el eje X, Y y Z se expresa en las ecuaciones (5.1), (5.2) y (5.3). La figura 5-2 tiene el origen en X = 0.32m, Y = 0.48m, Z = 0 m.

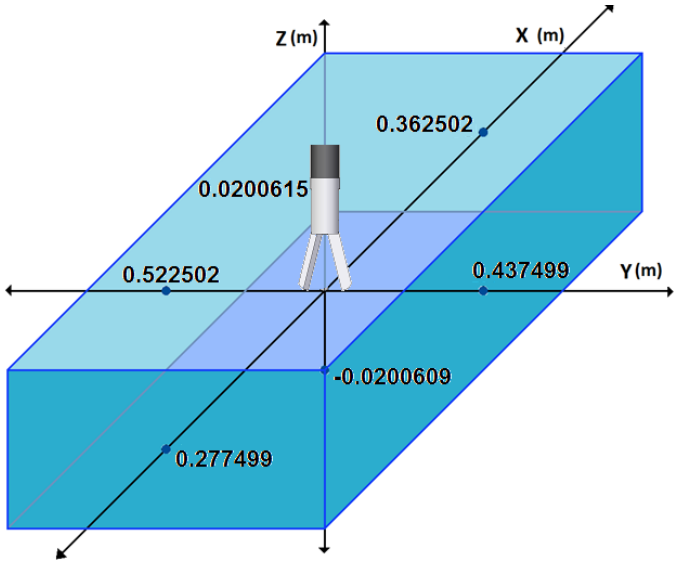

Figura 5-2. Área de trabajo del órgano terminal.

$$\Delta X = Xmax - Xmin = 0.362502m - 0.277499m = 0.085003m \ (5.1)$$

$$\Delta Y = Ymax - Ymin = 0.522502m - 0.437499m = 0.085003m \ (5.2)$$

$$\Delta Z = Zmax - Zmin = 0.0200615\text{m} \ - (-0.0200609\text{m}) = 0.0401224m \ (5.3)$$

Posteriormente se calcula el área de trabajo que recorre el efector final, con respecto a las posiciones cartesianas de Blender. El área de trabajo en Blender se calcula llevando a la pinza en el entorno de Blender a las posiciones que visualmente llega el efector final en el entorno SVRQ (Figura 5-

3). El origen del plano cartesiano en Blender es Xb = 0 m, Yb =0 m, y Zb = 0 m. Donde Xb, Yb y Zb son los ejes X, Y y Z de Blender.

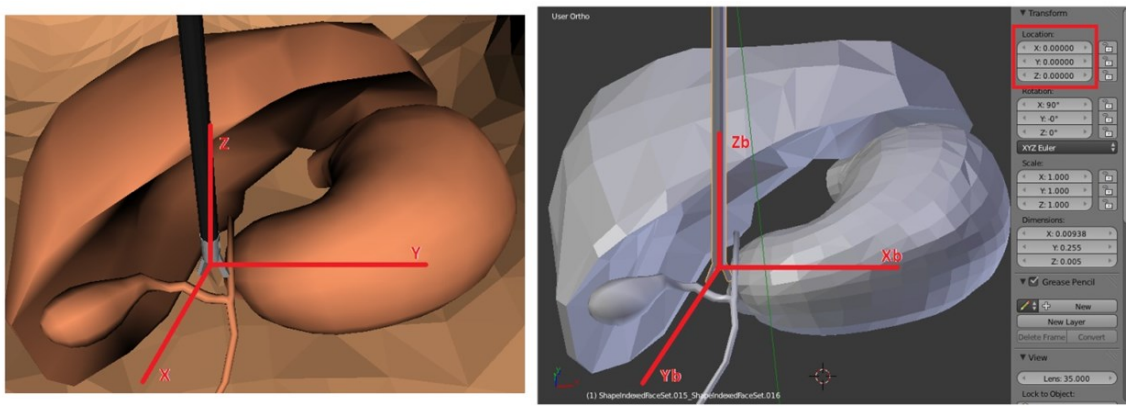

Figura 5-3. Visualización en SVRQ y Blender.

En la figura 5-4 se comparan los planos cartesianos de los dos entornos. En este se puede apreciar que un desplazamiento en el eje Y en SVRQ realiza un desplazamiento en el eje Xb de Blender, de igual forma un desplazamiento en el eje X de SVRQ genera un desplazamiento en el eje Yb de Blender. Cabe resaltar que el eje Y de SVRQ se encuentra invertido con respecto al eje Xb de Blender, es decir cuando Y alcance un Ymax en SVRQ en Blender Xb alcanza un Xbmin. Para pasar del plano cartesiano de SVRQ al plano cartesiano de Blender, es necesario calcular la matriz de transformación (5.4) que permita realizar este procedimiento.

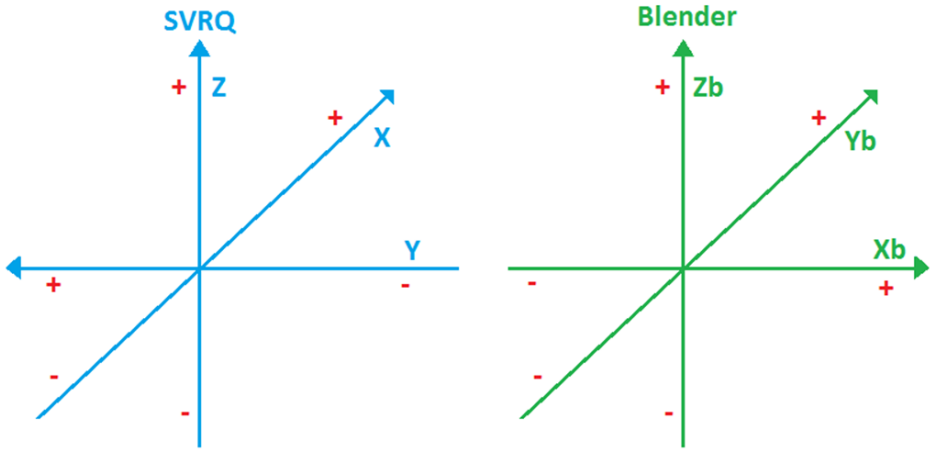

Figura 5-4. Planos cartesianos en SVRQ y Blender.

$$\begin{bmatrix} 0 & 1 & 0 & X \\ -1 & 0 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.4)$$

Ahora se calcula el desplazamiento en el eje Xb de Blender cuando la pinza se mueve en el eje Y de SVRQ. Este desplazamiento se encuentra encerrado en el recuadro de color rojo. Se debe tener en cuenta que el eje Y de SVRQ se encuentra invertido con respecto al eje Xb de Blender, por tal razón un desplazamiento que alcance el Ymax en SVRQ alcanza un desplazamiento en Xbmin en Blender (Figura 5-5). De igual forma si el efector final alcanza el desplazamiento Ymin en SVRQ en Blender alcanzara el Xbmax (Figura 5-6).

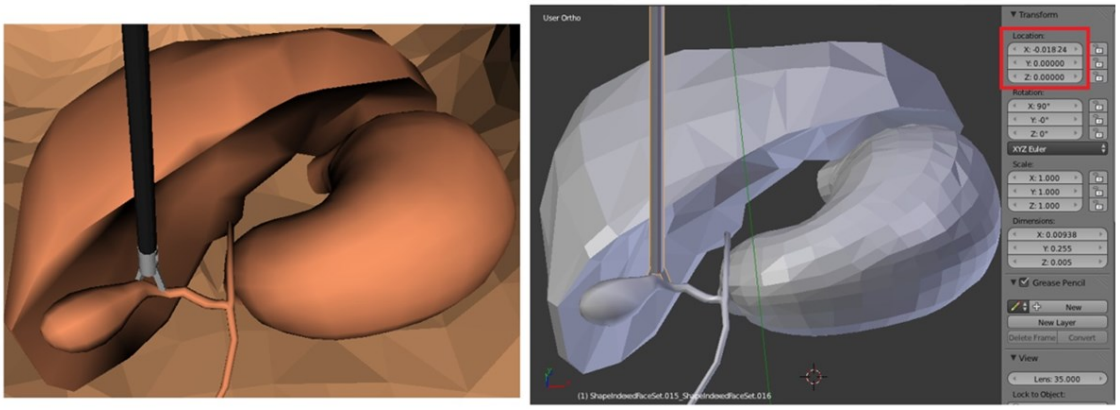

Figura 5-5. Desplazamiento en Ymax (SVRQ) y en Xbmin (Blender).

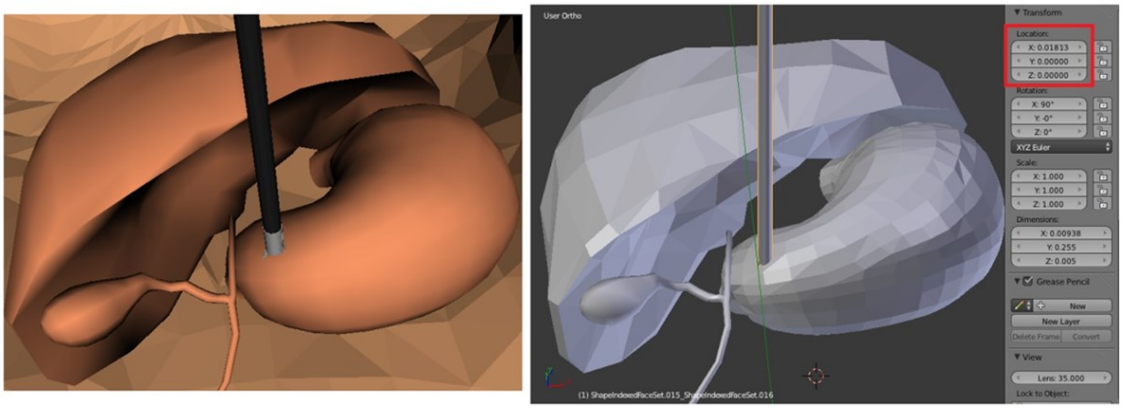

Figura 5-6. Desplazamiento en Ymin (SVRQ) y en Xbmax (Blender).

La distancia recorrida del efector final en el eje Xb de Blender se expresa en la ecuación (5.5).

$$\Delta Xb = Xbmax - Xbmin = 0.01813m - (-0.01824m) = 0.03637m \quad (5.5)$$

Posteriormente se calcula el desplazamiento en el eje Yb de Blender cuando la pinza se mueve en el eje X de SVRQ. Este desplazamiento se encuentra encerrado en el recuadro de color rojo. En este caso un desplazamiento que alcance el Ymax en SVRQ alcanza un desplazamiento en Xbmax en Blender

(Figura 5-7). De igual forma si el efector final alcanza el desplazamiento Ymin en SVRQ en Blender alcanzara el Xbmin (Figura 5-8).

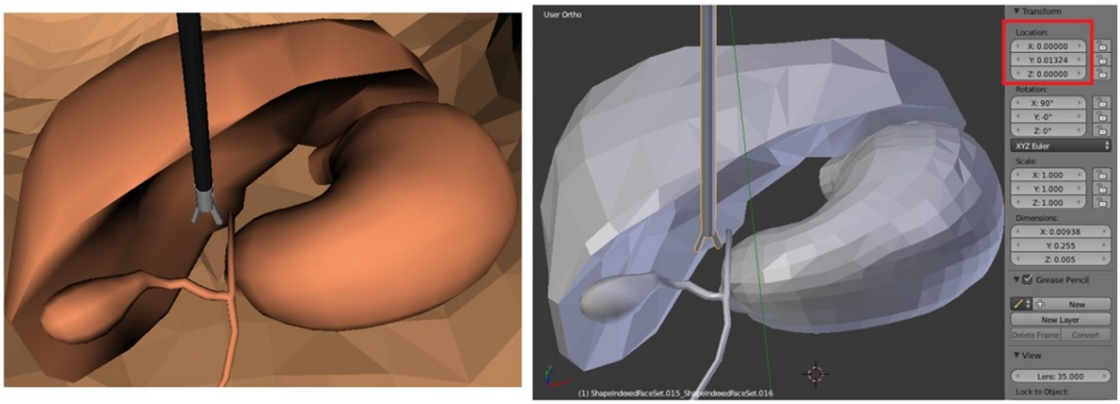

Figura 5-7. Desplazamiento en Ymax (SVRQ) y en Ybmax (Blender).

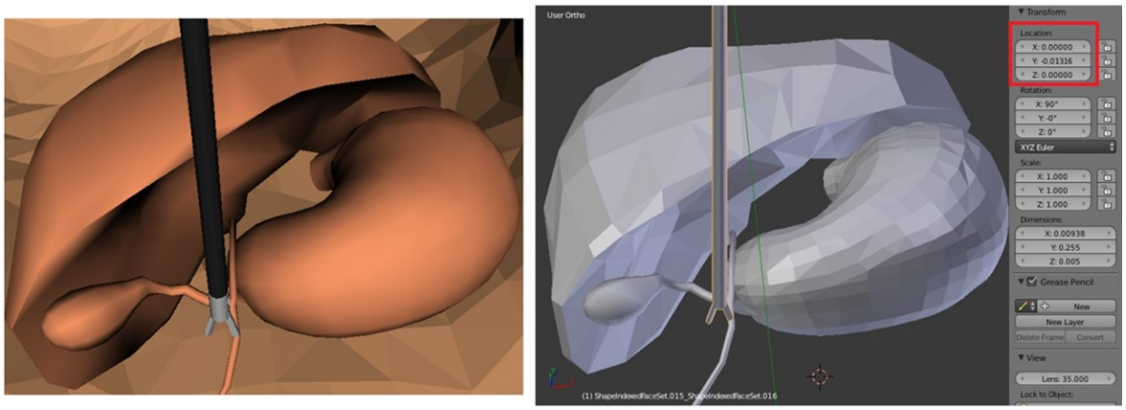

Figura 5-8. Desplazamiento en Ymin (SVRQ) y en Ybmin (Blender).

La distancia recorrida del efector final en el eje Yb de Blender se expresa en la ecuación (5.6).

$$\Delta Yb = Ybmax - Ybmin = 0.01324m - (-0.01316m) = 0.0264m \quad (5.6)$$

Luego se calcula el desplazamiento en el eje Zb de Blender cuando la pinza se mueve en el eje Z de SVRQ. Este desplazamiento se encuentra encerrado en el recuadro de color rojo. En este caso el eje Z de SVRQ corresponde con el eje Zb de Blender, por consiguiente un desplazamiento que alcance el Zmax en SVRQ alcanza un desplazamiento en Zbmax en Blender (Figura 5-9). De igual forma si el efector final alcanza el desplazamiento Zmin en SVRQ en Blender alcanzara el Zbmin (Figura 5-10).

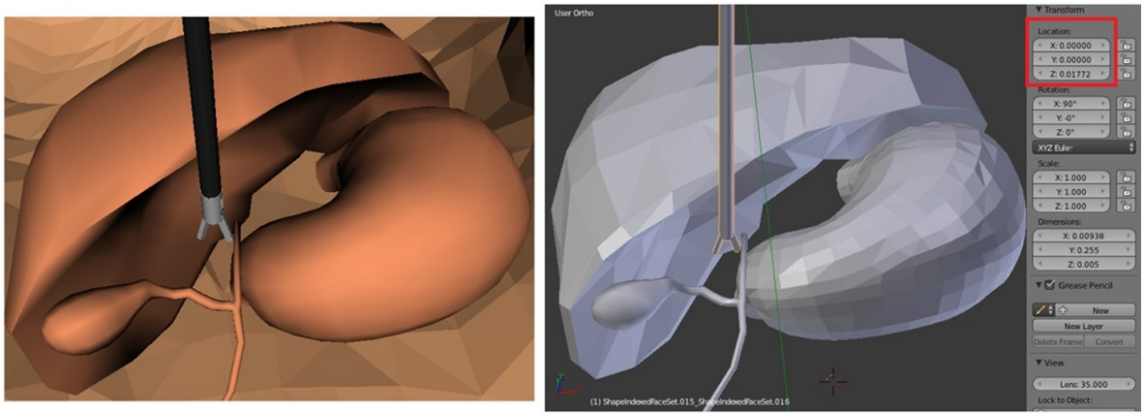

Figura 5-9. Desplazamiento en Zmax (SVRQ) y en Zbmax (Blender).

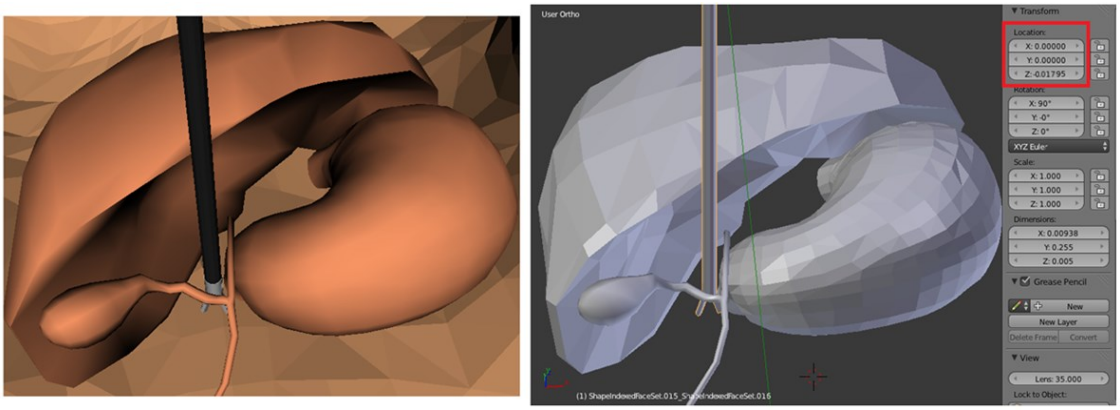

Figura 5-10. Desplazamiento en Zmin (SVRQ) y en Zbmin (Blender).

La distancia recorrida del efector final en el eje Zb de Blender se expresa en la ecuación (5.7).

$$\Delta Zb = Zbmax - Zbmin = 0.01772\text{m} - (-0.01795\text{m}) = 0.03567m \quad (5.7)$$

Después de calcular el área de trabajo en los dos planos con diferente origen, se procede a escalizar estos resultados, para que un desplazamiento en el eje Y de SVRQ se vea reflejado en el eje Xb de Blender, un desplazamiento en el eje X de SVRQ se vea reflejado en el eje Yb de Blender y un desplazamiento en el eje Z de SVRQ corresponda al eje Zb de Blender. Además de esto, hay que llevar el origen de coordenadas de la figura 5-2, al origen de coordenadas de la figura 5-3, es decir $Xb_{origen} = Y_{origen} - 0.48m$, así mismo, para los otros ejes. Como se dijo anteriormente el eje Y de SVRQ se encuentra invertido con respecto al eje Xb de Blender, por lo que es necesario anteponer un signo menos ante él como se muestra en la matriz de transformación (5.4), logrando así, que cuando Y alcance el valor Ymax, Xb debe alcanzar igualmente Xbmax (Figura 5-11).

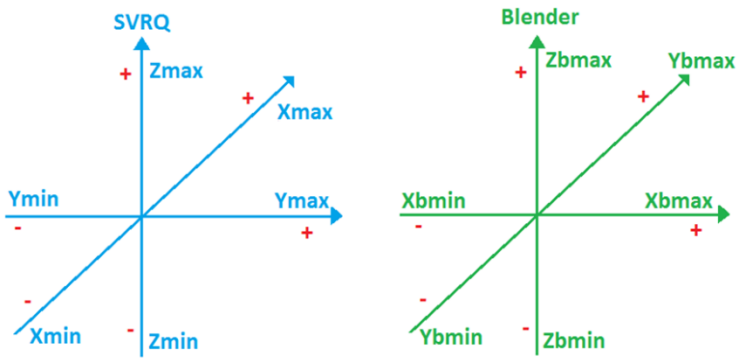

Figura 5-11. Inversión del eje Y de SVRQ.

Las ecuaciones que representan la escalización entre los dos planos cartesianos son las (5.8), (5.9) y (5.10).

$$\Delta CoordenadaXb = \frac{-(Yorigen - 0.48m) * \Delta Xb}{\Delta Y}$$

$$\Delta CoordenadaXb = \frac{-(Yorigen - 0.48m) * 0.03637m}{0.085003m} \quad (5.8)$$

$$\Delta CoordenadaYb = \frac{(Xorigen - 0.32m) * \Delta Yb}{\Delta X}$$

$$\Delta CoordenadaYb = \frac{(Xorigen - 0.32m) * 0.0264m}{0.085003m} \quad (5.9)$$

$$\Delta CoordenadaZb = \frac{Z * \Delta Zb}{\Delta Z}$$

$$\Delta CoordenadaZb = \frac{Z * 0.03567m}{0.0401224m} \quad (5.10)$$

Después de hallar las ecuaciones que transforman el desplazamiento del plano de SVRQ al plano de Blender es necesario tener en cuenta el plano de la "`Matriz`" instancia de la clase "vtkMatrix4x4" (Figura 5-12). En esta figura se puede apreciar que un desplazamiento en la coordenada Xb corresponde a un desplazamiento en el eje X de la matriz, un desplazamiento en la coordenada Yb produce un desplazamiento en el eje Z de la matriz y un desplazamiento en la coordenada Zb se ve reflejado en el eje Y de la matriz. Es de tener en cuenta que la coordenada Yb se encuentra invertida con respecto al eje Z de la matriz por lo que es necesario anteponer un signo menos ante ella. Para pasar del plano cartesiano de Blender al plano cartesiano de la "`Matriz`" instancia, es necesario calcular la matriz de transformación (5.11) que permita realizar este procedimiento.

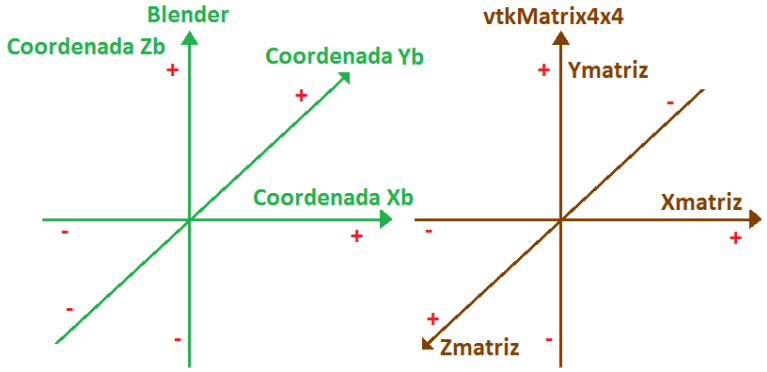

Figura 5-12. Análisis de los planos cartesianos de Blender y de vtkMatrix4x4.

$$\begin{bmatrix} 1 & 0 & 0 & Xb \\ 0 & 0 & -1 & Yb \\ 0 & 1 & 0 & Zb \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.11)$$

Las ecuaciones (5.12), (5.13) y (5.14) son las que se almacenan en el elemento "`Matriz`".

$$Coordenada[0] = \Delta CoordenadaXb \quad (5.12)$$

$$Coordenada[1] = \Delta CoordenadaZb \quad (5.13)$$

$$Coordenada[2] = -\Delta CoordenadaYb \quad (5.14)$$

Donde la coordenada [0] corresponde al eje X de la matriz, la coordenada [1] corresponde al eje Y de la matriz y la coordenada [2] corresponde al eje Z de la matriz.

5.3. Detección de colisiones y deformación.

Después de obtener la matriz de posición del actor que va a deformar a los órganos, esta matriz es enviada a la clase "Eventos", por medio de la instancia "`PA_10`" de la clase "Robot_PA_10". En la clase "Eventos" también interactúan otras dos clases que son "DeteccionColisiones" y "MotorDeformacion" con sus respectivas instancias "`colisiones`" y "`deformacion`".

A continuación se expresa en código lo dicho anteriormente:

```
interaccion->SetRobot(PA_10);
interaccion->SetDeteccionColisiones(colisiones);
interaccion->SetDeformador(deformacion);
```

Donde "`interaccion`" es una instancia de la clase "Eventos".

En el .cpp de la clase "Eventos" se ejecuta la siguiente porcion de codigo, en la cual se lee la instancia "`Matriz`" y se envía la posición del actor rígido a la instancia "`deformacion`" y "`colisiones`", para verificar si existe colisión y posteriormente empezar a deformar el órgano.

```
deformacion->SetPosicionPinza(ObtenerPosActor(PA_10->GetMatriz()->Element));
colisiones->ActualizarMatriz(PA_10->GetMatriz()->Element);
```

Las clases de los componentes de colisión y deformación que se explican a continuación, son implementadas de igual forma como se desarrollaron en la tesis "Visualización y deformación de objetos virtuales 3D" [23]. Aquí se explicara su funcionamiento, pero para mas información remitirse a ella.

5.3.1. Componente de colisión.

Este componente se hace necesario para que el sistema pueda verificar si existe contacto entre el órgano terminal del robot PA-10 y los órganos a deformar. Por tal razón se hace uso de la clase "DeteccionColisiones" que permite un contacto directo entre el efector final y los órganos. Para ello, a la instancia "`colisiones`" se le envían los objetos a colisionar, además debe leerse la posición actual del elemento "`Matriz`" para conocer donde se encuentra el actor, y retornar un dato booleano (contacto) que indica si existe o no colisión, además de reconocer el triángulo donde se produce el contacto.

```
//Envio de los objetos a colisionar
colisiones->SetObjeto(Cav_Abdominal->Organos[9]->Limpieza->GetOutput());
colisiones->SetObjeto(PA_10->piezasRobot[15]->polyData);

//Actualiza la Matriz con respecto a la posción del efector final
colisiones->ActualizarMatriz(PA_10->GetMatriz()->Element);

//Indica si existe colision o no
colisiones->Contacto();

//Reporta el triangulo donde se produce contacto
colisiones->GetIdCeldaContacto();
```

En esta clase, la clase V-Collide se configuró para reportar un solo triangulo, de manera que este sistema entregará el valor del primer contacto que se registre entre el efector final y los órganos deformables [23].

5.3.2. Componente de deformación.

Este componente se encarga de deformar los órganos cuando entran en contacto con el efector final del PA-10 y de regresar a su forma original cuando haya pasado la presión del actor rígido. Se encuentra compuesto por las clases "ReformadorSuperficial", "PropagacionMovimiento", y "MotorDeformacion", siendo esta ultima la más importante dentro de este componente.

La clase "ReformadorSuperficial" contiene los datos de las coordenadas de los vértices de los órganos, por tanto la malla de los órganos a deformar está formada por esta información. Estos datos podrán ser modificados por la clase "MotorDeformacion". "PropagacionMovimiento" entrega información de los vértices que serán afectados por la deformación en los diferentes niveles de propagación. "MotorDeformacion" define el cómo se deben modificar las coordenadas de los vértices de acuerdo al modelo deformable [23].

```
//Envío de las coordenadas de los vértices de los órganos a la clase
MotorDeformacion
deformacion->SetReformador(reformador);

//Información de los vértices afectados por la deformación
deformacion->SetPropagacionVerts( propagacion->GetPropagacionVertices());
```

5.3.2.1. Clase "ReformadorSuperficial"

La clase "ReformadorSuperficial" es la encargada de modificar la posición de los vértices de los órganos a deformar. Se encuentra conformada por las clases vtkProgrammableFilter, vtkDoubleArray, vtkPolyDataNormals.

vtkProgrammableFilter permite modificar la posición de todos los vértices de los órganos, permitiendo variar su superficie, la clase vtkDoubleArray es un arreglo dinámico de tipo *doublé* donde se almacenan los datos de las coordenadas de los vértices de los órganos y este arreglo es utilizado por vtkProgrammableFilter para modificar los valores de los vértices. La clase vtkPolyDataNormals elimina los bordes afilados de la superficie de los órganos recuperando su apariencia original [23].

```
//Convierte los órganos rígidos a un objeto deformable
reformador->SetEntrada(Cav_Abdominal->Organos[9]->Limpieza->GetOutput());

//Retorna los órganos para ser deformados
reformador->GetSalida();
```

5.3.2.2. Clase “PropagacionMovimiento”

Para explicar cómo funciona la clase “PropagacionMovimiento”, es necesario conocer cómo se propaga el movimiento dentro de un mallado triangular. Para ello se tiene la figura 5-13 en la que se observan los órganos con su respectiva maya triangular.

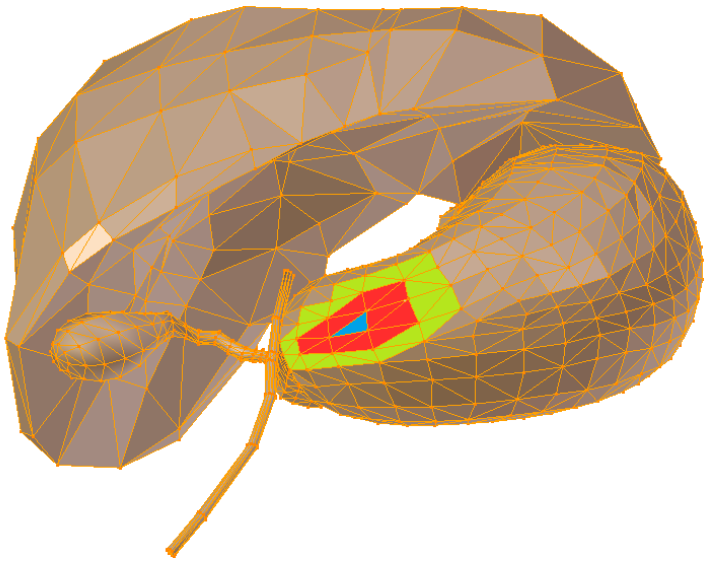

Figura 5-13.Propagación de movimiento en un mallado triangular.

Si se supone que el efector final del PA-10 impacta el triángulo de color azul y lo empieza a presionar hacia adentro, la propagación de movimiento primero se debe efectuar sobre los triángulos que están alrededor de él, es decir, los triángulos de color rojo. Estos triángulos conforman el primer nivel de propagación. Si la presión se sigue efectuando sobre el triángulo azul, se van a ver también afectados los triángulos de color verde, estos triángulos hacen parte del segundo nivel de propagación. El mismo comportamiento se sigue presentando hasta los triángulos del nivel X. Esta forma de propagación permite obtener el nivel de propagación de los vértices. Cabe destacar que este comportamiento se presenta para todos los polígonos triangulares que conforman el objeto.

El propósito de los niveles de propagación por vértices es en definitiva asignar una relación de movimiento a los vértices según el nivel en que se encuentren [23].

La clase “PropagaciónMovimiento” se encuentra compuesta por las clases “PropagacionCeldas” y “PropagacionVetices”. La clase “PropagacionCeldas” se encarga de encontrar los triángulos vecinos de cada triángulo y almacena

esta informacion en un arreglo dinámico de tipo entero. Mientras que la clase “PropagacionVertices” utiliza este arreglo de celdas para reflejar la misma organización con vértices [23].

```
//Obtiene los órganos a los que se le acomodan los vértices para una
propagación
propagacion->SetEntrada(Cav_Abdominal->Organos[9]->Limpieza->GetOutput());

//Contiene el arreglo de vértices que permite una propagación
propagacion->GetPropagacionVertices();
```

5.3.2.3. Clase “MotorDeformacion”

La clase “MotorDeformacion” es la encargada de deformar y de volver acomodar los órganos a su condición inicial. Para deformar los órganos se utiliza un modelo deformable basado en el modelo de masa resorte amortiguador. Este modelo permite encontrar la relación de movimiento entre los distintos nodos respecto al desplazamiento del nodo a mover. Una breve explicación de este modelo se muestra a continuación:

En la figura 5-14 se observa una unión de resortes en serie y paralelo unidos a los nodos x_1, x_2 y x_3. Estos son perturbados por una fuerza F que los hala a la derecha, por consiguiente se efectuará un movimiento de translación en x_1, x_2 y x_3. Se debe encontrar el desplazamiento que se realiza en cada nodo con respecto a la fuerza aplicada, donde k es la constante elástica del resorte.

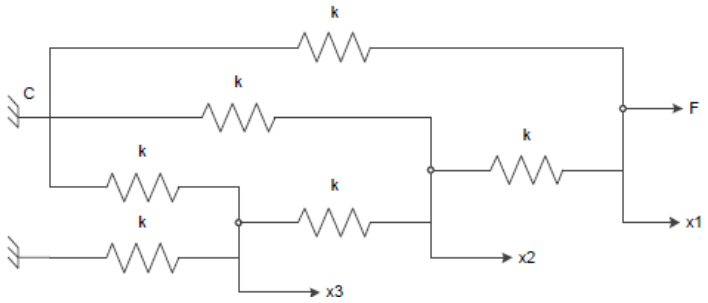

Figura 5-14. Unión de resortes en serie y paralelo.

Para x_3.

$$2kx_3 + k(x_3 - x_2) = 0 \rightarrow x_3 = \frac{1}{3}x_2 \quad (5.15)$$

Para x_2.

$$kx_2 + k(x_2 - x_3) + k(x_2 - x_1) = 0 \quad (5.16)$$

Remplazando (5.15) en (5.16).

$$kx_2 + k\left(x_2 - \frac{1}{3}\,x_2\right) + k(x_2 - x_1) = 0 \rightarrow\ x_2 = \frac{3}{8}\,x_1 \ (5.17)$$

Para x_1.

$$kx_1 + k(x_1 - x_2) = F \ (5.18)$$

Remplazando (5.17) en (5.18).

$$kx_1 + k\left(x_1 - \frac{3}{8}\,x_1\right) = F \rightarrow\ F = \frac{13}{8}\,kx_1 \ (5.19)$$

De los anteriores resultados se puede observar que entre más lejos esté el nodo de donde se aplica la fuerza, menor va a ser la perturbación en él. Esta conclusión es importante, ya que explica como los vértices del objeto se ven afectados por el movimiento de un vértice al que se le ha aplicado una acción externa [23].

Para restaurar la forma de los órganos, la clase “MotorDeformacion” consta de dos elementos. El elemento vértices propagados proporciona información de los vértices afectados en la malla y el elemento sistema discreto MBK contiene la ecuación diferencial discretizada del sistema masa resorte amortiguador que ejecuta la dinámica de restauración [23].

Ecuación discretizada:

$$x(t) = Ax(t-1) - Bx(t-2) + C \ (5.20)$$

Dónde:

$$A = \left(2 - \frac{b}{m}\Delta t - \frac{k}{m}\Delta t^2\right) \ (5.21)$$

$$B = \left(1 - \frac{b}{m}\Delta t\right) (5.22)$$

$$C = \frac{kx}{m}\Delta t^2 \ (5.23)$$

A continuación se representa en código como interactúa la clase deformación.

```
//Recibe el objeto a deformar
deformacion->SetEntrada(Cav_Abdominal->Organos[9]->Limpieza->GetOutput());

//Envío de las coordenadas de los vértices de los órganos a la clase
MotorDeformacion
deformacion->SetReformador(reformador);
```

```
//Información de los vértices afectados por la deformación
deformacion->SetPropagacionVerts( propagacion->GetPropagacionVertices());

//Se encarga de deformar los órganos
deformacion->Deformar();

//Restaura la forma de los órganos
deformacion->RestaurarForma();
```

6. Pruebas de la aplicación.

Las pruebas y resultados obtenidos se realizaron con las siguientes características hardware (para conocer cómo se maneja la aplicación diríjase al anexo C):

Ordenador:

- Equipo: Toshiba.
- Procesador: Intel(R) Core(TM) i5-2430M 2.40GHz.
- Memoria (RAM): 4,0 GB.

Joystick:

- Equipo: Extreme 3D Pro.
- Requisitos del sistema: Windows 8, Windows 7 o Windows Vista.
- Puerto: USB
- Funciones: Control de timón con eje de torsión de gran precisión. Base sólida y estable. Disparador de acción rápida.

Gamepad:

- Equipo: MaxFire Blaze 3.
- Requisitos del sistema: Windows 7 o Windows Vista.
- Puerto: USB.
- Numero de botones: 12.
- Compatibilidad: PC / PS2 / PS3.

Al abrir la aplicación SVRQ se aprecia la *GUI* desarrollada, donde se puede seleccionar dos escenarios a trabajar (Escenario Quirúrgico y Colisión y Deformación). En el “Escenario Quirúrgico” existen tres ventanas. La primera ventana presenta un ambiente donde se encuentra el paciente con los robots. En la segunda ventana se observan tres herramientas quirúrgicas (tijera, retractor y pistola endoclip), las cuales son utilizadas por el robot tipo PA-10. Por último, la tercera ventana permite visualizar el interior de la cavidad abdominal del paciente, por medio de una cámara que es manipulada por el robot porta endoscopio Hibou.

Al seleccionar el segundo escenario “Colisión y Deformación”, el usuario puede interactuar con algunos órganos de la cavidad abdominal, llevando el efector final del PA-10 a colisionar con los órganos y defórmalos si realiza una presión sostenida sobre él. En este escenario se removió los robots Hibou y

Lapbot buscando velocidad de renderizado para que la deformación se pueda apreciar mejor.

6.1. Manipulación del robot.

La prueba de manipulación del robot PA-10 se llevó a cabo en el escenario quirúrgico de la aplicación. Para manipular el robot se debe conectar el dispositivo de mando (*joystick* o *gamepad*). El efector final del robot seguirá los movimientos del dispositivo de mando en el sentido que este se lo indique.

A continuación se muestran los movimientos del órgano terminal del robot en las diferentes direcciones X, Y y Z. Es de destacar que el robot realiza el movimiento contrario ejercido por el *joystick* para que el órgano terminal alcance la posición deseada, es decir si el *joystick* se mueve hacia la derecha el robot se mueve hacia la izquierda para que el efector final se dirija hacia la derecha. De igual forma esta inversión se produce para todos los movimientos en los ejes X (Figura 6-1), (Figura 6-2), (Figura 6-3), y Y (Figura 6-4), (Figura 6-5). Mientras que en el eje Z (Figura 6-6), (Figura 6-7), el robot sigue el movimiento indicado por el *joystick*.

Movimientos en X:

Figura 6-1. El efector final y el *joystick* no realizan ningún movimiento.

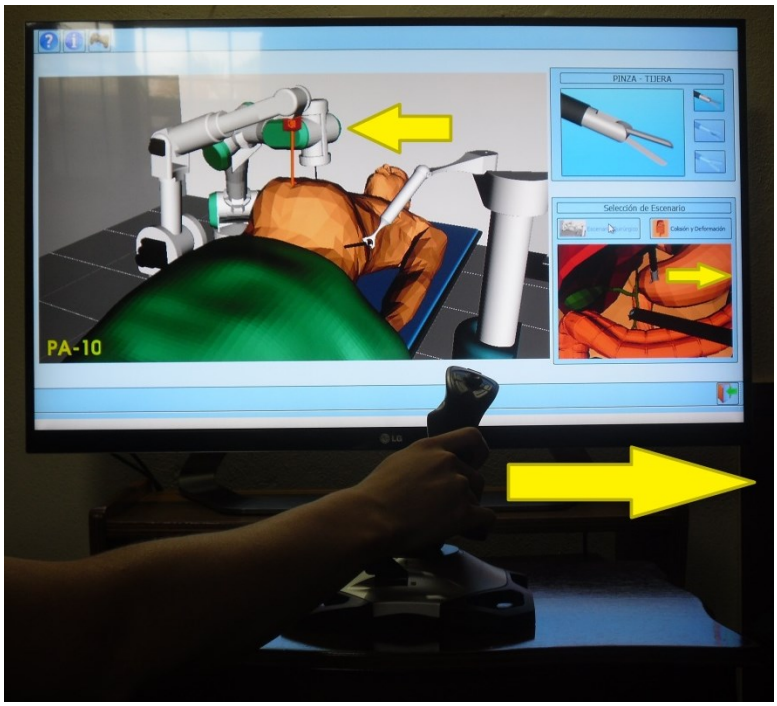

Figura 6-2. El efector final y el *joystick* se mueven hacia la derecha.

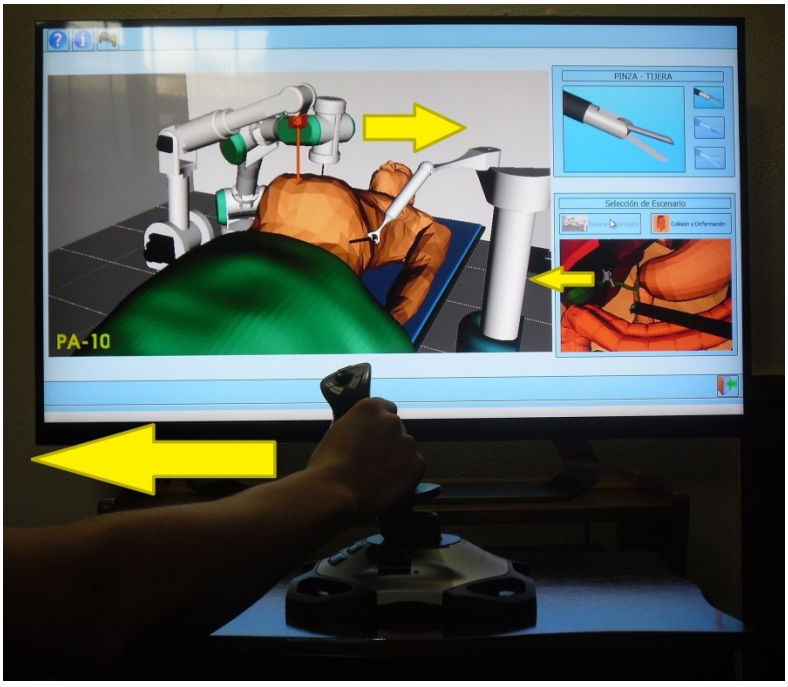

Figura 6-3. El efector final y el *joystick* se mueven hacia la izquierda.

Movimientos en Y:

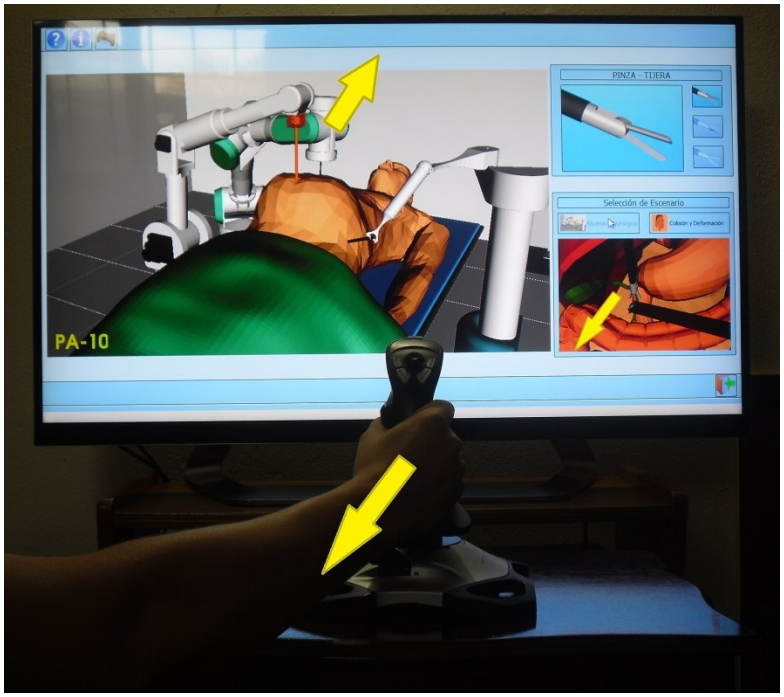

Figura 6-4. El efector final y el *joystick* se mueven hacia atrás.

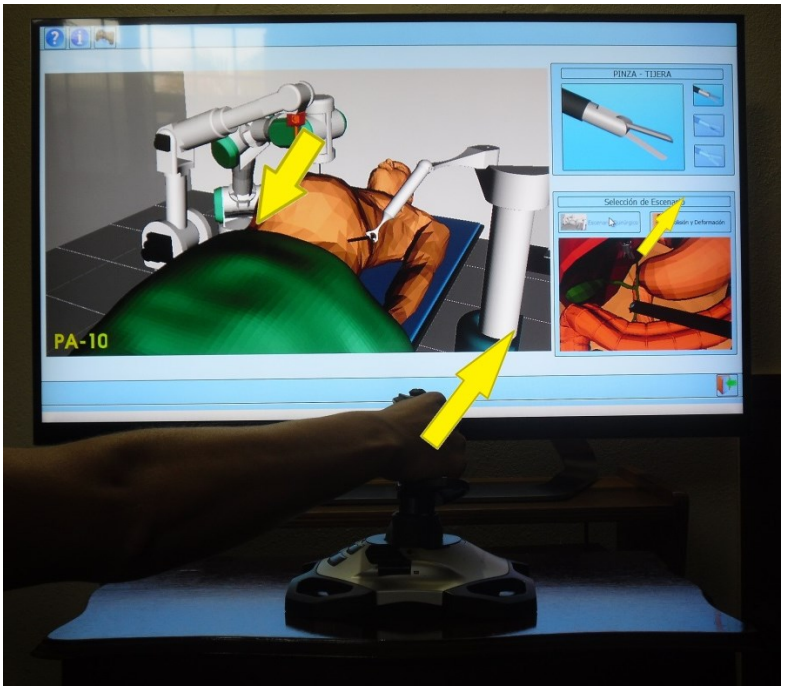

Figura 6-5. El efector final y el *joystick* se mueven hacia adelante.

Movimientos en Z:

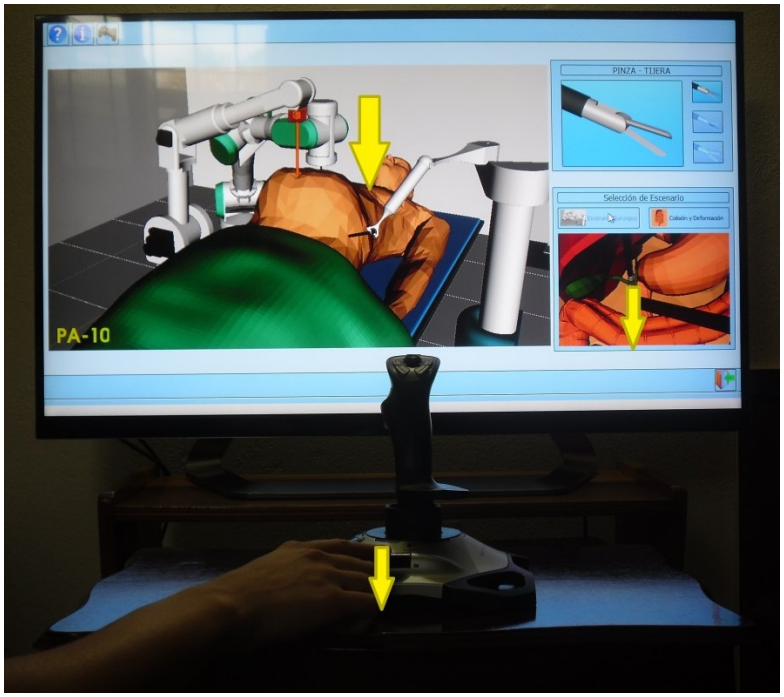

Figura 6-6. El efector final y el *joystick* se mueven hacia abajo.

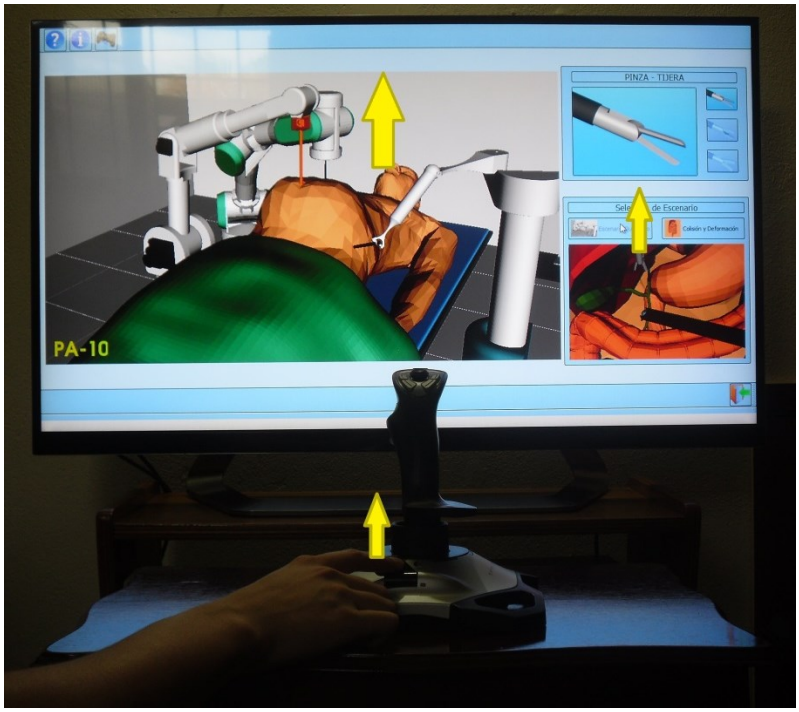

Figura 6-7. El efector final y el *joystick* se mueven arriba.

Con los movimientos anteriormente mostrados se puede ver el espacio de trabajo alcanzado por el robot, el cual permite interactuar con tres órganos:

hígado, estómago y vesícula. En esté, el robot puede posicionar dos endoclip sobre la cística[18] (procedimiento que se realiza en una colecistectomía), siempre y cuando se encuentre en el escenario de “Colisión y Deformación” y se seleccione el instrumento quirúrgico “Pistola Endoclip”.

6.2. Deformación de órganos.

Las pruebas realizadas para la etapa de deformación de órganos se realizan en el escenario de “Colisión y Deformación” y se muestran en las siguientes imágenes:

Primero se selecciona el instrumento quirúrgico “Tijera” y se hace colisionar con el hígado (Figura 6-8), el estómago (Figura 6-9) y la vesícula (Figura 6-10). Se observa que al mantener una presión sostenida sobre estos órganos, el órgano cede y se termina deformando. Para que el órgano retorne su forma inicial se aleja el efector final del órgano afectado.

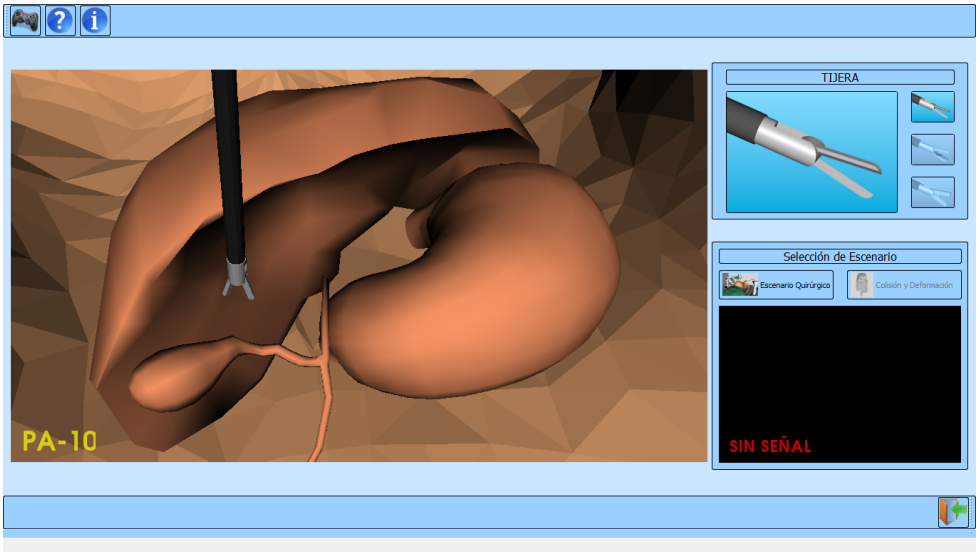

Figura 6-8. Deformación del higado con el instrumento quirúrgico "Tijera".

[18] Cística: Conducto que une la vesícula biliar con el punto de unión de los conductos colédoco y hepático.

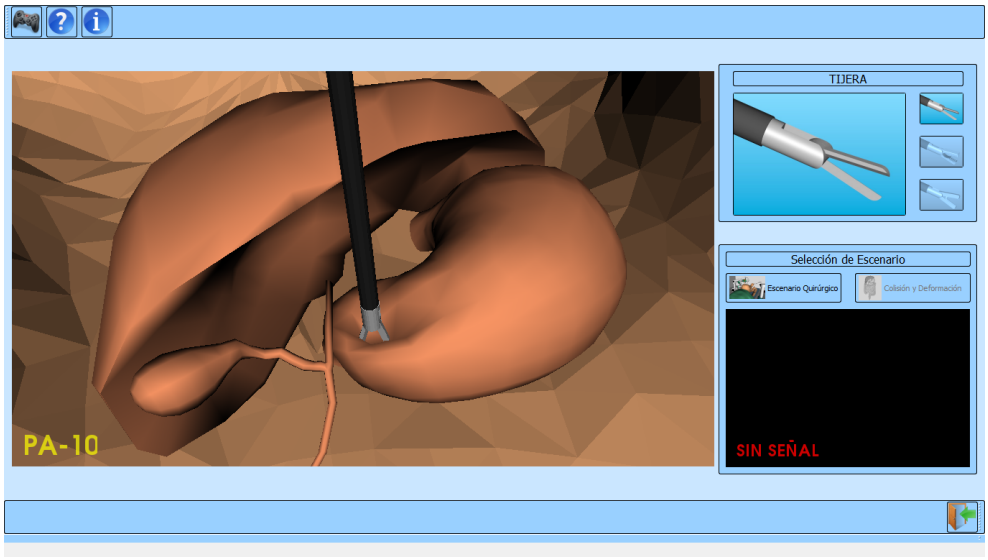

Figura 6-9. Deformación del estómago con el instrumento quirúrgico "Tijera".

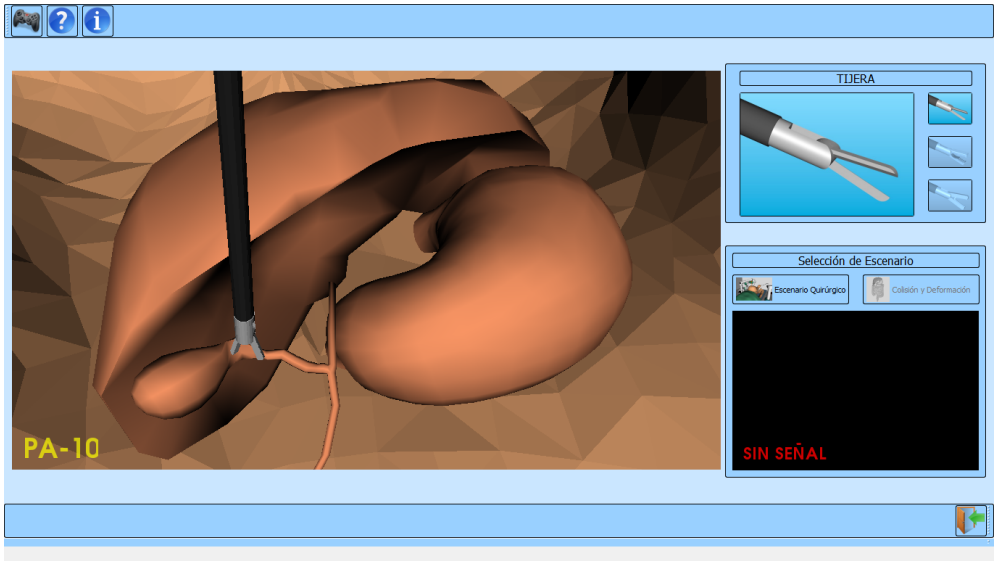

Figura 6-10. Deformación de la vesícula con el instrumento quirúrgico "Tijera".

Posteriormente se selecciona el instrumento quirúrgico “Retractor”, este cambio de instrumento se ejecuta al presionar un botón asignado en el *joystick* (ver Anexo C) y se lleva a cabo la prueba de deformación de órganos.

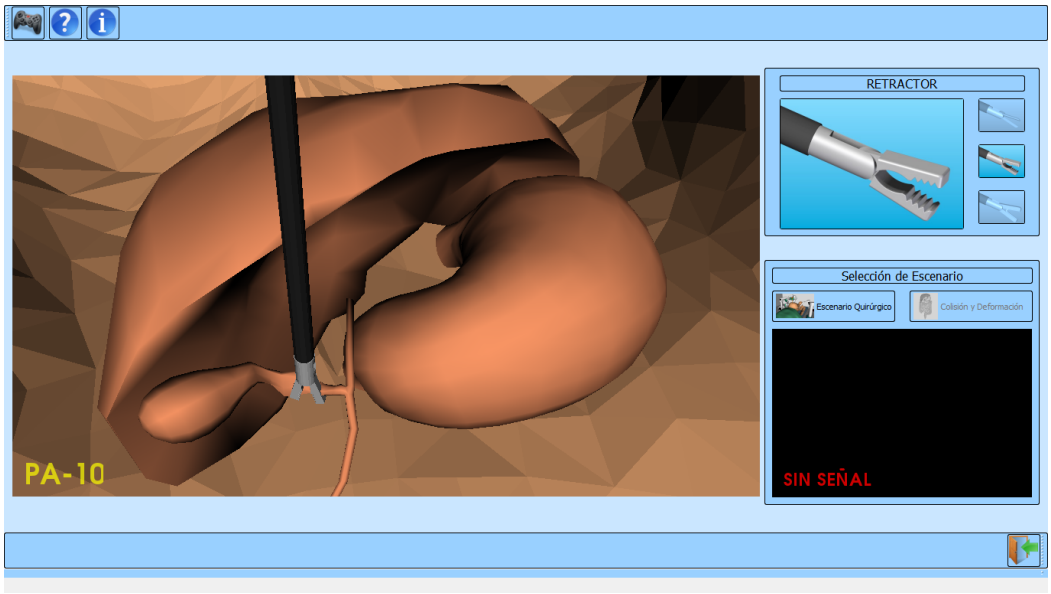

Figura 6-11. Deformación de la cística con el instrumento quirúrgico "Retractor".

La tercera prueba que se realiza es utilizando la “Pistola Endoclip”. Al seleccionar esta pinza se pueden realizar dos acciones. La primera acción es la de deformar los órganos como se hace con las otras pinzas (Figura 6-12). La segunda acción es la de colocar dos endoclips en la cística (Figura 6-14). Para colocar los endoclips, se debe llevar la pistola sobre la cística, cuando esta se encuentre en una posición cercana al conducto cístico, el efector final tendrá un giro de 90 grados (Figura 6-13), facilitando la ubicación de estos (ver Anexo C).

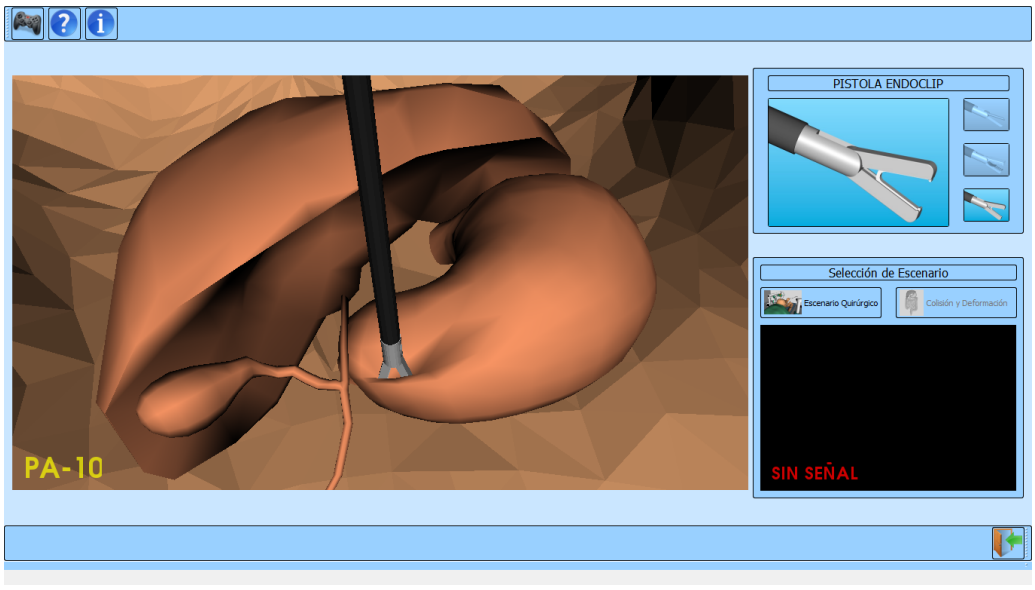

Figura 6-12. Deformación del estómago con el instrumento quirúrgico "Pistola Endoclip".

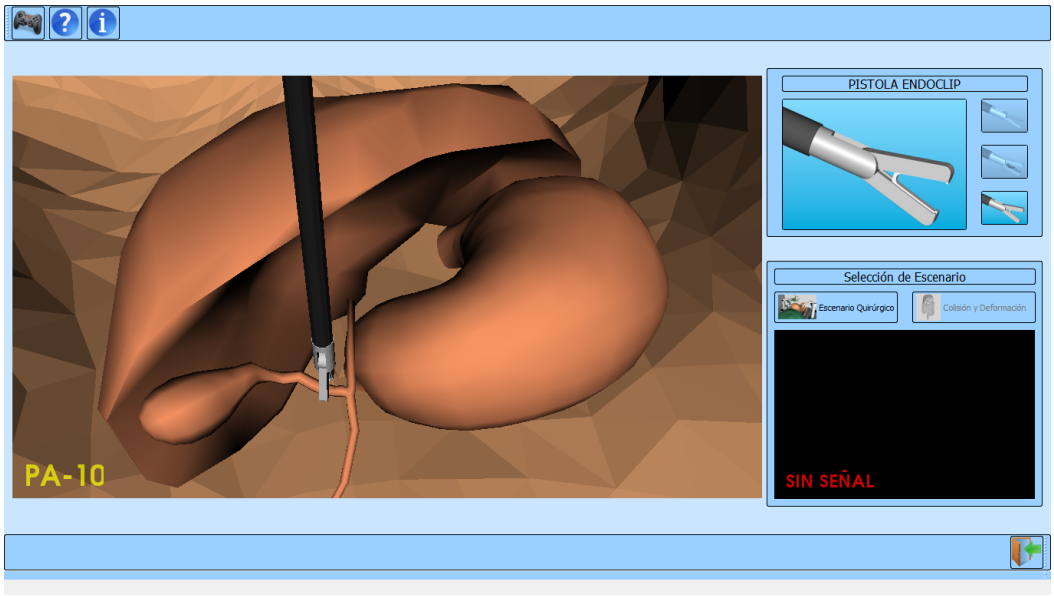

Figura 6-13. Ubicación de la "Pistola Endoclip" para colocar los endoclip.

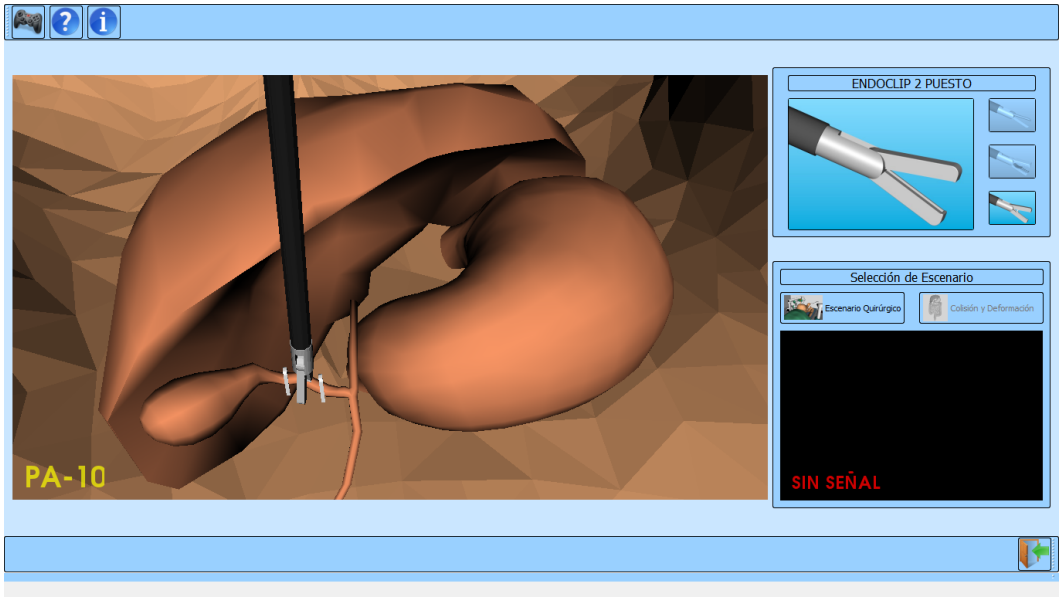

Figura 6-14. Endoclips ubicados.

7. Conclusiones y trabajos futuros.

7.1. Conclusiones.

En este trabajo de grado se diseñó el robot PA-10 y se implementó una solución software con el fin de verificar el comportamiento del robot en aplicaciones quirúrgicas. Para ello se analizó la estructura dinámica del robot PA-10, concluyendo que los siete grados de libertad que tiene el robot permiten posicionar sin mayor inconveniente el efector final dentro de la cavidad abdominal. Esto se logró mediante un algoritmo de optimización que restringe los movimientos del codo y de la muñeca, lo cual permite que el efector final se encuentre alineado con la muñeca en cada instante de tiempo (colinealidad), respetando el paso por el trocar.

Los métodos que se utilizaron para solucionar este problema de optimización, fueron los métodos de Levenberg Mardquart y de Nelder y Mead. De estos se puede concluir que el método de Levenberg Mardquart es un buen algoritmo de optimización siempre y cuando no se necesite obtener los datos en tiempo real, ya que hace al sistema muy lento debido a la dependencia de derivadas parciales para su funcionamiento. Por tal razón, en C++ se escogió el método de optimización de Nelder y Mead, técnica que no depende de las derivadas parciales del sistema de ecuaciones por lo que agiliza la solución del problema.

Después de verificar que el robot puede ser utilizado en aplicaciones quirúrgicas, se realizó el diseño del robot PA-10 con miras a su construcción real. Para esto se utilizó un software de diseño que permite la creación de piezas, que ensambladas forman la estructura completa del robot y esta a su vez, respetan las distancias establecidas previamente entre las articulaciones.

Posteriormente se implementó el diseño del robot en el software desarrollado para observar el comportamiento del robot PA-10 sobre un paciente virtual, verificándose una vez más el respeto del punto fijo en la cavidad abdominal. Además se comprobó el movimiento del robot bajo las órdenes de un dispositivo de mando externo (*joystick* o *gamepad*) conectado al puerto USB.

Por último se hizo uso de las librerías V-Collide y RAPID como herramienta de detección de colisiones para otorgarle una mayor realidad virtual al software y que el usuario pueda conocer con más detalle la interacción que tiene el robot PA-10 dentro de la cavidad abdominal. Del uso de estas librerías se concluye

que estas no ejercen demasiada carga computacional, algo que es muy importante cuando se desea tener velocidad de renderizado.

7.2. Trabajos futuros.

Por medio de la simulación desarrollada en el entorno MATLAB, se puede calcular los torques máximos requeridos para cada articulación, con esto se logra un dimensionamiento aproximado del par necesario para la escogencia de los actuadores en una futura construcción. Cuando la estructura del robot sea construida se debería implementar en el ambiente en que se encuentran inmersos los robots Hibou y Lapbot construidos en trabajos de grado anteriores.

En cuanto al software, se podría implementar el algoritmo de detección de colisiones y deformación al robot LapBot, para que exista en la aplicación dos robots que puedan interactuar con los órganos de la cavidad abdominal del paciente. También se podría desarrollar un sistema de deformación que permita realizar cortes en la superficie de los órganos, haciendo que la aplicación se acerque más a la realidad, permitiendo una práctica más realista y de la cual se puede sacar provecho en muchas áreas.

8. Referencias bibliográficas.

[1] V. Muñoz, “Control Cartesiano de un Asistente Robótico para Cirugía Laparoscópica”. Primer Work Shop Español de Robótica, Zaragoza, 2007.

[2] M. Marohn, E. Hanly. “Twenty-first century surgery using twenty-first century technology: surgical robotics”. Current Surgery, Vol. 61, pp. 466-473. 2004.

[3] “Intuitive Surgical. Da Vinci System”. URL: http://www.intuitivesurgical.com/index.aspx. Consultado: Diciembre 2013.

[4] G. Tan, R. Goel, J. Kaouk and A. Tewari, “Technological advances in Laparoscopic surgery”, Urologic Clinics of North America, Vol. 36, No. 2, pp. 237-250, 2009.

[5] C. Méndez, V. Torres, “Diseño y Simulación en 3D de un Robot Porta Endoscopio para Cirugía Laparoscópica”, Tesis de pregrado de Ingeniería en Automática Industrial, Universidad del Cauca, Colombia, 2010.

[6] S. Salinas, “Modelado, Simulación en 3D y control de un Robot para cirugía Laparoscópica”, Tesis de Maestría, Universidad del Cauca, Colombia, 2009.

[7] O. Mora, B. Garces, Estudio del desempeño de un robot Puma en operaciones de laparoscopia”, Tesis de pregrado de Ingeniería en Automática Industrial, Universidad del Cauca, Colombia, 2007.

[8] “Mitsubishi Heavy Industries”. URL: http://www.sdia.or.jp/mhikobe/products/mechatronic. Consultado: Febrero 2014.

[9] “LAP Mentor”. URL: http://simbionix.com/simulators/lap-mentor/. Consultado: Enero 2014.

[10] W. Khalil y D. Creusot, “Symoro+: A System for the Symbolic Modelling of Robots”, Robotica, Vol. 15, pp. 153-161, 1997.

[11] R. Gutiérrez, J. Lambás, E. Pascual y T. Vázquez, “Solid Edge v16. Guía de Referencia Diseño Gráfico”, Universidad Politécnica de Madrid, 2006.

[12] J. Samboni y C. Fuentes, “Base para sistema de entrenamiento quirúrgico: Robot HIBOU” Tesis de pregrado de Ingeniería en Automática Industrial, Universidad del Cauca, Colombia, 2013.

[13] “Modelado 3D de equipos de instrumentación electrónica disponibles en el Laboratorio de Instrumentación. Aplicaciones VRML de la interfaz virtual. Optimización de los modelos 3D”. URL: http://bibing.us.es/proyectos/abreproy/11281/fichero/1.+Memoria+PFC%252F7.+Capitulo4.pdf. Consultado: Junio 2014.

[14] “Simulink 3D Animation”. URL: http://www.mathworks.com/products/3d-animation/. Consultado: Junio 2014.

[15] VTK. URL: http://www.vtk.org. Consultado: Julio 2014.

[16] Microsoft Visual Studio. URL: msdn.microsoft.com/es-co/vstudio. Consultado: Julio 2014.

[17] QT. URL: http://qt-project.org/doc/qt-5/qtdesigner-manual.html. Consultado: Septiembre 2014.

[18] Librerias V-Collide y RAPID. URL: http://catarina.udlap.mx/u_dl_a/tales/documentos/msp/de_l_ea/apendiceB.pdf. Consultado: Septiembre 2014.

[19] M. Pinto, "Análisis e Implementación de una Interfaz Háptica en Entornos Virutales," Tesis de maestría, Universidad Nacional de Colombia, Bogotá, 2009.

[20] Librerías VTK. URL: http://www.vtk.org/doc/nightly/html/annotated. Consultado: Septiembre 2014.

[21] D. Guzmán, “Herramienta software para la práctica y experimentación de la robótica quirúrgica”, Tesis de pregrado de Ingeniería en Automática Industrial, Universidad del Cauca, Universidad del Cauca, Colombia, 2013.

[22] J. Nelder and R. Mead, “A simplex method for function minimation”, Computer Journal, Vol. 7, pp. 308-313, 1965.

[23] R. Gómez y F. Montero, “Visualización y deformación de objetos virtuales 3d”, Tesis de pregrado de Ingeniería en Automática Industrial, Universidad del Cauca, Colombia, 2012.

A. Anexo A: Instalación de herramientas software.

1. Instalación de Visual Studio 2008.

En las siguientes imágenes se mostrarán los pasos que se deben seguir para instalar esta herramienta. El tipo de instalación que se llevará a cabo es estándar.

Figura A-1. Instalación de Visual Studio 2008 (Paso 1).

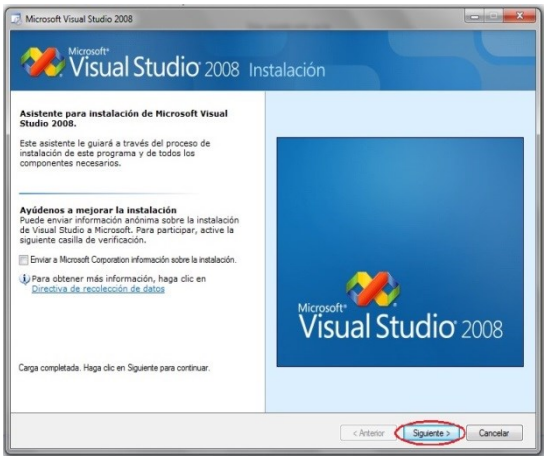

Figura A-2. Instalación de Visual Studio 2008 (Paso 2).

Figura A-3. Instalación de Visual Studio 2008 (Paso 3).

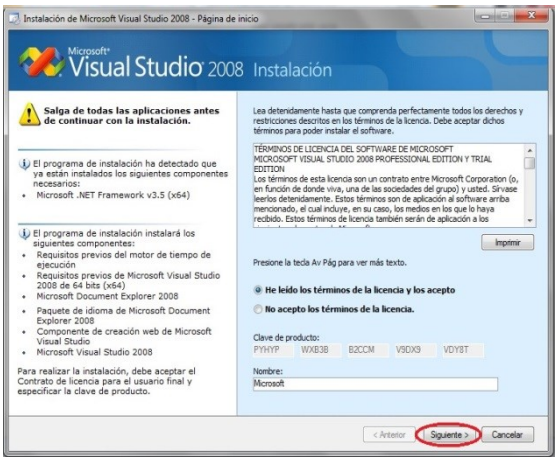

Figura A-4. Instalación de Visual Studio 2008 (Paso 4).

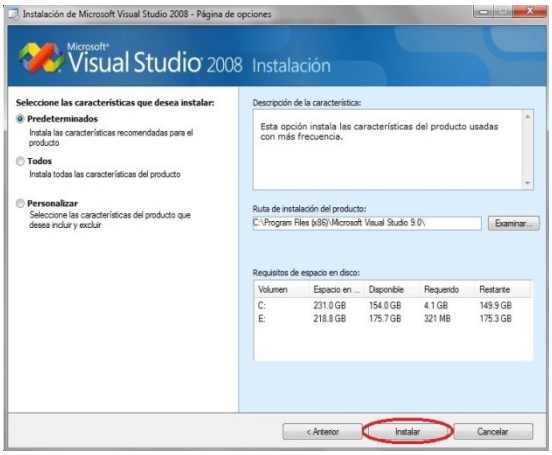

Figura A-5. Instalación de Visual Studio 2008 (Paso 5).

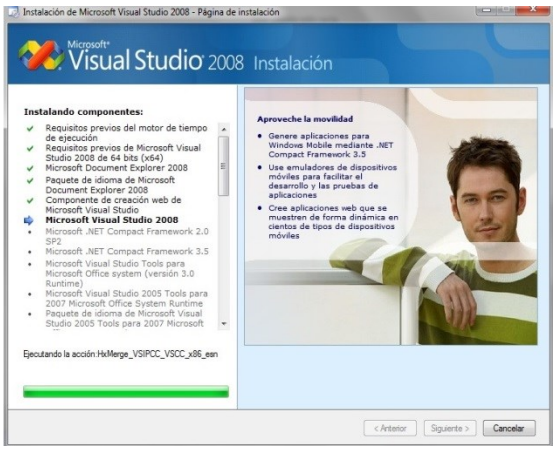

Figura A-6. Instalación de Visual Studio 2008 (Paso 6).

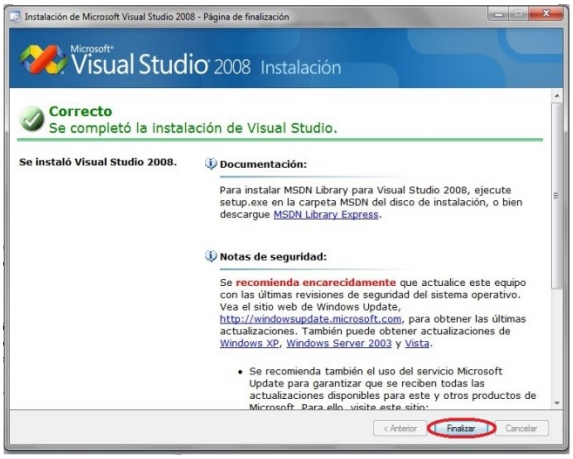

Figura A-7. Instalación de Visual Studio 2008 (Paso 7).

2. Instalación de QT

Esta herramienta se utilizó para el desarrollo de la interfaz gráfica SVRQ. QT Creator se puede descargar de la página http://qt-project.org/downloads. La instalación que se realiza es estándar. En las siguientes imágenes se muestra el proceso de instalación.

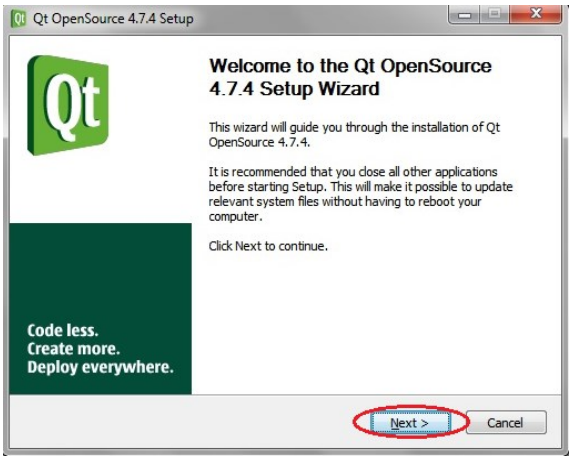

Figura A-8. Instalación de QT (Paso 1).

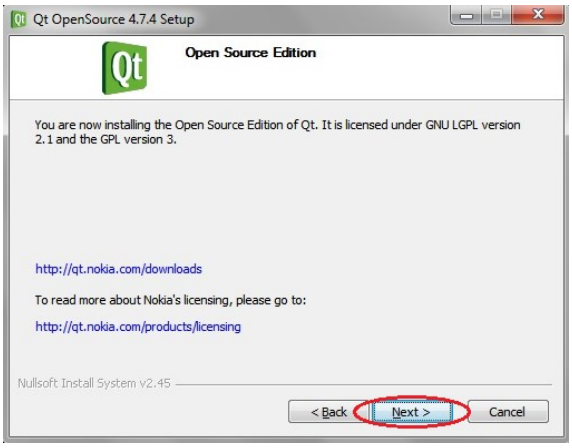

Figura A-9. Instalación de QT (Paso 2).

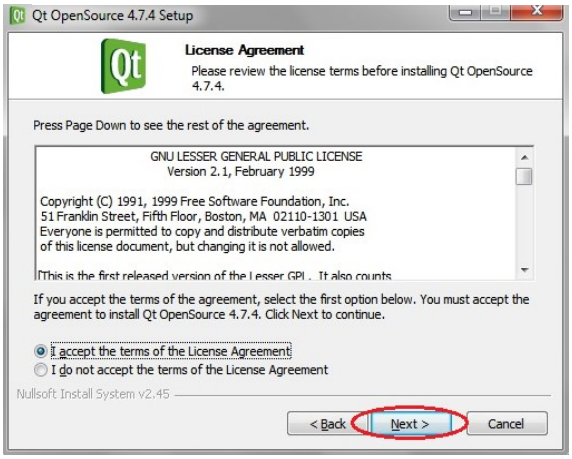

Figura A-10. Instalación de QT (Paso 3).

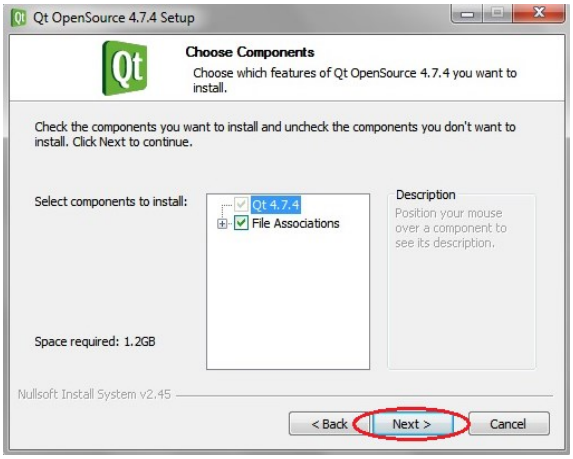

Figura A-11. Instalación de QT (Paso 4).

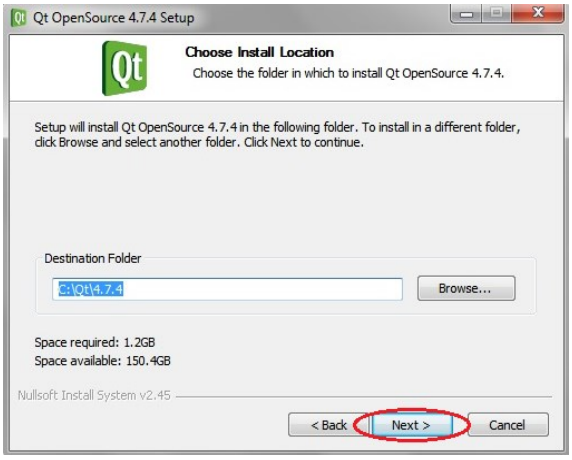

Figura A-12. Instalación de QT (Paso 5).

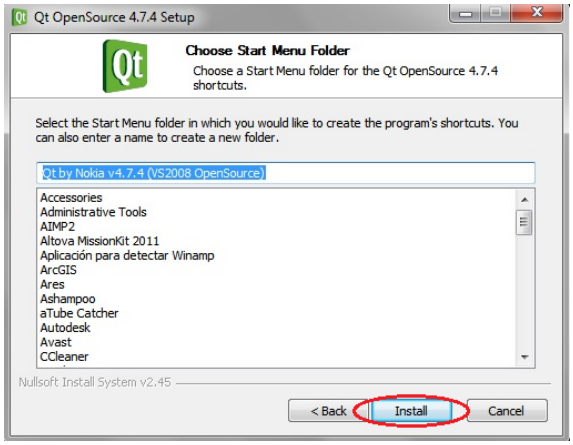

Figura A-13. Instalación de QT (Paso 6).

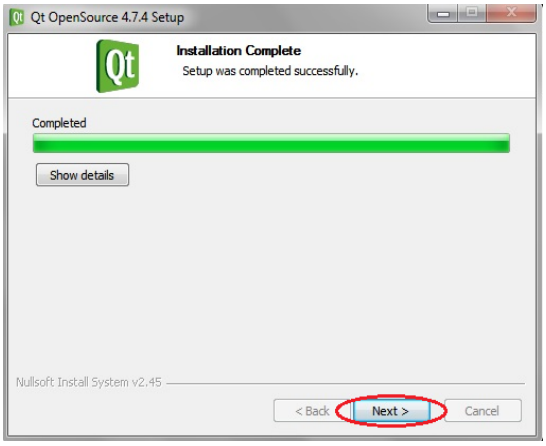

Figura A-14. Instalación de QT (Paso 7).

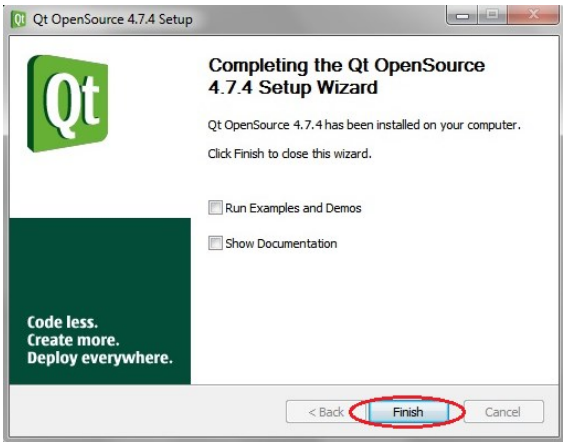

Figura A-15. Instalación de QT (Paso 8).

Al finalizar la instalación de qt-win-opensource-4.7.4-vs2008.exe se deben añadir las variables de entorno, para ello ir a: Equipo>>Propiedades del Sistema>>Configuración avanzada del sistema, y seguir los pasos indicados.

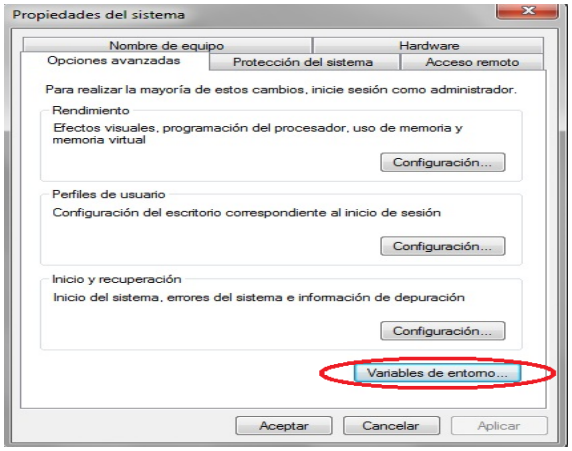

Figura A-16. Instalación de QT (Paso 9).

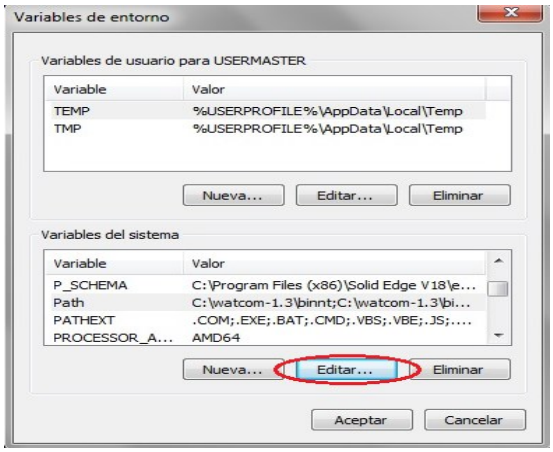

Figura A-17. Instalación de QT (Paso 10).

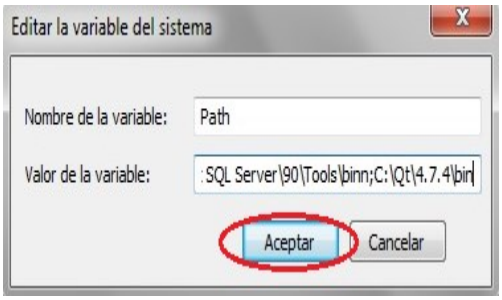

Figura A-18. Instalación de QT (Paso 11).

Una vez hecho esto se debe compilar QT para que funcione con Microsoft Visual Studio. Para esto, una vez se tiene instalado Visual Studio 2008, se abre el prom desde Visual Studio (click en Inicio>>Todos los programas>>Microsoft Visual Studio 2008>>Visual Studio Tools>>Símbolo del Sistema de Visual Studio 2008). Se abre una ventana en negro con la ubicación actual. Nos ubicamos en la carpeta donde se instaló QT (para ir atrás en una carpeta se puede dar “cd.”):

C:\Qt\4.7.4\

Escribir la siguiente configuración después de ubicarnos en la dirección anterior:

>configure –release –static –platform win32-msvc2008, (luego Enter).

Después de conocer los términos de la licencia, se pulsa la tecla “y” (yes), el sistema empezara a configurar la compilación, el proceso de configuración tardara unos minutos. Luego se compila el programa para Visual Studio, se debe escribir:

>nmake, (luego Enter).

Una vez compilado QT se procede a instalar QT Creator de la misma forma como se instaló Qt OpenSource hasta la figura A-15. Los pasos posteriores a esta figura no son necesarios. Después de realizar todo el proceso de instalación se verifica que en Visual Studio 2008 que añadido QT (Figura A-19).

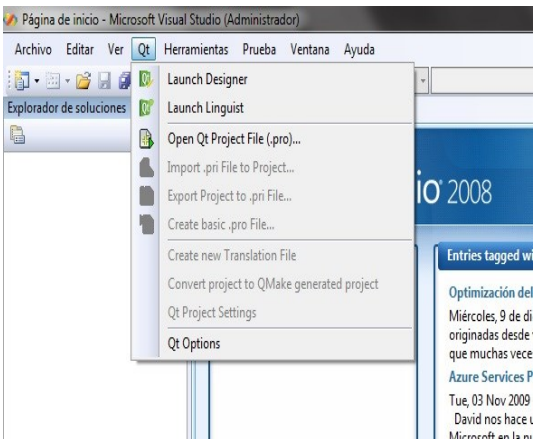

Figura A-19. Instalación de QT (Paso 12).

3. Instalación de CMake.

A continuación se mostrarán los pasos para instalar el software CMake 2.8 con las opciones predeterminadas. Este compilador se puede descargar de la página web http://www.cmake.org/cmake/resources/software.html.

Figura A-20. Instalación de CMake (Paso 1).

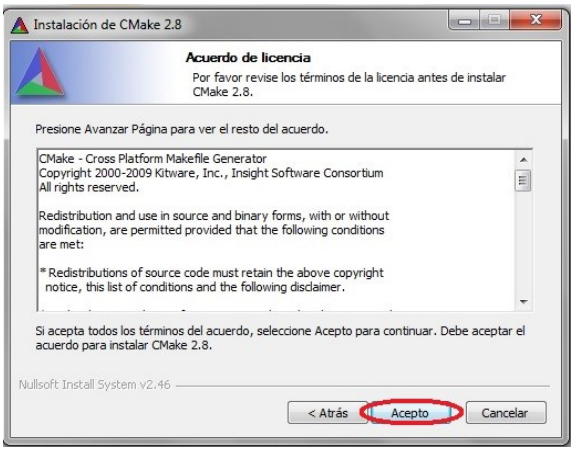

Figura A-21. Instalación de CMake (Paso 2).

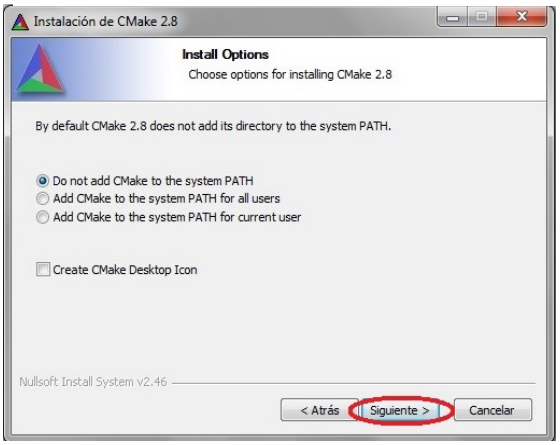

Figura A-22. Instalación de CMake (Paso 3).

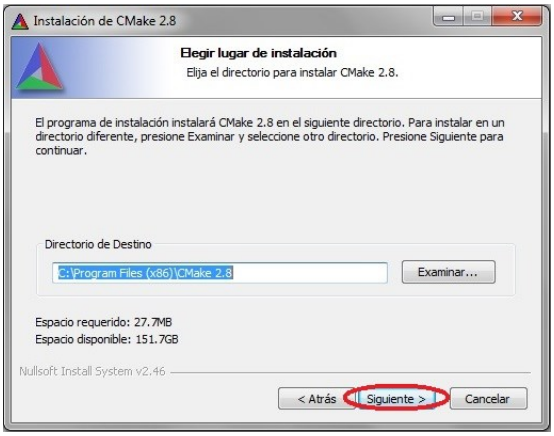

Figura A-23. Instalación de CMake (Paso 4).

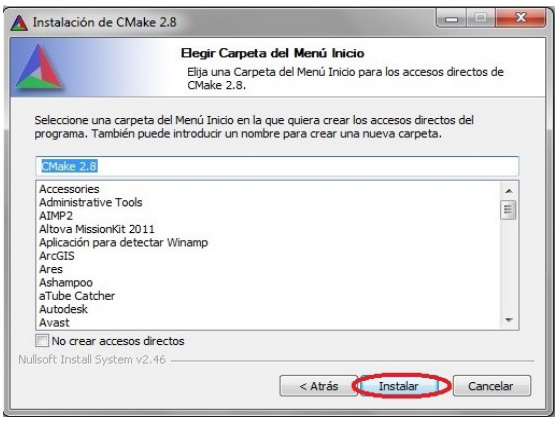

Figura A-24. Instalación de CMake (Paso 5).

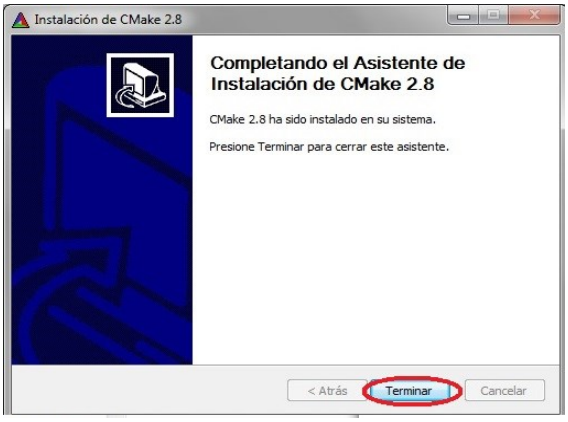

Figura A-25. Instalación de CMake (Paso 6).

4. Instalación de la librería VTK.

Para instalar la librería de VTK - 5.8.0 se deben crear tres carpetas en la raíz del C: VTK, VTK_Source y VTK_Build. Iniciar CMake y seguir los siguientes pasos:

En la opción Browse Source seleccionar C:/VTK_Source y en Browse Build seleccionar C:/VTK_ Build. Luego dar click en configure

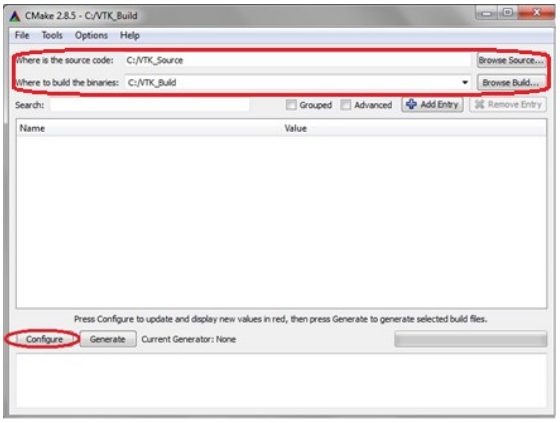

Figura A-26. Instalación de la librería VTK (Paso 1).

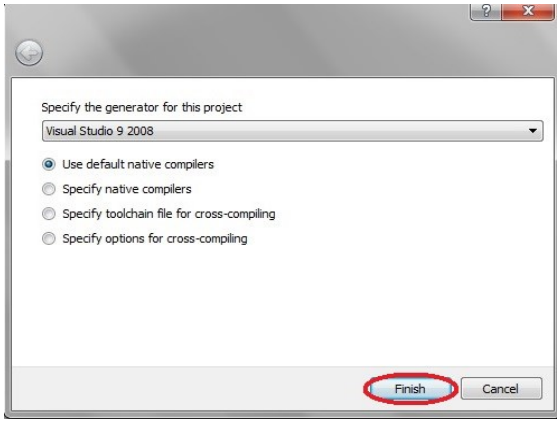

Figura A-27. Instalación de la librería VTK (Paso 2).

Luego en las opciones en rojo, cambiar el path de "C:/Program Files/VTK", al correspondiente a la ubicación de la carpeta "VTK".

Marcar las casillas de verificación.

- vtk_use_guisuport
- vtk_use_qt
- vtk_build_Examples

Luego presionar *configure*.

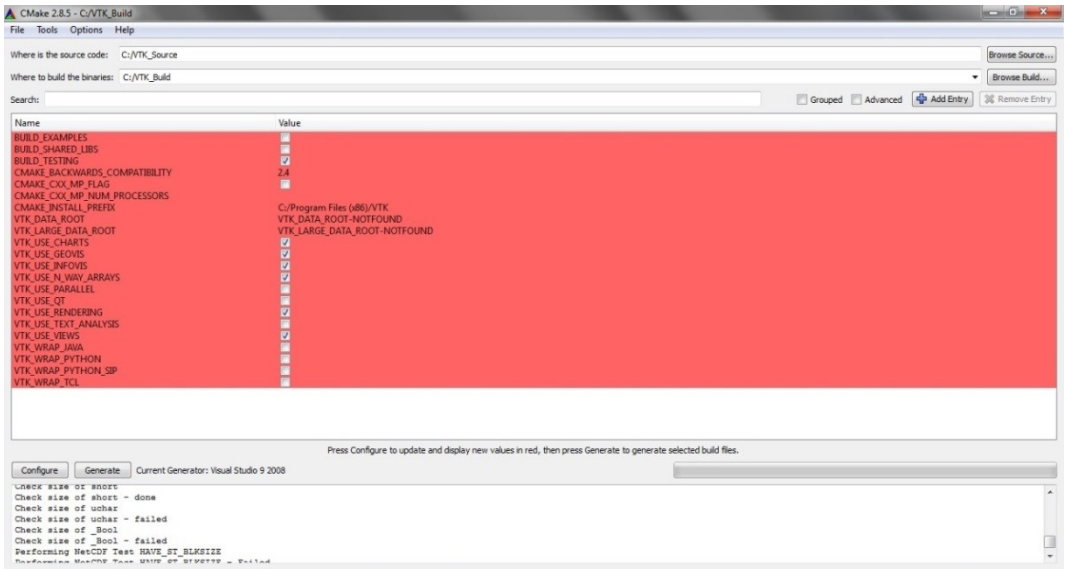

Figura A-28. Instalación de la librería VTK (Paso 3).

Posteriormente seleccionar:

- vtk_to_opengl

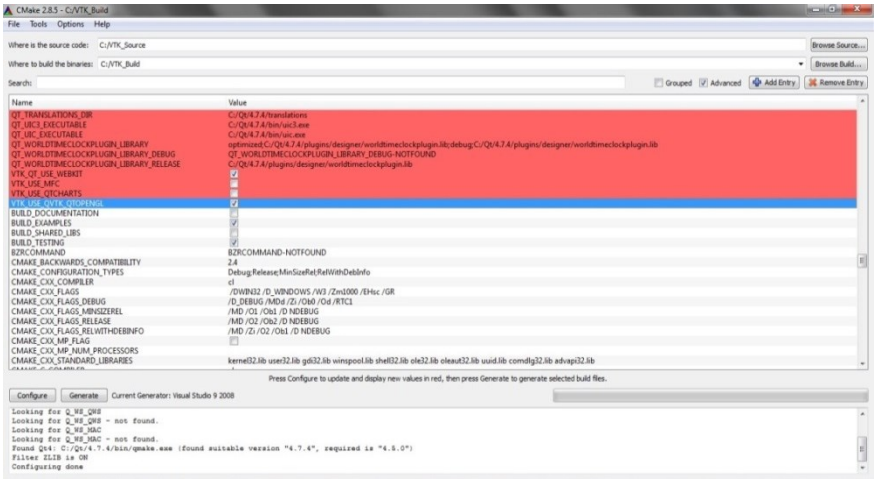

Figura A-29. Instalación de la librería VTK (Paso 4).

Luego presionar *configure*, después *generate* y a continuación cerrar CMake.

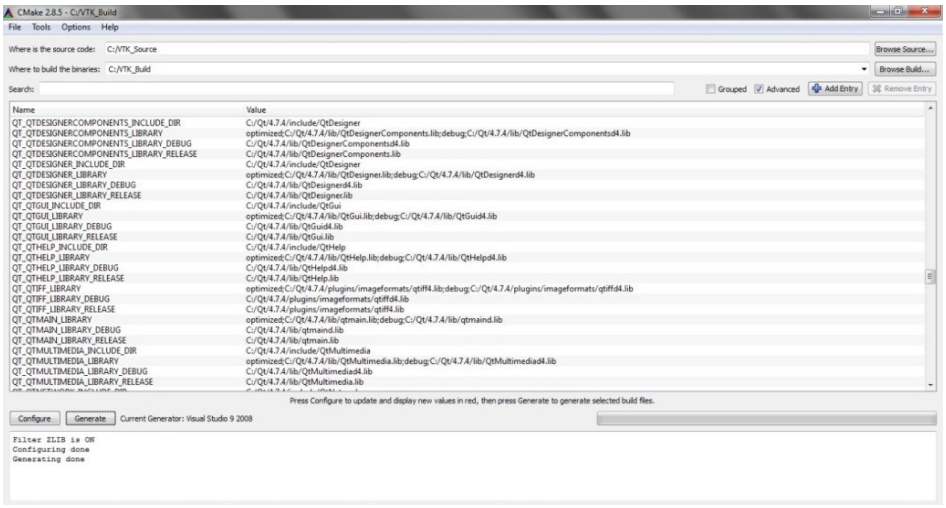

Figura A-30. Instalación de la librería VTK (Paso 5).

Buscar en la carpeta VTK_Build, el archivo VTK.sln (archivo de Visual Studio) y abrirlo.

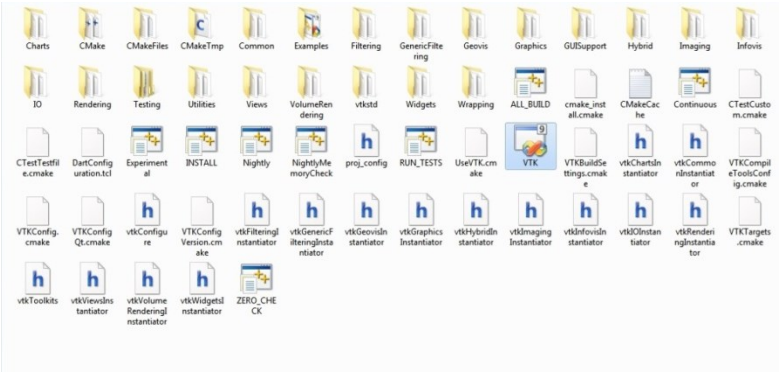

Figura A-31. Instalación de la librería VTK (Paso 6).

En la parte superior, en la barra de herramientas bajar, buscar "*debug*" y cambiarlo por "*release*".

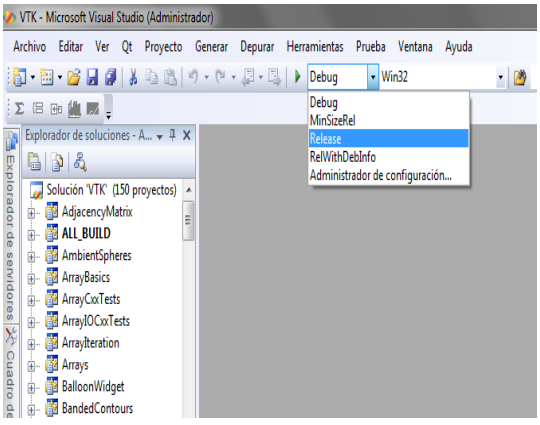

Figura A-32. Instalación de la librería VTK (Paso 7).

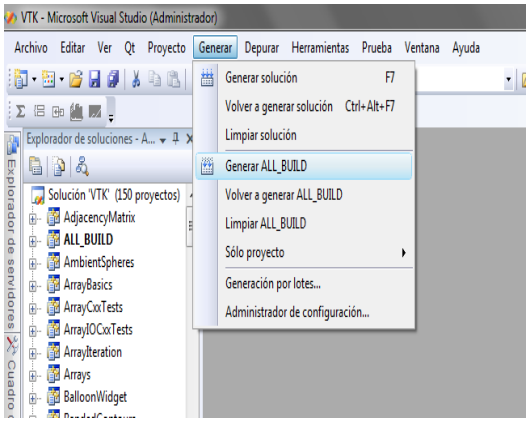

Figura A-33. Instalación de la librería VTK (Paso 8).

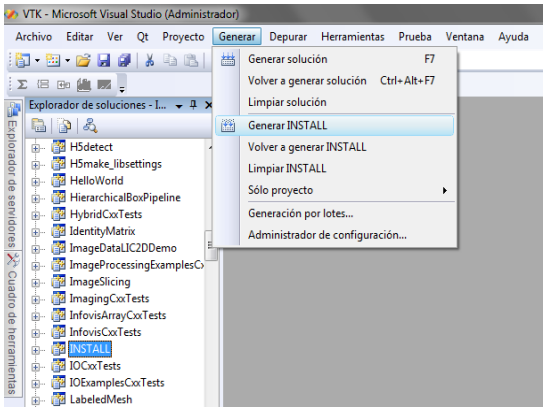

Figura A-34. Instalación de la librería VTK (Paso 9).

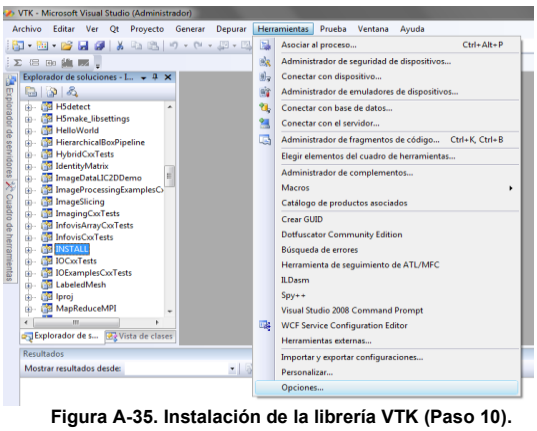

Figura A-35. Instalación de la librería VTK (Paso 10).

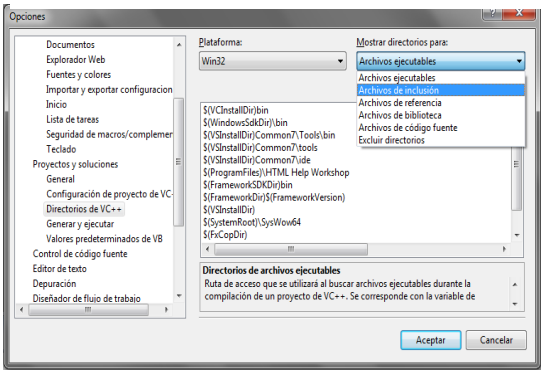

Figura A-36. Instalación de la librería VTK (Paso 11).

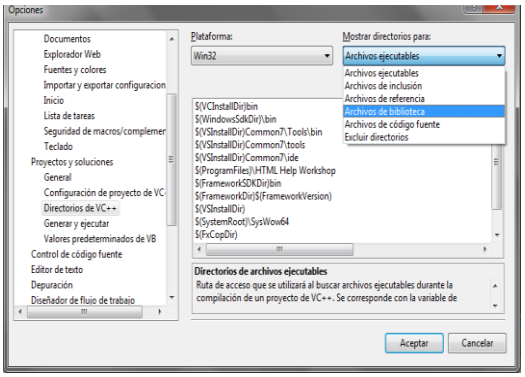

Figura A-37. Instalación de la librería VTK (Paso 12).

5. Instalación del SDL, V-Collide y RAPID.

Para instalar el SDL, V-Collide y Rapid se realiza el siguiente procedimiento.

Se abren las variables del sistema como indican las figuras A-16 y A-17 y se declaran las variables de entorno:

- SDL DIR C:\SDL\1.2.14
- VCOLLIDE DIR C:\VCollide201
- RAPID DIR C:\RAPID201

En el path (Figura A-18) se agrega

- ;C:\ SDL\1.2.14\lib
- ;C:\VCollide201\lib
- ;C:\RAPID201

Reiniciar el computador, pues Windows no reconoce las nuevas variables de entorno hasta que se reinicie el sistema. Luego abrir Visual Studio e ir a proyectos y soluciones, abrir los archivos de inclusión y agregar lo siguiente: C:\SDL\1.2.14\include, C:\VCollide201\include, C:\RAPID201\include.

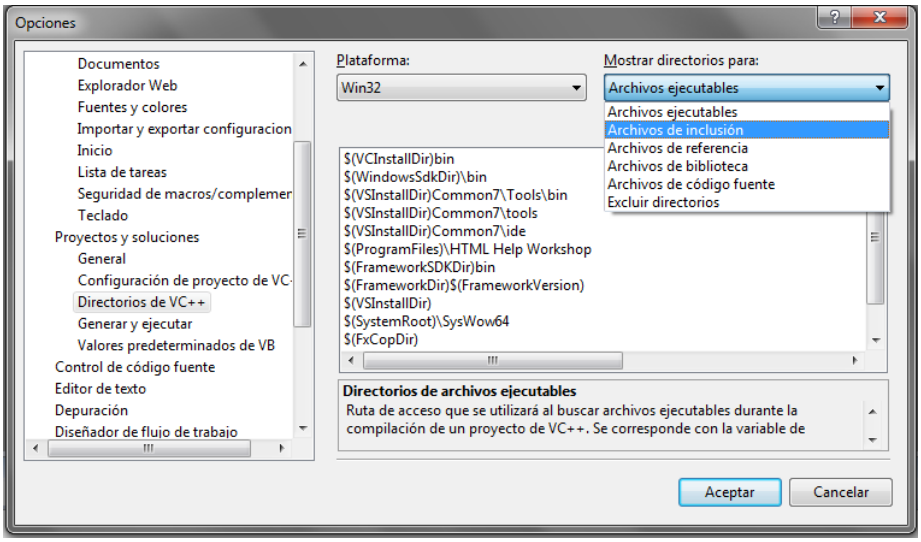

Figura A-38. Instalación de la librería V-Collide y RAPID (Paso 4).

Luego adicionar en archivos de biblioteca: C:\SDL\1.2.14\lib C:\VCollide201\lib C:\RAPID.

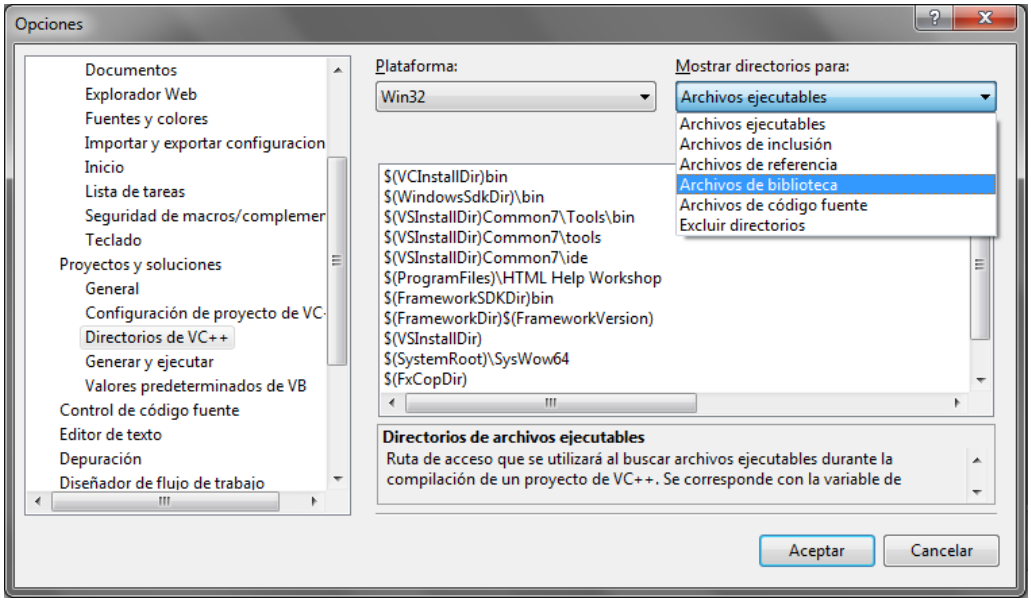

Figura A-39. Instalación de V-Collide y RAPID (Paso 5)

Luego ir a propiedades de proyecto>>Vinculador>>Entrada>>Dependencias adicionales e incluir las librerías VCollide.lib y RAPID.lib.

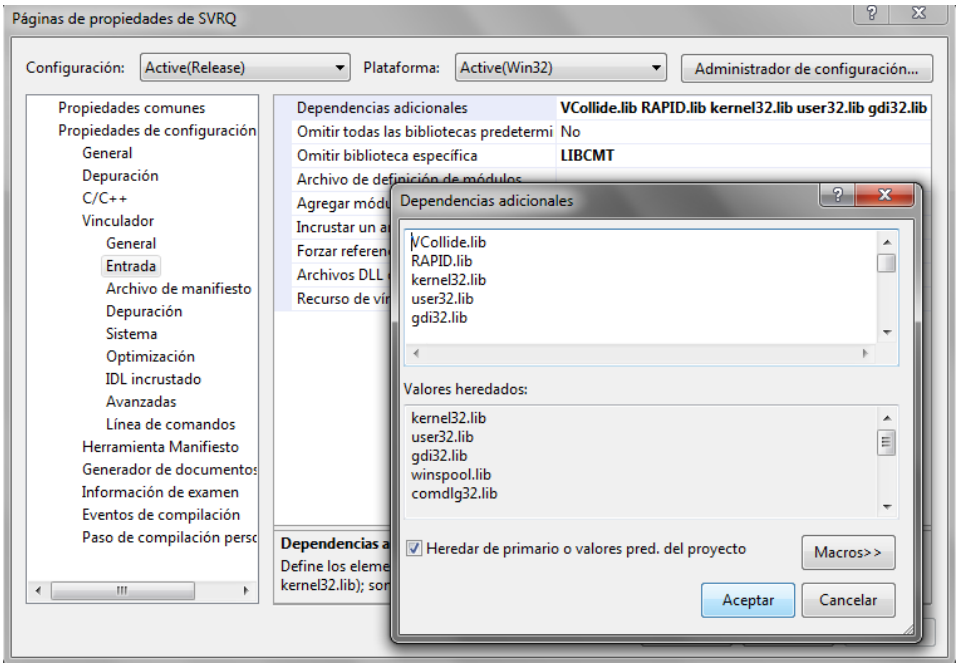

Figura A-40. Instalación de V-Collide y RAPID (Paso 6).

B. Anexo B: Uso de Solid Edge, Blender y VTK para el ensamblado del robot.

Para la construcción del robot se debe diseñar las piezas del robot en el entorno pieza de Solid Edge. Después de diseñar las piezas y elementos que se van a necesitar en la interfaz estas se guardan en formato *.wrl, como lo indican las figuras B-1 y B-2. Las piezas que se guardan en este formato se pueden importar en el entorno Blender y en *Virtual Reality* de MATLAB – Simulink.

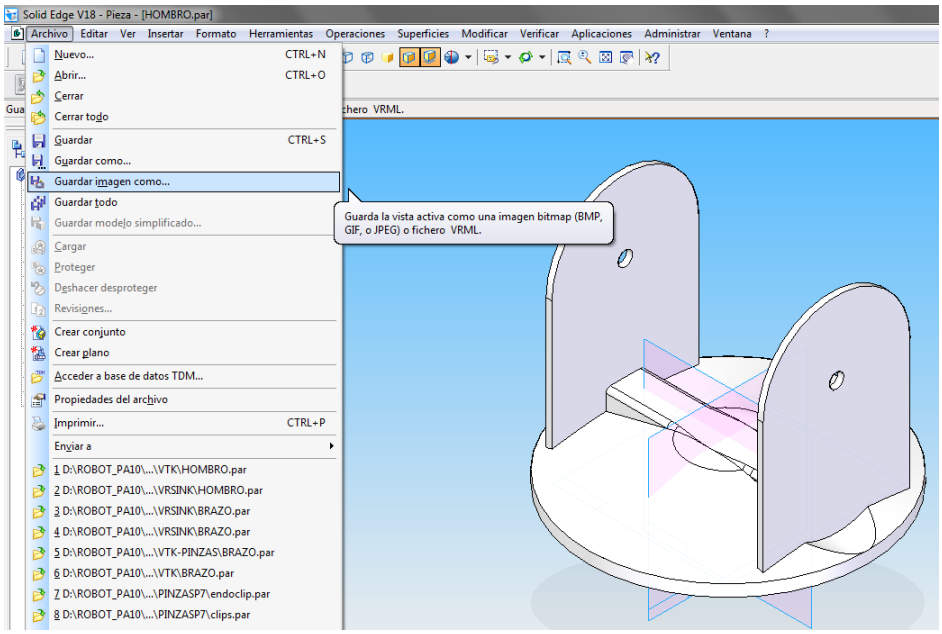

Figura B-1. Guardar pieza en formato *.wrl (Paso 1).

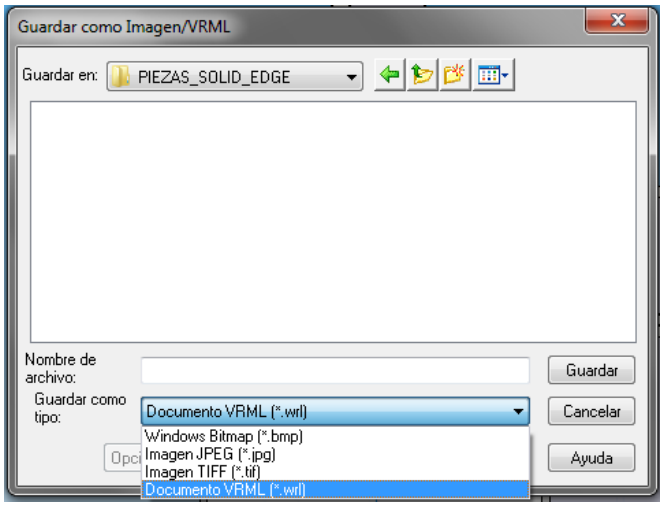

Figura B-2. Guardar pieza en formato *.wrl (Paso 2).

Luego de guardar todas las figuras en el formato *.wrl, se procede a importar las figuras en el entorno Blender con el fin de cambiar el formato a *.obj que es utilizado para cargar los objetos en la aplicación por medio de la librería vtkOBJReader.

La forma como se importan los objetos en Blender se muestran en las figuras B-3 y B-4.

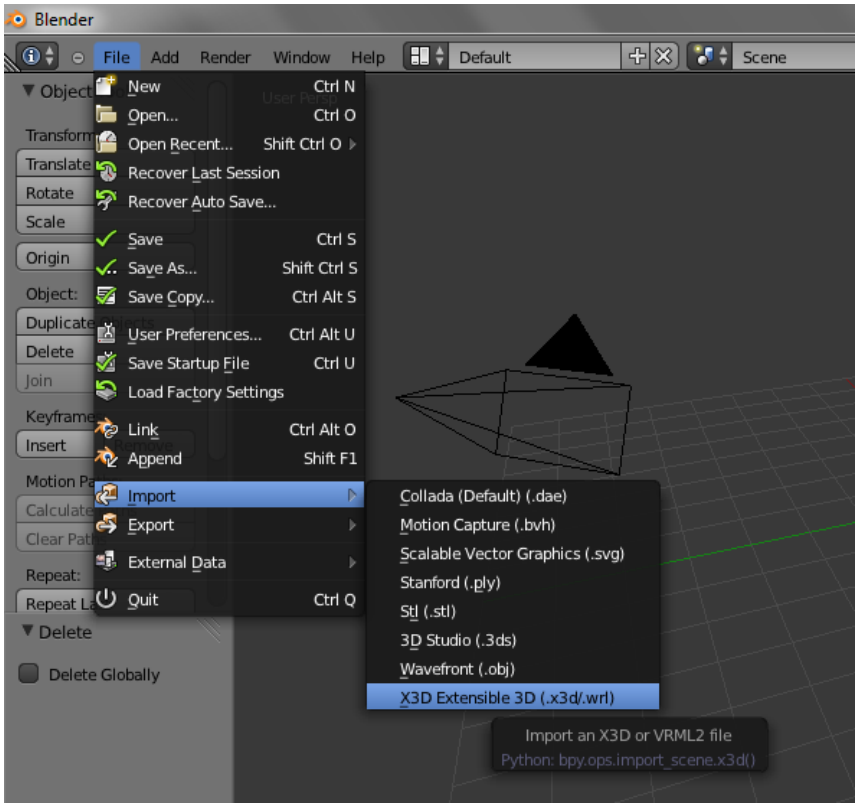

Figura B-3. Importando objetos en Blender (Paso 1).

Figura B-4. Importando objetos en Blender (Paso 2).

Luego de importar las piezas estas son cargadas en varios componentes por lo que es necesario unir estos en un solo objeto. Para ello se selecciona todos los componentes y se presiona Ctrl + J. La pieza unida se muestra en la figura B-5.

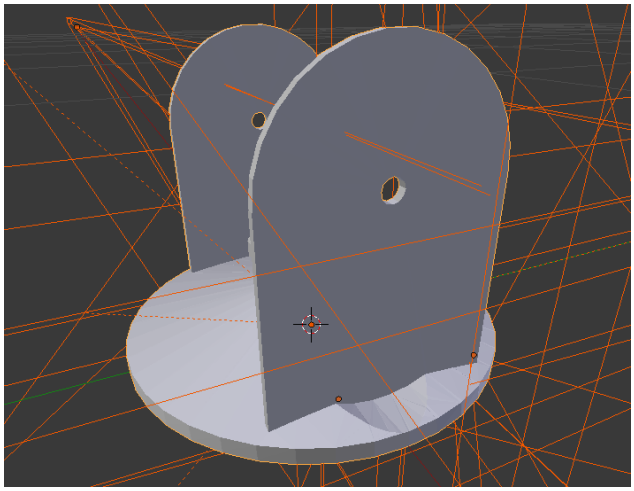

Figura B-5. Pieza unida en Blender.

Después de unir el objeto se procede a exportarlo en el formato *.obj como se muestra en la figura B-6.

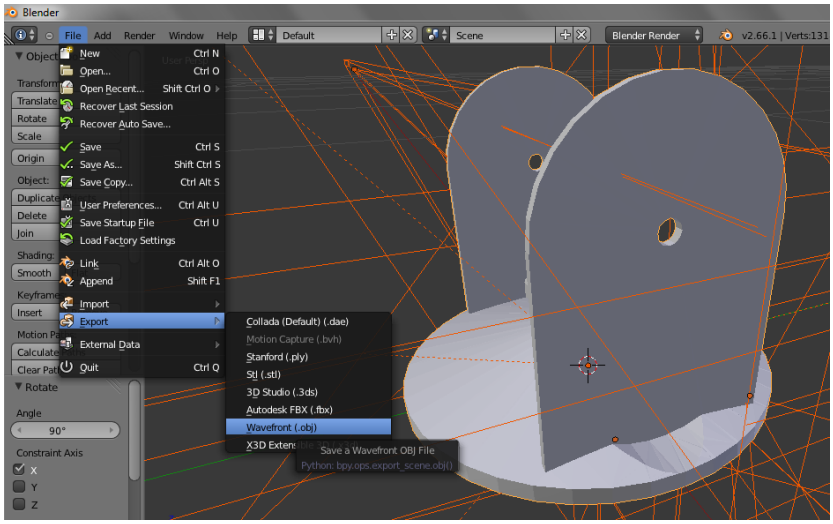

Figura B-6. Exportar objeto en Blender.

Blender maneja una seria de comandos que facilitan la ubicación de los objetos (Tabla B-1). Hay que destacar que para acceder al comando hay que presionar la letra y en seguida la letra del eje (X, Y y Z) en que se desea realizar el cambio, posterior se escribe el número que se desea desplazar o rotar el objeto y se presiona *Enter* (Figura B-7).

Comando	Función
R	Rotar los objetos
G	Mover los objetos
S	Cambiar el tamaño de los objetos
N	Propiedades del panel de transformación

Tabla B-1. Comandos de Blender.

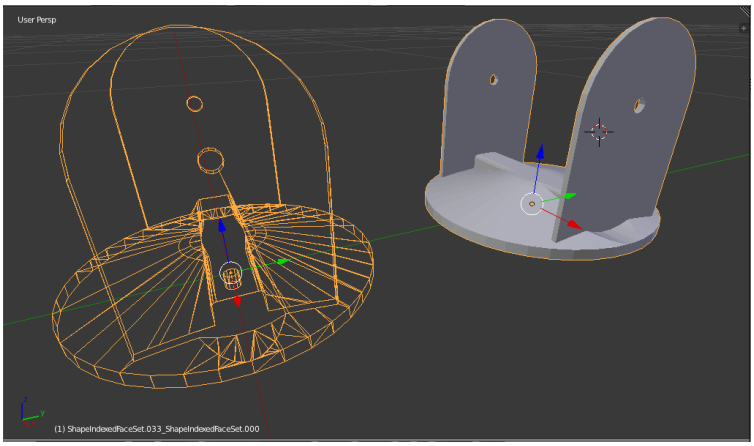

Figura B-7. Desplazamiento del objeto mediante el comando G.

Hay dos formas de ensamblar el robot en VTK, la primera de ella es ensamblar el robot en Blender como lo muestra la figura B-8 y después en Visual Studio con el comando AddPosition (X, Y, Z) de la clase vtkActor se les asigna la posición (0, 0, 0) y las piezas se van cargando según su posición y rotación en Blender.

La otra forma de ensamblar el robot es un poco más compleja, primero se debe guardar las piezas en el entorno Blender con el mismo origen y luego empezar a variar los valores en X, Y y Z en el comando AddPosition de la clase vtkActor hasta lograr un ensamble completo del robot.

Por tal razón se recomienda ensamblar el robot en Blender como se muestra en la figura B-8, luego guardar pieza por pieza en el formato *.obj, por ejemplo se tienen las piezas ensambladas de la figura B-8 y se desea guardar la pieza brazo, para ello se debe suprimir las otras piezas como se muestra en la figura B-9. Esta pieza unitaria se debe guardar en el formato *.obj siguiendo los pasos explicados anteriormente. Después de guardar la pieza, utilizando el

comando Ctrl + Z se retorna a la estructura que se tiene en la figura B-8. Para guardar las demás piezas se debe seguir el mismo procedimiento.

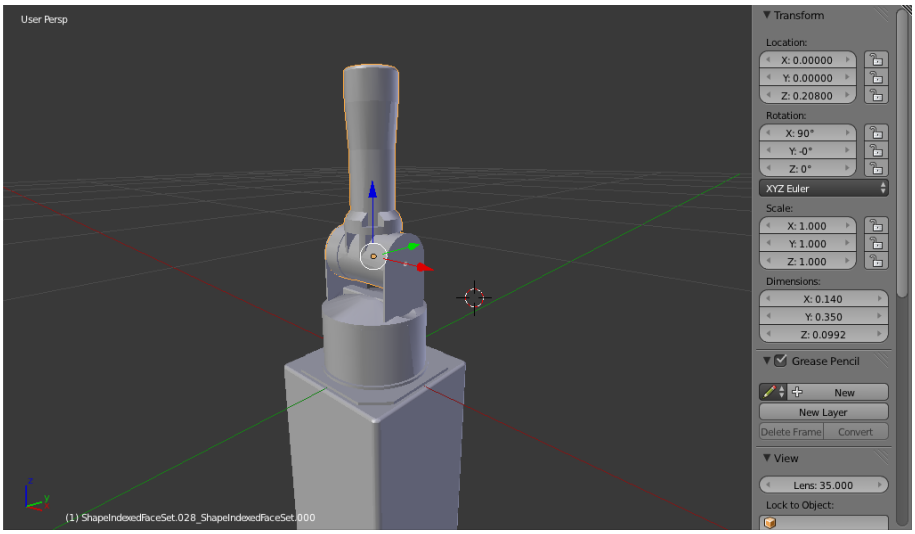

Figura B-8. Ensamblando robot en Blender.

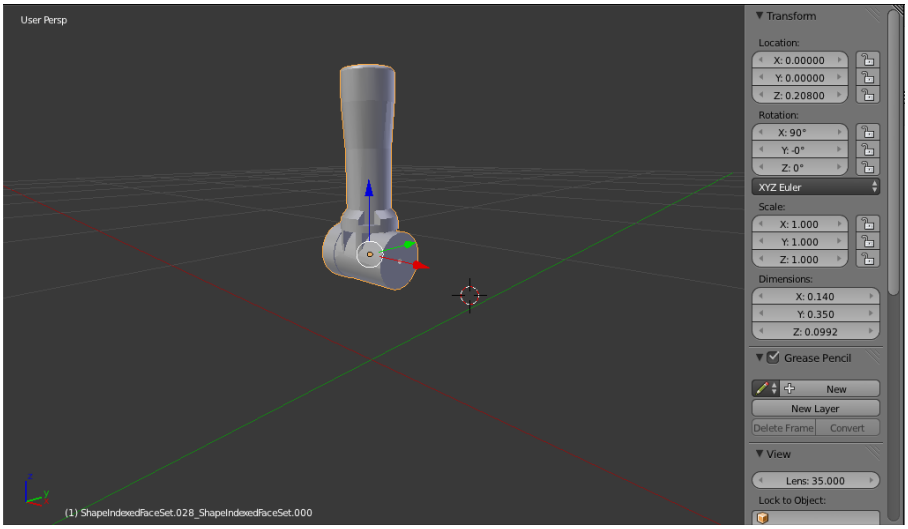

Figura B-9. Pieza brazo.

Después de leer los objetos en Visual Studio por medio del comando vtkOBJReader y dándole la posición a cada uno por medio del comando AddPosition de la clase vtkActor se debe crear una unión padre – hijo entre las piezas para que un movimiento de una afecte a la otra. Para ello se hace uso de la clase vtkAssembly la cual permite a una unión de piezas otorgarle un movimiento de una articulación. A continuación se muestra el código que permite ensamblar las piezas mostradas anteriormente.

```
articulacionesR[1] = vtkAssembly::New();//BRAZO
articulacionesR[1]->AddPart(piezasRobot[2]->objActor);
```

```
articulacionesR[1]->AddPart(articulacionesR[2]);
articulacionesR[1]->SetOrigin(0,0.208,0);

articulacionesR[0] = vtkAssembly::New();//HOMBRO
articulacionesR[0]->AddPart(piezasRobot[1]->objActor);
articulacionesR[0]->AddPart(piezasRobot[9]->objActor);
articulacionesR[0]->AddPart(piezasRobot[10]->objActor);
articulacionesR[0]->AddPart(articulacionesR[1]);

//variable del tipo vtkassembly que retorna todas los objetos de la
escena
robotEnsamblado = vtkAssembly::New();//BASE
robotEnsamblado->AddPart(articulacionesR[0]);
robotEnsamblado->AddPart(piezasRobot[0]->objActor);
robotEnsamblado->AddPart(piezasRobot[8]->objActor);
 robotEnsamblado->AddPosition(-0.075,-0.23,-0.45);
```

El objeto robotEnsamblado, contiene la estructura del robot que se observa en la figura B-10. La combinación de las funciones “vtkAssembly::New”, “AddPart”, “AddPosition”, y el vector “articulacionesR[n]” permiten simular el comportamiento de las articulaciones del robot, ya que convierten cada articulación en la referencia necesaria para que el objeto haga el movimiento en el espacio.

Cuando se está ensamblando el robot en Visual Studio se debe remover mientras tanto el método Mover_Robot, ya que este método proporciona rotaciones sobre las articulaciones que impiden ensamblar adecuadamente el robot (Figura B-10).

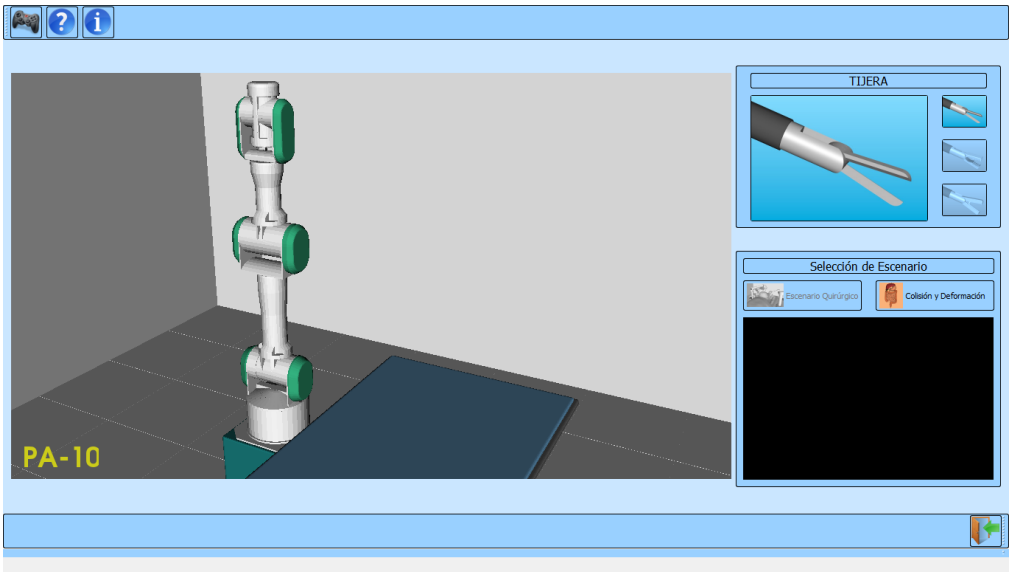

Figura B-10. Robot ensamblado.

Después de tener el robot ensamblado se activa el método Mover_Robot, el cual entrega los parámetros necesarios para posicionar las articulaciones con el fin de alcanzar la posición deseada (Figura B-11).

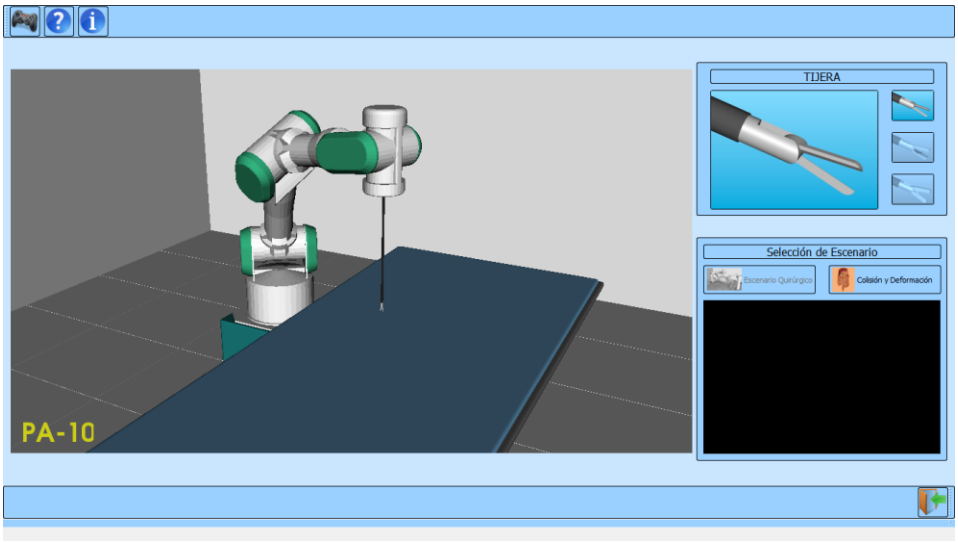

Figura B-11. Robot ensamblado y método Mover_Robot activo.

C. Anexo C: Manual de Usuario.

Para instalar la aplicación se debe abrir el setup de SVRQ que se encuentra en la carpeta Instalador. Una vez abierto el setup se debe seguir los pasos que muestre el asistente de instalación. Es importante seleccionar la carpeta donde se va a guardar los archivos de la aplicación, para ello se debe crear una carpeta con el nombre “SVRQ” en una unidad del disco duro.

C:\SVRQ\

Después de realizar el anterior procedimiento se debe dar click en siguiente y seguir los pasos mostrados por el asistente de instalación. Al terminar el proceso de instalación se puede abrir la aplicación desde escritorio mediante el acceso directo. Al inicializar la aplicación se abre la siguiente ventana de comandos (Figura C-1).

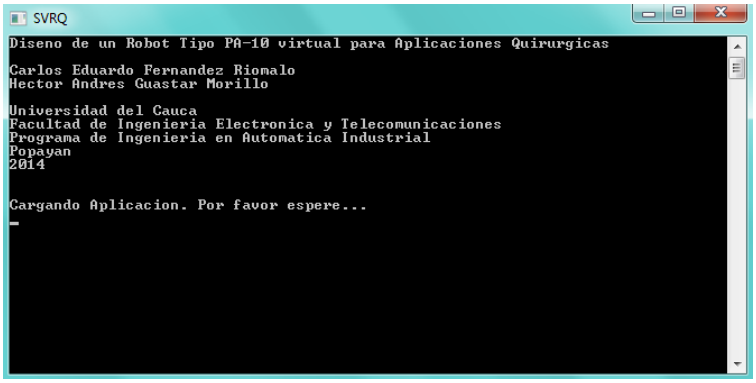

Figura C-1. Inicio de la aplicación.

Si no se tiene un dispositivo de mando conectado como un *joystick* o *gamepad*, la aplicación desplegará un cuadro de mensaje impidiendo acceder a la interfaz principal (Figura C-2).

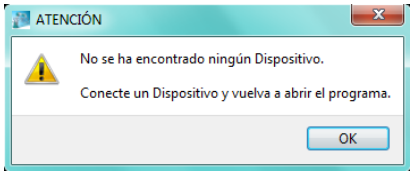

Figura C-2. Mensaje de atención.

Una vez conectado el dispositivo de mando, se puede abrir el programa para acceder a la pantalla principal del software implementado, la cual se ilustra a continuación.

La figura C-3 corresponde a la interfaz gráfica de usuario de la aplicación SVRQ y la figura C-4 a la barra de herramientas de la aplicación.

Figura C-3. Interfaz gráfica de SVRQ.

Figura C-4. Barra de herramientas.

La barra de herramientas presenta tres opciones: “Dispositivo”, “Ayuda”, y “Acerca de”. “Dispositivos” permite habilitar el joystick para la manipulación de los robots (Figura C-5). “Ayuda” presenta una guía de usuario donde se puede encontrar respuesta a las posibles dudas sobre el manejo de la herramienta (Figura C-6) y finalmente la opción “Acerca De” la cual muestra información sobre el desarrollador del trabajo realizado (Figura C-7).

Figura C-5. Opción Dispositivo.

Figura C-6. Opción Ayuda.

Figura C-7. Opción Acerca De.

Al dar clic sobre el icono de “Dispositivo”, se abre la ventana de configuración de dispositivo, donde se puede ver la distribución de los botones y la palanca de control principal del *joystick* o *gamepad.* Una vez realizado este proceso se habilitan los botones para acceder a los dos escenarios de la aplicación, los cuales son “Escenario quirúrgico” y “Colisión Deformación” (Figura C-8).

Figura C-8. Selección de Escenario.

En el primer escenario (“Escenario Quirúrgico”) existen tres ventanas. La primera ventana se presenta un ambiente donde se encuentra el paciente con los robots (Figura C-9). En la segunda ventana se observan tres herramientas quirúrgicas (tijera, retractor y pistola endoclip), las cuales son utilizadas por el robot tipo PA-10 (Figura C-10). Por último, la tercera ventana permite visualizar el interior de la cavidad abdominal del paciente, por medio de una cámara que es manipulada por el robot porta endoscopio Hibou (Figura C-11).

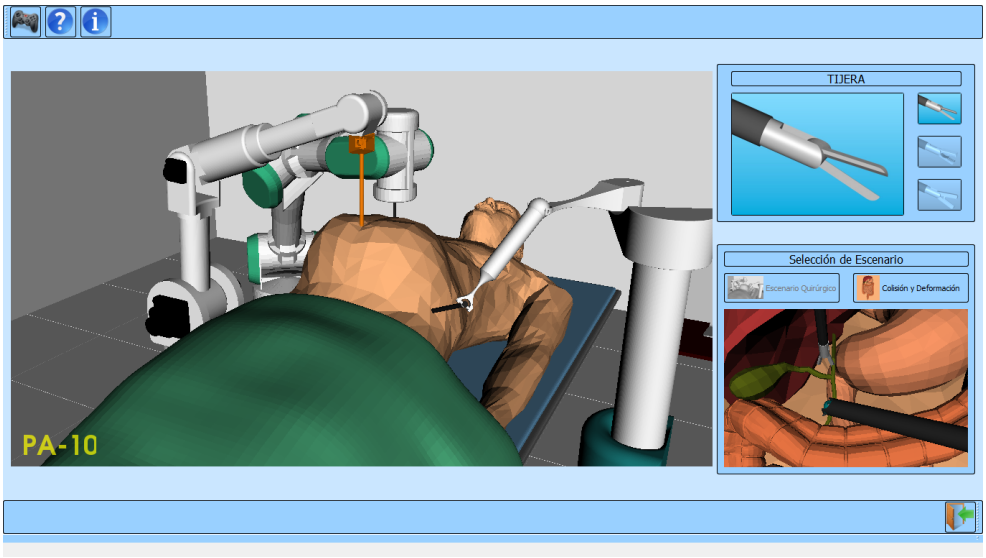

Figura C-9. Escenario Quirúrgico.

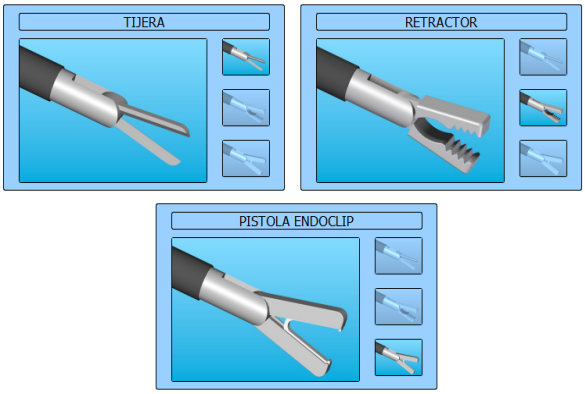

Figura C-10. Instrumentos quirúrgicos.

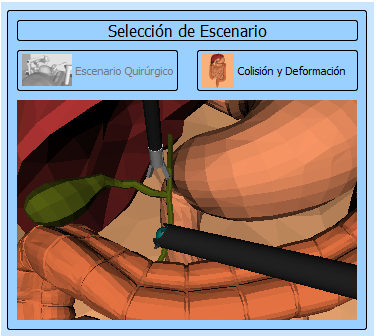

Figura C-11. Cámara del robot porta endoscopio HIBOU.

Al seleccionar el segundo escenario “Colisión y Deformación”, la aplicación desplegara el ambiente mostrado en la figura C-12. En este escenario solo se puede interactuar con el robot PA-10.

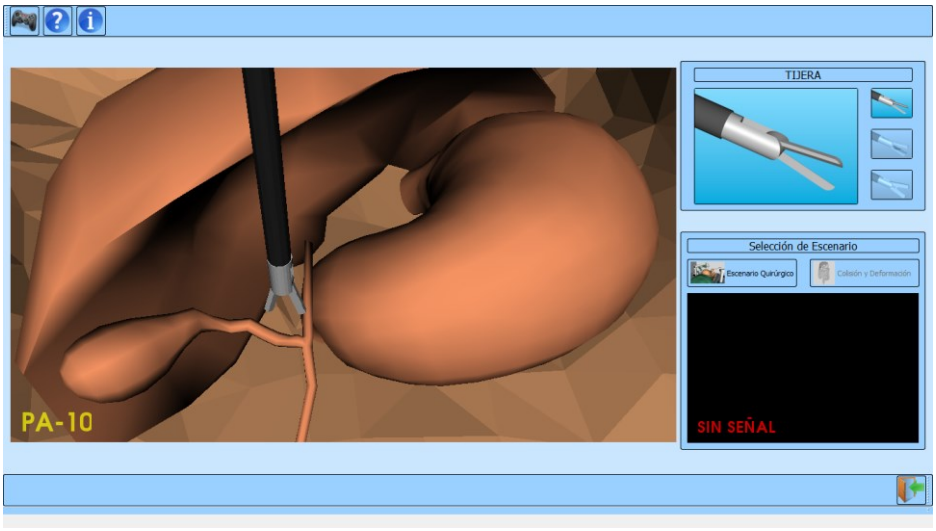

Figura C-12. Escenario Colisión y Deformación.

En este escenario, el usuario puede interactuar con algunos órganos de la cavidad abdominal, llevando el efector final del PA-10 a colisionar con los órganos y defórmalos si realiza una presión sostenida sobre él (Figura C-13).

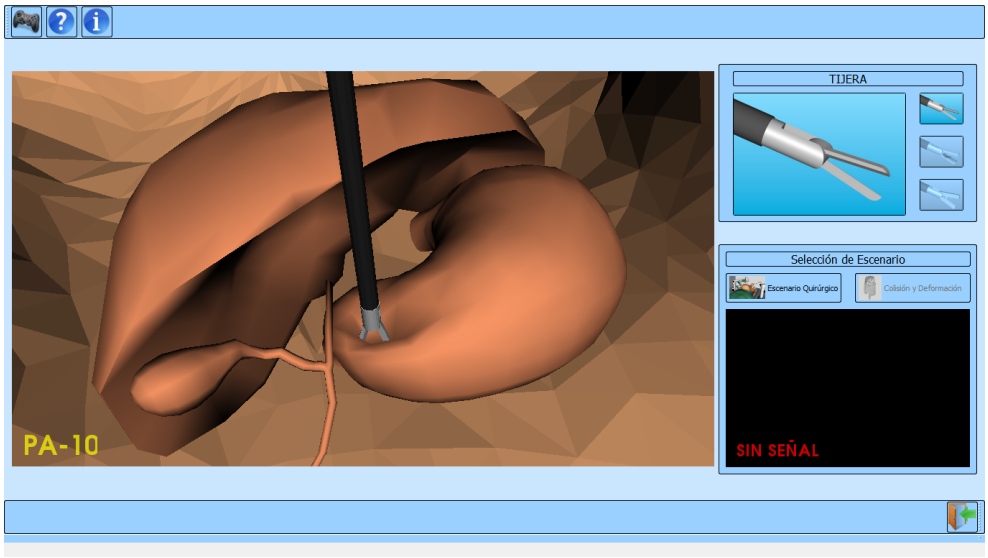

Figura C-13. Deformación sobre el estómago.

Seleccionando la pistola endoclip, por medio del botón “Selección de Pinza”, el usuario podrá colocar el endoclip sobre la cística, para ello debe ubicar el efector final del PA-10 sobre ella. Cuando este se encuentre en una posición cercana a la cística el efector final tendrá un giro de 90 grados, posicionando la pinza para colocar los endoclip (Figura C-14).

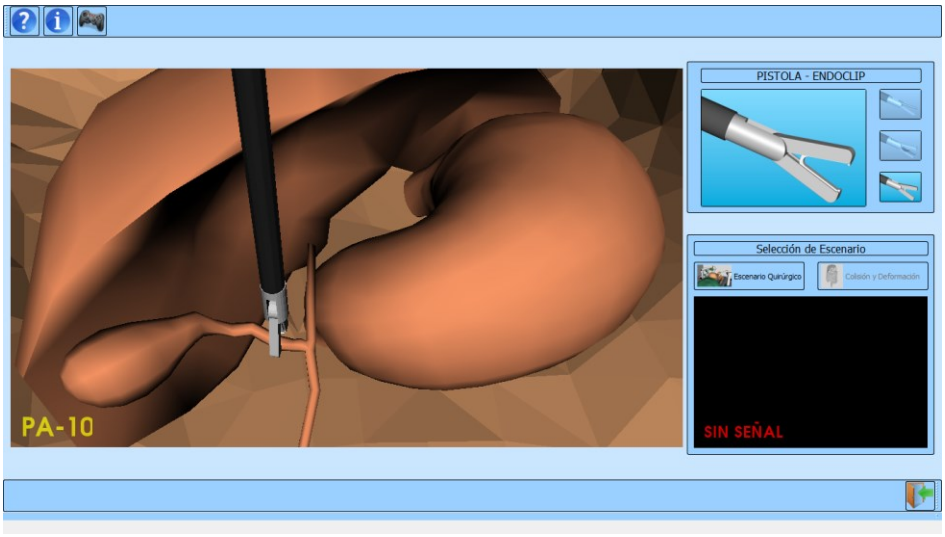

Figura C-14. Posicionamiento pistola endoclip.

El endoclip es puesto por medio del botón “Colocar Endoclip”, la posición en que se ubique el endoclip dependerá de la precisión del usuario, ya que si este no acierta sobre la arteria el endoclip se ubicará en el lugar que se presionó el botón (Figura C-15). Si la ubicación del endoclip no es la deseada el usuario tiene la opción de remover el endoclip a través del botón “Remover Endoclip”. Después de poner el primer endoclip el usuario podrá colocar un segundo endoclip siguiendo los pasos anteriormente mencionados. Si la posición del endoclip no es la deseada el usuario podrá igualmente removerlo. Cabe aclarar que solo se podrá remover el endoclip anteriormente puesto. Al cambiar de pinza el usuario ya no podrá manipular los endoclip anteriormente puestos y si retorna de nuevo a la pinza endoclip no los podrá colocar nuevamente ya que se encuentran ubicados en la cística.

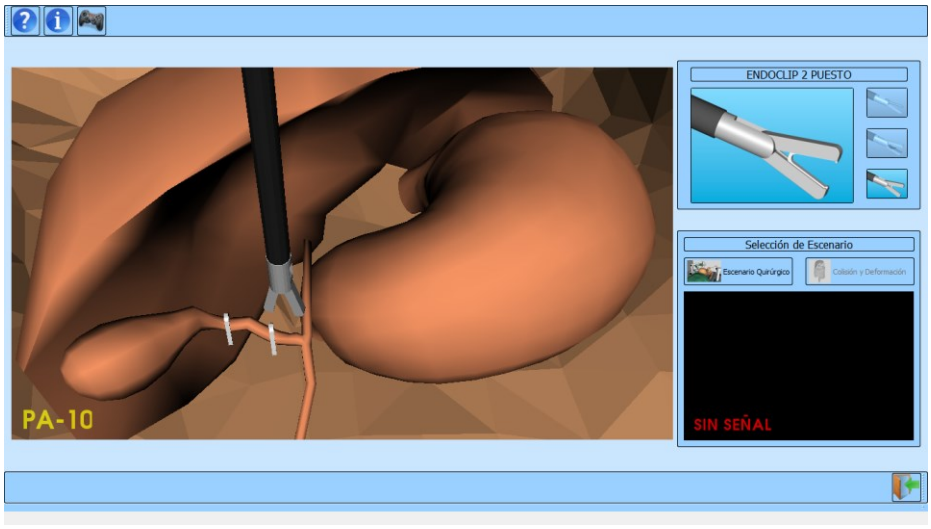

Figura C-15. Endoclip colocados.

Para salir de la aplicación el usuario deberá dar clic sobre el icono que aparece en la esquina inferior de la pantalla (Figura C-16).

Figura C-16. Opción salir.