UNIVERSIDAD POLITÉCNICA DE PUEBLA
Ingeniería en Informática

Proyecto de Estadía Profesional

“Implementación de rutinas de movilidad para robot Baxter”

Área temática del CONACYT: VII
Ingenierías y tecnologías

Presenta:
José Alberto Figueroa García.

Asesor técnico
Dr. Antonio Benítez Ruiz.

Asesor académico
M.C. Rebeca Rodríguez Huesca.

Juan C. Bonilla, Puebla, México. 19-Diciembre-2018

Resumen

Es muy fácil interactuar con la nueva creación de Rethink Robotics, aunque las innovaciones que incorpora el robot muestran las múltiples soluciones a diversos problemas tanto de la vida cotidiana como industriales.

En este documento se muestra la manera correcta de instalar el software necesario para el correcto funcionamiento de Baxter. De igual manera se describe cómo utilizar dos simuladores gráficos para generar rutinas de movilidad y visión sobre un robot, al igual que la correcta configuración de los parámetros de los mismos.

Índice

Índice de figuras

Índice de Tablas

1. Introducción

En este capítulo se hablará acerca del proyecto que ha sido seleccionado para su desarrollo y poder completar la etapa final de prácticas profesionales de la materia Estadía y así concluir con éxito la carrera. Hablaremos del por qué se escogió el proyecto y de igual manera la justificación del uso de diferente software para la simulación y generación de rutinas para el robot Baxter.

1.1 Descripción del problema o necesidad

En la actualidad, muchas empresas utilizan la robótica a su favor, ya que puede realizar tareas con pasos repetitivos mejor que un ser humano, pero en la gran mayoría de casos los robots diseñados para la industria no son de código abierto, esto nos limita su programación si se desea desarrollar o implementar otras aplicaciones.

La Universidad Politécnica de Puebla (UPPue), adquirió recientemente un robot Baxter (las especificaciones y descripción del robot se presentan en el siguiente capítulo), dado que no se cuenta con la documentación adecuada para su correcto funcionamiento, se pretende generar, desde la instalación del software necesario, hasta los pasos de guardar y ejecutar rutinas con el mismo robot.

1.2 Justificación

La Universidad Politécnica de Puebla, adquirió recientemente el robot Baxter, se deben desarrollar aplicaciones a mediano plazo, esto se pretende para dar nuevas posibilidades al robot Baxter dentro del campo de la investigación y en un futuro en el campo de la industria. Esto permitirá que los alumnos de Informática o Posgrado puedan desarrollar aplicaciones en la plataforma de Ubuntu. Esto permitirá que más alumnos tengan acceso al robot Baxter, para esto necesitamos establecer las operaciones mínimas para operar y programar a Baxter, y para eso, se necesita tener un equipo de cómputo dentro del laboratorio con la correcta configuración.

1.3 Objetivo General y Específicos

Objetivo General:

1. Implementar aplicaciones para la movilidad e identificación de objetos para el robot manipulador Baxter

Objetivos específicos:

1. Desarrollar rutinas de movilidad tipo demo para el robot Baxter

1.1 Implementar al menos 3 rutinas de movilidad para el robot Baxter que involucre ambos brazos
1.2 Almacenar las rutinas que puedan ser reproducidas en cualquier momento
1.3 Elaboración y edición de videos de cada una de las rutinas videos cortos y videos largos
1.4 Elaborar reporte técnico para describir como se desarrolló cada rutina

2. Revisar herramientas de software para el desarrollo de aplicaciones en el robot Baxter

   2.1. Elaborar manual completo de los pasos para la instalación
   2.2. Implementar al menos 3 rutinas de las herramientas de simulación
   2.3. Elaborar videos de ejecución de rutinas
   2.4. Elaborar documento que describa como utilizar las herramientas de simulación

3. Implementar identificación de formas a través del uso de cámaras para manipular objetos.

   3.1. Revisar documentación respecto a la versión de OpenCV que está instalada en el robot
   3.2. Determinar la versión de OpenCV e instalar en el equipo de computo
   3.3. Programar rutinas para la identificación de objetos

2. Metodología y herramientas

El creciente uso de los robots en la vida cotidiana del ser humano, genera la necesidad de nuevas y mejores herramientas tanto de software como de hardware, en algunas ocasiones el software con el que trabajan son bajo licencia y no son de código abierto, el robot Baxter fue creado por la compañía Rethink Robotics, para satisfacer necesidades tanto industriales como de desarrollo de software con código abierto.

2.1. Actividades a realizar

1. Configurar un equipo de cómputo con las herramientas necesarias para su correcto funcionamiento.
2. Generar 3 rutinas de movilidad mediante los controles manuales de Baxter.
3. Instalación de simuladores Gazebo Y RvIz.
4. Generar 3 rutinas de movilidad para cada uno de los dos simuladores ya instalados
5. Generar rutina de movimiento mediante de dispositivo joystick de Xbox 360
6. Programar rutinas para la detección de color y figuras.

2.2 Descripción de robot Baxter

Baxter (Figura 1) es un robot industrial construido por Rethink Robotics, una empresa de nueva creación fundada por Rodney Brooks. Baxter se presentó en septiembre de 2011. Se usa para trabajos industriales simples como cargar, descargar, clasificar y manipular materiales. [1]

Baxter es un robot de tipo humanoide que posee dos brazos con siete grados de libertad y tecnologías de detección de vanguardia, que incluyen la fuerza, la posición, la detección, el control de torque en cada articulación y cámaras integradas que permiten aplicaciones de visión por computadora. En la actualidad existen dos versiones del robot Baxter y la universidad cuenta con la versión para desarrolladores. [1]

Figura 1: Robot Baxter

Ventajas de Baxter Research Robot

- Configurado con SDK de código abierto.
- Mayor libertad de manipulación.
- Manipuladores individuales.
- El robot incluye varios sensores que permiten el desarrollo de aplicaciones a través de visión, proximidad color, forma y etc.

Desventajas de Baxter Research Robot

- Tiempo necesario y el número de actualizaciones para el software.
- Costo
- Actualmente ya no se fabrica esta versión del robot por lo que el soporte técnico puede ser un riesgo al corto plazo.
- Elaborar rutinas especificas requiere conocimientos básicos de Python y C++

Especificaciones de brazo(s) Baxter

Los nombres de las articulaciones de Baxter se mencionan varias veces en esta documentación [2]. Se puede observar una imagen con el etiquetado en la Figura 2, con la nomenclatura utilizada para todas las articulaciones:

- S (Shoulder) = Hombro
- E (Elbow) = Codo
- W (Wrist) = Muñeca

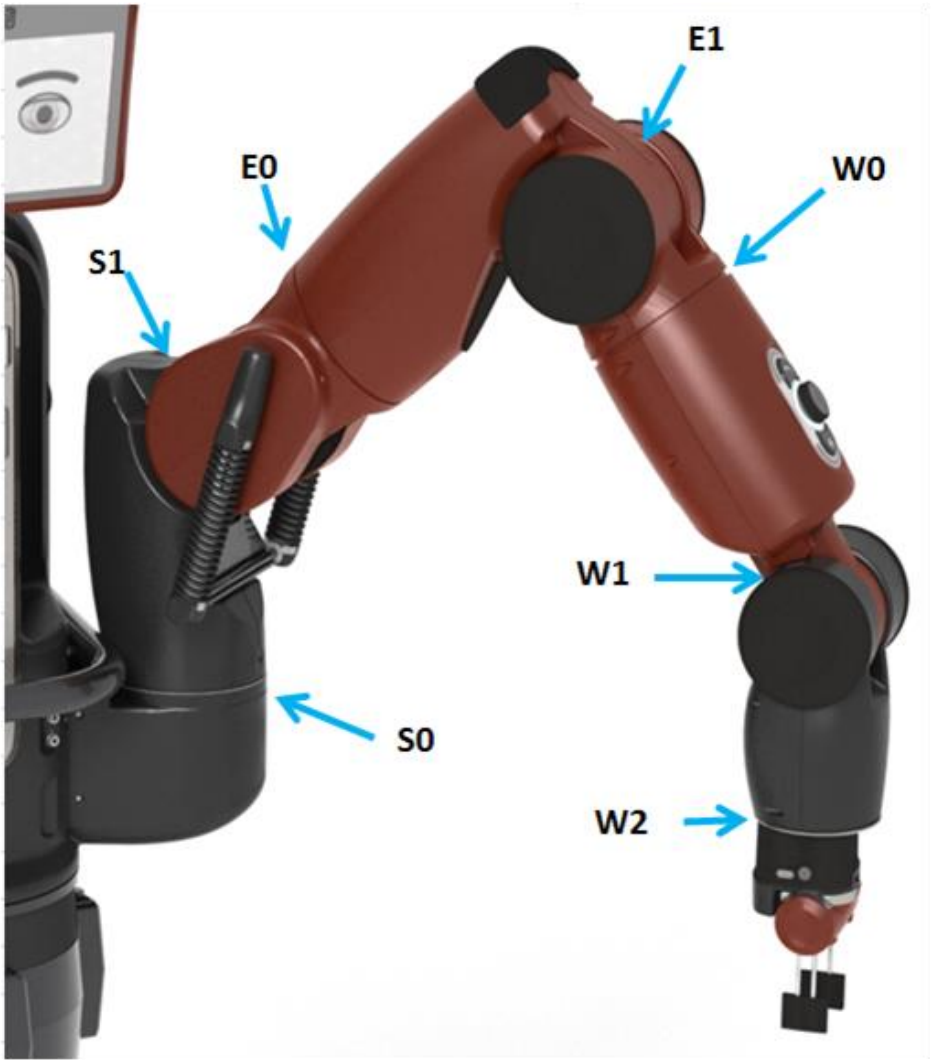

Figura 2: Articulaciones Baxter

Las longitudes de conexión para las articulaciones de Baxter se miden en mm, desde el centro de una conexión al centro de la siguiente, como se observa en la Figura 3. [2]

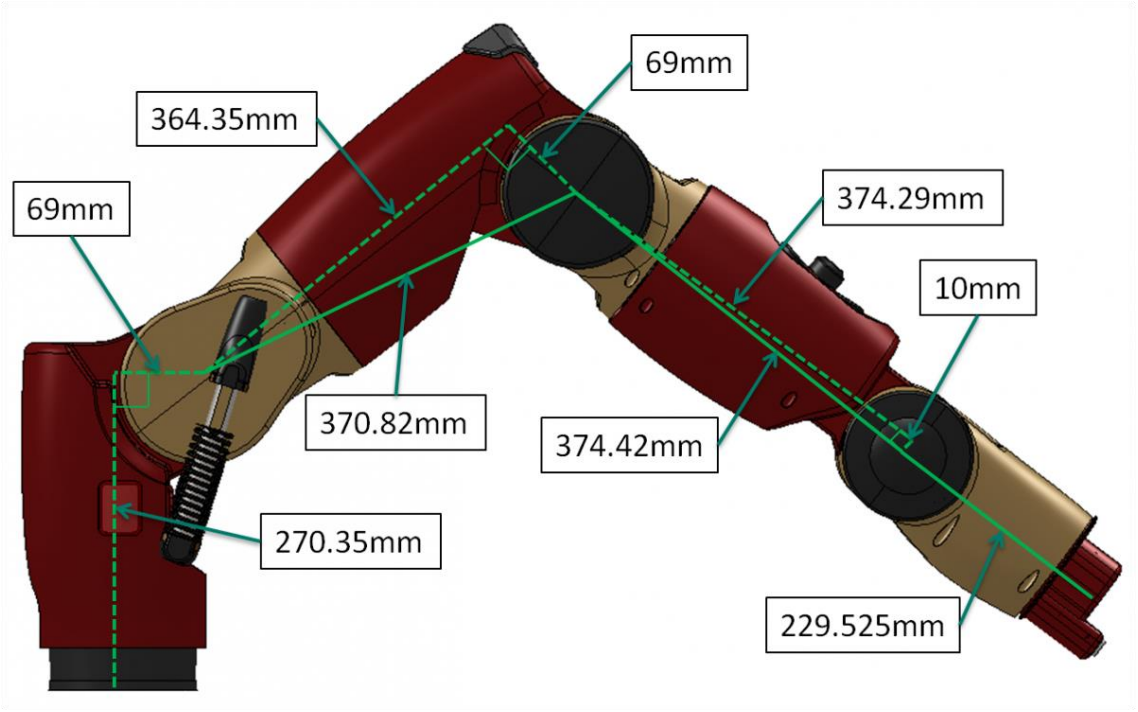

Figura 3: Longitudes de conexión de articulaciones

El rango de movimiento para cada articulación se observa en la Figura 4. La Tabla 1 muestra las medidas en grados y radianes. [2]

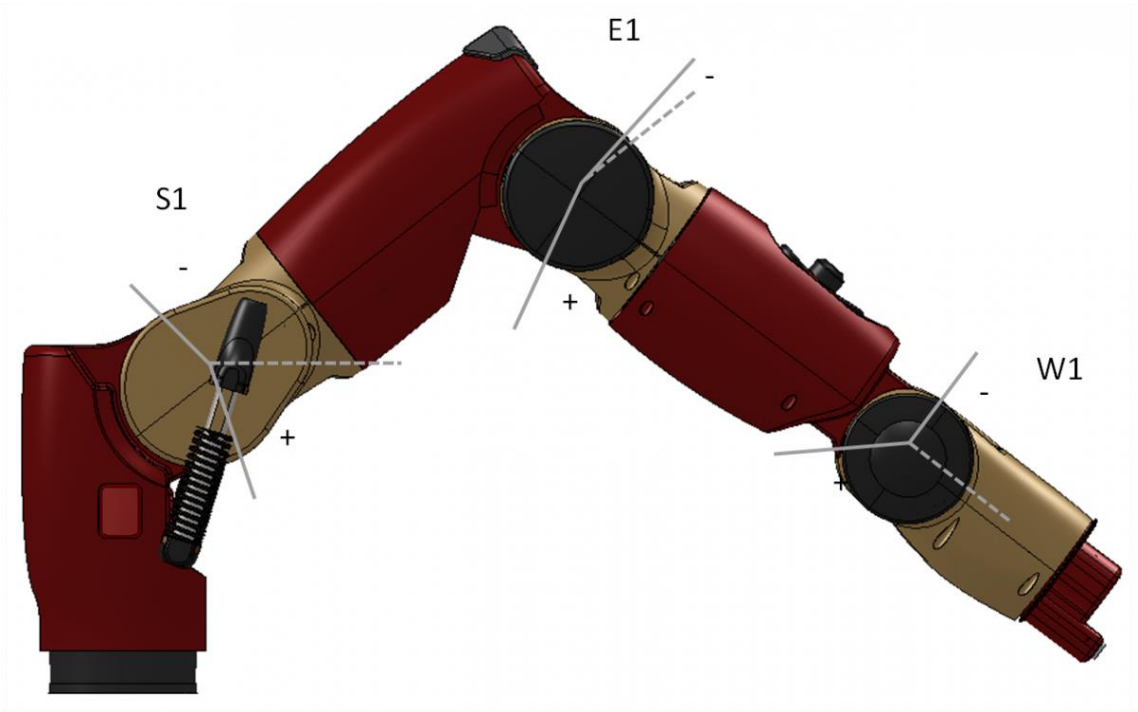

Figura 4: Rango de movimiento - Articulaciones

Articulación	(Grados) límite mínimo	Límite Max	Distancia	(Radianes) límite mínimo	Límite Max	Distancia
S1	-123	+60	183	-2.147	+1.047	3.194
S2	-2.864	+150	153	-0.05	+2.618	2.67
S3	-90	+120	210	-1.5707	+2.094	3.6647

Tabla 1: Tabla de rango de articulaciones (articulaciones de flexión)

En la Figura 5 se observa el rango de movimiento para cada articulación. La Tabla 2, muestra las medidas en grados y radianes y la Tabla 3 muestra las velocidades máximas por articulación en radianes sobre segundos. [2]

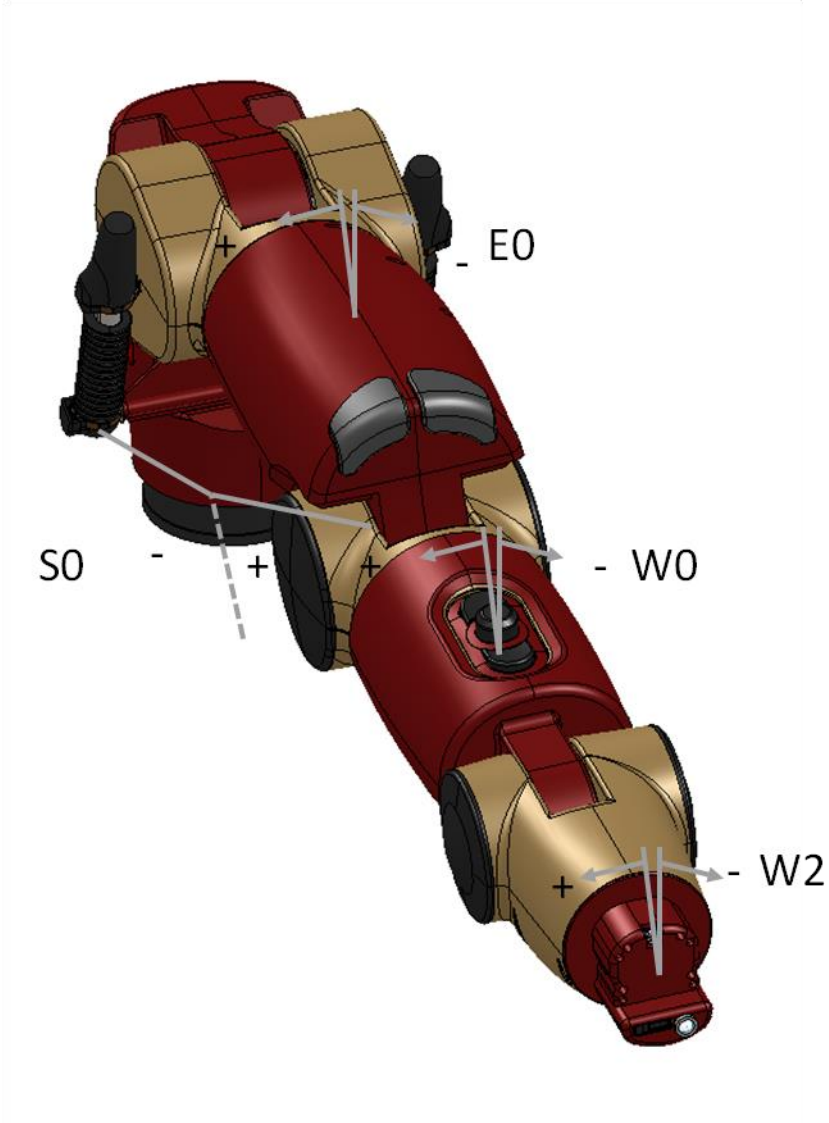

Figura 5: Rango de movimiento - Articulaciones torcidas

Articulación	(Grados) límite mínimo	Límite Max	Distancia	(Radianes) límite mínimo	Límite Max	Distancia
S0	-97.494	+97.494	194.998	-1.7016	+1.7016	*3.4033*
E0	-174.987	+174.987	349.979	-3.0541	+3.0541	*6.1083*
W0	-175.25	+175.25	350.5	-3.059	+3.059	6.117
W2	-175.25	+175.25	350.5	-3.059	+3.059	6.117

Tabla 2: Rango de movimiento - Articulaciones torcidas

Articulación	Velocidad Máxima
S0	2.0
S1	2.0
E0	2.0
E1	2.0
W0	4.0
W1	4.0
W2	4.0

Tabla 3: Velocidades máximas de la articulación (rad / seg)

Otras especificaciones de hardware.

En esta sección se habla de las características generales del robot Baxter como:

- Las cámaras posicionadas en los brazos, como se muestra en la Tabla 4.
- Especificaciones del CPU dentro del robot Baxter, como se muestra en la Tabla 5
- Conjunto general de especificaciones del robot, como se muestra en la Tabla 6.

Ya que son de gran ayuda para alcanzar los objetivos de este proyecto. [2]

Descripción	Especificaciones
Resolución Máxima	1280 x 800 pixeles
Resolución Efectiva	640 x 400 pixeles
Cuadros por segundo	30 cuadros por segundo
Longitud focal	1.2 mm

Tabla 4: Especificaciones de cámaras en brazos

Descripción	Especificaciones
Procesador	Procesador Intel Core i7-3770 de 3ra generación (8MB, 3.4GHz) con gráficos HD4000
Memoria	4GB, NO ECC, 1600MHZ DDR3
Disco Duro	Unidad de estado sólido de 128 GB

Tabla 5: CPU a bordo

Descripción	Especificaciones
Resolución de pantalla	1024 x 600 píxeles
Precisión posicional	+/- 5 mm
Carga útil máxima (incluido el efector final)	5 lb / 2.2 kg
Fuerza de agarre (máx.)	35N o 8 lbs
Rango de sensor infrarrojo	1.5 - 15 in / 4 - 40 cm

Tabla 6: Especificaciones misceláneas

2.3 Sublime Text 3

Sublime Text es un editor de texto y editor de código fuente, está escrito en C++ y Python para los plugins. Se puede descargar y evaluar de forma gratuita. Sin embargo no es software libre o de código abierto y se debe obtener una licencia para su uso continuo, aunque la versión de evaluación es plenamente funcional y no tiene fecha de caducidad. [3]

Ventajas de Sublime Text 3

- Es mucho más rápido que Sublime Text 2.
- Selección de texto múltiple.
- Más rápido y estable que Atom.
- Entrada de cursor múltiple.

Desventajas de Sublime Text 3

- No es 100% gratuito la licencia cuesta 70 dls.
- No es un editor de texto por default en ningún sistema operativo.

2.4 ROS Melodic

Sistema Operativo Robótico (en inglés Robot Operating System, ROS) es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. [4]

Se desarrolló originalmente en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford. [4]

Soporte requerido para:

- Ubuntu Artful (17.10)
- Ubuntu Bionic (18.04)

Ventajas de ROS Melodic

Tiene el soporte más amplio de todas las distribuciones de ROS [4], por ser el más nuevo:

- De Mayo de 2018 hasta Mayo de 2023 [4].

Soporta los siguientes lenguajes:

- C++ (14)
- Python (2.7)
- Python >= 3.5 (Se recomienda primero hacer pruebas)
- Lisp SBCL 1.3.14. [4]

Desventajas de ROS Melodic

Algunas de las desventajas de ROS Melodic son mayormente por ser un sistema demasiado nuevo estas son algunas de sus fallas:

- Si instala Melodic desde binarios durante la versión beta, tendrá que eliminar todos sus paquetes y volver a instalarlos.
- Actualmente no hay paquetes de depuración (que contengan los símbolos de depuración) disponibles para los paquetes Ubuntu Bionic o Artful.
- Muy poca documentación para su uso. [5]

2.5 ROS Indigo

ROS Indigo Igloo se dirigirá principalmente a la versión Ubuntu 14.04 LTS (Trusty Tar), Indigo solo admite la publicación, la documentación y las pruebas de integración de paquetes basados en catkin. Esto está impulsado especialmente por el objetivo de proporcionar soporte a largo plazo de esta distribución. Sin embargo, aún se admite la creación de paquetes basados en rosbuild desde la fuente. [6]

Soporte requerido para:

- Ubuntu Trusty (14.04)
- Ubuntu Saucy (13.10)

Ventajas de ROS Indigo

- Cmake_modules.
- Gazebo 2.0.
- Catkin [6].

Desventajas de ROS Indigo

- No cuenta con ninguna de las actualizaciones que se hacen en las versiones más nuevas de ROS. [7]

2.6 SDK Baxter

Baxter Research Robot SDK proporciona una interfaz de software que permite a los investigadores de todas las disciplinas desarrollar aplicaciones personalizadas para ejecutar en la plataforma Baxter. [8]

El SDK se conecta con el robot de investigación Baxter mediante ROS (Robot Operating System). Baxter proporciona un ROS Master autónomo al que cualquier estación de trabajo de desarrollo puede conectarse y controlar Baxter a través de varias API de ROS. [8]

Ventajas de SDK Baxter

- Contamos con un SDK libre para investigación y desarrollo.
- Rápida comunicación con los diversos sensores de Baxter.
- Comunicación universal con cualquier sistema de cómputo.

Desventajas de SDK Baxter

- Configuración larga.
- Se necesita de una conexión a internet forzosamente.

2.7 Gazebo

Un simulador bien diseñado permite probar rápidamente algoritmos, diseñar robots, realizar pruebas de regresión y entrenar el sistema IA (Inteligencia Artificial) utilizando escenarios realistas. Gazebo ofrece la capacidad de simular de manera precisa y eficiente poblaciones de robots en entornos interiores y exteriores complejos. [9]

Ventajas de Gazebo

- Simulación dinámica.
- Modelos de robots.
- Podemos simular sensores.
- Avanzados gráficos de 3D. [9]

Desventajas de Gazebo

- Necesita altos recursos de Hardware para su correcto funcionamiento.
- Actualizaciones pesadas y robustas que pueden alentar procesos.

2.8 MoveIt!

MoveIt! es un software de vanguardia para la manipulación móvil, que incorpora los últimos avances en planificación de movimiento, manipulación, percepción 3D, cinemática, control y navegación. Proporciona una plataforma fácil de usar para el desarrollo de aplicaciones de robótica avanzadas, la evaluación de nuevos diseños de robots y la construcción de productos integrados de robótica para áreas industriales, comerciales y otros. [10]

Ventajas de MoveIt!

- El complemento de planificación de movimiento.
- La solicitud del plan de movimiento. [11]

Desventajas de MoveIt!

- Al igual que Gazebo, requiere altos recursos de Hardware para su correcto funcionamiento.

2.9 RvIz

Es un visualizador 3D para mostrar datos de sensores e información de estado de ROS. Usando rviz, puede visualizar la configuración actual de Baxter en un modelo virtual del robot. También puede visualizar representaciones en vivo de los valores de los sensores que aparecen en ROS Topics. [12]

Ventajas de RvIz

Muestra en tiempo real los valores de:

- Datos de cámaras tanto frontales como de ambos brazos.
- Sensor infrarrojo para medidas de distancia.
- Información de sonares.
- Información de cámaras. [12]

Desventajas de RvIz

- Al igual que Gazebo y MoveIt!, requiere altos recursos de Hardware para su correcto funcionamiento.

2.10 Python 3

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad. [13]

Python 3.0 ("Python 3000" o "Py3k") es una nueva versión del lenguaje que no es compatible con las versiones inferiores de mismo Python.

El lenguaje es prácticamente el mismo, pero muchos detalles, especialmente el funcionamiento de objetos incorporados como diccionarios y cadenas de caracteres, han cambiado considerablemente.

Además, la biblioteca estándar se ha reorganizado en algunos lugares prominentes. [13]

Ventajas de Python 3

- Es una versión de Python más actualizada.
- Mayor compatibilidad que otras versiones de Python.

Desventajas de Python 3

- La curva de aprendizaje puede ser un poco amplia

3. Resultados

En este capítulo daremos a conocer los resultados de aplicar la metodología tanto al equipo de cómputo, como al robot Baxter, explicando paso a paso y con el apoyo de imágenes.

3.1 Configurar un equipo de cómputo con las herramientas necesarias para su correcto funcionamiento

El proceso de instalación completo se encuentra en el manual de "*Instalación de herramientas para configuración de equipo de cómputo*", en donde se puede encontrar la forma de instalar en Ubuntu 14.04 (Trusty Tar) el siguiente software:

- ROS Indigo
- SDK Baxter
- Gazebo
- MoveIt!
- RvIz

A continuación se mencionaran los pasos de instalación, de forma ordenada como se describe en el manual de instalación, como punto de partida, verificaremos que el ordenador donde estaremos trabajando, cumpla con los requerimientos mínimos para el correcto funcionamiento del software a descargar, una vez verificado procederemos con los siguientes pasos:

- Instalación de Sublime Text.
- Instalación de ROS Indigo.
- Instalación de SDK Baxter.
- Instalación de Gazebo.
- Instalación de MoveIt!
- Instalación de RvIz.

3.2 Generar 3 rutinas de movilidad mediante los controles manuales de Baxter

Todas las rutinas se encuentran en el manual de "*Implementación de rutinas de movilidad tipo demo usando ambos brazos de Baxter*", en este apartado se presenta sólo una rutina.

Como primer paso se necesita generar la conexión entre el ordenador y el robot Baxter, para eso necesitamos abrir una terminal como se muestra en la Figura 6, mediante el comando "*cd ~/ ros_ws*".

Figura 6: Cambio de directorio mediante terminal

Una vez dentro de este directorio se ejecuta el comando "*. baxter.sh*", que ayuda a generar la configuración más rápida y sencilla del entorno ROS para comunicarse con Baxter, así como podemos observar en la Figura 7.

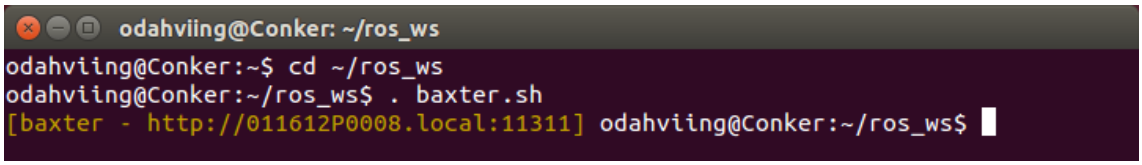

Figura 7: Conexión entre ordenador y Baxter

Teniendo la conexión ya establecida entre el ordenador y Baxter se ejecuta el comando con el cual se habilita a Baxter en gravedad cero y podremos manipular de manera libre los brazos de Baxter, "*rosrun baxter_tools enable_robot.py -e*", como se observa en la Figura 8.

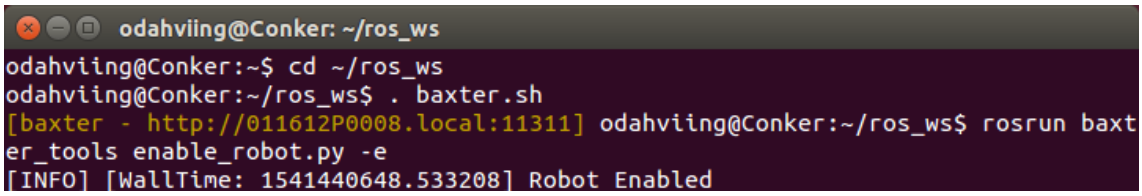

Figura 8: Gravedad cero en brazos de Baxter

Cuando se tiene habilitada la gravedad cero en los brazos de Baxter, se ejecuta el siguiente comando "rosrun baxter_examples joint_recorder.py -f <<Nombre de la rutina>>", este comando permite grabar el movimiento que se genera en tiempo real con Baxter, en el comando de grabación de movimientos en tiempo real, la parte de "*<<Nombre de la rutina>>*", tiene que ser substituido por el identificador con el cual nosotros llamaremos a la rutina creada, en este caso la nombraremos "*test*", esto se puede observar en la Figura 9.

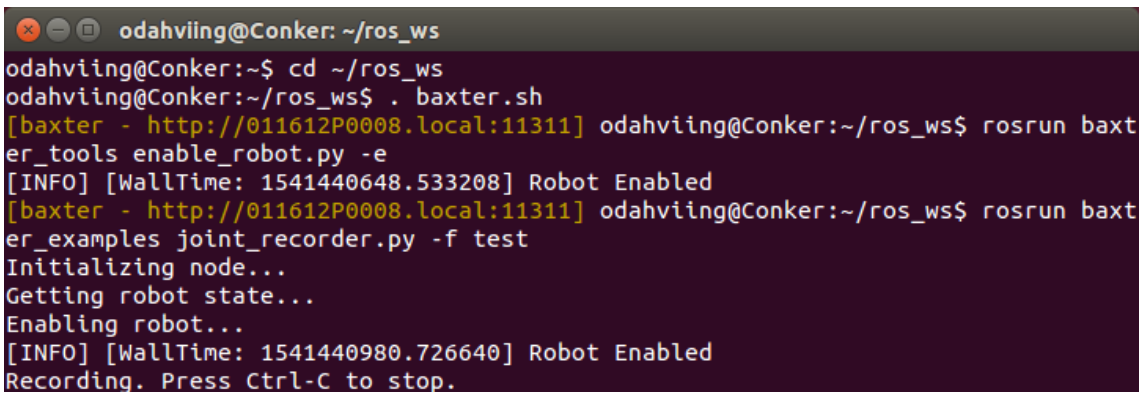

Figura 9: Grabación de una rutina

Cuando se terminen de realizar los movimientos para la rutina, se presionará la combinación de teclas "*Ctrl+C*" para finalizar con la grabación, como se muestra en la Figura 10.

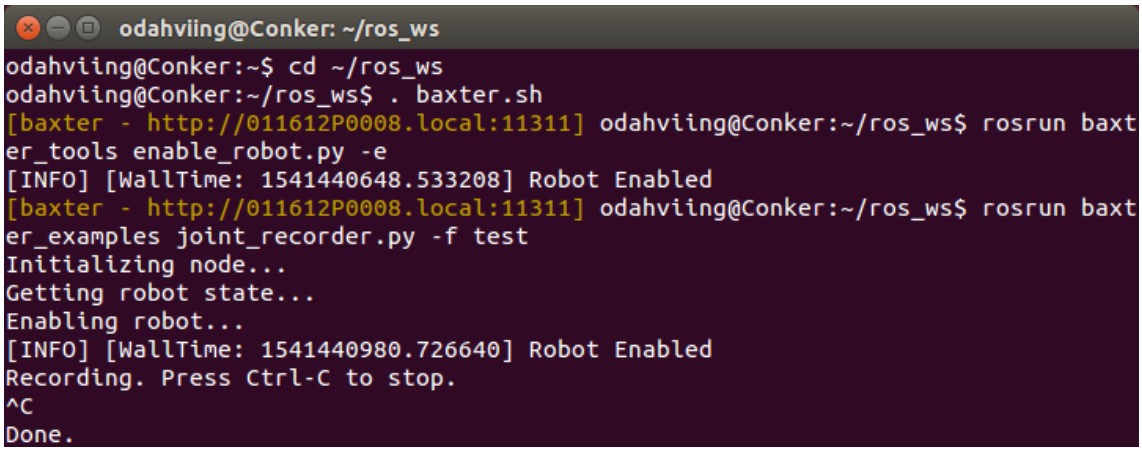

Figura 10: Rutina grabada

El archivo automáticamente será guardado dentro de la carpeta "*ros_ws*", como se observa en la Figura 11, la extensión del archivo es "*.txt*", al momento de dar doble click al documento este se abrirá y mostrará su contenido tal como lo muestra la Figura 12.

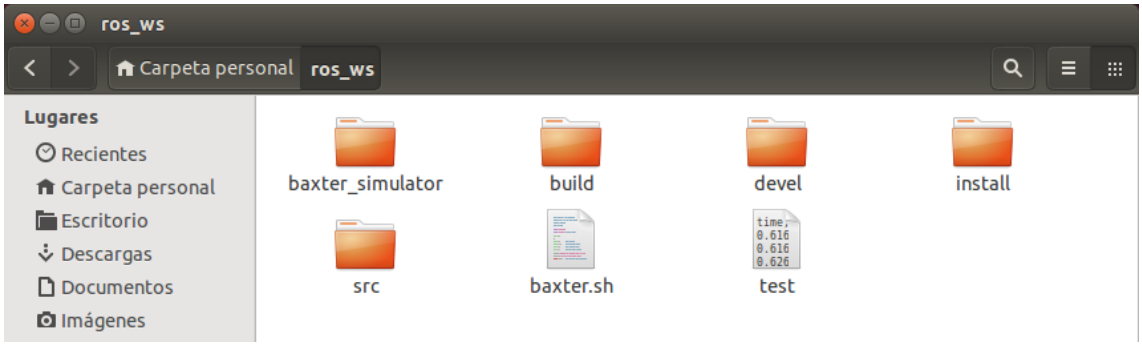

Figura 11: Carpeta "ros_ws"

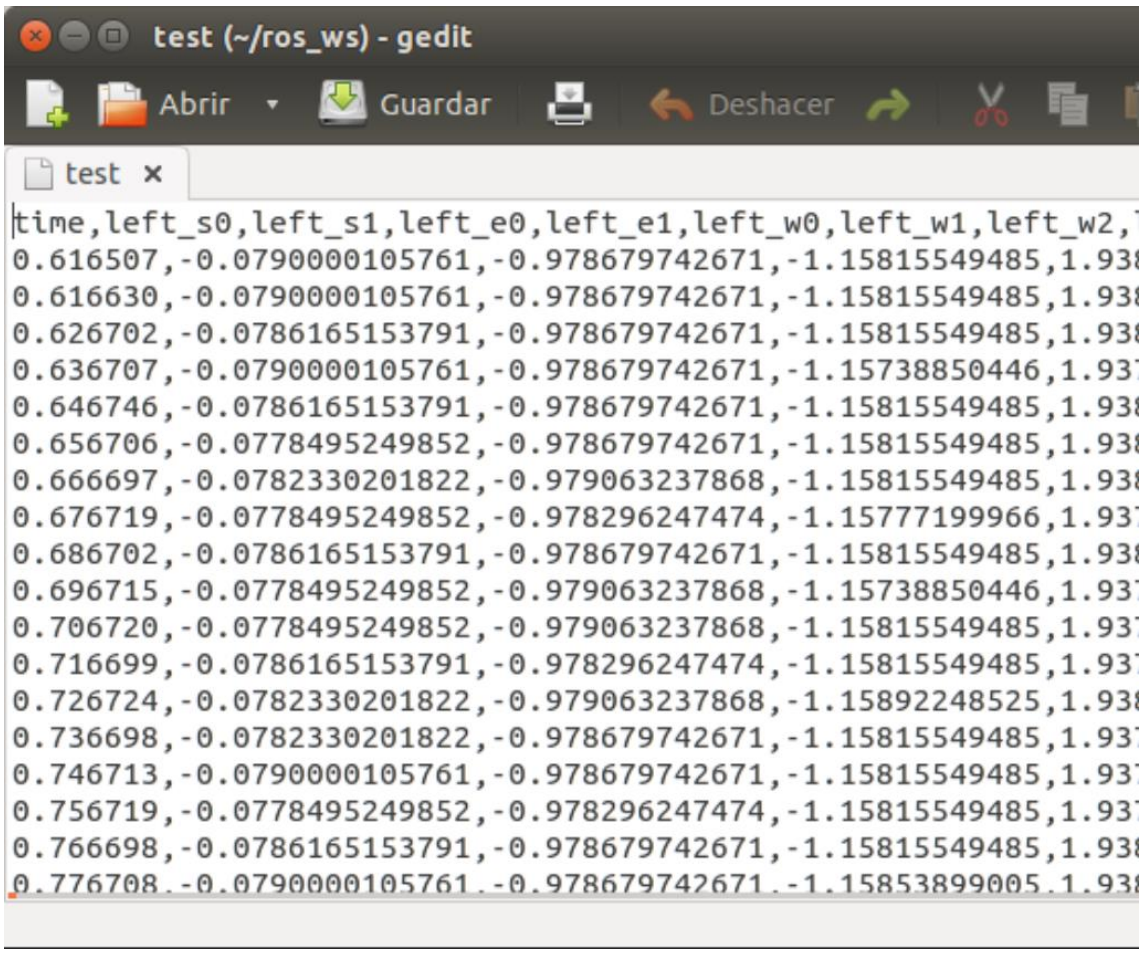

Figura 12: Contenido del archivo test

Como se observa en la figura 12, en la parte superior del archivo se encuentran descritas todas las articulaciones de Baxter, esto lo podemos observar como una matriz, el ejemplo se encuentra la Tabla 7.

time	**Left_s0**	**Left_s1**	**Left_e0**	**……**
0.616507	-0.0790000	-0.97867974	-1.15815549	……
0.616630	-0.0790000	-0.97867974	-1.15815549	……
0.626702	-0.0786165	-0.97867974	-1.15815549	……
……	……	……	……	……

Tabla 7: Ejemplo de archivo test

Si se toma como ejemplo la columna del tiempo (time), es el valor por el cual se van a regir los valores de las articulaciones en radianes, esto será para todas las articulaciones del robot, se graban las posiciones en las que se encuentra Baxter conforme pasa el tiempo, esos son los valores que son replicados por Baxter.

Ahora se ejecuta el archivo que se genera al grabar la rutina "*rosrun baxter_examples joint_position_file_playback.py -f <<Nombre de la rutina>>*", donde "*<<Nombre de la rutina>>*" será sustituido por el nombre del archivo que se generó, en este caso será "*test*", como se observa en la Figura 13.

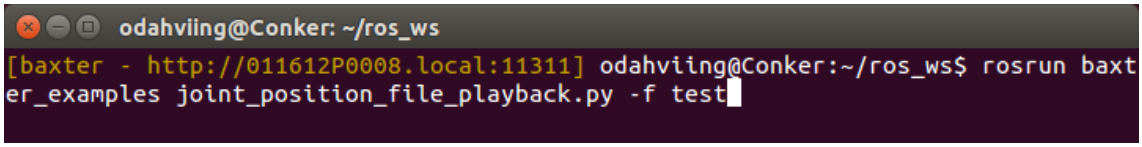

Figura 13: Ejecución del archivo test

Al momento de pulsar la tecla enter, se verificará si el robot se encuentra habilitado en modo de gravedad cero, una vez está habilitado el robot, mostrara en terminal el nombre del archivo con el cual se genera la rutina, como se observa en la Figura 14, una vez termine el ciclo saldrá de manera exitosa del archivo.

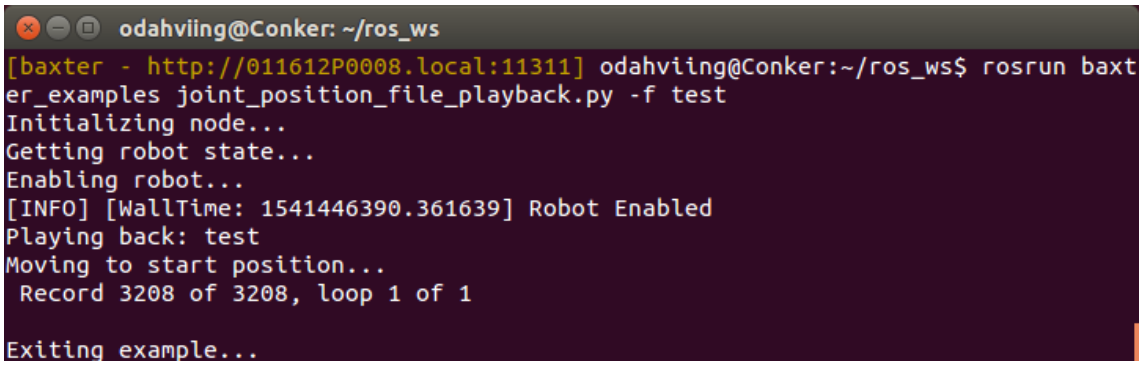

Figura 14: Ejecución terminada del archivo test

3.3 Instalación de simuladores Gazebo Y RvIz

El proceso de instalación completo se encuentra en un manual de "*Instalación de herramientas para configuración de equipo de cómputo*", en el apartado de "*Instalación de Gazebo*" e "*Instalación de RvIz*", este manual se encuentra en el apartado de anexos, a continuación se da un breve resumen de lo que son Gazebo y RvIz.

Gazebo

Ofrece la capacidad de simular de manera precisa y eficiente las poblaciones de robots en entornos complejos interiores y exteriores.

RvIz

Es un visualizador 3D para mostrar datos de sensores e información de estado de ROS. Usando rviz, puede visualizar la configuración actual de Baxter en un modelo virtual del robot.

3.4 Generar 3 rutinas de movilidad para cada uno de los dos simuladores ya instalados

Todas las rutinas se encuentran en el manual de "*Implementación de rutinas tipo demo usando RvIz y Gazebo*", en este apartado se presenta solo una rutina con RvIz y otra con Gazebo.

Rutina de RvIz

Se debe de generar la creación de dos modelos que son la representación virtual tanto de un escritorio, como de una caja con eso se pone a prueba la habilidad de RvIz para la detección de objetos virtuales y evitar así su colisión tanto de manera virtual como real, Los objetos siempre son guardados con extensión "*.scene*", la estructura de estos archivos se observa en la Figura 15.

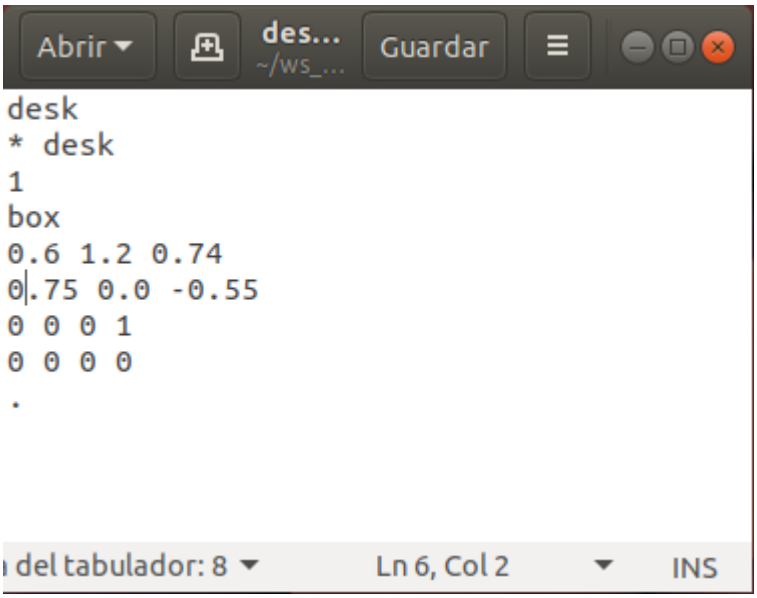

Figura 15: Archivo ".scene"

La explicación del contenido de estos archivos se encuentra en el manual de "*Implementación de rutinas tipo demo usando RvIz y Gazebo*", al igual que los pasos para agregarlos a RvIz.

Una vez se tienen los elementos del escritorio y la caja agregados en el ambiente virtual de "*RvIz*", se debe observar como la Figura 16.

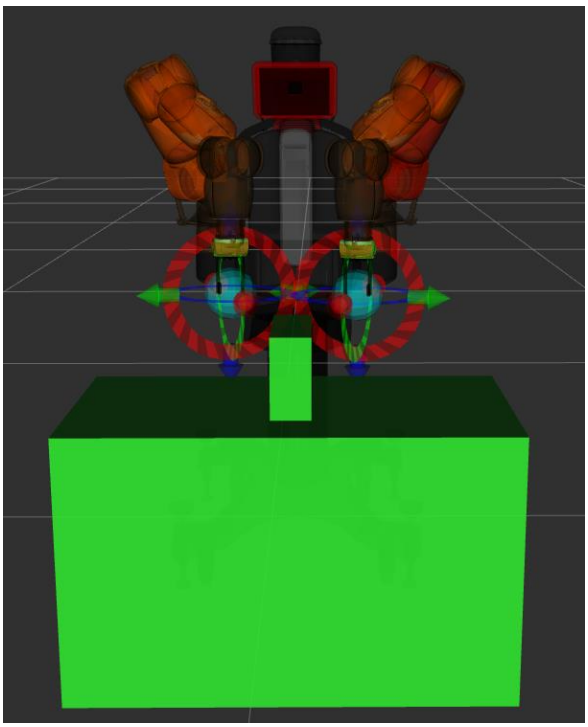

Figura 16: Ambiente grafico de Baxter y objetos

Una vez se tienen los objetos dentro del ambiente gráfico, se genera el mismo ambiente de trabajo con Baxter en tiempo real, como se observa en la Figura 17.

Figura 17: Ambiente de trabajo en tiempo real

Se selecciona la pestaña de “*Planning*” y procedemos a generar la trayectoria de la rutina, como se observa en Figura 18.

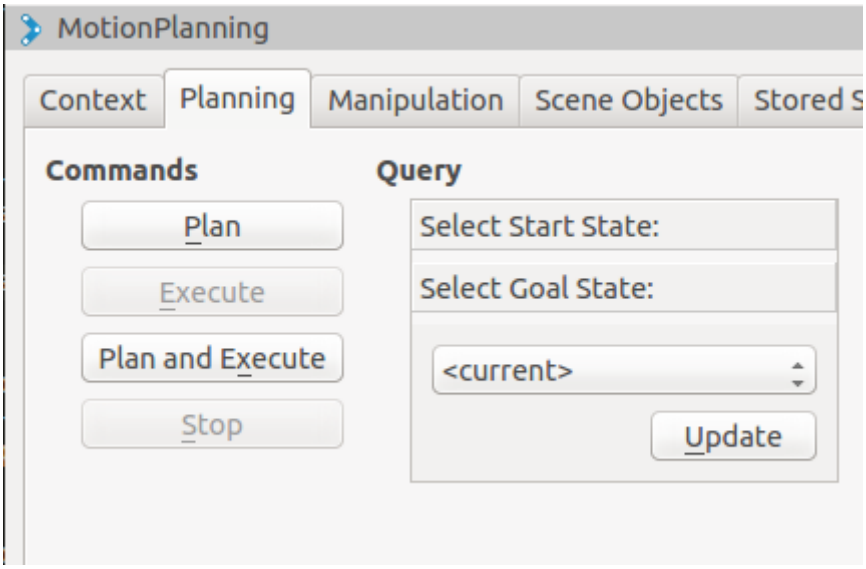

Figura 18: Pestaña de "Planning"

Ahora se puede observar la posición de los brazos en las Figura 19 y 20, se procede a presionar el botón “*Plan*” dentro de la pestaña de “*Planning*”, como se observa en la figura 18, y se observa el movimiento en tiempo real en la pantalla de RvIz.

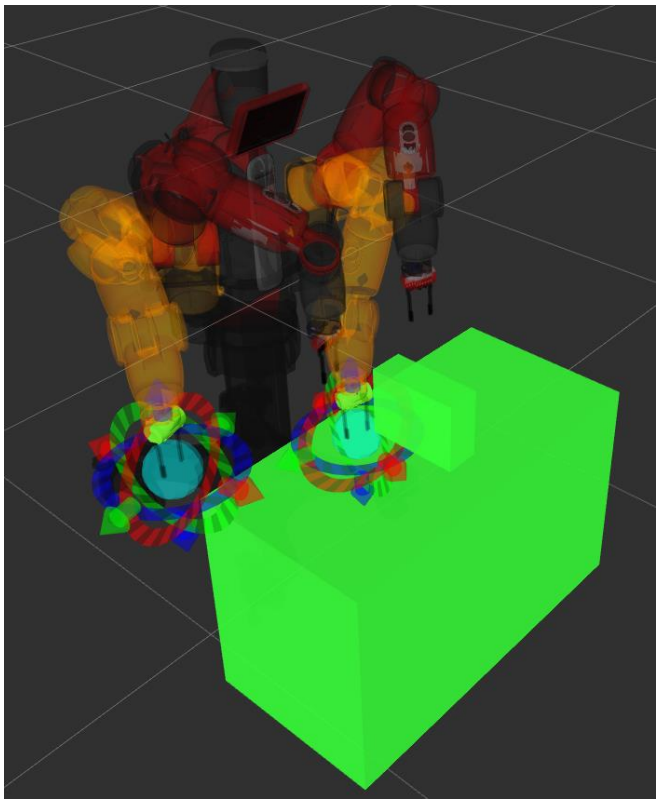

Figura 19: Baxter en RvIz

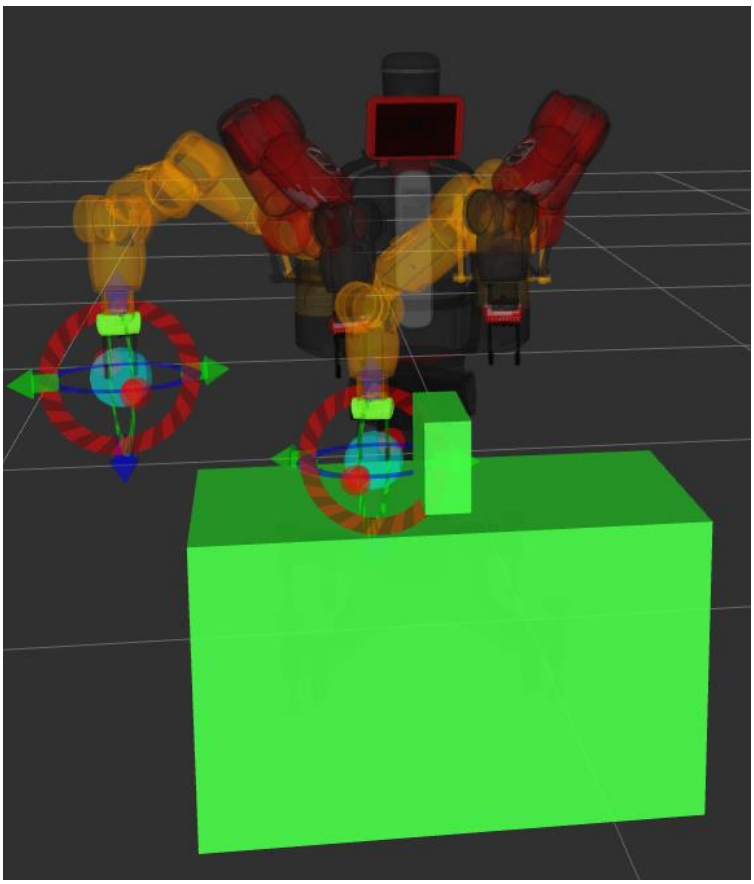

Figura 20: Baxter planeando ruta sin colisión

Una vez se realizó la trayectoria del movimiento se observa que nos permite ejecutar dando click en el botón de "*Execute*" y veremos el movimiento tanto en pantalla como en tiempo real, el punto final de la trayectoria en RvIz se muestra en la Figura 21 y en tiempo real en la Figura 22.

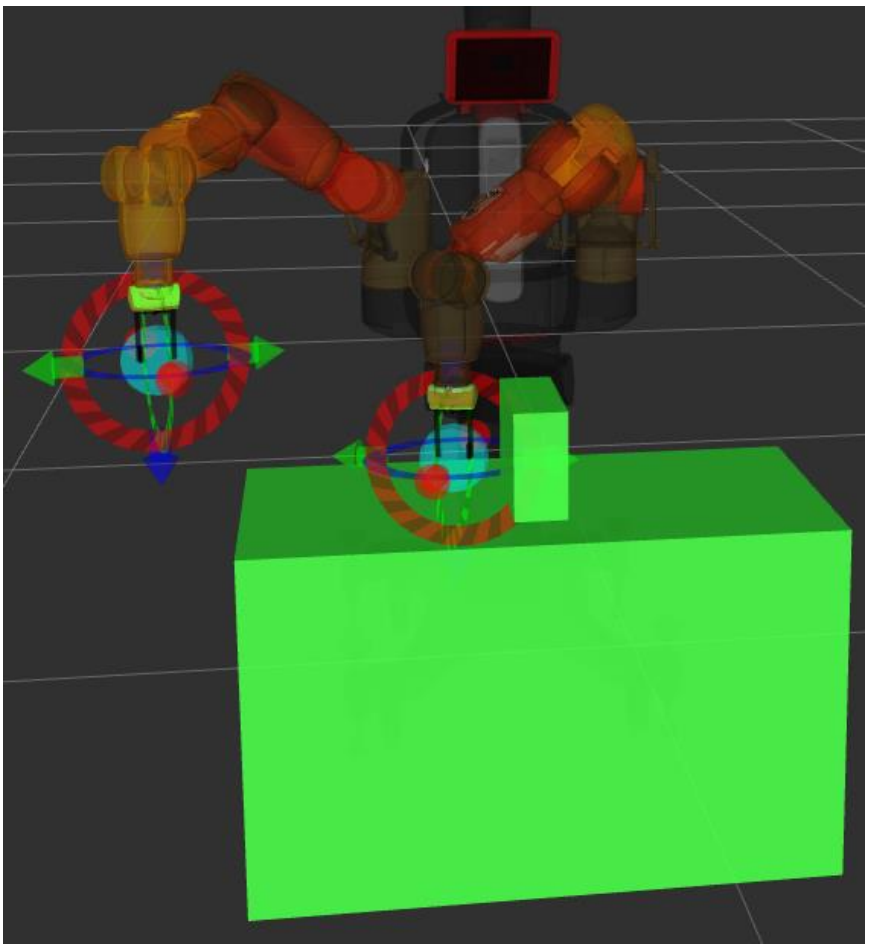

Figura 21: Ruta planeada y ejecutada con éxito

Figura 22: Respuesta de Baxter en tiempo real

Gazebo

Se abre una nueva terminal dentro de Ubuntu, se genera una nueva conexión con Baxter y se habilita la gravedad cero en sus brazos como se muestra en la Figura 23. Una vez se tiene la terminal preparada se ejecuta el comando que nos permite abrir Gazebo con un escenario vacío sólo se observará el modelo de Baxter, como se muestra en la figura 24.

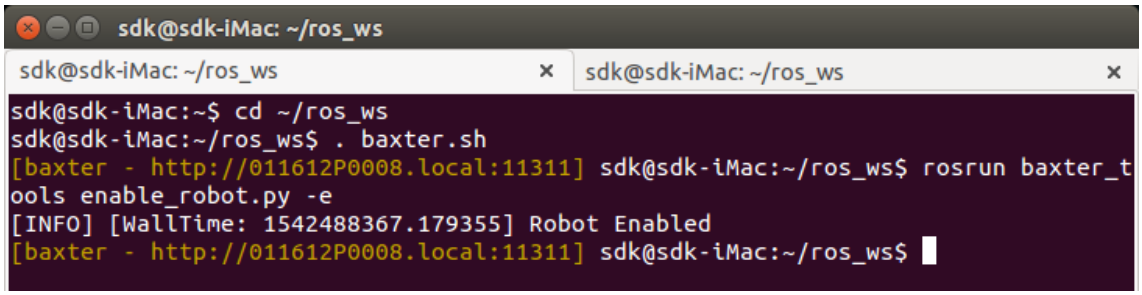

Figura 23: Conexión con Baxter

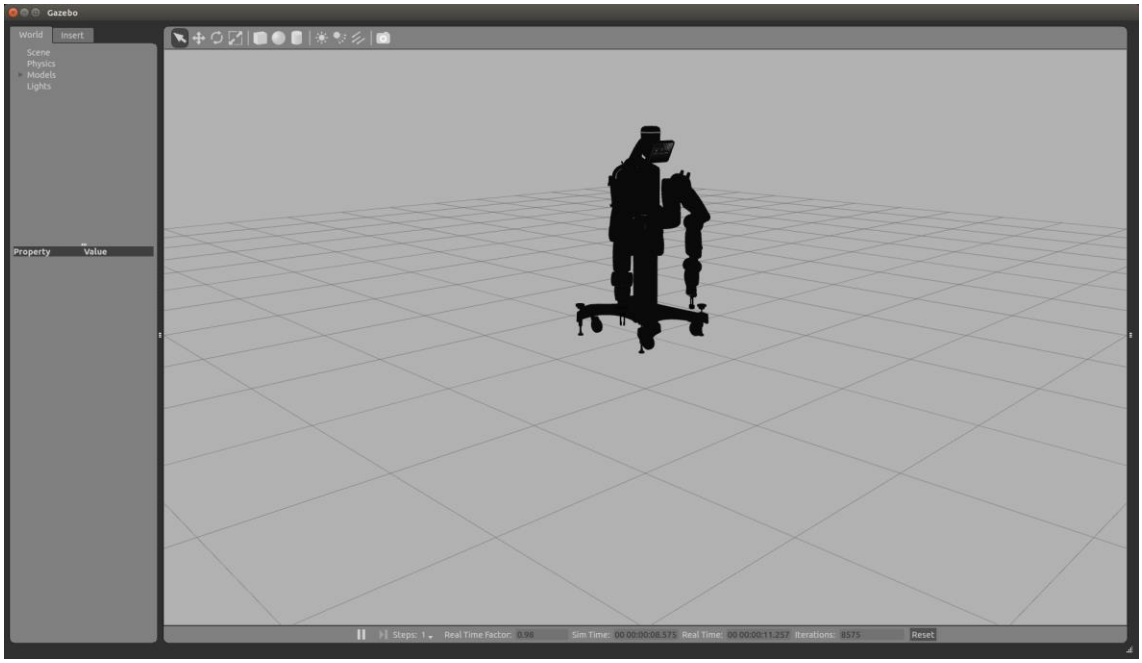

Figura 24: Escenario vacío solo con el modelo de Baxter

Se genera una nueva conexión con Baxter, y se ejecuta el comando con el cual se cargara un modelo de mesa y un modelo de cubo en Gazebo, esto se observa en la figura 25.

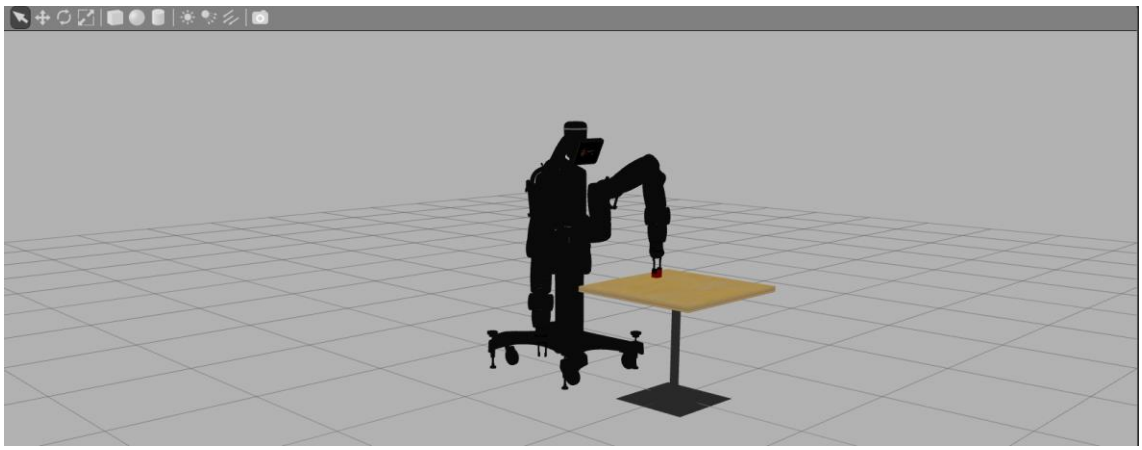

Figura 25: Ambiente vacío en Gazebo solo modelo de Baxter

Una vez esté siendo ejecutada, en la terminal donde se ejecuta el modelo de la mesa y el cubo, aparece las coordenadas de la trayectoria del brazo derecho de Baxter, como se observa en la Figura 26.

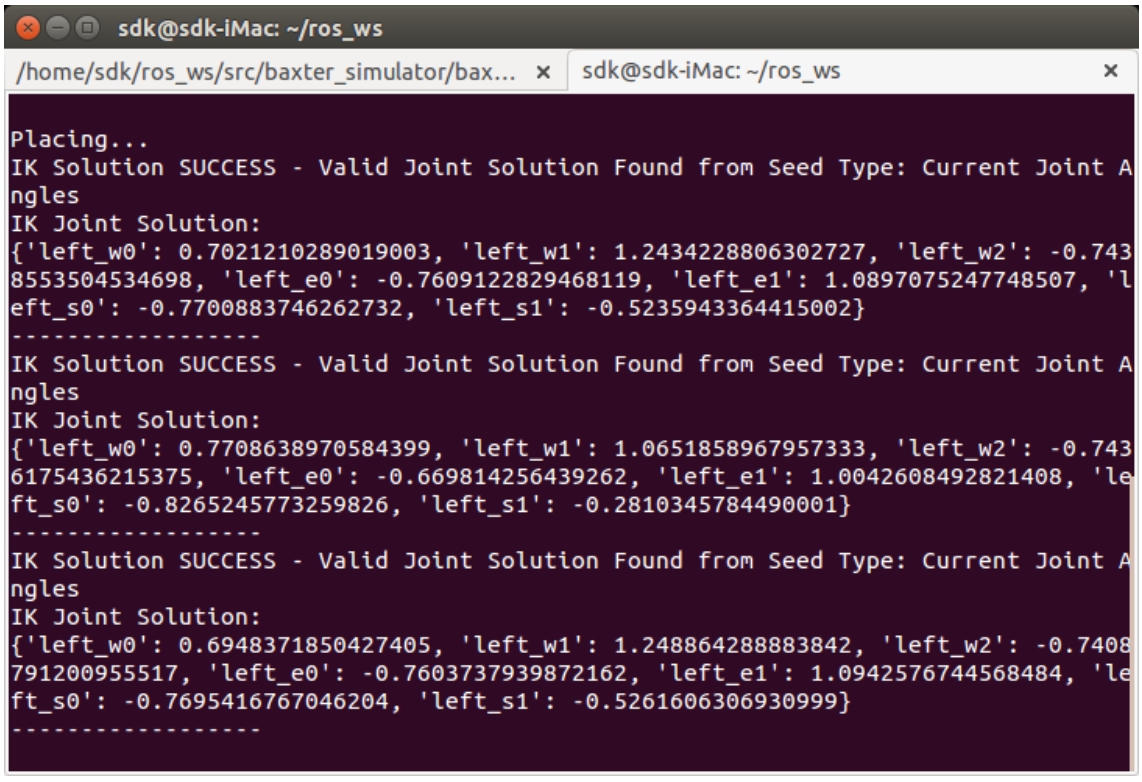

Figura 26: Coordenadas con las que trabaja baxter

La rutina será ejecutada mientras el proceso siga activo para terminar con el proceso se presiona la combinación de teclas “Ctrl + C”

3.5 Control de movimiento para robot Baxter a través de dispositivo joystick de Xbox 360

Todo el desarrollo de la rutina se encuentra en el manual de "*Control de movimiento para robot Baxter a través de dispositivo joystick de Xbox 360*", en el apartado de "*Control de Baxter por joystick*", a continuación se da un breve resumen de lo que es un joystick de Xbox 360.

Xbox 360

Es el principal controlador o joystick de la consola Xbox 360 de Microsoft. El mismo tiene dos versiones, con cable o inalámbrico y ambos son compatibles con la PC

3.5 Rutina experimental para detección de figuras

El uso de las cámaras es una gran herramienta para poder generar rutinas más complejas con Baxter, para esto se utilizó el código "*Pick and learn*" [14], que se encuentra en la página de rethink robotics, mediante este código se verifica la correcta configuración de Baxter, ya que hace uso de altos requerimientos de BAXTER.

Para esto se necesita estar un poco más familiarizado con los siguientes temas:

OpenCV

OpenCV (Open Source Computer Vision Library) se publica bajo una licencia BSD y, por lo tanto, es gratis para uso académico y comercial.

Tiene interfaces C ++, Python y Java y es compatible con Windows, Linux, Mac OS, iOS y Android. OpenCV fue diseñado para la eficiencia computacional y con un fuerte enfoque en las aplicaciones en tiempo real.

Escrita en C / C ++ optimizado, la biblioteca puede aprovechar el procesamiento multi-core. Habilitado con OpenCL, puede aprovechar la aceleración de hardware de la plataforma de cómputo heterogénea subyacente. [15]

A continuación se muestra en las Figuras 27, las librerías utilizadas tanto de "*OpenCV*", como las librerías que se encuentran instaladas para la interacción de Baxter "*ROS*", al igual que una breve descripción de su uso dentro del código.

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <baxter_core_msgs/CameraSettings.h>
#include <baxter_core_msgs/CameraControl.h>
#include <baxter_core_msgs/OpenCamera.h>
#include <baxter_core_msgs/CloseCamera.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <math.h>
#include "piece.h"
```

Figura 27: Librerías utilizadas en reconocimiento de objetos 1

<ros/ros.h>

Es una conveniencia que incluye todos los encabezados necesarios para usar las piezas públicas más comunes del sistema ROS. [16]

<image_transport/image_transport.h>

Proporciona clases y nodos para transportar imágenes en representaciones arbitrarias por cable, al tiempo que abstrae esta complejidad para que el desarrollador solo vea mensaje de "*sensor_msgs/Image*". [16]

<cv_bridge/cv_bridge.h>

Clase de mensaje de imagen que es interoperable con "*sensor_msgs/Image*", pero usa una representación cv :: Mat más conveniente para los datos de imagen. [17]

<sensor_msg/image_encoding.h>

Convierta entre el formato antiguo (sensor_msgs :: PointCloud) y el nuevo (sensor_msgs :: PointCloud2). [18]

<baxter_core_msgs/CameraSettings.h>

Mensajes y servicios necesarios para la comunicación con el robot de investigación Baxter de Rethink Robotics. Genera la configuración rápida de la cámara. [19]

<baxter_core_msgs/CameraControl.h>

Mensajes y servicios necesarios para la comunicación con el robot de investigación Baxter de Rethink Robotics. Genera la configuración rápida del control de la cámara. [19]

<baxter_core_msgs/OpenCamera.h>

Mensajes y servicios necesarios para la comunicación con el robot de investigación Baxter de Rethink Robotics. Genera la configuración rápida para la apertura de la cámara. [19]

<baxter_core_msgs/CloseCamera.h>

Mensajes y servicios necesarios para la comunicación con el robot de investigación Baxter de Rethink Robotics. Genera la configuración para detener la comunicación con la cámara. [19]

<opencv2/imgproc/imgproc.hpp>

Encuentra una plantilla arbitraria en la imagen en escala de grises usando la Transformada de Hu's Generalizada. [20]

<opencv2/highgui/highgui.hpp>

Si bien OpenCV se diseñó para su uso en aplicaciones a gran escala y se puede usar dentro de marcos de UI funcionalmente ricos (como Qt, WinForms o Cocoa) o sin ninguna interfaz de usuario, a veces es necesario probar la funcionalidad rápidamente y visualizar los resultados. Esto es para lo que se diseñó el módulo HighGUI. [21]

Ahora que se tiene una idea general de las librerías utilizadas tanto del sistema interno de Baxter y de OpenCV, para descargar este código de ejemplo se ejecutan los siguientes comandos en terminal, "*cd ~/ros_ws/src*" para cambiar el directorio de terminal, ejecutamos el siguiente comando para la correcta descarga del código "*git clone 'https://github.com/haxelion/baxter_pickandlearn.git'*", como se muestra en la Figura 28.

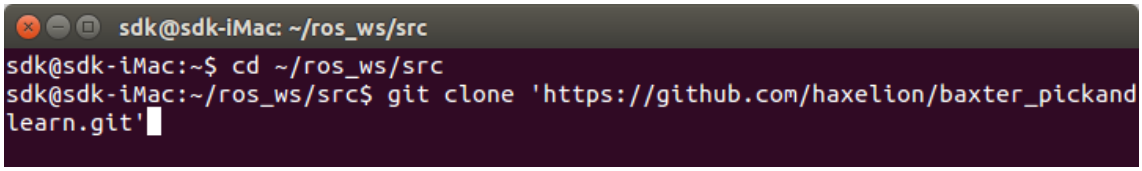

Figura 28: Descarga de pick and learn de Baxter

Una vez se completa la descarga de los archivos se ejecuta el siguiente comando para cambiar el directorio de terminal "*cd ~/ros_ws*", una vez estemos en el directorio correcto se genera una conexión con Baxter con el siguiente comando "*. baxter.sh*" y se compila "*catkin*" con el siguiente comando "*catkin_make*", como se muestra en la Figura 29.

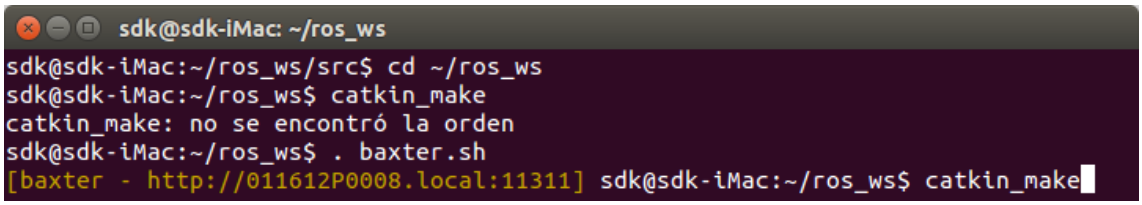

Figura 29: Compilando catkin

Una vez que termina de compilar "*catkin*", se ejecuta el siguiente comando para la ejecución de "*Baxter pick and learn*", "*rosrun baxter_pickandlearn pickandlearn*", genera un post con Baxter para establecer las conexiones correctas esto es mostrado en la pantalla de terminal donde se está ejecutando el programa, como se muestra en la Figura 30.

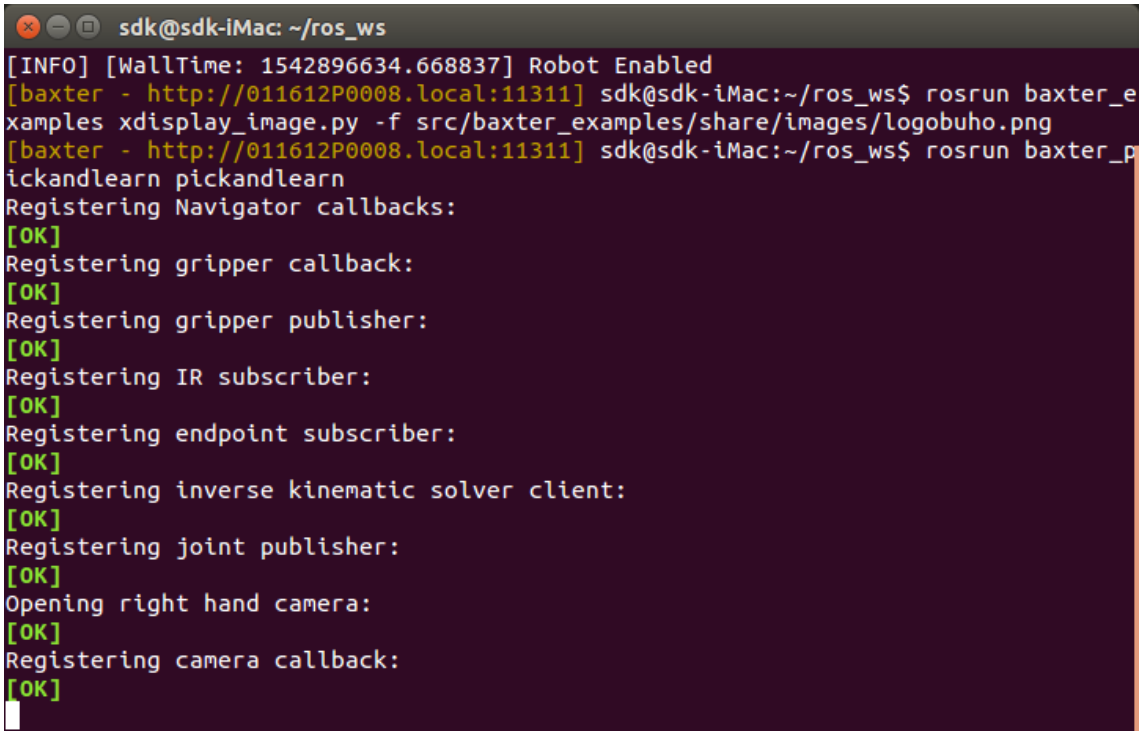

Figura 30: Post para establecer las conexiones

Se despliega una nueva terminal donde se observa la entrada de datos de la cámara, como se muestra en la Figura 31.

Figura 31: Datos captados por cámara

Para guardar un objeto basta con presionar la perilla controladora que se encuentra en la parte frontal del brazo de Baxter como se observa en la figura 32.

Figura 32: Control de brazo

Al presiona la perilla del brazo de Baxter en terminal nos muestra el mensaje de "*Piece 1 saved*", como se muestra en la figura 33.

```
Opening right hand camera:
[OK]
Registering camera callback:
[OK]
Piece 1 saved:
```

Figura 33: Mensaje de pieza guardada con éxito

Para ejecutar la búsqueda de la pieza en tiempo real se presiona el botón debajo de la perilla que se encuentra en el brazo de Baxter, como se muestra en la Figura 34.

Figura 34: Botón para inicio de búsqueda

Comienza la búsqueda del objeto, al encontrar un objeto marca las coordenadas, como se muestra en la Figura 35, comienza a bajar el brazo para recoger el objeto encontrado como se muestra en la Figura 36 y 37.

```
Opening right hand camera:
[OK]
Registering camera callback:
[OK]
Piece 1 saved:
Found at x: 0.808043 y: -0.376421 z: -0.193171
```

Figura 35: Coordenadas de la pieza encontrada

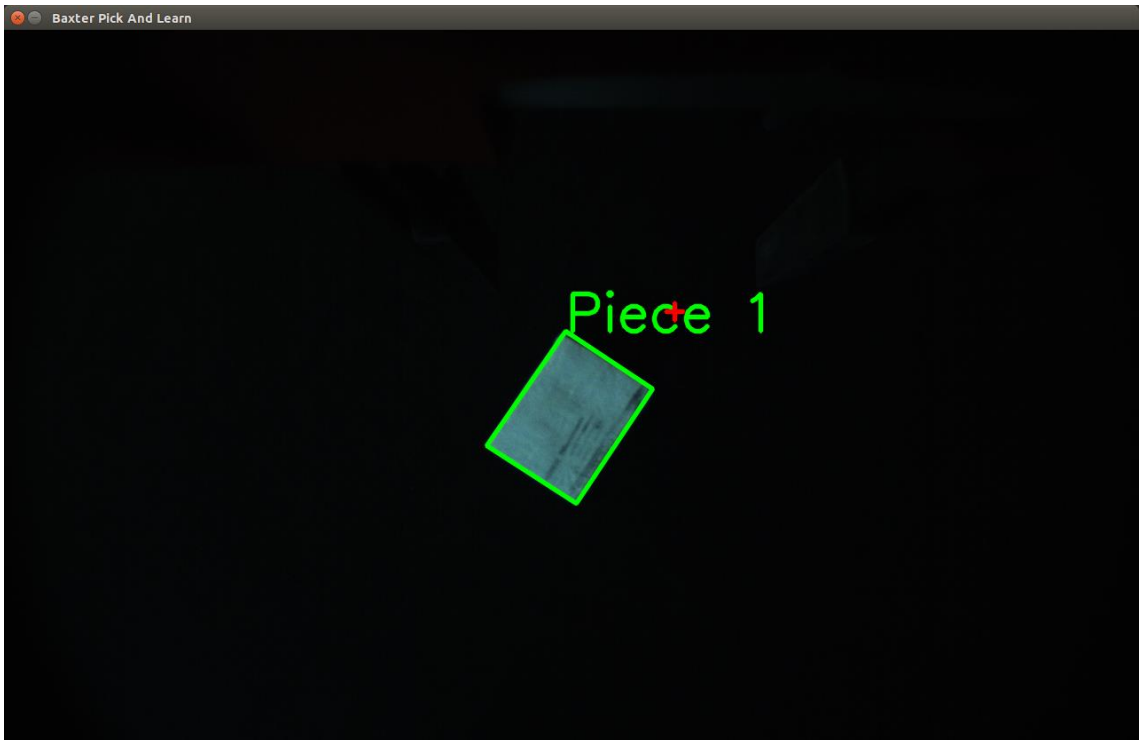

Figura 36: Detección de objeto

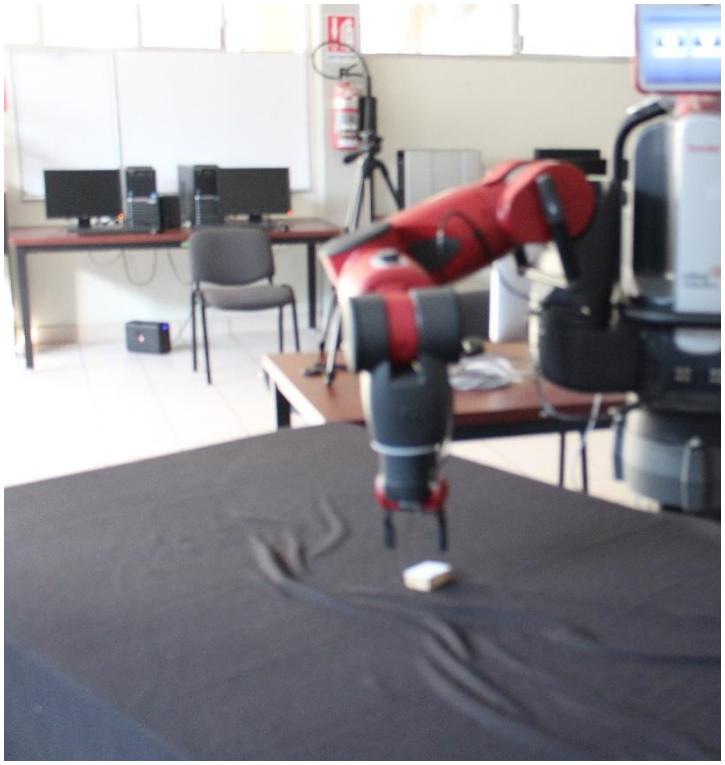

Figura 37: Desplazamiento de brazo en tiempo real

4. Conclusiones y recomendaciones

El principal objetivo de este trabajo era ser capaces de realizar la correcta configuración de un equipo de cómputo dentro del laboratorio, al igual que la descarga, instalación y configuración del software necesario para el correcto funcionamiento de Baxter, ser capaces de interactuar con el robot simulado desde RvIz y Gazebo.

Cuando se comenzó a elaborar este trabajo se desconocía el funcionamiento del Sistema Operativo Robótico (ROS), RvIz y Gazebo. Teniendo en cuenta que todo el proceso de elaboración del trabajo ha sido bajo este “sistema operativo”, se considera primordial aprender a manejarlo. Después de una labor de investigación y haber trabajado con él todo este tiempo, se consideran adquiridos los conocimientos necesarios acerca de su funcionamiento y de la forma de cómo se comunican sus procesos, además de haber aprendido a utilizar un gran número de herramientas que nos ofrece.

El siguiente objetivo consistía en familiarizarse con el simulador Gazebo, pues tampoco se conocía su funcionamiento, desde el comienzo del proyecto. De igual manera que con ROS, tras la labor de investigación llevada a cabo y el trabajo desarrollado, se considera superado este objetivo. Siendo capaces de simular un entorno y un robot en él correctamente, y comprendiendo el funcionamiento del programa.

Debíamos ser capaces de integrar todas estas herramientas para que interactuasen con Baxter. Como ha quedado demostrado, este objetivo también se ha logrado, siendo capaces de mover el robot en la simulación pasando mensajes desde nodos de ROS y, de igual manera, siendo capaces de transmitir la información de la simulación a través de los tópicos de éste.

Por último, fue muy difícil trabajar con un robot conociendo el precio del mismo, fue un poco intimidante ya que desconocía mucho del tema, a mi parecer fue tanto emocionante como frustrante el querer intentar nuevas cosas, con el miedo de no perder la configuración realizada, aprendí cosas que desconocía en un 100% y algo que me puedo llevar es que trabaje mucho con Linux, fue bueno tomar un reto ya que de ellos es donde más se aprende.

5. Referencias bibliográficas

[1] "Baxter – Redefining Robotics and Manufacturing – Rethink Robotics". Rethink Robotics.

[2] "Baxter". http://sdk.rethinkrobotics.com/wiki/Hardware_Specifications Consultado el 21 de Octubre de 2018.

[3] Haughee, E. (2013). Instant Sublime Text Starter. Packt Publishing Ltd.

[4] “ROS Melodic Morenia”. http://wiki.ros.org/melodic. Consultado el 14 de Septiembre de 2018

[5] “Target Platforms”. http://www.ros.org/reps/rep-0003.html, Consultado el 14 de Septiembre de 2018

[6] “ROS indigo”. http://wiki.ros.org/indigo, Consultado el 14 de Septiembre del 2018

[7] “ROS melodic migration”, http://wiki.ros.org/melodic/Migration, Consultado el 14 de Septiembre de 2018

[8] “Baxter Research Robot Software Developers Kit (SDK)”. http://sdk.rethinkrobotics.com/wiki/Baxter_Research_Robot_Software_Developers_Kit_(SDK). Consultado el 14 de Septiembre de 2018

[9] “Why Gazebo”. http://gazebosim.org/. Consultado el 14 de Septiembre de 2018

[10] “MoveIt!”. https://moveit.ros.org/. Consultado el 14 de Septiembre de 2018

[11] “Concepts”. https://moveit.ros.org/documentation/concepts/ Consultado el 14 de Septiembre de 2018

[12] “Rviz”. http://sdk.rethinkrobotics.com/wiki/Rviz Consultado el 14 de Septiembre de 2018

[13] “Phyton 3”. https://www.python.org/download/releases/3.0/ Consultado el 14 de Septiembre de 2018

[14] “Baxter pick and learn”. http://sdk.rethinkrobotics.com/wiki/Baxter_Pick_And_Learn Consultado el 14 de Noviembre de 2018

[15] “OpenCV”. https://www.opencv.org/ Consultado el 22 de Noviembre de 2018

[16] “ROS ros.h”. http:/ /wiki.ros.org /ROS/Tutorials /WritingPublisherSubscriber%28c%2B%2B%29 Consultado el 22 de Noviembre de 2018

[17] “ROS cv_bridge.h”. http: //docs.ros.org /jade/api/cv_bridge/ html/c++/classcv__bridge_1_1CvImage.html#details Consultado el 22 de Noviembre de 2018

[18] “ROS sensor_msgs”. http:/ /docs.ros.org/ diamondback/ api/ sensor_msgs/ html/namespacesensor__msgs.html#_details Consultado el 22 de Noviembre de 2018

[19] “ROS baxter_core_msgs”. http://wiki.ros.org/baxter_core_msgs Consultado el 22 de Noviembre de 2018

[20] “OpenCV imgproc.hpp”. https:/ /docs.opencv.org /3.0.0/d7/dd4/ classcv_1_1GeneralizedHough.html#details Consultado el 22 de Noviembre de 2018

[21] “OpenCV HighGUI”. https:/ /docs.opencv.org/ 2.4/modules/ highgui/doc/highgui.html Consultado el 22 de Noviembre de 2018

Universidad Politécnica de Puebla
Ingeniería en Informática

José Alberto Figueroa García
Antonio Benítez Ruiz
Rebeca Rodríguez Huesca