Programación del prototipo de un robot clasificador

Yaling Zhang

Resumen–En la actualidad el uso de robots está cada vez más presente en la vida cotidiana, desde robots de uso habitual para tareas domésticas (por ejemplo *Roomba Vacuum Cleaning*) hasta robots industriales que realizan tareas complejas o repetitivas sustituyendo el trabajo humano y además están apareciendo robots humanoides que ayudan a las personas dependientes.

En el presente artículo se presenta un caso real de la programación de un robot autónomo; desde el ensamblado *hardware* hasta la automatización a través del *software* **ROS** (*Robot Operating System*).

Parablas claves– ROS, *Kinect*, *Publisher*, *Subscriber*, cinemática, servos, filtros morfológicos, HSV, luminancia, onda de luz.

Abstract– Nowadays the use of robots is constantly increasing in our daily life, not only for household work, but also for industrial field to perform complex or recurring tasks easing the human labour. Moreover, humanoid robots are developed to help dependant people.

In this article, a real autonomous robot programming case will be seen; beginning by the hardware assembly and ending by the automation through **ROS** (Robot Operating System).

Keywords– ROS, Kinect, Publisher, Subscriber, kinematics, servos, morphological filters, HSV, lightness, lightwave.

1 Introducción

Este proyecto consiste en estudiar el funcionamiento y manipulación de un brazo robot. Para el dicho estudio, se ha optado por utilizar el robot ***PhantomX Pincher Arm*** [14] [13] [4] mediante la plataforma **ROS**, ***Robot Operating System*** [15], la justificación de la cual se detallará en la sección de estado de arte 3 y herramientas 6.

El presente documento está estructurado de la siguiente manera: una explicación de los objetivos para el alcance de este estudio (sección 2); en qué cosiste ROS (sección 3); la metodología utilizada (sección 4); ensamblado *hardware* (sección 5); presentación de las herramientas (sección 6); el desarrollo de la cinemática (sección 7) y la cinemática inversa (sección 8); como alcance opcional de este proyecto: visión sobre la *Kinect* (sección 9) y en consecuencia trabajo futuro (sección 10) y finalmente las conclusiones (sección 11), referencias utilizadas para el documento y el apéndice.

• E-mail de contacto: yaling.zhang@e-campus.uab.cat
• Mención realizada: Computación
• Trabajo tutorizado por: Dr. Ricardo Toledo Morales (Ciencias de la Computación) y Dr. Joan Oliver Malagelada (Departamento de Microelectrónica y sistemas electrónicas)
• Curs 2015/16

2 Objetivos

El objetivo de este proyecto consiste en investigar el comportamiento y funcionamiento de los robots autónomos industriales para la clasificación e implementar el prototipo al robot académico ***PhantomX Pincher Arm*** [14] [13] [4]. Para ello se define los subobjetivos siguientes: analizar las partes del sistema funcional; estudio profundo del sistema **ROS** para manipular el brazo robot; llevar el conocimiento al desarrollo físico del robot mencionado: posicionar el brazo robot en coordenadas deseadas y realizar movimientos automáticos.

Finalmente, se hará introducción al mundo de la visión mediante *Kinect*. Éste forma parte de un objetivo ambicioso de automatizar la detección de la posición del objeto mediante visión por computación. Dado que el proyecto es acotado por el tiempo, este objetivo está fuera del alcance del proyecto, sin embargo, es una buena línea de mejora.

3 Estado de arte

ROS (Robot Operating System) [15] [22] es una plataforma *open-source* flexible para desarrollar *software* de robots. Se puede entender ROS como un sistema distribuido, donde los nodos se comunican entre ellos mediante *Publishers* y *Subscribers*.

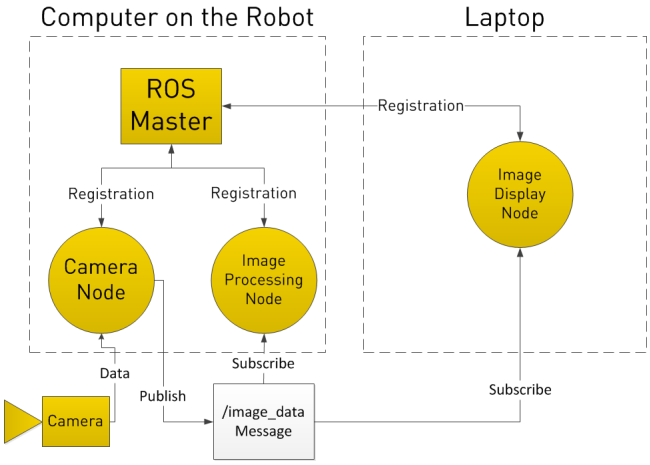

Fig. 1: Esquema de funcionamiento de nodos de ROS

Hoy en día, ROS se ha convertido en una plataforma común donde se comunican y colaboran los ingenieros de unos campos con los otros, ya sean de *hardware* o *software*; el cual está incrementando la innovación en robótica en el mundo.

4 Metodología

Después de estudiar los diferentes modelos de ciclos de vida del *software* [19], se ha seleccionado el modelo incremental como metodología para el proyecto, el cual consiste en añadir funcionalidades implementadas y testeadas a un modelo base diseñado.

La ventaja de usar éste método es que se genera el *software* de forma rápida y al ser flexible reduce el coste en los cambios de requisitos y de esta manera es más fácil gestionar los riesgos.

Éste método es seleccionado porque es el que se adapta más al presente proyecto dado sus objetivos. La inconveniencia del método es que, si falla el modelo base, el coste de reparación y el tiempo requerido en el caso es elevado.

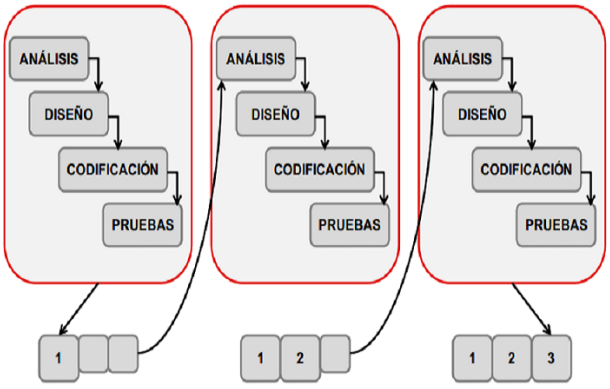

Fig. 2: Esquema modelo incremental

5 Ensamblaje del robot

El proyecto se ha empezado por el montaje del brazo robot ***PhantomX Pincher Arm*** [14]. El motivo de la realización de esta parte es adquirir el conocimiento *hardware* sobre el robot; al tener consideración de las limitaciones del *hardware* ayuda a comprender mejor la integración del conocimiento *hardware* y *software*.

El kit del robot *PhantomX Pincher Arm* está formado básicamente por los servos AX-12A *Dynamixel Actuators*, construcción ABS, el controlador *Arbotix Robocontroller* y el componente *Parallel Gripper*. El resultado de este montaje se muestra en la Figura 3.

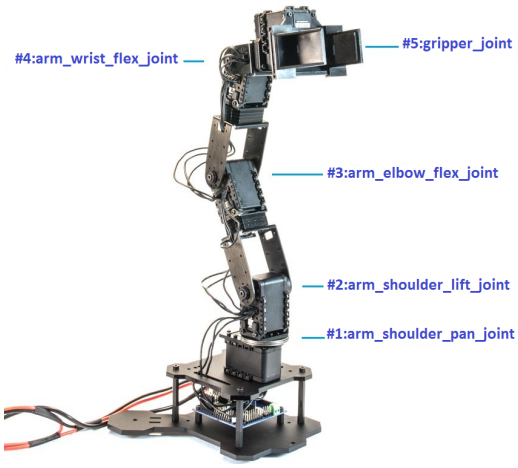

Fig. 3: IDs de PhantomX Pincher Arm

Los números que se muestran en la figura son los identificadores únicos para controlar los servos. Estos identificadores se pueden configurar con *arbotix_terminal* que está explicado en la sección de herramientas 6.

Los pasos más significativos del montaje se puede localizar en la sección del apéndice 11 y el video del ensamblado del robot se puede encontrar en el siguiente link [21].

6 Herramientas

El proyecto tiene una finalidad de estudiar el comportamiento de un robot autónomo. Por ello, para la parte *software* se utilizó ROS [15] como *framework* y ***Python*** para la programación. El *hardware* utilizado es el brazo robot *PhantomX Pincher Arm* [14] y finamente para la visión, la cámara *Kinect* [3].

Para el entorno, se ha configurado *Arduino*, ***Pypose*** y ***Arbotix***. En las siguientes secciones se entrarán algunos detalles de las configuraciones mencionadas.

6.1 PhantomX Pincher Arm

El brazo robot *PhantomX Pincher Arm* tiene 5 grados de libertad: 4 de rotación y 1 prismático (4R1P) y es simple de simular con el *TurtleBot ROS robot platform*. Algunas de sus características destacadas son: actuadores *Dinamixel AX-12A*, construcción robusta ABS, *Arbotix Robocontroller* para procesamiento a bordo y soporte de montaje para cámaras y sensores. [4].

6.2 Kinect

La *Kinect* o *Kinect XBOX 360* [4] es una cámara que ha sido fabricada por *Microsoft*. La *Kinect* permite obtener una escena 3D cuya característica de campo de visión es la siguiente [5]:

- Campo de visión horizontal de 57 grados.
- Campo de visión vertical de 43 grados.
- Rango de inclinación física de $\pm$ 27 grados.
- Mayor resolución 640 x 480.
- Rango de profundidad del sensor de [1.2, 3.5] metros.

Fig. 4: Kinect

6.3 ROS

ROS, ya citado en la sección 3, facilita una serie de herramientas, librerías, abstracción de hardware, control de dispositivos de bajo nivel, etc., que simplifica la complejidad de programación del comportamiento de robots. Por estas ventajas ROS es escogido para desarrollar el presente proyecto.

Para este proyecto se utiliza la versión *Indigo* de ROS ya que es la última versión estable. Las librerías de ROS están adaptadas para varios sistemas operativos y en este proyecto se utiliza el sistema *Linux Mint 17.1 Rebecca* que tiene equivalencia con *Ubuntu 14.04 Trusty Tahr* y es compatible con la versión *Indigo*.

En cuanto a ROS *Indigo*, hay que destacar en concreto, el módulo *TurtleBot ROS robot platform* que será ampliamente utilizada a lo largo del proyecto.

6.4 Arduino

El *Arduino* [12] es una plataforma electrónica *open-source* basada en un *hardware* fácil de usar y un entorno de desarrollo diseñada para facilitar los proyectos multidisciplinares. En este proyecto se usa para cargar el *Pypose*, *Arbotix*, *ROS* o cualquier módulo *software* se quiere incorporar al controlador del brazo robot *Phamton X pincher*. Las librerías utilizadas: *Arbotix* y *Pypose* se pueden descargar en *Trossen Robotics* [4].

6.4.1 Arbotix

Arbotix es un paquete de librerías para la placa del robot *Phantom X Pincher*. Entre todos los comandos, hay que destacar *arbotix_terminal* y *arbotix_gui* para configurar el robot. El *arbotix_terminal* sirve para establecer los identificadores de los servos, como se explica en la sección 5 y listar los servos conectados. El *arbotix_gui*, verifica mediante *ROS*, el correcto conexionado y funcionamiento de los servos, tal como se muestra en la imagen 5.

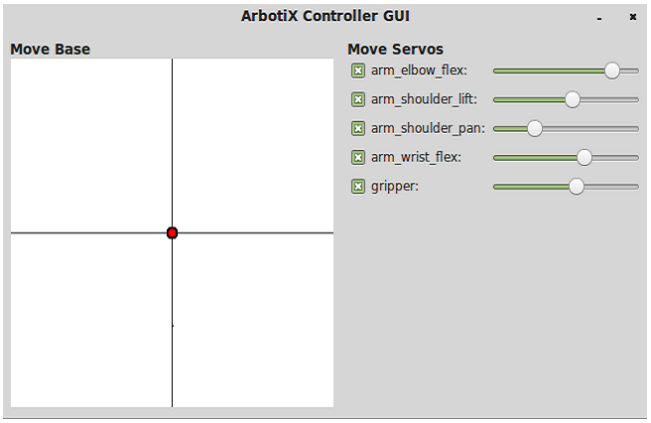

Fig. 5: Ejecución de *Arbotix Gui*

6.4.2 Pypose

Una trayectoria es definida por una secuencia de posiciones en *Pypose*. Un parámetro muy importante es el delta-T que se ve en la Figura 6; ya que indica la velocidad del movimiento. Por lo tanto, con valores muy bajos ($\leq$ 500) puede ocasionar daños al robot físico.

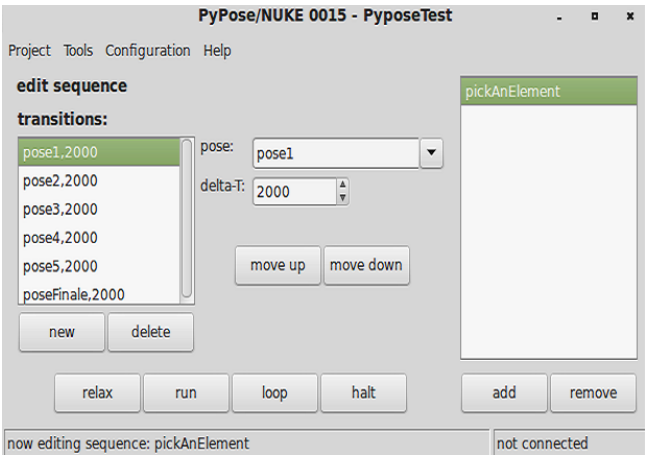

Fig. 6: Definir una trayectoria en Pypose.

La finalidad de utilizar *Pypose* es investigar la manera de programar una trayectoria; los parámetros a considerar y controlar, etc. Como resultado de la aplicación de este módulo, se ha planificado una repetición de movimientos (*loop*), tal como se ve en el vídeo [6] y una trayectoria de 5 posiciones que traslada un objeto de una posición a otra tal como se ve en el vídeo [8].

7 Cinemática

Después de la investigación realizada, se ha obtenido la información necesaria para realizar la implementación de la cinemática [24]. Como ya se ha comentado, ROS funciona

como un sistema distribuido, en el cual los nodos se comunican a través de *publishers* y *subscribers*, tal como se ve en la Figura 7.

```
Published topics:
 * /arm_controller/follow_joint_trajectory/status [actionlib_msgs
/GoalStatusArray] 1 publisher
 * /gripper_controller/gripper_action/result [control_msgs/Grippe
rCommandActionResult] 1 publisher
 * /gripper_joint/command [std_msgs/Float64] 1 publisher
 * /joint_states [sensor_msgs/JointState] 1 publisher
 * /rosout [rosgraph_msgs/Log] 2 publishers
 * /gripper_controller/gripper_action/feedback [control_msgs/Grip
perCommandActionFeedback] 1 publisher
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /arm_controller/follow_joint_trajectory/feedback [control_msgs
/FollowJointTrajectoryActionFeedback] 1 publisher
 * /arm_controller/follow_joint_trajectory/result [control_msgs/F
ollowJointTrajectoryActionResult] 1 publisher
 * /diagnostics [diagnostic_msgs/DiagnosticArray] 1 publisher
 * /gripper_controller/gripper_action/status [actionlib_msgs/Goal
StatusArray] 1 publisher

Subscribed topics:
 * /gripper_controller/gripper_action/cancel [actionlib_msgs/Goal
ID] 1 subscriber
 * /arm_shoulder_pan_joint/command [std_msgs/Float64] 1 subscribe
r
 * /arm_controller/follow_joint_trajectory/cancel [actionlib_msgs
/GoalID] 1 subscriber
 * /gripper_controller/gripper_action/goal [control_msgs/GripperC
ommandActionGoal] 1 subscriber
 * /gripper_joint/command [std_msgs/Float64] 1 subscriber
 * /joint_states [sensor_msgs/JointState] 1 subscriber
 * /rosout [rosgraph_msgs/Log] 1 subscriber
 * /arm_wrist_flex_joint/command [std_msgs/Float64] 1 subscriber
 * /arm_controller/follow_joint_trajectory/goal [control_msgs/Fol
lowJointTrajectoryActionGoal] 1 subscriber
 * /arm_shoulder_lift_joint/command [std_msgs/Float64] 1 subscrib
er
 * /arm_controller/command [trajectory_msgs/JointTrajectory] 1 su
bscriber
 * /arm_elbow_flex_joint/command [std_msgs/Float64] 1 subscriber
```

Fig. 7: *rostopic list -v*

7.1 Resultado de la cinemática

Una vez implementada la cinemática, se ha realizado dos tests: uno que verifica el funcionamiento del *gripper* y otro que abarca todas las articulaciones.

7.1.1 Test del *gripper*

El test del *gripper*, realiza un bucle de abrir y cerrar según la apertura máxima y mínima predefinida y la velocidad está controlada por un valor definido para *rospy.sleep()*. El resultado de este test se puede ver en la referencia [9]. Los nodos activos y la comunicación de ellos, se ve en la gráfica 8.

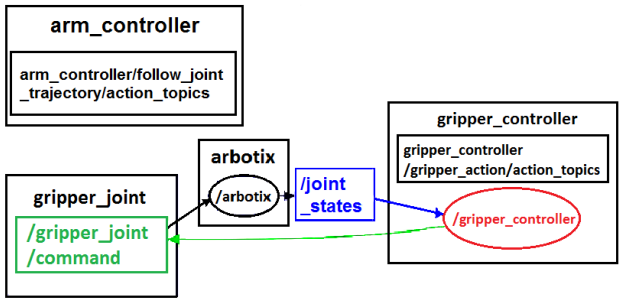

Fig. 8: *Rqt graph* del *gripper*

En la gráfica del movimiento 9 se ve un ascenso y descenso. Esta gráfica tiene sentido porque el *gripper* va abriendo y cerrando. Cuando se abre el *gripper*, la gráfica asciende y viceversa.

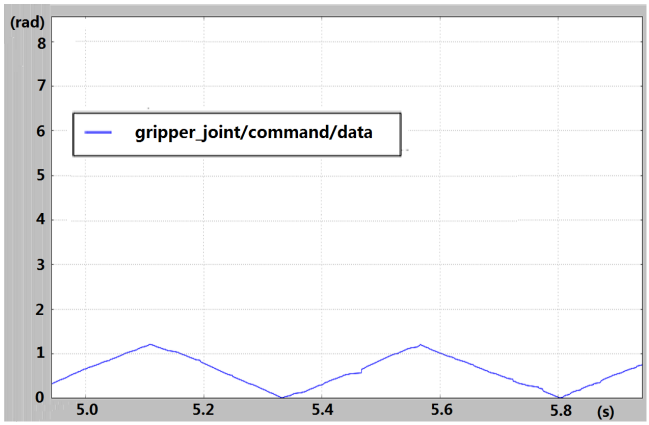

Fig. 9: rqt_plot del *gripper*

7.1.2 Test de las articulaciones

En el test de todas las articulaciones, el *gripper* sigue realizando el mismo movimiento que la sección anterior y los de más servos giran hacia arriba y abajo, según el giro máximo y mínimo predefinido. El resultado de este test se puede ver en la referencia [7]. En este caso hay más nodos activos que el caso anterior, como se ve en la segunda gráfica de la sección de apéndice.

En la gráfica de movimientos 10 se ve el *gripper* con el mismo movimiento y el movimiento de los otros servos se solapan en una ya que el *step* o el incremento de giro es el mismo para todos los servos y es constante.

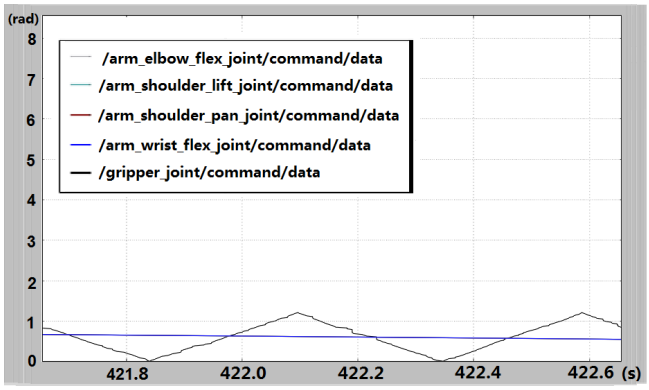

Fig. 10: rqt_plot de todas las articulaciones

8 Cinemática Inversa

El problema de la cinemática inversa consiste en calcular, el conjunto de ángulos de articulación que se debe aplicar a la cinemática implementada; dada la posición y la orientación de las articulaciones.

Este problema no es lineal. La existencia de la solución depende del espacio de trabajo del brazo. En general, el espacio de trabajo es el volumen de espacio que puede ser alcanzado por el robot. Para que exista una solución, el punto de destino debe estar dentro del espacio de trabajo. Este

espacio es limitado por las restricciones físicas del brazo robot. Esta información de las restricciones se encuentra en el fichero de la especificación del robot.

Otro posible problema que se puede encontrar al resolver la cinemática inversa es el de múltiples soluciones, ya que cualquier posición en el interior del espacio de trabajo se puede alcanzar con cualquier orientación. En este caso, para simplificar el problema, se elige la primera solución; aunque se podría escoger la solución más cercana a la posición de las articulaciones [23].

8.1 OpenRAVE

OpenRAVE [2] es una librería que proporciona un entorno para *testing*, desarrollo e implementación de algoritmos de *motion planning* para aplicaciones de robots reales. Contiene comandos para trabajar con planificadores y el tiempo de ejecución es suficiente pequeño para ser utilizado en controladores.

En este proyecto, *OpenRAVE* se utiliza en la resolución del problema de la cinemática inversa. En concreto, se aplica en la parte de análisis de la cinemática y la información geométrica relacionada con *motion planning* que facilita la integración con el sistema robótica ROS.

La instalación de la librería requiere una serie de pasos, los cuales están especificados en la referencia [1].

8.2 Solución de la cinemática inversa

La solución de la cinemática inversa para el brazo robot, *IKFast*, es generada mediante la combinación de ROS, la especificación de restricciones del brazo robot y la librería *OpenRAVE*. Se debe indicar que este método de generador de *IKFast* no funciona con robots de más de 7 grados de libertad [18].

La solución que proporciona *IKFast* se invoca espefícando la matriz de rotación del estado actual del brazo robot y la posición final que se desea alcanzar:

$$\begin{pmatrix} cos\theta & -sin\theta & 0 & P_x \\ sin\theta & cos\theta & 0 & P_y \\ 0 & 0 & 1 & P_z \end{pmatrix}$$

8.3 Resultado

El resultado de la cinemática inversa es un vector de ángulos, los cuales indican la apertura de las articulaciones. Aplicando este resultado a la cinemática implementada en la sección anterior, se ha visto que el brazo robot es capaz de alcanzar la posición objetiva automáticamente dado la posición. El vídeo de este resultado se ve en el enlace [20].

Para lograr el resultado visto en el vídeo, se ha definido 6 posiciones a alcanzar en la trayectoria para suavizar el movimiento. Este movimiento se puede automatizar realizando una interpolación de puntos entre la coordenada inicial y la coordenada final, e invocando la cinemática inversa por cada uno de los puntos. Cuantos más puntos se define, más suave es el movimiento, pero también más coste de computación. Ésta interpolación se puede implementar con algoritmos basados en la *Jacobiana*, interpolación polinómica o algoritmos similares.

9 Visión con Kinect

El objetivo de esta sección es introducir la implementación de la visión a través de la *Kinect*. La realización de esta parte viene dada por la detección automática de la posición mediante el análisis de las imágenes obtenidas de la *Kinect*. Se ha comenzado a trabajar para la incorporación de la visión y se ha conseguido algunos objetivos, pero el desarrollo completo de esta parte queda fuera del alcance de este proyecto debido a su complejidad, como se ha mencionado en la sección de objetivos 2. En esta sección se verán los detalles y sus respectivas líneas de continuación.

9.1 El entorno de la *Kinect*

Para empezar a usar la *Kinect* al entorno *Linux Mint 17 Rececca*, es necesario instalar el controlador de la *Kinect*. Existen varias opciones; pero este proyecto se ha utilizado la librería *freenect* porque es la más nueva [10] y la que más que más se ajusta al entorno y la necesidad.

El funcionamiento de la *Kinect* se puede ver en la siguiente Figura 11.

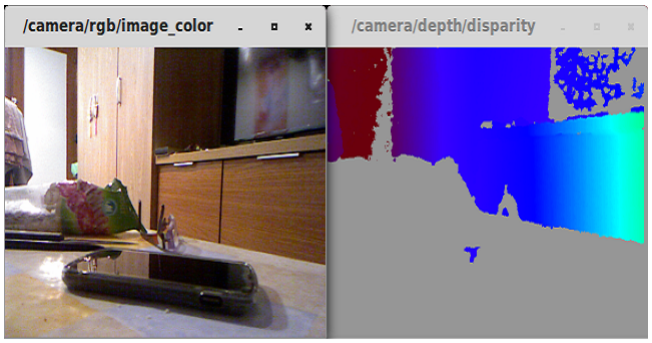

Fig. 11: Muestra de una captura de imagen con la *Kinect*

En la lista de nodos de *subscribers* y *publishers* 12 se añadieron los nodos correspondientes a la *Kinect*.

```
develop/turtlebot_arm_bringup/launch $ rostopic list
/camera/debayer/parameter_descriptions
/camera/debayer/parameter_updates
/camera/depth/camera_info
/camera/depth/disparity
/camera/depth/image
/camera/depth/image/compressed
/camera/depth/image/compressed/parameter_descriptions
/camera/depth/image/compressed/parameter_updates
/camera/depth/image/compressedDepth
/camera/depth/image/compressedDepth/parameter_descriptions
/camera/depth/image/compressedDepth/parameter_updates
/camera/depth/image/theora
/camera/depth/image/theora/parameter_descriptions
/camera/depth/image/theora/parameter_updates
/camera/depth/image_raw
/camera/depth/image_raw/compressed
/camera/depth/image_raw/compressed/parameter_descriptions
/camera/depth/image_raw/compressed/parameter_updates
/camera/depth/image_raw/compressedDepth
/camera/depth/image_raw/compressedDepth/parameter_descriptions
/camera/depth/image_raw/compressedDepth/parameter_updates
/camera/depth/image_raw/theora
/camera/depth/image_raw/theora/parameter_descriptions
/camera/depth/image_raw/theora/parameter_updates
/camera/depth/image_rect
/camera/depth/image_rect/compressed
/camera/depth/image_rect/compressed/parameter_descriptions
/camera/depth/image_rect/compressed/parameter_updates
/camera/depth/image_rect/compressedDepth
/camera/depth/image_rect/compressedDepth/parameter_descriptions
/camera/depth/image_rect/compressedDepth/parameter_updates
/camera/depth/image_rect/theora
/camera/depth/image_rect/theora/parameter_descriptions
/camera/depth/image_rect/theora/parameter_updates
/camera/depth/image_rect_raw
/camera/depth/image_rect_raw/compressed
/camera/depth/image_rect_raw/compressed/parameter_descriptions
/camera/depth/image_rect_raw/compressed/parameter_updates
```

Fig. 12: Los nodos asociados a la *Kinect*

Entre todos los nodos, se interesa, */camera/rgb/image_raw*, el cual permite obtener un mapa RGB de la imagen escaneada por la *Kinect*. Con este dato se puede implementar la parte de detección de objetos [11].

9.2 OpenCV

En la implementación de la visión, se ha utilizado la librería *OpenCV* [16]; una librería libre para procesamiento de imágenes. En ROS, las imágenes se publican en formato *sensor_msgs/Image message* y para integrar este dato en la librería *OpenCV* se usa *CvBridge* que es una librería de interfaz entre ROS y *OpenCV* com se ve en la Figura 13.

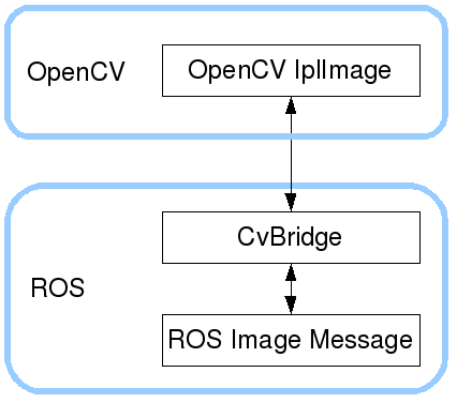

Fig. 13: Interfaz entre ROS y *OpenCV*: *CvBridge*

Esta función se puede invocar con *cv_image = bridge.imgmsg_to_cv(image_message, encoding)* donde *encoding* es el tipo de dato. En general, *CvBridge* hace la conversión de datos especificando el *encoding* con los siguientes parámetros:

- mono8: CV_8UC1, imagen en escala de gris.
- mono16: CV_16UC1, 16-bit imagen en escala de gris.
- bgr8: CV_8UC3, imagen en color con valores en orden azul-verde-rojo (BGR).
- rgb8: CV_8UC3, imagen en color con valores en orden rojo-verde-azul (RGB).
- bgra8: CV_8UC4, imagen BGR con canal *alpha*.
- rgba8: CV_8UC4, imagen RGB con canal *alpha*.

9.3 Visión implementada

La detección de objetos es basada en colores predefinidos. Para este fin, se define los rangos de valores para los colores BGR (límites inferiores y superiores) [17]. De esta manera filtrando los píxeles por rangos se obtiene una imagen binaria o segmentada, la cual indica la zona donde se encuentra el objeto a detectar. En un caso concreto, los límites son:

- *[17, 15, 100] y [50, 56, 200]* para el color rojo.
- *[86, 31, 4] y [220, 88, 50]* para el azul.
- *[0,102, 102] y [51,255, 255]* para el amarillo.

Los valores tienen el siguiente significado, tomando el ejemplo del color rojo especificado: $R \geq 100, B \geq 15 y G \geq 17$ para límite inferior, y $R \leq 200, B \leq 56$ y $G \leq 50$ para el límite superior. La imagen segmentada usando el caso anterior y aplicando los límites de colores mencionados se puede ver en la imagen 14:

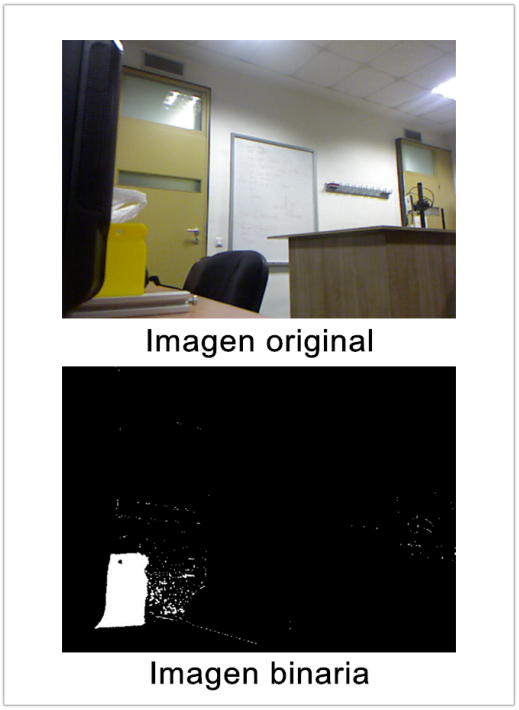

Fig. 14: Imagen binaria de detección de objeto amarillo

Como se ve, en la imagen aparecen islas o ruido y esta información no es útil para el resultado. Para resolver este problema se aplica el filtro morfológico *open* que se verá en la siguiente subsección 9.3.1.

9.3.1 El filtro morfológico *open*

Los filtros morfológicos son filtros cuyo valor de cada píxel de salida depende del valor del píxel a tratar y sus vecinos (píxeles que quedan dentro de una ventana alrededor del píxel a tratar).

Antes de entrar en detalle sobre el filtro *open* se debe explicar unos conceptos previos.

- **Elemento estructurante:** Define el tamaño de la ventana y la forma de ésta en la que se aplicará la operación morfológica [25].
- **Erosión:** El píxel de salida es el mínimo de los píxeles dentro de la ventana definida por el elemento estructurante [25].

  Su fórmula viene dada por:

  $$\varepsilon_b x[n] = \bigwedge_{k=-\infty}^{\infty} (x[k] - b[k-n])$$

  Donde b es el elemento estructurante, x es la imagen y las operaciones $\bigvee$ y $\bigwedge$ son operaciones supremos e ínfimos respectivamente.
- **Dilatación:** El píxel de salida corresponde al máximo de los píxeles dentro de la ventana definida por el elemento estructurante.

$$\delta_b x[n] = \bigvee_{k=-\infty}^{\infty}(x[k] + b[n-k])$$

Una vez explicado los conceptos anteriores, la operación *Open* es una combinación de una erosión seguido de una dilatación utilizando el mismo elemento estructurante.

$$y = (x \ominus b) \oplus b = \gamma_b(x)$$

Este filtro es utilizado para eliminar las islas (ruido) creado al aplicar el rango de colores. Esto es posible ya que, al aplicar primero una erosión, el ruido es eliminado por completo y al aplicar la dilatación, el objeto detectado recupera la deformación ocurrida por la erosión.

Por último, en caso de que sea necesario, se aplica un rellenado *filling* para rellenar posibles agujeros que pueda tener el objeto detectado, tal como se puede ver en la imagen 15.

Fig. 15: Imagen rellenada

9.4 Resultado de la visión

Tras tests realizados sobre la visión implementada, se ha visto que el resultado se ve afectado por la luminancia de la imagen capturada. Se ha probado otros espacios de colores, como **HSV** (*Hue Saturation Value*).

En consecuencia, para resolver el problema comentado, se debe hacer investigaciones profundas: usando el espacio de color HSL, *Hue Saturation Lightness*, que proporciona la información de la luminancia de la imagen y encontrar la relación entre el tipo de intensidad y las escalas de colores. Dicho en otras palabras, obtener un mapa de correspondencia de escala de colores respecto a la curva de intensidad de luminancia.

La mejora usando HSV no se es notable en el resultado. El motivo del cual se justifica que la **onda de luz** varia durante el trascurso del día, por lo tanto, la intensidad y la escala del color cambia a lo largo del día.

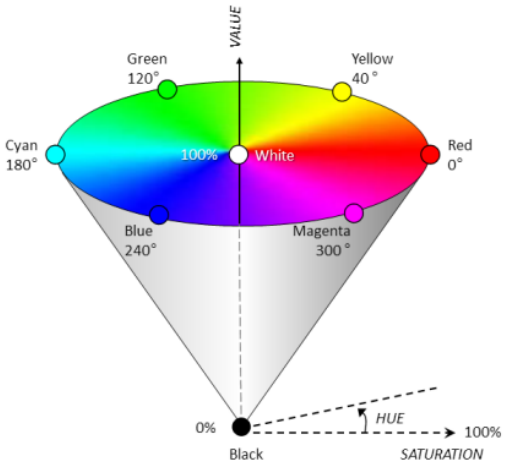

Fig. 16: Espacio de color HSV

Por la anterior explicación, si la imagen ha sido capturada de día, el color se observa, por ejemplo, rojo; pero si es capturada por la tarde, el color es rosa intenso. Se puede resolver este problema de manera fácil, admitiendo más tolerancia o flexibilidad a los rangos de colores mencionados anteriormente, pero, en consecuencia, la imagen segmentada obtenida contiene más ruido, en otras palabras, elementos del fondo que no es el objeto interesado.

En consecuencia, para resolver el problema comentado, se debe hacer investigaciones profundas: usando el espacio de color HSL, *Hue Saturation Lightness*, que proporciona la información de la luminancia de la imagen y encontrar la relación entre el tipo de intensidad y las escalas de colores. Dicho en otras palabras, obtener un mapa de correspondencia de escala de colores respecto a la curva de intensidad de luminancia.

10 Trabajo futuro

El presente proyecto queda cerrado con la investigación analizada hasta el momento. Sin embargo, quedan abiertas las líneas de continuación para investigaciones futuras, tal como ya se ha comentado en las secciones anteriores. Las líneas de mejoras se clasifican en 4 clases:

- Cinemática inversa: Para lograr el resultado que se muestra en la sección de cinemática inversa, se ha definido varias posiciones manualmente para formar una trayectoria y así, suavizar el movimiento. Este movimiento se puede automatizar con algoritmos basados en la *Jacobiana* o similares como ya se ha mencionado.
- Visión: En la sección de visión 9 se ha mencionado varias mejoras posibles, que se resumen en los siguientes puntos:
  - Algortimo sofisticado para resolver el problema de dependencia de la **onda de luz**, como se ha comentado en la sección 9.4.
  - Detección de objetos por formas, en lugar de colores; mediante primero, la aplicación de filtro *canny* para obtener los contornos de la imagen y después, utilizando los contornos como características y detectar el objeto deseado usando

la transformada de *Hough* o CNN (*Convolution Neuronal Net*).

- ICP: Una vez resuelto el problema de onda de luz, otra mejora pendiente es la detección de la posición de los objetos. La *Kinect* permite reconstruir el mapa 3D a través de captura de *Point Clouds*. Para el procesamiento de los datos *Point Clouds* se puede aplicar el algoritmo *ICP*(*Iterative closest point*). De esta manera combinándolo con el análisis de la imagen se obtiene la coordenada 3D de la posición del objeto. La coordenada obtenida en este caso, es respecto a la *Kinect*, por lo tanto, se debe realizar una transformada de coordenada a la coordenada del robot, conociendo la relación de distancia entre ambos 17.
- Escena no estática: En este trabajo se ha adaptado simplificaciones al problema real, considerando la escena estática. Sin embargo, en el mundo real no siempre los objetos son estáticos, como por ejemplo los clasificadores de la cinta transportadora. Por consiguiente, se espera la implementación de casos más complejos como la escena dinámica para investigaciones futuras.

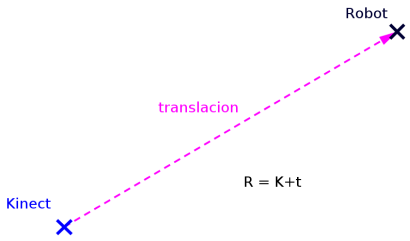

Fig. 17: Transformada de coordenada

11 Conclusions

En síntesis, el objetivo de este proyecto: programación del robot autónomo, *Phantom X Pincher Arm*, para la clasificación de objetos, queda cerrado. Actualmente, el brazo robot es capaz de realizar movimientos de forma automática dada una posición. Una vez logrado el punto mencionado, se ha abierto varias líneas de continuación posible para investigaciones futuras, como se ha mencionado anteriormente.

En la realización de este proyecto, se ha visto el funcionamiento del robot autónomo desde su ensamblado *hardware* hasta la programación del *software*, además de comprender la visión que esconde detrás y las limitaciones al llevarla a la práctica.

Por último, este proyecto ha introducido al autor a la plataforma ROS, la cual es altamente utilizado en el mundo de la robótica, para las posibles investigaciones personales futuras al presente robot y proyectos personales.

Agradecimientos

El autor quiere aprovechar este espacio para expresar sus agradecimientos al Dr. Ricardo Toledo Morales por haber dirigido y supervisado el proyecto. También quiere agradecer al Dr. Joan Oliver Malagelada por la idea original y la orientación del proyecto. Finalmente, a su pareja por todos los apoyos y ánimos.

Referències

[1] Openrave instalación. http://www.aizac.info/installing-openrave0-9-on-ubuntu-trusty-14-04-64bit/, Consulta: 1 de enero de 2016.

[2] Openrave sitio oficial. http://openrave.org/, Consulta: 1 de enero de 2016.

[3] Kinect. https://msdn.microsoft.com/en-us/library/hh438998.aspx, Consulta: 1 de octubre de 2015.

[4] Phantomx pincher robot arm kit v2. http://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.aspx, Consulta: 1 de octubre de 2015.

[5] Kinect developers. http://www.kinectfordevelopers.com/es/2012/03/01/diferencias-entre-kinect-xbox-360-y-kinect-for-windows/, Consulta: 12 de noviembre de 2015.

[6] Video: Demo de la trajectoria mediante pypose. https://drive.google.com/open?id=0B1yhzSs3E5n2VVpKREtkSXFEZ00, Consulta: 12 de noviembre de 2015.

[7] Video: Demo del control de las articulaciones mediante script. https://drive.google.com/open?id=0B1yhzSs3E5n2UE1YaUk2Ymc4VlE, Consulta: 12 de noviembre de 2015.

[8] Video: Demo del desplazamiento de un objeto mediante pypose. https://drive.google.com/open?id=0B1yhzSs3E5n2R1JiY2J1MmpEelE, Consulta: 12 de noviembre de 2015.

[9] Video: Demo del gripper. https://drive.google.com/open?id=0B1yhzSs3E5n2VzlFaTZlMWN2NFk, Consulta: 12 de noviembre de 2015.

[10] Kinect: página ros. http://wiki.ros.org/kinect, Consulta: 14 de noviembre de 2015.

[11] Kinect: Ros subscriber list. http://wiki.ros.org/kinect_camera, Consulta: 14 de noviembre de 2015.

[12] Arduino web oficial [en línea]. https://www.arduino.cc/en/Guide/Libraries, Consulta: 15 de octubre de 2015.

[13] Assembly phantomx basic robot arm. http://www.trossenrobotics.com/productdocs/assemblyguides/phantomx-basic-robot-arm.html, Consulta: 15 de septiembre de 2015.

[14] Phantomx pincher arm [en línea]. http://www.trossenrobotics.com/Shared/productdocs/Phantom_Pincher_Arm_Quickstart.pdf, Consulta: 15 de septiembre de 2015.

[15] Ros. http://www.ros.org/about-ros/, Consulta: 15 de septiembre de 2015.

[16] Opencv librería. http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.htm, Consulta: 20 de noviembre de 2015.

[17] Opencv detector de colores. http://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/, Consulta: 28 de noviembre de 2015.

[18] Ikfast librería. http://docs.ros.org/indigo/api/moveit_ikfast/html/doc/ikfast_tutorial.html, Consulta: 4 de enero de 2016.

[19] Ingeniería del software: metodologías y ciclos de vida. https://www.inteco.es/file/N85W1ZWFHifRgUc_oY8_Xg, Consulta: 4 de octubre de 2015.

[20] Video: Demo pick and place. https://drive.google.com/file/d/0B0Mpx0I9LHAsX3hkemhMQXJQWTA/view?usp=sharing, Consulta: 8 de febrero de 2016.

[21] Video: Montaje brazo robot. https://drive.google.com/file/d/0B0Mpx0I9LHAsbFkxcS1WQUxNZ1k/view?usp=sharing, Consulta: 8 de febrero de 2016.

[22] Enrique Fernández Aaron Martinez. *Learning ROS for Robotics Programming*. Packt Publishing Ltd, 1nd edition, 2013.

[23] JOHN J.CRAIG. *ROBÓTICA*. PEARSON PRENTICE HALL, 3nd edition.

[24] George H martins. *Kinematics and Dynamics of Machines*. Waveland press, 2nd edition, 1982.

[25] Pierre Soille. *Morphological image analysis: principles and applications*. Springer Verlag, 2nd edition, 2003.

APÉNDICE

A.1 Fotografías del ensamblado *hardware*

B.2 *Nodos de las articulaciones del brazo robot*

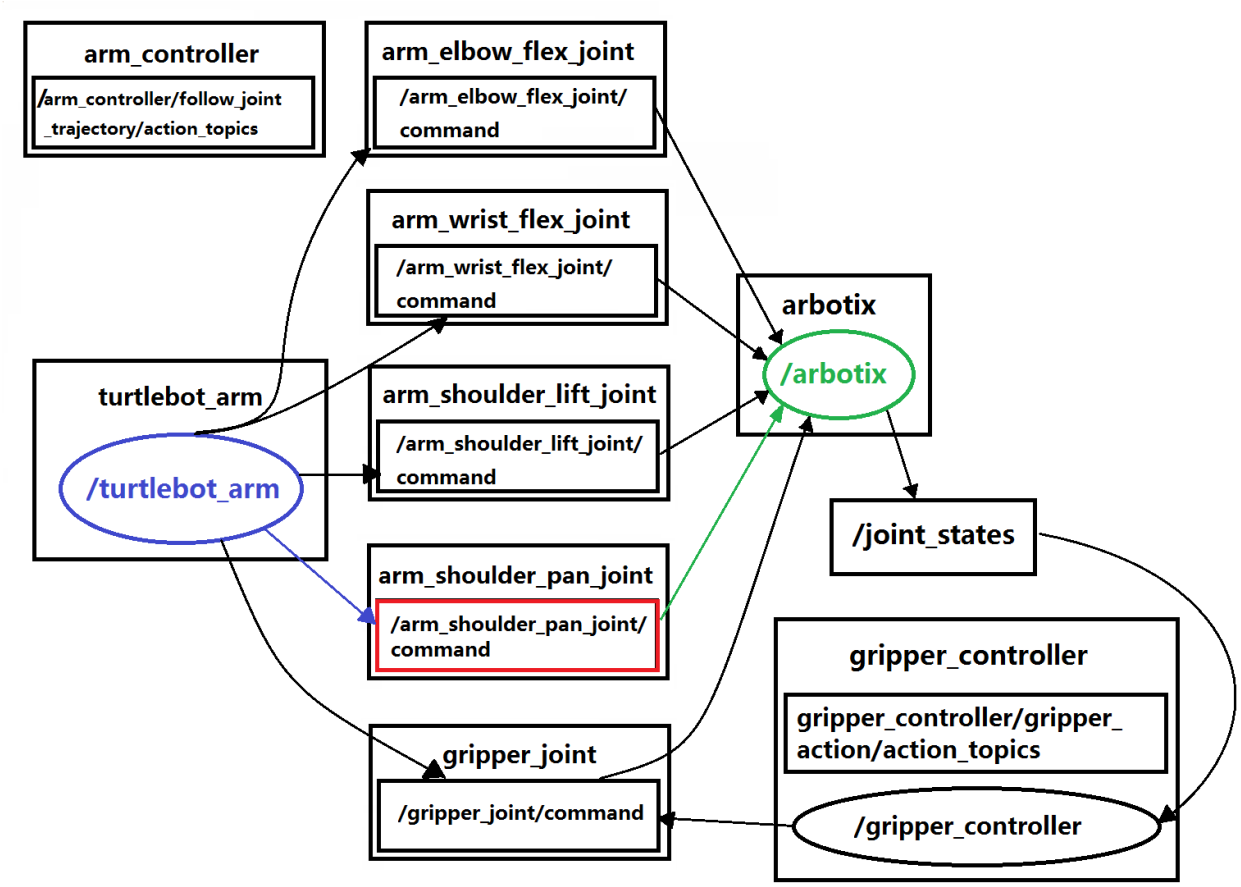

Fig. 18: *Rqt graph* de todas las articulaciones