Proyecto IROBOT

Manual de usuario

Visión por computador

Octubre de 2019

Sara Roos Hoefgeest Toribio
roossara@uniovi.es

ÍNDICE

1 LISTADO DE PAQUETES

- ***aruco_detect:*** paquete de detección de ArUcos.
- ***map_aruco:*** paquete que elabora un mapa de los marcadores presentes en el entorno.
- ***tracking_camera:*** paquete que estima la posición y orientación del robot en función de una sola cámara.
- ***fusing_poses:*** paquete que, a partir de la información de pose estimada por *tracking_camera* para cada una de las 8 cámaras, proporciona una estimación de la pose del robot.
- ***panorama:*** permite realizar un panorama de la chapa en vista cenital y devuelve la posición de los defectos marcados sobre la chapa.
- ***aruco_calibration:*** permite realizar la calibración de una cámara a partir de una serie de imágenes con una tabla ChArUco.
- ***getImages_uEye:*** programa para la captura de imágenes con una cámara uEye XS.

Para más información sobre el funcionamiento básico de ROS, consultar el documento Anexo_ROS adjuntado.

Para todos estos paquetes, es necesario tener instalado ROS Kinetic [1] y OpenCV [2].

2 DETECCIÓN ARUCO

La detección de los marcadores se ha realizado utilizando un paquete para ROS ya desarrollado, *aruco_detect* [3], basado en el módulo Aruco de OpenCV [4].

Este paquete detecta los marcadores presentes en las imágenes procedentes de una cámara. Una vez detectados, publica las posiciones 2D de los vértices de los marcadores detectados en la imagen, así como su identificador según el diccionario escogido.

Dada una imagen con varios ArUco presentes en ella, el proceso de detección devuelve una lista con los marcadores detectados. Cada uno incluye la posición de sus cuatro esquinas en la imagen y su identificador. El detector es capaz de encontrar los marcadores rotados en el entorno.

Con el fin de utilizar el paquete para llevar a cabo la detección de los marcadores, es necesario definir algunos parámetros clave, como son la medida del lado, en metros, y el diccionario al que pertenecen los ArUco que se pretende detectar.

2.1 INSTALACIÓN

Para instalar los paquetes necesarios para la detección de ArUcos, es necesario ejecutar el siguiente comando en el terminal:

```
$ sudo apt-get install ros-kinetic-fiducials
```

2.2 TOPICS A LOS QUE SE SUSCRIBE

Para que el nodo se ejecute correctamente, es necesario subscribirse al *topic* que proporciona las imágenes obtenidas por la cámara.

/nombre_camara/image_raw (sensor_msgs/Image.msg)

El tipo de mensaje Image se detalla en la Tabla 1.

Tabla 1. Parámetros detallados de los mensajes de tipo sensor_msgs::Image

Parámetro	**Tipo**	**Descripción**
header	Header	Metadatos estándar. Utilizados para comunicar datos con marca de tiempo en un sistema de coordenadas en particular.
height	uint32	Alto de la imagen, número de filas.
width	uint32	Ancho de la imagen, número de columnas.
string	encoding	Codificación de los píxeles (canal, orden, tamaño)
step	uint32	Longitud de fila completa en bytes
data	uint8[]	Datos de la matriz imagen. Tamaño: step * filas

2.3 TOPICS EN LOS QUE PUBLICA

La información sobre la localización de los marcadores detectados es publicada en el *topic* correspondiente:

/fiducial_vertices (fiducial_msgs/FiducialArray)

El tipo de mensaje FiducialArray se detalla en la Tabla 2.

Tabla 2. Parámetros detallados de los mensajes de tipo fiducial_msgs::FiducialArray

Parámetro	**Tipo**	**Descripción**
header	Header	Metadatos estándar. Utilizados para comunicar datos con marca de tiempo en un sistema de coordenadas en particular.
image_seq	int32	Identificador de la imagen en la que se han detectado los marcadores.
fiducials	Fiducial[]	Array con los marcadores detectados. Explicado en la Tabla 2.

Tabla 3. Parámetros detallados de los mensajes de tipo fiducial_msgs::Fiducial

Parámetro	**Tipo**	**Descripción**
Fiducial_id	int32	Identificador del marcador detectado
x0	float64	Posición x de la esquina 0
y0	float64	Posición y de la esquina 0

x1	float64	Posición x de la esquina 1
y1	float64	Posición y de la esquina 1
x2	float64	Posición x de la esquina 2
y2	float64	Posición y de la esquina 2
x3	float64	Posición x de la esquina 3
y3	float64	Posición y de la esquina 3

2.4 EJECUCIÓN

Para ejecutar el paquete, escribir en el terminal:

$ *roslaunch aruco_detect aruco_detect.launch*

A continuación, se muestra un fragmento con los parámetros que se pueden modificar del archivo *aruco_detect.launch* que ejecuta este programa.. El archivo completo se puede ver en el código adjuntado junto a este documento.

```
1  <launch>
2    <arg name="camera" default="/summit_xl_a/camera1"/>
3    <arg name="image" default="/image_raw"/>
4    <arg name="transport" default="compressed"/>
5    <arg name="fiducial_len" default="0.1778"/>
6    <arg name="dictionary" default="16"/>
7    <arg name="do_pose_estimation" default="true"/>
8    <arg name="ignore_fiducials" default="" />
9    <arg name="fiducial_len_override" default="" />
…    ….
```

Parámetros importantes:

Camera: Nombre en la que la cámara publica las imágenes. (línea 2)
Image: Nombre de la imagen
Fiducial_len: tamaño del marcador en metros (línea 5).
Dictionary: diccionario al que pertenecen los ArUcos que se quiere detectar (línea 6).

Siendo el *topic* en el que se publican las imágenes: ***/Camera/Image***

3 MAPEADO DE MARCADORES

Este programa proporciona un mapa de los marcadores presentes en el entorno, en un archivo .csv, a partir de la información proporcionada por el paquete de detección de ArUcos.

3.1 TOPICS A LOS QUE SE SUSCRIBE

/fiducial_vertices (fiducial_msgs/FiducialArray)

3.2 EJECUCIÓN

Es necesario que se esté ejecutando el programa de detección de marcadores simultáneamente. Por ello, escribir en el terminal:

```
$ roslaunch aruco_detect aruco_detect.launch
```

```
$ roslaunch map_aruco map_aruco.launch
```

A continuación, se muestra un fragmento con los parámetros que se pueden modificar del archivo *map_aruco.launch* que ejecuta este programa. El archivo completo se puede ver en el código adjuntado junto a este documento.

```
1 <launch>
2   <arg name="n_markers" default="8"/>
3   <arg name="marker_size" default="0.1778"/>
4   <arg name="path_fiducials_map" default="/home/aruco_map.yml"/>
5   <arg name="path_camera_params" default="/home /cameraParams/info_cameras.yml"/>
6   <arg name="topic_arucoDetect" default="/fiducial_vertices1"/>
… …
```

Parámetros importantes:

n_markers: Número de marcadores en el entorno. (línea 2)
marker_size: Tamaño de los marcadores. (línea 3)
path_fiducials_map: Ruta al archivo de tipo YAML donde se almacenará la localización de los marcadores, respecto al sistema de coordenadas de referencia, dispuestos sobre la escena. (línea 4)
path_camera_params: Ruta al archivo de tipo YAML donde se almacenarán los parámetros de calibración de las cámaras. (línea 5)
topic_arucoDetect: *Topic* en el que se publican los marcadores detectados. (línea 6)

4 LOCALIZACIÓN DEL ROBOT

La localización del robot mediante los marcadores ArUco se ha implementado en dos paquetes para ROS, desarrollados en C++. En primer lugar, se ha desarrollado uno de manera que se subscribe a los *topics* que contienen la información publicada por el paquete *aruco_detect* ejecutándose sobre las imágenes de una sola cámara. Así, ejecutándose sobre todas ellas, cada cámara proporcionará una posible localización junto con una estimación de su fiabilidad. Se referirá a este paquete como *tracking_camera*.

Posteriormente, el segundo paquete se subscribe a todas las poses arrojadas por cada una de las cámaras y proporciona una localización más fiable utilizando el Filtro de Kalman. Se referirá a este paquete como *fusing_poses*.

Como se ha explicado previamente, es necesario conocer la localización de los marcadores en el entorno referida al sistema del mundo. Por ello, dicha información se debe registrar en un archivo que será leído al comienzo de la ejecución del paquete de localización. Así como los parámetros de las cámaras, tanto intrínsecos como extrínsecos, dispuestas sobre el robot.

4.1 TRACKING_CAMERA

4.1.1 *TOPICS* A LOS QUE SE SUSCRIBE

Para que el nodo se ejecute correctamente, es necesario subscribirse al *topic* que almacena la información proporcionada por *aruco_detect*.

/fiducial_vertices (fiducial_msgs/FiducialArray)

El tipo de mensaje FiducialArray se detalló en la Tabla 2.

4.1.2 *TOPICS* EN LOS QUE PUBLICA

La posición y orientación del robot estimada en este paquete es publicada en otro *topic*, en mensajes de tipo *geometry_msgs::PoseWithCovariance,* de manera que otros módulos del proyecto puedan acceder a esta información.

Tabla 4. Parámetros detallados de los mensajes de tipo geometry_msgs::PoseWithCovariance

Parámetro	**Tipo**	**Descripción**
pose	Pose	Pose (Detallado en la Tabla 5)
covariance	Float64[36]	Matriz de covarianzas

Tabla 5. Parámetros detallados de los mensajes de tipo geometry_msgs::Pose

Parámetro	**Tipo**	**Descripción**
position	point	Posición 3D (X,Y,Z)
orientation	quaternion	Orientación expresada en cuaterniones (x,y,z,w)

La posición se almacena en un punto 3D, mientras que la orientación se almacena en forma de cuaterniones [5].

4.2 FUSING_POSES

4.2.1 *TOPICS* A LOS QUE SE SUSCRIBE

Para que el nodo se ejecute correctamente, es necesario subscribirse a los 8 *topic* que proporcionan una estimación de localización del robot según cada una de las cámaras de tipo geometry_msgs/PoseWithCovariance.

El tipo de mensaje geometry_msgs/PoseWithCovariance se detalló en laTabla 4.

4.2.2 *TOPICS* EN LOS QUE PUBLICA

Publica en un topic de tipo de mensaje geometry_msgs/PoseWithCovariance una estimación más robusta de la pose del robot. Lo que se considera como la localización final del robot.

4.3 EJECUCIÓN

Primero, hay que ejecutar la detección de marcadores y el *tracking_camera* para cada una de las ocho cámaras. Después, ejecutar el paquete *fusing_poses*.

$ *roslaunch aruco_detect aruco_detect_all.launch*
(Ejecuta el paquete para las ocho cámaras)

$ *roslaunch tracking_camera tracking_camera_all.launch*
(Ejecuta el paquete para las ocho cámaras)

$ *roslaunch fusing_poses fusing_poses.launch*

A continuación, se muestra un fragmento con los parámetros que se pueden modificar del archivo *tracking_camera.launch* que ejecuta este programa. El archivo completo se puede ver en el código adjuntado junto a este documento.

```
1   <launch>
2     <arg name="path_fiducials_map" default="/home/aruco_map.yml"/>
3     <arg name="path_camera_params" default="/home/cameraParams/info_cameras.yml"/>
4     <arg name="topic_arucoDetect" default="/fiducial_vertices1"/>
5     <arg name="id_cam" default="1"/>
…   …
```

Parámetros importantes:

path_fiducials_map: Ruta al archivo de tipo YAML donde se almacena la localización de los marcadores. (línea 2)
path_camera_params: Ruta al archivo de tipo YAML donde se almacenarán los parámetros de calibración de las cámaras. (línea 3)
topic_arucoDetect: *Topic* en el que se publican los marcadores detectados según la cámara correspondiente. (línea 4)
id_cam: ID de la cámara sobre la que se realiza el *tracking*.

5 PANORAMA

En este caso, los programas son ejecutados y coordinados mediante FlexBe. La documentación se podrá ver más en profundidad en otro de los documentos adjuntados.

5.1 CAPTURA DE IMÁGENES Y TRANSFORMACIÓN

5.1.1 *TOPICS* A LOS QUE SE SUSCRIBE

Se subscribe a la información RGB y de profundidad proporcionada por la cámara RGBD y almacena las imágenes obtenidas en el directorio correspondiente. La información RGB es de tipo sensors_msgs/Image, ver Tabla 6. y la nube de puntos correspondiente es sensors_msgs/PointCloud2, ver Tabla 7.

Tabla 6. Parámetros detallados de los mensajes *de tipo sensors_msgs/Image*

Parámetro	**Tipo**	**Descripción**
header	header	Metadatos estándar. Utilizados para comunicar datos con marca de tiempo en un sistema de coordenadas en particular.
height	uint32	Alto de la imagen. Número de filas.
width	uint32	Ancho de la imagen. Número de columnas.
encoding	sring	Codificación de los píxeles: canal, tamaño...
step	uint32	Longitud de fila completa en bytes.
data	uint8[]	Datos de la matriz de la imagen.

Tabla 7. Parámetros detallados de los mensajes *de tipo sensors_msgs/PointCloud2*

Parámetro	**Tipo**	**Descripción**
header	header	Metadatos estándar. Utilizados para comunicar datos con marca de tiempo en un sistema de coordenadas en particular.
height	uint32	Estructura de la nube de puntos.
width	uint32	Estructura de la nube de puntos.
fields	pointField[]	Describe el canal y el diseño de los datos binarios.
point_step	uint32	Longitud de un punto en bytes.
row_step	uint32	Longitud de una fila en bytes.
data	uint8[]	Puntos que forman la nube de puntos.
is_dense	bool	*True* si no hay puntos inválidos.

5.1.2 PARÁMETROS IMPORTANTES

path_images (string): Ruta al directorio donde se guardarán las imágenes.
H_matrix (float[]): Matriz de homografía que relaciona el plano del suelo y el de la cámara.

5.2 CREACIÓN PANORAMAS PARCIALES

A partir de las imágenes obtenidas en el apartado anterior se construye el panorama completo.

5.2.1 PARÁMETROS IMPORTANTES

path_images (string): Ruta al directorio donde se guardan las imágenes. Asimismo, se guardará el panorama creado.

num_images (int): Número de imágenes

direction (string): Dirección que sigue el robot en su trayectoria. Necesario para completar correctamente el panorama completo.

5.3 CREACIÓN PANORAMA COMPLETO

A partir de las imágenes obtenidas en el apartado anterior se construye el panorama completo.

5.3.1 PARÁMETROS IMPORTANTES

path_images (string): Ruta al directorio donde se guardan las imágenes. Asimismo, se guardará el panorama creado.

5.4 EXTRACCIÓN DE POLÍGONOS

5.4.1 PARÁMETROS

path_images (string): Ruta al directorio donde se guarda la imagen del panorama.

5.5 PAQUETES ADICIONALES

El resto de las librerías se pueden instalar directamente mediante la instrucción de Linux:

```
$ sudo apt-get install ros-kinetic-*
```

Siendo * el nombre del paquete. En ocasiones la instalación de varios paquetes se realiza por el mismo comando por lo que se recomienda la consulta de la página de la wiki de ROS de cada uno de los paquetes.

Para este paquete, es necesario tener instalada la librería PCL (*Point Cloud Library*) [6]. Para ello, seguir los siguientes pasos:

- Clonar el repositorio PCL

```
$ git clone https://github.com/PointCloudLibrary/pcl.git pcl-trunk
$ ln -s pcl-trunk pcl
$ cd pcl
```

- Instalar librerías requeridas

```
$ sudo apt-get install g++
$ sudo apt-get install cmake cmake-gui
$ sudo apt-get install doxygen
$ sudo apt-get install mpi-default-dev openmpi-bin openmpi-common
$ sudo apt-get install libflann1.8 libflann-dev
$ sudo apt-get install libeigen3-dev
$ sudo apt-get install libboost-all-dev
$ sudo apt-get install libvtk6-dev libvtk6.2 libvtk6.2-qt
$ sudo apt-get install libvtk5.10-qt4 libvtk5.10 libvtk5-dev
$ sudo apt-get install 'libqhull*'
$ sudo apt-get install libusb-dev
$ sudo apt-get install libgtest-dev
$ sudo apt-get install git-core freeglut3-dev pkg-config
$ sudo apt-get install build-essential libxmu-dev libxi-dev
$ sudo apt-get install libusb-1.0-0-dev graphviz mono-complete
$ sudo apt-get install qt-sdk openjdk-9-jdk openjdk-9-jre
$ sudo apt-get install phonon-backend-gstreamer
$ sudo apt-get install phonon-backend-vlc
$ sudo apt-get install libopenni-dev libopenni2-dev
```

- Compilar e instalar PCL

```
$ mkdir release
$ cd release
$ cmake -DCMAKE_BUILD_TYPE=None -DBUILD_GPU=ON -DBUILD_apps=ON -
DBUILD_examples=ON ..
$ make
$ sudo make install
```

6 CALIBRACIÓN CÁMARA CON CHARUCO

Se desarrolló un código para realizar la calibración de una cámara uEye, facilitando así el paso a un sistema físico en un futuro. Además, también se desarrolló un pequeño programa de captura de imágenes.

Ejecutando el programa de captura de imágenes, las imágenes se irán guardando, en el directorio correspondiente, al pulsar la tecla INTRO.

La calibración se realiza ejecutando el programa desarrollado para tal efecto, a partir de las imágenes en las que se encuentra presente el chArUco capturadas previamente. Es necesario indicar el directorio donde se encuentran dichas imágenes.

Para poder llevar a cabo la calibración es necesario tener OpenCV instalado.

7 UEYE EN RASPBERRY PI 3

Se realizaron pruebas utilizando cámaras uEye de IDS Imaging, en concreto el modelo XS [7]. Actualmente este modelo está descatalogado, la nueva versión sería el modelo uEye XS2. Se propuso la opción de utilizar una Raspberry pi 3 B+ [8] para cada una de las ocho cámaras para detectar los marcadores. Es necesario instalar ROS en la Raspberry. A continuación, se exponen las nociones básicas para manejar conjuntamente los dos sistemas.

- El primer paso consiste en descargar e **instalar en la Raspberry los drivers** correspondientes de la página de IDS:

Link: https://en.ids-imaging.com/download-ueye-emb-hardfloat.html
Driver: ARMv7 Cortex-A/ARMv8 (32 bit).

Una vez descargado, para instalarlo ejecutar en el terminal:

```
$ sudo sh ueye_4.91.1.0_armhf.run
```

Ahora se puede abrir la cámara con:

```
$ idscameramanager
```

- Instalar el **paquete *ueye_cam***.

Link*:* http://wiki.ros.org/ueye_cam

Una vez instalado, ejecutar el comando siguiente para capturar imágenes y publicarlas en un *topic*.

```
$ roslaunch ueye_cam xs_bayer_rgbd.launch
```

Para ejecutar correctamente este paquete, es necesario tener los parámetros de la cámara en el directorio *.ros*:

.ros/camera_conf/camera.ini – Parámetros de configuración de la cámara. Extraídos desde el *idscameramanager*.

.ros/camera_info/camera.yaml – Parámetros de calibración de la cámara.

- Instalar el **paquete *aruco_detect***.

Para instalarlo ejecute el siguiente comando:

```
$ sudo apt_get install ros-kinetic-fiducials
```

Para ejecutar el paquete:

```
$ roslaunch aruco_detect aruco_detect.launch
```

Es necesario modificar el archivo *aruco_detect.launch* para subscribirse al topic correspondiente. Para ello, modificar los parámetros *camera* e *image* como corresponda.

<arg name="camera" default="/**nombre_camara**"/>
<arg name="image" default="/**nombre_imagen**"/>

Siendo el *topic* en el que se publican las imágenes*: /nombre_camara/nombre_imagen*

7.1 CONEXIÓN RASPEBERRY Y ORDENADOR

Se planteó que cada cámara llevase asociada una *Raspberry* que realizase el procesamiento de las imágenes para detectar los marcadores. Posteriormente, esa información será manejada por un ordenador central para estimar la pose del robot. Para ello, es necesario conectar las *Raspberrys* con el ordenador. Se realizaron unas pruebas en este sentido. A continuación, se exponen los pasos a seguir para lograr la comunicación entre el ordenador y una *Raspberry* utilizando ROS.

Ejecutar en el *master* (en este caso el ordenador), siendo *direccionIP_master* la dirección IP del *master*:

```
$ export ROS_MASTER_URI = http://localhost:11311/

$ export ROS_HOSTNAME = direccionIP_master

$ export ROS_IP = direccionIP_master
```

Ejecutar en la *raspberry* que se conecta al *master*.

```
$ export ROS_MASTER_URI = http://direccionIP_master:11311/

$ export ROS_HOSTNAME = direccionIP_raspberry

$ export ROS_IP = direccionIP_raspberry
```

Tras estos pasos, la comunicación ha sido establecida.

Para saber la dirección IP de cada dispositivo:

```
$ ifconfig
```

8 REFERENCIAS

[1] «ROS.org | Powering the world's robots». .

[2] «OpenCV». [En línea]. Disponible en: https://opencv.org/. [Accedido: 16-jul-2019].

[3] «aruco_detect - ROS Wiki». [En línea]. Disponible en: http://wiki.ros.org/aruco_detect. [Accedido: 27-nov-2018].

[4] *Repository for OpenCV's extra modules. Contribute to opencv/opencv_contrib development by creating an account on GitHub*. OpenCV, 2019.

[5] J. Huerta, «Introducing The Quaternions», p. 25.

[6] «PCL - Point Cloud Library (PCL)». [En línea]. Disponible en: http://pointclouds.org/. [Accedido: 08-oct-2019].

[7] «uEye XS». [En línea]. Disponible en: https://es.ids-imaging.com/store/xs.html. [Accedido: 23-jul-2019].

[8] «Buy a Raspberry Pi 3 Model B+ – Raspberry Pi». [En línea]. Disponible en: https://www.raspberrypi.org. [Accedido: 23-jul-2019].