OpenCV ROS - CV Camera Node	
Facilitador	José Carlos Rangel Ortiz

1. Utilización de una cámara CV_Camera

1. Esta sección no esta disponible para Sistemas Operativos Windows, debido a la falta de disponibilidad de ciertas librerías.
2. Dentro de los puntos fuertes de ROS esta la facilidad de uso de diversos dispositivos
3. Su configuración permite que se utilice *hardware* sin tener que configurar toda la comunicación entre el dispositivo y la computadora.
4. Uno de los dispositivos más empleados es una cámara, ya sea por medio del puerto USB o la empotrada en la parte superior de un portátil
5. Para trabajar con los dispositivos en ROS el primer paso siempre es activar o correr un nodo que se encarga de establecer la comunicación.
6. El siguiente paso es conocer los $topics$ que produce este dispositivo
7. Luego debemos identificar cual es el $topic$ que nos interesa utilizar
8. Por ultimo se procede entonces a utilizar la información enviada por el nodo.
9. Para este ejemplo utilizaremos una cámara para captar imágenes en tiempo real
10. Se utiliza el paquete de ROS `cv_camera`
11. Más información cv_camera
12. Para activar la captura de imágenes se activa el `roscore` (si no se había hecho anteriormente)
13. Luego en una consola se escribe el comando:

```
rosrun cv_camera cv_camera_node
```

14. Este comando no activará ni producirá ningún resultado visible en la pantalla del computador, solo activa la interfaz de la cámara
15. Para conocer los $topics$ que envía el nodo escribimos en la consola:

```
rostopic list
```

16. Lo cual produce el siguiente resultado, la lista de los nodos activos

```
/cv_camera/camera_info
/cv_camera/image_raw
/cv_camera/image_raw/compressed
/cv_camera/image_raw/compressed/parameter_descriptions
/cv_camera/image_raw/compressed/parameter_updates
/cv_camera/image_raw/compressedDepth
/cv_camera/image_raw/compressedDepth/parameter_descriptions
/cv_camera/image_raw/compressedDepth/parameter_updates
/cv_camera/image_raw/theora
/cv_camera/image_raw/theora/parameter_descriptions
/cv_camera/image_raw/theora/parameter_updates
/rosout
/rosout_agg
```

17. De la lista anterior todo *topic* que contenga la frase `cv_camera`, estará relacionado con nuestro nodo de cámara

18. El siguiente paso es visualizar las capturas de la cámara en una ventana dentro del ordenador.

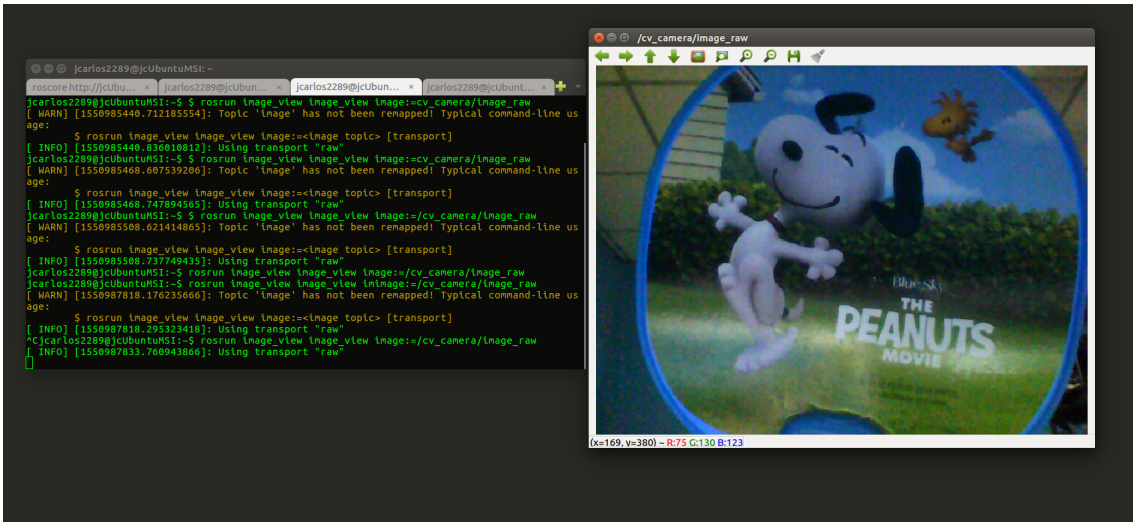
19. En este ejemplo se utilizará el nodo `/cv_camera/image_raw`

20. Para esto nos serviremos del paquete `image_view` que es capaz de crear una ventana que permite la presentación de imágenes.

21. Para lanzar esta ventana se escribe lo siguiente, en una consola

```
rosrun image_view image_view image:=/cv_camera/image_raw
```

22. Obteniendo lo siguiente



23. Para utilizar otro visor integrado con una interfaz de usuario con más opciones podemos escribir esto en la consola

```
rosrun rqt_image_view rqt_image_view
```

2. Almacenar una Imagen

1. Crear un *package* que contendrá un nodo que almacena las imágenes desde una cámara.
2. Nuestro *package* se debe crear dentro de la carpeta `/catkin_ws/src/`
3. Este ejemplo se creara utilizando *Python* como lenguaje.
4. Este *package* tendrá como dependencias `sensor_msgs, cv_bridge, rospy y std_msgs`.
5. Se abre un nuevo terminal y se escribe lo siguiente :

```
cd ~/catkin_ws/src
catkin_create_pkg guadar_imagen sensor_msgs cv_bridge rospy std_msgs
```

6. Esto creará el *package*, `CMakeList.txt` y `package.XML` para nuestro código.
7. Similar a los demás ejemplos, se debe crear la carpeta `scripts` para guardar nuestro código fuente en Python
8. Dentro de la carpeta `scripts` de nuestro recién creado *package* crearemos un archivo que será el código fuente de nuestro programa.
9. Se puede hacer desde consola con las siguientes instrucciones:

```
cd ~/catkin_ws/src/guadar_imagen/scripts
touch capturar.py
```

10. El comando `touch` crea un archivo con el nombre especificado en la dirección del terminal.
11. Si tenemos Sublime Text, podemos utilizar el comando `subl`, el cual nos creará el archivo y abrirá el editor Sublime Text y modificar el contenido de los archivos.

```
subl capturar.py
```

12. Se copia lo siguiente dentro del archivo `capturar.py`

```
#!/usr/bin/env python
from __future__ import print_function

import sys
import rospy
import cv2
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
```

```
class image_converter:

  def __init__(self):
    self.image_pub = rospy.Publisher("image_topic_2",Image, queue_size=1)

    self.bridge = CvBridge()
    self.image_sub = rospy.Subscriber("/cv_camera/image_raw",Image,self.callback)

  def callback(self,data):
    try:
      cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
    except CvBridgeError as e:
      print(e)

    (rows,cols,channels) = cv_image.shape
    if cols > 60 and rows > 60 :
      cv2.circle(cv_image, (50,50), 10, 255)

    cv2.imshow("Image window", cv_image)

    if cv2.waitKey(20) & 0xFF == ord('s'):
      cv2.imwrite('camera_image.jpeg', cv_image)

    cv2.waitKey(3)

    try:
      self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "bgr8"))
    except CvBridgeError as e:
      print(e)

def main(args):
  ic = image_converter()
  rospy.init_node('image_converter', anonymous=True)
  try:
    rospy.spin()
  except KeyboardInterrupt:
    print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)
```

13. Antes de compilar se deben agregar las siguientes lineas al archivo `CMakeLists.txt` del *package*

14. En la sección de *Install* del `CMakeLists` descomentar la sección de `catkin_install_python` y agregar el nombre y ubicación de nuestro script, debe quedar como el siguiente fragmento.

```
 catkin_install_python(PROGRAMS
   scripts/capturar.py
   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
 )
)
```

15. Esta modificación el indica al sistema que se debe utilizar una distribución de Python 3 para la ejecución del *script*. En este caso la versión Noetic de ROS, solo utiliza Python3, estas líneas aseguran entonces que la que ejecución de los *scripts* indicados será usando Python3.

16. Los *scripts* de Python requieren que se establezcan los permisos para la ejecución, lo cual se hace la siguiente manera en la consola:

```
chmod +x captura.py
```

17. Para compilar el $package$ se utiliza la siguiente instrucción en la consola

```
cd ~/catkin_ws/
catkin_make
```

18. Antes de ejecutar cualquier nodo se debe ejecutar el $roscore$ en una consola diferente.

```
roscore
```

19. Para la ejecución del nodo utilizamos el comando $rosrun$ de la siguiente manera en una nueva consola, de igual manera antes de la ejecución se indica a la consola donde puede encontrar los ejecutables compilados en este *workspace*.

```
source ~/catkin_ws/src/devel/setup.bash
rosrun guadar_imagen captura.py
```

20. Cuando el programa se ejecute, no se observará nada en la consola, esto se debe a que nuestro programa consume la información producida por el $topic$ de la cámara, el cual no se ha lanzado, para hacerlo se debe ejecutar un nodo de cv_camera en una consola diferente de la siguiente manera:

```
rosrun cv_camera cv_camera_node
```

21. Una vez en ejecución el programa guardara imágenes en la carpeta del $package$ dentro del ws, siempre que se presione la tecla "s".