

IIC2685 - Robótica Móvil I – 2022

Clase práctica 2

Kinect

Profesor: Gabriel Sepúlveda V.
grsepulveda@uc.cl

Kinect

EMISOR LUZ ESTRUCTURADA IR



Kinect

- Depth: proyección de luz estructurada IR es capturada por cámara IR y procesada
- Procesamiento: análisis de intensidad, densidad, etc.
- Salida: profundidad estimada (eje z en Kinect)



Kinect

- Ojo con las sombras



Kinect

- Ojo con las sombras



- Emisor y receptor en distinta posición = sombra a luz estructurada = sectores sin estimación de profundidad (con valor 0 o NaN)

Kinect

- En términos prácticos
 - Imágenes VGA: 640 pixeles (ancho) x 480 pixeles (alto)
 - Imagen de profundidad “cercana” a imagen RGB en términos de ajuste
 - 30 frames por segundo
 - En ROS
 - Imagen RGB: 24 bits (8 bits por color R,G,B), cada canal entre 0 y 255
 - Imagen Depth: 16 bits, un canal, profundidad en milímetros
 - Medición de profundidad funciona en ≥ 45 [cm] (unos 15 [cm] del robot)
 - Ángulo de visión de cámaras (FOV): $\sim 57^\circ$ horizontal, $\sim 43^\circ$ vertical
 - Sirve para estimar (x, y) a partir de z , el punto $(x_{\text{pix}}, y_{\text{pix}})$ y la resolución de la imagen

OpenCV

- Imágenes son **arreglos multidimensionales**
 - Imagen RGB: (480, 640, 3)
 - Imagen Depth: (480, 640)
 - Pixel (0, 0): esquina superior izquierda de la imagen



What We See

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

What Computers See

Kinect y ROS

- En ROS: paquete **openni_launch**

```
$ roslaunch oppenni_launch oppenni.launch
```

Para visualizar rápidamente

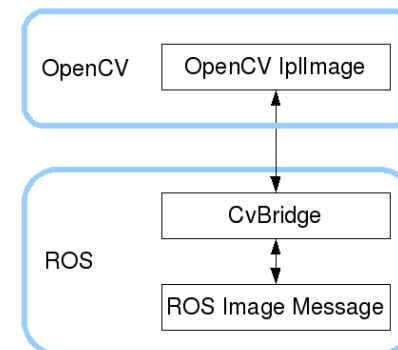
```
$ rosrun rqt_image_view rqt_image_view
```

- Tópicos importantes

- **/camera/rgb/image_color**
- **/camera/depth/image_raw**

- Los mensajes de imágenes son del tipo **sensor_msgs/Image**

- Para convertirlas a OpenCV: **cv_bridge**



Kinect y ROS: ejemplo de captura

```
import rospy
import roslib
import numpy
from sensor_msgs.msg import Image
import cv2
from cv_bridge import CvBridge

class Turtlebot_Kinect( object ):
    def __init__( self ):
        self.depth_sub = rospy.Subscriber( '/camera/depth/image', Image, self.depth_cb )
        self.rgb_sub = rospy.Subscriber( '/camera/rgb/image_color', Image, self.rgb_cb )
        self.bridge = CvBridge()
        self.current_cv_depth_image = None
        self.current_cv_rgb_image = None

    def depth_cb( self, data ):
        self.current_cv_depth_image = self.bridge.imgmsg_to_cv2( data )

    def rgb_cb( self, data ):
        self.current_cv_rgb_image = self.bridge.imgmsg_to_cv2( data )

if __name__ == '__main__':
    rospy.init_node( 'test_kinect' )
    handler = Turtlebot_Kinect()
    rospy.spin()
```

Bibliografía

- *Digital Image Processing*, Gonzalez, R. and Woods, R.
- Learning OpenCV, Bradski, G. and Kaehler, A.
- <https://opencv.org/>
- <https://numpy.org/>