

INTRODUCCIÓN A LA VISIÓN ARTIFICIAL CON ROS

ADQUISIÓN, PROCESAMIENTO Y DISTRIBUCIÓN DE
IMÁGENES EN SISTEMAS ROBÓTICOS

M.C. Alexis Guijarro

25 de Septiembre del 2021

Contenido i

1. Introducción

2. Motivación

3. Hardware

4. Software

5. Demostración

6. Bibliografía

Introducción

Introducción i

- La utilización de cámaras web permite la captura de hechos que pasan al efectuar los experimentos
- Información: **activa o pasiva**
- En el caso de ser pasivo, usualmente es usado para la captación de vídeo, donde se efectúe un análisis posterior al experimento

Introducción ii

- El rol activo permite la utilización de la cámara para incluirla dentro del experimento
- En ambos casos es necesario efectuar la calibración
- Correcta selección del hardware

Motivación

Motivación i



Figura 1: Detección de caras en grupos de personas

Motivación ii

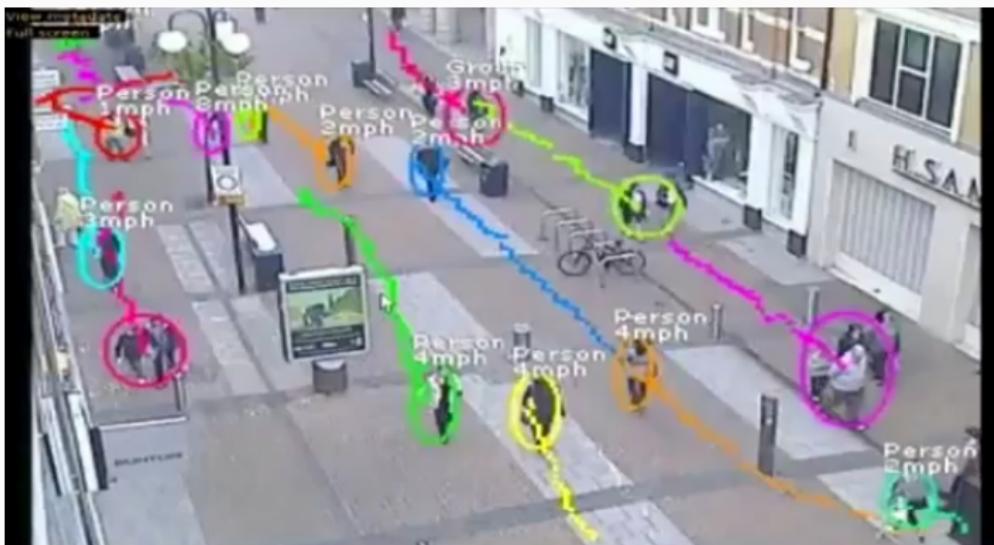


Figura 2: Seguimiento de personas

Motivación iii

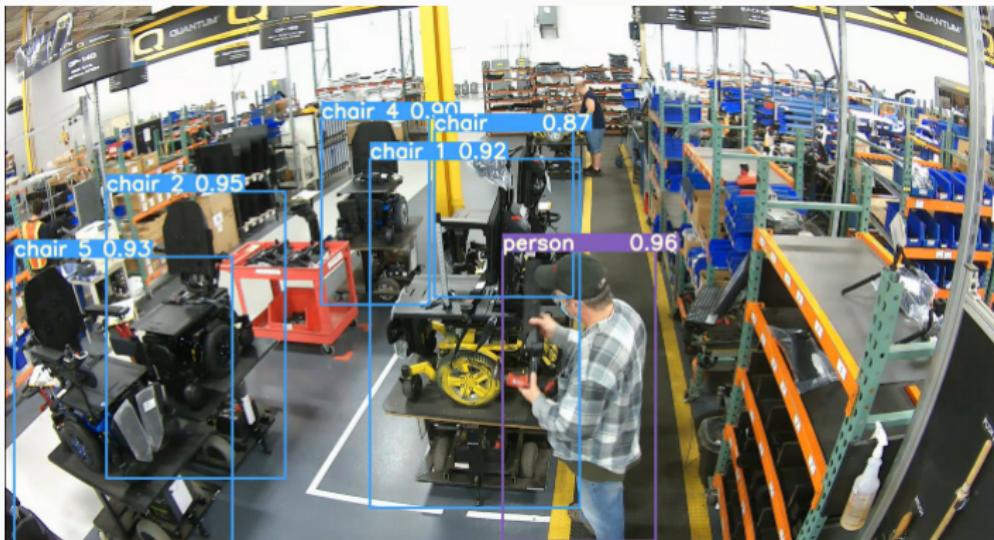


Figura 3: Reconocimiento de objetos en línea de ensamblaje

Motivación iv

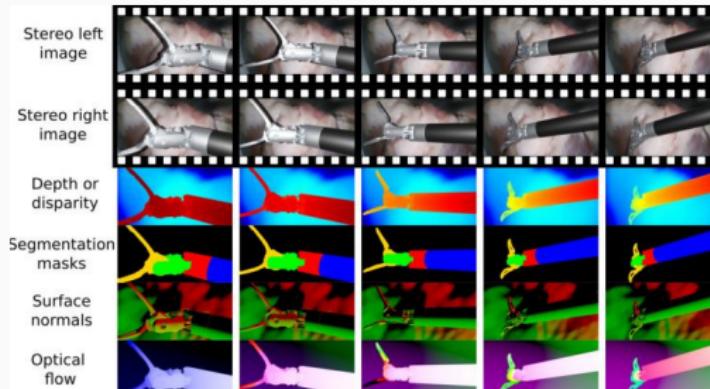


Figura 4: Diferentes tipos de imágenes de un robot cirujano

Motivación v

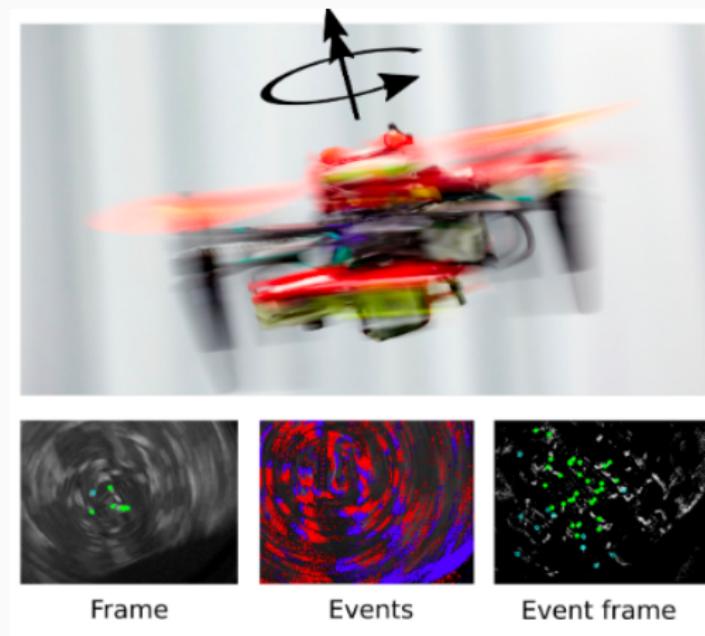


Figura 5: Drone reponiéndose ante averías usando una cámara de eventos

Motivación vi



Figura 6: Vector de Anki

Hardware

Selección del hardware i

- La configuracion varia dependiendo de las necesidades y capacidades de la aplicacion asi como el sistema en el que se ejecuta
- Para este ejercicio se consideran dos cámaras, aunque como requerimiento mínimo solo una puede satisfacer lo básico del taller:
 - Web Cams Integradas
 - Web Cams USB (Logitech C920)
 - Cámaras de Color/Profundidad como el Kinect
 - Cámaras WiFi (Cámaras IP, GoPro)

Selección del hardware ii

- Las capacidades de la camara y el sistema al que se adhieren deben ser revisadas minusciosamente
- Parámetros:
 - Suficientes cuadros por segundo (FPS)
 - Resolución adecuada
 - Ancho de banda
- Por fortuna, se cuenta con cierta flexibilidad que permite hacer funcionar *X* configuracion ajustando ciertos parámetros

Ejemplos de Cámaras i

- Webcams, gracias al módulo v4l2 (Logitech, Elgato, Microsoft LifeCam, sin marca)



Figura 7: Logitech C920 (Logitech)

Ejemplos de Cámaras ii

- Red, conectadas a una red local o remota (Cámaras de vigilancia, GoPro)



Figura 8: GoPro HERO5 Black (GoPro)

Ejemplos de Cámaras iii

- Profundidad, pueden ser RGB-D o no (Kinect, Intel RealSense, Asus Xtion)



Figura 9: Kinect (Xbox 360)

Ejemplos de Cámaras iv

- DLSR, gracias al módulo gphoto2 (Canon, Nikon, Panasonic)



Figura 10: Canon EOS Rebel T7 (Canon)

Cámaras conectadas al Kernel Linux i

- Gracias a la masificación de la distribución de Linux
- Alta posibilidad de encontrar equipos tipo Plug & Play
- De ser aceptados, se verá montado por el kernel dentro del directorio \dev

Cámaras conectadas al Kernel Linux ii

- En el caso de las cámaras, serán montadas como `\dev\video*`
- Siendo * el numero de cámara que el sistema le asigno
- El orden de los dispositivos empieza con el numero 0 en adelante

Cámaras conectadas al Kernel Linux iii

```
11  v10
12  vga_arbiter
13  vhci
14  vhost-net
15  vhost-vsock
16  video0
17  wmi
18  zero
19
```

Figura 11: Cámara montada en el kernel

```
ls /dev
```

Cámaras conectadas al Kernel Linux iv

- De otra forma, se deben de crear estos puntos de montaje en forma manual, la cual se encuentra fuera del objetivo del presente taller
- Es recomendable que se instalen las herramientas requeridas para visualizar la transmision que proviene de cada una, entre los mas recomendables se encuentran:
 - VLC
 - Cheese

Cámaras conectadas al Kernel Linux v

- Aunque para un uso más avanzado se recomienda:
 - FFMPEG
 - GStreamer

Cámaras conectadas al Kernel Linux vi



Figura 12: Aplicaciones multimedia

Cámaras conectadas al Kernel Linux vii

- Documentación en sus respectivas páginas Web
- Las cámaras ancladas al Kernel de Linux entregan datos en bruto al punto de montaje asignado

Cámaras conectadas al Kernel Linux viii

- Son manejadas por el recurso “Video4Linux2” (V4L2), quien se encarga de hacer el tratamiento y transporte de las imágenes al entorno del sistema

Cámaras conectadas al Kernel Linux ix

Comando:

```
ffmpeg -f v4l2 -list_formats all -i /dev/video0
```

```
totonzx@plibish:~$ ffmpeg -f v4l2 -list_formats all -i /dev/video0
ffmpeg version 3.4.6-0ubuntu0.18.04.1 Copyright (c) 2000-2019 the FFmpeg developers
  built with gcc 7 (Ubuntu 7.3.0-16ubuntu3)
configuration: --prefix=/usr --extra-version=0ubuntu0.18.04.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/
--enable-avisynth --enable-gnutls --enable-ladspa --enable-libaass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcd
fribidi --enable-libgme --enable-libgsm --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopennpt --enable-libop
e --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvorbis --en
bxml2 --enable-libvid --enable-libzmq --enable-libzvbi --enable-omx --enable-opengl --enable-sdl2 --enable-libdc1394
nable-libopencv --enable-libx264 --enable-shared
libavutil      55. 78.100 / 55. 78.100
libavcodec     57.107.100 / 57.107.100
libavformat    57. 83.100 / 57. 83.100
libavdevice    57. 10.100 / 57. 10.100
libavfilter     6.107.100 / 6.107.100
libavresample   3.  7.  0 / 3.  7.  0
libswscale      4.  8.100 / 4.  8.100
libswresample   2.  9.100 / 2.  9.100
libpostproc    54.  7.100 / 54.  7.100
[video4linux2,v4l2 @ 0x56309a70d8c0] Compressed:      mjpeg :          Motion-JPEG : 1280x720 848x480 960x540
[video4linux2,v4l2 @ 0x56309a70d8c0] Raw       :      yuyv422 :          YUVV 4:2:2 : 640x480 160x120 320x180 320x240 424x240 640x360
/dev/video0: Immediate exit requested
```

Figura 13: Características de una cámara web

Configuración de las cámaras i

- V4L2 fue diseñado para soportar una variedad de dispositivos multimedia como:
 - La interfaz para captura de vídeo, cámaras de video
 - Salida de video, televisores, monitores
 - Puente de vídeo, manda directamente el contenido de un dispositivo de entrada a uno de salida sin pasar por algún procesamiento de parte del CPU

Configuración de las cámaras ii

- Una gran cantidad de dispositivos de video son soportados por este proyecto
- Es altamente configurable

Configuración de las cámaras iii

- Ofrece una multitud de opciones
 - Frecuencia
 - Tamaño de imagen
 - Cropping
 - Compresión de video
 - Formatos
 - Encodificación / Decodificación¹
 - Brillos
 - Contrastes
 - Entre otros

¹http://gazebosim.org/tutorials/?tut=ros_depth_camera

Ejemplo de parámetros en una cámara web

Comando:

```
v4l2-ctl -d /dev/video0 -l
```

```
totonzx@plibsh:~$ v4l2-ctl -d /dev/video0 -l
        brightness 0x00980900 (int)      : min=-64 max=64 step=1 default=0 value=0
        contrast 0x00980901 (int)       : min=0 max=95 step=1 default=0 value=0
        saturation 0x00980902 (int)     : min=0 max=100 step=1 default=64 value=64
        hue 0x00980903 (int)           : min=-2000 max=2000 step=1 default=0 value=0
white_balance_temperature_auto 0x0098090c (bool)   : default=1 value=1
        gamma 0x00980910 (int)         : min=100 max=300 step=1 default=100 value=100
        gain 0x00980913 (int)          : min=1 max=8 step=1 default=1 value=1
power_line_frequency 0x00980918 (menu)    : min=0 max=2 default=2 value=2
white_balance_temperature 0x0098091a (int)   : min=2800 max=6500 step=1 default=4600 value=4600 flags=inactive
        sharpness 0x0098091b (int)     : min=1 max=7 step=1 default=2 value=2
backlight_compensation 0x0098091c (int)   : min=0 max=3 step=1 default=3 value=3
        exposure_auto 0x009a0901 (menu) : min=0 max=3 default=3 value=3
        exposure_absolute 0x009a0902 (int) : min=10 max=626 step=1 default=156 value=156 flags=inactive
exposure_auto_priority 0x009a0903 (bool)   : default=0 value=1
```

Figura 14: Parámetros configurables de la cámara (v4l2-ctl)

Encodificación / Decodificación

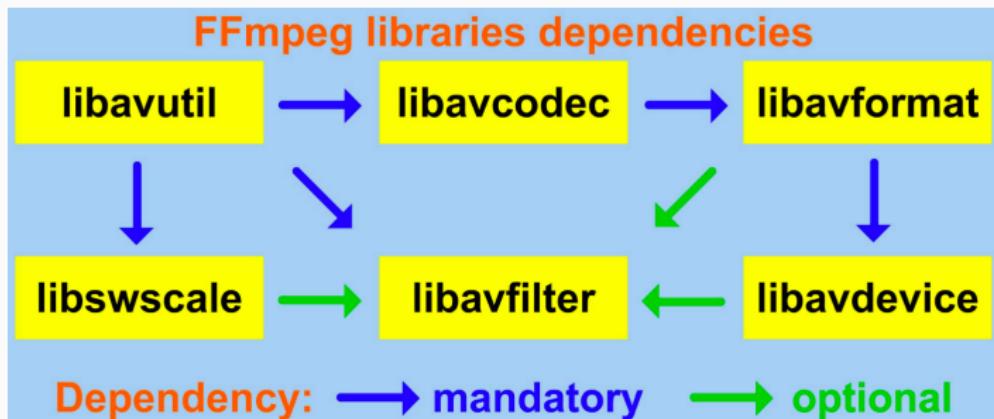


Figura 15: Dependencias de FFmpeg

Cámaras conectadas por la red i

- En esta configuración es importante conocer los dispositivos con los que se van a trabajar
- Es posible requerir de acceso privilegiado a la red
- Puede tener alta latencia
- Ayuda tener algo de conocimiento orientado formatos y protocolos de streaming
 - MPEG
 - RTPS
 - M3U8

Cámaras simuladas

- Es posible hacer uso de cámaras simuladas
 1. `v4l2loopback`
 2. **Gazebo**
 - 2.1 Kinect²
 - 2.2 RGB

²http://gazebosim.org/tutorials/?tut=ros_depth_camera

Software

Imágenes en ROS i

- Las imágenes es como cualquier otro tipo de dato intercambiable
- En ROS, esto aplica para los mensajes
- Cambia de acuerdo con el lenguaje que se use ROS: `rospy` y `roscpp`
- Se recomienda utilizar C++, ya que tiene acceso a módulos como `image_transport`

Herramientas disponibles

- Diagnóstico
 - `rqt_image_view`
 - `ros topic echo`
 - `ros topic hz`
- Aplicaciones
 - `openni_launch` y `rgbd_launch`(Kinect)
 - `ros-gopro-driver` (GoPro)
 - `usb_cam`

Calibración de la cámara i

- Se recomienda utilizar el paquete `camera_calibration`
 - Puede ser usada para cámaras monoculares y estéreo
 - Se utiliza un patrón como objeto de calibración, como un patrón de ajedrez
- Si el resultado es aceptado, la información generada es guardada
 - `sensor_msgs/CameraInfo`
 - Archivo `YAML` para su uso posterior

Calibración de la cámara ii

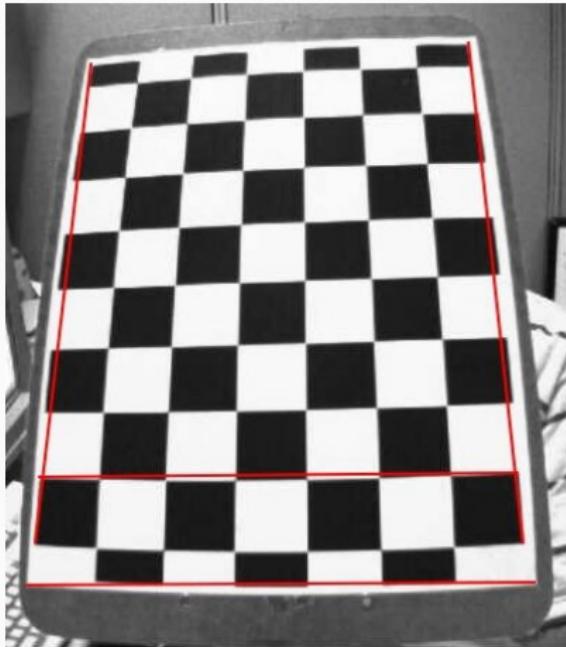


Figura 16: Patrón de ajedrez para la calibración

Calibración de la cámara iii

```
image_width: 640
image_height: 480
camera_name: example_camera
camera_matrix:
    rows: 3
    cols: 3
    data: [1035.4274095705, 0, 318.75669433335, 0,
           1035.33281408, 239.6921230165, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
    rows: 1
    cols: 5
```

Calibración de la cámara iv

```
data: [-0.001604963198, -0.0018789847996,  
       0.00011358878232, -0.0002804393593, 0]  
rectification_matrix:  
    rows: 3  
    cols: 3  
    data: [1, 0, 0, 0, 1, 0, 0, 0, 1]  
projection_matrix:  
    rows: 3  
    cols: 3  
    data: [1033.642944335938, 0, 318.1732831388, 0, 0,  
           1033.126708984, 239.2189583120708, 0, 0, 0,  
           1, 0]
```

OpenCV y ROS (`cv_bridge`) i

- OpenCV es un proyecto independiente a ROS
- Altamente popular por la unificación de técnicas del campo en Vision Artificial
- Últimamente aplicado a la Inteligencia Artificial
- La interacción entre los dos no es posible de manera directa y sin nada en medio
- `cv_bridge` permite nos puede ayudar para resolver esta problemática

OpenCV y ROS (cv_bridge) ii

- Se encarga de la transferencia de las imágenes de un dominio a otro
- Otorga abstracción entre las imágenes de ROS y las `Mat` e `IplImage` de *OpenCV*
- `opencv_apps` contiene varios algoritmos usando OpenCV

OpenCV y ROS (cv_bridge) iii

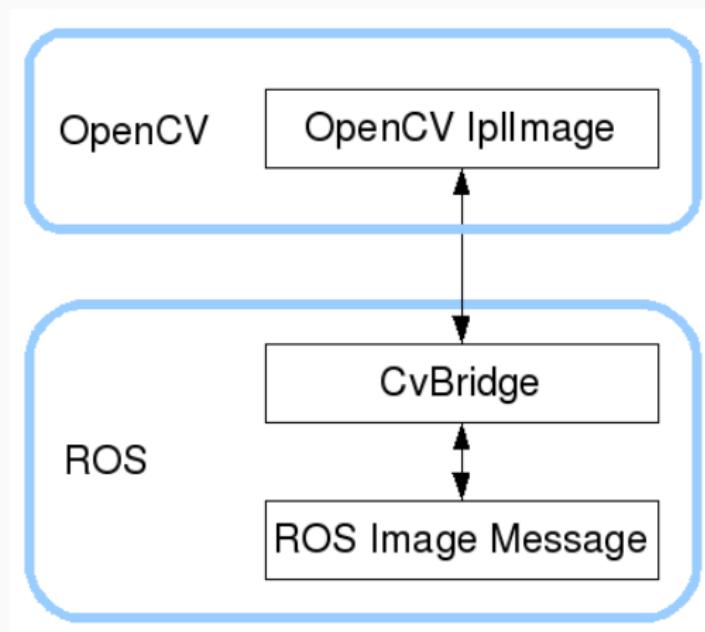


Figura 17: Diagrama de `cv_bridge`

OpenCV y ROS (cv_bridge) iv

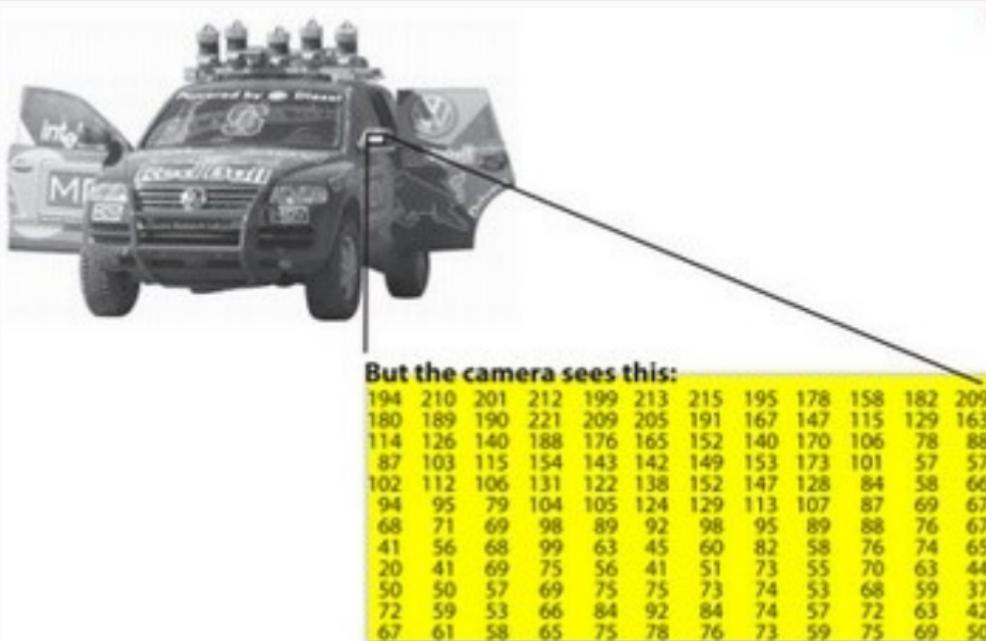


Figura 18: Objeto Mat (OpenCV)

Configuración importante (C++) i

- Es importante agregar estas líneas a tu proyecto de OpenCV y ROS:

Configuración importante (C++) ii

```
find_package(OpenCV REQUIRED)
...
include_directories(
    include
    ${catkin_INCLUDE_DIRS}
    ${OpenCV_INCLUDE_DIRS}
)
...
target_link_libraries(simple_camera
    ${catkin_LIBRARIES}
    ${OpenCV_LIBRARIES}
)
```

Por qué `usb_cam` en lugar de `VideoCapture` (OpenCV) i

- `VideoCapture` es sencillo y familiar para quienes hayan utilizado *OpenCV* en el pasado
- Tiene un control limitado sobre los parámetros de la cámara
- `usb_cam` se especializa en el manejo directo de *V4L2*
- Los cambios generados afectaran a todas las aplicaciones que usen la cámara

Código i

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/video/video.hpp>
#include <iostream>

//Window's name
static const std::string OPENCV_WINDOW = \
```

Código ii

```
"Camera viewer";  
  
//Class definition  
class singleVideo  
{  
    //NodeHandler and attached image_transport  
    ros::NodeHandle n;  
    image_transport::ImageTransport it;  
    image_transport::Subscriber img_sub;  
  
    //Name of the used topic  
    std::string sub_topic = "";
```

Código iii

```
//Read the parameters
bool readConfig()
{
    bool load_param = true;
    if(!n.getParam("cam_1",sub_topic))
    {
        ROS_ERROR("\nFailed to load \
                  a camera topic\n");
        load_param = false;
    }
}
```

Código iv

```
    return load_param;  
  
}  
  
public:  
  
    //Constructor that uses the image_transport  
    //to start receiving images  
    singleVideo() : it(n)  
    {  
        if(!readConfig()) ROS_WARN("＼nNO PARAMETERS \  
                                RECEIVED＼n");  
    }
```

Código v

```
    img_sub = it.subscribe(sub_topic, 1, \
                          &singleVideo::imageCb, this);
    cv::namedWindow(OPENCV_WINDOW);
}

//Destructor that gets rid of OpenCV
//window once the process is about to finish
~singleVideo()
{
    cv::destroyWindow(OPENCV_WINDOW);
}
```

Código vi

```
//ImageCallback used to acquire the image,  
//process it, and show it  
void imageCb(const sensor_msgs::ImageConstPtr& msg)  
{  
    cv_bridge::CvImagePtr cv_ptr;  
    try  
    {  
        cv_ptr = cv_bridge::toCvCopy(msg, \  
            sensor_msgs::image_encodings::BGR8);  
    }  
    catch(cv_bridge::Exception &e)  
    {
```

Código vii

```
ROS_ERROR("cv_bridge \n"
exception: %s", e.what());
return;
}

cv::imshow(OPENCV_WINDOW, cv_ptr->image);
cv::waitKey(3);
}
};
```

Código viii

```
//Main function
int main(int argc, char* argv[])
{
    ros::init(argc,argv,"single_video");
    singleVideo sv;
    ros::spin();
    return 0;
}
```

Demostración

Bibliografía

Bibliografía i

- Brown, Roger. 2021. "Role of Computer Vision in AI for Developing Robotics, Drones & Self-Driving Cars". *Medium*.
<https://becominghuman.ai/role-of-computer-vision-in-ai-for-developing-robotics-drones-self-driving-cars-9c92b89d57c>.
- "Camera_calibration - ROS Wiki". s/f.
http://wiki.ros.org/camera_calibration.
- Frank, Blair Hanley. 2017. "Amazon Web Services' Computer Vision Gets Real-Time Face Recognition". *VentureBeat*.
- Irving, Michael. 2018. "Hands-on with Vector, Anki's New Emotive Home Assistant Robot". *New Atlas*.
<https://newatlas.com/anki-vector-robot-hands-on/55806/>.

Bibliografía ii

Kucherenko, Ihor. 2017. "Android NDK. How to Integrate Pre-Built Libraries in Case of the FFmpeg". *Medium.*

<https://proandroiddev.com/android-ndk-how-to-integrate-pre-built-libraries-in-case-of-the-ffmpeg-7ff24551a0f>.

"OpenCV: Camera Calibration". s/f.

https://docs.opencv.org/3.4.15/dc/dbb/tutorial_py_calibration.html.

"OpenCV: Mat - The Basic Image Container". s/f.

https://docs.opencv.org/master/d6/d6d/tutorial_mat_the_basic_image.html

ROS. 2013. "Cv_bridge - ROS Wiki". http://wiki.ros.org/cv_bridge/.

Bibliografía iii

- Sun, Sihao, Giovanni Cioffi, Coen de Visser, y Davide Scaramuzza.
2021. "Autonomous Quadrotor Flight Despite Rotor Failure With
Onboard Vision Sensors: Frames vs. Events". *IEEE Robot. Autom.*
Lett. 6 (2): 580–87. <https://doi.org/10.1109/LRA.2020.3048875>.
- Techtales. 2020. "VisionBlender Generates Computer Vision
Datasets for Robotic Surgery".
<https://tectales.com/bionics-robotics/visionblender-computer-vision-datasets-robotic-surgery.html>.