gRPC and Protobuf A01283525 Ian De La Garza

Generated by Doxygen 1.9.7

Chapter 1

Proyecto de Captura y Procesamiento de Coordenadas de Objeto utilizando ROS, gRPC y REST-API

1.1 Introducción

Este proyecto tiene como objetivo principal capturar una imagen de la webcam, detectar un objeto verde en la imagen y calcular sus coordenadas X e Y. Utiliza el Robot Operating System (ROS), Python, C++, gRPC y REST-API para lograr esta funcionalidad.

1.2 Pasos del Proyecto

1. Captura de imagen y detección del objeto verde utilizando un ROS Node en Python.
2. Creación de una librería en C++ para multiplicar las coordenadas por 100.
3. Integración de la librería C++ en el ROS Node para realizar la operación de multiplicación.
4. Publicación de las coordenadas resultantes en un ROS Topic junto con su timestamp.
5. Desarrollo de un Wrapper de gRPC para convertir el ROS Topic en un servicio RPC.
6. Implementación de un programa en C# que consume el servicio RPC y muestra las coordenadas en la terminal.
7. Uso de grpc-Gateway para exponer el servicio como un REST-API.
8. Adquisición de las coordenadas del objeto desde el nuevo REST-API utilizando Postman o Flask, y guardar los datos en un archivo JSON.

1.3 Lenguajes y Plataformas

El proyecto utiliza los siguientes lenguajes y plataformas:

- Python: como wrapper para mandar el topico de ros por medio de un serivicio RPC.
- C++: para la captura de imagen y detección de objeto utilizando el ROS Node. Tambien la creación de la librería que realiza la multiplicación de las coordenadas.
- ROS: como plataforma de desarrollo y comunicación entre los componentes.
- gRPC: para la comunicación entre el ROS Node y el programa en Java a través de un servicio RPC.
- Java: para el programa que consume el servicio RPC y muestra las coordenadas en la terminal.
- grpc-Gateway junto con Go: para exponer el servicio como un REST-API.
- Postman o Flask: para adquirir las coordenadas desde el REST-API y guardar los datos en un archivo JSON.
- Protoc: para definir los mensajes y servicios utilizados en la comunicación gRPC.

1.4 Componentes Principales

Los componentes principales del proyecto son:

- ROS Node en C++: captura una imagen de la webcam, detecta el objeto verde y calcula las coordenadas X e Y.
- Librería en C++: realiza la multiplicación de las coordenadas por 100.
- Wrapper de gRPC: convierte el ROS Topic en un servicio RPC.
- Protoc: define los mensajes y servicios utilizados en la comunicación gRPC.
- Programa en C#: muestra las coordenadas del objeto en la terminal al consumir el servicio RPC.
- grpc-Gateway: expone el servicio como un REST-API para obtener las coordenadas.
- Postman o Flask: adquiere las coordenadas desde el REST-API y las guarda en un archivo JSON.

1.5 Paso 1: Captura de imagen y detección del objeto verde utilizando un ROS Node en Python

1.5.1 Código Fuente

El código fuente para este paso se encuentra en el archivo `enable_camera.cpp`.

```
#include "ros/ros.h"
#include "ros/console.h"
#include <geometry_msgs/PointStamped.h>
#include <image_transport/image_transport.h>
#include <opencv2/opencv.hpp>
#include <cv_bridge/cv_bridge.h>
#include <aruco_ros/aruco_ros_utils.h>
#include "pycam/multiply.h"
int main(int argc, char** argv) {
    ros::init(argc, argv, "webcam_publish_test");
    ros::NodeHandle nh;
    image_transport::ImageTransport it(nh);
```

```
    image_transport::Publisher pub = it.advertise("camera/image", 1);
    image_transport::Publisher marker_pub = it.advertise("camera/marker_image", 1);
    ros::Publisher marker_center_pub = nh.advertise<geometry_msgs::PointStamped>("camera/marker_center", 1);
    ros::Rate rate(30);
    cv::VideoCapture cap(
        "nvarguscamerasrc ! video/x-raw(memory:NVMM), width=(int)720, height=(int)480,format=(string)NV12,
      framerate=(fraction)30/1 ! nvvidconv ! video/x-raw, format=(string)BGRx ! videoconvert !  appsink",
        cv::CAP_GSTREAMER);
    if (!cap.isOpened()) {
        ROS_ERROR("Could not open camera");
        return -1;
    }
    bool stuff = cap.set(cv::CAP_PROP_FPS, 30);
    aruco::CameraParameters camParam = aruco::CameraParameters();
    cv::Mat frame, frame2;
    aruco::MarkerDetector mDetector;
    float min_marker_size; // percentage of image area
    nh.param<float>("min_marker_size", min_marker_size, 0.01);
    std::vector<aruco::Marker> markers;
    mDetector.setDetectionMode(aruco::DM_VIDEO_FAST, min_marker_size);
    while (ros::ok()) {
        markers.clear();
        cap » frame;
        if (frame.rows) {
            // resize image to 640x480
            cv::resize(frame, frame, cv::Size(640, 480));
             //make frame sharper
            cv::GaussianBlur(frame, frame2, cv::Size(0, 0), 3);
            cv::addWeighted(frame, 1.5, frame2, -0.6, 0, frame);
            cv::resize(frame, frame2, cv::Size(320, 240));
            // detect markers
            sensor_msgs::ImagePtr msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", frame2).toImageMsg();
            pub.publish(msg);
            mDetector.detect(frame, markers, camParam);
              // for each marker, draw info and its boundaries in the image
            for (std::size_t i = 0; i < markers.size(); ++i)
            {
                markers[i].draw(frame, cv::Scalar(0, 0, 255), 2);
                //get center of marker
                cv::Point2f center = markers[i].getCenter();
                //publish center of marker
                int x = center.x;
                int y = center.y;
                multiply(&x, &y);
                geometry_msgs::PointStamped marker_center;
                marker_center.header.stamp = ros::Time::now();
                marker_center.point.x = x;
                marker_center.point.y = y;
                marker_center_pub.publish(marker_center);
            }
            if(markers.size() > 0)
            {
                cv::resize(frame, frame2, cv::Size(320, 240));
                marker_pub.publish(cv_bridge::CvImage(std_msgs::Header(), "bgr8", frame2).toImageMsg());
            }
        }
        rate.sleep();
    }
    cap.release();
    return 0;
}
```

1.5.2 Descripción del Código

En esta etapa, inicializamos un nodo ROS llamado ¨webcam_publish_test¨ y configuramos varios publicadores y suscriptores. El código entonces comienza a capturar imágenes desde la webcam utilizando OpenCV y detecta marcadores ArUco en la imagen.

Cada vez que se captura una imagen, se publica en el tópico ¨camera/image¨. Si se detectan marcadores ArUco en la imagen, se dibujan en la imagen y se publica en el tópico ¨camera/marker_image¨. El centro del marcador detectado se calcula, se multiplica por 100 y se publica en el tópico ¨camera/marker_center¨.

La detección de marcadores ArUco se realiza utilizando la biblioteca ArUco. Los marcadores son objetos cuadrados en el mundo real que pueden ser fácilmente detectados y identificados en imágenes. Son útiles para la calibración de cámaras y la estimación de la pose en aplicaciones de realidad aumentada. El código detecta los marcadores utilizando el modo de detección rápido de video (`DM_VIDEO_FAST`) con un tamaño mínimo de marcador especificado por el parámetro ¨min_marker_size¨.

El código continúa en un bucle hasta que se detiene el nodo ROS. Luego, libera la cámara y finaliza.

1.5.3 Uso del Código

Para usar este código, debes tener una webcam compatible y un marcador ArUco visible en la imagen de la webcam. Puedes ajustar el tamaño mínimo del marcador y la frecuencia de captura de imágenes cambiando los parámetros ¨min_marker_size¨ y ¨rate¨, respectivamente.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Namespace Documentation

5.1 grpc_wrapper Namespace Reference

Classes

- class PositionServiceServicer

Variables

- **servicer** = PositionServiceServicer()

  Instancia del servidor gRPC.

5.1.1 Detailed Description

```
@file grpc_wrapper.py
@author Ian De La Garza
@date 28 de mayo de 2023
@brief Este archivo contiene la clase PositionServiceServicer que
sirve como servidor para enviar datos de posición a través de gRPC.
```

5.2 PositionService_pb2 Namespace Reference

Variables

- **_sym_db** = _symbol_database.Default()
- **DESCRIPTOR** = _descriptor_pool.Default().AddSerializedFile(b'\n%src/pycam/proto/PositionService.↩
  proto\x12\x0fobject_position\"\x07\n\x05\x45mpty\"3\n\x08Position\x12\t\n\x01x\x18\x01 \x01(\x01\x12\t\n\x01y\x18\x02
  \x01(\x01\x12\x11\n\ttimestamp\x18\x03 \x01(\x03\x32T\n\x0fPositionService\x12\x41\n\x0cSendPosition\x12\x16.↩
  object_position.Empty\x1a\x19.object_position.Positionb\x06proto3')
- **_options**
- **_serialized_start**
- **_serialized_end**

5.2.1 Detailed Description

```
Generated protocol buffer code.
```

5.3 PositionService_pb2_grpc Namespace Reference

Classes

- class PositionService
- class PositionServiceServicer
- class PositionServiceStub

Functions

- **add_PositionServiceServicer_to_server** (servicer, server)

5.3.1 Detailed Description

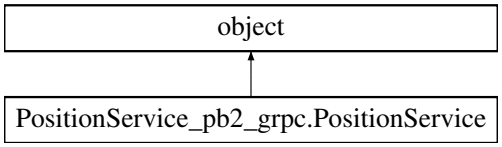
```
Client and server classes corresponding to protobuf-defined services.
```

Chapter 6

Class Documentation

6.1 PositionService_pb2_grpc.PositionService Class Reference

Inheritance diagram for PositionService_pb2_grpc.PositionService:



Static Public Member Functions

- **SendPosition** (request, target, options=(), channel_credentials=None, call_credentials=None, insecure=False, compression=None, wait_for_ready=None, timeout=None, metadata=None)

6.1.1 Detailed Description

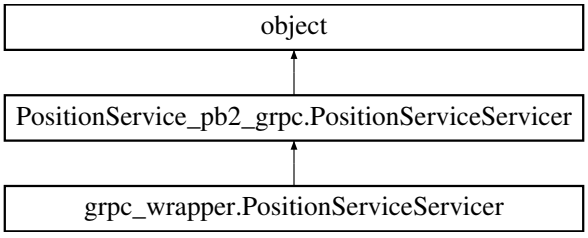
`The service definition.`

The documentation for this class was generated from the following file:

- src/pycam/src/PositionService_pb2_grpc.py

6.2 grpc_wrapper.PositionServiceServicer Class Reference

Inheritance diagram for grpc_wrapper.PositionServiceServicer:



Public Member Functions

- __init__ (self)
- callback (self, data)
- SendPosition (self, empty, context)
- serve (self)
- serverStop (self)

- SendPosition (self, request, context)

Public Attributes

- **latest_message**

  Ultimo mensaje de posición recibido.
- **callback**

  Callback para el suscriptor ROS.
- server

  Callback para el suscriptor ROS.

6.2.1 Detailed Description

```
@brief Servidor gRPC que transmite datos de posición.
```

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
grpc_wrapper.PositionServiceServicer.__init__ (
              self )
```

```
@brief Inicializa el nodo ROS, se suscribe a "/camera/marker_center"
y configura el servidor gRPC.
```

6.2.3 Member Function Documentation

6.2.3.1 callback()

```
grpc_wrapper.PositionServiceServicer.callback (
              self,
              data )
```

```
@brief Callback para el suscriptor ROS, actualiza el mensaje más reciente.

@param data El nuevo mensaje de posición a guardar.
```

6.2.3.2 SendPosition()

```
grpc_wrapper.PositionServiceServicer.SendPosition (
              self,
              empty,
              context )
```

```
@brief Envía los datos de posición actuales a través de gRPC.

@param empty El objeto vacío enviado por el cliente, no se utiliza.
@param context El contexto gRPC, no se utiliza.

@return Una respuesta de posición que contiene los datos de posición más recientes.
```

Reimplemented from PositionService_pb2_grpc.PositionServiceServicer.

6.2.3.3 serve()

```
grpc_wrapper.PositionServiceServicer.serve (
              self )
```

```
@brief Inicia el servidor gRPC.
```

6.2.3.4 serverStop()

```
grpc_wrapper.PositionServiceServicer.serverStop (
              self )
```

```
@brief Detiene el servidor gRPC.
```

6.2.4 Member Data Documentation

6.2.4.1 server

```
grpc_wrapper.PositionServiceServicer.server
```

Callback para el suscriptor ROS.

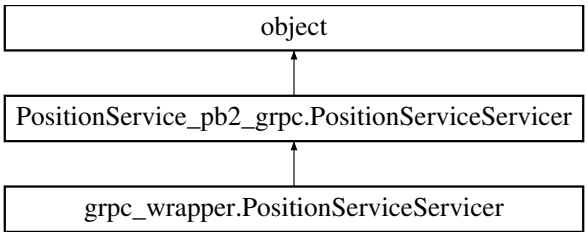
Servidor gRPC.

The documentation for this class was generated from the following file:

- src/pycam/src/grpc_wrapper.py

6.3 PositionService_pb2_grpc.PositionServiceServicer Class Reference

Inheritance diagram for PositionService_pb2_grpc.PositionServiceServicer:



Public Member Functions

- SendPosition (self, request, context)

6.3.1 Detailed Description

```
The service definition.
```

6.3.2 Member Function Documentation

6.3.2.1 SendPosition()

```
PositionService_pb2_grpc.PositionServiceServicer.SendPosition (
                self,
                request,
                context )
```

```
Sends a greeting
```

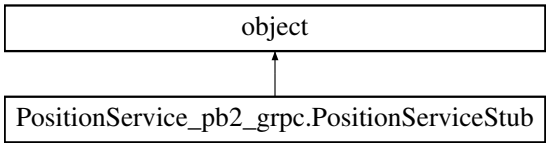
Reimplemented in grpc_wrapper.PositionServiceServicer.

The documentation for this class was generated from the following file:

- src/pycam/src/PositionService_pb2_grpc.py

6.4 PositionService_pb2_grpc.PositionServiceStub Class Reference

Inheritance diagram for PositionService_pb2_grpc.PositionServiceStub:



Public Member Functions

- __init__ (self, channel)

Public Attributes

- **SendPosition**

6.4.1 Detailed Description

```
The service definition.
```

6.4.2 Constructor & Destructor Documentation

6.4.2.1 __init__()

```
PositionService_pb2_grpc.PositionServiceStub.__init__ (
              self,
              channel )
```

```
Constructor.

Args:
    channel: A grpc.Channel.
```

The documentation for this class was generated from the following file:

- src/pycam/src/PositionService_pb2_grpc.py

Index