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.

Se puede un video del proyecto funcionando en https://drive.google.com/file/d/1gf↩KTAAFY7mNjZSFFG5LzwTAjmISHOkOd/view?usp=sharing

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 C++

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;
                //call multiply .so function
                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, debses 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.

1.6 Paso 2: Creación de una librería en C++ para multiplicar las coordenadas por 100

1.6.1 Código Fuente

El código fuente para este paso se encuentra en los archivos `multiply.h` y `multiply.cpp`.

`multiply.cpp`

```
extern "C" {
    void multiply(int* x, int* y) {
        *x *= 100;
        *y *= 100;
    }
}
```

`multiply.h`

```
#ifdef __cplusplus
extern "C" {
#endif
void multiply(int* a, int* b);

#ifdef __cplusplus
}
#endif
```

1.6.2 Descripción del Código

En esta etapa, creamos una librería en C++ que proporciona una función para multiplicar las coordenadas por 100.

El archivo `multiply.h` contiene la declaración de la función `multiply` que toma dos punteros a enteros y multiplica sus valores por 100.

El archivo `multiply.cpp` contiene la implementación de la función `multiply`. En esta implementación, los valores apuntados por los punteros se multiplican por 100.

1.7 Paso 3: Integración de la librería C++ en el ROS Node para realizar la operación de multiplicación.

1.7.1 Uso del Código

Para utilizar esta librería en tu proyecto, debes incluir el archivo de encabezado `multiply.h` en tus archivos fuente donde necesites utilizar la función `multiply`. Luego, enlaza tu proyecto con el archivo objeto generado a partir del archivo fuente `multiply.cpp`.

Puedes llamar a la función `multiply` pasando los valores que deseas multiplicar por 100 como argumentos. Los valores se multiplicarán directamente en su ubicación de memoria. Por lo tanto, no es necesario devolver un valor de la función. Por ejemplo en enable_camera.cpp su puede ver como se utiliza la librería:

```
int x = center.x;
int y = center.y;
//call multiply .so function
multiply(&x, &y);
```

Para lograr que el codigo usara el .so al compilar se tuvo que agregar lo siguiente en el `CMakelists.txt`:

```
include_directories(
    ${CMAKE_CURRENT_SOURCE_DIR}/include
)
target_link_libraries(enable_camera ${OpenCV_LIBRARIES} ${catkin_LIBRARIES}
      ${CMAKE_CURRENT_SOURCE_DIR}/lib/libmultiply.so)
install(DIRECTORY include/${PROJECT_NAME}/
  DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
  FILES_MATCHING PATTERN "*.h"
  PATTERN ".svn" EXCLUDE
)
```

1.7.2 Como utilizar una libreria compartida (.so) en python

Si le interesa saber como utilizar un .so en python puede ver la siguiente pagina agregar_so_python

1.8 Paso 4: Publicación de las coordenadas resultantes en un ROS Topic junto con su timestamp

En este paso del proyecto, se publican las coordenadas resultantes en un ROS Topic junto con su timestamp. Esto permite que otros nodos de ROS puedan suscribirse a este topic y obtener las coordenadas actualizadas del objeto.

El proceso de publicación de las coordenadas se realiza utilizando el framework ROS y su sistema de publicación y suscripción de mensajes. A continuación, se describe el flujo de trabajo para realizar esta tarea:

1. Definir un mensaje: Antes de publicar las coordenadas, se debe definir un mensaje personalizado en ROS que contenga los campos necesarios para almacenar las coordenadas X e Y, así como el timestamp correspondiente. Para ello se escogio un mensaje de tipo PointStamped que ya viene definido en ROS. Este mensaje contiene los campos necesarios para almacenar las coordenadas X e Y, así como el timestamp correspondiente.

2. Crear un publicador: En el nodo responsable de la detección y cálculo de las coordenadas, se debe crear un objeto publicador que esté configurado para publicar mensajes en el topic específico. El publicador se crea utilizando la biblioteca de ROS y se especifica el tipo de mensaje que se publicará (el mensajes definido en el paso anterior). Esto se puede ver en el archivo `enable_camera.cpp`:
   ```
   ros::Publisher marker_center_pub = nh.advertise<geometry_msgs::PointStamped>("camera/marker_center",
   1);
   ```

3. Actualizar y publicar las coordenadas: En cada iteración del bucle de detección y cálculo de coordenadas, se actualizan los valores de las coordenadas y se establece el timestamp actual. A continuación, se empaquetan los valores de las coordenadas y el timestamp en un objeto de mensaje y se publica a través del publicador creado en el paso anterior. Esto se puede ver en el archivo `enable_camera.cpp`:
   ```
   // 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;
       //call multiply .so function
       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);
        }
```

Una vez que las coordenadas se publican en el topic, otros nodos de ROS pueden suscribirse a este topic y recibir las coordenadas actualizadas en tiempo real. Esto permite la integración con otros componentes del sistema o la realización de acciones específicas basadas en las coordenadas detectadas.

1.9 Paso 5: Desarrollo de un Wrapper de gRPC para convertir el ROS Topic en un servicio RPC

En este paso del proyecto, se desarrolla un Wrapper de gRPC que convierte el ROS Topic de las coordenadas en un servicio RPC. Esto permite la comunicación entre el nodo ROS y otros programas o sistemas que pueden consumir el servicio RPC.

El Wrapper de gRPC se implementa utilizando el lenguaje de programación Python y las bibliotecas de gRPC y ROS. A continuación, se describe el flujo de trabajo para convertir el ROS Topic en un servicio RPC utilizando el Wrapper de gRPC:

1. Definir el servicio RPC: Antes de implementar el Wrapper de gRPC, se debe definir el servicio RPC que se proporcionará. El servicio debe tener una interfaz clara que especifique los métodos que estarán disponibles y los tipos de datos que se utilizarán. Para esto, se utiliza la herramienta `protoc` para definir el mensaje enviado por gRPC. El archivo `.proto` contiene la definición del mensaje y se utiliza para generar los archivos de código necesarios. A continuación se puede observar el archivo `.proto` usado en este proyecto pycam/proto/PositionService.proto:

```
syntax = "proto3";

package object_position;
message Empty {
}
 // El mensaje de respuesta que contiene la posición
 message Position {
   double x = 1;
   double y = 2;
   int64 timestamp = 3; // Este campo representa el timestamp del mensaje.
 }

 // La definición del servicio
 service PositionService {
   // Envía una posición
   rpc SendPosition (Empty) returns (Position);
 }
```

2. Generar los archivos de código: Utilizando el compilador de gRPC y el archivo `.proto`, se deben generar los archivos de código necesarios para implementar el servicio RPC. Estos archivos incluyen los stubs del cliente y del servidor, así como los mensajes definidos en el servicio. Para generar los archivos se utilizo el siguiente comando:

```
$ python -m grpc_tools.protoc --python_out=. --grpc_python_out=. nombre_del_archivo.proto
```

   Y obtuvimos los archivos PositionService_pb2.py y PositionService_pb2_grpc.py que se pueden ver en la carpeta `pycam/src/`.

3. Configurar y ejecutar el wrapper gRPC: Se crea un wrapper cuya función es configurar el servidor gRPC para que escuche en un puerto específico y acepte conexiones entrantes. Y ademas el wrapper va acruat om un nodo de ros el cual se va a subscribir al topico de posición para siempre tener el valor mas actualizado. De esta manera cuando un cliente quiere obtener la posición del objeto, el wrapper le va a devolver la posición que tiene guardada en el momento. Se puede observar el codigo implementado en el archivo grpc_wrapper.py:

```python
#!/usr/bin/env python3


"""
@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.
"""

import time
from concurrent import futures

import grpc
import rospy
from geometry_msgs.msg import PointStamped

import PositionService_pb2
import PositionService_pb2_grpc

class PositionServiceServicer(PositionService_pb2_grpc.PositionServiceServicer):
    """
    @brief Servidor gRPC que transmite datos de posición.
    """
    def __init__(self):
        """
        @brief Inicializa el nodo ROS, se suscribe a "/camera/marker_center"
        y configura el servidor gRPC.
        """

        self.latest_message = PointStamped()
        rospy.init_node('position_listener', anonymous=True)


        rospy.Subscriber("/camera/marker_center", PointStamped, self.callback)

        self.server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))


    def callback(self, data):
        """
        @brief Callback para el suscriptor ROS, actualiza el mensaje más reciente.

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

        self.latest_message = data

    def 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.
        """
        if self.latest_message is not None:
            response = PositionService_pb2.Position()
            response.x = self.latest_message.point.x
            response.y = self.latest_message.point.y
            response.timestamp = self.latest_message.header.stamp.nsecs
            return response
        else:
            raise grpc.RpcError("No position data received yet")

    def serve(self):
        """
        @brief Inicia el servidor gRPC.
        """
        PositionService_pb2_grpc.add_PositionServiceServicer_to_server(self, self.server)
        self.server.add_insecure_port('[::]:50051')
        self.server.start()

    def serverStop(self):
        """
        @brief Detiene el servidor gRPC.
        """
        self.server.stop(0)



if __name__ == '__main__':
    """
    @brief El punto de entrada del programa.
    Se crea una instancia de PositionServiceServicer y se inicia el servidor gRPC.
```

```
    Cuando se interrumpe el programa (p.ej., se presiona Ctrl+C), se detiene el servidor gRPC.
    """


    servicer = PositionServiceServicer()
    servicer.serve()
    try:
        rospy.spin()
    except KeyboardInterrupt:
        servicer.serverStop()
```

Una vez que se ha implementado el Wrapper de gRPC y configurado el servidor, otros programas o sistemas pueden consumir el servicio RPC para obtener las coordenadas del objeto. Esto permite la integración con diferentes plataformas y la comunicación eficiente de las coordenadas.

1.10 Paso 6: Implementación de un programa en Java que consume el servicio RPC y muestra las coordenadas en la terminal

En este paso del proyecto, se implementa un programa en Java que consume el servicio RPC generado en el paso anterior y muestra las coordenadas del objeto en la terminal.

A continuación, se describe el flujo de trabajo para implementar el programa en Java que consume el servicio RPC:

1. Configurar el entorno de desarrollo: Antes de comenzar, nos aseguramos de tener instalado el entorno de desarrollo de Java, como JDK (Java Development Kit) y un IDE (Integrated Development Environment) en nuestro caso usamos Visual Studio Code.
2. Importar el código generado: Importamos los archivos de código generados por el compilador de gRPC en el proyecto Java. Estos archivos incluyen los stubs del cliente y los mensajes definidos en el servicio RPC. Utilizamos Maven para manejar las dependencias y compilar el archivo .proto en el proyecto de Java.
3. Configurar la conexión al servicio RPC: En el programa Java, configuramos la conexión al servicio RPC proporcionando la dirección y el puerto del servidor gRPC. Se utilizo los stubs del cliente generados para establecer la conexión y crear un cliente gRPC.
4. Llamar al método del servicio RPC: Utilizando el cliente gRPC, llamamos el método del servicio RPC que devuelve las coordenadas del objeto. Pasa los parámetros necesarios según la definición del servicio y captura la respuesta del servidor.
5. Procesar la respuesta y mostrar las coordenadas: Procesamos la respuesta recibida del servidor gRPC y muestra las coordenadas del objeto en la terminal, en el formato deseado.

El codigo se puede observar en el archivo `App.java`:

```
package com.example;

import com.google.protobuf.Empty;

import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import object_position.PositionServiceGrpc;
import object_position.PositionServiceOuterClass.Position;
public class App
{
    public static void main(String[] args) throws Exception {
        ManagedChannel channel = ManagedChannelBuilder.forTarget("192.168.55.1:50051")
        .usePlaintext()
        .build();

        // Reemplace 'MyServiceGrpc' y 'MyRequest' con los nombres de tu servicio y tu mensaje.
        PositionServiceGrpc.PositionServiceBlockingStub stub = PositionServiceGrpc.newBlockingStub(channel);
        try {
            while(true) {
                Position response = stub.sendPosition(null);
```

```
                System.out.println("Response x: " + response.getX() + "Response y: " + response.getY() + "
    Timestamp: " + response.getTimestamp());
                Thread.sleep(10);  // Add delay of 1 second between each request
            }
          } catch (InterruptedException e) {
              System.out.println("Shutting down due to user request.");
          } finally {
              channel.shutdown();
          }
      }
}
```

1.11 Paso 7: Uso de grpc-Gateway para exponer el servicio como un REST-API

En este paso del proyecto, se utiliza grpc-Gateway para exponer el servicio RPC generado anteriormente como un REST-API. Esto permite a otros sistemas o aplicaciones consumir el servicio RPC utilizando protocolos RESTful estándar.

A continuación, se describe el flujo de trabajo para utilizar grpc-Gateway y exponer el servicio RPC como un REST-API:

1. Configurar grpc-Gateway: Antes de comenzar, instalamos el entorno de desarrollo, se utilizo el github oficial de grpc-Gateway para obtener instrucciones sobre cómo instalarlo.
2. Definir el archivo de configuración: Generamos una copia del archivo proto y creamos otro arhivo configuración proto pero le agregamos las especificaciones de la configuración del gateway. En este archivo se definio cómo se mapean los endpoints REST a los métodos del servicio RPC. A continuación se puede observar el archivo `.proto` usado en este proyecto go/positionGateway/PositionServiceGo.proto↩ :

   ```
   syntax = "proto3";

   package object_position;
   option go_package = ".";
   import "google/api/annotations.proto";

   message Empty {
   }
   // The response message containing the greetings
   message Position {
     double x = 1;
     double y = 2;
     int64 timestamp = 3; // timestamp as UNIX epoch time
   }

   // The service definition.
   service PositionService {
     // Sends a greeting
     rpc SendPosition (Empty) returns (Position){
       option (google.api.http) = {
         get: "/position"
       };
     }
   }
   ```

3. Generar el servidor proxy: Utilizando el compilador de grpc-Gateway, generamoos el código del servidor proxy que actuará como intermediario entre las solicitudes REST y el servicio RPC. El servidor proxy se encargará de traducir las solicitudes REST a llamadas RPC y viceversa.
4. Implementar el servidor proxy: Implementamos el servidor proxy en el lenguaje de Go. El servidor proxy debe configurarse para escuchar en un puerto específico y gestionar las solicitudes REST entrantes.

La implementación del servidor proxy se puede observar en el archivo `go/main.go`:

```
package main

import (
    "context"
    "log"
    "net/http"

    "google.golang.org/grpc"
    "github.com/grpc-ecosystem/grpc-gateway/v2/runtime"

    pb "positionGate/positionGateway"
)

func main() {
    ctx := context.Background()
    mux := runtime.NewServeMux()

    // Create a gRPC connection to your existing Python gRPC server
    grpcServerEndpoint := "localhost:50051" // Replace with the actual address of your Python gRPC server
    opts := []grpc.DialOption{grpc.WithInsecure()}
    err := pb.RegisterPositionServiceHandlerFromEndpoint(ctx, mux, grpcServerEndpoint, opts)
    if err != nil {
        log.Fatalf("failed to register gateway: %v", err)
    }

    // Start the HTTP server
    httpAddr := ":8080" // Set the listening address for the gateway server
    log.Printf("gRPC gateway server listening on %s", httpAddr)
    if err := http.ListenAndServe(httpAddr, mux); err != nil {
        log.Fatalf("failed to serve: %v", err)
    }
}
```

1. Ejecutar el servidor proxy: Inicia el servidor proxy y asegúrate de que esté en funcionamiento y escuchando en el puerto especificado. Puedes probar el servidor proxy enviando solicitudes REST utilizando herramientas como cURL o Postman y verificar las respuestas recibidas.

Si quiere ver el servidor proxy funcionando puedes ver el video de la demostración del proyecto que esta al inicio.

1.12 Paso 8: Adquisición de las coordenadas del objeto desde el nuevo REST-API utilizando Postman o Flask, y guardar los datos en un archivo JSON

A continuación, se describe el flujo de trabajo para adquirir las coordenadas del objeto utilizando Postman o Flask:

1. Configurar el entorno: Insalamos Postman. Postman es una herramienta de desarrollo de API que permite enviar solicitudes HTTP y recibir respuestas.
2. Crear una solicitud HTTP: Utilizando Postman, creamos una solicitud HTTP a la ruta específica del REST-API que proporciona las coordenadas del objeto.
3. Enviar la solicitud: Envíamos la solicitud HTTP al REST-API y esperamos la respuesta. Se puede ver la respuesta directamente en la interfaz de Postman.
4. Guardar los datos en un archivo JSON: Una vez recibido la respuesta del REST-API, extramos los datos de coordenadas del objeto y guardarlos en un archivo JSON.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all 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

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/api/source_info.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/api/system_parameter.pro
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/api/usage.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/cloud/audit/audit_log.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/geo/type/viewport.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/logging/type/http_request.
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/logging/type/log_severity.p
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/longrunning/operations.pro
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/rpc/code.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/rpc/error_details.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/rpc/status.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/rpc/context/attribute_conte
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/calendar_period.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/color.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/date.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/datetime.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/dayofweek.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/expr.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/fraction.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/latlng.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/money.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/postal_address.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/quaternion.proto
??
src/pycam/windows/demo/target/protoc-dependencies/626bed7ab4ce352886b6de85447453e5/google/type/timeofday.proto
??

Chapter 6

Namespace Documentation

6.1 Package com.example

Classes

- class App
- class AppTest

6.2 grpc_wrapper Namespace Reference

Classes

- class PositionServiceServicer

Variables

- servicer = PositionServiceServicer()

  Instancia del servidor gRPC.

6.2.1 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.
```

6.2.2 Variable Documentation

6.2.2.1 servicer

```
grpc_wrapper.servicer = PositionServiceServicer()
```

Instancia del servidor gRPC.

6.3 Package object_position

Classes

- class PositionServiceGrpc
- class PositionServiceOuterClass

6.4 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

6.4.1 Detailed Description

```
Generated protocol buffer code.
```

6.4.2 Variable Documentation

6.4.2.1 _options

```
PositionService_pb2._options  [protected]
```

6.4.2.2 _serialized_end

```
PositionService_pb2._serialized_end  [protected]
```

6.4.2.3 _serialized_start

```
PositionService_pb2._serialized_start  [protected]
```

6.4.2.4 _sym_db

```
PositionService_pb2._sym_db = _symbol_database.Default()  [protected]
```

6.4.2.5 DESCRIPTOR

```
PositionService_pb2.DESCRIPTOR = _descriptor_pool.Default().AddSerializedFile(b'\n%src/pycam/proto/Position↩
Service.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\x0f↩
PositionService\x12\x41\n\x0cSendPosition\x12\x16.object_position.Empty\x1a\x19.object_↩
position.Positionb\x06proto3')
```

6.5 PositionService_pb2_grpc Namespace Reference

Classes

- class PositionService
- class PositionServiceServicer
- class PositionServiceStub

Functions

- add_PositionServiceServicer_to_server (servicer, server)

6.5.1 Detailed Description

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

6.5.2 Function Documentation

6.5.2.1 add_PositionServiceServicer_to_server()

```
PositionService_pb2_grpc.add_PositionServiceServicer_to_server (
              servicer,
              server )
```

Chapter 7

Class Documentation

7.1 com.example.App Class Reference

Static Public Member Functions

- static void main (String[ ] args) throws Exception

  Punto de entrada del programa. Se establece una conexión gRPC con el servidor y se realizan solicitudes de posición continuas.

7.1.1 Member Function Documentation

7.1.1.1 main()

```
static void com.example.App.main (
            String[ ] args ) throws Exception  [inline], [static]
```

Punto de entrada del programa. Se establece una conexión gRPC con el servidor y se realizan solicitudes de posición continuas.

Parameters

args	Argumentos de línea de comandos, no se utilizan en este programa.

Exceptions

Exception	Si ocurre algún error durante la comunicación gRPC.

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

- src/pycam/windows/demo/src/main/java/com/example/App.java

7.2 com.example.AppTest Class Reference

Public Member Functions

- void shouldAnswerWithTrue ()

7.2.1 Detailed Description

Unit test for simple App.

7.2.2 Member Function Documentation

7.2.2.1 shouldAnswerWithTrue()

```
void com.example.AppTest.shouldAnswerWithTrue ( )  [inline]
```

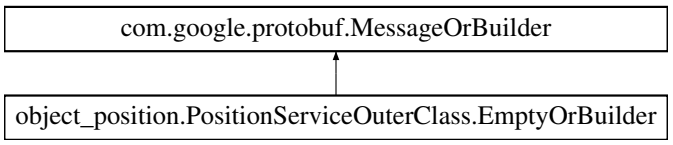
Rigorous Test :-)

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

- src/pycam/windows/demo/src/test/java/com/example/AppTest.java

7.3 object_position.PositionServiceOuterClass.EmptyOrBuilder Interface Reference

Inheritance diagram for object_position.PositionServiceOuterClass.EmptyOrBuilder:

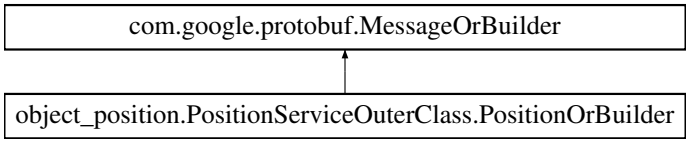


The documentation for this interface was generated from the following file:

- src/pycam/windows/demo/src/main/java/object_position/PositionServiceOuterClass.java

7.4 object_position.PositionServiceOuterClass.PositionOrBuilder Interface Reference

Inheritance diagram for object_position.PositionServiceOuterClass.PositionOrBuilder:



Public Member Functions

- double getX ()
- double getY ()
- long getTimestamp ()

7.4.1 Member Function Documentation

7.4.1.1 getTimestamp()

```
long object_position.PositionServiceOuterClass.PositionOrBuilder.getTimestamp ( )
```

```
timestamp as UNIX epoch time
```

```
int64 timestamp = 3;
```

Returns

The timestamp.

7.4.1.2 getX()

```
double object_position.PositionServiceOuterClass.PositionOrBuilder.getX ( )
```

```
double x = 1;
```

Returns

The x.

7.4.1.3 getY()

```
double object_position.PositionServiceOuterClass.PositionOrBuilder.getY ( )
```

```
double y = 2;
```

Returns

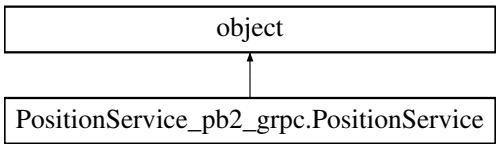
The y.

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

- src/pycam/windows/demo/src/main/java/object_position/PositionServiceOuterClass.java

7.5 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)

7.5.1 Detailed Description

```
The service definition.
```

7.5.2 Member Function Documentation

7.5.2.1 SendPosition()

```
PositionService_pb2_grpc.PositionService.SendPosition (
              request,
              target,
              options = (),
              channel_credentials = None,
              call_credentials = None,
              insecure = False,
              compression = None,
              wait_for_ready = None,
              timeout = None,
              metadata = None )  [static]
```

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

- src/pycam/src/PositionService_pb2_grpc.py

7.6 object_position.PositionServiceGrpc Class Reference

Classes

- class **PositionServiceBlockingStub**
- class **PositionServiceFutureStub**
- class **PositionServiceImplBase**
- class **PositionServiceStub**

Static Public Member Functions

- static io.grpc.MethodDescriptor< object_position.PositionServiceOuterClass.Empty, object_position.↩ PositionServiceOuterClass.Position > getSendPositionMethod ()
- static PositionServiceStub newStub (io.grpc.Channel channel)
- static PositionServiceBlockingStub newBlockingStub (io.grpc.Channel channel)
- static PositionServiceFutureStub newFutureStub (io.grpc.Channel channel)
- static io.grpc.ServiceDescriptor getServiceDescriptor ()

Static Public Attributes

- static final String SERVICE_NAME = ¨object_position.PositionService¨

7.6.1 Detailed Description

```
The service definition.
```

7.6.2 Member Function Documentation

7.6.2.1 getSendPositionMethod()

```
static io.grpc.MethodDescriptor< object_position.PositionServiceOuterClass.Empty, object_↩
position.PositionServiceOuterClass.Position > object_position.PositionServiceGrpc.getSend↩
PositionMethod ( )  [inline], [static]
```

7.6.2.2 getServiceDescriptor()

```
static io.grpc.ServiceDescriptor object_position.PositionServiceGrpc.getServiceDescriptor ( )
[inline], [static]
```

7.6.2.3 newBlockingStub()

```
static PositionServiceBlockingStub object_position.PositionServiceGrpc.newBlockingStub (
            io.grpc.Channel channel )  [inline], [static]
```

Creates a new blocking-style stub that supports unary and streaming output calls on the service

7.6.2.4 newFutureStub()

```
static PositionServiceFutureStub object_position.PositionServiceGrpc.newFutureStub (
            io.grpc.Channel channel )  [inline], [static]
```

Creates a new ListenableFuture-style stub that supports unary calls on the service

7.6.2.5 newStub()

```
static PositionServiceStub object_position.PositionServiceGrpc.newStub (
            io.grpc.Channel channel )  [inline], [static]
```

Creates a new async stub that supports all call types for the service

7.6.3 Member Data Documentation

7.6.3.1 SERVICE_NAME

```
final String object_position.PositionServiceGrpc.SERVICE_NAME = ¨object_position.Position↩
Service¨  [static]
```

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

- src/pycam/windows/demo/src/main/java/object_position/PositionServiceGrpc.java

7.7 object_position.PositionServiceOuterClass Class Reference

Classes

- class **Empty**
- interface EmptyOrBuilder
- class **Position**
- interface PositionOrBuilder

Static Public Member Functions

- static void registerAllExtensions (com.google.protobuf.ExtensionRegistryLite registry)
- static void registerAllExtensions (com.google.protobuf.ExtensionRegistry registry)
- static com.google.protobuf.Descriptors.FileDescriptor getDescriptor ()

7.7.1 Member Function Documentation

7.7.1.1 getDescriptor()

```
static com.google.protobuf.Descriptors.FileDescriptor object_position.PositionServiceOuter↩
Class.getDescriptor ( )  [inline], [static]
```

7.7.1.2 registerAllExtensions() [1/2]

```
static void object_position.PositionServiceOuterClass.registerAllExtensions (
            com.google.protobuf.ExtensionRegistry registry )  [inline], [static]
```

7.7.1.3 registerAllExtensions() [2/2]

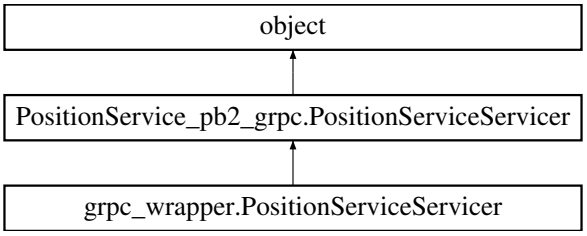
```
static void object_position.PositionServiceOuterClass.registerAllExtensions (
            com.google.protobuf.ExtensionRegistryLite registry )  [inline], [static]
```

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

- src/pycam/windows/demo/src/main/java/object_position/PositionServiceOuterClass.java

7.8 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.

7.8.1 Detailed Description

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

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

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

7.8.3 Member Function Documentation

7.8.3.1 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.
```

7.8.3.2 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.

7.8.3.3 serve()

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

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

7.8.3.4 serverStop()

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

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

7.8.4 Member Data Documentation

7.8.4.1 callback

```
grpc_wrapper.PositionServiceServicer.callback
```

Callback para el suscriptor ROS.

7.8.4.2 latest_message

```
grpc_wrapper.PositionServiceServicer.latest_message
```

Ultimo mensaje de posición recibido.

7.8.4.3 server

```
grpc_wrapper.PositionServiceServicer.server
```

Callback para el suscriptor ROS.

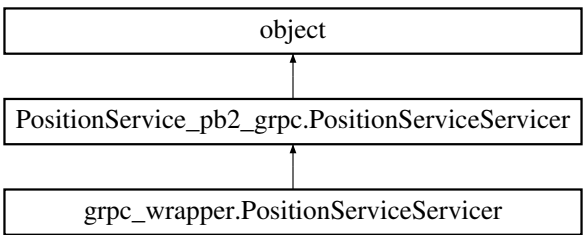
Servidor gRPC.

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

- src/pycam/src/grpc_wrapper.py

7.9 PositionService_pb2_grpc.PositionServiceServicer Class Reference

Inheritance diagram for PositionService_pb2_grpc.PositionServiceServicer:



Public Member Functions

- SendPosition (self, request, context)

7.9.1 Detailed Description

```
The service definition.
```

7.9.2 Member Function Documentation

7.9.2.1 SendPosition()

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

```
Sends a greeting
```

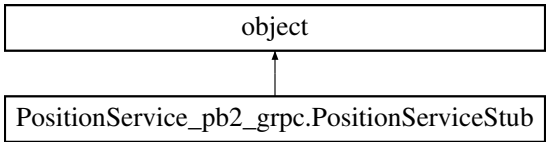
Reimplemented in grpc_wrapper.PositionServiceServicer.

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

- src/pycam/src/PositionService_pb2_grpc.py

7.10 PositionService_pb2_grpc.PositionServiceStub Class Reference

Inheritance diagram for PositionService_pb2_grpc.PositionServiceStub:



Public Member Functions

- __init__ (self, channel)

Public Attributes

- SendPosition

7.10.1 Detailed Description

```
The service definition.
```

7.10.2 Constructor & Destructor Documentation

7.10.2.1 __init__()

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

Constructor.

Args:
    channel: A grpc.Channel.

7.10.3 Member Data Documentation

7.10.3.1 SendPosition

```
PositionService_pb2_grpc.PositionServiceStub.SendPosition
```

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

- src/pycam/src/PositionService_pb2_grpc.py

Chapter 8

File Documentation

8.1 src/pycam/documentation/documentacion.dox File Reference

Functions

- page agregar_so_python Agregar un archivo de biblioteca compartida (.so) a Python utilizando ctypes La biblioteca ctypes de Python proporciona una forma de acceder a funciones en bibliotecas compartidas desde Python sin necesidad de escribir extensiones de Python en C. A continuación

Variables

- page agregar_so_python Agregar un archivo de biblioteca se muestra una guía paso a paso para agregar un archivo so a Python utilizando ctypes section paso1 Paso
- page agregar_so_python Agregar un archivo de biblioteca se muestra una guía paso a paso para agregar un archivo so a Python utilizando ctypes section paso1 puedes utilizar el tipo ctypes POINTER para crear un puntero a un tipo de datos específico Por ejemplo
- page agregar_so_python Agregar un archivo de biblioteca se muestra una guía paso a paso para agregar un archivo so a Python utilizando ctypes section paso1 puedes utilizar el tipo ctypes POINTER para crear un puntero a un tipo de datos específico Por si tienes una función en la biblioteca compartida que toma un puntero a un entero

8.1.1 Function Documentation

8.1.1.1 compartida()

```
page agregar_so_python Agregar un archivo de biblioteca compartida (
            .  so )
```

8.1.2 Variable Documentation

8.1.2.1 ejemplo

```
page agregar_so_python Agregar un archivo de biblioteca se muestra una guía paso a paso para
agregar un archivo so a Python utilizando ctypes section paso1 puedes utilizar el tipo ctypes
POINTER para crear un puntero a un tipo de datos específico Por ejemplo
```

8.1.2.2 entero

page agregar_so_python Agregar un archivo de biblioteca se muestra una guía paso a paso para agregar un archivo so a Python utilizando ctypes section paso1 puedes utilizar el tipo ctypes POINTER para crear un puntero a un tipo de datos específico Por si tienes una función en la biblioteca compartida que toma un puntero a un entero

8.1.2.3 Paso

page agregar_so_python Agregar un archivo de biblioteca se muestra una guía paso a paso para agregar un archivo so a Python utilizando ctypes section paso1 Paso

8.2 src/pycam/go/positionGateway/PositionServiceGo.proto File Reference

8.3 src/pycam/include/pycam/multiply.cpp File Reference

Archivo que contiene la definición de la función multiplicar.

Functions

- void multiply (int *x, int *y)
  Multiplica dos enteros por 100.

8.3.1 Detailed Description

Archivo que contiene la definición de la función multiplicar.

8.3.2 Function Documentation

8.3.2.1 multiply()

```
void multiply (
            int * x,
            int * y )
```

Multiplica dos enteros por 100.

Esta es una función externa ¨C¨ que puede ser llamada desde otros lenguajes de programación. Toma dos punteros a enteros, multiplica los enteros por 100, y almacena el resultado de nuevo en las ubicaciones de memoria originales a las que apuntan los punteros.

Parameters

x	Un puntero al primer entero a ser multiplicado por 100.
y	Un puntero al segundo entero a ser multiplicado por 100.

8.4 src/pycam/include/pycam/multiply.h File Reference

Archivo de cabecera para la función multiplicar.

Functions

- void multiply (int *a, int *b)

  Multiplica dos enteros por 100.

8.4.1 Detailed Description

Archivo de cabecera para la función multiplicar.

8.4.2 Function Documentation

8.4.2.1 multiply()

```
void multiply (
            int * x,
            int * y )
```

Multiplica dos enteros por 100.

Esta función es parte de una interfaz de C que puede ser usada por código C++. La función toma dos punteros a enteros, los multiplica por 100, y guarda el resultado en las ubicaciones de memoria originales.

Parameters

a	Un puntero al primer entero que se multiplicará por 100.
b	Un puntero al segundo entero que se multiplicará por 100.

Esta es una función externa ¨C¨ que puede ser llamada desde otros lenguajes de programación. Toma dos punteros a enteros, multiplica los enteros por 100, y almacena el resultado de nuevo en las ubicaciones de memoria originales a las que apuntan los punteros.

Parameters

x	Un puntero al primer entero a ser multiplicado por 100.
y	Un puntero al segundo entero a ser multiplicado por 100.

8.5 multiply.h

Go to the documentation of this file.

```
00001
00006 #ifdef __cplusplus
00007 extern "C" {
```

```
00008 #endif
00019 void multiply(int* a, int* b);
00020
00021 #ifdef __cplusplus
00022 }
00023 #endif
```

8.6 src/pycam/proto/PositionService.proto File Reference

8.7 src/pycam/windows/demo/src/main/resources/proto/PositionService.proto File Reference

8.8 src/pycam/windows/demo/target/classes/PositionService.proto File Reference

8.9 src/pycam/windows/demo/target/classes/proto/PositionService.proto File Reference

8.10 src/pycam/src/enable_camera.cpp File Reference

```
#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¨
```

Functions

- int main (int argc, char ∗∗argv)

  Función principal.

8.10.1 Function Documentation

8.10.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

Función principal.

Inicializa el nodo ROS, configura los publicadores y se suscribe a los temas relevantes. Captura imágenes de la cámara y realiza la detección de marcadores ARUCO, luego publica las imágenes y las posiciones de los marcadores detectados.

Parameters

argc	Número de argumentos.
argv	Argumentos de la línea de comandos.

Returns

0 si se ejecuta con éxito, -1 si no se pudo abrir la cámara.

[multiply example]

[multiply example]

8.11 src/pycam/src/grpc_wrapper.py File Reference

Classes

- class grpc_wrapper.PositionServiceServicer

Namespaces

- namespace grpc_wrapper

Variables

- grpc_wrapper.servicer = PositionServiceServicer()

  Instancia del servidor gRPC.

8.12 src/pycam/src/PositionService_pb2.py File Reference

Namespaces

- namespace PositionService_pb2

Variables

- PositionService_pb2._sym_db = _symbol_database.Default()
- PositionService_pb2.DESCRIPTOR = _descriptor_pool.Default().AddSerializedFile(b'\n%src/pycam/proto/Position↩
  Service.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')
- PositionService_pb2._options
- PositionService_pb2._serialized_start
- PositionService_pb2._serialized_end

8.13 src/pycam/src/PositionService_pb2_grpc.py File Reference

Classes

- class PositionService_pb2_grpc.PositionServiceStub
- class PositionService_pb2_grpc.PositionServiceServicer
- class PositionService_pb2_grpc.PositionService

Namespaces

- namespace PositionService_pb2_grpc

Functions

- PositionService_pb2_grpc.add_PositionServiceServicer_to_server (servicer, server)

8.14 src/pycam/windows/demo/src/main/java/com/example/App.java File Reference

Este archivo contiene la clase App, que implementa un cliente gRPC para el servicio PositionService.

Classes

- class com.example.App

Packages

- package com.example

8.14.1 Detailed Description

Este archivo contiene la clase App, que implementa un cliente gRPC para el servicio PositionService.

Author

Tu nombre

Date

28 de mayo de 2023

8.15 src/pycam/windows/demo/src/main/java/object_position/PositionServiceGrpc.java File Reference

Classes

- class object_position.PositionServiceGrpc
- class **object_position.PositionServiceGrpc.PositionServiceImplBase**
- class **object_position.PositionServiceGrpc.PositionServiceStub**
- class **object_position.PositionServiceGrpc.PositionServiceBlockingStub**
- class **object_position.PositionServiceGrpc.PositionServiceFutureStub**

Packages

- package object_position

8.16 src/pycam/windows/demo/src/main/java/object_position/PositionServiceOuterClass.java File Reference

Classes

- class object_position.PositionServiceOuterClass
- interface object_position.PositionServiceOuterClass.EmptyOrBuilder
- class **object_position.PositionServiceOuterClass.Empty**
- class **object_position.PositionServiceOuterClass.Empty.Builder**
- interface object_position.PositionServiceOuterClass.PositionOrBuilder
- class **object_position.PositionServiceOuterClass.Position**
- class **object_position.PositionServiceOuterClass.Position.Builder**

Packages

- package object_position

8.17 src/pycam/windows/demo/src/test/java/com/example/AppTest.java File Reference

Classes

- class com.example.AppTest

Packages

- package com.example

8.18 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/any.proto File Reference

8.19 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/api.proto File Reference

8.20 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/compiler/p File Reference

8.21 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/descriptor. File Reference

8.22 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/duration.pr File Reference

8.23 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/empty.prot File Reference

8.24 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/field_mask.proto File Reference

8.25 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/source_context.proto File Reference

8.26 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/struct.prot File Reference

8.27 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/timestamp File Reference

8.28 src/pycam/windows/demo/target/protoc-dependencies/3ab7ef9f8540c2917da277871632308e/google/protobuf/type.proto File Reference

8.29 src/pycam/windows/demo/target/protoc-

Index