OPENCV/CV_BRIDGE

ROBOTICS

POLITECNICO
MILANO 1863

CAMERAS

OPENCV

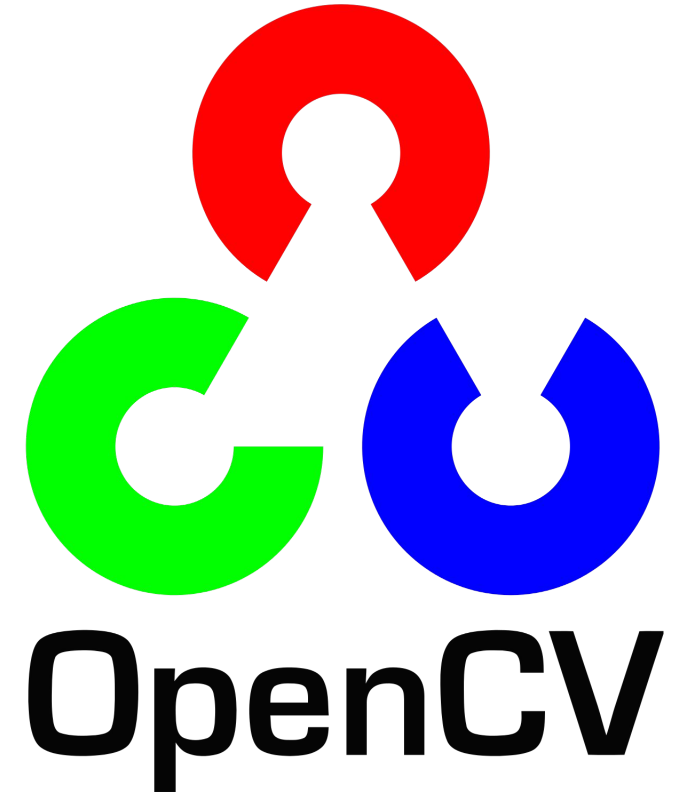
-One of the most used library for computer vision

-Fully integrated in ROS

-ROS package for topic to cv::Mat conversion

```
sudo apt-get install
ros-kinetic-opencv3
```

```
sudo apt-get install
ros-kinetic-cv-bridge
```

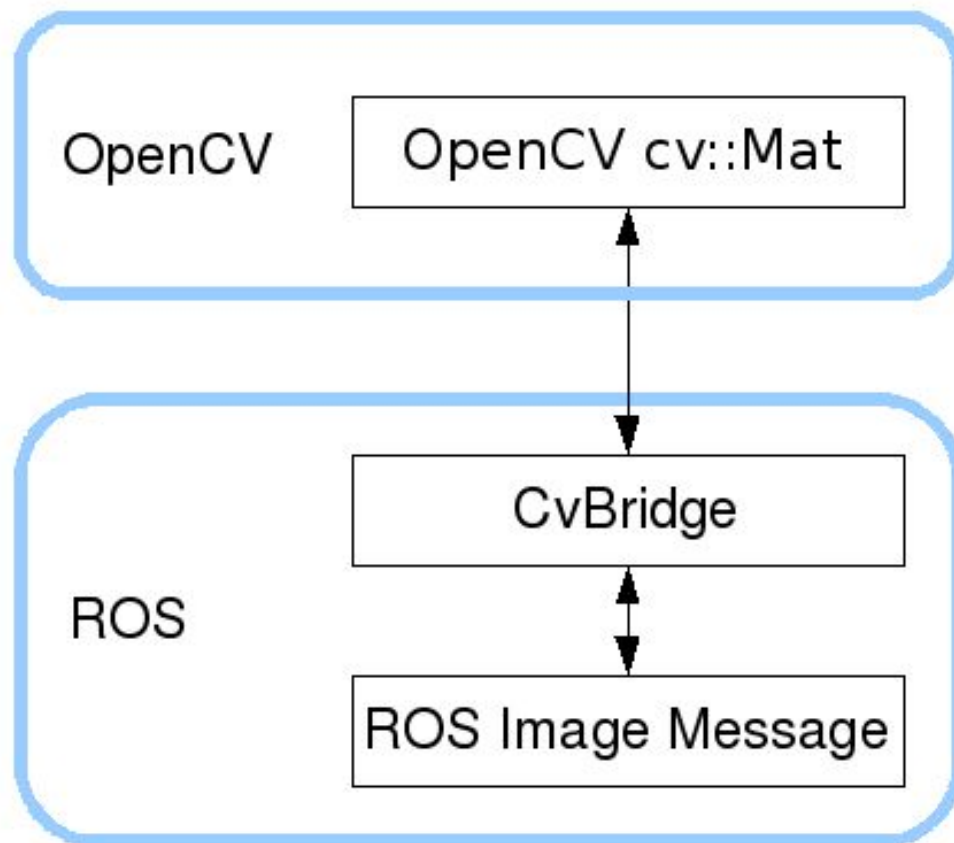


CvBridge

-ROS uses sensor_msgs/Image to transport images onto the ROS network

-OpenCV uses cv::Mat to store Images

-CVBridge is a package that handle the conversion between the two formats



cv_example.cpp

```
#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/opencv.hpp>
```

ROS include

Image transport include

CV bridge

Opencv library include

cv_example.cpp

```
int main(int argc, char** argv)
{
  ros::init(argc, argv, "image_converter");
  ImageConverter ic;
  ros::spin();
  return 0;
}
```

Main function:

Init ros
Image converter
keep spinning

cv_example.cpp

```
using namespace cv;


class ImageConverter
{
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Subscriber image_sub_;
  image_transport::Publisher image_pub_;
```

Create Node Handle

Create Image transport

Subscribe and publish images using Image transport

cv_example.cpp

```
public:
 ImageConverter()
  : it_(nh_)
 {

  image_sub_ = it_.subscribe("/rgb/image_rect_color", 1,
   &ImageConverter::imageCb, this);
  image_pub_ =
it_.advertise("/image_converter/output_video", 1);

  cv::namedWindow(OPENCV_WINDOW);
 }
```

Subscribe to image topic, using image transport

Publish elaborated image

Create opencv image for visualization

cv_example.cpp

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

Callback

Convert topic to opencv data

cv_example.cpp

```
int row = cv_ptr->image.rows;
int cols = cv_ptr->image.cols;
ROS_INFO ("row: %d- cols: %d", row, cols);
for (int i=0; i<cols;i++){ //cols
    for (int j=row/2.2; j<row-40;j++){ //row
        Vec3b color = cv_ptr->image.at<Vec3b>(Point(i,j));
        int B = color.val[0]; //B
        int G = color.val[1];
       int R = color.val[2];

        if (B>160 && G>160 && R>160 ){
        cv_ptr->image.at<Vec3b>(Point(i,j)) = Vec3b(255, 0, 255);
```

Example operation on the converted image

cv_example.cpp

```
  // Update GUI Window
  cv::imshow(OPENCV_WINDOW, cv_ptr->image);
  cv::waitKey(3);

  // Output modified video stream
  image_pub_.publish(cv_ptr->toImageMsg());
}
```

Show image in a window using OpenCV (annotation for `cv::imshow(OPENCV_WINDOW, cv_ptr->image);`)

Publish elaborated image (annotation for `image_pub_.publish(cv_ptr->toImageMsg());`)

CmakeLists.txt

```
find_package(catkin  REQUIRED COMPONENTS roscpp std_msgs geometry_msgs message_filters
image_transport cv_bridge )
```

← Add cv_bridge and image transport

```
find_package(OpenCV REQUIRED core highgui)
```

← Add OpenCV

```
include_directories(
  ${OpenCV_INCLUDE_DIRS}
  ${catkin_INCLUDE_DIRS}
  include
)
```

← Include both opencv and catkin library

```
link_libraries(
  ${OpenCV_LIBS}
  ${catkin_LIBRARIES}
)
```

← Link both opencv and catkin library

PCL

ROBOTICS

POLITECNICO
MILANO 1863

LIDAR

STEREO and RGBD CAMERAS

PCL

One of the most used library for PointCloud processing

Integrated in ROS to easily convert ros messages to pcl data

sudo apt-get install pcl

sudo apt-get install pcl-tools

sudo apt-get install ros-kinetic-pcl-conversions

sudo apt-get install ros-kinetic-pcl-msgs

sudo apt-get install ros-kinetic-pcl-ros

sudo apt-get install ros-kinetic-velodyne-pointcloud

rotate.cpp

```
#include "ros/ros.h"                                   ← ROS include


#include "std_msgs/String.h"
#include "sensor_msgs/PointCloud2.h"                   ← pcl include
#include <pcl_conversions/pcl_conversions.h>
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/common/transforms.h>
#include <pcl/filters/conditional_removal.h>
#include <pcl/point_types.h>
#include <velodyne_pointcloud/point_types.h>
#define VPoint velodyne_pointcloud::PointXYZIR         ← Define velodyne pointcloud
```

rotate.cpp

```
int main(int argc, char **argv)
{
    ROS_INFO ("Node rotate starting");
    ros::init(argc, argv, "rotate");
    pcl_process my_pcl_process;
    ros::spin();
    return 0;
}
```

Main function

initialize node

keep spinning

rotate.cpp

```
class pcl_process
{

public:
    pcl_process(){
    sub = n.subscribe("/velodyne_points", 1, &pcl_process::callback, this);

    pub = n.advertise<sensor_msgs::PointCloud2>("/velodyne_points_rotated", 1);


}
```

Our class

subscribe to belodyne

publish elaborated pointcloud

rotate.cpp (using pcl only)

```
void callback(const sensor_msgs::PointCloud2ConstPtr& point_cloud){        <------ callback
    pcl::PCLPointCloud2 pcl_pc2;
    pcl::PCLPointCloud2 pcl_pc2_out;
    pcl::PointCloud<VPoint>  projected;
    pcl_conversions::toPCL (*point_cloud,pcl_pc2);        <------ convert from message to pcl2
    pcl::PointCloud<VPoint>::Ptr pcl_cloud (new pcl::PointCloud<VPoint>);
    pcl::PointCloud<VPoint>::Ptr transformed_cloud (new pcl::PointCloud<VPoint>);
    pcl::fromPCLPointCloud2(pcl_pc2, *pcl_cloud);        <------ convert from pcl2 to pcl
```

rotate.cpp (using pcl only)

```
Eigen::Affine3f transform = Eigen::Affine3f::Identity();
    float theta = 4.9*0.0174533;
    transform.rotate (Eigen::AngleAxisf (theta, Eigen::Vector3f::UnitY()));
    pcl::transformPointCloud (*pcl_cloud, *transformed_cloud, transform);

    pcl::toPCLPointCloud2(*transformed_cloud, pcl_pc2_out);


    sensor_msgs::PointCloud2 output;
    pcl_conversions::fromPCL(pcl_pc2_out, output);
    pub.publish (output);
```

- `transform.rotate (...)` ← Create transform
- `pcl::transformPointCloud (...)` ← apply transform
- `pcl::toPCLPointCloud2(...)` ← go back to pcl2
- `pcl_conversions::fromPCL(...)` ← go back to ROS message
- `pub.publish (output);` ← publish

rotate_ros.cpp (using ros integration)

```
void callback(const sensor_msgs::PointCloud2 point_cloud){
    Eigen::Affine3f transform = Eigen::Affine3f::Identity();
    float theta = 4.9*0.0174533;
    transform.rotate (Eigen::AngleAxisf (theta, Eigen::Vector3f::UnitY()));
    Eigen::Matrix4f transformation;
    transformation = transform.matrix();

    sensor_msgs::PointCloud2 output;
    pcl_ros::transformPointCloud(transformation, point_cloud, output );

    pub.publish (output);
```

apply transformation directly to msg

publish

CmakeLists.txt

```
link_directories(${PCL_LIBRARY_DIRS})
find_package(catkin REQUIRED COMPONENTS  roscpp  pcl_conversions  pcl_ros   velodyne_pointcloud
)
find_package(PCL 1.8 REQUIRED)
add_definitions(${PCL_DEFINITIONS})
find_package(Eigen3 REQUIRED)
catkin_package(CATKIN_DEPENDS velodyne_pointcloud)
include_directories(
  include ${catkin_INCLUDE_DIRS}
  ${PCL_INCLUDE_DIRS}
  ${EIGEN3_INCLUDE_DIRS}
)
```

ROS Library

Libs needed only if you don't use ros_pcl but pcl without ROS integration

Needed to specify geometric transform

Launcher.launch

```
<launch>
  <node pkg="nodelet" type="nodelet" name="pcl_manager" args="manager" output="screen" />
   <node pkg="pcl_example" type="rotate_ros" name="rotate_ros" output="screen" />
  <node pkg="nodelet" type="nodelet" name="voxel_grid" args="load pcl/VoxelGrid pcl_manager" output="screen">
   <remap from="~input" to="/velodyne_points_rotated" />
   <rosparam>
    filter_field_name: z
    filter_limit_min: -1.1
    filter_limit_max: 3.0
    filter_limit_negative: False
    </rosparam>
  </node>
</launch>
```

Start the nodelet manager

Our node

Start the VaxelGrid node inside the nodelet