

dummy-mate

Documentation

Responsible: Kabács Máté

Developers: Kabács Máté

# \ USER DOCUMENTATION

## \ INTRODUCTION

### Package description

The dummy\_img\_mate node can concatenate 2 images horizontally and publish CAN message with different frequency. This message contains the publishing frequency of the image topic.

## \ Usage

### Dependencies

|  |  |
| --- | --- |
| **Dependency** | **Source** |
| roscpp | <https://github.com/ros/ros_comm> |
| rospy | <https://github.com/ros/ros_comm> |
| std\_msgs | <https://github.com/ros/std_msgs> |
| sensor\_msgs | <https://github.com/ros/common_msgs> |
| can\_msgs | <https://github.com/BME-FRT/roscan_open> |
| message\_runtime | <https://github.com/ros/message_runtime> |
| cv\_bridge | <https://github.com/ros-perception/vision_opencv> |
| image\_transport | <https://github.com/ros-perception/image_common> |
| message\_filters | <https://github.com/ros/ros_comm> |

### Installation

Here are the instructions on how to build your package:

1. cd [path\_to\_your\_ws]/src
2. git clone [repository]
3. cd [path\_to\_your\_ws]
4. catkin\_make

### Configuration

There is none.

### Launching

1. cd [path\_to\_your\_ws]
2. source devel/setup.bash
3. roslaunch [package] [node]

The remap function is working currently only if the bag file is started in the launch file. The given launch file does contain this.

### Topics

Subscribed topics

|  |  |  |
| --- | --- | --- |
| **Topic name** | **Type** | **Description** |
| /zed\_right\_img\_comp/compressed | sensor\_msgs/CompressedImage | The right ZED camera image for the dummy project. (This is a remapped topic) |
| /zed\_left\_img\_comp/compressed | sensor\_msgs/CompressedImage | The left ZED camera image for the dummy project. (This is a remapped topic) |

Published topics

|  |  |  |
| --- | --- | --- |
| **Topic name** | **Type** | **Description** |
| /dummy\_img\_mate | sensor\_msgs/Image | Contains the horizontally concatenated images. |
| /sent\_messages | can\_msgs/Frame | This is a CAN frame what is periodically published and it’s frequency can be changed. |

## \Sources

1. [Tutorial: Create Package](http://wiki.ros.org/ROS/Tutorials/CreatingPackage)
2. [Tutorial: Building a ROS Package](http://wiki.ros.org/ROS/Tutorials/BuildingPackages)
3. [Tutorial: Understanding ROS Topics](http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics)
4. [Tutorial: Examining the Simple Publisher and Subscriber](http://wiki.ros.org/ROS/Tutorials/ExaminingPublisherSubscriber)
5. [Tutorial: Recording and playing back data](http://wiki.ros.org/ROS/Tutorials/Recording%20and%20playing%20back%20data)
6. [Converting between ROS images and OpenCV images (C++)](http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages)
7. [message\_filters 4. Time Synchronizer](http://wiki.ros.org/message_filters#Time_Synchronizer)
8. [Synchronizing LiDAR and Camera Data for Offline Processing Using ROS](https://medium.com/@kidargueta/synchronizing-lidar-and-camera-data-for-offline-processing-using-ros-de000a3e29cc)

# \Developer documentation

## \ Brief description

### Datatypes and constants

|  |  |
| --- | --- |
| **Enum/Constant name** | **Description** |
| - |  |

### Classes

ImageProcess

Variables: All of the variables are private.

ros::NodeHandle **nodeHandle\_** : The main object of the node. This should be declared at the place of use of the class.

ros::Publisher **pubImg**: The publisher of the topic which can concatenate two images.

ros::Publisher **pubCan**: The publisher of the topic which publishes CAN Frame messages.

ros::Timer **canTimer**: This timer provides the fix rate with the CAN message is published.

unsigned int **imgFreq**: This variable stores the frequency the publishing of the images.

ros::Time **prevTime**, ros::Time **currTime**: These help at the frequency calculating.

message\_filters::Subscriber<sensor\_msgs::CompressedImage> **leftImgSub**: Subscriber object which can synchronize topics. This is for the left image.

message\_filters::Subscriber<sensor\_msgs::CompressedImage> **rightImgSub**: Subscriber object which can synchronize topics. This is for the right image.

typedef message\_filters::sync\_policies::ApproximateTime<sensor\_msgs::CompressedImage, sensor\_msgs::CompressedImage> **MySyncPolicy**: This variable defines the policy of the synchronization the two images.

typedef message\_filters::Synchronizer<MySyncPolicy> **sync**: This is the synchronizer which can synchronize the two necessary images with the defined policy.

boost::shared\_ptr<sync> **syncPtr**: This pointer points the previously created synchronizer object.

Functions:

* **ImageProcess**(ros::NodeHandle& nodeHandle) : nodeHandle\_(nodeHandle): This is the constructor.
* ~**ImageProcess**(): Destructor
* cv\_bridge::CvImageConstPtr **convertImage**(sensor\_msgs::CompressedImage& img, std::string imageEncoding) : Converts the image pointer in sensor\_msgs::CompressedImage& into cv\_bridge::CvImageConstPtr type and returns it. It is important because of the concatenate function.
* cv\_bridge::CvImage **hConcatImg**(const cv\_bridge::CvImageConstPtr img1, const cv\_bridge::CvImageConstPtr img2) : Concatenates the two given images horizontally into one and returns it.
* void **syncCallback**(const sensor\_msgs::CompressedImage& leftImgSub, const sensor\_msgs::CompressedImage& rightImgSub): This is the callback function which can concatenate the synchronized images and calculates the publishing frequency.
* void **canPub**(const ros::TimerEvent& event): This is the timer controlled callback which publishes the CAN messages with 50Hz.

## \ Detailed description

* **ImageProcess**(ros::NodeHandle& nodeHandle) : nodeHandle\_(nodeHandle): The constructor first gives default values of the variables (imgFreq, currTime, prevTime). After that, defines the subscriptions and add the synchronized callback which concatenates the images. This function also makes a timer with the CAN messages are sent with fixed rate (50Hz). Finally adds the published topics to the NodeHandle object.
* **~ImageProcess:** The destructor stops the timer, so that the CAN publisher does not send more messages.
* cv\_bridge::CvImageConstPtr **convertImage**(sensor\_msgs::CompressedImage& img, std::string imageEncoding) : The function converts the given image with the given encoding type into the cv\_bridge::CvImageConstPtr type. This is the case for the other function, which can concatenate two images horizontally.
* cv\_bridge::CvImage **hConcatImg**(const cv\_bridge::CvImageConstPtr img1, const cv\_bridge::CvImageConstPtr img2) : The function expects two images provided by the previous function. The used cv::hconcat(cv::Mat M1, cv::Mat M2, cv::Mat HM) function expects cv::Mat types, and the explained cv\_bridge::CvImageConstPtr’s *image* attribute contains this type. The cv::hconcat(…) function saves the new image into the HM variable. After that the function creates a cv\_bridge::CvImage from this image, and returns. This image will be published into the /dummy\_img\_mate topic.
* void **syncCallback**(const sensor\_msgs::CopmressedImageConstPtr& leftImgSub, const sensor\_msgs::CompressedImageConstPtr rightImgSub) : The function is a synchronized callback function. Saves the synchronized images when they are published. It converts the images into the right type and then concatenate them with the previously presented functions. The image will be published in the /dummy\_img\_mate topic. This function also calculates the frequency of the publishing from the elapsed time between two publishing. This value will be stored in the imgFreq argument.
* void **canPub**(const ros::TimerEvent& event): This is the callback which is called by the timer periodically. When it happens it creates a can\_msgs::Frame canFrame variable and fills its fields with datas defined in the task description. The completed message will be published into the /sent\_messages topic.
* int **main**(int argc, char \*\*argv): A ros::NodeHandle object is defined and an ImageProcess object is instantiated with the NodeHandle. The ros::spin() function is also called, with which ensures the callbacks run.

# \ Changelog

## Changes

-

|  |  |  |
| --- | --- | --- |
| Change Description | Affected Function | Reason |
| - |  |  |