

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 messages with different frequency.

## \ 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 download and 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. |
| /zed\_left\_img\_comp/compressed | sensor\_msgs/CompressedImage | The left ZED camera image for the dummy project. |

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

This part will be the bigger pain in the ass to write. It consists of two parts. A brief explanation on what each function does, and a simple overview on how the package works. This is intended to be used if someone has to work with the code but does not necessarily has to understand the code deeper. For example, if someone has to write a visualization function for the path planner node, they have to understand where they should call the function, where certain values are stored etc.

The second part is a more detailed description of each function. This should be detailed enough, if for example a bug is found someone is able to work with the code without having to figure things out for themselves.

## \ Brief description

### Datatypes and constants

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

### Classes

This simply node does not contain any classes.

Functions:

* cv\_bridge::CvImageConstPtr **convertImage**(sensor\_msgs::CompressedImage& img, std::string imageEncoding) : Converts the image 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.

## \ Detailed description

* 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 by the node.
* 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 also sets flags that shows the availability of the data.
* void **canThread**() : This Node uses two threads in the processor, that’s because the CAN communication expects accurate publishing frequency (50 Hz). This function defines the functionality of this thread. It creates a publisher, which publishes into the “/sent\_messages” topic with the given frequency. For this it creates a can\_msgs::Frame message object, and fills with data that was defined in the task description. It publishes the message as long as the Node is running.
* In the **main** function the synchronized subscriptions are defined and the second thread is started. While the Node is running (ros::ok() gives true) the synchronized images are concatenated and published into the /dummy\_img\_mate topic. The previously mentioned flags are tested here: if they are true (2 new synchronized images are ready for the concatenating) the function runs in the while cycle and after that they are set false. This prevents it from repeating the operation multiple times.

# \ Changelog

## Changes

-

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