panopticon Documentation

Release 0.1.0

Jan-Christoph Klie

April 18, 2016

Contents

This ROS package is used to localize robots by putting markers on their top. The infrastructure in which it will be used has four ceiling cameras with overlapping field of views. Panopticon detects the robots and gives their position in a global world frame. The source code can be obtained on Github .

Contents:

CHAPTER 1

Architecture

1.1 Frames

This section describes the differnt coordinate systems used. There are two special marker, `map` and `center`. `map` is the origin of the global frame (0,0), `center` has the property to be seen by all cameras. Every camera has its own frame, as do the `center` and `map`.

The coordinate systems (frames) used can be seen in this illustration. The four cameras are depicted by C0 to C3 and are connected via the center marker. C0 can see the marker which represents the origin. When a marker is now seen in C1, C2 or C3, the position in the `map` frame can be computed by going `C0->center->map` . The arrow have to be read like `a->b` means `a is parent of b`.

For dealing with the different coordinate systems, tf2 is used, the de facto standard transform library for ROS. In the current setup, `map` and `center` have to be seen at all time. If they are moved, then the coordinate system changes because the relation between them changes. But as long as they adhere to their position requirements, the package runs. As the transformations obtained by ArUco are from camera to marker, some have to be inversed to get the above graph.

1.2 Cameras

We use four ceiling mounted Logitech USB cameras. They are interfaced with the usb_cam package and launched by a custom launch file **launch/usb_cam_by_id.xml** . They have to have overlapping field of view, at least one marker, the `center`, has to be seen by all four cameras. For accurate results, the cameras have to be calibrated.

Right now, the cameras are not mounted to a fixed device address, i.e. camera IDs can change when restarting the computer. Therefore, make sure that *usb_cam0* is the one which can see the origin marker.

1.3 Marker

The marker used are from ArUco and detected by vanilla ar_sys. We use a slightly changed multi_board version launch file (see *launch/aruco_multi.xml*). The list of boards which can be detected is in **config/boards.yml**. The origin and center marker **HAVE** to be called `map` and `center` respectively. Other marker have to be prefixed *marker*. Each board consists of two configuration files: one in pixel and one in meter. We only need the later. These can be found in *config/boards* and can be generated. To add a new board, either do the way of ArUco and use the command line tools directly or use the scripts in this package. More on that in Scripts.

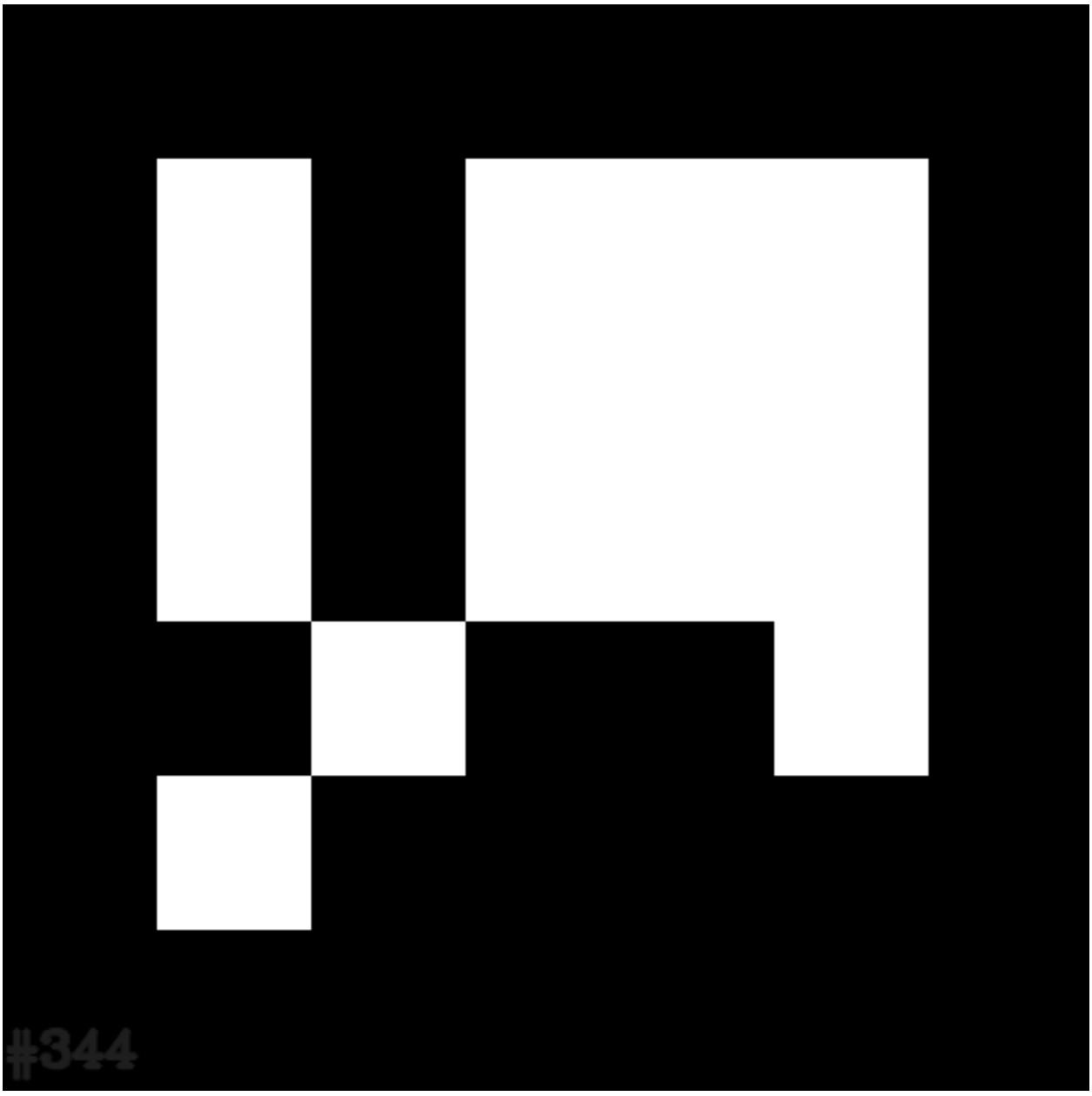

Fig. 1.1: A typical ArUco marker. Each marker encodes a number.

A good overview on ArUco marker can be found in this OpenCV ArUco tutorial or directly on the authors homepage.

1.4 Nodes

The following section describes which nodes the *panopticon* package uses to get the marker positions.

1.4.1 usb_cam

We have one `usb_cam` node for every camera used. These publish the raw image on `/usb_cam$(camera_id)/transform`. They are started with the `usb_cam_by_id` launchfile.

1.4.2 ar_sys

We have one `ar_sys` multi-board node for every camera. These publish the transformation from camera to marker for each detected marker on `cam$(camera_id)/transform`. These are started with the `aruco_multi` launchfile.

1.4.3 panopticon_transformer

This node takes the transformations from the four `ar_sys` nodes and creates the transformation graph depicted in *Frames* . That is done by filtering markers and inversing some of the transformations. Marker filtering is necessary, as tf only allows one parent, but a marker can be seen potentially in four cameras, so could have four parents.

1.4.4 panopticon_poser

This node takes the transformations for vanilla markers (not center or map) from the four `ar_sys` nodes, computes the pose in the world frame and publishes them. As every marker can potentially be seen by four cameras, each marker used has four accompanying topics.

1.5 Subscribed topics

The following section lists the topics subscribed and published. These are only the topics of this very package. Topics from `ar_sys` and `usb_cam` are omitted, see their respective documentations.

`camera0/transform` (geometry_msgs/TransformStamped) The transform from the first camera to each detected marker.

`camera1/transform` (geometry_msgs/TransformStamped) The transform from the second camera to each detected marker.

`camera2/transform` (geometry_msgs/TransformStamped) The transform from the third camera to each detected marker.

`camera3/transform` (geometry_msgs/TransformStamped) The transform from the fourth camera to each detected marker.

1.6 Published topics

For every marker listed in the board configuration file that is also detected, panopticon publishes the raw position in world coordinate for every camera. As every marker can potentially be seen by four cameras, each marker used has four accompanying topics.

`pose/marker$(markerId)/cam0` (geometry_msgs/PoseWithCovarianceStamped) The pose of marker *$(markerId)* detected by cam0 in world coordinates.

`pose/marker$(markerId)/cam1` (geometry_msgs/PoseWithCovarianceStamped) The pose of marker *$(markerId)* detected by cam1 in world coordinates.

`pose/marker$(markerId)/cam2` (geometry_msgs/PoseWithCovarianceStamped) The pose of marker *$(markerId)* detected by cam2 in world coordinates.

`pose/marker$(markerId)/cam3` (geometry_msgs/PoseWithCovarianceStamped) The pose of marker *$(markerId)* detected by cam3 in world coordinates.

CHAPTER 2

Installing

The following chapter describes the steps necessary to install this package.

2.1 Installing dependencies

This package requires two external ROS packages, *ar_sys* and *usb_cam*. These have binary distributions in Ubuntu and can be installed via:

```
sudo apt install ros-indigo-ar-sys
sudo apt install ros-indigo-usb-cam
```

If they should be installed only locally, then one needs to download them manually in the used catkin workspace. This requires `git` installed and an already existing and sourced catkin workspace:

```
cd <your_catkin_workspace>
mkdir -p src
cd src
git clone https://github.com/Sahloul/ar_sys.git
git clone https://github.com/bosch-ros-pkg/usb_cam.git
cd ..
catkin_make
```

2.2 Installing the package itself

As the source code for this package is also hosted on Github, installing simply means checking its repository out:

> cd <your_catkin_workspace> mkdir -p src cd src git clone https://github.com/Rentier/ros-panopticon cd .. catkin_make

CHAPTER 3

Usage

The goal for this package is to publish the position of previously defined markers in a global world frame. These markers can be in view of only some of the cameras. The output is therefore the position of each detected marker in world coordinates for every camera attached.

3.1 Start the package

In order to launch the `usb_cam` and `ar_sys` nodes together with the `panopticon` node, a handy launch file is provided. Just run the following in the root of the package:

```
roscd panopticon
roslaunch launch/panopticon.xml
```

Make sure that the `map` marker can be seen by `usb_cam0`. Make sure that the `center` marker can be seen by all four cameras. The pose of all other markers defined and detected then are published under `pose/marker$(markerId)/cam$[1-4]`.

3.2 Configuration

`usb_cam` and `ar_sys` can be heavily configured. See their respective docmentations and change the launch files under **launch** as your require.

3.3 Adding markers

There are two steps needed to add markers: Generate a `board_configuration` and add an entry to `config\boards.yml` . Make sure that you specify the right ID and right size. A script has been provided to generate the configuration, see Scripts for more. Markers used for localization have to be prefixed with `marker` !

CHAPTER 4

Scripts

4.1 Marker generation

To add additional markers, generate them with:

```
./create_board.sh <MARKER_ID>
```

from the **scripts** folder. Put the boardconfig into the **config/boards** folder and add a line to **config/boards.yml**. One can adjust which marker is `map`, `middle` and plain marker in `config/boards.yml`. Change the script for different marker sizes. In order for this script to work, the command line tools of ArUco have to be in path.

CHAPTER 5

ToDo

5.1 Camera mount point

At the moment, the cameras are not mounted to a fixed device adress. In order to make sure that the origin is always in the same camera, fix the mount by udev rules. A guide for that can be found in libuvc_camera . The device ID of the cameras can be found by the following neat snippet:

```
sudo lsusb -v -d 046d:082c | grep -i serial
```

Easiest way to get it done is removing all cameras but one, check which area it sees, and then assign a fixed device ID. Repeat until all for have been assigned.

5.2 Data fusion

Right now, for every marker of interest, the system returns at most four estimated positions (depends on how many cameras see the marker). To make it more accurate and useful, this information has to be merged into one. There are many different approaches, from ad-hoc trial-and-error to dedicated ROS packages, like robot_localization . A launch file for that has already been started, some work is missing there. Another idea is to write a custom node that weights a (moving) average with the time of the positions and the distance to the corners.

5.3 Covariance

It was planned to use `robot_localization` as the data fusion package of choice. As it requires information about the covariance, `panopticon` publishes `geometry_msgs/PoseWithCovarianceStamped` messages. Right now, the covariance is fixed. In order to make it more reasonable, it has to be computed somehow instead of fixing it.

5.4 Calibration

The cameras have to be calibrated in order to get a good position estimation. In the current setup, only one camera is calibrated, all other use the same data. To make it more accurate, each camera has to be calibrated on its own. This calibration can best be stored in a *calibration* folder in the package and given to the *aruco_multi* launch file.

5.5 Visualization

In order to get a grasp about the marker positions in the world grid, a good visualization is handy. The camera, map and center frames itself can be already seen in rviz (see `page` for an example). The marker are not yet there and have to be implemented. That can be done by sending messages to *rviz*, as described for instance in this rviz tutorial.

5.6 Make map and center marker removable

In the current setup, the *map* and *center* marker have to be always in position. But in general, they have to be only detected once to get the relation between the cameras. This can be saved and published to keep the frame in tf2 active, whereas the marker can then be removed until the next restart of the package. This also prevents errors like occlusion of the center marker by a robot.

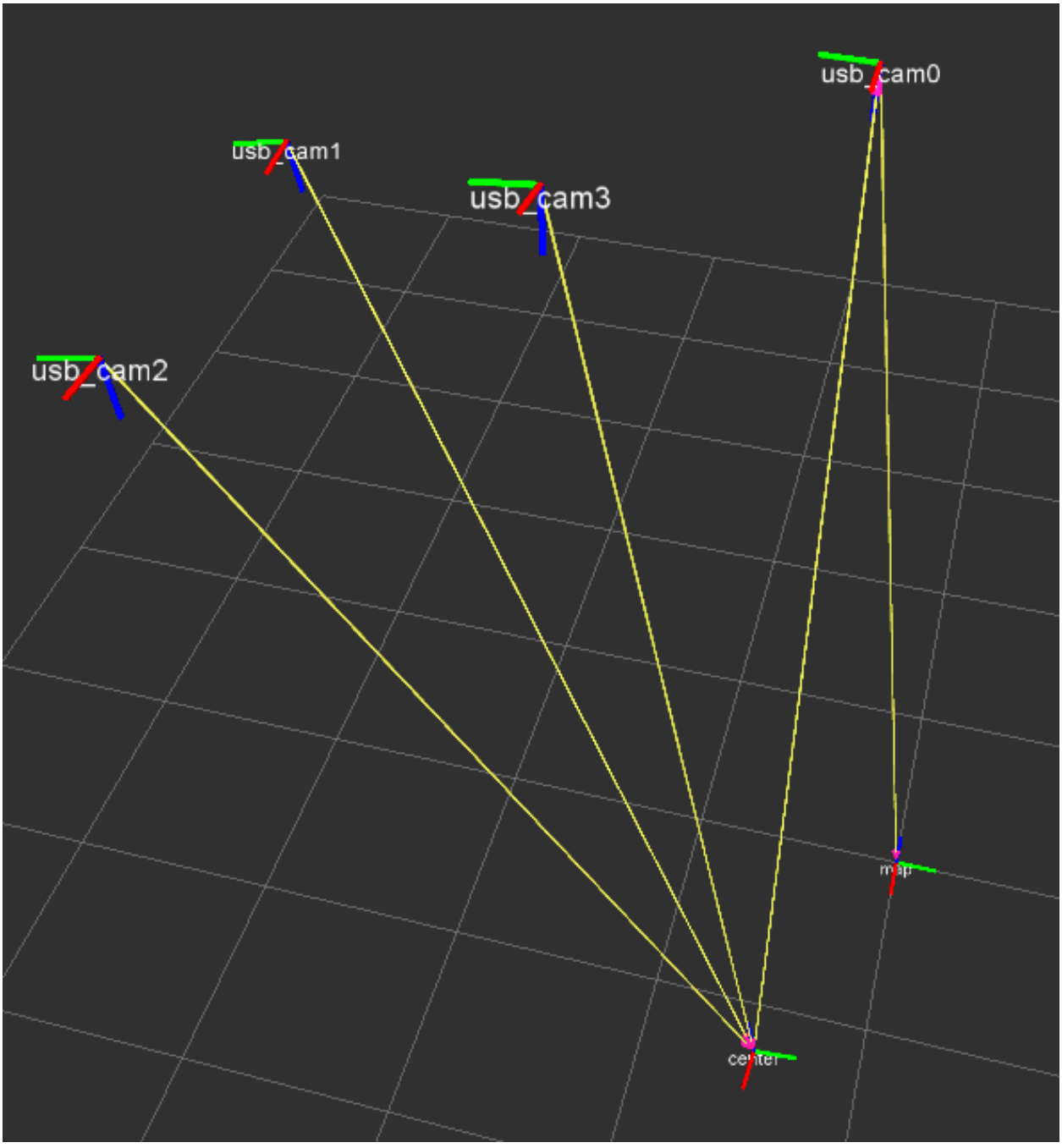

Fig. 5.1: Overview of the camera and frame setup. The four ceiling cameras can see a center marker. One camera can see the marker that serves as the origin. The position of any marker seen by atleast one camera can be transformed into the world frame.