Camera Calibration Package

Camera Calibration Package

Description:

This page discusses the *camera_calibration_package* in detail. The package uses a set of calibration images (checkerboard images) and OpenCV libraries to calibrate one or more cameras through a singular ROS2 node.

Note that image sets for each calibrating camera must be provided prior to calibration! This code is can also be used on any monocular camera, not just LUCID Vision cameras - as such, it does NOT require the Arena SDK.

The OpenCV Calibration Tutorial was used as a reference, however the tutorial includes lots of unnecessary components so the code in this package is a much simpler version.

Requirements:

- Ubuntu Focal Fossa
- ROS2 Foxy Fitzroy
- C++17 or higher

Configuration and Launch Files

Before using this package, make sure ALL paths in the configuration and launch files are set correctly. This will especially cause issues when cloning the repository to a new machine as the paths cloned from the remote repo are (most likely) not valid for the new local repo.

Note: All paths are set globally. Paths are also rarely set outside of configuration/launch files. This allows the user to point to different files in a more automated manner. Editing the configuration file DOES NOT require you to rebuild the package, but editing the launch file DOES!

***Configuration File*:**
`camera_calibration_config.yaml` is shown below:

src > camera_calibration_package > config > ! camera_calibration_config.yaml

```yaml
#TRI028S-CC
camera1:
  camera_calibration:
    ros__parameters:
      camera_name: "TRI028S-CC"
      image_path: "/home/tahnt/T3_Repos/camera_packages/ros2_ws/src/camera_calibration_package/calibration_images/TRI028S_CC"
      extension: "/*.jpg"
      pattern_size: [8.0, 8.0]
      frame_size: [1936.0, 1464.0]
      camera_output_file: "/home/tahnt/T3_Repos/camera_packages/ros2_ws/src/camera_calibration_package/camera_params/TRI028S-CC_params.yaml"

#CAMERA2
camera2:
  camera_calibration:
    ros__parameters:
      camera_name: "<camera_name>"
      image_path: "<image_path>"
      extension: "/*.<extension>"
      pattern_size: [<checker_dim, checker_dim>]
      frame_size: [<image_width>, <image_height>]
      camera_output_file: "<camera_output_file>"

#CAMERA3
camera3:
  camera_calibration:
    ros__parameters:
      camera_name: "<camera_name>"
      image_path: "<image_path>"
      extension: "/*.<extension>"
      pattern_size: [<checker_dim, checker_dim>]
      frame_size: [<image_width>, <image_height>]
      camera_output_file: "<camera_output_file>"
```

Multiple namespaces are included to represent different calibrating cameras. This repo only has one set of calibration images associated with the TRI028S_CC camera. Notes to add calibration images for different cameras are discussed later.

ROS2 Parameters Configured:

- **camera_name:** Name of the camera
- **image_path:** Path to the calibration image set
- **extension:** Extension/format of calibration image (Note all images per set must share one format, but image formats CAN vary among sets!)
- **pattern_size:** Checkerboard pattern (square x square) - discussed in detail later
- **frame_size:** Calibrating image ROI (width X height)
- **camera_output_file:** Path to camera parameter output file

***Launch File*:**

`camera_calibration_launch.py` is shown below:

src > camera_calibration_package > launch > camera_calibration_launch.py > ...

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():

    config_path = '/home/tahnt/T3_Repos/camera_packages/ros2_ws/src/camera_calibration_package/config/camera_calibration_config.yaml'

    return LaunchDescription([
        Node(
            package="camera_calibration_package",
            executable="calibrate",
            namespace="camera1",
            name="camera_calibration",
            output="screen",
            parameters=[config_path]
        ),
        #Node(                                      #New launch nodes must be included to calibrate multiple cameras!
            #package="camera_calibration_package",
            #executable="calibrate",
            #namespace="camera2",
            #name="camera_calibration",
            #output="screen",
            #parameters=[config_path]
        #),
        #Node(                                      #New launch nodes must be included to calibrate multiple cameras!
            #package="camera_calibration_package",
            #executable="calibrate",
            #namespace="camera3",
            #name="camera_calibration",
            #output="screen",
            #parameters=[config_path]
        #)
    ])
```

Typically, the launch file does not need to be edited often. Make sure **config_path** correctly points to `camera_calibration_config.yaml` . If more cameras are added, additional launch nodes can be added - be sure to indicate each **namespace** correctly!

Generating Calibration Images

Using the Camera Capture Package

New calibration images can be generated using the *camera_capture_package*! The package was actually designed to take calibration images.

You can directly link the save path in `camera_capture_config.yaml` to point into the `/calibration_images` directory in the *camera_calibration_package*. For more details on setting paths, check the Camera Capture Package page!

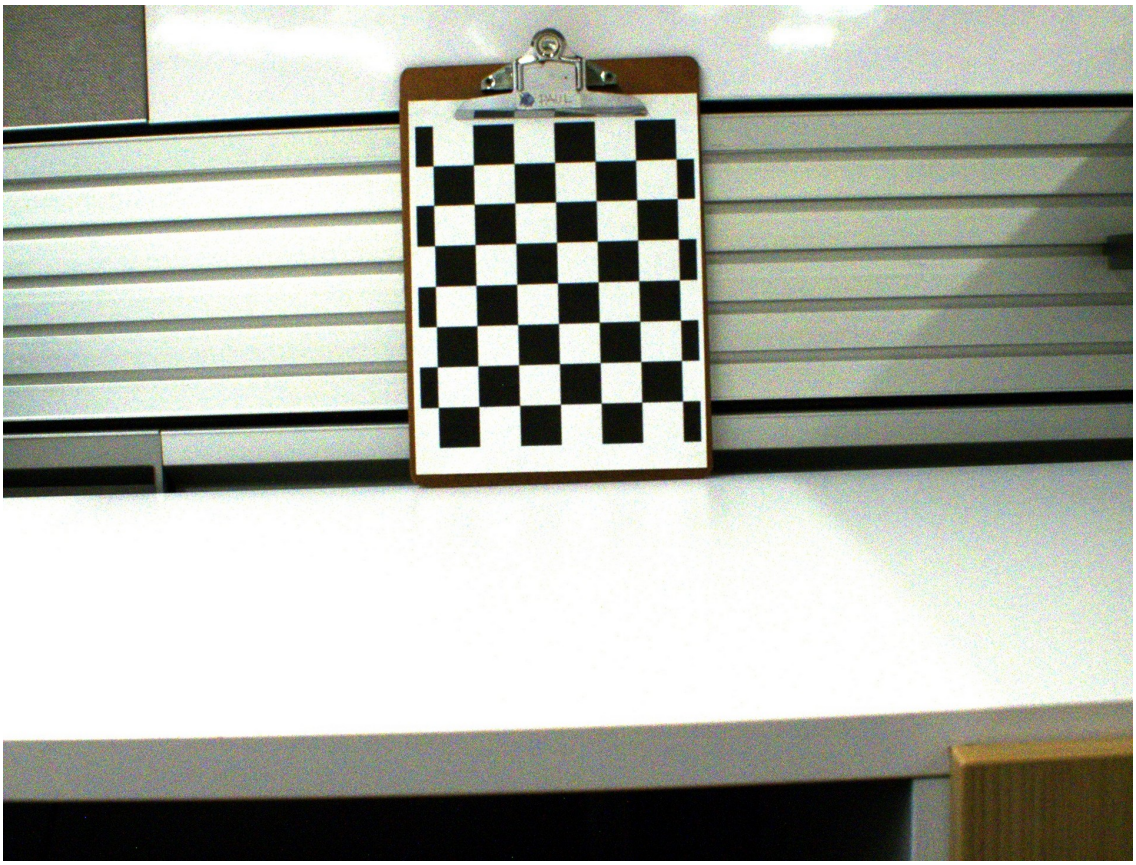
Taking Effective Calibration Images

There are a few things to keep in mind when generating calibration images to ensure the best possible camera intrinsics estimates.

- Take multiple calibration images - the more images the better the estimate! You should have 10 minimum per image set.
- Keep the camera stationary and move the checkerboard - including different orientations improve the estimate. The checkboard can be translated AND rotated!
- Make sure the checkerboard pattern is completely flat and visible before taking the image.

Specifying the Correct Pattern Size

Consider a calibration image below:



The specified pattern size should be the ***number of outer squares*** so in this case, (8x8) as there are 8 black and white outer squares (even the half squares are counted!).

Note: The calibration algorithm actually takes the ***number of inner squares*** or (pattern_size - 1, pattern_size - 1), but that is accounted for in code!

Using the Package

After an image set(s) is/are populated and all paths are correctly set, follow the steps below to calibrate your camera(s) (also included in the `README.md`):

Before Use:

- **Make sure ALL PATHS ARE SET CORRECTLY in the launch and config files before use!**
- **These steps assume you have already created a workspace folder and a `/src` directory within it!**

Steps:

1. Navigate into the `\src` directory of your workspace and clone the repo using `git clone`
2. Navigate back into the workspace directory and source `$ source /opt/ros/foxy/setup.bash`
3. Build package `$ colcon build` or `$ colcon build --packages-select <package_name>`
4. Open a new terminal and source it `$ . install/setup.bash`
5. Run launch file `$ ros2 launch <package_name> <launch_file_name>` in this case it is `$ ros2 launch camera_calibration_package camera_calibration_launch.py`

If executed correctly, the stream window should open and the terminal should output the following (or similar):

```
tahnt@pelican-glide:~/T3_Repos/camera_packages/ros2_ws$ ros2 launch camera_calibration_package camera_calibration_la
unch.py
[INFO] [launch]: All log files can be found below /home/tahnt/.ros/log/2023-12-06-10-57-21-084069-pelican-glide-2102
2
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [calibrate-1]: process started with pid [21025]
[calibrate-1] [INFO] [1701889045.629572786] [camera1.camera_calibration]: TRI028S-CC Image List: /home/tahnt/T3_Repo
s/camera_packages/ros2_ws/src/camera_calibration_package/calibration_images/TRI028S_CC/*.jpg
[calibrate-1] [INFO] [1701889045.629797605] [camera1.camera_calibration]: TRI028S-CC Frame Size: 1936 1464
[calibrate-1] [INFO] [1701889045.629810739] [camera1.camera_calibration]: Calibrating TRI028S-CC ...
[calibrate-1] [INFO] [1701889046.050457308] [camera1.camera_calibration]: Calibration successful
[calibrate-1] [INFO] [1701889046.058819958] [camera1.camera_calibration]: Write successful
[calibrate-1] [INFO] [1701889046.058895137] [camera1.camera_calibration]: TRI028S-CC parameters written to /home/tah
nt/T3_Repos/camera_packages/ros2_ws/src/camera_calibration_package/camera_params/TRI028S-CC_params.yaml
[INFO] [calibrate-1]: process has finished cleanly [pid 21025]
```

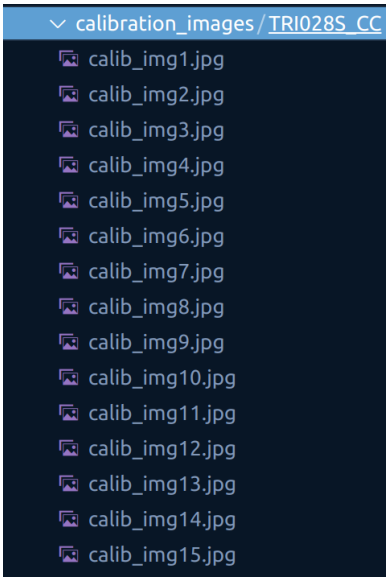
Failure Case

If one or more calibration images in a set are invalid, the terminal will output the following (or similar):

```
tahnt@pelican-glide:~/T3_Repos/camera_packages/ros2_ws$ ros2 launch camera_calibration_package camera_calibration_launch.py
[INFO] [launch]: All log files can be found below /home/tahnt/.ros/log/2023-12-06-11-09-13-472418-pelican-glide-24660
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [calibrate-1]: process started with pid [24662]
[calibrate-1] [ERROR] [1701889759.628787329] [camera1.camera_calibration]:
[calibrate-1] Failed to detect pattern for /home/tahnt/T3_Repos/camera_packages/ros2_ws/src/camera_calibration_package/calibration
_images/TRI028S_CC/calib_img16.jpg during calibration for TRI028S-CC
[calibrate-1]
[calibrate-1] [INFO] [1701889760.198415817] [camera1.camera_calibration]: TRI028S-CC Image List: /home/tahnt/T3_Repos/camera_packa
ges/ros2_ws/src/camera_calibration_package/calibration_images/TRI028S_CC/*.jpg
[calibrate-1] [INFO] [1701889760.198433646] [camera1.camera_calibration]: TRI028S-CC Frame Size: 1936 1464
[calibrate-1] [INFO] [1701889760.198438329] [camera1.camera_calibration]: Calibrating TRI028S-CC ...
[calibrate-1] terminate called after throwing an instance of 'cv::Exception'
[calibrate-1]   what():  OpenCV(4.2.0) ../modules/calib3d/src/calibration.cpp:3332: error: (-2:Unspecified error) in function 'voi
d cv::collectCalibrationData(cv::InputArrayOfArrays, cv::InputArrayOfArrays, cv::InputArrayOfArrays, int, cv::Mat&, cv::Mat&, cv::
Mat*, cv::Mat&)'
[calibrate-1] >  (expected: 'nimages == (int)imagePoints1.total()'), where
[calibrate-1] >     'nimages' is 15
[calibrate-1] > must be equal to
[calibrate-1] >     '(int)imagePoints1.total()' is 16
[calibrate-1]
[ERROR] [calibrate-1]: process has died [pid 24662, exit code -6, cmd '/home/tahnt/T3_Repos/camera_packages/ros2_ws/install/camera
_calibration_package/lib/camera_calibration_package/calibrate --ros-args -r __node:=camera_calibration -r __ns:=/camera1 --params-
file /home/tahnt/T3_Repos/camera_packages/ros2_ws/src/camera_calibration_package/config/camera_calibration_config.yaml'].
tahnt@pelican-glide:~/T3_Repos/camera_packages/ros2_ws$
```

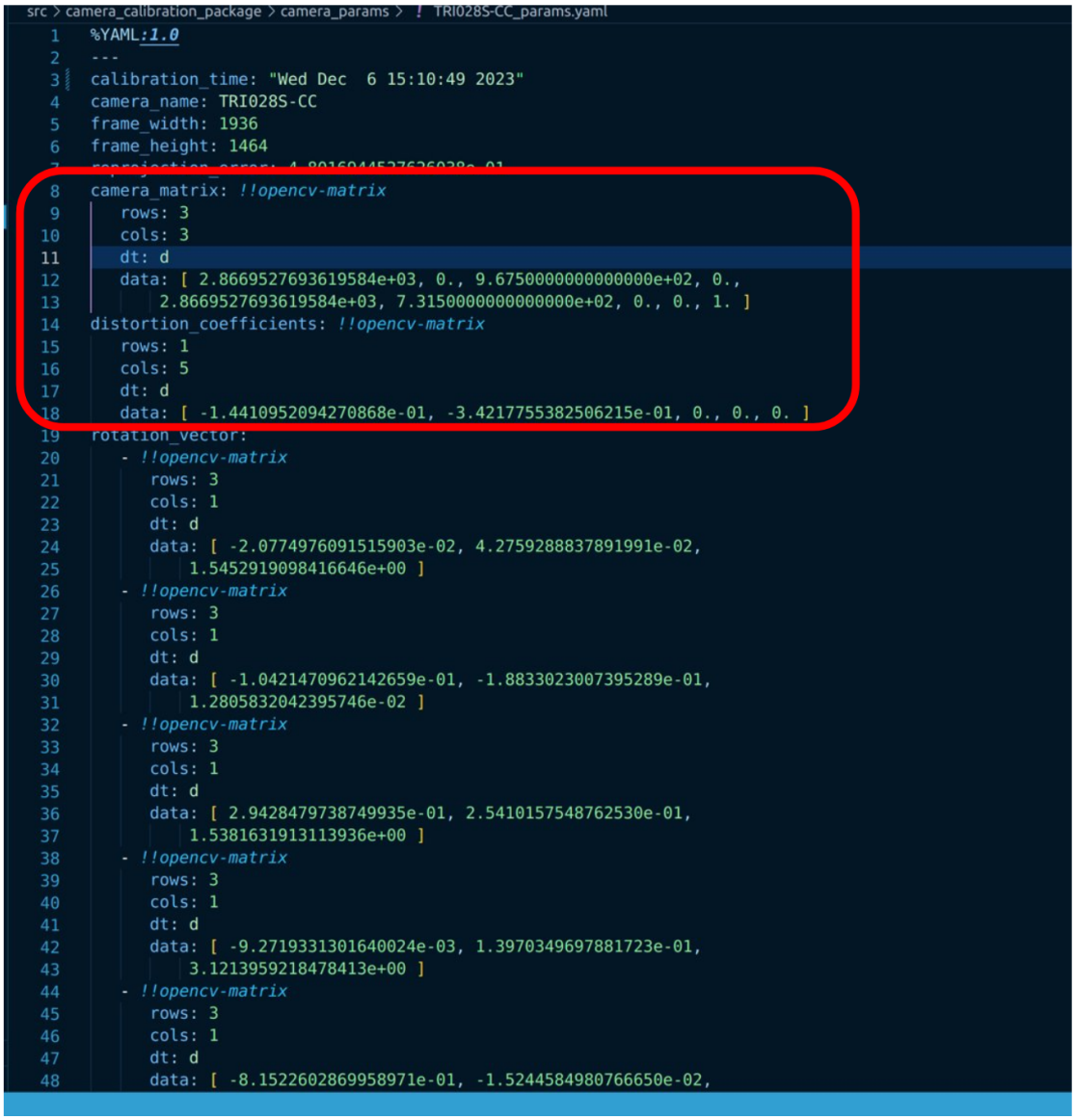
The portion circled in red indicates which image for which set (path and camera specified in the error message) failed the calibration process.

Instead of deleting an entire calibration set you can just delete the failed image HOWEVER you must adjust the names of your image list so they are in sequential order (as shown below)!



Parameter Output

Once the calibration has run successfully, the output file should look like the following (or similar):



The output file will include the camera matrix, distortion coefficients, rotation vectors, and translation vectors. The camera intrinsics which include the camera matrix and distortion coefficients are circled in red in the above image. These are the parameters used to undistort the image, and used in most other post processing applications.

I have not explored the rotation and translation vectors and they are NOT required for any of the post processing that I've uploaded to T3.

Additional Validation

Checking an undistorted image can be a good way to check if the estimated camera intrinsics are reliable. To do so, uncomment the following display piece in `camera_calibration.cpp` , build and run the node again:

```
176     //Show lens corrected images
177     for (auto const &f : file_names_)
178     {
179         //std::cout << std::string(f) << std::endl;
180 
181         cv::Mat img = cv::imread(f, cv::IMREAD_COLOR);
182         cv::Mat imgUndistorted;
183 
184         //Remap the image using precomputed interpolation maps
185         cv::remap(img, imgUndistorted, mapX, mapY, cv::INTER_LINEAR);
186 
187         //Display undistorted images
188         /*
189         cv::imshow("undistorted image", imgUndistorted);
190         int key = cv::waitKey(0);
191         if (key == 's')
192         {
193             cv::imwrite("<enter_write_path>", imgUndistorted);
194         }
```