Documentation

Release 0.1

Yunhyeok Han

Nov 25, 2024

CONTENTS:

CHAPTER

ONE

REAL-TIME CAMERA POSE ESTIMATOR

This program provides a solution for estimating the 6-DoF (Degrees of Freedom) camera pose, which includes position and orientation relative to a global coordinate system defined by a chessboard pattern. It operates in **real-time**.

—

1.1 Prerequisites

Hardware Requirements

- **Camera:** Basler camera.
- **Chessboard Pattern:** A printed or projected chessboard pattern with known dimensions.

Software Requirements

- **Pylon SDK:** Required for connecting the camera to the computer.
- Download it here: Pylon Viewer.

—

1.2 Installation

1. Clone the repository by using git clone with the repository link. .. code-block:: bash

   git clone https://github.com/YunhyeokHan/PoseEstimator.git

2. (Optional) Create a Python 3.9 environment by using conda create command with the name *PoseEstimator* and Python version *3.9*. .. code-block:: bash

   conda create –name PoseEstimator python=3.9

3. Install the required Python libraries by using pip install with the *requirements.txt* file. .. code-block:: bash

   pip install -r requirements.txt

—

1.3 Usage

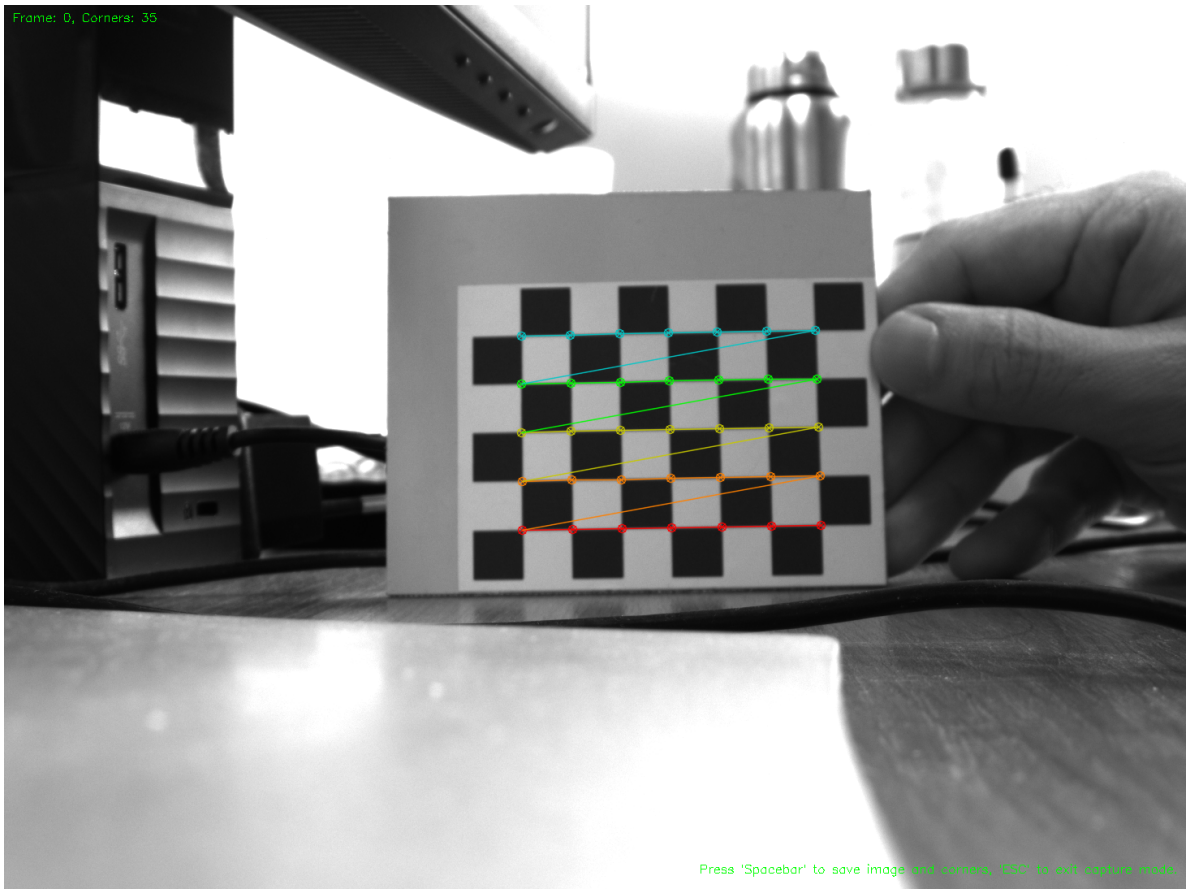
Camera Calibration

Before running the pose estimation, the camera must be calibrated using a chessboard pattern.

1. Start the calibration process by running the camera calibrator script and providing arguments for the checkerboard dimensions, square size, and exposure time.
    - Define the number of corner points in the rows and columns of the pattern (`--checkerboard <rows> <columns>`).
    - Square size: Specify the size of each square on the chessboard in millimeters (`--square_size <size>`).
    - Exposure time: Set the camera exposure time in microseconds (`--exposure_time <value>`).

```
python PoseEstimator/camera_calibrator.py --checkerboard 7 5 --square_size 10 --
↪exposure_time 20000
```

2. Place the chessboard in the camera's view and verify that all corner points are detected. The program will display real-time feedback.



3. Press Spacebar to save detected points for calibration. Move the chessboard to different positions and orientations, capturing sufficient data points.
4. Press ESC to end the capture session. The program will perform intrinsic camera calibration and save the results for future use.

—

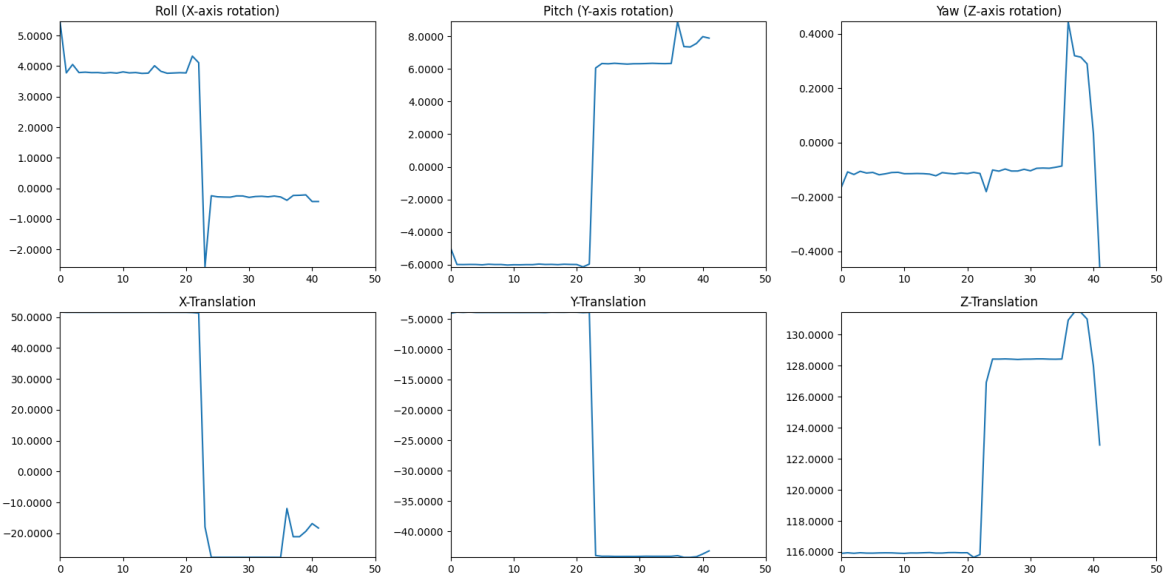
Pose Estimation

Once the camera is calibrated, you can run the pose estimation program.

1. Start the pose estimator by running the pose estimation script with the same arguments as the calibration process.
   - Number of corner points in the rows and columns of the pattern (`--checkerboard <rows> <columns>`).
   - Square size: Specify the size of each square on the chessboard in millimeters (`--square_size <size>`).
   - Exposure time: Set the camera exposure time in microseconds (`--exposure_time <value>`).

```
python PoseEstimator/pose_estimator.py --checkerboard 7 5 --square_size 10 --
↪exposure_time 20000
```

2. The program will display real-time camera feed and calculate the 6-DoF pose relative to the chessboard pattern. Ensure that the pattern is visible to the camera and its corners are detected.



5. Press ESC to terminate the program.

—

1.4 Notes

- Calibration results are saved automatically and reused for pose estimation.
- Ensure the chessboard pattern dimensions and exposure settings are correctly provided in all steps for accurate results.
- For real-time performance, use a compatible camera and a well-lit environment.

CHAPTER

TWO

POSEESTIMATOR

2.1 PoseEstimator package

2.1.1 Submodules

2.1.2 PoseEstimator.camera_calibrator module

PoseEstimator.camera_calibrator.**calibrate_camera**(*objpoints*, *imgpoints*, *image_shape*, *output_file='PoseEstimator/calib.npz'*)

Perform camera calibration and save the results.

Parameters

- **objpoints** (*list*) – List of 3D object points.
- **imgpoints** (*list*) – List of 2D image points.
- **image_shape** (*tuple*) – Shape of the images used for calibration.
- **output_file** (*str*) – File to save calibration results.

Returns
Camera matrix and distortion coefficients.

Return type
tuple

PoseEstimator.camera_calibrator.**capture_chessboard_images**(*camera*, *checkerboard*, *objp*, *save_dir='PoseEstimator/calibration_images'*)

Capture images with detected chessboard corners.

Parameters

- **camera** (*pylon.InstantCamera*) – Initialized camera object.
- **checkerboard** (*tuple*) – Checkerboard size (rows, cols).
- **objp** (*numpy.ndarray*) – 3D object points for the chessboard.
- **save_dir** (*str*) – Directory to save calibration images.

Returns
Object points list, image points list, saved images list.

Return type
tuple

PoseEstimator.camera_calibrator.**initialize_camera**(*fps=100*, *exposure_time=20000*)

Initialize the camera with user-defined settings.

Parameters

- **fps** (*int*) – Frames per second.
- **exposure_time** (*int*) – Exposure time in microseconds.

Returns
Initialized camera object.

Return type
pylon.InstantCamera

PoseEstimator.camera_calibrator.**load_calibration_data**(*calib_file*)

Load camera calibration data from a file.

Parameters
calib_file (*str*) – File containing camera calibration data.

Returns
Camera matrix and distortion coefficients.

Return type
dict

PoseEstimator.camera_calibrator.**prepare_object_points**(*checkerboard*, *square_size*)

Prepare 3D object points for the given checkerboard size.

Parameters

- **checkerboard** (*tuple*) – Checkerboard size (rows, cols).
- **square_size** (*float*) – Size of each square in real-world units.

Returns
Object points for the chessboard.

Return type
numpy.ndarray

PoseEstimator.camera_calibrator.**undistort_image**(*img*, *calib_data*)

Undistort the input image using camera calibration data.

Parameters

- **img** (*numpy.ndarray*) – Input image to undistort.
- **calib_data** (*dict*) – Camera calibration data containing mtx and dist.

Returns
Undistorted image.

Return type
numpy.ndarray

2.1.3 PoseEstimator.pose_estimator module

PoseEstimator.pose_estimator.**generate_objp**(*checkerboard*, *scale*)

Generate the 3D object points for a given checkerboard size and scale.

Parameters

- **checkerboard** (*tuple*) – Checkerboard inner corners as (rows, cols).
- **scale** (*float*) – Scale of the checkerboard squares.

Returns
3D object points.

Return type
numpy.ndarray

PoseEstimator.pose_estimator.**initialize_camera**(*fps=5*, *exposure_time=20000*)

Initialize the camera with user-defined settings.

Parameters

- **fps** (*int*) – Frames per second.
- **exposure_time** (*int*) – Exposure time in microseconds.

Returns
Initialized camera object.

Return type
pylon.InstantCamera

PoseEstimator.pose_estimator.**load_calibration_data**(*filepath*, *img_size*)

Load calibration data from a file and compute the new camera matrix.

Parameters

- **filepath** (*str*) – Path to the calibration data file.
- **img_size** (*tuple*) – Size of the images as (width, height).

Returns
Calibration data containing mtx, dist, mapx, mapy.

Return type
dict

PoseEstimator.pose_estimator.**plotXYZ**(*img*, *rvec*, *tvec*, *newcameramatrix*)

PoseEstimator.pose_estimator.**plot_to_image**(*rx*, *ry*, *rz*, *tx*, *ty*, *tz*, *ind_range*)

Draw and Convert Matplotlib plots to an OpenCV-compatible image.

Parameters

- **rx** (*list*) – Rotation and translation data.
- **ry** (*list*) – Rotation and translation data.
- **rz** (*list*) – Rotation and translation data.
- **tx** (*list*) – Rotation and translation data.
- **ty** (*list*) – Rotation and translation data.
- **tz** (*list*) – Rotation and translation data.

- **ind_range** (*int*) – Number of data points to display.

Returns
Rendered plot as a BGR image.

Return type
numpy.ndarray

PoseEstimator.pose_estimator.**process_camera_feed**(*camera*, *calib_data*, *objp*, *checkerboard*, *ind_range*, *update_interval*)

Process the camera feed and visualize rotation and translation data.

Parameters

- **camera** (*pylon.InstantCamera*) – Initialized camera object.
- **calib_data** (*dict*) – Calibration data containing mtx, dist, mapx, mapy.
- **objp** (*numpy.ndarray*) – 3D object points.
- **checkerboard** (*tuple*) – Checkerboard size (rows, cols).
- **ind_range** (*int*) – Number of frames to display in the plots.
- **update_interval** (*int*) – Interval for updating the plots.

PoseEstimator.pose_estimator.**rotationMatrixToEulerAngles**(*R*)

Convert a rotation matrix to Euler angles (XYZ convention).

Parameters
R (*numpy.ndarray*) – A 3x3 rotation matrix.

Returns
Euler angles (in radians) as [roll, pitch, yaw].

Return type
numpy.ndarray

2.1.4 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX