****

**Embedded Visual Control**

Assignment 2

|  |  |
| --- | --- |
| **Grade: Group (10%)** | **Individual (10%)** |
| **Title:** Vision | |

**Instructions**

Typically, in a vision application, we would have a publisher node, which captures and publishes the images, a processing node, which runs a pipeline of processes through the incoming messages, and lastly a subscriber node, which simply views the output image. While in Workshop 2, we developed the publisher and subscriber nodes, here you will create your own image processing algorithms.

As a group, create a copy of the vision\_ws directory and name it workshop2\_<group\_id> under the workshops folder. Furthermore, individually create another copy and name it workshop2\_<student\_id> under the same directory.

**Group assignment**

While experimenting with Workshop 2, you may have noticed that the images captured by the camera appear distorted, creating a fisheye effect. In vision-based applications, accurate object detection is crucial, as you rely on the camera to represent objects at their true positions. However, distortion can cause objects to appear at incorrect locations and distances, potentially affecting your robot's perception and decision-making.

These distortions originate from the camera’s intrinsic characteristics. Especially pinhole cameras, such as the CSI camera used, introduce radial and tangential distortions. You can undistort the images captured by the camera by running OpenCV’s undistort function. However, this function requires as inputs not only the raw image but also the camera matrix containing the focal lengths and optical centers of the specific camera.

With the provided chessboard and the related OpenCV Camera Calibration [tutorial](https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html), extract the camera’s matrix and then create a ROS application described as:

* A camera publisher node which publishes the camera’s raw image
* A processing node which takes the incoming raw messages and undistorts the images using OpenCV’s function. It then publishes both the raw and undistorted images in a single message. Details on how to make custom messages can be found in [ROS/Tutorials/CreatingMsgAndSrv.](https://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv)
* A subscriber node which will project the incoming custom messages

**Questions:**

1. **Camera Undistortion**: Capture a side-by-side video of the distorted and undistorted images (max length ~ 10s). Instead of capturing the video from the X11 forwarding window (which could introduce large network lag), you can store the video locally in Jetson and then upload it. (**9pts**)
2. **ROS graph**: Illustrate the ROS architecture. (**1pts**)

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **Assessment Criteria:**   * Short and concise answers * Arguments must be supported by comprehensive visuals (and logging information) * Code quality | **Submission Guidelines:**   * Each group must submit their workshop directory (in zipped format) with the video and report inside * Answers are written in the space between questions |

**Individual assignment:**

Notice: The individual assignment has as prerequisite the undistort node developed in the Group assignment.

After creating the processing node above, you can move to an actual vision application. Each student must select a different vision application to implement. Some applications can be but are not limited to:

* Real-time object detection
* Lane detection
* Face detection and recognition
* Hand gesture recognition
* QR Code/AprilTags/Barcode Scanner
* Traffic Sign recognition
* Optical Character recognition (OCR) for Text Extraction
* People Counter
* Motion Detection and Intrusion Alert
* Object Tracking

NVIDIA Jetson Nano has a 128-core Maxwell GPU enabling it to run small CNN models, like YOLO or MobileNet locally. Furthermore, in case the application you develop is limited by the required environmental data, e.g. for lane detection and traffic sign recognition, you can use online videos instead of raw camera feed for processing images. If you use external data, the camera calibration function can be omitted. However, the application must still be implemented using ROS with an architecture similar to the one explained above.

**Questions:**

1. **Vision Application:** Capture a side-by-side video of the original and processed video (max length ~ 1min). (**9pts**)
2. **ROS graph**: Illustrate the ROS architecture. (**1pts**)

|  |
| --- |
| For My individual assignment, I worked on the Person counter task. The full ROS architecture used was: |

|  |  |
| --- | --- |
| **Assessment Criteria:**   * Short and concise answers * Arguments must be supported by comprehensive visuals (and logging information) * Code quality | **Submission Guidelines:**   * Each individual must submit their workshop directory (in zipped format) with the video and report inside * Answers are written in the space between questions |

****