Task 4B : Camera Calibaration

Objective:

The task 4b has two objectives:

1. To calibrate the camera using camera-calibration package of ros2.
2. Implement ArUco Detection (as done in Task 2A) with the calibrated feed and\ Publish the message on the specified pen pose topic for all three robots.

---

1. Calibration :

Raw image from the USB camera has what is known as the fish-eye effect. We need to remove this fish-eye effect to effectively detect Aruco markers in further tasks. Hence it is necessary to calibrate our USB camera. For more details refer **Why To Calibrate?**📏 in learning resources section.

Setup :

1. You will need a checkerboard in order to calibrate your camera. Print out the image given on this 👉 **link**.
2. You must install the camera calibration package in ROS2. Open a terminal and type the following command:

```
sudo apt install ros-humble-camera-calibration-parsers
sudo apt install ros-humble-camera-info-manager
sudo apt install ros-humble-camera-calibration
```

3. Now, you will launch the camera node i.e

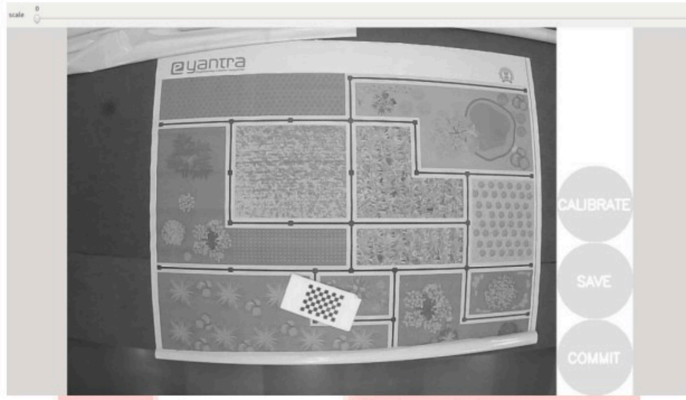
```
ros2 launch usb_cam camera.launch.py
```

You should see `/camera1/image_raw` and `/camera1/camera_info` when you run `ros2 topic list`

4. Open new terminal and run the camera calibrator

```
ros2 run camera_calibration cameracalibrator --size 8x6 --square 0.02 --ros-args
```

5. You should now see a new window as shown below.



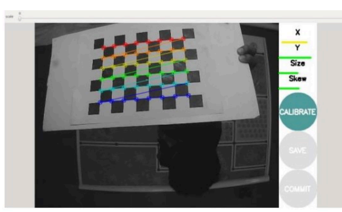
Please Note: Content of the image (arena etc) are for representation purposes only (from a different theme), not relevant to the the HB theme.

---

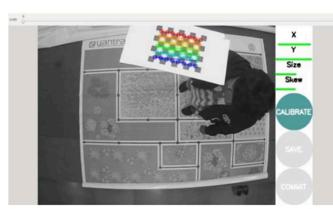
6. Hold up the checkerboard in front of the camera. Zig-zag lines should be displayed on the checkerboard. To calibrate:
    - X axis – Move the checkerboard left to right and right to left.
    - Y axis – Move the checkerboard top to bottom and bottom to top.
    - Size – Move the checkerboard close to and away from the camera.
    - Skew – Tilt the checkerboard in all directions.

    Note: The more samples you take, the better is the output.

    Size Calibration

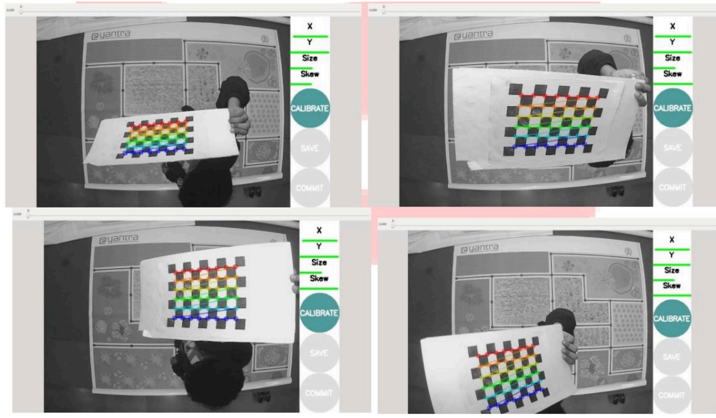
    Y axis Calibration

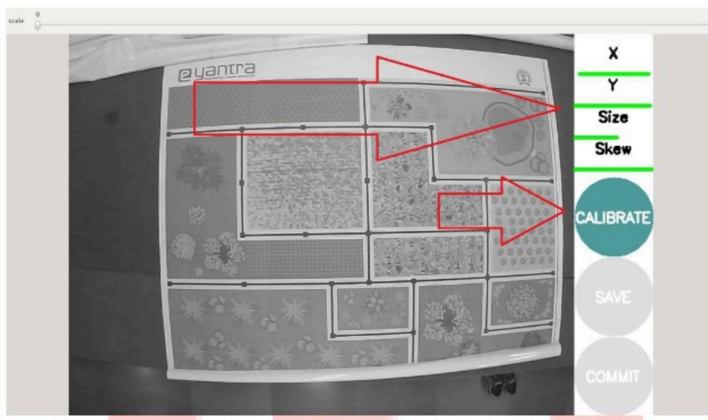
    X axis Calibration

Skew Calibration



7. You must perform all these steps until you get maximum green for X, Y, Size and Skew in both directions on the panel on the right-hand side. When complete, your final progress should look like the figure below.



8. When you get green progress for X, Y, Size and Skew, your 'CALIBRATE' button will be highlighted. Click that button in order to generate the calibration matrix. This might take some time so please wait while it generates the matrix. It might appear your computer has hung, but that is not the case.
9. After the calibration is completed the save and commit buttons light up. And you can also see the result in terminal.
10. Press the save button to see the result. Data is saved to `/tmp/calibrationdata.tar.gz` . After saving the result, terminate all running processes in the terminal.
11. To use the the calibration file unzip the `calibration.tar.gz`
12. In the folder images used for calibration are available and also `ost.yaml` and `ost.txt` files. You can use the yaml file which contains the calibration parameters as directed by the camera driver
13. Copy the content of `ost.yaml` and paste it at `camera_info.yaml` . \ Use VScode to paste the content for convenience. Since you cannot modify the files present at root folder without root access.
14. After pasting the content, you have to download the image_proc package to obtain a rectified video output.

```
sudo apt-get install ros-humble-image-proc
```

15. Copy the script from the `e_camera_config.py` file available in the **Task_3_resources** and paste it into the `camera_config.py` file located in the launch directory of the usb_cam package. **Note: Use VS Code to paste the content. Since the camera_config.py file is located in the root folder, you can easily save it with the help of VS Code.**
16. Now, you will relaunch the camera node

```
ros2 launch usb_cam camera.launch.py
```

17. Launch the image_proc by running the following command

```
ros2 launch image_proc image_proc.launch.py
```

18. After launching the camera node you can run the script `ros2_camera_test.py` which we have provided to you in `Task_3_resources` folder.

Note : Camera calibration should be done with the arena setup only.

---

2. Pen Pose Publish :

- After the calibration process, you need to write the script called `feedback_node.py` .
- You can reuse some code snippets from feedback.py in Task 2A for ArUco detection and perspective transformation.
- **IMPORTANT:** The image after perspective transform must be 500px x 500px. Also the outer corners of the corner ArUco markers (ID: 4, 8, 10, 12) must be used to ensure that the image after transformation perfectly lines up with the 220cm x 220cm borders of the arena.
- Now you can also reuse some code snippets from Task 2A to estimate marker positions and orientations.
- All teams must strictly follow `\pen1_pose` , `\pen2_pose` and `\pen3_pose` for topic names and `/geometry_msgs/2DPose` message type for publishing the positions of the pens and orientation of the bot, since this will matter in auto-evaluation.
- **IMPORTANT:** The units for x and y of the message is in pixels. `x=0` , `y=0` is the top left corner, `x=500` , `y=0` is the top right corner, `x=0` , `y=500` is the bottom left corner. theta of the message must of course be in radians, where `theta=0` is the desired robot orientation in HB23 theme.
- **IMPORTANT:** So far we have been sending the center of the marker (which is the center of the bot) as feedback to controller. And the controller is trying to take the center of the bot to the desired goal positions (and zero orientation).\ \ But now we'll need to take into account **the offset in pen position** from the center of the marker, since we want the pen to reach the desired goal position and which may not be the center of the bot in many bot designs.\ \ Making this compensation/correction is quite simple **if the bot's orientation is zero**. It is just a matter of adding/subtracting a fixed offset (in pixels) to the x and y coordinates of center of the robot to get the pen position.\ \ But when the orientation of the robot changes, the offset stays constant in bot frame but not in the global frame, therefore a rotation matrix will come into picture before the addition/subtraction. More explicitly, the constant offset in body frame must be transformed by the rotation matrix before adding to the (x,y) of the center.
- For theta simply publish the bot orientation.
- **(BONUS):** Create a visualization of the pen poses using openCV functions.

Submission

- Capture snapshots of the fisheye, undistorted, and perspective-transformed views, as well as snapshots of three bots positioned in the arena.
- Execute the feedback node and run rqt_graph.
- Capture a screenshot of the graph.
- **(BONUS)** Screen record a demo of the pen pose being visualized using openCV. This visualization will be very useful for debugging.
- Combine these images into videos.
- Upload the video with the title HB23__Task4B(For example: If your team ID is 1234 then, save it as HB23_1234_Task4B).
- Please note that while uploading the video on YouTube select the privacy setting option as **Unlisted**.
- Submit the unlisted youtube link on the eYRC Portal.
