Exercise 1

Duckiebot Assembly and Basic Development

How to access duckiebot dashboard

Once the duckiebot is turned on and the bot's Wi-Fi adapter starts blinking with the Wi-Fi logo (📶) appears on the LCD. The logo indicates that the bot is connected to a wireless network which enables remote control or communication with other devices over the internet or a local network (Desktop). The Dashboard for the duckiebot can be accessed via hyperlink (http://csc22907.local/) on your browser.

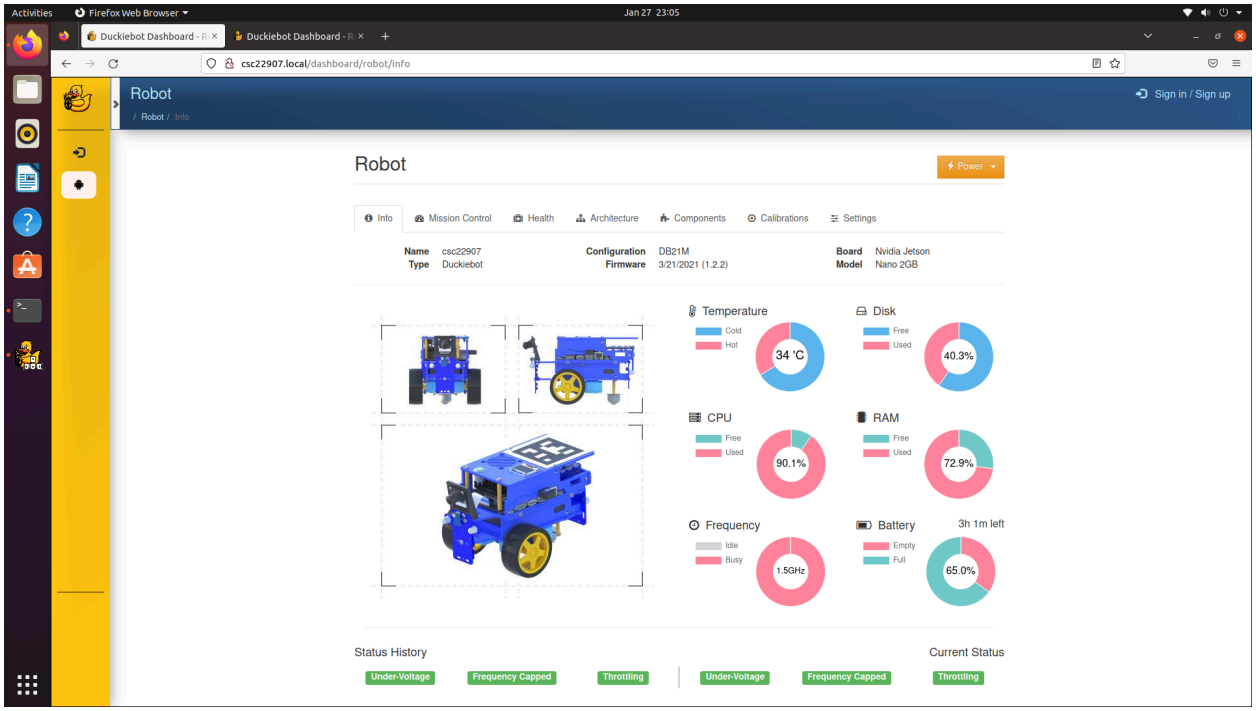

Fig 1: Illustration of dashboard (info tab)

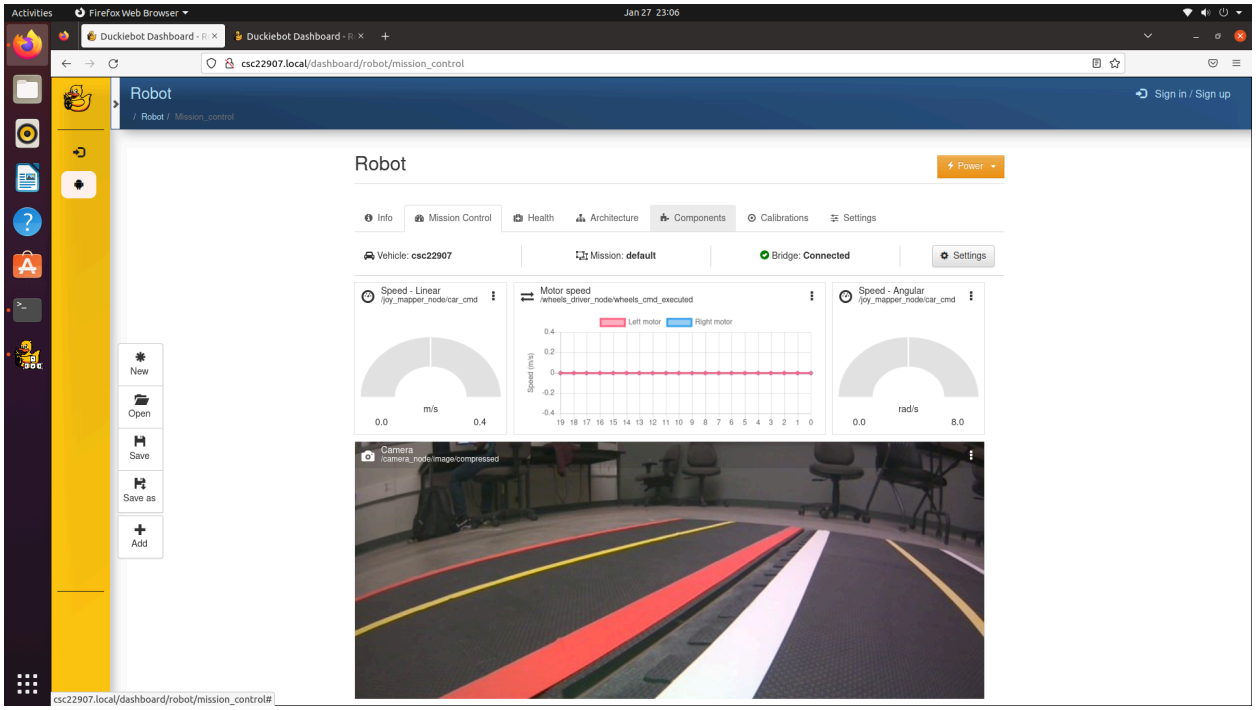

Fig 2: Illustration of dashboard (Mission Control tab)

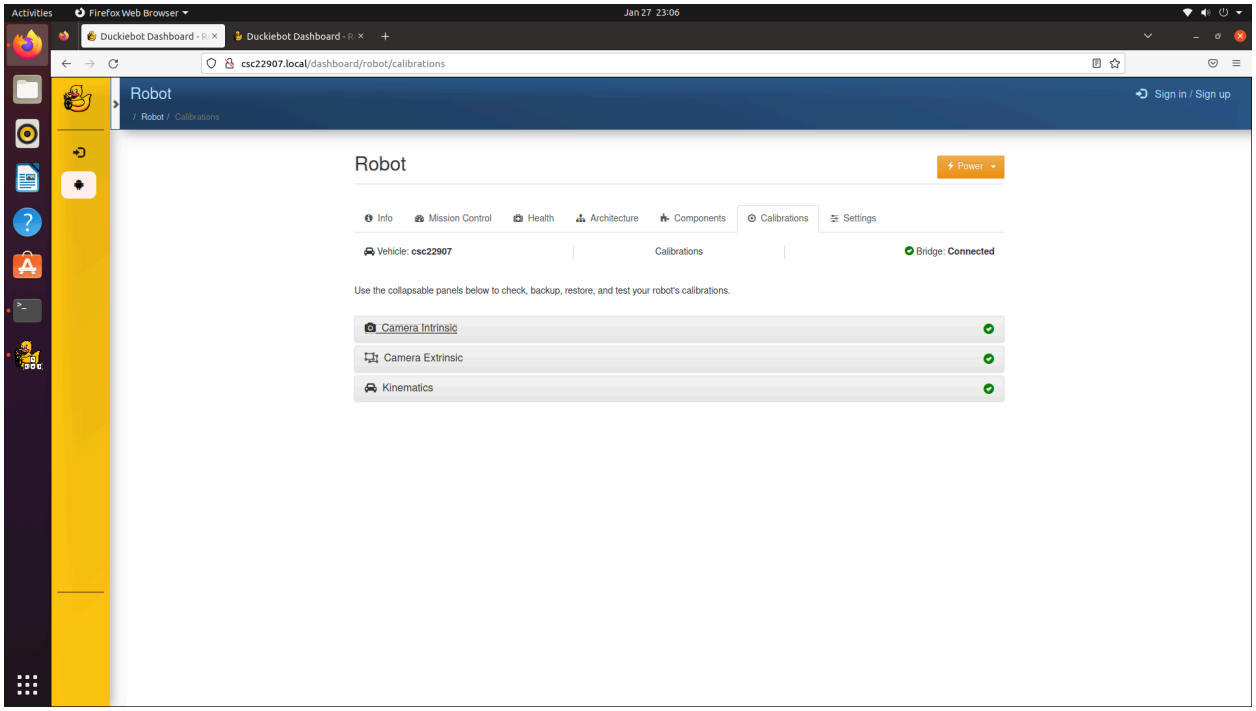

Fig 3: Illustration of dashboard (Calibration tab)

Dashboard Analysis

Fig 1 above the current status of the duckiebot. For example: Temperature, CPU and RAM usage, and the battery level etc.,.

Fig 2 illustrates the mission control tab where linear speed refers to how fast the robot moves along a straight line, Angular speed refers to how fast the robot is rotating around its center or another point (which could be the center point between the two wheels or any point along the axes of the wheels). The angular speed (to change direction) is important in order to make your robot move anywhere on a 2-D surface. Moverover, motor speed refers to the speed at which the motors (2 DC motors) of the robot are spinning. The graph represents the spinning speed of each motor with two different colors and spinning speed could be positive or negative depending on the motion of the robot (positive if moving forward and negative if robot is in reverse motion).

At last, Fig 3, which shows the calibration panel which could be used in order to check, backup, restore and test the calibration of different aspects of the robot. Calibration of a mobile robot involves adjusting its systems to ensure accurate movement, perception, and control. Different aspects of the robot require calibration to function correctly and efficiently.

How to communicate with Duckiebot

Following link is used to get the information about docker and used some part of it: https://docs-old.duckietown.org/daffy/duckietown-robotics-development/out/docker_basics.html

What is docker

Docker is a tool for portable, reproducible, and self-contained computing. It is used to perform operating-system-level virtualization, something often referred to as *containerization*.

Why is docker useful

Docker is useful when parameters, libraries, packages written in different languages, binary executables, system configurations, and anything else that your software might need to run correctly, also it makes sure that user has all of the setup correctly. Simply

put, it is like a virtual environment in python which is not associated with one language only. Most importantly, docker helps software to work across different hardware and operating systems along with true portability. Moreover, Programs running in such a container have access only to the resources they are allowed to and are completely independent of libraries and configurations of the other containers and the host machine.

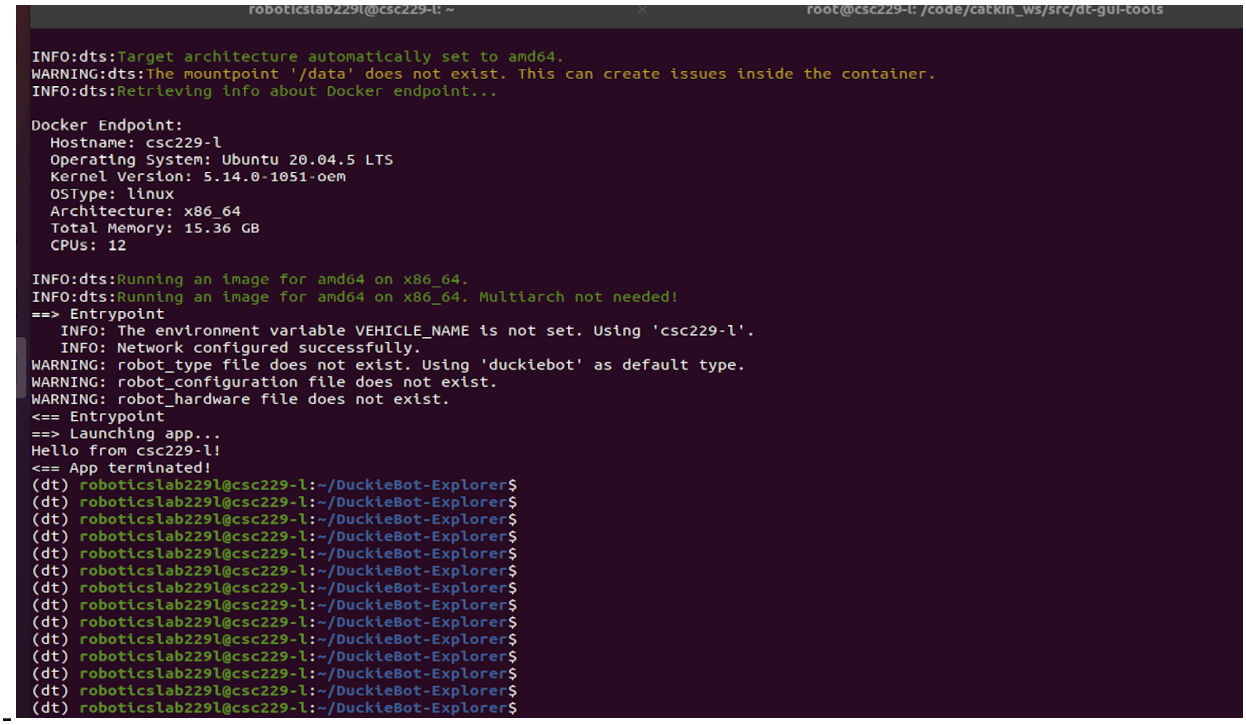

Fig 4: Screenshot of Duckiebot saying “Hello from csc22907!”

Analyzing Fig. 4

To run software on a robot’s operating system, we are using docker which provides containerization that helps software to meet the requirements and run successfully.

In order to achieve the objective of duckiebot saying “Hello from csc22907”, we have to fork a repository given in the duckietown documentation which provides Duckietown-compliant Docker image which helps to run your software on duckiebot. Adding a script.py file which contains the following code in the packages/my_package directory:

```
import os
message = "Hello from %s!" % os.environ['VEHICLE_NAME']
print(message)
```

```
// This code basically takes the vehicle name and adds it to a string which then
// gets printed on the terminal.
```

To run this code on duckiebot, make changes in launchers/default.sh by replacing the launching app with **dt-exec python3 -m "my_package.my_script".** The command clearly specifies that docker executes the python script “my_script” which is in directory “my_package”.

The output of the command also shows the docker endpoint which is the desktop where the command was executed.

Camera Calibration

Camera intrinsics calibration

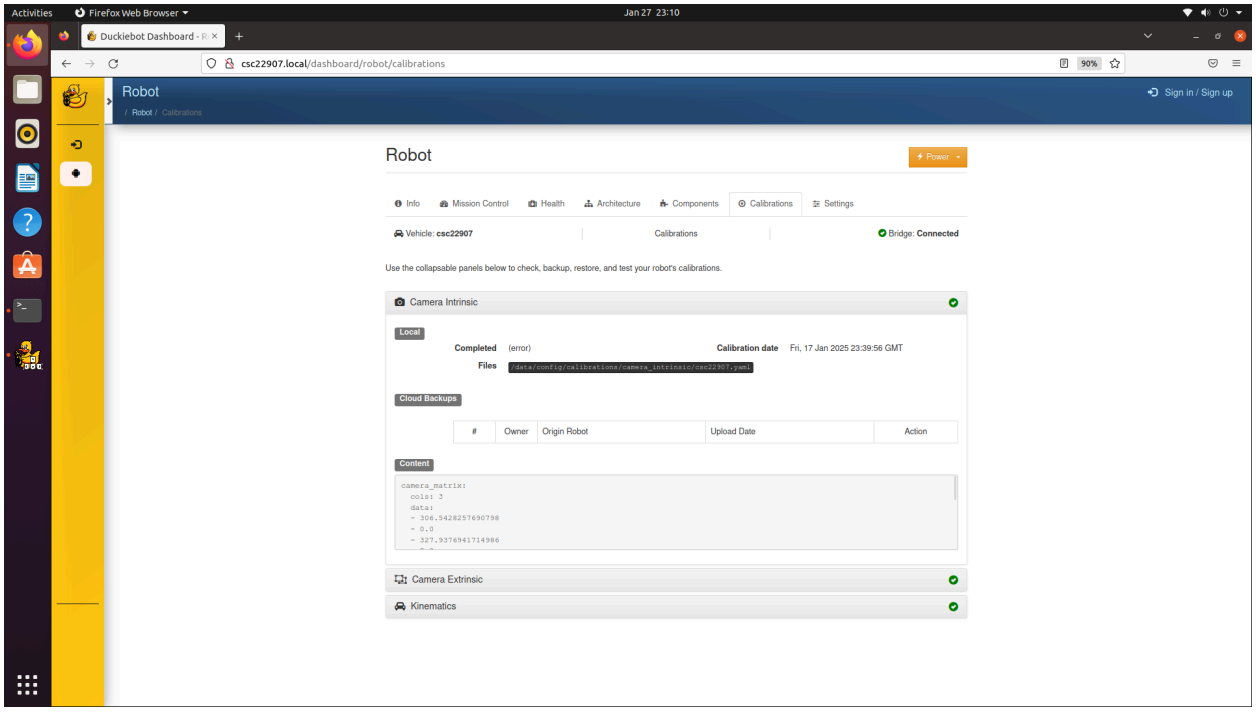

Fig 5: Screenshot of Camera Intrinsics panel

Following link is used to get the information about intrinsics calibration and used some part of it:
https://docs.duckietown.com/daffy/opmanual-duckiebot/operations/calibration_camera/index.html

What is the purpose of intrinsics calibrating the camera and how to achieve it?

Every camera is a little bit different, so we need to do a camera calibration procedure to account for the small manufacturing discrepancies. This process will involve displaying a predetermined pattern to the camera, and using it to solve for the camera parameters. The procedure is basically a wrapper around the ROS camera calibration tool.

To launch the intrinsics calibration program, command used "dts duckiebot calibrate_intrinsics csc22907". Now, Position the checkerboard in front of the camera until you see colored lines overlaying the checkerboard. You will only see the colored lines if the entire checkerboard is within the field of view of the camera. Move the checkerboard right/left, up/down, and tilt the checkerboard through various angles relative to the image plane. After each movement, make sure to pause long enough for the checkerboard to become highlighted.

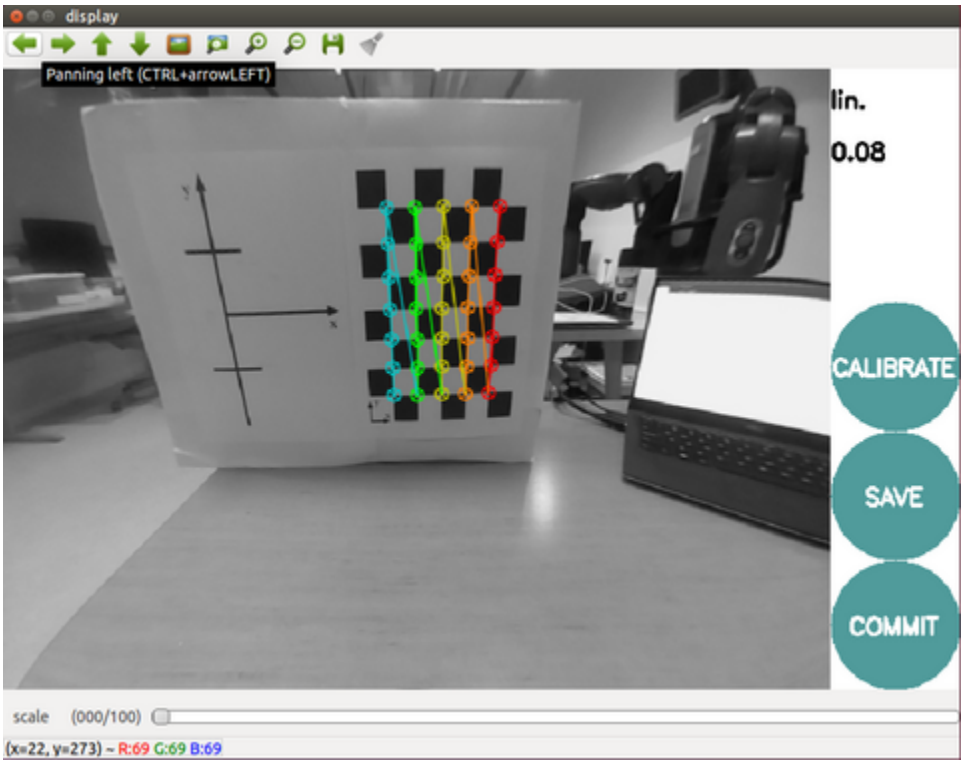

Fig 6: Screenshot of Camera Intrinsics calibration in process

Once satisfied with the calibration, you can save the results by pressing the "COMMIT" button in the sidebar. Then Save the calibration results on your Duckiebot: /data/config/calibrations/camera_intrinsic/csc22907.yaml

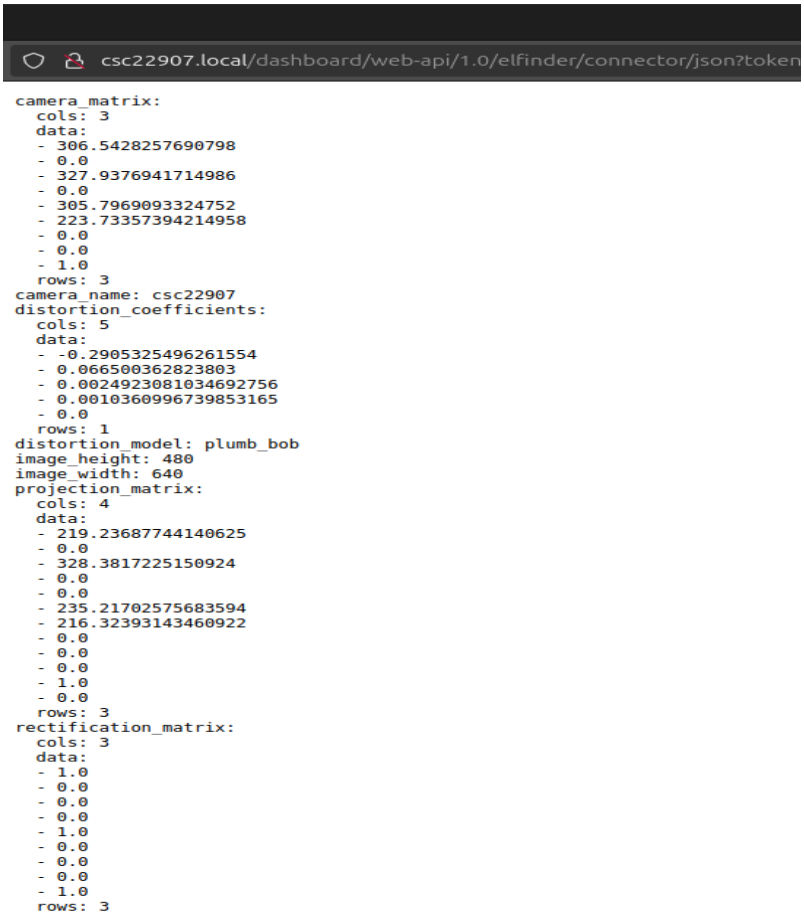

Fig 7: Screenshot of Camera Intrinsics .yaml file

Analysis of intrinsics .yaml file

Following link is used to get the analysis for intrinsics.yaml file and used some part of it: https://chatgpt.com/c/6799481e-7854-8004-b5f1-39a3bf721fd3

Fig. 7 shows that there is a camera matrix which is of size (3 X 3) with different values which is crucial for mapping 3D world coordinates to 2D image coordinates. Along with distortion_model: plumb_bob which is used for dealing with distortions along with distortion_coefficients captured by the model which is used to compensate for the radial and tangential distortions. Correcting these distortions ensures that straight lines in the real world appear straight in the image. The figure also shows the height and width of the image which represents the resolution of the camera feed. Projection matrix

combines intrinsics and extrinsic parameters and uses it to project 3D points to 2D image coordinates. At last, Rectification Matrix which is used in stereo vision to align images from multiple cameras., which is a identity matrix as duckiebot is only equipped with one camera in front.

Camera extrinsic calibration

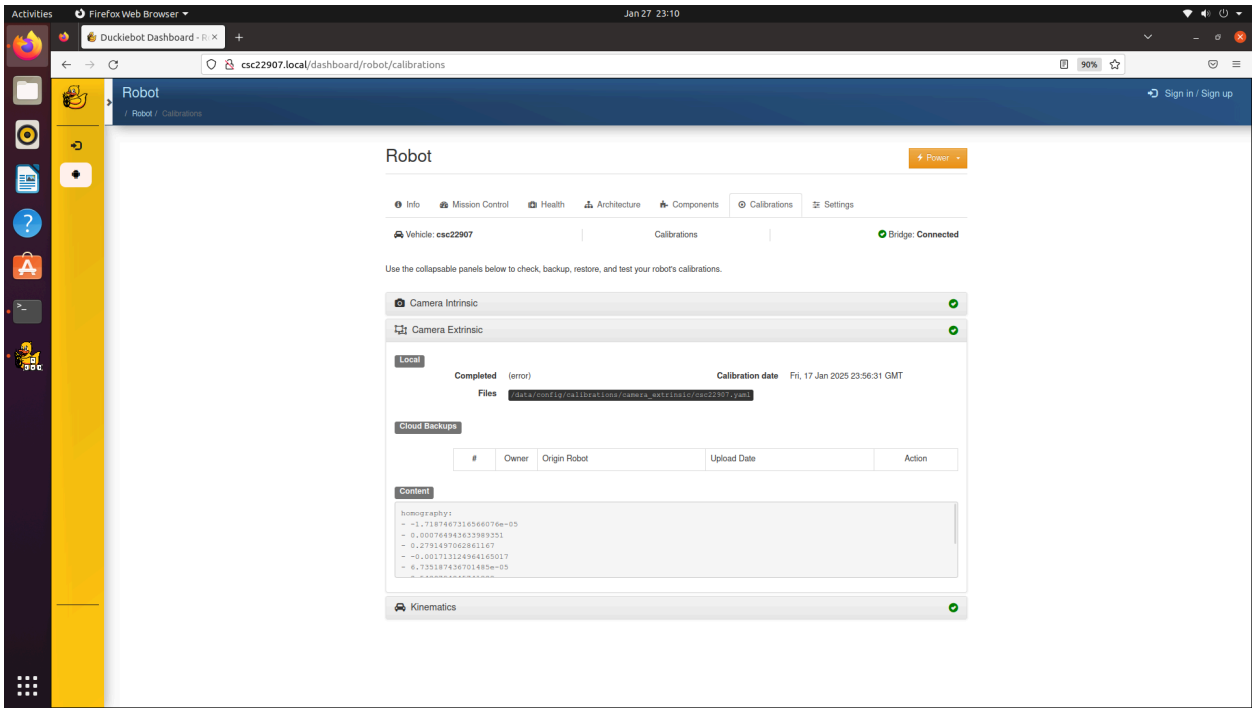

Fig 8: Screenshot of Camera extrinsics panel

Following link is used to get the information about extrinsics calibration and used some part of it:
https://docs.duckietown.com/daffy/opmanual-duckiebot/operations/calibration_camera/index.html

What is the purpose of extrinsic calibrating the camera and how to achieve it?

The purpose of **extrinsic calibration** of the Duckiebot is to determine the relationship between the camera and the robot's frame of reference. This calibration is crucial for ensuring accurate perception and navigation.

To launch the extrinsics calibration program, command used “dts duckiebot calibrate_extrinsics csc22907”. To save the calibration results on your Duckiebot: /data/config/calibrations/camera_extrinsics/csc22907.yaml

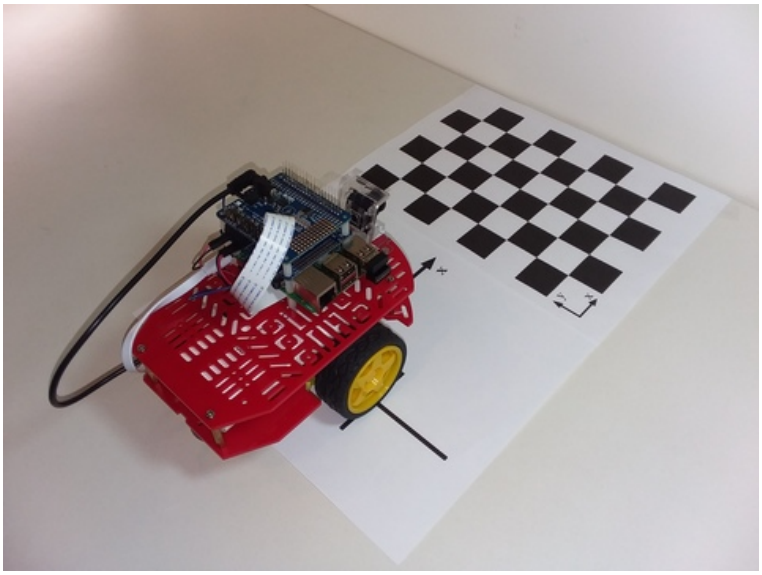

Fig 9: Screenshot of Camera extrinsics calibration in process

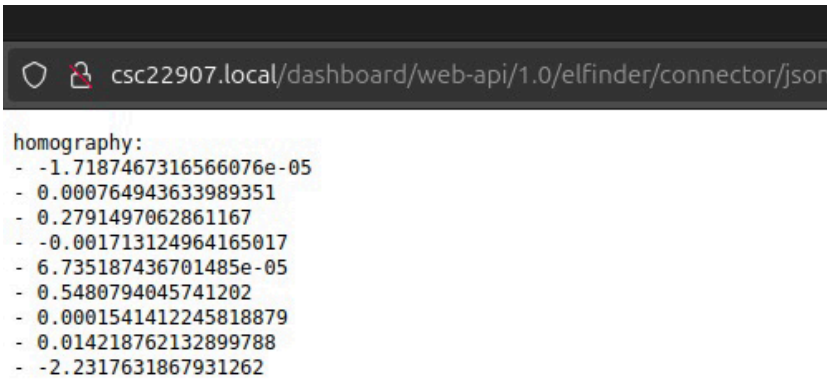

Fig 10: Screenshot of Camera extrinsics .yaml file

Analysis of extrinsics .yaml file

Following link is used to get the analysis for extrinsics.yaml file and used some part of it: https://chatgpt.com/c/67998eaf-db94-8004-822a-1ef7f4937a22

The extrinsics.yaml file represents the homography matrix which is used to describe the relationship between two images of the same scene taken from different viewpoints or perspectives. It is essentially a matrix that maps points from one plane to another, and it's crucial for tasks like camera calibration. Also, the homography matrix helps map the 2D points in the image to 3D points in the world. The value in the matrix is used to compensate for distortion introduced by the camera's position.

Kinematics Calibration

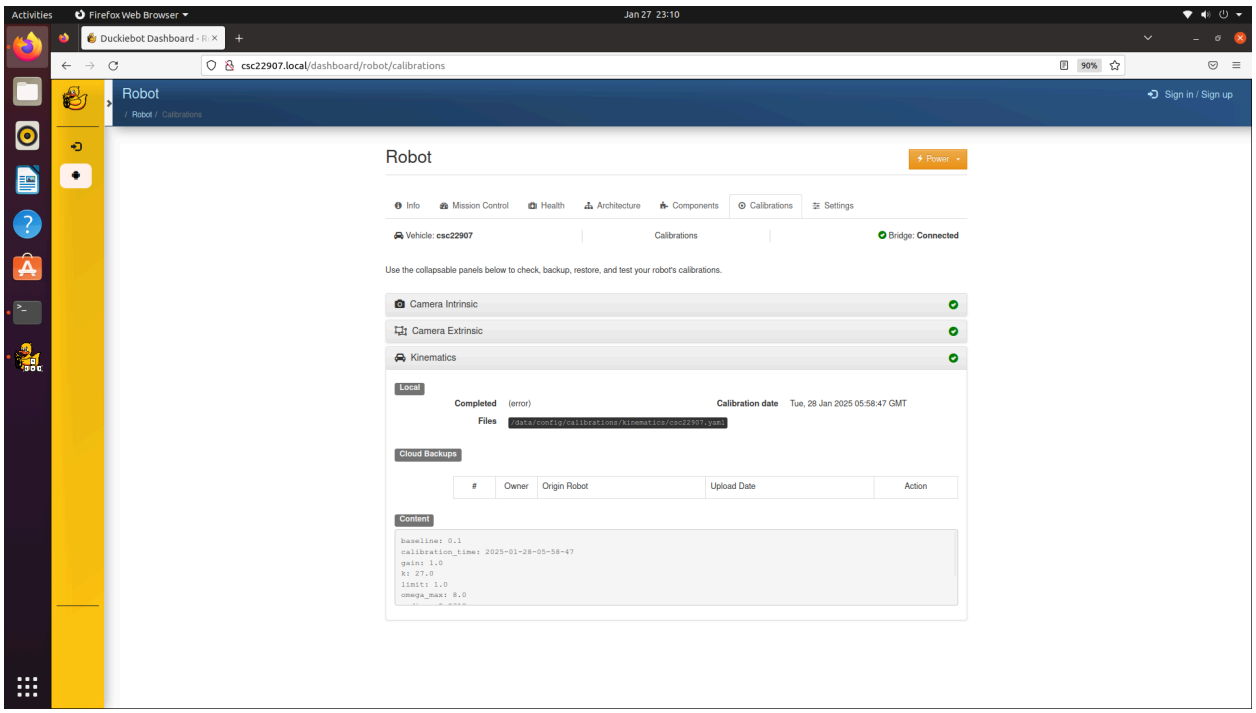

Fig 11: Screenshot of Camera kinematics panel

Following link is used to get the information about kinematics calibration and used some part of it: https://docs.duckietown.com/daffy/opmanual-duckiebot/operations/calibration_wheels/index.html

Say your robot started to veer right when driving in a straight line. What do you think is causing the drift? What could your robot do to detect and adjust for this drift?

Calibrating the motors on a Duckiebot ensures that the robot's movement is accurate and consistent. The motors may have slight variations in speed and power due to manufacturing differences, which can affect the robot's performance, such as how straight it drives or how it turns.

To perform the calibration, calibrating the trim parameter is crucial. The trim parameter is set to 0.00 by default, under the assumption that both motors and wheels are perfectly identical. Change the value of the trim parameter by running the following commands:
"dts start_gui_tools csc22907"
"rosparam set /csc22907/kinematics_node/trim [trim_value]"

Now, If the Duckiebot drifted to the left side of the tape which represents a straight line, use negative value for trim_value and if the Duckiebot drifted to the right side of the tape, use positive value for trim_value. The trim_value should be in between 0.0 to 1.0 which is maximum. When all done, save the parameters by running:

"rosservice call /csc22907/kinematics_node/save_calibration".

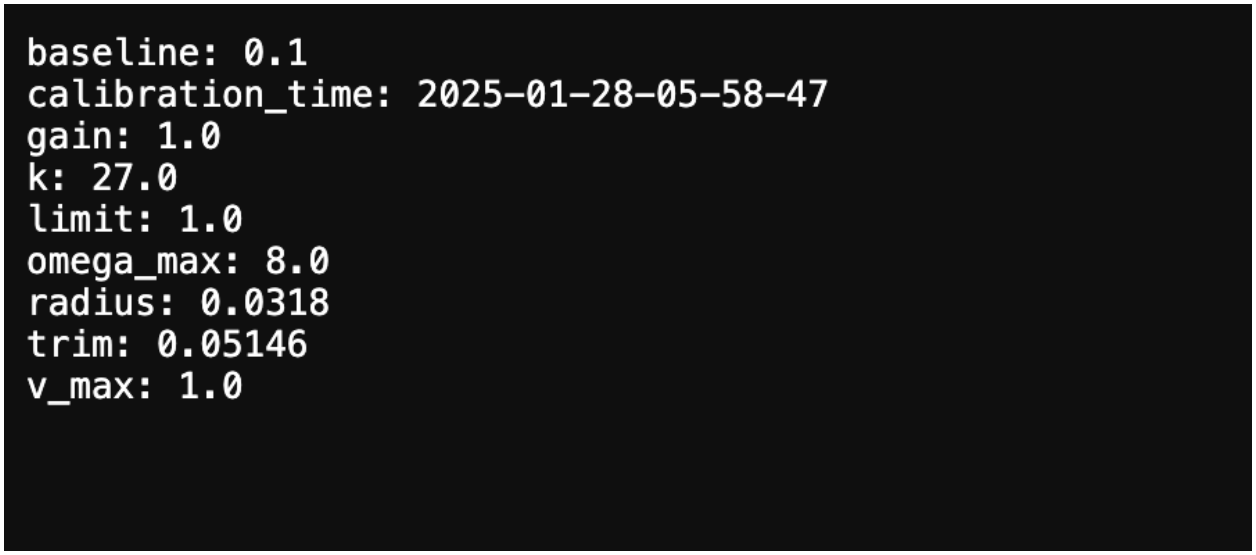

Fig 12: Screenshot of Camera kinematics .yaml file

Analysis of kinematics .yaml file

Following link is used to get the analysis for kinematics.yaml file and used it:
https://chatgpt.com/c/67998eaf-db94-8004-822a-1ef7f4937a22

Fig.12 illustrates the kinematics.yaml file contains parameters obtained after motor calibration in the Duckiebot. Observations:

Baseline: 0.1 (The distance between the wheels is 0.1 meters.)
Gain: 1.0 (The motor gain is set to 1.0, which is by default)
Limit: 1.0 (The maximum command limit is set to 1.0, indicating full power usage when required. This states that the trim value should not exceed 1.0 which is maximum).
Omega_max: 8.0 (The maximum angular velocity is 8.0 rad/s, determining how fast the Duckiebot can rotate.)
Radius: 0.0318 (The wheel radius is 3.18 cm, affecting speed calculations.)
Trim: 0.05146 (Indicates an adjustment to balance motor outputs, likely compensating for small mechanical imperfections.)
V_max: 1.0 (Maximum linear velocity is 1.0, meaning the bot operates at full speed under normal conditions.)

The kinematics.yaml file indicates a balanced and well-calibrated motor setup, but the trim value suggests minor adjustments were made to ensure equal motor performance.

Video of 2m straight driving

Click the link to download the video

Explanation of video

Making a non-calibrated Duckiebot to move in a straight line is an important task. To achieve this objective only wheel calibration is required to make sure that the duckie bot moves in a straight line and this task is the foundation for other complex tasks performed by a bot. Calibration can be done by following commands:
“dts start_gui_tools csc22907”
“rosparam set /csc22907/kinematics_node/trim [trim_value]”

To get a straight walk, trial and error method is used to get a perfect trim_value which makes bot to follow a straight line.

Duckiebots are sensitive to even minor irregularities in their environment, such as bumps or inconsistencies in the flooring. If the robot encounters a bump where two mats connects, this can cause sudden, unexpected shifts in the bot's movement. If the Duckiebot drifts less than 10 centimeters after adjusting the trim value, you can stop calibrating the trim parameter, as this is good enough for most tasks. However, if the drift exceeds 10 centimeters, further adjustments are needed, and you should continue tweaking the trim value until the drift is within the acceptable range.

Video of lane following demo

Click the link to download the video (demo 1)

Click the link to download the video (demo 2)

Explanation of video

Before performing the lane following demo, wheel calibration and camera (Intrinsics and extrinsics) calibration is MUST. To run DuckieTown build in Lane following demo, with the following command, then press 'a':

```
"dts duckiebot demo --demo_name lane_following --duckiebot_name csc22907 --package_name duckietown_demos"
```

and

```
"dts duckiebot keyboard_control csc22907"
```

How it works and what could affect this?

In the **lane following demo**, the camera in front of Duckiebot captures real-time data and feeds it to the operating system for processing. Then Lane Detection Algorithm detects the lane markings on the road which can be seen in the video above. In the video, it can be observed that the Duckiebot sometimes runs over the marked lines on the road because if the lane has tight curves or sharp turns, the Duckiebot may struggle to adjust its steering quickly enough to stay centered in the lane.

If the demo is run under low lighting conditions, the camera might not capture enough data to follow the lane effectively which is not the case in the video above. Also, with uneven flooring (intersections between mats), the Duckiebot could experience issues maintaining its trajectory.

Key takeaway

- **Importance of Calibration:** Camera and motor calibration are essential for ensuring the Duckiebot operates correctly. Camera calibration ensures the robot can accurately interpret its surroundings, while motor calibration helps it move in a straight line or follow a path without veering off.

- **Docker as a Useful Tool and its importance:** Docker provides a consistent development environment, which reduces the complexity of setting up dependencies ensures that the code works the same way across different systems

- **Challenges with Motor Drift:** If the robot veers off course, it could be due to motor imbalances, calibration issues, or environmental factors like lighting. Addressing these issues requires fine-tuning.

- **Learning the Interaction Between Sensors and Actuators:** The lab highlights how sensors (like the camera) and actuators (like motors) work together to enable autonomous behavior which will be the focus of this course in upcoming exercises.

This course takes a lot of patience and time to get the expected outcome. To get to that level where implementing your own path following algorithms, this exercise was a good start to get to know all the tools and basics skills to achieve it.

References:

- https://wiki.ros.org/camera_calibration
- https://docs-old.duckietown.org/daffy/duckietown-robotics-development/out/docker_basics.html
- https://docs.duckietown.com/daffy/opmanual-duckiebot/operations/calibration_camera/index.html
- https://chatgpt.com/c/6799481e-7854-8004-b5f1-39a3bf721fd3

- https://docs.duckietown.com/daffy/opmanual-duckiebot/operations/calibration_camera/index.html
- https://chatgpt.com/c/67998eaf-db94-8004-822a-1ef7f4937a22
- https://docs.duckietown.com/daffy/opmanual-duckiebot/operations/calibration_wheels/index.html
- https://chatgpt.com/c/67998eaf-db94-8004-822a-1ef7f4937a22
- https://chatgpt.com/