Narwhal

Chess game manipulator

Operating Guide

INSTITUTE OF FIELD ROBOTICS AND AUTOMATION, KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI

This project was done under the supervision of Unicorn Tech

First release, August 2022

Contents

1. Chessboard State Estimation and Chess AI

1.1 Environment Initialization

1.1.1 Initialize environment

All softwares (except low-level control) were written as nodes in **module89** ROS2 package which located at `/home/fiborobotlab/dev_ws/src`. ROS2 environment automatically intialized each time when new section is opened by several commands inside `/home/fiborobotlab/.bashrc`, thus you don't need to source ROS2 installation. However, you still need to source the package installation before run or launch node(s) inside the package, this can be done in 2 ways.
Using standard `bash` command with setup script.

» `bash /home/fiborobotlab/dev_ws/install/setup.bash`

Or use shorter command written in alias.

» `ins`

Use this command to run all nodes at once

» `ros2 launch module89 main.launch.py`

1.2 Camera Setup

1.2.1 Camera Installation

Install single RGB camera above target chessboard and face down toward target chessboard with resolution of 1080p (1920x1080 pixels). Make sure target chessboard is inside camera frame, side-alignment is unnecessary (software can perform auto-alignment after calibration). We use camera coordinate as diagram below

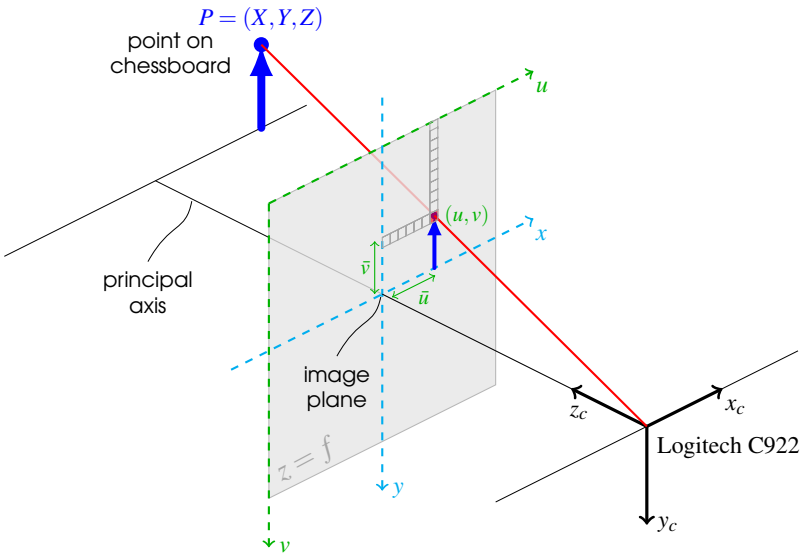

Figure 1.1: Camera coordinate used in Narwhal project

1.2.2 Camera Calibration

Default camera intrinsic matrix and distortion coefficients is for Logitech C922 USB camera. When change the camera, you need to modify these values at `package://config/camera_config.json` for new installed camera.

Chessboard Pose Calibration & Color Clustering Lock

Source package when open new terminal to include module89 in the environment.

```
» ins
```

Next, run `chessboard estimator` node to begin calibration of chessboard using command:

```
» ros2 run module89 chessboard_estimator.py
```

If camera is connected correctly, this window will opened and begin streaming automatically at 720p (scaled down from 1080p) along with several buttons as in Figure 1.2. After each installation or camera displaced, the chessboard pose matrix might be changed, so new calibration might be done by

- Click at 4 corners of chessboard (regarding black/white side bar).
- If you mistaken click any point other than chessboard corner, you can use **Clear Points** button to reset point(s) stored in buffer.
- Make sure you click exactly 4 points (any other amount of point will not be accepted by auto-calibration function and nothing will happen) before press **Confirm** button to confirm and apply selections. This node will use these 4 points with `solvePnP()` along with some image processing to estimate aligned pose of chessboard.

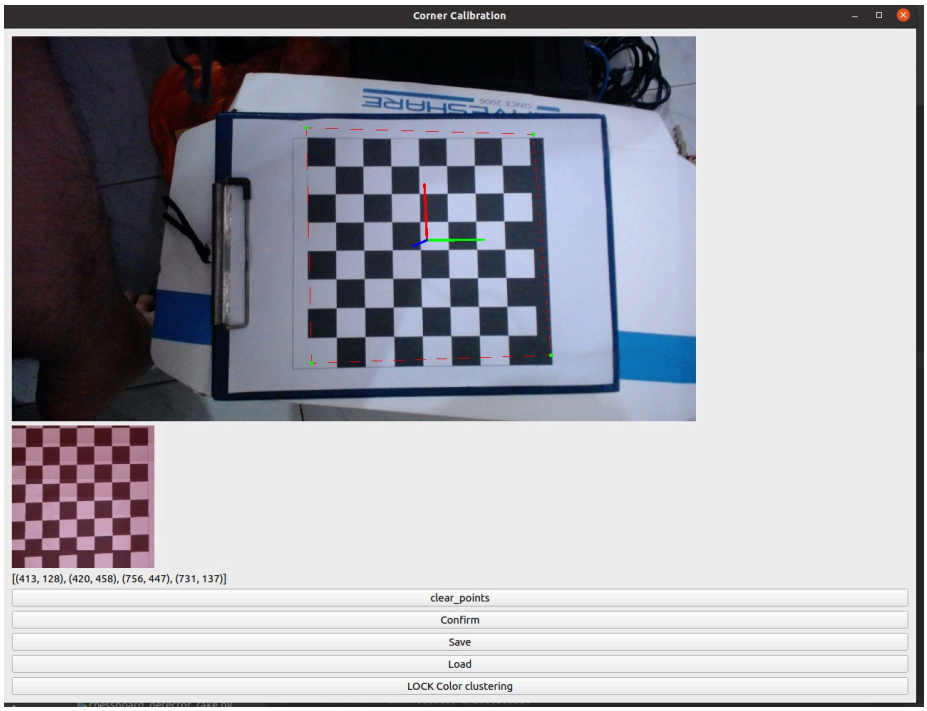

Figure 1.2: GUI for chessboard pose calibration written by Qt5

- Applied corners will only stored in memory, press **Save** button if you want to store calibration for later use without 4 points input again.
- By clicking **Load** button, saved calibration will restored from disk if exist. If nothing has moved after most recent calibration, this is the only button you need to press after run this node.
- After chessboard is perfectly aligned in GUI, the system will beginning clustering process automatically then press **LOCK Color clustering** button to stop updating the model and lock color which most of them near corresponding side (note: this button can be toggle to unlock-relock)

1.2.3 Chessboard tile preparation

Before feeding image to classification model, we have to preprocess them to a Tensor with shape (64, 32, 32, 3) which is 3 channel images with size 32x32 using fixed batchsize of 64. Thus, we need to divide raw image for each tile, resize, and zero-pad chessboard tile.

1.3 AI (Vision)

1.3.1 Chess piece classification

We use only one lightweight-CNNs based classification model created with **TensorFlow2** and optimized by OpenVINO which be able to run on **Intel UHD Graphics Xe G4 48EUs** via **OpenCL** using **OpenVINO runtime** at **>200fps**. This model located at

`/home/fiborobotlab/models/classification/chess/chess_ir`

it has **8 nodes** in output layer corresponding to probabilities of that tile being

- empty tile
- occupied tile with black piece
- occupied tile with gold piece
- occupied tile with green piece
- occupied tile with pink piece
- occupied tile with silver piece
- occupied tile with wood piece
- occupied tile with yellow piece

After feed data in to the model and output node which corresponding to empty tile has the highest score, it will be interpreted as empty tile.

1.3.2 Chess piece color clustering

Outputs from **Chess piece classification** are used as feature for clustering except feature from first node which is empty tile indicator. We use K-mean clustering to occupied tile in to 2 cluster and bind it with nearest side automatically.

1.4 AI (Chess Game)

Use command below to run both **Game State Controller** and **Game AI** in same node.

» `ros2 run module89 pseudo_state.py`

1.4.1 Game State Controller

Our classification model is simple and quite accurate. However, this procedure only give us chess piece existence and their color. Role of this node is to track completed state of chessboard from the beginning using just chess piece existence and their color by indicate if state of chessboard has changed and match the legal move. To interact with game state you need to open main GUI (Figure 1.3) from directory below

» `/home/fiborobotlab/Narwhal/x86-64`

This GUI also has tab to control Narwhal robot (Figure 1.4).

1.4.2 Game AI

We replace original Chess engine with Stockfish and build bit-board move generator for **AVX-512** running on unsupported CPUs might got error.

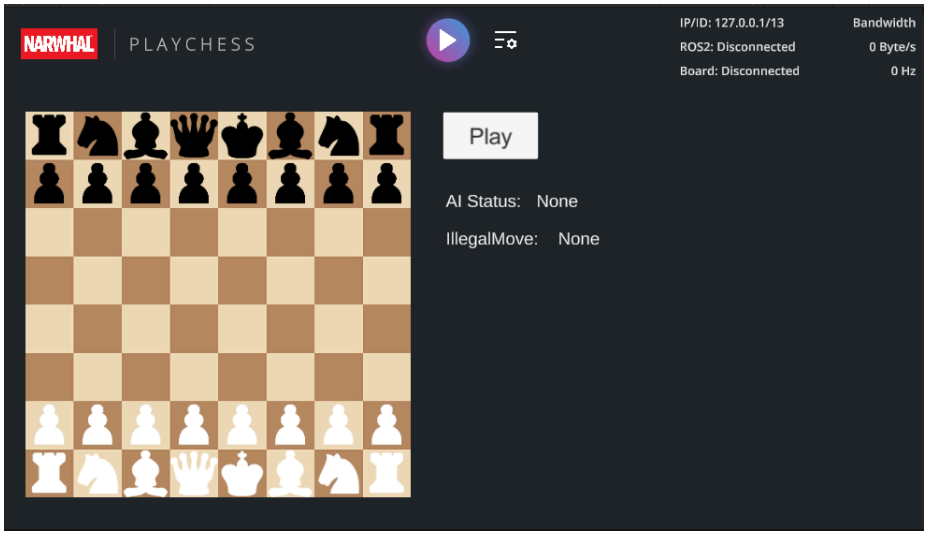

Figure 1.3: GUI for chessboard state visualization written by Unity3D

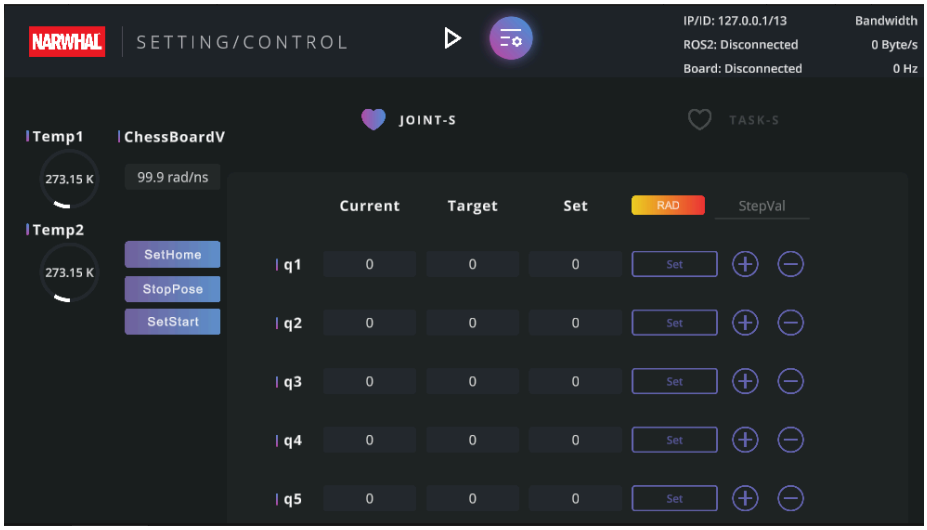

Figure 1.4: GUI for robot manual control written by Unity3D