OpenGL Project User Guide

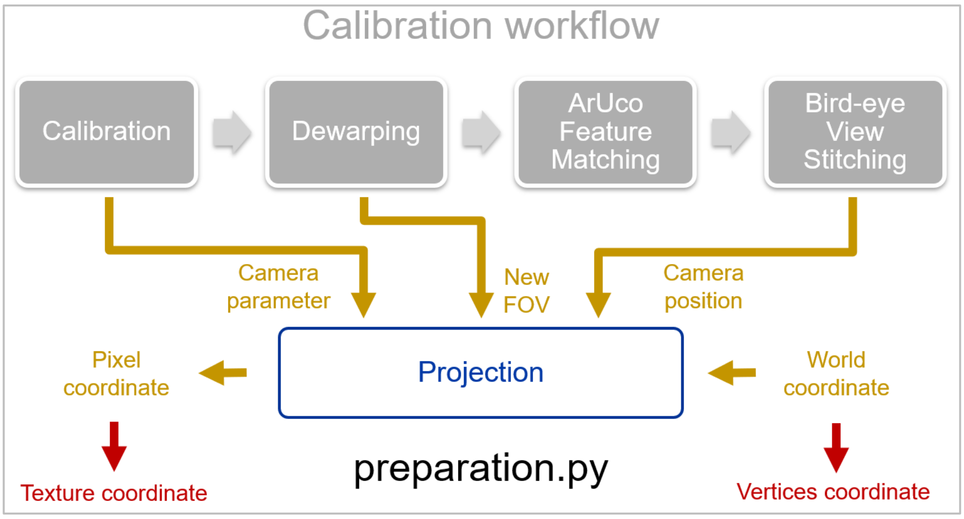
Gitlab project: ProjectiveTexture_BowlGrids – original branch: dewarp

Gitlab link: http://ntustcgalgit.synology.me:8000/360-surround-view/projectivetexture_bowlgrids

Objective: to create a 360 view from multiple fisheye images

Requirement: the camera parameters, dewarping parameters, position and angle of all camera are already known from the previous steps

1. Preparation step



Preparation before OpenGL rendering

Requirement	Example
Camera parameters	In “preparation.py”: `K = np.array([[696.0405939705771, 0.0, 640.0],` `[0.0, 694.8901322581238, 640.0],` `[0.0, 0.0, 1.0]])` `xi = np.array([[0.7100823796212375]])` `D = np.array([[-0.2894618100329196, 0.06539145347156397, 0.0002772246603313237, 9.3`
Dewarping parameters	In “preparation.py”: `vehicle_size = 532 / DIM[0]` `rescale = 1 / 12.0` (*Please refer to Appendix A for more detail)
Camera position	In “camera_config_180.json”:

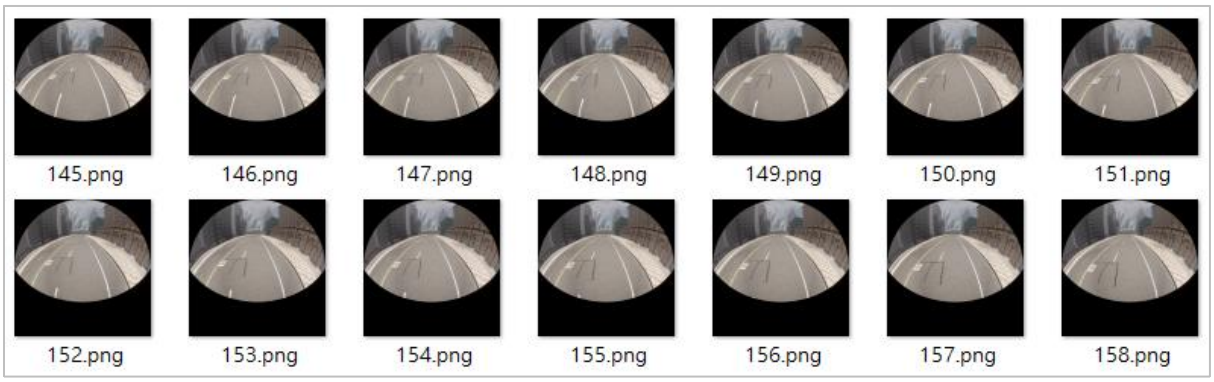
	`` "camera_config2": { "camera1": 0, "camera2": 1, "camera3": 8, "camera4": 4, "camera5": 9, "camera6": 7 }, "camera_config3": { "camera1": 0, "camera2": 1, "camera3": 2, "camera4": 3, "camera5": 4, "camera6": 5, "camera7": 6, "camera8": 7 }, "camera_list": [{ "fov": 180, "position": [5.17, 0, 2.695], "rotation": [0, -45, 0] }, ```

List of parameters needed

First, prepare the data as the following structure:

```
ProjectiveTexture
├── resources
|   ├── Vehicle1_Way0
|   |   ├── 180
|   |       ├── 0
|   |       └── 1
|   |       └── 2
|   |       └── 3
|   |       └── 4
|   |       └── 5
|   |       └── 6
|   |       └── 7
|   |       └── camera_config_180.json
|   └── Bowl_vertices.txt
|   └── Bowl_indices.txt
└── preparation.py
└── main.cpp
└── ...
```

Copy image data into resources



Images in folder “/0”

Then run the “preparation.py” to complete the preparation step. It will generate two files: “Bowl_vertices.txt” and “Bowl_indices.txt”. They will be used in OpenGL rendering flow. For more information about the configuration, please refer to Appendix B.

```
$python3 preparation.py --min_R 0.5 --max_R 0.5 --
height 0.26 --stackCount 14 --json
./resources/Vehicle1_Way0/180/camera_config_180.jso
n --config camera_config2 --sixcam 1
```

Command line for “preparation.py”

2. OpenGL Implementation

First, install these packages for OpenGL (or GLES):

```
sudo apt-get install pkg-config
sudo apt-get install libglfw3-dev
sudo apt-get install libgles2
sudo apt-get install libgles2-mesa-dev
sudo apt-get install libgles2-mesa
sudo apt-get install libxinerama-dev libxcursor-dev libxi-dev
```

Then, change the mode of the following files:

```
chmod +x ./doall.sh ./buildrun.sh ./run.sh
```

For the first time of compiling the C code, you have to run “./doall.sh” command to make and build, after it make successfully once, later on you can just use “./buildrun.sh” whenever you modify the code, images or “Bowl_vertices.txt”.

```
$./doall.sh
```

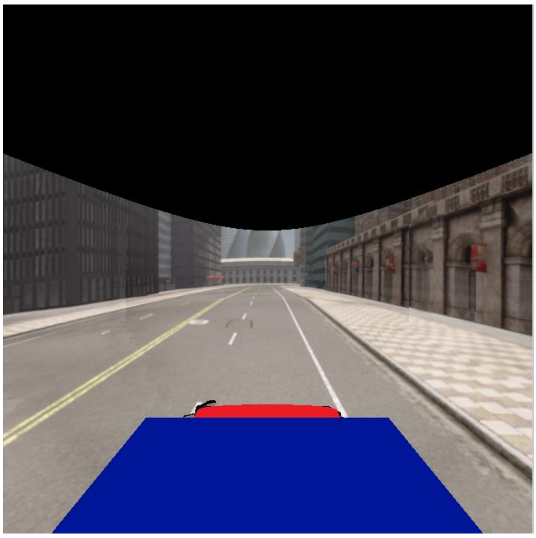
Compile the C code

For this step, if you are running on PC and encounter some errors, you may modify these lines in “CMakeLists.txt” and disable WAYLAND (WAYLAND is required for development boards), then run “./doall.sh” again.

```
set(GLFW_BUILD_WAYLAND TRUE)
set(GLFW_BUILD_X11 FALSE)
```

```
#set(GLFW_BUILD_WAYLAND TRUE)
#set(GLFW_BUILD_X11 FALSE)
```

Comment out these 2 lines in “CMakeLists,txt”



Rendering result

Additional information:

1. If you have prepared a sequence of input images (such as “resources/Vehicle1_Way0”) and want to see the truck moving forward, simply uncomment these lines of code in “main.cpp”:

```cpp
        while (!glfwWindowShouldClose(window))
        {
                // std::this_thread::sleep_until(nextTime);
                // nextTime += framerate{ 1 };

                glfwPollEvents();
                RenderScene();
                //RenderGUI();
                glfwSwapBuffers(window);

                // Calculate FPS
                crntTime = glfwGetTime();
                timeDiff = crntTime - prevTime;
                counter++;
                if (timeDiff >= 2.0){
                        // Creates new title
                        std::string FPS = std::to_string((1.0 / timeDiff) * counter);
                        std::string ms = std::to_string((timeDiff / counter) * 1000);
                        //std::string newTitle = "Simple example - " + FPS + "FPS / " + ms + "ms";
                        //glfwSetWindowTitle(window, newTitle.c_str());
                        fflush(stdout);
                        std::cout << "\r" + FPS + "FPS / " + ms + "ms ";

                        // Resets times and counter
                        prevTime = crntTime;
                        counter = 0;
                }
                // if (sourceImageData.size() > 0)
                // {
                //      std::lock_guard lock(surroundTextureLock);
                //      //std::cout << sourceImageData.size() << std::endl;
                //      const std::vector<TextureData>& tdata = sourceImageData.front();
                //      surroundViewTexture.UpdateTextures(tdata);
                //      for (int i = 0; i < tdata.size(); ++i)
                //      {
                //              delete[] tdata[i].data;
                //      }
                //      sourceImageData.pop();
                // }
        }
```

Allow updating textures

2. You can change the size of the truck model in "VehicleModel.cpp":

VehicleModel.cpp 9.39 KiB

```cpp
#include "VehicleModel.h"
#define POSITION_LOCATION  0
#define TEX_COORD_LOCATION 1
using namespace std;

VehicleModel::VehicleModel(const std::string& Filename)
{
        scale_x = 1.0;
        scale_y = 1.0;
        scale_z = -0.75;
```

NXP Development Board – iMX8QM:

1. **Development Board Installation**
   A. Install UUU (Universal Update Utility) from https://github.com/nxp-imx/mfgtools/releases.

   B. Download Linux release from https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/embedded-linux-for-i-mx-applications-processors:IMXLINUX.

Linux Current Release

Release and Documentation	Build Sources	Supported Platforms/Binary Demo Files	Incremental Releases
Linux 6.1.22_2.0.0 Release Date: Jun 2023 Documentation • i.MX Linux Release Notes • i.MX Linux User's Guide • i.MX Linux Reference Manual • i.MX Porting Guide • i.MX Yocto Project User's Guide • i.MX VPU API Reference Manual • i.MX Graphics User's Guide • i.MX Machine Learning User's Guide • Harpoon User's Guide • i.MX 8M Plus Camera and Display Guide • i.MX Digital Cockpit HW Partitioning Enablement for i.MX 8QuadMax • i.MX DSP User's Guide • uPower User Guide • NXP Wi-Fi Driver Features and Release Notes • NXP Demo Experience User Guide	• See README on instructions for each release. • SCFW Porting Kit 1.15.0 • AACPlus Codec • Verisilicon IDE • uPower Firmware Porting Kit 1.3.0	• i.MX 93 EVK (Pre-production) • i.MX 8ULP EVK • i.MX 8DXL EVK • i.MX 8M Plus EVK • i.MX 8M Nano DDR3L EVK • i.MX 8M Nano EVK • i.MX 8M Mini EVK • i.MX 8M Quad EVK • i.MX 8QuadXPlus(C0) MEK • i.MX 8QuadMax MEK • i.MX 7ULP EVKB • i.MX 6UltraLite EVK, i.MX 6ULL EVK, i.MX 6ULZ EVK, i.MX 7Dual SABRE-SD • i.MX 6SLL EVK • i.MX 6QuadPlus, i.MX 6Quad, i.MX 6DualPlus, i.MX 6Dual, i.MX 6DualLite, i.MX 6Solo, i.MX 6SoloX SABRE-SD • i.MX 8M EVKs boot image(SystemReady-IR certified)	

   C. Install Linux latest version into eMMC with UUU.

Windows

- `git clone https://github.com/nxp-imx/mfgtools.git`
- `cd mfgtools`
- `git submodule init`
- `git submodule update`
- `open msvs/uuu.sln with Visual Studio 2017`

   D. Turn iMX8QM to serial download mode and install it with UUU.

```
D:\shared_folder\mfgtools\msvc\x64\Debug>uuu -b emmc_all LF_v6.1.22-2.0.0_images_IMX8QMMEK/imx-image-full-imx8qmmek.wic
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.5.109-10-g6881886

Wait for Known USB Device Appear...
```

   E. Setup uboot environment to enable hdmi output using following command:

```
=> setenv fdt_file imx8qm-mek-hdmi.dtb
=> saveenv
Saving Environment to MMC... Writing to MMC(0)... OK
=> boot
switch to partitions #0, OK
```

2. Toolchain Installation

A. Install imx-manifest repo https://github.com/nxp-imx/imx-manifest.

B. Using following command to choose machine and distro.

```
(base) Kneron@ubuntu:~/Desktop/imx8$ MACHINE=imx8qmmek DISTRO=fsl-imx-xwayland s
ource ./imx-setup-release.sh -b build
```

C. Build the toolchain with bitbake.

```
(base) Kneron@ubuntu:~/Desktop/imx8/build$ bitbake meta-toolchain
Loading cache: 100% |##########################################| Time: 0:00:01
Loaded 5305 entries from dependency cache.
Parsing recipes: 100% |##########################################| Time: 0:00:01
Parsing of 3444 .bb files complete (3442 cached, 2 parsed). 5307 targets, 380 skipped, 17 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
```

D. Use iMX8QM's environment to build the program after installation.

```
(base) Kneron@ubuntu:~/Desktop/imx8/build/tmp/deploy/sdk$ source /opt/fsl-imx-xwayland/6.1-mickledore/environment-setup-armv8a-poky-linux
(base) Kneron@ubuntu:~/Desktop/imx8/build/tmp/deploy/sdk$ aarch64-poky-linux-gcc --version
aarch64-poky-linux-gcc (GCC) 12.2.0
Copyright (C) 2022 Free Software Foundation, Inc.
```

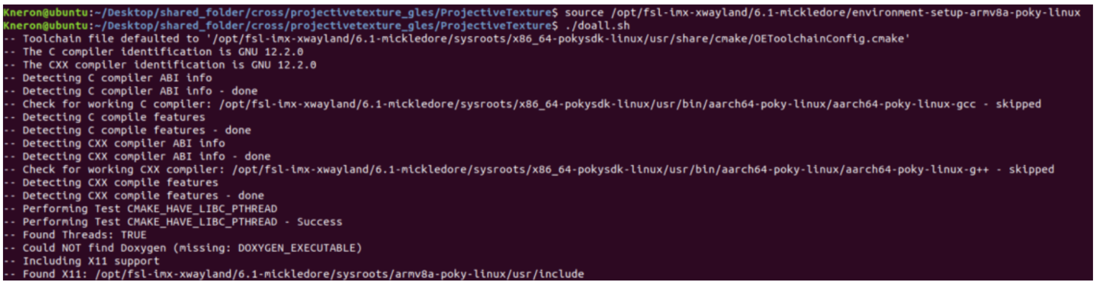
E. Install complete toolchain.

```
(base) Kneron@ubuntu:~/Desktop/imx8/build$ bitbake imx-image-multimedia -c populate_sdk
Loading cache: 100% |#########################################################################
Loaded 5305 entries from dependency cache.
Parsing recipes: 100% |#######################################################################
Parsing of 3444 .bb files complete (3442 cached, 2 parsed). 5307 targets, 380 skipped, 17 mas
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "2.4.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "aarch64-poky-linux"
MACHINE              = "imx8qmmek"
DISTRO               = "fsl-imx-xwayland"
DISTRO_VERSION       = "6.1-mickledore"
TUNE_FEATURES        = "aarch64 armv8a crc crypto"
TARGET_FPU           = ""
meta
```

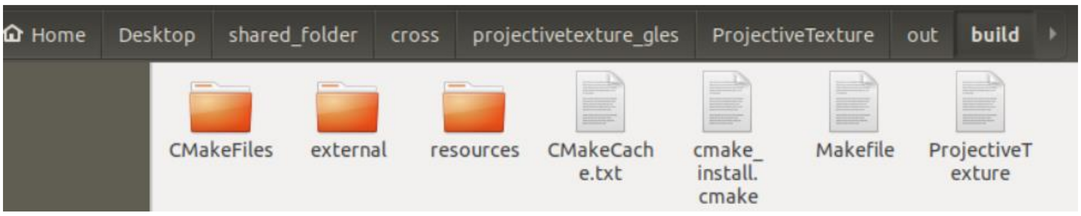
F. Get the script after installation.

G. Execute the script and then build the environment.

H. Source the environment and compile the program.



I. Get the binary code after cross-compiling.



J. Transport the output folder to iMX8QM and execute.

References:

1. https://www.nxp.com/document/guide/getting-started-with-the-i-mx-8quadmax-mek:GS-iMX-8QM-MEK
2. https://www.nxp.com/docs/en/user-guide/IMX_LINUX_USERS_GUIDE.pdf

Camera Positioning

1. Download the project http://ntustcgalgit.synology.me:8000/360-surround-view/camera_positioning.

2. Install python packages:
   $pip install -r requirements.txt

3. Execute
   A. Camera Calibration:
      $cd src
      $python calibration.py --root {your_root}
   B. Generate Homograph Matrices
      $cd src
      $python stitching_aruco.py --root {aruco_root}
   C. Output Json File
      $cd src
      $python positioning.py --root {aruco_root}

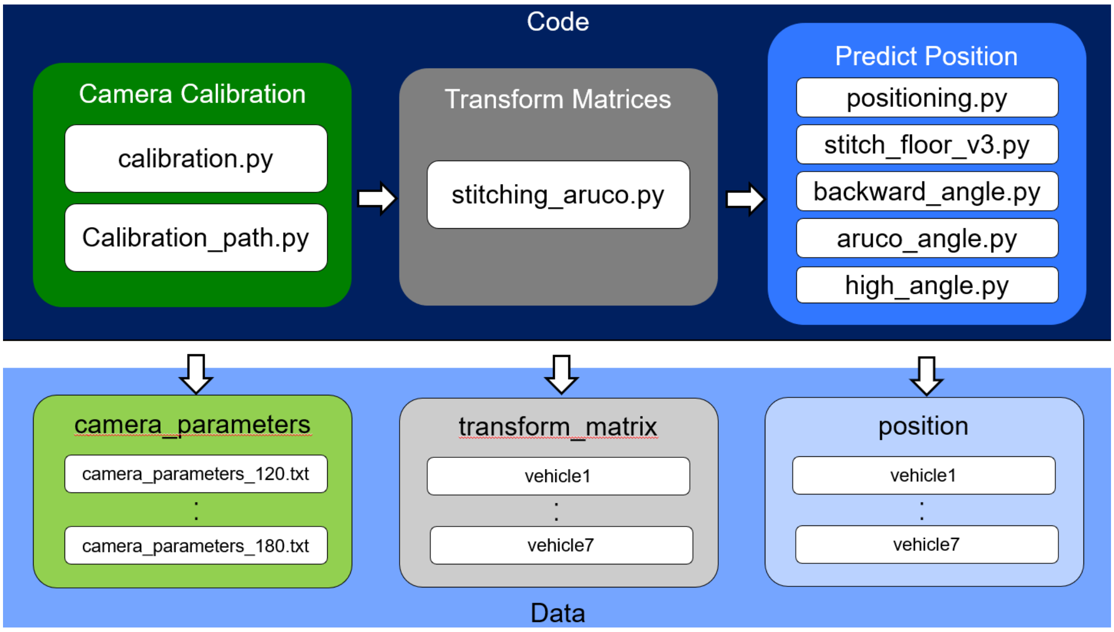
4. Dataset architecture:
   A. Calibration:

```
root/
    pattern120/
                *.png
    pattern140/
                *.png
    pattern160/
                *.png
    pattern180/
                *.png
```

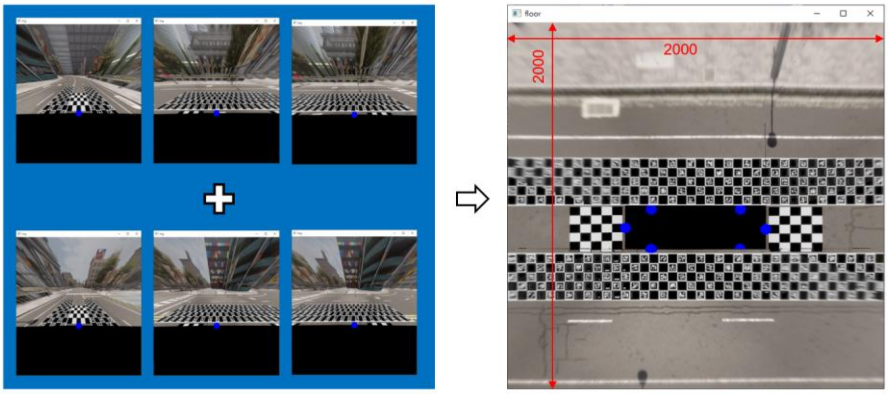
B. ArUco markers:

```
root/
    chessboard/
            Charuco1.png
    vehicle1/
        camera_config.json
        120/
            *.png
        140/
            *.png
        160/
            *.png
        180/
        *.png
    vehicle2/
        camera_config.json
        120/
            *.png
        140/
            *.png
        160/
            *.png
        180/
        *.png
    .
    .
    .

    vehicle7/
        camera_config.json
        120/
            *.png
        140/
            *.png
        160/
            *.png
        180/
        *.png
```

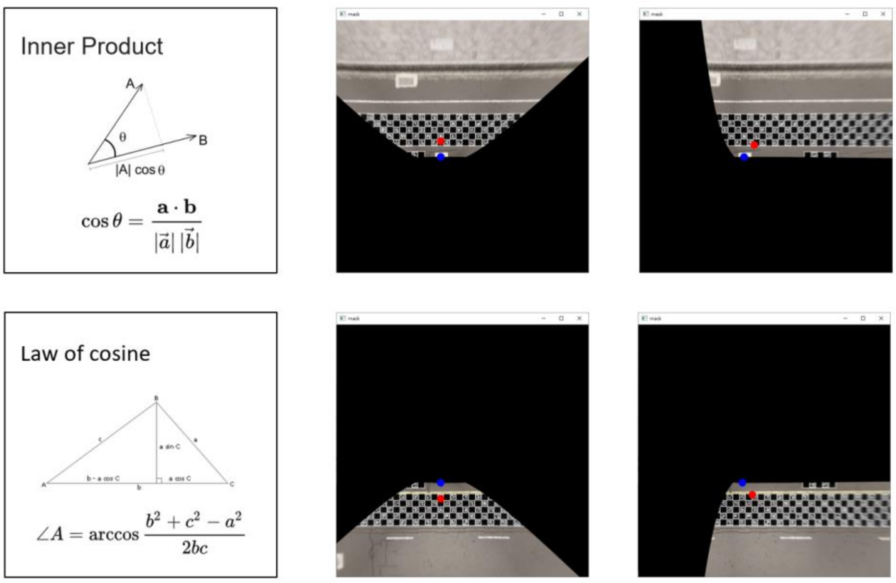
5. Project architecture:



A. stitch_floor_v3.py: Compute camera plane position through convert blue points from origin images to stitched image.



B. backward_angle.py: Compute backward angle.

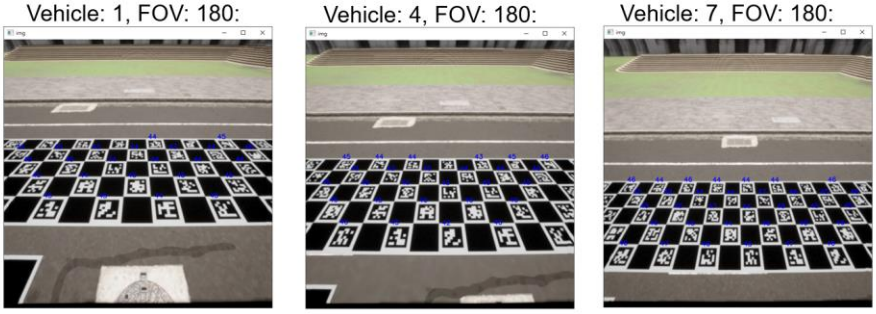


C. high_angle.py: Using OpenCV ArUco markers to Predict angle.

```
mtx = np.array([[640,0,640],[0,640,640],[0,0,1]])
dist = np.array([0,0,0,0])
```

```
ang_tmp = rvec[j][0][0]*180/math.pi - 90
```

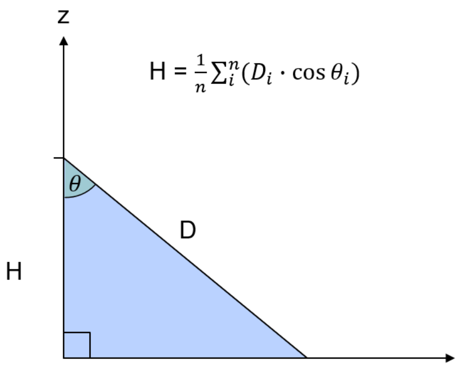
Vehicle: 1, FOV: 180: | Vehicle: 4, FOV: 180: | Vehicle: 7, FOV: 180:



angle mean:-44.75(deg) | angle mean:-44.81(deg) | angle mean:-45.69(deg)

D. high_angle.py: Compute the distance and the angle of ArUco markers through the function “estimatePoseSingleMarkers”. Then, compute camera’s high through the following equation:

$$H = \frac{1}{n}\sum_{i}^{n}(D_i \cdot \cos\theta_i)$$



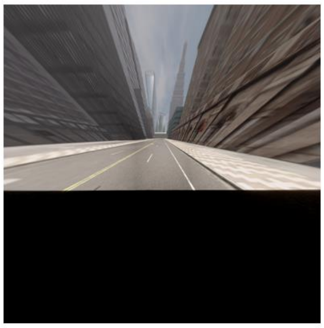
Appendix

A. Dewarping parameter

When you change a new set of camera, in addition to doing camera calibration, you have to do the dewarping again to find the suitable rescale value and vehicle size in new dewarped image.

Original Image

Default Dewarping



Current Dewarping

“Undistort.py” can help you to do the job.

First, you have to modify “Undistort.py”, to update the camera matrix and distortion parameters which are known from calibration process.

◆ initUndistortRectifyMap()

```
void cv::omnidir::initUndistortRectifyMap ( InputArray        K,
                                            InputArray        D,
                                            InputArray        xi,
                                            InputArray        R,
                                            InputArray        P,
                                            const cv::Size &  size,
                                            int               m1type,
                                            OutputArray       map1,
                                            OutputArray       map2,
                                            int               flags
                                          )
```

Python:

```
cv.omnidir.initUndistortRectifyMap( K, D, xi, R, P, size, m1type, flags[, map1[, map2]] ) -> map1, map2
```

```
#include <opencv2/ccalib/omnidir.hpp>
```

Computes undistortion and rectification maps for omnidirectional camera image transform by a rotation R. It output two maps that are used for **cv::remap()**. If D is empty then zero distortion is used, if R or P is empty then identity matrices are used.

Parameters

K	Camera matrix $K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$, with depth CV_32F or CV_64F
D	Input vector of distortion coefficients (k_1, k_2, p_1, p_2), with depth CV_32F or CV_64F
xi	The parameter xi for CMei's model
R	Rotation transform between the original and object space : 3x3 1-channel, or vector: 3x1/1x3, with depth CV_32F or CV_64F
P	New camera matrix (3x3) or new projection matrix (3x4)
size	Undistorted image size.
m1type	Type of the first output map that can be CV_32FC1 or CV_16SC2 . See **convertMaps()** for details.

List of parameters needed in dewarping step

Undistort.py 3.13 KiB

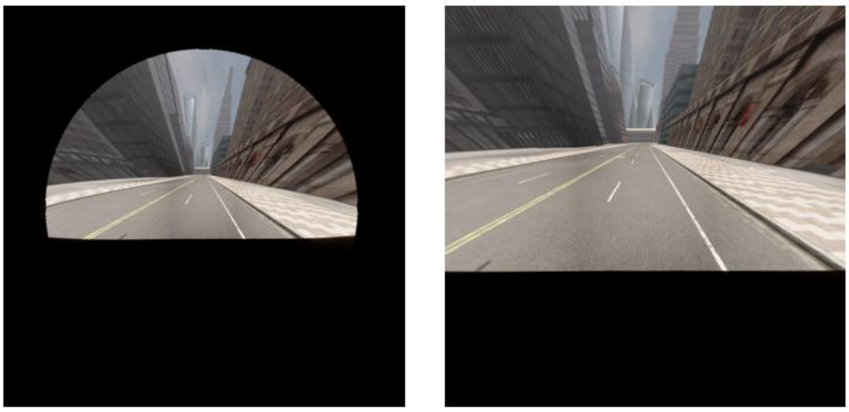
```python
import numpy as np
import cv2
import glob
import argparse

############### Configure ################

# Image to be dewarped (different image for different FOV)
image_path = './resources/Vehicle1_Way0/180/0/1313.png'

# Intrinsic and distortion parameters for each camera (from calibration)
# 180
DIM    = (1280, 1280)
K      = np.array([[696.0405939705771, 0.0, 640.0],
                              [0.0, 694.8901322581238, 640.0],
                              [0.0, 0.0, 1.0]])
xi     = np.array([[0.7100823796212375]])
D      = np.array([[-0.2894618100329196, 0.06539145347156397, 0.0002772246603313237, 9.335154355321922e-05]])

```

Modify these lines if camera parameters were changed

Second, you have to do the experiments, try to run "Undistort.py" with different arguments to find the optimal rescale value and vehicle size:

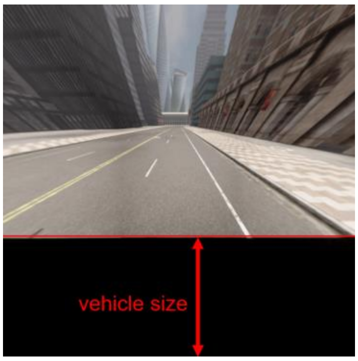
```
python3 Undistort.py -r 6 -s 440
```

Command line for "Undistort.py"



Rescale = 12 Rescale = 6

The effect of different rescale value



The meaning of vehicle size

FOV	Rescale	Vehicle size
180	12	532
160	6	440
140	3.5	280
120	2.5	113

Recommended values for different FOV

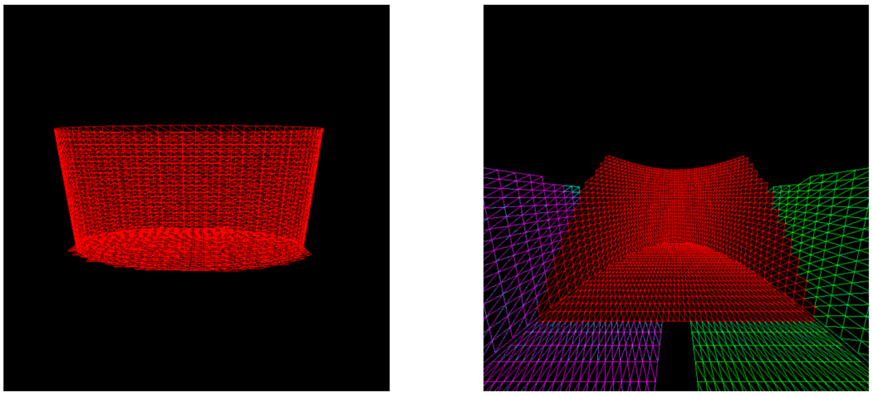
Now you have the correct rescale value and vehicle size, don't forget to modify "preparation.py" as well:

```
vehicle_size = 440 / DIM[0]
rescale = 1 / 6.0
K = np.array([[732.2980640352715, 0.0, 640.0],
              [0.0, 734.1208095538103, 640.0],
              [0.0, 0.0, 1.0]])
xi = np.array([[0.5852392164668814]])
D = np.array([[-0.30279907728116356, 0.07291584096650669, 0.0008114609772521696, 3.0358130968293173e-05]])
```

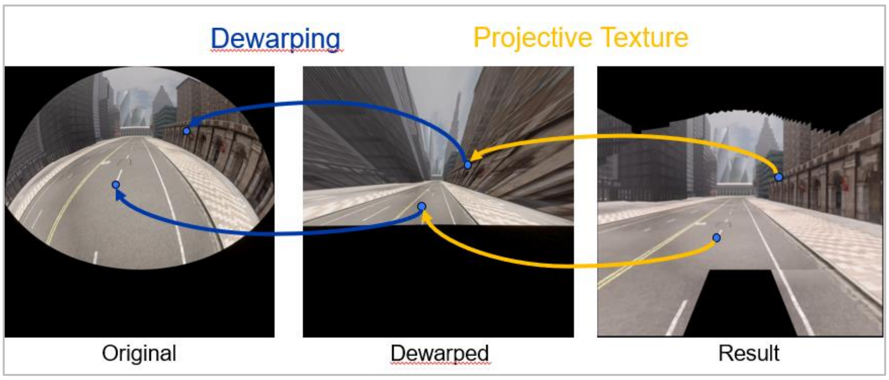
Parameters to be modified in "preparation.py"

B. Detail about "preparation.py"

The bowl model is constructed from vertices, each vertex can be projected to different image planes. The world coordinate from bowl model is projected to "Dewarped" plane by using projection matrix and view matrix. The coordinate from "Dewarped" plane is projected to "Original" image plane by "Undistort" function.



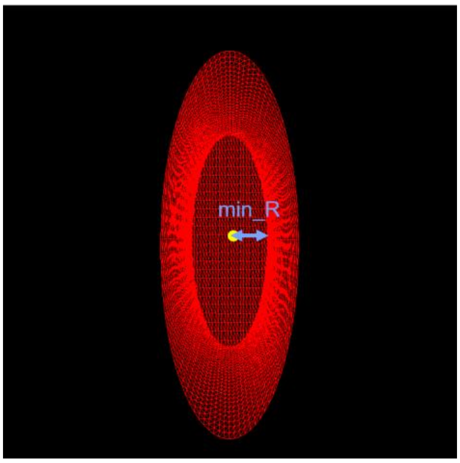
Bowl model　　Grid projection



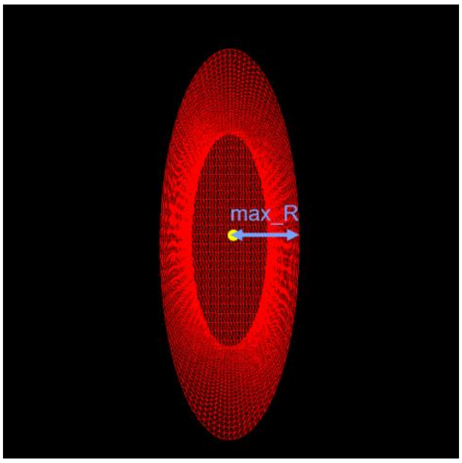
Demonstration of the idea behind Projective Texture

Argument explanation:

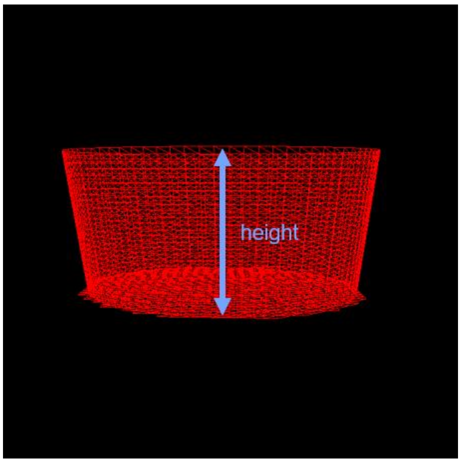
--min_R: minimum radius (bowl bottom)



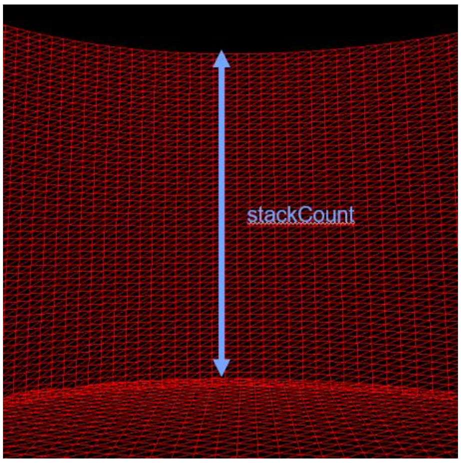
--max_R: maximum radius (bowl top)



--height: the bowl's height



--stackCount: number of stack (or resolution)



The resolution for other parts of the bowl will be inferred based on the configurations above, for example:

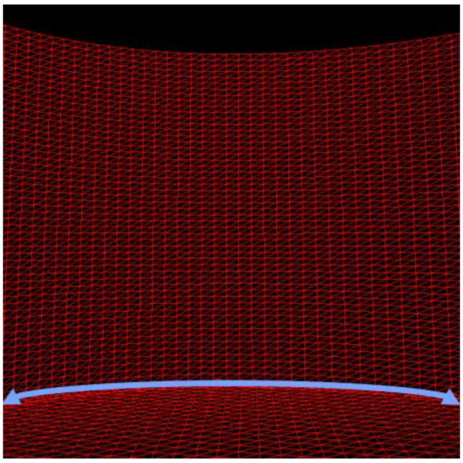
For the bottom of the bowl, its circumference is:

$$C = 2 \times \pi \times \min\_R$$

To calculate the number of stacks for C:

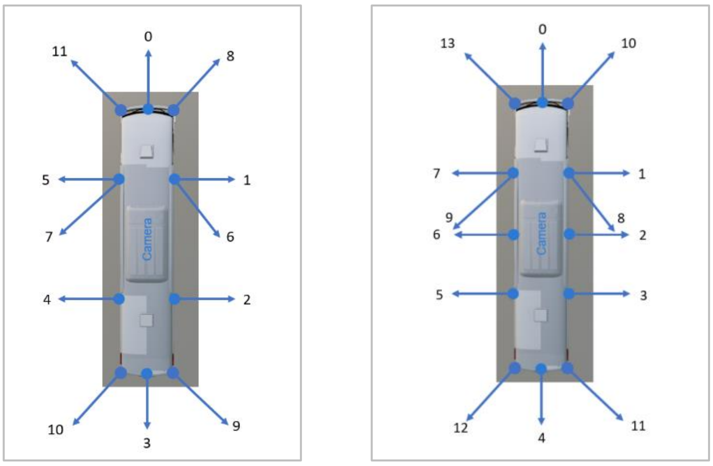
$$\frac{C}{stack_C} = \frac{height}{stackCount}$$

$$\rightarrow stack_C = \frac{C \times stackCount}{height}$$



The number of stacks for C

--sixcam: to scoop out the floor below the vehicle, we have to know the coordinate of the edges surrounding the vehicle, these parameters can be extracted from json file. Depend on which vehicle model we are going to use (6 camera or 8 camera), the top, right, bottom and left camera ID are different, so we have to choose the correct camera ID for each model.



Different vehicle models have different camera position
(Vehicle1 ~ Vehicle4 are 6 cam model, and Vehicle5 ~ Vehicle7 are 8 cam model)

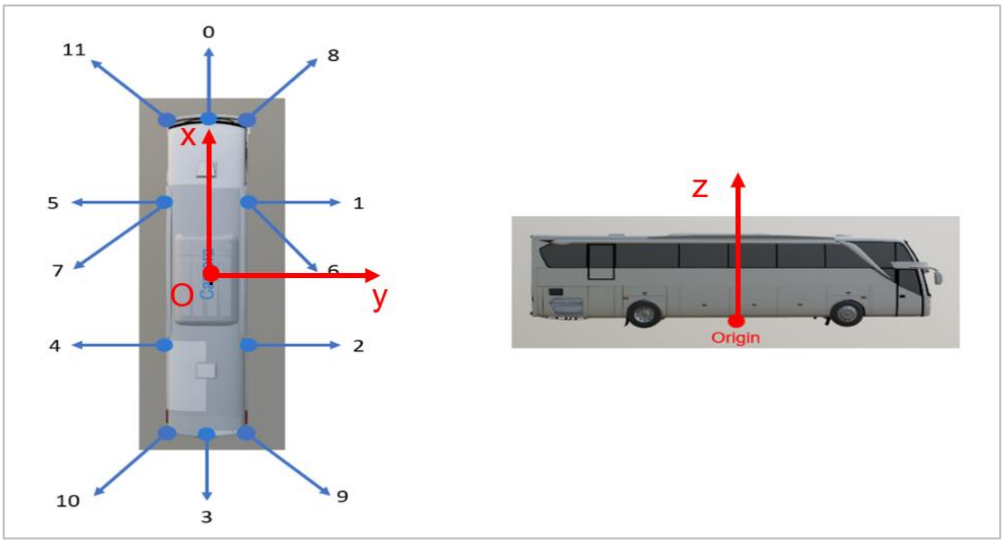
```
# Camera position (for precise vehicle edges)
if(six_cam):
        top_id    = 0
        right_id  = 1
        bottom_id = 3
        left_id   = 5
else:
        top_id    = 0
        right_id  = 1
        bottom_id = 4
        left_id   = 7
```

The chosen camera for reference in different vehicle models

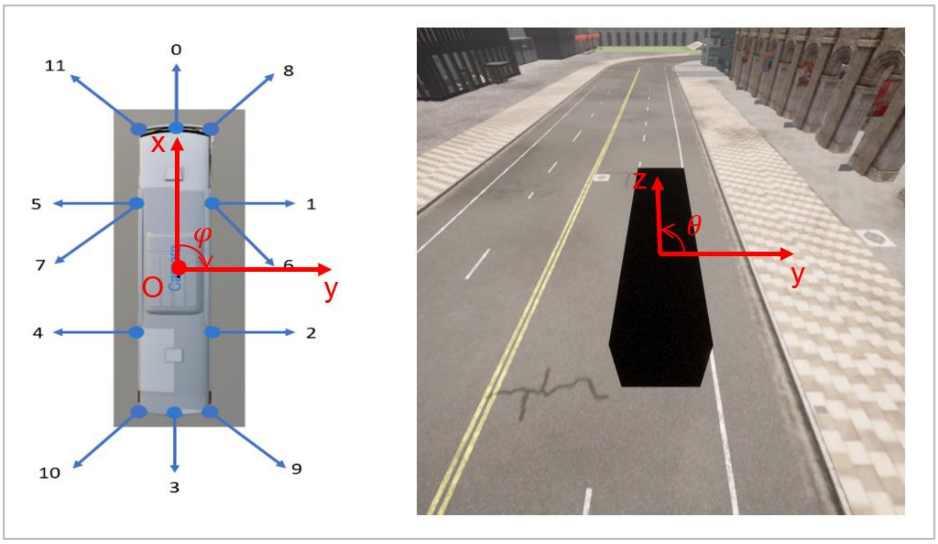
C. Coordinate system in json file

```
"camera_list": [
    {
        "fov": 180,
        "position": [
           x 5.17,
           y 0,
           z 2.695
        ],
        "rotation": [
            0,
           θ -45,
           φ 0
        ]
    },
```

Example: camera 0



Position coordinate axes



Rotation coordinate axes