# Term Project:Real-Time Texture Mapping of an Industrial Equipment

Zheng-Lin Wu[a]

[a]*Department of Mechanical Engineering, National Taiwan University, Taipei, Taiwan*

**Abstract**

In this project, I organize a process by using image segmentation model to reconstruct industrial equipment in a virtual environment, this innovative approach allows us to create virtual representations of equipment with textures derived from the real world. While the technology is still progressing towards real-time mapping, it presents a promising solution for accurate texture reconstruction. The significance of this project extends beyond not only visual fidelity, but also offers substantial benefits in terms of environmental sustainability[1], operational efficiency[2], and logistical simplicity.

*Keywords:* Texture Mapping, Real-Time Rendering, Salient Object Detection, 3D Visualization, Virtual Reality

## 1. Introduction

### *1.1. Motivation and Relevant Works*

According to an informal field research from a Taiwanese equipment vendor, gaining attention from various nations requires participating in more exhibitions to showcase their equipment and techniques. However, the size of the equipment significantly affects shipping costs. In [3], the authors reviewed how the transportation mode selection influence the shipment cost. Start-up companies, which often need to promote their innovations, typically cannot afford such high shipment fees. Consequently, they are exploring alternative methods to enable potential customers to

*Email address:* R12522636@ntu.edu.tw (Zheng-Lin Wu)

interact with real equipment components. Even if it's just a small part of the process, allowing customers to engage with actual equipment can significantly boost their purchasing intentions. Additionally, finding ways to attract international customers to Taiwan could be a beneficial strategy.

In [4], which is the most relevant to this project, the authors focus on the application of virtual reality to assist CNC machine tool demonstrations and exhibitions. This system was presented at the 2017 Taipei International Machine Tool Show (TIMTOS 2017). In [5], the authors explore the emerging trends and preliminary practices in digital design within the field of virtual exhibitions. In [6], the authors propose an approach to reconstruct industrial heritage in Spain using Virtual Reality. In [7], the authors present a methodology that aids programmers in developing virtual and augmented reality systems. Finally, in [8], the authors propose three different model creation methods to facilitate data transfer between CAD models and virtual environments.

### *1.2. Contribution and Scope*

The primary focus of this project is to facilitate real-time video streaming of equipment, specifically targeting the projection of its actuating parts. This is achieved through the use of an image segmentation model, which leverages deep learning neural network techniques for semantic segmentation to identify and extract specific regions of interest (ROI). This approach contrasts significantly with the current state-of-the-art (SOTA) solutions[9], which typically involve deploying CAD models into virtual environments. As highlighted in the introduction, there is a preference among customers for viewing real equipment rather than virtual representations, given that CAD models often appear less realistic.

Moreover, traditional 3D reconstruction methods, such as point clouds, meshes, and voxels, are computationally expensive. For further reading on this topic, refer to the relevant research in [10], [11], and [12]. In contrast, this project introduces a compromise solution that addresses these issues effectively. This novel approach not only enhances the visual authenticity of the equipment representation but also reduces the computational overhead associated with 3D model processing.

Thus, the major contribution of this paper lies in presenting a cost-effective, realistic, and computationally feasible method for equipment visualization in real-time applications. This research extends the scope of image segmentation applications to practical industrial equipment visualization, providing a more accessible and efficient solution compared to conventional methods.

## 2. Datasets

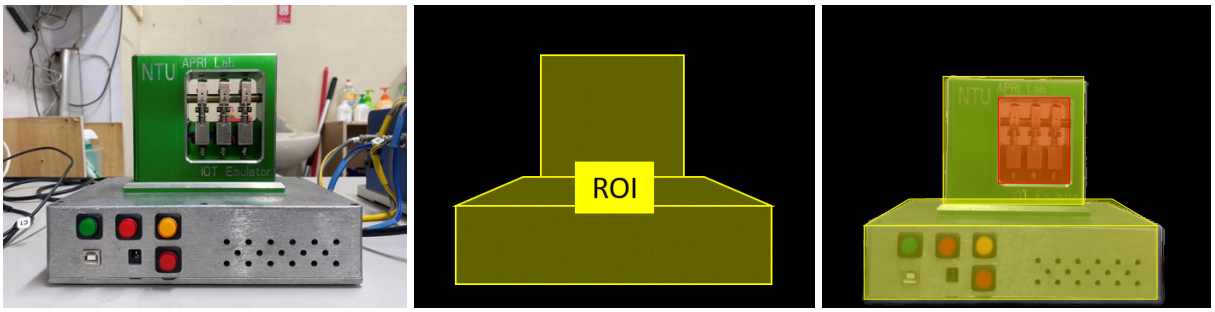
At the outset of this project, I aimed to develop an image segmentation model from scratch, focusing on an emulator constructed by graduate alumni of my laboratory. The initial plan was to use masking data from the emulator as training data. However, after discussing with my advisor, we concluded that annotating a large volume of training data for a generic piece of equipment was not an efficient approach. Consequently, I explored existing general models that could be adapted for our project. This search led me to discover "rembg," a package containing several pre-trained image segmentation models. After some experimentation, I decided to utilize U²-Net for the segmentation tasks.

The original research paper [13] for U²-Net indicates that it utilizes the DUTS-TR dataset [14] for pre-training. DUTS-TR consists of 10,553 images, making it one of the largest available datasets for salient object detection. Each image is accompanied by high-quality, pixel-level annotations that clearly delineate salient objects from the background. These images, sourced from a variety of environments, embody a wide spectrum of scenarios, complexities, and backgrounds. Such diversity is crucial for training models that are robust and generalizable across different settings.

In its training phase, U²-Net leverages this extensive dataset to learn detailed features of salient objects. The comprehensive annotations enable the model to accurately learn the boundaries and distinctions between salient and non-salient regions. Utilizing the DUTS-TR dataset ensures that U²-Net is not only effective on standard benchmark datasets but also performs well in practical applications where precise detection and segmentation of salient objects are essential.

## 3. Methodology

In the initial phase of the methodology, I defined the regions of interest (ROI) for the emulator as shown in Figure 1, which are categorized into stationary and actuating parts.




(a) (b) (c)

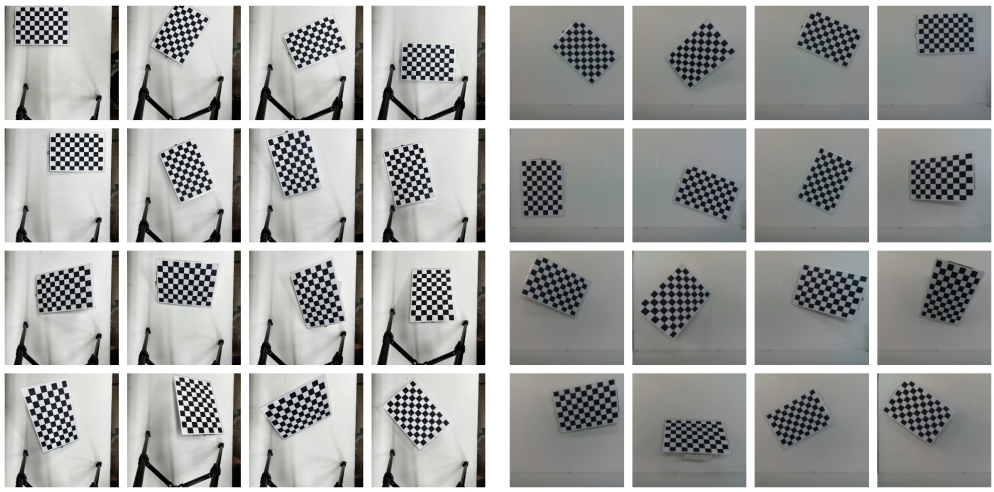
Figure 1: Illustration of the emulator, constructed by graduate alumni: (a) Front view of the emulator, displaying the project's target object and the background that needs to be removed. (b) The ROI masking section of the emulator. (c) The yellow region indicates the stationary part, and the red region highlights the actuating part.

To minimize image distortion during post-processing, I conducted camera calibration. This approach is based on methods detailed in the following references: [15], [16], and [17]. For this purpose, I used an iPhone 13 to capture pre-mapping images due to its high-resolution capabilities. For real-time texture mapping, I selected the RealSense D435i camera for its stability in image processing. For the calibration process, as shown in Figure 2. I used a total of 17 checkerboard images, each featuring different transformations and rotations. These were performed using a 10x7 checkerboard. The projection errors recorded were 0.162 for the iPhone 13 and 0.032 for the RealSense D435i, as shown in Figure 3 are the results for camera calibration.

For the stationary parts of the emulator, I applied pre-texture mapping using Blender. This approach was suitable as these parts do not change during the process. For the actuating parts, I utilized the U²-Net segmentation model[13] to generate masks that isolate and remove the background of the emulator, which could be considered noise in this context. Subsequently, I transformed the black background into a transparent property.

Finally, I employed Unity to create a virtual environment for practical applications. This en-

vironment is designed to process and display the post-processed videos generated by a Python program, effectively integrating the segmented and mapped textures into a simulated real-world setting.



(a) (b)

Figure 2: Illustration of the camera calibration process, these figures show a selection of the images used: (a) Checkerboard images used for iPhone 13 calibration. (b) Checkerboard images used for RealSense D435i calibration.

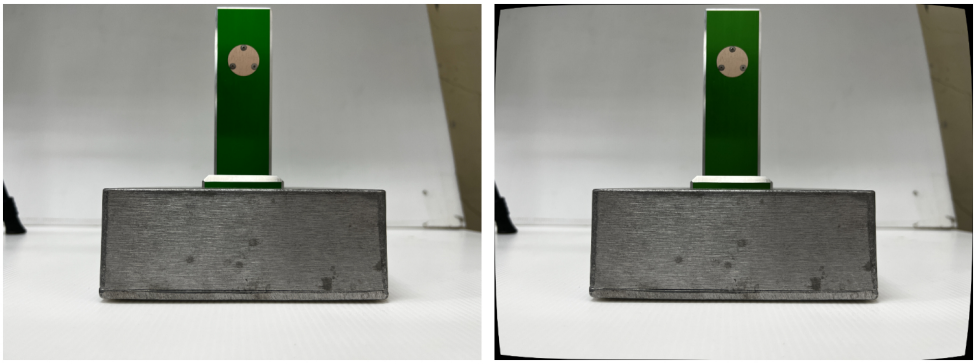


Figure 3: The left image shows the original capture from the camera, while the right image displays the result after applying camera calibration parameters.

## 4. Results and Discussions

As discussed in the methodology section, I performed pre-texture mapping on the stationary parts of the emulator using Blender. As depicted in Figure 4, it is observable that the emulator

retains a metallic texture. In contrast, the actuating part was left uncolored, resulting in a white, blank texture.

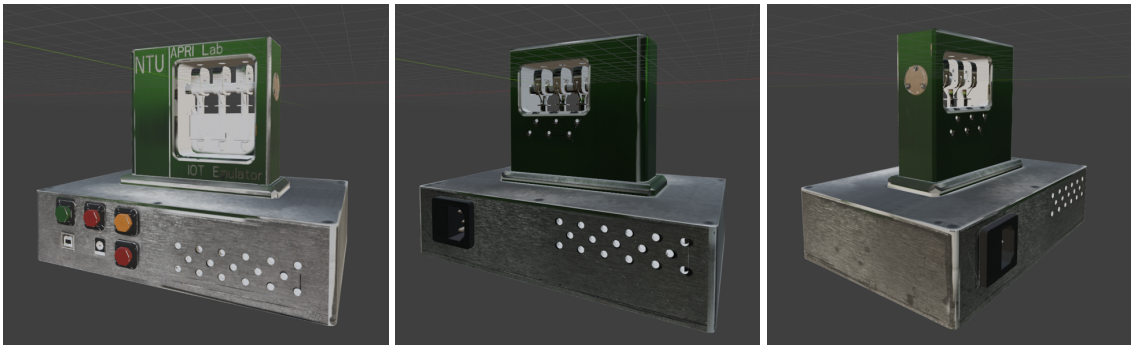
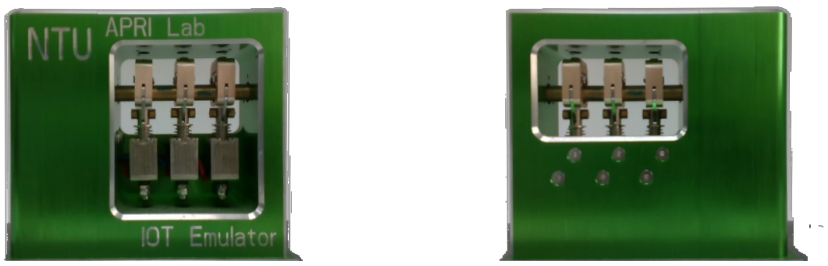



Figure 4: Illustration of the output results from Blender.

For the actuating parts, I utilized the rembg package in Python, which supports multiple image segmentation models. The results are displayed in Figure 5. Although some noise is still visible around the contours of the ROI, the majority of the emulator's region demonstrates this method is an efficient way to remove the background.




(a) (b)

Figure 5: Illustration of the output results from python program : (a) Front View of the emulator apply rembg result. (b) Back View of the emulator apply rembg result.

The next step involved importing the pre-textured emulator into Unity to facilitate real-time streaming of the actuating part. As illustrated in Figure 6, I utilized a white blank board to display the video stream output from the Python program.

Ultimately, the video streamed from the Python program allowed us to observe the results, which are shown in Figure 7.

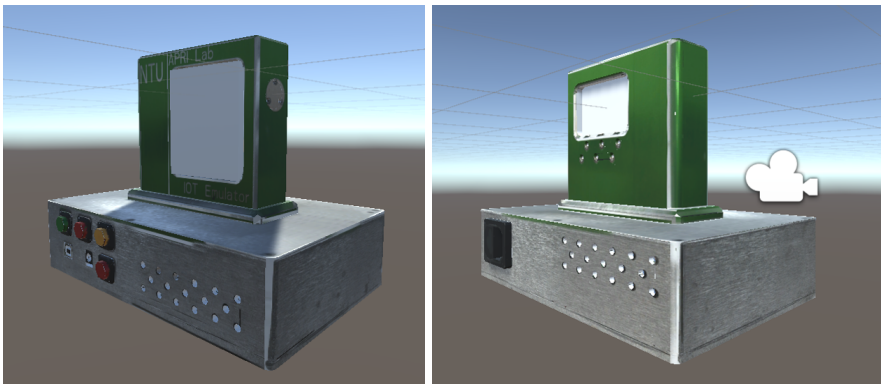



Figure 6: Illustration of the deploy results in Unity.

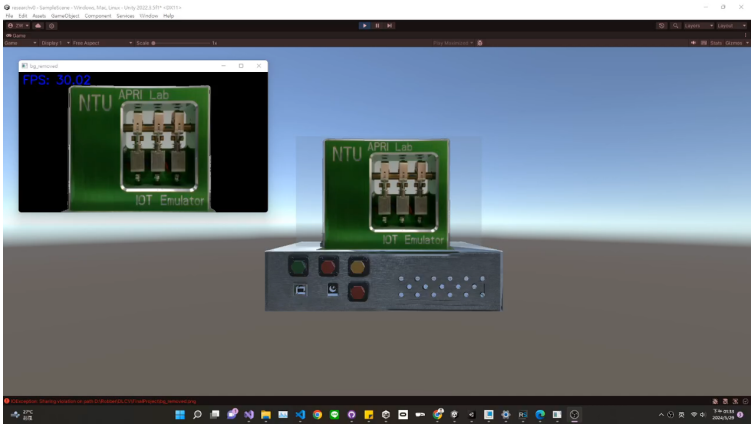



Figure 7: Illustration of the output results from python program and the deploy results in Unity.

## 5. Conclusion

In this project, I introduced a novel real-time texture mapping solution. The primary method involves defining the region of interest (ROI) of the target object, conducting pre-texture mapping on the stationary parts, and employing an image segmentation model for real-time streaming of the actuating parts. Although the results from the source program are beneficial, deploying them in the virtual environment introduces a significant delay. In the future, I plan to explore more efficient data transfer solutions to enhance deployment in virtual environments.

Additionally, I aim to integrate the concept of the Internet of Things (IoT) into this project. This integration is intended to transform the project into a more generic solution capable of addressing the needs of a wide range of industrial equipment in practical scenarios.

## References


[1] Askiner Gungor and Surendra M Gupta. Issues in environmentally conscious manufacturing and product recovery: a survey. Computers Industrial Engineering, 36(4):811–853, 1999.

[2] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. European Journal of Operational Research, 187(3):985–1032, 2008.

[3] Erna Engebrethsen and Stéphane Dauzère-Pérès. Transportation mode selection in inventory models: A literature review. European Journal of Operational Research, 279(1):1–25, 2019.

[4] Kao, Yung-Chou, Lee, Chung-Shuo, Liu, Zhi-Ren, and Lin, Yu-Fu. Case study of virtual reality in cnc machine tool exhibition. MATEC Web Conf., 123:00004, 2017.

[5] Fan Chen. Virtual exhibition hall based on blender and u3d: "cloud weaving dream" chu embroidery virtual exhibition hall design. Highlights in Art and Design, 4(2), 2023.

[6] David Checa, Mario Alaguero, and Andres Bustillo. Industrial heritage seen through the lens of a virtual reality experience. In Lucio Tommaso De Paolis, Patrick Bourdot, and Antonio Mongelli, editors, Augmented Reality, Virtual Reality, and Computer Graphics, pages 116–130, Cham, 2017. Springer International Publishing.

[7] I. Calvo, F. López, E. Zulueta, et al. Towards a methodology to build virtual reality manufacturing systems based on free open software technologies. International Journal on Interactive Design and Manufacturing (IJIDeM), 11(3):569–580, 2017.

[8] J Whyte, N Bouchlaghem, A Thorpe, and R McCaffer. From cad to virtual reality: modelling approaches, data exchange and interactive 3d building design tools. Automation in Construction, 10(1):43–55, 2000.

[9] J.E. Naranjo, D.G. Sanchez, A. Robalino-Lopez, P. Robalino-Lopez, A. Alarcon-Ortiz, and M.V. Garcia. A scoping review on virtual reality-based industrial training. Appl. Sci., 10(22):8224, 2020.

[10] Yusheng Xu, Xiaohua Tong, and Uwe Stilla. Voxel-based representation of 3d point clouds: Methods, applications, and its potential use in the construction industry. Automation in Construction, 126:103675, 2021.

[11] P. Hübner, M. Weinmann, and S. Wursthorn. Voxel-based indoor reconstruction from hololens triangle meshes, 2020.

[12] Francis Engelmann, Konstantinos Rematas, Bastian Leibe, and Vittorio Ferrari. From points to multi-object 3d reconstruction, 2020.

[13] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R. Zaiane, and Martin Jagersand. U2-net: Going deeper with nested u-structure for salient object detection. Pattern Recognition, 106:107404, 2020.

[14] Lijun Wang, Huchuan Lu, Yifan Wang, Mengyang Feng, Dong Wang, Baocai Yin, and Xiang Ruan. Learning to detect salient objects with image-level supervision. In CVPR, 2017.

[15] Z. Zhang. A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330–1334, 2000.

[16] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. IEEE Journal on Robotics and Automation, 3(4):323–344, 1987.

[17] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 1106–1112, 1997.

# main program

In [ ]:
```
from rembg import remove
import pyrealsense2 as rs
import numpy as np
import pickle
import time
import cv2
```

In [ ]:
```
# Load camera calibration data
with open('cameraCalibration_srcdata/calibration_data_RSd435i.pkl', 'rb') as f:
    calibration_data = pickle.load(f)
    cameraMatrix = calibration_data["cameraMatrix"]
    dist = calibration_data["dist"]

# Initialize the camera and the background removal model
pipe = rs.pipeline()
cfg = rs.config()

cfg.enable_stream(rs.stream.color, 640, 360, rs.format.bgr8, 30)
cfg.enable_stream(rs.stream.depth, 640, 360, rs.format.z16, 30)

pipe.start(cfg)

stop_display = False

# Initialize: capture the initial image and use it to generate a mask for backgr
init_frame = pipe.wait_for_frames()
init_color_frame = init_frame.get_color_frame()
if init_color_frame:
    init_color_image = np.asanyarray(init_color_frame.get_data())
    init_bg_removed = remove(
        init_color_image,
        alpha_matting=True,
        alpha_matting_foreground_threshold=240,
        alpha_matting_background_threshold=5,
        alpha_matting_erode_size=15,
        alpha_matting_base_size=1000,
    )
    # Extract the Alpha channel from the RGBA image as the mask
    if init_bg_removed.shape[2] == 4:
        init_mask = init_bg_removed[:, :, 3]
        init_mask = cv2.cvtColor(init_mask, cv2.COLOR_GRAY2BGR)
        init_mask = init_mask.astype(np.uint8)

# Precompute the undistortion map
h, w = init_color_image.shape[:2]
newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, dist, (w, h),
map1, map2 = cv2.initUndistortRectifyMap(cameraMatrix, dist, None, newCameraMatr

# Initialize variables for FPS calculation
fps = 0
frame_count = 0
start_time = time.time()

while not stop_display:
    frame = pipe.wait_for_frames()
    color_frame = frame.get_color_frame()
    depth_frame = frame.get_depth_frame()
```

```
    if not color_frame:
        continue
    color_image = np.asanyarray(color_frame.get_data())
    depth_image = np.asanyarray(depth_frame.get_data())

    bg_removed = cv2.bitwise_and(color_image, color_image, mask=init_mask[:, :,

    depth_cm = cv2.applyColorMap(cv2.convertScaleAbs(depth_image, alpha=0.5), cv

    alpha = np.sum(bg_removed, axis=-1) > 0
    alpha = np.uint8(alpha * 255)
    bg_removed = np.dstack((bg_removed, alpha))

    h, w = bg_removed.shape[:2]
    newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, dist, (w,
    bg_removed = cv2.undistort(bg_removed, cameraMatrix, dist, None, newCameraMa

    # Calculate and display FPS
    frame_count += 1
    elapsed_time = time.time() - start_time
    if elapsed_time > 1:
        fps = frame_count / elapsed_time
        frame_count = 0
        start_time = time.time()

    # Display FPS on the image
    cv2.putText(bg_removed, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLE

    cv2.imshow('bg_removed', bg_removed)

    saved = False
    if not saved:
        saved = True
        cv2.imwrite('bg_removed2.png', bg_removed)

    key = cv2.waitKey(1)
    if key == ord('q'):
        stop_display = True

pipe.stop()
cv2.destroyAllWindows()
```

# camera calibration program

In [ ]:
```
import numpy as np
import cv2 as cv
import glob
import pickle
```

In [ ]:
```
################ FIND CHESSBOARD CORNERS - OBJECT POINTS AND IMAGE POINTS ######
chessboardSize = (10,7)
frameSize = (1280,720) # realsense d435i
# frameSize = (3024,4032) # iphone 13

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)

size_of_chessboard_squares_mm = 20
objp = objp * size_of_chessboard_squares_mm

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

# for RSd435i camera
images = glob.glob('cameraCalibration_srcdata/RSd435i/*.png')
# for iphone 13 camera
# images = glob.glob('cameraCalibration_srcdata/iphone13/*.png')
print(images)

for image in images:
    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)
    # If found, add object points, image points (after refining them)
    if ret == True:

        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)

        # Draw and display the corners
        cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
        img = cv.resize(img, (720, 1080))
        cv.imshow('img', img)
        cv.waitKey(1000)

cv.destroyAllWindows()

############## CALIBRATION #######################################################

ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,

# Save the camera calibration result
calibration_data = {
    "cameraMatrix": cameraMatrix,
    "dist": dist,
    "rvecs": rvecs,
```

```
    "tvecs": tvecs
}

with open('cameraCalibration_srcdata/calibration_data_RSd435i.pkl', 'wb') as f:
    pickle.dump(calibration_data, f)

print("Calibration parameters saved successfully.")
```

```
['cameraCalibration_srcdata/RSd435i\\10_Color.png', 'cameraCalibration_srcdata/RS
d435i\\11_Color.png', 'cameraCalibration_srcdata/RSd435i\\12_Color.png', 'cameraC
alibration_srcdata/RSd435i\\13_Color.png', 'cameraCalibration_srcdata/RSd435i\\14
_Color.png', 'cameraCalibration_srcdata/RSd435i\\15_Color.png', 'cameraCalibratio
n_srcdata/RSd435i\\16_Color.png', 'cameraCalibration_srcdata/RSd435i\\17_Color.pn
g', 'cameraCalibration_srcdata/RSd435i\\1_Color.png', 'cameraCalibration_srcdata/
RSd435i\\2_Color.png', 'cameraCalibration_srcdata/RSd435i\\3_Color.png', 'cameraC
alibration_srcdata/RSd435i\\4_Color.png', 'cameraCalibration_srcdata/RSd435i\\5_C
olor.png', 'cameraCalibration_srcdata/RSd435i\\6_Color.png', 'cameraCalibration_s
rcdata/RSd435i\\7_Color.png', 'cameraCalibration_srcdata/RSd435i\\8_Color.png',
'cameraCalibration_srcdata/RSd435i\\9_Color.png']
Calibration parameters saved successfully.
```

In [ ]:
```
# Reprojection Error
mean_error = 0

for i in range(len(objpoints)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], cameraMat
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
    mean_error += error

print( "total error: {}".format(mean_error/len(objpoints)) )
```

```
total error: 0.03223261797395963
```

In [ ]:
```
# read the calibration data
with open('cameraCalibration_srcdata/calibration_data_RSd435i.pkl', 'rb') as f:
    calibration_data = pickle.load(f)
    cameraMatrix = calibration_data["cameraMatrix"]
    dist = calibration_data["dist"]
    rvecs = calibration_data["rvecs"]
    tvecs = calibration_data["tvecs"]

# read the image
img = cv.imread('cameraCalibration_srcdata/distorted/RSd435i/bg_removed.png')
h, w = img.shape[:2]

newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w, h),

# Undistort
dst = cv.undistort(img, cameraMatrix, dist, None, newCameraMatrix)

# crop the image
x, y, w, h = roi
dst_crop = dst[y:y+h, x:x+w]

cv.imwrite('cameraCalibration_srcdata/undistorted/RSd435i/bg_removed.jpg', dst)
cv.imwrite('cameraCalibration_srcdata/undistorted/RSd435i/bg_removed_crop.jpg',
print("Undistorted image saved successfully.")
```

```
Undistorted image saved successfully.
```