

# 3D Vision Using Stereo Camera

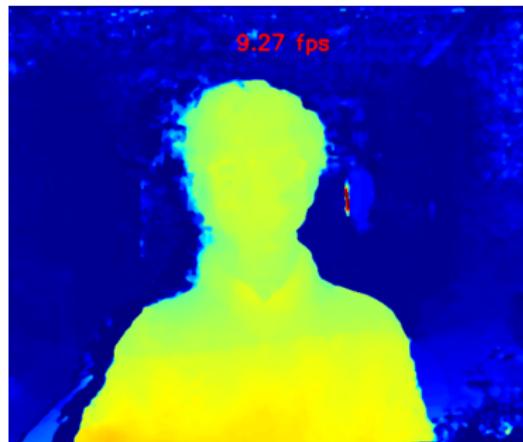
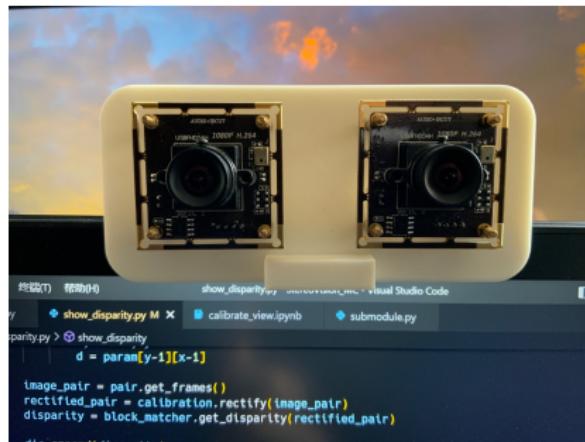
## Project for course Media & Cognition

Jida Zhang, Ruiming Zhang, Mingli Liu

Dept. of EE, Tsinghua University

January, 2022

# Overview



code: [https://github.com/GiddaZhang/StereoVision\\_MC.git](https://github.com/GiddaZhang/StereoVision_MC.git)

# Outline

① Frame design

② Calibration

③ Depth algorithm

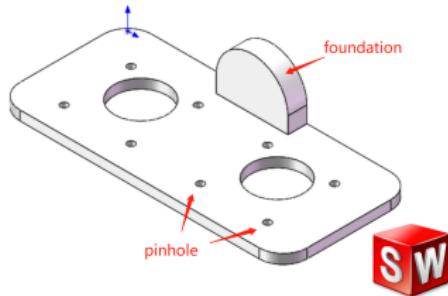
# Structure & Parameter

## Structure

plane structure with pinholes and foundation

## Parameter

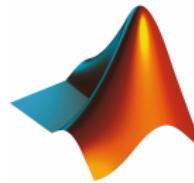
- (width, height, thickness) = (120, 55, 5)mm
- baseline = 57mm (designed value)



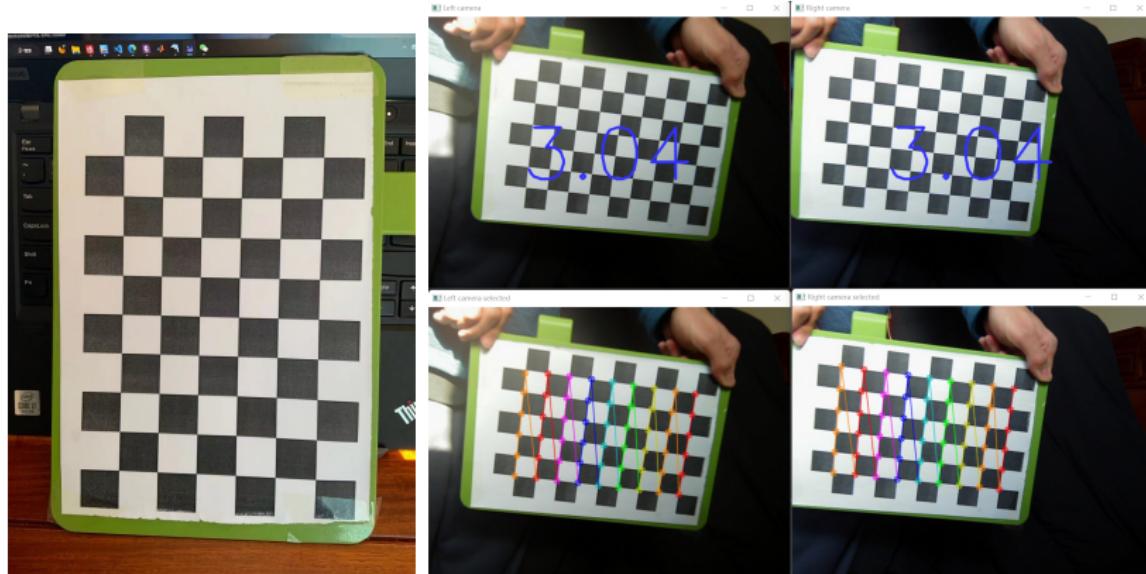
# Calibration Procedure

steps

- chessboard preparation & capture  
[ref] <https://github.com/erget/StereoVision>
- calculate camera parameters with Matlab  
[ref] [https://blog.csdn.net/weixin\\_43956351/article/details/94394892](https://blog.csdn.net/weixin_43956351/article/details/94394892)
- calculate & save rectify mapping with OpenCV  
[ref] <https://www.bilibili.com/video/BV1H34y157Q5>

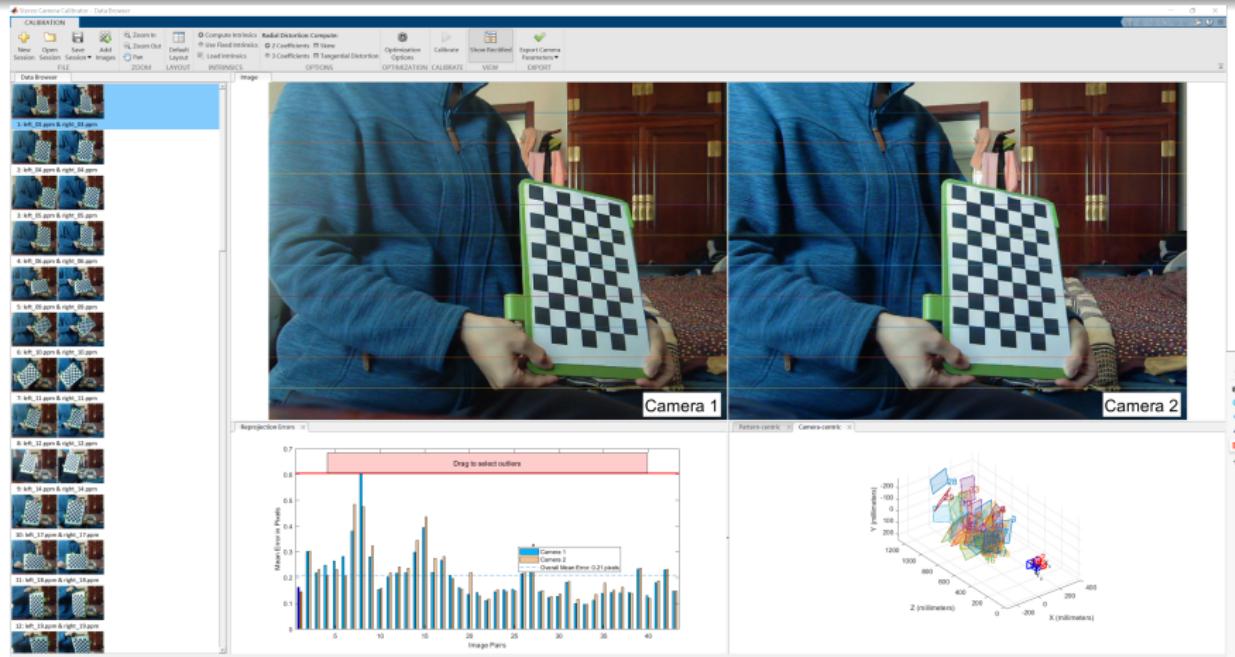


# Chessboard preparation & Capture



9\*6 chessboard & auto capture program screenshot

# Calculate camera parameters with Matlab



# Calculate & Save rectify mapping with OpenCV

Results from matlab (inner parameters)

intrinsic matrix, distortion coefficients, rotation matrix,  
translation vector, essential matrix & fundamental matrix.

$$\begin{bmatrix} 645.40 & 0 & 311.00 \\ 0 & 645.87 & 227.58 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 644.64 & 0 & 323.14 \\ 0 & 645.00 & 226.36 \\ 0 & 0 & 1 \end{bmatrix}.$$

intrinsic matrix<sub>L</sub>                           intrinsic matrix<sub>R</sub>

$$\begin{bmatrix} 1.0000 & -0.002 & 0.004 \\ 0.002 & 1.0000 & 0.006 \\ -0.004 & -0.006 & 1.0000 \end{bmatrix} \approx \mathbf{I}_3, \quad \begin{bmatrix} -56.4 \\ 0.35 \\ -0.11 \end{bmatrix}.$$

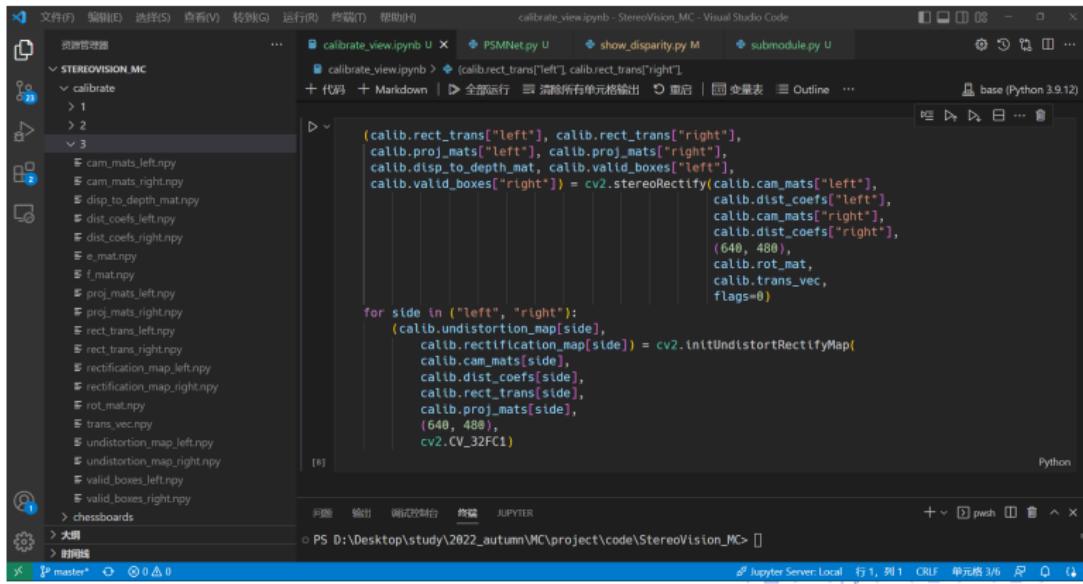
rotation matrix                                   transposition vector

⇒ focal length  $\approx$  645mm, baseline  $\approx$  56.4mm.

# Calculate & Save rectify mapping with OpenCV

## Results from opencv (for calculation)

rectification transform, projection matrix, undistortion mapping, rectification mapping & disparity to depth mapping...



The screenshot shows a Visual Studio Code interface with a Python file open. The file contains code for calculating rectification maps and depth maps from stereo camera parameters. The code uses OpenCV functions like `stereoRectify` and `initUndistortRectifyMap`.

```
(calib.rect_trans["left"], calib.rect_trans["right"],
 calib.proj_mats["left"], calib.proj_mats["right"],
 calib.dist_to_depth_mat, calib.valid_boxes["left"],
 calib.valid_boxes["right"]) = cv2.stereoRectify(calib.cam_mats["left"],
 calib.dist_coefs["left"],
 calib.cam_mats["right"],
 calib.dist_coefs["right"],
 (640, 480),
 calib.rot_mat,
 calib.trans_vec,
 flags=0)

for side in ("left", "right"):
    (calib.undistortion_map[side],
     calib.rectification_map[side]) = cv2.initUndistortRectifyMap(
         calib.cam_mats[side],
         calib.dist_coefs[side],
         calib.rect_trans[side],
         calib.proj_mats[side],
         (640, 480),
         cv2.CV_32FC1)
```

# Depth algorithm

## Algorithm procedure

At time- $t$  with images from two “eyes”  $L_t$  and  $R_t$  as input:

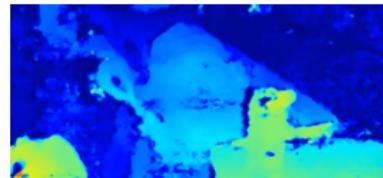
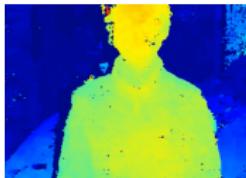
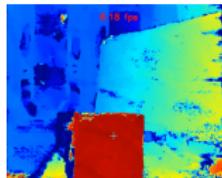
- apply rectify mapping to get rectified pair  $\tilde{L}_t, \tilde{R}_t = \mathcal{R}(L_t, R_t);$
- calculate disparity map via semi-global block matching (SGBM):  $D_t = \text{SGBM}(\tilde{L}_t, \tilde{R}_t);$
- apply filtering in spatial domain to  $D_t$ :  $\tilde{D}_t = \mathcal{F}(D_t);$
- apply a  $k$ -tap sequential average:  $\bar{D}_t = \frac{\tilde{D}_{t-k+1} + \dots + \tilde{D}_{t-1} + \tilde{D}_t}{k};$
- calculate depth and output:  $d_t = \frac{\text{baseline} \times \text{focal length}}{\bar{D}_t}.$

1. high frequency “noise” in both spatial and time domain;
2. cheaper in time & memory compared with learning based algorithm (tried, failed).

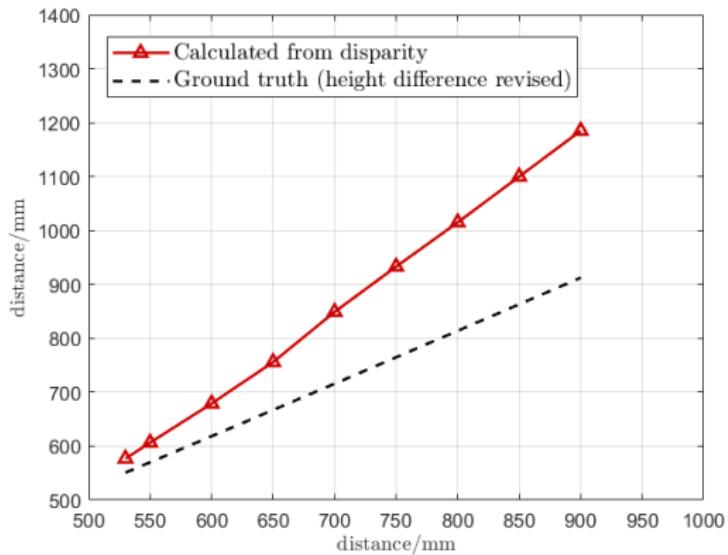
# Algorithm performance

## criterias and realization

- Speed:  $\sim 10$  fps,  
via non-learning algorithm and simple filterings;
- Quality: “rather” smooth edges, with “some” holes,  
via parameter adjustment of the SGBM algorithm  
(matched block size, number of disparities, etc.);
- Precision: See below;
- Robust: None.



# Precision



1. depth is the inverse of disparity, disparity is in pixel...
2. the error is also linear, maybe easy to rectify...