2. Astra camera calibration

Wiki: http://wiki.ros.org/camera_calibration

Official website link: https://orbbec3d.com/develop/

Astra camera: https://github.com/orbbec/ros_astra_camera

Developer community: https://developer.orbbec.com.cn/download.html?id=53

Due to some internal and external reasons of the camera, the image will be greatly distorted, mainly radial deformation and tangential deformation, causing the straight line to become curved. The farther the pixel is from the center of the image, the more serious the distortion will be. In order to avoid errors caused by data sources, the parameters of the camera need to be calibrated. Calibration essentially uses a known and determined spatial relationship (calibration plate) to reversely deduce the inherent and real parameters of the camera (internal parameters) by analyzing the pixels of the photographed pictures.

Disadvantages of infrared depth camera ranging:

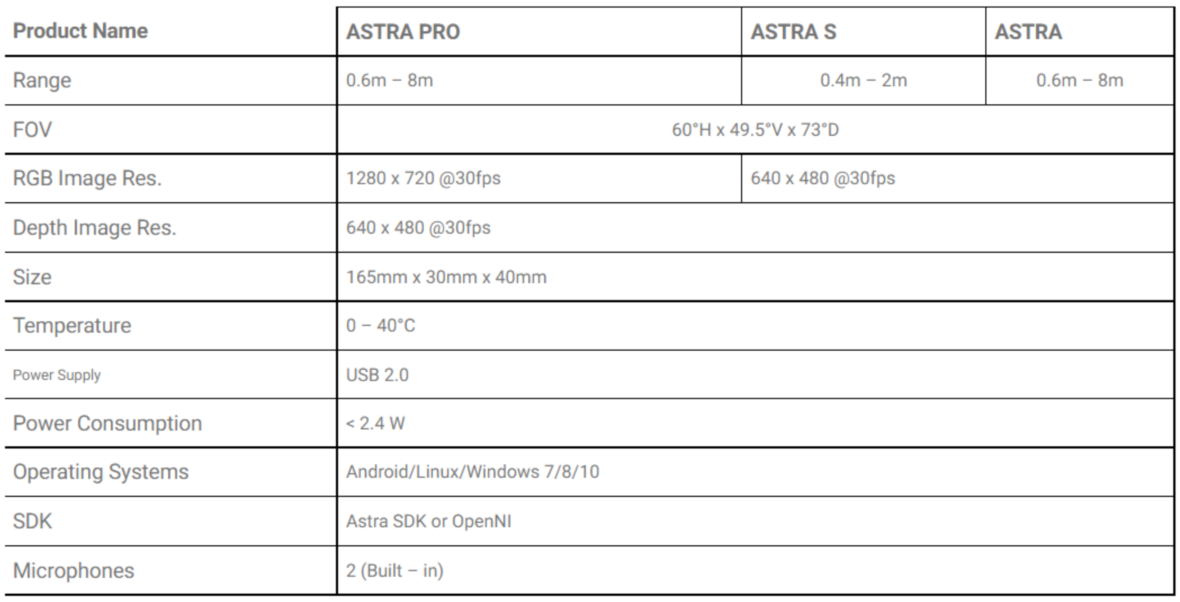
(1) It is impossible to accurately measure the distance of black objects because black substances can absorb infrared rays and the infrared rays cannot return, so the distance cannot be measured.

(2) It is impossible to accurately measure the distance of specular objects, because only when the depth camera is on the center vertical line of the specular object, the receiver can receive the reflected infrared rays, which will lead to overexposure.

(3) It is impossible to accurately measure the distance of transparent objects because infrared rays can pass through transparent objects.

(4) Unable to accurately measure distances for objects that are too close. Principle briefly

Astra Series



Product Name	ASTRA PRO	ASTRA S	ASTRA
Range	0.6m – 8m	0.4m – 2m	0.6m – 8m
FOV	60°H x 49.5°V x 73°D		
RGB Image Res.	1280 x 720 @30fps	640 x 480 @30fps	
Depth Image Res.	640 x 480 @30fps		
Size	165mm x 30mm x 40mm		
Temperature	0 – 40°C		
Power Supply	USB 2.0		
Power Consumption	< 2.4 W		
Operating Systems	Android/Linux/Windows 7/8/10		
SDK	Astra SDK or OpenNI		
Microphones	2 (Built – in)		

2.1. Preparation before calibration

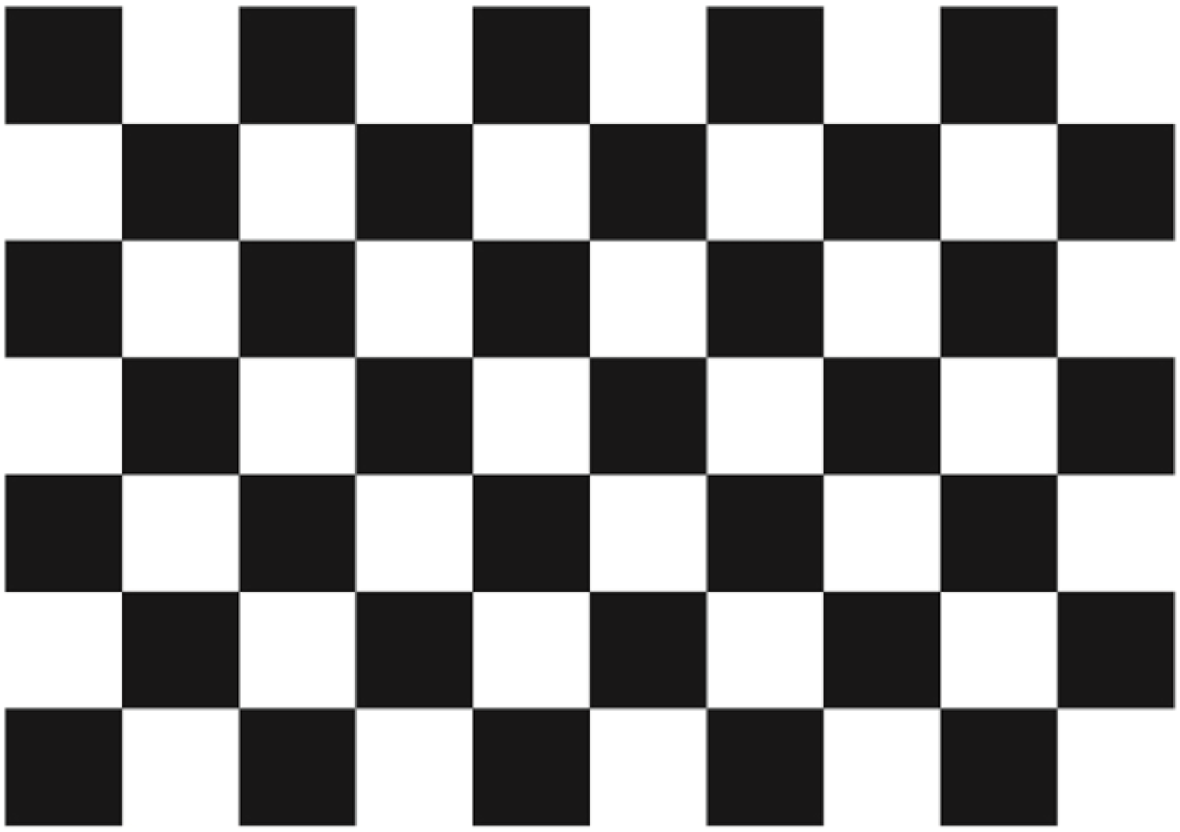
- A large [checkerboard] of known dimensions(http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration?action=AttachFile&do=view&target=check-108.pdf). This tutorial uses a 9x6 checkerboard and a 20mm square, which needs to be flattened during calibration.

  The calibration uses the internal vertices of the checkerboard, so a "10x7" checkerboard uses the internal vertex parameters "9x6", as shown in the example below.

  Any calibration board can be used, as long as the parameters are changed.

- An open area without obstacles and calibration board patterns
- Monocular camera for publishing images via ROS

Checkerboard (calibration board)



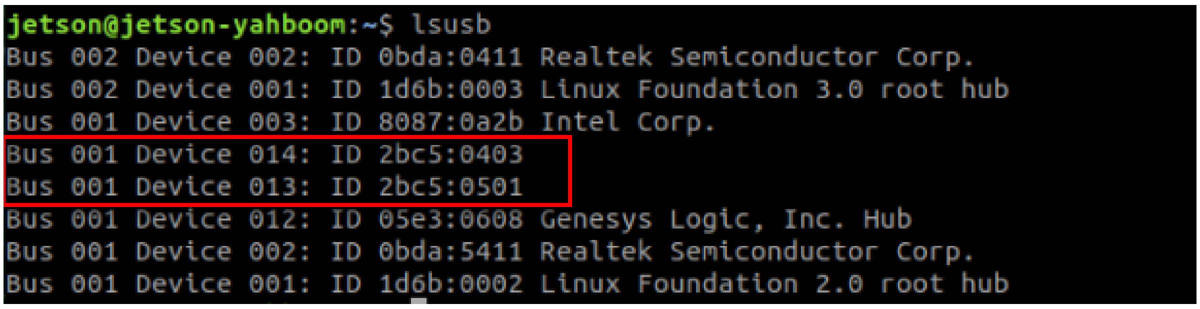
7×10 | Size: 20mm

Obi mid-range camera model and corresponding launch file

Launch file	Launch camera model
astra.launch	Astra, Astra S, Astra mini, Astra mini S
astraproplus.launch	Astra plus/Astraproplus
astrapro.launch	Astra pro
embedded_s.launch	Deeyea
dabai_u3.launch	Dabai
gemini.launch	Gemini

Device view

```
lsusb
```



```
jetson@jetson-yahboom:~$ lsusb
Bus 002 Device 002: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 8087:0a2b Intel Corp.
Bus 001 Device 014: ID 2bc5:0403
Bus 001 Device 013: ID 2bc5:0501
Bus 001 Device 012: ID 05e3:0608 Genesys Logic, Inc. Hub
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Depth camera ID: [2bc5:0403]

Color camera ID: [2bc5:0501]

The appearance of these two IDs indicates that the device is connected.

2.2. Astra calibration

Start the camera before calibration, and then turn off the camera until all calibrations are completed.

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

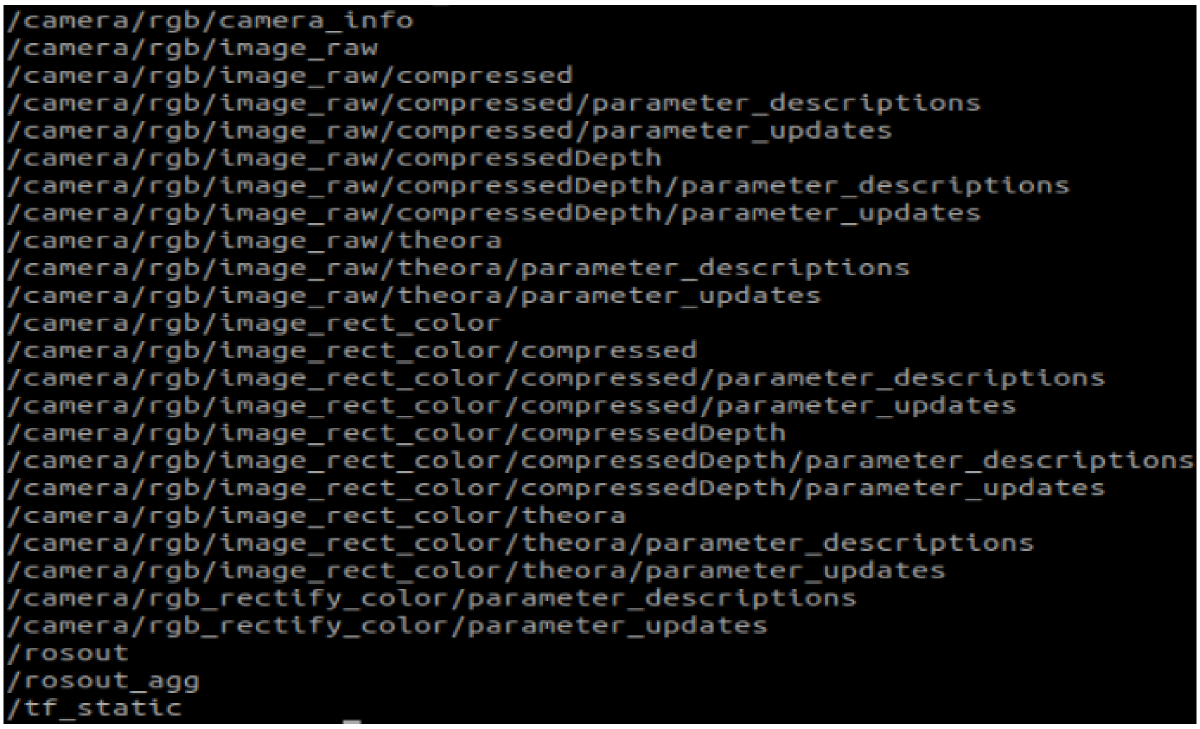
Launch Astra Camera

```
roslaunch yahboomcar_visual astra_calibration.launch
```

This startup command includes an IR image conversion node. The conversion is because the IR infrared camera views a 16-bit image during calibration, and the picture cannot be clearly seen. The 16-bit needs to be normalized into a value range of 0-255. 8-bit picture so that you can see it clearly.

View Image Topics

```
rostopic list
```

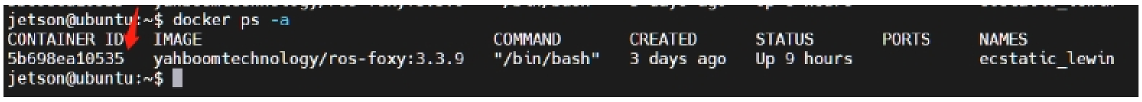


```
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/camera/rgb/image_rect_color
/camera/rgb/image_rect_color/compressed
/camera/rgb/image_rect_color/compressed/parameter_descriptions
/camera/rgb/image_rect_color/compressed/parameter_updates
/camera/rgb/image_rect_color/compressedDepth
/camera/rgb/image_rect_color/compressedDepth/parameter_descriptions
/camera/rgb/image_rect_color/compressedDepth/parameter_updates
/camera/rgb/image_rect_color/theora
/camera/rgb/image_rect_color/theora/parameter_descriptions
/camera/rgb/image_rect_color/theora/parameter_updates
/camera/rgb_rectify_color/parameter_descriptions
/camera/rgb_rectify_color/parameter_updates
/rosout
/rosout_agg
/tf_static
```

<PI5 needs to open another terminal and enter the same docker container

1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

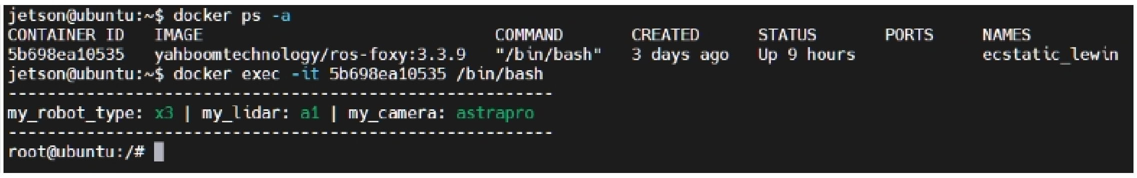
```
docker ps -a
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND       CREATED      STATUS       PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9    "/bin/bash"   3 days ago   Up 9 hours             ecstatic_lewin
jetson@ubuntu:~$
```

2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND       CREATED      STATUS       PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9    "/bin/bash"   3 days ago   Up 9 hours             ecstatic_lewin
jetson@ubuntu:~$ docker exec -it 5b698ea10535 /bin/bash
-------------------------------------------------------
my_robot_type: x3 | my_lidar: a1 | my_camera: astrapro
-------------------------------------------------------
root@ubuntu:/#
```

After successfully entering the container, you can open countless terminals to enter the container.

2.2.1. Color icon definition

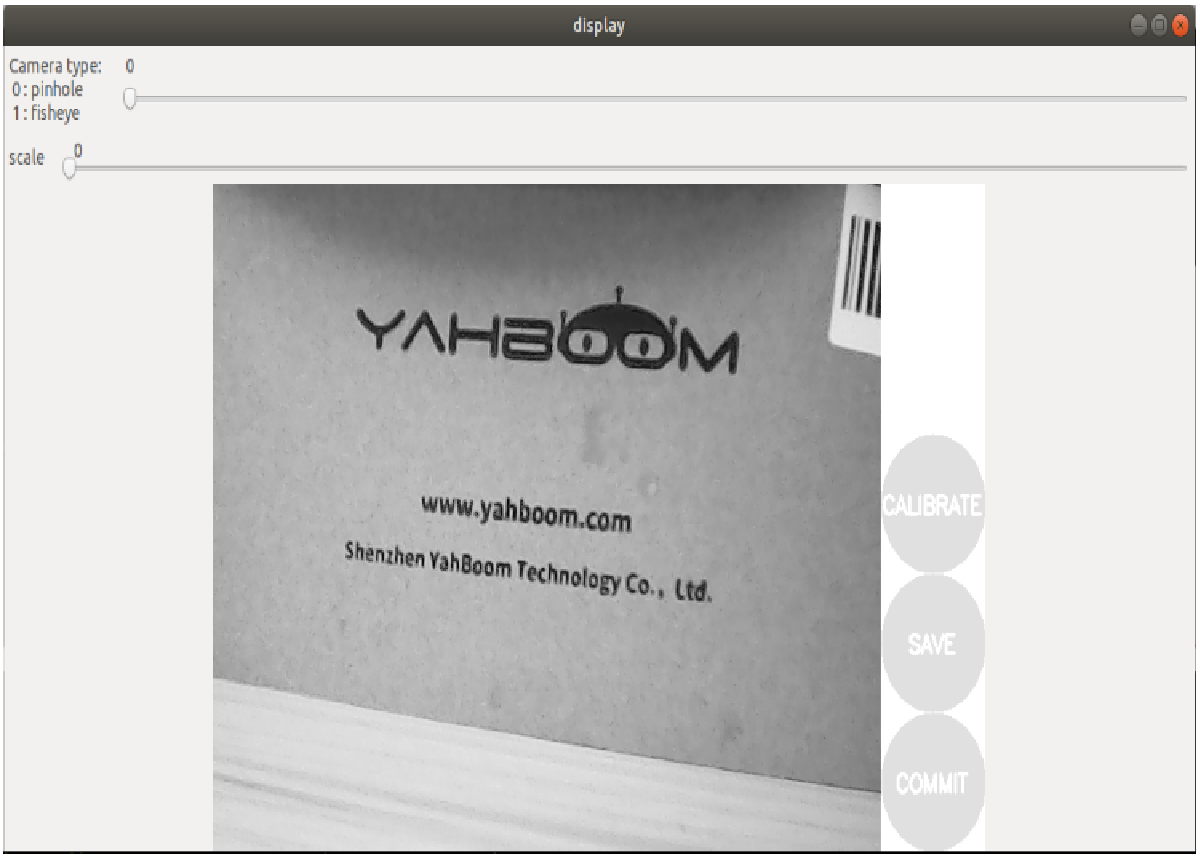
Start calibration node

```
rosrun camera_calibration cameracalibrator.py image:=/camera/rgb/image_raw
camera:=/camera/rgb --size 9x6 --square 0.02
```

size: Calibrate the number of internal corner points of the checkerboard, for example, 9X6, with a total of six rows and nine columns of corner points.

square: The side length of the checkerboard, in meters.

image and camera: Set the image topic published by the camera.



Calibration interface

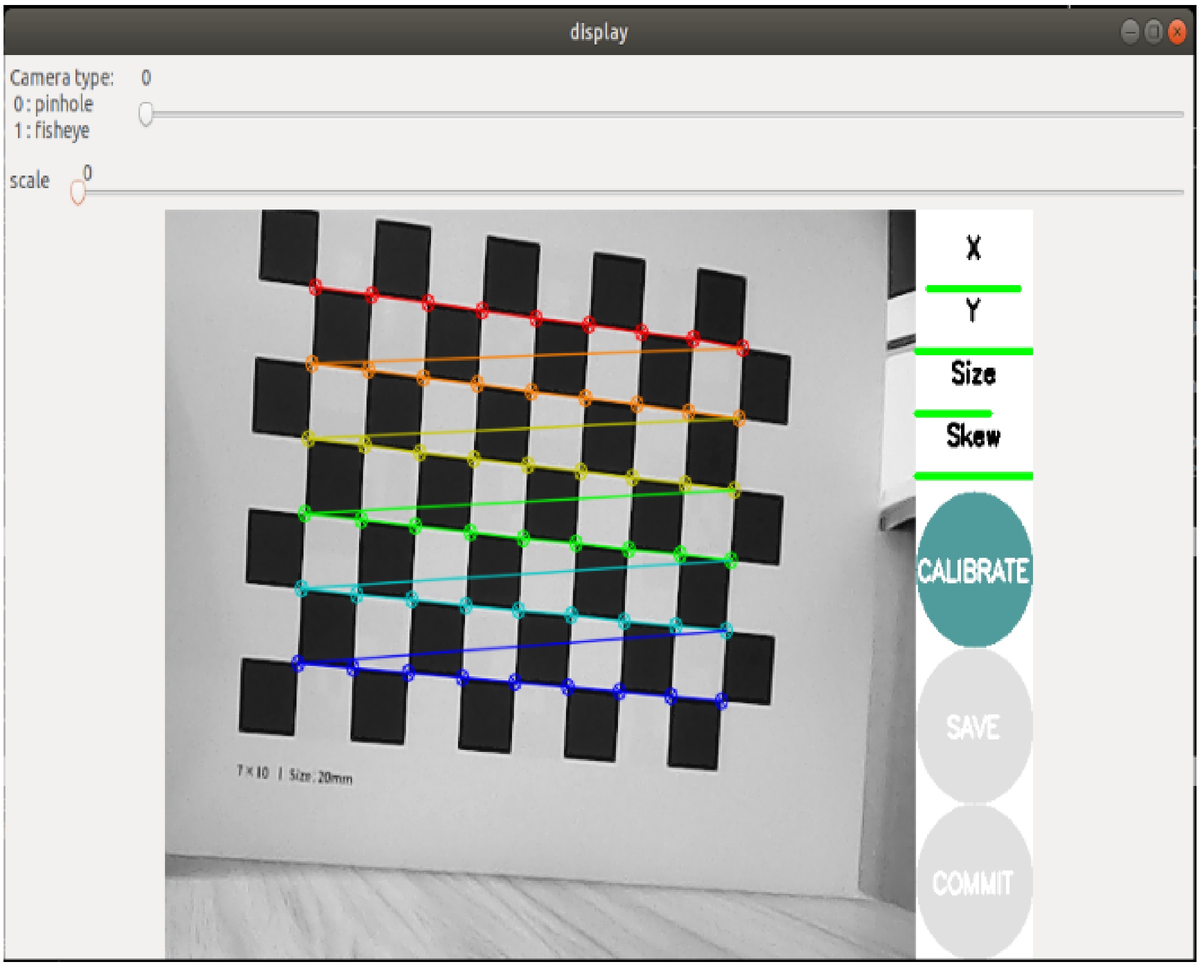
X: The left and right movement of the checkerboard in the camera field of view

Y: The checkerboard moves up and down in the camera field of view

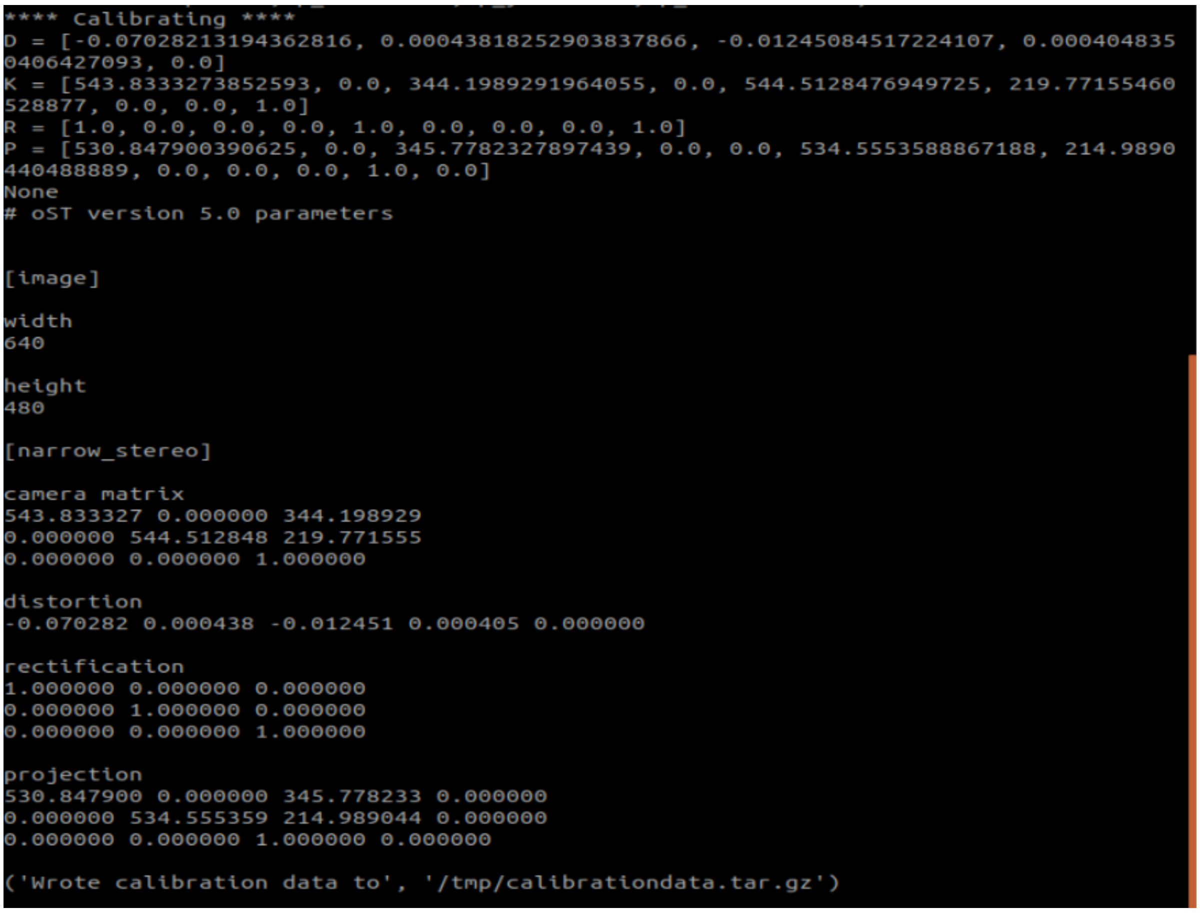
Size: the movement of the checkerboard back and forth in the camera field of view

Skew: The tilt and rotation of the checkerboard in the camera's field of view

After successful startup, place the checkerboard in the center of the screen and change to different positions. The system will identify it independently. The best situation is that the lines under [X], [Y], [Size], and [Skew] will first change from red to yellow and then to green as the data is collected, filling them as fully as possible.



- Click [CALIBRATE] to calculate the internal parameters of the camera. The more pictures you have, the longer it will take. Just wait. (Sixty or seventy is enough, too many can easily get stuck).
- Click [SAVE] to save the calibration results to [/tmp/calibrationdata.tar.gz].
- Click [COMMIT] to write the calibration file into the [.ros/camera_info/rgb_camera.yaml] file. The next time you start the camera, the calibration results will be automatically read.



```
**** Calibrating ****
D = [-0.07028213194362816, 0.00043818252903837866, -0.01245084517224107, 0.000404835
0406427093, 0.0]
K = [543.8333273852593, 0.0, 344.1989291964055, 0.0, 544.5128476949725, 219.77155460
528877, 0.0, 0.0, 1.0]
R = [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P = [530.847900390625, 0.0, 345.7782327897439, 0.0, 0.0, 534.5553588867188, 214.9890
440488889, 0.0, 0.0, 0.0, 1.0, 0.0]
None
# oST version 5.0 parameters


[image]

width
640

height
480

[narrow_stereo]

camera matrix
543.833327 0.000000 344.198929
0.000000 544.512848 219.771555
0.000000 0.000000 1.000000

distortion
-0.070282 0.000438 -0.012451 0.000405 0.000000

rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
530.847900 0.000000 345.778233 0.000000
0.000000 534.555359 214.989044 0.000000
0.000000 0.000000 1.000000 0.000000

('Wrote calibration data to', '/tmp/calibrationdata.tar.gz')
```

After the calibration is completed, you can move out the [/tmp/calibrationdata.tar.gz] file to see the content.

```
sudo mv /tmp/calibrationdata.tar.gz ~
```

After decompression, there are the image just calibrated, an ost.txt file and an ost.yaml file.

2.2.2, ir infrared calibration

After the data normalization problem is dealt with, another problem will arise. Because the RGBD camera, which uses structured light as the depth imaging principle, the infrared light projected by it is a special disordered spot, causing the infrared receiving device to be unable to receive clear and complete images. screen content.

At this time we can have several special processing methods:

- Forcibly find various angles and let the camera find the corners as much as possible (poor accuracy)
- Spread the infrared light spots evenly by pasting some frosted translucent paper in front of the red hair emitter (moderate accuracy, more convenient)
- Block the infrared projection camera and use an external infrared camera to fill in the light (high accuracy, additional equipment is required)

Choose the processing method according to your needs.

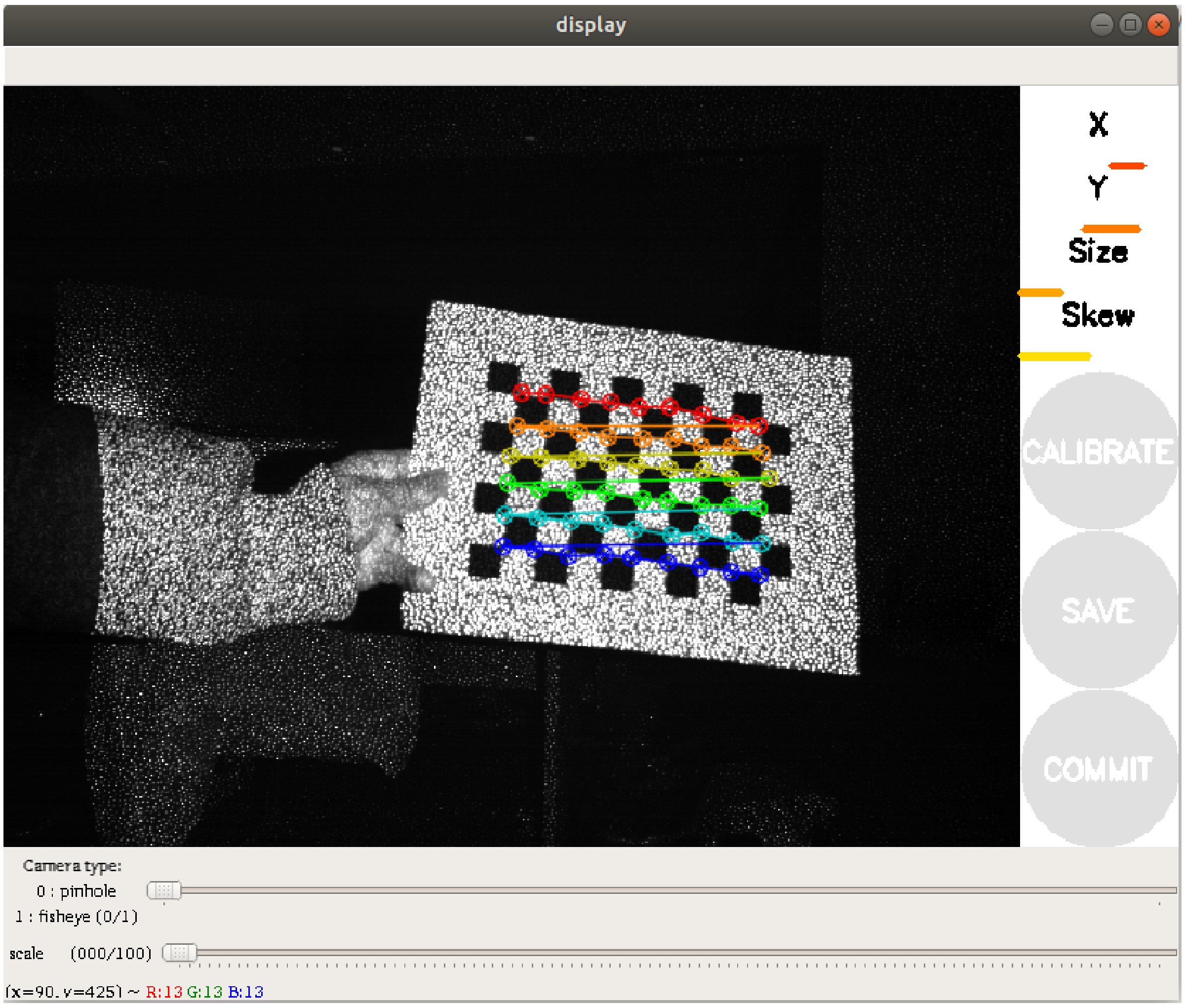
Start calibration node

```
rosrun camera_calibration cameracalibrator.py image:=/camera/ir/image_mono8 --
size 9x6 --square 0.02
```

size: Calibrate the number of internal corner points of the checkerboard, for example, 9X6, with a total of six rows and nine columns of corner points.

square: The side length of the checkerboard, in meters.

image and camera: Set the image topic published by the camera.



The following operations are similar to color camera calibration, changing different poses. The system will identify it independently. The best situation is that the lines under [X], [Y], [Size], and [Skew] will first change from red to yellow and then to green as the data is collected, filling them as fully as possible.

- Click [CALIBRATE] to calculate the internal parameters of the camera. The more pictures you have, the longer it will take. Just wait. (Sixty or seventy is enough, too many can easily get stuck).
- Click [SAVE] to save the calibration results to [/tmp/calibrationdata.tar.gz].
- Click [COMMIT] to write the calibration file into the [.ros/camera_info/ir_camera.yaml] file. The next time you start the camera, the calibration results will be automatically read.

After the calibration is completed, you can move out the [/tmp/calibrationdata.tar.gz] file to see the content.

```
sudo mv /tmp/calibrationdata.tar.gz ~
```

After decompression, there are the image just calibrated, an ost.txt file and an ost.yaml file.

2.3. Single target setting

The principle of setting the color map in section [2.2.1] is the same, except that the startup command and topic name are different. This section is suitable for monocular color image calibration.

Start monocular camera

```
roslaunch usb_cam usb_cam-test.launch
```

Start calibration node

```
rosrun camera_calibration cameracalibrator.py image:=/usb_cam/image_raw
camera:=/usb_cam --size 9x6 --square 0.02
```

The single-purpose calibration results are stored in the file [.ros/camera_info/head_camera.yaml].