5. mediapipe fun gameplay

1. Introduction to mediapipe

MediaPipe is an open-source framework for developing machine learning applications for data stream processing developed by Google. It is a graph-based data processing pipeline for building applications that utilize multiple forms of data sources, such as video, audio, sensor data, and any time-series data. mediaPipe is cross-platform, running on embedded platforms (Raspberry Pi, etc.), mobile devices (iOS and Android), workstations, and servers, and supports mobile GPU acceleration. MediaPipe provides a cross-platform, customizable ML solution for real-time and streaming media, with its core framework implemented in C++ and language support for Java and Objective C. MediaPipe's key concepts include Packet, Stream, Calculator, Graph, and Stream. The main concepts of MediaPipe include Packet, Stream, Calculator, Graph and Subgraph.

MediaPipe features:

- End-to-End Acceleration: Built-in fast ML reasoning and processing accelerates even on common hardware;
- Build Once, Deploy Anywhere: A unified solution for Android, iOS, desktop/cloud, web, and IoT;
- Ready-to-Use Solution: A cutting-edge ML solution that demonstrates the full functionality of the framework;
- Free and open source: framework and solution under Apache 2.0, fully extensible and customizable.

2. Use

2.1 Program Run

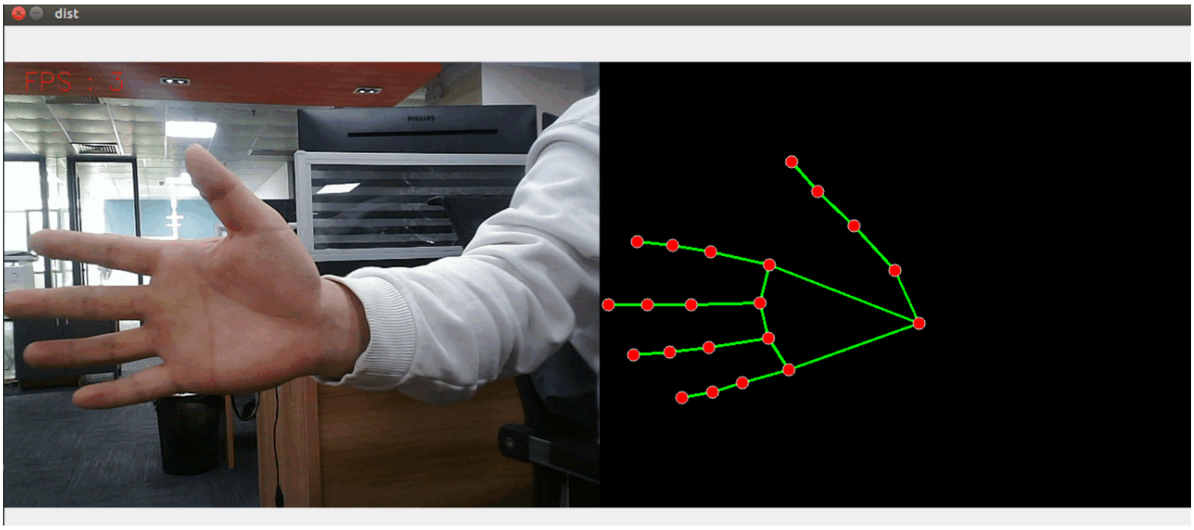
source code path reference.

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe
```

docker terminal input.

```
#手部检测 #Hand detection
ros2 run yahboomcar_mediapipe 01_HandDetector
# 姿态检测 # Attitude detection
ros2 run yahboomcar_mediapipe 02_PoseDetector
# 整体检测 # Overall testing
ros2 run yahboomcar_mediapipe 03_Holistic
# 面部检测 # Facial Inspection
ros2 run yahboomcar_mediapipe 04_FaceMesh
# 人脸识别 #  Face Recognition
ros2 run yahboomcar_mediapipe 05_FaceEyeDetection
```

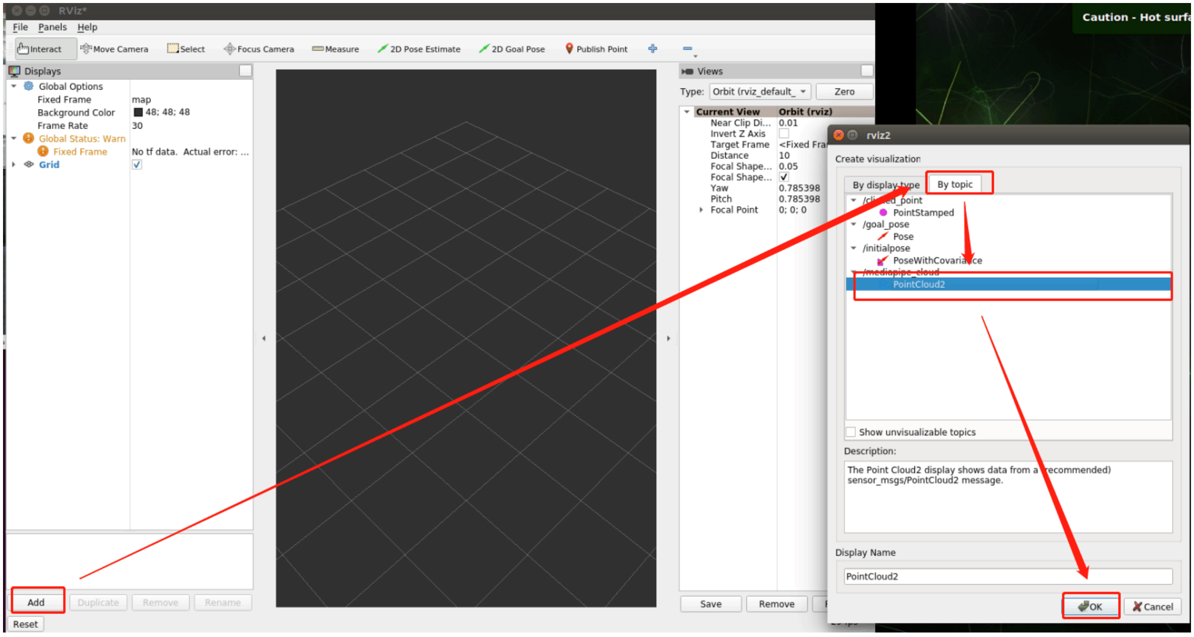
Taking hand detection as an example, the screenshot of the run is as follows



Alternatively, to view the point cloud data, docker terminal input, the

```
#运行点云发布程序
# Run the point cloud publishing program
ros2 run yahboomcar_point pub_point
#开启rviz查看点云 # Open rviz to view point cloud
rviz2
```

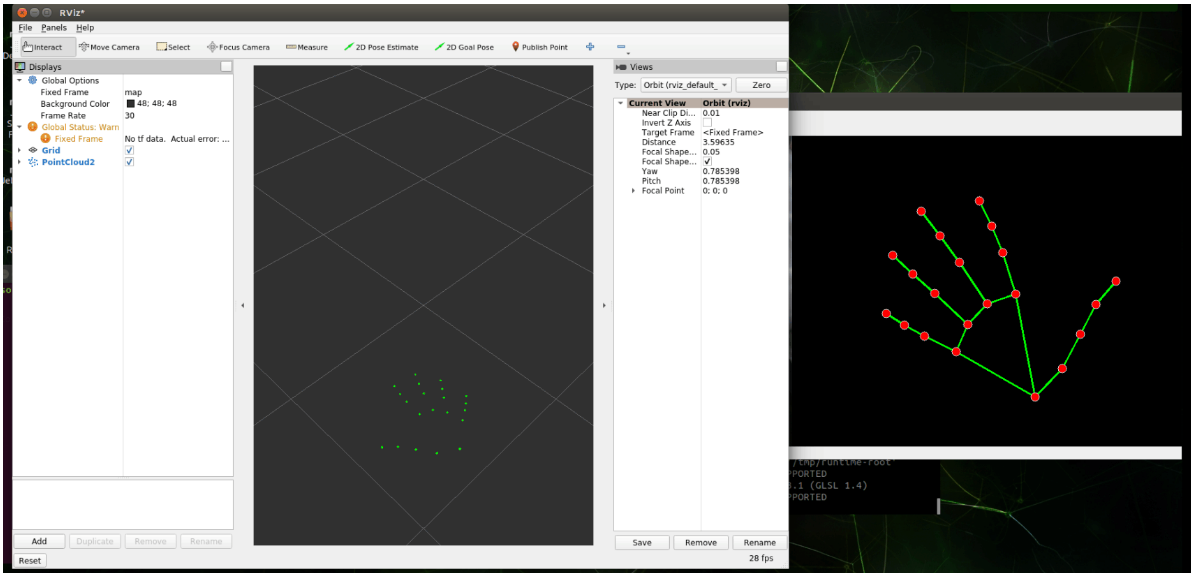
Follow these steps to add a point cloud topic to rviz,



Displays:
Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Global Status: Warn
Fixed Frame No tf data. Actual error: ...
Grid ☑

Views:
Type: Orbit (rviz_default_
Current View Orbit (rviz)
Near Clip Di... 0.01
Invert Z Axis ☐
Target Frame <Fixed Fra
Distance 10
Focal Shape... 0.05
Focal Shape... ☑
Yaw 0.785398
Pitch 0.785398
Focal Point 0; 0; 0

Create visualization
/clicked_point
PointStamped
/goal_pose
Pose
/initialpose
PoseWithCovariance
/mediapipe_cloud
PointCloud2
☐ Show unvisualizable topics
Description:
The Point Cloud2 display shows data from a (recommended) sensor_msgs/PointCloud2 message.
Display Name
PointCloud2



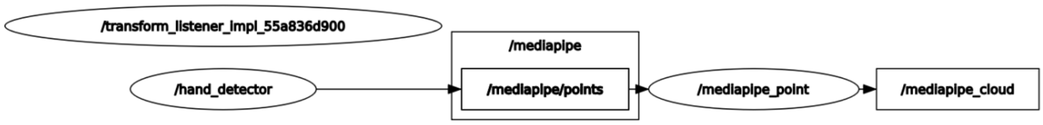
Global Options
Fixed Frame: map
Background Color: 48; 48; 48
Frame Rate: 30
Global Status: Warn
Fixed Frame: No tf data. Actual error: ...
Grid: ☑
PointCloud2: ☑

Type: Orbit (rviz_default_ ▾ Zero
Current View: Orbit (rviz)
Near Clip Di...: 0.01
Invert Z Axis: □
Target Frame: <Fixed Frame>
Distance: 3.59635
Focal Shape...: 0.05
Focal Shape...: ☑
Yaw: 0.785398
Pitch: 0.785398
Focal Point: 0; 0; 0

Above is a point cloud image of the program running the palm detector, known as 01_HandDetector. Point cloud viewing supports only the 01-05 demo.

You can view the node topic communication graph using rqt_graph,

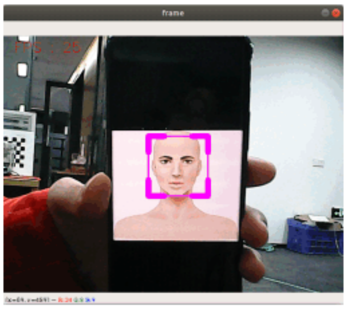
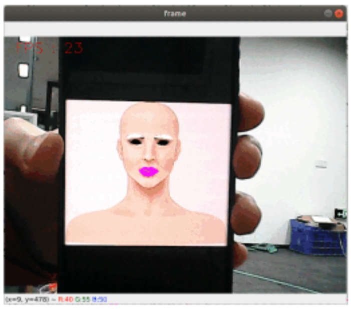
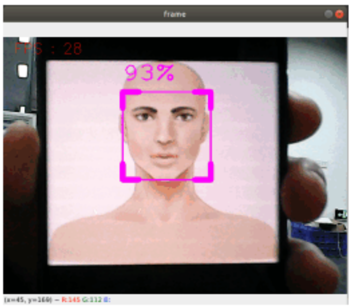
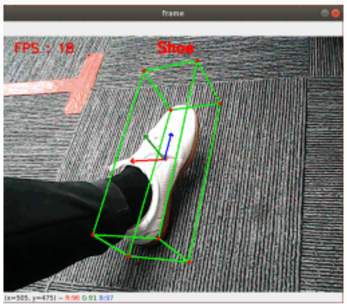
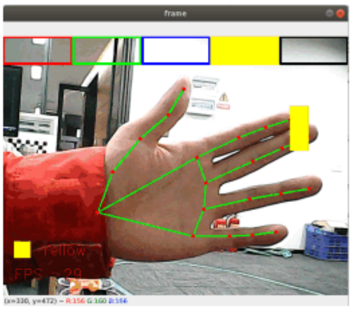
```
ros2 run  rqt_graph rqt_graph
```

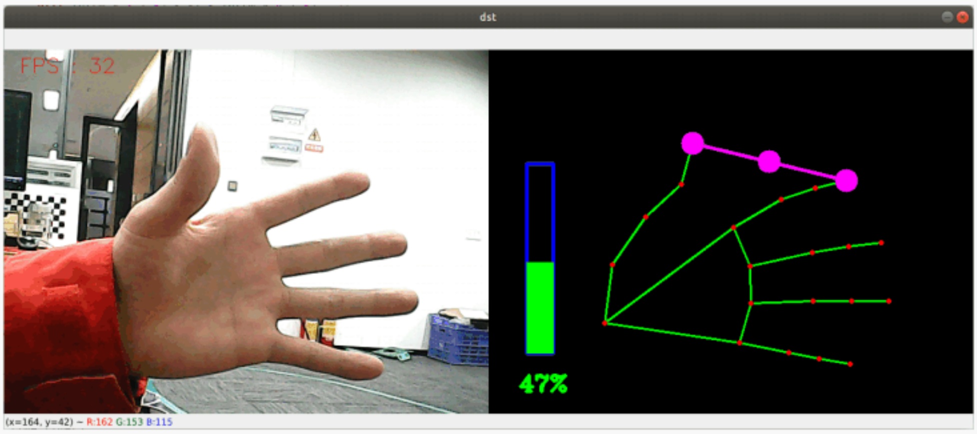
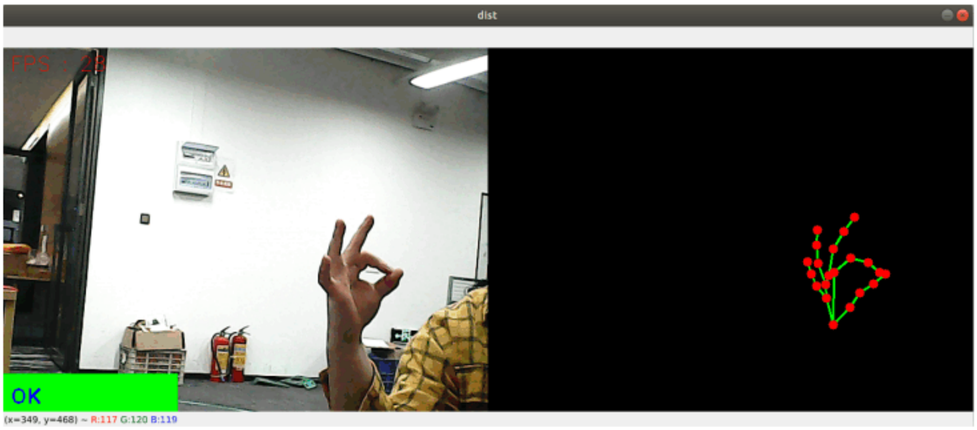


2.2. Other fun games

docker terminal input,

```
cd
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_media
pipe
#人脸特效 # Face effects
python3 06_FaceLandmarks.py
#人脸检测 # Face detection
python3 07_FaceDetection.py
#三维物体识别 # 3D object recognition
python3 08_Objectron.py
#画笔 #brush
python3 09_VirtualPaint.py
#手指控制 #Fingerprint control
python3 10_HandCtrl.py
#手势识别 #Fingerprint Recognition
python3 11_GestureRecognition.py
```

5. Face Recognition	6. Face Effect	7. Face Detection
		
8. 3D object recognition		9. Strokes
		

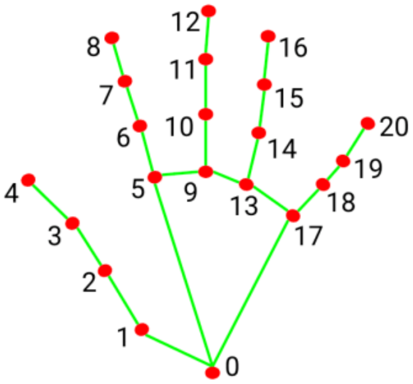
10. Finger control	
11. Gesture recognition	

3. Mediapipe Hands

MediaPipe Hands is a high fidelity hand and finger tracking solution. It utilizes machine learning (ML) to infer the 3D coordinates of 21 hands from a single frame.

After hand detection is performed on the entire image, the coordinates of the 21 3D hand joints within the detected hand regions are accurately key-point localized by regression based on a hand marking model, i.e., direct coordinate prediction. The model learns a consistent internal representation of the hand pose and is robust even to partially visible hands and self-occlusion.

In order to obtain face-toface realistic data, approximately 30K real-world images were manually annotated using 21 3D coordinates, as shown below (Z-values were obtained from the image depth map, if Z-values were available for each corresponding coordinate). To better cover possible hand poses and to provide additional supervision on the nature of hand bases, high-quality synthetic hand models in various backgrounds were also drawn and mapped to the corresponding 3D coordinates.

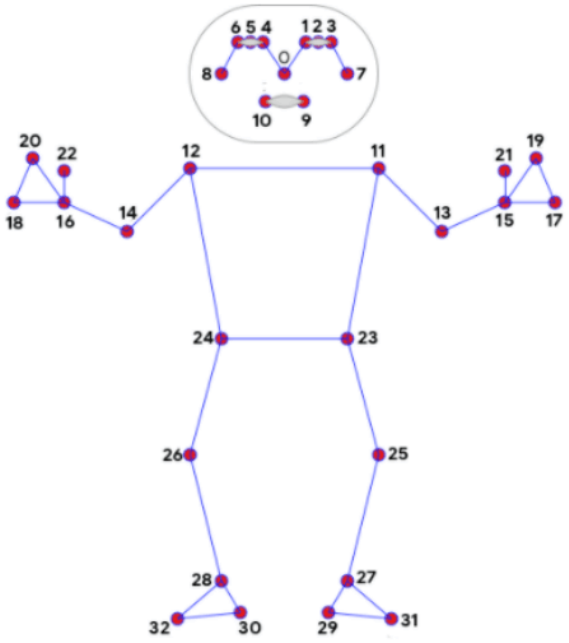


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

4. Mediapipe Pose

MediaPipe Pose, an ML solution for high-fidelity body pose tracking, utilizes the BlazePose study to infer 33 3D coordinates and full-life background segmentation masks from RGB video frames, which also powers the ML Kit pose detection API.

The landmark model in the MediaPipe pose predicts the locations of the 33 pose coordinates (see figure below).



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

5. dlib

The counterpart is the human face effect.

DLIB is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real-world problems. It is widely used by industry and academia in areas such as robotics, embedded devices, cell phones, and large, high-performance computing environments. The dlib library uses 68 points to mark important parts of the human face, such as 18-22 points for the right eyebrow and 51-68 points for the mouth. Faces are detected using the get_frontal_face_detector module of the dlib library, and the shape_predictor_68_face_landmarks.dat feature data is used to predict the values of facial features.