3. Mediapipe fun gameplay

1. Introduction to mediapipe

MediaPipe is an open-source data stream processing machine learning application development framework developed by Google. It is a graph-based data processing pipeline used to build data sources in various forms, such as video, audio, sensor data, and any time series data. MediaPipe is cross-platform and can run on embedded platforms (Raspberry Pi, etc.), mobile devices (iOS and Android), workstations and servers, and supports mobile GPU acceleration. MediaPipe provides cross-platform, customizable ML solutions for real-time and streaming media. The core framework of MediaPipe is implemented in C++ and provides support for languages such as Java and Objective C. The main concepts of MediaPipe include packets, streams, calculators, graphs, and subgraphs.

Features of MediaPipe:

- End-to-end acceleration: built-in fast ML inference and processing can be accelerated even on ordinary hardware;
- Build once, deploy anywhere: unified solution for Android, iOS, desktop/cloud, web and IoT;
- Ready-to-use solution: cutting-edge ML solution that demonstrates the full functionality of the framework;
- Free and open source: framework and solution under Apache2.0, fully extensible and customizable.

2. Use

2.1. Program running

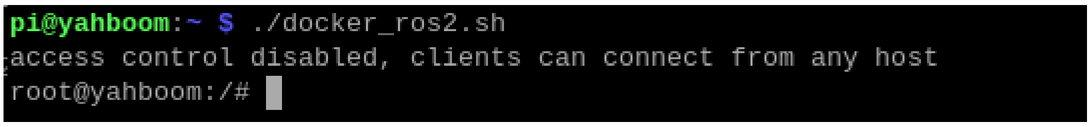
Source code path reference,

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

Open a terminal and enter the following command to enter docker,

```
./docker_ros2.sh
```

The following interface appears, which means that you have successfully entered Docker.

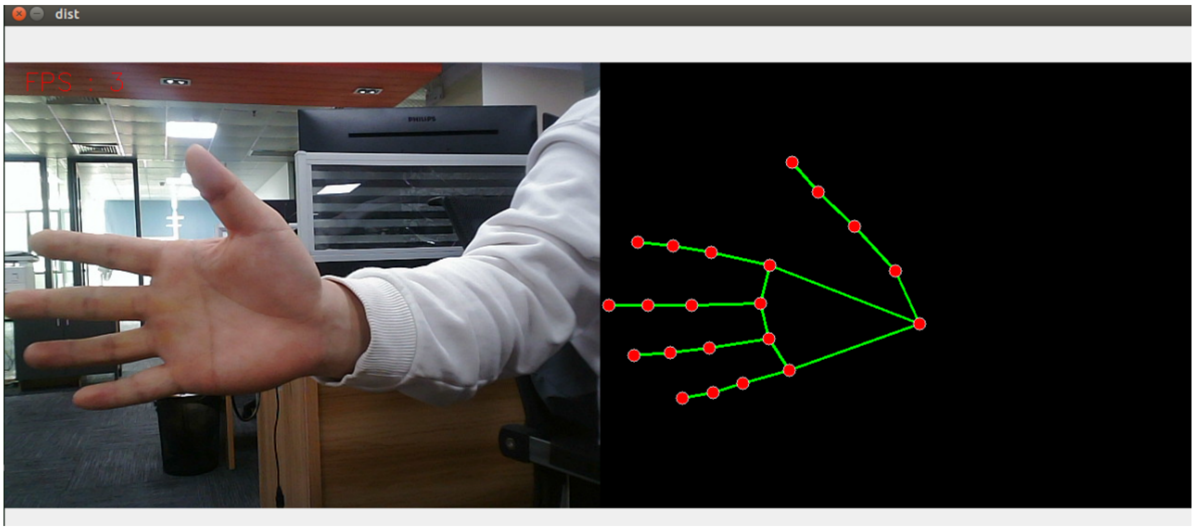


```
pi@yahboom:~ $ ./docker_ros2.sh
access control disabled, clients can connect from any host
root@yahboom:/#
```

Enter in the docker terminal:

```
#HandDetector
ros2 run yahboomcar_mediapipe 01_HandDetector
# PoseDetector
ros2 run yahboomcar_mediapipe 02_PoseDetector
# Holistic
ros2 run yahboomcar_mediapipe 03_Holistic
# FaceMesh
ros2 run yahboomcar_mediapipe 04_FaceMesh
# FaceEyeDetection
ros2 run yahboomcar_mediapipe 05_FaceEyeDetection
```

Taking hand detection as an example, the running screenshot is as follows

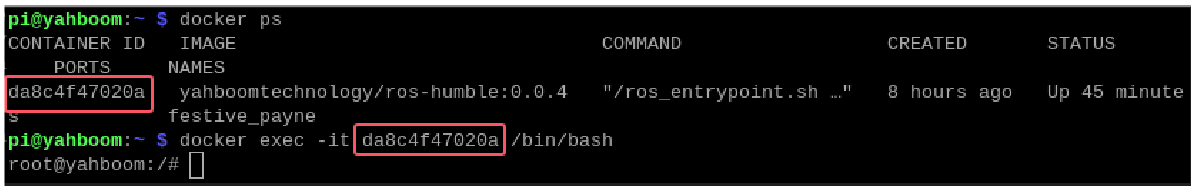


In addition, you can also view the point cloud data.

Open a new terminal, enter the same docker, and change the following da8c4f47020a to the ID displayed in the actual terminal.

```
docker ps
```

```
docker exec -it da8c4f47020a   /bin/bash
```

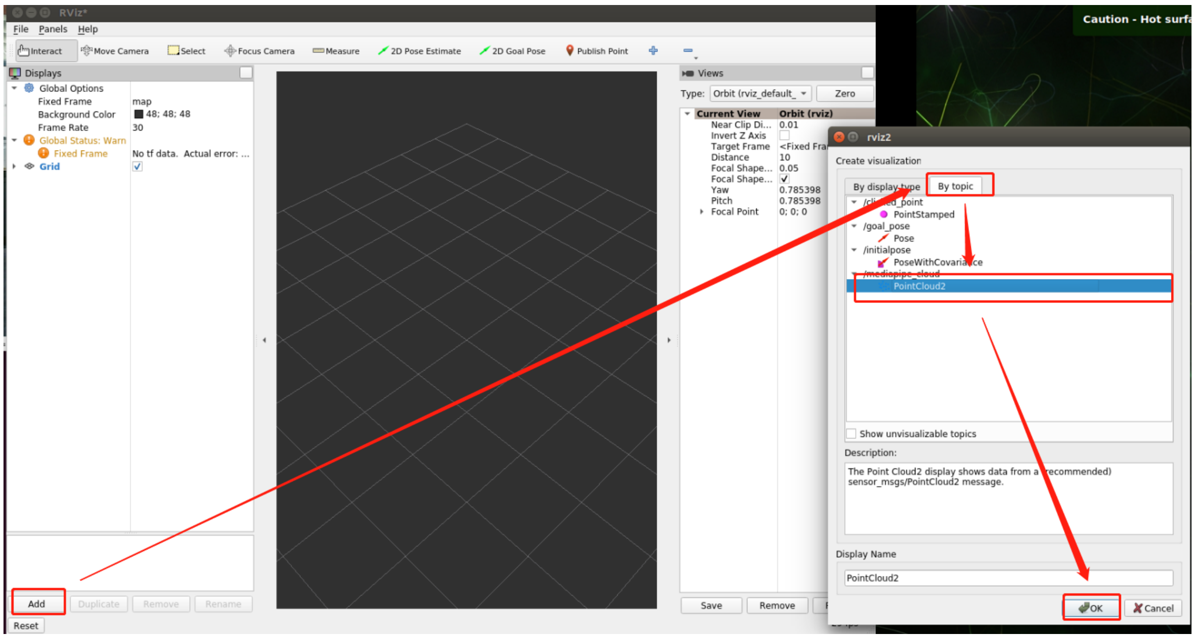


```
pi@yahboom:~ $ docker ps
CONTAINER ID   IMAGE                                 COMMAND                  CREATED         STATUS
    PORTS     NAMES
da8c4f47020a   yahboomtechnology/ros-humble:0.0.4   "/ros_entrypoint.sh …"   8 hours ago     Up 45 minute
s             festive_payne
pi@yahboom:~ $ docker exec -it da8c4f47020a /bin/bash
root@yahboom:/# 
```

Enter in the docker terminal:

```
#Run the point cloud publishing program
ros2 run yahboomcar_point pub_point
#Open rviz to view the point cloud
rviz2
```

Follow the steps below to add a point cloud topic in rviz.



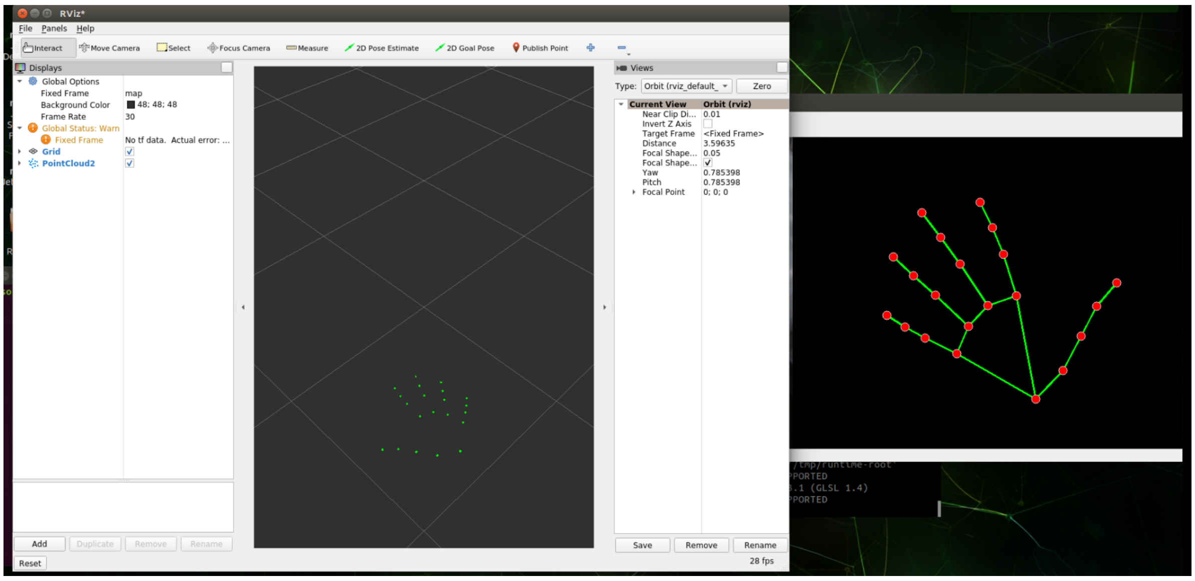
Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Global Status: Warn
Fixed Frame No tf data. Actual error: ...
Grid ✓

Type: Orbit (rviz_default_ Zero
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

Description:
The Point Cloud2 display shows data from a (recommended) sensor_msgs/PointCloud2 message.

Display Name
PointCloud2



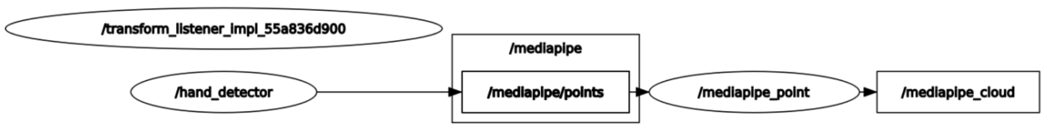
Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Global Status: Warn
Fixed Frame No tf data. Actual error: ...
Grid ✓
PointCloud2 ✓

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

The above is the point cloud diagram of the program running hand detection, that is, 01_HandDetector. Point cloud viewing only supports demos 01-05.

You can view the node topic communication diagram through rqt_graph,

```
ros2 run  rqt_graph rqt_graph
```

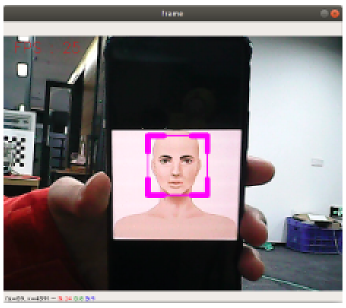
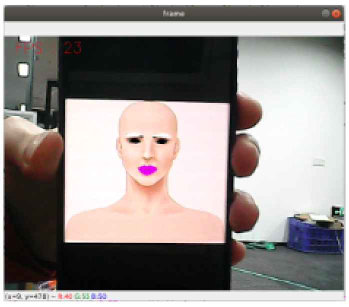
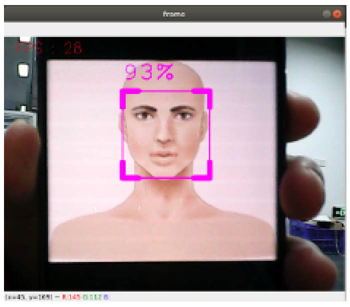
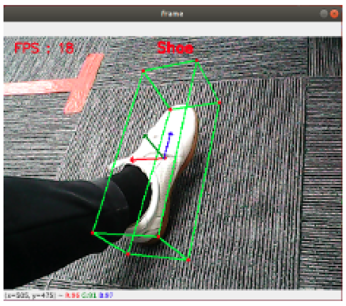
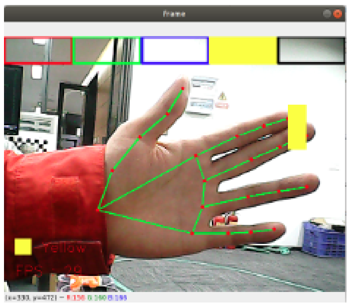


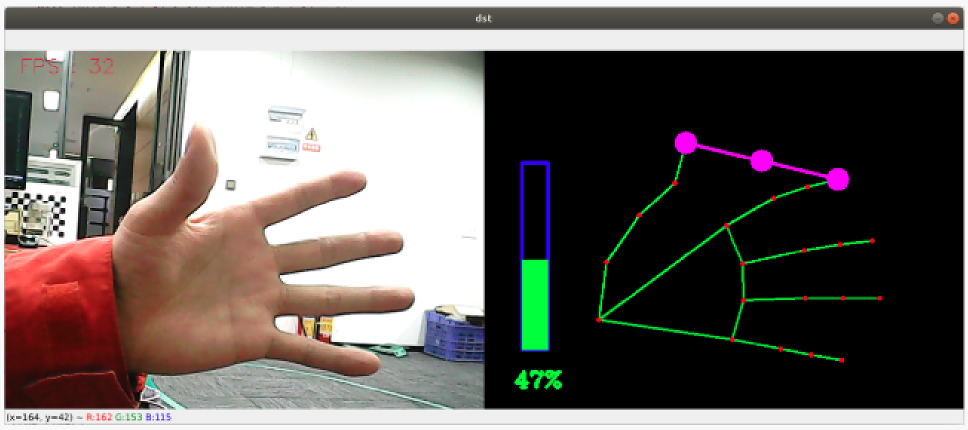
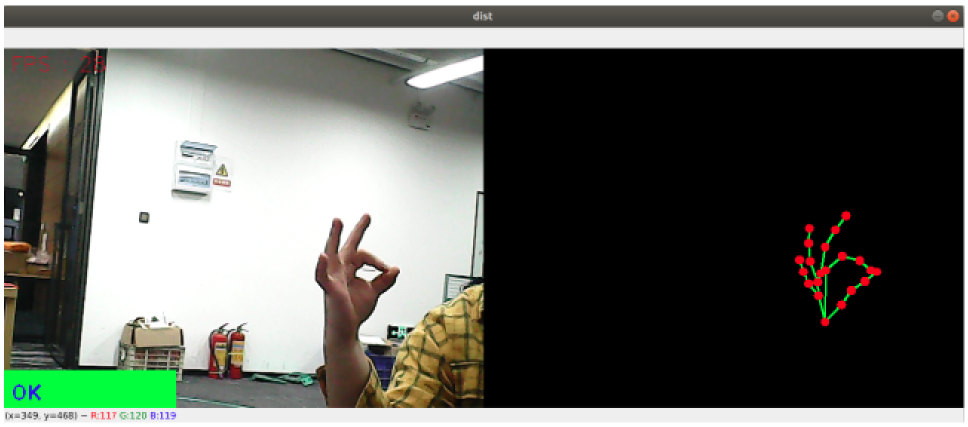
2.2. Other fun ways to play

Enter in the docker terminal,

```
cd
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_media
pipe
```

```
#FaceLandmarks
python3 06_FaceLandmarks.py
#FaceDetection
python3 07_FaceDetection.py
#Objectron
python3 08_Objectron.py
#irtualPaint
python3 09_VirtualPaint.py
#HandCtrl
python3 10_HandCtrl.py
#GestureRecognition
python3 11_GestureRecognition.py
```

05. Face recognition	06. Face special effects	07. Face detection
		
08. 3D object recognition		09. Brush
		

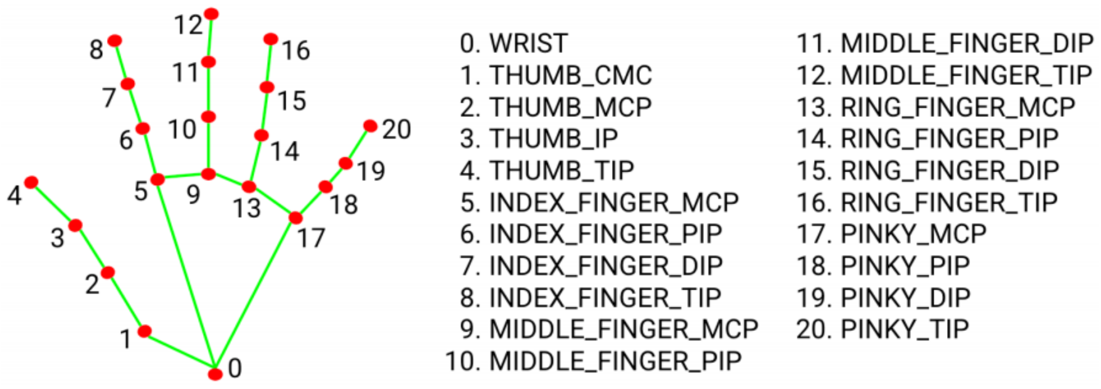
10. Finger control	
11. Gesture recognition	

3. Mediapipe Hands

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

After palm detection for the entire image, the 21 3D hand joint coordinates in the detected hand region are accurately localized by regression based on the hand marker model, i.e. direct coordinate prediction. The model learns a consistent internal hand pose representation that is robust even to partially visible hands and self-occlusion.

To obtain ground truth data, about 30K real-world images were manually annotated with 21 3D coordinates, as shown below (Z values are obtained from the image depth map, if there is a Z value for each corresponding coordinate). To better cover possible hand poses and provide additional supervision on the properties of hand geometry, high-quality synthetic hand models are also rendered against various backgrounds and mapped to their corresponding 3D coordinates.

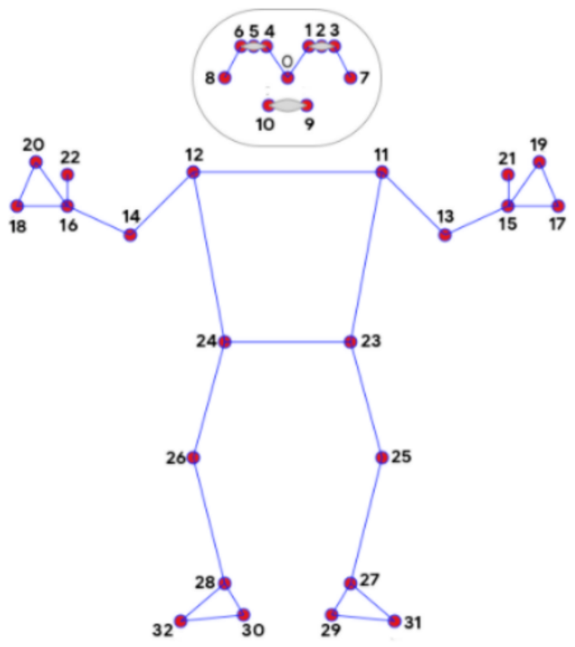


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP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

4. MediaPipe Pose

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

The landmark model in MediaPipe Pose predicts the locations of the 33 pose coordinates (see the 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 corresponding case is face effects.

DLIB is a modern C++ toolkit that contains machine learning algorithms and tools for creating complex software in C++ to solve real-world problems. It is widely used in the industry and academia in fields such as robots, embedded devices, mobile phones, and large-scale high-performance computing environments. The dlib library uses 68 points to mark important parts of the face, such as 18-22 points to mark the right eyebrow, and 51-68 points to mark the mouth. Use the get_frontal_face_detector module of the dlib library to detect the face, and use the shape_predictor_68_face_landmarks.dat feature data to predict the facial feature values.