11. MediaPipe development

mediapipe github： https://github.com/google/mediapipe

mediapipe official website: https://google.github.io/mediapipe/

dlib official website: http://dlib.net/

dlib github： https://github.com/davisking/dlib

11.1. Introduction

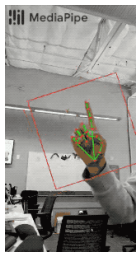
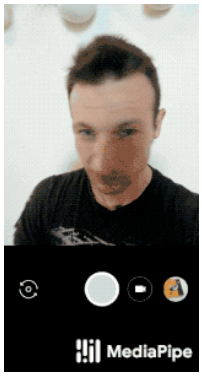
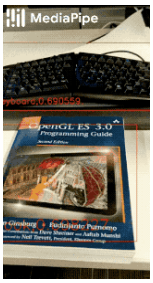
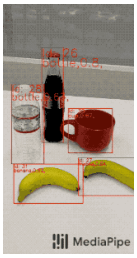
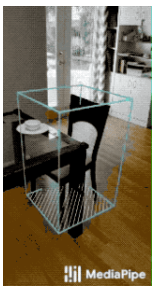
MediaPipe is a data stream processing machine learning application development framework developed and open sourced by Google. It is a graph-based data processing pipeline for building data sources that use many forms, such as video, audio, sensor data, and any time series data. MediaPipe is cross-platform 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 live and streaming media.

The core framework of MediaPipe is implemented in C++ and supports languages such as Java and Objective C. The main concepts of MediaPipe include Packet, Stream, Calculator, Graph and Subgraph.

Features of MediaPipe:

- End-to-end acceleration: Built-in fast ML inference and processing accelerates even on common hardware.
- Build Once, Deploy Anywhere: Unified solution for Android, iOS, desktop/cloud, web and IoT.
- Ready-to-use solutions: Cutting-edge ML solutions that demonstrate the full capabilities of the framework.
- Free and open source: frameworks and solutions under Apache2.0, fully extensible and customizable.

Deep Learning Solutions in MediaPipe

Face Detection	Face Mesh	Iris	Hands	Pose	Holistic
					
Hair Segmentation	**Object Detection**	**Box Tracking**	**Instant Motion Tracking**	**Objectron**	**KNIFT**
					

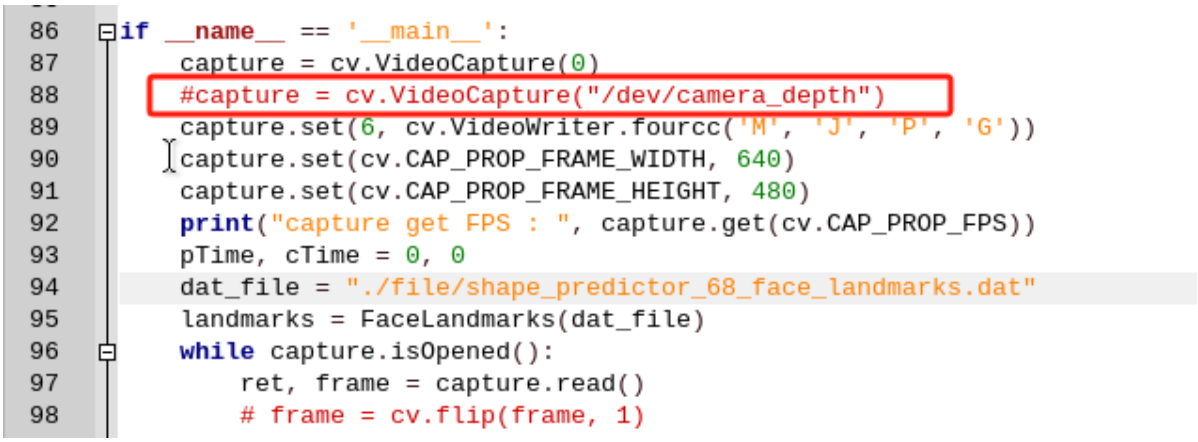
	Android	iOS	C++	Python	JS	Coral
Face Detection	✅	✅	✅	✅	✅	✅
Face Mesh	✅	✅	✅	✅	✅	
Iris	✅	✅	✅			
Hands	✅	✅	✅	✅	✅	
Pose	✅	✅	✅	✅	✅	
Holistic	✅	✅	✅	✅	✅	
Selfie Segmentation	✅	✅	✅	✅	✅	
Hair Segmentation	✅		✅			
Object Detection	✅	✅	✅			✅
Box Tracking	✅	✅	✅			
Instant Motion Tracking	✅					
Objectron	✅		✅	✅	✅	
KNIFT	✅					
AutoFlip			✅			

	Android	iOS	C++	Python	JS	Coral
MediaSequence			✅			
YouTube 8M			✅			

11.2. Use

```
---------------------------------- ROS ----------------- ------------------------
-----------
#pi5 needs to be run in docker
roslaunch yahboomcar_mediapipe cloud_Viewer.launch # Point cloud view: supports
01~04
roslaunch yahboomcar_mediapipe 01_HandDetector.launch # Hand detection
roslaunch yahboomcar_mediapipe 02_PoseDetector.launch # Pose detection
roslaunch yahboomcar_mediapipe 03_Holistic.launch # Overall detection
roslaunch yahboomcar_mediapipe 04_FaceMesh.launch # Face detection
roslaunch yahboomcar_mediapipe 05_FaceEyeDetection.launch # Face recognition
---------------------------------- not ROS ------------------ -------------------
-------
#pi5 does not run in docker
cd ~/yahboomcar_ws/src/yahboomcar_mediapipe/scripts # Enter the directory where
the source code is located
python3 06_FaceLandmarks.py # Face special effects
python3 07_FaceDetection.py # Face detection
python3 08_Objectron.py # 3D object recognition
python3 09_VirtualPaint.py # Brush
python3 10_HandCtrl.py # Finger control
python3 11_GestureRecognition.py # Gesture recognition
```

If the astra depth camera is not started, you can modify the .py file



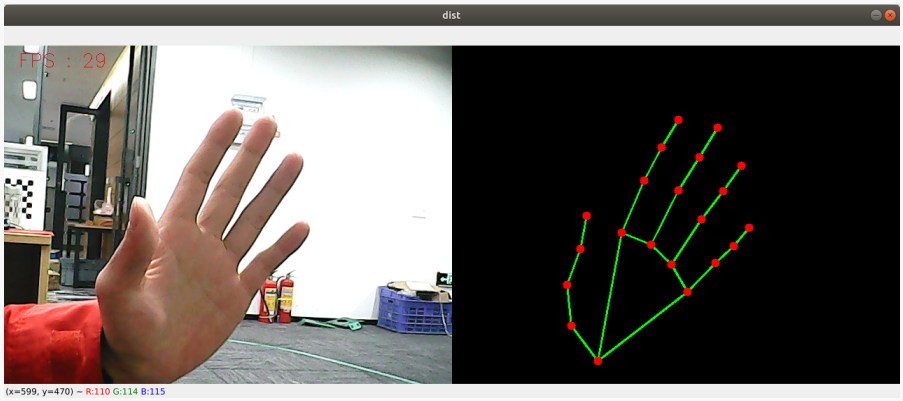
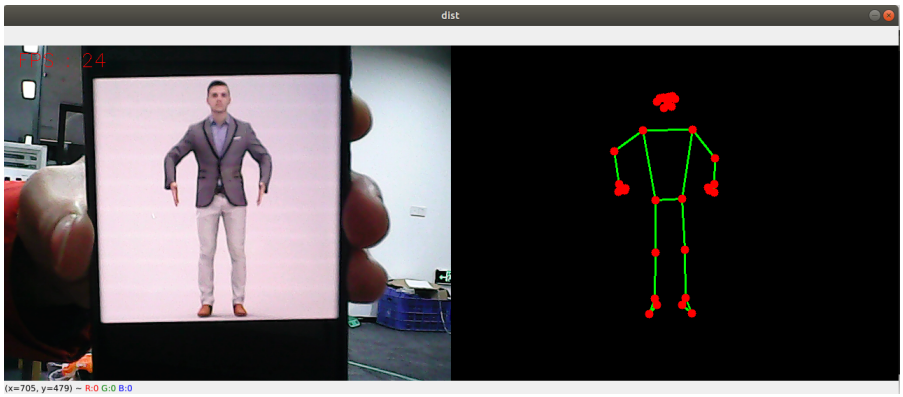
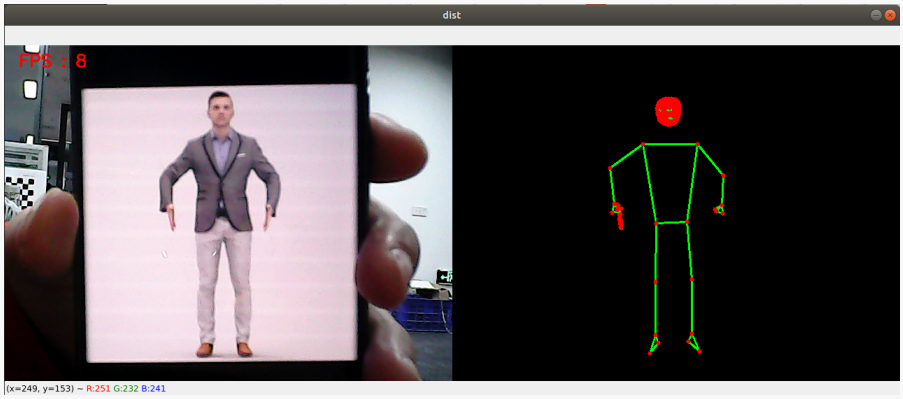
```
if __name__ == '__main__':
    capture = cv.VideoCapture(0)
    #capture = cv.VideoCapture("/dev/camera_depth")
    capture.set(6, cv.VideoWriter.fourcc('M', 'J', 'P', 'G'))
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    print("capture get FPS : ", capture.get(cv.CAP_PROP_FPS))
    pTime, cTime = 0, 0
    dat_file = "./file/shape_predictor_68_face_landmarks.dat"
    landmarks = FaceLandmarks(dat_file)
    while capture.isOpened():
        ret, frame = capture.read()
        # frame = cv.flip(frame, 1)
```

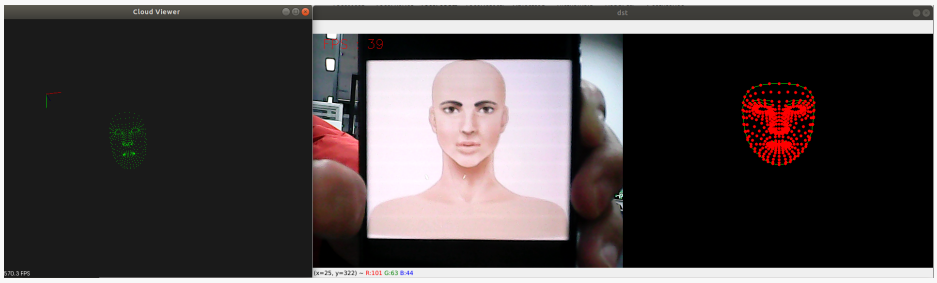
```
capture = cv.VideoCapture("/dev/camera_depth")
```

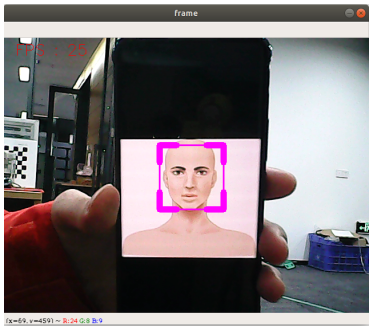
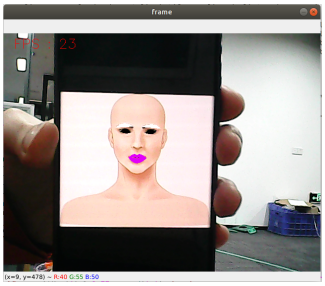
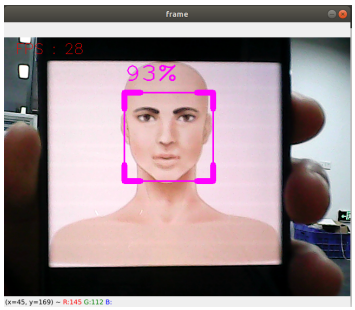
During use, you need to pay attention to the following:

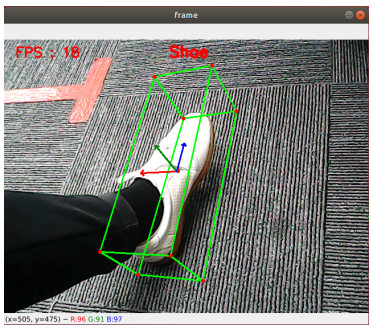
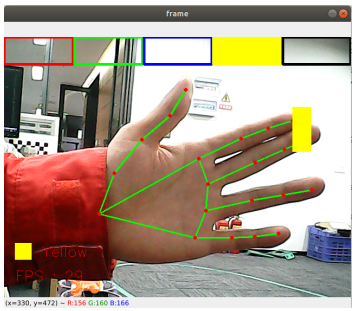
- Hand detection, posture detection, overall detection, and face detection all have point cloud viewing functions. Take face detection as an example.
- All functions【q key】to exit.
- Overall detection: including hand, face, body pose detection.

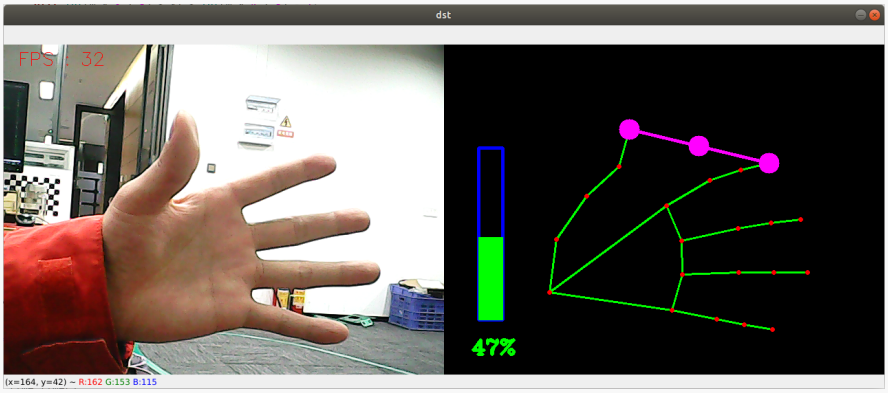
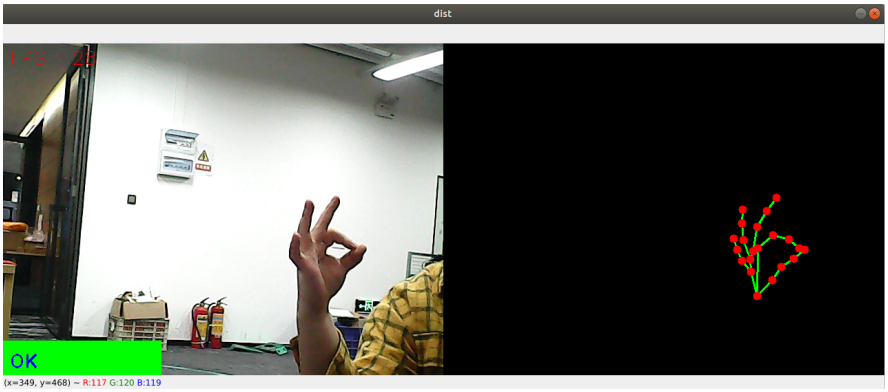
- 3D Object Recognition: Recognized objects are: ['Shoe', 'Chair', 'Cup', 'Camera'], a total of 4 categories; click [f key] to switch the recognition object; jetson series cannot use keyboard keys to switch recognition The object needs to change the [self.index] parameter in the source code.
- Brush: When the index finger and middle finger of the right hand are combined, it is in the selected state, and the color selection box will pop up at the same time. When the two fingertips move to the corresponding color position, select the color (black is the eraser); the index finger and the middle finger start to be in the drawing state, which can be displayed on the drawing board. Draw arbitrarily.
- Finger control: Click [f key] to switch the recognition effect.
- Finger recognition: gesture recognition designed with the right hand as the criterion, can be accurately recognized when certain conditions are met. Recognized gestures are: [Zero, One, Two, Three, Four, Five, Six, Seven, Eight, Ok, Rock, Thumb_up (like), Thumb_down (thumb down), Heart_single (one-handed heart)] , a total of 14 categories.

01. Hand detection	
02. Attitude detection	
03. Overall inspection	

04. Face Detection	

05. Face recognition	06. Face special effects	07. Face detection
		

08. 3D object recognition		09. Brush
		

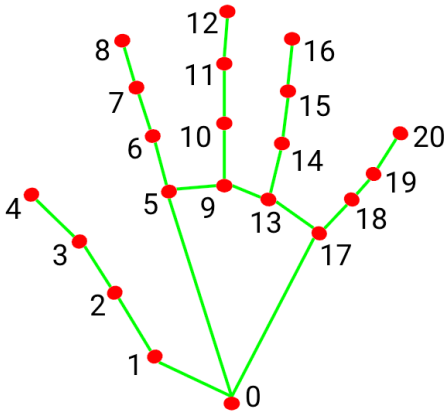
10. Finger control	
11. Gesture recognition	

11.3. MediaPipe Hands

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

After palm detection is performed on the entire image, accurate key point positioning is performed on the 21 3D hand joint coordinates in the detected hand region by regression according to the hand marker model, that is, direct coordinate prediction. The model learns consistent internal hand pose representations and is robust to even 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 the possible hand poses and provide additional supervision on the nature of the hand geometry, high-quality synthetic hand models in various contexts are also drawn and mapped to the corresponding 3D coordinates.

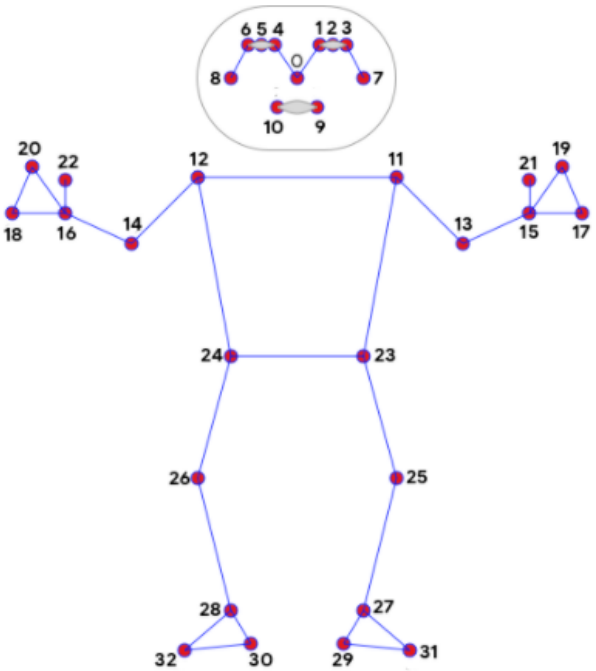


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP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

11.4. MediaPipe Pose

MediaPipe Pose is an ML solution for high-fidelity body pose tracking that infers 33 3D coordinates and a full-body background segmentation mask from RGB video frames using the BlazePose research that also powers the ML Kit pose detection API.

The landmark model in MediaPipe pose predicts the location of 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

11.5. dlib

The corresponding case is the 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 fields such as robotics, 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 face feature value