1. Hand detection

1.1. Introduction

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 supports languages such as Java and Objective C. The main concepts of MediaPipe include packets, streams, calculators, graphs, and subgraphs.

Features of MediaPipe:

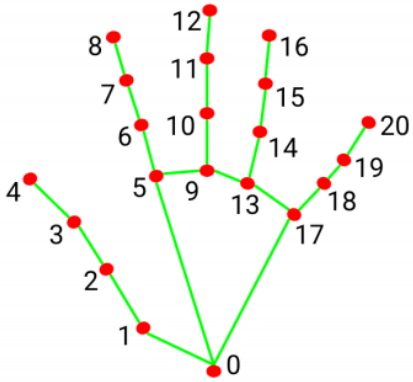
- End-to-end acceleration: built-in fast ML inference and processing can be accelerated even on commodity hardware.
- Build once, deploy anywhere: unified solution for Android, iOS, desktop/cloud, web and IoT.
- Ready-to-use solution: cutting-edge ML solution that demonstrates the full capabilities of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

1.2, 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 area are accurately located by regression based on the hand marker model, that is, direct coordinate prediction. The model learns a consistent internal hand pose representation that is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, ~30K real-world images were manually annotated with 21 3D coordinates, as shown below (Z values were taken from the image depth map, if available for each corresponding coordinate). To provide better coverage of possible hand poses and provide additional supervision on the properties of the hand geometry, high-quality synthetic hand models were also drawn against various backgrounds 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

1.3, Hand detection

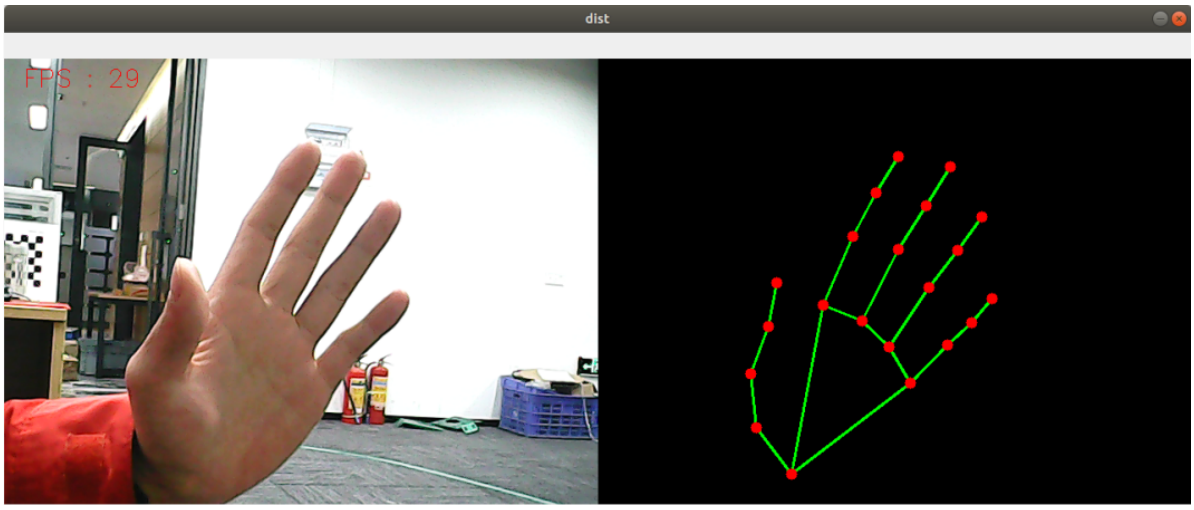
1.3.1, Start

Start the camera

```
roslaunch ascamera hp60c.launch
```

Terminal input,

```
roslaunch yahboomcar_mediapipe 01_HandDetector.launch
```



1.3.2, Source code

Source code location:
/home/yahboom/ascam_ws/src/yahboomcar_mediapipe/scripts/01_HandDetector.py

```
#!/usr/bin/env python3
# encoding: utf-8
import rospy
import time
import cv2 as cv
import numpy as np
import mediapipe as mp
from geometry_msgs.msg import Point
from yahboomcar_msgs.msg import PointArray
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

class HandDetector:
def __init__(self, mode=False, maxHands=2, detectorCon=0.5, trackCon=0.5):
self.mpHand = mp.solutions.hands
self.mpDraw = mp.solutions.drawing_utils
self.hands = self.mpHand.Hands(
static_image_mode=mode,
max_num_hands=maxHands,
min_detection_confidence=detectorCon,
min_tracking_confidence=trackCon )
self.pub_point = rospy.Publisher('/mediapipe/points', PointArray,
queue_size=1000)
```

```
self.lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0, 255),
thickness=-1, circle_radius=6)
self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255, 0),
thickness=2, circle_radius=2)
self.bridge = CvBridge()
self.sub_image = rospy.Subscriber('/ascamera_hp60c/rgb0/image', Image,
self.image_callback, queue_size=1)
self.pTime = 0

def pubHandsPoint(self, frame, draw=True):
pointArray = PointArray()
img = np.zeros(frame.shape, np.uint8)
img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
self.results = self.hands.process(img_RGB)
if self.results.multi_hand_landmarks:
'''
draw_landmarks(): Draws landmarks and connections on an image.
image: A three-channel RGB image represented as a numpy ndarray.
landmark_list: A list of normalized landmarks to be annotated on the image.
connections: A list of landmark index tuples specifying how the landmarks are
connected in the drawing.
landmark_drawing_spec: A drawing parameter for specifying the drawing settings of
landmarks, such as color, line thickness, and circle radius.
connection_drawing_spec: A drawing parameter for specifying the drawing settings
of connections, such as color and line thickness.
'''
for i in range(len(self.results.multi_hand_landmarks)):
if draw: self.mpDraw.draw_landmarks(frame, self.results.multi_hand_landmarks[i],
self.mpHand.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
self.mpDraw.draw_landmarks(img, self.results.multi_hand_landmarks[i],
self.mpHand.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
for id, lm in enumerate(self.results.multi_hand_landmarks[i].landmark):
point = Point()
point.x, point.y, point.z = lm.x, lm.y, lm.z
pointArray.points.append(point)
self.pub_point.publish(pointArray)
return frame, img

def frame_combine(slef,frame, src):
if len(frame.shape) == 3:
frameH, frameW = frame.shape[:2]
srcH, srcW = src.shape[:2]
dst = np.zeros((max(frameH, srcH), frameW + srcW, 3), np.uint8)
dst[:, :frameW] = frame[:, :]
dst[:, frameW:] = src[:, :]
else:
src = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
frameH, frameW = frame.shape[:2]
imgH, imgW = src.shape[:2]
dst = np.zeros((frameH, frameW + imgW), np.uint8)
dst[:, :frameW] = frame[:, :]
dst[:, frameW:] = src[:, :]
return dst

def image_callback(self, msg):
```

```
try:
# Convert ROS Image message to OpenCV image
frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")

# Process the frame and publish hand points
frame, img = self.pubHandsPoint(frame, draw=True)

# Calculate and display FPS
cTime = time.time()
fps = 1 / (cTime - self.pTime)
self.pTime = cTime
text = "FPS : " + str(int(fps))
cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 1)

# Combine and display the frames
dist = self.frame_combine(frame, img)
cv.imshow('dist', dist)

# Check for 'q' key press to quit
if cv.waitKey(1) & 0xFF == ord('q'):
rospy.signal_shutdown("User requested shutdown")
except Exception as e:
rospy.logerr("Could not process image: {e}")

if __name__ == '__main__':
rospy.init_node('handDetector', anonymous=True)
hand_detector = HandDetector(maxHands=2)
rospy.spin()
cv.destroyAllWindows()
```