Hand Detection

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 (such as Jetson nano), 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:

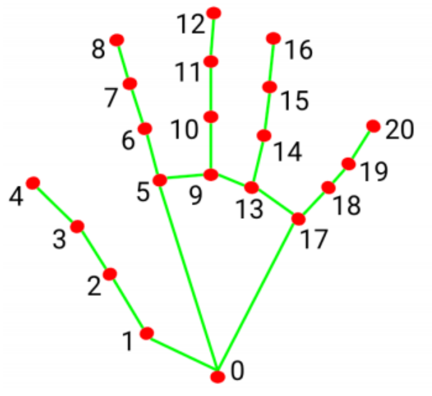
- End-to-end acceleration: built-in fast ML inference and processing can be accelerated even on ordinary hardware.
- Build once, deploy anywhere: unified solution for Android, iOS, desktop/cloud, web and IoT.
- Ready-to-use solution: cutting-edge ML solution that demonstrates the full capabilities of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

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 of 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-occlusion.

To obtain ground truth data, about 30K real-world images were manually annotated with 21 3D coordinates, as shown below (Z values are taken 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 the hand geometry, high-quality synthetic hand models against various backgrounds are also drawn and mapped to the corresponding 3D coordinates.



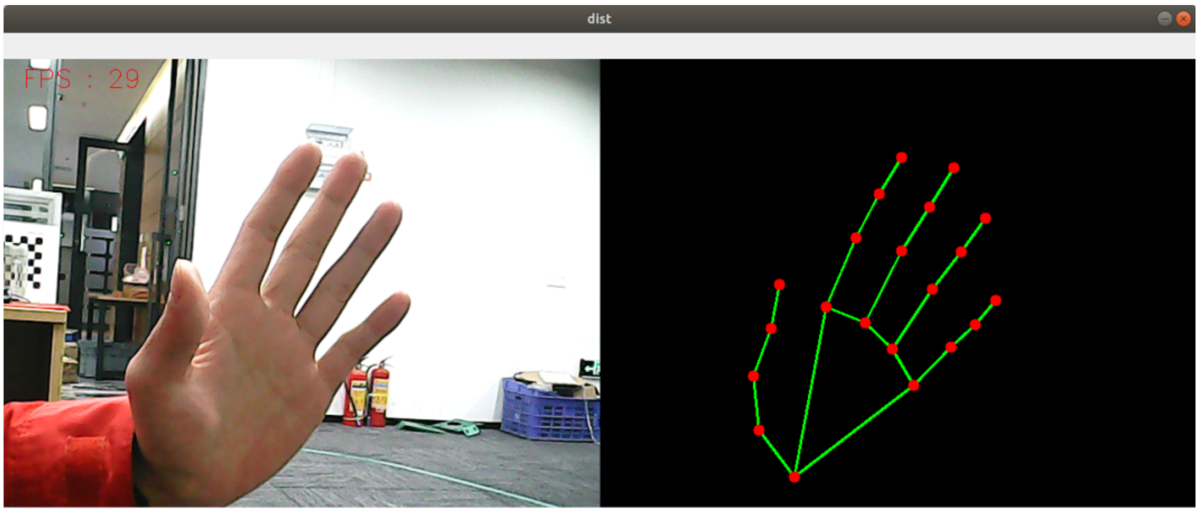
0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP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

3. Hand Detection

3.1. Startup

- Enter the following command to start the program

```
ros2 run dofbot_pro_mediapipe 01_HandDetector
```



3.2, Source code

Source code location: ~/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/01_HandDetector.py

```
#!/usr/bin/env python3
# encoding: utf-8
import rclpy
import cv2 as cv
import numpy as np
import mediapipe as mp
import time
from rclpy.node import Node
from geometry_msgs.msg import Point
from dofbot_pro_msgs.msg import PointArray

class HandDetector(Node):
    def __init__(self):
        super().__init__('hand_detector')
        self.mpHand = mp.solutions.hands
        self.mpDraw = mp.solutions.drawing_utils
        self.hands = self.mpHand.Hands(
            static_image_mode=False,
            max_num_hands=2,
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5
        )

        self.publisher_ = self.create_publisher(PointArray, '/mediapipe/points',
10)
        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)

        # Initialize the camera
        self.capture = cv.VideoCapture(0, cv.CAP_V4L2)
        self.capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
        self.capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)

        if not self.capture.isOpened():
            self.get_logger().error("Failed to open the camera")
            return

        self.get_logger().info(f"Camera FPS:
{self.capture.get(cv.CAP_PROP_FPS)}")
        self.pTime = time.time()

        # Create a timer to process frames approximately at 30 FPS
        self.timer = self.create_timer(0.03, self.process_frame)

    def process_frame(self):
        ret, frame = self.capture.read()
        if not ret:
            self.get_logger().error("Failed to read frame")
            return

        frame, img = self.pubHandsPoint(frame, draw=True)

        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)

        HandDetector = self.frame_combine(frame, img)
        cv.imshow('HandDetector', HandDetector)

        if cv.waitKey(1) & 0xFF == ord('q'):
            self.get_logger().info("Exiting program")
            self.capture.release()
            cv.destroyAllWindows()
            self.destroy_node()
            rclpy.shutdown()
            exit(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:
            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 lm in 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.publisher_.publish(pointArray)
        return frame, img

    def frame_combine(self, 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 main(args=None):
    rclpy.init(args=args)
    node = HandDetector()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.capture.release()
        cv.destroyAllWindows()
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```