Brush

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 functionality of the framework.
- Free and open source: framework and solution under Apache2.0, fully extensible and customizable.

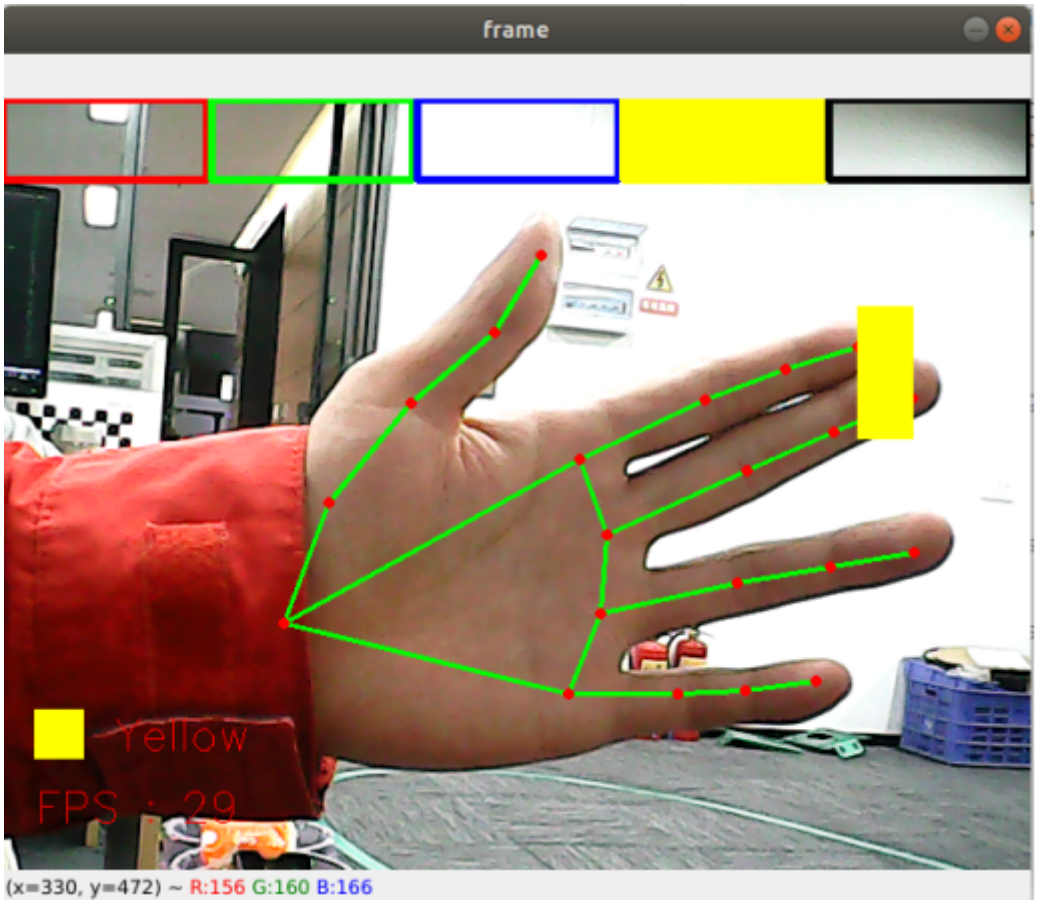
2. Brush

When the index finger and middle finger of the right hand are combined, it is the selection state, and the color selection box pops up at the same time. When the two fingertips move to the corresponding color position, the color is selected (black is an eraser); when the index finger and middle finger are separated, it is the drawing state, and you can draw arbitrarily on the drawing board.

2.1. Start

- Enter the following command to start the program

```
ros2 run dofbot_pro_mediapipe 08_VirtualPaint
```



2.2. Source code

Source code location:
~/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/08_VirtualPaint.py

```
#!/usr/bin/env python3
# encoding: utf-8

import math
import time
import cv2 as cv
import numpy as np
import mediapipe as mp
import rclpy
from rclpy.node import Node
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.tipIds = [4, 8, 12, 16, 20]
        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.lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0,
255), thickness=-1, circle_radius=15)
```

```
        self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255, 0),
thickness=10, circle_radius=10)

    def findHands(self, frame, draw=True):
        self.lmList = []
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.hands.process(img_RGB)
        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw: self.mpDraw.draw_landmarks(frame, handLms,
self.mpHand.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
                else: self.mpDraw.draw_landmarks(frame, handLms,
self.mpHand.HAND_CONNECTIONS)
            for id, lm in
enumerate(self.results.multi_hand_landmarks[0].landmark):
                h, w, c = frame.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                self.lmList.append([id, cx, cy])
        return frame, self.lmList

    def fingersUp(self):
        fingers=[]
        if (self.calc_angle(self.tipIds[0], self.tipIds[0] - 1, self.tipIds[0] -
2) > 150.0) and (
                self.calc_angle(self.tipIds[0] - 1, self.tipIds[0] - 2,
self.tipIds[0] - 3) > 150.0): fingers.append(1)
        else:
            fingers.append(0)
        for id in range(1, 5):
            if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] - 2]
[2]:
                fingers.append(1)
            else:
                fingers.append(0)
        return fingers

    def get_dist(self, point1, point2):
        x1, y1 = point1
        x2, y2 = point2
        return abs(math.sqrt(math.pow(abs(y1 - y2), 2) + math.pow(abs(x1 - x2),
2)))

    def calc_angle(self, pt1, pt2, pt3):
        point1 = self.lmList[pt1][1], self.lmList[pt1][2]
        point2 = self.lmList[pt2][1], self.lmList[pt2][2]
        point3 = self.lmList[pt3][1], self.lmList[pt3][2]
        a = self.get_dist(point1, point2)
        b = self.get_dist(point2, point3)
        c = self.get_dist(point1, point3)
        try:
            radian = math.acos((math.pow(a, 2) + math.pow(b, 2) - math.pow(c, 2))
/ (2 * a * b))
            angle = radian / math.pi * 180
        except:
            angle = 0
        return abs(angle)
```

```
class HandPaintingNode(Node):
    def __init__(self):
        super().__init__('hand_painting_node')
        self.publisher_ = self.create_publisher(Image, 'hand_painting_image', 10)
        self.timer = self.create_timer(0.1, self.timer_callback)
        self.bridge = CvBridge()
        self.hand_detector = HandDetector(detectorCon=0.85)
        self.xp = self.yp = self.pTime = self.boxx = 0
        self.tipIds = [4, 8, 12, 16, 20]
        self.imgCanvas = np.zeros((480, 640, 3), np.uint8)
        self.brushThickness = 5
        self.eraserThickness = 100
        self.top_height = 50
        self.Color = "Red"
        self.ColorList = {
            'Red': (0, 0, 255),
            'Green': (0, 255, 0),
            'Blue': (255, 0, 0),
            'Yellow': (0, 255, 255),
            'Black': (0, 0, 0),
        }
        self.capture = cv.VideoCapture(0, cv.CAP_V4L2)
        self.capture.set(6, cv.VideoWriter.fourcc('M', 'J', 'P', 'G'))
        self.capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
        self.capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)

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

        h, w, c = frame.shape
        frame, lmList = self.hand_detector.findHands(frame, draw=False)
        if len(lmList) != 0:
            x1, y1 = lmList[8][1:]
            x2, y2 = lmList[12][1:]
            fingers = self.hand_detector.fingersUp()
            if fingers[1] and fingers[2]:
                if y1 < self.top_height:
                    if 0 < x1 < int(w / 5) - 1:
                        self.boxx = 0
                        self.Color = "Red"
                    if int(w / 5) < x1 < int(w * 2 / 5) - 1:
                        self.boxx = int(w / 5)
                        self.Color = "Green"
                    elif int(w * 2 / 5) < x1 < int(w * 3 / 5) - 1:
                        self.boxx = int(w * 2 / 5)
                        self.Color = "Blue"
                    elif int(w * 3 / 5) < x1 < int(w * 4 / 5) - 1:
                        self.boxx = int(w * 3 / 5)
                        self.Color = "Yellow"
                    elif int(w * 4 / 5) < x1 < w - 1:
                        self.boxx = int(w * 4 / 5)
                        self.Color = "Black"
```

```
                cv.rectangle(frame, (x1, y1 - 25), (x2, y2 + 25),
self.ColorList[self.Color], cv.FILLED)
                cv.rectangle(frame, (self.boxx, 0), (self.boxx + int(w / 5),
self.top_height), self.ColorList[self.Color], cv.FILLED)
                cv.rectangle(frame, (0, 0), (int(w / 5) - 1, self.top_height),
self.ColorList['Red'], 3)
                cv.rectangle(frame, (int(w / 5) + 2, 0), (int(w * 2 / 5) - 1,
self.top_height), self.ColorList['Green'], 3)
                cv.rectangle(frame, (int(w * 2 / 5) + 2, 0), (int(w * 3 / 5) - 1,
self.top_height), self.ColorList['Blue'], 3)
                cv.rectangle(frame, (int(w * 3 / 5) + 2, 0), (int(w * 4 / 5) - 1,
self.top_height), self.ColorList['Yellow'], 3)
                cv.rectangle(frame, (int(w * 4 / 5) + 2, 0), (w - 1,
self.top_height), self.ColorList['Black'], 3)
            if fingers[1] and fingers[2] == False and math.hypot(x2 - x1, y2 -
y1) > 50:
                if self.xp == self.yp == 0: self.xp, self.yp = x1, y1
                if self.Color == 'Black':
                    cv.line(frame, (self.xp, self.yp), (x1, y1),
self.ColorList[self.Color], self.eraserThickness)
                    cv.line(self.imgCanvas, (self.xp, self.yp), (x1, y1),
self.ColorList[self.Color], self.eraserThickness)
                else:
                    cv.line(frame, (self.xp, self.yp), (x1, y1),
self.ColorList[self.Color], self.brushThickness)
                    cv.line(self.imgCanvas, (self.xp, self.yp), (x1, y1),
self.ColorList[self.Color], self.brushThickness)
                cv.circle(frame, (x1, y1), 15, self.ColorList[self.Color],
cv.FILLED)
                self.xp, self.yp = x1, y1
            else: self.xp = self.yp = 0

        imgGray = cv.cvtColor(self.imgCanvas, cv.COLOR_BGR2GRAY)
        _, imgInv = cv.threshold(imgGray, 50, 255, cv.THRESH_BINARY_INV)
        imgInv = cv.cvtColor(imgInv, cv.COLOR_GRAY2BGR)
        frame = cv.bitwise_and(frame, imgInv)
        frame = cv.bitwise_or(frame, self.imgCanvas)

        if cv.waitKey(1) & 0xFF == ord('q'):
            self.capture.release()
            cv.destroyAllWindows()
            rclpy.shutdown()

        cTime = time.time()
        fps = 1 / (cTime - self.pTime)
        self.pTime = cTime
        text = "FPS : " + str(int(fps))
        cv.rectangle(frame, (20, h - 100), (50, h - 70),
self.ColorList[self.Color], cv.FILLED)
        cv.putText(frame, self.Color, (70, h - 75), cv.FONT_HERSHEY_SIMPLEX, 0.9,
(0, 0, 255), 1)
        cv.putText(frame, text, (20, h-30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0,
255), 1)

        msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
        self.publisher_.publish(msg)
```

```
        cv.imshow('frame', frame)

def main(args=None):
    rclpy.init(args=args)
    hand_painting_node = HandPaintingNode()
    rclpy.spin(hand_painting_node)
    hand_painting_node.destroy_node()
    rclpy.shutdown()

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