Fingertip trajectory recognition

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.

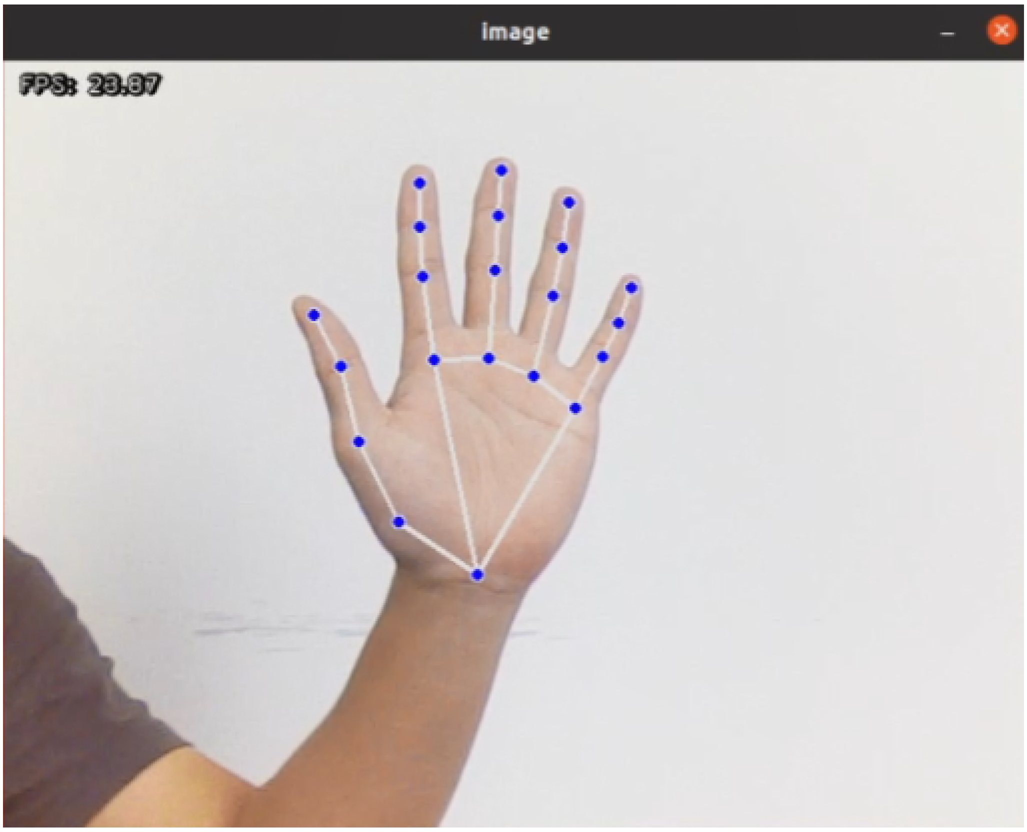
Features of MediaPipe:

- 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.

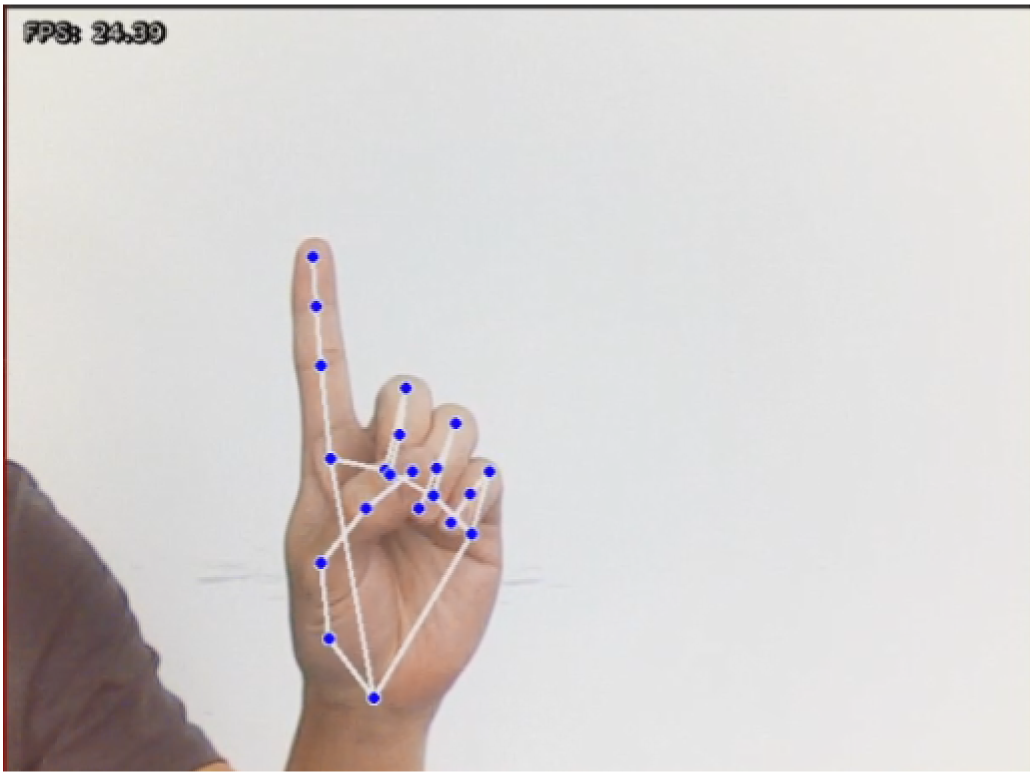
2. Startup

2.1. Program description

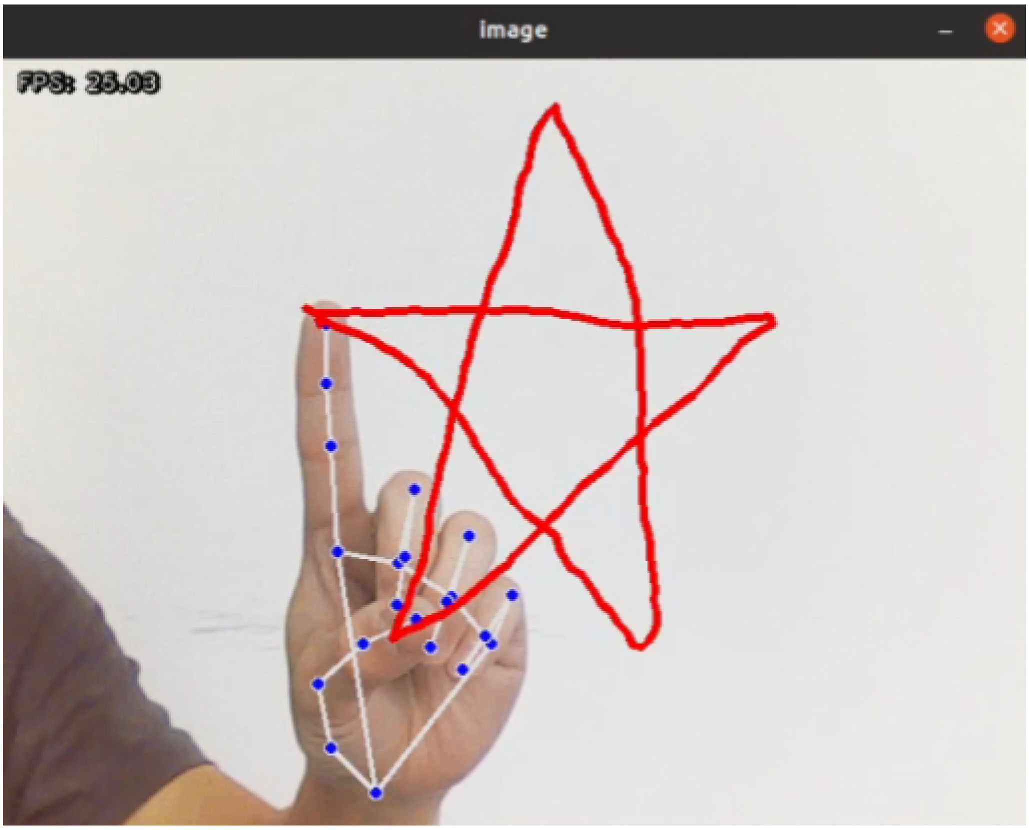
After the program is started, the camera captures the image, put your palm flat in the camera screen, open your fingers, and face the palm of your hand to the camera, similar to the gesture of the number 5, and the image will draw the joints on the entire palm. Adjust the position of your palm and try to be in the upper middle part of the screen.



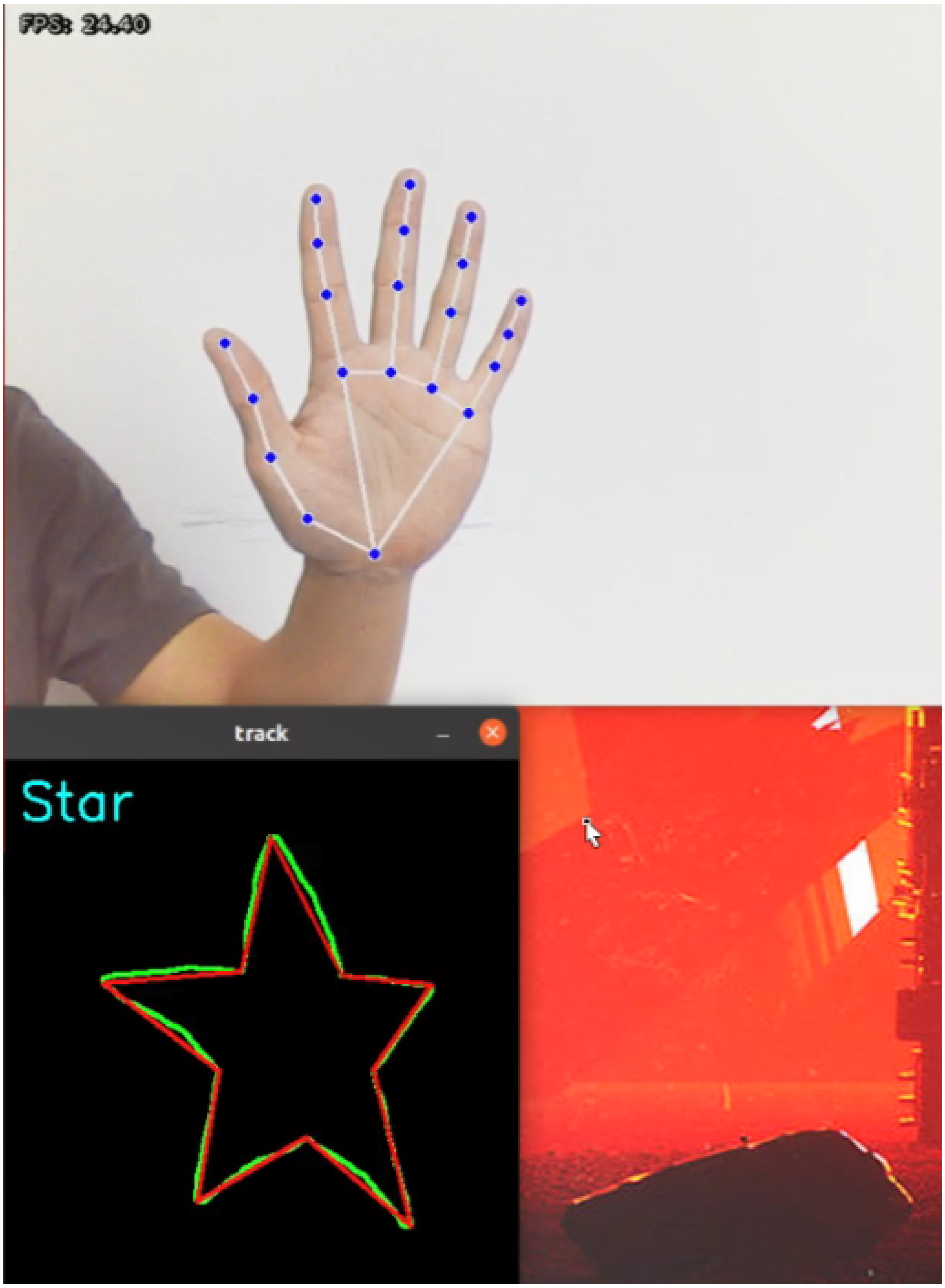
At this time, the index finger remains unchanged, and the other fingers are retracted, similar to the gesture of the number 1.



While keeping the gesture 1 unchanged, move the position of the finger, and a red line will appear on the screen to draw the path of the index finger.



When the figure is drawn, open all fingers, similar to the gesture of the number 5, and the drawn figure will be generated below.



Note: The drawn graphics need to be closed, otherwise part of the content may be missing.

Currently, there are four trajectory graphics that can be recognized, namely: triangle, rectangle, circle, and five-pointed star.

2.2, Program Startup

- Enter the following command to start the program

```
roslaunch dofbot_hand finger_trajectory.launch
```

Press the q key in the image or press Ctrl+c in the terminal to exit the program.

3, Source Code

Code path:

```
~/dofbot_ws/src/dofbot_hand/scripts/finger_trajectory.py
```

```python
#!/usr/bin/env python3
# coding: utf8
import os
import enum
import cv2
import time
import numpy as np
import mediapipe as mp
import rospy
import queue
from sensor_msgs.msg import Image
from dofbot_utils.fps import FPS
import gc
from dofbot_utils.vutils import distance, vector_2d_angle, get_area_max_contour


def get_hand_landmarks(img, landmarks):
    """
    Convert landmarks from the normalized output of medipipe to pixel coordinates
    :param img: The image corresponding to the pixel coordinates
    :param landmarks: Key points of normalization
    :return:
    """
    h, w, _ = img.shape
    landmarks = [(lm.x * w, lm.y * h) for lm in landmarks]
    return np.array(landmarks)


def hand_angle(landmarks):
    """
    Calculate the bending angle of each finger
    :param landmarks: Hand key points
    :return: The angle of each finger
    """
    angle_list = []
    # thumb 大拇指
    angle_ = vector_2d_angle(landmarks[3] - landmarks[4], landmarks[0] - landmarks[2])
    angle_list.append(angle_)
    # index 食指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[6], landmarks[7] - landmarks[8])
    angle_list.append(angle_)
    # middle 中指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[10], landmarks[11] - landmarks[12])
    angle_list.append(angle_)
    # ring 无名指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[14], landmarks[15] -
landmarks[16])
    angle_list.append(angle_)
    # pink 小拇指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[18], landmarks[19] -
landmarks[20])
    angle_list.append(angle_)
    angle_list = [abs(a) for a in angle_list]
    return angle_list


def h_gesture(angle_list):
    """
    Determine the gesture of the finger through two-dimensional features
    :param angle_list: The bending angle of each finger
    :return : Gesture name string
    """
    thr_angle, thr_angle_thumb, thr_angle_s = 65.0, 53.0, 49.0
    if (angle_list[0] < thr_angle_s) and (angle_list[1] < thr_angle_s) and
(angle_list[2] < thr_angle_s) and (
            angle_list[3] < thr_angle_s) and (angle_list[4] < thr_angle_s):
        gesture_str = "five"
    elif (angle_list[0] > 5) and (angle_list[1] < thr_angle_s) and
(angle_list[2] > thr_angle) and (
            angle_list[3] > thr_angle) and (angle_list[4] > thr_angle):
        gesture_str = "one"
    else:
        gesture_str = "none"
    return gesture_str


class State(enum.Enum):
    NULL = 0
    TRACKING = 1
    RUNNING = 2


def draw_points(img, points, tickness=4, color=(255, 0, 0)):
    """
    Draw lines connecting the recorded points on the screen
    """
    points = np.array(points).astype(dtype=np.int32)
    if len(points) > 2:
        for i, p in enumerate(points):
            if i + 1 >= len(points):
                break
            cv2.line(img, tuple(p), tuple(points[i + 1]), color, tickness)

def get_track_img(points):
    """
    Generate a black background and white line trajectory map using the recorded
points
    """
    points = np.array(points).astype(dtype=np.int32)
    x_min, y_min = np.min(points, axis=0).tolist()
    x_max, y_max = np.max(points, axis=0).tolist()
    track_img = np.full([y_max - y_min + 100, x_max - x_min + 100, 1], 0,
dtype=np.uint8)
```

```
    points = points - [x_min, y_min]
    points = points + [50, 50]
    draw_points(track_img, points, 1, (255, 255, 255))
    return track_img


class FingerTrajectoryNode:
    def __init__(self):
        rospy.init_node('finger_trajectory')
        self.drawing = mp.solutions.drawing_utils
        self.timer = time.time()

        self.hand_detector = mp.solutions.hands.Hands(
            static_image_mode=False,
            max_num_hands=1,
            min_tracking_confidence=0.05,
            min_detection_confidence=0.6
        )


        self.fps = FPS()  # fps计算器 FPS Calculator
        self.state = State.NULL
        self.points = []
        self.start_count = 0
        self.no_finger_timestamp = time.time()

        self.gc_stamp = time.time()
        self.image_queue = queue.Queue(maxsize=1)
        source_image_topic = rospy.get_param('~source_image_topic',
'/camera/color/image_raw')
        rospy.loginfo("source_image_topic = {}".format(source_image_topic))
        self.image_sub = rospy.Subscriber(source_image_topic, Image,
self.image_callback, queue_size=1)


    def image_callback(self, ros_image: Image):
        try:
            self.image_queue.put_nowait(ros_image)
        except Exception as e:
            pass

    def image_proc(self):
    #    rospy.loginfo('Received an image! ')
        ros_image = self.image_queue.get(block=True)
        rgb_image = np.ndarray(shape=(ros_image.height, ros_image.width, 3),
dtype=np.uint8, buffer=ros_image.data)
        rgb_image = cv2.flip(rgb_image, 1) # 水平翻转 Flip Horizontal
        result_image = np.copy(rgb_image)
        result_call = None
        if self.timer <= time.time() and self.state == State.RUNNING:
            self.state = State.NULL
        try:
            results = self.hand_detector.process(rgb_image) if self.state !=
State.RUNNING else None
            if results is not None and results.multi_hand_landmarks:
                gesture = "none"
                index_finger_tip = [0, 0]
                self.no_finger_timestamp = time.time()  # 记下当期时间，以便超时处理
Note the current time for timeout processing
                for hand_landmarks in results.multi_hand_landmarks:
                    self.drawing.draw_landmarks(
                        result_image,
                        hand_landmarks,
                        mp.solutions.hands.HAND_CONNECTIONS)
                    landmarks = get_hand_landmarks(rgb_image,
hand_landmarks.landmark)
                    angle_list = (hand_angle(landmarks))
                    gesture = (h_gesture(angle_list))
                    index_finger_tip = landmarks[8].tolist()

                if self.state == State.NULL:
                    if gesture == "one":  # 检测到单独伸出食指，其他手指握拳 Detect
that only the index finger is extended and the other fingers are clenched into a
fist
                        self.start_count += 1
                        if self.start_count > 20:
                            self.state = State.TRACKING
                            self.points = []
                    else:
                        self.start_count = 0

                elif self.state == State.TRACKING:
                    if gesture == "five": # 伸开五指结束画图 Stretch out your five
fingers to end drawing
                        self.state = State.NULL

                        # 生成黑白轨迹图 Generate black and white trajectory map
                        track_img = get_track_img(self.points)
                        contours = cv2.findContours(track_img,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]
                        contour = get_area_max_contour(contours, 300)
                        contour = contour[0]
                        # 按轨迹图识别所画图形
                        # cv2.fillPoly在图像上绘制并填充多边形
                        # Identify the drawn graphics according to the
trajectory diagram
                        # cv2.fillPoly draws and fills the polygon on the image
                        track_img = cv2.fillPoly(track_img, [contour,], (255,
255, 255))
                        for _ in range(3):
                            # 腐蚀函数 Corrosion function
                            track_img = cv2.erode(track_img,
cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))
                            # 膨胀函数 Dilation function
                            track_img = cv2.dilate(track_img,
cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))
                        contours = cv2.findContours(track_img,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]
                        contour = get_area_max_contour(contours, 300)
                        contour = contour[0]
                        h, w = track_img.shape[:2]

                        track_img = np.full([h, w, 3], 0, dtype=np.uint8)
                        track_img = cv2.drawContours(track_img, [contour, ], -1,
(0, 255, 0), 2)
```

```
                        # 对图像轮廓点进行多边形拟合 Polygon fitting of image contour
points
                        approx = cv2.approxPolyDP(contour, 0.026 *
cv2.arcLength(contour, True), True)
                        track_img = cv2.drawContours(track_img, [approx, ], -1,
(0, 0, 255), 2)

                        print(len(approx))
                        # 根据轮廓包络的顶点数确定图形
                        # Determine the shape based on the number of vertices of
the outline
                        if len(approx) == 3:
                            cv2.putText(track_img, 'Triangle', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
                        if len(approx) == 4 or len(approx) == 5:
                            cv2.putText(track_img, 'Square', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
                        if 5 < len(approx) < 10:
                            cv2.putText(track_img, 'Circle', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
                        if len(approx) == 10:
                            cv2.putText(track_img, 'Star', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)

                        cv2.imshow('track', track_img)

                    else:
                        if len(self.points) > 0:
                            if distance(self.points[-1], index_finger_tip) > 5:
                                self.points.append(index_finger_tip)
                        else:
                            self.points.append(index_finger_tip)

                    draw_points(result_image, self.points)
                else:
                    pass
            else:
                if self.state == State.TRACKING:
                    if time.time() - self.no_finger_timestamp > 2:
                        self.state = State.NULL
                        self.points = []

        except BaseException as e:
            rospy.logerr("e = {}".format(e))


        self.fps.update_fps()
        self.fps.show_fps(result_image)
        result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
        cv2.imshow('image', result_image)
        key = cv2.waitKey(1)

        if key == ord(' '): # 按空格清空已经记录的轨迹 Press the spacebar to clear
the recorded tracks.
            self.points = []
        if time.time() > self.gc_stamp:
            self.gc_stamp = time.time() + 1
            gc.collect()
```

```python
if __name__ == "__main__":
    finger_track_node = FingerTrajectoryNode()
    while not rospy.is_shutdown():
        try:
            finger_track_node.image_proc()
        except Exception as e:
            rospy.logerr(str(e))
```