- simple_run.launch
  - 定义地图大小、里程计话题
  - Include 这个advanced_param.xml导入主算法参数：相机姿态、加速度、深度相机内参、全局路径点

    以及调用一个节点： ego_planner_node
    - ego_planner_node.cpp
      - init初始化ros节点
      - 创建EGOReplanFSM类型的对象：rebo_replan;
        - 加载fast planner的参数
        - 创建两个定时器，
          触发回调函数 execFSMCallback
          触发回调函数 checkCollisionCallback
        - 订阅里程计话题 odom_world
        - 创建发布者1：bspline_pub_，发布话题 /planning/bspline
          发布者2：data_disp_pub_， 发布话题/planning/data_display
        - 如果rviz中指定目标，订阅消息： /waypoint_generator/waypoints
          触发回调函数 waypointCallback
          - planGlobalTraj全局规划
            - planner_manager.cpp
              - 插入中间点
              - 计算时间、路径点个数
              - minSnapTraj
              - one_segment_traj_gen
            - 显示目标点
              显示轨迹
              设置状态机
          - 如果按照预设点：按给定点进行，检查是否获得到里程计信息
            并且按照 planGlobalTrajbyGivenWps 全局规划
            - 传入给定的路径点
            - planGlobalTrajWaypoints
              - 把开始点插入路径点集合中
              - 规划时间，路径点个数
              - minSnapTraj
              - one_segment_traj_gen
            - 可视化显示
      - ros spin
  - 启动轨迹规划服务器节点：traj_server
    - main函数：
      - 初始化节点，名称 "traj_server."
        节点句柄：node
        nh("~") 句柄用于访问私有参数。
      - 创建了一个订阅者 bspline_sub，用于接收ego_planner::Bspline类型的轨迹规划消息，回调函数是 bsplineCallback。
        - 主要用来接收消息，将 B 样条轨迹信息解析并保存下来，以便后续的路径跟踪和控制。
        - 解析轨迹规划消息：轨迹上的位置点、节点（knots）信息、起始时间、轨迹ID
        - 创建轨迹对象，储存节点信息、ID、起始时间
        - 新创建的轨迹对象存储在全局向量 traj_ 中，包括位置轨迹、速度轨迹和加速度轨迹
        - 根据位置轨迹中的节点信息，计算整个轨迹的总时长，并将结果保存在全局变量 traj_duration_ 中。
        - 将全局变量 receive_traj_ 设置为 true，表示已经成功接收到一个新的轨迹，这个标志在后续的控制回调函数 cmdCallback 中会用来判断是否可以执行路径跟踪和控制。
      - 创建一个发布者pos_cmd_pub，发布 /position_cmd 话题上的 PositionCommand 类型的消息。队列长度50，可以调用发布者进行发布
      - 创建一个定时器，周期为0.01秒，定期触发 cmdCallback 回调函数。
        - 执行路径跟踪和控制的操作，并将控制指令发布到 "/position_cmd" 话题上
        - 检查轨迹是否已经接收：receive_traj_
        - 获取当前时间 time_now，然后计算 t_cur，表示当前时刻与轨迹规划的起始时间的时间差。
        - 获取当前时刻的位置、速度和加速度
          - 调用evaluateDeBoorT函数获取
          - 修改时间，t_cur 调整为 u_(p_) + t_cur，u_(p_) 可视为时间的起点
          - 继续调用evaluateDeBoor ： 递推算法
          - 通过递推算法，获取B样条在某一时刻的值
        - calculate_yaw 来计算当前时刻的航向和航向角速度。
        - 获取轨迹结束时刻的位置： 计算一个时刻 tf 等于当前时刻 t_cur 加 2.0 秒，然后获取 tf位置 pos_f，用于后续的悬停控制。
        - 判断是否到达轨迹结束： 如果当前时刻 t_cur 大于等于轨迹总时长 traj_duration_，说明已经到达轨迹结束，此时将位置设置为轨迹结束时刻的位置，并将速度和加速度设置为零，同时将航向和航向角速度设置为最后一次计算得到的值。
        - 设置控制指令的相关信息： 将上述计算得到的位置、速度、加速度、航向和航向角速度等信息填充到 cmd 消息中。
        - 发布控制指令： 通过 pos_cmd_pub.publish(cmd) 将控制指令 cmd 发布到 "/position_cmd" 主题上，这样飞行器就会根据指令执行路径跟踪和控制。
      - 初始化控制参数，具体为位置(cmd.kx)和速度(cmd.kv)的增益。这些增益可能在控制算法中使用。
      - 从参数服务器加载参数 "traj_server/time_forward"。如果找不到，则默认为 -1.0。
      - ros::Duration(1.0).sleep();：休眠1秒，可能是为了等待其他节点初始化完成。
      - ROS_WARN("[Traj server]: ready.");：打印警告消息，表示轨迹服务器已准备就绪。
      - 进入ROS Spin循环，使节点保持运行并等待接收ROS消息。节点会一直运行，直到接收到关闭信号为止。
  - 路径点生成器：waypoint_generator
    - main函数：
      - 初始化节点，名称 "waypoint_generator"
        节点句柄：node
        nh("~") 句柄用于访问私有参数。
      - 获取参数： 使用 n.param 从ROS参数服务器中获取参数值，如 "waypoint_type" 参数，用于设置路径点的类型，默认为 "manual"。
      - 创建一个名为 sub1 的ROS订阅者，订阅 "odom" 主题，队列大小为10，接收到消息后调用回调函数odom_callback
        - odom_callback回调函数：处理接收到的里程计信息
        - 全局变量 is_odom_ready 设置为 true，表示收到
        - 将接收到的里程计信息 *msg 复制给全局变量 odom
        - 获取待处理路径段的预期时间戳，即路径段的开始时间。
        - 判断当前里程计时间戳是否大于等于预期时间戳，即检查是否到达或超过了路径段的开始时间
        - 路径段赋值给 waypoints。
        - 打印一条包含路径段信息的日志，包括路径段的起始时间、每个路径点的位置和姿态信息。
        - 调用 publish_waypoints_vis 将路径点进行可视化发布。
        - 调用 publish_waypoints 将路径点发布到 "waypoints" 主题。
        - 从 waypointSegments 的前端弹出已处理的路径段。
      - 创建订阅者2，订阅 goal 话题，调用函数goal_callback
      - 创建订阅者3，订阅 traj_start_trigger 话题，有信息传入，调用回调函数 traj_start_trigger_callback ，触发路径生成
      - 路径可视化： 调用 publish_waypoints_vis 函数将路径点进行可视化发布。
      - 路径发布： 调用 publish_waypoints 函数将路径点发布到 "waypoints" 主题。
  - 使用仿真器：simulator.xml，生成地图，旋翼等等信息
    - 启动节点：quadrotor_simulator_so3
    - 启动节点：so3_control
  - 使用Rviz节点：rviz