丛林寻路系统 C ++ 实现

NWPU 自动化学院：刘振博

2021/04/23

目录

1 类的设计

1.1 姿态类

主要是为本项目做一个通用的姿态类型，便于管理.

包括各种姿态表示方法的转换 (欧拉角，四元数，旋转矩阵).

```
class Attitude
{
public:
    // Zero by default
    Attitude();
    // rotation matrix r## and origin o##
    Attitude(float r11, float r12, float r13, float o14,
             float r21, float r22, float r23, float o24,
             float r31, float r32, float r33, float o34);
    // should have 3 rows, 4 cols and type CV_32FC1
    Attitude(const cv::Mat &AttitudeationMatrix);
    // x,y,z, roll,pitch,yaw
    Attitude(float x, float y, float z, float roll, float pitch, float yaw);
    // x,y,z, qx,qy,qz,qw
    Attitude(float x, float y, float z, float qx, float qy, float qz, float qw);
    // x,y, theta
    Attitude(float x, float y, float theta);

    Attitude clone() const;

    float r11() const { return data()[0]; }
    float r12() const { return data()[1]; }
    float r13() const { return data()[2]; }
    float r21() const { return data()[4]; }
    float r22() const { return data()[5]; }
    float r23() const { return data()[6]; }
    float r31() const { return data()[8]; }
    float r32() const { return data()[9]; }
    float r33() const { return data()[10]; }

    float o14() const { return data()[3]; }
    float o24() const { return data()[7]; }
    float o34() const { return data()[11]; }

    float &operator[](int index) { return data()[index]; }
    const float &operator[](int index) const { return data()[index]; }
    float &operator()(int row, int col) { return data()[row * 4 + col]; }
    const float &operator()(int row, int col) const { return data()[row * 4 + col]; }

    bool isNull() const;
    bool isIdentity() const;
    void setNull();    void setIdentity();

    const cv::Mat &dataMatrix() const { return data_; }
    const float *data() const { return (const float *)data_.data; }
    float *data() { return (float *)data_.data; }
    int size() const { return 12; }

    float &x() { return data()[3]; }
    float &y() { return data()[7]; }
    float &z() { return data()[11]; }
    const float &x() const { return data()[3]; }
    const float &y() const { return data()[7]; }
    const float &z() const { return data()[11]; }

    float theta() const;

    Attitude inverse() const;
    Attitude rotation() const;
    Attitude translation() const;
    Attitude to3DoF() const;

    cv::Mat rotationMatrix() const;
    cv::Mat translationMatrix() const;

    void getTranslationAndEulerAngles(float &x, float &y, float &z, float &roll, float &pitch,
     float &yaw) const;
    void getEulerAngles(float &roll, float &pitch, float &yaw) const;
    void getTranslation(float &x, float &y, float &z) const;
    float getAngle(float x = 1.0f, float y = 0.0f, float z = 0.0f) const;

    Attitude interpolate(float t, const Attitude &other) const;
    void normalizeRotation();

    Attitude operator*(const Attitude &t) const;
    Attitude &operator*=(const Attitude &t);
    bool operator==(const Attitude &t) const;
    bool operator!=(const Attitude &t) const;

    Eigen::Matrix4f toEigen4f() const;
    Eigen::Matrix4d toEigen4d() const;
    Eigen::Affine3f toEigen3f() const;
    Eigen::Affine3d toEigen3d() const;

    Eigen::Quaternionf getQuaternionf() const;
    Eigen::Quaterniond getQuaterniond() const;

```

```
public:
    static Attitude getIdentity();    static Attitude fromEigen4f(const Eigen::Matrix4f &matrix
    );
    static Attitude fromEigen4d(const Eigen::Matrix4d &matrix);
    static Attitude fromEigen3f(const Eigen::Affine3f &matrix);
    static Attitude fromEigen3d(const Eigen::Affine3d &matrix);
    static Attitude fromEigen3f(const Eigen::Isometry3f &matrix);
    static Attitude fromEigen3d(const Eigen::Isometry3d &matrix);

private:
    cv::Mat data_;
};
```

1.2 栅格类

```
    class Cell
    {
    public:
        Cell();
        Cell(uint8_t class_state, float current_height);
        ~Cell();
        uint8_t Update(uint8_t class_state, float current_height);
        float GetZ(int id);
        uint8_t GetState();

    private:
        // 格子的类别, 0 表示未探索, 1 表示是路, 2 表示是障碍物, 3 表示不确定
        uint8_t state;
        float height;
        int seen_times;
        // 对比一个格子不同点云的数量来判断路和障碍物

        int obs_seentimes;
        int ground_seentimes;
        float obs_height;
        float ground_height;
    };
```

1.3 地图类

```
    enum Color
{
  COLOR_BLACK,
  COLOR_RED,
  COLOR_BLUE,
  COLOR_GREEN,   COLOR_WHITE,
  COLOR_CYAN,
  COLOR_YELLOW,
  COLOR_MAGENTA,
};
class Map
{
public:
  Map();
  ~Map();
  void init(ros::NodeHandle &nh);

  // 点云拼接融合
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr get_allmap();
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr get_allObsmap();
  bool fusion(pcl::PointCloud<pcl::PointXYZRGB>::Ptr current_pts);
  bool fusion_obs(pcl::PointCloud<pcl::PointXYZRGB>::Ptr current_pts);
  // 方格融合
  bool fusion_cell(pcl::PointCloud<pcl::PointXYZRGB>::Ptr current_pts, uint8_t class_id);

  void addOnePoint(float x, float y, float z, uint8_t class_id);
  pcl::PointCloud<pcl::PointXYZ>::Ptr get_cellMap();
  pcl::PointCloud<pcl::PointXYZ>::Ptr get_localMap();
  pcl::PointCloud<pcl::PointXYZ>::Ptr get_ObscellMap();
  int Corrd2Id(float x, float y, float z);
  std::vector<float> Id2Corrd(int id);

  int Initialization_Newground(pcl::PointCloud<pcl::PointXYZRGB>::Ptr clouds, pcl::PointCloud<
    pcl::PointXYZRGB>::Ptr obstaclesCloud);
  int Track(pcl::PointCloud<pcl::PointXYZRGB>::Ptr groundCloud);
  int Init_Clouds2Localmap(pcl::PointCloud<pcl::PointXYZRGB>::Ptr clouds, pcl::PointCloud<pcl::
    PointXYZRGB>::Ptr obstaclesCloud, int Initializing);
  int Init_Fusion2Localmap(pcl::PointCloud<pcl::PointXYZRGB>::Ptr clouds, pcl::PointCloud<pcl::
    PointXYZRGB>::Ptr obstaclesCloud);
  int Fusion2Localmap(pcl::PointCloud<pcl::PointXYZRGB>::Ptr clouds, pcl::PointCloud<pcl::
    PointXYZRGB>::Ptr obstaclesCloud);
  void Add2Globalmap();
  void SetPose(Attitude pose);
  void SetInitFlag(int flag);
  pcl::PointCloud<pcl::PointXYZ>::Ptr get_obsMap();
  pcl::PointCloud<pcl::PointXYZ>::Ptr get_unsureMap();

private:
  // 点云  fusion
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr all_pc;
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr all_pc_obs;
  // 方格fusion
  std::vector<Cell *> cellDataset_;
  //global map
  std::set<int> road_ids;
  std::set<int> obs_ids;
  std::set<int> unsure_ids;

  // local map
  std::map<int, Cell *> localmap_;
  std::set<int> localnew_id_;
  std::set<int> localupdate_id1_;
  std::set<int> localupdate_id2_;
  int InitializeFromScratch_;
  std::map<int, int> cell_olds;
  int current_old;
  int localmap_size_;

  // location
  Attitude pose_;

  double length = 30;
  double width = 30;
  double resolution = 0.2;
  int length_size;
  int width_size;
  int sum_size;
  int origin_id;
  int origin_id_x;
  int origin_id_y;
  double rd_x;
  double rd_y;
};
```

1.4 点云分割类

```


class PC_Segment
{
public:
  PC_Segment(/* args */);
  ~PC_Segment();
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr segmentCloud(
    const pcl::PointCloud<pcl::PointXYZRGB>::Ptr &cloudIn,
    const pcl::IndicesPtr &indicesIn,
    const Attitude &pose,
    const cv::Point3f &viewPoint,
    pcl::IndicesPtr &groundIndices,     pcl::IndicesPtr &obstaclesIndices,
    pcl::IndicesPtr *flatObstacles);

  void segmentObstaclesFromGround(
    const pcl::PointCloud<pcl::PointXYZRGB>::Ptr &cloud,
    const pcl::IndicesPtr &indices,
    pcl::IndicesPtr &ground,
    pcl::IndicesPtr &obstacles,
    int normalKSearch,
    float groundNormalAngle,
    float clusterRadius,
    int minClusterSize,
    bool segmentFlatObstacles,
    float maxGroundHeight,
    pcl::IndicesPtr *flatObstacles,
    const Eigen::Vector4f &viewPoint);

  void pc_init(ros::NodeHandle &nh);
  void double2float();

private:
  bool preVoxelFiltering_ = true;
  bool projMapFrame_ = false;

  bool normalsSegmentation_ = true;
  bool groundIsObstacle_ = false;
  bool flatObstaclesDetected_ = true;
  int minClusterSize_ = 4;
  int noiseFilteringMinNeighbors_ = 5;
  int normalKSearch_ = 10;

  //float
  float cellSize_ = 0.04;
  float footprintLength_ = 0.0;
  float footprintWidth_ = 0.0;
  float footprintHeight_ = 0.0;
  float minGroundHeight_ = 0.0;
  float maxObstacleHeight_ = 2.0;
  float maxGroundAngle_ = 0.785;
  float clusterRadius_ = 0.1;
  float noiseFilteringRadius_;
  float maxGroundHeight_ = 0.0;

  // double
  double cellSize = 0.04;
  double footprintLength = 0.0;
  double footprintWidth = 0.0;
  double footprintHeight = 0.0;  double minGroundHeight = 0.0;
  double maxObstacleHeight = 2.0;
  double maxGroundAngle = 0.785;
  double clusterRadius = 0.1;
  double noiseFilteringRadius;
  double maxGroundHeight = 0.0;

  // new parameter
  double _min_region = 2.0;
};
```

1.5 系统类

```

class System
{
public:
  System();
  ~System();
  void callback(const sensor_msgs::PointCloud2ConstPtr &cloudMsg);
  void Init_parameter(ros::NodeHandle &);
  void run();
  void vis_map();
  void Sent2MapHandle(pcl::PointCloud<pcl::PointXYZRGB>::Ptr groundCloud, pcl::PointCloud<pcl::
    PointXYZRGB>::Ptr obstaclesCloud);

private:
  std::string frameId_;
  std::string mapFrameId_;
  bool waitForTransform_;
  tf::TransformListener *tfListener_;
  bool mapFrameProjection_;
  double pointcloud_xu_ = 5.0;
  double pointcloud_xd_ = -5.0;
  double pointcloud_yu_ = 5.0;
  double pointcloud_yd_ = -5.0;
  double pointcloud_zu_ = 5.0;
  double pointcloud_zd_ = -5.0;

  int system_status_;
  ros::Publisher groundPub_;
  ros::Publisher localgroundPub_;
  ros::Publisher obstaclesPub_;
  ros::Publisher unsurePub_;
  ros::Publisher projObstaclesPub_;

  ros::Subscriber cloudSub_;
  // 点云切割器
  PC_Segment PC_Processor_;

  Map map_;

  const sensor_msgs::PointCloud2ConstPtr cloudMsg_;
  Attitude pose_;
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr obstaclesCloud_;
};
```

1.6 工具函数

工具函数主要

2 系统流程

2.1 坐标系

使用的坐标系为:

机体坐标系设为 B，对应 launch 文件中的 frame_id

地图坐标系设为M，对应 launch 文件中的 map_frame_id

原始点云的坐标系 P ,代码中是通过回调函数的参数(sensor_msgs::PointCloud2ConstPtr)中的 header.frame_id 来确定的.

tf 树中的坐标变化类为 tf::StampedTransform，其数据成员定义如下:

```
std::string    child_frame_id_
  // The frame_id of the coordinate frame this transform defines.
std::string    frame_id_
  // The frame_id of the coordinate frame in which this transform is defined.
ros::Time    stamp_
  // The timestamp associated with this transform.
```

其是 tf::Transform 的派生类，tf::Transform 的数据成员为:

```
Matrix3x3    m_basis    // Storage for the rotation.
Vector3    m_origin    // Storage for the translation.
```

其中的 m_basis 和 m_origin 记录的是 frame_id_ 坐标系到 child_frame_id_ 坐标系的坐标变换矩阵, 可以简写为: frame_id_ -> child_frame_id_, 也可以理解为 child_frame_id_ 坐标系下 frame_id_ 的位置(child_frame_id_ 经过什么样的旋转与平移才能转到 frame_id_).

在 ROS 系统中可以接收 tf 树中某些变换:

```
  Attitude localTransform = Attitude::getIdentity();
  try
  {
    if (waitForTransform_)
    {
      if (!tfListener_->waitForTransform(frameId_, cloudMsg->header.frame_id, ros::Time(0), ros::Duration(3)))
      {
        ROS_ERROR("Could not get transform from %s to %s after 1 second!", frameId_.c_str(), cloudMsg->header.frame_id.c_str());
        return;
      }
    }
    tf::StampedTransform tmp;
    tfListener_->lookupTransform(frameId_, cloudMsg->header.frame_id, ros::Time(0), tmp);
    localTransform = Utils_transform::transformFromTF(tmp);
```

```
}
catch (tf::TransformException &ex)
{    ROS_ERROR("%s", ex.what());
  return;
}

```

```
waitForTransform(const std::string &target_frame, const std::string &source_frame, const ros
  ::Time &time, const ros::Duration &timeout): 给定源坐标系和目标坐标系，等待两个坐标系指定
  时间的变换关系，该函数会阻塞程序运行，因此要设置超时时间。
lookupTransform(const std::string &target_frame, const std::string &source_frame, const ros::
  Time &time, StampedTransform &transform): 给定源坐标系和目标坐标系，得到两个坐标系指定时间
  的变换关系，ros::Time(0)表示我们想要得到最新一次的坐标变换。
```

所以，上面代码的 tmp 表示的就是 frameId_->mapFrameId_ 的坐标变换.

2.2 回调函数

回调函数 callback():

算法步骤:

(1) 通过 tf 树获得点云坐标系 P (cloudMsg->header.frame_id) 到机体系 B (frameId) 的坐标变换，计为 T^B_P; 通过 tf 树获得机体系 B 与地图系 M 的坐标变换，计为 T^M_B.

(2) 将 ROS 的点云格式 (const sensor_msgs::PointCloud2ConstPtr) 转换到 pcl 的固定格式 (pcl::PointCloud<pcl::PointXYZRGB>::Ptr); 并去除无效点云数据，主要利用 pcl 库的 pcl::removeNaNFromPointCloud，去除含有 NaN 的数据成员.

(3) 根据 launch 文件中的 pointcloud_xu, pointcloud_xd, pointcloud_yu, pointcloud_yd, pointcloud_zu, pointcloud_zd 参数，来对原始点云的范围进行限定，去除多余的点云. 注意: 435i 的 x 方向是正前方，y 方向是左边，z 方向朝上.

(4) 对点云进行分割，设计如下函数:

```
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud = PC_Processor_.segmentCloud(
    inputCloud,
    pcl::IndicesPtr(new std::vector<int>),
    pose,
    cv::Point3f(localTransform.x(), localTransform.y(), localTransform.z()),
    ground,
    obstacles,
   &flatObstacles);
```

具体算法流程以及工程细节见 2.3

（5）因为在分割函数中对原始点云进行了高度（launch 文件中的 mapFrameProjection 函数决定）或 row,patch 变换，所以这里将分割得到的点云坐标恢复，并且将点云的坐标投影到全局坐标系（M全局地图系）.

具体算法流程以及工程细节见 2.4

（6）将分割好的道路点云以及障碍物点云传送到建图系统：Sent2MapHandle(groundCloud, obstaclesCloud);

具体算法流程以及工程细节见 2.5

此外回调函数的调用方式使用 spinOnce 函数：

```
void System::run()
{
  // 设置rosspin的调用频率10HZ
  ros::Rate rate(10);
  bool status = ros::ok();
  while (status)
  {
    ros::spinOnce();
    status = ros::ok();
    rate.sleep();
  }
}
```

2.3 分割函数

```
pcl::PointCloud<pcl::PointXYZRGB>::Ptr PC_Segment::segmentCloud(
  const pcl::PointCloud<pcl::PointXYZRGB>::Ptr &cloudIn,
  const pcl::IndicesPtr &indicesIn,
  const Attitude &pose,
  const cv::Point3f &viewPoint,
  pcl::IndicesPtr &groundIndices,
  pcl::IndicesPtr &obstaclesIndices,
  pcl::IndicesPtr *flatObstacles)
```

计 pose 为 T^M_B，viewPoint 为 T^B_P 中的 t^B_P.

步骤：

（1）下采样（Downsampling）

使用 pcl 库的内置对象：pcl::VoxelGrid<pcl::PointXYZRGB> filter;

下采样是通过构造一个三维体素栅格，然后在每个体素内用体素内的所有点的重心近似显示体素中的其他点，这样体素内所有点就用一个重心点来表示，进行下采样的来达到滤波的效果，这样就大大的减少了数据量，特别是在配准，曲面重建等工作之前作为预处理，可以很好的提高程序的运行速度.

计此时的点云为 I，考虑到坐标系计为：^{P}I

（2）对原始点云进行高度或旋转变换.

```

float roll , pitch , yaw;
pose.getEulerAngles(roll , pitch , yaw);
cloud = Utils_transform::transformPointCloud(cloud , Attitude(0, 0, projMapFrame_ ? pose.z() :
    0, roll , pitch , 0));

pcl::PointCloud<pcl::PointXYZRGB>::Ptr transformPointCloud(
    const pcl::PointCloud<pcl::PointXYZRGB>::Ptr &cloud ,
    const Attitude &transform)
{
    pcl::PointCloud<pcl::PointXYZRGB>::Ptr output(new pcl::PointCloud<pcl::PointXYZRGB>);
    pcl::transformPointCloud(*cloud , *output , transform.toEigen4f());
    return output;
}
```

这里对原始点云进行坐标变换 Attitude(0, 0, projMapFrame_ ? pose.z() : 0, roll, pitch, 0).

经过这个变换之后，点云系的，

2.4 恢复点云坐标

2.5 基于道路点云与障碍物点云的环境重建

3 项目展示

3.1 Github 代码地址

https://github.com/liuzhenboo/mav_find_road
email:2746443306@qq.com

3.2 实际效果