高精度地图数据存储框架Lanelet2

项目介绍---引言

与传统的机器人开放场景不同，无人驾驶中的规划大多基于结构化道路，尤其是涉及到车道和交通规则。其中全局规划很依赖高精地图，没有高精地图，全局规划也无从谈起。一般来说，全局规划是规划模块的第一步，负责给下游的局部规划和控制提供全局路径。类似百度地图中的导航，全局规划算法的解决方案很成熟，一般是A*或者Dijstra等经典算法。但是无人车中，全局规划不仅仅只是搜索最短路径，还要根据高精地图丰富的信息，符合交通规则，做出更多合理的路径选择。这就显示了高精地图的重要性。高精地图方案有Opendirve，vector map，lanelet2等格式。

本次项目选用的是成熟的lanelet2的osm地图格式，方便学员快速上手和使用高精地图，实现车道级别的全局规划。

对于不熟悉高精地图，或者没有Lanelet2相关基础的同学，首先提供了一些参考资料，如下。已经熟悉的同学可以跳过本节。

> Paper：[1] Poggenhans F, Pauls J H, Janosovits J, et al. Lanelet2: A high-definition map framework for the future of automated driving[C]//2018 21st international conference on intelligent transportation systems (ITSC). IEEE, 2018: 1672-1679.
>
> 源码链接：GitHub - fzi-forschungszentrum-informatik/Lanelet2: Map handling framework for automated driving
>
> 博客资料：
>
> Lanelet2：一种面向未来自动驾驶的高精地图框架
> Lanelet2地图框架代码解析

简单介绍

Lanelet2是一个C++库，用于处理在自动驾驶情况下的地图数据。 它兼容并扩展了之前的lanelets库, 能够利用高清地图数据，以有效应对复杂交通情况下车辆所面临的挑战。 灵活性和可扩展性是应对未来地图挑战的一些核心原则。Lanelet2中的地图是自底向上（从定义车道的单个边缘开始，从车道到整个街道）定义的，通过与可观察对象相关联来验证地图上的所有信息。

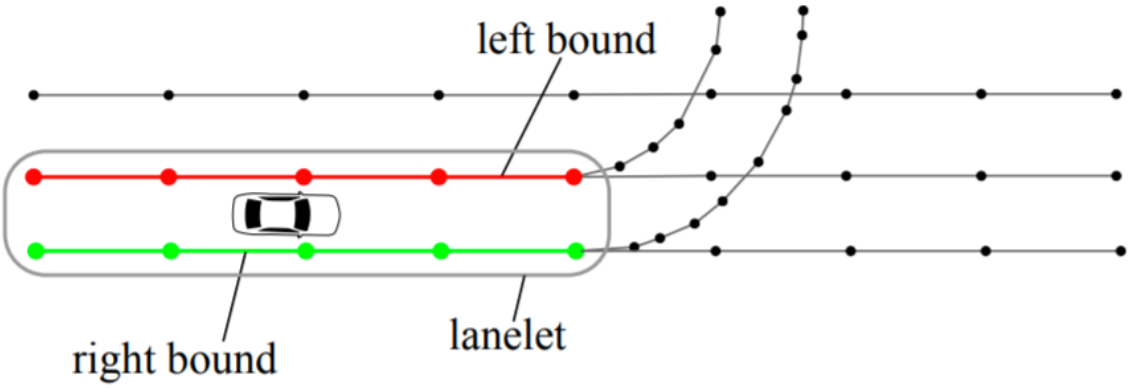
lanelet2框架下的相关内容在上面给出的论文和源码说明中介绍得十分详细，网上也有许多相关辅助参考。源码中 lanelet2_examples/ 下给出的教程较为详细地说明了增删改查各种元素、地图、路径等基本操作。

上手

建议有时间可以阅读一下GitHub上的Lanelet2 代码示例解读，这样能帮助想入门的同学快速上手。

1. Lanelet2介绍

LaneLets是自动驾驶领域高精度地图的一种高效表达方式，它以彼此相互连接的LaneLets来描述自动驾驶可行驶区域，不仅可以表达车道几何，也可以完整表述车道拓扑，同时可以集成交通规则和人的驾驶习惯。

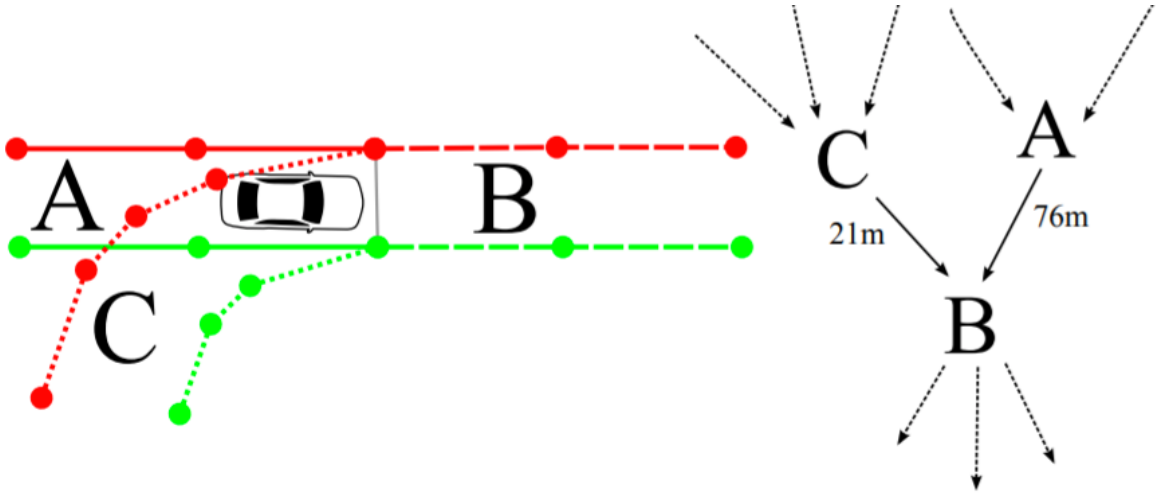


如上图所示，每个Lanelet由left bound和right bound组成，left/right bound有一系列点序列组成，因此可以以任意精度逼近任意车道形状。

1.1 用于Routing的Lanelets Graph

为了能够基于Lanelets进行路径规划，我们可以构建Lanelets邻接图结构。当Lanelets A的左右边界的终点与Lanelets B的左右边界的起点相同时，我们就称Lanelets A和Lanelets B是相邻接的。

如下图所示，图(右)是对图(左)构建的Graph，同时将每个Lanelets的长度作为Graph Edge的权重。基于该Graph，我们就可以采用Dijkstra算法，实现从任意起点到终点的路径规划。当然读者也可以给Graph Edge赋予道路边界类型、权重因子等属性，从而实现其它类型的Routing规划算法。

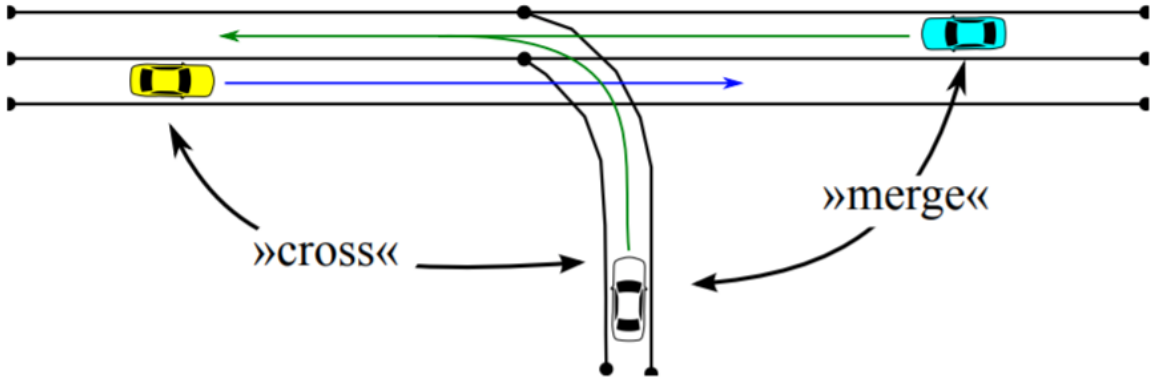


1.2 Lanelets中的交通规则

开放的公共道路上存在各种各样的交通控制要素，比如红绿灯、交通标牌等。我们将这些交通规则按照一定的方式组织起来，并关联到对应的Lanelets上。行驶在Lanelets上的车辆必须遵守该Lanelets关联的交通规则。

交通规则通常由两部分内容组成：1、规则的名称和内容；2、遵守这一规则的静态信息或者参数。举个路口红绿灯的例子，它的规则为车辆必须在交通灯为红色的时候，必须停止在路口停止线前等待；它的参数为停止线和关联交通灯的位置。

这里要特别提到是没有红绿灯的十字路口，它的通行规则必须以尽可能少的阻碍其它拥有通行权的交通参与者为准则。



交通规则表达

在实际数据中，交通规则通过"**type=regulatory element**"标识，再通过名称为maneuver的tag区分不同的交通规则。

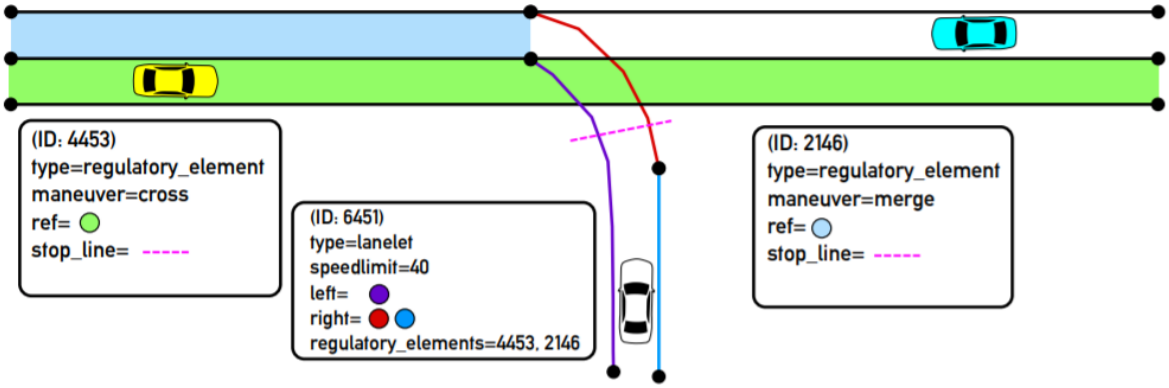
merge and cross

当maneuver=merge时，该规则的参数是：merge发生的第一个Lanelets。该规则期望车辆在进入merge的第一个Lanelets时，尽可能的与同向车道的车辆的运动速度趋同，并且保持安全距离。

当maneuver=cross时，该规则的参数是：与当前Lanelets发生cross的Lanelets，以及为了避免碰撞发生主车的停止位置。

traffic light

当maneuver=traffic light，该规则的参数是：路口的停止线和关联红绿灯的位置。该规则期望当红绿灯为红色时，车辆停止在停止线之前。



1.3 Lanelets中高效的距离计算和测量

在使用Lanelets的过程中，计算车辆Pose到Lanelets边界的距离非常重要。由于LaneLets的左右边界是由折线组成的，因此我们可以先看看单个Segment的距离如何计算。

假设单个Segment G的定义如下:

$$G = (p_b, p_t, t_b, t_t) \tag{1}$$

其中，p_b是Segment的起点，p_t是Segment的终点，t_b是起点的切向量，t_t是终点的切向量。我们可以通过*lambda*对Segment G进行插值，即

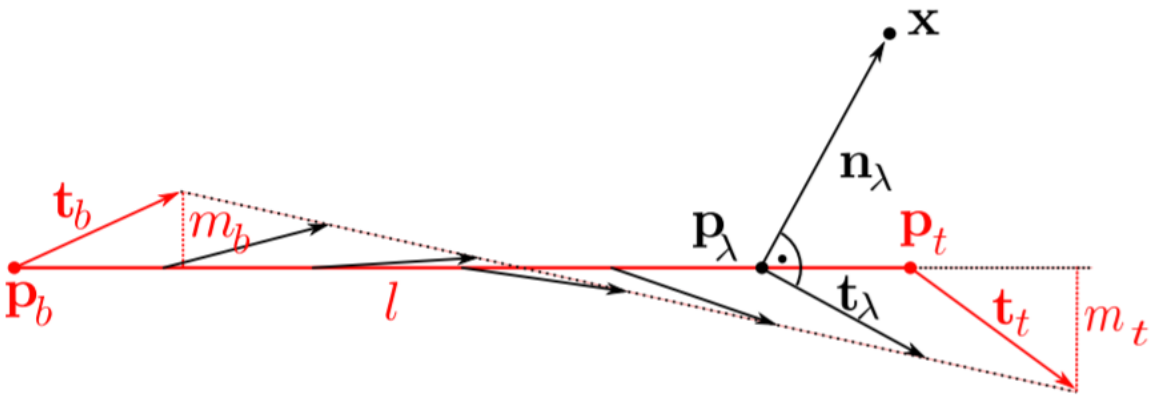
$$\begin{aligned} t_\lambda &= \lambda t_t + (1-\lambda) t_b \\ p_\lambda &= \lambda p_t + (1-\lambda) p_b \end{aligned} \tag{2}$$

点$X = (x, y)^T$到Segment G的距离定义如下:

$$d = ||n_\lambda|| = ||X - p_\lambda|| \tag{3}$$

并且n_λ满足如下约束：

$$n_\lambda^T t_\lambda = 0 \tag{4}$$



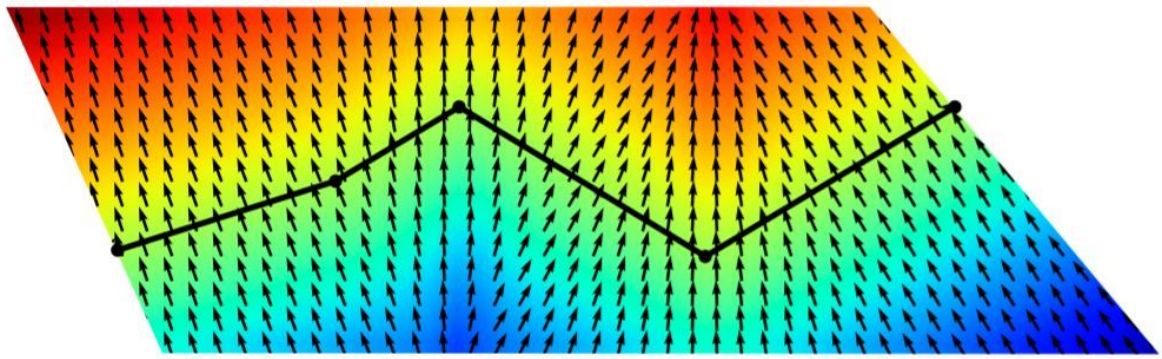
不失一般性，我们假设

$$p_b = (0,0)^T, p_t = (l,0)^T, t_b = (1,m_b)^T, t_t = (1,m_t)^T \tag{5}$$

将这些信息代入公式(2)(3)(4)，可以得到：

$$\lambda = \frac{x + y m_b}{l - y(m_t - m_b)} \tag{6}$$

也就是说，只要知道了车辆的Pose(位置和朝向)，我们就能迅速计算出车辆到边界的距离。除此之外，通过计算$\frac{n_\lambda}{||n_\lambda||}$就可以得到任意一点的梯度信息。



1.4 开源的Lanelets地图加载库-libLanelet

libLanelet使用Boost C++代码库实现，它提供了读取、加载和查询XML文件的功能；使用RTree检索查询空间要素的功能；使用诸如Dijkstra进行Routing路线规划的功能；

随着地图范围的不断扩张，Lanelets的数量会快速膨胀，为了提升数据检索的速度，libLanelets使用RTree对Lanelets进行检索，可以做到在O(logn)时间内实现对任意Object的查询。

2. lanelet2存储框架

2.1 lanelet2特点

Lanelet2地图层次划分：
• 物理层（physical layer，可观测到的真实元素）
• 关联层（relational layer，与物理层相关联的车道，区域以及交通规则）
• 拓扑层（topological layer）
设计Lanelet2地图基于以下原则：

（1）通过与可观察对象相关联来验证地图上的所有信息；

（2）地图需要覆盖到所有可能区域，包括道路外的部分；

（3）地图上各个车道和区域之间的交互作用必须是可识别和可理解的。必须能找出在哪些车道之间可能发生变道，或者在哪里可能由于交叉车道而引起冲突；

（4）必须包含有关road user使用的区域的信息以及适用于他们的交通规则；

（5）必须区分交通规则的来源及其对road user的影响；

（6）可扩展性/模块化；

（7）容易修改和更新。

Lanelet2格式基于Liblanelet已知的格式，并设计为可在基于XML的OSM数据格式上表示，该格式的编辑器和查看器可公开使用。但是，我们认为地图的实际数据格式无关紧要，并且可以互相转换，只要可以确保该格式可以无损地传输到内部表示即可， 因此能够轻松转换为其他格式。

地图存储的时候，最重要的是准确性，一般用无损地理坐标系（经纬度）。而在加载地图时，为了能够进行有效计算，将其转换为平面投影系，比如UTM系，但是高度信息也十分重要。

2.2 地图格式

Lanelet2格式基于Liblanelet已知的格式，并设计为可在基于XML的OSM数据格式上表示，该格式的编辑器和查看器可公开使用。

OSM—OpenStreetMap是lanelet2软件输出地图的标准格式。

首先，看一下OpenStreetMap的数据结构：

OpenStreetMap的元素（数据基元）主要包括三种：

（1）点（Nodes）

（2）路（Ways）

（3）关系（Relations）

这三种元素构成了整个地图画面。其中，Nodes定义了空间中点的位置；Ways定义了线或区域；Relations（可选的）定义了元素间的关系。示例为源码提供的osm地图mapping_example.osm中的部分：

1. Node
   node通过经纬度定义了一个地理坐标点。同时，还可以height=表示物体的海拔；通过layer= 和level=，可以表示物体所在的地图层与所在建筑物内的层数；通过place= and name=*来表示对象的名称。同时，way也是通过多个点（node）连接成线（面）来构成的。

```
<node id='38992' visible='true' version='1' lat='49.00345654351'
lon='8.42427590707' />
```

2. Way
   通过2-2000个点（nodes）构成了way。way可表示如下3种图形事物（非闭合线、闭合线、区域）。对于超过2000 nodes的way，可以通过分割来处理。
   Open polyline way(非闭合线)：收尾不闭合的线段。通常可用于表示现实中的道路、河流、铁路等。
   Closed polyline closed way(闭合线)：收尾相连的线。例如可以表示现实中的环线地铁。
   Area area(区域)：闭合区域。通常使用landuse=* 来标识区域等。

```
<way id='43148' visible='true' version='1'>
  <nd ref='39196' />
  <nd ref='39046' />
  <tag k='type' v='pedestrian_marking' />
</way>
```

3. Relation
   一个Relation可由一系列nodes, ways 或者其他的relations来组成，相互的关系通过role来定义。
   一个元素可以在relation中被多次使用，而一个relation可以包含其他的relation。

```
<relation id='45142' visible='true' version='1'>
  <member type='way' ref='43980' role='left' />
  <member type='way' ref='43698' role='right' />
  <tag k='location' v='urban' />
  <tag k='one_way' v='no' />
  <tag k='region' v='de' />
  <tag k='subtype' v='bicycle_lane' />
  <tag k='type' v='lanelet' />
</relation>
```

4. Tag
   标签不是地图基本元素，但是各元素都通过tag来记录数据信息。通过 'key' and 'value' 来对数据进行记录。
   例如，可以通过highway=residential来定义居住区道路；同时，可以使用附加的命名空间来添加附加信息，例如：maxspeed: winter=* 就表示冬天的最高限速。

2.3 数据结构

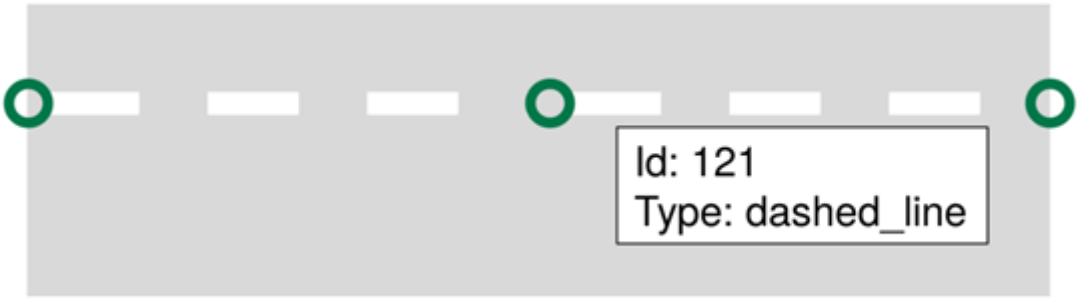
2.3.1 Points

points由ID，3d坐标和属性组成，是唯一存储实际位置信息的元素，ID必须是唯一的。其他基本元素都是直接或者间接由points组成的。在Lanelet2中，Points本身并不是有意义的对象，Points仅与Lanelet2中的其他对象一起使用有意义。

2.3.2 Linestrings

线串是两个或者多个点通过线性插值生成的有序数组，用来描述地图元素的形状。线串可以是虚线，它可以通过高度离散化实现，来描述任何一维形式，并应用于地图上的任何可物理观察到的部分。与样条曲线相比，线串可以高效计算，并且可以用来描述尖角，最终转化为非线性微分方程的求解问题。

线串必须至少包含一个点才能有效，并且不能自相交。它们不能重复包含点（即，不允许p1-> p2-> p2-> p3）。线串必须始终具有type属性，以便可以确定其用途。



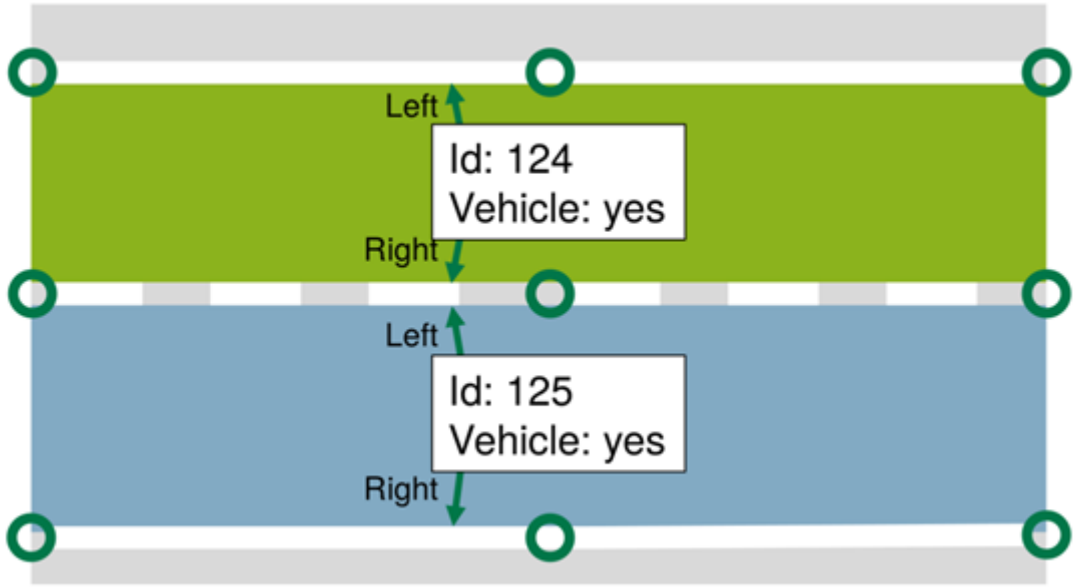
2.3.3 Polygon

多边形与线串非常相似，但会形成一个Area。假定多边形的第一个点和最后一个点以闭合形状被连接。多边形很少用于传输地图信息（交通标志除外）。相反，它们通常用作将有关区域的自定义信息添加到地图（例如感兴趣区域）的一种手段。

2.3.4 Lanelets

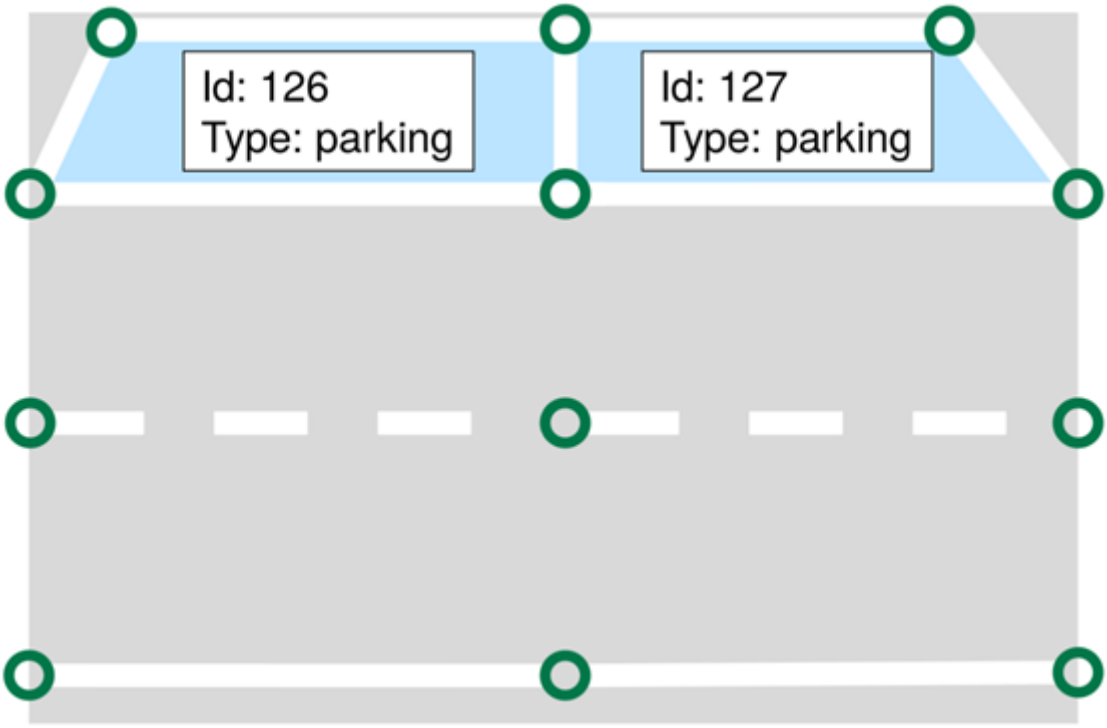
Lanelets定义了发生定向运动时，地图车道的原子部分。原子表示沿当前lanelet行驶时，有效的交通规则不会改变，而且与其他Lanelet的拓扑关系也不会更改。

lanelet可以引用表示适用于该lanelet的交通规则的regulatory elements。多个lanelet可以引用同一regElem。必须始终可以直接从车道上确定车道的当前速度限制。可以通过引用SpeedLimit监管元素或标记小车的位置来完成。在这种情况下，假定道路类型的最大速度（例如，如果位置是德国城市，则为最大50公里/小时）。



2.3.5 Areas

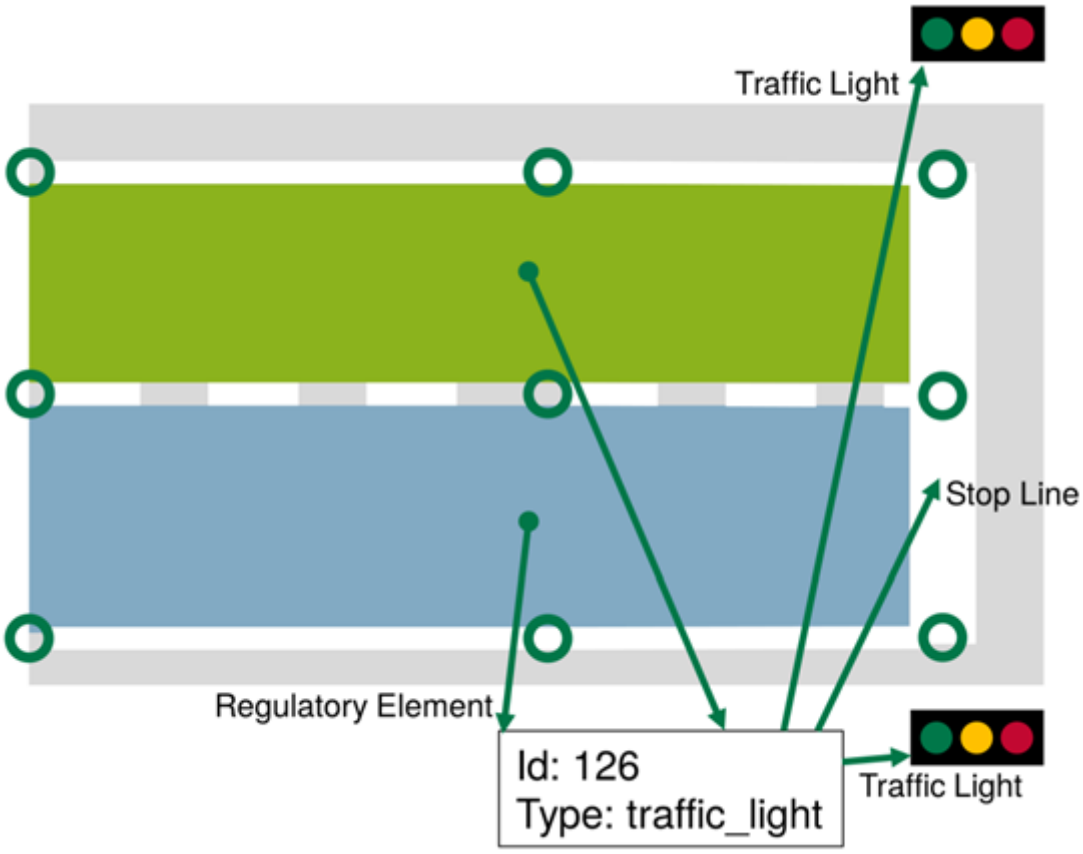
Areas是地图上没有方向或者是无法移动的部分区域，比如路标、停车位、绿化带等。他们是由一条或者多条linestring组成的闭合的区域。Area也有相关联的regulatory elements。



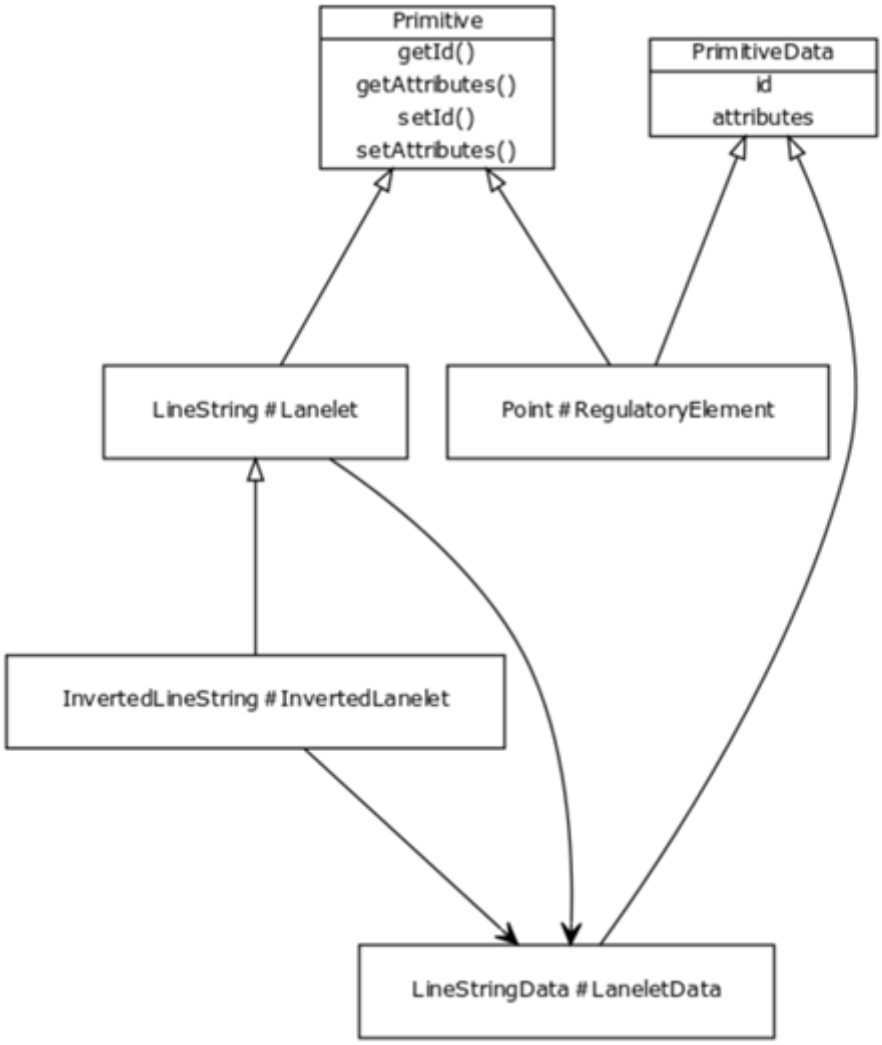
2.3.6 regElem （Regulatory elements ）

regElem是表达交通规则的通用方式，它们由适用的lanelet或Area引用。在应用的时候，regElem 会和一个或者多个Lanelets、Areas相关联。regElem是动态变化的，意味着它只是在某些条件下是有效的。诸如限速、道路优先级规则、红绿灯等，交通规则有许多不同的类型，因此每个regElem的准确结构大都不一样。他们通常引用定义规则的元素（例如交通标志），并在必要时引用取消规则的元素（例如速度区末尾的标志）。

通常，regElem元素由通常表示规则类型的标签（即交通信号灯regElem）和有关对此规则具有特定作用的可观察事物的特定信息（例如交通信号灯本身和停车线）组成。 其他类型的regElem是通行权和交通标志regElem。



2.4 软件模块



2.4.1 Core

此模块包含所有的图元和以上描述的图层，并且还包括几何计算，比如计算中心线、距离和重叠区域等。

2.4.2 Traffic Rules

根据不同的road user类型和国家，来解释相应的交通规则。

2.4.3 Physical

可以直接访问物理层元素。

2.4.4 Routing

根据交通规则，创建路由图，来确定精准的行驶路线。也可能通过组合相邻的Areas和lanelets来构建易通行区域。

2.4.5 Matching

用来给road user分配lanelets或者基于传感器的观测来确定可能的行驶方向。

2.4.6 Projection

提供全球地理坐标系到局部平面坐标系的准换。

2.4.7 IO

用于从各种地图格式（特别是OSM格式）读取和写入地图。

2.5 OSM高精度地图

OSM—OpenStreetMap是lanelet2软件输出地图的标准格式。

首先，看一下OpenStreetMap的数据结构：

OpenStreetMap的元素（数据基元）主要包括三种：点（Nodes）、路（Ways）和关系（Relations），这三种元素构成了整个地图画面。其中，Nodes定义了空间中点的位置；Ways定义了线或区域；Relations（可选的）定义了元素间的关系。

参考资料

Lanelets: 高效的自动驾驶地图表达方式