开发者说｜Apollo感知分析之跟踪对象信息融合

原创：Apollo社区开发者 Apollo开发者社区 2018-12-19

自动驾驶使用的感知类的传感器，**主要有激光雷达、毫米波雷达、摄像头、组合导航。**

> 激光雷达安装在车顶，360度同轴旋转，可以提供周围一圈的点云信息。另外，激光雷达不光用于感知，也可用在定位和高精度地图的测绘。
>
> 毫米波雷达安装在保险杠上，与激光雷达原理类似，通过观察电磁波回波入射波的差异来计算速度和距离。
>
> 组合导航分为两部分，一部分是GNSS板卡，另一部分是INS。当车辆行驶到林荫路或是建筑物附近，GPS会产生偏移或是信号屏蔽的情况，这时可通过与INS进行组合运算解决问题。

而在Apollo多传感器融合定位模块的融合框架中，

包括两部分：惯性导航解算、Kalman滤波；

融合定位的结果会反过来用于GNSS定位和点云定位的预测；

融合定位的输出是一个6-dof的位置和姿态，以及协方差矩阵。

自动驾驶中感知学习最大问题是系统对模块的要求，对准确率/召回率/响应延时等，要求很高，牵扯到安全。**如果在自动驾驶中的感知学习中，出现一些障碍物的漏检、误检，就会会带来安全问题。漏检会带来碰撞，影响事故；误检会造成一些急刹，带来乘车体验的问题。**

那么，在在进行几何计算+时序计算后，如何进行环视融合？

以下是Apollo社区开发者朱炎亮在Github-Apollo-Note上分享的**《跟踪对象信息融合》**，感谢他为我们在融合这一步所做的详细注解和释疑。

面对复杂多变、快速迭代的开发环境，只有开放才会带来进步，Apollo社区正在被开源的力量唤醒。

在上一步**HM对象跟踪**步骤中，Apollo对每个时刻检测到的Object与跟踪列表中的TrackedObject进行**匈牙利算法的二分图匹配**，匹配结果分三类：

1. 如果成功匹配，那么使用卡尔曼滤波器更新信息(重心位置、速度、加速度)等信息；
2. 如果失配，缺少对应的TrackedObject，将Object封装成TrackedObject，加入跟踪列表；
3. 对于跟踪列表中当前时刻目标缺失的TrackedObject(Object都无法与之匹配)，使用上时刻速度，跟新当前时刻的重心位置与加速度等信息(无法使用卡尔曼滤波更新，缺少观测状态)。对于那些时间过长的丢失的跟踪目标，将他们从跟踪队列中删除。

而在跟踪对象信息融合的阶段，Apollo的主要工作是，给定一个被跟踪的物体序列：

(Time_1,TrackedObject_1),

(Time_2,TrackedObject_2),

...,

(Time_n,TrackedObject_n)

每次执行LidarSubNode的回调，都会刷新一遍跟踪列表，那么一个被跟踪物体的信息也将会被刷新(重心位置，速度，加速度，CNN物体分割--前景概率，CNN物体分割--各类别概率)。N次调用都会得到N个概率分布(N个CNN物体分割--前景概率score，N个CNN物体分割--各类物体概率Probs)，我们需要在每次回调的过程中确定物体的类别属性，当然最简单的方法肯定是`argmax(Probs)`。但是CNN分割可能会有噪声，所以最好的办法是将N次结果联合起来进行判断！

跟踪物体的属性可以分为4类：

- UNKNOWN--未知物体
- PEDESTRIAN--行人
- BICYCLE--自行车辆
- VEHICLE--汽车车辆

E.g. 5次跟踪的结果显示某物体的类别/Probs分别为：

跟踪时间戳	UNKNOWN概率	PEDESTRIAN概率	BICYCLE概率	VEHICLE概率	Argmax结果
frame1	0.1	0.2	0.6	0.1	BICYCLE
frame2	0.1	0.1	0.7	0.1	BICYCLE
frame3	0.1	0.2	0.2	0.5	VEHICLE
frame4	0.1	0.2	0.6	0.1	BICYCLE
frame5	0.1	0.2	0.6	0.1	BICYCLE

如果直接对每次跟踪使用argmax(Probs)直接得到结果，有时候会有误差，上表frame3的时候因为误差结果被认为是汽车，所以需要根据前面的N次跟踪结果一起联合确定物体类别属性。Apollo使用维特比算法求解隐状态概率。这里做一个简单地描述，具体参考维特比Viterbi算法：

维特比算法前提是状态链是马尔可夫链，即下一时刻的状态仅仅取决于当前状态。(假设隐状态数量为m，观测状态数量为n)，隐状态分别为s1,s2,s3,....sm；可观测状态分别为o1,o2,...,on。 则有：

状态转移矩阵P(mxm)：P[i,j]代表状态i到状态j转移的概率。$\sum_{j=0}^m P[i,j] = 1$ 发射概率矩阵R(mxn)：R[i,j]代表隐状态i能被观测到为j现象的概率。 $\sum_{j=0}^n P[i,j] = 1$

现在假设初始时刻的m个隐状态概率为：(s0_1，s0_2，...，s0_m)，第一时刻观测到的可观察状态为ok，那么如何求第一时刻的隐状态：

1、上时刻隐状态si，第一时刻隐状态sj的联合概率为：

$$ p(s1_j, s0_i) = p(prv_state=s0_i) * P(sj|si) $$

因此可以得到p(s1_1，s0_i)，p(s1_2，s0_i)，...，p(s1_m，s0_i)，也可以得到p(s1_j，s0_1)，p(s1_j，s0_2)，p(s1_j，s0_3),...，p(s1_j，s0_m)。最终是一个mxm的联合概率矩阵。

2、同时上时刻隐状态si，第一时刻隐状态sj情况下可观测到观察状态ok的概率为：

$$ p(ok|s1_j, s0_i) = p(s1_j, s0_i) * R(ok|sj) $$

同理可得到p(ok|s1_0, s0_i), p(ok|s1_1, s0_i), p(ok|s1_2, s0_i),..., p(ok|s1_m, s0_i)，也是一个mxm的条件概率矩阵。

若最终的条件概率矩阵中p(ok|s1_jj, s0_ii) 值最大，那么就可以得出结论这个时刻的隐状态为s_jj。
Apollo也采取类似的Viterbi算法做隐状态的修正。

```
2
3   /// file in apollo/modules/perception/obstacle/onboard/lidar_process_subnode.cc
4   void LidarProcessSubnode::OnPointCloud(const sensor_msgs::PointCloud2& message) {
5     /// call hdmap to get ROI
6     ...
7     /// call roi_filter
8     ...
9     /// call segmentor
10    ...
11    /// call object builder
12    ...
13    /// call tracker
14    ...
15    /// call type fuser
16    if (type_fuser_ != nullptr) {
17      TypeFuserOptions type_fuser_options;
18      type_fuser_options.timestamp = timestamp_;
19      if (!type_fuser_->FuseType(type_fuser_options, &(out_sensor_objects->objects))) {
20        ...
21      }
22    }
23  }
24
25  /// file in apollo/modules/perception/common/sequence_type_fuser/sequence_type_fuser.cc
26  bool SequenceTypeFuser::FuseType(
27      const TypeFuserOptions& options,
28      std::vector<std::shared_ptr>* objects) {
29    if (options.timestamp > 0.0) {
30      sequence_.AddTrackedFrameObjects(*objects, options.timestamp);  // step1. add current
31      std::map<int64_t, std::shared_ptr> tracked_objects;
32      for (auto& object : *objects) {
33        if (object->is_background) {                // step2. check whether is background
34          object->type_probs.assign(static_cast<int>(ObjectType::MAX_OBJECT_TYPE), 0);
```

```
          object->type = ObjectType::UNKNOWN_UNMOVABLE;
          continue;
        }
        const int& track_id = object->track_id;
        // step3. crop sequence to generate a new sequence which length smaller or equal tha
        sequence_.GetTrackInTemporalWindow(track_id, &tracked_objects, temporal_window_);
        // step4. CRF to rectify object type use Viterbi algorithm
        if (!FuseWithCCRF(&tracked_objects)) {
          ...
        }
      }
    }
  }
```

从上述代码可以看到使用CRF进行隐状态(物体类别)修正，主要步骤为：

1. 将TrackedObject加入到sequence(sequence数据结构为map<int, map
2. 筛选，对于背景物体直接标记为UNKNOWN::UNMOVABLE类别
3. 新序列生成，sequence[track_id]里面的已存储的记录过多，而修正只需要最近一段时间的序列即可，所以使用`GetTrackInTemporalWindow`函数可以获取近期`temporal_window_`((默认20)以内的所有物体跟踪序列
4. CRF隐状态矫正，使用上述new_sequence配合Viterbi算法进行矫正

上述的难点就在于Step 4，因此本节从代码分析描述CRF修正的原理。在FuseWithCCRF函数中共分两步

- **Step1**. 状态平滑(物体状态整流)
- **Step2**. Viterbi算法推离状态

```
/// file in apollo/modules/perception/common/sequence_type_fuser/sequence_type_fuser.cc
bool SequenceTypeFuser::FuseWithCCRF(std::map<int64_t, std::shared_ptr>* tracked_objects)
  /// Step1. rectify object type with smooth matrices
  fused_oneshot_probs_.resize(tracked_objects->size());
  std::size_t i = 0;
  for (auto& pair : *tracked_objects) {
    std::shared_ptr& object = pair.second;
```

```
      if (!RectifyObjectType(object, &fused_oneshot_probs_[i++])) {
        AERROR << "Failed to fuse one shot probs in sequence.";
        return false;
      }
    }
  }

  bool SequenceTypeFuser::RectifyObjectType(const std::shared_ptr& object, Vectord* log_prob
    Vectord single_prob;
    fuser_util::FromStdVector(object->type_probs, &single_prob);
    auto iter = smooth_matrices_.find("CNNSegClassifier");

    static const Vectord epsilon = Vectord::Ones() * 1e-6;
    single_prob = iter->second * single_prob + epsilon;
    fuser_util::Normalize(&single_prob);

    double conf = object->score;
    single_prob = conf * single_prob + (1.0 - conf) * confidence_smooth_matrix_ * single_prob
    fuser_util::ToLog(&single_prob);
    *log_prob += single_prob;
    return true;
  }
```

从上述代码可以得到以下结论：

1. 原始CNN分割与后处理得到当前跟踪物体对应4类的概率为object->type_probs
2. 原始CNN分割与后处理得到当前跟踪物体前景概率为conf=object->score，那么背景的概率为1-conf
3. 平滑公式为：

   single_prob = iter->second * single_prob + epsilon single_prob = conf * single_prob + (1.0 - conf) * confidence_smooth_matrix_ * single_prob

iter->second(CNNSegClassifier Matrix)矩阵为：

0.9095 0.0238 0.0190 0.0476

0.3673 0.5672 0.0642 0.0014

0.1314 0.0078 0.7627 0.0980

0.3383 0.0017 0.0091 0.6508

confidence_smooth_matrix_(Confidence)矩阵为：

1.00 0.00 0.00 0.00

0.40 0.60 0.00 0.00

0.40 0.00 0.60 0.00

0.50 0.00 0.00 0.50

Viterbi算法推理代码如下：

```
/// file in apollo/modules/perception/common/sequence_type_fuser/sequence_type_fuser.cc
bool SequenceTypeFuser::FuseWithCCRF(std::map<int64_t, std::shared_ptr>* tracked_objects)
  /// rectify object type with smooth matrices
  ...
  /// use Viterbi algorithm to infer the state
  std::size_t length = tracked_objects->size();
  fused_sequence_probs_.resize(length);
  state_back_trace_.resize(length);
  fused_sequence_probs_[0] = fused_oneshot_probs_[0];
  /// add prior knowledge to suppress the sudden-appeared object types.
  fused_sequence_probs_[0] += transition_matrix_.row(0).transpose();
  for (std::size_t i = 1; i < length; ++i) {
    for (std::size_t right = 0; right < VALID_OBJECT_TYPE; ++right) {
      double max_prob = -DBL_MAX;
      std::size_t id = 0;
      for (std::size_t left = 0; left < VALID_OBJECT_TYPE; ++left) {
        const double prob = fused_sequence_probs_[i - 1](left) +
                            transition_matrix_(left, right) * s_alpha_ +
                            fused_oneshot_probs_[i](right);
        if (prob > max_prob) {
          max_prob = prob;
          id = left;
        }
      }
      fused_sequence_probs_[i](right) = max_prob;
      state_back_trace_[i](right) = id;
    }
  }
  std::shared_ptrobject = tracked_objects->rbegin()->second;
  RecoverFromLogProb(&fused_sequence_probs_.back(), &object->type_probs, &object->type);
  return true;
```

```
32   }
```

上述代码中，fused_oneshot_probs_是每个时刻独立的4类概率，经过平滑和log(·)处理。transition_matrix_是状态转移矩阵P，维度为4x4，经过log(·)处理，fused_sequence_probs_为Viterbi算法推理后的修正状态(也就是真实的隐状态)。

那么根据上时刻的真实的隐状态fused_sequence_probs_[i-1]和状态转移矩阵transition_matrix_，可以求解开始提到的mxm(4x4)联合状态矩阵，其中矩阵中元素fused_sequence_probs_[i][j]的求解方式为：

p(si_j, si-1_k) = p(prv_state=si-1_k) * P(sj|sk)

对应代码：

```
1   const double prob = fused_sequence_probs_[i - 1](left) +
2                       transition_matrix_(left, right) * s_alpha_ +
3                       fused_oneshot_probs_[i](right);
```

上述存在几个问题：

- **问题1： 为什么代码用的加法?**

因为代码中是将乘法转换到log(·)做运算，上述已提到transition_matrix_和fused_oneshot_probs_都是经过log(·)处理。那么上述公式等价于：

log p(si_j, si-1_k) = log p(prv_state=si-1_k) + log P(sj|sk)

只要最终将log p(si_j, si-1_k) 经过 exp(·)处理还原真实的概率即可。

- **问题2. s_alpha_是什么意思?**

有待后续深入研究，这里还不曾研究透。

- **问题3. 为什么代码会额外多乘一个概率**

p(ok|sj) -- used_oneshot_probs_[i](right)

Apollo中对于当前状态的推理就是，求解前后两个时刻关联最紧密(上时刻各状态中与当前时刻状态sj

变换概率最大的状态si)，本质就是求解开始例子中提到联合概率矩阵。在开始的例子中，我们举例隐状态s共m个。紧接着计算前后两个时刻状态的联合概率矩阵。在这个例子中，隐状态是onehot类型，也就是[0,0,0,si-1=1,0,0,0]，那么当我们的隐状态不是onehot，也就是这种类型[0.1,0.1,0.1,si-1=0.6,0,0,0.1]，那么如何求解联合概率矩阵？

做法也很类似，只需要将原先计算目标：

p(si_j, si-1_k) = p(prv_state=sk) * P(sj|si)

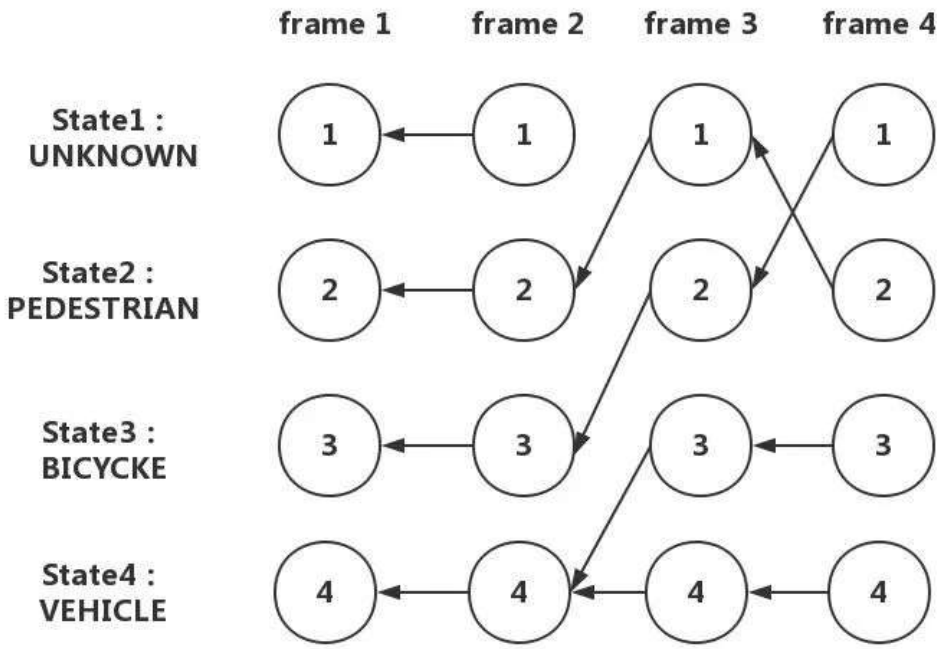
变成计算目标：

p_new(si_j, si-1_k) = p(prv_state=sk) * P(sj|si) * p(current_state=sj)

上述代码中`const double prob`就是联合概率矩阵:p_new(si_j，si-1_k)，最终取最大元素对应的(ii,jj)组，也就是(left，right)组，ii(left)就是当前时刻的隐状态：

```
1   for (std::size_t right = 0; right < VALID_OBJECT_TYPE; ++right) {  // time sequence loop
2     double max_prob = -DBL_MAX;
3     std::size_t id = 0;
4     for (std::size_t left = 0; left < VALID_OBJECT_TYPE; ++left) {  // previous state loop
5       const double prob = fused_sequence_probs_[i - 1](left) +     // each elem p[i,j] in ur
6                           transition_matrix_(left, right) * s_alpha_ +
7                           fused_oneshot_probs_[i](right);
8       if (prob > max_prob) {   // find the previous state and current hidden state which hav
9         max_prob = prob;
10        id = left;
11      }
12    }
13    fused_sequence_probs_[i](right) = max_prob; // update 2 state connected intersity
14    state_back_trace_[i](right) = id; // 2 frame's connection
15  }
```

从上述代码和注释，可以很明显的看到求解的过程，`fused_sequence_probs_`是各个时刻物体的真实修正状态，`state_back_trace_`是一条由后往前连接的最佳回溯状态链，如下图：



Apollo状态转移矩阵为：

$$\begin{matrix} 0.34 & 0.22 & 0.33 & 0.11 \\ 0.03 & 0.90 & 0.05 & 0.02 \\ 0.03 & 0.05 & 0.90 & 0.02 \\ 0.06 & 0.01 & 0.03 & 0.90 \end{matrix}$$

经过上述计算得到的`fused_sequence_probs_`矩阵就是所有时刻跟踪物体的状态概率(log(·)处理过，所以必须经过exp(·)还原概率)，**最后就是求解当前时刻，也就是sequence最后一列各类物体的概率。**

```
// get current time tracked object, last elem in sequence
std::shared_ptrobject = tracked_objects->rbegin()->second;
// post-process
RecoverFromLogProb(&fused_sequence_probs_.back(), &object->type_probs, &object->type);

bool SequenceTypeFuser::RecoverFromLogProb(Vectord* prob,
                                           std::vector<float>* dst,
                                           ObjectType* type) {
  fuser_util::ToExp(prob);     // probability log(·) format to origin format by using exp(
  fuser_util::Normalize(prob);
  fuser_util::FromEigenVector(*prob, dst);  // assign probability from origin 6 classes tc
  *type = static_cast(std::distance(dst->begin(), std::max_element(dst->begin(), dst->end(
```

```
13      return true;
14    }
15
16    void FromEigenVector(const Vectord& src_prob, std::vector<float>* dst_prob) {
17      dst_prob->assign(static_cast<int>(ObjectType::MAX_OBJECT_TYPE), 0);
18      dst_prob->at(0) = src_prob(0);
19      for (std::size_t i = 3; i < static_cast<int>(ObjectType::MAX_OBJECT_TYPE); ++i) {
20        dst_prob->at(i) = static_cast<float>(src_prob(i - 2));
21      }
22    }
```

上面代码之所以有FromEigenVector过程，主要是Apollo原始定义了6中object type：

- UNKNOWN--未知物体
- UNKNOWN_MOVABLE--未知可移动物体
- UNKNOWN_UNMOVABLE--未知不可移动物体
- PEDESTRIAN--行人
- BICYCLE--自行车辆
- VEHICLE--汽车车辆

实际只用到了0,3,4,5四类。

自Apollo平台开放已来，我们收到了大量开发者的咨询和反馈，越来越多开发者基于Apollo擦出了更多的火花，并愿意将自己的成果贡献出来，这充分体现了Apollo**『贡献越多，获得越多』**的开源精神。为此我们开设了**『开发者说』**板块，希望开发者们能够踊跃投稿，更好地为广大自动驾驶开发者营造一个共享交流的平台！

阅读原文