外部核心组件

这里的外部指不属于openpilot这个git仓库，但也都是comma.ai自己研发的开源项目（大佬们就是喜欢自己重新做更趁手的工具）

cereal

实现了一套消息通讯机制，支持single publisher multiple subscriber，早期使用ZeroMQ作为底层，后改为自己实现的msgq

```
import cereal.messaging as messaging

# in subscriber
sm = messaging.SubMaster(['sensorEvents'])
while 1:
  sm.update()
  print(sm['sensorEvents'])
```

```
# in publisher
pm = messaging.PubMaster(['sensorEvents'])
dat = messaging.new_message('sensorEvents', size=1)
dat.sensorEvents[0] = {"gyro": {"v": [0.1, -0.1, 0.1]}}
pm.send('sensorEvents', dat)
```


pub/sub 的使用例子

除了提供消息通讯外，该库封装了Cap'n Proto，把大部分系统中的涉及到交换的数据结构都用capnp定义了，如cereal/log.capnp中

```
796
797   struct ModelDataV2 {
798     frameId @0 :UInt32;
799     frameIdExtra @20 :UInt32;
800     frameAge @1 :UInt32;
801     frameDropPerc @2 :Float32;
802     timestampEof @3 :UInt64;
803   modelExecutionTime @15 :Float32;
804   gpuExecutionTime @17 :Float32;
805   rawPredictions @16 :Data;
806
807   # predicted future position, orientation, etc..
808   position @4 :XYZTData;
809   orientation @5 :XYZTData;
810   velocity @6 :XYZTData;
811   orientationRate @7 :XYZTData;
812   acceleration @19 :XYZTData;
813
814   # prediction lanelines and road edges
815   laneLines @8 :List(XYZTData);
816   laneLineProbs @9 :List(Float32);
817   laneLineStds @13 :List(Float32);
818   roadEdges @10 :List(XYZTData);
819   roadEdgeStds @14 :List(Float32);
820
```


用于在进程间交换，代表模型输出的数据的数据结构

或者cereal/car.capnp中

```
144  # ******* main car state @ 100hz *******
145  # all speeds in m/s
146
147  struct CarState {
148    events @13 :List(CarEvent);
149
150    # CAN health
151    canValid @26 :Bool;        # invalid counter/checksums
152    canTimeout @40 :Bool;      # CAN bus dropped out
153
154    # car speed
155    vEgo @1 :Float32;          # best estimate of speed
156    aEgo @16 :Float32;         # best estimate of acceleration
157    vEgoRaw @17 :Float32;      # unfiltered speed from CAN sensors
158    vEgoCluster @44 :Float32;  # best estimate of speed shown on car's instr
159
160    yawRate @22 :Float32;      # best estimate of yaw rate
161    standstill @18 :Bool;
162    wheelSpeeds @2 :WheelSpeeds;
163
164    # gas pedal, 0.0-1.0
165    gas @3 :Float32;           # this is user pedal only
166    gasPressed @4 :Bool;       # this is user pedal only
167
168    # brake pedal, 0.0-1.0
169    brake @5 :Float32;         # this is user pedal only
170    brakePressed @6 :Bool;     # this is user pedal only
171    regenBraking @45 :Bool;    # this is user pedal only
172    parkingBrake @39 :Bool;
173    brakeHoldActive @38 :Bool;
174
175    # steering wheel
176    steeringAngleDeg @7 :Float32;
177    steeringAngleOffsetDeg @37 :Float32; # Offset betweens sensors in case t
178    steeringRateDeg @15 :Float32;
179    steeringTorque @8 :Float32;        # TODO: standardize units
180    steeringTorqueEps @27 :Float32;    # TODO: standardize units
181    steeringPressed @9 :Bool;          # if the user is using the steering whe
```


用于在进程间交换，代表车辆状态信息的数据结构（car.capnp）

cereal > car.capnp
Adeeb Shihadeh, 3 weeks ago | 23 authors (George Hotz and others)

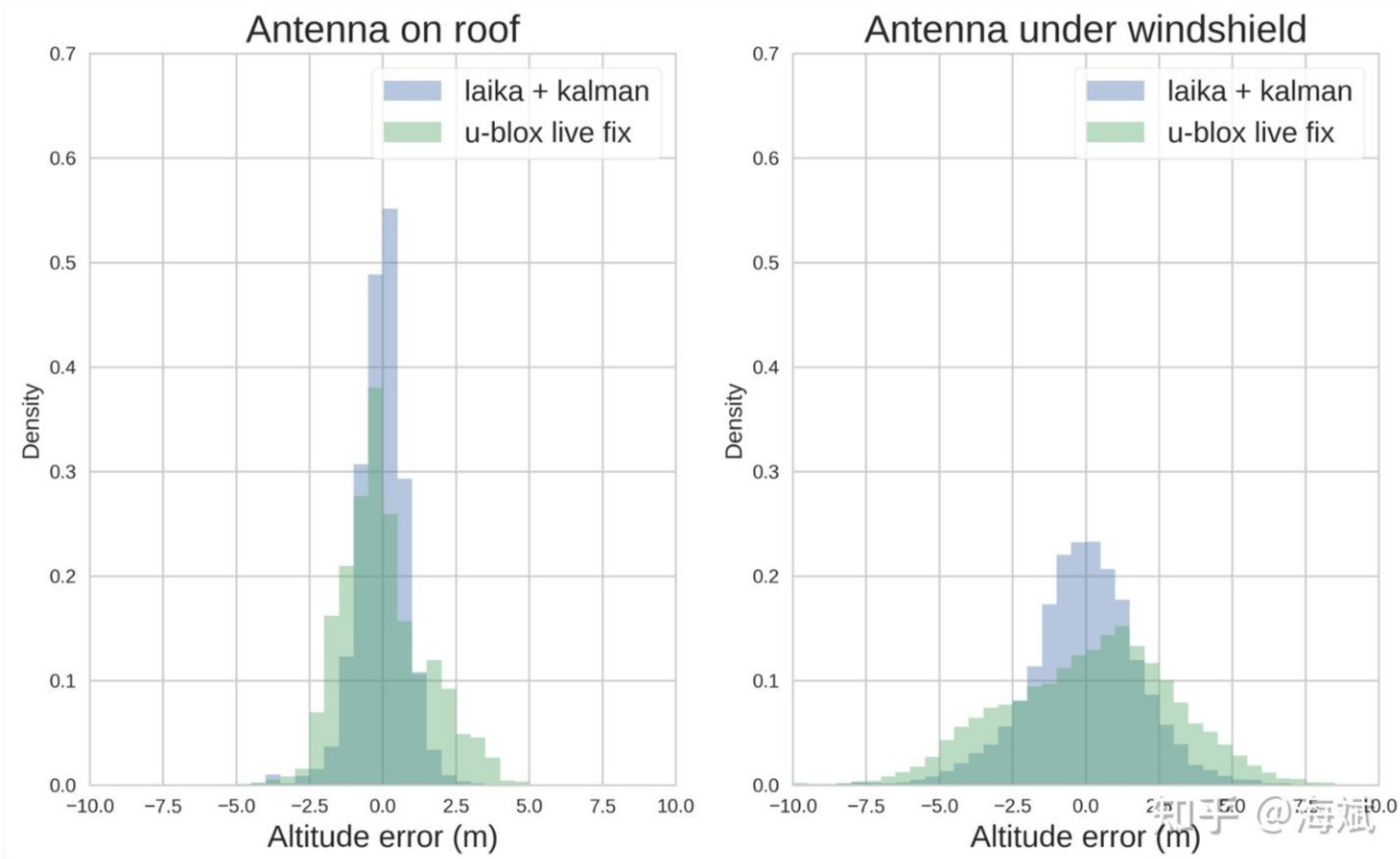
```
using Cxx = import "./include/c++.capnp";
$Cxx.namespace("cereal");

@0x8e2af1e708af8b8d;

# ******* events causing controls state machine transition **

struct CarEvent @0x9b1657f34caf3ad3 {…
}

# ******* main car state @ 100hz *******
# all speeds in m/s

struct CarState {…
}

# ******* radar state @ 20hz *******

struct RadarData @0x888ad6581cf0aacb {…
}

# ******* car controls @ 100hz *******

struct CarControl {…
}

# ****** car param ******

struct CarParams {…
}
```


car.capnp定了这些数据结构

cereal中还实现了一个VisionIpcServer的服务，通过共享内存的方式用来提供摄像头数据的多进程间的传递，减少不必要的数据拷贝带来的压力。

laika

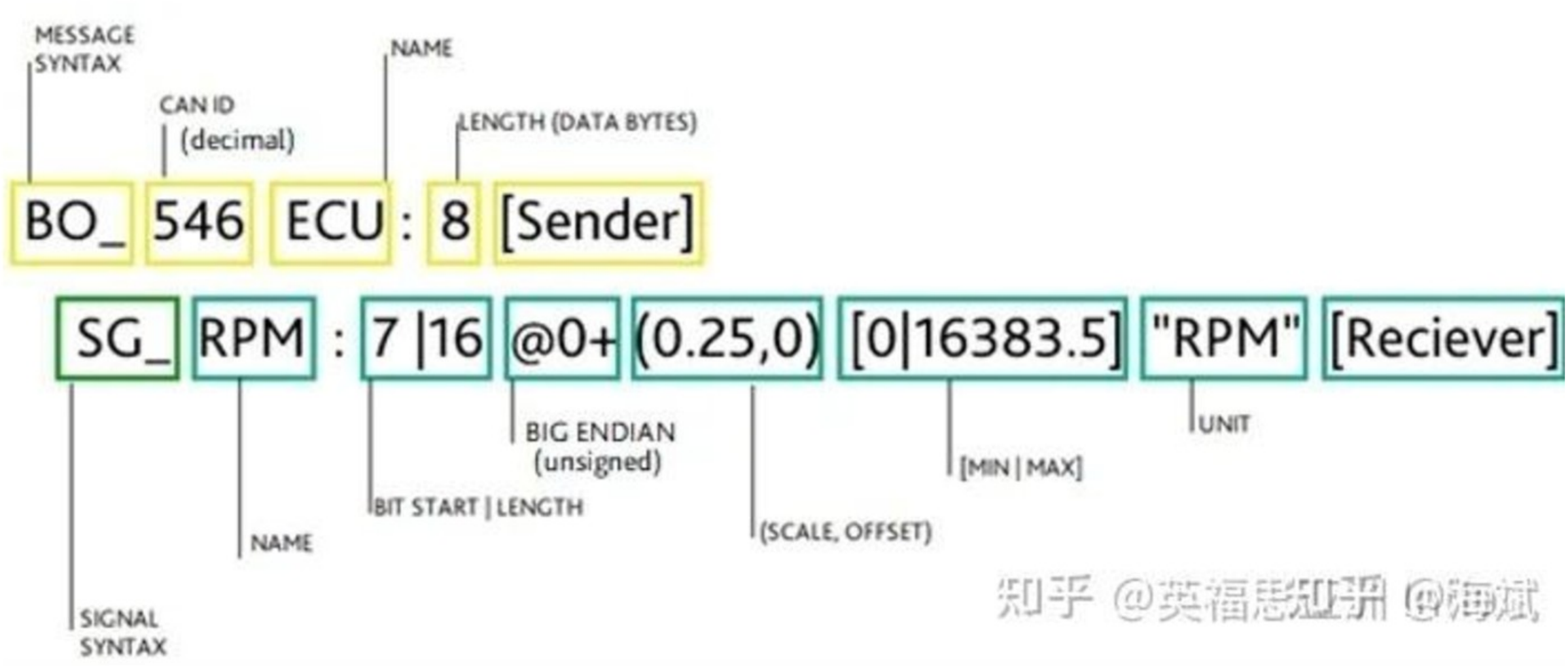
类似RTKlib，但使用python编写，处理原始GNSS数据形成高精度的位置/速度输出。测试数据表明它比u-blox GNNS芯片默认提供的定位算法的输出结果的精度要更高不少。



opendbc

DBC文件定义了每个CAN消息的具体格式，可以依据该定义进行CAN消息的解码和编码。它定义了如下内容：

- 当前信号报文的CAN ID
- 信号在CAN 报文中出现的位置
- 信号的字节顺序
- 信号的转换细节
- 信号的单位



opendbc就是comma.ai实现的一套依据DBC文件来编码和解码CAN消息的库。openpilot项目中已提供了对所支持的各种车辆的DBC文件，如opendbc/chrysler_ram_dt_generated.dbc的部分内容是：

```
BO_ 35 STEERING: 8 XXX
 SG_ STEERING_ANGLE : 5|14@0+ (0.5,-2048) [-2048|2047] "deg" XXX
 SG_ STEERING_RATE : 21|14@0+ (0.5,-2048) [-2048|2047] "deg/s" XXX
 SG_ STEERING_ANGLE_HP : 48|4@1+ (0.1,-0.4) [-0.4|0.4] "deg" XXX
 SG_ COUNTER : 55|4@0+ (1,0) [0|15] "" XXX
 SG_ CHECKSUM : 63|8@0+ (1,0) [0|255] "" XXX

BO_ 37 ECM_1: 8 XXX
 SG_ ENGINE_RPM : 7|16@0+ (1,0) [0|65535] "" XXX
 SG_ ENGINE_TORQUE : 20|13@0+ (0.25,-500) [-500|1547.5] "Nm" XXX
 SG_ EXPECTED_ENGINE_TORQUE : 36|13@0+ (0.25,-500) [-500|1547.5] "Nm" XXX
 SG_ COUNTER : 55|4@0+ (1,0) [0|15] "" XXX
 SG_ CHECKSUM : 63|8@0+ (1,0) [0|255] "" XXX

BO_ 181 ECM_TRQ: 8 XXX
 SG_ ENGINE_TORQ_MAX : 4|13@0+ (.25,-500) [-500|1547.5] "NM" XXX
 SG_ ENGINE_TORQ_MIN : 20|13@0+ (.25,-500) [-500|1547.5] "NM" XXX

BO_ 121 ESP_8: 8 XXX
 SG_ BRK_PRESSURE : 3|12@0+ (1,0) [0|1] "" XXX
 SG_ BRAKE_PEDAL : 19|12@0+ (1,0) [0|1] "" XXX
 SG_ Vehicle_Speed : 39|16@0+ (0.0078125,0) [0|511.984375] "km/h" XXX
 SG_ COUNTER : 55|4@0+ (1,0) [0|15] "" XXX
 SG_ CHECKSUM : 63|8@0+ (1,0) [0|255] "" XXX

BO_ 131 ESP_1: 8 XXX
 SG_ Brake_Pedal_State : 2|2@1+ (1,0) [0|0] "" XXX
 SG_ Vehicle_Speed : 33|10@0+ (0.5,0) [0|511] "km/h" XXX
 SG_ COUNTER : 55|4@0+ (1,0) [0|15] "" XXX
 SG_ CHECKSUM : 63|8@0+ (1,0) [0|255] "" XXX
 SG_ BRAKE_PRESSED_ACC : 6|1@0+ (1,0) [0|3] "" XXX
```

rednose

comma.ai自己实现的一套卡尔曼滤波的框架，可用在视觉里程计，多传感器融合定位，SLAM等场景。openpilot中的GNNS定位，车辆参数估计，相机外参估计等都用到了rednose。

rednose的一大特点是使用sympy来计算非线性系统状态方程的jacobian矩阵，来实现EKF，相比手动计算更不容易出错，也更方便。

主要服务进程

openpilot将不同的子功能用独立的进程来运行，进程间使用cereal提供的pub/sub消息机制来通信。这样设计的第一个好处是可以充分利用CPU的多核。每个服务进程可以根据其特点指定它的实时优先级和希望它运行在哪个核上（845上有4大核4小核）。第二个好处是，部分速度很关键的服务使用c++编写，部分速度不关键的服务使用python编写，他们之间使用pub/sub消息通信机制来配合很方便。

boardd

入口文件：selfdrive/boardd/boardd.cc

功能：使用libusb与panda设备（openpilot的配套外设）进行通讯，收发CAN总线设备的数据。

camerad

入口文件：system/camerad/main.cc

功能：摄像头处理栈，与内核通讯读取道路摄像头和司机摄像头的数据，并控制摄像头的对焦、曝光。设想图数据buffer会直接通过VisionIpcServer来申请，并将收到的结果通过VisionIpcServer传递给接收者（模型推理）。

sensord

入口文件：selfdrive/sensord/sensors_qcom2.cc

功能：读取和上报手机里的的加速度计、陀螺仪、磁力计、温度传感器、光线传感器、GPS的数据。

modeld

入口文件：selfdrive/modeld/modeld.cc

功能：通过VisionIpcClient读取摄像头数据，结合通过cereal读取订阅的的lateralPlan、navModel等的消息，调用模型推理，并把推理结果通过cereal的消息以modelV2和caeraOdometry的message名字pub出去。这里的模型推理只处理道路摄像头。考虑到摄像头采样频率设定为20HZ，该进程调用supercombo模型推理也是按照此频率。

```
// messaging
PubMaster pm({"modelV2", "cameraOdometry"});
SubMaster sm({"lateralPlan", "roadCameraState", "liveCalibration", "driverMo
```

navmodeld

入口文件：selfdrive/modeld/navmodeld.cc

功能：对navmodel进行推理，模型输入是导航的地图，输出是一个1x64的特征向量，供给modelc中的模型推理使用。

openpilot的导航相关功能中，navd进程负责调用mapbox的API产生导航路线(turn by turn)的一步步指令，mapsd进程（map renderer）负责将导航路线绘制在地图上，并通过VisionIpcServer("navd")将图像数据传出，navmodeld接下来通过VisionIpcClient("navd")取得地图图像，进行推理。

locationd/ubloxd

入口文件：selfdrive/locationd/locationd.cc，selfdrive/locationd/ubloxd.cc

功能：ubloxd负责读取u-blox芯片的数据，并通过cereal的消息机制中gpsLocationExternal消息广播出去。locationd读取gpsLocationExternal消息，对数据进行卡尔曼滤波，并将结果通过cereal的消息机制中liveLocationKalman消息广播出去。

calibrationd

入口文件：selfdrive/locationd/calibrationd.py

功能：因为用户安装comma设备时会产生yaw/pitch/roll这样的欧拉角旋转，以及车辆在行驶中产生的震动和用户点击comma设备屏幕上UI元素时的力，都会改变相机外参。所以该进程通过不断读取cameraOdometry消息，计算出相机需要校准的数据用liveCalibration消息广播出去，modeld在将摄像头图像交给模型推理前，会根据当前最新的liveCalibration中的欧拉角度信息对图像进行变换。

controlsd

入口文件：selfdrive/controls/controlsd.py

功能：该进程负责车辆控制，核心循环以100HZ进行。每轮循环第一个任务是读取所有CAN总线设备上积压的待接收消息，并依次调用相关解码模块来更新carState这个数据结构，并把该更新后的carState消息通过cereal广播出去。第二个任务是读取planerd发来的lateral和longitudinal plan，再结合更多信息并使用PID等算法，计算actuators的控制量，并调用CAN总线编码和发送，把控制信号发送给车辆硬件来执行。

plannerd

入口文件：selfdrive/controls/plannerd.py

功能：supercombo模型输出的规划轨迹并不足够好，所以该进程会对模型输出的规划进行再次处理以优化，主要是使用MPC控制算法，加入多种约束产生新的规划结果。优化后的规划结果通过longitudinalPlan和lateralPlan消息广播出去。从这里也可以看出，comma.ai宣称的端到端横向/纵向控制目前并不太准确。

radard

入口文件：selfdrive/controls/radard.py

功能：该进程主要读取ACC雷达CAN总线上待接收数据，解析成前方车辆信息后，通过radarState消息广播出去。前方车辆信息会被longitudinal plan模块所使用到。从这里也可以看出，openpilot并不是纯视觉方案，还是部分依赖原车提供的感知能力。

paramsd

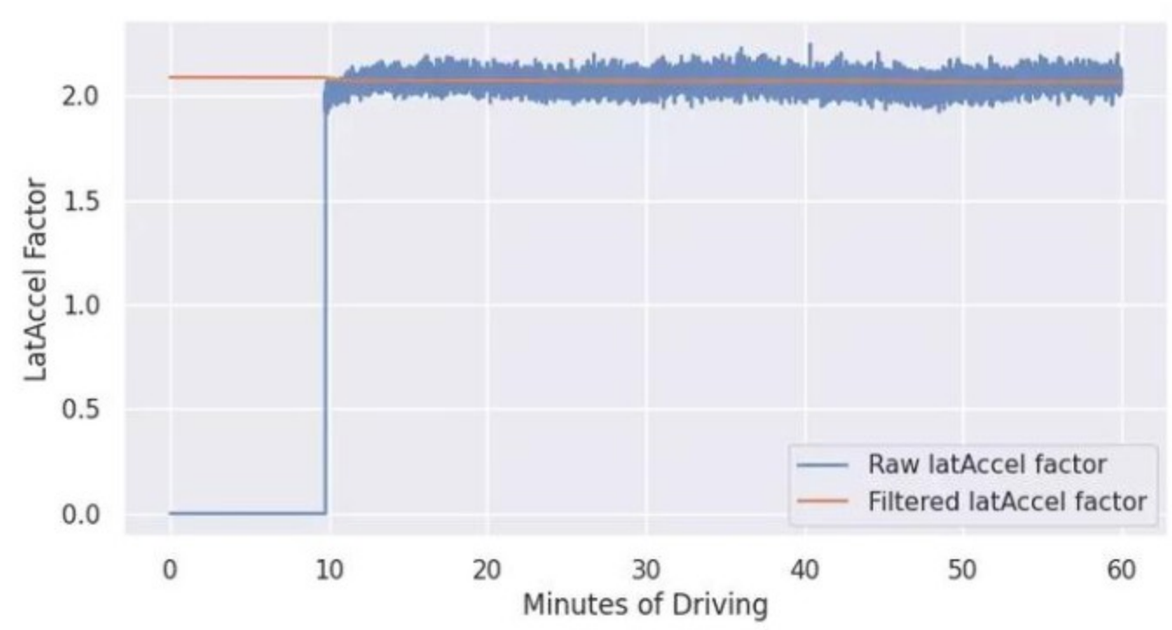
入口文件：selfdrive/locationd/paramsd.py

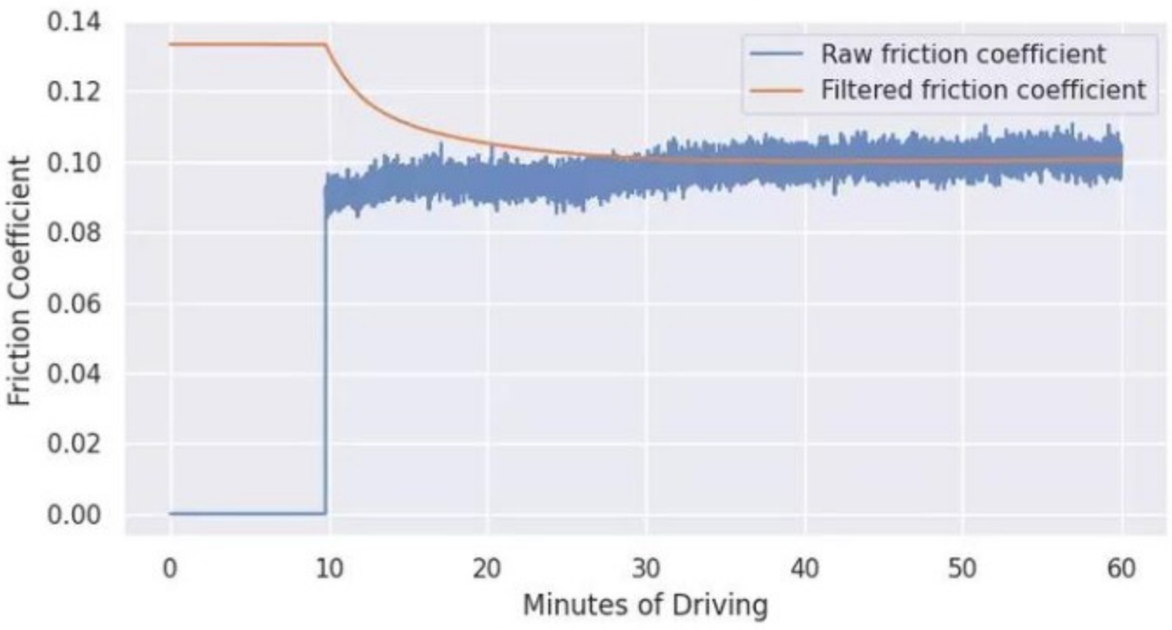
功能：利用liveLocationKalman, carState消息中的数据，用卡尔曼滤波方式估计与更新当前车辆的与控制相关的参数，如tire stiffness, steering angle offset, steer ratio，center to front，center to rear，rotational inertia 。将估计的数据用liveParameters消息广播出去。

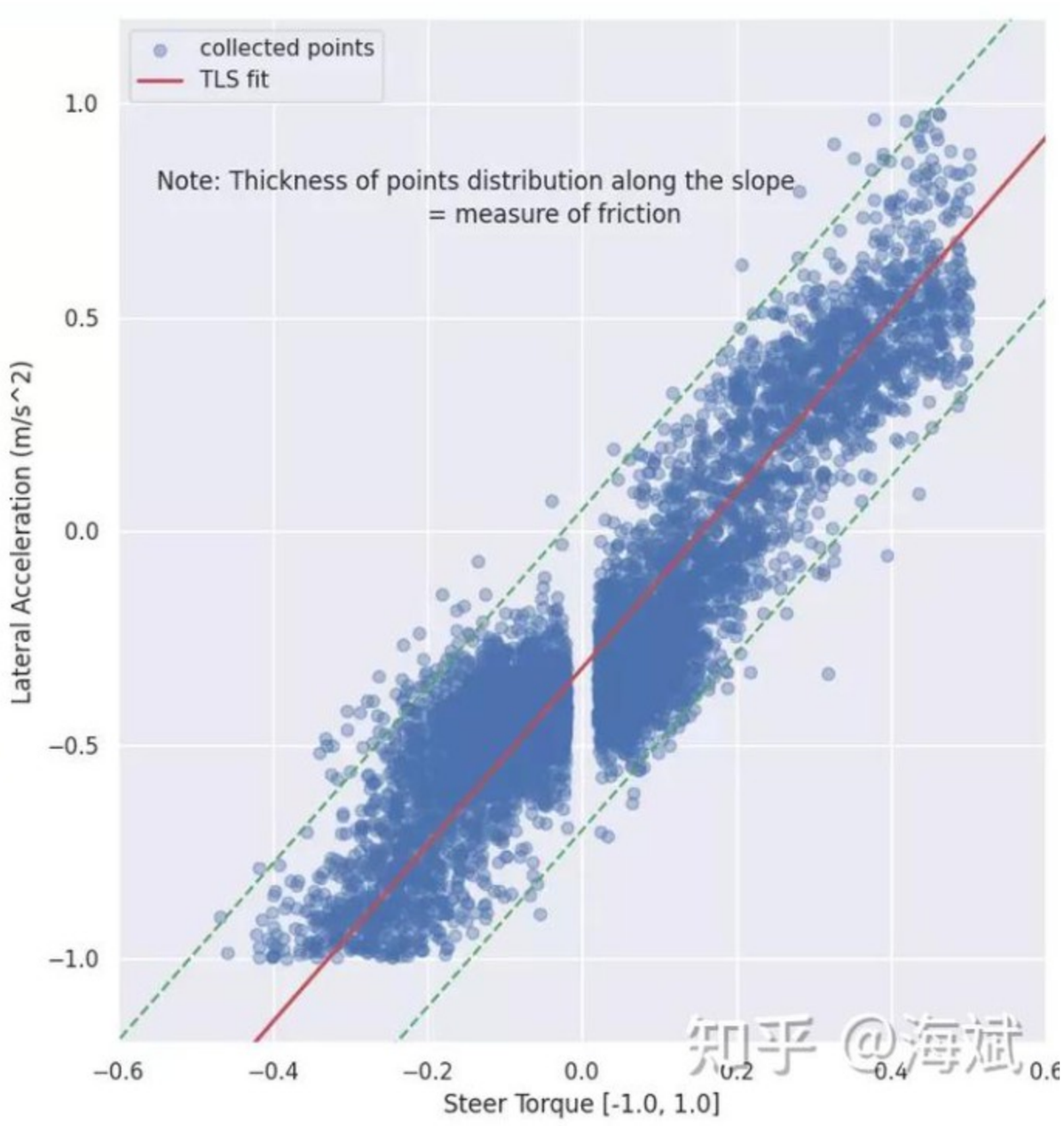
torqued

入口文件：selfdrive/locationd/torqued.py

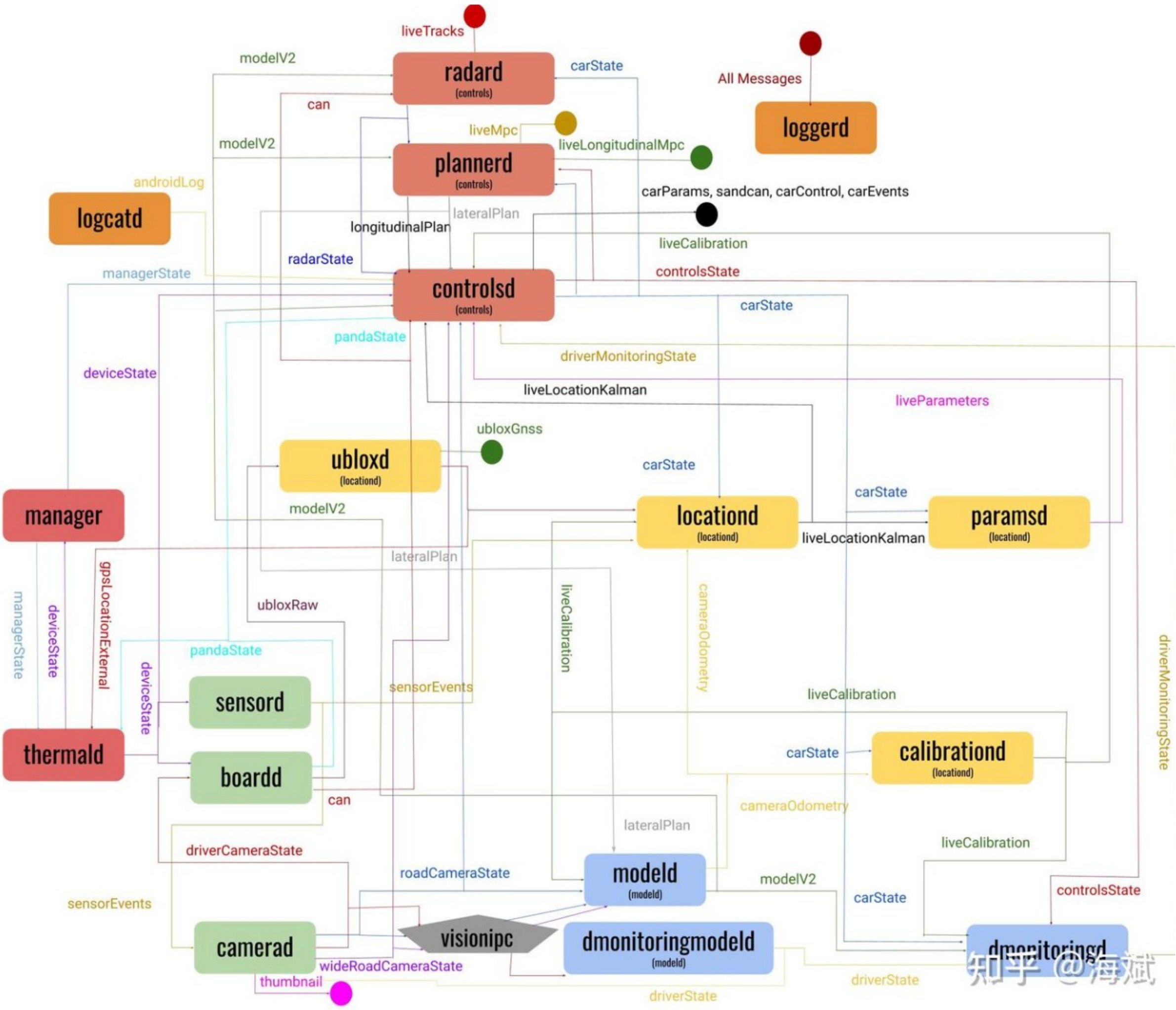
功能：一个观察是，方向盘扭矩和车辆横向加速度之间是近似的线性关系，openpilot中的控制模块会采用该线性关系来根据横向加速度需求设置方向盘扭矩。不过即使是同一款车，LatAccelFactor和FrictionCoefficient也和轮胎、胎压、路面等多种因素有关，所以使用每款车型的统计平均值并不足够好，而在每辆车上实时估计这些参数效果会更好。该进程负责实时估计车辆横向加速度和方向盘扭矩的关系，估计的结果如LatAccelFactor和FrictionCoefficient等参数使用cereal广播出去供控制模块使用。







各服务间的通讯



主要后台进程的通信，线条上的名字是消息名

数据结构

openpilot的代码中有几个关键数据结构以简写的方式大量出现在很多地方，这里进行一些罗列

- CS：CarState，定义了车辆的动态状态，如车速。其数据部主要赖CAN消息上报。
- CC：CarControl，定义了对车辆的控制操作，如油门开度，由底层解析后发送CAN消息给硬件。
- CP：CarParams，定义了车辆的一些参数，如轮胎刚度。其数据部分依赖实时的参数估计，部分依赖具体车辆的配置文件中定义。
- CI：Car Interface，他不是数据结构，但和前面几个简称常一起出现。对应的超类定义在

selfdrive/car/interface.py，每个车辆有具体的继承实现。主要用来读取和解析CAN消息，和将CarControl进行具体执行，发送CAN消息到执行硬件。