从愿景到架构：如何使用 openpilot 和 live

开放飞行员

2020 年 3 月 11 日

自动驾驶在理论上很简单，但总体上仍未得到解决。openpilot 如何解决实际行驶中出现的问题?

在我们探索 openpilot 的内部工作原理之前，让我们先做一个小小的思想实验。

我们自己的 openpilot

想象一下，我们正在为自动驾驶汽车设计非常简单的软件。为了让它对环境（其他汽车、车道分离）做出反应，我们需要能够读取传感器数据、根据这些数据采取行动并更新执行器，例如方向盘或油门。

从这个描述中，我们可以得出核心系统功能的一个非常简单的抽象实现：

```
while (true) {
  read sensor data
  compute adjustments using machine-learning model
  apply adjustments to actuators
}
```

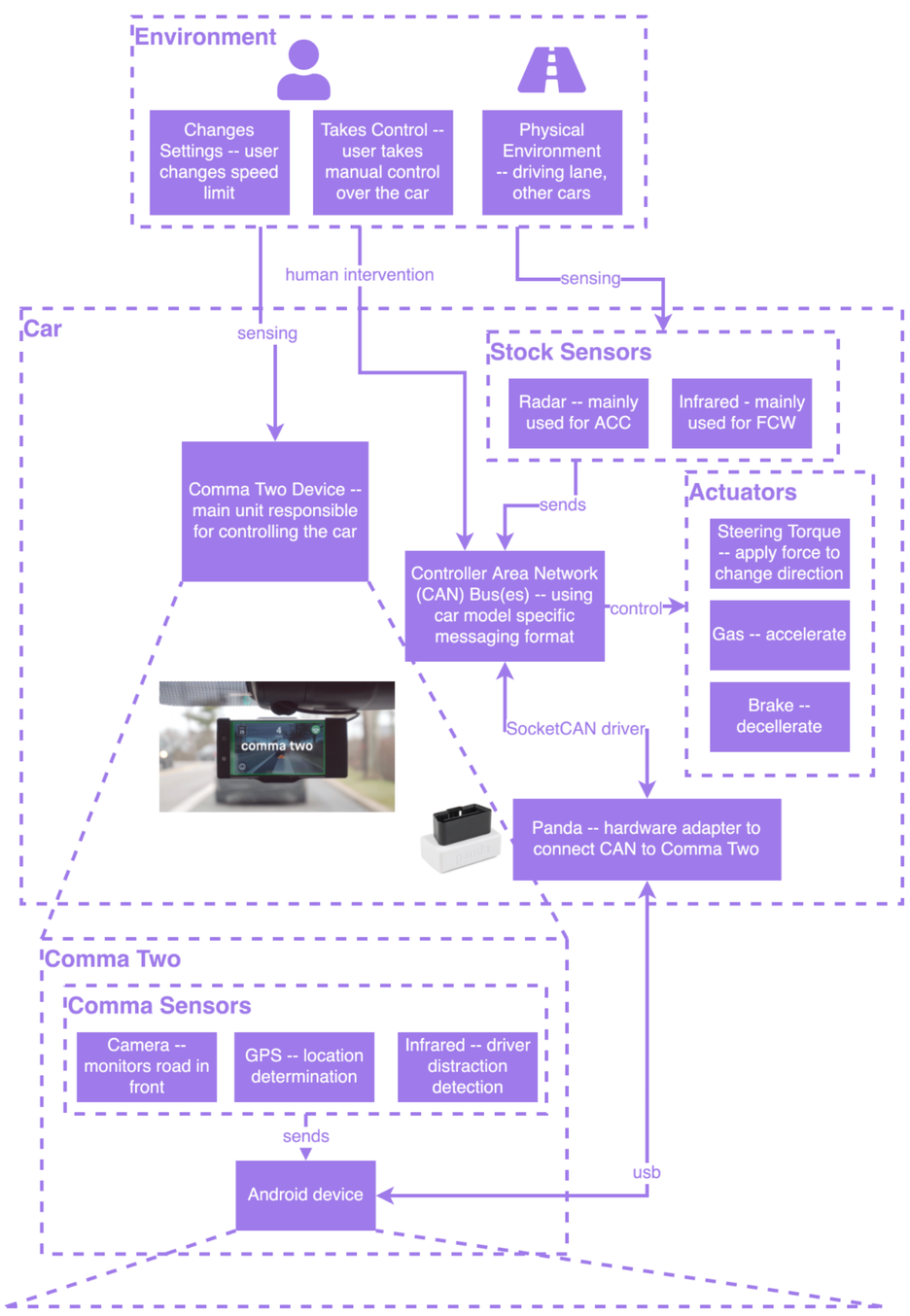
虽然这看起来过于简单，但本质上这就是 openpilot 内部的工作方式。实际上，openpilot 使用 300 多个 Python 文件（分为各种子模块、依赖项和多个硬件组件）来运行该系统。

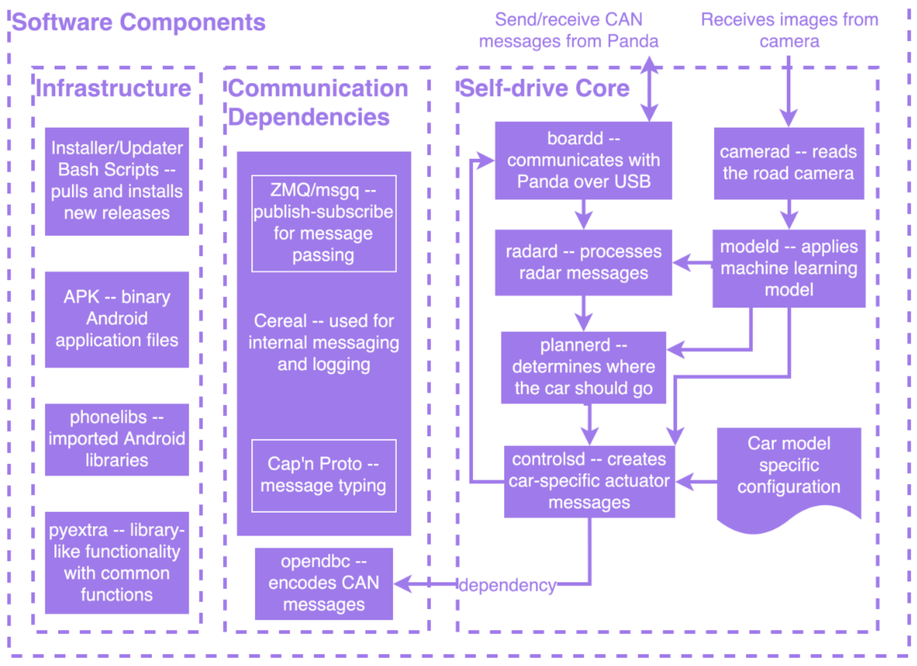
这篇文章旨在概述从传感器输入到执行器输出的路径上的主要架构组件和流程、它们的职责、组件内或组件之间应用的设计模式以及在该架构中遇到的权衡。

架构

现在我们对自主系统的工作原理有了基本的了解，让我们看看 openpilot 实际上是怎样做到的。

我们通过分析和运行GitHub 存储库中的源代码确定了 openpilot 的主要组件，并根据该分析创建了一个简洁明了的图表。如果这张图看起来太复杂，请不要担心，在接下来的部分中，我们将尽力解释每个组件的含义和作用。鼓励您（读者）在文本和此图之间来回切换，以充分了解 openpilot 的内部工作原理。





架构图是运行时视图和开发视图的组合，因为它们是相关的，并且最好在这个系统中一起解释。没有讨论逻辑视图，因为系统主要执行单一任务（控制您的汽车），并且没有太多的最终用户功能。也没有讨论流程/部署视图。这是因为 comma.ai 的部署程序和云基础设施的某些部分不是开源的，导致分析不完整。

多个传感器可观察汽车内部和周围的环境。制造商提供的常规传感器包括雷达和红外传感器。Comma Two 配备前置摄像头、GPS 和红外驾驶员注意力分散摄像头。这些输入统一起来以提供所需的功能。

该图传达了 openpilot 开发人员做出的另一个重要架构决策：openpilot 不是单个整体流程，而是协调的流程集群。能够优先处理某些流程，使 openpilot 能够遵守自动驾驶所需的严格时间要求。

现在，我们将通过分析两种情况来理清图表底部的所述过程：

1. 一辆车在我们前面刹车
2. 道路是弯曲的

在下文中，我们将逐步探讨这些场景，解释软件组件是什么、它们的职责以及它们如何通信。

场景 1：领头车刹车/减速

现在想象一下你前面有一辆车突然刹车。配备 ACC 的汽车会注意到这一点，其雷达会向 CAN 总线发送一条消息。CAN 总线由 comma.ai 制造的硬件设备 panda 持续*读取*。

panda 将此信息转发给 openpilot 的comma two。此 Android 设备运行多个后台进程（总共约 20 个），它们相互通信并协调其操作。

所有发送至和来自 panda 的消息均由*boardd*后台进程管理。boardd 将这些消息发布给其他进程使用。

*radard*正在密切监听 boardd 发布的消息，并且会注意到来自汽车雷达系统的消息。radard 将 boardd 发布的雷达 CAN 消息与 modeld 发布的数据（参见下一部分）相结合，并发布一条`radarState`消息，其中包含前车距离等信息。

然后，plannerd `radarState`会接收该消息。plannerd 的职责包括（正如其名称所传达的）规划汽车需要行驶的路线。它会发布这条路线，然后其他进程就可以使用它。

此场景涉及的最后一个过程是*controlsd*。还记得本文开头的小伪代码循环吗？该 while 循环几乎逐字逐句地在 controlsd 中实现。controlsd 将 plannerd 发布的路径转换为实际的 CAN 消息，并以与汽车型号无关的方式进行。这是一个重要的细节，因为 openpilot 旨在支持尽可能多的汽车，而让 controlsd 使用接口而不是具体实现有助于实现这一愿景。增加对新车型的支持主要涉及实现 controlsd 使用的接口的新具体实现。

最后，controlsd 将 CAN 消息发送回 boardd，然后 boardd 通过 panda 将其写入汽车的 CAN 总线。

场景 2：方向改变/道路弯曲

高速公路也有弯道。openpilot 如何感知线标并根据弯道采取行动？让我们来探究一下。

这次分析的切入点不是 boardd，而是*camerad*。从表面上看，camerad 是一个非常简单的进程，其唯一职责就是发布来自后置和前置摄像头的视频帧。然后其他进程就可以使用这些帧。

对这些帧特别感兴趣的一个过程是*modeld*。*modeld*使用其机器学习模型转换相机帧并发布结果。

从此时起，将涉及与场景 1 中相同的过程。即，plannerd 监视 modeld 的输出并将其纳入其路径规划中。plannerd 发出编码为 CAN 消息的方向盘扭矩变化，然后由 boardd 接收并发送到汽车的 CAN 总线。

基础设施组件

在上述两个场景中，我们访问了 openpilot 中可以说是最重要的流程。然而，它们并不是 openpilot 生态系统中唯一的组件。

与核心功能不直接相关的其他组件并不是那么有趣，因此我们不会花太多篇幅来介绍它们：

- 一组 bash 脚本用于远程提取和安装新版本。
- 用户需要安装的附加 APK 文件位于 APK 文件夹中。
- Phonelibs 包含用于与 Android 通信的 Comma Two 上的库。
- Pyextra 包含多个组件使用的类似库的功能，包括明确的异常消息。

将其与代码结合起来

有了上述架构概述，我们可以研究实现它的过程。因此，请站在自动驾驶系统软件架构师的角度，思考系统必须遵守的要求：

- 原子性（第 6.3 章）在信号处理方面。因为我们处理的数据最终会实际移动汽车，所以模棱两可的指令是不能容忍的。
- 兼容且可扩展。由于 comma.ai 的愿景包括为所有流行汽车提供支持，因此代码也应为此进行设计。
- 通用。当您的产品依赖于开源社区时，应用软件设计中的常见思想会很有帮助。
- 模块化（第 3.5 章）。当更改或改进系统中的某些部分时，您不希望触及整个代码库。此外，模块化允许独立测试各个模块，并允许使用多种编程语言或协议。

Openpilot 满足了这些要求，并为我们提供了实施设计模式的良好示例，其中最值得一提的是发布和订阅。该模式在自动驾驶系统中起着基础性作用，并强制执行上面列出的所有内容。简而言之，发布和订阅是一种协调不同软件系统或组件之间消息传递的系统。发布者仅发送消息，而订阅者仅接收来自其订阅的发布者的消息。

当应用程序与其他应用程序异步通信时，发布-订阅特别有用，这些应用程序不一定以相同的语言实现，也不一定在同一平台或系统上执行。此外，在每个组件中，它允许从可用的发布者中挑选。这确保关键信息只存在于至关重要的地方，并引入了可扩展性。

总体可扩展性

除了发布和订阅模式之外，openpilot 还采用了软件工程中的其他最佳实践，以实现兼容性和灵活性。示例包括使用基类和接口。我们可以将这些现象与产品愿景联系起来，该产品愿景旨在提供 由社区构建和维护的分布式去中心化自动驾驶汽车平台。从部署的角度来看，cereal、opendbc 和 panda 将在 2020 年成为独立项目。结合新提出的开发流程，这应该会让开源社区的人们非常有兴趣做出贡献并创建一个以质量和社区为关键词的系统。

权衡

我们已经看到了系统的一些功能属性，现在我们将看看一些非功能属性及其权衡。

性能与隐私

George Hotz 在 2019 年 6 月的一次演讲中提到，他认为“驾驶的定义是人们在驾驶时所做的事情”，这意味着驾驶助手应该基于驾驶员提供的数据。正如架构图中所提到的和看到的，系统的关键部分之一是建模的驾驶模型。由于驾驶模型在系统中起着至关重要的作用，因此它需要准确和一致，以防止发生潜在的事故。该模型所需的数据直接来自在汽车中使用 openpilot 系统的驾驶员，这引发了数据隐私问题。有些人可能不希望他们的汽车数据进入 comma.ai。然而，在 comma.ai，他们对数据隐私非常清楚，如果你使用他们的系统，你就同意他们使用驾驶时生成的所有数据，这也是 George Hotz 在上述演讲中所说的。因此，他们在模型性能和数据隐私之间进行了权衡，他们更看重性能和准确性，而不是驾驶员的隐私。

兼容性与可维护性

正如场景 1 结尾处所提到的，openpilot 旨在支持尽可能多的汽车。因此，以与所有不同汽车和车型兼容的方式构建系统非常重要。如架构图所示，并在场景 1 中提到，openpilot 通过创建 controlsd 使用的接口的新实现来处理所有这些不同的汽车和车型。但是，创建所有这些文件也意味着需要单独维护它们。随着 openpilot 支持的汽车数量不断增加，这意味着他们最终需要维护数百甚至数千个这样的接口实现。这意味着公司选择在兼容性和可维护性之间做出权衡，创建一个具有高兼容性的系统，但反过来会失去一些可维护性。

总结

最初只是一个简单的 while 循环，最后却变成了一场跨越流程、模块和设计原则的旅程。我们看到了发布-订阅如何创建模块化、可扩展的设计。我们探讨了开发人员多年来做出的权衡及其影响。

openpilot 是一个非常复杂的项目，旨在解决一个非常复杂的问题。