OCS2 优化控制工具箱（二） 控制框架 汉化

一、前言

简要说明为解决具体问题而应用 OCS2 所需的步骤。根据安装页面的描述，我们已经假定您已经成功安装了程序库。

一、从何处开始?

第一步，也许也是最重要的一步，是决定要针对手头的问题使用哪种复杂模型。例如，对于您计划使用 MPC 解决的任务，运动学模型是否足够？一些常见的模型如下

运动学模型（Kinematic model）： 这种模型通常用于固定或移动基座机械手。该问题的目标是：末端执行器跟踪、避免自碰撞和无碰撞导航。有关示例，请参阅移动机械手示例。https://leggedrobotics.github.io/ocs2/robotic_examples.html#doxid-ocs2-doc-robotic-examples-mobile-manipulator

动力学模型（Dynamic model）： 这是最完整的刚体模型，同时考虑了系统的运动学和动力学。示例请参阅 Ballbot 示例。https://leggedrobotics.github.io/ocs2/robotic_examples.html#doxid-ocs2-doc-robotic-examples-ballbot

中心模型（Centroidal model）： 这是一种常用于足式机器人的模型，因为它具有两方面的优点。首先，它考虑了系统的全部运动学。其次，它只考虑受外力影响的部分动力学，而不是计算成本高昂的全部动力学。有关示例，请参阅足式机器人示例。https://leggedrobotics.github.io/ocs2/robotic_examples.html#doxid-ocs2-doc-robotic-examples-legged-robot

选择好模型后，与其从头开始，不如从类似的示例开始。OCS2 库中已经包含了多种机器人示例，涵盖了大多数常用模型。有关这些示例的详细信息，请阅读机器人示例页面。https://leggedrobotics.github.io/ocs2/robotic_examples.html#doxid-ocs2-doc-robotic-examples

二、最优控制公式

OCS2 专门用于解决切换系统（switched systems）的最优控制问题。切换系统由有限个动态子系统组成，这些子系统会受到离散事件的影响，而离散事件会导致这些子系统之间的转换。切换系统模型在许多实际应用中都会遇到，如带有不同齿轮的汽车和机车、DC-DC 转换器、制造过程、生物系统和行走机器人等。此类系统的最优控制可表述为

$$
\begin{aligned}
&\text{minimize } \mathbf{u}(.) && \sum_i \phi_i(\mathbf{x}(t_{i+1})) + \int_{t_i}^{t_{i+1}} l_i(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \\
&\text{subject to} && \mathbf{x}(t_0) = \mathbf{x}_0 && \text{initial state} \\
& && \dot{\mathbf{x}}(t) = \mathbf{f}_i(\mathbf{x}(t), \mathbf{u}(t), t) && \text{system flow map} \\
& && \mathbf{x}(t_{i+1}^+) = \mathbf{j}(\mathbf{x}(t_{i+1})) && \text{system jump map} \\
& && \mathbf{g_{1i}}(\mathbf{x}(t), \mathbf{u}(t), t) = \mathbf{0} && \text{state} - \text{input equality constraints} \\
& && \mathbf{g_{2i}}(\mathbf{x}(t), t) = \mathbf{0} && \text{state} - \text{only equality constraints} \\
& && \mathbf{h_i}(\mathbf{x}(t), \mathbf{u}(t), t) \geq \mathbf{0} && \text{inequality constraints} \\
& && \text{for } t_i < t < t_{i+1} \text{ and } i \in \{0, 1, \cdot\cdot, I-1\}
\end{aligned}
$$

其中， t_i为切换时间，t_I 为最终时间。对于每种模式，非线性成本函数都包括预跳跃成本和中间成本。乍一看，这种表述似乎有点吓人，但实际上它的解释非常简单，并为 MPC 问题的定义带来了最大的灵活性。我们可以把这个问题想象成一系列优化控制子问题，这些子问题通过系统跳跃图相互连接。每个子问题甚至可以有不同的状态和输入维度，只要它们之间的跳转图定义正确即可。如前所述，该方案中的切换是基于时间触发事件的。这意味着需要定义模式顺序和发生切换的事件时间。

对于许多机器人平台，如四旋翼机器人、球形机器人和手推车，系统只有单一模式（即单域系统）。在这种情况下，上述公式可简化为一个常规的最优控制问题。

$$
\begin{aligned}
&\text{minimize } \mathbf{u}(.) && \phi(\mathbf{x}(t_I)) + \int_{t_0}^{t_I} l(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \\
&\text{subject to} && \mathbf{x}(t_0) = \mathbf{x}_0 && \text{initial state} \\
& && \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) && \text{system flow map} \\
& && \mathbf{g_1}(\mathbf{x}(t), \mathbf{u}(t), t) = \mathbf{0} && \text{state} - \text{input equality constraints} \\
& && \mathbf{g_2}(\mathbf{x}(t), t) = \mathbf{0} && \text{state} - \text{only equality constraints} \\
& && \mathbf{h}(\mathbf{x}(t), \mathbf{u}(t), t) \geq \mathbf{0} && \text{inequality constraints}
\end{aligned}
$$

三、如何设置最优控制问题?

为了定义最优控制问题，至少需要定义系统动力学和成本函数，其中成本函数在输入时为正定黑塞矩阵，在状态时为正半定黑塞矩阵。幸运的是，OCS2 为定义最优控制问题提供了一个紧密匹配的接口，即 Optimal Control Problem（请参阅本页）。
https://leggedrobotics.github.io/ocs2/optimal_control_modules.html#doxid-ocs2-doc-optimal-control-modules

```
/** Optimal Control Problem definition */
struct OptimalControlProblem {
  /* Cost */
  /** Intermediate cost */
  std::unique_ptr<StateInputCostCollection> costPtr;
  /** Intermediate state-only cost */
  std::unique_ptr<StateCostCollection> stateCostPtr;
  /** Pre-jump cost */
  std::unique_ptr<StateCostCollection> preJumpCostPtr;
  /** Final cost */
  std::unique_ptr<StateCostCollection> finalCostPtr;

```

```
  /* Soft constraints */
  /** Intermediate soft constraint penalty */
  std::unique_ptr<StateInputCostCollection> softConstraintPtr;
  /** Intermediate state-only soft constraint penalty */
  std::unique_ptr<StateCostCollection> stateSoftConstraintPtr;
  /** Pre-jump soft constraint penalty */
  std::unique_ptr<StateCostCollection> preJumpSoftConstraintPtr;
  /** Final soft constraint penalty */
  std::unique_ptr<StateCostCollection> finalSoftConstraintPtr;

  /* Constraints */
  /** Intermediate equality constraints, full row rank w.r.t. inputs */
  std::unique_ptr<StateInputConstraintCollection> equalityConstraintPtr;
  /** Intermediate state-only equality constraints */
  std::unique_ptr<StateConstraintCollection> stateEqualityConstraintPtr;
  /** Intermediate inequality constraints */
  std::unique_ptr<StateInputConstraintCollection> inequalityConstraintPtr;
  /** pre-jump constraints */
  std::unique_ptr<StateConstraintCollection> preJumpEqualityConstraintPtr;
  /** final constraints */
  std::unique_ptr<StateConstraintCollection> finalEqualityConstraintPtr;

  /* Dynamics */
  /** System dynamics pointer */
  std::unique_ptr<SystemDynamicsBase> dynamicsPtr;

  /* Misc. */
  /** The pre-computation module */
  std::unique_ptr<PreComputation> preComputationPtr;

  ...
}
```

四、如何设置 MPC 循环？

到目前为止，您已经创建了一个最优控制问题。要设置 MPC，您需要在每次控制时，使用最新的状态测量值重复解决这个问题。虽然对于简单的系统来说，实时解决这个问题是可能的，但对于许多板载计算能力有限且控制频率较高的机器人平台来说，这是不可能的。为此，您需要做到以下几点：(1) 利用最新的状态测量结果，尽可能快地运行 MPC。(2) 使用最新的 MPC 输出，而不必担心在读取其输出时出现任何堵塞问题。为此，您需要一些同步机制来满足这些要求。OCS2 通过引入 MPC 接口和 MRT（Model Reference Tracking，模型参考跟踪）接口的概念来提供这些功能。

4.1 MPC 接口

MPC 接口负责用最新的测量结果安全地更新求解器。因此，用户可以安全地为求解器设置最新状态并推进它。如果求解器尚未结束上一次调用，则状态将被缓冲，直到求解器准备就绪，缓冲区大小为 1，因此求解器将始终获得最新状态。

4.2 MRT 接口

MRT 接口负责安全访问求解器的结果。它提供两种输出视图：基于时间的视图和基于状态的视图。在基于时间的方法中，MRT 只根据优化状态输入轨迹的线性插值输出查询时间的优化状态输入对。另一方面，基于状态的技术会使用给定时间和状态的反馈策略来评估最优输入。请注意，反馈策略选项应在求解器设置中激活（前提是求解器支持反馈策略）。

4.3 ROS 和非 ROS 版本

MPC和MRT接口协同工作，在机器人上部署MPC时需要同时使用这两个接口。根据您是在一台机器上运行 MPC 还是在不同机器上运行 MRT，您应该使用 MpcMrtInterface 或一对 MpcRosInterface 和 MrtRosInterface。顾名思义，后者使用 ROS 在 MPC 和 MRT 节点之间进行通信。

五、如何测试 MPC 输出？

最后一个阶段是调整成本和其他算法超参数。为了将规划问题与将 MPC 输出转换为机器人指令输入（如扭矩、所需关节角度和速度）的跟踪控制器分开，OCS2 配备了一个所谓的假人模拟器。在最简单的形式下，MRT 虚拟模拟器仅对优化后的状态输入进行插值，并在 rviz 中将其可视化。然而，如果您为虚拟模拟器设置了一个 Rollout 实例，它就会使用该实例来模拟 MPC 策略。在这种情况下，如果采用与优化控制问题中相同的动态，就可以用规划时使用的精确模型模拟 MPC 输出。更先进的模拟器（如 RaiSim）也可用作推出实例。