OCS2 优化控制工具箱（三）基本模块 汉化

前言

在本篇中，将为您提供如何定义 MPC 问题的一些提示。在 OCS2 中，OptimalControlProblem 结构定义了优化问题的主要组成部分，即动力学、成本和约束条件。

除此之外，您可能还希望为 MPC 提供一些参考轨迹、预定义模式计划（用于切换系统）和其他外部信息，如模型参数的更新（如在自适应控制设置中）、环境地图（如用于避免碰撞的 SDF 地图）等。所有这些都可以借助参考管理器（Reference Manager）和求解器同步模块（Solver Synchronized Module）来实现。

我们从最优控制问题的组成部分开始：成本函数、软约束、（硬）约束、动力学和预计算。一般来说，成本和（硬/软）约束条件是在三个不同的时间实例中定义的：(1) 中间时间间隔，(2) 切换（预跳跃）时间，(3) 优化范围的最终时间。注意：预跳变部分只在切换系统问题中有效，根据切换前的状态值（预跳变值）进行评估。

优化控制问题（Optimal Control Problem）收集的是一组成本和约束条件，而不是单一的成本或约束条件。您可以将这些收集器想象成一个容器，在其中您可以使用唯一的名称添加术语，之后再以相同的名称将其请求回来。每个术语都可以通过覆盖术语的 isActive 方法来决定是否处于活动状态。这有助于改变 MPC 行为。例如，你可以使用外部触发信号在移动机械手的基础跟踪或末端执行器跟踪模式之间进行切换。此外，这些收集器还为您的代码引入了一定程度的模块化，并为在整个项目中扩展和修改 MPC 提供了方便的工具。例如，如果您决定在问题中添加新的约束或成本，您可以将新项添加到最优控制问题中，而无需修改其他项。

一、Costs - 成本

如前所述，成本函数是为三个不同的时间实例定义的：中间、预跳和最终。中间成本项可以是时间、状态和输入（OptimalControlProblem::costPtr）的函数，也可以是时间和状态（OptimalControlProblem::stateCostPtr）的函数。不过，预跳跃和最终代价项应该只是时间和状态的函数。成本项应继承自 StateCost 或 StateInputCost 类。派生类应定义代价值及其二次近似值。对于复杂函数，可以使用这些类的自动微分版本，即 StateCostCppAd 或 StateInputCostCppAd，用户只需提供代价值（cost value）即可。有关这些类的实现细节，请参阅 "ocs2_core/cost"，有关简单示例，请查看 QuadraticStateCost 和 QuadraticStateInputCost。

注意：我们的所有最优控制求解器都假设，在每个中间时间，与输入相关的中间成本项的 Hession（黑塞）之和为正定值。此外，除 PISOC 外，其他求解器都要求在每个时间实例（中间、预跳和最终）中，所有与状态有关的项的 Hession（黑塞）总和也是正半定的。需要注意的是，虽然在线性搜索策略和信任区域全局化策略中使用 Hession（黑塞）修正在一定程度上可以处理非定义性问题，但如果能保证这种正半定义性，求解器的工作通常会更可靠。避免非定域性的一种方法是对 f(x,u)^2形式

的成本函数使用高斯-牛顿近似技术，其中 f (x,u)的线性近似值将用于形成正定的 Hessian。更多详情，请参阅状态输入成本高斯-牛顿近似StateInputCostGaussNewtonAd。

二、Constraints - 约束

与成本类似，也为三个不同的时间实例定义了约束条件：中间、预跳和最终，其中中间约束条件是时间、状态、输入或时间、状态和预跳的函数，而最终约束条件仅是时间和状态的函数。

约束条件应继承自 StateConstraint 或 StateInputConstraint 类。派生类应根据约束的顺序（ConstraintOrder）定义约束值及其线性或二次近似值。对于复杂函数，可以使用这些类的自动微分版本，即 StateConstraintCppAd 或 StateInputConstraintCppAd，其中只需提供约束值。有关这些类的实现细节，请参阅 "os2_core/constraint"，有关简单示例，请查看 LinearStateConstraint 和 LinearStateInputConstraint。

在 OCS2 中处理约束时，可以使用硬约束或软约束方法。

软约束由 OptimalControlProblem 单独收集。软约束的处理基于惩罚方法，在这种方法中，约束被用户定义的惩罚函数包裹（关于这些惩罚函数的列表，请参阅 "ocs2_core/soft_constraint/penalties"）。要从约束项创建软约束，可以使用 StateSoftConstraint 和 StateInputSoftConstraint 类。这些类可以获取约束项实例和选择的惩罚函数，并创建软约束收集器可以收集的代价项。在设置约束的同时设置惩罚函数，可以灵活地为每个约束使用不同的惩罚函数（不同类型和/或不同的超参数）。这使得违反约束条件的调整变得更加容易。

硬约束条件（简称约束条件）根据其类型的不同，通过不同的技术以更高的精度进行处理。通过投影法处理状态-输入等式约束。纯状态等式和不等式通过松弛壁垒法(relaxed-barrier method)或增强拉格朗日技术(an augmented Lagrangian technique)处理（此功能暂时禁用，将在下一版本中添加）。

注意：由于状态-输入等式约束是通过投影法处理的，OCS2 假定约束的雅可比（Jacobin）相对于输入是全行列的。如果无法保证这一条件，则应使用软约束技术。

三、Dynamics - 动力学

动力学由其流动图、跳跃图及其一阶近似值（参见 SystemDynamicsBase）定义。对于常规系统，跳跃映射是一个标识映射，但对于切换系统，该映射可以是状态的非线性函数。对于复杂函数，可以使用自动微分版动力学 SystemDynamicsBaseAD。

四、Pre-computation - 预计算

OCS2 对缓存友好，这意味着可以在成本、约束条件、系统动力学及其近似值之间共享计算。为此，OCS2 使用了预计算（PreComputation）。在评估成本、约束条件和系统动力学之前，OCS2 求解器会调用 PreComputation::request （在切换时间调用 requestPreJump 或在最终时间调用 requestFinal），并发送请求消息，说明下一步将进行哪些操作。因此，您可以根据表示请求集的输入参数来实现 request() 方法。

注意：一般情况下，应避免使用缓存，仅在以后实施缓存版本以提高性能。有关示例，请参阅 "ocs2_robotic_examples/ocs2_mobile_manipulator"。

五、Changing parameters of the Optimal Control Problem - 改变最优控制参数

一旦您向求解器定义并设置了最优控制问题，求解器就会在内部创建一个副本（实际上是多个副本）。因此，如果您决定更改最优控制问题中的任何参数，如参考轨迹或模型参数，您就不能通过简单修改您可以访问的最优控制问题的参数来实现这一目的。此外，无论技术上如何，在任何时候都应避免任意更改这些参数。因为当您更改参数时，您的 MPC 求解器可能正处于迭代过程中，这将导致求解器出现未定义的行为。为了避免这个问题，OCS2 引入了参考管理器和求解器同步模块的概念。一般来说，这些都是同步概念，可确保在正确的时间（MPC 每次迭代之前和/或之后）更新参数。换句话说，它们使参数更新与 MPC 迭代同步。

要访问最优控制组件（如成本、约束、动力学.....）中的更新参数/信息，需要采取以下步骤：(1) 创建同步模块的共享指针。(2) 与成本、约束或动力学共享该实例的地址。(3) 通过 SolverBase::setReferenceManager、SolverBase::addSynchronizedModule 或 SolverBase::setSynchronizedModules 将其设置给求解器。

注意：在整个 MPC 问题中，每个同步模块应该只有一个实例。

六、Reference Manager Interface - 参考管理器接口

ReferenceManagerInterface 创建了一个通用接口，用于定义目标轨迹和模式时间表（仅用于切换系统）。OCS2 的每个求解器在开始新的 MPC 迭代之前都会调用参考管理器的 preSolverRun()。关于该接口的实现，请参考 ReferenceManager 类。ReferenceManager 有两个装饰器类：ReferenceManagerRos（将 ROS 通信添加到 ReferenceManager 中）和 LoopshapingReferenceManager（将其扩展到环形最优控制问题中）。

由于参考管理器是按照 MPC 主循环的顺序运行的，因此为了提高效率，应避免在 preSolverRun 中进行复杂操作。为此，应在不同的线程中处理这些参数，并将结果保存在缓冲存储器中。然后在 preSolverRun 中通过地址交换更新活动参数即可。OCS2 为此提供了一个名为 BufferedValue 的辅助类。

七、Solver Synchronized Modules - 求解器同步模块

SolverSynchronizedModules 类似于 ReferenceManagerInterface，但适用于通用应用程序。它只有两个纯虚拟方法 preSolverRun 和 postSolverRun，顾名思义，这两个方法分别在每次 MPC 迭代之前和之后调用。preSolverRun 方法还可以访问最近更新的 ReferenceManagerInterface。相比之下，postSolverRun 方法可以访问 MPC 解决方案。

与 ReferenceManagerInterface 类似，SolverSynchronizedModules 也是按照 MPC 主循环的顺序运行的。因此，为提高效率，应避免在 preSolverRun 和 postSolverRun 中进行复杂操作。为此，应在不同的线程中保存/计算这些参数，并将结果保存在缓冲区中。然后在 preSolverRun 或 postSolverRun 中，可以通过地址交换更新活动参数。为此，您可以使用 BufferedValue 类。