Ungar —— 一个使用模板元编程实现实时最优控制的 C++框架

Flavio De Vincenti and Stelian Coros

摘要—我们介绍了 Ungar，一个开源库，用于辅助实现高维最优控制问题（OCP）。我们采用现代模板元编程技术，在编译时对复杂系统进行建模，同时保持最大的运行时效率。我们的框架提供了语法糖，允许以表达性的方式定义一系列结构化动力学系统。虽然核心模块仅依赖于纯头文件的 Eigen 和 Boost.Hana 库，但我们还是将代码库与自动微分、代码生成及非线性规划的可选包和自定义包装捆绑在一起。最后，我们在各种模型预测控制应用中展示了 Ungar 的灵活性，具体包括四足行走和多个一臂四足机器人协作的运动操作。Ungar 在 https://github.com/fdevinc/ungar 下以 Apache 许可证 2.0 提供。

I. 介绍

模型预测控制（MPC）方法的进步已经赋予了机器人卓越的运动技能。最近的人形 [8] 和四足机器人 [4, 11] 的展示展示了曾经属于科幻小说中的壮举。然而，仍需大量的工程努力才能使这种实用的 MPC 实现成为可能。在实时速率下解决大规模非线性规划（NLP）问题与机械系统的固有高维度和快速动态特性相冲突。这些相互矛盾的方面转化为需要在几秒钟内完成的沉重计算成本，因此需要复杂的数据结构和巧妙的软件设计。

我们致力于促进手动工作投入到 MPC 控制器的开发中。在我们的愿景中，易用性和对广泛应用的相关性至关重要。这些目标只有通过谨慎避免任何运行时计算开销才能实现。同时，用户界面必须提供一种直观的语法，这种语法反映最优控制问题 (OCPs) 的标准数学公式。

This research was supported by the Swiss National Science Foundation through the National Centre of Competence in Digital Fabrication (NCCR DFAB) and through Grant No. 200021 200644, and by the European Research Council (ERC) under the European Union' s Horizon 2020 research and innovation program (Grant No. 866480).

We express our gratitude to Miguel Angel Zamora Mora and Zijun Hui for their precious contributions to the development of Ungar.

The authors are with the Computational Robotics Lab, ETH Zurich, Switzerland. flavio.devincenti@inf.ethz.ch

实现稳健的 MPC 性能在很多方面都具有挑战性。高效的 NLP 求解器必须与快速导数计算相结合。由于状态和控制变量的数量庞大且相互关联，手动实现一阶和二阶导数将导致一个繁琐且容易出错的过程。此外，尽管成熟的自动微分（AD）和 NLP 库存在，但大多数实现要求所有变量堆叠在一个向量中，这需要一些索引维护逻辑。这一事实提出了一个问题：什么样的数据结构可以存储它们同时保证零成本的访问操作并适应不同的系统设计。

与 Ungar 一起，我们提供了一种元语言来应对这些建模挑战。我们的解决方案引入了显著简化通常在最优控制中出现的自然语言处理问题定义的结构。我们使用模板元编程（TMP）将必要实现的生成委托给编译器，同时用户可以完全专注于所需 MPC 应用的架构细节。我们的方法使结构化变量集的转录无缝，并在编译时编码所有层次和索引信息。因此，对相应创建的对象的所有读/写操作不会产生额外开销，就像即席编程解决方案一样。由于 Ungar 的核心是仅头文件的，它在 C++项目中的集成非常轻松。然而，我们也包含了一个可选接口到 CppADCodeGen[3, 15] 以自动生成导数，并提供一个可选的序列二次规划（SQP）求解器使用 OSQP[20] 作为后端；如果启用，则所有外部依赖项会通过 CMake 自动下载。最后，我们通过为越来越复杂的系统实现 MPC 控制器来展示 Ungar 的能力，包括四足机器人和团队合作搬运物体的四足操作装置。

相关工作

存在许多开源软件包来协助创建 MPC 控制器。一个不全面的列表包括用于建模、仿真以及机械系统优化控制的框架，例如 Control Toolbox [10] 及其继任者

```cpp
// Define integral constants.
constexpr auto N          = 30_c;
constexpr auto NUM_ROTORS = 4_c;

// Define "leaf" variables.
constexpr auto position         = var_c<"position">,         3>;
constexpr auto orientation      = var_c<"orientation">,      Q>;
constexpr auto linear_velocity  = var_c<"linear_velocity">,  3>;
constexpr auto angular_velocity = var_c<"angular_velocity">, 3>;
constexpr auto rotor_speed      = var_c<"rotor_speed">,      1>;

// Define "branch" variables.
constexpr auto x = var_c<"x"> <<= (position, orientation, linear_velocity, angular_velocity);
constexpr auto X = var_c<"X"> <<= (N + 1_c) * x;
constexpr auto u = var_c<"u"> <<= NUM_ROTORS * rotor_speed;
constexpr auto U = var_c<"U"> <<= N * u;
constexpr auto decision_variables = var_c<"decision_variables"> <<= (X, U);
```

Listing 1. Decision variables of an OCP for controlling a quadrotor with Ungar.

OCS2 [9], Drake [21], Crocoddyl [18], TOWR [25], FROST [13], Quad-SDK [19], SymForce [17] 等。这些库针对机器人应用，它们解决的 OCP 需要特定的结构。相比之下，面向 NLP 的框架解决了更广泛类别的非线性优化问题；值得注意的例子有 IPOPT[24], ACADOS[14, 22], PSOPT[2], CasADi[1] 等。Ungar 通过提供新颖的系统建模特性补充了上述库的功能。特别是，它允许快速设置广泛采用的 AD 实现和 OCP 求解器所需的数据结构。最终，虽然 MPC 是其主要关注点，但 Ungar 的设计使其适用于任何需要解决有限维 NLP 问题的应用。

II. 数据结构

在本节中，我们介绍了 Ungar 的核心两种主要数据结构：变量和映射。我们在机器人领域的动机示例的伴随下描述了它们。我们注意到，我们讨论的类仅基于两个外部依赖项，即 Eigen[12] 和 Boost.Hana[7]。前者是一个线性代数模板库，在机器人代码库中无处不在，因为它的高性能和灵活性；后者是一系列实用程序和容器的集合，极大简化了 TMP 算法的实现。由于两者都是仅包含头文件的库，我们将它们包裹在 Ungar 内部，以便尽可能简单地将其集成到 C++项目中。

A. 变量

Ungar 的核心是Variable 模板类。变量描述了感兴趣的数量的结构，例如状态、输入或参数。每个变量都有

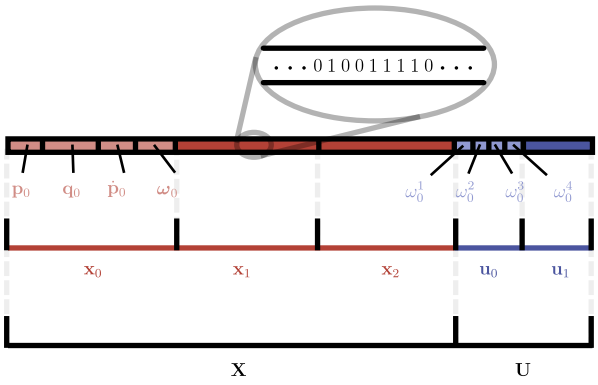

图 1. 分层关系在控制四旋翼的 OCP 的状态和输入变量之间及其底层记忆表示。在每个时间步长 k，我们将机器人的状态 $_k$ 定义为一个堆叠向量，其中包含其位置 $\boldsymbol{p}_k \in \mathbb{R}^3$、方向 $\boldsymbol{q}_k \in \mathbb{S}^3 \subseteq \mathbb{R}^4$、线速度 $\dot{\boldsymbol{p}}_k \in \mathbb{R}^3$ 和角速度 $\boldsymbol{\omega}_k \in \mathbb{R}^3$。输入 $\boldsymbol{u}_k$ 包括四个转子速度 $\omega_k^i \in \mathbb{R}, \forall i \in \{1,2,3,4\}$。最后，$\boldsymbol{X} \in \mathbb{R}^{13(N+1)}$ 和 $\boldsymbol{U}^{4N}$ 表示堆叠的状态和输入向量，其中 $N \in \mathbb{N}$ 是离散时间范围—在图中，$N = 2$。使用 TMP 技术，Ungar 支持生成用于高效操作原始数据数组的数据结构。

一个名称和一种类型 [1]，并且通过层次关系与其他所有变量相关联。当Variable 对象被实例化时，所有这些信息都通过三个模板参数编码在其类型中：用于名称的编译时常量字符串、标识其类型的整数编号以及表示变量层次结构的编译时常量映射。我们使用boost::hana::string 和boost::hana::map 类型来分别指定名称和变量层次结构。特别是，后者将变量名映射到子Variable 对象或其数组。

[1]在本文中，我们赋予术语类型和类型不同的含义。为了澄清，类型表示变量所属的数学群组，而类型专指 C++数据类型。

为了澄清并解释上述设计选择，让我们考虑一个用于控制四旋翼飞行器的有限时间范围 MPC 的决策变量。系统在时间步 k 的状态 $_k$ 包括机器人的姿态和速度；控制输入 $\boldsymbol{u}_k$ 由四个电机转速组成。给定一个离散的时间范围 $N \in \mathbb{N}$ 并假设直接多重射程公式 [6]，优化变量分别对应于堆叠的状态向量 $\boldsymbol{X} = [\begin{smallmatrix}\top & \top & \dots^\top\end{smallmatrix}_0 \ \begin{smallmatrix}\top\end{smallmatrix}_1 \ \dots_N]^\top$ 和输入向量 $\boldsymbol{U} = [\boldsymbol{u}_0^\top \ \boldsymbol{u}_1^\top \ \dots \boldsymbol{u}_{N-1}^\top]^\top$。然后，我们可以使用 Ungar 模板对象var_c 来实例化如 Listing 1 所示的相关变量。

var_c 构造接受一个表示变量名称的固定字符串和一个可选整数参数，该参数代表其类型：如果存在，则实例化一个 “叶子” 变量，即没有附加子结构的变量；否则，它创建一个 “分支” 变量，这种变量仅在其子变量关系中定义。我们将标量类型的叶子变量标识为1，将单位四元数类型的实现定义常量标识为Q，并将任何正整数对应于相应大小的向量。如 figure 1 所示意表示，Ungar 允许将变量编码的结构映射到连续内存缓冲区 (参见 Section II-B)。

在我们的示例中，行 1-3 定义了一个离散时间区间 N，该区间包含 30 个时间步骤，并通过用户定义的字面量_c 将旋翼的数量作为整数常量 [7]。线条 5-10 定义了四旋翼的姿态、速度以及旋翼转速的叶变量。相反，行 12-17 分别引入了变量 $_k$,$\boldsymbol{X}$,$\boldsymbol{u}_k$ 和 $\boldsymbol{U}$ 的分支变量。我们观察到 $_k$ 由机器人的堆叠位置、方向、线性速度和角速度组成，而 $\boldsymbol{X}$ 包含 $N+1$ 个堆叠状态。然后，我们可以立即使用重载函数operator<<=、operator, 和operator*表达上述结构信息，如在行 13-14 所示。类似地，我们定义了变量 $\boldsymbol{u}_k$ 和 $\boldsymbol{U}$ 的分支，并最终在对象 decision_variables 的第 17 行内堆叠了 $\boldsymbol{X}$ 和 $\boldsymbol{U}$。

在 Listing 1 中定义的所有变量都是constexpr，可以在编译时查询两种主要信息：它们的大小和它们在层次结构中的索引。例如，参考 figure 1 中的图表以及对于 $N = 30$，我们可以写出

```
static_assert(
    x.Size() == 13 &&
    X.Size() == 403 &&
    u.Size() == 4 &&
    U.Size() == 120 &&
    decision_variables.Size() == 523
);
```

和

```
static_assert(
    X(x, 0).Index() == 0 &&
    X(x, 1).Index() == x.Size() &&
    X(x, 1, linear_velocity).Index() == 20 &&
    decision_variables(U).Index() == X.Size() &&
    U(u, 0).Index() == 0 &&
    U(u, 1).Index() == 4 &&
    U(u, 1, rotor_speed, 0).Index() == 4 &&
    U(u, 1, rotor_speed, 1).Index() == 5
);
```

如上所示，我们可以通过函数调用运算符operator() 访问任何分支变量的所有子变量。如果存在多个副本的子变量，我们必须写明我们要访问的子变量的零基础索引。最重要的是，如果从分支变量到其任何子变量的路径没有歧义，我们可以跳过所有中间变量，如 Listing 2 所示：当变量组织在深层层次结构中时，此功能非常方便。最后，Ungar 提供了根据惯例定义变量的宏，即它们的名称与相应的对象名称相同。因此，Listing 1 可以重写为如 Listing 3 所示的更简洁的方式。

B. 变量映射

Variable 框架提供了用最少且表达力强的语法描述复杂系统的方法，而不会产生运行时成本。为了将系统描述转换为有用的数据结构，Ungar 提供了模板类 VariableMap。变量映射将一个变量与标量数组关联，并将每个子变量与子数组关联。为了执行各种映射，我们采用了Eigen::Map 类 [12]，它允许原始缓冲区与密集矩阵表达式无缝对接。所有必要的 Eigen 映射都在 VariableMap 构造函数的执行过程中创建；因此，访问任何子变量数据没有运行时成本。这只有在我们采用 TMP 技术的情况下才有可能，并使 Ungar 类似于一种元语言。

给定一些用户定义的标量类型scalar_t，我们可以为在 Section II-A 中引入的四旋翼决策变量创建一个变量映射：

```
auto vars = MakeVariableMap<scalar_t>(
    decision_variables);
```

然后，我们可以通过将相应的子变量传递给Get 方法来访问所有子映射。例如，我们可以将所有单位四元数初始化为恒等旋转，并将所有剩余的决策变量初始化为零 [12]：

```
vars.Get(X).setZero();
for (auto k = 0; k < N + 1; ++k) {
    vars.Get(orientation, k).setIdentity();
}
```

```
static_assert(X(x, 1, linear_velocity)                    == X(linear_velocity, 1));
static_assert(U(u, 1, rotor_speed, 0)                     == U(rotor_speed, 1, 0));
static_assert(U(u, 1, rotor_speed, 1)                     == U(rotor_speed, 1, 1));
static_assert(decision_variables(X, x, 1, linear_velocity) == decision_variables(linear_velocity, 1));
static_assert(
    decision_variables(U, u, 2, rotor_speed, 3) == decision_variables(u, 2, rotor_speed, 3) &&
    decision_variables(U, u, 2, rotor_speed, 3) == decision_variables(rotor_speed, 2, 3)
);
```

Listing 2. Equivalent expressions for unambiguous variable hierarchies.

```
// Define integral constants.
constexpr auto N          = 30_c;
constexpr auto NUM_ROTORS = 4_c;

// Define "leaf" variables.
UNGAR_VARIABLE(position,           3);
UNGAR_VARIABLE(orientation,        Q);
UNGAR_VARIABLE(linear_velocity,    3);
UNGAR_VARIABLE(b_angular_velocity, 3);
UNGAR_VARIABLE(rotor_speed,        1);

// Define "branch" variables.
UNGAR_VARIABLE(x) <<= (position, orientation, linear_velocity, b_angular_velocity);
UNGAR_VARIABLE(X) <<= (N + 1_c) * x;
UNGAR_VARIABLE(u) <<= NUM_ROTORS * rotor_speed;
UNGAR_VARIABLE(U) <<= N * u;
UNGAR_VARIABLE(decision_variables) <<= (X, U);
```

Listing 3. Equivalent transcription of Listing 1 with improved readability using macros.

```
vars.Get(U).setZero();
static_assert(
    std::same_as<
        decltype(vars.Get(U)),
        Eigen::Map<Eigen::VectorX<scalar_t>>&
    >
);
```

我们注意到通过Get 方法返回的所有对象都具有引用类型，因此它们不会执行复制操作，而是直接操纵底层数据。此外，返回的类型取决于相应变量的类型，因此它可以是引用scalar_t，一个到单位四元数的 Eigen 映射，或者是一个向量的 Eigen 映射。分支变量总是映射到涵盖所有相应子变量的向量（第 6 行）。

我们通过内部采用将子变量与相应数据子数组关联的编译时映射来实现这种灵活性。虽然此解决方案可确保最佳的运行时性能，但它可能在编译时间和内存消耗方面要求较高。如果，例如，此类变量映射仅用于中间代码生成步骤，这可能是不希望的；事实上，大多数代码生成器都会优化输入代码，从而使此阶段不需要任何性能优化。我们通过命名为VariableLazyMap 的VariableMap 的延迟版本来解决这一需求。延迟映射按需实例化 Eigen 映射，这是一项廉价的操作，涉及两个整数的副本。它们的构造函数需要一个具有正确大小的数据缓冲区，如下所示：

```
VectorXr underlying{decision_variables.Size()};
auto lvars = MakeVariableLazyMap(underlying,
    decision_variables);
```

特别是，我们可以将我们的初始化示例重写为：

```
lvars.Get(X).setZero();
for (auto k = 0; k < N + 1; ++k) {
    lvars.Get(orientation, k).setIdentity();
}
lvars.Get(U).setZero();
static_assert(
    std::same_as<
        decltype(lvars.Get(U)),
        Eigen::Map<Eigen::VectorX<scalar_t>>
    >
);
```

我们强调VariableMap 和VariableLazyMap 对象之间的唯一区别在于Get 的返回类型。如第 6 行的静态断言所示，惰性映射通过值而不是引用返回 Eigen 映射。

III. 实验

我们通过实现两种不同的但相关的系统中的 MPC 方案来验证 Ungar。前者是一个基于单刚体动力学（SRBD）模型 [5] 的四足行走控制器；后者是一个使用一臂四足机器人的集中式 MPC 控制器，用于协作行走和操作 [23]。特别是，合作行走操作示例将简单的行走控制器扩展到多个带有手臂的机器人共同携带共享负载的情况。我们附带的视频展示了使用这两种控制器进行仿真实验的记录。这些应用使我们能够展示通过仅更改几行代码即可定义完全不同的变量层次结构的 Ungar 的灵活性。虽然我们没有提供讨论中的控制器的源代码，但我们捆绑了包含多个示例的库，其中包括四旋翼和微型无线电控制汽车 [16] 的非线性 MPC 实现。

A. 实现细节

我们完全依赖 Ungar 提供的功能来实现我们的控制器。在我们的测试中，我们使用核心数据结构以及可选的 CppADCodeGen 包装器 [15] 和 SQP 求解器。我们的求解器基于 Grandia et al. [11] 的近期工作，并建议感兴趣的读者参考 [23] 以获取更多详情。

为了生成必要的导数，CppADCodeGen 需要所有函数依赖于一个包含独立变量和参数的单一数据数组。我们通过如 Section II-B 中所述的 Ungar 映射来符合此接口。对于每个控制器，我们定义了变量层次结构 decision_variables 和parameters。决策变量包括状态和控制输入，而参数包含惯性属性、参考轨迹、物理常数等。在以下各节中，我们将仅展示各种 MPC 控制器的决策变量定义以保持简洁明了。尽管如此，在所有实现中我们始终采用这种划分，并邀请读者探索库中的示例进行详细浏览。

Ungar 需要支持 C++20 的编译器 [2]。我们在 Ubuntu 22.04.2 LTS 和 GCC 11 上实现了并彻底测试了我们的控制器，但 Ungar 的核心模块不依赖于任何特定于操作系统的指令。然而，我们注意到我们的可选 CppADCodeGen 包装器使用了 Linux 提供的运行时编译和动态链接功能。

[2]为了方便起见，我们在一个单独的 Git 分支上提供了一个符合 C++17 标准的库版本。虽然这种改编缺少一些功能，但它实现了 Ungar 的所有最重要功能。我们请读者参阅 GitHub 网页上的文档以获取更多信息。

编译时间：Ungar 对象在构造时没有运行时开销。因此，我们只为不同大小的系统提供编译时间的基准测试。为此，我们测量了构建VariableMap 和 VariableLazyMap 对象所需的时间，这些对象用于定义 Listing 3 中的决策变量。我们在配备 i7-11800H、2.30 GHz、16 核 CPU 的笔记本电脑上针对不同值的N和NUM_ROTORS 进行此测量，并在 figure 2 中绘制相应的热图。我们可以看到，懒惰映射具有更优的编译时间，只需 14 s 即可构建八轴飞行器的数据结构，该飞行器有 $N = 390$。相比之下，在相同设置下，VariableMap 的编译时间为 39 s 长。尽管与VariableLazyMap 相比，VariableMap 扩展性较差，但我们可以注意到对于时间跨度小于 100 个时间步的情况，它们具有相似的编译性能。然而，我们建议在原型设计时使用惰性映射，并切换到VariableMap 来获取最快的运行时性能。

B. 四足运动

我们基于 Bledt and Kim [5] 的控制器构建了四足运动的 MPC 公式，但有三个显著的不同之处。首先，我们使用非线性的 SRBD 方程，不进行线性化或简化。其次，我们用单位四元数表示方向而不是欧拉角以防止奇异性问题。最后，我们采用 Lie 群时间步进方法来集成动力学，保持四元数的单位范数约束 [23]。

我们定义我们的变量层次结构如 Listing 4 所示。特别是，我们可以看到少于 19 行代码就足以生成用于操作机器人状态和输入所需的数据结构。虽然一个 MPC 行走控制器需要额外的组件才能实际使用，例如步态规划器、逆运动学求解器和全身控制器，但我们已经可以体会到 Ungar 在简化具有复杂结构的 NLP 问题公式方面的潜力。

C. 协作式移动操作

我们使用 Ungar 实现了一个 MPC 控制器，该控制器同时优化一组单臂四足机器人集体操作物体时的地反力、操纵力矩、步态位置和身体轨迹。由此产生的最优控制问题维度非常高，并且状态和输入之间存在耦合动态和深层层次结构。例如，每个机器人有 4 条腿，每条腿在每一个时间步骤都与一个地反力和一个步态位置相关联；此外，每个机器人有一个手臂，该手臂对应一个操纵力和扭矩。

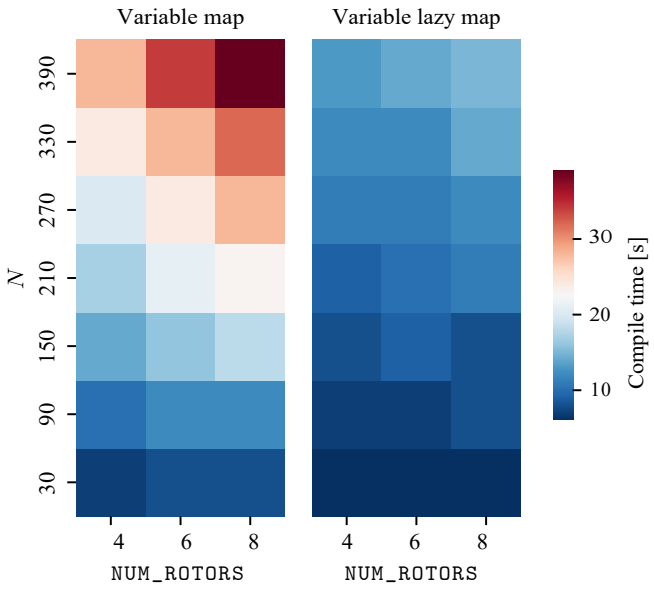

图 2. 编译生成变量映射 (**左侧**) 和变量懒加载映射 (**正确**) 的实现时间，用于 Listing 3 的多旋翼示例。我们通过改变 2 和 3 这几行来，针对不同的时间范围和旋翼数量对 Ungar 进行基准测试。热图显示了延迟映射与非延迟映射替代方案相比，更理想的编译时间。因此，变量映射类型仅应在寻求最佳 MPC 性能时使用。

我们将协作式行进操作（CLM）的 MPC 公式化为 SRBD 在多智能体系统中的扩展，并请感兴趣的读者参阅 [23] 以获取我们模型的详细描述。如 Listing 5 所示，创建 Ungar 变量用于看似复杂的 CLM 设置只需要对 Listing 4 的行进控制问题进行一些小的改动。

D. 限制

Ungar 的最佳运行时效率归因于大量编译时计算。然而，如果变量层次结构变得过深或嵌套，则编译时间可能会显著增加。此外，我们在实例化VariableMap 对象用于非常大的 OCPs 时观察到了编译器崩溃，这再次是由过多的编译时计算引起的。在这些罕见情况下，采用VariableLazyMap 类型就足够了，该类型计算量小得多，并且提供了几乎与非延迟版本相同的表现。对于未来的工作，我们将优化 Ungar 的设计以改进其编译时间。我们还计划扩展库，增加更多工具以促进高性能 MPC 控制器的实现。

IV. 结论

在这篇论文中，我们介绍了 Ungar，一个用于实时 MPC 应用的 C++模板库。我们的框架使用 TMP 技术来解决现有 NLP 和最优控制软件包中被忽略的建模需求。特别是，它提供了一种元语言，以变量层次结构描述复杂系统。然后，它允许编译器基于这些层次结构生成高度高效的代码，用于操作原始数据缓冲区。如我们在四足动物运动和协作位姿操作实验中所示，这些特性使得在制定 NLP 问题时具有极大的灵活性，并简化了符合 AD 规范的实现。

参考文献

[1] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. Mathematical Programming Computation, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

[2] Victor M. Becerra. Solving complex optimal control problems at no cost with psopt. 2010 IEEE International Symposium on Computer-Aided Control System Design, pages 1391–1396, 2010.

[3] Bradley M. Bell. CppAD: a package for c++ algorithmic differentiation. [Online]. Available: https://github.com/coin-or/CppAD.

[4] Marko Bjelonic, Ruben Grandia, Moritz Geilinger, Oliver Harley, Vivian Suzano Medeiros, Vuk Pajovic, Edo Jelavic, Stelian Coros, and Marco Hutter. Offline motion libraries and online mpc for advanced mobility skills. The International Journal of Robotics Research, 41:903 – 924, 2022.

[5] G. Bledt and Sangbae Kim. Implementing regularized predictive control for simultaneous real-time footstep and ground reaction force optimization. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6316–6323, 2019.

[6] M. Diehl, H. Bock, H. Diedam, and Pierre-Brice Wieber. Fast direct multiple shooting algorithms for optimal robot control. 2005.

[7] Louis Dionne et al. Boost.Hana: Your standard library for metaprogramming. [Online]. Available: https://github.com/boostorg/hana.

[8] Boston Dynamics. Atlas Gets a Grip | Boston Dynamics. https://www.youtube.com/watch?v=-e1_QhJ1EhQ, January 2023.

[9] Farbod Farshidian et al. OCS2: An open source library for optimal control of switched systems. [Online]. Available: https://github.com/leggedrobotics/ocs2.

[10] Markus Giftthaler, Michael Neunert, M. Stäuble, and Jonas Buchli. The control toolbox — an open-source c++ library for robotics, optimal and model predictive control. 2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), pages 123–129, 2018.

[11] Ruben Grandia, Fabian Jenelten, Shao-Hua Yang, Farbod Farshidian, and Marco Hutter. Perceptive locomotion through nonlinear model predictive control. ArXiv, abs/2208.08373, 2022.

[12] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[13] Ayonga Hereid and Aaron D. Ames. Frost: Fast robot optimization and simulation toolkit. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, September 2017. IEEE/RSJ.

[14] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. Optimal Control Applications and Methods, 32, 2011.

[15] Joao Rui Leal et al. CppADCodeGen. [Online]. Available: https://github.com/joaoleal/CppADCodeGen.

```
// Define integral constants.
constexpr auto N        = 30_c;
constexpr auto NUM_LEGS = 4_c;

// Define "leaf" variables.
UNGAR_VARIABLE(position,           3);
UNGAR_VARIABLE(orientation,        Q);
UNGAR_VARIABLE(linear_velocity,    3);
UNGAR_VARIABLE(angular_velocity,   3);
UNGAR_VARIABLE(force,              3);
UNGAR_VARIABLE(relative_position,  3);

// Define "branch" variables.
UNGAR_VARIABLE(leg_input) <<= (force, relative_position);
UNGAR_VARIABLE(x)         <<= (position, orientation, linear_velocity, angular_velocity);
UNGAR_VARIABLE(X)         <<= (N + 1_c) * x;
UNGAR_VARIABLE(u)         <<= NUM_LEGS * leg_input;
UNGAR_VARIABLE(U)         <<= N * u;
UNGAR_VARIABLE(decision_variables) <<= (X, U);
```

Listing 4. Decision variables of an OCP for quadrupedal locomotion using the single rigid body model.

[16] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale rc cars. Optimal Control Applications and Methods, 36:628 – 647, 2015. URL https://api.semanticscholar.org/CorpusID:11242645.

[17] Hayk Martiros, Aaron Miller, Nathan Bucki, Bradley Solliday, Ryan Kennedy, Jack Zhu, Tung Dang, Dominic Pattison, Harrison Zheng, Teo Tomic, Peter Henry, Gareth Cross, Josiah VanderMey, Alvin Sun, Samuel Wang, and Kristen Holtz. SymForce: Symbolic Computation and Code Generation for Robotics. In Proceedings of Robotics: Science and Systems, 2022. doi: 10.15607/RSS.2022.XVIII.041.

[18] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control. In IEEE International Conference on Robotics and Automation (ICRA), 2020.

[19] Joseph Norby, Yanhao Yang, Ardalan Tajbakhsh, Jiming Ren, Justin K. Yim, Alexandra Stutt, Qishun Yu, Nikolai Flowers, and Aaron M. Johnson. Quad-SDK: Full stack software framework for agile quadrupedal locomotion. In ICRA Workshop on Legged Robots, May 2022.

[20] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. Mathematical Programming Computation, 12(4):637–672, 2020. doi: 10.1007/s12532-020-00179-2. URL https://doi.org/10.1007/s12532-020-00179-2.

[21] Russ Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016. URL http://drake.mit.edu.

[22] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados – a modular open-source framework for fast embedded optimal control. Mathematical Programming Computation, Oct 2021. ISSN 1867-2957. doi: 10.1007/s12532-021-00208-8. URL https://doi.org/10.1007/s12532-021-00208-8.

[23] Flavio De Vincenti and Stelian Coros. Centralized model predictive control for collaborative loco-manipulation. In Robotics: Science and Systems, 2023. URL https://api.semanticscholar.org/CorpusID:259319056.

[24] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming, 106:25–57, 2006.

[25] Alexander W Winkler, Dario C Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. IEEE Robotics and Automation Letters (RA-L), 3:1560–1567, July 2018. doi: 10.1109/LRA.2018.2798285.

```
// Define integral constants.
constexpr auto N          = 10_c;
constexpr auto NUM_ROBOTS = 2_c;
constexpr auto NUM_LEGS   = 4_c;

// Define "leaf" variables.
UNGAR_VARIABLE(position,           3);
UNGAR_VARIABLE(orientation,        Q);
UNGAR_VARIABLE(linear_velocity,    3);
UNGAR_VARIABLE(angular_velocity,   3);
UNGAR_VARIABLE(force,              3);
UNGAR_VARIABLE(relative_position,  3);
UNGAR_VARIABLE(torque,             3);

// Define "branch" variables.
UNGAR_VARIABLE(leg_input)     <<= (force, relative_position);
UNGAR_VARIABLE(arm_input)     <<= (force, torque);
UNGAR_VARIABLE(robot_input)   <<= (NUM_LEGS * leg_input, arm_input);
UNGAR_VARIABLE(payload_state) <<= (position, orientation, linear_velocity, angular_velocity);
UNGAR_VARIABLE(robot_state)   <<= (position, orientation, linear_velocity, angular_velocity);
UNGAR_VARIABLE(  x  )             <<= (payload_state, NUM_ROBOTS * robot_state);
UNGAR_VARIABLE(X)             <<= (N + 1_c) * x;
UNGAR_VARIABLE(  u  )             <<= NUM_ROBOTS * robot_input;
UNGAR_VARIABLE(U)             <<= N * u;
UNGAR_VARIABLE(decision_variables) <<= (X, U);
```

Listing 5. Decision variables of an OCP for collaborative loco-manipulation with two robots modeled as single rigid bodies. We highlight the differences from the locomotion controller formulated in Listing 4. In particular, we mark newly added variables in yellow and modified variables in light blue.