Notes on Control Conventions

Marc Toussaint

October 9, 2020

[first version July 7, 2020]

In typical hierarchical control approaches the low-level controllers are black boxes. E.g., in hierarchical RL, an option is fully defined by the initiation set $\mathcal{I}$, termination set $\mathcal{T}$, and a policy π. No further assumptions are made about of what nature π is other than that, if initiated in $\mathcal{I}$, π runs autonomously until it stops somewhere in $\mathcal{T}$. By *black box* I mean that the reasoning mechanisms on the next higher level do not need to know any internals of π other than the abstract specification in terms of $\mathcal{I}$ and $\mathcal{T}$. Similarly, action operators in PDDL describe the nature of a controller only in terms of abstract pre-conditions and effects. What exactly the action's real-world controller does is not exposed – and does not need to be exposed – to the abstract reasoning level. Similarly, if finite state machines or behavior trees are used to coordinate sub-level controllers – for these approaches it is sufficient to have rather abstract specifications of pre-, post- and switching conditions of the controllers they coordinate (or semantic attachments). Of course, formalisms that treat low-level controllers as black boxes are highly desirable from the perspective of the generality: they can in principle coordinate any sub-policies in any contexts.

We here describe an alternative approach where we restrict ourselves to a very particular kind of lowest-level controllers. All controllers are related to establishing geometric[1] constraints that are defined via differentiable features over world configurations. More specifically, they are defined as mathematical programs that describe model-predictive control to achieve or maintain geometric constraints. The benefit of restricting ourselves to this kind of lowest-level control is that higher level reasoning and coordination does have access to these differentiable features, and can evaluate them to estimate feasibility or bounds. In particular, the features can be used directly to make geometrically-informed(!) decisions on how to coordinate the controllers, e.g. which ones can currently be executed and be combined, and how they could be sequenced to achieve long-term constraints – in contrast to merely symbolically informed higher-level reasoning or coordination systems. The low-level controllers are fully transparent to the high-level reasoning and coordination, rather than being black boxes.

Consider the following example to constrast this to symbolic abstractions of (pre-/post-) conditions: Assume various symbols that are grounded as geometric constraints. If you want to query whether some geometric constraints can hold simulatneously you might consider the conjunctions of these abstract symbols. However, that is not correct: When the robot can reach for A, and can also reach for B, this does not mean it can simultaneously reach for A and B. Working directly with geometric constraint rather than symbolic

[1] We continue using the term 'geometric' as in Logic-Geometric Programming – but it more generally means: defined in terms of *any* differentiable feature over sequences of consecutive world configurations.

abstractions, the higher level system has more options to reason about possibilities to simultaneously establish constraints, e.g. for chaining or concurrently executing controllers.

1 Control Convension

We provide here basic conventions on how to describe controllers to enable high-level coordination and reasoning with them.

1.1 Mathematical Program

As a convention of notation, we describe a non-linear mathematical program as a tuple (X, ϕ, ϱ), where X is the decision space, $\phi : X \to \mathbb{R}^K$ a set of features, and $\varrho : \{1, .., K\} \to \{\mathtt{S}, \mathtt{C}, \mathtt{I}, \mathtt{E}\}$ speficies for each feature whether it is a sum-of-squares, normal cost, inequality, or equality:

$$\min_{x \in X} \ \phi_{\mathtt{S}}(x)^\top \phi_{\mathtt{S}}(x) + \mathbf{1}^\top \phi_{\mathtt{C}}(x) \quad \text{s.t.} \quad \phi_{\mathtt{I}}(x) \le 0,\ \phi_{\mathtt{E}}(x) = 0 \ , \tag{1}$$

where $\phi_{\mathtt{S}} \equiv \phi_{\varrho^{-1}(\mathtt{S})}$ is the feature vector containing only the set of sum-of-square features, and analogously for $\phi_{\mathtt{C}}$, $\phi_{\mathtt{I}}$, and $\phi_{\mathtt{E}}$.

1.2 Controllers & transient features

Given a robot configuration space X, we describe a controller as a tuple (X, ϕ, ϱ) plus additional information $\theta_k \in \{0, 1\}$ for each feature that specifies whether it is immediate or *transient*. The controller then solves a short-horizon mathematical program (as in model-predictive control) defined by these features.

Specifically, if all features are immediate, the controller would directly solve for the 1-step MPC problem (X, ϕ, ϱ), where X is the space of the next robot configuration at time $t + \tau$ (where t is the current time and τ the time stepping).

If instead a feature is **transient**, $\theta_k = 1$, then the feature does not have to hold (or be minimized) immediately. Instead, the feature's target (=set point, or zero point) is monotonously shifted from its current value to its final target to produce a transient behavior from its current violation to final convergence. Such a transient moving target can, e.g., be realized by limiting the step size in target shift ("MaxCarrot"), and typically requires an additional parameter to characterize the transient dynamics (e.g., max step). In other terms, transient features modify the mathematical program by imposing small residual reducing steps in the constraints rather than fulfilling them immediately.

1.3 Feasibility of Controller

Given a candidate controller (ϕ, ϱ, θ) we can estimate feasibility, costs, and execution time if that controller was to be imposed.

Specifically, we can solve for **immediate feasibility** $\mathcal{F}_I(\phi)$, where we consider the NLP that drops all transient and non-hard constraint features. (Here, for brevity, with ϕ we implicitly refer to the full controller specification (ϕ, ϱ, θ).)

We can solve for **final feasibility** $\mathcal{F}_T(\phi)$, where we consider the full NLP (X, ϕ, ϱ) that includes transient constraints, but remove/reduce any control cost penalties or constraints, and the decision variable x describes a configuration in the far future – a potential point of convergence of the controller. (In LGP, this is related to the pose bound.)

In relation to options, $\mathcal{F}_I(\phi)$ and $\mathcal{F}_T(\phi)$ are indicators for the initiation and termination sets, respectively.

The same NLP also allows us to **estimate control costs**. In fact, by solving for a whole path (with potentially coarse time discretization) from current state to a point of final convergence we can compute lower bounds on the control costs (and execution time) with variable fidelity.

Note that all our estimates, immediate feasibility, final feasibility, and the control costs estimates can be made strict lower bounds[2] of the true control costs *neglecting stochasticity*. As always in decision making, such lower bounds are valuable pieces of information of higher-level coordination of such sub-routines.

1.4 Chain of Controllers

Consider two controllers ϕ_1 and ϕ_2 (where for brevity we omit notating also $\varrho_1, \theta_1, \varrho_2, \theta_2$). We want to estimate feasibility of chaining them.

We can solve for **sequence feasibility** $\mathcal{F}(\phi_{1:2})$, where the NLP includes all features ϕ_1 (as for final feasibility $\mathcal{F}_T(\phi_1)$), but only immediate features ϕ_2 (as for $\mathcal{F}_I(\phi_2)$). This solves for a potential switching configuration between the controllers.

This of course generalizes to any chain of controllers, $\mathcal{F}(\phi_{1:n})$. (In LGP, this is related to the sequence bound.)

1.5 THE PRIORITY CONTROLLER

We assume we have a finite set of controllers (grounded from a relational description, given a finite domain).

We assume that a higher-level planner returns a prioritized list $\phi_{1:n}$ of controllers. The execution then executes the first in this list that is feasible to initiate, $\mathcal{F}_I(\phi_i) = 1$.

Note that, instead of a priority list $\phi_{1:n}$, the high level controller could also return a Q-value Q_i for each controller ϕ_i; the priority controller would then choose the feasible-to-initiate controller with highest Q-value.

In this view, this approach is not limited to chains of controllers.

However, this control approach does not look ahead on a geometric level. It might choose a first controller that needs to be followed by a second controller which might be infeasible (due to som perturbation not accounted for during planning). E.g., the planner planned (pick a)-(place a on b), but meanwhile the placement b was moved very far away – and picking a makes no sense anymore. The reactive controller should account for this.

[2]Also in the sense of infeasibility.

1.6 CONTROL TREES

We assume that the high-level controller returns a priority list of controller chains. The control executes the first controller in the first feasible-to-sequence chain. Note that the list would typically contain all sub-chains of chains. E.g., if it contains A-B-C, then it should also contain B-C, and C, separately, and separately prioritized (with higher prioritization).

The format of a 'priority list of chains' might seem awkward. However, this is equivalent to a backward tree: Consider all chains to be 'backwardly' attached to a common root, where the root is the 'end' state. And merge the chains to a backward tree when they share a sub-sequence. Further consider all nodes of this tree to have a (consistent, decreasing from root) Q-value. The control executes the node with highest Q-value if its chain is feasible-to-sequence.