Trajectory Optimization Framework

Johnathan Van Why

September 17, 2013

1 Introduction

This framework is designed for the optimization of trajectories for robotic systems. It can handle underactuated and hybrid systems. In the future, it will have support for multiple "phases", or time periods in which specific dynamics are enforced.

The framework makes use of the MATLAB Symbolic Toolbox to aid in problem formulation and manipulation. The actual optimization, however, is almost always numerical. By keeping the problem in a symbolic form, we allow for the implementation of a mode where the optimization algorithm and problem are formulated together, leading to a final solver optimized for the specific problem's structure.

This document gives a basic description of how to use the framework. It also covers the architectural design of the framework and details on some of the most important algorithms contained within the framework. In addition to this document, code samples are located within the samples/ directory and function as additional documentation for the optimizer.

2 Usage

At the moment, the interface to the optimizer is undergoing a re-design, so it is likely to change in the near future. This section will hopefully be kept up to date with changes.

One thing that currently appears to be invariant is the usage of a central structure, called the "scenario" structure. The scenario structure contains all information necessary to run an optimization – it is incrementally filled out by the optimizer as it completes the various stages of problem setup and optimization. The final scenario structure contains the results of the optimization, and should contain all information that may need to be saved for future use.

The overall workflow for using the optimizer is as follows:

1. Incrementally set up the problem by calling optimizer functions to generate structures.
2. Call traj_optimize() on the scenario structure to do the optimization.

3 Structures

This section contains information on all the values in the structures used to define and optimize the problem. Since the optimizer is built around these structures, correctly setting them up is the majority of the work required to interface with the framework.

Note: This documentation reflects the plans for the next revision of the framework's API.

Each structure contains a struct_type string, which states the "type" of the structure. This allows for error checking and for mixing and matching optimizer functions with structure types.

Additionally, each structure contains a version integer. This version is incremented every time a breaking change is made to one of the structures. All versions are kept in-sync, since the structures contain each other. Each time the version number is incremented, functions for updating all relevant structures should be created for backwards compatibility. The optimizer will automatically update any outdated structures passed in to it. In this way, a scenario structure that is saved at one point in time will remain usable, even if the optimizer is updated.

The ordering of these structures in this section of the documentation reflects the nesting of the structures inside the scenario structure.

3.1 Constraint

This structure represents one or more constraints (inequality and/or equality).

Field	Type	Description
c	Symbolic expression or function handle	The inequality constraints function. This gets constrained such that $c <= 0$ at the solution point. If this is created by the user, it will contain a symbolic variable. Once it is processed by the optimization framework, it will be turned into a function handle. The set of variables of which it is a function will change based on the context it's in.
ceq	Symbolic expression or function handle	This is the same as for c, but for equality constraints (i.e. ceq is constrained such that $ceq = 0$ at the solution point).
struct_type	String	The type of the struct – always 'constraint'
version	Positive Integer	The version number of this structure.

3.2 Cost

This structure represents a cost.

Field	Type	Description
cost	Symbolic expression or function handle	The value of this cost
struct_type	String	The type of the struct – always 'constraint'
version	Positive Integer	The version number of this structure.

3.3 Phase

This structure encodes information on one dynamic phase.

Field	Type	Description
add_params	Cell array of strings	A list of all additional optimization parametrs for this phase. These are optimization parameters used by the user functions – they are not parameters generated by the optimizer.
dcost	Function handle	The derivative of cost for this phase. This is a function of the form $\dot{C} = c(x, u)$.
dx	Function handle	The dynamics for this phase. This is a function of the form $\dot{x} = f(x, u)$.
input	Cell array of strings	A column vector containing the names of each input for this phase.
intervals	Positive Integer	The number of time intervals in the discrete approximation to the underlying continuous-time problem.
name	String	The name of this phase.
noopt_params	Cell array of strings	A list of all parameters that will not be optimized for. These parameters may be quickly changed in between optimization runs.
state	Cell array of strings	This is a column vector cell array containing the names of each of the states for this phase.
struct_type	String	This struct's type. Always 'phase'
version	Positive Integer	The version number of this structure.

3.4 Scenario

This is the main structure, encoding all information necessary to perform the optimization.

Field	Type	Description
phases	Phase Array	All the phases for this optimization scenario. Ordered.
struct_type	String	This struct's type. Always 'scenario'
version	Positive Integer	The version of this structure.

4 Functions

This section documents the various functions within the optimizer.

4.1 traj_create_dynamics(dynamicsfcn, state, input)

traj_create_dynamics() creates a new dynamics structure. The dynamics function represents the system dynamics and is of the form $\dot{x} = f(x, u)$ where x is the state and u is the input. In MATLAB code, this means it is of the form dx = dynamicsfcn(state, input). The state and input are parameters structures created by traj_create_state () and traj_create_input (), respectively.

4.2 traj_create_input(...)

This creates a new input structure. Each parameter represents a single variable in this input and should be a string naming that variable (such as 'actuator rate' or 'force').

4.3 traj_create_scenario(...)

This creates the scenario structure. If you pass it phases, costs, or constraints, it'll go ahead and initialize the scenario with the given structures.

4.4 traj_create_state(...)

This creates the state structure. Each parameter represents a single variable in the state and should be a string naming that variable (such as 'position' or 'velocity').

4.5 traj_version()

This returns the current version of the optimizer.