Appendix I

RoboDyn

I.1 Introduction

RoboDyn is a C++ class library that simulates the dynamics (i.e. the motion) of articulated rigid bodies. The numerical engine is based on the PhD work of Scott McMillan [76, 77, 78, 79], which he implemented in the DynaMechs library. RoboDyn provides very efficient simulation: it is based on Featherstone's recursive articulated-body algorithm [34] which executes in $O(n)$ time where n is the number of links in the system. Numerical integration is user selected from Euler, midpoint or Runge-Kutta (in the future Runge-Kutta-Nyström methods should be implemented for extra speed [25]). This appendix gives some superficial details about the RoboDyn library.

I.2 Links, joints and the AB tree

An articulated rigid-body (AB) is a collection of "links" that are connected together by "joints". Each link is a solid body with its own mass and rotational inertia. Each joint is a constraint between two links so that they can only move in a certain way relative to each other.

Featherstone's AB algorithm requires that all the links are in a tree structure, with one link designated the system root. Closed loops of links are not allowed. This means that all links except the root have one inboard (or parent) link. So each link except the root has an associated joint which attaches it to its parent link. Each link has a coordinate system (CS) which is fixed relative to that link. The CS is transformed (rotated and translated) with respect to the CS of the parent link.

Each joint has between zero and six degrees of freedom. That is, the joint outboard link position is a function of n parameters of the inboard link position. Mobile root links have 6 joint parameters because they can be positioned arbitrarily in 3D space. Some special "spacer" links have no parameters because the joints are always in a fixed position.

The joint parameters are called the positional state variables. They parameterize the transformation from the inboard to the outboard CS. The time derivatives of the positional state variables are the velocity state variables. The simulation has n position and n velocity state variables, where n is the sum of the number of joint parameters for all links. The variables are arranged into an n-sized vector in a standard order which comes from a depth-first traversal of the AB tree.

Different link types in RoboDyn correspond to different types of transformations that get the link CS from the parent CS.

The articulated-body (AB) algorithm has three stages, which each recurse over the entire AB tree:

1. Forward kinematics: kinematic (positional) quantities are computed based on joint positions, from inboard links to outboard links.
2. Backward dynamics: some dynamic quantities such as the inboard reflected inertia and the inboard force are computed, from outboard links to inboard links.
3. Forward accelerations: the joint accelerations are computed, from inboard links to outboard links.

I.3 Widgets

The RoboDyn kernel computes the forces on the links that arise from the constraints at the joints. (In fact these forces are not computed explicitly, but the accelerations arising from them are). However, other "external" forces must be applied to the links and joints to make the simulation interesting. These include:

- Link-to-link and link-to-ground collision forces.
- Spring forces between links.
- Damping forces on the links, e.g. if the link is moving in a viscous medium.
- "Thruster" forces on the links, e.g. if a link is motivated by a turbine or reaction engine (rocket motor).
- Joint frictional forces.
- Joint limits (the stopping force when a joint comes to the end of its motion).
- Motor forces, using simple or detailed motor models.

Gravity is not on this list because it is handled specially in the root links.

In RoboDyn, "widgets" are used to apply forces to links or joints. Widgets can interact with the external force and joint force variables of the links. The forces applied must be a function of the system state (position and velocity) only, no accommodation is made for internal widget states. This restriction means it is always possible to compute the widget's potential (position-related) and dissipated (velocity-related) energies. The widgets are not allowed to have internal state because that would greatly complicate the structure of the library, i.e. allowances would have to be made for arbitrary dynamical system structures.

Each widget can have a number of actuator and sensor variables. These variables are supplied/read by the user of RoboDyn at each time step. The actuators parameterize the widget effects: for example, actuators could be thruster force or motor signals. The sensor values allow the user to get feedback from the widgets. For example, a sensor could be contact force for a collision detection widget.

I.4 RoboDyn configuration file

The general structure of the configuration file is very simple. There are only two kinds of data that can be written: numbers and strings. Strings are always quoted `"like this"`. There are two kinds of data structures: arrays and associations. Arrays are written by enclosing the values in parentheses. For example, here is a four element array:

```
( 1 "foo" 4.8e-2 "bar" )
```

Associations are written by enclosing key-value pairs in curly brackets, like this:

```
{ mass 1 size 4.0 name "foo" }
```

This associates `mass` with 1, `size` with 4.0, etc. The key values are unquoted names. In both arrays and associations the values can be other arrays and associations, so complex hierarchical data structures can be constructed.

The RoboDyn configuration file is just a single un-bracketed association. All links are stored as associations in an association called `Links`. All link widgets are stored as associations in an array called `LinkWidgets`. So the configuration file has this overall structure:

```
Links{
        link_1_name { type "Link1Type" outboard_links (...) ... }
        link_2_name { type "Link2Type" outboard_links (...) ... }
        ...
}
LinkWidgets (
        { type "Widget1Type" ... }      # first link widget
        { type "Widget2Type" ... }      # second link widget
        ...
)
```

Most RoboDyn classes initialize themselves from associations in the configuration file.

I.5 RoboDyn class hierarchy

`ABSystem`	An articulated body system. This is the container for the entire AB system, in a typical simulation this is the only object you will need to make.
`ArticulatedBody`	This represents a node in an AB tree. It contains pointers to the outboard (child) ABs. It represents the joint connecting this to the parent (or inboard) AB (or none if this is the system root).
`RigidBody`	This adds mass properties to the AB: the total mass, inertia tensor and center of mass position.
`MobileRootLink`	Mobile system root link.
`Link`	This class adds a few AB implementation variables to `RigidBody`, and has abstract definitions for all the coordinate system transformation functions.
`MDHLink`	An MDH link has a single degree of freedom joint. Its transformation is defined by modified Denavit-Hartenberg parameters.

`RevoluteLink`	Revolute, or hinge-type joint.
`PrismaticLink`	Prismatic, or piston-type joint.
`BallnSocketLink`	Ball and socket link. The joint has 3 degrees of freedom of movement and 6 degrees of freedom of placement.
`FixedRootLink`	Fixed system root link.
`ZScrewLink`	This is used as a spacer to provide two extra degrees of freedom to the placement of another link (typically an MDH link).
`Widget`	The widget base class.
`GroundPointsWidget`	Collision detection between a link and the ground.
`PositionSensorWidget`	Sense the absolute position of some point in a link.
`SpringWidget`	A spring between two points in two links.
`SimpleMotorWidget`	A simple linear force joint motor.
`SpatialThrusterWidget`	An arbitrary linear and rotational force.
`JointLimiterWidget`	Spring plus damper limits on joint movement.
`RootLink`	Extra data stored for system root links.