Trajectory Input Standard Definition

Friday, February 24, 2017 12:34 PM

Purpose: Define how trajectories for the DCP should be structured

Structure: There are three primary stages that trajectories for the DCP can be broken up into. They are task trajectories (xtraj); joint trajectories (qtraj); and Simulink trajectories (slxtraj). These trajectory types are structured as follows:

- Xtraj: Structure of the form {[t],[dcp_xtraj],[at40_xtraj],[kuka_xtraj],[tool_traj],[en_traj]}
  - Each cell in a given column corresponds to a trajectory segment. All columns must have the same number of segments.

  - Segments in different columns have the following formats:
    - T:[t]. These values are timestamps for this segment, from 0 to the maximum length of time that any cell within the segment takes to execute. **This segment defines the length in time of the entire segment.**
    - Dcp_xtraj: [x(t),y(t),z(t)]. These values are relative positions, measured relative to an internal [0,0,0].
      - Dcp_xtraj = at40_xtraj + kuka_xtraj
    - At40_xtraj: [x(t),y(t),z(t)]. These values are relative positions, measured relative to an internal [0,0,0].
    - Kuka_xtraj: [x(t),y(t),z(t)]. These values are relative positions, measured relative to the KUKA's starting point.
    - Tool_traj: [h1(t), h2(t), h3(t), h4(t), h5(t), h6(t)]. These values can be used to apply whatever inputs are necessary as a function of time to the tool trajectory.
    - En_traj: [en_at40(t), en_kuka(t), en_tool(t)]. These values indicate whether a given device is enabled or disabled for this particular segment. For most devices, this is not important (e.g.simply feeding 0 position/velocity to the KUKA will cause it not to move), but will be for some (e.g. this flag can be used to turn off the AT40GW controller, to avoid sensor noise causing the system to move).

        - En_traj(n) is default high.
        - En_traj should generally be constant for the entire segment - a system should not be enabled & disabled within a given segment.

    - Any segment (except for en_traj and t) may be either:
        - Sparse - e.g., it has only one row. In this case:
            - The single row is copied to fill in the rest of the trajectory
            - Optionally, depending on the type of trajectory used, the en_traj flag will be set low. In the case of the LIFX bulb tool, the enable channel would not be set low (the bulb would just have a single color/brightness set the entire time); in the case of the AT40GW, it likely would be.
        - Full - e.g., it has a number of rows equal to [t].

- Qtraj: Structure of the form {[t], [dcp_xtraj], [at40_qtraj], [kuka_xtraj], ([kuka_qtraj]), [tool_traj], [en_traj]}
    - Qtraj is identical in structure to xtraj, except that:
        1) Dcp_xtraj[] is now absolute, measured relative to the DCP's origin;
        2) At40_qtraj[] is now absolute and in joint space for the AT40GW
        3) Kuka_xtraj[] may optionally be converted to kuka_qtraj(), but does not have to be, depending on what mode the KUKA is being controlled in. If it is not converted, kuka_qtraj = [0], and en_traj(3) = [0]
        4) En_traj has been lengthened by one element, to include kuka_qtraj.

    - The following operations need to be conducted to construct qtraj:
        - [x] 1) Identify origin of relative trajectory xtraj in workspace
        - [x] 2) Offset at40_xtraj and dcp_xtraj to that location
        - [x] 3) Convert at40_xtraj to joint space;
        - [x] 4) If required, convert kuka_xtraj to joint space
        - [ ] 5) Lengthen en_traj, and set appropriately to control kuka.

- Slxtraj: slxtraj simply takes qtraj and compacts all segments into a single vector representation.
    - At this point, the trajectory should be checked to verify that:
        - All segments are the same length;

        - Segments and enables are in agreement (e.g. there are no segments for which a trajectory contains motion, but the enable flag is set low).

Example use case, in short term:

- Img2rasttoolpath is used to create a (T X Y Z H) trajectory for the DCP. This trajectory contains a Cartesian path to execute, as well as tool values for each point along the path.
- Before passing the trajectory on, img2rasttoolpath also adds cells for at40_xtraj, kuka_xtraj, and en_xtraj.
- Xtraj2qtraj (in development) performs the following functions:
    - Offsets the trajectory to the correct location in the AT40GW's workspace
    - Converts AT40GW task trajectories to joint trajectories
- Qtraj2slxtraj (mostly built, needs renaming) compresses and checks the resulting qtraj

Example use case, in long term:

- A user has a trajectory of a structure they would like to light-paint. They generate a [T X Y Z H] trajectory for the DCP in Rhino or some other composition program.
- They import the [T X Y Z H] trajectory into dcp_toolpather (program to be developed). Dcp_toolpather allows them to:
    - Position the [T X Y Z H] trajectory within the DCP's workspace, with visualization of the trajectory & robot --> creates complete xtraj with absolute positions
    - Use one of a series of IK solvers/joint space trajectory generators to create a joint-space trajectory --> creates qtraj as needed
    - Save out a slxtraj

To-do:

- [x] Want to change everything over to structs instead of cells, so we can use names to reference specific parts of the trajectory.
- [x] Need to add back in functionality to create transition segments between sections in img2vecttoolpath
- [x] Create qtraj2slxtraj