Staubli TX90 Motion Verification Procedure

Kris Hauser

5/16/2011

# 1. Introduction

This document describes a procedure for verifying the safety of raw motion commands before executing them on a Staubli TX90L robot. These commands are executed on a remote computer and sent over the network to the CS8 controllers through the TrajClient protocol. *Only version 1.0 is currently supported*.

**Debugging Cycle.** The procedure involves a debugging cycle whereby the user runs the TrajClient program while being connected to software emulators. The user inspects the output and corrects any errors caused by the program. Once errors are eliminated, the program is finally connected to the real robot. These safeguards reduce (but do not completely eliminate) the risk of executing unsafe motion on the real robot.

**Motion Constraints.** Safety is checked against joint limits, velocity limits, acceleration limits, and collision avoidance in a static environment given as a CAD model. The checked limits are the software limits that can be controlled in the VAL3 interface (and are always less than or equal to the hardware limits).

**TrajClient Protocol.**  This simple protocol was developed to allow a remote computer to command arbitrary motions in real-time on the CS8 controller. For more information please consult the TrajClient documentation.

# 2. System Architecture

In summary, a TrajClient on the remote computer connects to a TrajServer host in order to send motion commands and to read robot state. The TrajServer can reside on a variety of devices:

* On the CS8 controller, which controls the actual robot;
* On a CS8 emulator provided in the Staubli Robotics Studio (SRS) package, which is a near-exact copy of the CS8 controller but controls a simulated robot;
* Or a virtual emulator provided in RobotSim, which also controls a simulated robot and provides collision checking.

The results of connecting a TrajClient program to an emulated controller should be approximately the same as connecting to the real controller. So, the testing procedure proceeds in reverse order: the program is verified **first on RobotSim, then the emulator, and finally on the real robot.** At each stage the user is responsible for checking the output of the server and correcting any reported errors.

## 2.A. Installation and Setup

**TrajClient** can be found on Val3Interface folder on the Desktop of the lab computer attached to the controllers. Python and C++ implementations are available. TrajServer is already installed in the SRS installation and on the CS8 controller. For use on other machines, the latest TrajClient package can be downloaded from the lab Oncourse site.

**Staubli Robotics Studio (SRS)** is distributed on a Staubli CD and can be installed on PCs running Windows XP or Vista 32 bit. Vista 64 bit and Windows 7 are not supported.

**RobotSim** (contains Val3Server, Val3Client, and SimTest) binaries are available in the Val3Interface folder, and can be built from scratch from SVN on <https://cs.indiana.edu/svn/hauserk/RobotSim>. To build the Val3Server program use ‘make Val3Server’. You may also wish to build the ‘Val3Client’ and ‘SimTest’ programs, which are useful debugging utilities. Linux, Windows under Cygwin, and MacOSX are supported.

**CS8 controllers** can only be accessed through the attached lab computer. IP addresses are listed on the notes attached to the controllers.

# 3. Testing Procedure

## 3.A. Basic Testing

The following procedure shall be used to test a TrajClient program P, residing on client machine C, before connecting P to a TrajServer on the robot R.

1. Open up a virtual TrajServer on a server machine S. (S might be equal to C). In the RobotSim directory, run ‘Val3Server –r 250 –c –w [world file] [val3 setup file] [emulator spec file]’. Ensure that the world file refers to the correct robot.
2. Run P with the TrajClient connected to the address of S (port 1000).
3. Inspect the Val3Server output on S for errors.
4. If any errors are detected, correct them in P and start from step 1.
5. Run the SRS emulator on a machine E. Run the trajServer program on the emulator.
6. Run P with the TrajClient connected to the address of E (port 1000).
7. Inspect the emulator log file for any errors.
8. If any errors exist, correct them in P and start from step 1.
9. The program passes all safety tests. Connect P to the address of R and execute the motion. *It is highly recommended to record the performance with a video camera*.

## 3.B. Visual Debugging

Motion correctness may be inspected visually using a 3D simulator. There are three procedures for doing so:

**Val3Server, online viewing.** Launch Val3Server with the ‘-v’ flag.

**Val3Server, offline viewing.** 1) When launching Val3Server, provide the flag ‘–o [file]’. The path that is produced by the TrajClient program will then be recorded to [file]. 2) Kill Val3Server after the program is finished. 3) Run ‘SimTest –path [file] [world file]’. 4) Click ‘Simulate’ to observe the physical simulation of the path.

**SRS emulator, online viewing.** 1) While the SRS emulator is running, launch the 3D Studio program from SRS. Connect 3D Studio to ‘127.0.0.1’. 2) Observe the motion while the TrajClient program is being executed.

## 3.C. Robot and Environment Models

In the Val3 directory of RobotSim, there are XML files of the following format:

1. Setup files (tx90right.xml, tx90right\_hispeed.xml), which describe a RobotSim-TrajServer mapping. tx90right.xml is a safely constrained robot file with slow movement and restrictive joint limits. This should be your standard robot setup while testing. You may wish to edit these files to provide different start-up dynamics and kinematics constraints.
2. Spec files (tx90l\_spec.xml, tx90lsafe\_spec.xml), which describe specifications for emulated controllers. It is unlikely that you will need to alter these files. Usually you will use tx90l\_spec.xml.

World models are XML files located in the RobotSim directory. Currently we only have a plane (val3plane.xml) that can be used for collision detection. The Val3Client program uses the <val3> tag of the world file to specify a Setup file and to connect to a TrajServer instance.

TODO: construct a world file for the Info E 010 room, both for the left and right robots.

# 4. Nondeterminism and Caveats

Users must be aware of the following caveats:

* Emulators usually start with a virtual robot at the zero configuration, while the real robot may be in any configuration when it starts up. **So, the user’s TrajClient program must not rely on starting at the home configuration.** To test program behavior under random initial configurations, run the Val3Server program with the ‘-i random’ flag.
* Changes in the environment or the robot will invalidate the correctness of collision tests. Most importantly:
  + **Attachments to the robot (e.g., cameras, tools) must be modeled in the robot’s CAD file** to ensure collisions are detected.
  + **The other robot arm must be positioned outside of the robot’s workspace** so that it does not have a chance to impede the robot’s movement. *The controller has no knowledge of the other arm’s configuration*.
* If Val3Server is run on a slow machine or the environment model is very complex, then its performance may suffer. In particular, it may miss collisions between configurations, and network performance may be slow.
* Because Val3Server is programmed in a completely different language it has a chance of being behaving differently from the CS8 controller. (We intend to correct these problems as they arise as the system is used more thoroughly, which will eventually eliminate this caveat.) The emulator provided by SRS runs the VAL3 language natively, so results should be identical to those of the CS8 controller.
* The network behavior of emulators may be different from the CS8. More precisely, PC communication may be lower latency than that of the CS8. The CS8 also has some internal buffering scheme that is not identical to that of PCs.
* The SRS emulator does not provide real-time performance. It has an undocumented, hard-coded “batch processing” behavior in which the emulator waits until 0.1s physical clock time elapses, and then processes controller cycles in 0.1s batches. *So, errors in this stage of the procedure might be judiciously ignored if your system relies on very fine-grained temporal feedback. If you choose to ignore such errors, proceed with caution.*

**Important: if your program performs real-time feedback control on the order of 5Hz or higher, network and emulation delays have a very real effect on performance. Do not proceed without careful consultation.**