Tarsim: A Realtime Frontend Integratable and Configurable Robot Kinematics Simulator

Kamran Shamaei, PhD

1. Introduction

Tarsim is a kinematics simulator that can simulate the kinematics of any open-chain rigid body system. Tarsim can simulate any robotic arm, robotic hand, humanoid robot, bipedal robot, Cartesian stages, and any other robot that is composed of an open-chain system of rigid bodies. The current version of Tarsim cannot simulate closed-chain rigid bodies such as delta robots, four-bar linkages and other similar mechanisms. The main features of Tarsim include:

1) Tarsim is a realtime-safe simulator. Setting and getting robot kinematics parameters takes a few micro seconds and the kinematics engine can calculate the whole kinematics chain in a few micro seconds.
2) Tarsim is integratable. The client of Tarsim can be integrated in any C++ code running under a POSIX system such as Linux Ubunutu. The service of Tarsim can be run as a separate process or be integrated in a Qt application.
3) It is easy to set up a robot in Tarsim. You can set up your robot in Tarsim in a couple of hours by creating a configuration file and dropping the STL files of your robot.
4) Tarsim window is configurable. Almost every behavior of Tarsim can be configured such as its background color, position of buttons, button icons, and other window properties.
5) Tarsim has a very small foot-print. Tarsim uses very small memory and cpu. For example under an i7 core it uses ~20 mb of memory and 1-2% cpu.
6) Multiple clients can communicate with Tarsim and move different parts of the robot or robots.
7) Tatrsim simulates collisions, max joint velocity, and max joint acceleration.
8) Tarsim allows you to simulate the exact commands you send to the robot while the robot is running or without the robot running. For example, you can send the position commands that you send to your robot to Tarsim to display your robot. Also you can create a shadow robot along with your robot to display where the robot actually is by sending the feedback you get from your robot.
9) Tarsim keeps track of time and reports if the update rate from the software deviates from what it actually must be by a settable tolerance.

These features in addition to many other features integrated into Tarsim, makes it an ideal simulator for roboticists who are developing software and algorithms for a robot without actually running their code on the actual robot. This becomes even more important because robots are inherently expensive to purchase, develop, and use making it very difficult for many roboticists to have extensive tests on a robot. Especially, nowadays there are so many engineers who work off-site and do not have daily access to robots. Moreover, there are so many engineers who develop software for a robot, but they don't

really need to test their software on an actual robot but need to know how the robot would behave such as UI/UX developers. Tarsim facilitates all these scenarios.

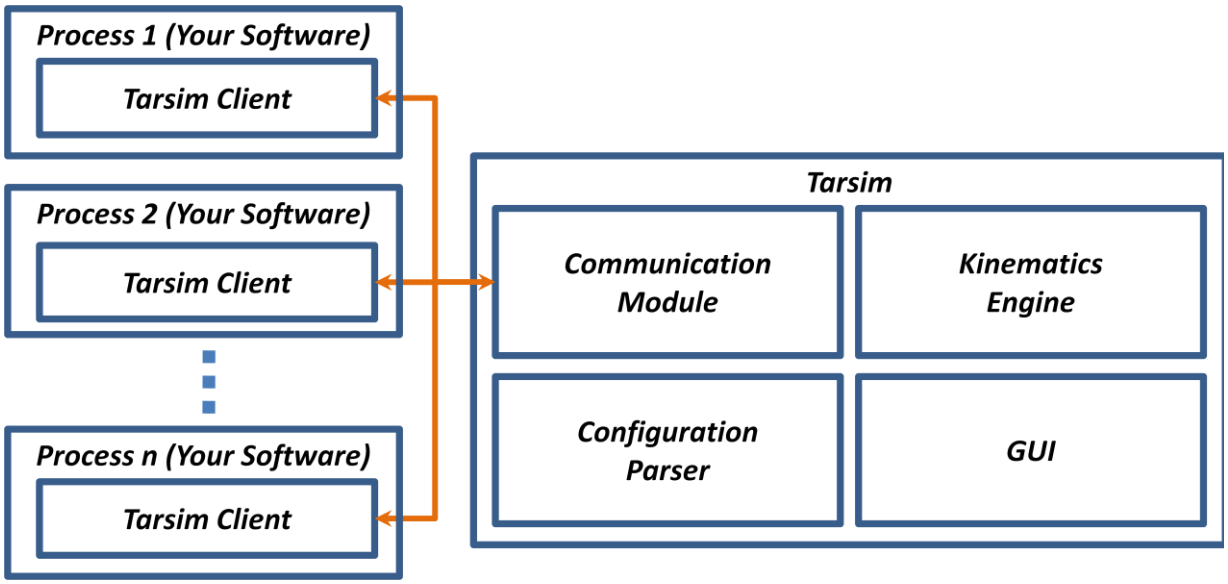

Fig. 1 Tarsim relies on a client-service design. Tarsim service receives messages from its client and executes them. Tarsim client is employed inside your code to communicate with the simulator the same way you communicate with your robot.

2. Design

Tarsim relies on a client-service mechanism as shown in Fig. 1. Tarsim service can be run standalone or inside your software as a process or integrated in your Qt application. Tarsim client is a C++ class that can be instantiated in your code to communicate with Tarsim service.

Tarsim service has multiple components:

1. Communication Module: This module employs a series of classes to communicate with the outside world. It relies on a POSIX message queue mechanism to communicate.
2. Kinematics Engine: It is responsible for all kinematics calculations. It has an event-based mechanism in that it only performs kinematics calculations when there is a message from the client asking for it.
3. Configuration Parser: All the properties of the rigid body system (robot) and the window (what appears to the user) are processed by this module. It employs Google protobuf to process the configuration data.
4. GUI: It translates the robot kinematics to a graphical representation for the user. It relies on VTK graphics package. The gui has multiple scenes including:
    a. Robot scene: The main scene where the robot is displayed.
    b. Buttons scene: A series of buttons that can be used by the user:

i. View Reset
ii. Show/Hide Frames
iii. Show/Hide Planes
iv. Show/Hide Lines

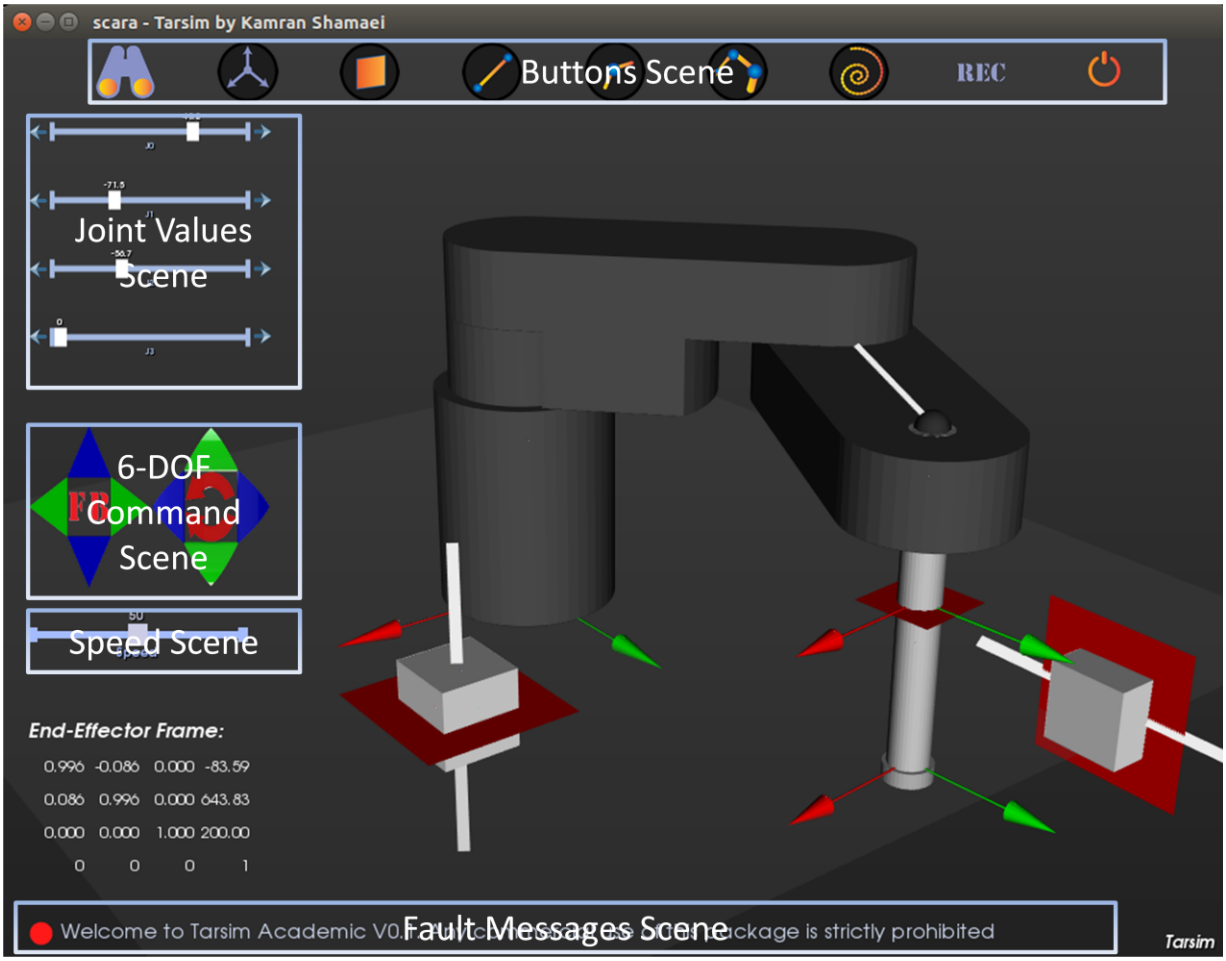

Fig. 2 Tarsim GUI is composed of a series of scenes. The scenes have different functionalities. The robot shown above is a Kuka KR6 robot with the CAD models from www.grabcad.com

v. Show/Hide Points/Spheres
vi. Show/Hide CAD model
vii. Show/Hide End-Effector Path
viii. Record button: The scene will be recorded as PNG files under ~/tarsimRecord folder
ix. Exit

c. Joint values scene: It displays one slider per robot joint that can be used to set the robot joint values as well as one increment and one decrement button per joint that can be used for joint jogging.
d. Fault scene: This scene display potential faults. For example if you set the control cycle of your software to 1 ms and a tolerance of 0.1 ms, Tarsim will show a message every

time this requirement is not met. The user can also send text messages to be shown on this scene through Tarsim client.

e. Speed scene: This slider can be used to set the robot pace.

f. End-effector pose scene: It merely reports the end-effector frame with respect to the

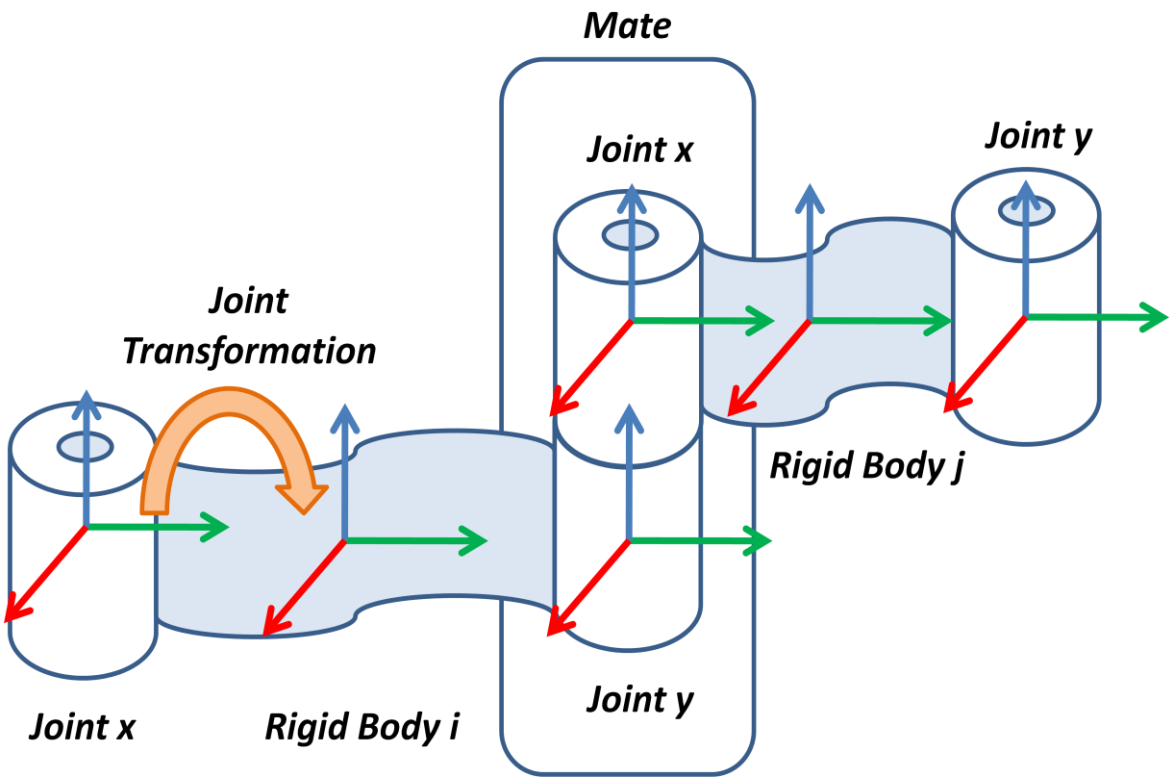

Fig. 3 A robot is modeled as a system of rigid bodies mated together. Each mate is composed of two joints with their z-axes along each other. The joint value is defined as the angular or linear distance between the x-axes of the joints.

world frame.

g. 6 DOF incremental command scene: This scene has a 12 buttons that can be used to jog in the space. It has one increment and one decrement button per degree of freedom. The status of these buttons can be retrieved using Tarsim client every control cycle.

3. How to Use

Tarsim is designed for POSIX systems. These instructions assume you are running Tarsim in Linux Ubuntu.

3.1 Robot Configuration

Tarsim interprets a robot as a rigid body system, as shown in Fig. 3. Every robot link is a rigid body that can have a number of joints. A robot joint is created when a joint of a rigid body is mated with a joint of another rigid body. Therefore, a robot joint in fact is a mate in the rigid body system that represents the

robot. A rigid body has a pose frame that defines where the rigid body is in space. All other frames of the rigid body are defined with respect to this frame. A rigid body joint has a coordinate frame with its z-axis along the joint axis both for revolute and prismatic joints. The x and y axes of the joint can be chosen arbitrarily. Every rigid body also has an appearance that defines how it should appear on the robot scene of Tarsim. The appearance can include a CAD model with a STL format in mm units or a series of geometrical items such as points/spheres, lines, and planes. All these appearances are defined in the coordinate frame of the robot. To define a robot you need to follow these steps (Follow one of the examples in samples folder and refer to rbs.proto file that explains every field):

1. Identify all the rigid bodies of the robot. One rigid body should be fixed with respect to the world frame
2. Assign a coordinate frame on each rigid body
3. Assign a coordinate frame on each joint and calculate its transformation with respect to the rigid body coordinate frame
4. Create a folder for the robot
5. Create a file for the rigid body system with name: rbs.txt
6. For each rigid body create a rigid_bodies{} in the rbs file.
7. Create one mates {} per robot joint in rbs.txt file.
8. Drop the STL file of the robot if you have them in the same folder.
9. Define other configurations such as the camera, collision detection, control cycle and it tolerance, etc.

3.2 GUI Configuration

Copy win.txt from one of the sample into the folder you created for your robot. If you like to change the appearance of your gui follow this section otherwise just skip the rest. The file win.txt is composed of a series of scenes. Each scene has a number of characteristics such as color, transparency, location, font data, sliders and buttons data. File win.proto details what each of these parameters do. If you like to change the button icons or background images, it is straight forward, jut replace the icon image with your desired image; just make sure the format is the same. The size does not have to be the same, but if you choose a picture with a very large size, it might affect the performance of Tarsim. Tarsim does not have any icon intentionally so that it allows the users to choose their own software icon. To do so, all you have to do is to create a tarsm.desktop file that defines what icon should be used.

3.3 Run Tarsim Service

To run Tarsim, you need to point to the location of the configuration folder you created that at least includes one win.txt and one rbs.txt files as:

```
/path/to/tarsim/tarsim -c /path/to/configuration/folder
```

Tarsim also create a log filder under the /tmp folder. If you run into issues, it is always good to check that file.

3.4 Tarsim Client

Tarsim client is a C++ library that is available in folder user. You can integrate the library into your C++ code. Tarsim package includes a clientExample folder that shows how you can use Tarsim client in your code. You can also refer to tarsimClient.h that includes more details on each API that can be used. Note that Tarsim client can communicate only after Tarsim is up and running.

3.5 Qt Application Integration

Tarim service is available as a C++ class that can be integrated into a Qt application as schematically shown in Fig. 4. To do so, you need to make sure that VTK is installed with Qt modules. Please follow the instruction on VTK website for installation with Qt widgets. Under samples folder, there is a folder named qt that includes a simple example of how to integrate Tarsim into your Qt application.

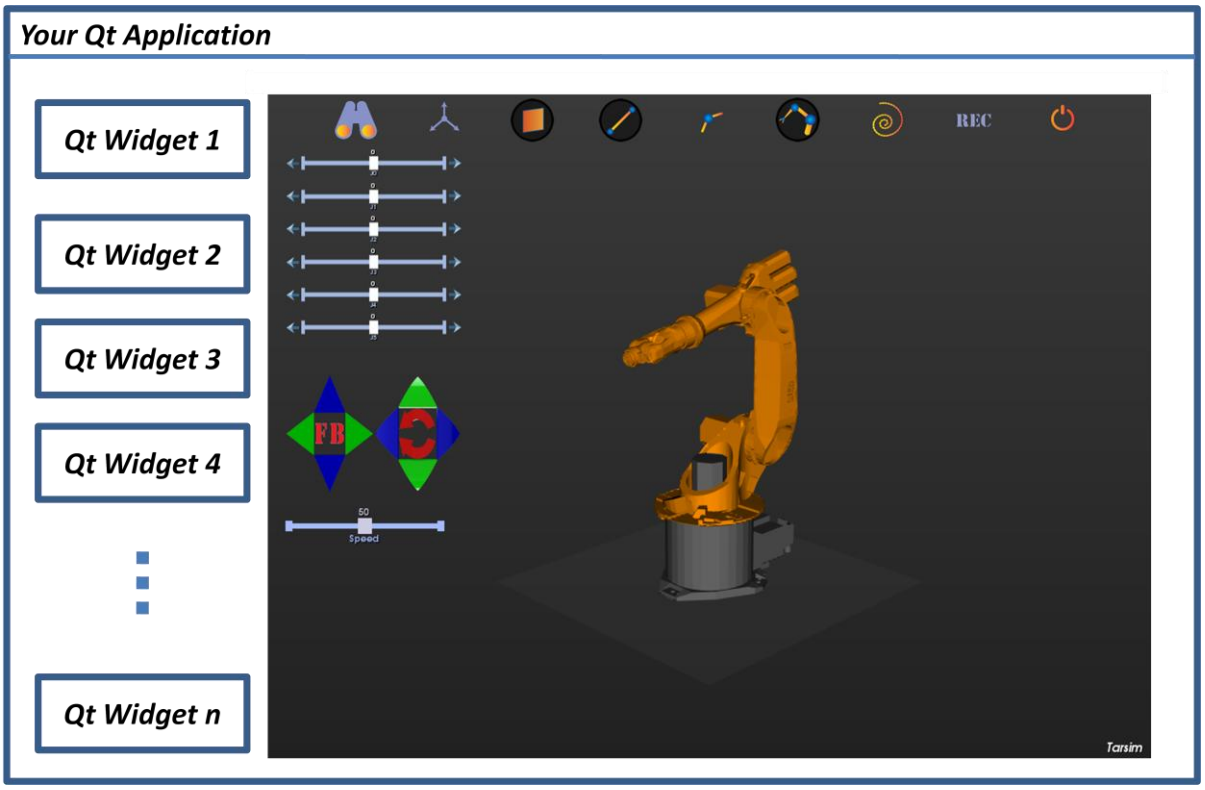

Fig. 4 A schematic of how Tarsim can be integrated into your Qt application. The robot shown above is a Kuka KR6 robot with the CAD models from www.grabcad.com