nVIZ Virtual Test Drive Server

Unity Plugin - User Guide

Release 2018.5 Beta 2

August 30th , 2018

Contents

1. Introduction

nVIZ GmbH provides a plugin DLL for Unity 3D. This plugin can be used to communicate with Virtual Test Drive (VTD) server at runtime. VTD Server provides all the quantities in SI units in right-hand coordinate system as received from simulation solver software e.g. IPG CarMaker. These values can be accessed through the plugin as they are received from the VTD server or they can also be accessed in the modified form ready to be used directly in Unity 3D scene i.e. left hand coordinate system.

The plugin can also be used to directly apply all the transformations to the vehicle and the camera. The plugin will read the vehicle rig from a text file and apply all the transformations to the respective nodes defined in the text file.

Initial setup and some important instructions are discussed in the next section. All the different ways to use the data received from VTD server in Unity are discussed in the following sections.

2. Setup Guide

- A DLL named "**unityVtdPlugin.dll**" is provided by nVIZ GmbH to allow communication of VTD Server with scripts in Unity3D.
- The Dll can be found under:
  "C:\Program Files\nVIZ\VirtualTestDrive\renderengines\unity\"
- The Dll needs to be added to the Unity project under:
  "<Unity Project Directory> / Assets / Plugins"
- Unity will automatically load the Dll to the project. It can be verified by checking "References" in the Visual Studio Project.
- User script in the Unity project will have to include the namespace "unityVtdPlugin" with the following command:
  "using unityVtdPlugin;"

Important Note

- All the values received from CarMaker are in Z-Up right hand coordinate system, so they must be handled accordingly if using the original data.
- The DLL is targeted at ".NET runtime 4.x", so it must be set in the unity project settings
  - (Edit --> Project Settings --> Player --> Other Settings --> Configuration --> Scripting Runtime Version)
  - This option is only available since Unity 2017.1 (according to the documentation of unity).

Specifications of Our Development Setup

- Visual Studio 2017
- Unity 2018.2.0f2
- .NET Runtime Framework 4.7.2
- .NET Developer Pack Framework 4.7.1

3. Using Original Data

In order to setup the communication between unity plugin and Virtual Test Drive server, the user needs to know the IP address of the system on which VTD server is running. Also, the port number selected for render engine in VTD server is required for setting up the communication. This can be found under "Render Engine" tab on VTD Server GUI as "UDP Port".

- Class "NvizDataParser" is responsible for communication with VTD Server via UDP connection.
- The constructor of this class will take 2 parameters (IP Address of the machine running VTD Server & the port number).
- The constructor will do all the initializations for the communication.
- Struct "CarMakerData" is defined in the Dll which is used to provide the data received from VTD server to the user defined scripts.
- Member function "**getData()**" will return the struct "CarMakerData" with all quantities subscribed from IPG CarMaker.
- After receiving data, user script can use any quantity as required.
- User script will have to call this function in a loop/ Update / FixedUpdate to get new data periodically.
- Please note that all the values are in Z-Up, right hand coordinate system. So, they must be handled accordingly in this case.
- Please note that the function uses synchronous UDP connection with a timeout of 1 second, it returns fail signal in case of timeout.
- The boolean variable in CarMakerData named "successful" will reflect the status "false", if it has failed to receive data.
- Please note that the quantity variables might have garbage values in case of failure. So, it is important to check the status before using quantities.
- Communication will be closed in the destructor of "NvizDataParser" class.
- The member function "**getDataString()**" can be used to get the original data in comma separated string format.

The details of the functions are given below:

- getData()

Returns an instance of CarMakerData struct which contains all the data received form the VTD server. It also includes a Boolean variable named "successful" which shows the status of the communication between the plugin and the VTD server. If the communication is down, the variable will be false. In this case, the data fields in the struct might have garbage values.

Input Parameters:	---
Return Value:	CarMakerData

- getDataString()

This function can be used to get the original data received from VTD server in a comma separated string format. Return value of this function is of type *string*.

Input Parameters:	---
Return Value:	String

4. Using Modified Data

Setup for this case is also similar to the one used to get original data. IP address and port number will be required of the system on which Virtual Test Drive server is running. This IP address and port number will be passed to the constructor of “NvizDataParser” class. It will initialize the communication with VTD server. A number of functions are provided by this class which provide modified data which can be directly used by the user script to drive the vehicle in real time.

Member function “**getData()**” still needs to be called before accessing any other function in every iteration because it will communicate with VTD server and update the data which will be modified and returned by other functions. The functions which provide modified values are as follows:

- **getFR1Position()**

  Returns the exact coordinates of the vehicle reference system FR1 at the specific time. The return value is of type *Vector3* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Vector3

- **getFR1Rotation()**

  Returns the rotations of the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- **getWheelFRPosition()**

  Returns the translation coordinates of the front right wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Vector3* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Vector3

- **getWheelFRRotation3d()**

  Returns the rotation of the rotating part of front right wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- **getWheelFRRotation2d()**

  Returns the rotation of the non-rotating part of front right wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- ### getWheelFLPosition()

  Returns the translation coordinates of the front left wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Vector3* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Vector3

- ### getWheelFLRotation3d()

  Returns the rotation of the rotating part of front left wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- ### getWheelFLRotation2d()

  Returns the rotation of the non-rotating part of front left wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- ### getWheelRRPosition()

  Returns the translation coordinates of the rear right wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Vector3* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Vector3

- ### getWheelRRRotation3d()

  Returns the rotation of the rotating part of rear right wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- ### getWheelRRRotation2d()

  Returns the rotation of the non-rotating part of rear right wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- **getWheelRLPosition()**

  Returns the translation coordinates of the rear left wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Vector3* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Vector3

- **getWheelRLRotation3d()**

  Returns the rotation of the rotating part of rear left wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

- **getWheelRLRotation2d()**

  Returns the rotation of the non-rotating part of rear left wheel relative to the vehicle reference system FR1 at the specific time. The return value is of type *Quaternion* defined in *UnityEngine* namespace.

Input Parameters:	---
Return Value:	Quaternion

5. Using Transformer Class

The plugin also provides "Transformer" class which applies all the transformations to the respective nodes to simplify the integration of Virtual Test Drive server with Unity 3D. Just like other cases, this class will also need the IP address and the port number of the Virtual Test Drive server. The constructor will try to read vehicle rig from the assets of Unity Project. This file must be saved with the following path:

"<Unity Project Directory>/Assets/Resources/defaultVehicleRig.txt"

A sample file can be found under the given path in the example project. The constructor of the class will try to read the file and find every node in the unity scene. It will print out warnings if one or more nodes are not found in the scene hierarchy. If the file read was successful, it will initialize the communication with VTD server and save the initial translation and rotation of all the nodes, so that they can be handled accordingly later on.

The member function "**updateScene()**" can be used to receive the latest data form VTD server and apply it to all the nodes. There are also a few other functions that can be used for some settings. All these functions are explained below.

- updateScene()

This function sends a request to VTD server to get latest data. Once the data is received, it transforms the camera and the vehicle according to the received data. If there is any problem in the communication and the data is not received from Virtual Test drive server, then it will not update the camera and the vehicle.

The function will return true, if data was received from VTD server and returns false if there was a problem in the communication. The return type is "*bool*".

Input Parameters:	---
Return Value:	Boolean status

- setCameraDOF(Vector3 , Vector3)

This function can be used to set the freedom of movement of the camera. First parameter of the function is of type *Vector3* defined in *UnityEngine* namespace. It includes 3 multiplication factors for X, Y & Z, typically ranging between [0-1]. Each factor will define the freedom of movement of the camera in the respective direction. For example, translation vector (1, 0, 0.5) will move the camera along with vehicle in X-direction but it will not move in the Y-direction. However, in the Z-direction, camera will move only half as much as the vehicle.

Second parameter of the function is also of type Vector3 defined in UnityEngine namespace. This parameter defines the freedom of rotation of the camera. It includes 3 multiplication factors for Roll, Pitch & Yaw, typically ranging between [0-1]. For example, rotation vector (0,0,1) will not apply any roll or pitch on the camera but only the yaw. This will work similar to the dolly view camera. For the driver view, all the 6 DOF should be set to 1, so that the driver feels all the translations and rotations similar to that of vehicle.

Input Parameters:	Vector3 Translation Factors, Vector3 Rotation Factors
Return Value:	---

- **setSteeringWheelGain(float)**

  Steering Wheel gain can be adjusted by using this function to match with the actual model of the vehicle. Typically, this value should be 1.

Input Parameters:	float
Return Value:	---

- **setTachoNeedleGain(float)**

  The gain for needle of tachometer can be adjusted by using this function to match with the actual model of the vehicle. The function can also take a negative value, if the needle needs to be rotated in the opposite direction.

Input Parameters:	float
Return Value:	---

- **setRevCountNeedleGain(float)**

  The gain for needle of revolution counter meter can be adjusted by using this function to match with the actual model of the vehicle. The function can also take a negative value, if the needle needs to be rotated in the opposite direction.

Input Parameters:	float
Return Value:	---