Manual for the use of the kinematic model in Unity

Begnis Ruben, Benedetti Isacco,
Buonocore Pasquale, Fadini Gabriele

December 19, 2018

1 Kinematic and Controller

This Unity project allows to simulate the dynamic of a four wheeled car.
Good results with this scripts were obtained in test for the car-like front steering rear traction model and for the differential tricycle-like kinematic model.
The main script Controller is attached to the GameObject named "vehicle" in the SampleScene.
The parent object "vehicle" has a rigidBody component with the main characteristic of the vehicle such as the overall mass, drag, angular drag.
The "vehicle" is composed by a visual body, named "truck", and another child, "wheels", containing the wheel colliders as shown in Fig. 1.
Each wheel collider parameters, such as the spring stiffness, can be tuned in-

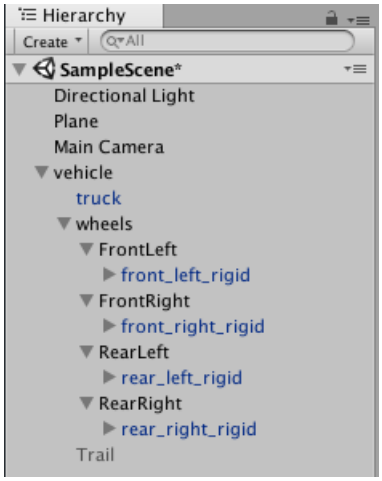

Figure 1: Hierarchy of the project

dependently, and this allows a great variety of setups for the vehicle. Moreover, the kinematic of the vehicle is completely general: each wheel can be both motor and steering. To set such behaviour it's necessary to tick the box from the inspector the tab of the script attached to the vehicle object Fig. 2.
The object "truck " is simply a mesh representing the main body of the vehicle,

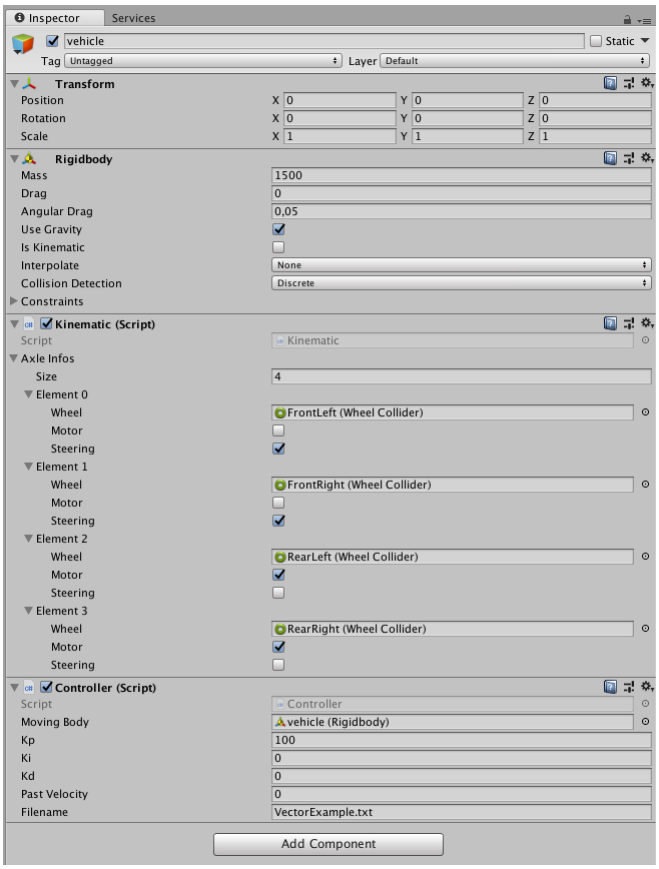

Figure 2: Components of the vehicle game object

while the "wheels" contains the components that allow the real physical simulation of the chassis dynamics in particular the shock absorber, the steering and motor of each wheel.
To simulate the dynamic of the wheels it's enough to modify the values of the applied torque and the steering angle in the Controller script attached to the "vehicle" gameObject.
The values will be modified in the Controller script, while the Kinematic will find the appropriate wheel collider and update the values in real time.
Each wheel collider is identified by in Unity a unique tag, selectable from the inspector, namely:

- Front_right
- Front_left
- Rear_right
- Rear_left

The values of each steering angle and Torque can be modified continuously in time.

In order for the script Kinematic to work (to find and access the values associated to the 4 wheel colliders) such tags must be left attached.

The Torque is the value $[Nm]$ applied to the wheel with respect to the rotary axis.
The Steering angle is the angle [°] that the wheel has with respect to the local reference frame attached to the chassis.
Thus, a positive angle applied to the front wheels will make the car turn right and a negative value to turn to the left.
To specify the wheel to which change the value, access the value of the Steering angle from Controller with the names:

- Kinematic.SteeringFrontLeft
- Kinematic.SteeringFrontRight
- Kinematic.SteeringRearLeft
- Kinematic.SteeringRearRight

To specify the wheel to which change the value, access the value of the Torque from Controller with the names:

- Kinematic.TorqueFrontLeft
- Kinematic.TorqueFrontRight
- Kinematic.TorqueRearLeft
- Kinematic.TorqueRearRight

By default, all of the aforementioned values are initialized to 0.

Solution to possible errors Please notice that not every possible assigned values set is physically sound.
Be aware that in some cases kinematic singularities are possible, and this may result in undesirable results in the simulation.
That may happen for instance if the wheels point towards different directions, an thus the vehicle becomes disarticulated. Or else, if the angle between the front and the rear wheels is close to the value of 90°, in a rear traction vehicle, then the front wheels drag force will be very high: depending on the torque provided to the vehicle either the motion will be impossible or else the vehicle will move with great vibrations and unstable motion.
To avoid this it's sufficient to properly set the constraints between the wheels, for example impose the same value to the front steering angle in a front steering car-like or use a more sophisticated Ackermann steering algorithm, after calculating the instantaneous center of rotation.
Moreover it's advisable to set a range of maximum and minimum values to the steering angle and the torque.
Lastly, for debugging, the use of the function Debug.Log to print desired value on the console.

2 PID function

If needed, a PIDControl is provided in Controller: such function, given a target, for instance the desired forward velocity of the vehicle, provides in runtime the Torque to the wheel in order to achieve it and corrects it frame by frame. In order for the algorithm to work, the actual velocity of the vehicle, and the velocity at the previous time frame must be given to the function. From that, by difference, the error from the target is calculated.
The output torque τ is a float value obtained with the sum of three components:

1. proportional to the error $e(t)$, with constant K_p
2. proportional to the derivative of the error $e(t)$, with constant K_d
3. proportional to the integral of the error $e(t)$, with constant K_i

where $e(t) = v_{ref}(t) - v(t)$ is the difference between the reference value v_{ref} and the actual value v.

$$\tau = K_p e(t) + K_i \int e(t)dt + K_d \dot{e}$$

3 Camera script

A simple camera script cameraController is provided in order to test and debug. It's attached to the main Camera in the Sample Scene, two main behaviour can be selected for the Unity inspector:

- Follow position
- Follow rotation

The first will keep the distance between the vehicle named object and the main camera constant, while the rotation is free.
On the other hand, the second will follow the rotation of the body in the space by copying and utilizing its quaternion vector.

4 Read from file function

Use the function Init from the script ReadTrajectory to import data in Unity.
The source of such function is a text file that has to be placed in the root directory of the Unity project.
This filepath must coincide with the root path of the project, for example if you saved your project as "MyProject", then the file to be read must be in "./MyProject/" not in the Script subfolder nor in any other subfolder.
The filename can be changed in the Unity console directly from the inspector of the Controller.
The format of the vector must divided by commas, while decimal separators are points. For example

```
(1.0,    1.0,    1.0)
(0.0,    0.0,    0.0)
```

The format of each line can either be in parentheses or not. So also the following is accepted:

```
1.0,    1.0,    1.0
0.0,    0.0,    0.0
```

5 Adding other wheels

If for some particular reason another wheel has to be added, two cases may arise:

- If it's an idle wheel with fixed axis or freely rotating one can attach a wheel collider to the main body through a proper unity constraint. For example a caster wheel can be obtained with revolute joint and a wheel collider attached to the vehicle body (and not to the visual body).
- If it's a controlled steering or motor wheel instead, please tag it with a proper non redundant name from the Unity inspector, for example "New_wheel". Then in the Kinematic script add two new variables of type float for the desired torque and steering angle.

```
public static float newTorque;
public static float newSteering;
```

Remember to change the values in the axle object, the tagged object will be selected and their steering angle changed in accordance to the value of newSteering and newTorque

```
foreach (AxleInfo axleInfo in axleInfos)
{
 if (axleInfo.steering)
  {
   switch (axleInfo.Wheel.tag)
   {
    ...
    case "New_wheel":
     axleInfo.Wheel.steerAngle = newSteering;
    break;
   }
  }
 if (axleInfo.motor)
  {
   switch (axleInfo.Wheel.tag)
    {
     ...
     case "New_wheel":
      axleInfo.Wheel.motorTorque = newTorque;
     break;
```

```
        }
    }
}
```

¡

Then it will be sufficient to change the value of *Kinematics.newSteering* and *Kinematics.newTorque* from the Controller script along the values of the other wheels.