

VBI-2D

User guide

Author: Daniel Cantero
Contact: daniel.cantero@ntnu.no

Document version: 02 February 2023

Summary

This document contains information about the Vehicle-Bridge Interaction model in 2 dimensions (VBI-2D) by Daniel Cantero. With this software, the user can easily define a traffic event with several vehicles crossing a bridge. The simulation includes the effects of road profile irregularities and the interaction between vehicle(s) and bridge. The software is designed to allow the user to define a huge range of possibilities. The traffic event can be composed of several vehicles, each represented by different models. The user can specify the vehicle(s) initial position, velocity, and acceleration. These vehicles travel on a road profile, which can be generated according to standards, given a specific geometry, or loaded from existing profiles. The bridge is modelled as a beam using a Finite Element formulation. It is possible to define multiple number and types of boundary conditions. The software even allows the representation of damage in the structure or supports. The whole vehicle-bridge system is solved numerically by direct integration. The results of the simulation are stored in variables and can easily be retrieved by the user. The software also offers a range of calculation options and graphical outputs.

This document provides a detailed description of each aspect in the model (Vehicle, Profile, Bridge, Calculation) together with relevant options, comments, and recommendations. Then it guides the user on how to use the software for the first time. The document finishes with a compendium of relevant comments regarding various aspects of the problem implementation. Note, however, that this document provides little or no explanation of the underlying calculations.

The program is written in Matlab. It relies on the use of a variable type called “structures”. The user needs to be familiar with this variable type before running the program.

License

This software has been licensed under the GNU General Public License v3.0. Author: Daniel Cantero (daniel.cantero@ntnu.no). However, it is encouraged to contact the author for help, modifications and collaborations using this software.

Warranty and liability

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the copyright owner or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

1. Introduction

The software performs the dynamic analysis of vehicles moving along a road profile and traversing a bridge for 2-dimensional models. For short, the acronym VBI-2D (Vehicle-Bridge Interaction) is used to refer to the software throughout this document.

The following text explains the relevant details for somebody to be able to use the model. This is, only the input, options and output will be discussed. Little or no explanation is provided of the underlying calculations. The program is written in Matlab. It relies on the use of a variable type called “structures”. The user needs to be familiar with this variable type before running the program.

The text is divided into two sections, namely, model description and a guide on how to run the model. The model description section presents an explanation of the implemented vehicle(s) model (Section 2.1), possible road profile definitions (Section 2.2), options for bridge definition (Section 2.3), and numerical procedure to solve the interaction (Section 2.4). The guide in Section 3 aims to give indications on how to use the software for the first time and describe the modelling alternatives available to the user. For completeness, the document finishes with some additional notes and comments elaborating on some aspects of the model and the theoretical background.

2. Model description

Figure 1 presents an overview of the model implemented in VBI-2D. These scripts can be used to model the 2D dynamic response of vehicles traversing a beam. The current implementation offers a range of possibilities. One or more vehicles can cross the bridge with different velocities and/or travelling direction. The velocities for each vehicle can vary during the event. The vehicle model can be varied from very simple 1-DOF to more complex and elaborate configurations. Road profile can be included. The bridge is modelled as a beam, but its support conditions can be arbitrarily defined, ranging from simply supported configuration to multiple spans with elastic supports. The model offers also different possibilities to solve the interaction procedure.

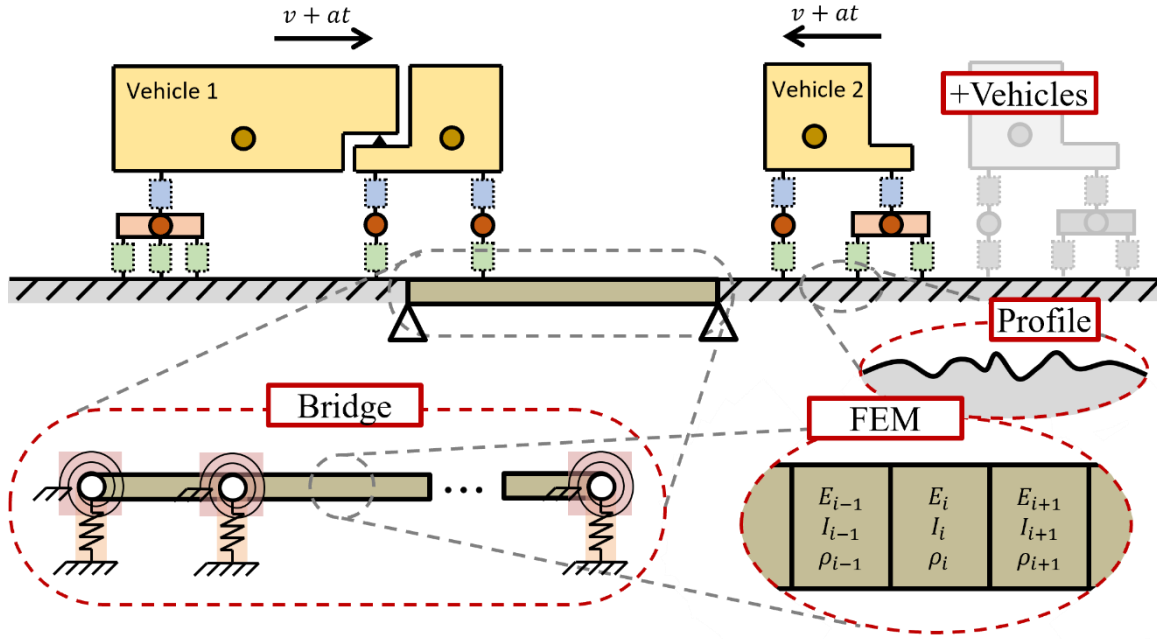


Figure 1: VBI-2D model overview

VBI-2D simulates the responses of traffic events crossing the bridge. A traffic event might consist of multiple vehicles. These traffic events might consist of several vehicles crossing the bridge at the same time or even vehicles meeting on the bridge. The possibilities are almost infinite. To define the desired traffic event to be simulated the user needs to specify travelling direction, acceleration, speed, and initial position. The reference system for these definitions is centred at the left end of the bridge as show in Figure 2, where negative values of the x-coordinate correspond to location on the left side of the bridge.

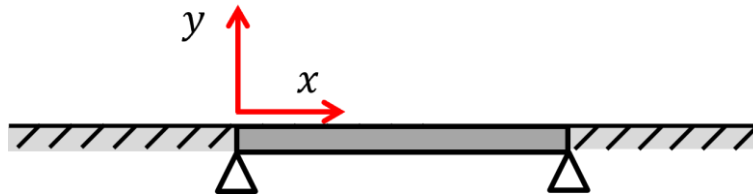


Figure 2: Global reference system for VBI-2D

2.1. Vehicle model

The equations of motion of particular vehicle models are defined in separate scripts located in the folder *Vehicle_equations*. This folder includes already a range of predefined models. These models have been generated using VEqMon2D tool [1].

Use VEqMon2D tool to generate other vehicle models with different bodies and axle configurations. With this tool it is possible to generate the equations of motion for a large range of vehicle configuration possibilities. The user can specify the number of bodies, the existence of articulations between bodies, number of suspension systems for each body, and number of axles for each suspension by creating single axles or grouped axles. See Figure 3 for a schematic overview of vehicle modelling possibilities.

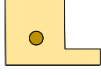
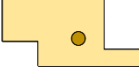
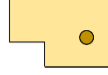
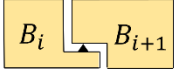
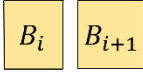




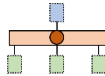



Body (B_i)	$i = 1$  + $i = 2$  ... $i = n_B$ 	
Articulation	Yes = 	No = 
Suspension (S_j)	$j = 1$  + $j = 2$  ... $j = n_s$ 	
Axle/Group (A_j)	Single axle = 	Axle group = 
Tyre (T_k)	$k = 1$  + $k = 2$  ... $k = n_T$ 	

Figure 3: Overview of vehicle modelling possibilities (Source: [1])

The resulting equations of motion are given in terms of the mechanical properties and geometrical definitions for each of the elements of a vehicle model (bodies, articulations, suspensions and axles). A body is defined in terms its mass m_{B_i} and moment of inertia I_{B_i} , and corresponding m_{G_j} and I_{G_j} for the body of eventual axle groups. Spring and dashpot systems characterized by linear spring stiffness and viscous damping define the suspensions (k_{S_j}, c_{S_j}) and tyres (k_{k_j}, c_{k_j}) systems. The locations of all these elements are expressed in terms geometric parameters centre following the notation presented in Figure 4.

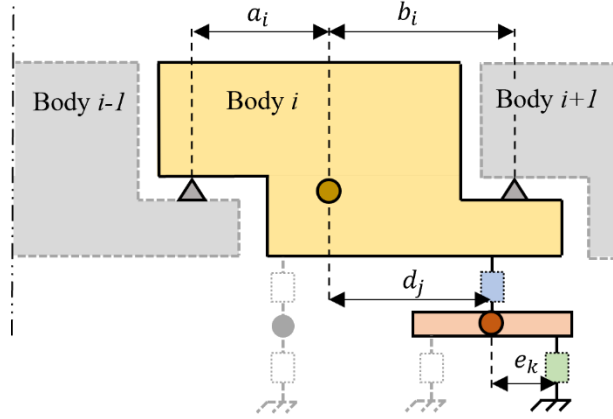


Figure 4: Notation for parameters defining the vehicle's geometry (Source: [1])

The outcome of VEqMon2D is the formulation of the equations of motion of the desired vehicle configuration. This formulation is provided in form of a Matlab function compatible with VBI-2D.

The notation followed to name any possible vehicle model is as follows:

Vehicle + Num. of axles body 1 + A (if articulation) + Number of axles body 2 + ...

Therefore, a typical 5-axle articulated truck is defined with the script *Vehicle_3A2*.

The vehicle model also offers the possibility of having axle groups. Then more than one axles are linked together into one suspension. In this case the previous notation is extended to:

$$\text{Previous notation} + G + \text{Num. axle group 1} + \text{Num. axles group 2} + \dots$$

Therefore, a typical 5-axle articulated truck can be described with the model in *Vehicle_3A2_G_1_2_2*.

Every vehicle function generated with VEqMon2D contains additional to help the user. The function has comments indicating what are the vehicle model characteristics and what are the model variables. When the function is executed, it generates the system matrices that define the equations of motion of the vehicle for the particular set of numerical values. In addition, the function lists the names and characteristics of the degrees of freedom (DOF) that describe the system. The possible degrees of freedom are defined in the order as shown in Table 1.

Table 1: Possible DOFs and order for a vehicle model

DOF	Description
y_{B1}	Vert. displacement Body 1
θ_{B1}	Rotation Body 1
y_{B2}	Vert. displacement Body 2
θ_{B2}	Rotation Body 2
...	Additional, if more Bodies
y_{S1}	Vert. displacement Suspension 1
y_{S2}	Vert. displacement Suspension 2
...	Additional, if more suspensions
θ_{S1}	Rotation Suspension 1
θ_{S2}	Rotation Suspension 2
...	Addition, if more suspension groups

Depending on the vehicle model, some DOF might not be defined. For example:

- A vehicle with no axle groups has no θ_s
- A two body articulated vehicle has no y_{B2} , because this DOF can be expressed in terms of the others (it is a dependent DOF)

Figure 5 shows the DOF numbering for an articulated truck with a total of 5 axles. The notation of the truck model is *Vehicle_3A2_G_1_2_2* (Body 1 with 3 axles, Articulation, Body 2 with 2 axles, Group 1 with 1 axle, Group 2 with 2 axles, Group 3 with 2 axles). The DOF are numbered according to the criteria shown in Table 1. First the DOF of Body 1, then the DOF of Body 2. Because both bodies are articulated, one can express (for instance) the vertical displacement of Body 2 in terms of the other DOFs of the bodies. Then the vertical displacement of all axles or suspension groups. Then the rotation of any axle groups. The figure shows also the numbering of the dependent DOFs, indicated with grey dashed lines. They are included for completeness and because they might be mentioned and used within the code. These dependent DOFs appear due to the presence of the articulation. The vertical displacement of Body 2 can be expressed in terms of the others. The vertical displacement of the articulation point (also called fifth wheel) is not strictly speaking a DOF but is an auxiliary reference point, which is convenient to simplify calculations.

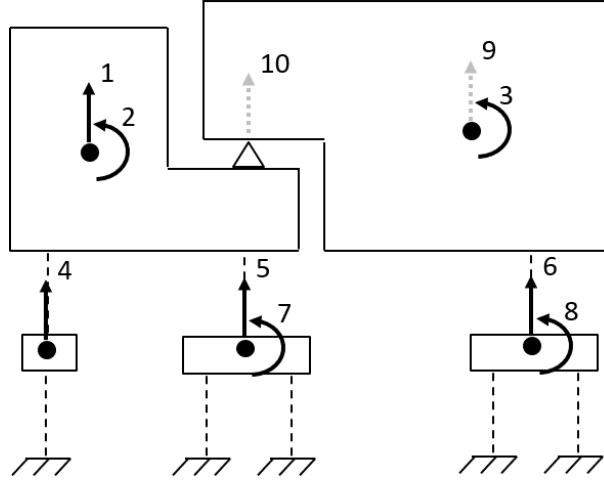


Figure 5: DOF numbering for model *Vehicle_3A2_G_1_2_2*

Note that the equations of motion have been developed using the sign criteria as shown in Figure 5, where upward displacements and anti-clockwise rotation are positive. This sign criterion is consistent with a vehicle moving from right to left. However, when using the same model to represent a vehicle moving from left to right, then there is change in the sign of the rotations. This means, that the DOF rotations obtained, when simulating a vehicle moving from left to right, have a positive value for clockwise rotations. To define the geometry and properties of a vehicle use the definitions for the “right to left” vehicle. To specify the travelling direction of the vehicle, define a negative velocity for a vehicle moving from right to left, and a positive velocity for left to right.

The user needs to define the mechanical properties listed in Table 2 for each vehicle:

Table 2: Mechanical properties of vehicle models

Variable	Description
m_{Bi}	Mass of Body i
I_{Bi}	Moment of inertia of Body i
m_{Gj}	Mass of Group/Suspension j
I_{Gj}	Moment of inertia of Group/Suspension j
k_{Sj}	Stiffness of Suspension j
c_{Sj}	Viscous damping of Suspension j
k_{Tk}	Stiffness of Tyre k
c_{Tk}	Viscous damping of Tyre k

Figure 6 shows the mechanical properties for the model *Vehicle_3A2_G_1_2_2*.

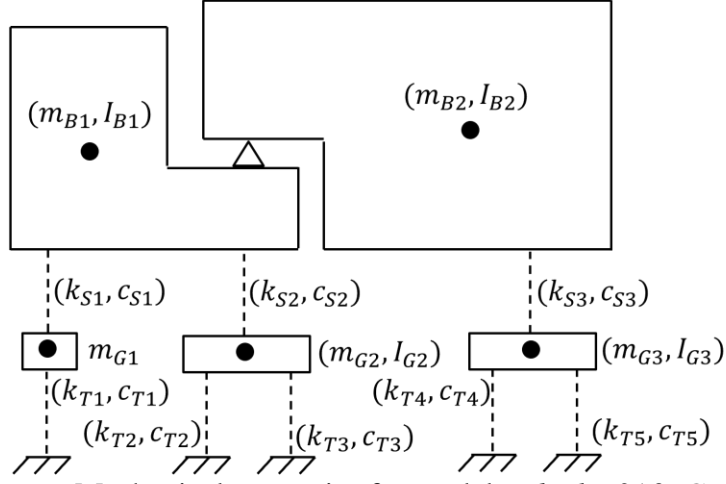


Figure 6: Mechanical properties for model *Vehicle_3A2_G_1_2_2*

The user needs to define the geometric properties listed in Table 3 for each vehicle. However, strictly speaking the variable a_1 for Body 1, and b_n for the last Body are not necessary.

Table 3: Geometric properties of vehicle models

Variable	Description
a_i	Front distance of Body i to end or articulation
b_i	Back distance of Body i to end or articulation
d_j	Horizontal coordinate of Suspension j with respects to centre of gravity of its corresponding Body. (Note that it can have negative values)
e_k	Horizontal distance of Tyre k with respect to the centre of gravity of its corresponding Axle Group centre. (Note that it can have negative values)

Figure 7 shows the geometric properties for the model *Vehicle_3A2_G_1_2_2*.

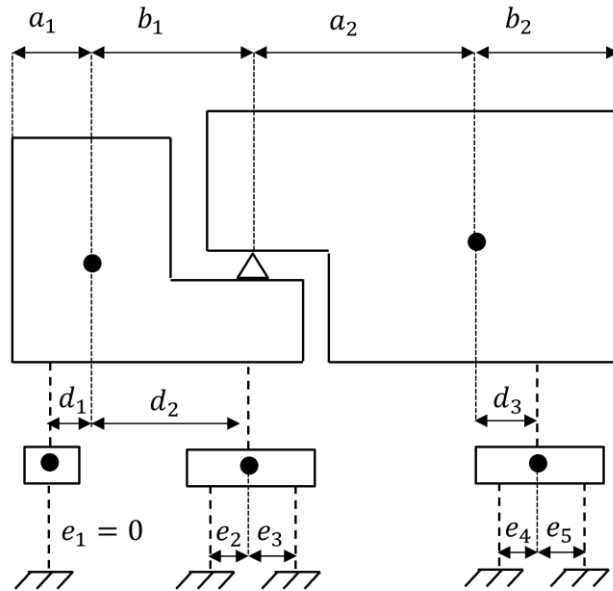


Figure 7: Geometric properties for model *Vehicle_3A2_G_1_2_2*

The folder *Vehicle_equations* contains a sample of different vehicle models. It is possible to generate similar scripts for other vehicle models using use VEqMon2D. Refer to [1] for extended explanations on how to use VEqMon2D to generate additional models.

2.1.1. Multiple vehicles

It is possible to model multiple vehicle events. These vehicles can travel in the same or opposite direction. In order to define the desired event configuration each vehicle has to be defined separately. Include the properties of vehicle i , in the structure element $Veh(i)$. Key parameters to define the event are the vehicle's initial position and velocity.

2.1.2. Vehicles position

The position in time of each vehicle can be defined in 3 different ways. The first option is to consider that the vehicles will move with constant speed. However, it is also possible to define variable velocities at time for each vehicle. In other words, the vehicles can be modelled traversing the bridge while accelerating or decelerating. VBI-2D offers 3 options to define the velocity of each vehicle:

Option 1 – Constant speed

The initial position of vehicle i is $Veh(i).Pos.x0$ and indicates the location of the front axle of the vehicle with respect to the left bridge support (see Figure 2). A vehicle starting at some approach distance from the bridge (and moving left to right) will have a negative $x0$ value, as shown in Figure 8(a).

The vehicle velocity is specified in $Veh(i).Pos.vel$. The sign of the velocity indicates if the vehicle is moving from left to right (positive velocity, as in Figure 8(a)) or from right to left (negative velocity, as in Figure 8(b)).

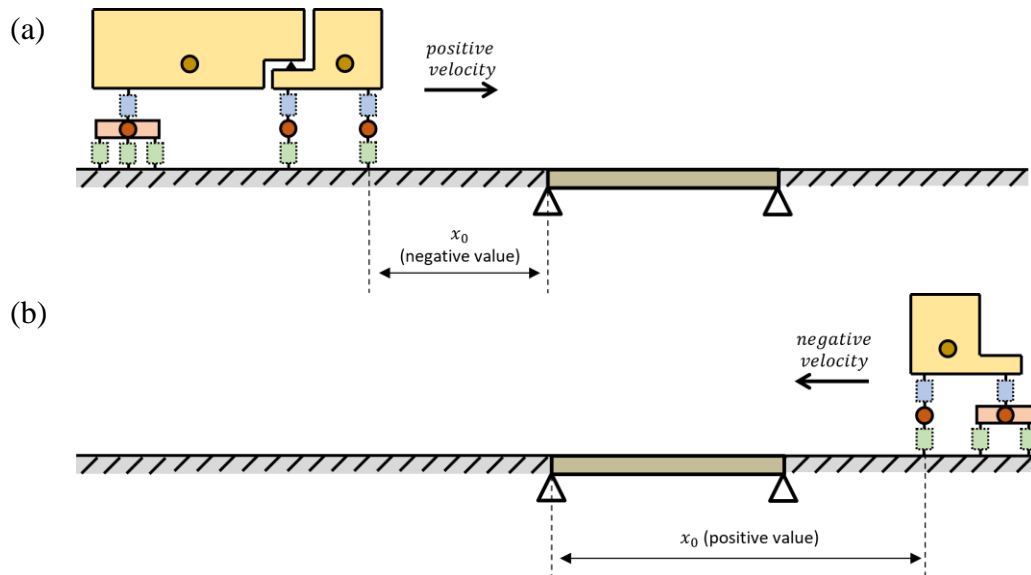


Figure 8: Definition of vehicle initial position and moving direction; (a) Left to right vehicle movement; (b) Right to left vehicle movement

Option 2 – Custom speed intervals

The velocities are defined in terms of spatial intervals. $Veh(i).Pos.x0_values$ lists the locations along the coordinate system where the different velocities are specified ($Veh(i).Pos.vel_values$).

The first axle of vehicle i starts moving from $Veh(i).Pos.x0_values(1)$ with initial velocity $Veh(i).Pos.vel_values(1)$. For the remaining time steps, the model adopts a

The user can specify as many additional intervals to define a custom vehicle velocity variation. The only condition is that the vehicle cannot change moving direction. This means that for a given vehicle all $Veh(i).Pos.vel_values$ should be of the same sign. When a vehicle moves outside the provided $Veh(i).Pos.x0_values$ the model assumes that the velocity is constant and equal to the end velocity of the last interval. It is recommended to define constant velocity when the vehicle has fully crossed the bridge.

Example:

Single vehicle crossing a 25 m bridge

Vehicle position defined by:

$Veh(1).Pos.x0_values = [-10, 0, Beam.Prop.Lb, Beam.Prop.Lb+10];$

$Veh(1).Pos.vel_values = [20, 20, 30, 30];$

Graphical representation of input and vehicle position in Figure 9.

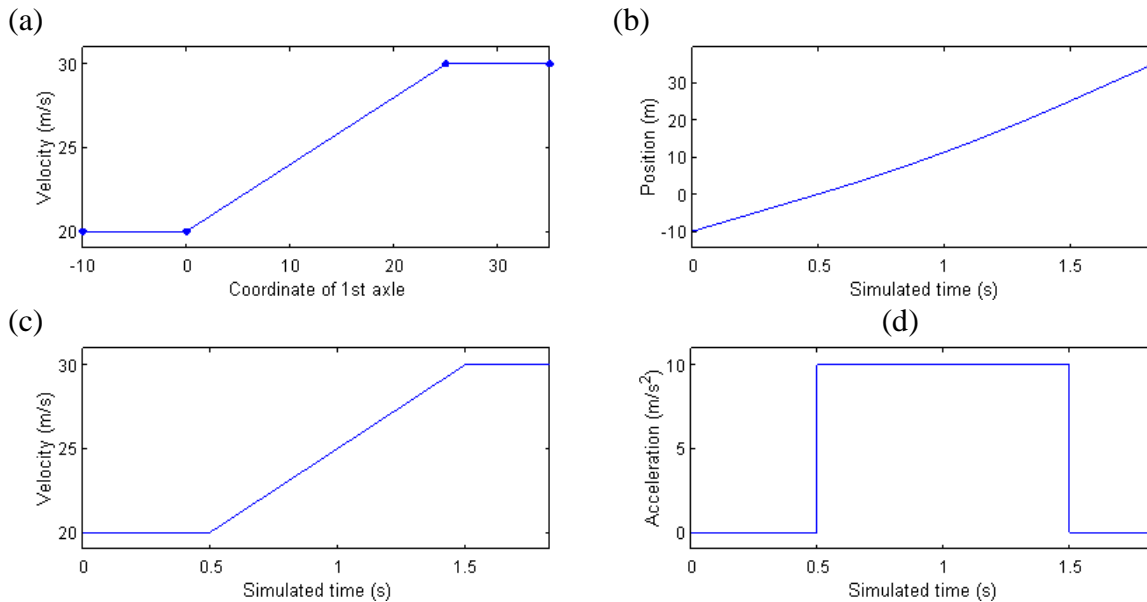


Figure 9: (a) Representation of inputs x_0_values and vel_values ;
 (b) Position of 1st axle in time;
 (c) Velocity of vehicle in time;
 (d) Acceleration of vehicle in time

Option 3 – Uniform acceleration

With this option the user defines the variation in velocity of the vehicle i directly by specifying the change in velocity (acceleration), with the variable $Veh(i).Pos.a$. Note that the accelerated motion of the vehicle is considered only when it is on the bridge. While the vehicle is approaching the bridge or it has left the bridge, the velocity of the vehicle is set constant (no acceleration).

Example

Single vehicle crossing a 25 m bridge

Moving from left to right (positive velocity)

With decelerated motion (negative acceleration)

Vehicle position defined by:

$Veh(veh_num).Pos.x0 = -10;$

$Veh(veh_num).Pos.vel = 20;$

$Veh(veh_num).Pos.a = -5;$

Addition of free vibration to the simulation

$Calc.Opt.free_vib_seconds = 0.5;$

Graphical representation of input and vehicle position in Figure 10.

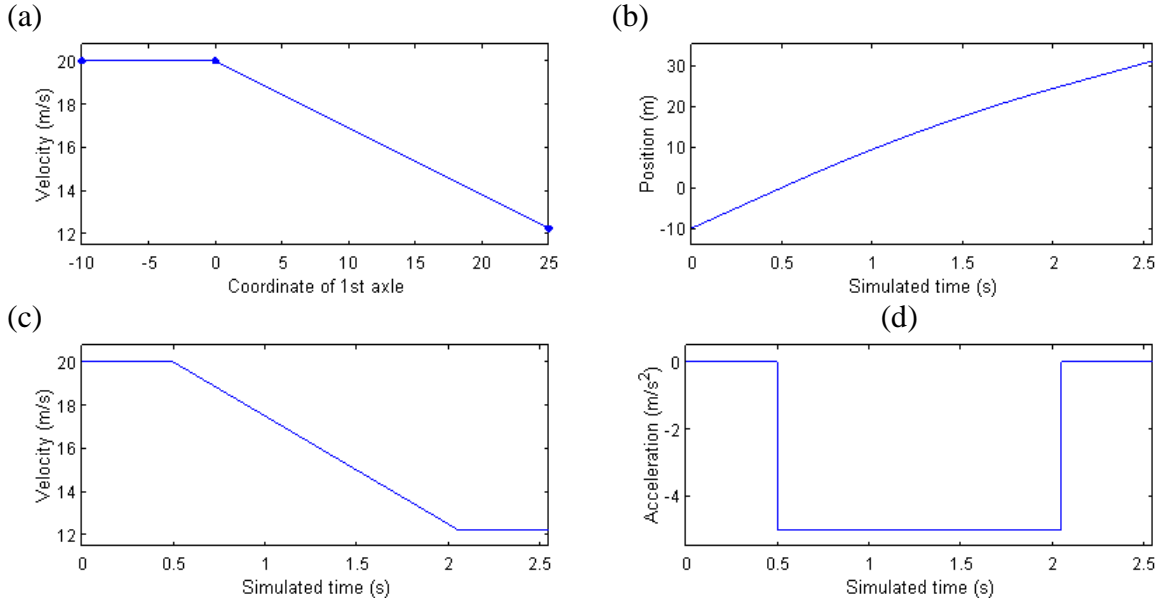


Figure 10: (a) Adopted x_0 values and vel values based on the inputs for this example

(b) Position of 1st axle in time;

(c) Velocity of vehicle in time;

(d) Acceleration of vehicle in time

2.2. Road profile

The irregularities of the road profile can also be specified in VBI-2D. The reference system for the profile is also at the left support of the beam (see Figure 2). This means that the x -coordinate is equal to zero at the left end of the bridge.

2.2.1. Road profile types

There are several possibilities to define the road irregularities, which have been classified in the following types:

- *Type 0 = Smooth profile (perfectly flat)*

This is the default profile used in any simulation. In this case no irregularities are considered, and the road profile is perfectly smooth. The vehicle(s) only excitation comes from the bridge deformation due to the vehicle passage.

- *Type 1 = Load existing profile*

It is possible to load a previously saved profile. Define the type of profile as:

$Calc.Profile.type = 1$

and provide the name of the file as:

Calc.Profile.Load.file_name = 'Test_Profile'

The file to be loaded should be available in the subfolder *Profiles*.

- *Type 2 = Randomly generated profile based on ISO standard*

It is also possible to generate a random profile based on the specifications by the ISO-8608 standard [2]. This procedure is based on the Power Spectral Density (PSD) specified in the standard. The magnitude of the PSD values define the roughness of the profile, which has been characterized in several classes, denoted with letters A to H (See Figure 11(a)). A profile is then generated based on the PSD of the selected road class and random sample of phase angles. The result is a randomly generated profile for the specified class. Figure 11(b) shows an example of a Class A randomly sampled 1000 m long profile.

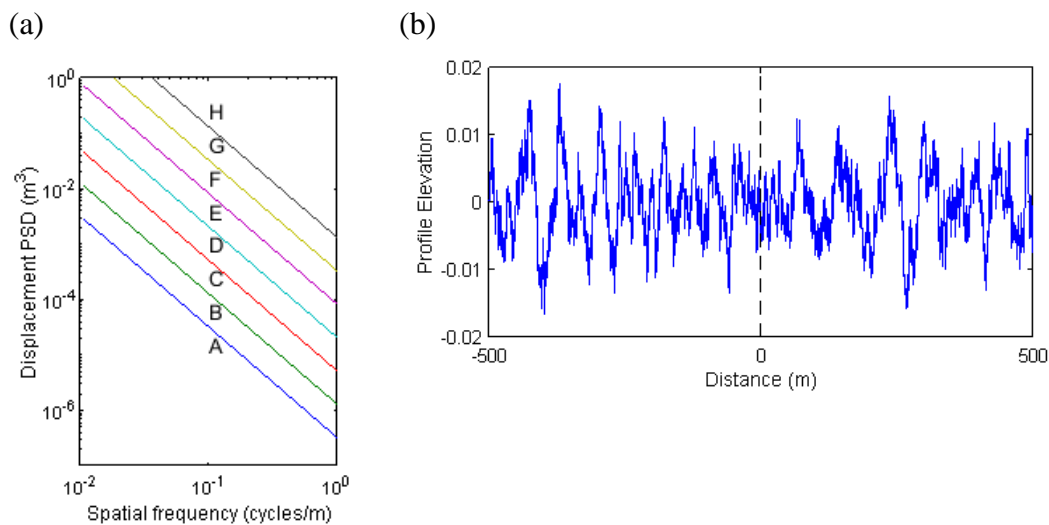


Figure 11: (a) Class definitions according to ISO-8608; (b) Example of a Class A profile

The user has one additional option to control the roughness level of the generated profile. The PSD lines defined by the standard, specify the limit between two road classes. Therefore, a profile of class B should be based on a PSD laying between the upper limit for Class A and the upper limit for class B. By default, VBI-2D defines the reference PSD as the one in the middle between both limits. This is specified with the parameter:

Calc.Profile.Opt.class_var = 0

However, it is also possible to define randomly the reference magnitude of the PSD. In that case, VBI-2D defines the PSD of the specified class using a reference magnitude random sampled within the valid limits of the selected road class. To activate this option set the following parameter to

Calc.Profile.Opt.class_var = 1

- *Type 3 = Step profile*

With this type of profile, the user defines a road irregularity in the form of a sharp step of desired elevation and location. For example for the parameter definition below the corresponding profile is shown in Figure 12.

Example for step profile
 $\text{Calc.Profile.type} = 3$
 $\text{Calc.Profile.step_loc_x} = 5$
 $\text{Calc.Profile.step_height} = 0.05$
 $\text{Calc.Profile.L} = 100$

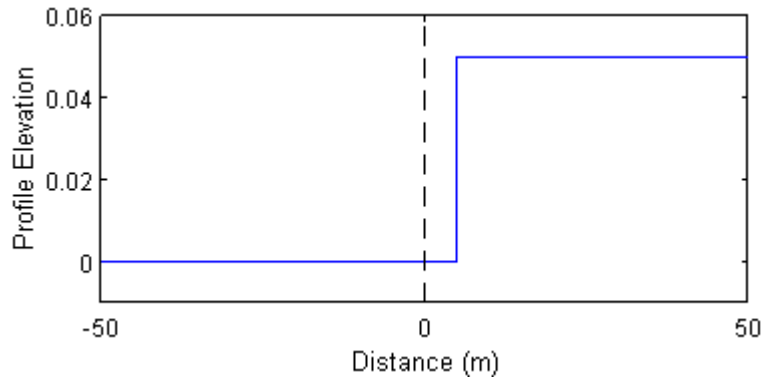


Figure 12: Step profile example

- *Type 4 = Ramp profile*

Similarly, the user can define a ramp profile. The length, height and location of the ramp must be defined with the user. Figure 13 shows the resulting profile for the example below.

Example for ramp profile
 $\text{Calc.Profile.type} = 4$
 $\text{Calc.Profile.ramp_loc_x_0} = -25$
 $\text{Calc.Profile.ramp_loc_x_end} = 25$
 $\text{Calc.Profile.ramp_height} = 0.01$
 $\text{Calc.Profile.L} = 100$

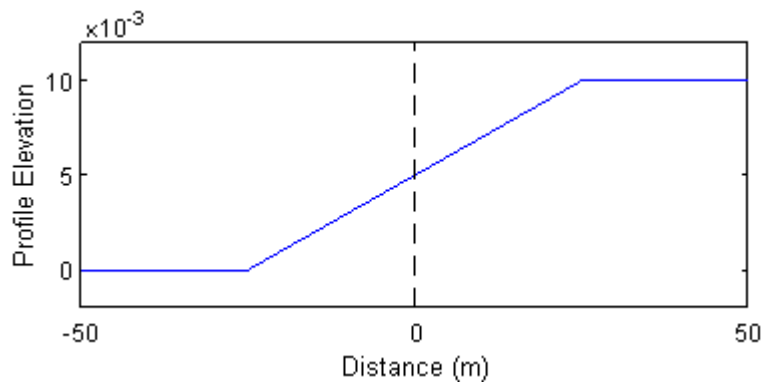


Figure 13: Ramp profile example

- *Type 5 = Sinewave*

Another predefined profile geometry consists of imposing a sinusoidal shape for the road irregularity. The user has the option to define the wavelength and phase angle (at coordinate $x = 0$) and amplitude of the sinewave. Figure 14 shows the resulting profile for the example below.

Example for sinewave profile

Calc.Profile.type = 5

Calc.Profile.sine_wavelength = 20

Calc.Profile.sin_Amp = 0.01

Calc.Profile.sin_phase = 0

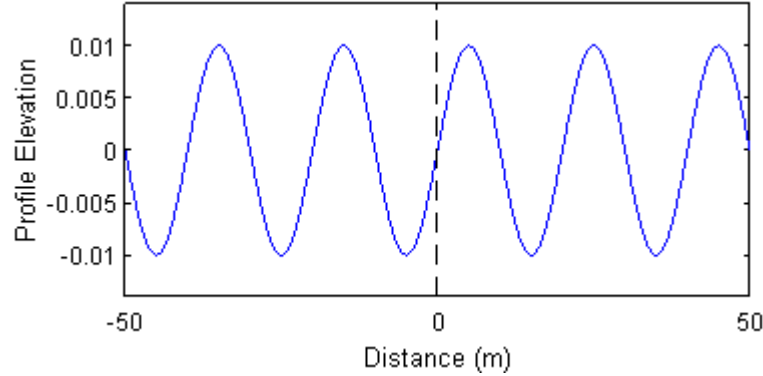


Figure 14: Sinewave profile example

2.2.2. Saving a road profile

The user can choose to save the road profile generated during a simulation. By default this operation is not performed. The user can switch this option on and define the name of the file where all relevant information about the profile should be saved. The file is saved in the subfolder *Profiles*. The variables to define to save a profile are:

Calc.Profile.Save.on = 1

Calc.Profile.Save.file_name = 'Test_Profile'

Saving a profile is particularly useful when studying the VBI problem for one particular ISO profile (*Type=2*). To repeat the simulation but for different vehicle velocity or a different vehicle model (for example), the user can recover the generated profile if it has been previously saved. To recover a previously saved profile use *Type=1*, as discussed in Section 2.2.1.

2.2.3. Moving average filter

The generated profiles can be processed by a moving average filter. With this operation it is possible to approximate the effect of an actual tyre contact patch, as suggested in [3]. To activate this option the user needs to define the logical flag for this option and specify the length of the tyre patch to consider. For the example below, Figure 15 shows a comparison of the original and filtered profiles. In this case a 24 cm long patch is assumed. After the profile is generated and processed, the simulation proceeds using the filtered profile.

Example for moving average option

Calc.Profile.Opt.MovAvg.on = 1

Calc.Profile.Opt.MovAvg.window_L = 0.24

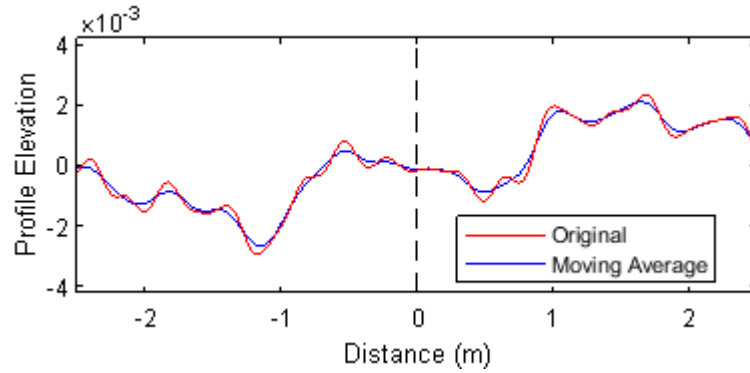


Figure 15: Sinewave profile example

2.2.4. Other comments

There is a distinction between generated and needed profile. The software generates a profile that is *Calc.Profile.L* long. Then, depending on the number of vehicles, initial positions and velocities, only parts of that profile are needed. The recommendation is to make sure that the total distance is sufficiently long to allow future uses of the same profile with different vehicles involved in the event.

2.3. Bridge

The bridge is modelled as an Euler-Bernoulli beam using a Finite Element Model (FEM) discretization. Each beam element has two nodes, and each node has two DOF, namely vertical displacement and rotation. Figure 16 shows a sketch of the model together with the associated variables.

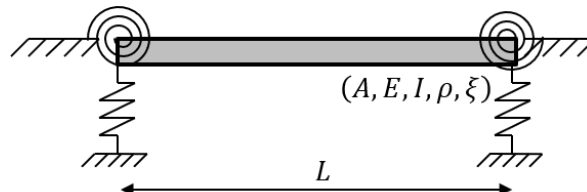


Figure 16: Sketch of bridge model

The list of parameters that define the bridge are given in Table 4.

Table 4: List of beam variables

Variable	Description
L	Bridge span
A	Cross-section area
I	Section's second moment of area
ρ	Mass per unit length
ξ	Damping ratio

2.3.1. Mechanical properties

The mechanical properties of the beam whole beam can be defined as variables part of the structure element *Beam.Prop* (for beam properties). The possible fields are: *Lb* (beam's length),

ρ (density per unit length), E (elastic modulus), I (second moment of area), A (cross-sectional area), $damp_per$ (damping ratio in percentage).

Alternatively, the user can define the beam properties from a table of properties that could be regarded as typical concrete bridge properties. These properties are obtained from [4] and are classified as indicated in Figure 17. In this case the user should define the total span of the bridge (*Bridge.Prop.Lb*) and the type of bridge (*Bridge.Prop.type*). In addition the user needs to specify the desired damping ratio (*Bridge.Prop.damp_per*). The possibilities and valid range of spans are:

T beam bridge (*type* = 'T') for spans 9 m to 21 m
Y beam bridge (*type* = 'Y') for spans 17 m to 31 m
Super-Y beam bridge (*type* = 'SY') for spans 33 m to 43 m

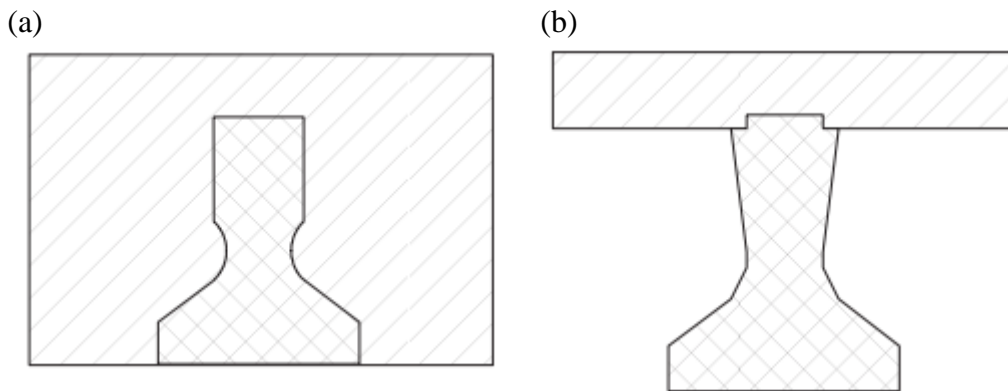


Figure 17: (a) T beam bridge; (b) Y and Super-Y beam bridge (Source: [4])

2.3.2. Mesh

The mesh of the beam is made of beam elements of equal length. The user simply defines the number of elements with the variable *Beam.Mesh.Ele.num*. Therefore, the adopted element size is the total beam length over the number of elements (*Beam.Prop.Lb/Beam.Mesh.Ele.num*). The number of elements should be (recommendation) an even number to ensure that the mesh has a node at mid-span. The node at mid-span might not be the most relevant node in all structural configurations. But the mid-span is generally the section of greatest interest for the case of single span bridge. VBI-2D calculates some additional results for that point by default. However, similar analysis could be done for any section along the beam since the output of the simulation contains the response of all the DOFs of the model.

2.3.3. Boundary conditions

With VBI-2D it is possible to define several boundary conditions of different types. This flexibility gives the user the possibility to model a wide range of structural configurations. It is necessary to define 3 arrays to indicate the location, vertical stiffness, and rotational stiffness of each boundary condition. Each entry in an array gives the properties of one particular boundary condition. The location is defined in terms of the x-coordinate where the boundary condition should be placed. The vertical stiffness (Figure 18(a)) is the value of the linear spring that opposes the vertical motion of the DOF of the beam at that location. The rotational spring (Figure 18(b)) is the value of the rotational spring that opposes the rotations of the DOF at that location. Possible values of these stiffnesses are either positive integers or the value *Inf*. When

a stiffness value is equal to *Inf*, then no vertical deformation or rotation will occur at the corresponding DOF.

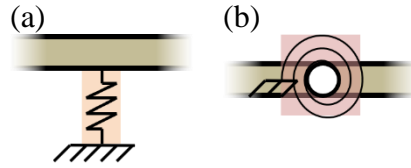


Figure 18: (a) Vertical stiffness; (b) Rotational stiffness

As an example, a multi-span continuous bridge with different types of boundary conditions can be easily defined. The schematic representation of the boundary conditions for this example is shown in Figure 19.

Example of multi-span beam

Beam.BC.loc = [0, *Beam.Prop.Lb*/2, *Beam.Prop.Lb*]

Beam.BC.vert_stiff = [*Inf*, 1e6, 0]

Beam.BC.rot_stiff = [0, 1e4, *Inf*]

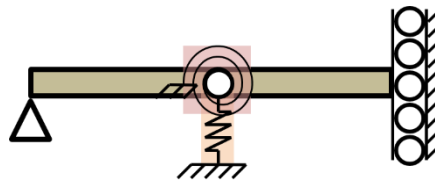


Figure 19: Example of boundary conditions definition

Therefore, it is possible to define a bridge with a wide range of boundary conditions. The location of the boundary conditions can be specified, which allows the definition of multiple span configuration. The vertical and rotational stiffness can vary from zero to infinity, which permits having boundary conditions with certain stiffness or fully fixed.

2.3.4. Damage

It is also possible to define variations of the mechanical properties along the bridge mesh. This has been termed damages because it is a good approximation of possible damages that a structure might have. All options and information regarding damages is store in *Beam.Damage*. Different types of damages are implemented in VBI-2D, and can be selected by setting the value of *Beam.Damage.type*. Each type is discussed bellow separately.

Damage on the bridge can be modelled specifying the location and type of damage. The damage is represented as a reduction of one of the beam properties at one particular element.

- No damage (for *Beam.Damage.type* = 0)

This is the default option. Basically, this means that all elements of the beam mesh have the same mechanical properties.

- Single element damage (for *Beam.Damage.type* = 1)

With this damage type the user can define different mechanical properties for one single element of the beam mesh. More precisely, it is defined as a reduction of one of the mechanical properties, compared to the mechanical properties of the rest of the elements. For this the user needs to define the location of the damage in percentage of total beam length, and the percentage of reduction in the corresponding property. All mechanical properties of the beam can be reduced, this is: E , I , ρ , and A . Figure 20 shows a schematic representation of what a single element damage could be represented.

Example: Beam damage at $\frac{1}{4}$ of the span with a stiffness reduction of 10%
 $Beam.Damage.loc_per = 25$
 $Beam.Damage.E.per = 10$

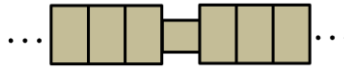


Figure 20: Schematic representation of a single element damage

- Global damage (for $Beam.Damage.type = 2$)

With this damage type the user can reduce the magnitude of one of the mechanical properties in the mesh. This reduction is then applied to all the elements of the beam.

Example: Global change of by reducing E by 20% along the whole beam
 $Beam.Damage.E.per = 20$

- Change in boundary conditions (for $Beam.Damage.type = 3$)

It is possible to increase/reduce the stiffness of one or more DOFs with boundary conditions. This change in stiffness can be done to the vertical and rotational stiffness separately. The change is defined by specifying the location and magnitude of stiffness change, stored in the variable $Beam.Damage.BC$.

Example: Increase of rotational stiffness at the right support by 10^6 N/rad
 $Beam.Damage.BC.loc = Beam.Prop.Lb$
 $Beam.Damage.BC.rot_stiff = 1e6$

- Custom function (for $Beam.Damage.type = 4$)

For more flexibility in the definition of mechanical properties, VBI-2D has type = 4 damage. With this damage type the user can define factors to multiply one mechanical property of the beam. These factors can be defined via an anonymous function stored in the $Beam.Damage$ variable. With this tool it is possible to represent more complex structural configurations. This damage type also offers the possibility of plotting the factor applied to each of the elements, as show in Figure 21 for the example below.

Example: Parabolic change of E along the beam length
 $Beam.Damage.Plot.on = 1$
 $Beam.Damage.E.Inputs.values = [0.4, Beam.Prop.Lb]$
 $Beam.Damage.E.fun = @(x,Inputs)...$
 $-(1-Inputs.values(1))*4/Inputs.values(2)^2*x.^2 + ...$
 $+(1-Inputs.values(1))*4/Inputs.values(2)*x + Inputs.values(1)$

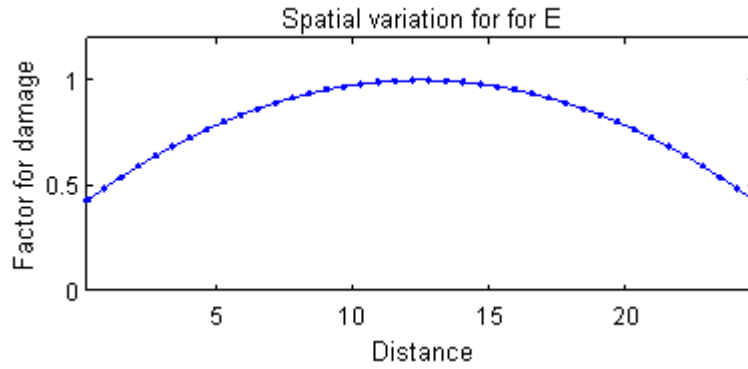


Figure 21: Function defined factors to multiply each beam element

2.4. Numerical solution

The simulation of the vehicle-bridge interaction considering multiple vehicle vehicles travelling over a road profile and traversing the bridge is solved numerically. For that, first the separate systems (vehicles and bridge) have to be coupled together, which is achieved via different by equivalent interaction procedures. Then the solution is found by numerical integration of the problem. Each of these aspects are describe with greater detail below.

2.4.1. Interaction procedure

The models presented above for the vehicle(s) and beam are defined as separate models. However, during a crossing event, the models interact, and the systems are then coupled together. To solve the coupled system, the equations of motion need to be coupled or solved by iterations. The software has three alternative procedures to impose the interaction between vehicle(s) and bridge, which are: *Full_Sim_Iter*, *StepByStep_Iter*, and *Coupled*. The user selects which procedure to use via the variable *Calc.Proc.name*. All the implemented procedures are equivalent and provide the same answer. However, note that VBI-2D has been optimized to solve the interaction via the *Coupled* procedure, which is also set as the default. Therefore, it is recommended use the default procedure due to its computational efficiency.

- Full simulation iteration (FI)

In this approach, the vehicle and bridge are solved iteratively. First, the vehicle is solved for a given profile. The contact forces are applied to the bridge. The road profile is updated with the beam deformation. The vehicle is simulated again running over the updated profile. This is repeated until certain convergence criterium is reached. The conceptual representation of this solution procedure is presented in Figure 22. To solve the interaction via this approach the user needs to define *Calc.Proc.name* = '*Full_Sim_Iter*'.

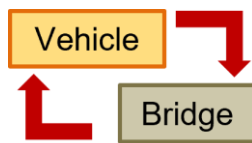


Figure 22: Conceptual representation of full simulation iteration (FI)

The convergence criteria can be defined in two different ways. The convergence is achieved when the difference of certain result for two consecutive iterations is below certain value (*Calc.Proc.Iter.tol*). Select *Calc.Proc.Iter.criteria = 1* to define the convergence criteria based on the beam deformation under the vehicle wheels. Select *Calc.Proc.Iter.criteria = 2* to define the convergence criteria based on the bending moment of the beam. The latter is set as the default criteria, since it is a more demanding criteria that leads to more iterations and more accurate results.

- Step-by-step interaction (SSI)

This procedure is similar to the previous one, but the convergence is checked for every time step during the numerical integration. This is, for every time step the vehicle and beam are solved iteratively. The vehicle(s) is solved for the road profile at that time step. The contact force is then applied to the beam. From the beam response, the beam deformation under the wheels are obtained, which are then used to update the profile. This is repeated iteratively until the convergence criteria is fulfilled. The same convergence criteria and options are available to the user as those described for FI procedure. To activate the step-by-step interaction procedure, the user needs to define *Calc.Proc.short_name = 'SSI'*. The conceptual representation of this solution procedure is presented in Figure 23.

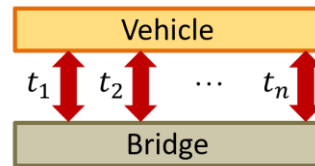


Figure 23: Conceptual representation of step-by-step interaction (SSI)

- Coupled (Coup)

Alternatively, the vehicle-bridge interaction can be solved without any iterations. For this solution procedure the equations of motion, expressed in terms of the system matrices, are defined including the coupling relations between the vehicle(s) and the bridge. Because the vehicle(s) position changes in time, the global system matrices of the coupled system also change in time. Therefore, the system matrices of the coupled vehicle and bridge system are updated for every new position of the vehicle. The equations of motion are then solved numerically. The conceptual representation of this solution procedure is presented in Figure 24. To solve the interaction via this approach the user needs to define *Calc.Proc.name = 'Coup'*.

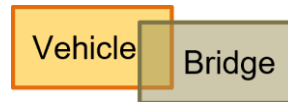


Figure 24: Conceptual representation of couple procedure (Coup)

2.4.2. Numerical integration

Regardless of the adopted interaction procedure (see Section 2.4.1), the equations of motion of the vehicle, the bridge or the coupled system are solved by numerical integration. The system of second order differential equations is solved by direct numerical integration with the Newmark- β algorithm. The adopted formulation for the algorithm is taken from [5].

3. Running VBI-2D

In order to run VBI-2D for the first time, the user can use the script *A01_VBI_input_and_run.m*, which already includes the definition of all the inputs and options. This main script calls the rest of the necessary functions and script listed in the folder and subfolders.

3.1. Model definition and configuration of simulation

The script *A01_VBI_input_and_run.m* has two distinct parts, first the definitions and then the calculations. The user can simulate a wide range of different traffic events, road conditions and structural configurations, with an extensive set of calculation options and predefined graphical outputs. The definitions of the event to simulate are sorted in descending order, starting from the vehicle(s), adding the desired road profile and continued with the definition of the structural system. In addition, the user can define options to specify certain aspects of the simulation and graphical output. The script has been edited with sufficient comments to clarify the purpose of each definition.

Below is a discussion of the main features and definitions done in *A01_VBI_input_and_run.m*. The following list is sorted in the same order as the different definitions are set in the script *A01_VBI_input_and_run.m*.

- Vehicle(s)
 - It is possible to define as many vehicles as desired. Simply define the properties of each vehicle in a different element of the *Veh* structure. For example, the properties of the first vehicle are in *Veh(1)*, for vehicle 2 in *Veh(2)*, and so on.
 - *Veh(i).Model.type* is a string with the name of the vehicle model to use. All vehicle models are saved in the subfolder *Vehicle_equations*. To add more vehicle models to this folder review See Section 2.1 and use the tool VEqMon2D [1].
 - In *Veh(i).Model.Prop* the user needs to define the numerical values of all the mechanical and geometrical properties of the selected vehicle model. See inside the function of the selected vehicle model for the full list and description of the required properties.
 - The position in time of each vehicle is defined in terms of its initial position of the vehicle's front wheel (*Veh(i).Pos.x0*) with respect to the global reference system of the model, its velocity (*Veh(i).Pos.vel*). Note that, positive velocity indicates that the vehicle moves from left to right, while negative velocity corresponds to a right to left moving vehicle.
 - It is also possible to describe vehicle motion with varying velocity in time. One possibility is to define an array of locations along the vehicle path (*Veh(i).Pos.x0_values*) and the desired velocity when the front axle of the vehicle passes that point (*Veh(i).Pos.vel_values*). If subsequent values in the *vel_values* array are different, the vehicle will be modelled with accelerated motion. The other possibility is to define a constant value of acceleration with *Veh(i).Pos.a*. Then the vehicle will move from *x0* position with velocity *vel* while it approaches the bridge. As soon as the vehicle enters the bridge, the vehicle will start moving with the specified acceleration. After the last axle has left the bridge, the acceleration will be reset to zero, and the vehicle will continue moving with the constant velocity reached at that moment. Note that the sign criteria adopted here is consistent with the global reference system of the model. For vehicles from left to right, a positive acceleration indicates that the vehicle increases velocity over time, while a negative acceleration corresponds to a decelerated motion. The signs in acceleration indicate the opposite for vehicles moving in

the opposite direction. For a vehicle moving from right to left, a negative acceleration corresponds to an accelerated motion.

- For additional comments on the details of vehicle modelling refer to Section 2.1.
- Road profile (Irregularities)
- There is the possibility to generate different types of profiles. The user needs to specify the variable *Calc.Profile.type* according to one of the following:
 - Type 0 = Perfectly smooth profile
 - Type 1 = Load existing profile
 - Type 2 = Generate random profile according to ISO-8608 standard [2]
 - Type 3 = A profile with a step
 - Type 4 = A profile with a ramp
 - Type 5 = A single sinewave profile
- The user has the option to save the generated profile with the option *Calc.Profile.Save.on = 1*. The profile will be saved in the subfolder *Profiles*. This profile can then be used in another simulation by loading it using *Calc.Profile.type = 1*.
- Noteworthy is the profile type 2. This will generate a random instance of a profile based on the specifications found in the ISO-8608 standard [2]
- With the options in *Calc.Profile.Opt.MovAvg*, the user can specify the properties of a moving average to be applied to the generated profile. The intention of this filter is to try to mimic the effect of the tyre contact patch.
- The variable *Calc.Profile.L* defines the total length of the profile to be generated. It is recommended to generate a much longer profile than necessary. During the simulation, VBI-2D will only use the parts relevant for each vehicle.
- For additional comments on the details of road profile modelling refer to Section 2.2.
- Bridge
- All properties related to the bridge definition are stored in the structure *Beam*.
- The user can define all the properties of the beam either manually or based on predefined values for a given span and beam type. Set *Beam.Prop.type* to “T”, “Y” or “SY” and specify the desired span *Beam.Prop.span*. See the function *B51_BeamProperties.m* for more information.
- The boundary conditions are defined at the closest nodes specified by coordinates listed in *Beam.BC.loc*. For each node, the user needs to define what is the vertical stiffness and rotational stiffness, with the variables *Beam.BC.vert_stiff* and *Beam.BC.rot_stiff*. To define infinite stiffness values, select the value *Inf*.
- Define the number of elements that the beam mesh should have with the variable *Beam.Mesh.Ele.num*. An even number of elements leads to a mesh with a node at mid-span. By default, the software calculates the load effects at mid-span, so it needs a mesh with a node at that location.
- The user can also induce some damage on the bridge. The user can modify some of the properties of the bridge model, which can be regarded as damage modelling. Use *Beam.Damage.type = 1* to damage one single element of the mesh, *Beam.Damage.type = 2* to change one property of all elements, and *Beam.Damage.type = 3* to increase/reduce the stiffness on the boundary conditions. In addition, there is the option *Beam.Damage.type = 4* that allows the user to define a function to specify multiplicative factors to the stiffness of each of the elements in the beam. This option provides great flexibility to the user. For

example, with this option it is possible to model either complex damage layouts, randomness in the bridge properties or a beam with varying cross-sections.

- For additional comments on the details of bridge, boundary conditions and damage modelling refer to Section 2.3.
- Calculation options
 - *Calc.Solver.max_accurate_freq* = Sets the size of the time step depending on how high the accuracy should be. The larger the desired maximum accurate frequency the smaller the time step is. This variable is used to specify the size of the time step to be used in the numerical solver. It is expressed in terms of Hz. It specifies up to what frequency the solver should provide accurate results. The resulting time step is equal to the inverse of twice the selected frequency.
 - *Calc.Solver.min_t_steps_per_second* = Specify the minimum number of time steps per second needed to be simulated.
 - *Calc.Solver.min_Beam_modes_considered* = Number of beam modes to be considered for time step selection.
 - Based on the defined time steps requirements by the user, the software finds the most demanding condition and takes this as the criteria to define the time step to be used in the numerical analysis. In other words, the time step will be the smallest one according to all possible conditions.
 - *Calc.Solver.t_steps_per_second* = Alternatively, the user can fix the desired time step to use for the numerical analysis
 - *Calc.Proc.name* = Specify the name of the procedure to use to solve the coupling between systems. Options are: 'Full_Sim_Iter', 'StepByStep_Iter', 'Coupled'. Note that the default and recommended procedure is 'Coupled'. Greater efforts were done in the optimization of the computational efficiency of this procedure. However, the other procedures are valid and give the same result (disregarding small numerical differences).
 - For procedures requiring iterations, the user has additional options. *Calc.Proc.Iter.max_num* defines the maximum number of iterations to perform. Sets an upper limit to the number of possible iterations. *Calc.Proc.Iter.criteria* defines what criteria to use for the convergence check. The convergence can be in terms of beam displacements (*criteria* = 1) or beam bending moments (*criteria* = 2). *Calc.Proc.Iter.tol* defines the numerical value of the convergence criteria tolerance. When the difference between two consecutive iterations is below the specified value, the software considers that convergence has been reached and the iterative process stops.
 - *Calc.Opt.VBI* = Option to connect or disconnect the Vehicle-Bridge Interaction (VBI). If 0 then no interaction is considered, and the solution is the moving load problem. If equal to 1, then vehicle and bridge interact, and the model solves the coupled problem.
 - *Calc.Opt.free_vib_seconds* = Indicates how many extra seconds should the simulation be calculated after the last vehicle has left the bridge.
 - *Calc.Opt.verbose* = Option to display on the command window a detailed report of the simulation to be run.
 - *Calc.Opt.show_progress_every_s* = Number of seconds that should pass between consecutive progress displays on the command window.
 - For additional comments on the calculation options see Section 2.4. Also refer to the variables glossary in *A00_Variables_glossary.m* (Section 5) for a comprehensive list and description of all the variables and additional options available in VBI-2D.

- Plotting options
- The user can request graphical outputs after the simulation has finished. VBI-2D tool has several implementations that process the results from a simulation and produce a graphical output. By default, no plotting will be generated. The user needs to switch on the option for the particular type of plot. For example, if the user sets *Calc.Plot.P27_Veh_A = 1*, then the software automatically plots the accelerations for all the DOFs of the vehicle(s). Refer to *A01_VBI_input_and_run.m* for the complete list of available graphical outputs.
- Nevertheless, the users can easily generate their own custom plots with the results from a simulation. All the information of a completed simulation is stored in the corresponding structure variables: *Veh* (with all the details about the vehicles), *Beam* (containing all variables related to the structural system), *Calc* (with every option and auxiliary variable related to the simulation, including the profile), and *Sol* (that contains all the results and outcomes from the simulation). Refer to Section 3.3 for an extended explanation of the contents in *Sol* structure.

3.2. Calculations

Once all the inputs and options are specified, the script continues with the execution the actual calculations to solve the model. All these calculations are grouped into a separate script called *A02_calcuations.m*. The user does not need to edit or understand the calculations performed in this script. Advanced user can explore and modify the calculations. The scripts and functions have been created aiming for simplicity and clarity. This is enhanced by the adoption of clear and understandable notation supported by complementary comments. The software uses over +40 different functions and scripts. This user guide does not provide an extended explanation of each. Nevertheless, the list below provides an overview of the calculations performed in *A02_calculations.m*:

- Checks the correctness of the inputs. However, note that these checks do not cover all possible errors that the user might produce during the input definition.
- Creates the system matrices of vehicle and beam, together with additional auxiliary variables needed for the calculation.
- Solves the coupled equations of motions by direct integration with the selected procedure.
- Generates a series of outputs based on the nodal displacements, including contact forces, deformations, bending moments, shear forces and accelerations.

3.3. Working with the output

All the results of the model are included in the Matlab structure *Sol*. The results are organized in various sub-structures, namely *Veh* and *Beam* substructures for the vehicle and beam results respectively.

- Vehicle *i* results (in *Sol.Veh(i)*)
- Matrices *U*, *V* and *A* are the deformations, velocities, and accelerations of each of the DOFs of vehicle *i*.
- *Sol.Veh(i).Under* is a substructure with information about the results at the contact point between wheel and road.
 - *Sol.Veh(i).Under.def* and *Sol.Veh(i).Under.vel* are the deformation and velocities of the beam respectively.

- *Sol.Veh(i).Under.h* and *Sol.Veh(i).Under.hd* are the elevation and first derivative of the profile respectively.
 - *Sol.Veh(i).Under.onBeamF* is the contact force for each wheel in time.
- *Sol.Veh(i).Wheels* is a substructure with information about the results at the wheels of the vehicle. This is the point above the contact point. The wheel point and the contact point are linked together by the spring and dashpot of the tyre. For a single axle (no group) the wheel position is the same as the suspension mass position. For an axle group, the wheel position is calculated considering the suspension vertical displacement and its rotation.
 - *Sol.Veh(i).Wheel.U* is the vertical displacement in time with respect to the global reference system. This includes the road profile.
 - *Sol.Veh(i).Wheel.Urel* is the relative vertical displacement in time. In other words, the vertical distance between the wheel and the contact point.
 - *Sol.Veh(i).Wheel.V* is the vertical velocity in time with respect to the global reference system. This includes the road profile first derivative.
 - *Sol.Veh(i).Wheel.Vrel* is the relative vertical velocity in time. In other words, the vertical velocity between the wheel and the contact point.
- Beam results (in *Sol.Beam*)
 - Sub-structures *U*, *V*, *A*, with the displacement, velocity and acceleration of all DOFs in the model.
 - Sub-structures *BM*, *Shear*, *U_static*, *BM_static* and *Shear_static* containing information about bending moment, shear, static deformation, static bending moment and static shear of each node of the beam model.
 - Sub-structures *Disp*, *Disp_static*, *Disp_dot*, and *Disp_ddot* containing the vertical displacement for each node for corresponding responses: total, static, first time derivative, second time derivative.
 - Sub-structures *Rot*, *Rot_static*, *Rot_dot*, and *Rot_ddot* containing the rotation for each node for corresponding responses: total, static, first time derivative, second time derivative.
 - Each of these results are called Load Effects (LE) and contain the following sub-structures storing the information in a systematic manner.
 - *Sol.Beam.(LE).value_DOFT* = Results for all the DOFs of the beam in time. Note that each node has 2 DOFs. The first row is for DOF1 (vertical displacement of node 1), second row is for DOF2 (rotation of node 1), third row is for DOF3 (vertical displacement of node 2), and so on ...
 - *Sol.Beam.(LE).value_xt* = Only results of vertical displacements in time.
 - *Sol.Beam.(LE).Max* and *Sol.Beam.(LE).Min* = sub-structures with information about the extreme values of the corresponding load effect. Where the subfields are:
 - value = numerical value of the extreme
 - COP = Critical Observation Point, which is the location along the beam where the extreme value occurred
 - pCOP = is COP expressed in percentage of bridge span
 - cri_t_ind = is the critical time index, or the time step in the beam simulation when the recorded extreme occurred.
 - value05 = Value of the extreme response at the mid-span section

4. Additional notes

This section contains a mix of comments regarding particularities of some functions or relevant information about the implementation.

- The vehicle system matrices include one matrix called *Veh(i).SysM.N2w* (Nodal to wheel) and is used to obtain the displacements at the wheels from the vehicle's nodal displacements. This is particularly relevant when the vehicle has axle groups. It allows to easily transform the vertical displacement and rotation of a group into the vertical displacement of each particular wheel.
- There are three solution procedures available. All of them give the same result. However, the Coup procedure is recommended because no iteration stopping criteria needs to be defined.
- The existing literature regarding VBI, often mentions the Coriolis effect of moving masses, as for example in [6]. This is directly relevant when modelling a railway vehicle, where the wheel (rotating mass) is in direct contact with the track. On the other hand, for road vehicles the wheel mass is generally considered as a lumped mass (together with the suspension mass) and is connected to the road profile by a spring and a dashpot. The contributions of the Coriolis effects naturally appear when deriving the VBI formulation as can be seen for instance in [7]. For the case of no rotating mass there are still contributions to what is termed here Coriolis effect. These terms appear because of the application of the derivation chain rule. The bridge deformation depends on space and time. Thus, when calculating the time derivatives, the chain rule needs to be applied and additional terms appear. These terms then need to be included in the contact force calculation (for the iterative procedures) or in the system matrices (for the coupled solution). Even though, strictly speaking this is not due to a rotating mass, this contribution has been termed Coriolis effect in this software.
- This software includes the option to switch on/off the option to include the Coriolis effect. However, not including the Coriolis effect is wrong, and should always be on for correct results. The default option is to include this effect. However, this on/off possibility is included for comparison purposes because other available solutions in literature might not include this effect.

5. Glossary of variables

This user guide has covered all the basic inputs and options for anybody to be able to use VBI-2D. With this knowledge the user can simulate a huge range of traffic scenarios, road conditions and structural systems for the bridge. VBI-2D includes some additional undocumented options that have not been described here. The default values adopted by the software for these options are usually appropriate.

Nevertheless, and for completeness, the script *A00_Variables_glossary.m* lists all the variables used in the software, either as input, output, option, or auxiliary variable. Find there also a short description for each of them together with the list of possible options.

References

- [1] Cantero, D. VEqMon2D – Equations of motion generation tool of 2D vehicles with Matlab. Software X, 2022; 19: 101113. DOI: [10.1016/j.softx.2022.101103](https://doi.org/10.1016/j.softx.2022.101103)
- [2] International organization for standardization. *Mechanical vibration - Road surface profiles - Reporting of measure data*. ISO, 1995, ISO 8608.

- [3] Harris, N.K., OBrien, E.J. & González, A. 2007. Reduction of bridge dynamic amplification through adjustment of vehicle suspension damping. *Journal of Sound and Vibration* 302(3):471-485.
- [4] Li Y. Factors Affecting the Dynamic Interaction of Bridges and Vehicle Loads. PhD thesis, University College Dublin, Ireland, 2006.
- [5] Smith IM, Griffiths DV. *Programming the Finite Element Method*. Wiley, 2004.
- [6] Lou P, Au FTK. Finite element formulae for internal forces of Bernoulli-Euler beams under moving vehicles. *Journal of Sound and Vibration* 2013; 332: 1533-1552. DOI: [10.1016/j.jsv.2012.11.011](https://doi.org/10.1016/j.jsv.2012.11.011).
- [7] Arvidsson, T. *Train-Track-Bridge interaction for the analysis of railway bridges and train running safety*. PhD Thesis, KTH, Sweden, 2018.