VREP Documentation

Ratnangshu Das

February 11, 2019

1 Why Simulation?

Simulation modeling solves real-world problems safely and efficiently. It provides an important method of analysis which is easily verified, communicated, and understood. Simulation enables experimentation on a valid digital representation of a system.

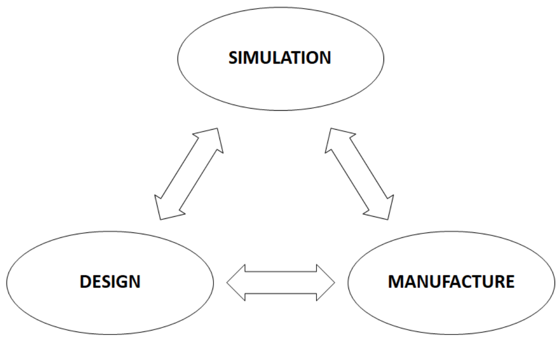


Figure 1:

2 Introduction to VREP

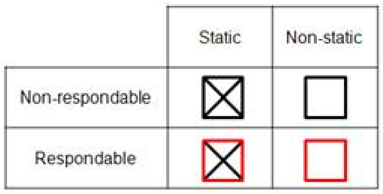
The robot simulator V-REP, with integrated development environment, is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, ROS nodes and remote API clients. Controllers can be written in C/C++, Python, Java, Matlab or Lua.

3 Designing Dynamic Simulations

In V-REP, only a limited number of objects will be dynamically simulated. Those are shapes, joints and force sensors, but it will depend on the scene structure and object properties.

3.1 Shapes

Shapes can be classified into 4 groups depending on their behavior during dynamic simulation:



	Static	Non-static
Non-respondable	☒	☐
Respondable	☒	☐

Figure 2:

- Static: Position relative to parent object is fixed (not influenced)
- Non-Static: Directly influenced by gravity or other constraints
- respondable: Influence each other, during dynamic collision

Dynamic and Respondable Will Fall(due to gravity) and able to react to collisions with other respondable shapes. **(simulated by the physics engine)**

3.2 Dynamically enabled joints

Non-static shapes will fall (i.e. be influenced by gravity) if they are not otherwise constrained. Dynamic constraints between shapes can be set-up by attaching two shapes together with a dynamically enabled joint (or with a dynamically enabled force sensor).

- Are in force or torque mode (or that operate in hybrid fashion)
- Have a shape as parent object
- Have exactly one child object which must be a non-static shape

3.3 Design Consideration

Use convex shapes instead of random shapes(pure shapes): Whenever possible, try using pure shapes as respondable shapes. This is however not always easy to do, or practical (think of a torus for instance). In that case,we generate a convex shape, or a convex decomposed shape (i.e. a simple/compound shape containing only convex meshes). Convex shapes perform faster and are more stable than random shapes (but they are still not as fast and stable as pure shapes!).

Select the shapes you wish to simplify as convex shapes, then select **[Menu bar → Edit → Morph selected shapes into convex shapes].**

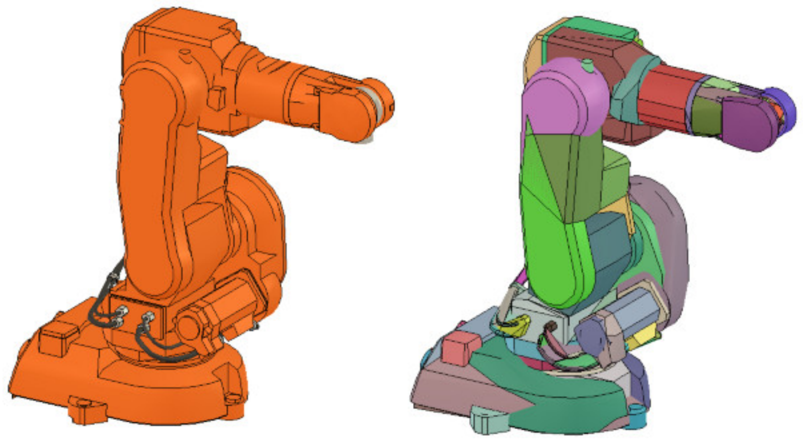


Figure 3: Non-convex model (left) and corresponding convex-decomposed model (right)

Note: To make an object usable by the other calculation modules (eg: minimum distance calculation module), enable: Collidable, Measurable, Renderable and Detectable in the 'object common properties' for that shape.

4 STEPS

1. **IMPORT 2 sets of the same stl file**:

   We'll import stl file of the entire bot.(We'll divide them later)

   [**File → Import → Mesh**]

   - **stl 1:** Break Down(morph) into convex shapes; on which physics engine will work. Hide them in the final assembly.

     [**object common properties → Visibility → all cameras and vision sensors ↔ never visible**].
   - **stl 2:** Just Divide for similar grouping as stl 1; this is visible in final simulation.

2. **ORIENT both the stl files**:

   We'll have to orient them properly such that, the model doesn't fall, when Physics Engine starts working on them.

   Trick: set orientation wrt world frame to (0,0,0) (or replace some 0 with 180), such that, body axis and world axis are aligned.

3. **DIVIDE both the stl files**:

   [**Edit → Grouping/Merging → Divide selected shapes**]

4. **MERGE divided pieces**:

   We'll now merge those divided pieces that belong to the same physical part(share the same visual attributes).

   [**select multiple sub parts by pressing CTRL → Edit → Grouping/Merging → Merge selected shapes**]

   Do this for both the imported meshes.

   Trick: Merging before Morphing gives us the advantage of having same numbers for same parts of two different imported meshes. We can also note down the corresponding numbers and rename the final merged sub part accordingly.

5. **GROUP parts**:

   Now, group those parts together, that are supposed to move together(belong to the same rigid entity).

   [**select multiple sub parts by pressing CTRL → Edit → Grouping/Merging → Group selected shapes**]

   Trick: Rename the final grouped parts accordingly.

6. **MORPH**:

   Select all the parts belonging to stl 1 and hide them. Now MORPH all of these parts into convex shapes.

   [**Right on the part → Edit → Morph selection into its convex decomposition...**

   These are the parts on which Physics Engine will work.

7. **ADD JOINTS**:

   Insert a revolute joint into the scene:

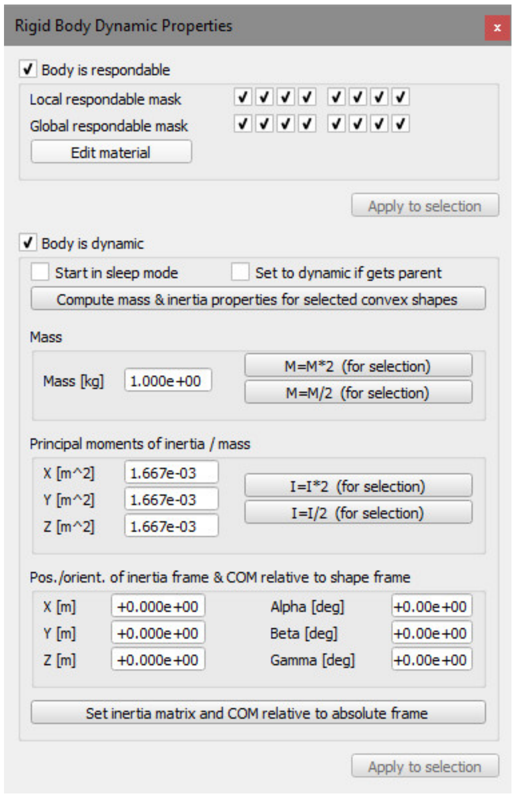
   [**Menu bar → Add → Joint → Revolute**]

   - The default position is at (0;0;0) and its default orientation is vertical, and so the joint might be hidden by the bot.
   - While the joint is still selected, ctrl-select the part, then open the position dialog on the position tab and click the Apply to selection. This just positioned the joint at the exact same coordinates as that part.
   - Set the correct position of the joint, using **Mouse Translation** or **Position**. Similarly set the correct orientation.
   - adjust the joint sizes (check the Joint length and Joint diameter items) in the joint properties dialog (that you can open by double-clicking a joint icon in the scene hierarchy)

- Select all joints, then in the joint dialog:
  "Show dynamic properties dialog"
  → **Motor enabled**
  → **Control loop enabled**

8. **SHAPE DYNAMIC PROPERTIES**

   The shape dynamics dialog is part of the shape properties. The dialog displays the dynamics settings and parameters of the last selected shape.



Rigid Body Dynamic Properties

✓ Body is respondable

Local respondable mask ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

Global respondable mask ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

Edit material

Apply to selection

✓ Body is dynamic

☐ Start in sleep mode ☐ Set to dynamic if gets parent

Compute mass & inertia properties for selected convex shapes

Mass

Mass [kg] 1.000e+00

M=M*2 (for selection)

M=M/2 (for selection)

Principal moments of inertia / mass

X [m^2] 1.667e-03

Y [m^2] 1.667e-03

Z [m^2] 1.667e-03

I=I*2 (for selection)

I=I/2 (for selection)

Pos./orient. of inertia frame & COM relative to shape frame

X [m] +0.000e+00 Alpha [deg] +0.00e+00

Y [m] +0.000e+00 Beta [deg] +0.00e+00

Z [m] +0.000e+00 Gamma [deg] +0.00e+00

Set inertia matrix and COM relative to absolute frame

Apply to selection

Figure 4:

→ Make the body **"respondable"**

→ Make the body **"dynamic"**

- **Body is respondable:** if enabled, then the shape will produce a collision reaction with other respondable shapes, however only if the respective respondable masks overlap

- **Respondable mask:** indicates when a collision response is generated. The mask is composed by two 8-bit values, local and global. If two colliding shapes share any of their parents (direct or indirect), then the local masks are used, otherwise the global masks are used. If two shapes AND-combined masks (local or global) is different from zero, then a collision response will be generated.

  → AND-combined result zero implies share no tick.

→ Set Accurate Mass

→ Set Accurate **Principle moments of inertia/mass**, obtained from **Inventor**

→ Set Accurate **Pos./orient. of inertia frame ...**, obtained from **Inventor**

9. **HIERARCHY**:

   We'll build the kinematic chain, going from main body to end of the links: select the child object, then ctrl-select the parent object and click

   [**Menu bar → Edit → Make last selected object parent**]

   If object A is built on top of object B, then object B is the parent and object A is the child. Create a parent-child relationships between object B and object A(Make last selected object parent). Both objects keep their respective configuration. However, looking at the scene hierarchy, you can see that object A became child of object B. If you now move object B, object A will automatically follow, since object A is attached to object B. Object A can be detached by selecting it, then selecting "Make selected object(s) orphan". Doing so will detach object A without changing its configuration.

   - **Set Hierarchy of the Joints**

     → Child of that link, to which it is fixed.

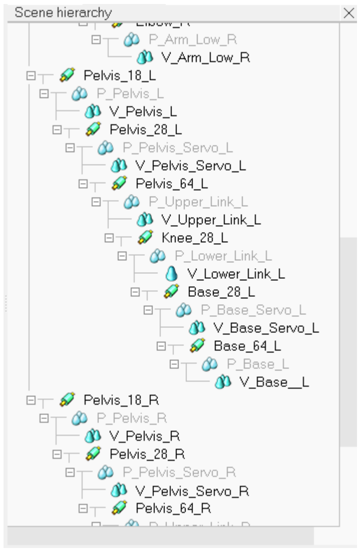
     → Parent of that link, which it is moving.

     Note: The links mentioned above, belong to stl 1,i.e, on which physics will work.

   - **Set Hierarchy of stl 1 and stl 2**

     → Similar part of stl 2 should be a child of stl 1. Understand that, it is links of stl 1, which is moving, because of constraints. We decide to hide them and show links of stl 2, which will just move along with the morphed stl 1 links.

   In my project the end result looks something like this:



```
Scene hierarchy
        P_Arm_Low_R
          V_Arm_Low_R
Pelvis_18_L
  P_Pelvis_L
    V_Pelvis_L
  Pelvis_28_L
    P_Pelvis_Servo_L
      V_Pelvis_Servo_L
    Pelvis_64_L
      P_Upper_Link_L
        V_Upper_Link_L
      Knee_28_L
        P_Lower_Link_L
          V_Lower_Link_L
        Base_28_L
          P_Base_Servo_L
            V_Base_Servo_L
          Base_64_L
            P_Base_L
              V_Base__L
Pelvis_18_R
  P_Pelvis_R
    V_Pelvis_R
  Pelvis_28_R
    P_Pelvis_Servo_R
      V_Pelvis_Servo_R
    Pelvis_64_R
```

Figure 5: