Departamento de Ingeniería de Sistemas y Automática

Universitat Politècnica de València

Simulation in robotics with *VirtualRobot*. Introduction to *VRM*

Martin Mellado Arteche

Escuela Técnica Superior de Ingeniería **Informática**

September, 2012

Contents

Introduction to the graphical modelling system for robotics *VRM*

1. Introduction

The aim of this document is to present a tutorial to introduce the use of a modelling system for robot objects, parts and environments called *Virtual Robot Modeller (VRM)* and its use within *VirtualRobot Simulator (VRS). VRM* is a set of external graphical modelling programs or applications aimed at robotics applications for their use in *VRS*. *VRM* is installed on installing *VRS*. The early part of the tutorial demonstrates how to use such programs by means of a series of tasks to be carried out. Later on it is shown how the results may be used in *VRS*. It is assumed that the user is familiar with the *VRS* program and is able to run applications in such environment.

2. *VRM* applications

In *VRS* and environment may be a loaded to make the simulation more realistic. The environments comprise objects and parts. The difference between objects and part is that whilst objects are merely "decorative" elements, that is, only graphical information to improve the visual aspect of a simulation, parts contain more information. In particular, they have associated with them coordinate systems used for the robots to act on them, both by means of relative movement and for grasping parts.

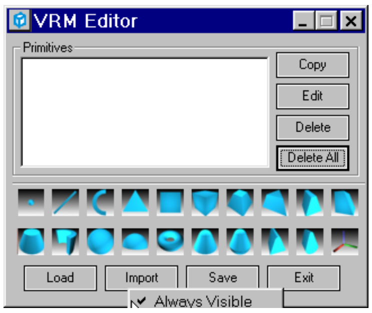
VRM applications are used to create and edit objects and parts and to build environments with them. *VRM* consists, mainly, of the following three applications ***VRM Tools*** (which can be run from the **File>>VRM Tools** option):

- ***VRM Editor*** for creating and editing both objects and parts.
- ***VRM Object Mover*** locating, deleting and saving objects in an environment loaded in *VRS*.
- ***VRM Part Mover*** for locating, deleting and saving parts in an environment loaded in *VRS*.

Using these applications environments may be generated which are later saved with the corresponding ***VRS Loader*** option.

3. Object modelling with *VRM Editor*

For object modelling in *VRS* the *VRM Tool* called ***VRM Editor*** is available. ***VRM Editor*** is an external application which is used for modelling objects for creating environments. As an external program, its executable file **VRMEditor.exe** may be run directly from the operating system, located in the **Applications\VRMTools** folder from the *VRS* directory The default configuration installation of *VRS* is accessible via the **File>>VRM Tools>>VRM Editor** menu option. Once run, it is a program like any other, with access via the task bar. If ***VRM Editor*** is running it will re-open in the event of it being launched again.

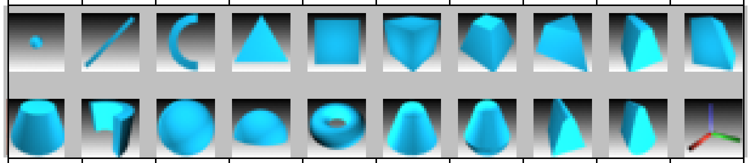


Its interface is the one shown on the right, having, like all *VRM Tool* type applications, its own *pop-up* menu, accessible on clicking on the window (outside of the buttons) with the right mouse button, enabling, if the **Always Visible** option is selected (as appears by default), this application to be always visible on top of any other window, including the *VRS* one. The interface comprises four parts: a list of primitives, editing buttons, primitive creating buttons and file buttons.

Creating primitives

The ***VRM Editor*** modeller enables objects to be created such as the joining of graphical primitives. These primitives are created with the corresponding graphic buttons, one for each type of primitive giving a total of 20 buttons. The available graphic primitives and their icons are the following:

POINT	LINE	DISK	TRIANGLE	3DFACE	BOX	PYRAMID	TRIANGULAR_ PYRAMID	TENT	WEDGE

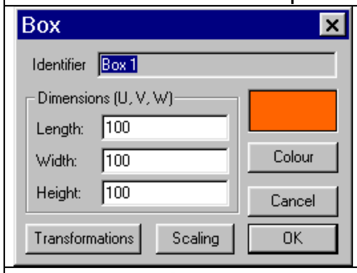


CONE	TUBE	SPHERE	DOME	TORUS	CONE_ SPHERE	CONE_TWO_ SPHERES	TENT_ CYLINDER	TENT_TWO_ CYLINDERS	FRAME

Each geometric primitive is defined using a series of geometric parameters (dimensions). The definition of the geometric parameters for each primitive is explained in **Appendix A**.

To create a primitive the corresponding button is selected. For example, when the **Box** primitive icon is selected the following dialogue appears to introduce the data:

Identifier: name of the primitive (this field is not modifiable)


Box

Identifier: Box 1

Dimensions (U, V, W)

Length: 100

Width: 100

Height: 100

Colour

Cancel

Transformations

Scaling

OK

Dimensions (U, V, W): dimensions or geometric parameters. In the case of the prism they are: **Length**, **Width**, **Height**
Colour: opens a dialogue for selecting the primitive's colour. The colour selected is represented in the top rectangle
Cancel: cancels the definition of the primitive
OK: Saves and ends the definition of the primitive

Transformations: opens a dialogue enabling the primitive to be located. This dialogue is explained below
Scaling: opens a dialog enabling the primitive to be scaled (it multiplies all its geometric parameter scaling factor greater than zero)

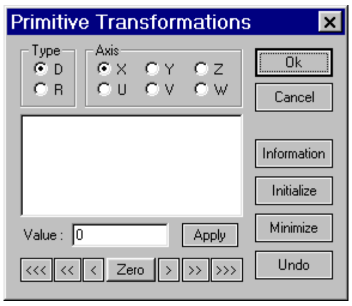
In this dialogue the dimensions should be important (optionally, scaling them) and the right colour selected. Transformations are available to locate the primitives as explained in the next section.

Task 3.1:
- Click on the Box icon to define a prism (**Do NOT click on the OK button** in this task, as it continues in Task 3.2).
- Try changing its dimensions to see how they fit dynamically.
- Change the colour of the primitive.
- Scale the primitive to twice the size (**Scale Factor**=2) and half the size (**Scale Factor**=0.5).

Placing primitives

The **Transformations** button opens the following dialogue:

Type: defines the type of transformation: Displacement (D) or Rotation (R)
Axis: selects not what axis the transformation will be applied this may be a global axis (X,Y,Z) or a local one (U,V,W)
Ok: ends the definition of the transformations and returns to the previous dialogue (primitive creation) saving the transformations made



Cancel: cancel the definition of the transformations and returns to the previous dialogue (primitive creation) returning the transformations to the state they were in on opening this dialogue
Information: shows the homogeneous transformation matrix
Initialize: removes all the transformations from the list
Minimize: minimizes the transformation to six or fewer basic transformations (RX,RY,RZ,DX,DY,DZ)
Undo: undoes the last basic transformation in the list and removes it
Value: inputs a displacement value or angle (in degrees)
Apply: applies the basic transformation defined by term (**Type**,**Axis**,**Value**) and adds to the end of the list
Zero: resets the **Value field to zero**
<<<, **<<**, **<**: subtract 100,10,1 from the value of the **Value field**
>, **>>**, **>>>**: add 1,10,100 respectively to the value of the **Value field**

The use of this dialogue, which appears somewhat difficult, is easier than what it seems. It enables the application of displacements and rotations of the object on their main axes U,V,W (which move together with the object) or the global axes of the “world” X,Y,Z (which remained static or fixed). For help using this dialogue, the axes of both coordinate systems are drawn (initially overlapped), using the nomenclature of the three axis with the colours red, green and blue (RGB), which represent the X,Y,Z or the U,V,W axes respectively of such systems.

Displacement or transformation operations may be carried out on each of these axes. Unless axes are parallel, the effect of the displacements on one or the other will be different. Likewise, unless the two axes are coincident, the effect of the rotations will be different.

To help you understand how they work, do the following tasks.

Task 3.2:
- Carry out displacements (Type=D) on axes X,Y,Z (fixed axes) with the value 100 (or any other) to familiarize yourself with their use. Notice that once the type is defined, the axis and the basic transformation value must be applied using the **Apply** button.
- Carry out displacements on axes U,V,W. Notice that the effect is the same (displacing on U does the same as displacing on X) since the axes remain parallel.
- Test the **Information**, **Undo**, **Minimize** and **Initialize** options.

Remember that you are using VRS, so you have all the graphical viewing facilities at your disposal (zoom, scroll, point of view, etc).

Task 3.3:
- Initialize the transformation (**Initialize** button).

- Carry out a 60° angle rotation (Type=R) on the Z axis.
- Carry out displacements on the X and U axes. Deduct the difference between the displacements in respect of the fixed and local axes.
- Carry out displacements on the Y and V axes. Deduct the difference between the displacements in respect of the fixed and local axes.

Task 3.4:

- Initialize the transformation (**Initialize** button).
- Carry out displacements on X, Y, Z.
- Carry out rotations on the U,V,W axes (main axes of the object) choosing the desired angle.
- Carry out rotations on the X,Y,Z axes (fixed axes of the world). Deduct the difference between the rotations in respect of the fixed and local axes.

As this dialogue is essential for locating the primitives which form objects, you need to be skilled about how to handle the dialog before continuing with the next section.

Task 3.5:

- Choose the place of whichever primitive you like.
- Close the transformations dialogue (**Ok** button).
- Close the definition dialogue of the **Box** primitive (**OK** button). From this moment you have a primitive in the modeller.

Copying, editing and deleting primitives

Already created primitives may be duplicated using the **Copy** button. You just have to select the primitive to duplicate from the list of existing primitives and click on the **Copy** button. The create dialogue will open for the dimensions, colour or place of the copy to be changed (otherwise both primitives will overlap each other). In case of cancelling, the copy is lost.

Any primitive input may be edited, that is, any of its dimensions or geometric parameters may be modified, its colour changed, scaled or its place modified. To do this its name must be selected from the list of primitives and the **Edit** button clicked on and the create dialogue of the primitive to be able to change the dimensions, colour or place of the primitive. On the other hand, any primitive input may be deleted. To do so its name must be selected from the list of primitives and the **Delete** button clicked on. Note that it does NOT ask for confirmation and that you cannot recover a primitive once it has been deleted. Finally, all primitives created in *VRS* may be deleted using the **Delete All** button which will ask for confirmation before carrying out the action.

Task 3.6:

- Input several primitive of different types (except **Frame**) and locate them in space by making use of the different options.
- Practice the edit, copy and delete options for primitives so as to be used with them.

Saving objects

Once all the primitives that make up an object have been input, it may be saved in a file using the **Save** button. After a dialogue for selecting the directory and inputting the name of the file, the existing primitives are saved in a file with an **OBF** extension. To keep the file structure consistent, it is recommended that the file be saved in the **Models\User** folder (from which the files may be organized by folder). The object is saved in a file in ASCII format (modifiable therefore with any text editor). The **OBF** format is explained in Appendix B.

Task 3.7:

- Freely model a table with dimensions of 120cm x 60cm y 75cm high (remember to use when modelling the object).
- Save the table in a file (in the **Models\User** folder).

Loading and importing the primitives of objects

Primitives saved in object files may be loaded using the **Load** button. If the primitives created have not been saved confirmation will be sought, since otherwise they will be lost. On the other hand, during creation of an object all the primitives of an already created object may be imported and saved in a file using the **Import** button which enables the object file to be selected. All primitives of which the object is made up stored in the file are added to the object currently being created.

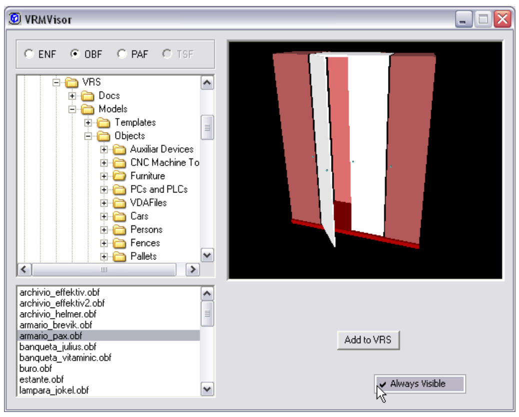
Task 3.8:

- Freely model a couple of simple objects (maximum 4 primitives) and save them in two files.
- Create a new object by importing the table and the previous objects. Notice that once the objects are imported only their primitives may be accessed and the grouping is lost, so the object as a whole may not be re-located (this may be resolved with the definition of environments explained in the next section).
- Delete this new object.

4. Modelling environments with objects

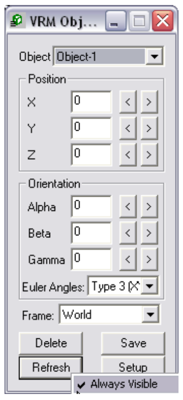
As seen in the previous section, with the aid of *VRM Editor* primitives for modelling object may be represented in *VRS* and stored in files. If you want *VRS* can represent these objects as grouped entities, environment files must be created. An environment is the union of objects (later we shall see that parts may also be included).

To load objects which define environments use of the find environments option available in *VRS Loader* (button) is recommended, this runs the ***VRM Visor*** program. Its interface is the one shown on the right, having, like all accessible mouse *VRM Tool* type applications, its own *pop-up menu* on clicking on the window (outside of the buttons) with the right button, enabling, if the **Always Visible** option is selected (as appears by default), this application to be always visible on top of any other window, including the *VRS* one.



With this application you can view and load environment files into *VRS* by launching the **ENF** option and object files by launching the **OBF** option. It may also be used for parts files by launching the **PAF** option. ***VRM Visor*** shows on the left the directory tree (top) and the content of the directory (bottom) and on the right a view of the content of the file in a graphical window. By double clicking with the mouse on the graphical window you can configure the use of the mouse in the graphical window. Using the ***Add to VRS*** button you can add objects to the currently available environment in *VRS*.

Once at least one object is loaded into *VRS* (otherwise it will not start), you can run the ***VRM Object Mover*** application, accessible using the **File>>VRM Tools>>VRM Object Mover** menu option This application enables objects to be located, deleted and saved in an environment, although its main function is to change the position and orientation of an already existing object in the environment and to enable environments to be modelled by proceeding in this way. Its interface is the one shown on the right, having, like all accessible mouse *VRM Tool* type applications, its own *pop-up menu* on clicking on the window (outside of the buttons) with the right button, enabling, if the **Always Visible** option is selected (as appears by default), this application to be always visible on top of any other window, including the *VRS* one.



From the **Object** field the object to be re-located may be selected and using the "**<**" y "**>**" buttons or typing directly the value of position components x,y,z or orientation ones α,β,γ may be changed. Orientation can be specified in any of the three types of Euler angles and the place can be represented in respect of the world system or of the environment (initially coincident, they only vary if this relationship has been modified using **Place Environment**). The **Delete** button is for deleting an object from the environment, the **Save** button for saving an object in its current place, the **Refresh** button for updating the list of objects (required when another application has loaded new objects) and the **Setup** button for configuring the "**<**" y "**>**" button increments.

Using these two applications objects can be loaded and placed in VRS in order to create environments which may be saved using the **Save Environment** button in ***VRS Loader***. It is recommended that files be saved in the **Models\User** folder. Environments are saved in files with the **ENF** extension whose format is described in Appendix B. As previously explained, these files may be loaded into *VRS* using the **Load Environment** button in ***VRS Loader***.

Task 4.1:
- Make sure you have an empty environment in *VRS*.
- Add the table defined in the previous section into *VRS*.
- Add one of the objects defined in the previous section into the *VRS*.
- Locate this object so that it is situated on top of the table.
- Add the other object into the *VRS* and locate it on top of the table too.
- Save the environment in the **Models\User** folder and delete the environment.
- Load the environment created.

5. Modelling parts and environments with parts

Objects modelled with *VRM* may be viewed in *VRS* to get the feeling that the robot is in the environment being worked in. However, these objects are inert objects, on which a robot cannot operate. On the contrary, parts may be defined using *VRM*, which can be operated by robots (for example handled) in *VRS*. The difference between objects and parts is that the latter, besides a set of primitives for defining their geometric shape, have at least one coordinate system will allow the robot to make movements relative to this coordinate system or pick up the part using this coordinate system.

To define parts, you work in the VRM Editor the same as for defining objects, with the only proviso that at least one **Frame** must be created using the relevant button. Once this type has been launched, *VRM* adds a coordinate system for operating the part (*operation frame*) initially with transformation identity. The **Display Size** field as well as the scaling option are only used for viewing the **Frame**, not being a definition field of a geometric parameter. To place the **Frame** the **Primitive Transformations** dialogue must be used which can be accessed using the **Transformations** button.

Parts are created in its geometric part the same as objects are created, it being possible to input, edit, copy or delete primitives in the same way. The only requisite is that a primitive **Frame** be created. Parts are saved in files with the **PAF** extension whose format is described in Appendix B. To create parts object files **OBF** and/or parts files **PAF** may be imported. It is recommended that part files be saved in the **Models\User** folder.

Parts may be included in environment files which can be viewed in *VRS* by making use of the ***VRM Visor*** application as explained earlier. The PAF option must be launched to be able to view existing parts files. Once input, they can be placed using the ***VRM Part Mover*** program, with similar features to the ***VRM Object Mover***, but for placing parts and not objects.

Now environments containing objects and parts can be saved (**Save Environment** button in ***VRS Loader***).

Task 5.1:
- Make sure you have an empty environment in *VRS*.
- Model a simple part (a pair of primitives and an operating system).
- Save the part in a file in the **Models\User** folder.

Task 5.2:
- Make sure you have an empty environment in *VRS*.
- Load the environment created in **Task 4.1**.
- Add the part created in the previous task into the environment.
- Load the **CRS A465.rkf** robot into the *VRS* and place it on top of the table.
- Active the operating system view mode by activating the **PARTS** field in the **Display>>Frames...** menu option

6. Operation on parts in *VRS*

The definition of the operating systems is essential for programming in *VRS* the operations of:
- Moving a robot so that the active coordinate system of one of its tools coincides with the operating system of a part (*MoveToPart*).
- Moving a robot so that the active coordinate system of one of its tools approximates to the operating system of a part (*ApproxToPart*).
- Making a robot tool pick up a part forcing the relationship between active coordinate system of one of its tools and the operating system of a part to be constant (*PickPart*).

To define where to put a part's operating system for it to be useful, what type of operation is to be carried out on the part and where the tool system of the robot which will carry out the operation needs to be known.

For example, let us suppose that we want to design a part that can be handled by the CRS A465 robot which has a pneumatic vacuum-type tool (Figure 2). The coordinate system of this tool is modelled in *VRS* with its source situated in the centre of its outer face, the X axis pointing outwards from the vacuum tool and the Z axis vertically upwards in the rest position of the robot.

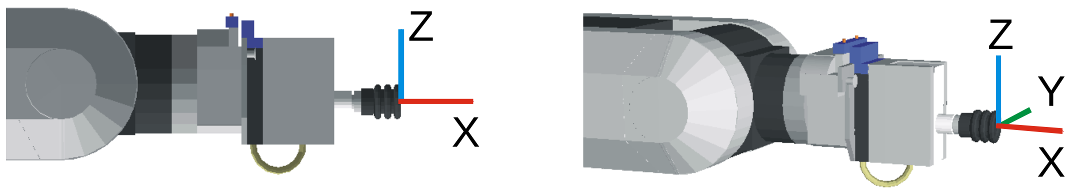


Figure 2. Model of the CRS A465 vacuum tool and its ToolFrame.

In order to simulate that the robot with the vacuum tool pick up a part, an operating system has to be placed on the part as shown in Figure 3a, so that the robot can be instructed to make a movement to place the vacuum tool coordinate system to coincide with the operating system as can be seen in Figure 3b, so that the vacuum tool can pick up the part.

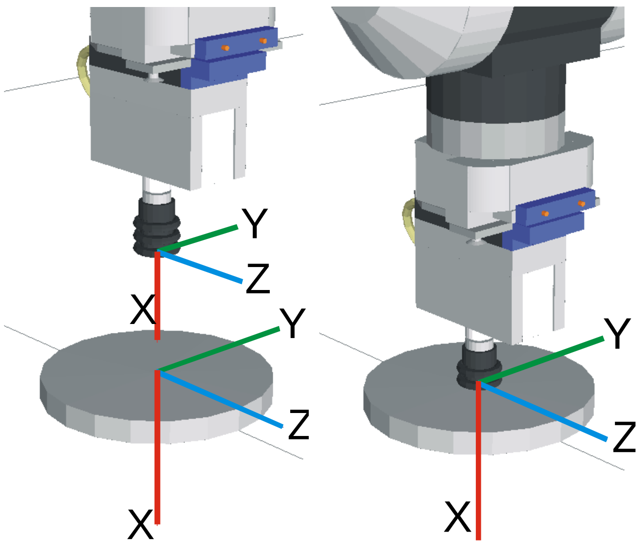


Figure 3. a) Operating system b) Gripper situation.

If the part is a cylinder located on the floor, the following two transformations will be sufficient:

- 90º V rotation to direct the system with the vertical axis X downwards.
- Displace the height of the cylinder along Z to locate the source of the operating system on the upper side of the cylinder.

Task 6.1:

- Model a part comprising a cylinder which can be handled by the CRS A465 using the vacuum tool by means of the operating system.
- Save the part to a file.
- Create an environment which has as many objects as you like but with the previous cylinder as the only part placed with a single transformation which is a displacement of 500mm on the X axis.
- Save the environment in the **Models\User** folder.
- Load the **CRS A465 Vacuum.rkf** robot from the **Models\Tutorial** folder.
- Using ***VRS PartHandling***, handle the part.

7. Practical exercises

The CRS A465 robot may also have a servo-controlled electromechanical gripper as a tool. The coordinates system of this tool (Figure 4) is modelled in *VRS* with its source situated on the plane on which the fingers touch on closing, protruding 21mm from the base plate of the fingers and sunk 4mm from the edge of the fingers (25mm long). Orientation is similar to that of the vacuum tool coordinate system.

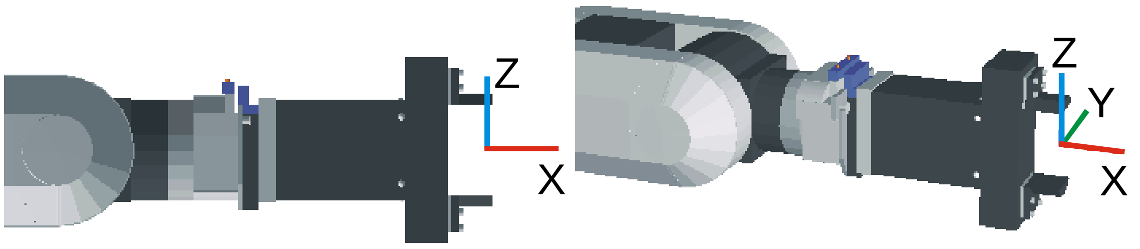


Figure 4. Model of the CRS A465 gripper tool and its ToolFrame.

The movement of the gripper fingers is modelled by means of 4 states with 51mm openings between the fingers (open gripper), 33mm (semi-open) 15mm (semi-closed) and 1mm (closed gripper) as shown in Figure 5.

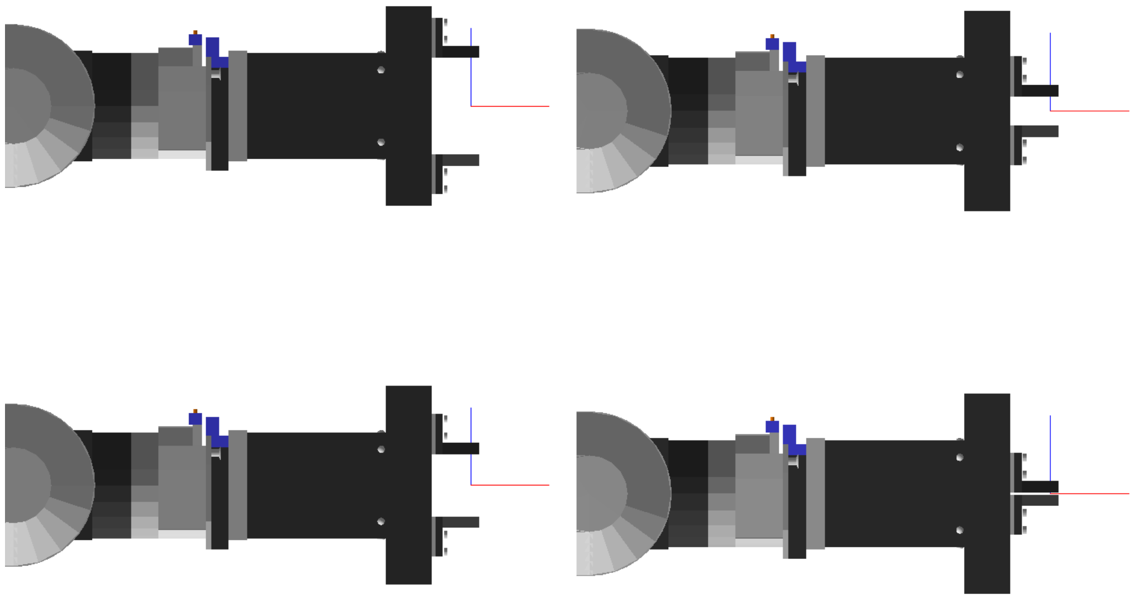


Figure 5. Models of gripper states of the CRS A465.

Task 7.1:
Model an environment which has **two parts** for handling by the CRS A465 robot (place the part in a place reachable by the robot) in the following way:

- The first part must be grasped properly with the gripper in the semi-open state.
- The second part must be grasped properly with the gripper in the semi-closed state.

Handle the parts in *VRS* using ***VRS PartHandling***.

Task 7.2:
Model an environment which has a part for handling by the CRS A465 robot in the following way:

- Using the operating system the part must be grasped properly with the gripper in the semi-open state.
- Using the operating system the part must be grasped properly with the gripper in the semi-closed state.

Handle the parts in *VRS* using ***VRS PartHandling***.

A. Primitive definition

Geometric primitives and their definition parameters are outlined in the VRPD (VirtualRobot Primitive Definition) document but a summary graphic of them is provided below by way of help.

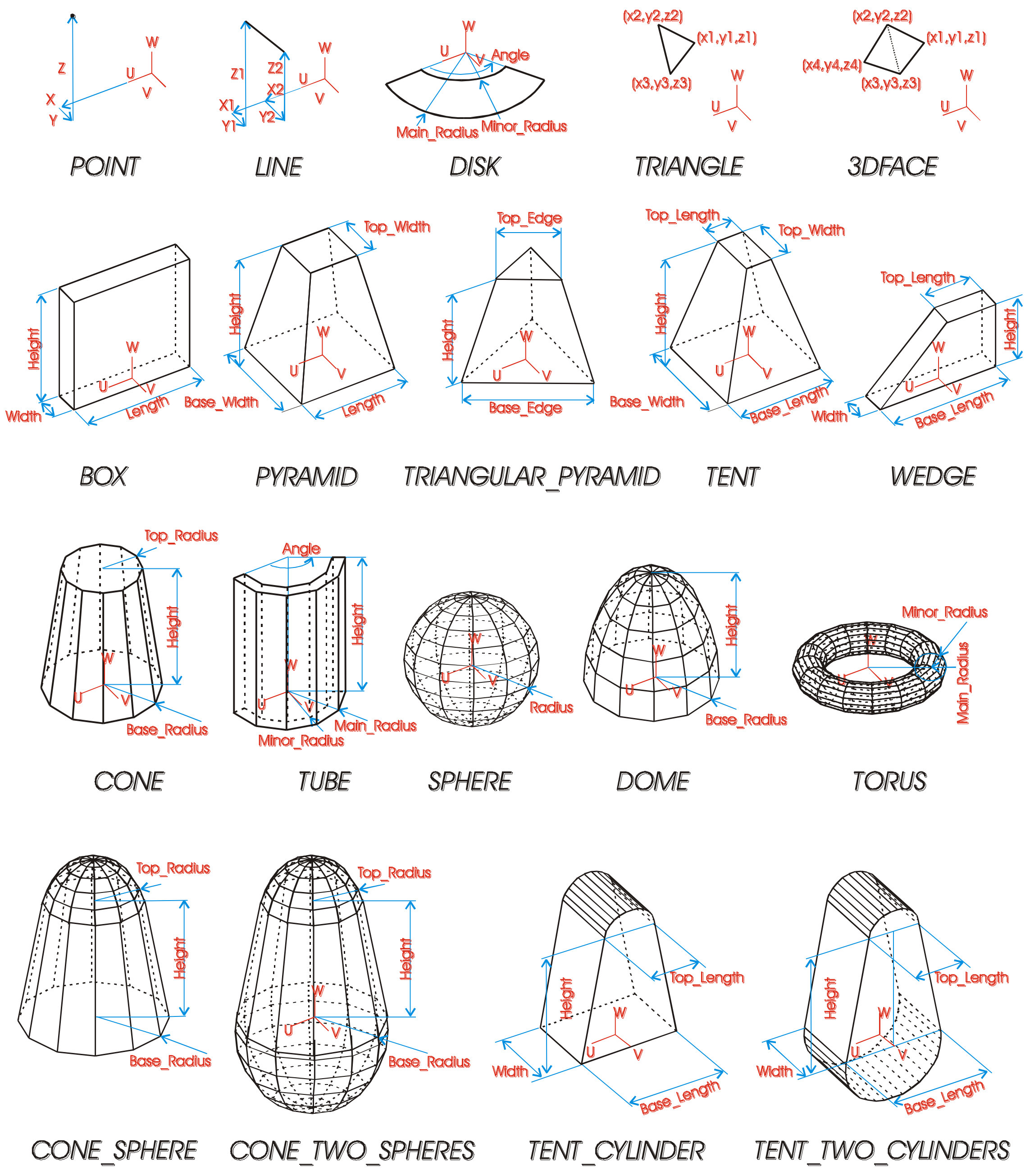


Figure A1. Geometrical Primitives and Definition Parameters.

B. File formats

The format of OBF, PAF and ENF files is outlined in the VRFFD (VirtualRobot File Format Definition) document and is summarized below (“;” indicate commentary to line end, the tabulators are not significant).

```
; OBJECT DESCRIPTION FILE (OBF)
; This file describes an Object as a set of Primitives

[GENERAL]                                        ; General Information
        Name            = Table                  ; Object Name
        Manufacturer    =                        ; Robot Manufacturer (for robot components)
        Model           =                        ; Robot Model (for robot components)
        Author          = Juan Vte. Catret       ; Author
        Company         = DISA - UPV             ; Company
        Date            = 9/2/99                 ; Date

[PRIMITIVE1]                                     ; First Primitive of Object
        Type    = BOX                            ; Type of Primitive:
           ; POINT (x, y, z)
           ; LINE (x1, y1, z1, x2, y2, z2)
           ; DISK (Main_Radius>0, Minor_Radius>=0, 0<Angle<=360 [Main_Radius>=Minor_Radius])
           ; TRIANGLE (x1, y1, z1, x2, y2, z2, x3, y3, z3)
           ; 3DFACE (x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4)
           ; BOX (Length>0, Width>0, Height>0)
           ; PYRAMID (Length>0, Base_Width>0, Top_Width>=0, Height>0)
           ; TRIANGULAR_PYRAMID (Base_Edge>0, Top_Edge>=0, Height>0)
           ; TENT (Base_Length>0, Base_Width>0, Top_Length>0, Top_Width>=0, Height>0)
           ; WEDGE (Base_Length>0, Width>0, Top_Length>0, Height>0)
           ; CONE (Base_Radius>0, Top_ Radius>=0, Height>0)
           ; TUBE (Main_Radius>0, Minor_Radius>=0, 0<Angle<=360 [Main_Radius>Minor_Radius] , Height>0)
           ; SPHERE (Radius>0)
           ; DOME (Base_Radius>0, Height>0)
           ; TORUS (Main_Radius>0, Minor_Radius>0)
           ; CONE_SPHERE (Base_Radius>=0, Top_Radius>0, Height>0)
           ; CONE_TWO_SPHERES (Base_Radius>0, Top_Radius>0, Height>0)
           ; TENT_CYLINDER (Width>0, Base_Length>=0, Top_Length>0, Height>0)
           ; TENT_TWO_CYLINDERS (Width>0, Base_Length>0, Top_Length>0, Height>0)
           ; VDA_CURVE (No_Segments, ...)
           ; VDA_SURF (No_Patches, ...)
           ; VDA_FILE (FileName, Thickness>0, Curve_Sections>0, Surface_Sections>0)
           ; OBF_FILE (FileName)
           ;     For VDA_FILE and OBF_FILE, the file name must include extension and start from VRS Path
        RGB     = 255,0,0        ; Red-Green-Blue (RGB) components for Colour (0..255,0..255,0..255)
                        ; This field can be avoided for OBF_FILE primitive (it has no effect if exists)
        Length  = 30.00          ; Name and value of parameters according to type of Primitive
        Width   = 100.00         ; Dimensions on mm (degrees for angles)
        Height  = 150.00
        Transformations = DX>5.000, RZ>90.000    ; Location of Primitive1 related to Object Frame
                ; Defined with as many Basic Transformations as required
                ; Transformations can be: Displacements (D) or Rotations (R)
        ; The axis is indicated with (X,Y,Z,U,V,W) before the symbol '>'
        ; Transformations related to Object (X,Y,Z) or Primitive (U,V,W) Frames
        ; The transformations will be applied in the specified order
        ; Dimensions expressed on mm or degrees (real numbers)
        ; Transformations = I must be used for identity (no rotation or translation)

[PRIMITIVE2]                                     ; Second Primitive of Object
        ..
        ..              ; As many Primitives as required to define Object (at least one)
```

```
; PART DESCRIPTION FILE (PAF)
; This file describes a Part as an Object with a Operating Frame defined for Robot Operation

[GENERAL]                                         ; General Information
        Name            = Screw                   ; Part Name
        Manufacturer    =                         ; Robot Manufacturer (for robot components)
        Model           =                         ; Robot Model (for robot components)
        Author          = David Puig              ; Author
        Company         = DISA - UPV              ; Company
        Date            = 9/2/99                  ; Date

[OPERATION1]                              ; Location of Operation Frame 1 related to Part Frame
      Transformations   = I               ; Defined with Basic Transformations (related to Part Frame)

[OPERATION2]                              ; Location of Operation Frame 2 related to Part Frame
      Transformations   = I               ; Defined with Basic Transformations (related to Part Frame)

        ..                                ; As many Operation Frames as required to define Part
                                          ;         (at least one)

[PRIMITIVE1]                        ; Defined as a Primitive in Object Geometric Description File
        ..                          ; Transformations related to Part (X,Y,Z) or Primitive (U,V,W) Frames

[PRIMITIVE2]
        ..

        ..                                ; As many Primitives as required to define Part (at least one)
```

```
; ENVIRONMENT GEOMETRIC DESCRIPTION FILE (ENF)
; This file describes an Environment as a set of Objects and Parts

[GENERAL]                                          ; General Information
        Name            = RoboticsLaboratory       ; Environment Name
        Manufacturer    =                          ; No sense for Environment
        Model           =                          ; No sense for Environment
        Author          = Carlos Correcher         ; Author
        Company         = DISA - UPV               ; Company
        Date            = 9/2/99                   ; Date

[OBJECT1]
      Name              = Table         ; Object Name
      Transformations   = I             ; Location of Object1 related to Environment Frame
[OBJECT1_PRIMITIVE1]                    ; Defined as a Primitive in Object Geometric Description File
      ..                                ; Transformations related to Object (X,Y,Z) or
                                        ; Primitive (U,V,W) Frames
[OBJECT1_PRIMITIVE2]                    ; As many Primitives as required to define Object1 (at least 1)
      ..

[OBJECT2]
        ..

        ..                              ; As many Objects as required to define Environment (even none)

[PART1]
      Name              = Screw         ; Part Name
      Transformations   = I             ; Location of Part1 related to Environment Frame
[PART1_OPERATION1]
      Transformations   = I             ; Location of Operation Frame 1 related to Object Frame
[PART1_OPERATION2]                      ; As many Operation Frames as required to define part (at least 1)
      ..
[PART1_PRIMITIVE1]                      ; Defined as a Primitive in Object Geometric Description File
      ..                                ; Transformations related to Part (X,Y,Z) or Primitive (U,V,W) Frames
[PART1_PRIMITIVE2]                      ; As many Primitives as required to define Part1 (at least 1)
      ..

[PART2]
        ..

        ..                              ; As many Parts as required to define Environment (even none)
```