http://doi.org/10.35784/iapgos.5856

received: 01.02.2024 | revised: 27.04.2024 | accepted: 12.05.2024 | available online: 30.06.2024

MODELING ROBOTECHNICAL MECHATRONIC COMPLEXES IN V-REP PROGRAM

Laura Yesmakhanova
Taraz Regional University named after Dulati M. K., Taraz, Kazakhstan

Abstract. *The article clarifies the issues of modeling robotic systems in the V-REP program and provides skills in modeling the process of robotic and mechatronic complexes operation taking into account the laws of physics. The aim of the research paper is to investigate a 3D robotic simulator based on a distributed control architecture: control programs (or scripts) can be directly attached to objects in the scene and executed simultaneously in a streaming or non-streaming mode. V-REP can be used for remote monitoring, for hardware control, for rapid prototyping and verification, for rapid algorithm development/parameter tuning, for safety retesting, for robotics education, factory automation simulation, etc.*

Keywords: robotics, simulator, simulated environment, V-Rep

SYMULACJA ROBOTYCZNYCH ZESPOŁÓW MECHATRONICZNYCH W PROGRAMIE V-REP

Streszczenie. *Artykuł wyjaśnia kwestie modelowania systemów zrobotyzowanych w programie V-REP i zapewnia umiejętności modelowania procesu działania robotów i zespołów mechatronicznych z uwzględnieniem praw fizyki. Celem artykułu jest zbadanie symulatora robotycznego 3D opartego na rozproszonej architekturze sterowania: programy sterujące (lub skrypty) mogą być bezpośrednio dołączane do obiektów na scenie i wykonywane jednocześnie w trybie strumieniowym lub niestrumieniowym. V-REP może być wykorzystywany do zdalnego monitorowania, kontroli sprzętu, szybkiego prototypowania i weryfikacji, szybkiego opracowywania algorytmów / dostrajania parametrów, ponownego testowania bezpieczeństwa, edukacji robotyki, symulacji automatyzacji fabryki itp.*

Słowa kluczowe: robotyka, symulator, środowisko symulowane, V-Rep

Introduction

Nowadays, robotics is one of the most demanded directions in the development of automated technological systems and is actively used in such fields as medicine, telecommunications, military and industry, as well as education. The development of robotics and modeling technologies has led to the emergence of a whole class of software - robotics simulators. A simulator is a software platform that provides an opportunity to create various scenarios using a user interface. The special significance of simulators should be emphasized in robotics training [7]. Virtual simulators differ significantly from general-purpose simulation environments. They use an approach in which the model has components that realize the same functions as similar components in real robots. Therefore, while students are still working with the virtual model, they learn how to work with electrical actuators, mechanical elements, and control system, which allows them to reduce risks without harming the learning process.

Testing is the most common [5], but by no means the only purpose of developing experimental platforms. For example, most modern industrial robots have their own offline programming systems, which includes an experimental modeling system. Such systems perform all the necessary calculations for simulation and visualize the process. Fig. 1 shows the interface of specialized software for offline programming and testing of the executable program of industrial robots [1] produced by ABB Robotics.

The modeling goal depends on the tasks at hand, and when creating new robotic solutions, mainly on the development stage. It can be hypothesis testing, design optimization, testing of software implementing new algorithms of sensor information processing and controlling behavior, and at later stages – debugging of executable code before its launch on the manipulator workstation controller.

There exists a number of software solutions for robotic systems that allow creating simulations with high accuracy. These include Gazebo [9] and V-REP [10], as the most popular and having more functionality. These platforms have a wide range of capabilities for modeling robots of different types, ranging from floating to flying. The distinctive feature of the platforms is also the ability to scale easily. However, the developer interface in the V-REP program is significantly better than the interface in Gazebo, where there is no development interface per se. In addition, V-REP program regularly receives new updates, new features and functions appear, while Gazebo is poorly supported and no significant improvements have been seen in the last 3 years. In view of these facts, V-REP is currently the best solution for those who are just starting to explore the possibilities of modeling robotic systems and the peculiarities of creating simulations for robots in order to start using them in practice.

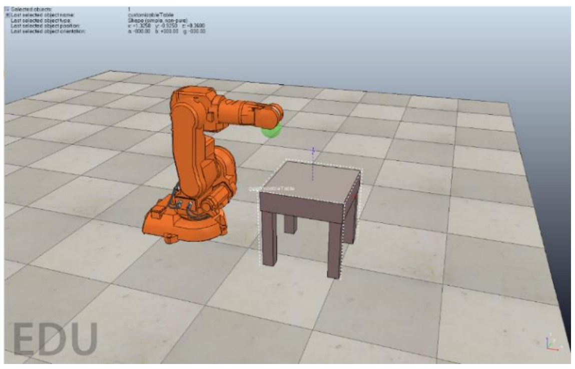


Fig. 1. ABB Robot Studio program interface

1. V-REP software capabilities

The V-REP program interface is divided into several parts depending on the purpose and is implemented in a window with a graphical interface: the graphical window of the program is used to control all built-in tools. Also worth mentioning is the console window, which can be observed during the application startup, but by default this window is hidden immediately. If necessary, you can configure V-REP to always display the console by calling "User settings" using the first button in the vertical toolbar. The console window displays the plug-ins to be loaded and their routines, which are only needed when working with plug-ins.

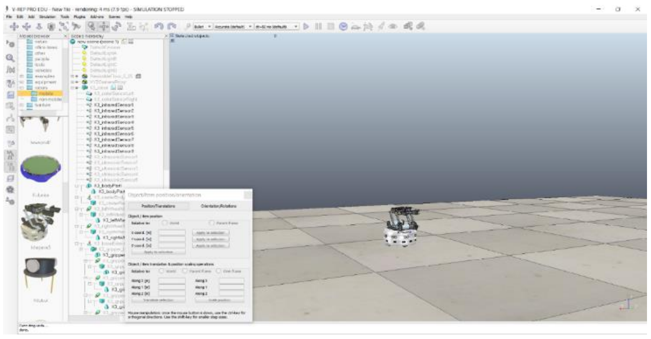


Fig. 2. V-REP application window

Figure 2 shows the V-REP program interface with a rather large set of active tools, but it should be understood that simultaneous activation of all tools will lead to their mutual overlapping.

1.1. Overview of V-REP tools

Change position, orientation and zoom. These are some of the basic tools needed to enter the initial conditions of the simulation. The "Object/Item Shift" tool (Fig. 3. left) can be used to set the position of objects in the scene and scale (reduce or enlarge the object). The "Object/Item Rotation" tool (Fig. 3. right) allows you to set the object orientation in space.

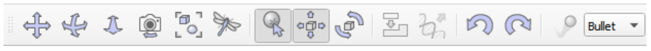


Fig. 3. Position and Orientation tools

Changing object properties. The "Properties" tool is no less frequently used when creating simulations in the V-REP program and allows you to set various properties of objects and simulation components. After activating the tool, the context menu is displayed, which has two tabs. The first tab contains the list of properties, which is specialized and depends on the type of the selected object. The second tab of the tool's context menu contains general properties that are similar for most objects.

Starting and stopping the simulation. For these tasks, there is a set of toolbar buttons in the horizontal panel (Fig. 4) that allow you to start the simulation (shown in Fig. 4, left), stop (Fig. 4, middle) and end the simulation (Fig. 4, right).

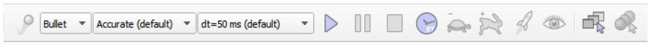


Fig. 4. Tools for starting, stopping and finishing a simulation

Also in Fig. 4 to the left of the simulation start tool shows the settings of the solver (library for modeling the laws of physics), where you can change the simulation step, the simulation mode and the used kernel of the physics engine.

Visual property management. The V-REP program implements a mechanism of layer-by-layer separation of all visual objects. This functionality is necessary because in the program it is necessary to use the combination of components of different types, and in such cases it becomes impossible to further work with objects. For example, the presence of identical components that have the same position, but model different properties, leads to the fact that they become visually indistinguishable. That is why it is necessary to use the "Layers" tool. The recommended principle for layering is that components of the same type should be on a separate layer (e.g. all joints). A total of 10 layers are available in V-REP, and in the properties of each component you can select the layer on which the element will be placed. Once the elements have been layered, you can activate the "layers" tool from the left toolbar and switch between layers.

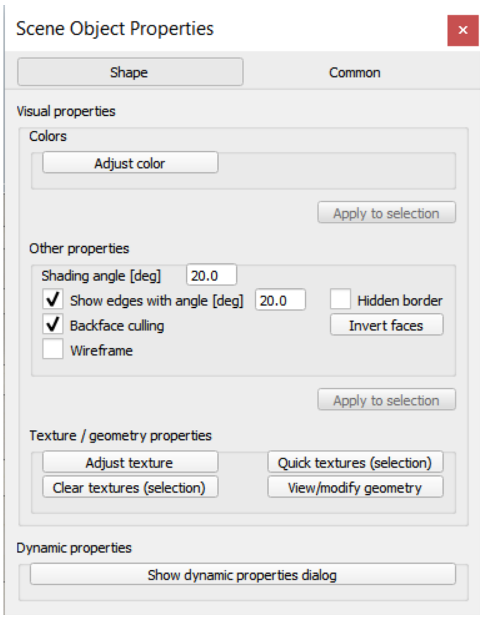
Script management tool. Adding and removing scripts, as well as changing the binding to individual components, is easily accomplished through the simulation script hierarchy. However, V-REP also has a duplicate "Scripts" tool that also allows you to perform similar operations and often does so faster. But for beginners in V-REP this tool can be difficult, because the principle of its work is not as intuitive as a similar operation through component hierarchies.

Form Editor. This tool is designed to edit the mesh of the mechanical elements of the system being modeled. Setting a finer mesh allows to obtain higher simulation accuracy, but if you chop the mesh in all components, it leads to an increase in the computational power consumed. Therefore, it is necessary to set variable concentration grid sizes through manual grid editing. This tool is most often needed when creating a simulation based on 3D models that have been imported from a CAD system. There are 3 modes of operation available in the shape editor: triangle editing mode, vertex editing mode and edge editing mode.

Triangle editing mode. In this mode, the individual triangles that make up the shape can be selected, after which they can be deleted, the remaining voids can be closed, and they can be subdivided into smaller triangles. Using the "Subdivide triangles" button you can subdivide all triangles into smaller triangles, i.e. each click reduces the size of triangles by 2 times.

Vertex editing mode. In this mode individual vertices that make up the shape can be selected and then deleted. You can also create new triangles by selecting 3 or more vertices and pressing "Insert triangles".

Edge editing mode. In this mode the functions similar to those described above are available. The only difference is that separate edges of the shape are available for selection and editing.



Scene Object Properties

Shape | Common

Visual properties

Colors

Adjust color

Apply to selection

Other properties

Shading angle [deg] 20.0

✓ Show edges with angle [deg] 20.0 ☐ Hidden border

✓ Backface culling Invert faces

☐ Wireframe

Apply to selection

Texture / geometry properties

Adjust texture | Quick textures (selection)

Clear textures (selection) | View/modify geometry

Dynamic properties

Show dynamic properties dialog

Fig. 5. Forms editing tool

It should be noted that component forms cannot be edited directly with this tool, but must be grouped in advance and modified individually.

1.2. Writing scripts in V-REP program

As it was mentioned above, the V-REP platform supports working with different programming languages, including the most common ones, such as C++ and Python. Implementation is performed using the Lua language, as this language is a built-in language of V-REP, and you can start writing scripts immediately after starting the program. A distinctive feature of Lua is its simplicity: in many respects the syntax is similar to the popular C language, which greatly simplifies the work for those who are familiar with the language. Scripts in V-REP open great opportunities for realizing control of both separate objects of the scene and the platform.

To begin with, it is necessary to remember that scripts in V-REP are of two types: streaming and non-streaming.

Let's consider the structure of a non-streaming script, as recommended by developers and most often used. Such a script always consists of several blocks.

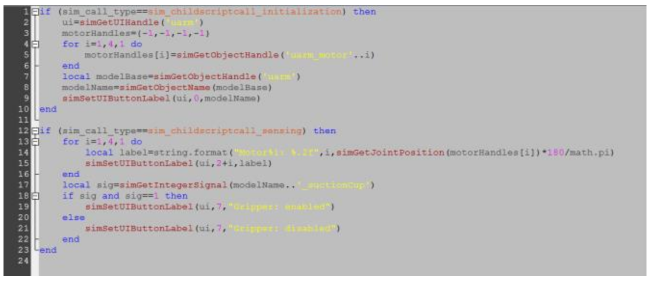
Initialization block. The contents of this block are executed only once when the simulation is started. This block contains such operations as declaring the necessary variables and assigning initial values to them. Also in this part of the script handlers for controlling the scene objects are set.

Activation block. The contents of this block are iteratively executed at regular intervals (simulation step). The part of the script written in this block will be repeated until the simulation is stopped or a critical error occurs (in which case the simulation will also be stopped). The basic control algorithm is described here.

Sensor control code block. The contents of this block are executed as many times as the activation block. However, the main V-REP script accesses this block only after the script from the activation block has finished executing. This block is designed to retrieve data from the sensors.

Simulation Completion Block. This block allows you to selectively erase the data acquired during the simulation. The script from this block starts executing once before the simulation is terminated or the script is erased. This block remains empty in most cases.

As a rule, no script is without the use of variables. In Lua, you can declare new variables at any time, and you can also specify their initial value when declaring them (e.g., Fig. 6, line 9). It is also not necessary to specify the type of the variable. Lua will determine it depending on the value that is assigned to the variable for the first time. Variables with a numeric value are treated as a floating point number. Variables with character values and variables of logical type (possible values: "true" or "false") are also allowed. There is also a function for destroying (releasing) variables. All variables in Lua are global by default, i.e. they can be accessed from any area.

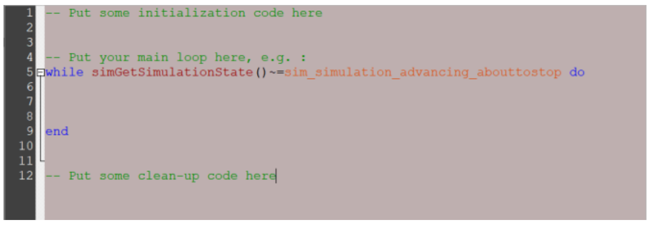


```
 1 if (sim_call_type==sim_childscriptcall_initialization) then
 2     ui=simGetUIHandle('uarm')
 3     motorHandles={-1,-1,-1,-1}
 4     for i=1,4,1 do
 5         motorHandles[i]=simGetObjectHandle('uarm_motor'..i)
 6     end
 7     local modelBase=simGetObjectHandle('uarm')
 8     modelName=simGetObjectName(modelBase)
 9     simSetUIButtonLabel(ui,0,modelName)
10 end
11
12 if (sim_call_type==sim_childscriptcall_sensing) then
13     for i=1,4,1 do
14         local label=string.format("Motor%i: %.2f",i,simGetJointPosition(motorHandles[i])*180/math.pi)
15         simSetUIButtonLabel(ui,2+i,label)
16     end
17     local sig=simGetIntegerSignal(modelName..'_suctionCup')
18     if sig and sig==1 then
19         simSetUIButtonLabel(ui,7,"Gripper: enabled")
20     else
21         simSetUIButtonLabel(ui,7,"Gripper: disabled")
22     end
23 end
24
```

Fig. 6. Example of an initialization script

Branching and loops are implemented in Lua as in most C-like programming languages in slightly different terms.

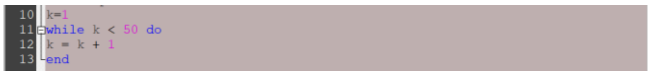
The "If" conditional operator requires the mandatory use of the "then/end" construct. A logical expression or variable must be specified after the "If" keyword. The logical expression is followed by the keyword "then", which begins the "body" of the condition - the part of the script that will be executed if the condition specified in the script is true. The conditional operator has an optional component – an additional condition "elseif" (elseif), the "body" of this condition – a part of the script that will be executed only if the condition specified in the script is false. Fig. 7 shows an example of using a conditional operator.



```
-- Put some initialization code here


-- Put your main loop here, e.g. :
while simGetSimulationState()~=sim_simulation_advancing_abouttostop do



end


-- Put some clean-up code here
```

Fig. 7. An example of using a conditional operator and functions for outputting information to the console window and status window

As with conditional statements, a control system is rarely without at least one loop. Loops in Lua are defined using keywords that denote the type of loop together with the keywords "do" and "end". The order is as follows: first the type is specified, then the loop execution condition, then the word "do", followed by the "loop body" – the script fragment that will be executed cyclically, and the loop ends with the word "end". The most common loops are "while" and "for" loops. Fig. 8 shows an example of the simplest while loop, which increases the value of the variable "k" by one until the condition "k<50" becomes false.

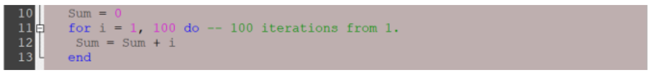


```
k=1
while k < 50 do
k = k + 1
end
```

Fig. 8. An example of using a "while" loop

Loops in Lua, like conditional branching, must end with the keyword "end". Undefined variables have a default value of "nil". In this case, only variables with the value "nil" and "false" (the logical variable type "false") return false in the loop condition, while the variable value "0" and ' ' return true.

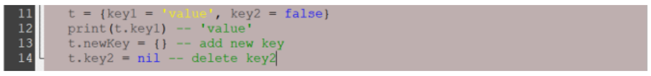
The second type of loops that is most often used is "for". This type is well suited for tasks where a counter loop is needed, but it should be mentioned that in most cases "for" can be replaced by a "while" loop with a few extra variables. An example of using the "for" loop is shown in Fig. 9, this script fragment executes 100 iterations starting from 1 (including 1 and 100). In this case, you can change the conditions, put 100 instead of 1 and 100 instead of 1, then the loop will be executed with changing the value of the variable "i" from 100 to 1. Also in the "for" loop there is an optional step parameter, by default the step is equal to one, and there is no need to specify it. However, for cases when it is necessary to use a step with a value other than 1, it is necessary to specify it with a comma after the final value (if a step 2 is required, then for the example in Fig. 9 it will be the condition "i = 1, 100, 2").



```
    Sum = 0
    for i = 1, 100 do -- 100 iterations from 1.
     Sum = Sum + i
    end
```

Fig. 9. An example of using a "for" loop

Tables in Lua are the only structural element, they combine the properties of an array, hash table ("key" – "value"), structure and object. Most often tables are used as dictionaries, and the key is of string (character) type by default. An example is shown in Fig. 10. Line 11 declares a variable of type "table" and sets 2 keys and their corresponding values. The values can be accessed by specifying the name of the table and the key with a dot (example in Fig. 10).

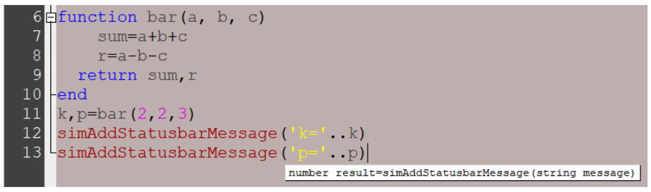


```
11    t = {key1 = 'value', key2 = false}
12    print(t.key1) -- 'value'
13    t.newKey = {} -- add new key
14    t.key2 = nil -- delete key2
```

Fig. 10. Example of using tables in Lua

As you can see from the example, it is allowed to use as a key not only strings, but also all other types of variables that are available in Lua.

Another and extremely important part of the programming language are functions. You can create your own functions, or you can use ready-made ones, which can be found in the Lua reference book [7]. An example of a function defined by the developer is shown in Fig. 11. As you can see from the example, the function can take several values. Also functions can return several values.

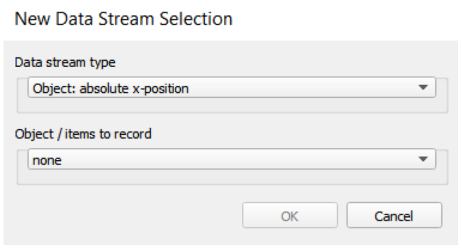


```
function bar(a, b, c)
     sum=a+b+c
     r=a-b-c
  return sum,r
end
k,p=bar(2,2,3)
simAddStatusbarMessage('k='..k)
simAddStatusbarMessage('p='..p)
                          number result=simAddStatusbarMessage(string message)
```

Fig. 11. An example of using a custom function in Lua

Regular functions are already integrated into Lua when writing V-REP scripts and begin with "sim". It is these functions that are used to interact with the simulation: control, data reading, debugging and many other tasks [8].

The V-REP program has a separate object class "Graph" (graph), which can be used to easily visualize both data from individual sensors and user data (regardless of the method of acquisition). To add graphs to the simulation scenario, just execute [Add→Graph] in the main menu. Next, it is necessary to set the properties. To do this, select an object in the scenario hierarchy and activate the "Object/Item Properties" tool. In the context menu under "Data stream recording list" you must click on "Add new data stream to record". In the context window that appears (Fig. 12) it is necessary to specify the data type and data source. If it is necessary to output the data from the sensor to the chart, it is necessary to specify the data type in the drop-down field "Data stream type" and the sensor name in the field "Object/Item to record". Often there is a need for preliminary processing of sensor data before displaying it on a chart: this can be done by using a script as an intermediate point where data processing will be performed. In this case, it is necessary to specify in the chart settings that the data will be displayed as user data.


New Data Stream Selection

Data stream type: Object: absolute x-position

Object / items to record: none

Fig. 12. Context window of data stream creation in V-REP

After clicking "OK" in the "Data stream recording list" section, a new data stream with the default name will appear. Double-click on the name of the data stream to enter the name editing mode. This name will be used to refer to the graph from the script when outputting data.

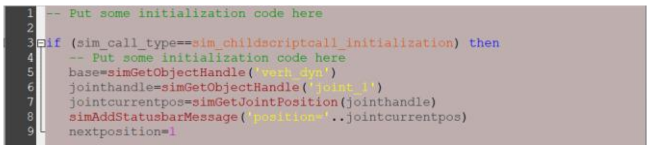

```
1  -- Put some initialization code here
2
3  if (sim_call_type==sim_childscriptcall_initialization) then
4      -- Put some initialization code here
5      base=simGetObjectHandle('verh_dyn')
6      jointhandle=simGetObjectHandle('joint_1')
7      jointcurrentpos=simGetJointPosition(jointhandle)
8      simAddStatusbarMessage('position='..jointcurrentpos)
9      nextposition=1
```

Fig. 13. Script fragments for outputting user data to the chart using the V-REP program

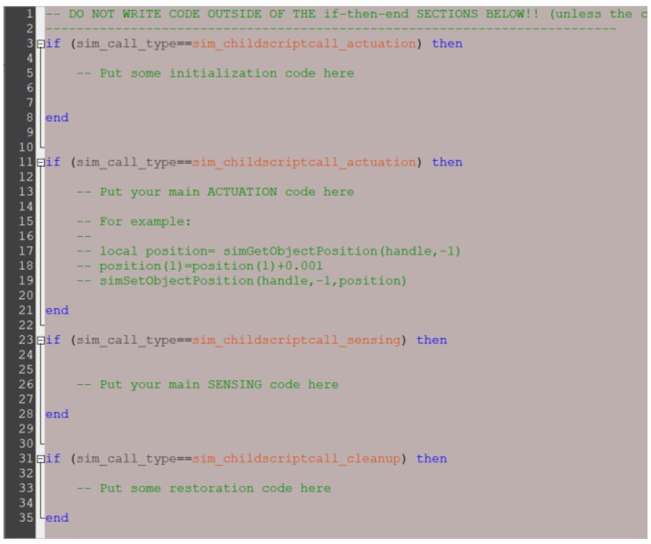

```
1  -- DO NOT WRITE CODE OUTSIDE OF THE if-then-end SECTIONS BELOW!! (unless the c
2  ------------------------------------------------------------------------------
3  if (sim_call_type==sim_childscriptcall_actuation) then
4
5       -- Put some initialization code here
6
7
8  end
9
10
11 if (sim_call_type==sim_childscriptcall_actuation) then
12
13      -- Put your main ACTUATION code here
14
15      -- For example:
16      --
17      -- local position= simGetObjectPosition(handle,-1)
18      -- position(1)=position(1)+0.001
19      -- simSetObjectPosition(handle,-1,position)
20
21 end
22
23 if (sim_call_type==sim_childscriptcall_sensing) then
24
25
26      -- Put your main SENSING code here
27
28 end
29
30
31 if (sim_call_type==sim_childscriptcall_cleanup) then
32
33      -- Put some restoration code here
34
35 end
```

Fig. 14. Script fragments for reading data from sensors in the V-REP program

Figure 13 shows an example of a script fragment that outputs user data from a script to a graph. In the example, on line 4 "Graph" is the name of the object, on line 7 "Red" is the name of the data stream, and "data" is the name of the variable whose value is displayed on the graph. To learn how to get data from a sensor, see "Sensors".

Sensors. Different types of sensors are available in V-REP program: force and moment sensors (force sensor), video sensors (vision sensor), distance sensors (proximity sensor). Reading data from sensors is performed in a script using special functions (API-functions). Examples of reading data from "vision sensor" and "proximity sensor" are shown in Fig. 14.

2. Conditions and methods of research

To model the operation of an industrial manipulator, we need to simulate the process of an industrial manipulator equipped with a sensor. The manipulator should be programmed to search for an object of a given color that lies within the manipulator's working area. After the object is detected, the manipulator should stop the search and bring the end link to grab the object.

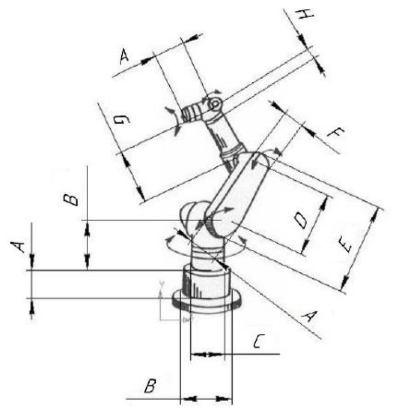


Fig. 15. Schematic representation of a manipulator

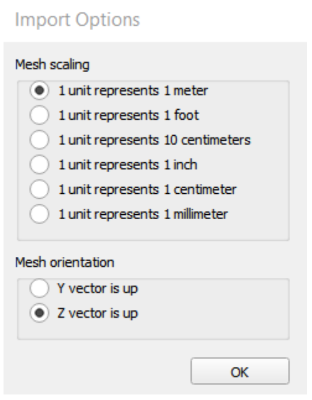
An industrial multi-link manipulator is a classic example of a dynamic system in mechatronics and robotics. The development of a scenario that allows to simulate the process of operation and the whole physics of interaction of the manipulator components allows the most complete study of the main aspects of modeling a large class of systems.

This paper reviews the creation of the simulation without delving into the mathematical aspects of modeling, allowing us to focus on the practical application aspects. The basic approaches for modeling robot dynamics, as well as the Newton-Euler method, which is used in V-REP to calculate the dynamics, are discussed in detail in textbooks [4] and [11]. It is also worth remembering that simulation does not give 100% accuracy, small deviations are allowed at different stages of execution.

This paper assumes that the student already has the skills to create three-dimensional models in any CAD system. If there are no such skills, it is recommended to refer to the textbook [8], or other similar literature. First, it is necessary to import the already created 3D model of the manipulator from CAD-system into V-REP. For this purpose it is necessary to save the manipulator assembly with *.STL extension. Each element of the assembly will be saved as a separate file. Files saved in STL format save their coordinates, and in the future it will be convenient to work with them in the V-REP program.

Start V-REP and in the main menu run [File → Import → Mesh...], select all saved STL files and click "Open".

In the window that appears, select the item "1 unit represents 1 millimeter", which will allow you to set the correct scale in the new coordinate system, you may also need to change the "Mesh orientation". After all the settings have been correctly set, it remains to click "OK" (Fig. 16).



Import Options

Mesh scaling

- [✓ 1 unit represents 1 meter]
- 1 unit represents 1 foot
- 1 unit represents 10 centimeters
- 1 unit represents 1 inch
- 1 unit represents 1 centimeter
- 1 unit represents 1 millimeter

Mesh orientation

- Y vector is up
- [✓ Z vector is up]

Fig. 16. File import settings

After importing, the main window should display all models (an example is shown in Fig. 17). You can also see that in the hierarchy of scene objects there are new components corresponding to each part of the industrial manipulator.

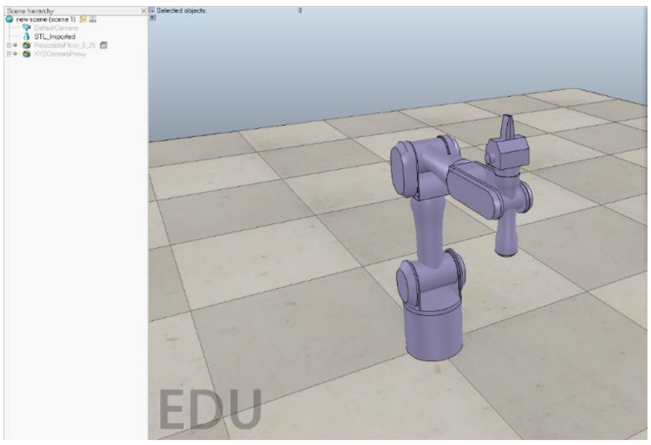


Fig. 17. Example of a manipulator imported from CAD system

To identify the types it is convenient to use the icons to the left of the component name. Now all components have the type "Composite random forms".

In the future, to make it more convenient to work with the scene hierarchy, we will give each part of the manipulator a name. The name should contain information about what kind of component it is (for example, proximity to the base or other information for easy identification).

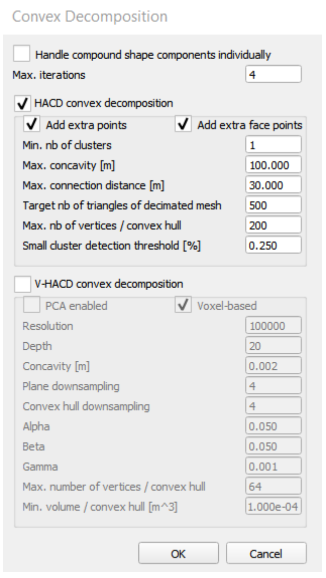
The elements available at this stage can be used for simulation, but this is not recommended because elements of this type are not optimal for calculating physical properties. Therefore, let's create modified copies of these elements that will be used for physics modeling.

Select each component, right-click on its name and choose [Add → Convex decomposition of selection...] in the pop-up menu.

By changing the “Target nb of triangles of decimated mesh" parameter, you can adjust the detail of the mesh of the created object. The recommended range of values is from 500 to 1000. Also in the context window of the tool there is a number of other parameters that also allow you to create an optimized mesh.

Having set the necessary parameters (Fig. 18), click "OK". This should be repeated for all components of the manipulator. Note that there are new elements, and the icons to the left of them differ from the icons of the elements we had.

Next, it is recommended to rename the new components so that they have the same names as the original components, only with the addition of the ending "_dyn" (remember that only Latin letters are allowed in the name).

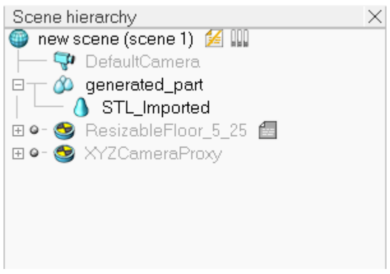


Convex Decomposition

☐ Handle compound shape components individually	
Max. iterations	4
✓ HACD convex decomposition	
✓ Add extra points	✓ Add extra face points
Min. nb of clusters	1
Max. concavity [m]	100.000
Max. connection distance [m]	30.000
Target nb of triangles of decimated mesh	500
Max. nb of vertices / convex hull	200
Small cluster detection threshold [%]	0.250
☐ V-HACD convex decomposition	
☐ PCA enabled	✓ Voxel-based
Resolution	100000
Depth	20
Concavity [m]	0.002
Plane downsampling	4
Convex hull downsampling	4
Alpha	0.050
Beta	0.050
Gamma	0.001
Max. number of vertices / convex hull	64
Min. volume / convex hull [m^3]	1.000e-04

Fig. 18. Window for creating a copy with an optimized mesh for dynamics simulations

After that, you need to link the source components to their copies ("_dyn") in such a way that the dynamic components are a "parent" to the source components. This can be done using the "drag and release" method by selecting the source component in the scene hierarchy (with the left mouse button held down) and dragging it onto its copy with dynamic properties. The same operation should be done for all components of the manipulator. The result should be similar to the one shown in Fig. 19.



Scene hierarchy

```
new scene (scene 1)
├─ DefaultCamera
⊟─ generated_part
│   └─ STL_Imported
⊞─ ResizableFloor_5_25
⊞─ XYZCameraProxy
```

Fig. 19. Example of the manipulator scene hierarchy

Now there are objects in the scene that overlap each other, and it is necessary to separate these objects into different display layers. To do this, select an element with an optimized grid and activate the "Properties" tool. In the "Common" tab find the item "Camera visibility layers" and set the values for the first row 0000 0000 and 1000 0000 for the second row of fields. The same operation should be done with other elements that have optimized grid.

As a result, we get visually the original manipulator, but with the help of the "Layers" tool you can switch to displaying components of another type. Fig. 20 shows the display of only elements with optimized grid.

Now it remains to configure the elements in such a way that the dynamic-optimized elements simulate dynamics, while the original components do not participate in the calculations, but only repeat the behavior of dynamic elements. So, open the "Properties" of the dynamic element and click "Show dynamic properties dialog". Next, we need to enable the "Body is Respondable" and "Body is dynamic" functions. In the dynamic parameters click on "Compute mass and inertia properties for selected..." to automatically calculate mass and moments of inertia based on geometry (Fig. 21), after that it is necessary to enter material density and V-REP will calculate all dynamic properties.

The above operation should be repeated for all parts of the manipulator except the base. The manipulator base is fixed and should not be considered as a dynamic element, but the "Respondable" property should be enabled.

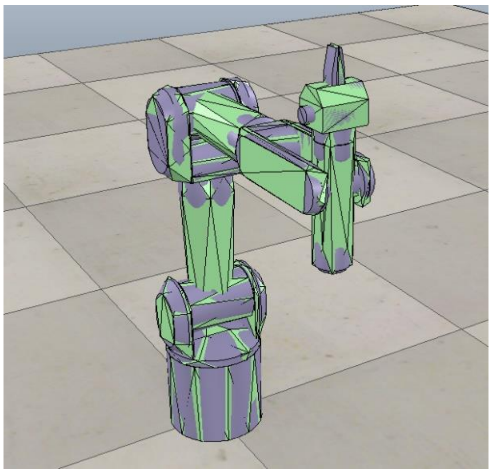
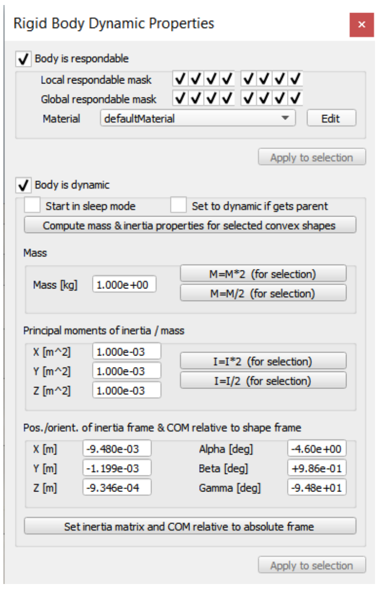


Fig. 20. Example of a manipulator



```
Rigid Body Dynamic Properties                              ×
[✓] Body is respondable
    Local respondable mask    ✓ ✓ ✓ ✓  ✓ ✓ ✓ ✓
    Global respondable mask   ✓ ✓ ✓ ✓  ✓ ✓ ✓ ✓
    Material   defaultMaterial                       Edit
                                          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]   1.000e-03
    Y [m^2]   1.000e-03            I=I*2 (for selection)
    Z [m^2]   1.000e-03            I=I/2 (for selection)
  Pos./orient. of inertia frame & COM relative to shape frame
    X [m]   -9.480e-03      Alpha [deg]   -4.60e+00
    Y [m]   -1.199e-03      Beta [deg]    +9.86e-01
    Z [m]   -9.346e-04      Gamma [deg]   -9.48e+01
    Set inertia matrix and COM relative to absolute frame
                                          Apply to selection
```

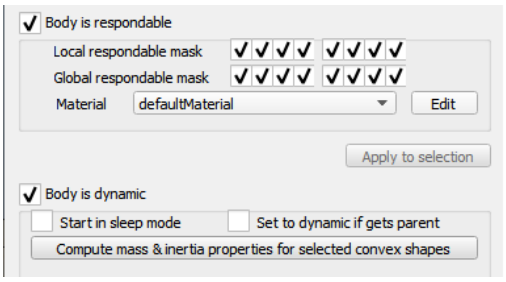
Fig. 21. Window for setting dynamic properties of objects

The V-REP program has a function for setting the interaction mask. The mask can be used to tell the program which elements should be ignored. It is often the case that two solids are connected by a hinge, and in a real situation they would not act on each other. But in the V-REP program, when modeling, we can get a coarse mesh, which is the basis for the calculation. This causes the meshes of two adjacent elements to touch each other, and to solve this problem we would have to recreate a finer mesh, which would increase computer resources. Using masks, you can easily solve such problems.

Let's use object masks to exclude the interaction of neighboring elements of the manipulator from the calculation. To do this, open the properties of the main element (manipulator base, fixed link) and set the mask as shown in Fig. 22.

Further, each subsequent element of the tree structure must have a local mask different from the mask of the previous element. For example, the "Local respondable mask" of the element following the base element will be 11110000, and the next element will have the same mask as the base ("00001111"), and so on with alternation.

If you try to run the resulting scenario, all parts of the manipulator should fall apart, because there are no joints.



```
[✓] Body is respondable
    Local respondable mask    ✓ ✓ ✓ ✓  ✓ ✓ ✓ ✓
    Global respondable mask   ✓ ✓ ✓ ✓  ✓ ✓ ✓ ✓
    Material   defaultMaterial                       Edit
                                          Apply to selection
[✓] Body is dynamic
    [ ] Start in sleep mode     [ ] Set to dynamic if gets parent
    Compute mass & inertia properties for selected convex shapes
```

Fig. 22. Setting up a mask

To connect the parts of the manipulator components you need to add joints ("Joint") of rotational type ("Revolute") to the scene. This can be done through the main menu [Add → Joint → Revolute], after which a new object is added to the scene. The joint represents a certain axis, relative to which the movements of the elements connected to the joint will be performed. Proceeding from this it is necessary to precisely set the position and orientation of the joints at the points of relative rotation of the manipulator links. This task is one of the most difficult, and in this case it is possible to resort to some tricks. One of the possible approaches to this problem is described below.

To position the hinges, we will use the coordinates of the already existing elements of the manipulator. Press "Ctrl" to select the hinge and the manipulator base, then go to the "Position" tab of the "Object/Item Shift" tool and press "Apply to selection". Switch to the "Object/Item Rotate" tool and click "Apply to selection" in the "Orientation" tab. But it happens that the hinge must be placed not in the center of the pattern by all three coordinates, but in the middle, in a non-standard place in a groove or on a ledge. In such a situation there are two ways out: use relative displacement or relative rotation. To do this, select only the hinge (two items can be selected at once after the previous operation) and activate the "Object/Item Shift" tool. In the "Translation" tab, select "Own frame" (coordinate system associated with the current object) and enter the distance from the current position to which you want to move the selected object in the corresponding axes, then click "Translate selection". Similarly, you can rotate the hinge in the "Rotation" tab of the "Object/Item rotate" tool.

The above method solves the hinge positioning problem, but in most cases it can be solved even faster if you take advantage of the ability to separate elements with a mesh. Before you do this, it is better to use an additional "draft" script. Create another script and copy all the elements of the scene there. Next, we should perform positioning in the draft scene and then copy the already correctly positioned joint into the working scenario. When copying objects from one scenario to another, their properties are preserved. Copying and pasting are performed using the keyboard shortcuts "Ctrl + C" and "Ctlr + V" respectively.

In the draft script, select an element of the "compound convex shape" type and in the right-click context menu choose "Edit"→"Grouping/Merging"→"Ungroup selected shapes". Thus, we can divide the initial object into several very simple shapes, and now we can use coordinates of any of the obtained shapes for positioning the hinge. Using the already known method, we set the necessary position and orientation, and then copy the hinge from the draft to the working script.

Results. As a result of manipulations we managed to set the exact location and orientation of the hinge. Having made similar actions with other joints, it remains to set readable names, so that it would be convenient to work with them in the future.

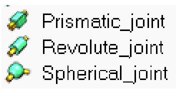
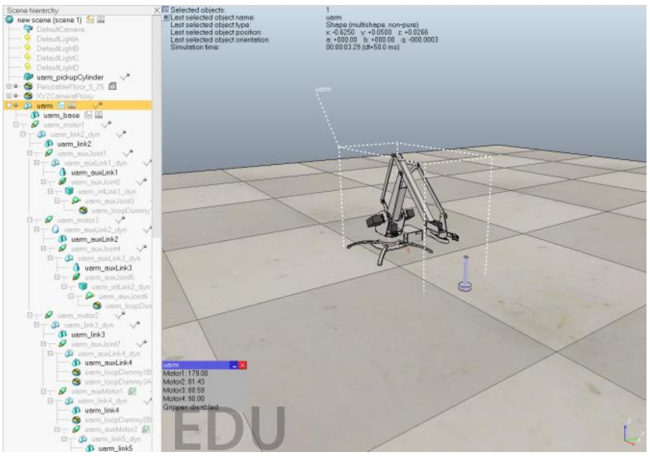


Fig. 23. Pictograms of different hinge types

Now that all the hinges are in place, we can set the links between the elements. To do this, we create a tree structure in the scene hierarchy, exposing some objects as children relative to others. The hierarchy starts with the dynamic object of the manipulator base, which is the first parent, followed by the hinge, followed by the dynamic object closest to the base, then the hinge, and so on.

Fig. 24 shows an example of the result to be obtained after performing all the above actions.



Selected objects: 1
Last selected object name: uarm
Last selected object type: Shape (multishape, non-pure)
Last selected object position: x: -0.6250 y: +0.0500 z: +0.0266
Last selected object orientation: a: +000.00 b: +000.00 g: -000.0003
Simulation time: 00:00:03.29 (dt=50.0 ms)

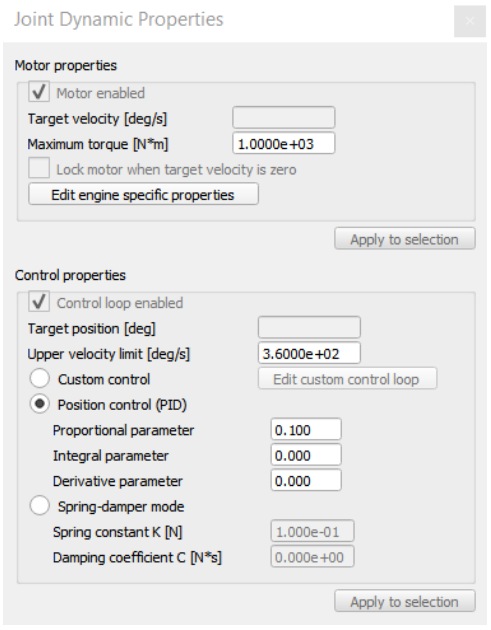
uarm
Motor1: 179.00
Motor2: 81.43
Motor3: 80.59
Motor4: 90.00
Gripper: disabled

Fig. 24. Hierarchy of objects and location of manipulator joints

For a more convenient display of the manipulator, you can also remove the display of the visual properties of the joints.

Let's proceed to customizing the joints. Select the joint in the scene hierarchy and execute the commands [Scene object properties→Show dynamic properties dialog]. In the context window "Joint Dynamic Properties" we will mark three important items (Fig. 25).

- motor enabled (motor enabled/disabled),
- lock motor when target velocity is zero,
- control loop enabled (enable/disable cyclic control).


Joint Dynamic Properties

Motor properties
- [✓] Motor enabled
- Target velocity [deg/s]:
- Maximum torque [N*m]: 1.0000e+03
- [ ] Lock motor when target velocity is zero
- Edit engine specific properties
- Apply to selection

Control properties
- [✓] Control loop enabled
- Target position [deg]:
- Upper velocity limit [deg/s]: 3.6000e+02
- ○ Custom control — Edit custom control loop
- ● Position control (PID)
- Proportional parameter: 0.100
- Integral parameter: 0.000
- Derivative parameter: 0.000
- ○ Spring-damper mode
- Spring constant K [N]: 1.000e-01
- Damping coefficient C [N*s]: 0.000e+00
- Apply to selection

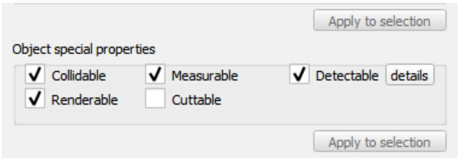
Fig. 25. Editing dynamic properties of a joint

Check the "Motor enabled" and "Control loop enabled" checkboxes in all joints of the manipulator. The next step is to install sensors.

Add a sensor via [Add→Proximity sensor→Ray type] commands, then select this object and set the required position and orientation (to the final link of the manipulator). Set the binding to the final link of the manipulator in the scene hierarchy. This sensor will determine the distance to the object. Let's proceed to its configuration. Select the sensor and go to [Scene object properties→Show volume parameters]. The "Offset" parameter is responsible for the distance from which the counting starts, and the "Range" parameter is responsible for the range of the sensor.

Add one more sensor, which will allow to determine the color of the object. Execute the commands [Add→Vision sensor→Orthographic type], set the necessary position and orientation, then set the binding to the manipulator's end link. We will not customize it in this work, as the default settings meet all tasks.

Now let's add the cube that the manipulator will need to search for. In the main menu choose [Add→Primitive shape→Cuboid], select the necessary dimensions of the cube and press "OK". Set the color in the window [Scene object properties→Adjust color→ Ambient/diffuse component] and set its location within the working area of the manipulator. To make the cube visible for sensors, you should change its properties through the tool [Scene object properties→Common]. It is necessary to set the properties "Collidable", "Measurable", "Detectable" and "Renderable" active, as shown in Fig. 26.


Apply to selection

Object special properties
- [✓] Collidable
- [✓] Measurable
- [✓] Detectable — details
- [✓] Renderable
- [ ] Cuttable

Apply to selection

Fig. 26. Special properties of objects

For convenient display of the information received from the sensor, let's add a dependency graph, which will be displayed during modeling. Create the graph through the main menu [Add → Graph], go to its properties and select "Add new data stream to record". In "Data stream type" select "Various: user-defined", and in "Object/items to record" select "User data" and click "OK".

Let's proceed to the program part. Having selected the base of the robot in the scene hierarchy, select [Add → Associated child script → Non threaded] in the main menu. The script icon will appear opposite the name, double-clicking on it opens the script editing mode.

The following is a manipulator control script that was implemented by one of the students. The script is far from optimal both in terms of programming and implementation of the control algorithm. The presented script is also universal, since many parameters depend on the dimensions and features of the manipulator. The algorithm searches for and attempts to bring the end link to a cube of red color. Cubes of other colors will be ignored by the manipulator.

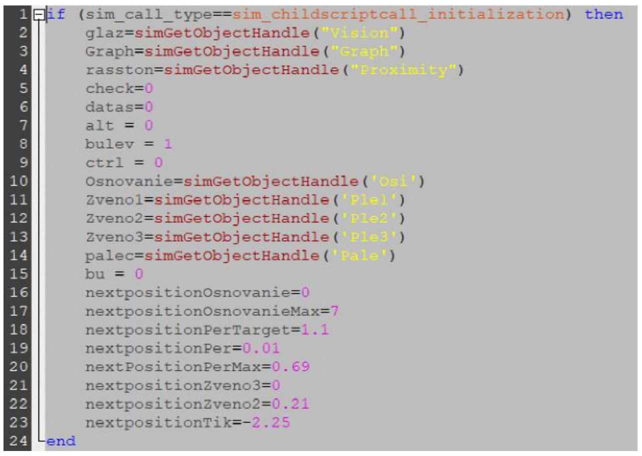
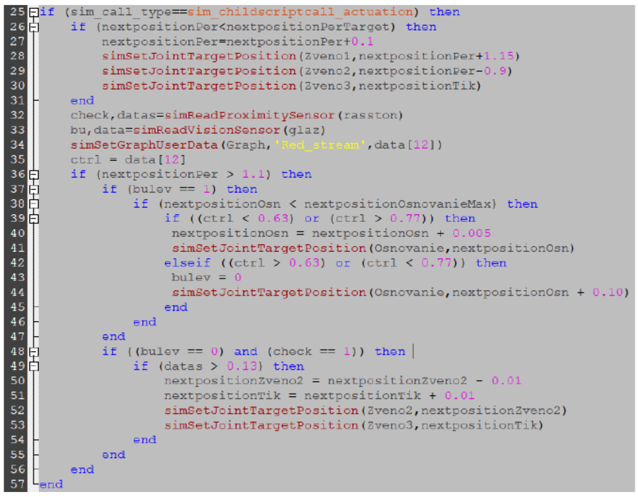

```
if (sim_call_type==sim_childscriptcall_initialization) then
    glaz=simGetObjectHandle("Vision")
    Graph=simGetObjectHandle("Graph")
    rasston=simGetObjectHandle("Proximity")
    check=0
    datas=0
    alt = 0
    bulev = 1
    ctrl = 0
    Osnovanie=simGetObjectHandle('Osi')
    Zveno1=simGetObjectHandle('Ple1')
    Zveno2=simGetObjectHandle('Ple2')
    Zveno3=simGetObjectHandle('Ple3')
    palec=simGetObjectHandle('Pale')
    bu = 0
    nextpositionOsnovanie=0
    nextpositionOsnovanieMax=7
    nextpositionPerTarget=1.1
    nextpositionPer=0.01
    nextPositionPerMax=0.69
    nextpositionZveno3=0
    nextpositionZveno2=0.21
    nextpositionTik=-2.25
end
```

Fig. 27. Initialization block of the final script



```
if (sim_call_type==sim_childscriptcall_actuation) then
    if (nextpositionPer<nextpositionPerTarget) then
        nextpositionPer=nextpositionPer+0.1
        simSetJointTargetPosition(Zveno1,nextpositionPer+1.15)
        simSetJointTargetPosition(Zveno2,nextpositionPer-0.9)
        simSetJointTargetPosition(Zveno3,nextpositionTik)
    end
    check,datas=simReadProximitySensor(rasston)
    bu,data=simReadVisionSensor(glaz)
    simSetGraphUserData(Graph,'Red_stream',data[12])
    ctrl = data[12]
    if (nextpositionPer > 1.1) then
        if (bulev == 1) then
            if (nextpositionOsn < nextpositionOsnovanieMax) then
                if ((ctrl < 0.63) or (ctrl > 0.77)) then
                 nextpositionOsn = nextpositionOsn + 0.005
                 simSetJointTargetPosition(Osnovanie,nextpositionOsn)
                elseif ((ctrl > 0.63) or (ctrl < 0.77)) then
                 bulev = 0
                 simSetJointTargetPosition(Osnovanie,nextpositionOsn + 0.10)
                end
            end
        end
        if ((bulev == 0) and (check == 1)) then
            if (datas > 0.13) then
                nextpositionZveno2 = nextpositionZveno2 - 0.01
                nextpositionTik = nextpositionTik + 0.01
                simSetJointTargetPosition(Zveno2,nextpositionZveno2)
                simSetJointTargetPosition(Zveno3,nextpositionTik)
            end
        end
    end
end
```

Fig. 28. Control code block

Running the simulation.

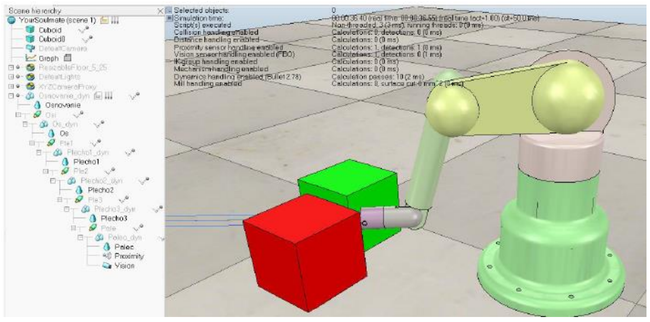
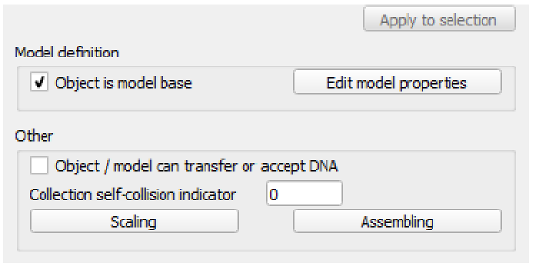


Fig. 29. Result of modeling the manipulator

In order to save the manipulator as a whole model, it is enough to additionally set "Object is model base" in the properties of the base object (manipulator base) (Fig. 30).



Apply to selection

Model definition

[✓] Object is model base   Edit model properties

Other

[ ] Object / model can transfer or accept DNA

Collection self-collision indicator: 0

Scaling   Assembling

Fig. 30. Manipulator base properties window, "Model definition" section

3. Conclusion

Modeling plays an important role in the development of robotics. It allows us to create virtual robot models and test them in different scenarios without the need for physical implementation. It saves time and resources and also allows us to study and optimize the behavior of robots before their physical implementation.

The V-REP program uses the main approaches for modeling the dynamics of robots, as well as the Newton-Euler method, which is used to calculate the dynamics. The development process is non-linear, and a good computer simulation environment avoids many problems at an early stage, allowing to test both individual robot units and simplified models implementing individual functions.

In this paper, the basics of modeling robotic systems in V-REP (Virtual Robotics Experimentation Platform) software, developed by Coppelia Robotics and distributed freely under a free license for educational purposes, were discussed in detail. The basic approaches for creating simulations of the main types of robotic systems are considered.

Reference

[1] ABB Robotic Studio software documentation [http://developercenter.robotstudio.com/].
[2] Documentation for Lua developers. [http://www.lua.ru/doc/].
[3] Documentation of the open library of motion planning. [http://ompl.kavrakilab.org].
[4] Kolyubin S. A.: Dynamics of robotic systems. ITMO University, 2017.
[5] Kolyubin S.: Modelling Metallic Shell. Robot modelling tools. Control Engineering Russia 4(52), 2014.
[6] List of V-REP regular functions with detailed description. [http://www.coppeliarobotics.com/helpFiles/en/apiFunctionListCategory.htm]
[7] Monitoring the development of educational robotics and IT-education in the city of Moscow. Publishing Center ANO AIR Moscow 2017.
[8] Sorokin S. V., Soldatenko I. S.: Fundamentals of development and programming of robotic systems. Tver 2017.
[9] The official website of the Gazebo developers. [http://gazebosim.org].
[10] The official website of the V-REP developers. [http://coppeliarobotics.com/].
[11] Zhurbenko P. A., Guznenkov V. N., Bondareva T. P.: SolidWorks 2016. Three-dimensional modeling of parts and execution of electronic drawings. Tutorial.

Ph.D. Laura Yesmakhanova
e-mail: laura060780@mail.ru

Yesmakhanova Laura – graduate of the doctoral program of the Lublin University of Technology. Currently, she works as an associate professor at Taraz Regional University at the Department of Automation and Telecommunications. She is the author and co-author of several scientific publications in the field of the use of optoelectronic components in automated control systems and fiber-optic communication lines.

https://orcid.org/0000-0002-3308-9676