Virtual Robot Experimentation Platform

V-REP

www.coppeliarobotics.com
V-REP Overview

What is it? General purpose robot simulator with integrated development environment

What can it do? Sensors, mechanisms, robots and whole systems can be modelled and simulated in various ways >> Play overview video

Typical applications:
• Fast prototyping and verification
• Fast algorithm development
• Robotics related education
• Remote monitoring
• Hardware control
• Simulation of factory automation systems
• Safety monitoring
• Product presentation
• etc.
3 Central Elements

- Scene Objects
- Calculation Modules
- Control Mechanisms
Scene Objects

- Basic building blocks
- 14 different types
- Can be combined with each other
- Can form complex systems together with calculation modules and control mechanisms
Proximity Sensors & Graphs

Proximity Sensors

• More than simple ray-type detection
• Configurable detection volume
• Fast minimum distance calculation within volume
• Much more realistic simulation than with ray-type sensors

Graphs

• Time graphs
• X/Y graphs
• 3D curves
• Can be exported

>> Play demo video1   >> Play demo video2
Vision Sensors

• Integrated image processing
• Extendable via plugin mechanism
• Ray-traced rendering also available

>> Play demo video
Paths and Mills

Paths

• 6 dim. trajectory definition

• Path can be shaped
  (i.e. automatically generate extruded meshes)

Mills

• Customizable cutting volume

• Cuts shapes (i.e. meshes)

>> Play demo video
Cameras

- Perspective / orthographic projection
- Tracking & automatic view-fitting function

Lights

Spotlight / directional / omnidirectional

Mirrors

Mirror or scene / object clipping function

>> Play demo video
Joints, Shapes, Force/Torque Sensors, and Dummies

**Shapes**
- Random mesh, convex mesh, primitive mesh or heightfield mesh
- Can be grouped/ungrouped
- Optimized for fast calculations

**Force/Torque Sensors**
- Measures force and torque
- Can conditionally break apart

**Joints**
- Revolute-type
- Prismatic-type
- Screw-type
- Spherical-type

**Dummies**
Auxiliary reference frame & helper object
Octrees and Point Clouds

Octrees

- Spatial partitioning, made up by a tree data structure for fast data access
- Voxel-based, can be modified during simulation
- Can be used as a simplified representation of meshes, as an occupancy grid/space, etc.
- Can be used for fast collision detection, minimum distance calculation, proximity sensor detection

Point Clouds

- Point container
- Octree-based, for fast data access
- Can be used for fast collision detection, minimum distance calculation, proximity sensor detection

>> Play demo video1  >> Play demo video2  >> Play demo video3
3 Central Elements

- **Calculation modules**
  - 5 basic algorithms
  - Can be combined with each other
  - Can form complex systems together with scene objects and control mechanisms
Calculation Modules

Collision detection

Minimum distance calculation

Forward / Inverse kinematics

Physics / Dynamics

Path / motion planning
Kinematics and Distance Calculation

Inverse / forward Kinematics

• Any mechanism: redundant, branched, closed, etc.
• Damped / undamped resolution
• Weighted resolution
• Conditional resolution
• Obstacle avoidance

Minimum Distance Calculation

• Any mesh (also open / concave / polygon soups)
• Any octree
• Any point cloud
• Any individual point

>> Play demo video1 >> Play demo video2
Dynamics

Dynamics / Physics

• 4 physics engines: Bullet Physics
  Open Dynamics Engine
  Vortex Dynamics
  Newton Dynamics

• Simple mouse click to switch

• Dynamic particles to simulate air or water jets

• Can work hand-in-hand with kinematics module

>> Play demo video
Collision Detection

• Any mesh (also open / concave / polygon soups)
• Any octree
• Any point cloud
• Any individual point

Path / Motion Planning

Supported via an OMPL plugin for V-REP

>> Play demo video1

>> Play demo video2
3 Central Elements

- Scene Objects
- Calculation Modules

Control Mechanisms

- 6 methods or interfaces
- >7 languages
- 6 methods can be used at the same time, and even work hand-in-hand
Control Mechanisms

- Embedded scripts
- Plugins
- Add-ons
- Remote API clients
- Custom solutions
- ROS nodes
Local and Remote Interfaces

- **Local (i.e. same process)**
  - Sync.* or async.* operation

- **Remote (i.e. different process / hardware)**
  - Async.* operation

**Control Mechanisms**

- Embedded scripts
- Plugins
- Add-ons
- Remote API clients
- Custom solutions
- ROS nodes

*Synchronous to the simulation loop*
Local Interfaces

Plugins
- > 500 API functions. Extendable
- C/C++ interface
- Can customize the simulator
- Can register new embedded script commands

Embedded Scripts
- > 500 API functions. Extendable
- Can be attached to any scene object
- Many Lua extension libraries available
- Threaded or non-threaded. Threads can be synchronized easily
- Various types: main script, child scripts, callback scripts (e.g. custom joint controllers)

Add-ons
- > 400 API functions. Extendable
- Lua interface
- Can customize the simulator
- Lightweight and easy to set-up
- Many Lua extension libraries available
Remote Interfaces

Remote API
- > 100 API functions. Extendable
- C/C++, Python, Java, Matlab, Octave, Lua & Urbi interfaces
- Data streaming and partitioning modes
- Lightweight and easy to use

ROS Interfaces
- Plugin-based
- Supports all standard messages, extendable
- Naturally duplicates the ROS C++ API
Embedded Script Advantages 1/6

Controller Integration

Plugins

Simulation model
(meshes, joints, sensors, etc.)

+ Plugin
(controller)

2 items

Embedded scripts

Simulation model
(meshes, joints, sensors, etc.)

+ controller

1 item
Embedded Script Advantages 2/6

Scalability

Plugins

- Simulation model 1
- Simulation model 2
- Simulation model 3

Embedded scripts

- Simulation model 1
- Simulation model 2
- Simulation model 3

Plugin has to manage instances

Scalability is inherent
Embedded Script Advantages 3/6

Version Conflicts

Plugins

Simulation model 1 (version 1)

Simulation model 2 (version 1)

Simulation model 1 (version 2)

Plugin (version 1)

Embedded scripts

Simulation model 1 (version 1)

Simulation model 2 (version 1)

Simulation model 1 (version 2)

Plugin (version 2)

High chances for conflicts

No chances for conflicts
<table>
<thead>
<tr>
<th></th>
<th>Plugins</th>
<th>Embedded scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same OS</td>
<td>Simulation model</td>
<td>Simulation model</td>
</tr>
<tr>
<td></td>
<td>Plugin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 files</td>
<td>1 file</td>
</tr>
<tr>
<td>Different OS</td>
<td>Simulation model</td>
<td>Simulation model</td>
</tr>
<tr>
<td></td>
<td>Plugin source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Many files</td>
<td>1 file</td>
</tr>
<tr>
<td></td>
<td>Compilation required</td>
<td>No compilation required</td>
</tr>
<tr>
<td></td>
<td>OS-specific problems</td>
<td>No OS-specific problems</td>
</tr>
</tbody>
</table>

**Portability**
### Other Considerations

<table>
<thead>
<tr>
<th>Plugins</th>
<th>Embedded scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Creation, compilation and installation difficulty:</strong></td>
<td><strong>Creation, compilation and installation difficulty:</strong></td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Model modification difficulty:</strong></td>
<td><strong>Model modification difficulty:</strong></td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Maintenance over the years:</strong></td>
<td><strong>Maintenance over the years:</strong></td>
</tr>
<tr>
<td>OS-dependent</td>
<td>OS-independent</td>
</tr>
<tr>
<td>Compiler-dependent</td>
<td>Compiler-independent</td>
</tr>
<tr>
<td>Framework-dependant</td>
<td>Framework-independant</td>
</tr>
</tbody>
</table>
## Synchronization with Simulation Loop

<table>
<thead>
<tr>
<th>Plugins</th>
<th>Embedded scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-threaded</td>
<td>Non-threaded</td>
</tr>
<tr>
<td>Control routine called at each simulation pass</td>
<td>Control routine called at each simulation pass</td>
</tr>
<tr>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Threaded</td>
<td>Threaded</td>
</tr>
<tr>
<td>Complex synchronization mechanism required</td>
<td>Control routine thread can behave as a coroutine</td>
</tr>
<tr>
<td>Difficult</td>
<td>Easy</td>
</tr>
</tbody>
</table>

- Control routine thread can be considered as a coroutine.
- Examples:
  - `simSwitchThread()`
  - `simSetThreadSwitchTiming(delay)`
  - `simSetThreadIsFree(isFree)`
  - `simSetThreadResumeLocation(location, order)`
Embedded Scripts – Simple Example

Hexapod simulation model

Hokuyo simulation model

Distance sampling rays

Detected points

Embedded script

Proximity sensor

Hokuyo model hierarchical representation

Joint

Shape

Hokuyo_URG_04LX_UG01

Hokuyo_URG_04LX_UG01_joint

Hokuyo_URG_04LX_UG01_laser

>> Play demo video
Remote API Advantages 1/2

Runs on any hardware, lightweight, several languages

Matlab/Octave

C/C++

Lua

Java

Python

V-REP
Remote API Advantages 2/2

Easy to use, almost like a regular API

```c
int returnCode = simxGetJointPosition(jointHandle, &position, operationMode);
```

- simx_return_ok
- simx_return_timeout_flag
- simx_return_novalue_flag
- simx_return_remote_error_flag
- simx_return_local_error_flag
- etc.

Remote API function

Regular arguments

Blocking mode

- simx_opmode_oneshot
- simx_opmode_blocking
- simx_opmode_streaming
- simx_opmode_discontinue
- simx_opmode_buffer
- etc.
Collaborative Control Mechanisms

- Embedded scripts
- Remote API clients
- Custom solutions
- ROS nodes
- Add-ons
- Plugins
```c
float coords[3]={0.1f,0.2f,0.3f};
int retIntCnt;
int* retInts;
simxCallScriptFunction(...,"createDummy_function",...,3,coords,1,"MyDummyName",...,&retIntCnt,&retInts,...,simx_opmode_blocking);
printf("Dummy handle: %i\n",retInts[0]);
```

```
createDummy_function=function(inInts,inFloats,inStrings,inBuffer)
  -- Create a dummy object with specific name and coordinates
  if #inStrings>=1 and #inFloats>=3 then
    local dummyHandle=simCreateDummy(0.05)
    local position={inInts[2],inInts[3],inInts[4]}
    simSetObjectName(dummyHandle,inStrings[1])
    simSetObjectPosition(dummyHandle,-1,inFloats)
  return {dummyHandle},{},{},''
  -- return the handle of the dummy
end
end
```
void SCRIPT_DO_SOME_MAGIC_CALLBACK(SScriptCallBack* callbackStruct)
{
  // gets called when a script calls "simxExt_doSomeMagic"
  int inputOutputArgumentStack=callbackStruct->stackID;
  ...
}

// Initialization phase of plugin: register new script commands:
simRegisterScriptCallbackFunction(..., SCRIPT_DO_SOME_MAGIC_CALLBACK);

returnData1, returnData2=simExt_doSomeMagic(arg1, arg2)
Example of Collaborative Mechanism 3 / 3

-- Left motor speed subscriber callback
function LVel_cb(msg)
    simSetJointTargetVelocity(leftMotor,msg.data)
end

-- Right motor speed subscriber callback
function RVel_cb(msg)
    simSetJointTargetVelocity(rightMotor,msg.data)
end

-- Initialization
if (sim_call_type==sim_childscriptcall_initialization) then
    ...
    pub=simExtRosInterface_advertise('/sensorData','std_msgs/Bool')
    subL=simExtRosInterface_subscribe('/leftVel','std_msgs/Float32','LVel_cb')
    subR=simExtRosInterface_subscribe('/rightVel','std_msgs/Float32','RVel_cb')
end

-- Actuation phase, once per simulation step
if (sim_call_type==sim_childscriptcall_actuation) then
    local result=simReadProximitySensor(noseSensor)
    local detectionTrigger={}
    detectionTrigger['data']=result>0
    simExtRosInterface_publish(pub,detectionTrigger)
    simExtRosInterface_sendTransform(...)
end
## Control Mechanisms – Feature Overview

<table>
<thead>
<tr>
<th>Control mechanism</th>
<th>Embedded script</th>
<th>Add-on</th>
<th>Plugin</th>
<th>Remote API client</th>
<th>ROS node</th>
<th>Custom client/server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control entity is external (i.e., can be located on a robot, different machine, etc.)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Difficulty to implement</td>
<td>Easiest</td>
<td>Easiest</td>
<td>Relatively easy</td>
<td>Easy</td>
<td>Relatively difficult</td>
<td>Relatively difficult</td>
</tr>
<tr>
<td>Supported programming language</td>
<td>Lua</td>
<td>Lua</td>
<td>C/C++</td>
<td>c/c++, python, Java, Matlab, Octave, Lua, Urb</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td>Simulator functionality access (available API functions)</td>
<td>500+ functions, extendable</td>
<td>500+ functions, extendable</td>
<td>500+ functions, extendable</td>
<td>&gt;100 functions, extendable</td>
<td>Depends on the selected ROS interface</td>
<td>Custom implementation</td>
</tr>
<tr>
<td>The control entity can control the simulation and simulation objects (models, robots, etc.)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>The control entity can start, stop, pause and step a simulation</td>
<td>Start, stop, pause</td>
<td>Start, stop, pause</td>
<td>Start, stop, pause, step</td>
<td>Start, stop, pause, step</td>
<td>Start, stop, pause, step</td>
<td>Start, stop, pause, step</td>
</tr>
<tr>
<td>The control entity can customize the simulator</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Code execution speed</td>
<td>Relatively slow (fast with JIT compiler)</td>
<td>Relatively slow (fast with JIT compiler)</td>
<td>Fast</td>
<td>Depends on programming language</td>
<td>Depends on programming language</td>
<td>Depends on programming language</td>
</tr>
<tr>
<td>Communication lag</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Yes, reduced</td>
<td>Yes, reduced</td>
<td>Yes, can be reduced</td>
</tr>
<tr>
<td>Control entity is fully contained in a scene or model, and is highly portable</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>API mechanism</td>
<td>Regular API</td>
<td>Regular API</td>
<td>Regular API</td>
<td>Remote API</td>
<td>ROS</td>
<td>Custom communication + regular API</td>
</tr>
<tr>
<td>API can be extended</td>
<td>Yes, with custom Lua functions</td>
<td>Yes, with custom Lua functions</td>
<td>Yes, V-REP is open source</td>
<td>Yes, Remote API is open source</td>
<td>Yes, ROS plugin is open source</td>
<td>N/A</td>
</tr>
<tr>
<td>Control entity relies on</td>
<td>V-REP</td>
<td>V-REP</td>
<td>V-REP</td>
<td>Sockets + Remote API plugin</td>
<td>Sockets + ROS plugin + ROS framework</td>
<td>Custom communication + script/plugin</td>
</tr>
<tr>
<td>Synchronous operation</td>
<td>Yes, inherent, no delays</td>
<td>Yes, inherent, no delays</td>
<td>Yes, inherent, no delays</td>
<td>Yes, Slower due to comm. Lag</td>
<td>Yes, Slower due to comm. Lag</td>
<td>Yes, Slower due to comm. Lag</td>
</tr>
<tr>
<td>Asynchronous operation</td>
<td>Yes, via threaded scripts</td>
<td>No</td>
<td>No threads available, but API access tenable</td>
<td>Yes, default operation mode</td>
<td>Yes, default operation mode</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. Depends on what ROS currently supports
2. The execution of API functions is however very fast. Additionally, there is an optional JIT (Just in Time) compiler option that can be activated
3. Lag reduced via streaming and data partitioning modes
4. Synchronous in the sense that each simulation pass runs synchronously with the control entity, i.e., simulation step by step
Architecture Overview

- **V-REP framework**
  - **V-REP, shared library (open source)**
  - **V-REP engine**
    - **Main script (customizable)**
    - **Child scripts (custom)**
    - **Callback scripts (custom)**
  - **C/C++ API to V-REP**
    - **C/C++ API calls**
    - **Add-ons (custom)**
  - **Lua API to V-REP**
    - **Lua API calls**
  - **Plugins (custom)**
    - **Remote API plugin (customizable)**
    - **ROS interfaces (customizable)**
  - **Custom clients/servers (robots, etc.) (custom)**
  - **Remote API clients (robots, etc.) (custom)**
  - **ROS nodes (robots, etc.) (custom)**

- **Main client application (i.e., vrep.exe)**
  - **Customization scripts (custom)**
  - **Add-ons (custom)**

- **Communication and Interactions**
  - **Remote API function calls**
  - **ROS transit**
  - **Custom communication (socket, serial, pipes, etc.)**
  - **Add-on calls to Lua API**
  - **Script callback calls**

- **Advanced Features**
  - **Cascaded child script execution**
  - **Lua API callbacks**
  - **V-REP event callbacks**
  - **Remote API function calls**
  - **Custom communication (socket, serial, pipes, etc.)**
  - **Add-on calls to Lua API**
  - **Script callback calls**
Other Feature: Custom User Interfaces

Custom User Interfaces

• OpenGL-based or
• Qt-based
Mesh Edit Modes

- Triangle, vertex or edge edit mode
- Modify meshes (adjust vertices, add/remove triangles)
- Semi-automatic primitive shape extraction function
- Triangle, vertex or edge extraction
- Mesh decomposition
- Convex decomposition
- Convex hull extraction
- Mesh decimation
More Features

• Headless mode support (i.e. via command line)
• Import formats: OBJ, STL, 3DS, DXF, COLLADA & URDF
• Integrated Reflexxes motion library: www.reflexxes.com
• Model browser and scene hierarchy
• Multilevel undo / redo
• Movie recorder
• Simulation of wireless communication
• Simulation of paint or welding seams
• Static & dynamic textures
• Exhaustive documentation
• Etc.
State-of-the-art distributed control architecture

- Embedded scripts
- Remote API
- 2 ROS interfaces

Extremely fine-grained and large amount of features

- >500 different API function
- 14 types of simulation objects (force/torque sensor, joint, camera, etc.)
- Integrated physics, kinematics, collision/distance calculation & path planning

V-REP sets on several horses

- Interfaces (plugins, embedded scripts, add-ons, Remote API, ROS interfaces)
- Languages (C/C++, Java, Python, Lua, Matlab, Octave, Lua, Urbi)
- Physics engines (Bullet, ODE, Vortex, Newton)
- Platforms (Windows, MacOS, Linux)
V-REP Flavours

**V-REP PRO EDU**
- For hobbyists, students, teachers, professors, schools and universities
- Free
- No limitations (i.e. fully functional)
- No registration
- Not for commercial applications
- Not for companies, research institutions, non-profit organizations, etc.

**V-REP PRO**
- For companies, research institutions, non-profit organizations, etc.
- Not free
- No limitations (i.e. fully functional)
- For commercial applications

**V-REP PLAYER**
- For everyone
- Free, can be distributed
- Limited editing capability, saving is disabled
- For any application
V-REP Source Code Licensing

PLUGINS educational license
(where 'PLUGIN' may refer to 'DYNAMICS PLUGIN' or 'MESH CALCULATION PLUGIN'):

The PLUGIN educational license applies ONLY to EDUCATIONAL ENTITIES composed by following people and institutions:

1. Hobbyists, students, teachers and professors
2. Schools and universities

EDUCATIONAL ENTITIES do NOT include companies, research institutions, non-profit organisations, foundations, etc.

An EDUCATIONAL ENTITY may use, modify, compile and distribute the modified/unmodified PLUGIN under following conditions:

1. Distribution should be free of charge.
2. Distribution should be to EDUCATIONAL ENTITIES only.
3. Usage should be non-commercial.
4. Altered source versions must be plainly marked as such and distributed along with any compiled code.
5. When using the PLUGIN in conjunction with V-REP, the "EDU" watermark in the V-REP scene view should not be removed.
6. The origin of the PLUGIN must not be misrepresented. You must not claim that you wrote the original software.

The PLUGIN is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. In no event will the original author be held liable for any damages arising from the use of this software.

The PLUGIN is copyrighted by Dr. Marc Andreas Freese (the original author). All rights reserved.

contact Coppelia Robotics for details
Resources

V-REP website: [www.coppeliarobotics.com](http://www.coppeliarobotics.com)


V-REP forum: [www.forum.coppeliarobotics.com](http://www.forum.coppeliarobotics.com)

V-REP YouTube channel: [VirtualRobotPlatform](http://VirtualRobotPlatform)

V-REP Twitter account: [coppeliaRobotic](http://coppeliaRobotic)

V-REP contact: info_at_coppeliarobotics_dot_com