Documentation

HOCHSCHULE FURTWANGEN UNIVERSITY HFU

IIoT Simulator

Java controlled V-REP simulation

Serkan Aktas

Index

Index

List of Figures

List of Figures

Introduction

This Documentation is about an IIoT Simulator. The goal of this simulator is to demonstrate how a robotic arm can be controlled and moved and gather some sensor data through a remote API.

For the simulation the software V-REP PRO EDU is used. V-REP is a robot simulator developed by Coppelia Robotics. V-REP is available in three versions. V-REP PRO EDU, V-REP PRO and V-REP PLAYER. V-REP PRO EDU is for educational purposes and free, V-REP PRO is the commercial version and V-REP PLAYER is a free software with limited editing capabilities to play templates.

To access the simulator Java is used. V-REP offers remote API for some programming language. Language currently supported are C/C++, Python, Java, Matlab, Octave, Urbi and Lua.

V-REP can be used on Windows 64 bit, Linux 64 bit and macOS. V-REP also offers source code to download. [1]

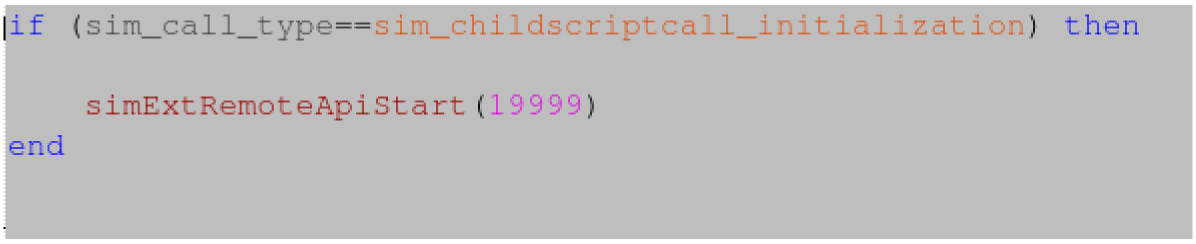
Getting started

After downloading and installing the desired version there is a package called coppelia containing 12 Java classes which can be used on Java projects and a remoteAPI Java library to register. All this files can be found in V-REP's installation directory. The shared library is platform specific.[2]

Every V-REP scene has a main script which usually launches child scripts. Main script is default and initializes everything. Child script are recommended to modify instead of the main script.

V-REP offers models which can be drag and drop to the scene. The models have some predefined script and behavior. It is possible to build or import own model to V-REP. V-REP supports file-formats like OBJ, DXF,3DS,STL, COLLADA and URDF.[3]

It is possible to create child scripts and assign them to models(Objects).
To communicate with Java it is important to give this command to a child script.

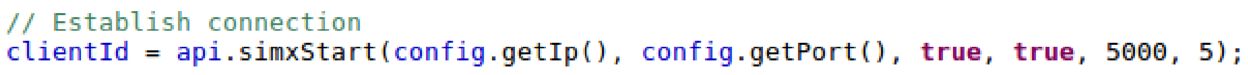


```
if (sim_call_type==sim_childscriptcall_initialization) then

    simExtRemoteApiStart(19999)
end
```

Abbildung 1: SimExtRemoteAPIStart

SimExtRemoteAPIStart(19999) enables communication with a Java client on the Port number 19999. In the Template this command is assigned to the “Dummy” object script. This scrip is a non-threaded-script. This non-threaded-script is launched by the main script.

An example how to connect to the server from a Java client is shown in Fig.2



```
// Establish connection
clientId = api.simxStart(config.getIp(), config.getPort(), true, true, 5000, 5);
```

Abbildung 2: simxStart

In this example the port number and the IP number are defined in the class Config. The other values are “waitUntilConnected “, “doNotReconnectOnceDisconnected”, “timeOutInMs”, and “commThreadCycleInMs”.[4]

When this is called from the client side and the child script in the V-REP scene is set up like in Fig. 1 than a connection should be possible. Important is to use the correct port number and the correct IP Number. Port number in this example must be 19999.

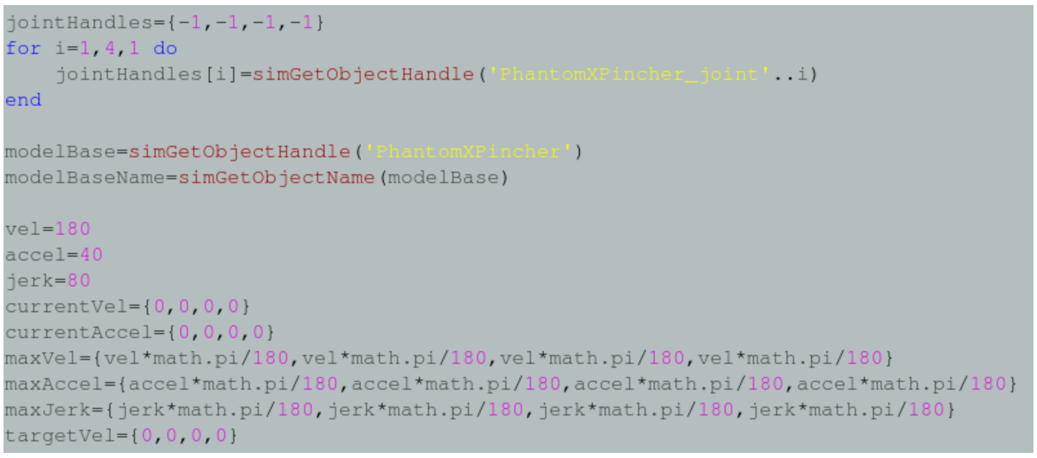
V-REP template

The Template is very important. The models must be defined and a behavior must be implemented to theme to bring theme to move.

The template which is used in this project has the robot model “PhantomXPincher” with added movement to control the robot with sliders created in Java. It also has a proximity sensor on the left gripper to detect a defined cylinder in the scene. The Model “PhantomXPincher” is a model with several links and joints that are put together to work as a unit. The Model “PhantomXPincher” has a child script assign to it which describes its behavior. This script is a threaded child script. To call threaded child scripts through remote API the script must be running and don't be finished. It is important to configure and disable the option “Execute just once ” in the “Scripts menu”.

To move robots V-REP uses Reflexxes Motion Library type II or IV. SimRMLMoveToJointPositions is an important function to move the robot. RML has vectors that must be set in the script. For the “PhantomXPincher” this can be like this:[5]

4 See http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsJava.htm#simxStart
5 See http://www.coppeliarobotics.com/helpFiles/en/regularApi/simRMLMoveToJointPositions.htm



```
jointHandles={-1,-1,-1,-1}
for i=1,4,1 do
    jointHandles[i]=simGetObjectHandle('PhantomXPincher_joint'..i)
end

modelBase=simGetObjectHandle('PhantomXPincher')
modelBaseName=simGetObjectName(modelBase)

vel=180
accel=40
jerk=80
currentVel={0,0,0,0}
currentAccel={0,0,0,0}
maxVel={vel*math.pi/180,vel*math.pi/180,vel*math.pi/180,vel*math.pi/180}
maxAccel={accel*math.pi/180,accel*math.pi/180,accel*math.pi/180,accel*math.pi/180}
maxJerk={jerk*math.pi/180,jerk*math.pi/180,jerk*math.pi/180,jerk*math.pi/180}
targetVel={0,0,0,0}
```

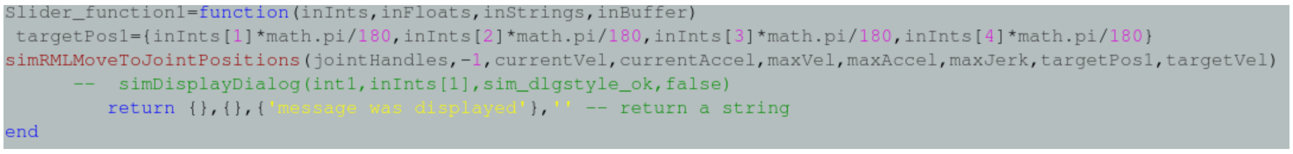
Abbildung 3: RML Vectors

JointHandles is handling the joint that are linked in the model.

Vel stands for the velocity of the joints, accel stands for acceleration of the joints and jerk stands for jerk of the joints. This vectors also have current values and max values that can be defined like in Fig 3.

Moving the “PhantomXPincher”

To move the “PhantomXPincher” there are functions created in child script of “PhantomXPincher”. This functions will be remotely called through Java.

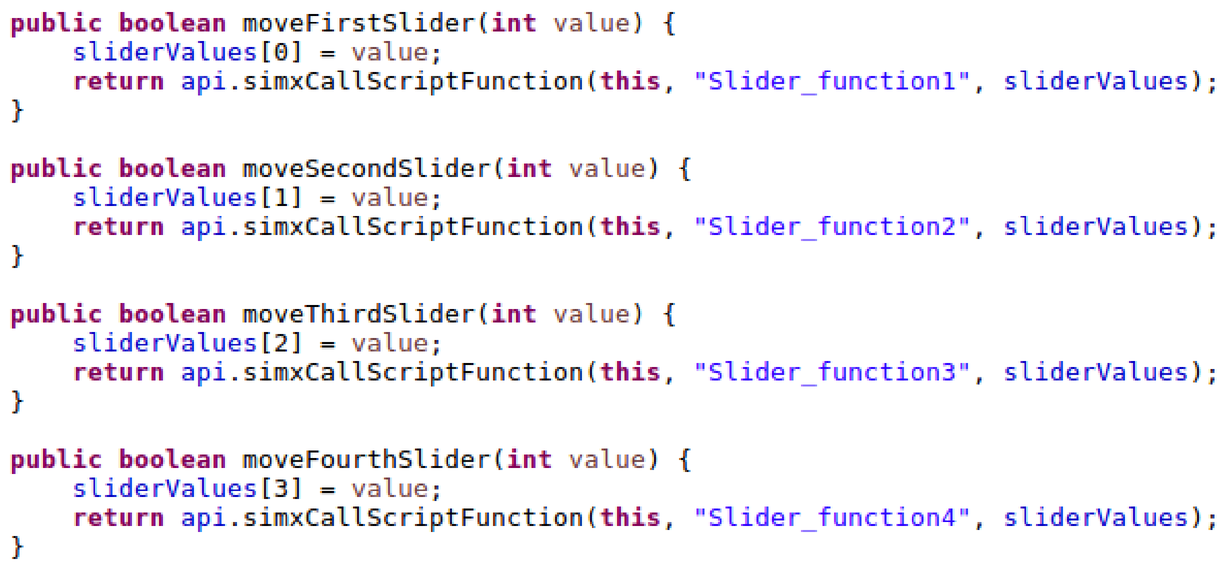


```
Slider_function1=function(inInts,inFloats,inStrings,inBuffer)
 targetPos1={inInts[1]*math.pi/180,inInts[2]*math.pi/180,inInts[3]*math.pi/180,inInts[4]*math.pi/180}
simRMLMoveToJointPositions(jointHandles,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,targetPos1,targetVel)
      --  simDisplayDialog(int1,inInts[1],sim_dlgstyle_ok,false)
         return {},{},{'message was displayed'},'' -- return a string
end
```

Abbildung 4: Slider_function1

The “Slider_function1” in the child script use the function simRMLMoveToJointPositions which is part of the regular API and moves the robot to a target position which is defined in the script.

In this example there is a defined targetPos1 with four inInts[1-4]. This inInts are controlled through Java. The inInts get filled with slider values created in Java. There are four functions called slider_function every of them has a number 1 to 4. Every number represents a slider. Every function is filled with the same code. The sliders are defined in Java. The reason why there are four slider_function is that otherwise the other values would be lost. Goal is not to lose the current position and just control a value with one slider. The other values will stay untouched till the other slider gets moved. For example if slider one gets moved than it will only change the value of inInts[1] and when moving slider two the value of inInts[2] will be changed. inInts[1] will stay changed from the movement of slider one and not altered from slider two.



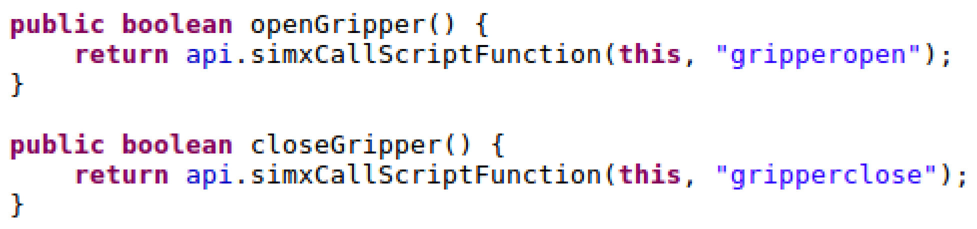
```java
public boolean moveFirstSlider(int value) {
    sliderValues[0] = value;
    return api.simxCallScriptFunction(this, "Slider_function1", sliderValues);
}

public boolean moveSecondSlider(int value) {
    sliderValues[1] = value;
    return api.simxCallScriptFunction(this, "Slider_function2", sliderValues);
}

public boolean moveThirdSlider(int value) {
    sliderValues[2] = value;
    return api.simxCallScriptFunction(this, "Slider_function3", sliderValues);
}

public boolean moveFourthSlider(int value) {
    sliderValues[3] = value;
    return api.simxCallScriptFunction(this, "Slider_function4", sliderValues);
}
```

Abbildung 5: moveSlider

Fig. 5 shows methods defined in the class PhantomXPincher. These are the methods that return the changed values from the sliders in Java to V-REP. The Important point is that it calls the Remote API function simxCallScriptFunction in Java which allows to call a defined function in V-REP simulation. Like told earlier threaded scripts must still be running. The method simxCallScriptFunction is defined in the class API and is been used here with the two additional information, name of the function and the given input to the simulation to fill the inInts in slider_function. In this example moveFirstSlider fills inInts[1] in function slider_function1 with sliderValues. This value will be used to determinate the target position to move the robot. The function will call the simRMLMoveToJointPositions to move the robot to the desired position.[6]

Gripper functions

It is also possible to control the grippers of the robot. PhantomXPincher has the functionality to close and to open its grippers. This is controlled through Java with buttons.

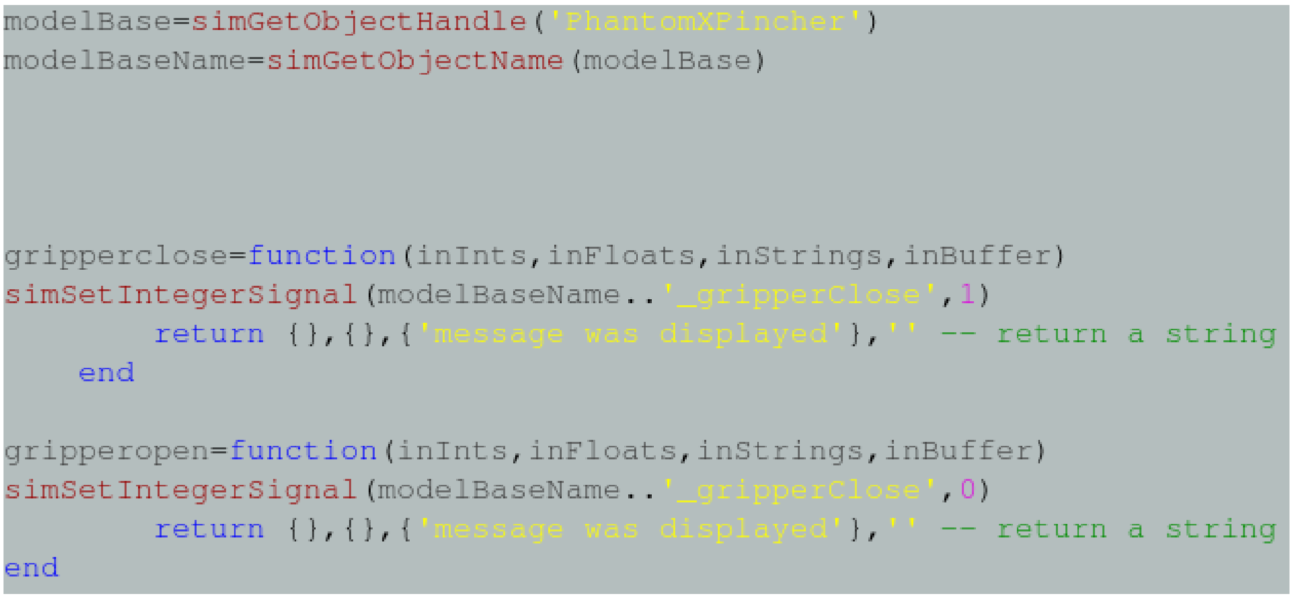


```java
public boolean openGripper() {
    return api.simxCallScriptFunction(this, "gripperopen");
}

public boolean closeGripper() {
    return api.simxCallScriptFunction(this, "gripperclose");
}
```

Abbildung 6: Gripper Java

6 See http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsJava.htm#simxCallScriptFunction

These methods do also use the remote API function simxCallScriptFunction. It does work similar to the move functions. The only difference is that there aren't inputs just a return value from the simulation. In this example it will just call the function “gripperopen”. “Gripperopen” is a defined function in the child script assigned to the Model “PhantomXPincher”.

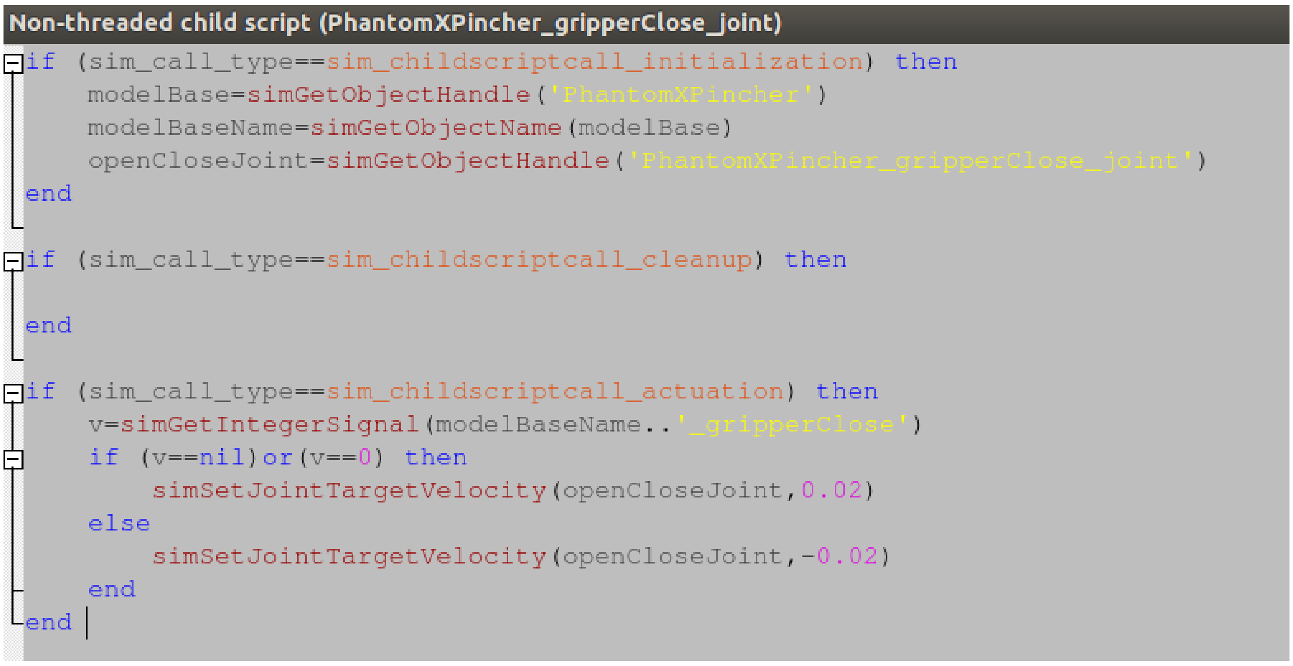


```
modelBase=simGetObjectHandle('PhantomXPincher')
modelBaseName=simGetObjectName(modelBase)




gripperclose=function(inInts,inFloats,inStrings,inBuffer)
simSetIntegerSignal(modelBaseName..'_gripperClose',1)
        return {},{},{'message was displayed'},'' -- return a string
    end

gripperopen=function(inInts,inFloats,inStrings,inBuffer)
simSetIntegerSignal(modelBaseName..'_gripperClose',0)
        return {},{},{'message was displayed'},'' -- return a string
end
```

Abbildung 7: Gripper V-REP

The functions “gripperclose" and "gripperopen" use the regular API function simSetIntegerSignal. To get the “modelBaseName” and the “ModelBase” the function “simGetObjectHandle” and “simGetObjectName” are called. The object is the “PhantomXPincher”. After getting the name it is possible to use the name and get to signal of “gripperClose” and set it to 1 to close the gripper. 0 would open the gripper.[7][8][9]

7 See http://www.coppeliarobotics.com/helpFiles/en/regularApi/simSetIntegerSignal.htm
8 See http://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm#simGetObjectHandle
9 See http://www.coppeliarobotics.com/helpFiles/en/regularApi/simGetObjectName.htm


Non-threaded child script (PhantomXPincher_gripperClose_joint)

```
if (sim_call_type==sim_childscriptcall_initialization) then
    modelBase=simGetObjectHandle('PhantomXPincher')
    modelBaseName=simGetObjectName(modelBase)
    openCloseJoint=simGetObjectHandle('PhantomXPincher_gripperClose_joint')
end

if (sim_call_type==sim_childscriptcall_cleanup) then

end

if (sim_call_type==sim_childscriptcall_actuation) then
    v=simGetIntegerSignal(modelBaseName..'_gripperClose')
    if (v==nil)or(v==0) then
        simSetJointTargetVelocity(openCloseJoint,0.02)
    else
        simSetJointTargetVelocity(openCloseJoint,-0.02)
    end
end
```

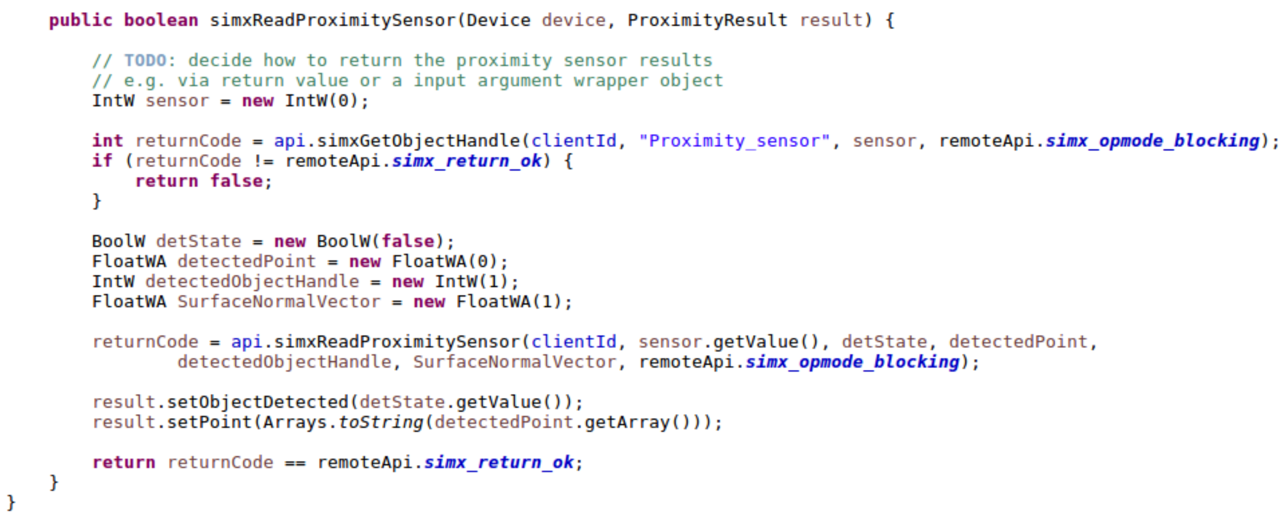
Abbildung 8: PhantomXPincher_gripperClose_joint

More details how to close the gripper are in the child script assigned to object "PhantomXPincher_gripperClose_joint" in Fig. 8

Read Proximity Sensor

SimxReadProximitySensor is a function of the remote API. This function reads the result of a proximity sensor and returns detection state and detection point. To use this function there must be a sensor handle. To get the sensor handle the function simxGetObjectHandle must be called. This will return the object handle of the sensor. [10][11]

10 See http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsJava.htm#simxReadProximitySensor
11 See http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsJava.htm#simxGetObjectHandle



```
    public boolean simxReadProximitySensor(Device device, ProximityResult result) {

        // TODO: decide how to return the proximity sensor results
        // e.g. via return value or a input argument wrapper object
        IntW sensor = new IntW(0);

        int returnCode = api.simxGetObjectHandle(clientId, "Proximity_sensor", sensor, remoteApi.simx_opmode_blocking);
        if (returnCode != remoteApi.simx_return_ok) {
            return false;
        }

        BoolW detState = new BoolW(false);
        FloatWA detectedPoint = new FloatWA(0);
        IntW detectedObjectHandle = new IntW(1);
        FloatWA SurfaceNormalVector = new FloatWA(1);

        returnCode = api.simxReadProximitySensor(clientId, sensor.getValue(), detState, detectedPoint,
                detectedObjectHandle, SurfaceNormalVector, remoteApi.simx_opmode_blocking);

        result.setObjectDetected(detState.getValue());
        result.setPoint(Arrays.toString(detectedPoint.getArray()));

        return returnCode == remoteApi.simx_return_ok;
    }
}
```

Abbildung 9: Proximity Sensor Java

In the example in Fig. 9 the sensor is called “Proximity_Sensor” .The return value will be saved in the sensor variable. After getting the sensor handle it is possible to use that result as sensor handling in the simxReadProximitySensor function of the remote API.

Future goals

The next steps in the project will be to develop a template with multiple robots and mange their control through java."PhantomXPincher" is currently the only robot programmed in the template to work with java. The goal is to demonstrate how remotely a robot can be controlled and give back data. Industry 4.0 plays an important role in the industry now days. This Project has the goal to control things remotely. This would make working with robots easier.