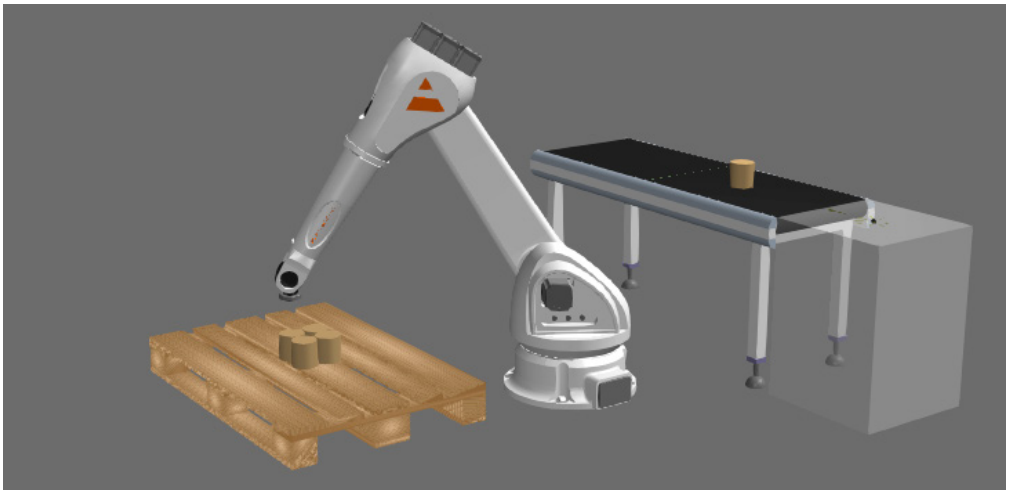
Control Robots

Visual Components 4.0 | Version: March 6, 2017



Python API can be used to control robots during a simulation. For example, you can write a component script that executes subroutines in a robot program for picking and placing parts at different locations. That, of course, would not require you to execute the entire robot program. You can create subroutines for tasks dynamically and automate robot motions using scripts and helper libraries.

In this tutorial you learn how to:

- Control the actions of a robot during a simulation with a script.
- Call subroutines in a robot program
- Move and drive robot joints.
- Use helper libraries to record and automate robot routines.
- Control a robot from a different component.

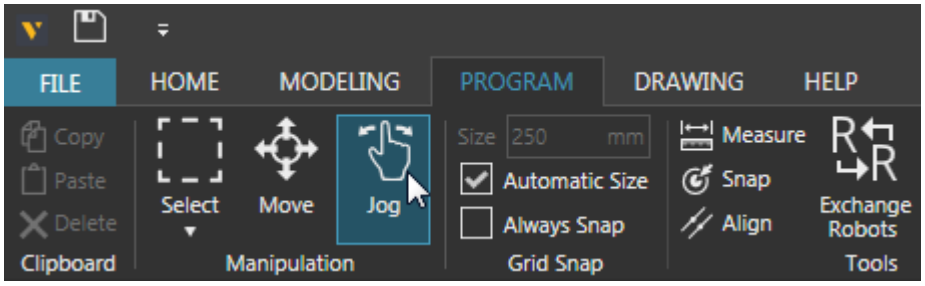
Support
support@visualcomponents.com

Community
community.visualcomponents.net

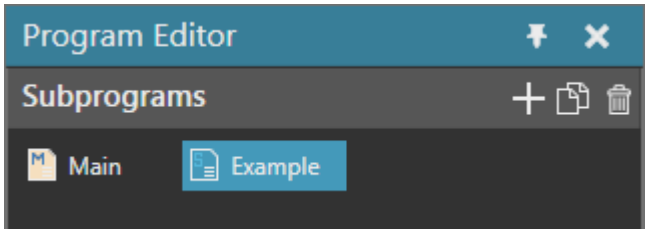
Getting Started

1. In the eCatalog panel, Collections view, browse to **Models by Type > Robots > Visual Components** and then add a **Generic Articulated Robot** to the 3D world.

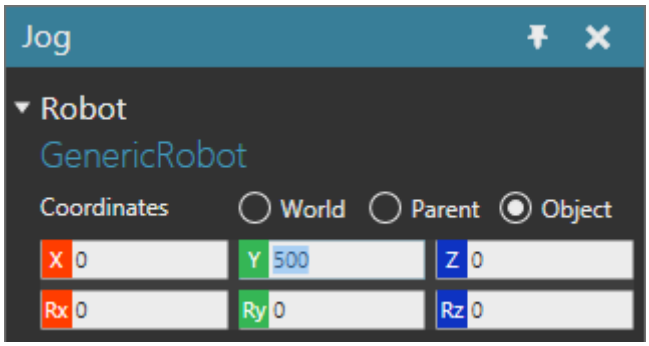
2. Click the **Program** tab, and then use the **Jog** command to select the **robot** in the 3D world. In some cases, the robot will be automatically selected for you when you enable the Jog command.



3. In the Program Editor panel, click **Add Sequence**, and then rename the new sequence **Example**.

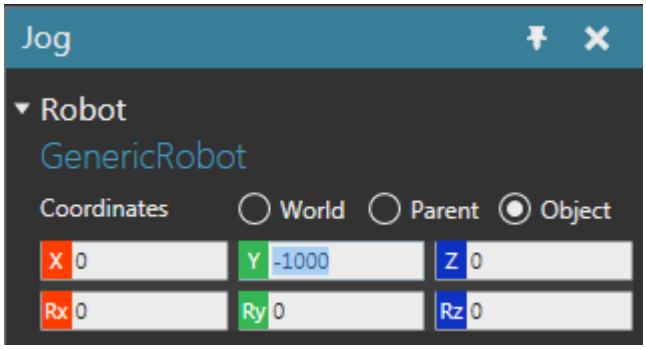


4. In the Jog panel, Robot section, set Coordinates to **Object**, and then set the Y-axis coordinate to **500**.

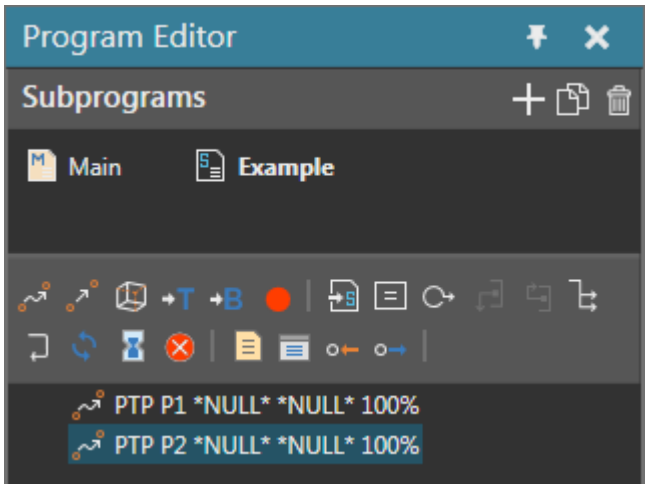


5. In the Program Editor panel, Example sequence, add a **Point-to-Point Motion** statement.

6. In the Jog panel, Robot section, set the Y-axis coordinate to **-1000**.



7. In the Program Editor panel, Example sequence, add a **Point-to-Point Motion** statement. There should now be two motion statements in the Example subroutine.



8. Click the **Modeling** tab, and then add a **Python Script** behavior. The script editor will open automatically when you add the behavior.

Call Subroutines

The executor of a robot program can be used to call subroutines in its executable program during a simulation. If you do not want a robot to execute its Main routine, set the **IsEnabled** property of its executor to False.

1. On the Simulation controls, click **Reset** to return the robot to its initial joint configuration.

2. In the script editor, add the following code to the OnRun event, and then compile the code.

```
from vcScript import *


def OnRun():
  comp = getComponent()
  robot_executor = comp.findBehaviour("Executor")
  robot_executor.IsEnabled = False


  routine = robot_executor.Program.findRoutine("Example")
  if routine:
    robot_executor.callRoutine(routine)
```

3. Run the simulation, verify the robot executes the Example subroutine, and then reset the simulation.

Another way to call routines in a robot program is to use the **vcHelpers.Robot2** library. The library contains many useful methods for controlling a robot and will be used for the remainder of this tutorial.

4. In the script editor, import the vcHelpers.Robot2 library, edit the OnRun event to have the robot call the Example subroutine, and then compile the code.

```
from vcScript import *
from vcHelpers.Robot2 import *


def OnRun():
  robot = getRobot()
  robot.callSubRoutine("Example")
```

5. Run the simulation, verify the robot executes the Example subroutine, and then reset the simulation.

Move Joints

The joints of a robot can be moved individually or together during a simulation. For example, you can use the **driveJoints()** method in vcHelpers.Robot2 or the **moveJoint()** method available to servo and robot controllers.

1. In the script editor, OnRun event, drive the joints of the robot to zero value before executing the Example subroutine, and then compile the code.

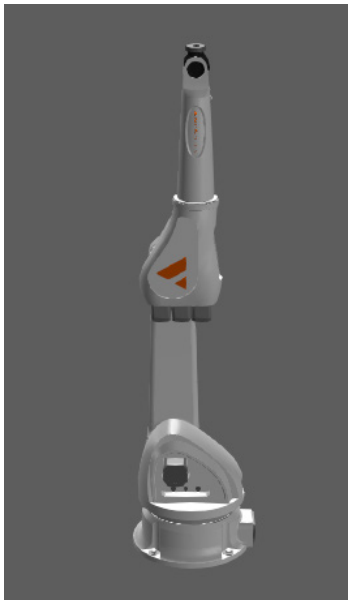
```
def OnRun():
  robot = getRobot()
  robot.driveJoints(0,0,0,0,0,0)
  robot.callSubRoutine("Example")
```

2. Run the simulation, verify the robot goes to its joint zero position, and then reset the simulation.

3. In the script editor, OnRun event, move the third joint of the robot to 90.0 immediately after the robot moves to its joint zero position, and then compile the code. This will move the robot to its joint initial position.

```
def OnRun():
  robot = getRobot()
  robot.driveJoints(0,0,0,0,0,0)
  robot.Controller.moveJoint(2,90.0)
  robot.callSubRoutine("Example")
```

4. Run the simulation, verify the robot moves to its joint initial position, and then reset the simulation.



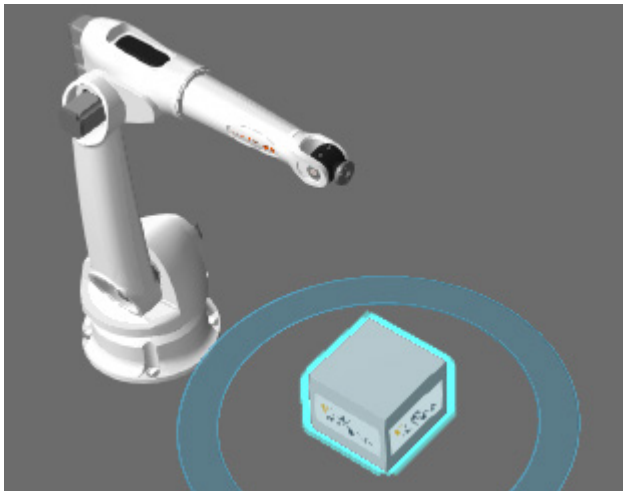
Joint zero position

Joint initial position

Pick Stationary Parts

You can instruct a robot to pick stationary parts during a simulation. For example, you can use the **pick()** method in vcHelpers.Robot2 to pick a component in the 3D world.

1. Click the **Home** tab, and then in the eCatalog panel, Collections view, browse to **Models by Type > Products and Containers** and then add a **Visual Components Box** to the 3D world and move it within reach of the robot.

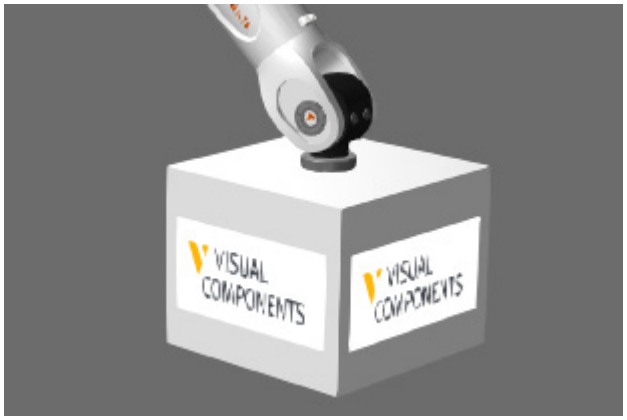


2. In the script editor, modify the OnRun event to pick the box using the pick() method available in vcHelpers.Robot2 with a 300.0 distance, and then compile the code.

```
from vcScript import *
from vcHelpers.Robot2 import *


def OnRun():
  app = getApplication()
  part = app.findComponent("VisualComponents_Box")
  robot = getRobot()
  robot.pick(part,300.0)
```

3. Run the simulation, verify the robot picks up the box, and then reset the simulation.



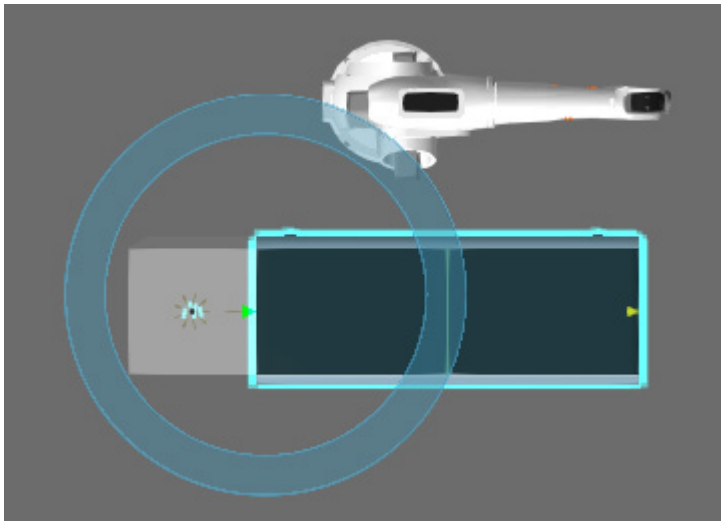
Pick Moving Parts

You can control a robot from a different component and instruct the robot to pick moving parts during a simulation. The robot and the controlling component do not have to be connected to one another. For example, you can add a component script to a conveyor that instructs a robot to pick parts that trigger a sensor. In such cases, the **pickMovingPart()** method in vcHelpers.Robot2 can be used to pick moving components.

1. In the 3D world, delete the **box**. This will not break the script in the robot used for picking the box because the robot will be told to pick nothing in that case.

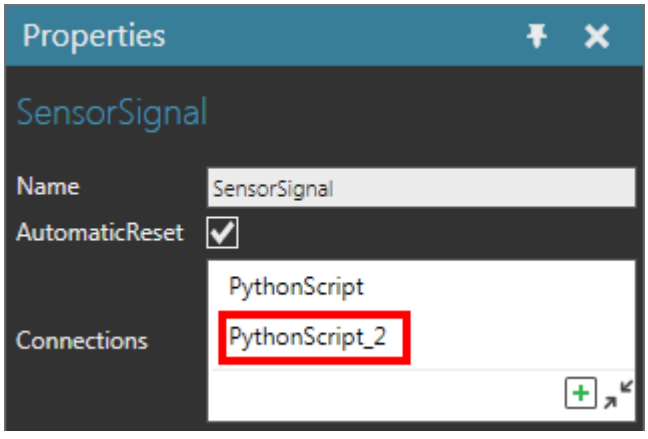
1. In the eCatalog panel, Collections view, browse to **Models by Type > Feeders** and then add a **Basic Feeder** to the 3D world.

2. Browse to **Models by Type > Conveyors > Visual Components** and then add and connect a **Sensor Conveyor** to the **feeder** in the 3D world, and then move those components so that the sensor line is within reach of the robot.



3. In the 3D world, select the **conveyor**.

4. Click the **Modeling** tab, and then add a **Python Script** behavior and connect that script to the **SensorSignal** behavior. This will allow the script to know what component triggers the path sensor.



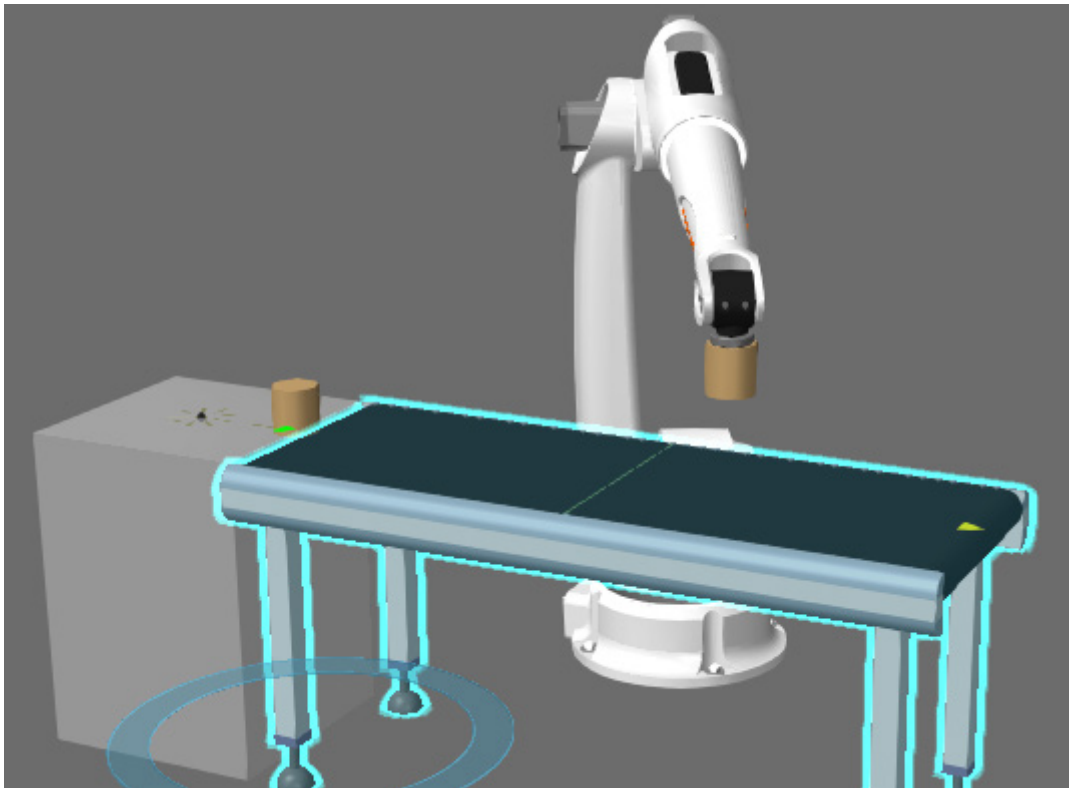
5. In the PythonScript_2 script editor, add the following code to instruct the robot to pick a part that triggers the path sensor, and then compile the code.

Sensor Conveyor::PythonScript2*

Find

```
1   from vcScript import *
2   from vcHelpers.Robot2 import *
3
4   def OnRun():
5       #start by getting part
6       comp = getComponent()
7       sensor_signal = comp.findBehaviour("SensorSignal")
8       triggerCondition(lambda: sensor_signal.Value != None)
9       part = sensor_signal.Value
10
11      #now get robot
12      app = getApplication()
13      robot_comp = app.findComponent("GenericRobot")
14      robot = getRobot(robot_comp)
15
16      #pick the part
17      robot.pickMovingPart(part)
```

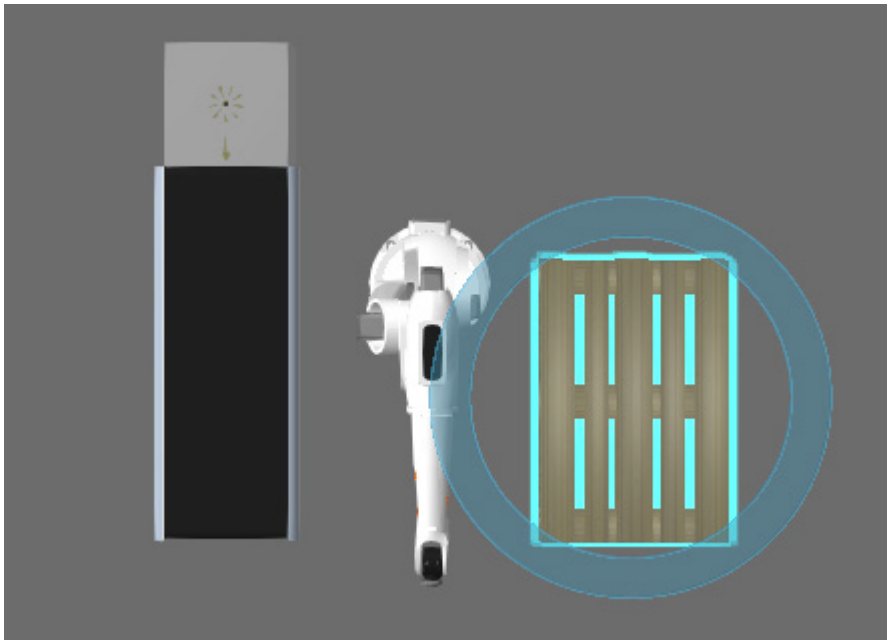
6. Run the simulation, verify the robot picks a moving part from the conveyor, and then reset the simulation.



Place Parts

You can instruct a robot to place parts during a simulation. For example, you can use the **place()** method in vcHelpers.Robot2 to place a component on top of a table, pallet or conveyor.

1. Click the **Home** tab, and then in the eCatalog panel, Collections view, browse to **Models by Type > Products and Containers** and then add a **Euro Pallet** to the 3D world that is within reach of the robot.

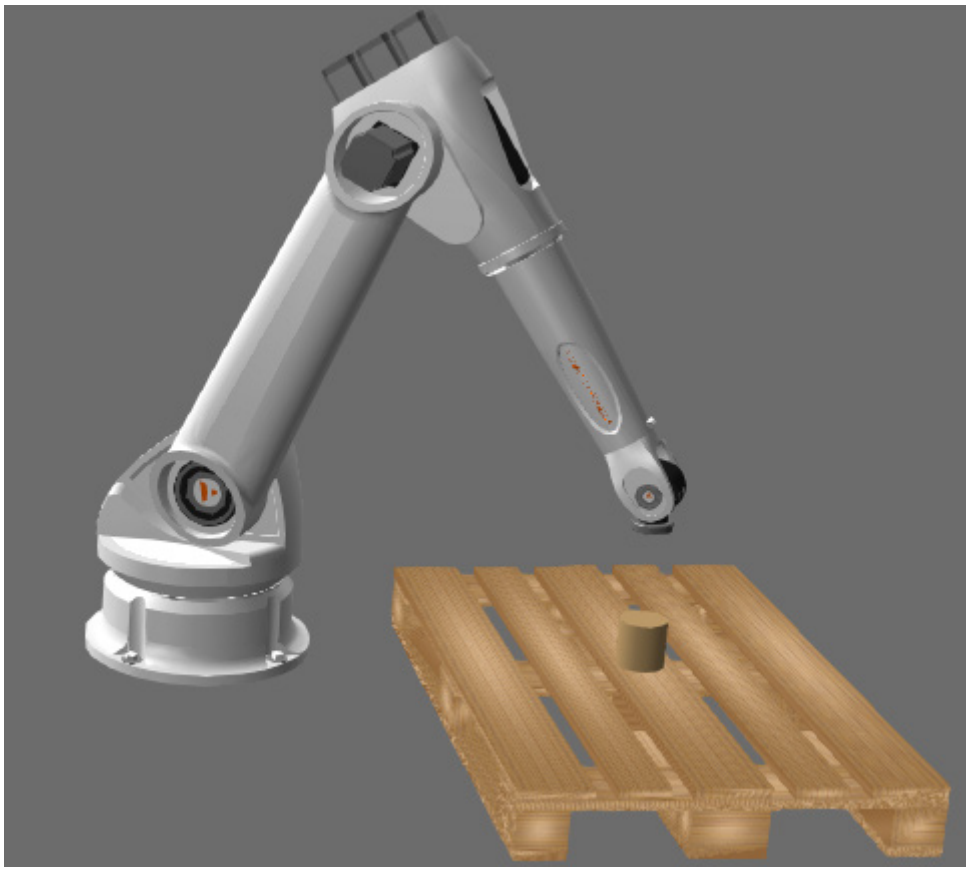


2. Access the editor for **PythonScript_2** in the conveyor, and then in the OnRun event, instruct the robot to place a picked part on top of the pallet, and then compile the code.

```
def OnRun():
...
  #pick the part
  robot.pickMovingPart(part)


  #place the part
  pallet = app.findComponent("Euro Pallet")
  robot.place(pallet)
```

3. Run the simulation, verify the robot picks and places a part on top of the pallet, and then reset the simulation.



Pick Parts From Pallet

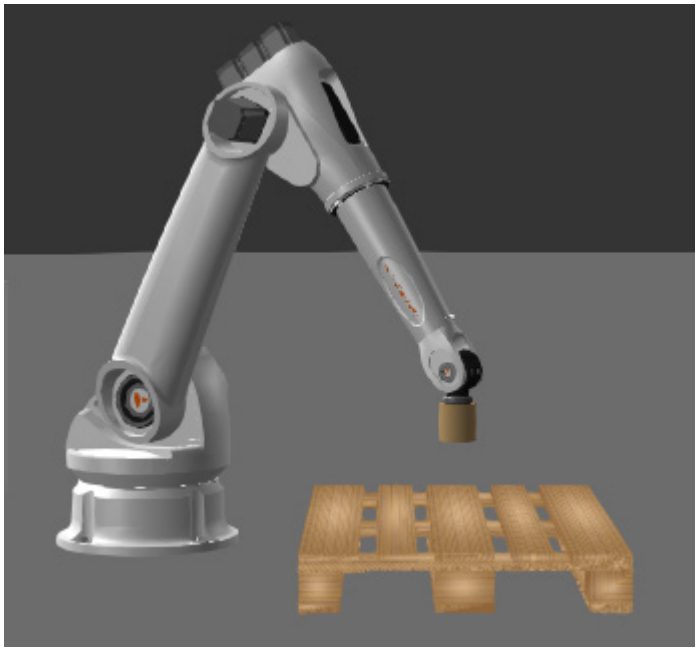
You can instruct a robot to pick parts from pallets during a simulation. For example, you can use the **pickPartFromPallet()** method in vcHelpers.Robot2 to pick components attached to a pallet.

1. Access the editor for **PythonScript_2** in the conveyor, and then in the OnRun event, instruct the robot to pick up the part it placed on the pallet, and then compile the code.

```
def OnRun():
...
  #place the part
  pallet = app.findComponent("Euro Pallet")
  robot.place(pallet)


  #pick same part from pallet
  robot.pickFromPallet(pallet)
```

2. Run the simulation, verify the robot picks up a part it placed on the pallet, and then reset the simulation.



Record Routines

During a simulation you may want to record the actions of a robot as a routine in its program. For example, you can use the **RecordRoutine** and **RecordRSL** properties in vcHelpers.Robot2 to record routines for picking and placing components.

1. In the 3D world, select the **conveyor**, and then in the Component Properties panel, set OnSensor to **Stop product**. This will stop a part that triggers the path sensor.

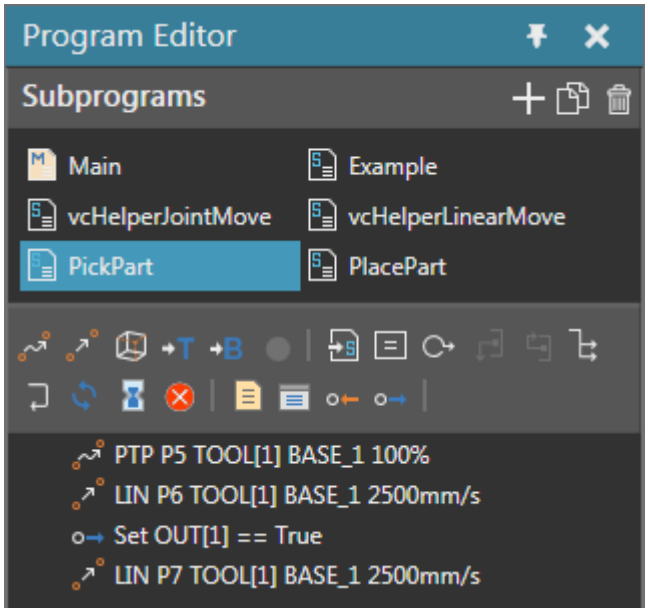
2. Access the editor for PythonScript_2 in the conveyor, and then in the OnRun event, modify the code to record two routines for picking and placing a part.

```
def OnRun():
...
  #record PickPart routine
  robot.RecordRoutine = "PickPart"
  robot.RecordRSL = True
  robot.pick(part)
  robot.RecordRSL = False

  #record PlacePart routine
  robot.RecordRoutine = "PlacePart"
  robot.RecordRSL = True
  pallet = app.findComponent("Euro Pallet")
  robot.place(pallet)
  robot.RecordRSL = False
```

3. Run the simulation, wait for the robot to pick and place a part, and then reset the simulation.

4. Click the **Program** tab, select the **robot**, and then verify in the Program Editor panel that two subroutines have been created for picking and placing a part.



NOTE! You need to define the name of a routine before recording. If the routine already exists in robot, statements will be appended to that routine. That is, recording will not override a routine.

Place Parts in Pattern

You can instruct a robot to place parts in a pattern during a simulation. For example, you can use the **placeInPattern()** method in vcHelpers.Robot2 to place components on top of a table, pallet or conveyor in a set pattern.

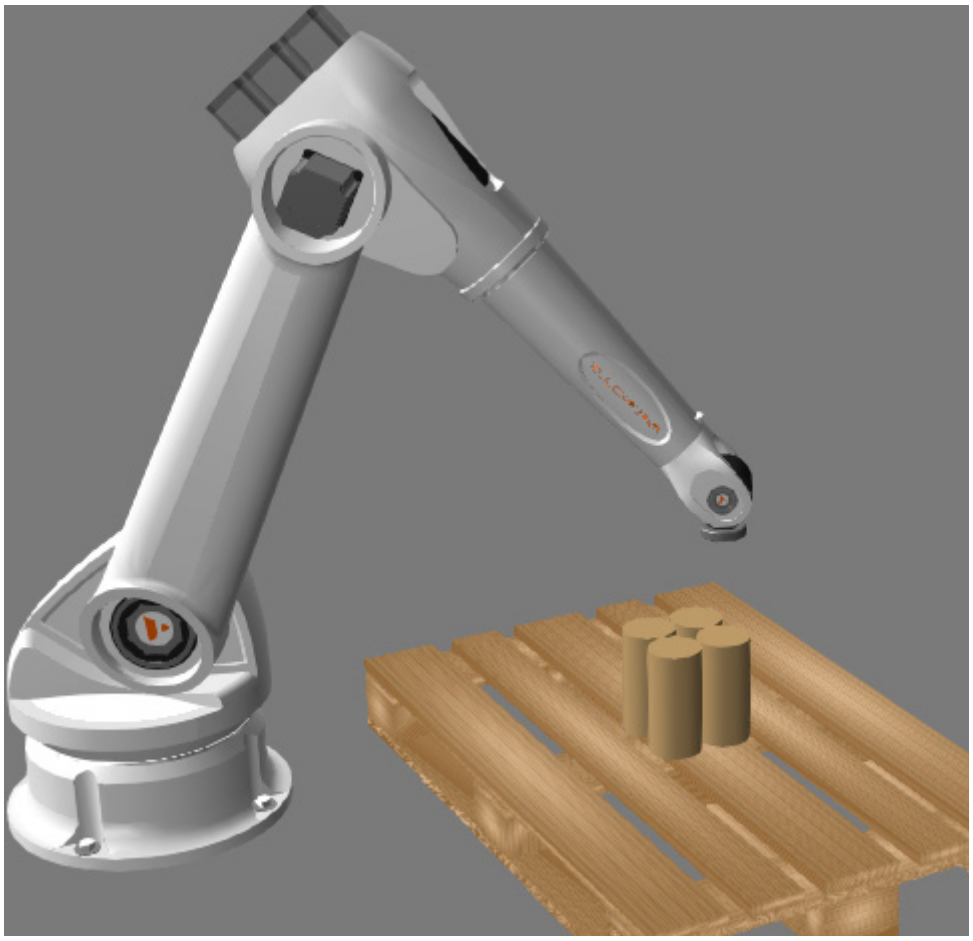
1. Access the editor for **PythonScript_2** in the conveyor, and then in the OnRun event, modify the code to call the recorded routine for picking parts, place the parts in an XYZ pattern of {2,2,2} on the pallet, and then compile the code.

```
def OnRun():
  #get robot
  app = getApplication()
  robot_comp = app.findComponent("GenericRobot")
  robot = getRobot(robot_comp)

  #use sensor signal to initiate part pick-up
  comp = getComponent()
  sensor_signal = comp.findBehaviour("SensorSignal")

  stack_size = 8
  x,y,z = 0,0,0
  while stack_size > 0:
    triggerCondition(lambda: sensor_signal.Value != None)
    part = sensor_signal.Value
    robot.callSubRoutine("PickPart")
    pallet = app.findComponent("Euro Pallet")
    robot.placeInPattern(pallet,x,y,z,2,2,2)
    stack_size -= 1
    #x,y,z are index values
    if x < 1:
      x += 1
    else:
      x = 0
      y += 1
    if y > 1:
      y = 0
      z += 1
```

2. Run the simulation, verify the robot stacks parts on the pallet, and then reset the simulation.



Review

In this tutorial you learned how to control a robot with a component script. You know how to manipulate the controller, executor and program of a robot. You also know how to use the vcHelpers.Robot2 library to teach and record automated routines for robots.