

## Applications with Universal Robots and *VeriSens*<sup>®</sup> with different heights as well as relative object positions

**AN202005/v0.2/2023-08-21**

### Description

Solutions for applications with Universal Robots (UR) and *VeriSens*<sup>®</sup> with different heights as well as relative object positions

### Products

*VeriSens*<sup>®</sup> XF900 and XC900 series

## Content

<b>1</b>	<b>Technical background</b>	<b>2</b>
<b>2</b>	<b>Option 1 – simply create several jobs</b>	<b>2</b>
2.1	Applications	2
2.2	Advantages and disadvantages of the option	3
2.3	Installation and calibration	3
2.4	Image processing job and robot program	3
<b>3</b>	<b>Option 2 – Movement affected by the robot program</b>	<b>4</b>
3.1	Applications	4
3.2	Advantages and disadvantages of the option	4
3.3	Installation and calibration	4
3.4	Image processing job and robot program	5
3.5	Programming example 1 – winner’s podium	6
3.5.1	Description of the application	6
3.5.2	Creating the variables	6
3.5.3	Creating the program	7
3.6	Programming example 2 – four screws depending on a variable position	10
3.6.1	Description of the application	10
3.6.2	Creating the variables	11
3.6.3	Creating the program	12
<b>4</b>	<b>Summary / special cases</b>	<b>17</b>
<b>5</b>	<b>Downloads</b>	<b>17</b>
<b>6</b>	<b>Support</b>	<b>17</b>
<b>7</b>	<b>Legal information</b>	<b>17</b>

## 1 Technical background

Image processing with *VeriSens*<sup>®</sup> is based on 2D images. Robotics operate in the 3D realm. The effect of the third dimension on the 2D scaling is already taught in during the calibration process via *SmartGrid* when *VeriSens*<sup>®</sup> / UR is integrated. During the subsequent setup, this makes it considerably easier to handle the different “Z” values such as object height, grabbing height, or work surface height, as all values can operate with a “reference level.”

The height of an object is measured and stored in the image processing job. This enables *VeriSens*<sup>®</sup> to convey to the robot not only the  $x$ ,  $y$ , and  $rotZ$  that were determined by the 2D image processing but also the value  $z$  of the object surface that is “seen”.

### Note

To put it simply,  $rotX$  and  $rotY$  are zero, accordingly are all 3D coordinates covered.

This leads to the justified question of whether *VeriSens*<sup>®</sup> and UR can also be used to handle objects with different heights in an application.

Imagine a winner’s podium (Figure 1).

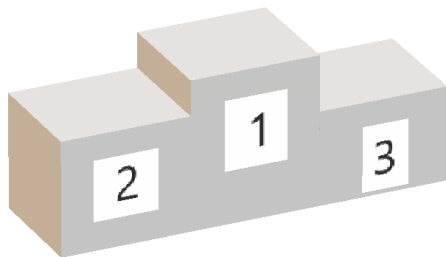


Figure 1: Winner’s podium

You want to use image processing to measure from above whether the platforms have the correct dimensions.

**Problem:** Capturing an image from an overhead camera position makes the captured platforms appear differently sized due to their varying distances from the capturing device.

How can such applications be resolved?

## 2 Option 1 – simply create several jobs

### 2.1 Applications

An object with varying heights is to be inspected or measured with limited effort. All heights are in the depth of field area of the *VeriSens*<sup>®</sup> in its calibrated capturing position.

Using the example of our winner’s podium, it must therefore be possible for all 3 steps to be in the depth of field area. With the *VeriSens*<sup>®</sup> XC series, the depth of field can be influenced by the choice of the respective lens.

*VeriSens*<sup>®</sup> is mounted dynamically according to the technical documentation – i.e. carried along. Alternatively, it can also be mounted statically, i.e., above the robot.

## 2.2 Advantages and disadvantages of the option

### Advantages:

- Very easy to comprehend and implement
- Can be implemented with both mounting options

### Disadvantages:

- Object heights must be located in a common depth of field area
- Specific object heights must be known when creating the program

## 2.3 Installation and calibration

The application is calibrated via *SmartGrid*. The focus must not be shifted anymore. The image capturing position is stored via *VeriSens® URCap* in step 2.

### Note

Please make sure that when the distance between the *SmartGrid* and the reference level is entered in both the *VeriSens® Application Suite* and *VeriSens® URCap*, the additional distance between the *SmartGrid* bottom side and reference level is added to the *SmartGrid* material thickness if the *SmartGrid* is not directly placed on the reference level.

After the automatic alignment of the coordinates is completed in installation step 3 of the *VeriSens® URCap*, the application can be created.

## 2.4 Image processing job and robot program

For each height to be inspected – in our case the three heights of the platforms – a *VeriSens®* job must be created or adapted (“Platform\_1”, “Platform\_2”, etc.).

For this purpose, the respective object height (height of the step to the reference level) must be saved in every job under Coordinates/Z-correction.

As *VeriSens®* is familiar with the overall system from the calibration, it is capable of correspondingly scaling objects that appear to have different sizes due to their heights for each job correctly in  $x$  and  $y$  and thus verifying the correct dimensions.

How can the robot be additionally moved to the different heights, e.g., to assemble the winner’s podium from the individual steps using the robot (Figure 2)?

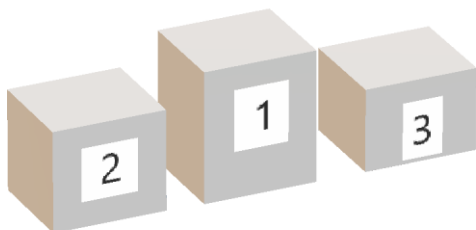


Figure 2: Individual steps of a winner’s podium

Very simply, the movement of the robot is based on the coordinates of the respective upstream “*VeriSens®* Job Execution” in the robot’s program using the individual *VeriSens®* job for each level. This provides the coordinates to the subsequent “*VeriSens®* Waypoint & Move”, including  $z$  of the object height. At this node, an additional, corresponding  $Z$  offset can be freely selected each time, allowing the grabber, e.g., to move to a position underneath the surface of the step.

If the individual steps are provided randomly, the selection of the associated job could take place based on the results of a distance sensor (measurement of the step height), for example.

### 3 Option 2 – Movement affected by the robot program

#### 3.1 Applications

- A screwdriver guided by a robot arm is to move to four screws in the corners of an object, depending on a position determined by *VeriSens*<sup>®</sup>.
- A distance sensor moving along with the UR provides the distance to the object, allowing the inspection to take place based on the height and thus flexibly.
- For application reasons, the robot program itself is to introduce an additional offset to the image-based coordinates provided by *VeriSens*<sup>®</sup> for the waypoint.
- With option 2, the robot in our example of a winner's podium moves to each step of the podium individually to allow measurement with a constant distance to *VeriSens*<sup>®</sup>.

Only dynamic mounting of the *VeriSens*<sup>®</sup>, i.e., moving along with the robot, is possible.

#### 3.2 Advantages and disadvantages of the option

##### Advantages:

- Different object heights do not have to be in the same depth of field area
- A distance sensor can be used for the automatic determination of the object height
- Positions depending on the heights  $z$  or  $x, y, \text{rot}Z$  can be approached

##### Disadvantages:

- More complex than option 1
- Limited to dynamic mounting with its advantages and disadvantages (see the documentation)

#### 3.3 Installation and calibration

In our example of a winner's podium, the height of precisely one platform is measured in relation to the reference level and permanently defined in the robot program. This offers the advantage that only one image processing job is required and the work is also done independently of the depth of field area of *VeriSens*<sup>®</sup>. The addition to or deduction from coordinate  $z$  required for steps of different heights is specified by the robot program itself.

The system is calibrated in *VeriSens*<sup>®</sup> *URCap* based on the top step of our winner's podium, to which the *SmartGrid* is applied. In this captured image, the robot position is stored under step 2 as standard. The coordinates are now calibrated in step 3; the focus of the *VeriSens*<sup>®</sup> should now no longer be shifted.

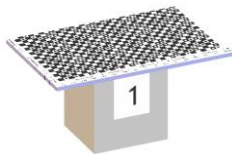


Figure 3: *SmartGrid* on the surface of the highest step

##### Note

Please make sure that when the distance between the *SmartGrid* and the reference level is entered, the additional distance between the *SmartGrid* bottom side and the reference level is added to the *SmartGrid* material thickness in both *VeriSens*<sup>®</sup> *Application Suite* as well as in *VeriSens*<sup>®</sup> *URCap*, as the *SmartGrid* is not directly placed on the reference level. The distance between the *SmartGrid* and the reference level therefore consists of the sum of the *SmartGrid* material thickness and the height of the step.

We have chosen the top step for the calibration to avoid conflicts with the maximum height during the subsequent process.

### 3.4 Image processing job and robot program

In our example of grabbing individual steps of the winner's podium, an offset  $z$  is required for each individual step 1, 2, and 3.

The applicative trick now consists of the robot program itself or even a distance sensor setting this additional offset  $z$  to specified waypoint coordinates or image-based coordinates by incorporating both coordinates in the robot program calculations.

Specifically, offset  $z$  is used twice in the robot program:

- 1st height adjustment: waypoint before "VeriSens® Job Execution"
- 2nd height adjustment: waypoint after "VeriSens® Waypoint & Move"

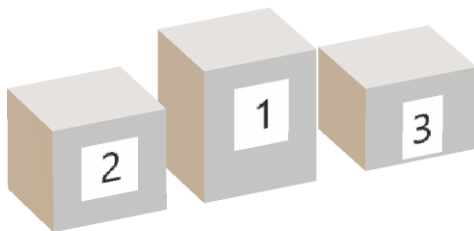


Figure 4: Individual steps of the winner's podium

#### 1st height adjustment

The same "VeriSens® Job Execution" takes place for every platform. However, before the job is executed via the program-based movement of the robot, a fixed working distance between the top of the step and VeriSens® is determined for each of the three steps so that the displayed platform always has same size in the image and can be correctly measured.

#### 2nd height adjustment

The coordinate  $z$  provided by VeriSens® or saved in the image processing job (under Coordinates / Z correction) is always the same. The stated value is the height of step 1 (top side) in relation to the reference level, as we have created one job for this.

This is why a script-based calculation is required in the robot, as both sensors or values are only combined in the robot, or an additional, fixed offset value is only added to the image-based waypoint there.

For this purpose, the script must first acquire the coordinates of the approached image-based waypoint for further calculation, which takes place by moving to a position with a fixed offset from the node "VeriSens® Waypoint & Move".

The waypoint to be approached for the grabbing task of the UR is then calculated from these stored coordinates and the above-mentioned additional value for the height adjustment of the respective step.

For grabbing tasks, which typically require a position underneath the surface of the object, an object-based offset from the robot program continues to be used.

### 3.5 Programming example 1 – winner’s podium

#### 3.5.1 Description of the application

We now look at the winner’s podium (Figure 1) and show how the different object heights  $z$  are handled: the robot program itself maintains the distance between *VeriSens*<sup>®</sup> and the object surface that is specified in the image processing job.

#### 3.5.2 Creating the variables

The following variables are used by the program and must therefore be created in advance (Figure 5). The initially determined values are adjusted while the program is running.

`offs_exec_z`

Vertical shift of the image capturing position in relation to the height during the coordinate calibration (installation).

`offs_move_z`

Vertical shift that was set manually in the *VeriSens*<sup>®</sup> waypoint.

`pos_WP`

Calculated position of a waypoint that is to be approached in the program.

Syntax: `p[x, y, z, rotX, rotY, rotZ]`

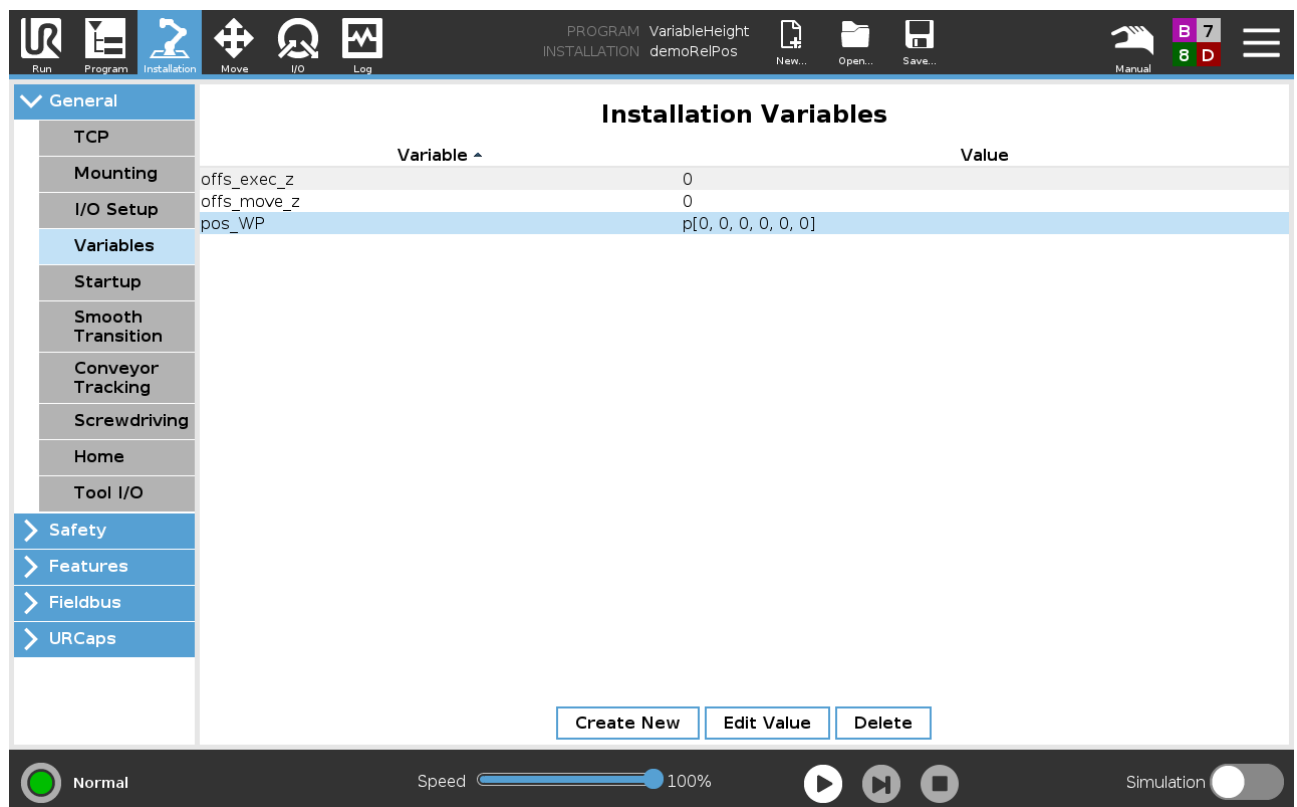


Figure 5: UR controls, Installation mode for variables

### 3.5.3 Creating the program

Robot programming can now commence (Figure 6).

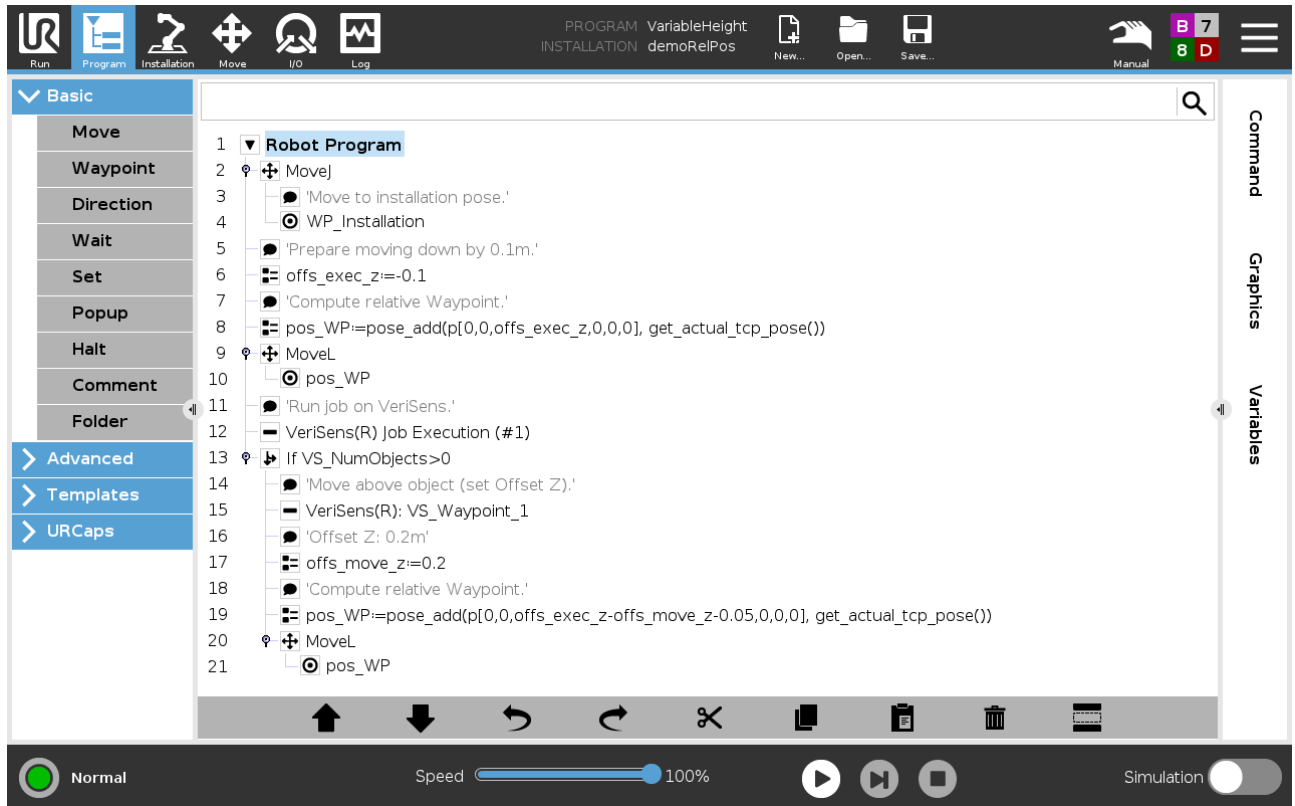


Figure 6: UR controls, programming

#### Program description

Line	Explanation
4	→ Basic → Waypoint  Move to a waypoint that either corresponds to the pose of the vision sensor during installation or a pose with the same height z only as during the installation.
6	→ Advanced → Assignment  Set the variable <code>offs_exec_z</code> to a z offset by which the job execution pose is to be shifted.  Example: The current inspection level is 100 mm lower than the pose during calibration (neg. z direction) => -0.1 meter  Alternatively, the result of an external distance sensor can also be used here.
8	→ Advanced → Assignment  1st height adjustment: Set variable <code>pos_WP</code> to a calculated value. This is calculated with <code>pose_add()</code> as the sum of the following two values: <ul style="list-style-type: none"> <li><code>p[0, 0, offs_exec_z, 0, 0, 0]</code> ... a relative shift created in the Z direction</li> <li><code>get_actual_tcp_pose()</code> ... the current position of the robot</li> </ul>

**Note:**

- For use in a formula, a pose can be created with the following syntax:  $p[x, y, z, rotX, rotY, rotZ]$ . In this formula  $x, y, z$  indicate the coordinates (e.g. metric in “meters”) and  $rotX, rotY, rotZ$  the rotation values of the TCP (in radians).
- In this example, the last three components (rotation values) remain unaffected in the case of calculated poses.

**Note**

Depending on the application, we generally recommend only altering  $rotZ$  to prevent unintended side effects; *VeriSens*® also only provides this value.

- $pos\_WP$  serves as a temporarily used variable for storing a position

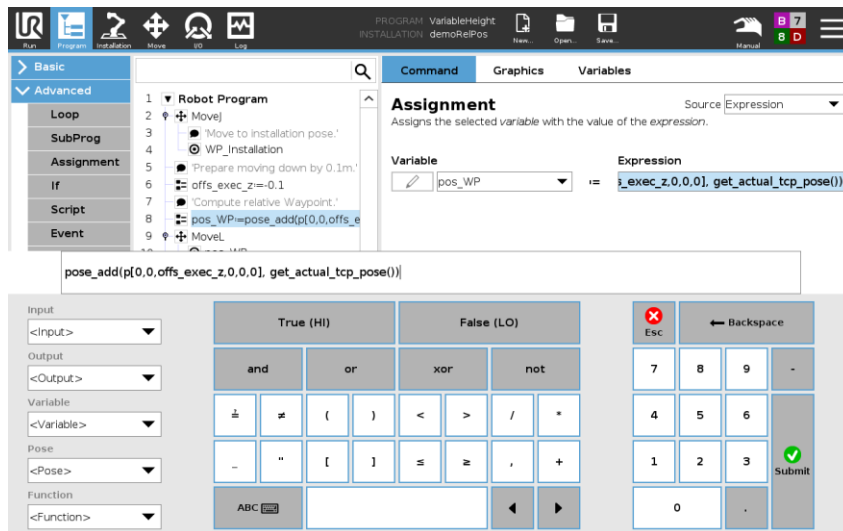


Figure 7

10

→ Basic → Waypoint

Move to the waypoint whose position corresponds to the value of the variable  $pos\_WP$ .

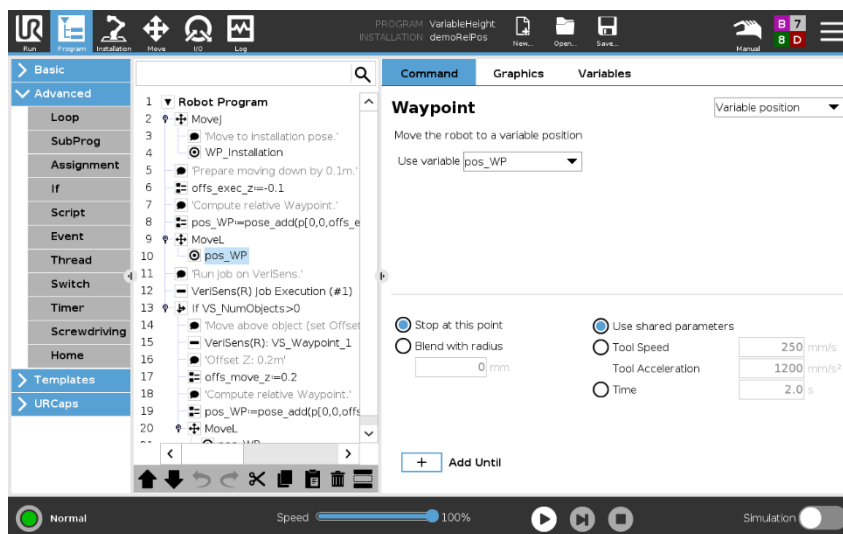
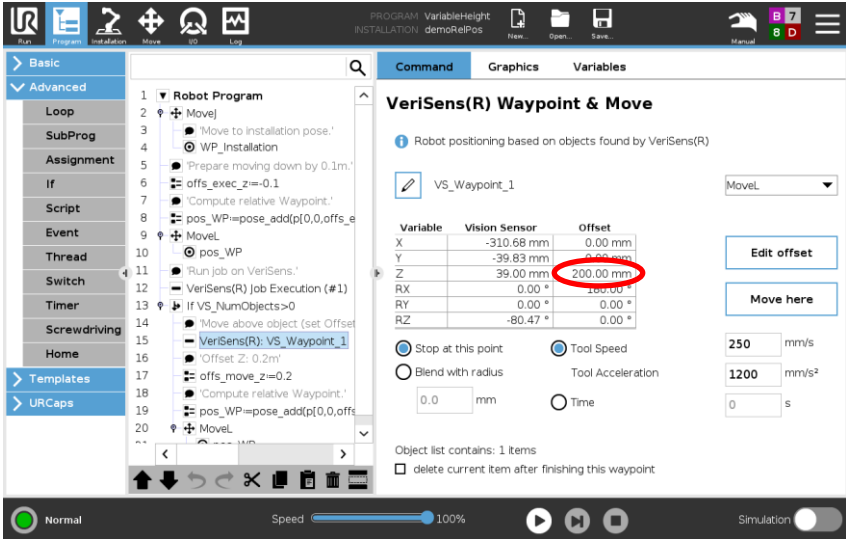


Figure 8



12	<p>→ URCap → VeriSens(R) Job Execution</p> <p>Execute the selected job on <i>VeriSens</i>®.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>Note</b></p> <p>In this example, <i>VeriSens</i>® is located 100 mm lower than normal. <i>VeriSens</i>® is not informed of this changed image capturing position and continues to provide the Z height set in the job in the <i>Application Suite</i> as the Z coordinate.</p> </div>
13	<p>→ Advanced → If</p> <p>Execute the following subprogram only if at least one object has been found.</p>
15	<p>→ URCap → VeriSens(R) Waypoint &amp; Move</p> <p>Move to the object position provided by <i>VeriSens</i>®, which is manually adjusted in the Z direction to prevent collisions.</p> <div data-bbox="316 815 1166 1352" style="border: 1px solid black; padding: 5px; margin: 10px 0;">  </div> <p>Figure 9</p> <p>In the example, this manual adjustment is 200 mm. This means that the robot moves 200 mm above the Z position specified in the job in the <i>Application Suite</i>. The value of this height adjustment must be selected so that the robot moves to a position above the object in all instances.</p> <p>Especially when <code>offs_exec_z</code> is &gt; 0, this manual adjustment should be a little larger than the value of <code>offs_exec_z</code>.</p>
17	<p>→ Advanced → Assignment</p> <p>Set the variable <code>offs_move_z</code> to the same value which was previously manually set as the offset in the “<i>VeriSens</i>® Waypoint &amp; Move” node (line 13) (in this example 0.2 m).</p>
19	<p>→ Advanced → Assignment</p> <p>2nd height adjustment: Set variable <code>pos_WP</code> to a calculated value. This is calculated with <code>pose_add()</code> as the sum of the following two values:</p> <ul style="list-style-type: none"> <li>- <code>p[0, 0, offs_exec_z - offs_move_z - 0.05, 0, 0, 0]</code> ... a relative shift created in the Z direction, which is composed of three Z components:             <ul style="list-style-type: none"> <li>o <code>offs_exec_z</code> ... takes into account the vertical shift during the job execution</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>o <code>offs_move_z</code> ... compensates the vertical shift of the approached waypoint above the object</li> <li>o <code>-0.05</code> ... manual adjustment: e.g. for a lower grabbing position (in this example shifted by 50 mm in the neg. Z direction)</li> </ul> <p>- <code>get_actual_tcp_pose()</code> ... the current position of the robot</p>
21	<p>→ Basic → Waypoint</p> <p>Move to the waypoint whose position corresponds to the value of the variable <code>pos_WP</code>.</p>

### 3.6 Programming example 2 – four screws depending on a variable position

#### 3.6.1 Description of the application

In the second example (Figure 10), four screws are to be tightened in fixed positions from the center based on the position detected in the middle by *VeriSens*<sup>®</sup>, using an electric screwdriver guided by a robot. In the captured image, Z is equivalent to the nominal height of the coordinate calibration. To “get” the position, the robot moves to a higher z. An adjustment is then made in z as well as x and y for the tightening position in each corner.

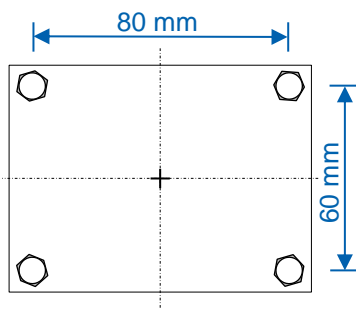


Figure 10

### 3.6.2 Creating the variables

The following variables are used by the program and must therefore be created in advance. The initially determined values are adjusted while the program is running.

`offs_corner_x`

X offset of the tightening position of a corner in relation to the center of the object

`offs_corner_y`

Y offset of the tightening position of a corner in relation to the center of the object

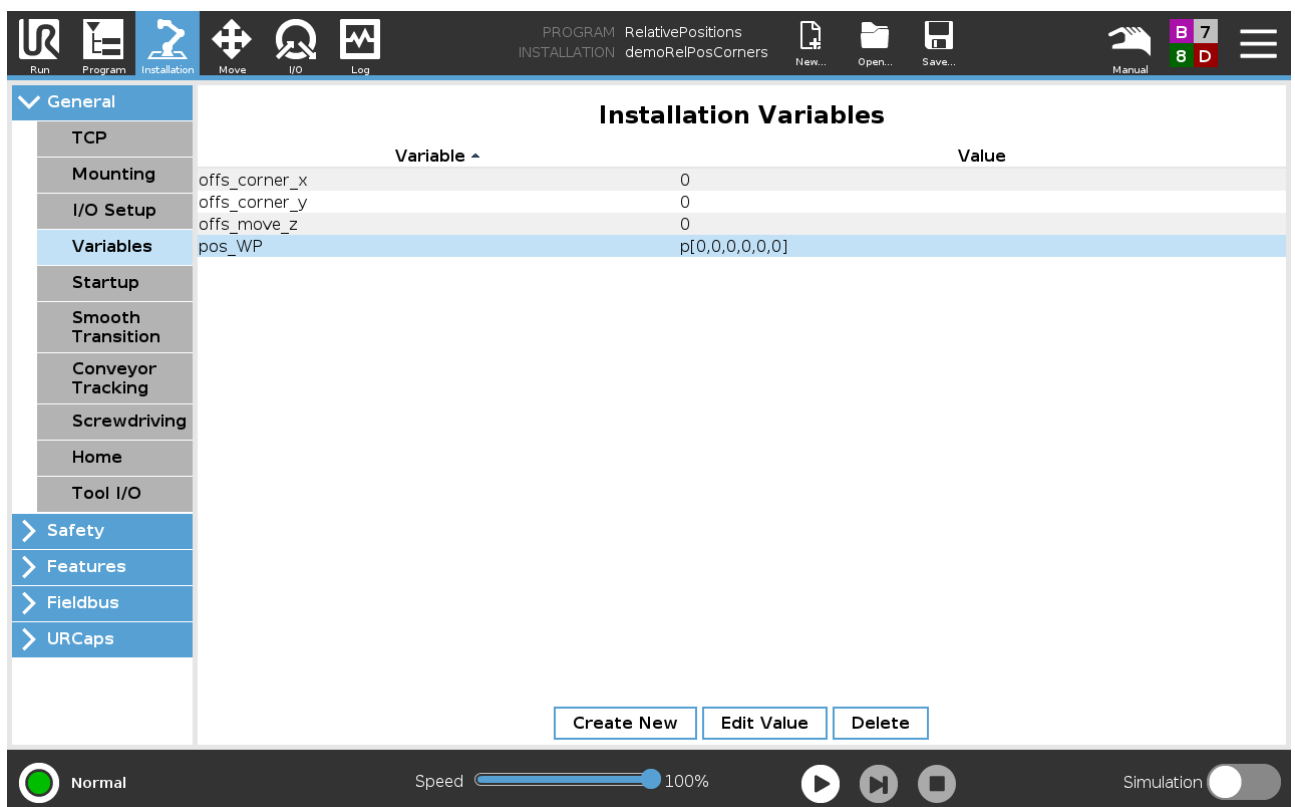
`offs_move_z`

Vertical shift that is set manually in the *VeriSens*<sup>®</sup> waypoint

`pos_WP`

Calculated position of a waypoint that is to be approached in the program.

Syntax: `p[x, y, z, rotX, rotY, rotZ]`



The screenshot shows the 'Installation Variables' window in the VeriSens software. The window has a menu bar at the top with icons for Run, Program, Installation, Move, I/O, and Log. Below the menu bar, the current program is identified as 'RelativePositions' and the installation as 'demoRelPosCorners'. The main area contains a table with the following data:

Variable	Value
<code>offs_corner_x</code>	0
<code>offs_corner_y</code>	0
<code>offs_move_z</code>	0
<code>pos_WP</code>	<code>p[0,0,0,0,0,0]</code>

At the bottom of the table, there are three buttons: 'Create New', 'Edit Value', and 'Delete'. The interface also includes a status bar at the bottom with a 'Normal' indicator, a 'Speed' slider set to 100%, and a 'Simulation' toggle switch.

Figure 11

### 3.6.3 Creating the program

Robot programming can now commence (Figure 11).

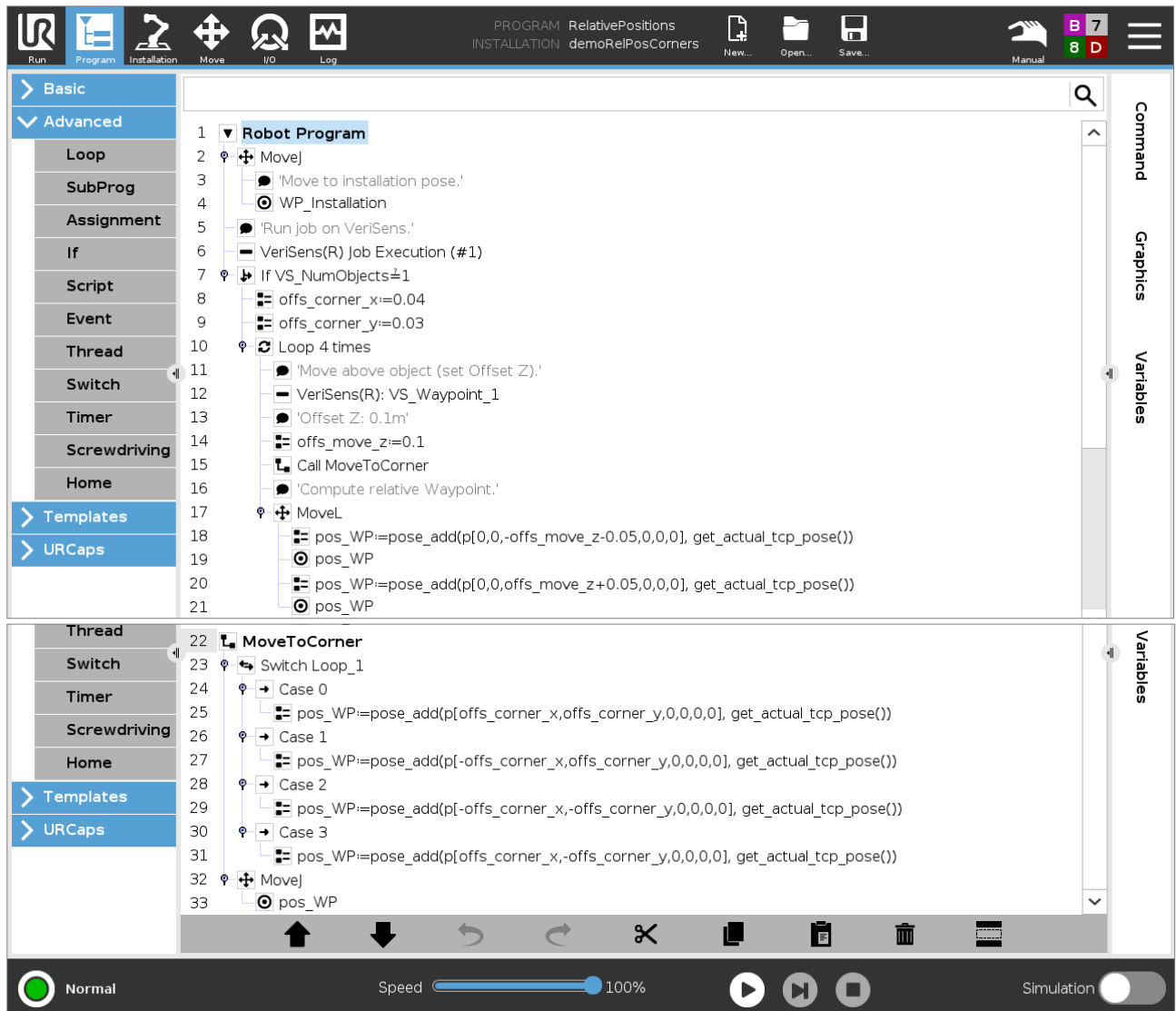
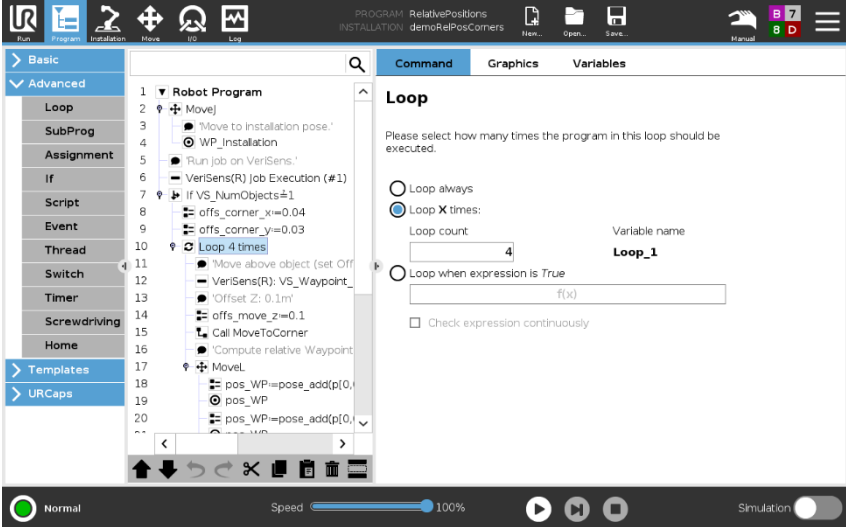


Figure 12

#### Program description

Line	Explanation
4	→ Basic → Waypoint  Move to a waypoint that either corresponds to the pose of the vision sensor during installation or a pose with the same height $z$ only as during the installation.
6	→ URcap → VeriSens(R) Job Execution  Execute the selected job on <i>VeriSens</i> <sup>®</sup> .
7	→ Advanced → If  Execute the following subprogram only if one object has been found.

8	<p>→ Advanced → Assignment</p> <p>Set the variable <code>offs_corner_x</code> to a fixed value.</p> <p><u>Example:</u> <i>VeriSens</i><sup>®</sup> provides the center of the object as the object position. The value <code>offs_corner_x</code> is set to half the distance between the tightening positions in the X direction, as a tightening position is found at this distance to the center of the object.</p>
9	<p>→ Advanced → Assignment</p> <p>Set the variable <code>offs_corner_y</code> to a fixed value.</p> <p><u>Example:</u> <i>VeriSens</i><sup>®</sup> provides the center of the object as the object position. The value <code>offs_corner_y</code> is set to half the distance between the tightening positions in the Y direction, as a tightening position is found at this distance to the center of the object.</p>
10	<p>→ Advanced → Loop</p> <p>Introduce a loop to move the robot to the 4 tightening positions in sequence.</p>  <p>Figure 13</p>
12	<p>→ URCap → VeriSens(R) Waypoint &amp; Move</p> <p>Move to the object position provided by <i>VeriSens</i><sup>®</sup>, which is manually adjusted in the Z direction to prevent collisions.</p>

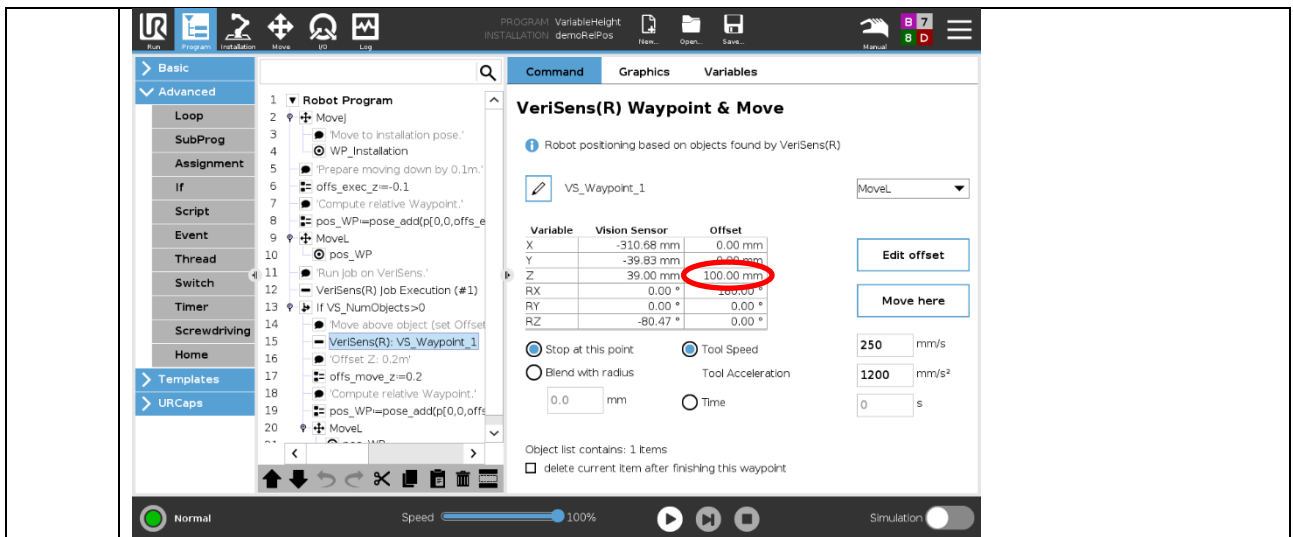


Figure 14

In the example, this manual adjustment is 100 mm. This means that the robot moves 100 mm above the Z position set in the job in the *Application Suite*. The value of this height adjustment must be selected so that the robot moves to a position above the object in all instances.

14 → Advanced → Assignment

Set the variable `offs_move_z` to the same value that was previously set manually as the offset in the “VeriSens® Waypoint & Move” node (line 12) (in this example, 0.1 m).

15 → Advanced → SubProg

Execute the “MoveToCorner” subprogram (see from line 22), which will cause the robot to move to a position above one of the four tightening positions, depending on the loop index.

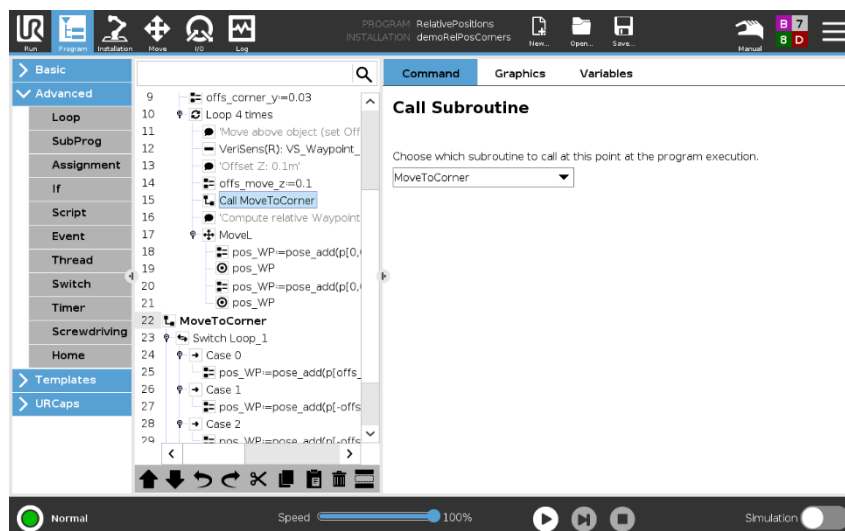


Figure 15

18 → Advanced → Assignment

Set variable `pos_WP` to a calculated value. This is calculated with `pose_add()` as the sum of the following two values:

- `p[0, 0, offs_exec_z - offs_move_z - 0.05, 0, 0, 0]` ... a relative shift created in the Z direction, which is composed of two Z components:

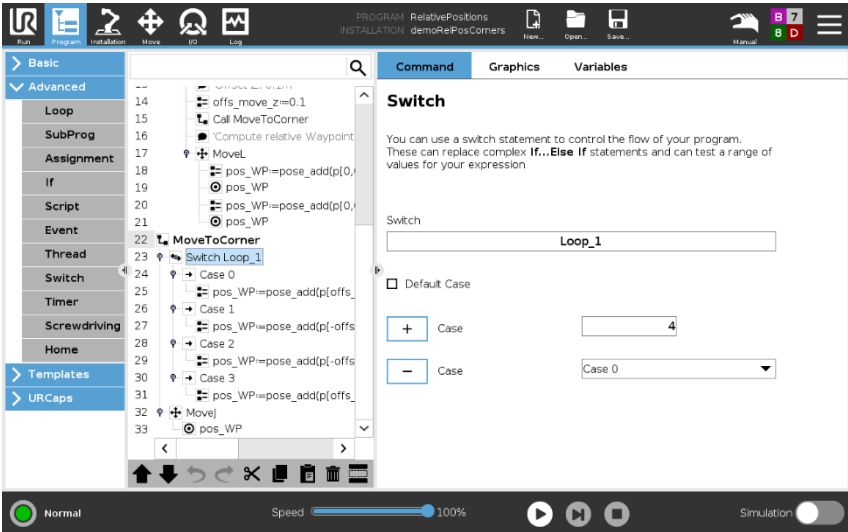
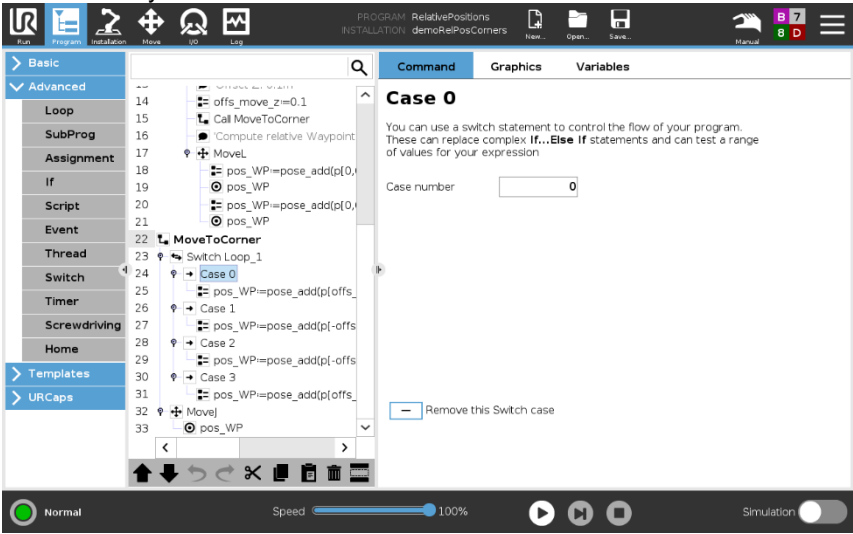
	<ul style="list-style-type: none"> <li>○ -offs_move_z ... compensates the vertical shift of the approached waypoint above the object</li> <li>○ -0.05 ... manual adjustment: e.g. for a lower tightening position (in this example, shifted by 50 mm in the neg. Z direction)</li> <li>• get_actual_tcp_pose() ... the current position of the robot</li> </ul>
19	<p>→ Basic → Waypoint</p> <p>Move to the waypoint whose position corresponds to the value of the variable <code>pos_WP</code>.</p> <div style="border: 1px solid black; background-color: #0056b3; color: white; padding: 2px;"><b>Note</b></div> <div style="border: 1px solid black; padding: 2px;">The tightening process could take place now.</div>
20	<p>→ Advanced → Assignment</p> <p>Set variable <code>pos_WP</code> to a calculated value. This reverses the vertical position change from line 18.</p>
21	<p>→ Basic → Waypoint</p> <p>Move to the waypoint whose position corresponds to the value of the variable <code>pos_WP</code>.</p> <div style="border: 1px solid black; background-color: #0056b3; color: white; padding: 2px;"><b>Note</b></div> <div style="border: 1px solid black; padding: 2px;">Now the robot is at a distance above the object again.</div>
22	<p>Start of the description of the “MoveToCorner” subprogram</p>
23	<p>→ Advanced → Switch</p> <p>Addition of 4 cases within a switch command. This way, each tightening position can be treated separately within a case command.</p>  <p>The screenshot shows the Universal Robots programming environment. The left sidebar displays a tree view of the program structure, with the 'MoveToCorner' subprogram expanded to show a 'Switch' block containing four cases (Case 0 to Case 3). Each case includes a 'MoveL' command with a specific pose. The right pane shows the configuration for the 'Switch' block, with 'Loop_1' selected as the switch expression and 'Default Case' unchecked. The bottom status bar indicates the robot is in 'Normal' mode and the simulation is running at 100% speed.</p>

Figure 16

24, 26, 28, 30	<p>Automatically inserted case command within which additional nodes can be added.</p>  <p>Figure 17</p>
25, 27, 29, 31	<p>→ Advanced → Assignment</p> <p>Set variable <code>pos_WP</code> to a calculated value. This is calculated with <code>pose_add()</code> as the sum of the following two values:</p> <ul style="list-style-type: none"> <li>• <code>p[±offs_corner_x, ±offs_corner_y, 0, 0, 0, 0]</code> ... a relative offset created in the X and Y direction, to subsequently (line 33) move the robot correctly over a tightening position each time.</li> <li>• <code>get_actual_tcp_pose()</code> ... the current position of the robot</li> </ul>
33	<p>→ Basic → Waypoint</p> <p>Move to the waypoint whose position corresponds to the value of the variable <code>pos_WP</code>.</p> <div style="background-color: #0056b3; color: white; padding: 5px;"><b>Note</b></div> <p>The robot is now over a tightening position depending on the respective case. The program execution continues at the end of this subprogram in line 16.</p>



## 4 Summary / special cases

It is possible to inspect or detect the position of objects with different heights and even control subsequent actions in this way.

The methods vary and depend on the application. While option 1 is distinguished by its simplicity and can be implemented quickly even by beginners, option 2 offers complete freedom of action to experienced programmers.

## 5 Downloads

Additional information can be found in the documentation for *VeriSens*<sup>®</sup> Vision sensors, specifically the section on Universal Robots.

[Product Finder](#)

## 6 Support

Please contact our Technical & Application Support Center with any questions.

### Worldwide

#### **Baumer Optronic GmbH**

Badstrasse 30 · DE-01454 Radeberg  
Deutschland

Phone +49 3528 4386 845

[support.cameras@baumer.com](mailto:support.cameras@baumer.com)

## 7 Legal information

All product and company names mentioned are trademarks or registered trademarks of their respective owners.

All rights reserved. Reproduction of this document in whole or in part is only permitted with previous written consent from Baumer Optronic GmbH.

Revisions in the course of technical progress and errors reserved.

**Baumer Group**

The Baumer Group is one of the worldwide leading manufacturers of sensors, encoders, measuring instruments and components for automated image processing. Baumer combines innovative technologies and customer-oriented service into intelligent solutions for factory and process automation and offers an unrivalled wide technology and product portfolio. With around 2,700 employees and 39 subsidiaries in 19 countries, the family-owned group of companies is always close to the customer. Baumer provides clients in most diverse industries with vital benefits and measurable added value by worldwide consistent high quality standards and outstanding innovative potential. Learn more at [www.baumer.com](http://www.baumer.com) on the internet.

**Baumer Optronic GmbH**

Badstrasse 30 · DE-01454 Radeberg  
Phone +49 3528 4386 0 · Fax +49 3528 4386 86  
[sales.cc-vt@baumer.com](mailto:sales.cc-vt@baumer.com) · [www.baumer.com](http://www.baumer.com)