UNIVERSAL ROBOTS Support

MODIFY ROBOT TRAJECTORY BY OVERLAYING CUSTOM MOTION

Explains how to use Online Path Offset to modify basic robot movements. Online Path Offset is a feature released in PolyScope 5.6.0

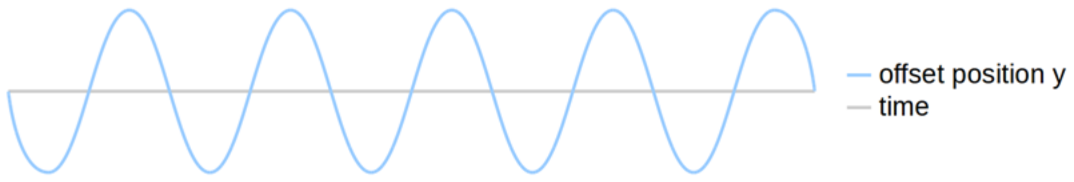
Last modified on Dec 20, 2022

Examples are valid for:
e-Series Software version 5.6.0.90886
Note that newer software may behave differently.

The URScript function "path_offset_set()" allows runtime modifications to the motions produced by a robot program. This can be useful in situations where the robot is needed to move in some predefined repeating pattern e.g. while welding or dispensing glue. For more advanced use, the robot motion can in principle be fully controlled by feeding the controller offsets from an external source.

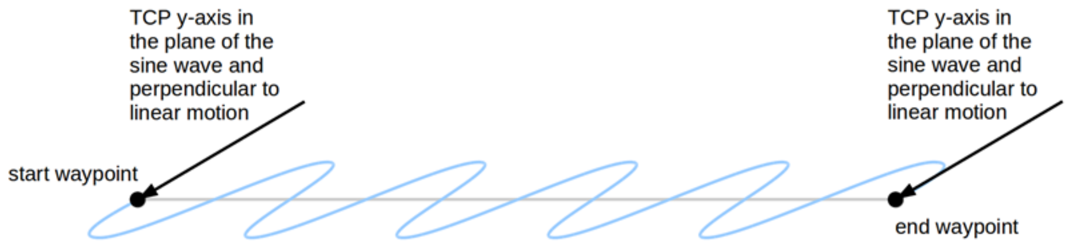
SINE WEAVING PATTERN

This example illustrates how to create a linear robot motion progressing at a fixed rate of 8 mm/s with an overlayed sine wave motion oscillating at 3 Hz with an amplitude of 6 mm (values can be replaced).



Previously, this scenario would require teaching potentially hundreds of waypoints and blending between them to produce an approximation of the desired motion. Using the “path_offset_”-script commands, this can be done in a more general and far less time-consuming fashion.

Firstly, the underlying motion must be defined. In this example, this will be a MoveP between two waypoints which will constitute the main robot program. The move can be defined using the PolyScope waypoint and move nodes. The waypoints in this example are taught in such a way that the TCP y-axis is perpendicular to the direction of the motion at either endpoint and lies in the plane is which the sine wave must be placed.


TCP y-axis in the plane of the sine wave and perpendicular to linear motion
TCP y-axis in the plane of the sine wave and perpendicular to linear motion
start waypoint
end waypoint

Having defined the underlying motion, a separate thread must be created for the calculation and application of the path offset. It will need to perform a few calculations based on the motion parameters. The e-Series controller runs at 500 Hz and the MoveP must have an overlayed motion that is a sine wave oscillating at 3 Hz and have an amplitude of 6 mm.

```
ctrl_frequency = 500.0
```

```
weave_frequency = 3.0
```

```
amplitude = 0.006
```

Given these parameters, it is possible to calculate the sine wave cycle duration in controller time steps.

```
cycle_duration = ctrl_frequency / weave_frequency
```

Having these parameters set up, the next step is to specify the initial state of the offset along with a counter for the number of controller cycles that have passed. In order to activate the dynamic path offset, the URScript function call "path_offset_enable()" must be added.

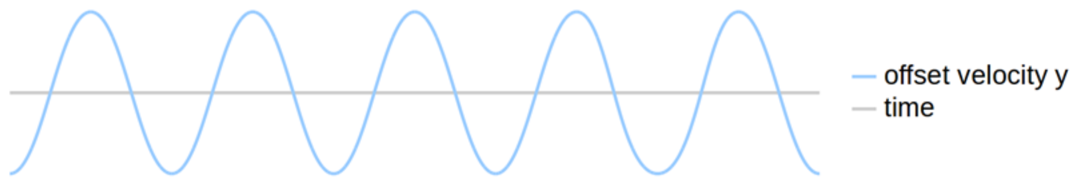
```
offset = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
i = 0
path_offset_enable()
```

The only thing remaining is to add a loop which calculates the offset at the given controller cycle, sets the offset via the URScript function "path_offset_set()" and increments "i".

```
Loop
    offset[1] = sin((i / cycle_duration) * 2.0 * 3.14159) * amplitude
    path_offset_set(offset, 2)
    i = i + 1
    sync()
end (Loop)
```

offset[1] corresponds to a positional offset along the y-axis and the second parameter (2) for "path_offset_set()" specifies that the offset must be interpreted to be in the TCP coordinate system. The "sync()" function call means that the next iteration of the loop will begin in the next controller time step.

However, attempting to run this program will result in a runtime exception at the very beginning. While sine is a continuous function with continuous derivative (corresponding to continuous position and velocity profiles for the offset), the offset at the very first time step jumps from adding zero velocity, to adding a fairly large velocity (and is thus discontinuous at the first time step).



It is a requirement, that the applied offset is smooth, and does not force jumps in the robot velocity profile. Making a large jump in velocity between one time step and the next, would require accelerations that the robot is unable to produce.

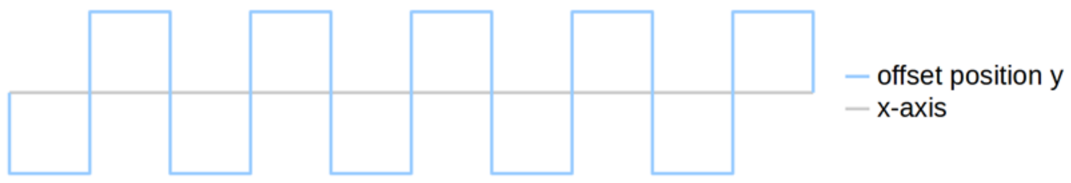
A discontinuous velocity profile can be remedied by smoothly ramping into the sine wave. An easier approach, is to use the URScript function "path_offset_set_alpha_filter()". Calling this function with a floating-point parameter between 0.0 and 1.0 will add an exponentially decaying filter to the specified offset which smoothens the velocity profile for the offset. For this example created on a UR10, a value of 0.27 is sufficient and results in a trajectory practically indistinguishable from the non-filtered version. The necessary alpha value will depend on robot calibration, robot mounting, payload mass, payload center of gravity, TCP offset, robot position in workspace, path offset rate of change and underlying motion.
A filter constant of 0.1 results in a filtration, where the filtered offset is updated to be 90 % of its previous value plus 10 % of the new offset. Given that the value is updated every 2 ms, leaving the offset constant will lead to more than 90 % convergence in 44 ms.

```
alpha = 0.27
path_offset_Set_alpha_filter(alpha)
```

See the files "path_offset_sine_weave_UR10" at the bottom of this page, to retrieve the example files.

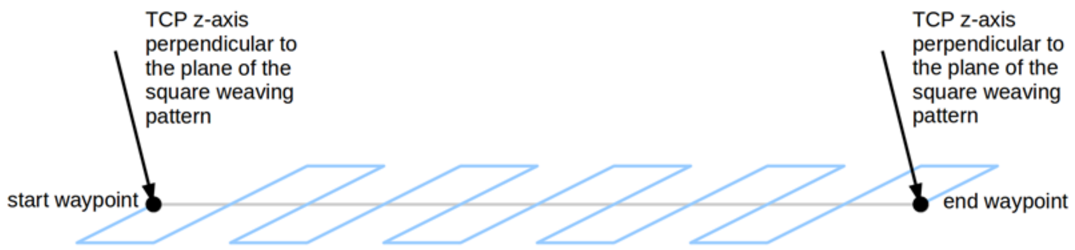
SQUARE WEAVE PATTERN

This example illustrates how to create a linear robot motion progressing at a fixed rate of 8 mm/s with an overlayed so called square weaving pattern oscillating at 3 Hz with an amplitude of 6 mm (values can be replaced).

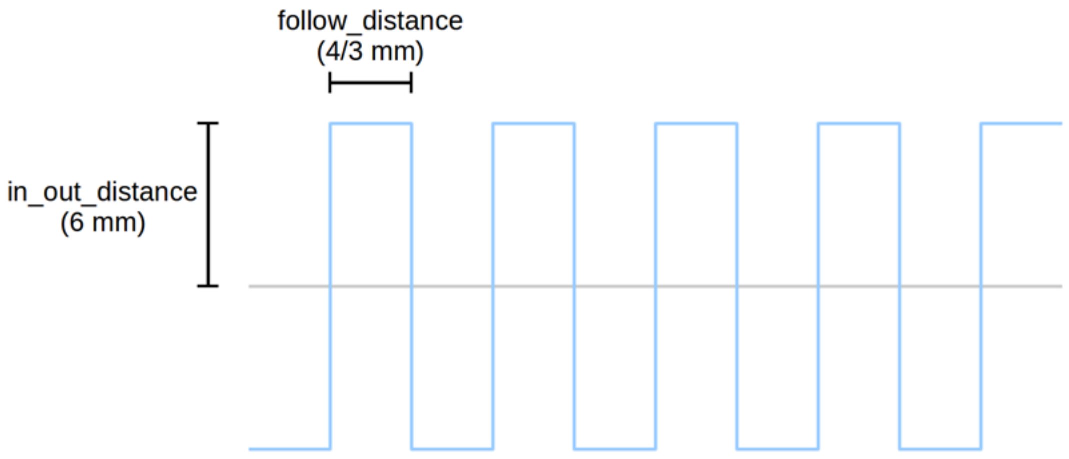


Previously, this scenario could be realized by either calculating or teaching a waypoint for each corner of the weave. Using the Online Path Offset feature, this can be done in a more general and far less time-consuming fashion.

Firstly, the underlying motion must be defined. In this example, this will be a MoveP between two waypoints which will constitute the main robot program. The move can be defined using the PolyScope waypoint and move nodes. The waypoints are taught in such a way, that the TCP z-axis is perpendicular to the plane is which the square weave pattern must be placed.


TCP z-axis perpendicular to the plane of the square weaving pattern
TCP z-axis perpendicular to the plane of the square weaving pattern
start waypoint
end waypoint

Having defined the underlying motion, a separate thread needs to be created for the calculation and application of the path offset. It will be needed for performing a few calculations based on the motion parameters. The e-Series controller runs at 500 Hz and the MoveP must have an overlayed motion that is a square weaving pattern oscillating at 3 Hz and have an amplitude of 6 mm.



```
ctrl_frequency = 500.0
```

```
speed = 0.008
```

```
weave_frequency = 3.0
```

```
amplitude = 0.006
```

Given these parameters, it is possible to calculate the square weaving pattern cycle duration in controller time steps. This requires the calculation of the duration of the different phases of the movement along with the offset to be added at each time step, in order to get a constant velocity of the TCP.

```
in_out_distance = amplitude
```

```
follow_distance = (speed / weave_frequency) / 2.0
```

```
cycle_distance = (4.0 * in_out_distance) + (2.0 * follow_distance)
```

```
in_out_duration = floor((ctrl_frequency / weave_frequency) * (in_out_distance / cycle_distance))
follow_duration = floor((ctrl_frequency / weave_frequency) * (follow_distance / cycle_distance))
cycle_duration = (4 * in_out_duration) + (2 * follow_duration)
in_out_addition = in_out_distance / in_out_duration
follow_mult = 2.0 * (in_out_duration / follow_duration)
```

Having these parameters set up, it is possible to specify the initial state of the offset along with a counter for the number of controller cycles that has passed. In order to activate the Online Path Offset, a call to the URScript function "path_offset_enable()" is added.

```
offset = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
i = 0
path_offset_enable()
```

Lastly, a loop which calculates the offset at the given controller cycle and sets the offset via the URScript function "path_offset_set()" and increments "i" is needed. This can be done using a set of if statements because the different parts of the cycle require different addition to the path offset. Note that in order to offset the movement along the MoveP, it is possible to use the utility URScript function "get_target_tcp_speed_along_path()", which as the name suggests returns the velocity of the TCP, had it not been modified by conveyor tracking or Online Path Offset.

```
Loop
    traj_vel = get_target_tcp_speed_along_path()
    traj_vel_pos = [traj_vel[0], traj_vel[1], traj_vel[2]]
    if i < in_out_duration
    (the first part of the cycle moving away from the seam)
        offset[0] = offset[0] - (norm(traj_vel_pos) / ctrl_frequency)
        offset[1] = offset[1] + in_out_addition
    elseif i < (in_out_duration + follow_duration)
    (The first part of the cycle moving parallel to the seam)
        offset[0] = offset[0] + (follow_mult * (norm(traj_vel_pos) / ctrl_frequency))
    elseif i < ((3 * in_out_duration) + follow_duration)
    (The movement from one extreme, across the seam and towards the other extreme)
        offset[0] = offset[0] - (norm(traj_vel_pos) / ctrl_frequency)
        offset[1] = offset[1] - in_out_addition
    elseif i < ((3 * in_out_duration) + (2 * follow_duration))
    (the last part of the cycle moving parallel to the seam)
        offset[0] = offset[0] + (follow_mult * (norm(traj_vel_pos) / ctrl_frequency))
    else
    (the last part of the cycle moving back to the seam)
        offset[0] = offset[0] - (norm(traj_vel_pos) / ctrl_frequency)
        offset[1] = offset[1] + in_out_addition
    path_offset_set(offset, 3)
    i = i + 1
    if i >= cycle_duration
```

```
        i = 0
    sync()
end Loop
```

offset[1] corresponds to a positional offset along the y-axis and the second parameter for "path_offset_set()" (3) specifies that the offset must be interpreted to be in the motion coordinate system.

Excerpt from script manual:

3: (MOTION) Use a coordinate system following the un-offset trajectory when applying. This coordinate system is defined as follows. X-axis along the tangent of the translational part of the un-offset trajectory (rotation not relevant here). Y-axis perpendicular to the X-axis above and the Z-axis of the tool (Z cross X). Z-axis given from the X and Y axes by observing the right-hand rule. This is useful for instance for superimposing a weaving pattern onto the trajectory when welding.

Since the offset is specified piecewise during the cycle, the counting variable "i" is reset after each cycle. The "sync()" function call means that the next iteration of the loop will begin in the next controller time step.

The generated movement has a lot of discontinuities in the velocity profile both along the x-axis and the y-axis. Therefore, some filtering is necessary to perform an approximated motion. The unfiltered motion would require the TCP to move at a constant speed of 0.08m/s but with velocity alternating between a movement purely along the x-axis or purely along the y-axis, requiring instant velocity change at each switch.

The URScript function "path_offset_set_alpha_filter()" can be used to smoothen the offset velocity profile. Since the pattern frequency and amplitude are relatively high, for this example created on a UR10, an alpha filter constant of 0.165 is needed to be within acceleration limits. Calling "path_offset_set_alpha_filter()" will add an exponentially decaying filter to the specified offset. The necessary alpha value will depend on robot calibration, robot mounting, payload mass, payload center of gravity, TCP offset, robot position in workspace, path offset rate of change and underlying motion.

```
alpha = 0.165
path_offset_set_alpha_filter(alpha)
```

See the files "path_offset_square_weave_UR10" at the bottom of this page, to retrieve the example files.

GENERAL OBSERVATIONS

In cases where the applied offset has continuous position and velocity profile except for the first time step, it may be desirable to initially have some filter applied, but ramping the filter value towards 1.0 (corresponding to no filter). This may in a lot of cases be done in less than one second.

As a rule of thumb, the interval of valid alpha filter constants can be split into the following parts:

- A value of 1.0 corresponds to disabling the filter entirely. This is the default setting.
- Values larger than 0.25 are unlikely to smoothen a jagged pattern enough to make the robot able to perform the desired motion.
- It is likely that some value between 0.1 and 0.25 smoothens the pattern enough to make the robot able to follow it while still approximating the desired motion sufficiently for the task at hand.
- Values less than 0.1 smoothen the pattern considerably, and may in some cases distort the pattern to such a degree that it becomes unusable.
- A value of 0.0 corresponds to ignoring changes to the specified offset altogether.