API quickstart

All the examples shown here are for **LARA5**; and for other robots please refer to Examples section.

Robot object initialization

When we instantiate a robot object, static properties of the robot can be accessed as the attributes of the robot object.

```
from neurapy.robot import Robot
r = Robot()
print(r.robot_name)
print(r.dof)
print(r.platform)
print(r.payload)
print(r.kURL)
print(r.robot_urdf_path)
print(r.current_tool)
print(r.connection)
print(r.version)
```

Note : Robot motion is possible only in Automatic mode and stop function needs to be called at the end of every script for a proper termination. Please refer to https://neurarobotics.atlassian.net/wiki/spaces/SAppEng/pages/edit-v2/305594923#Set-Mode and https://neurarobotics.atlassian.net/wiki/spaces/SAppEng/pages/edit-v2/305594923#Pause/Unpause/Stop functionalities for more information.

Move Joint

move_joint method from robot object is used to move the robot to specified joint configuration in joint space. This method takes the following arguments/keyword arguments

Input Parameters:

target_joint	• type : Keyword Argument • required: Yes • description: List of joint configurations • data_type: List of Joint Values - float • units: radians • default_value: N/A
speed	• type: Keyword Argument • required: No • description: Angular Speed • data_type: float • units: % of maximum angular speed • default_value: 0.25
acceleration	• type: Keyword Argument • required: No • description: Angular Acceleration • data_type: float • units: % of maximum angular acceleration • default_value: 0.25
current_joint_angles	• type: Keyword Argument • required: No • description: Current Robot Joint Configuration • data_type: List of Joint Values - float • units: radians • default_value: Joint Configuration obtained form Robot Status method

safety_toggle	• type: Keyword Argument
	• required: No
	• description: Safety toggle
	• data_type: Bool - True/False
	• units: N/A
	• default_value: if not set, value of the safety toggle in Program screen will be used.
	  • If set to True, Max speed is slashed to 25%
	  • False - No reduction in already set max speed

only_send	• type: Keyword Argument • required: No • description: only to plan the motion (This can be used when planning and execution needs to be separated, this needs to be used in conjunction with executor function) • data_type: Bool - True/False • units: N/A
non_blocking	• type: Keyword Argument • required: No • description: When non_blocking is set to true, then one can communicate with the robot and move_joint function is non-blocking. • data_type: Bool - True/False • units: N/A

Return Values:

True	If motion is executed successfully
False	If motion is not executed successfully

Example: (target_joint values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
joint_property = {
            "speed": 50.0,
            "acceleration": 50.0,
        "safety_toggle": True,
            "target_joint": [
                [
                    2.5995838308821924,
                    0.24962416292345468,
                    -1.8654403327490414,
                    0.04503286318691005,
                    -1.1740563715454926,
                    0.10337461241185522
                ],
                [
                    2.1372059994827075,
                    0.24939733788589463,
                    -1.8651270179353125,
                    0.044771940725327274,
                    -1.173860821592129,
                    0.10315646291502645
                ],
                [
                    1.9180047887810003,
                    -0.24855170101601043,
                    -1.3680228668892351,
                    0.12404421791100637,
                    -1.1914147150222498,
                    -0.13255713717112075
                ]
            ],
        "current_joint_angles":r.robot_status("jointAngles")
        }
r.move_joint(**joint_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Example: (set only_send to True)

```
from neurapy.robot import Robot
r = Robot()
joint_property = {
            "speed": 50.0,
            "acceleration": 50.0,
        "safety_toggle": True,
            "target_joint": [
                [
                    2.5995838308821924,
                    0.24962416292345468,
                    -1.8654403327490414,
                    0.04503286318691005,
                    -1.1740563715454926,
                    0.10337461241185522
                ],
                [
                    2.1372059994827075,
                    0.24939733788589463,
                    -1.8651270179353125,
                    0.044771940725327274,
                    -1.173860821592129,
                    0.10315646291502645
                ],
                [
                    1.9180047887810003,
                    -0.24855170101601043,
                    -1.3680228668892351,
                    0.12404421791100637,
                    -1.1914147150222498,
                    -0.13255713717112075
                ]
            ],
        "current_joint_angles":r.robot_status("jointAngles")
        }
r.move_joint(**joint_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Example: (set non_blocking to True)

```
from neurapy.robot import Robot
r = Robot()
joint_property = {
            "speed": 50.0,
            "acceleration": 50.0,
                "non_blocking": True,
        "safety_toggle": True,
            "target_joint": [
                [
                    2.5995838308821924,
                    0.24962416292345468,
                    -1.8654403327490414,
                    0.04503286318691005,
                    -1.1740563715454926,
                    0.10337461241185522
                ],
                [
                    2.1372059994827075,
                    0.24939733788589463,
                    -1.8651270179353125,
                    0.044771940725327274,
                    -1.173860821592129,
                    0.10315646291502645
                ],
                [
                    1.9180047887810003,
                    -0.24855170101601043,
                    -1.3680228668892351,
                    0.12404421791100637,
                    -1.1914147150222498,
                    -0.13255713717112075
                ]
            ],
        "current_joint_angles":r.robot_status("jointAngles")
        }
r.move_joint(**joint_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Move Linear

move_linear method is used to move the robot to specified poses in Cartesian/Task space.

Input Parameters:

<table>
<tr><td>target_pose</td><td>type : Keyword Argument
required: Yes
description: List of pose configurations(minimum two poses are required,i.e starting and ending pose)
data_type: Pose configuration - [X,Y,Z,A,B,C], float
units: Position values in meters and rotation values in radians
default_value: N/A</td></tr>
<tr><td>target_joint</td><td>type : Keyword Argument
required: No
description: List of joint configurations corresponding to the given target poses
data_type: List of Joint Values - float
units: radians
default_value: N/A</td></tr>
<tr><td>speed</td><td>type: Keyword Argument
required: No
description: Linear Speed
data_type: float
units: m/sec
default_value: 0.25</td></tr>
<tr><td>acceleration</td><td>type: Keyword Argument
required: No
description: Linear Acceleration
data_type: float
units: m/sec2
default_value: 0.25</td></tr>
<tr><td>blend_radius</td><td>type: Keyword Argument
required: No
description: Value of the blend radius ,If blending is needed between two segments of motion.
data_type: float
units: meters
default: 0</td></tr>
<tr><td>current_joint_angles</td><td>type: Keyword Argument
required: No
description: Current Robot Joint Configuration
data_type: List of Joint Values - float
units: radians
default_value: Joint Configuration obtained form Robot Status method</td></tr>
<tr><td>safety_toggle</td><td>type: Keyword Argument
required: No
description: Safety toggle
data_type: Bool - True/False
units: N/A
default_value: if not set, value of the safety toggle in Program screen will be used.
If set to True, Max speed is slashed to 25%
False - No reduction in already set max speed</td></tr>
</table>

weaving	• type: Keyword Argument
	• required: No
	• description: enables weaving modifier
	• data_type: Bool - True/False
	• units: N/A
	• default value: if not set, value of weaving is set to false.
	  ◦ True - Applies weaving modifier to path
	  ◦ False - No modification to the path
	• Additional Parameters
	  ◦ amplitude [m] or amplitude_left [m], amplitude_right [m]
	  ◦ frequency [Hz]
	  ◦ pattern - int
	    ▪ 1: Sine
	    ▪ 2: Trapezoidal
	    ▪ 3: Circle
	  ◦ dwell_time_left [s], dwell_time_right [s] (Trapezoidal only)
	  ◦ elevation [rad]
	  ◦ azimuth [rad]

input_reference_frame	• type: Keyword Argument • required: No • description: frame in which points were teached • data_type:[X,Y,Z,A,B,C] • units: Position values in meters and rotation values in radians • default value: if not set, robot base frame is considered
output_reference_frame	• type: Keyword Argument • required: No • description: frame in which points to be transformed • data_type:[X,Y,Z,A,B,C] • units: Position values in meters and rotation values in radians • default value: if not set, robot base frame is considered

Return Values:

True	If motion is executed successfully
False	If motion is not executed successfully

Example: (target_pose values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
linear_property = {
            "speed": 0.25,
            "acceleration": 0.1,
            "blend_radius": 0.005,
            "target_pose": [
                [
                    0.3287228886,
                    -0.1903355329,
                    0.4220780352,
                    0.08535207028439847,
                    -2.797181496822229,
                    2.4713321627410485
                ],
                [
                    0.2093363791501374,
                    -0.31711250784165884,
                    0.422149168855134,
                    -3.0565555095672607,
                    -0.3447442352771759,
                    -1.1323236227035522
                ],
                [
                    0.2090521916195534,
                    -0.5246753336643587,
                    0.4218773613553828,
                    -3.0569007396698,
                    -0.3448921740055084,
                    -1.1323626041412354
                ],
                [
                    0.3287228886,
                    -0.1903355329,
                    0.4220780352,
                    0.08535207028439847,
                    -2.797181496822229,
                    2.4713321627410485
                ]
            ],
        "current_joint_angles":r.robot_status("jointAngles")
        }
r.move_linear(**linear_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Example with Weaving: (target_pose values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
linear_property = {
            "speed": 0.008,
            "acceleration": 1.0,
            "blend_radius": 0.005,
            "target_pose": [
                [
                    0.3287228886,
                    -0.1903355329,
                    0.4220780352,
                    0.08535207028439847,
                    -2.797181496822229,
                    2.4713321627410485
                ],
                [
                    0.2093363791501374,
                    -0.31711250784165884,
                    0.422149168855134,
                    -3.0565555095672607,
                    -0.3447442352771759,
                    -1.1323236227035522
                ],
                [
                    0.2090521916195534,
                    -0.5246753336643587,
                    0.4218773613553828,
                    -3.0569007396698,
                    -0.3448921740055084,
                    -1.1323626041412354
                ],
                [
                    0.3287228886,
                    -0.1903355329,
                    0.4220780352,
                    0.08535207028439847,
                    -2.797181496822229,
                    2.4713321627410485
                ]
            ],
        "current_joint_angles":r.robot_status("jointAngles"),
        "weaving":True,
        "pattern": 1,
        "amplitude": 0.006,
        "amplitude_left": 0.0,
        "amplitude_right": 0.0,
        "frequency": 1.5,
        "dwell_time_left": 0.0,
        "dwell_time_right": 0.0,
        "elevation": 0.0,
```

```
        "azimuth": 0.0
        }
r.move_linear(**linear_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Move Linear From Current Position

move_linear_from_current_position method is used to move the robot to specified poses in Cartesian/Task space from current position

Input Parameters:

target_pose	• type : Keyword Argument • required: Yes • description: List of pose configurations(minimum two poses are required,i.e starting and ending pose) • data_type: Pose configuration - [X,Y,Z,A,B,C], float • units: Position values in meters and rotation values in radians • default_value: N/A
target_joint	• type : Keyword Argument • required: No • description: List of joint configurations corresponding to the given target poses • data_type: List of Joint Values - float • units: radians • default_value: N/A
speed	• type: Keyword Argument • required: No • description: Linear Speed • data_type: float • units: m/sec • default_value: 0.25
acceleration	• type: Keyword Argument • required: No • description: Linear Acceleration • data_type: float • units: m/sec2 • default_value: 0.25
blend_radius	• type: Keyword Argument • required: No • description: Value of the blend radius ,If blending is needed between two segments of motion. • data_type: float • units: meters • default: 0
current_joint_angles	• type: Keyword Argument • required: No • description: Current Robot Joint Configuration • data_type: List of Joint Values - float • units: radians • default_value: Joint Configuration obtained form Robot Status method

<table>
<tr><td>safety_toggle</td><td>type: Keyword Argument
required: No
description: Safety toggle
data_type: Bool - True/False
units: N/A
default_value: if not set, value of the safety toggle in Program screen will be used.
If set to True, Max speed is slashed to 25%
False - No reduction in already set max speed</td></tr>
<tr><td>weaving</td><td>type: Keyword Argument
required: No
description: enables weaving modifier
data_type: Bool - True/False
units: N/A
default value: if not set, value of weaving is set to false.
True - Applies weaving modifier to path
False - No modification to the path
Additional Parameters
amplitude [m] or amplitude_left [m], amplitude_right [m]
frequency [Hz]
pattern - int
1: Sine
2: Trapezoidal
3: Circle
dwell_time_left [s], dwell_time_right [s] (Trapezoidal only)
elevation [rad]
azimuth [rad]</td></tr>
<tr><td>input_reference_frame</td><td>type: Keyword Argument
required: No
description: frame in which points were teached
data_type:[X,Y,Z,A,B,C]
units: Position values in meters and rotation values in radians
default value: if not set, robot base frame is considered</td></tr>
<tr><td>output_reference_frame</td><td>type: Keyword Argument
required: No
description: frame in which points to be transformed
data_type:[X,Y,Z,A,B,C]
units: Position values in meters and rotation values in radians
default value: if not set, robot base frame is considered</td></tr>
</table>

Return Values:

True	If motion is executed successfully
False	If motion is not executed successfully

Example: (target_pose values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
linear_property = {
            "speed": 0.25,
            "acceleration": 0.1,
            "blend_radius": 0.005,
            "target_pose": [
                [
                    0.3287228886,
                    -0.1903355329,
                    0.4220780352,
                    0.08535207028439847,
                    -2.797181496822229,
                    2.4713321627410485
                ],
                [
                    0.2093363791501374,
                    -0.31711250784165884,
                    0.422149168855134,
                    -3.0565555095672607,
                    -0.3447442352771759,
                    -1.1323236227035522
                ],
            ],
        "current_joint_angles":r.robot_status("jointAngles")
        }
r.move_linear_from_current_position(**linear_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Move Circular

move_circular method is used to move the robot in circular path across the given poses

Input Parameters:

target_pose	• type : Keyword Argument • required: Yes • description: List of 3 pose configurations (starting, middle and end points) • data_type: Pose configuration - [X,Y,Z,A,B,C], float • units: Position values in meters and rotation values in radians • default_value: N/A
target_joint	• type : Keyword Argument • required: No • description: List of joint configurations corresponding to the given target poses • data_type: List of Joint Values - float • units: radians • default_value: N/A

<table>
<tr><td>speed</td><td>type: Keyword Argument
required: No
description: Linear Speed
data_type: float
units: m/sec
default_value: 0.25</td></tr>
<tr><td>acceleration</td><td>type: Keyword Argument
required: No
description: Linear Acceleration
data_type: float
units: m/sec2
default_value: 0.25</td></tr>
<tr><td>current_joint_angles</td><td>type: Keyword Argument
required: No
description: Current Robot Joint Configuration
data_type: List of Joint Values - float
units: radians
default_value: Joint Configuration obtained form Robot Status method</td></tr>
<tr><td>safety_toggle</td><td>type: Keyword Argument
required: No
description: Safety toggle
data_type: Bool - True/False
units: N/A
default_value: if not set, value of the safety toggle in Program screen will be used.
If set to True, Max speed is slashed to 25%
False - No reduction in already set max speed</td></tr>
<tr><td>weaving</td><td>type: Keyword Argument
required: No
description: enables weaving modifier
data_type: Bool - True/False
units: N/A
default value: if not set, value of weaving is set to false.
True - Applies weaving modifier to path
False - No modification to the path
Additional Parameters
amplitude [m] or amplitude_left [m], amplitude_right [m]
frequency [Hz]
pattern - int
1: Sine
2: Trapezoidal
3: Circle
dwell_time_left [s], dwell_time_right [s] (Trapezoidal only)
elevation [rad]
azimuth [rad]</td></tr>
<tr><td>input_reference_frame</td><td>type: Keyword Argument
required: No
description: frame in which points were teached
data_type:[X,Y,Z,A,B,C]
units: Position values in meters and rotation values in radians
default value: if not set, robot base frame is considered</td></tr>
<tr><td>output_reference_frame</td><td>type: Keyword Argument
required: No
description: frame in which points to be transformed
data_type:[X,Y,Z,A,B,C]
units: Position values in meters and rotation values in radians
default value: if not set, robot base frame is considered</td></tr>
</table>

Return Values:

True	If motion is executed successfully
False	If motion is not executed successfully

Example: (target_pose values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
circular_property = {
            "speed": 0.25,
            "acceleration": 0.1,
            "target_pose": [
                [
                    0.3744609827431085,
                    -0.3391784988266481,
                    0.23276604279256016,
                    3.14119553565979,
                    -0.00017731254047248513,
                    -0.48800110816955566
                ],
                [
                    0.37116786741831503,
                    -0.19686307684994242,
                    0.23300456855796453,
                    3.141423225402832,
                    -0.00020668463548645377,
                    -0.48725831508636475
                ],
                [
                    0.5190337951593321,
                    -0.1969996948428492,
                    0.23267853691809767,
                    3.1414194107055664,
                    -0.00017726201622281224,
                    -0.48750609159469604
                ]
            ],
        "current_joint_angles":r.robot_status("jointAngles")
        }
r.move_circular(**circular_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Example with Weaving: (target_pose values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
circular_property = {
            "speed": 0.008,
            "acceleration": 1.0,
            "target_pose": [
                [
                    0.3744609827431085,
                    -0.3391784988266481,
                    0.23276604279256016,
                    3.14119553565979,
                    -0.00017731254047248513,
                    -0.48800110816955566
                ],
                [
                    0.37116786741831503,
                    -0.19686307684994242,
                    0.23300456855796453,
                    3.141423225402832,
                    -0.00020668463548645377,
                    -0.48725831508636475
                ],
                [
                    0.5190337951593321,
                    -0.1969996948428492,
                    0.23267853691809767,
                    3.1414194107055664,
                    -0.00017726201622281224,
                    -0.48750609159469604
                ]
            ],
        "current_joint_angles":r.robot_status("jointAngles"),
        "weaving":True,
        "pattern": 2,
        "amplitude": 0.006,
        "amplitude_left": 0.0,
        "amplitude_right": 0.0,
        "frequency": 1.5,
        "dwell_time_left": 0.0,
        "dwell_time_right": 0.0,
        "elevation": 0.0,
        "azimuth": 0.0
        }
r.move_circular(**circular_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Move Composite

move_composite method is used to move the robot in the specified linear and circular motion combinations.

Input Parameters:

<table>
<tr><td>commands</td><td>type : Keyword Argument
required: Yes
description: List of linear and circular command combinations
linear command:
blend_radius - blend_radius value in meters
targets - list of target poses
target_joint - list of target joint configuration corresponding to the given target, not required
circular_command:
targets - list of target poses
target_joint - list of target joint configuration corresponding to the given target, not required
data_type: Pose configuration - [X,Y,Z,A,B,C], float
units: Position values in meters and rotation values in radians
default_value: N/A</td></tr>
<tr><td>speed</td><td>type: Keyword Argument
required: No
description: Linear Speed
data_type: float
units: m/sec
default_value: 0.25</td></tr>
<tr><td>acceleration</td><td>type: Keyword Argument
required: No
description: Linear Acceleration
data_type: float
units: m/sec2
default_value: 0.25</td></tr>
<tr><td>current_joint_angles</td><td>type: Keyword Argument
required: No
description: Current Robot Joint Configuration
data_type: List of Joint Values - float
units: radians
default_value: Joint Configuration obtained form Robot Status method</td></tr>
<tr><td>safety_toggle</td><td>type: Keyword Argument
required: No
description: Safety toggle
data_type: Bool - True/False
units: N/A
default_value: if not set, value of the safety toggle in Program screen will be used.
If set to True, Max speed is slashed to 25%
False - No reduction in already set max speed</td></tr>
</table>

weaving	- type: Keyword Argument - required: No - description: enables weaving modifier - data_type: Bool - True/False - units: N/A - default value: if not set, value of weaving is set to false. - True - Applies weaving modifier to path - False - No modification to the path - Additional Parameters - amplitude [m] or amplitude_left [m], amplitude_right [m] - frequency [Hz] - pattern - int 1: Sine 2: Trapezoidal 3: Circle - dwell_time_left [s], dwell_time_right [s] (Trapezoidal only) - elevation [rad] - azimuth [rad]
input_reference_frame	- type: Keyword Argument - required: No - description: frame in which points were teached - data_type:[X,Y,Z,A,B,C] - units: Position values in meters and rotation values in radians - default value: if not set, robot base frame is considered
output_reference_frame	- type: Keyword Argument - required: No - description: frame in which points to be transformed - data_type:[X,Y,Z,A,B,C] - units: Position values in meters and rotation values in radians - default value: if not set, robot base frame is considered
continuous_mode	- type: Keyword Argument - required: No - description: continuous mode - data_type: Bool - True/False - units: N/A - default value: False - If set to True, reduces the velocity in between paths if the acceleration would violate the set acceleration limit - If set to False, velocity is constant even if the acceleration limit is violated.

Return Values:

True	If motion is executed successfully
False	If motion is not executed successfully

Example: (targets values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
composite_motion_property = {
            "speed": 0.432,
            "acceleration": 0.2,
            "current_joint_angles": r.robot_status('jointAngles'),
            "commands": [
                {
                    "linear": {
```

```
                "blend_radius": 0.005,
                "targets": [
                    [
                        -0.000259845199876027,
                        -0.5211437049195536,
                        0.4429382717719519,
                        3.14123272895813,
                        -0.0007908568368293345,
                        -1.570908784866333
                    ],
                    [
                        -0.16633498440272945,
                        -0.5201452059140722,
                        0.4427486025872017,
                        3.140937089920044,
                        -0.0005319403717294335,
                        -1.571555495262146
                    ]
                ]
            }
        },
        {
            "circular": {
                "targets": [
                    [
                        -0.16633498440272945,
                        -0.5201452059140722,
                        0.4427486025872017,
                        3.140937089920044,
                        -0.0005319403717294335,
                        -1.571555495262146
                    ],
                    [
                        -0.16540090985202305,
                        -0.3983552679378624,
                        0.44267608017426174,
                        3.1407113075256348,
                        -0.00036628879024647176,
                        -1.5714884996414185
                    ],
                    [
                        -0.33446498807559716,
                        -0.3989652352814891,
                        0.4421152856242009,
                        3.1402060985565186,
                        0.00030071483342908323,
                        -1.572899580001831
                    ]
                ]
            }
```

```
                }
            ]
        }
r.move_composite(**composite_motion_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Example with Weaving: (targets values in this example are only valid for LARA5)

```
from neurapy.robot import Robot
r = Robot()
composite_motion_property = {
            "speed": 0.008,
            "acceleration": 1.0,
            "current_joint_angles": r.robot_status('jointAngles'),
            "commands": [
                {
                    "linear": {
                        "blend_radius": 0.005,
                        "targets": [
                            [
                                -0.000259845199876027,
                                -0.5211437049195536,
                                0.4429382717719519,
                                3.14123272895813,
                                -0.0007908568368293345,
                                -1.570908784866333
                            ],
                            [
                                -0.16633498440272945,
                                -0.5201452059140722,
                                0.4427486025872017,
                                3.140937089920044,
                                -0.0005319403717294335,
                                -1.571555495262146
                            ]
                        ]
                    }
                },
                {
                    "circular": {
                        "targets": [
                            [
                                -0.16633498440272945,
                                -0.5201452059140722,
                                0.4427486025872017,
                                3.140937089920044,
                                -0.0005319403717294335,
```

```
                                    -1.571555495262146
                                ],
                                [
                                    -0.16540090985202305,
                                    -0.3983552679378624,
                                    0.44267608017426174,
                                    3.1407113075256348,
                                    -0.00036628879024647176,
                                    -1.5714884996414185
                                ],
                                [
                                    -0.33446498807559716,
                                    -0.3989652352814891,
                                    0.4421152856242009,
                                    3.1402060985565186,
                                    0.00030071483342908323,
                                    -1.572899580001831
                                ]
                            ]
                        }
                    }
                ],
                    "weaving":True,
                    "pattern": 2,
                    "amplitude": 0.006,
                    "amplitude_left": 0.0,
                    "amplitude_right": 0.0,
                    "frequency": 1.5,
                    "dwell_time_left": 0.0,
                    "dwell_time_right": 0.0,
                    "elevation": 0.0,
                    "azimuth": 0.0
        }
r.move_composite(**composite_motion_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

Move Trajectory

move_trajectory is used to move the robot with a given joint trajectory.

Input Parameters:

timestamps	• type: Keyword Argument • required: Yes • description: List of times at which the joint configurations should be reached • data_type: float • units: sec • default_value: N/A

target_joint	• type : Keyword Argument • required: Yes • description: List of joint configurations • data_type: List of Joint Values - float • units: radians • default_value: N/A
current_joint_angles	• type: Keyword Argument • required: No • description: Current Robot Joint Configuration • data_type: List of Joint Values - float • units: radians • default_value: Joint Configuration obtained form Robot Status method

Return Values:

True	If motion is executed successfully
False	If motion is not executed successfully

Example:

```
from neurapy.robot import Robot
r = Robot()
trajectory_motion_property = {
  "current_joint_angles": r.robot_status('jointAngles'),
  "timestamps": [
    0.01, 0.02, 0.03
  ],
  "target_joint": [
    [0.0531381, 0.157485, -0.100272, 1.29569, -5.99211e-05, 0.700957,
-0.000371511],
    [-0.208897,0.461728,-0.433937,1.66485,-5.99211e-05,0.700382,
-0.000383495],
    [0.0120801,0.621298,0.0149563,1.67381,5.99211e-05,0.687403,
-0.000359527]
  ]
}
r.move_trajectory(**trajectory_motion_property)
r.stop() # this needs to be called only once at the end of the script
for a proper program termination
```

This is only an example and not tested on any robot. Please provide a complete Trajectory for move_trajectory to work properly.

Record Path

record_path method is used to move the robot in a pre-recorded path.

Input Parameters:

is_motion	• type: Keyword Argument • required: Yes • description: True, if constant velocity is needed during the motion • data_type: bool

file_location	• type: Keyword Argument • required: Yes • description: Location of the recorded path file • data_type: str
speed	• type: Keyword Argument • required: Yes • description: Linear Speed • data_type: float • units: m/sec
current_joint_angles	• type: Keyword Argument • required: No • description: Current Robot Joint Configuration • data_type: List of Joint Values - float • units: radians • default_value: Joint Configuration obtained form Robot Status method

safety_toggle	• type: Keyword Argument
	• required: No
	• description: Safety toggle
	• data_type: Bool - True/False
	• units: N/A
	• default_value: if not set, value of the safety toggle in Program screen will be used.
	  ◦ If set to True, Max speed is slashed to 25%
	  ◦ False - No reduction in already set max speed

use_recordings_filter	• type: Keyword Argument • required: No • description: Toggle to enable/disable the filtering on recorded data • data_type: Bool - True/False • units: N/A • default_value: True - Filtering is turned on by default

Return Values:

True	If motion is executed successfully
False	If motion is not executed successfully

Example:

```
from neurapy.robot import Robot
r = Robot()
recorded_path_property = {
            "is_motion": False,
            "file_location": "<path to file location>",
            "speed": 0.25,
            "use_recordings_filter": False,
                "current_joint_angles": r.robot_status('jointAngles'),
        }
r.record_path(**recorded_path_property)
```

Sample recorded paths for lara5 can be downloaded from here!

Power

power method is used to turn on/off the power to robot.

Input Parameters:

value	• type: Argument
	• required: Yes
	• description:
	  ◦ 'on' - for powering on the robot
	  ◦ 'off' - for powering off the robot
	• data_type: str

Return Values:

True	If operation is executed successfully
False	If operation is not executed successfully

Example

```
from time import sleep
from neurapy.robot import Robot
r = Robot()
r.power('on')
sleep(2)
r.power('off')
```

Motion status

motion_status method is used to query the motion status of the robot.

Input Parameters: N/A

Return Values:

NOT_RUNNING	If the robot is not running
RUNNING	If the robot is running
PAUSED	if the robot is in paused state
POWERED OFF	if the robot is powered off

Example

```
from neurapy.robot import Robot
r = Robot()
print(r.motion_status())
```

Program status

program_status method is used to query the program_status of the robot

Input Parameters: N/A

Return Values:

NOT_RUNNING	If the program is not running from Teach pendant
RUNNING	If the program is running from Teach pendant
PAUSED	if the program running from Teach pendant is in pause state

Example

```
from neurapy.robot import Robot
r = Robot()
status = r.program_status()
```

Warnings

get_warnings method is used to query the list of warnings present on the robot

Input Parameters: N/A

Return Values:

List of warning codes	Please refer to the code descriptions in List of Errors and Warnings page.

Example

```
from neurapy.robot import Robot
r = Robot()
warnings = r.get_warnings()
print(warnings)
```

Errors

get_errors method is used to query the list of errors present on the robot

Input Parameters: N/A

Return Values:

List of error codes	Please refer to the code descriptions in List of Errors and Warnings page.

Example

```
from neurapy.robot import Robot
r = Robot()
errors = r.get_errors()
print(errors)
```

Get point

get_point method is used to query the data of the point stored on the robot.

Input Parameters:

value	• type: Argument • required: Yes • description: name with which point was saved from GUI • data_type: str

Return Values:

Dictionary containing point data	If point exists in the database

Example

```
from neurapy.robot import Robot
r = Robot()
point = r.get_point("P1")
print(point)
```

IOs

io method is used to access and manipulate(output io's) io values on the robot.

Input Parameters:

operation_type	• type: Argument
	• required: Yes
	• description: type of operation
	    • 'get' - for getting the io value
	    • 'set' - for setting the output io value
	• data_type: str

io_name	• type: Keyword Argument • required: Yes • description: name of the io(list of available IO names can be found here) • data_type: str
target_value	• type: Keyword Argument • required: needed only for "set" operation • description: target value of the io

Return Values:

• value of the queried io_name for "get" operation • True/False - if the "set" operation has succeeded or not

Example

```
import time
from neurapy.robot import Robot
r = Robot()
io_get = r.io("get", io_name = "DO_1")
io_set = r.io("set", io_name = "DO_2", target_value = True)
tdo_set = r.io("set",io_name="TOD_Array",target_value=[0.0,1.0,0.0]) #
to set tool digital ouput 1 to true
tdo_get = r.io("get",io_name=TDO_1)
print(io_get,io_set)
""" DI - Contorl box digital inputs
    AI -Control box analog Inputs
    TAI - tool analog inputs
    TDI - tool digital inputs
    TDO - tool digital ouputs
    DO - Contorl box digital ouputs
"""
```

Gripper Control

gripper method is used to control the gripper connected to the robot

Input Parameters:

operation_type	• type: Argument
	• required: Yes
	• description: type of operation
	    • 'on' - for opening the gripper
	    • 'off' - for closing the gripper
	• data_type: str

Return Values:

True	If the operation succeds
False	If the operation fails

Example

```
from time import sleep
from neurapy.robot import Robot
r = Robot()
r.gripper('on')
sleep(2)
r.gripper('off')
```

Set Tool

set_tool method is used to notify the gripper/tool change to software components, after the physical change

Input Parameters:

tool_name	• type: Keyword Argument • required: Yes • description: name of the tool defined from GUI • data_type: str

Return Values:

True	If the operation succeeds
False	If the operation fails

Example

```
from neurapy.robot import Robot
r = Robot()
r.set_tool(tool_name="name given during tool creation in GUI")
```

Robot Status

robot_status method is used to query the current status of the robot.

Parameters and Return values

Parameter	Return Value
cartesianPosition	Robot Pose
jointAngles	Joint Positions
jointTorques	Joint Torque
commandedjointAngle	last commanded joint angle
taskStateTwist	taskStateTwist linear
loadSideEncValue	Primary encoder value
motorSideEncValue	Secondary encoder value

Example:

1.Robot status without timestamp

```
from neurapy.robot import Robot
r = Robot()
c = r.robot_status('jointAngles')
"""
            "cartesianPosition" -  returns Robot Pose in xyzwxyz format
(rotation in quaternion)
            "jointAngles" -  returns Joint Positions
            "jointTorques" -  returns Joint Torque
            "commandedjointAngle" - returns last commanded joint angle
            "taskStateTwist" - returns taskStateTwist linear
            "loadSideEncValue" - returns Primary Encoder Value
            "motorSideEncValue" - reutrns Secondary Encoder Value
"""
```

2.robot status with time stamp

```
from neurapy.robot import Robot
r = Robot()
c = r.robot_status('jonitAngles',time_stamp=True)
print(c) # tuple containing list of joint angles and utc timestamp
"""
            "cartesianPosition" -  returns Robot Pose in xyzwxyz format
(rotation in quaternion)
            "jointAngles" -  returns Joint Positions
            "jointTorques" -  returns Joint Torque
            "commandedjointAngle" - returns last commanded joint angle
            "taskStateTwist" - returns taskStateTwist linear
            "loadSideEncValue" - returns Primary Encoder Value
            "motorSideEncValue" - reutrns Secondary Encoder Value
"""
```

Zero G

zero_g method is used to toggle the freedrive/Gravity compensation mode.

Input Parameters:

value	• type: Argument
	• required: Yes
	• description:
	  ◦ 'on' - for turning on the free drive mode
	  ◦ 'off' - for turning off the free drive mode
	• data_type: str

Return Values:

True	If operation is executed successfully

False	If operation is not executed successfully

Example

```
import time
from neurapy.robot import Robot
r = Robot()
r.zero_g("on")
time.sleep(1)
r.zero_g("off")
```

Forward/Inverse Kinematics

ik_fk method is used to compute forward/inverse kinematics for a given configuration

Input Parameters:

operation_type	- type: Argument - required: Yes - description: 'ik' - for joint configuration calculation, for the given cartesian position 'fk' - for cartesian position calculation, for the given joint configuration - data_type: str
target_pose	- type: Keyword Argument - required: Yes, only for ik calculation - description: Pose, in the form of list in XYZABC format - data_type: list
target_angle	- type: Keyword Argument - required: Yes, only for fk calculation - description: list of joint angles - unit: radians - data_type: list
current_joint	- type: Keyword Argument - required: Yes, only for fk calculation - description: list of current joint angles - unit -radians - data_type: list

Return Values:

target_position in XYZABC format / target_angle	

Example

```
from neurapy.robot import Robot
r = Robot()
target_position = r.ik_fk("fk", target_angle =
[0.2,0.2,0.2,0.2,0.2,0.2])
target_angle = r.ik_fk("ik", target_pose = [0.140448, -0.134195,
1.197456, 3.1396, -0.589, -1.025],current_joint = [-1.55, -0.69, 0.06,
1.67, -0.02, -1.57, 0.11])
```

Override

override method is used to set the override to a given value on the robot

Input Parameters:

operation_type	• type: Argument
	• required: Yes
	• description: type of operation
	  ◦ 'get' - for getting the current override value
	  ◦ 'set' - for setting the override value
	• data_type: str

target_value	• type: Keyword Argument • required: Yes, only for set operation • description: override value • data_type: float

Return Values:

True/False	If the set operation succeeds/fails
Value	If the get operation succeeds

Example

```
import time
from neurapy.robot import Robot
r = Robot()
override_value = r.override("get")
print("override_value : " + str(override_value))
time.sleep(1)

override_value = r.override("set", target_value = 0.4)
print("Setting Override to 0.4")
time.sleep(1)

override_value = r.override("get")
print("override_value : " + str(override_value))
```

Set Mode

set mode method is used to toggle the robot modes(Teach/Automatic/SemiAutomatic).

Input Parameters:

value	• type: Argument
	• required: Yes
	• description:
	  ◦ 'Teach' - for changing to teach mode
	  ◦ 'Automatic' - for changing to Automatic mode
	  ◦ 'SemiAutomatic' - for changing to semi automatic mode
	• data_type: str

Return Values:

True	If operation is executed successfully
False	If operation is not executed successfully

Example

```
import time
from neurapy.robot import Robot
r = Robot()
r.set_mode("Teach")
time.sleep(1)
r.set_mode("Automatic")
time.sleep(1)
r.set_mode("SemiAutomatic")
```

Get Mode

get_mode method is used to query current robot mode(Teach/Automatic/SemiAutomatic).

Input Parameters: N/A

Return Values:

Teach/Automatic/SemiAutomatic	Based on the robot mode

Example

```
from neurapy.robot import Robot
r = Robot()
mode = r.get_mode()
print(mode)
```

Reset error

reset error method is used to reset the error on real robot

Input Parameters: N/A

Return Values:

True/False	If operation is executed successfully or not

Example

```
import time
from neurapy.robot import Robot
r = Robot()
r.reset_error()
```

Get tools

get tools method is used to get the data of available tools

Input Parameters: N/A

Return Values:

List of tool data

Example

```
import time
from neurapy.robot import Robot
r = Robot()
tools_data = r.get_tools()
```

Create tool

create tool method is used to create a new tool in robot database with the given information

Input Parameters:

tool_data	• type: Argument • required: Yes • description: refer to tool description table below • data_type: dictionary

Tool description:

Key	Value(sample value)	Description	Required or not
_controlOA	False	True,if the tool is controlled by controlbox analog outputs	
_controlOD	False	True,if the tool is controlled by controlbox digital outputs	
_toolOA	False	True,if the tool is controlled by analog outputs of port present on robot tool flange	
_toolOD	False	True,if the tool is controlled by digital outputs of port present on robot tool flange	

autoM	0	Mass of the tool	
autoMeasureX	0	x-offset of tool's Center of mass	
autoMeasureY	0	y-offset of tool's Center of mass	
autoMeasureZ	0	z-offset of tool's Center of mass	
closeInput	0		
cmdID	16		
description	Tool Description	Tool description	
force	0		
gripper			
grippertype	Standard Gripper		
inertiaXX	0	Ixx of tool	
inertiaXY	0	Ixy of tool	
inertiaXZ	0	Ixz of tool	
inertiaYY	0	Iyy of tool	
inertiaYZ	0	Iyz of tool	
inertiaZZ	0	Izz of tool	
name	NoTool	Name of the tool	
offCOA	[0, 0, 0, 0, 0, 0, 0, 0]		
offCOD1	0	if the tool is controlled via control box digital outputs, offCOD1 is the pin mapped to turn off(close) the tool	
offCOD2	0		
offTOA	[0, 0]		
offTOD	0	if the tool is controlled via tool digital outputs, offTOD is the pin mapped to turn off(close) the tool	
offsetA	0	TCP roll offset	
offsetB	0	TCP pitch offset	
offsetC	0	TCP yaw offset	
offsetX	0	TCP offset in X	
offsetY	0	TCP offset in Y	
offsetZ	0	TCP offset in Z	
onCOA	[0, 0, 0, 0, 0, 0, 0, 0]		
onCOD1	0	if the tool is controlled via control box digital outputs, onCOD1 is the pin mapped to turn on(open) the tool	
onCOD2	0		
onTOA	[0, 0]		
onTOD	0	if the tool is controlled via tool digital outputs, onTOD is the pin mapped to turn on(open) the tool	
openInput	0		
portID			
protocol	0		
robot_type	Tool		
slaveID	0		
speed	0		

Return Values:

True/False	If operation is executed successfully or not

Example

```
from neurapy.robot import Robot
r = Robot()
tool_data = { '_controlOA': False,
 '_controlOD': False,
 '_toolOA': False,
 '_toolOD': False,
 'autoM': 0,
 'autoMeasureX': 0,
 'autoMeasureY': 0,
 'autoMeasureZ': 0,
 'closeInput': 0,
 'cmdID': 16,
 'description': 'Tool Description',
 'force': 0,
 'gripper': '',
 'grippertype': 'Standard Gripper',
 'inertiaXX': 0,
 'inertiaXY': 0,
 'inertiaXZ': 0,
 'inertiaYY': 0,
 'inertiaYZ': 0,
 'inertiaZZ': 0,
 'name': 'NoTool',
 'offCOA': [0, 0, 0, 0, 0, 0, 0, 0],
 'offCOD1': 0,
 'offCOD2': 0,
 'offTOA': [0, 0],
 'offTOD': 0,
 'offsetA': 0,
 'offsetB': 0,
 'offsetC': 0,
 'offsetX': 0,
 'offsetY': 0,
 'offsetZ': 0,
 'onCOA': [0, 0, 0, 0, 0, 0, 0, 0],
 'onCOD1': 0,
 'onCOD2': 0,
 'onTOA': [0, 0],
 'onTOD': 0,
 'openInput': 0,
 'portID': '',
 'protocol': 0,
 'robot_type': 'Tool',
 'slaveID': 0,
 'speed': 0}

tools_data = r.create_tool(tool_data)
```

Get encoder offsets

get encoder offsets method is used to get the encoder offsets of robot

Input Parameters: N/A

Return Values:

List of encoder offsets

Example

```
import time
from neurapy.robot import Robot
r = Robot()
encoder_offsets = r.get_encoder_offsets()
```

Encoder to radian values

encoder2rad method is used to convert given encoder ticks to joint angles.

Input Parameters:

value	• type: Argument • required: Yes • description: list of encoder values • data_type: List of floats

Return Values:

List of joint angles in radians

Example

```
import time
from neurapy.robot import Robot
r = Robot()
encoder_ticks = r.robot_status('loadSideEncValue')
joint_angles = r.encoder2rad(encoder_ticks)
```

Quaternion to roll-pitch-yaw

Input Parameters: quaternions w,x,y,z

Return Values: rpy angles in radians

Example

```
from neurapy.robot import Robot
r = Robot()
rpy = r.quaternion_to_rpy(0.85,0,0.52,0)
```

Roll-pitch-yaw to quaternion

Input Parameters: rpy angles in radians

Return Values: quaternions w,x,y,z

Example

```
from neurapy.robot import Robot
r = Robot()
quaternion = r.rpy_to_quaternion(0,1.1,0)
```

Get ZeroG status

get zerog status method is used to get the status of the robot free drive mode

Input Parameters: N/A

Return Values:

Turned On/ Turned Off

Example

```
import time
from neurapy.robot import Robot
r = Robot()
status = r.get_zerog_status()
```

Get reference frame

get_reference_frame method is used to query the data of the frame stored on the robot.

Input Parameters:

value	• type: Argument • required: Yes • description: name with which frame was saved from GUI • data_type: str

Return Values:

list containing the frame data - pose of the frame wrt to the reference frame(frame in which points were taught) frame - [X,Y,Z,A,B,C] X,Y,Z - 3D position wrt to reference frame - in meters A,B,C - 3D rotation represented in roll, pitch, yaw values in radians - rotation order 'ZYX'	If frame exists in the database

Example

```
from neurapy.robot import Robot
r = Robot()
frame = r.get_reference_frame("tool_frame")
print(frame)
```

Jogging with NeuraPy

Perform both Cartesian and Joint jogging with Neurapy.

(This feature is available from software version v4.10.0)

Input Parameters:

jog_velocity	• type: Argument • required: Yes • description: either cartesian (X,Y,Z,R,P,Y) or joint velocity between [-1, 1] which represents the percentage velocity sent to robot of the max velocity defined. • data_type: array
jog_type	• type: Argument • required: Yes • description: Cartesian or Joint • data_type: string
set_jogging_external_flag	• type: Argument • required: Yes • description: Flag to run external jog command in each cycle. This has to be called in a loop each time so that jog command is passed. • data_type: bool

Example

Example code to run jogging with neurapy (Joint Jog).

```
from neurapy.robot import Robot

robot = Robot()
"""
jog_velocity - velocity ranging from [-1,1] for all joints
jog_type - can be either cartesian or joint jogging
turn_on_jog changes the state from interal jogging (from GUI) to
external jogging
"""
robot.turn_on_jog(jog_velocity=[0.2, 0.2, 0.2, 0.2, 0.2, 0.2],
jog_type='Joint')

# command to set flag for jogging in external mode.
robot.jog(set_jogging_external_flag = 1)
i = 0

"""
Requires minimum number of cycles in the loop for performing jogging.
Depends upon jogging velocity, override.
"""
while(i < 500):
    """
    command to set flag for jogging in external mode. This command has
to be used each time
    external jog command has to be sent
    """
    robot.jog(set_jogging_external_flag = 1)
    i+=1

"""
Change the state from external jog to internal jog  (GUI) and sets all
other
external parameters to false
"""
robot.turn_off_jog()
```

Example code to run jogging with neurapy (Cartesian Jog).

```
from neurapy.robot import Robot

robot = Robot()
robot.turn_on_jog(jog_velocity=[0.2, 0.2, 0.2, 0.2, 0.2, 0.2],
jog_type='Cartesian')
robot.jog(set_jogging_external_flag = 1)
i = 0

while(i < 500):
    robot.jog(set_jogging_external_flag = 1)
    i+=1
robot.turn_off_jog()
```

Get reference frame with offset

get_reference_frame_with_offset method is used to create a reference frame at a given offset to the existing reference frame

(This feature is available from software version v4.11.0)

Input Parameters:

existing_reference_frame	• type: Argument • required: Yes • description: name with which frame was saved from GUI • data_type: str
offset	• type: Argument • required: Yes • description: offset values(measured in existing reference frame) in meters in x,y,z directions • data_type: list of floats

Return Values:

list containing the frame data - pose of the frame wrt to the reference frame(frame in which points were taught) frame - [X,Y,Z,A,B,C] X,Y,Z - 3D position wrt to reference frame - in meters A,B,C - 3D rotation represented in roll, pitch, yaw values in radians - rotation order 'ZYX'	If the operation is succesful

Example

```
from neurapy.robot import Robot
r = Robot()
x_offset = 0.02 # in meters
y_offset = 0.1 # in meters
z_offset = 0.0 # in meters
frame = r.get_reference_frame_with_offset("world",[x_offset,y_offset,
z_offset])
print(frame)
```

Get tcp pose

get_tcp_pose method is used to query the robot's current tcp pose.

Input Parameters: N/A

Return Values:

list containing the tcp pose - tcp_pose - [X,Y,Z,A,B,C] X,Y,Z - 3D position wrt to reference frame - in meters A,B,C - 3D rotation represented in roll, pitch, yaw values in radians - rotation order 'ZYX'	If the operation is successful

Example

```
from neurapy.robot import Robot
r = Robot()
tcp_pose = r.get_tcp_pose()
print(tcp_pose)
```

Get Sim or Real

get_sim_or_real method is used to query current robot running context(Real/Simulation).

(This feature is available from software version v4.9.0)

Input Parameters: N/A

Return Values:

Real/Simulation	Based on the robot running context

Example

```
from neurapy.robot import Robot
r = Robot()
context = r.get_sim_or_real()
print(context) #Real/Simulation
```

Read safeio

read_safeio method is used to get the value of the given safety IO.

(This feature is available from software version v4.11.0)

Input Parameters:

SafetyIO number	type : int (range 1-8) required: yes

Return Values:

True/False	Based on the IO status

Example

```
from neurapy.robot import Robot
r = Robot()
safe_io = r.read_safeio(1)
print(safe_io) #True/False
```

Reset Teach Mode

reset_teach_mode needs to be called after executing motion and before set_mode/mode change from GUI, if user wants to switch the robot to teach mode.

(This feature is available from software version v4.11.0)

Input Parameters: N/A

Return Values:

True/False	If executed succesfully

Example

```
from neurapy.robot import Robot
r = Robot()
r.set_mode("Automatic")
r.move_joint([0]*6)
r.reset_teach_mode()
r.set_mode("Teach")
```

Set Sim Real

set_sim_real method is used to toggle the robot running context. i.e Real or Simulation

(This feature is available from software version v4.11.0)

Input Parameters:

True/False	True : To switch to Real False : To switch to simulation

Return Values:

True/False	If executed succesfully

Example

```
from neurapy.robot import Robot
r = Robot()
r.set_sim_real(True)
print(r.get_sim_or_real())
r.set_sim_real(False)
print(r.get_sim_or_real())
```

Pause/Unpause/Stop

Pause/Unpause/Stop is used to pause, unpause and stop the program using Neurapy.

Note: Clean up action has to be taken before stopping the program using stop function. (save the data if required, stop the python program)

Stop function needs to be called at the end of every neurapy script to terminate the execution properly and it can be used to stop the robot motion while handling the interrupt signals.

Input Parameters: N/A

Return Values:

True/False	If executed successfully

Example

```
from neurapy.robot import Robot
r = Robot()
r.pause() # to pause the program
r.unpause() # to pause the program
r.stop() # to pause the program
```

Wait for signal

wait for signal functionalities are used when waiting for a digital signal with or without delay timers

Example

```
from neurapy.robot import Robot
r = Robot()
r.wait_for_digital_input(1,True) #waits till digital input 1 is True
r.wait_for_digital_input_timer_on_delay(1,10) #waits for 10sec and
returns after digital input 1 reaches high
r.wait_for_digital_input_timer_off_delay(1,10) #waits for 10sec and
returns after digital input 1 reaches low
r.wait_for_analog_input(1,2.3) #waits till analog input 1 is 2.3
r.wait_for_tool_digital_input(1,True) #waits till tool digital input 1
is True
r.wait_for_tool_digital_input_timer_on_delay(1,10) #waits for 10sec and
returns after tool digital input 1 reaches high
r.wait_for_tool_digital_input_timer_off_delay(1,10) #waits for 10sec
and returns after tool digital input 1 reaches low
r.wait_for_tool_analog_input(1,2.3v) #waits till tool analog input 1 is
2.3v
```