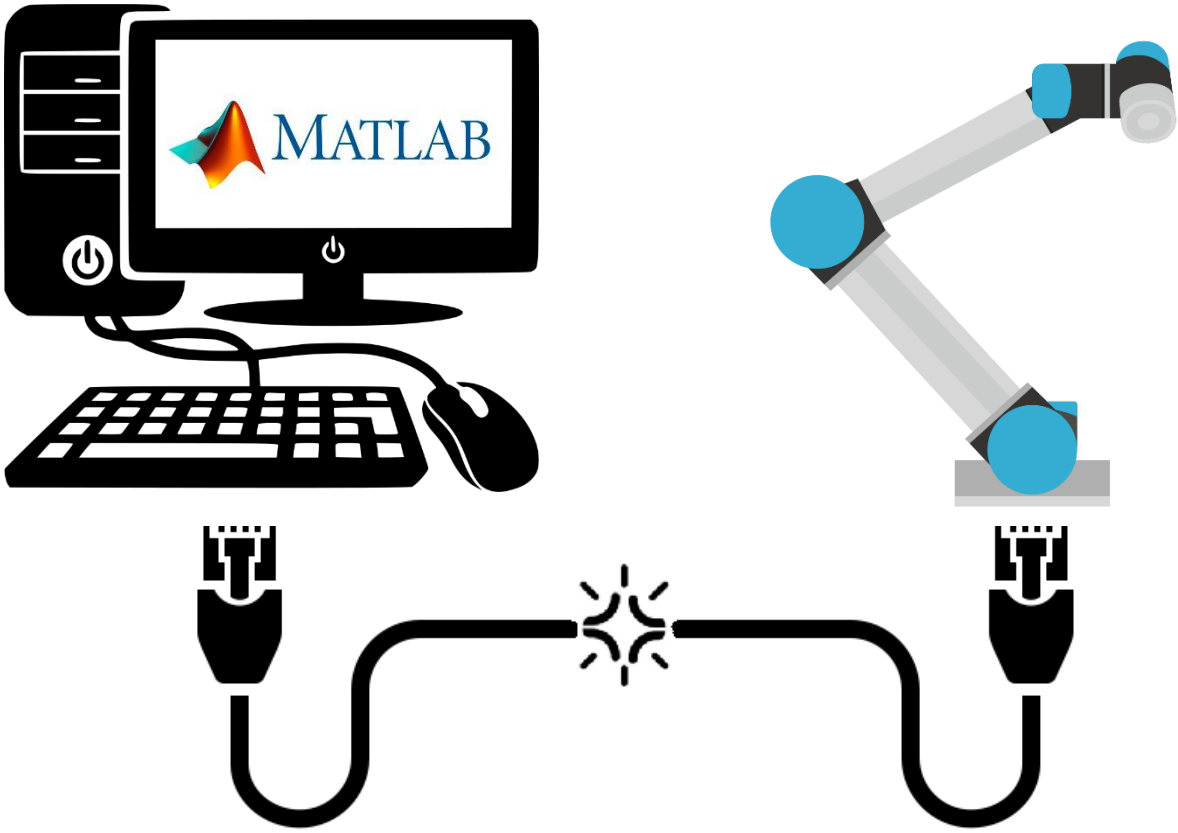

MATLAB – UR5e toolbox

Mechlab UMTMB

Last update: 3.3.2023 10:33

Manual in process

Contents:

1. Robot connection

Communication between UR5 and PC is possible through the TCP/IP connection. There are multiple ports available, the most important are mentioned below:

- 29999 – Dashboard server
- 30001 – Primary interface (10 Hz)
- 30003 – Real-time client interface (500 Hz)
- 30004 – Real-time data exchange (500 Hz)

The connection enables reading data from the robot (position, I/O, force, etc.) and simultaneously sending instructions to the robot controller. Outcoming instructions from Matlab are written in URScript language, which is described in detail here:

https://s3-eu-west-1.amazonaws.com/ur-support-site/61791/scriptManual.pdf

The Matlab toolbox created at UMTMB provides automatic port connection and configuration after the control object is created. The user must properly configure the network adapter of the UR and the PC.

1.1. UR5 setting

The first necessary procedure is to set a static IP address in UR Polyscope. The menu is accessible via the button in the upper right corner (Setting – System – Network). The values must be set accordingly to the connection type (whether the UR and PC are connected directly or over a subnet).
The toolbox has a default IP address of the robot, which can be used when the robot is connected directly to the PC. For this case, set the address in Polyscope to:

IP address:	10.1.1.2
Subnet mask:	255.255.255.0
Default Gateway:	10.1.1.1

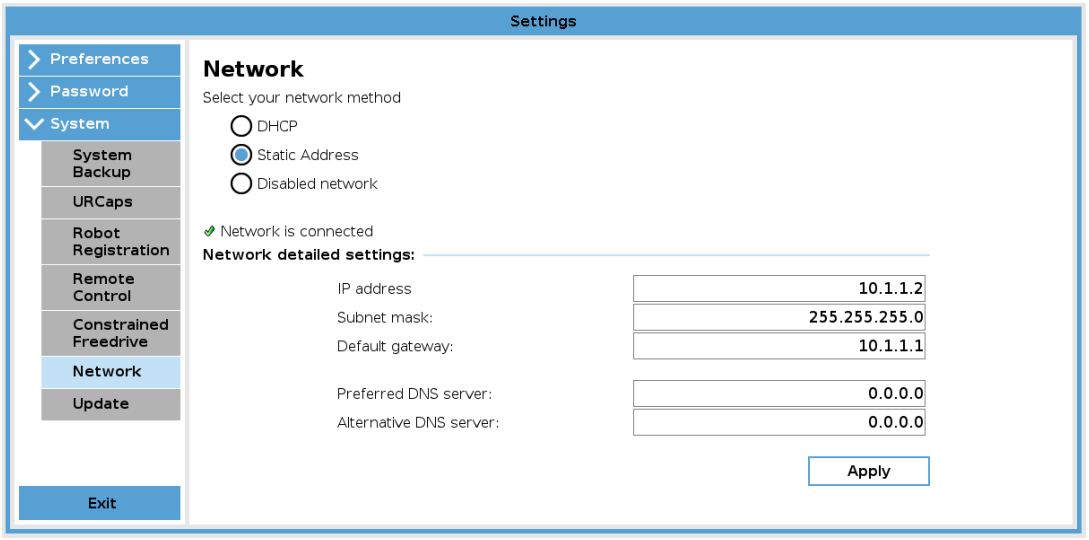

Fig. 1) Polyscope Network setting

If the robot and PC are part of LAN, the values must be set accordingly to that network. To obtain IP addresses in Mechlab, contact laboratory manager.

The robot can work in three modes, i.e.: “Manual”, “Automatic”, and “Remote”. The mode button is also in the right upper corner; see Fig. 3. To control the robot from Matlab, it must be switched to the remote mode. If the mode is not listed in the menu, it must be enabled in the setting menu; see Fig. 2.

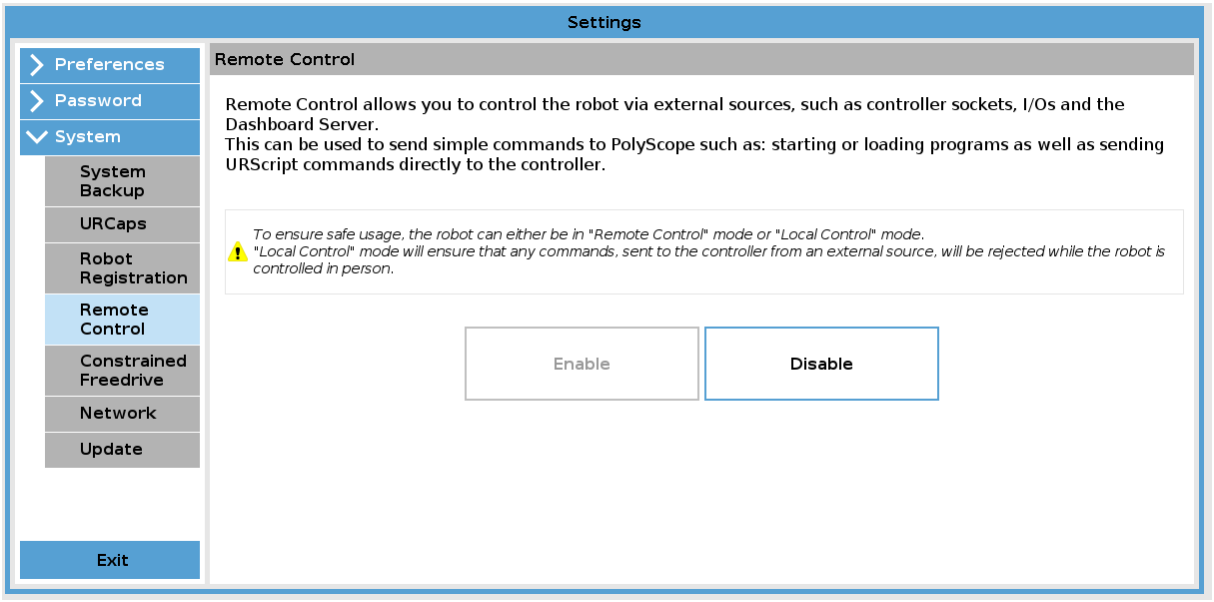

Fig. 2) Enabling Remote Mode Control

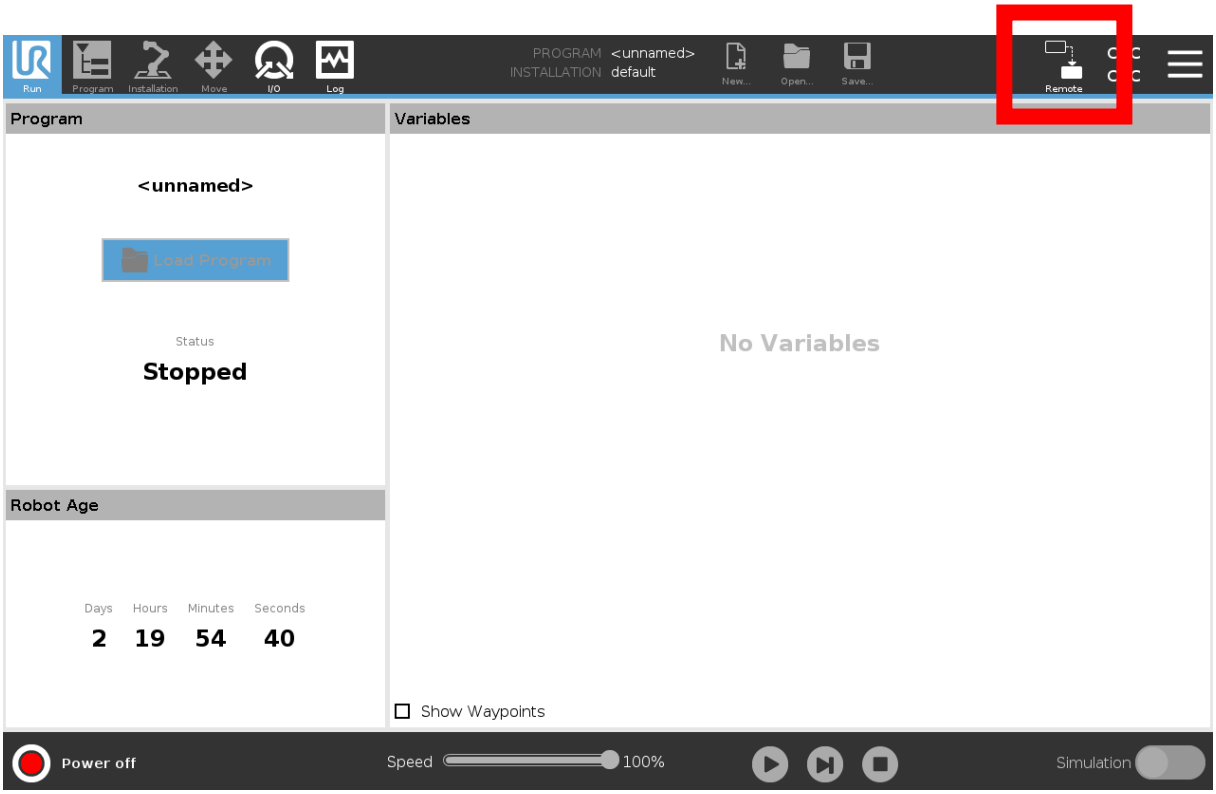

Fig. 3) Mode button

1.2. PC setting

PC IP address can be set as follows (Windows): Control Panel > Network and Internet > Network and Sharing Center > Change adapter settings; see Fig. 4.

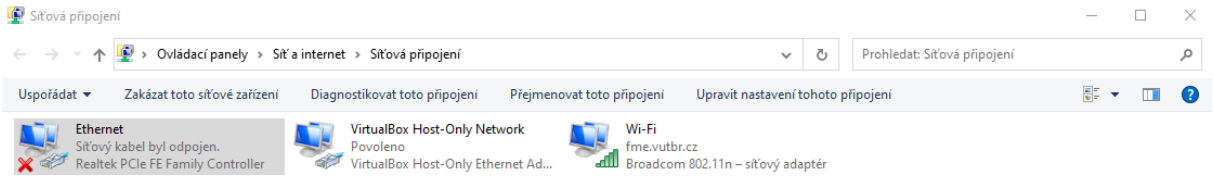

Fig. 4) Network adapter setting

An appropriate network adapter must be chosen and then continue to Properties > IP version 4 (TCP/IPv4) (see Fig. 5) > Properties > Select *"Use following IP address"* (instead of *"Obtain the IP address automatically"*).

Again, if the PC and the robot are part of some LAN, the values must be set according to the network. For the direct connection (PC – robot), the address can be set to:

- IP address: 10.1.1.1
- Subnet mask: 255.255.255.0
- Default gateway: 10.1.1.2 (IP address of UR)

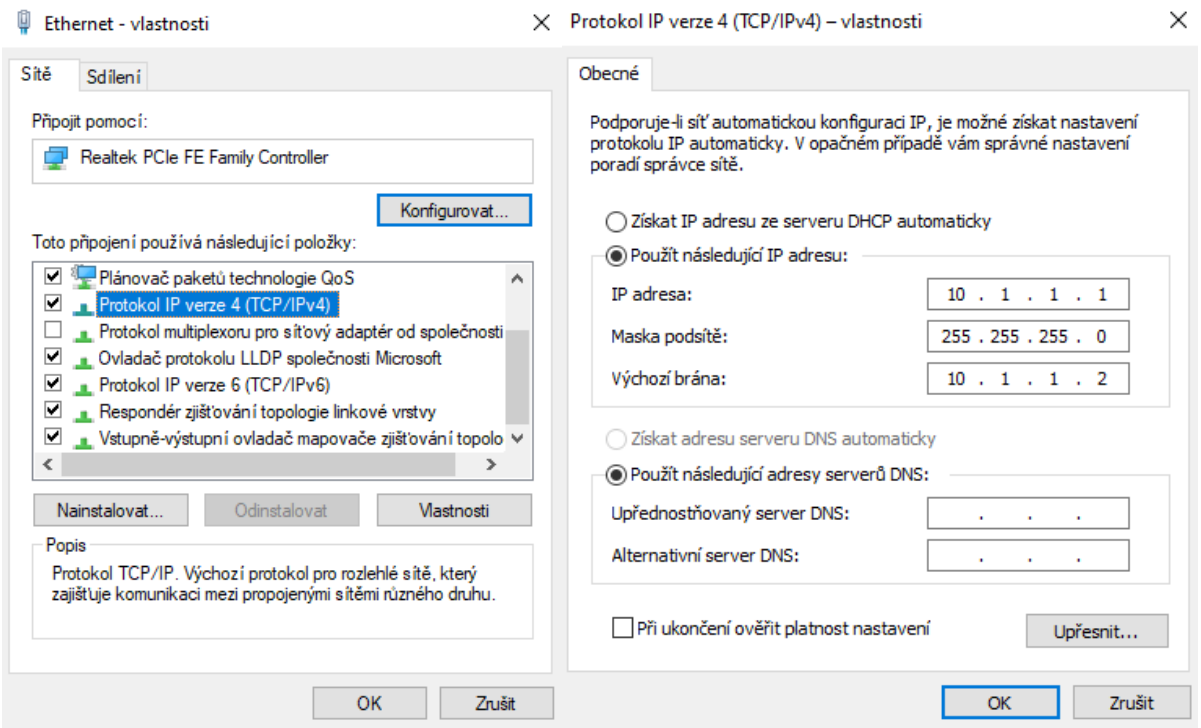

Fig. 5) a) Adapter properties, b) Setting The Static IP Address

2. Coordinate system

All position data are related to the Base coordinate system. It is placed at the bottom of the robot and the axis are visualized on the Fig. 6.

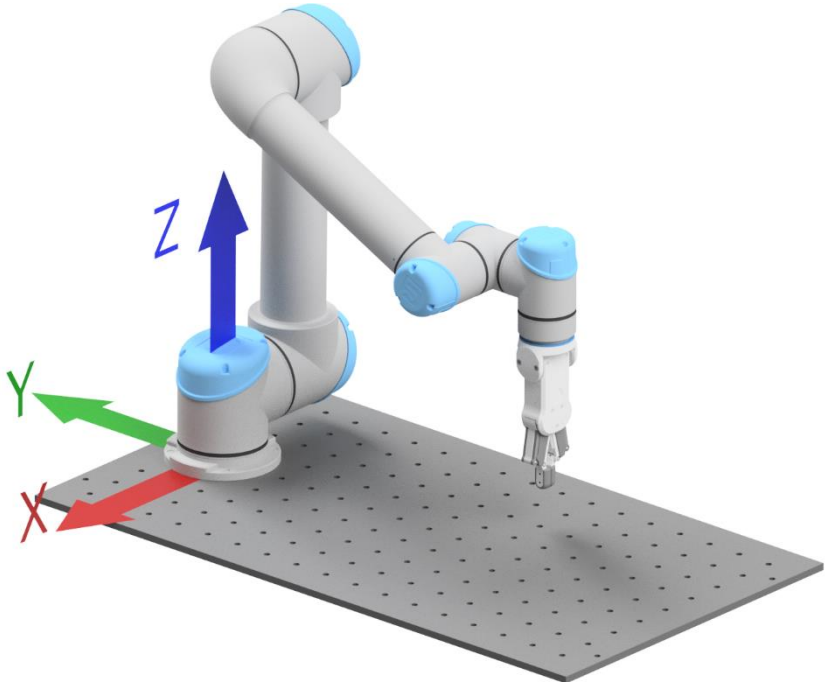

Fig. 6) Robot Base coordinate system

3. Ur_lib_demo

This section contains an example of using the most essential commands of the toolbox.

```
clc, clear, close all

%% Poses and configs declaration
pose1 = [0.05;-0.49;0.28;0;-pi;0];           % format [x;y;z;rx;ry;rz]
pose2 = [-0.1;-0.49;0.28;0;-pi;0];

config1 = [-pi/2;-pi/2;-pi/2;-pi/2;pi/2;0];  % format [q1;q2;q3;q4;q5;q6]
config2 = [-pi/3;-pi/2;-pi/3;-pi/2;pi/2;0];
config3 = [-pi/2;-pi/2;-pi/2;-pi/2;pi/2;0];

Q = [config1 config2 config3 config1];       % every column is one config

%% Object creation
IP = '10.1.1.5'                              % IP address of the robot
ur = UR_robot('robot','ip',IP);              % ur object is created

%% Set robot to config1
ur.a_joint = 0.4;                            % set joint acceleration [rad/s2]
ur.v_joint = 0.2;                            % set joint speed [rad/s]

ur.set_config(config1)                       % move robot to the config1

%% Move through all configs
ur.set_config(Q)                             % robot moves through all configs

%% Movej and Movel commands
ur.a_tool = 0.1;                             % set TCP acceleration [m/s2]
ur.v_tool = 0.03;                            % set TCP speed [m/s]

ur.movej(pose1)                              % robot execute joint movement
ur.movel(pose2)                              % robot executes linear movement

%% Reading TCP and joints positions
poseActual = ur.pose;                        % read actual position of TCP
configActual = ur.config;                    % read actual joints position
```

4. Class description

4.1. Object creation

Constructor - UR_robot

`ur = UR_robot(mode,options)`

Creates an ur object of the UR_robot class. Depending on the selected mode, it is possible to control a real arm or a simulation in the Matlab environment.

mode (2 options)

'sim'	• the created object can be used to control the robot in a simulation
'robot'	• the created object can be used to control the real UR5e robot • the TCP/IP communication is established via ports 29999, 30001, 30003

options

'user', U	U = 'student'	• default • for a student user, the robot's acceleration and speed are limited (max 5 cm/s^2, 5 cm/s, 20 deg/s^2, 10 deg/s)
	U = 'admin'	• unlimited acceleration and speed • for activation, it is necessary to insert the password as the *password* argument
'password', P		• necessary for admin user activation • P contains the password in text format
'ip', IP		• Set a required IP address of the robot • Default address: '10.1.1.2'

Examples:

```
pswrd = load('password');                                          % load the passport file
ur1 = UR_robot('robot', 'user', 'admin', 'password', pswrd);       % create an object

ur2 = UR_robot('sim')                                              % create another object
```

4.2. Methods

UR_robot.movej

Linear movement in joint-space. The trajectory of TCP is not known, but it is not linear or the shortest!

`ur.movej(pose)`
Moves the TCP from the current position to the position specified in the *pose* argument. Acceleration and speed of the move are set in properties *a_joint* [rad/s^2] and *v_joint* [rad/s].

Pose

- as a vector (6x1 – double)
  - format: [x ; y ; z ; rx ; ry ; rz]
  - x, y, z – TCP position in the base frame
  - rx, ry, rz - the orientation of the TCP, given in axis-angle notation
- as a matrix (6xn – double)
  - it is possible to send more positions in a single matrix
  - every column is one position, the format is explained above
  - in this format, it is possible to use a blend radius between each position which causes a continuous motion

Examples

```
pose1 = [0.0300;-0.3789;0.3959;-0.0269;3.1315;0.0000];   % defined poses
pose2 = [-0.0634;-0.4533;0.3731;0.1756;3.1366;0.0000];
pose3 = [-0.0066;-0.5688;0.3906;-0.1788;3.1365;0.0000];

ur.v_joint = 10;                                         % set joint speed
ur.a_joint = 40;                                         % set joint acceleration
ur.movej(pose1)                                          % move to the pose1

pose = [pose1,pose2,pose3];                              % create 6x3

ur.r = 0.005;                                            % set blend radius
ur.movej(pose)                                           % move through all positions
```

UR_robot.movel

Linear movement of TCP in cartesian.

`ur.movel(pose)`

Moves the TCP from the current position to the position specified in the *pose* argument. Acceleration and speed of the move are set in properties *a_tool [m/s^2]* and *v_tool* [m/s].

Pose

- as a vector (6x1 – double)
  - format: [x ; y ; z ; rx ; ry ; rz]
  - x, y, z – TCP position in the base frame
  - rx, ry, rz - the orientation of the TCP, given in axis-angle notation
- as a matrix (6xn – double)
  - it is possible to send more positions in a single matrix
  - every column is one position, the format is explained above
  - in this format, it is possible to use a blend radius between each position which causes a continuous motion

Examples

```
pose1 = [0.0300;-0.3789;0.3959;-0.0269;3.1315;0.0000];    % defined poses
pose2 = [-0.0634;-0.4533;0.3731;0.1756;3.1366;0.0000];
pose3 = [-0.0066;-0.5688;0.3906;-0.1788;3.1365;0.0000];

ur.v_tool = 0.01;                                         % set TCP speed
ur.a_tool = 0.05;                                         % set TCP acceleration
ur.movel(pose1)                                           % linear move to the pose1

pose = [pose1,pose2,pose3];                               % create 6x3

ur.r = 0.005;                                             % set blend radius
ur.movel(pose)                                            % move through all positions
```

UR_robot.movep

Linear movement of TCP in cartesian space with constant tool speed. Ideal for welding, glueing, drawing, etc.

`ur.movep(pose)`

Moves the TCP from the current position to the position specified in the *pose* argument. Acceleration and speed of the move are set in properties *a_tool [m/s^2]* and *v_tool* [m/s]. If the blend radius is set, the robot moves through the positions with constant speed.

pose

- as a vector (6x1 – double)
  - format: [x ; y ; z ; rx ; ry ; rz]
  - x, y, z [m] – TCP position in the base frame
  - rx, ry, rz [rad] - the orientation of the TCP, given in axis-angle notation
- as a matrix (6xn – double)
  - it is possible to send more positions in a single matrix
  - every column is one position, the format is explained above
  - in this format, it is possible to use a blend radius between each position which causes a continuous motion

Examples

```
pose1 = [0.0300;-0.3789;0.3959;-0.0269;3.1315;0.0000];   % defined poses
pose2 = [-0.0634;-0.4533;0.3731;0.1756;3.1366;0.0000];
pose3 = [-0.0066;-0.5688;0.3906;-0.1788;3.1365;0.0000];

ur.v_tool = 0.01;                                        % set TCP speed
ur.a_tool = 0.05;                                        % set TCP acceleration
ur.movep(pose1)                                          % linear move to the pose1

pose = [pose1,pose2,pose3];                              % create 6x3

ur.r = 0.005;                                            % set blend radius
ur.movep(pose)                                           % move through all positions
```

UR_robot.set_config

Linear movement in joint-space.

`ur.set_config(config)`

Moves the robot from the current position to the position specified in the *config* argument as the position of each joint. Acceleration and speed of the move are set in the properties *a_joint [rad/s^2]* and *v_joint* [rad/s].

config

- as a vector (6x1 – double)
  - format: [q1 ; q2 ; q3 ; q4 ; q5 ; q6]
  - q1 - q6 [rad] – the rotation angle of each joint numbered from the bottom of the robot
- as a matrix (6xn – double)
  - it is possible to send more positions in a single matrix
  - every column is one position, the format is explained above
  - in this format, it is possible to use a blend radius between each position which causes a continuous motion

Examples

```
config1 = [pi/2;pi/2;0;0;0;0];                % defined configs
config2 = [0;pi/2;0;-pi/2;0;0];
config3 = [0;pi/2;0;-pi/2;pi/2;0];

ur.v_joint = 0.1;                             % set joint speed
ur.a_tool = 0.4;                              % set joint acceleration
ur.set_config(config1)                        % move to the config1

config = [config1,config2,config3];           % create 6x3 matrix

ur.r = 0.005;                                 % set blend radius
ur.set_config(config)                         % move through all positions
```

UR_robot.movel_tcp

Linear movement of TCP in Cartesian space relative to the current TCP position.

`ur.movel_tcp(pose)`

Moves TCP from the current position to the position specified in the *pose* argument along the linear path. Acceleration and speed of the move are set in the properties *a_tool [m/s^2]* and *v_tool* [m/s].

pose (6x1 – double)

- format: [x ; y ; z ; rx ; ry ; rz]
- x, y, z [m] – TCP position relative to the current TCP position
- rx, ry, rz [rad] - the orientation of the TCP relative to the current TCP position; given in axis-angle notation

Examples

```
pose1 = [0.45;0;0;0;0;0];          % defined pose
ur.v_tool = 0.01;                  % set TCP speed
ur.a_tool = 0.05;                  % set TCP acceleration
ur.movel_tcp(pose1);               % linear move 0.45 m in x axis relative
                                   % to the current TCP position
```

UR_robot.movej_tcp

Linear movement in joint-space to the position given relative to the current TCP position.

`ur.movej_tcp(pose)`

Moves the robot from the current position to the position specified in the *pose* argument along the unspecified Cartesian space path. Acceleration and speed of the move are set in the properties *a_joint [rad/s^2]* and *v_joint* [rad/s].

pose (6x1 – double)

- format: [x ; y ; z ; rx ; ry ; rz]
- x, y, z [m] – TCP position relative to the current TCP position
- rx, ry, rz [rad] - the orientation of the TCP relative to the current TCP position; given in axis-angle notation

Examples

```
pose1 = [0.03;-0.37;0.39;-0.02;3.13;0];        % defined poses


ur.v_tool = 0.01;                              % set TCP speed
ur.a_tool = 0.05;                              % set TCP acceleration
ur.movej_tcp(pose1);                           % move 0.45 m in x axis relative to
                                               % the current TCP position
```

UR_robot.move_joint

Moves a chosen joint to an angle relative to the current position.

`ur.move_joint(joint_num, angle)`
Moves a chosen joint specified in the *joint_*num argument to the relative angle specified in the *angle* argument. Acceleration and speed of the move are set in the properties *a_joint [rad/s^2]* and *v_joint* [rad/s].

joint_num (double)
- number in range 1 – 6 specifying which joint is supposed to move
- joints are numbered from the bottom of the robot

angle (double)
- angle [rad] measured from the current joint position
- value can be a negative number

Examples

```
ur.v_joint = 0.01;                  % set joint speed
ur.a_joint = 0.05;                  % set joint acceleration

ur.move_joint(6,pi/2);              % move the 6th joint (wrist 3) pi/2 rad
```

UR_robot.freedrive_on

Sets robot in freedrive mode.

`ur.freedrive_on`
In this mode the robot can be moved around by hand in the same way as by pressing the "freedrive" button. The robot will not be able to follow a trajectory in this mode.

The mode is ended by *freedrive_off* command or automatically after 100 s.

UR_robot.freedrive_off

Sets robot back in normal position control mode after freedrive mode.

`ur.freedrive_off`

Examples

```
% The example shows how to use the freedrive mode to measure certain positions into
% which the robot is guided by hand.

ur.freedrive_on;            % switch to the freedrive mode

pose1 = ur.pose;            % get the robot position and save it to the pose1 variable
pose2 = ur.pose;            % get the robot position and save it to the pose2 variable

ur.freedrive_off;           % switch back to the normal mode
```

UR_robot.stop

Immediately stops any movement of the arm

`ur.stop`
Any movement of the arm is stopped no matter which move instruction was executed. Also interrupts any command which is currently executed.

Examples

```
forceAxis = [0 0 1 0 0 0];
forces = [0 0 -5 0 0 0];
lim = [0.1 0.1 0.01 0.3 0.3 0.3];        % define parameters

ur.apply_force(forceAxis,forces,lim);    % apply force
pause(5)                                 % wait 5 s
ur.stop                                  % interrupts apply_force command and stops
                                         %the movement
```

UR_robot.zero_force

Zeroes the TCP force/torque measurement from the built-in force/torque sensor

`ur.zero_force`
It is recommended to insert a short pause command of at least 0.05 s after zeroing the sensor.

Examples

```
ur.zero_force;      % force sensor is set to zero
pause(0.1);         % program waits 0.1 s
```

UR_robot.apply_force

Set robot to be controlled in force mode

`ur.apply_force(selection_vector, wrench, limits)`
It is possible to make a robot produce a certain force in a certain direction. If there is an obstacle, the robot pushes it by specified force. If there is no resistance, the robot starts to move in the direction of the specified force with the maximum specified speed.

Command is executed until the robot is stopped by **stop command** or interrupted by other instruction!

selection_vector (6x1 – double)
- possible values: 0 or 1
- 1 stands for compliant axis, i.e. controlled by force
- 0 stands for non-compliant axis, i.e. controlled by position (standard)
- format: [x ; y ; z ; rx ; ry ; rz]

wrench (6x1 – double)
- setting applied force/torque in each axis
- format: [Fx ; Fy ; Fz ; Mx ; My ; Mz]
- values have no effect for non-compliant axes!

limits (6x1 – double)
- For compliant axes, these values are the maximum allowed tcp speed along/about the axis
- For non-compliant axes, these values are the maximum allowed deviation along/about an axis between the actual tcp position and the one set by the program
- format: [x ; y ; z ; rx ; ry ; rz]

Example

```
axes = [1,0,0,0,0,0];                    % x axis is compliant
f = [15,0,0,0,0,0];                      % robot produces 15 N force
lim = [0.01,0.05,0.05,0.1,0.1,0.1];      % max speed in x is 0.01 m/s

ur.zero_force;                           % force sensor is set to zero
pause(0.1)
ur.apply_force(axes, f, lim)             % force mode begin
pause(5);                                % program waits 5 s
ur.stop                                  % force mode stops
```

UR_robot.force_movel

Execute movel command, but in the force-controlled mode

`ur.force_movel(selection_vector,wrench,limits,tgt_pose)`
Robot executes standard movel command to the position specified in tgt_pose. However, if some axes are set to compliant, the robot will produce a specified force in these axes. The movel trajectory is therefore unknown.

selection_vector (6x1 – double)

- possible values: 0 or 1
- 1 stands for compliant axis, i.e. controlled by force
- 0 stands for non-compliant axis, i.e. controlled by position
- format: [x ; y ; z ; rx ; ry ; rz]

wrench (6x1 – double)

- setting applied force/torque in each axis
- format: [Fx ; Fy ; Fz ; Mx ; My ; Mz]
- values have no effect for non-compliant axes!

limits (6x1 – double)

- For compliant axes, these values are the maximum allowed tcp speed along/about the axis
- For non-compliant axes, these values are the maximum allowed deviation along/about an axis between the actual tcp position and the one set by the program
- format: [x ; y ; z ; rx ; ry ; rz]

tgt_pose (6x1 – double)

- format: [x ; y ; z ; rx ; ry ; rz]
- x, y, z [m] – TCP position in the base frame
- rx, ry, rz [rad] - the orientation of the TCP, given in axis-angle notation

Example

```
axes = [1,0,0,0,0,0];                    % x axis is compliant
f = [15,0,0,0,0,0];                      % robot produces 15 N force
lim = [0.05,0.05,0.05,0.1,0.1,0.1];      % max speed in x is 0.05 m/s
P = [0.05;-0.49;0.28;0;-pi;0];           % final position
ur.zero_force;                           % force sensor is set to zero
pause(0.1)
ur.force_movel(axes, f, lim, P)          % force mode begin
```

UR_robot.RG2

Controlling the OnRobot RG2 gripper

`ur.RG2(wide,force,payload,setpayload,depthcomp)`

Command for controlling the width of gripper or force of the grip. Before the use of the command, the gripper must be activated by setting the property *gripper* to 'on'.

wide (double)

- target width that the RG2 will try to reach
- if the specified force is achieved, the RG2 will stop at a width that differs from the target width

force (double)

- target force that the RG2 will try to achieve
- if the target width is reached before the target force, the RG2 will stop moving and the target force may not be achieved at the anticipated width

payload (double)

- the mass of the manipulated object
- unit: kg
- do not have effect if the argument *setpayload* is set to False

setpayload (True/False)

- enable when the mass of the object should be taken into account

depthcomp (True/False)

- when is enabled, the robot arm will try to make a movement that compensates for the circular movement of the finger arms

Example

```
ur.gripper = 'on';                  % gripper is activated!

W = 80;                             % jaws are open to 80 mm
F = 0;                              % the grip force is 0 N
load = 0;                           % no load is carried
setLoad = false;                    % no load is
depth = true;                       % depth compensation is on

ur.RG2(W, F, load, setLoad, depth)  % x axis is compliant
```

4.3. Properties

UR_robot.a_joint

Joint acceleration limit [rad/s^2]

`ur.a_joint = a`
Set *a* value as the maximum joint acceleration [rad/s^2].

`a = ur.a_joint`
Writes the currently set joint acceleration to *a* variable [rad/s^2].

Default value = 0.35 rad/s^2

UR_robot.a_tool

TCP acceleration limit [m/s^2]

`ur.a_tool = a`
Set *a* value as the maximum TCP acceleration [m/s^2].

`a = ur.a_tool`
Writes the currently set TCP acceleration to *a* variable [m/s^2].

Default value = 0.05 m/s^2

UR_robot.v_joint

Joint speed limit [rad/s].

`ur.v_joint = v`
Set *v* value as the maximum joint speed [rad/s].

`v = ur.v_joint`
Writes the currently set joint speed to *v* variable [rad/s].

Default value = 0.085 rad/s

UR_robot.v_tool

TCP speed limit [m/s].

`ur.v_tool = v`
Set *v* value as the maximum TCP speed [m/s].

`v = ur.v_tool`
Writes the currently set joint speed to *v* variable [m/s].

Default value = 0.05 m/s

UR_robot.r

Blend radius [m]

`ur.r = radius`
Sets *radius* value as the blend radius. Blend radius is used for continuous movement between trajectory points. The exact position is not reached however! Poses must be sent at once as a matrix, see move commands.

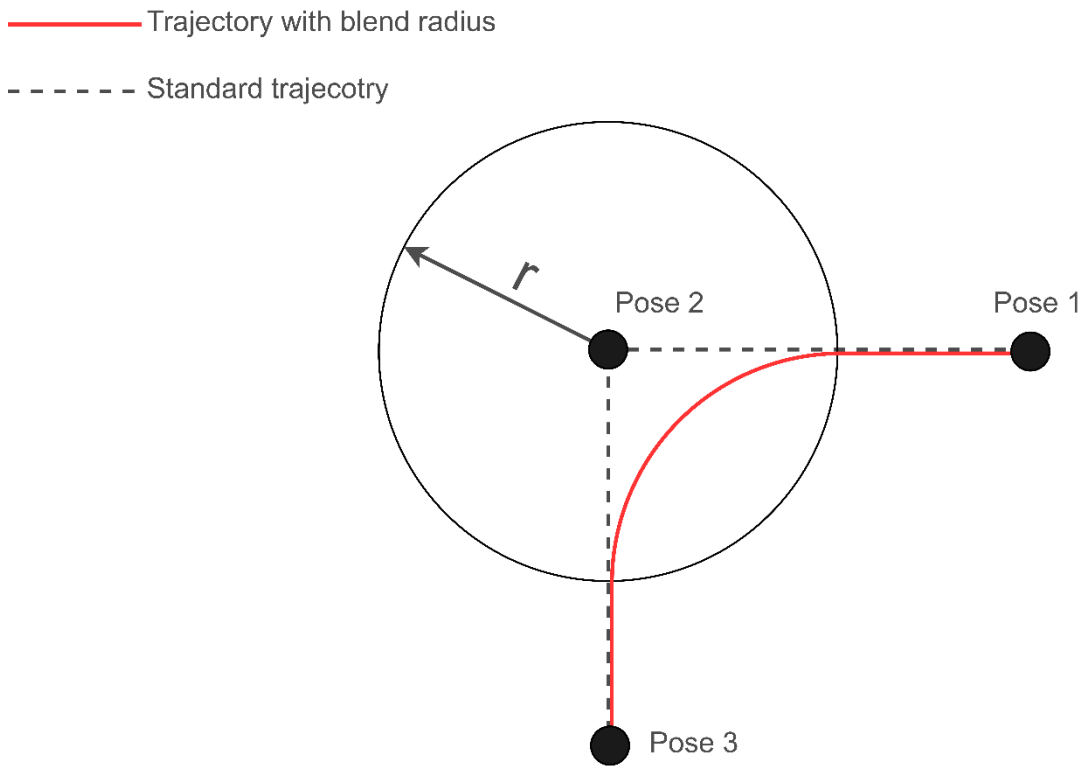

Fig. 7) Trajectory with blend radius

UR_robot.n_tcp_data

Determine which predefined tcp_data are active

`ur.n_tcp_data = n`
Activate the n^{th} set of tcp data defined in the *tcp_data* property. There are predefined three sets (default (1), RG2 gripper (2), user-defined (3)). For defining tcp_data, see the *tcp_data* property.

UR_robot.tcp_data

Definition of TCP and characteristics of objects attached to the robot's flange. There are predefined three sets (default (1), RG2 gripper (2), user-defined (3)).

`ur.tcp_data.mass(n) = m`
Consider *m* [kg] as the mass of the currently attached object to the flange (e.g. gripper). The *m* value is stored in the n^{th} set of tcp_data.

`ur.tcp_data.CoG(n) = cog`
Consider *cog* as the centre of gravity of the object currently attached to the robot's flange. The *cog* value is stored in the n^{th} set of tcp_data.

cog (3 x double)
- Format: [x, y, z]
- Units [m]
- Distances measured from the default TCP in the centre of the robot's flange

`ur.tcp_data.pose(n) = P`
Shift the TCP to the position defined in the *P*. The *P* value is stored in the n^{th} set of tcp_data.

P (6 x double)
- format: [x ; y ; z ; rx ; ry ; rz]
- x, y, z [m] – TCP position relative to the default TCP in the centre of the robot's flange
- rx, ry, rz [rad] - the orientation of the TCP relative to the default TCP in the centre of the robot's flange

Example

```
ur.n_tcp_data = 1;                          % set 1 is activated

ur.tcp_data.mass(3) = 1.5;                  % set 3 is edited
ur.tcp_data.CoG(3) = [0.01 0 0.01];
ur.tcp_data.pose(3) = [0.03 0 0.03 0 0 0];

ur.n_tcp_data = 3;                          % set 3 is activated
```

UR_robot.wait_mode

Determine whether Matlab waits until the move instruction is executed or not

`ur.wait_mode = mode`
Writes mode value to the *wait_mode* property. The mode determines whether Matlab stops the program until the move instruction is executed or sends the move instruction and continues in the program.

mode (2 options)

'on'
- When the move instruction is sent, the program waits until it is executed
- Default

'off'
- The program continues immediately after the move instruction is sent

Example

```
ur.movej(pose1);                % program waits until the pose1 is reached
ur.movej(pose2);                % program waits until the pose2 is reached
ur.wait_mode = 'off';           % wait_mode if set to 'off'
ur.movej(pose3);                % program do not wait for reaching the pose3
for i = 1 : 500                 % the loop measures the force during the movement
    Fx(i) = ur.force(1)
    pause(0.002);
end
```

UR_robot.gripper_fingertips

Fingertips offset on the OnRobot RG2 gripper

`ur.gripper_fingertips = tip`
Writes *tip* value [mm] to the gripper_fingertips property. It specifies the distance from the inside of the gripper's aluminum fingertip to the reference point on the attached fingertip, see Fig. 8. The value is essential for reading the *gripper_pose* and setting correct width. Default value is 4,6 mm.

Fig. 8) Fingertips offset

UR_robot.gripper

Enable/disable using OnRobot GR2 gripper

`ur.gripper = mode`
Enable using the gripper, sets appropriate TCP and enable reading gripper width.

mode (2 options)

'on'
- It is possible to read actual width
- TCP is set to the end of the gripper

'off'
- Default
- Gripper is disabled
- Default TCP is set

4.4. Read-only properties

UR_robot.s1

`ur.s1`
Access the tcpip object created for port 29999. It can be used for sending or reading data, but it cannot be edited.

UR_robot.s2

`ur.s2`
Access the tcpip object created for port 30003. It can be used for sending or reading data, but it cannot be edited.

UR_robot.s3

`ur.s3`
Access the tcpip object created for port 30001. It can be used for sending or reading data, but it cannot be edited.

UR_robot.pose

Read the actual position of the TCP

`P = ur.pose`
Save the current TCP position to the *P*.

P (6x1 – double)

- format: [x ; y ; z ; rx ; ry ; rz]
- x, y, z [mm] – TCP position in the base frame
- rx, ry, rz [rad] - the orientation of the TCP, given in axis-angle notation

UR_robot.config

Read the actual position of the robot's joints

`J = ur.config`
Save the current joints positions to the *J*.

J (6x1 – double)

- format: [q1 ; q2 ; q3 ; q4 ; q5 ; q6]
- q1 - q6 [rad] – the rotation angle of each joint numbered from the bottom of the robot

UR_robot.force

Read the actual force applied to the TCP

`F = ur.force`
Save the current force applied to the TCP to the *F*.

F (6x1 – double)

- format: [Fx ; Fy ; Fz ; Mx ; My ; Mz]
- Fx, Fy, Fz [N] – forces in axis of TCP (Base coordinates)
- Mx, My, Mz [Nm] – torques in axis of TCP (Base coordinates)

UR_robot.gripper_pose

Read the actual width of the gripper jaws

`P = ur.gripper_pose`
Save the current width of the gripper to the P [mm]. The fingertips offset value is included. The value cannot be read if the gripper is not activated by setting property *gripper* to 'on'.

UR_robot.active_tcp

Display currently active set of tcp_data

`n = ur.active_tcp`
Save the number of the active set of tcp_data to the *n*. There are predefined three sets (default (1), RG2 gripper (2), user-defined (3)). For defining tcp_data, see the *tcp_data* property.

5. Shortcuts

TCP	Tool centre point	The point at the end of the robotic arm to which the coordinates are related.