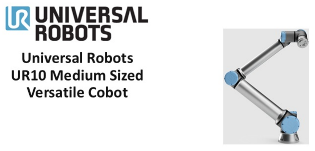
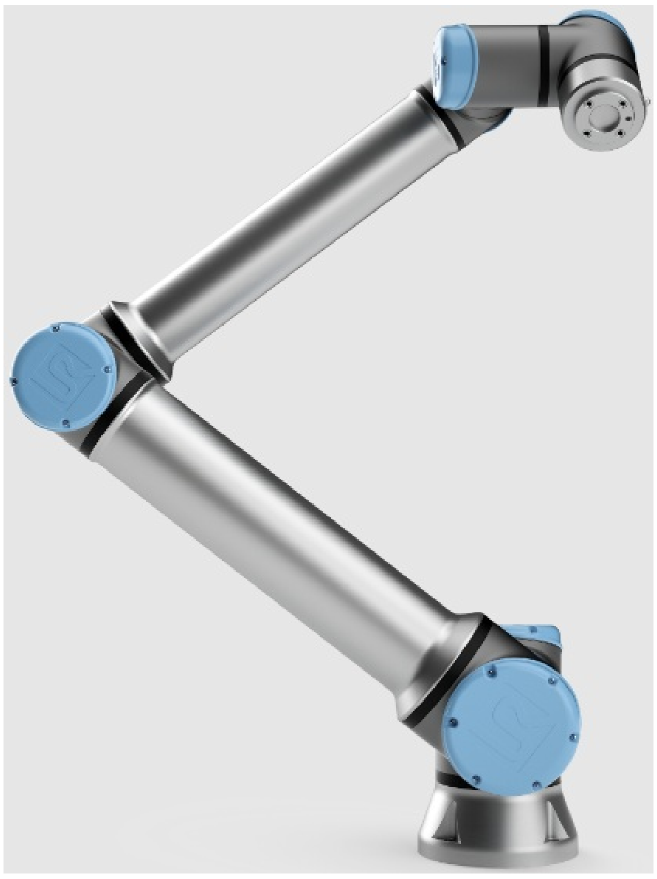


Universal Robots UR10 Medium Sized Versatile Cobot User Guide

Contents

Universal Robots UR10 Medium Sized Versatile Cobot



Product Specifications

- Software Version: 5.15.0
- Supported Robot Models: UR10, UR16, UR20
- Control Box Version: 5.5 for e-Series robots
- Maximum Tool Force and Elbow Force: 400N

Product Usage Instructions

Payload Transition Time
Users can define a transition time for payload changes to prevent sudden jumps in the robot motion, especially when handling heavy objects or using a vacuum gripper.

1. Access the Set Payload node in the PolyScope GUI.
2. Set the desired transition time in the input field.
3. Ensure the transition time is greater than zero to enable smoother payload changes.

Max Force for Elbow and Tool
The maximum Tool Force and Elbow Force limits can now be adjusted up to 400N for UR10, UR16, and UR20 models.

1. Navigate to the Robot limits settings in the PolyScope GUI.
2. Update the Tool Force and Elbow Force values as needed.
3. Save the changes to apply the new force limits.

EtherNet/IP and Conveyor Tracking Installation

The installation screens for EtherNet/IP and Conveyor Tracking have been improved for better usability.

1. Follow the on-screen instructions to configure EtherNet/IP or Conveyor Tracking settings.
2. Take note of the layout changes, such as checkbox replacements and improved visuals.

URCap Software Platform Updates
The URCap API and SDK have received enhancements for better payload management and support for new robot types.

1. Refer to the updated API documentation for details on payload transition time and new robot types support.
2. Explore the URCap SDK improvements for better code readability and communication handling.

URScript Functions
Structs and variable length lists can be utilized in URScript for data aggregation and manipulation.

1. Create structs using the struct function with named arguments.
2. Manipulate lists with fixed capacity and length attributes.
3. Utilize the estimate_payload function to estimate payload mass and center of gravity based on poses and force torque recordings.

Frequently Asked Questions (FAQ)

- **Q: Can the maximum Tool Force and Elbow Force limits be adjusted on all robot models?**
  A: No, the increased maximum limits of 400N apply only to UR10, UR16, and UR20 models.
- **Q: How can I estimate payload mass and center of gravity within a robot program?**
  A: Utilize the estimate_payload URScript function with a list of poses and force torque recordings to estimate payload properties.

Release Versions:

- UR Software Update: 5.15.0
- URCap Software Platform:
  - URCap API: 1.14.0
  - URCap SDK: 1.14.0
- URSim Linux: 5.15.0
- URSim Virtual Machine: 5.15.0 – 2.0.285
- User Manuals: 5.15.0
- Support Log Reader: xxx

Key Features

- Payload Transition Time (GUI)
  It is possible to define a duration for a payload change in the Set Payload node.
- Increased max force safety limit for elbow and tool for UR10, UR16, UR20
  Max allowed setting increased from 250N to 400N.

- Structs (complex data types) – Release for production

  In URScript, a set of variables can be aggregated into struct, which is transferred and stored as a single variable.
- Variable length lists in URScript

  Lists can be defined with length and capacity. New operations on lists supported.
- Methods for list operations, struct members access, and matrices

  New methods enable list length and content manipulation.
- The estimate_payload() URScript function

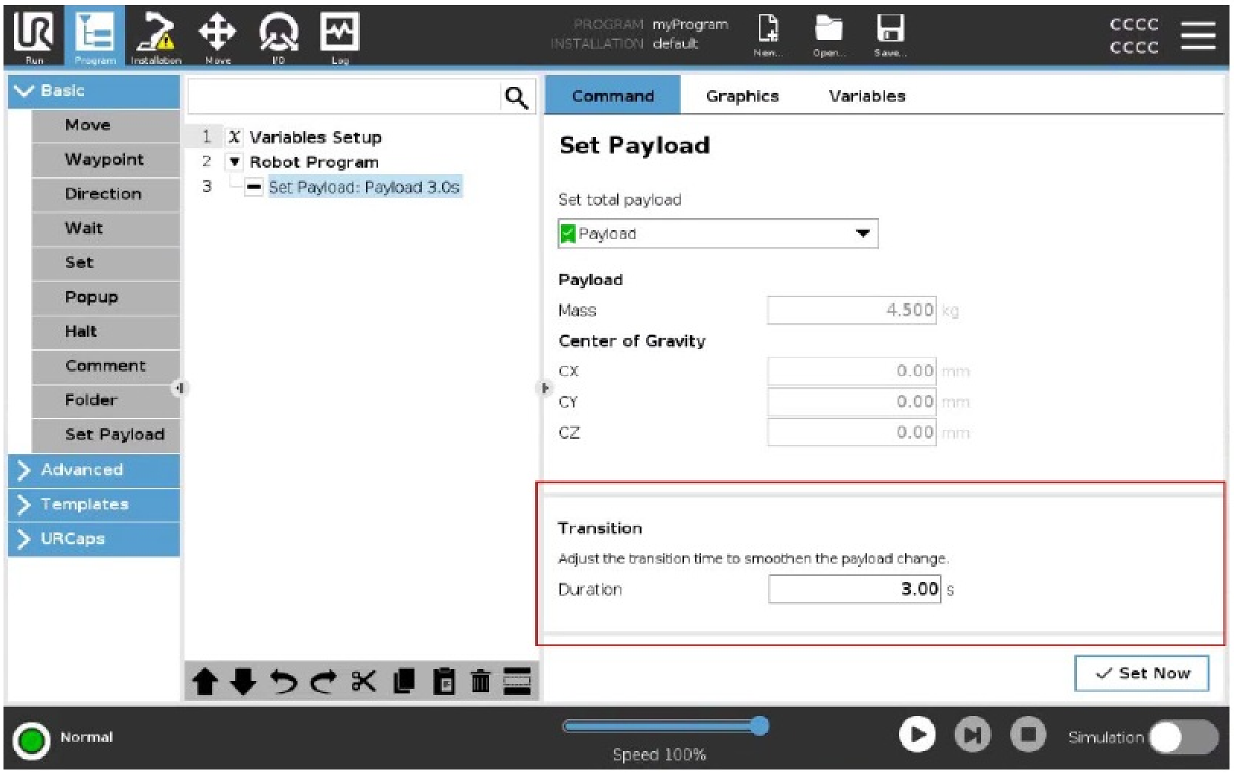
  Payload can be estimated during execution of a program.
- Flexible Ethernet/IP adapter assemblies – Engineering Preview

  Enables CNC machine connectivity with custom Ethernet/IP adapter data layouts.

PolyScope GUI

Payload Transition Time

We added a new input field in the Set Payload node, which allows users to define a duration for a payload change.
Setting a transition time larger than zero prevents the robot from doing a small "jump" when the payload changes and is easier on the joints. This is useful when picking up or releasing heavy objects or using a vacuum gripper.



1 Variables Setup
2 Robot Program
3 Set Payload: Payload 3.0s

Set Payload

Set total payload: Payload

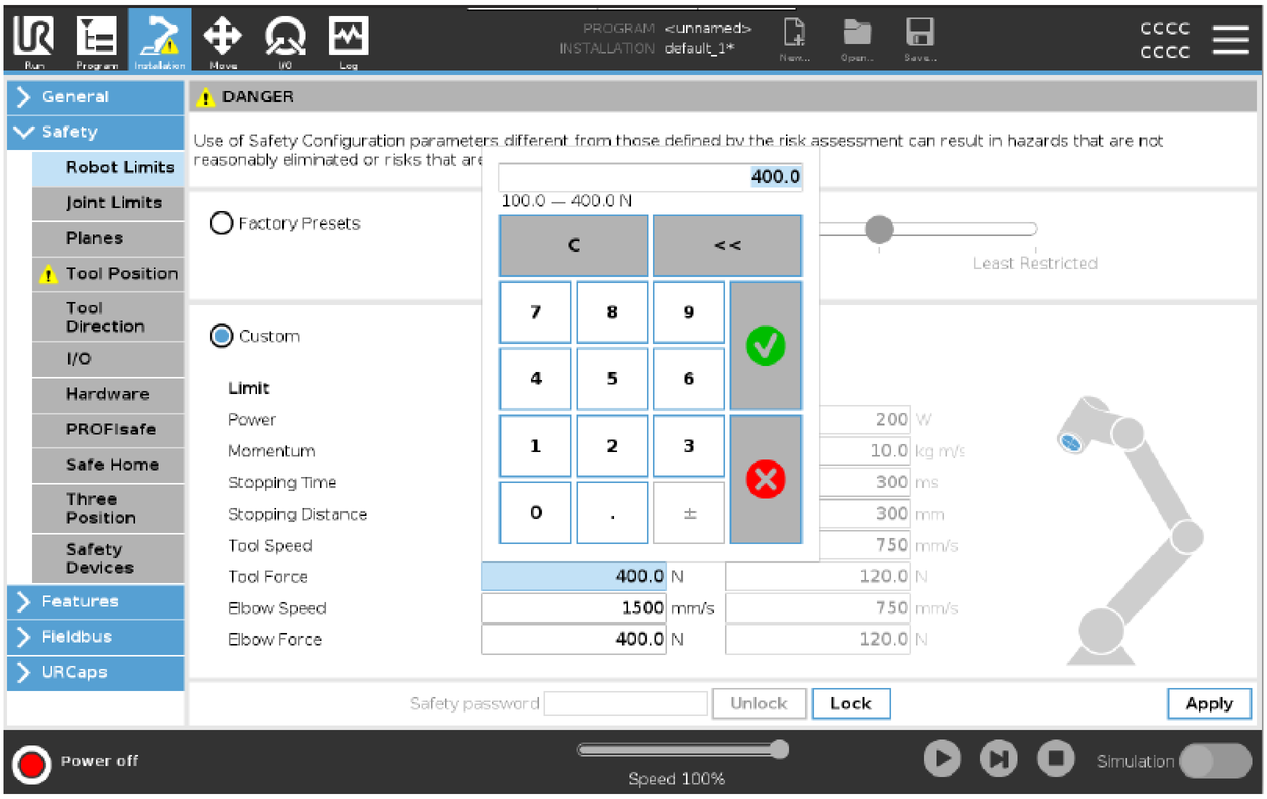
Payload
Mass 4.500 kg
Center of Gravity
CX 0.00 mm
CY 0.00 mm
CZ 0.00 mm

Transition
Adjust the transition time to smoothen the payload change.
Duration 3.00 s

Note that the transition time is now also part of the action for the "Set Now" button.

Max force for elbow and tool

The maximum value for "Tool Force" and "Elbow Force" in "Robot limits" can now be set to a maximum of 400N instead of 250N. This change only applies for UR10, UR16 and UR20.



PROGRAM <unnamed>
INSTALLATION default_1*

DANGER

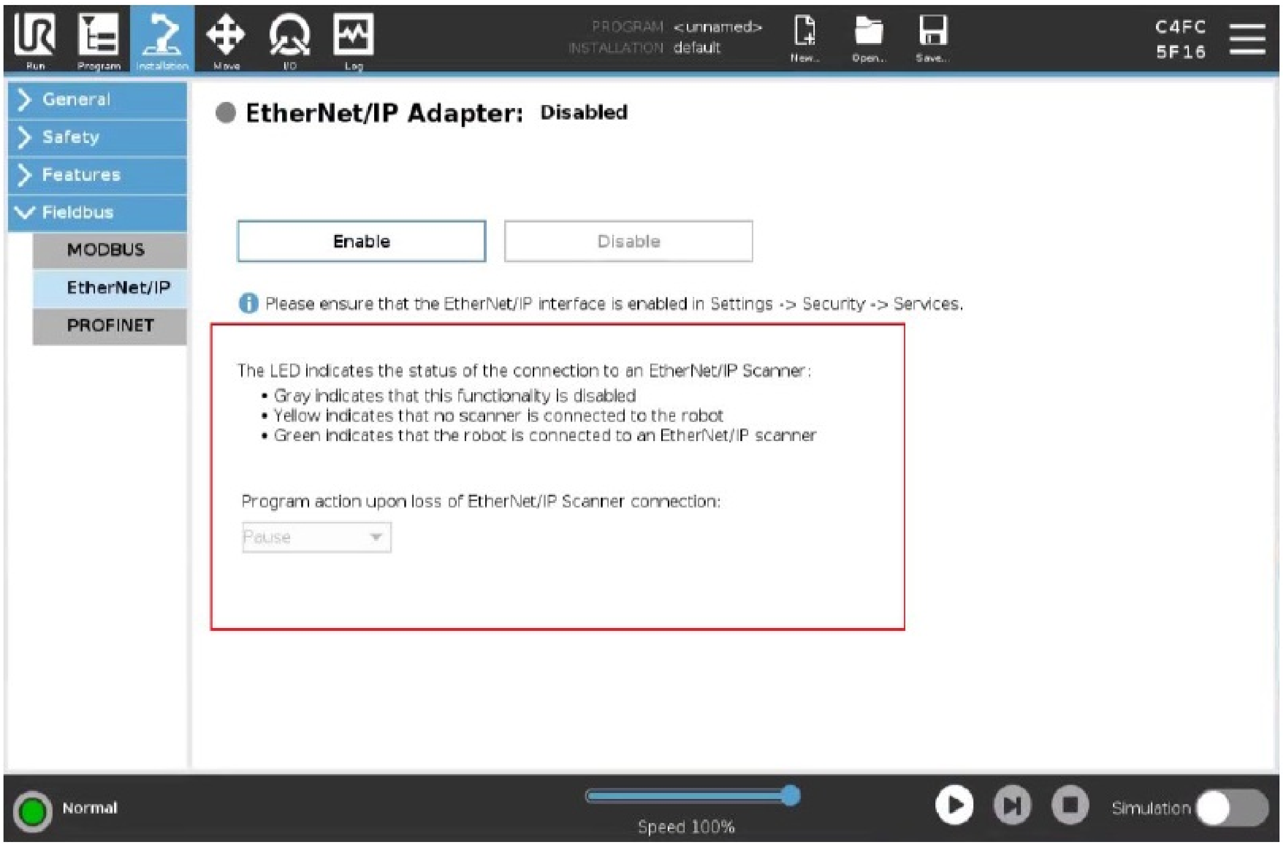
Use of Safety Configuration parameters different from those defined by the risk assessment can result in hazards that are not reasonably eliminated or risks that are

400.0
100.0 — 400.0 N

Limit			
Power			200 W
Momentum			10.0 kg m/s
Stopping Time			300 ms
Stopping Distance			300 mm
Tool Speed			750 mm/s
Tool Force	400.0	N	120.0 N
Elbow Speed	1500	mm/s	750 mm/s
Elbow Force	400.0	N	120.0 N

Safety password

Improved EtherNet/IP installation screen
The layout of the EtherNet/IP installation screen has been improved and aligned with the PROFINET installation screen.



PROGRAM <unnamed>
INSTALLATION default

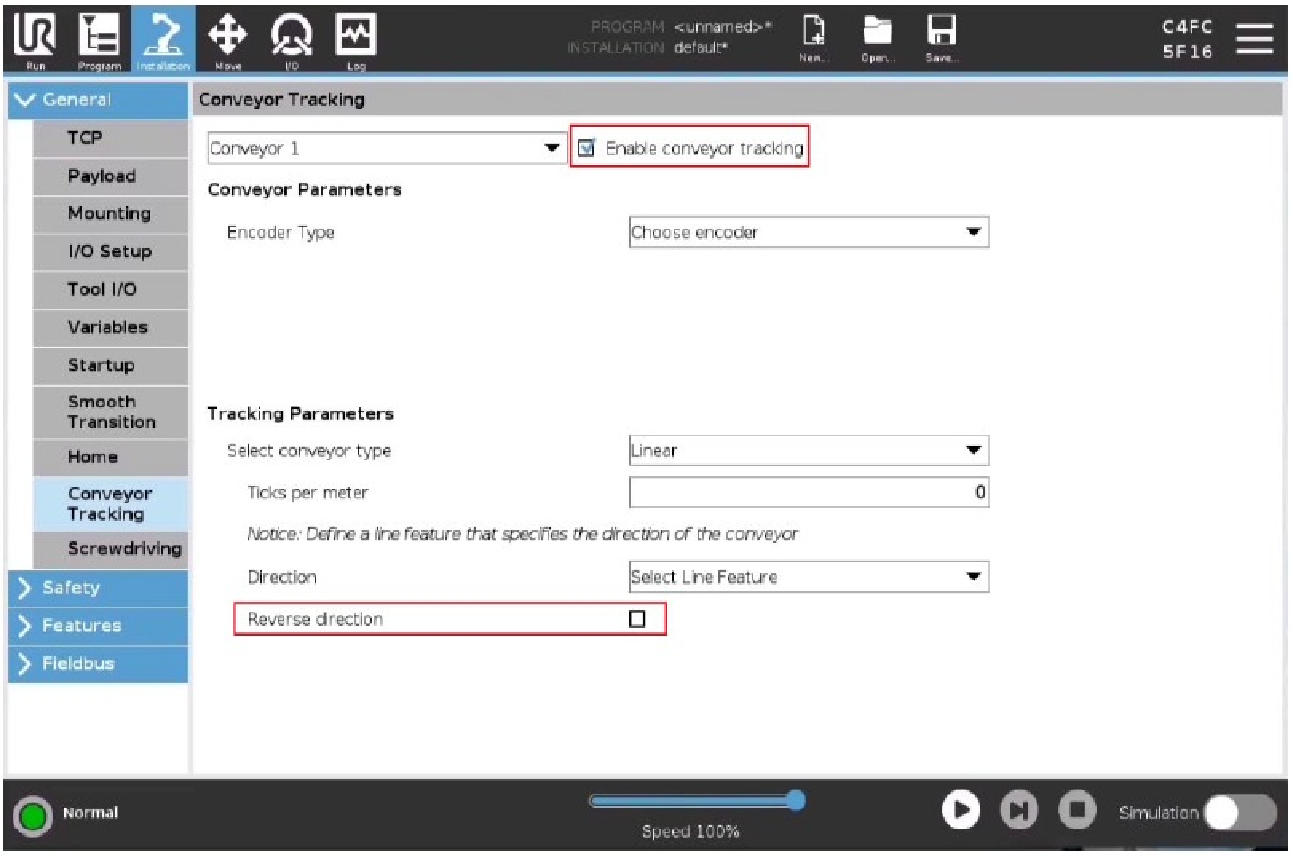
EtherNet/IP Adapter: Disabled

Please ensure that the EtherNet/IP interface is enabled in Settings -> Security -> Services.

The LED indicates the status of the connection to an EtherNet/IP Scanner:
- Gray indicates that this functionality is disabled
- Yellow indicates that no scanner is connected to the robot
- Green indicates that the robot is connected to an EtherNet/IP scanner

Program action upon loss of EtherNet/IP Scanner connection:
Pause

Improved Conveyer Tracking installation screen
The layout of the Conveyor Tracking installation screen has been improved.
In the configuration of linear conveyors, the previous “Reverse” direction button has been replaced with a checkbox, so that it is easy to see whether the direction of the selected Line feature is reversed.



Conveyor Tracking

Conveyor 1 [✓ Enable conveyor tracking]

Conveyor Parameters

Encoder Type	Choose encoder

Tracking Parameters

Select conveyor type	Linear
Ticks per meter	0

Notice: Define a line feature that specifies the direction of the conveyor

Direction	Select Line Feature
Reverse direction	☐

URCap Software Platform

URCap API:

Added support for payload transition time

- URCaps can create Set Payload node configurations with a specified transition time
- URCaps can read the transition time of a Set Payload node configuration
- API changes (packagecom.ur.urcap.api.domain.program.nodes.builtin.configurations.setpayloadnode):
  - New methods createSelectionConfig(Payload, Time, ErrorHandler<Time>) and createCustomParametersConfig(Mass, Position, Time, ErrorHandler<Time>) in the existing PayloadNodeConfigFactory interface
  - New getTransitionTime() method in the existing SelectionPayloadNodeConfig and CustomParametersPayloadNodeConfig interfaces

Added support for new robot types

- Extended the existing RobotType enum in the RobotModel interface with the new UR20 and UR30 robot types
- Deprecated the getRobotSeries() method and the RobotSeries enum in the RobotModel interface
  - Note:
    - The RobotType enum will no longer be extended with new values when new robot series (generations) are introduced.
    - The getRobotSeries() method will return the UNKNOWN enum value, if the underlying robot does not belong to either the CB3 or e-Series robot series. This will, for instance, be the case, if the robot is a UR20. The method can only be used to determine, if a robot is a CB3 robot.

    - For more information, see the Javadoc for the getRobotSeries() method.

URCap SDK:

URCap Samples

- Improved readability of the POM file (pom.xml) for all samples
- Updated the newURCap.sh script to the generate new URCap projects with the improved POM file format
- Improved the My Daemon Swing sample:
    - Simplified the code and improved readability
    - Improved the handling of the communication with the daemon and updating of the UI
    - Changes are located in the code for both the C++/Python daemon and the Java URCap bundle

Controller

Introduced support control box version 5.5 for e-Series robots

Structs (complex data types) – Release for production
In URScript, a set of variables can be aggregated into struct, and thus transferred and stored as a single variable.

Structs can be obtained through multiple means:

- Using the struct() function
- Executing an RPC call that returns a struct

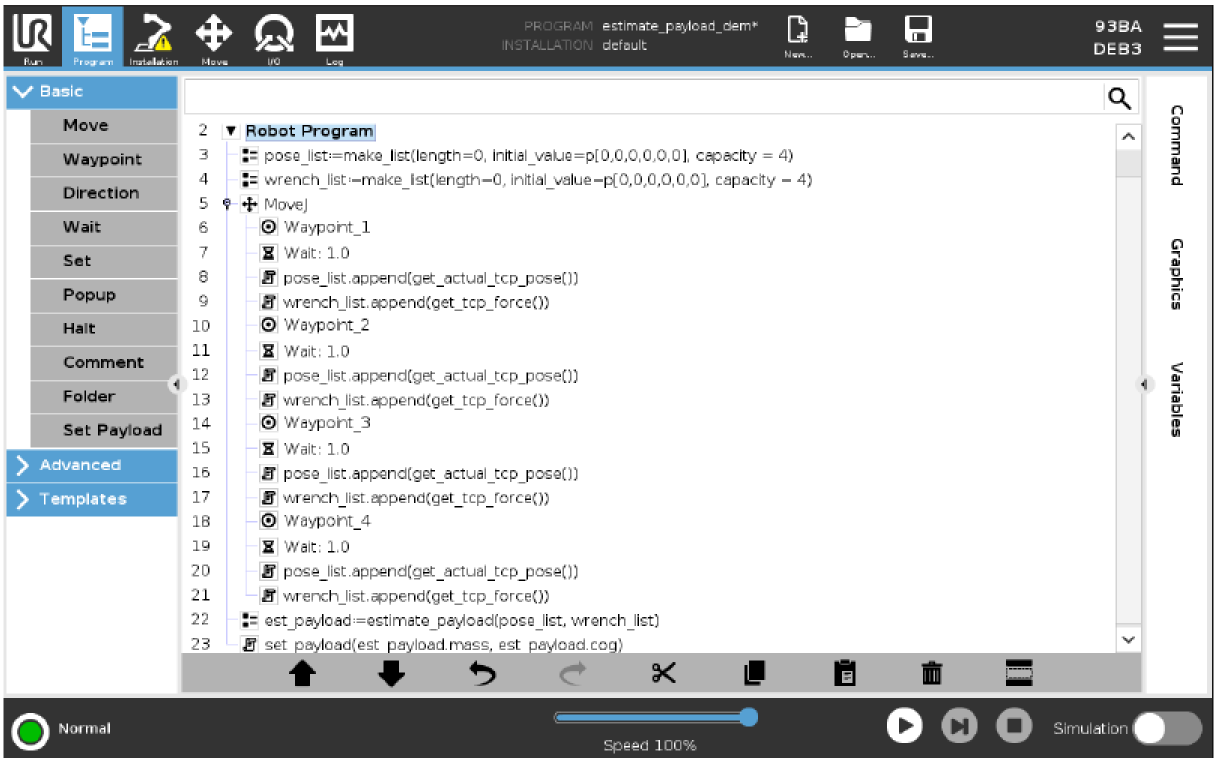
The struct function takes one or more named arguments, and each argument name becomes a member in the struct. All values must be initialized by value, and the type of the value cannot be changed subsequently.

Create a struct:
myStruct = struct(identifier1 = 1, identifier2 = 2, myMember = “Hello structs”, listMember = [1,2,3])

Use a member:
myVar = myStruct.myMember

Variable length lists
A list object in URScript has two attributes: length and capacity. The length indicates how many elements the list currently holding. The capacity tells how many elements the list can hold maximum. Once declared, the capacity of the list cannot be changed. Examples of struct and list manipulation could be found in URScript manual and support article “URScript examples for manipulating structs and lists”.

estimate_payload URScript function
Allows the payload mass and center of gravity to be estimated inside a robot program, based on a list of poses and force torque recordings.



```
▼ Robot Program
    pose_list:=make_list(length=0, initial_value=p[0,0,0,0,0,0], capacity = 4)
    wrench_list:=make_list(length=0, initial_value=p[0,0,0,0,0,0], capacity = 4)
    MoveJ
        Waypoint_1
        Wait: 1.0
        pose_list.append(get_actual_tcp_pose())
        wrench_list.append(get_tcp_force())
        Waypoint_2
        Wait: 1.0
        pose_list.append(get_actual_tcp_pose())
        wrench_list.append(get_tcp_force())
        Waypoint_3
        Wait: 1.0
        pose_list.append(get_actual_tcp_pose())
        wrench_list.append(get_tcp_force())
        Waypoint_4
        Wait: 1.0
        pose_list.append(get_actual_tcp_pose())
        wrench_list.append(get_tcp_force())
    est_payload:=estimate_payload(pose_list, wrench_list)
    set_payload(est_payload.mass, est_payload.cog)
```

Flexible Ethernet/IP adapter in URScript – Engineering Preview

NOTE: Feature is not recommended for production use. Script API and functionality can change, and production version will be released in the next major release. Contact Universal-Robots application engineer for further support.
This functionality allows programmer to add custom Ethernet/IP connections on top of connections that contains Universal-Robots real-time data.

Interface consists of 2 parts:

- Configuration interface over XML-RPC to enable the functionality.
- URScript functions to read and write structured data over connection assembly T→O, and O→T objects.

Functionality is intended to be used as a platform feature. It can be built into URCaps, or used directly from URScript.
One example of EDS file with “CNC” connection that contains custom Robot→PLC (T→O) and PLC→Robot (O→T) assemblies is available in UniversalRobot_CNC.eds. Both assemblies are 210 bytes long, and have instance id’s 101, and 111 respectively.

Performance
It is recommended to consider performance limitation before integrating custom Ethernet/IP adapter with PLC Scanner. Performance limits can be changed in the final production release. Contact Universal-Robots support if following limitations are preventing implementation of Ethernet/IP connectivity in customer application. All read and write operations are executed on non-realtime threads. In internal tests read a write turnaround cycle with PLC RPI set to 5ms was on average 40ms. Maximum at 90ms. This implementation is not recommended for application requiring data update rates below 20ms. Maximum recommended assembly size is 1000 bytes. This engineering preview does not limit sizes.

Configuration interface
Ethernet/IP service exposes XML-RPC server on port 40000.
add_configuration(Robot→PLC instance id, Robot→PLC assembly size, PLC→Robot instance id, PLC→Robot assembly size)

Function creates one input, and one output assembly that typically forms a Ethernet/IP connection object.

Parameters:

- Robot→PLC instance id – Integer, unique instance ID
- Robot→PLC assembly size – Integer, assembly data size in bytes
- PLC→Robot instance id – Integer, unique instance ID
- PLC→Robot assembly size – Integer, assembly data size in bytes

Returns:

- True if new assembly was created
- False when one of the assemblies already exists, or parameter error.

After function is called, external Ethernet/IP Scanner (on PLC, or CNC machine) can use new connection. Some PLCs require connection object to be defined in EDS (Electronic Data Sheet) file. Separate guide is available on request explaining how to add custom connections, and assemblies to existing Universal Robots EDS file.

NOTE: Universal Robots built-in assemblies 100, and 112 cannot be overwritten. Attempt to configure those assemblies will result in an error.

Example code for adding custom Ethernet/IP from URScript
# Before Start program section
# create XMLRPC connection to Ethernet/IP service global eip_handle=rpc_factory("xmlrpc", "http://127.0.0.1:40000/RPC2")
# one input, and one output assemblies will be available in Ethernet/IP adapter after the call global result = eip_handle.add_configuration(101, 210, 111, 210)
# dispose XMLRPC connection handle eip_handle.closeXMLRPCClientConnection()
# pause for 1s while Ethernet/IP service is configuring data exchange files for new assemblies sleep(1)
# next data layout for read and write should be configured

URScript interface
Payload data exchanged through the connection is accessible using read and write handle objects. Typically handle would be opened once in "Before Start" section of the program (or by script contributed by URCap installation contribution), then in the same section data layout will be defined. Afterwards individual reads and writes can be made with methods on the configured handles.

eip_reader_factory(assembly_name, data_size)
Opens handle to PLC→Robot data. Typically placed in "Before Start" section of the program.

Parameters:

- assembly_name – string representation of instance id.
- data_size – size of the assembly data in bytes

Returns:

- handle object

The handle will provide a read only access to the assembly data.

Example:
global cnc_status = eip_reader_factory(“111”, 210) # Open handle to PLC→Robot assembly with instance id 111, and data size 210 bytes.
eip_writer_factory(assembly_name, data_size)
Opens handle to Robot→PLC data. Typically placed in “Before Start” section of the program.

Parameters:

- assembly_name – string representation of instance id.
- data_size – size of the assembly data in bytes

Returns:

- handle object

The handle will provide a write only access to the assembly data.

Example:
global cnc_command = eip_writer_factory(“101”, 210) # Open handle to Robot→PLC assembly with instance id 101, and data size 210 bytes.

Methods on handles
Handles are used to define underlying assembly memory layout, and afterwards read and write data.

Methods are invoked on handles using ‘.’ operator. Methods are grouped in 2 families:

- memory layout definitions: informing handle on how data is visible to the external device, and assigning labels. All methods in this family start with “define_…”.
- data access methods: read and write process data.

Examples:
cnc_status.define_bit(0, 1, 0, “comm_check”)status = cnc_status.read(“comm_check”)cnc_command.define_uint8(6, “program_number”)
define_uint8(offset, data_name), define_uint16(offset, data_name), define_int16(offset, data_name), define_float32(offset, data_name), define_int32(offset, data_name)
Methods define individual labels in assemblies exchanged with PLC. Labels are used for writing and reading process data in URScript.

Parameters:

- offset: Integer, position of the data in assembly in bytes
- data_name: String, label

NOTE: Overlapping data definitions are not allowed, except for overlap with bit fields.

Examples:
cnc_status.define_uint8(11, “service_code”) # set label “service_code” to byte 11 of the assembly
define_bit(offset, bitfield_width, bit_number, data_name)
Method defines individual label in assemblies exchanged with PLC. Labels are used for writing and reading process data in URScript.

Parameters:

- offset: Integer, position of the data in assembly in bytes
- bitfield_width: Integer, can be 1, 2 or 4 representing bit in byte, word, or double word.
- bit_number Integer, bit number in a word. 0 = least significant bit. Depending on the bitfield_width most significant bit will be 7, 15, or 31.
- data_name: String, label

NOTE: Overlapping bit definitions are not allowed.

Example:
cnc_status.define_bit(0, 1, 0, “comm_check”) # set label “comm_check” to bit 0 in byte 0 of the assembly.
cnc_status.define_bit(16, 2, 9, “door_opened”) # set label “door_opened” to bit 9 in word starting on byte 16 of the assembly.
define_uint8_array(offset, length, data_name), define_uint16_array(offset, length, data_name), define_int16_array(offset, length, data_name), define_float32_array(offset, length, data_name), define_int32_array(offset, length, data_name)
Methods define individual labels in assemblies exchanged with PLC. Labels are used for writing and reading process data in URScript.

Parameters:

- offset: Integer, position of the data in assembly in bytes
- length: Integer, number of elements of the array with base size specified by the data type (e.g. for define_uint16_array length = 4 will use 8 bytes)
- data_name: String, label

Examples:
cnc_command.define_uint8_array(48, 32, “work_number”) # set label
“work_number” to 32 byte long array starting at byte 48 in the assembly cnc_command.define_int16_array(120, 3, “axes_offset”) # set label
“axes_offset” to 3 words (6 byte long) array starting at byte 120 in the assembly
define_string(offset, length, data_name)

Parameters:

- offset: Integer, position of the data in assembly in bytes
- length: Integer, number of characters in the string. String should be terminated with 0 if it’s shorter than length.
- data_name: String, label

read(data_names)
Method reads one or more data labels written by external PLC. Last data is read in non-blocking way.

Parameters:

- data_name:
    - Single string – one data label to read. Method returns basic data type.
    - Array of strings – multiple data labels to read. Method returns structure with fields named with data labels.

Examples:
comm_check = cnc_status.read(“comm_check”) # reads single bit to comm_check URScript variable
service_request = cnc_status.read([“door_closed”, “service_code”]) # reads multiple fields that can be accessed as service_request struct members if(service_request.door_closed):
do_service_code(service_request.service_code)
end

write(data_struct)
Method writes one or more data labels to memory read external PLC.

Parameters:

- data_struct: Struct where member names are data labels, and values are data contents

All labels in struct are guaranteed to be written at the same time to assembly data.

NOTE: method will do a range check based on the type of the data label. Numbers exceeding data type will stop a program with runtime exception.

Examples:
cnc_command.write(struct(work_number = [1, 2, 3, 4], search_start = True)) # set work_number label array, and search_start bit
cnc_command.write(struct(door_open = True))
# define global struct, and reuse in calls to write method
s = struct(robot_ready = False, machine_stop = False, robot_alarm = False)
s.robot_ready = True cnc_command.write(s)

clear()
Method sets all memory managed by handle to zero.
Can only be called on the write handle.

close()
Closes handle object, and frees memory associated. It’s recommended to close handles if they are used only once in a program. If handle is used repeatedly then it’s recommended to keep it open over the lifespan of the program. The same handle can not be opened again, new handle has to be created.

Bug Fixes

PolyScope GUI

- The ampere value for the “IO Current” field in the Readings panel on the Log tab is now shown as a decimal number instead of a whole number
- Fixed an issue where programs were slow to start, if the program contained Pallet (Palletizing) nodes
- Fixed issue on UR20 robots using a setup with an external 3-Position Enabling (3PE) Device where the Automove screen would show illustrations and text referring to the UR Teach Pendant with built-in 3PE functionality

Controller

- Introduced the RTDE channel “collision_detection_ratio” which publishes a non-negative double precision

floating point value which can be used to monitor how close the robot is to triggering a C157 or C158 “collision detected” protective stop. A protective stop is triggered if the value is ever equal to or exceeds 1.0. The channel is very similar to the existing RTDE channel “joint_position_deviation_ratio” which does the same for the C153 and C159 “joint position deviation” protective stops.
- Fixed issue where empty string variables were incorrectly evaluated as True; now empty strings correctly evaluate as False, ensuring accurate boolean logic in conditional statements.
- Fixed issue where a robot with IMMI attached would fault the first time the robot arm was powered on after a reboot of the controlbox.
- Fixed an issue where the get_inverse_kin_has_solution() function where it would incorrectly report that the solution had not been found after two or more calls with the same argument.
- Fixed a corner case with movec in “fixed” mode where the TCP could rotate in the opposite direction of the required rotation to maintained the “fixed’ orientation constraint.
- Fixed situations where UR20 will fail with 48V voltage drop (C740A11) or over-current (C740A15), or system reboot in very hard motions.

URCap Software Platform

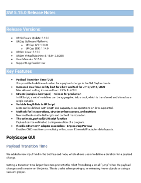
- Fixed issue where an unhandled null pointer Java exception would occur, in some situations, when the user tried to undo program changes. This could occur, if the user prior to the undo action had attempted twice to insert a new URCap node as child under another URCap node with locked child sequence.
- Fixed issue where a URCap would never get a callback after calling the UserInteraction.getUserDefinedRobotPosition(RobotPositionCallback2) method to request the end user to define a position of the robot. This would occur, if the robot was not powered on and brake released (“Robot needs to be powered ON to proceed.” tooltip displayed in the PolyScope footer). The RobotPositionCallback2.onCancel() callback function is now called, if the robot is not fully operational when the request is made.
- Fixed issue where a URCap would never get a callback after calling one of the methods in the RobotMovement interface, e.g. requestUserToMoveRobot(Pose, RobotMovementCallback), to request the end user to move the robot to a given target position. This would occur, if the robot was not powered on and brake released (“Robot needs to be powered ON to proceed.” tooltip displayed in the PolyScope footer). The RobotMovementCallback.onCancel() callback function is now called, if the robot is not fully operational when the request is made.

URSim

Embedded

- Removed generic user fault report “Start up check : Critical error”, and replaced it with more clear error reports in the specific scenarios
- Fixed issue where Energy Eater power dissipation monitoring can cause system to fault if the safety control board and I/O draws more than 50W.
- Fix for issue where robot would be stuck in booting if a fault occurs very early after the joints boot. Could e.g. occur if the brake solenoid cable was disconnected

Documents / Resources

	**Universal Robots UR10 Medium Sized Versatile Cobot** [pdf] User Guide UR10, UR16, UR20, UR10 Medium Sized Versatile Cobot, UR10, Medium Sized Versatile Cobot, Sized Versatile Cobot, Versatile Cobot, Cobot

SW 5.15.0 Release Notes

Release Versions:

- UR Software Update: 5.15.0
- URCap Software Platform:
  - URCap API: 1.14.0
  - URCap SDK: 1.14.0
- URSim Linux: 5.15.0
- URSim Virtual Machine: 5.15.0 - 2.0.285
- User Manuals: 5.15.0
- Support Log Reader: xxx

Key Features

- **Payload Transition Time (GUI)**
  It is possible to define a duration for a payload change in the Set Payload node.
- **Increased max force safety limit for elbow and tool for UR10, UR16, UR20**
  Max allowed setting increased from 250N to 400N.
- **Structs (complex data types) - Release for production**
  In URScript, a set of variables can be aggregated into struct, which is transferred and stored as a single variable.
- **Variable length lists in URScript**
  Lists can be defined with length and capacity. New operations on lists supported.
- **Methods for list operations, struct members access, and matrices**
  New methods enable list length and content manipulation.
- **The estimate_payload() URScript function**
  Payload can be estimated during execution of a program.
- **Flexible Ethernet/IP adapter assemblies - Engineering Preview**
  Enables CNC machine connectivity with custom Ethernet/IP adapter data layouts.

PolyScope GUI

Payload Transition Time

We added a new input field in the Set Payload node, which allows users to define a duration for a payload change.

Setting a transition time larger than zero prevents the robot from doing a small "jump" when the payload changes and is easier on the joints. This is useful when picking up or releasing heavy objects or using a vacuum gripper.

References

- **User Manual**