Using Universal Robots

with LEonard





Software Version 22.11.1.0

**LEonard Software by Lecky Engineering, LLC**

|  |  |  |
| --- | --- | --- |
| **Document Version** | **Date** | **Major Additions** |
| 21.11.4.0 | 11/04/2021 | Initial user interface and device management system, Java interpreter |
| 22.04.1.0 | 04/01/2022 | Universal Robot interface and grinding system, LEScript support |
| 22.08.1.0 | 08/15/2022 | LMI Gocator interface and demonstration |
| 22.11.1.0 | 11/14/2022 | Python support, screen sizing and display management |

**CONTENTS**

[Overview 6](#_Toc119332319)

[Basic Ethernet Connection 6](#_Toc119332320)

[The LEonard Interface 7](#_Toc119332321)

[Additions to the Run Tab for UR 9](#_Toc119332322)

[Run Tab- UR Dashboard Connection Only 10](#_Toc119332323)

[Run - UR Dashboard and Command Connection 11](#_Toc119332324)

[UR Grinding Option Controls 12](#_Toc119332325)

[Robot Jogging in LEonard 13](#_Toc119332326)

[Installing Programs on the UR 14](#_Toc119332327)

[Program Installation Method 1: FTP 14](#_Toc119332328)

[Program Installation Method 2: The Supplied Magic File 14](#_Toc119332329)

[Controlling the Robot 16](#_Toc119332330)

[General LEonard.urp Communications Strategy 16](#_Toc119332331)

[LElib.UR Library for Universal Robots 16](#_Toc119332332)

[LElib.UR.dashboard: Commands to control the UR dashboard 16](#_Toc119332333)

[string ur\_dashboard(string message, int timeout\_ms) 16](#_Toc119332334)

[LElib.UR.robot: The UR Robot PolyScope Interface 18](#_Toc119332335)

[select\_tool(string tool\_name) 18](#_Toc119332336)

[set\_part\_geometry(string FLAT|CYLINDER|SPHERE, double part\_diam\_mm) 18](#_Toc119332337)

[save\_position(position\_name) 18](#_Toc119332338)

[move\_linear(position\_name) 18](#_Toc119332339)

[move\_joint(position\_name) 18](#_Toc119332340)

[move\_relative(dx\_mm, dy\_mm) 18](#_Toc119332341)

[move\_tool\_home() 18](#_Toc119332342)

[move\_tool\_mount() 18](#_Toc119332343)

[free\_drive(0=OFF|1=ON) 18](#_Toc119332344)

[set\_linear\_speed(speed\_mm/s) 19](#_Toc119332345)

[set\_linear\_accel(accel\_mm/s^2) 19](#_Toc119332346)

[set\_joint\_speed(speed\_deg/s) 19](#_Toc119332347)

[set\_joint\_accel(double accel\_deg/s^2) 19](#_Toc119332348)

[set\_blend\_radius(double blend\_radius\_mm) 19](#_Toc119332349)

[get\_actual\_tcp\_pose() 19](#_Toc119332350)

[get\_target\_tcp\_pose() 19](#_Toc119332351)

[get\_actual\_joint\_positions() 19](#_Toc119332352)

[get\_target\_joint\_positions() 19](#_Toc119332353)

[get\_actual\_both() 19](#_Toc119332354)

[get\_target\_both() 20](#_Toc119332355)

[movej(double j1, j2, j3, j4, j5, j6) 20](#_Toc119332356)

[movel(double x, y, z, rx, ry, rz) 20](#_Toc119332357)

[get\_tcp\_offset() 20](#_Toc119332358)

[movel\_incr\_base(double x,y,z,rx,ry,rz) 20](#_Toc119332359)

[movel\_incr\_tool(double x,y,z,rx,ry,rz) 20](#_Toc119332360)

[movel\_incr\_part(x,y,z,rx,ry,rz) 21](#_Toc119332361)

[movel\_single\_axis(axis,value) 21](#_Toc119332362)

[movel\_rot\_only(rx,ry,rz) 21](#_Toc119332363)

[movel\_rel\_set\_tool\_origin(double x,y,z,rx,ry,rz) 21](#_Toc119332364)

[movel\_rel\_set\_tool\_origin\_here() 21](#_Toc119332365)

[movel\_rel\_tool(x,y,z,rx,ry,rz) 21](#_Toc119332366)

[movel\_rel\_set\_part\_origin(x,y,z,rx,ry,rz) 21](#_Toc119332367)

[movel\_rel\_set\_part\_origin\_here() 21](#_Toc119332368)

[movel\_rel\_part(x,y,z,rx,ry,rz) 21](#_Toc119332369)

[send\_robot(string message) 21](#_Toc119332370)

[set\_output(int port, bool value) 22](#_Toc119332371)

[robot\_socket\_reset() 22](#_Toc119332372)

[robot\_program\_exit() 22](#_Toc119332373)

[set\_tcp(x,y,x,rx,ry,rz) 22](#_Toc119332374)

[set\_payload(mass\_kg,cog\_x\_m, cog\_y\_m, cog\_z\_m) 22](#_Toc119332375)

[set\_door\_closed\_input(int dig\_in, int state) 22](#_Toc119332376)

[set\_footswitch\_pressed\_input(int dig\_in, int state) 23](#_Toc119332377)

[set\_tool\_on\_outputs(int dig\_out, int state, …) 23](#_Toc119332378)

[set\_tool\_off\_outputs(int dig\_out, int state, …) 23](#_Toc119332379)

[set\_coolant\_on\_outputs(int dig\_out, int state, …) 23](#_Toc119332380)

[set\_coolant\_off\_outputs(int dig\_out, int state, …) 23](#_Toc119332381)

[tool\_on() 23](#_Toc119332382)

[tool\_off() 23](#_Toc119332383)

[coolant\_on() 23](#_Toc119332384)

[coolant\_off() 23](#_Toc119332385)

[LElib.UR.grind: The UR Grinding Package 23](#_Toc119332386)

[grind\_line(dx\_mm, dy\_mm, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact) 24](#_Toc119332387)

[grind\_line\_deg(length\_mm, angle\_deg, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact) 24](#_Toc119332388)

[grind\_rect(dx\_mm, dy\_mm, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact) 24](#_Toc119332389)

[grind\_serp(dx\_mm, dy\_mm, n\_xsteps, n\_ysteps, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact) 24](#_Toc119332390)

[grind\_poly(circle\_diam\_mm, n\_sides, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact) 24](#_Toc119332391)

[grind\_circle(circle\_diam\_mm, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact) 24](#_Toc119332392)

[grind\_spiral(circle1\_diam\_mm, grind\_circle2\_diam\_mm, n\_spirals, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact) 24](#_Toc119332393)

[grind\_retract() 24](#_Toc119332394)

[grind\_contact\_enable(0=Touch OFF,Grind OFF|1=Touch ON,Grind OFF| 2=Touch ON,Grind ON) 25](#_Toc119332395)

[grind\_touch\_retract(touch\_retract\_mm) 25](#_Toc119332396)

[grind\_touch\_speed(touch\_speed\_mm/s) 25](#_Toc119332397)

[grind\_force\_dwell(dwell\_time\_ms) 25](#_Toc119332398)

[grind\_max\_wait(max\_time\_before\_retract\_ms) 25](#_Toc119332399)

[grind\_max\_blend\_radius(grind\_blend\_radius\_mm) 25](#_Toc119332400)

[grind\_trial\_speed(trial\_speed\_mm/s) 25](#_Toc119332401)

[grind\_linear\_accel(accel\_mm/s^2) 25](#_Toc119332402)

[grind\_point\_frequency(point\_frequency\_hz) 25](#_Toc119332403)

[grind\_jog\_speed(trial\_speed\_mm/s) 25](#_Toc119332404)

[grind\_jog\_accel(accel\_mm/s^2) 25](#_Toc119332405)

[grind\_force\_mode\_damping(damping: 0.0 – 1.0) 26](#_Toc119332406)

[grind\_force\_mode\_gain\_scaling(scaling: 0.0 – 2.0) 26](#_Toc119332407)

[enable\_user\_timers(integer 0=off, 1=on) 26](#_Toc119332408)

[zero\_user\_timers() 26](#_Toc119332409)

[return\_user\_timers() 26](#_Toc119332410)

[LElib.UR.grind Grinding Examples 26](#_Toc119332411)

[Remove Current Tool 26](#_Toc119332412)

[Install A Tool 27](#_Toc119332413)

[Integrated Example 27](#_Toc119332414)

[Computed Concentric Circles 28](#_Toc119332415)

[Lots of Grinds 29](#_Toc119332416)

# Overview

LEonard provides a custom interface for industrial cobots from Universal Robots (UR).

The interface is currently in use with UR-5e and UR-10e robots in many places.

For more information and documentation on the exciting UR product line, see [www.universal-robots.com](http://www.universal-robots.com).

# Basic Ethernet Connection

The UR and the computer running LEonard must have an Ethernet interface capable of communication. At Lecky Engineering, our test machine is on 192.168.0.252/24 and our UR-5e is on 192.168.0.2/24 (robotIP=192.168.0.2 for us).

You should be able to use ping robotIP and ssh root@robotIP to verify communication. SSH uses root with default password easybot on a UR!

Text

Description automatically generated

Figure Pinging and SSH into a UR to Verify Communication

If you’re like us, you may also want to be able to move files on and off your UR easily. We use WinSCP. Just connect your FTP client to robotIP:22 and use that root, easybot login to get access.

A screenshot of a computer

Description automatically generated

Figure WinSCP FTP access to a UR

# The LEonard Interface

To communicate with the UR, the **Devices** list in LEonard needs one or two entries for the robot:

1. If you just want to use the UR Dashboard interface and ask the robot to load/run existing PolyScope programs on the robot, you only need a **UrDashboard** device.
2. If you want to use the Lecky Engineering LEonard01.urp PolyScope program that allows commanding, driving, and sequencing the robot, as well as using the Lecky Engineering Grinding programs, you also need a **UrCommand** device.

Both devices connect to the same robot, just on different ports.

1. Port 29999 is a standard fixed port that provides dashboard control services on all UR robots. It connects to the robot as a **TcpClient** but has some special features that a basic **TcpClient** device does not.
2. Port 30000 is a custom port used in the Lecky Engineering LEonard01.urp PolyScope program. The program looks for a **TcpServer** on the machine running LEonard. The UrCommand device is a customized **TcpServer** device that has some special features to help with the UR interface.

Table

Description automatically generated

Figure Device Entries for Universal Robot

It is important to use the UrCommand CallBack as well as the displayed TxSuffix and RxTerminator. Default devices that have everything setup just the way we need although you might need to edit the IP address!

The PolyScope job that you wish to be loaded by the UrDashboard can be included in the Jobfile field of the dashboard device entry. For our example, this is set to LEonard/LEonard01.urp since the LEonard01.urp program is stored on the robot in programs/LEonard/LEonard01.urp.

## Additions to the Run Tab for UR

Special status and control annunciators and buttons appear when a UR is in communication with LEonard. These features are described below:

Diagram

Description automatically generated with low confidence

Annunciators controlled by **UrDashboard**

(These are talking to the UR.)

Annunciators controlled by **UrCommand**

**These are talking to the PolyScope program!**

Figure Gocator Status Annunciators

### Run Tab- UR Dashboard Connection Only

Connecting a UR Robot Dashboard makes the robot control annunciators visible:

Graphical user interface, text, application, table

Description automatically generated

Figure 5 LEonard Connected to UR Dashboard

Graphical user interface, application

Description automatically generated

Figure 6 LEonard Main Screen when Connected to UR Dashboard

**Dashboard OK:** Shows connection status to the Dashboard (this is a TCP client).

**Robotmode:** Press this to cycle from POWER\_OFF to BOOTING to IDLE to RUNNING

**Safetystatus:** Press this to clear robot stop conditions

**STOPPED/PLAYING:** Use this to toggle between a PolyScope program running or not on the robot. The program loaded is specified in the Device entry and is shown in this button.

Most of these options require that the UR is in Remote mode.

### Run - UR Dashboard and Command Connection

With a UR Command interface also connected, the full UR control system is available. This requires that a special PolyScope program be installed and running on the robot. This program is provided by Lecky Engineering and can be modified by the user if you need special functions. Contact Lecky Engineering for assistance if you are new to UR programming!

The images below assume the optional UR Grinding package is also licensed.

Graphical user interface, table

Description automatically generated

Figure 7 LEonard Connected to UR Dashboard and UR Command

Graphical user interface

Description automatically generated

Figure 8 LEonard Main Screen when Connected to UR Dashboard and UR Command

**Command Ready:** Shows status of UR Command connection (this is a TCP server).

**Robot Ready:** Shows if the robot is currently executing a LEonard command.

**Command Index Numbers:** Shows index number of last command sent and response ID of last response received. These should stay in lockstep and must be equal to allow the next command to be issued in the program.

Tools are selected in the **Tool Dropdown.** More information on Tools in in [Setup - Tools](#_Setup_-_Tools)

Tools have mount and home positions that can be moved into with the **toolname\_mount** and **toolname\_home** buttons.

Part geometry in specified in **Part Geometry Dropdown:** The UR Command program supports surfaces that are FLAT, CYLINDER, or SPHERE

### UR Grinding Option Controls

With the optional Grinding Package enabled we get 2 additional buttons:

**Grind Ready:** Has the current grinding process completed?

**Grind Process:** Are we leaving the tool in contact with the part and expecting the next grind command withing a few seconds?

Set the grinding mode with the **Touch/Grind button** which cycles through **No contact, Touch Only, and Touch+Grind**

**Protective Stops:** These show up in (and may be cleared with) the **SafetyStatus** button

**Door Status** is monitored (IO is configured in the **Setup Tab**). Door Open is treated like **Pause**.

**Footswitch Status** is monitored (IO is configured in the **Setup Tab**).

### Robot Jogging in LEonard

Jogging opens a separate screen. Jogging can be done in **BASE** or **TOOL** coordinates, or relative to a **PART**. The buttons move the robot by the specified increment in Z, XY, or rotation.

Holding a button down (mouse) or double-tapping and holding (tablet) makes the move repeat.

Diagram

Description automatically generated

Figure UR Jog Dialog

When jogging in **PART** mode, if a cylindrical or spherical geometry is selected, the tool will rotate around the center of the part instead of around the tool tip. This can be convenient for manually jogging to a defect using the touch screen instead of Freedrive

Freedrive is supported in a manner identical to on the UR pendant. The X, Y, X, RX, RY and RZ buttons may be used to enable or disable Freedrive in any desired axis. All, Trans, Plane, and Rot select pre-defined subsets of axes as on the UR.

Coordinate systems may be changed during Freedrive and the tool will allow motion relative to the world, the tool, or the center of the part of part geometry is cylinder or sphere.

Press the Freedrive button again to turn Freedrive mode off. Saving or exiting the dialog will also turn off freedrive.

The **Freedrive footswitch** puts all 6 axes in Freedrive whether this jog dialog is open or not.

## Installing Programs on the UR

As an aside, how do we get the programs on the UR? The program needs to wind up on the robot in programs/LEonard, and the PolyScope program also requires three somewhat complicated URScript programs to support robot and grinding applications.

### Program Installation Method 1: FTP

We already discussed one method, FTP. Just copy the files from your PC on LEonardRoot\3rdParty\UR\USB Image- Program Installer/LEonard onto the robot in programs/LEonard.

### Program Installation Method 2: The Supplied Magic File

Another method is to use UR magic files:

1. Make sure your existing robot programs are backed up! This process should not affect them, but it never hurts to be careful.
2. Take a fully erased USB thumb drive and insert it in your PC.
3. Copy all the files and subdirectories from LEonardRoot\3rdParty\UR\USB Image- Program Installer onto the thumb drive.
4. Plug the thumb drive into your robot USB port on the pendant- all of the necessary files will be installed on your robot in the programs/LEonard directory.

***You will likely need to manually open LEonard01.urp on your robot to associate it with the installation file you are using in your installation. You’ll be prompted to select an installation file. Once you do, resave the program and everything should be automatic from there.***

***There is also a set of three hard-coded IP address that the robot looks at to find LEonard. This allows a robot to be directly tied to only one LEonard. You will have to adjust this if the PolyScope program if your address is something different. Lecky Engineering can help if you need some assistance!***

***The LEonard01.urp PolyScope program currently looks for LEonard on:***

***169.254.254.200:30000***

***169.254.254.210:3000***

***192.168.0.252:30000***

Connection is initiated by selecting the desired row and pressing **Connect**. In addition, if you have selected **Auto Connect On Load** for your device file, the connection will be started automatically when LEonard starts.

LEonard always starts a UrDashboard connection with a set of commands to initialize the robot, load any specified program, and start it.

close safety popup

is in remote control (response must be true)

if JobFile <> “”

load <Jobfile> (as in Device… default LEonard/LEonard01.urp

get loaded program (make sure response matches above)

play

Upon successful connection, the Connected field should check itself and the UR Status annunciators should appear on the Run tab.

The Run tab in LEonard uses robotmode to determine whether the robot has booted, and regularly sends robotmode, safetystatus, and programstate to keep an eye on the robot.

# Controlling the Robot

The UR may be commanded from LEScript, Java, or Python with equal ease.

All three languages provide a set of functions that cover most point-to-point, grinding, and inspection tasks that you might want to do with your cell.

## General LEonard.urp Communications Strategy

The LeckyEngineering PolyScope program supplied for robot sequencing, control, and the grind functions is complex.

Our simple PolyScope program uses large set of underlying URScript code to perform most functions.

That said, UR-savvy users are able to add their own functions to the PolyScope program with relative ease. All communications between LEonard and the PolyScope program happen over a single socket using a message ID, a simple checksum, and the parameters which are all assumed to be numeric.

All return messages from the robot are asynchronous **LEonardMessages** and can set variables or execute LEScript, Java, or Python functions within LEonard. The whole architecture is quite powerful.

If you need to customize LEonard01.urp or the underlying URScript code for your own needs, don’t hesitate to contact Lecky Engineering for some start-up assistance!

## LElib.UR Library for Universal Robots

These functions work with Universal Robots robotic systems. The commands fall into three categories

1. Dashboard
2. Command Interface to Lecky Engineering’s LEonard01.urp PolyScope program
3. Grinding Package for force-controlled surface following

### LElib.UR.dashboard: Commands to control the UR dashboard

The UR robot provides a dashboard interface that allows controlling the robot operation.

#### string ur\_dashboard(string message, int timeout\_ms)

Sends the command message to the currently selected Universal Robot dashboard connection and waits for up to timeout\_ms milliseconds for a response.

Response:

LEScript: Any response received is placed in the variable ur\_dashboard\_response

Java, Python: Function returns any string received or and empty string.

The UR dashboard system provides many commands that are useful in loading, starting, and stopping the robot. The Run tab in LEonard uses robotmode to determine whether the robot has booted, and regularly sends robotmode, safetystatus, and programstate to keep an eye on the robot.

When you press the **Robot Mode** button, LEonard cycles through the robot modes as appropriate- RUNNING initiates sending power off. IDLE initiates sending brake release. And POWER\_OFF initiates sending power on. This allows you to cycle through UR operating modes.

The **Safety Status** button sends unlock protective stop and close safety popup when the robot is in safety stop but not in E-Stop.

The **Program State** button toggles between sending play and stop to start and stop the loaded PolyScope program. The UR Dashboard device sends a load JobFile command when the UR connects with the dashboard to get your default PolyScope program loaded.

A comprehensive discussion of the dashboard interface is available on the UR website:

<https://www.universal-robots.com/articles/ur/dashboard-server-e-series-port-29999/>

Here are the handiest ones that are used internally by LEonard!

**Useful UR Dashboard Commands**

get robot model Returns robot model number, as in “UR5”

get serial number Returns robot serial number, for example “20195501xxxx”

PolyscopeVersion Returns PolyScope version installed on robot

power on Power system up

brake release Release from IDLE to READY

load LEonard/LEonard01.urp Load a PolyScope program (default shown)

play Start it

close popup Close a popup prompt on the pendant

close safety popup Close a safety popup prompt on the pendant

unlock protective stop Recover from E-stop or safety stop

stop Stop execution of the program

robotmode Get mode POWER\_OFF POWER\_ON BOOTING IDLE RUNNING

programstate Return program state STOPPED file PLAYING file

power off Power servos down (and put brakes on)

### LElib.UR.robot: The UR Robot PolyScope Interface

Lecky Engineering supplies an extensive PolyScope program that supports robot control and grinding functions. This code is supplied with the LEonard installation and must be installed on the UR robot.

Getting robot communications working is discussed in [Basic Ethernet Connection](#_Basic_Ethernet_Connection) and [The LEonard Interface](#_The_LEonard_Interface).

Installation of the code on the robot is discussed in [Installing Programs on the UR](#_Installing_Programs_on)

#### select\_tool(string tool\_name)

Setup all the necessary environment to be able to use tool\_name. No motion is performed. Future tool moves, position moves, and grinds will assume this tool is attached.

#### set\_part\_geometry(string FLAT|CYLINDER|SPHERE, double part\_diam\_mm)

Future tool moves and grinds will assume the specified geometry.

#### save\_position(position\_name)

The current robot position is stored in the Positions Table as position\_name.

#### move\_linear(position\_name)

The robot moves along a linear path to Position position\_name.

#### move\_joint(position\_name)

The robot performs a joint move to Position position\_name.

#### move\_relative(dx\_mm, dy\_mm)

Move (dx\_mm, dy\_mm) relative to current tool position. If the part geometry selected is CYLINDER or SPHERE, robot moves along the part.

#### move\_tool\_home()

Perform a joint move to the home position associated with the current tool.

#### move\_tool\_mount()

Perform a joint move to the mounting position associated with the current tool.

#### free\_drive(0=OFF|1=ON)

Turn robot free drive mode on or off.

The commands below provide a programmatic way to set the default motion parameters.

#### set\_linear\_speed(speed\_mm/s)

Sets default linear speed used for robot linear moves.

#### set\_linear\_accel(accel\_mm/s^2)

Sets default linear acceleration used for robot linear moves.

#### set\_joint\_speed(speed\_deg/s)

Sets default joint speed used for robot joint moves.

#### set\_joint\_accel(double accel\_deg/s^2)

Sets default joint acceleration used for robot joint moves.

#### set\_blend\_radius(double blend\_radius\_mm)

Sets default blend radius used in all robot moves.

#### get\_actual\_tcp\_pose()

Ask the current robot to perform get\_actual\_tcp\_pose() and return the value in the LEonard variable actual\_tcp\_pose.

#### get\_target\_tcp\_pose()

Ask the current robot to perform get\_target\_tcp\_pose() and return the value in the LEonard variable target\_tcp\_pose.

#### get\_actual\_joint\_positions()

Ask the current robot to perform get\_actual\_joint\_positions() and return the value in the LEonard variable actual\_joint\_positions.

#### get\_target\_joint\_positions()

Ask the current robot to perform get\_target\_joint\_positions() and return the value in the LEonard variable target\_joint\_positions.

#### get\_actual\_both()

Performs both get\_actual\_joint\_positions() and get\_actual\_tcp\_pose() on the current robot and return the values to the LEonard variables actual\_joint\_positions and actual\_tcp\_pose.

#### get\_target\_both()

Performs both get\_target\_joint\_positions() and get\_target\_tcp\_pose() on the current robot and return the values to the LEonard variables target\_joint\_positions and target\_tcp\_pose.

#### movej(double j1, j2, j3, j4, j5, j6)

Performs a movej to **joint positions** on the current robot as follows:

q = [j1, j2, j3, j4, j5, j6]

movej(q, a=robot\_joint\_accel\_rpss, v=robot\_joint\_speed\_rps)

#### movel(double x, y, z, rx, ry, rz)

Performs a movel to a **pose** on the current robot as follows:

p = p[x, y, z, rx, ry, rz]

movej(q, a=robot\_joint\_accel, v=robot\_joint\_speed)

#### get\_tcp\_offset()

Ask the current robot to perform get\_tcp\_offset() and return the value in the LEonard variable tcp\_offset.

#### movel\_incr\_base(double x,y,z,rx,ry,rz)

Ask the current robot to move incrementally from the current position in base coordinates as in URScript:

local p0 = get\_target\_tcp\_pose()

local p1 = p[x,y,z,dx,dy,dz]

local p2 = pose\_add(p0, p1)

if p1[0] == 0 and p1[1] == 0 and p1[2] == 0: # Rotational move

movel(p2, robot\_joint\_accel\_rpss, robot\_joint\_speed\_rps)

else:

movel(p2, robot\_linear\_accel\_mpss, robot\_linear\_speed\_mps)

end

#### movel\_incr\_tool(double x,y,z,rx,ry,rz)

Ask the current robot to move incrementally from the current position in TCP coordinates as in URScript:

local p1 = p[x,y,z,rx,ry,rz]

local p2 = pose\_trans(get\_target\_tcp\_pose(), p1)

if p1[0] == 0 and p1[1] == 0 and p1[2] == 0: # Rotational move

movel(p2, robot\_joint\_accel\_rpss, robot\_joint\_speed\_rps)

else:

movel(p2, robot\_linear\_accel\_mpss, robot\_linear\_speed\_mps)

end

#### movel\_incr\_part(x,y,z,rx,ry,rz)

Ask the current robot to move incrementally from the current position in PART coordinates. X and Y are interpreted based on set\_part\_geometry(…). For cylinders, X is along the axis of the cylinder and Y is interpreted as a fixed-distance rotation about the cylinder.

#### movel\_single\_axis(axis,value)

Ask the current robot to move to its current pose with the coordinate axis changed to value.

#### movel\_rot\_only(rx,ry,rz)

Ask the current robot to move to its current pose with the new rotations rx, ry, and rz.

#### movel\_rel\_set\_tool\_origin(double x,y,z,rx,ry,rz)

#### movel\_rel\_set\_tool\_origin\_here()

Sets a tool-coordinate origin for the current robot either to a specified pose or to the current robot position. Subsequent calls to movel\_rel\_tool() will move in tool coordinates relative to this origin.

#### movel\_rel\_tool(x,y,z,rx,ry,rz)

Move to a tool coordinate position that is relative to the movel\_rel\_set\_tool\_origin.

#### movel\_rel\_set\_part\_origin(x,y,z,rx,ry,rz)

#### movel\_rel\_set\_part\_origin\_here()

Sets a part-coordinate (FLAT, CYLINDER, or SPHERE) origin for the current robot either to a specified pose or to the current robot position. Subsequent calls to movel\_rel\_part() will move in part coordinates relative to this origin.

#### movel\_rel\_part(x,y,z,rx,ry,rz)

Move to a part coordinate position that is relative to the movel\_rel\_set\_part\_origin.

#### send\_robot(string message)

Sends any command to the Lecky Engineering PolyScope program. All communications with the Lecky Engineering PolyScope program is handled by this command.

1. Commands are sent with a message ID and a checksum as follows:
   1. (ID, checksum, message)
2. ID can be any integer. LEonard sends an incrementing number between 100 and 999.
3. Checksum is expected to be 1000 – ID.
4. message is typically 1 or more comma-separated numeric values.
5. The command is non-blocking.
6. The PolyScope program is expected to send a start message:

robot\_starting = ID

robot\_ready = False

1. After the command is complete, the PolyScope program is expected to send back the following:

robot\_response = response\_message

robot\_ready=True

robot\_completed = ID (as it was received)

In addition, the UR Command device runs the general CallBack, so the UR robot can return **LeonardMessages** to set variables or trigger other actions in LEonard at any time.

#### set\_output(int port, bool value)

Set UR digital output port to value.

#### robot\_socket\_reset()

Commands the Lecky Engineering UR PolyScope program to reset (bounce) its socket connection to LEonard. Program must be running on the UR!

#### robot\_program\_exit()

Commands the Lecky Engineering UR PolyScope program to terminate. Program must be running on the UR!

Low-level Setup Calls

These are all called automatically by select\_tool(), set\_part\_geometry(),and the grind\_xxx() functions. They should be used through that high level interface except during testing or for special purposes!

#### set\_tcp(x,y,x,rx,ry,rz)

Executes set\_tcp(p[x,y,z,rx,ry,rz]) on the current robot only if x > 10. Always returns the current get\_tcp\_offset() in the LEonard variable robot\_tcp.

#### set\_payload(mass\_kg,cog\_x\_m, cog\_y\_m, cog\_z\_m)

Executes set\_paylod(mass\_kg, [cog\_x\_m, cog\_y\_m, cog\_z\_m]) on the current robot only if mass\_kg > 0. Always returns the current robot\_payload\_mass\_kg and robot\_paylod\_cog\_m in corresponding LEonard variables.

#### set\_door\_closed\_input(int dig\_in, int state)

Specifies what digital input and state is expected to signify that the door is closed to the current robot.

#### set\_footswitch\_pressed\_input(int dig\_in, int state)

Specifies what digital input and state is expected to signify that the footswitch is pressed on the current robot.

#### set\_tool\_on\_outputs(int dig\_out, int state, …)

Sets a set of digital output,state pairs (1 – 4) to specify what outputs should be controlled when tool\_on() is executed on the current robot.

#### set\_tool\_off\_outputs(int dig\_out, int state, …)

Sets a set of digital output,state pairs (1 – 4) to specify what outputs should be controlled when tool\_off() is executed on the current robot.

#### set\_coolant\_on\_outputs(int dig\_out, int state, …)

Sets a set of digital output,state pairs (1 – 4) to specify what outputs should be controlled when coolant\_on() is executed on the current robot.

#### set\_coolant\_off\_outputs(int dig\_out, int state, …)

Sets a set of digital output,state pairs (1 – 4) to specify what outputs should be controlled when coolant\_off() is executed on the current robot.

#### tool\_on()

Performs the tool\_on output list set in set\_tool\_on\_outputs() on the current robot.

#### tool\_off()

Performs the tool\_off output list set in set\_tool\_off\_outputs() on the current robot.

#### coolant\_on()

Performs the coolant\_on output list set in set\_coolant\_on\_outputs() on the current robot.

#### coolant\_off()

Performs the coolant\_off output list set in set\_coolant\_off\_outputs() on the current robot.

## LElib.UR.grind: The UR Grinding Package

The grinding commands use a set of common parameters described below:

dx\_mm, dy\_mm, diam\_mm**:** dimensions of the patterns in mm

n\_cycles**:** times to repeat the pattern (ignored if test grinding)

speed\_mm/s: speed to grind at (ignored if test grinding

force\_N: force in Newtons to apply

stay\_in\_contact: 0 to retract at end of grind, 1 to stay in contact

#### grind\_line(dx\_mm, dy\_mm, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact)

#### grind\_line\_deg(length\_mm, angle\_deg, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact)

Grind in a straight line centered on the current position, defined either by endpoints or angle.

#### grind\_rect(dx\_mm, dy\_mm, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact)

Grind along a rectangle centered on the current position at the current RZ angle of the tool.

#### grind\_serp(dx\_mm, dy\_mm, n\_xsteps, n\_ysteps, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact)

Grind a serpentine pattern within a rectangle centered on the current position. N\_xsteps and n\_ysteps is the number of moves needed to span the rectangle. One or the other of these must be equal to 1.

#### grind\_poly(circle\_diam\_mm, n\_sides, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact)

Grind along a polygon of n\_sides inscribed in circle\_diam\_mm centered on the current position.

#### grind\_circle(circle\_diam\_mm, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact)

Grind along a circle centered on the current position.

#### grind\_spiral(circle1\_diam\_mm, grind\_circle2\_diam\_mm, n\_spirals, n\_cycles, speed\_mm/s, force\_N, stay\_in\_contact)

Grind along a variable diameter circle centered on the current position. The circle goes from the first diameter to the second in n\_spirals full revolutions.

#### grind\_retract()

Ensure not in contact with the part. Happens automatically if a non-grind command is sent, if stop or pause is selected, or if grind\_max\_wait timer expires.

#### grind\_contact\_enable(0=Touch OFF,Grind OFF|1=Touch ON,Grind OFF| 2=Touch ON,Grind ON)

Set the grinding mode programmatically as shown.

The commands below provide a programmatic way to set the grinding parameters.

#### grind\_touch\_retract(touch\_retract\_mm)

Set grind retract speed used after touchoff.

#### grind\_touch\_speed(touch\_speed\_mm/s)

Set speed used to go in for touchoff in Z.

#### grind\_force\_dwell(dwell\_time\_ms)

A dwell time performed when force mode is turned on to allow the robot to settle against the grind surface.

#### grind\_max\_wait(max\_time\_before\_retract\_ms)

If the tool is left in contact with the surface awaiting the next grind command, it will retract if this timeout is exceeded.

#### grind\_max\_blend\_radius(grind\_blend\_radius\_mm)

Sets the maximum blend radius that will be used in any pattern. This will be reduces for small geometries.

#### grind\_trial\_speed(trial\_speed\_mm/s)

Sets the speed used for “air grinding” when not in Touch + Grind mode.

#### grind\_linear\_accel(accel\_mm/s^2)

Sets the linear acceleration used for grinding operations.

#### grind\_point\_frequency(point\_frequency\_hz)

Sets a point interpolation frequency used for complex figures. Obsolete.

#### grind\_jog\_speed(trial\_speed\_mm/s)

Sets the speed used when the grinding requires a robot move while not in contact with the part.

#### grind\_jog\_accel(accel\_mm/s^2)

Sets the acceleration used for grinding m,oves not in contact with the part.

#### grind\_force\_mode\_damping(damping: 0.0 – 1.0)

Sets the UR force\_mode\_damping parameter to assist in stabilizing force-mode performance.

#### grind\_force\_mode\_gain\_scaling(scaling: 0.0 – 2.0)

Sets the force\_mode\_gain\_scaling parameter to assist in stabilizing force-mode performance.

**Grind User Timers: Internal Use**

Enabling these will time each grind operation and place it in a circular buffer of user\_timers that can be returned to the variable list with return\_user\_timers(). Used primarily for internal testing.

#### enable\_user\_timers(integer 0=off, 1=on)

Turn the UR-internal user timers on or off.

#### zero\_user\_timers()

Zero all UR-internal user timers.

#### return\_user\_timers()

Return an array of timers. Each timer represents one grinding operation. Repeating the same grinding operations on different surface geometries can be used to validate Lecky Engineering’s internal speed calibration system.

## LElib.UR.grind Grinding Examples

Here are a few sequences that show the kinds of things that can be done in a recipe. The Examplessubdirectory in the Code folder has many more complicated examples that you can examine (and run!).

These examples are shown in LEScript and require slight edits in Java or Python sequences.

#### Remove Current Tool

Just remove the current tool from the robot. As long as the one actually mounted is selected, this goes to the tool home followed by the mount/demount position and prompts the operator when it is time to remove.

# Remove Current Tool

# Go through demount procedure

# Assumes you have selected whatever tool is actually mounted!

prompt(Please confirm: you wish to demount {robot\_tool}?)

move\_tool\_home()

move\_tool\_mount()

prompt(Please demount tool {robot\_tool})

select\_tool(none)

#### Install A Tool

This goes through prompting to mount a specific tool.

# Install 2F85

# Example to install a tool when none is currently installed

# We just select the new tool, move to the mount position, prompt the operator, and move to tool\_home

# Change to whatever tool you like

tool=2F85

# Operator confirmation

prompt(About to mount {tool})

# Mounting process

select\_tool({tool}) # This only informs the robot what is mounted

# This does the physical swap

move\_tool\_mount()

prompt(Please mount tool {tool})

move\_tool\_home()

#### Integrated Example

Here we start with the 2F85 tool ready to grind and swap tools and continue from the same location mid-recipe.

# Integrated Example

# Assumes we're where we want to grind initially but need to do a tool swap mid-way

tool1=2F85

tool2=vertest

# Program assumes we are starting with tool1- verify internally and with operator!

assert(robot\_tool,{tool1})

prompt(Confirming tool {tool1} is currently mounted and you are grinding on {robot\_geometry})

# This will always be our grind\_start position

save\_position(grind\_start)

# Do some grinding with tool1

move\_linear(grind\_start)

grind\_rect(30,30,3,10,10,1)

grind\_rect(20,20,3,10,10,1)

prompt(Ready to swap {tool1} to {tool2}?)

# Remove {tool1}

move\_tool\_home()

move\_tool\_mount()

prompt(Please remove {tool1})

# Install {tool2}

select\_tool({tool2})

move\_tool\_mount()

prompt(Please install {tool2})

move\_tool\_home()

# Do some grinding with tool2

move\_linear(grind\_start) # Returns us to the starting position

grind\_rect(30,30,3,10,10,1)

grind\_rect(20,20,3,10,10,1)

#### Computed Concentric Circles

Here’s a test recipe that grinds 3 concentric circles explicitly and in a loop, not lifting until the final one.

# 26 Concentric Circle Test

# Old school

grind\_circle(30,2,0.9,10,1)

grind\_circle(20,2,0.9,10,1)

grind\_circle(10,2,0.9,10,0)

# Do it with a loop

size = 30

count = 2

speed = 0.9

force = 10

repeat:

grind\_circle({size},{count},{speed},{force},1)

size -= 10

jump\_gt\_zero(size,repeat)

#### Lots of Grinds

By pre-teaching points and swapping geometries, a whole day’s work could be done (other than tool swaps!)

# Test all the patterns on all the geometries

size1=40

size2=10

count=3

speed=5

force=10

select\_tool(2F85)

cycleCount=0

redo:

move\_linear(demo\_flat)

set\_part\_geometry(FLAT,0)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(CYLINDER,400.1)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(CYLINDER,600.1)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(CYLINDER,800.1)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(CYLINDER,1000.1)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(SPHERE,400.2)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(SPHERE,600.2)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(SPHERE,800.2)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

set\_part\_geometry(SPHERE,1000.2)

grind\_line({size1},{size2},{count},{speed},{force},1)

grind\_line(-{size2},{size1},{count},{speed},{force},1)

grind\_rect({size1},{size2},{count},{speed},{force},1)

grind\_rect({size2},{size1},{count},{speed},{force},1)

grind\_serp({size1},{size1},1,3,{count},{speed},{force},1)

grind\_serp({size1},{size1},3,1,{count},{speed},{force},1)

grind\_circle({size1},{count},{speed},{force},1)

grind\_circle({size2},{count},{speed},{force},1)

grind\_spiral({size1},{size2},3,{count},{speed},{force},1)

cycleCount++

jump(redo)

**INDEX**

# 

Ethernet Connection 6

LElib.UR

dashboard

ur\_dashboard 16

free\_drive 18

grind

enable\_user\_timers 26

grind\_circle 24

grind\_contact\_enable 25

grind\_force\_dwell 25

grind\_force\_mode\_damping 26

grind\_force\_mode\_gain\_scaling 26

grind\_jog\_accel 25

grind\_jog\_speed 25

grind\_line 24

grind\_line\_deg 24

grind\_linear\_accel 25

grind\_max\_blend\_radius 25

grind\_max\_wait 25

grind\_point\_frequency 25

grind\_poly 24

grind\_rect 24

grind\_retract 24

grind\_serp 24

grind\_spiral 24

grind\_touch\_retract 25

grind\_touch\_speed 25

grind\_trial\_speed 25

return\_user\_timers 26

zero\_user\_timers 26

robot

coolant\_off 23

coolant\_on 23

get\_actual\_both 19

get\_actual\_joint\_positions 19

get\_actual\_tcp\_pose 19

get\_target\_both 20

get\_target\_joint\_positions 19

get\_target\_tcp\_pose 19

get\_tcp\_offset 20

move\_linear 18

move\_relative 18

move\_tool\_home 18

move\_tool\_mount 18

movej 20

movel 20

movel\_incr\_base 20

movel\_incr\_part 21

movel\_incr\_tool 20

movel\_rel\_part 21

movel\_rel\_set\_part\_origin 21

movel\_rel\_set\_part\_origin\_here 21

movel\_rel\_set\_tool\_origin 21

movel\_rel\_set\_tool\_origin\_here 21

movel\_rel\_tool 21

movel\_rot\_only 21

movel\_single\_axis 21

position\_name 18

robot\_program\_exit 22

robot\_socket\_reset 22

save\_position 18

select\_tool 18

send\_robot 21

set\_blend\_radius 19

set\_coolant\_on\_outputs 23

set\_door\_closed\_input 22

set\_footswitch\_pressed\_input 23

set\_joint\_accel 19

set\_joint\_speed 19

set\_linear\_accel 19

set\_linear\_speed 19

set\_output 22

set\_part\_geometry 18

set\_payload 22

set\_tcp 22

set\_tool\_off\_outputs 23

set\_tool\_on\_outputs 23

tool\_off 23

tool\_on 23