Using LMI Gocators with LEonard

LECKY ENGINEERING

Software Version 22.11.1.1

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
22.11.1.1	11/25/2022	Documentation fixes, small code cleanup and reorganization

CONTENTS

Overview

LEonard provides a custom interface for the LMI Gocator product line.

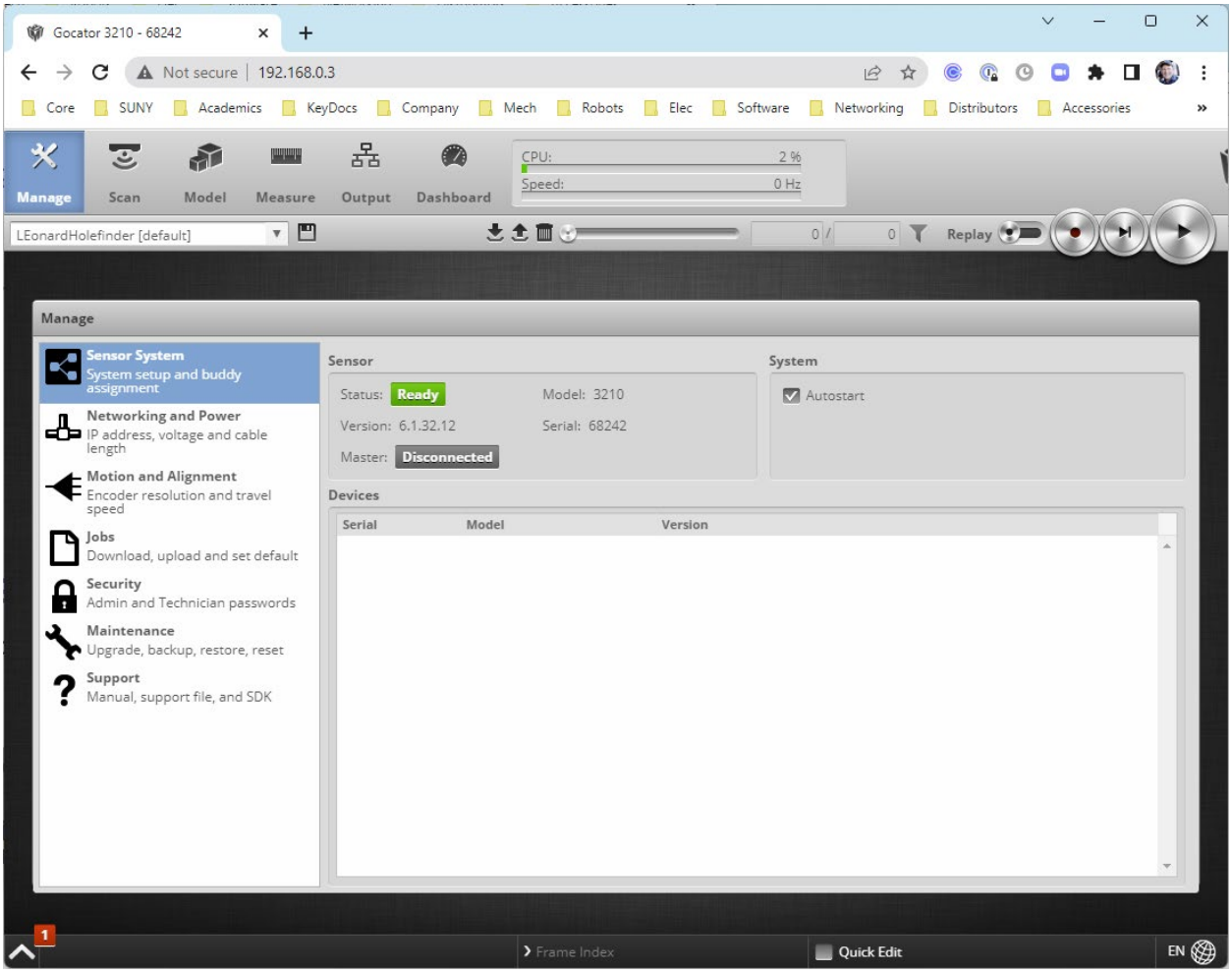
The interface is currently in use with and heavily tested with the Gocator 3200 Series Snapshot cameras.

For more information on these powerful systems, see the company website at lmi3d.com or see product-specific information at lmi3d.com/series/gocator-3210/.

Basic Ethernet Connection

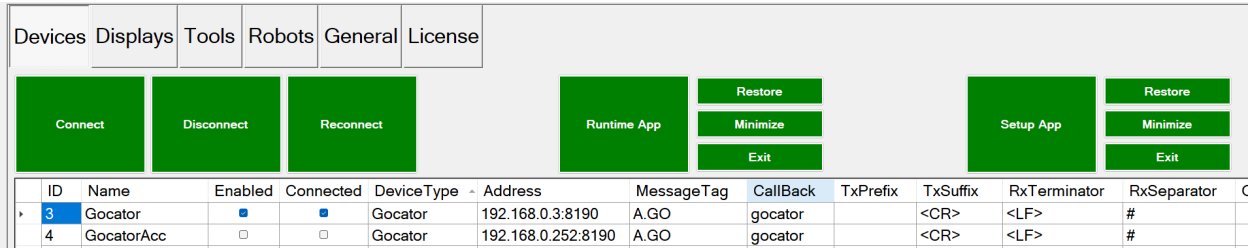
The Gocator and the computer running LEonard must have an Ethernet interface capable of communication. At Lecky Engineering, our test machine is on 192.168.0.252/24 and our Gocator is on 192.168.0.3/24.

You should be able to browse directly to the Gocator to verify communication.



The LEonard Interface

To communicate with the Gocator, the **Devices** list in LEonard needs an entry for the Gocator. The default devices include two properly setup entries for a Gocator- you will just need to verify and adjust your IP address appropriately.



ID	Name	Enabled	Connected	DeviceType	Address	MessageTag	CallBack	TxPrefix	TxSuffix	RxTerminator	RxSeparator
3	Gocator	☑	☑	Gocator	192.168.0.3:8190	A.GO	gocator		<CR>	<LF>	#
4	GocatorAcc	☐	☐	Gocator	192.168.0.252:8190	A.GO	gocator		<CR>	<LF>	#

Figure 1 Device Entries for LMI Gocator

It is important to use the `gocator` CallBack as well as the displayed `TxSuffix` and `RxTerminator`.

The Gocator job that you wish to be loaded can be included in the `Jobfile` field of the device entry.

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 Gocator-style connection with a set of commands:

```
stop
clearalignment
loadjob <jobFile>
start
```

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

Additions to the Run Tab for Gocator

Special status annunciators appear when a Gocator is in communication with LEonard. These are shown below.

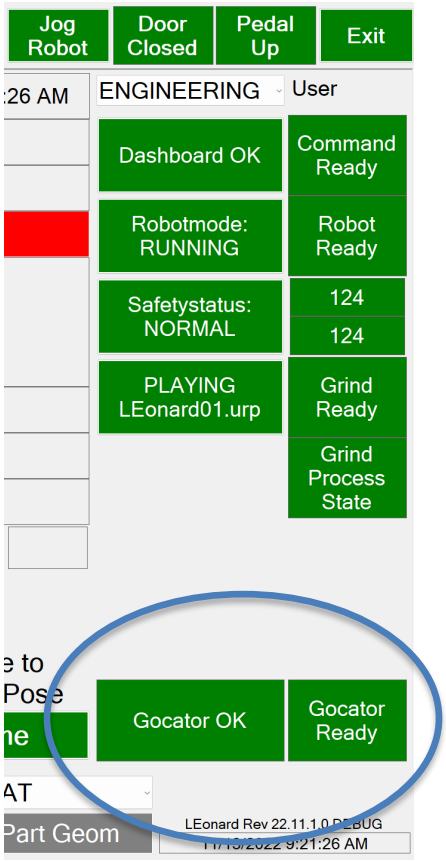


Figure 2 Gocator Status Annunciators

Gocator OK: Is the connection with the Gocator (a special version of `TcpClient`) healthy?

Gocator Ready: Green only if `gocator_ready == True`.

NOTE:

On `gocator_trigger(…)`, `gocator_ready` is set to `False`.

The Gocator must send `gocator_ready=True` at the end of processing!

Using the Accelerator

It is common to use PC-based acceleration with the Gocator to accelerate processing speed.

The Accelerator software is downloaded from the LMI website. Ours is at:
`"C:\Users\nedlecky\Desktop\Gocator 3210\14405-6.1.32.12_SOFTWARE_GO_Utilities\Emulator and Accelerator\bin\win64\GoAccelerator.exe"`

When this software is running, and started, you will see the dialog below. Sometimes it takes a few attempts to get the connection initiated.

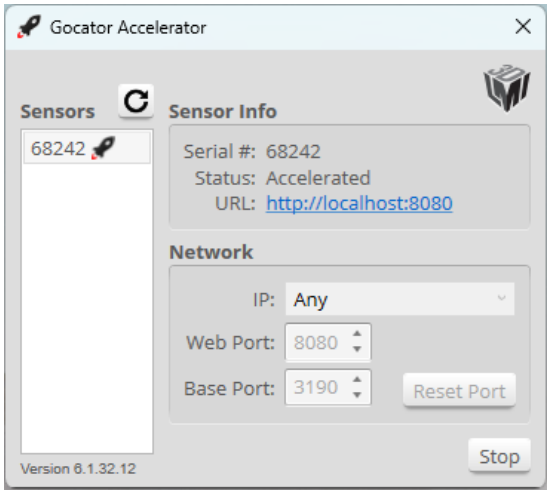


Figure 3 LMI Gocator Accelerator in Operation

For native connection to the Gocator, we would use 192.168.0.3:8190 to connect straight to the Gocator. To use the accelerators, this becomes localIP:8190. In our case, that would be 192.168.0.252:8190.

To browse to the Gocator for monitoring and programming while using the accelerator, just browse to localhost:8080

Your Gocator Job

The LEonard interface to Gocator is simple. The `gocator_trigger(int pre_delay_ms)` command sends the “trigger” command to the Gocator which runs whatever job you have loaded.

When the trigger is sent, LEonard clears a variable called `gocator_ready`.

Your Gocator job must send `gocator_ready=True` as its final output. This is how LEonard knows that processing is complete!

The example Gocator job `LEonardRoot/3rdParty/Gocator` is called `LeonardHolefinder.job`.

It measures hole sizes, angles, and locations for both countersunk and thru holes.

It does this using the Surface Countersunk Hole, Surface Hole, and Surface Plane tools as shown below.

Your job could use any Gocator tools!

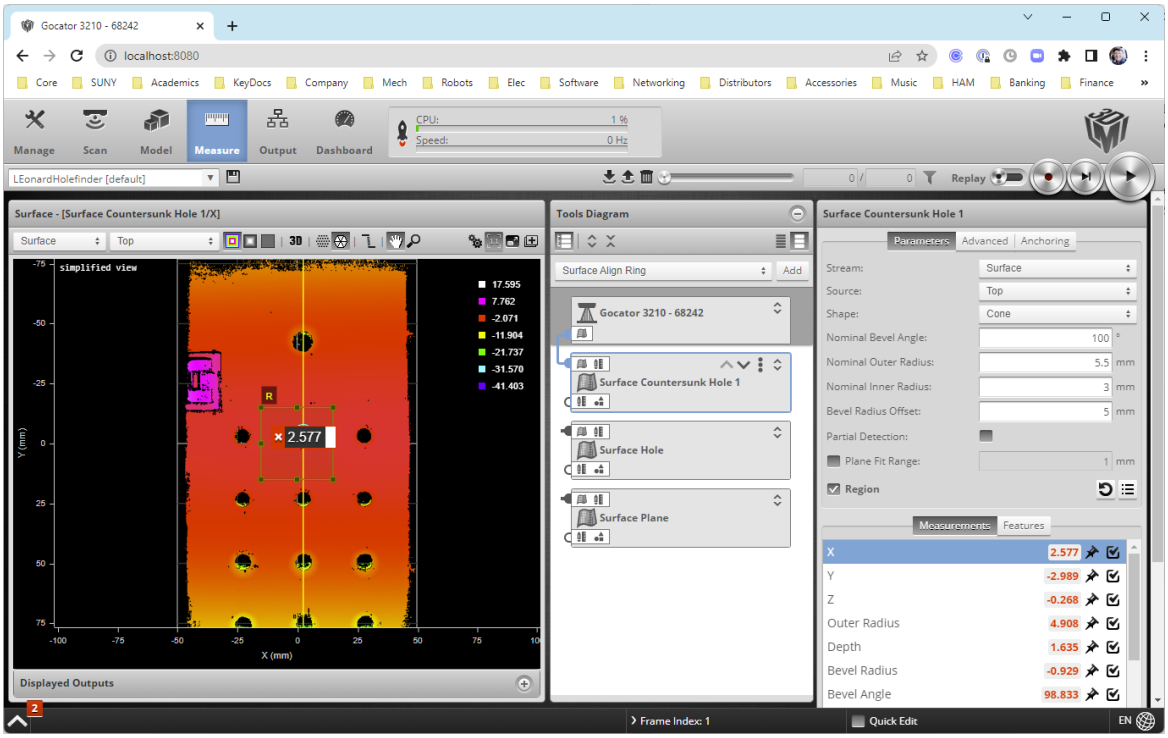


Figure 4 LEonardHolefinder.job Tools

After running your inspection, you need to send results back to LEonard, followed by that all-important `gocator_ready=True`.

Here's how `LEonardHolefinder.job` does it:

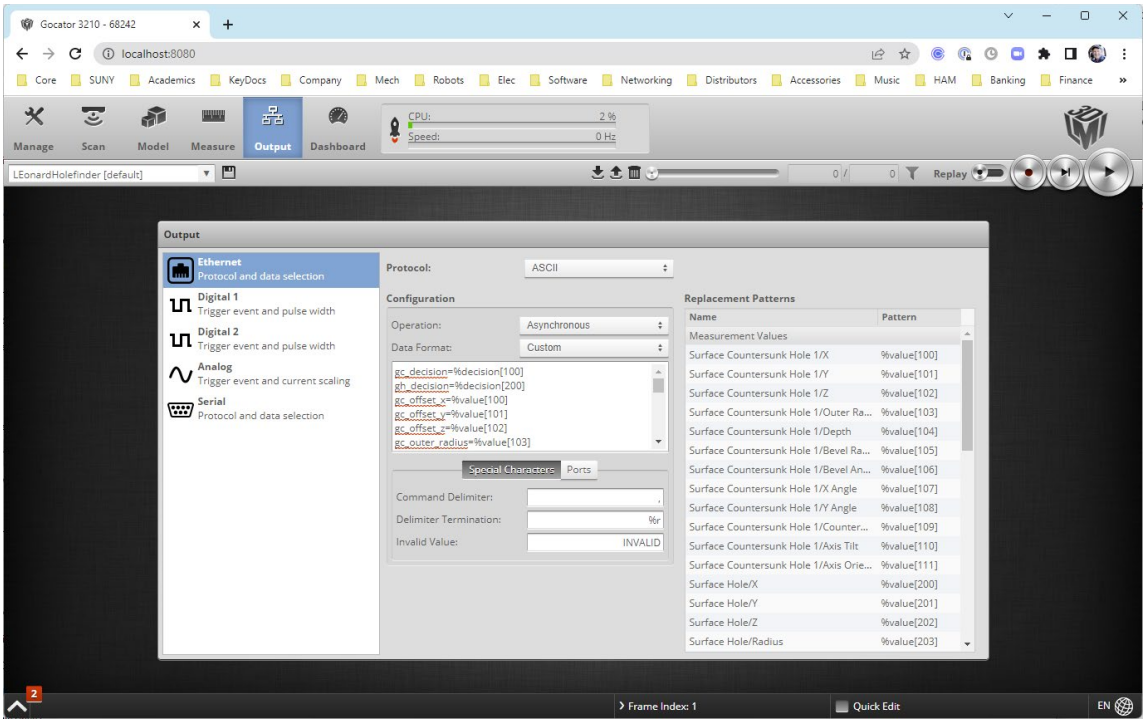


Figure 5 Output from the Gocator job to LEonard

The detail on the data format field is as follows:

```
gc_decision=%decision[100]
gh_decision=%decision[200]
gc_offset_x=%value[100]
gc_offset_y=%value[101]
gc_offset_z=%value[102]
gc_outer_radius=%value[103]
gc_depth=%value[104]
gc_bevel_radius=%value[105]
gc_bevel_angle=%value[106]
gc_xangle=%value[107]
gc_yangle=%value[108]
gc_cb_depth=%value[109]
gc_axis_tilt=%value[110]
gc_axis_orient=%value[111]
gh_offset_x=%value[200]
gh_offset_y=%value[201]
gh_offset_z=%value[202]
gh_radius=%value[203]
gp_xangle=%value[300]
gp_yangle=%value[301]
```

```
gp_z_offset=%value[302]
gp_std_dev=%value[303]
gocator_ready=True
```

DON'T FORGET THAT LAST LINE!
LEonard will assume the Gocator is still crunching data until it receives that. That will also trigger the Gocator Ready annunciator to go back to green.

That's it! You can send as many results back as you like, and they will be received and remembered by LEonard.

These variables are stored in the LEonard LEScript variable set as well as the variable lists in Java and Python.

The variable assignment statements are passed to Java and Python exactly as received, so the automatic typing rules implemented in those languages will automatically happen. The received values will be integers or floats as assumed in Python, for example.

Using the Results in LEonard

LEonard includes a hard-coded `gocator_adjust` function that automatically moves a UR robot to try to drive offsets and angles to 0. This code makes assumptions about orientation, scale, and offset that might not be what you need, however.

Look at the example Python alignment, which is entirely user-modifiable, in `LEonardRoot/Code/Lib/leGocatorSupport.py` to see how custom functions can be used to feed robot motion commands back as a result of measurements.

Look at the examples in `LEonardRoot/Code/Examples/Gocator` to see how these functions are used.

LElib.LMI Library for LMI Gocator

Here is the complete documentation for each of the Gocator-specific internal and Python-based calls available in LEonard.

Many example programs using the snapshot camera for alignment are in `LEonardRoot/Code/Examples/Gocator`.

To learn how to connect, configure, and setup programs for use on the Gocator with LEonard, consult the ***LMI Gocator QuickStart for LEonard*** manual.

LElib.LMI.gocator

These are Gocator support commands built-in to LEonard. As they are not user-modifiable, they should only be used in existing applications.

Lecky Engineering provides a Python library for LEonard that duplicates these capabilities in user-modifiable Python code.

These are discussed after the built-in functions here.

`gocator_send(string message)`

Sends the command `message` to the currently selected LMI Gocator. Non-blocking.

`gocator_trigger(int pre_delay_ms)`

Sends the command `trigger` to the currently selected LMI Gocator. Delays for `pre_delay_ms` milliseconds prior to sending the trigger to allow specification of robot mechanical settling time.

`gocator_adjust(int version)`

This is a built-in fixed adjustment routine in LEonard for hole aligning using the LMI Gocator and a UR robot. It has been used for several applications but is replaced by the new Python version that is user modifiable.

This is a hardcoded adjustment that assumes the use of counterbore, thru hole, and plane tools returning the following variables from the Gocator. The example Gocator job used for testing is at `LEonardRoot/3rdParty/Gocator/LEonardHolefinder.job`.

Counterbore Tool

`gc_decision`	0 if counterbore tool succeeded
`gc_offset_x`	Misalignment in X in microns
`gc_offset_y`	Misalignment in Y in microns

`gc_offset_z` Misalignment in Z in microns
`gc_xangle` Misalignment angle in X in deg/1000
`gc_yangle` Misalignment angle in Y in deg/1000
Also Included (Unused in examples)
`gc_outer_radius` Misalignment angle in Y in deg/1000
`gc_depth` Misalignment angle in Y in deg/1000
`gc_bevel_radius` Misalignment angle in Y in deg/1000
`gc_bevel_angle` Misalignment angle in Y in deg/1000

Hole Tool

`gh_decision` 0 if thru hole tool succeeded
`gh_offset_x` Misalignment in X in microns
`gh_offset_y` Misalignment in Y in microns
`gh_offset_z` Misalignment in Z in microns

Plane Tool

`gp_xangle` Misalignment angle in X in deg/1000
`gp_yangle` Misalignment angle in Y in deg/1000

```
gocator_write_data(string filename, string
tag_name={gocator_ID})
```

This built-in fixed routine appends the standard Gocator alignment variables to a file. The CSV file created has a timestamp, an optional `tag_name`, and a copy of all the variables returned by the standard Gocator application.

The function writes appropriate CSV headers on the file if the file does not yet exist.

Example file with the returned results in it as built by example program `LEonardRoot/Code/Examples/Gocator/10p Static Repeatability.txt`

```
timestamp,gocator_ID,gc_decision,gc_offset_x,gc_offset_y,gc_offset_z,g
c_outer_radius,gc_depth,dc_bevel_radius,gc_bevel_angle,gc_xangle,gc_ya
ngle,gc_cb_depth,gc_axis_tilt,gc_axis_orient,gh_decision,gh_offset_x,g
h_offset_y,gh_offset_z,gh_radius,gp_xangle,gp_yangle,gp_z_offset,gp_st
d_dev
,,,in,in,in,in,in,in,deg,deg,deg,in,deg,deg,,in,in,in,in,deg,deg,in,in
2022-11-06 14:30:56,static_pose,0,-0.0448,0.2962,-
0.0272,0.2635,3.1,0.7,100.5,0.9,-0.2,0.0000,2.9,99.2,0,-
0.0439,0.2826,-0.0363,0.1331,-0.1,0.0,-0.0385,0.0220
2022-11-06 14:47:53,static_pose,0,-0.0447,0.2961,-
0.0272,0.2636,3.1,0.7,100.5,0.9,-0.2,0.0000,2.8,99.2,0,-
0.0439,0.2827,-0.0363,0.1332,-0.1,0.0,-0.0386,0.0222
```

```
2022-11-06 14:47:56,static_pose,0,-0.0447,0.2964,-
0.0273,0.2635,3.1,0.7,100.5,0.9,-0.2,0.0000,2.9,99.6,0,-
0.0432,0.2823,-0.0365,0.1335,-0.1,0.0,-0.0387,0.0226
```

Python Library for Gocator

Lecky Engineering provides a Python library for LEonard that duplicates the built-in functions in user-modifiable Python code.

Load the commands into your sequence and switch into Python processing to use them:

```
exec_python(Lib/leGocatorSupport.py)
using_python()
```

These commands are:

```
start_operation()
end_operation()
adjust_alignment()
offset_to_probe()
write_results()
```

Many example programs using the snapshot camera for alignment are in `LEonardRoot/Code/Examples/Gocator`.

`start_operation()`

Just performs a movement to `cp_origin`, a previously taught LEonard position assumed to be the start position and orientation of the part to be inspected. We tell the robot that this is part(0,0,0,0,0,0).

```
def start_operation():
  move_linear('cp_origin')
  movel_rel_set_part_origin_here()
```

`adjust_alignment(int version)`

Identical in performance to `gocator_adjust`. Feel free to copy and modify!

`offset_to_probe()`

A demo routine that simply moves the robot in part coordinates the distance assumed from the Gocator 0,0,0 to the tip of the tool.

```
def offset_to_probe():
  movel_incr_part(-0.0235,0,0.165,0,0,0)
```

`write_results(string filename, string tag_name)`

Identical in performance to `gocator_write_data`. Feel free to copy and modify!

`end_operation()`

Just moves back to the `cp_origin` home position.

```
def end_operation():
  move_linear('cp_origin')
```

How Does Adjust Alignment Work?

Here's the code. In this version, we're just trying to drive all Gocator positions to 0,0,0,0,0,0.

Translation movements in x,y,z are done in tool coordinates.

Rotations are done in Leonatrd Part coordinates, which means that if a spherical or cylindrical part has been specified, LEonard will follow the surface trajectory as part of it's X,Y,Z movement.

The example programs assume a cylindrical part which is setup in `LEonardRoot/Code/Examples/Gocator/00p Origin.txt`:

```
# 00p Set Origin
exec_python(Lib/leGocatorSupport.py)
using_python()

start_operation()

# Coupon is 762mm diam (32in)
coupon_diameter = 762.0
le_write_sysvar('coupon_diameter',str(coupon_diameter))

# Force all speeds and accelerations to good inspection ones!
linear_speed = 100
linear_accel = 100
joint_speed = 20
joint_accel = 10
blend_radius = 2

set_linear_speed(linear_speed)
set_linear_accel(linear_accel)
set_joint_speed(joint_speed)
set_joint_accel(joint_accel)
set_blend_radius(blend_radius)

grind_trial_speed(linear_speed)
grind_linear_accel(linear_accel)
```

```
    grind_max_blend_radius(blend_radius)

    select_tool('gocator3210')
    set_part_geometry('CYLINDER',coupon_diameter)

    end_operation()
```

For reference, the exact Python code for the adjustment function is as follows:

```
# adjust_alignment.py
# LMI Gocator Interface Code for LEonard
# Lecky Engineering LLC
# Author:      Ned Lecky
# Description: Moves the robot to drive the offsets from Gocator to 0
#
# This version:
#   Implements 4 versions of alignment- see code below
#
# Customize as needed for your application

# Moves robot to (hopefully!) drive alignment values to zero
# Version:
#  1: only adjusts translation
#  2: only adjusts rotation
#  3: adjusts in translation first, pauses 1S, and then does rotation
#  4: adjusts all 5 axes simultaneously but 3D calcs are not spot-on
def adjust_alignment(version):
  le_print('adjust_alignment starting...')
  dx = 0.0
  dy = 0.0
  dz = 0.0
  drx = 0.0
  dry = 0.0

  # Assumes using counterbore and thruhole tools. Uses counterbore if
result, else tries thruhole
  # NOTE: dz and drx are negated below to match sensor/robot
alignment!
  if gc_decision == '0':
    le_print('Using counterbore')
    dx = float(gc_offset_x) / 1e6
    dy = float(gc_offset_y) / 1e6
    dz = -float(gc_offset_z) / 1e6
    drx = -float(gc_xangle) / 1e3
    dry = float(gc_yangle) / 1e3
```

```
  elif gh_decision == '0':
    le_print('Using thruhole')
    dx = float(gh_offset_x) / 1e6
    dy = float(gh_offset_y) / 1e6
    dz = -float(gh_offset_z) / 1e6
    drx = -float(gp_xangle) / 1e3
    dry = float(gp_yangle) / 1e3
  else:
    le_print('No result found')
    return

  abs_dx = abs(dx)
  abs_dy = abs(dy)
  abs_dz = abs(dz)
  abs_drx = abs(drx)
  abs_dry = abs(dry)
  drx_rad = math.radians(drx)
  dry_rad = math.radians(dry)

  if version == 1:
    le_print('Version 1 Translation Only')
    if abs_dx > 0.020 or abs_dy > 0.020 or abs_dz > 0.020:
      le_print('Excessive gocator_adjust (' + str(dx) + ',' + str(dy)
+ ',' + str(dz) + ',0,0,0)')
    else:
      movel_incr_part(dx,dy,dz,0,0,0)
  elif version == 2:
    le_print('Version 2 Rotation Only')
    if abs_drx > 15 or abs_dry > 15:
      le_print('Excessive gocator_adjust (0,0,0,' + str(drx) + ',' +
str(dry) + ',0)')
    else:
      movel_incr_tool(0,0,0,drx_rad,dry_rad,0)
  elif version == 3:
    le_print('Version 3 translate, Pause 1sec, Rotate')
    if abs_dx > 0.020 or abs_dy > 0.020 or abs_dz > 0.020 or abs_drx >
15 or abs_dry > 15:
      le_print('Excessive gocator_adjust (' + str(dx) + ',' + str(dy)
+ ',' + str(dz) + ',' + str(drx_rad) +',' + str(dry_rad) + ',0)')
    else:
      movel_incr_part(dx,dy,dz,0,0,0)
      time.sleep(1)
      movel_incr_tool(0,0,0,drx_rad,dry_rad,0)
  elif version == 4:
    le_print('Version 4 All 5 axes at once- not quite accurate!')
```

```
    if abs_dx > 0.020 or abs_dy > 0.020 or abs_dz > 0.020 or abs_drx >
15 or abs_dry > 15:
      le_print('Excessive gocator_adjust (' + str(dx) + ',' + str(dy)
+ ',' + str(dz) + ',' + str(drx) +',' + str(dry) + ',0)')
    else:
      movel_incr_tool(dx,dy,dz,drx_rad,dry_rad,0)

  le_print('adjust_alignment complete')
```

INDEX