AutoGrind User Manual

For

TOSOH QUARTZ, INC.

Where Innovation Meets Automation®

Author: Ned Lecky
nlecky@olympus-controls.com
Software Rev 2022.6.16.12

June 16, 2022

Contents

Overview

This manual describes the use of the AutoGrind software.

There are four Operation Tabs: **Run**, **Program**, **Setup**, and **Log**. These screens are described below.

There are three User Modes. The user mode is selected using the **User** field in the upper-right corner of the Run tab

1. **Operator** Mode can only load and run existing recipes.
2. **Editor** Mode is Operator plus the user can create and edit recipes.
3. **Engineering** Mode is Editor plus access to all setup and configuration functions.

Entering Operator or Engineering mode requires a fixed password. By default, these are 9 and 99, respectively. They are not currently user settable.

Many functions can be manually activated with buttons or automatically activated with recipe commands. The convention in this manual is the used boldface for buttons and italics for recipe commands, as in **This Is a Button** and *this_is_a_recipe_function(param1)*.

System Tabs

Run Tab

The **Run Tab** is where the bulk of program execution will typically be observed.

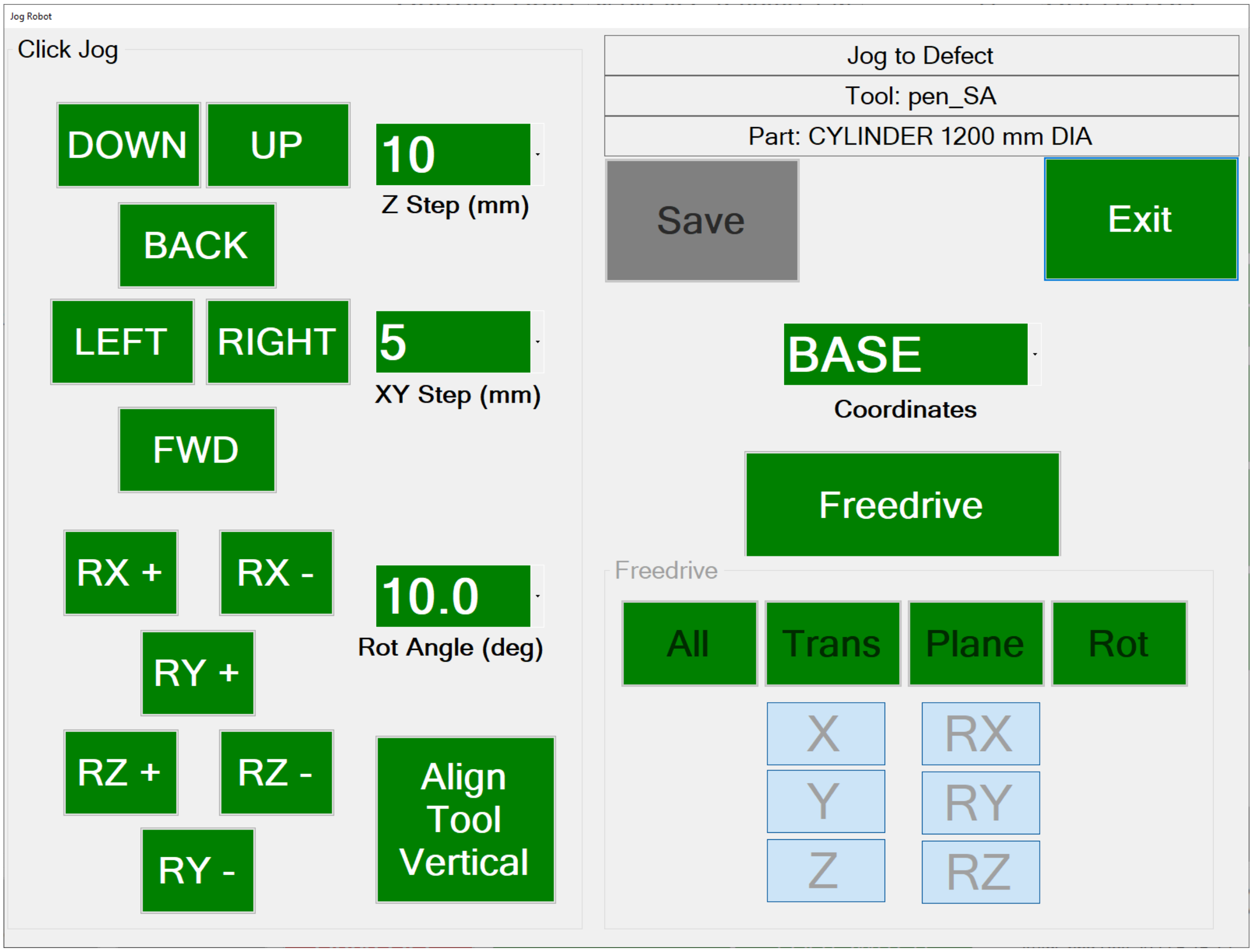
Tools are selected in the **Tool Dropdown.**
Part geometry in specified in **Part Geometry Dropdown:** FLAT, CYLINDER, and SPHERE
Tools have mount and home positions that can be moved into with the **toolname_mount** and **toolname_home** buttons.
A recipe is loaded using the **Recipe Name** button next to the **Log** tab.
Recipe file operations in addition to **Load** are **New, Save, Save As.**
Set the User in the **User Dropdown.**
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**).
Running a Recipe behaves as expected:

Start, **Stop**, **Step**, and **Pause / Continue**

Jog Robot is used to jog or freedrive to a defect. Jogging is described on the next page.

Robot Jogging in AutoGrind

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.



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.

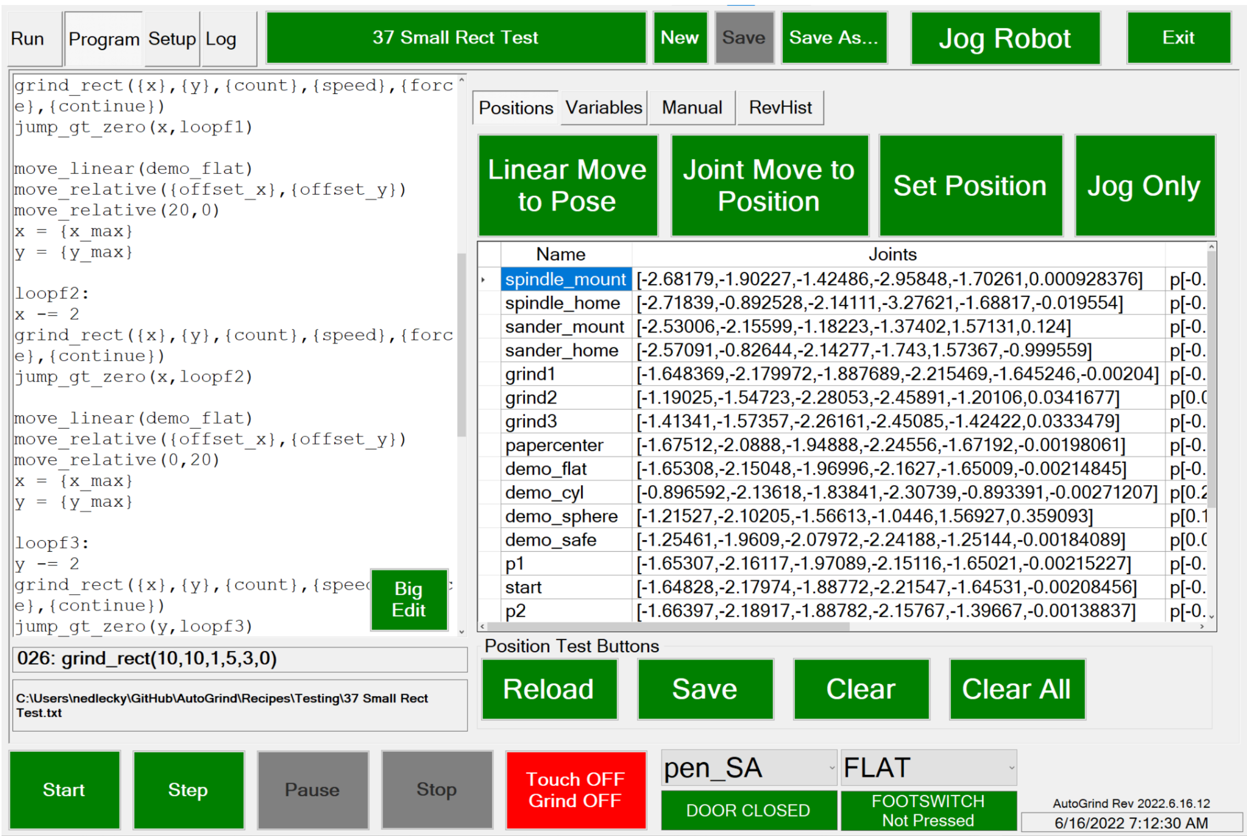
The **Freedrive footswitch** puts all 6 axes in freedrive whether this jog dialog is open or not.

Program Tab

The **Program Tab** has four sub pages: **Positions**, for teaching and manually moving to fixed positions, **Variables**, for monitoring or changing AutoGrind variables, **Manual,** to provide access to documentation, and **RevHist**, which displays what has been added in each version of the software. The current software version is always displayed in the lower right of the screen.

Program Tab - Positions

Below is the **Program Tab** when the Positions Subtab is selected.



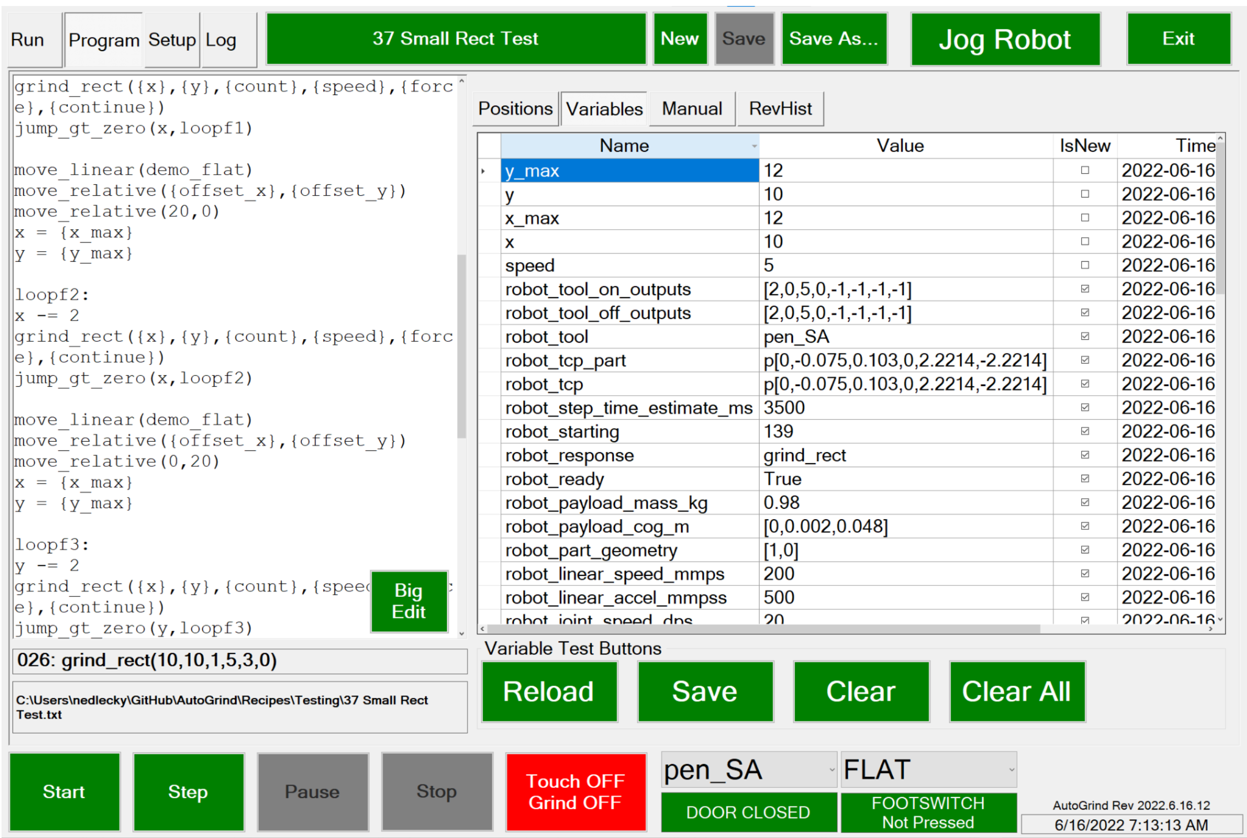
Positions can be saved manually (**Set Position**) or from the recipe with *save_position(name)*.

You can manually move to Positions in Joint (**Joint Move To Position**) or Linear (**Linear Move To Pose**) paths. These can also be executed from a recipe with *move_linear(position)* or *move_joint(position).*

Jogging is used here for setting or updating named positions or just for moving the robot. This uses the standard Jog screen.
Big Edit opens up a full-screen editor to make editing complex recipes easier.

Program Tab - Variables

Below is the **Program Tab** when the **Variables Subtab** is selected..



This tab shows all of the local variables maintained in AutoGrind for internal, system, and user purposes. They can be edited here as well.

System variables will not be erased by the **Clear** button, or by the recipe *clear()* command.

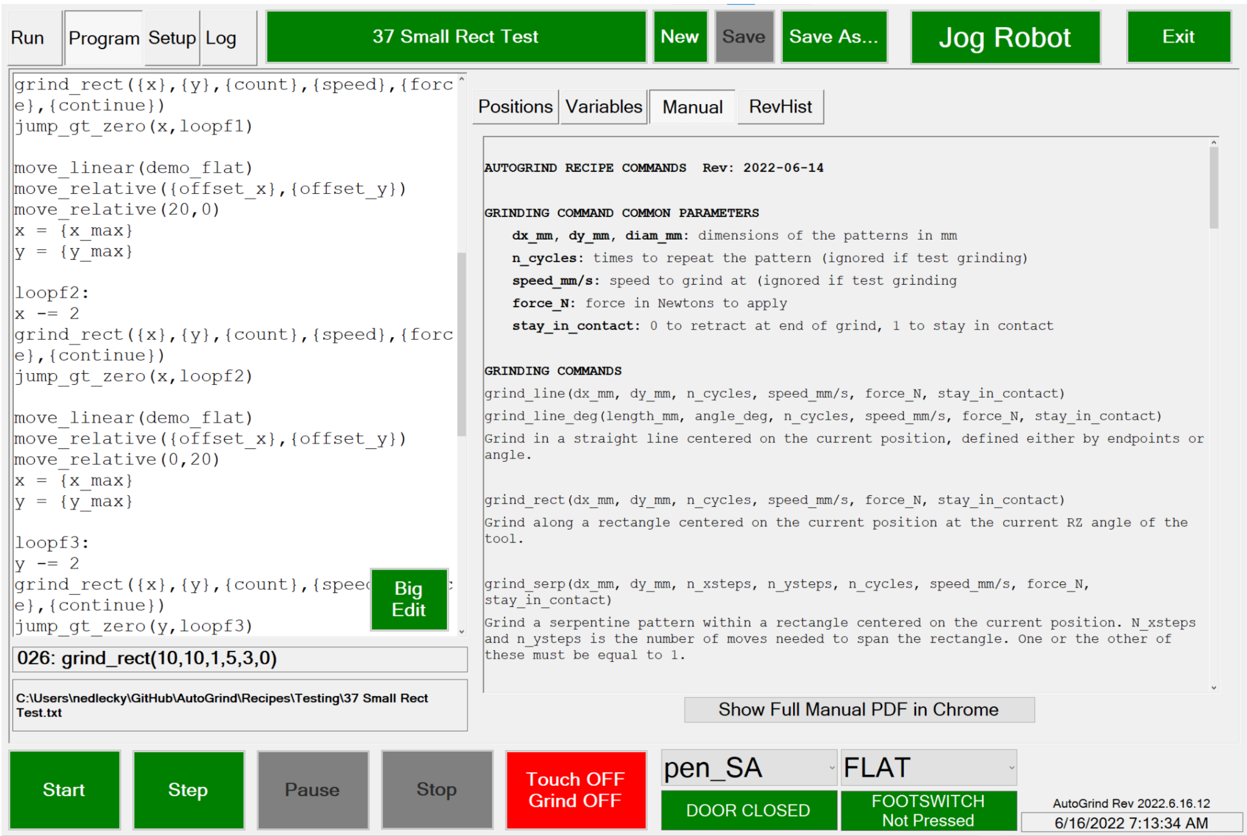
The **TimeStamp** shows when the variable was last written.

IsNew indicates whether the variable has ever been examined by the program since the last write.

The variables may be saved or reloaded from Recipes/Variables.xml with the **Save** and **Reload** buttons. Variables are automatically saved on program exit and reloaded when the program starts.

Program Tab - Manual

Below is the **Program Tab** when the **Manual Subtab** is selected..

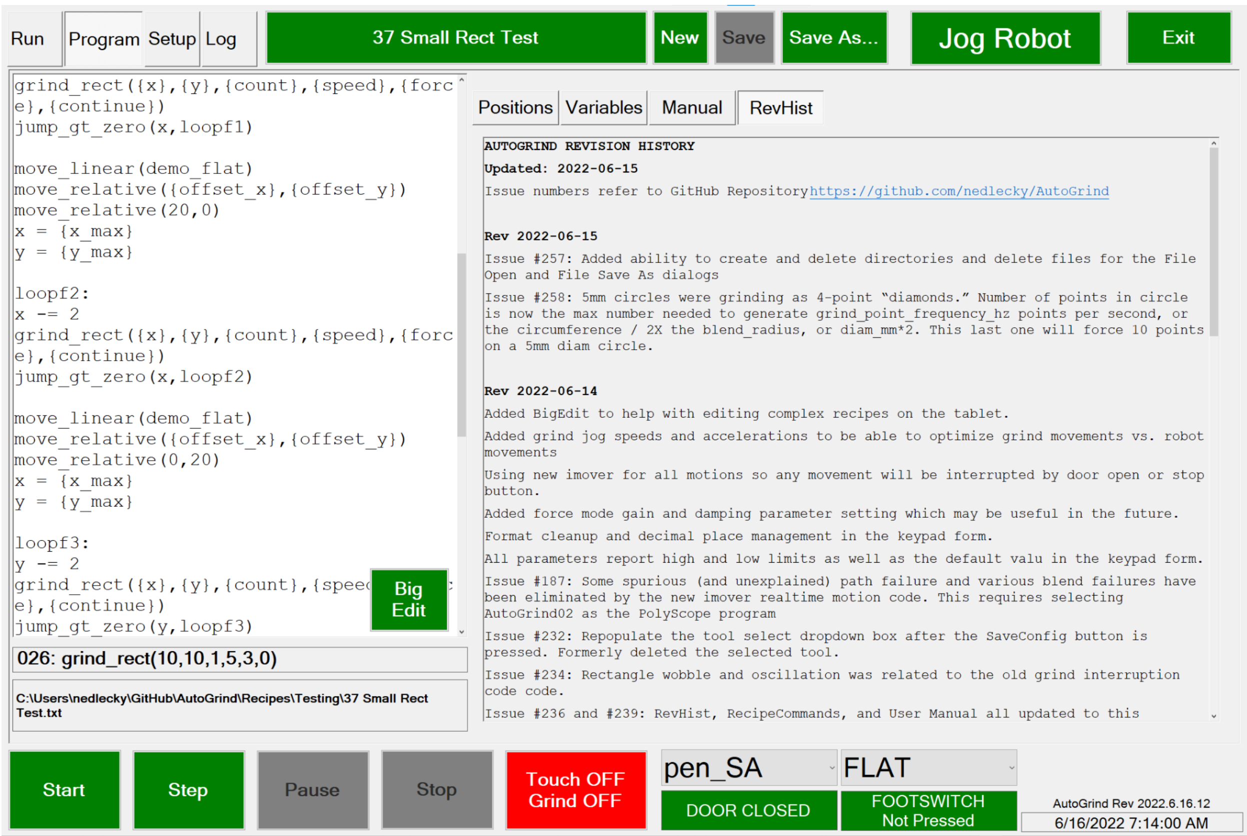


This tab displays the recipe commands to help you remember what each command does and how it is called..

The **Show Full Manual PDF in Chrome** button pulls up a PDF of the manual you are looking at in Chrome. It is assumed that Chrome is installed on the system.

Program Tab - RevHist

Below is the **Program Tab** when the **RevHist Subtab** is selected..

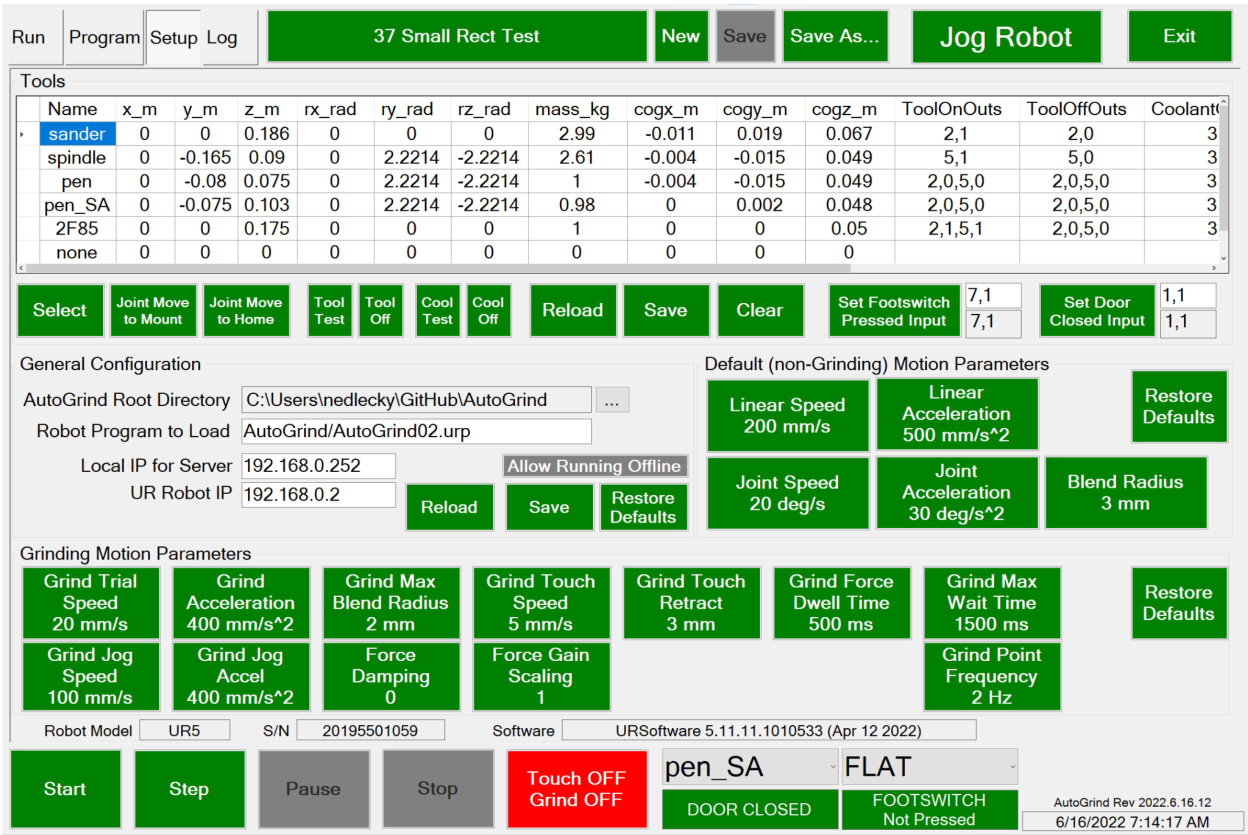


This tab displays the changes incorporated in the current (and prior) versions of the AutoGrind program. The issues addressed are described in more detail in the AutoGrind GitHub repository.

https://github.com/nedlecky/AutoGrind

Setup Tab

The **Setup Tab** is where all system configuration takes place.



Setup - General

1. **AutoGrind Root Directory:** Location where subdirectories Recipes and Logs will be created. Tools, Variables, and Positions are also stored here in the Recipes subfolder.
2. **Robot Program To Load:** Specifies the program on the UR that the UR will load and run when this software starts. Starting in version 2022-06-07, this should be **AutoGrind/AutoGrind02.urp**
3. **Local IP for Server:** This should be the IP address of the port on the host computer that is connected to the UR
4. **UR Robot IP:** The IP address that the UR is set to
5. **Allow Running Offline:** Testing only… will allow recipes to run with no UR attached

Setup - Tools

Tools are defined in the Tool Table. Each contains the following information. These are saved in the Tools.xml file in AutoGrindRoot/Recipes and are loaded and saved automatically.

1. **Tool TCP:** This is a copy of what we would teach for the tool on the UR including x, y, z offset and rx, ry, rz orientation. Teaching these is best done on the UR and then the values simply copied to the entry in AutoGrind

2. **Mass and Center of Gravity:** Set these as you would on the UR. Accurate settings improves behavior when in freedrive mode.
3. **ToolOnOuts, ToolOffOuts:** This is a list of up to 4 digital IOs that need to be turned on or off to enable the tool. This is only done during a grind in **Touch ON Grind ON** mode. Examples: “1,1,3,1” implies that output 1 should be set to 1 and output 3 should be set to 1. “3,1” implies that output 3 should be set to 1
4. **CoolantOnOuts, CoolantOffOuts:** Similarly, these are digital output commands to be executed when grinding in **TouchOn Grind ON** mode.
5. **MountPosition:** This is a position recommended for installing/removing this tool. The system will use joint moves to approach the position with **Joint Move To Mount** or *move_tool_mount()*. This must be a position that has been defined in the **Positions Table**.
6. **HomePosition:** This is a position recommended for homefor this tool. The system will use joint moves to approach the position with **Joint Move To Home** or *move_tool_home()*. This must be a position that has been defined in the **Positions Table**.
7. **Tool Test, Tool Off** and **Cool Test, Cool Off**: These allow manually verifying the outputs for the currently selected tool.
8. **Set Footswitch Pressed Input:** This is defaulted to 7,1 meaning that input 7 goes high when the footswitch is pressed.
9. **Set Door Closed Input:** This is defaulted to 1,1 meaning that input 1 goes high when the door is closed.

Setup - Default Motion Parameters

These are generally self-explanatory settings for speeds and accelerations used in jogging and non-grinding motion. These are saved in the Variables.xml file in AutoGrindRoot/Recipes and are sent to the robot whenever the software starts. New values are saved automatically.
Restore Defaults sets all to standard values used in all testing.

Setup - Grinding Motion Parameters

These are settings governing grind operations. These are saved in the Variables.xml file in AutoGrindRoot/Recipes and are sent to the robot whenever the software starts. New values are saved automatically.
Grind Trial Speed: When not in **Touch On Grind On** mode, the grind patterns are limited to one cycle and are performed at this speed.
Grind Acceleration: Linear acceleration used during grinding
Grind Max Blend Radius: Maximum blend radius used during grinding. Recommended 2 mm
Grind Touch Speed: Speed robot advances toward part for touchoff. Recommended 5-10 mm/s
Grind Touch Retract: Distance robot retracts from part after touchoff.

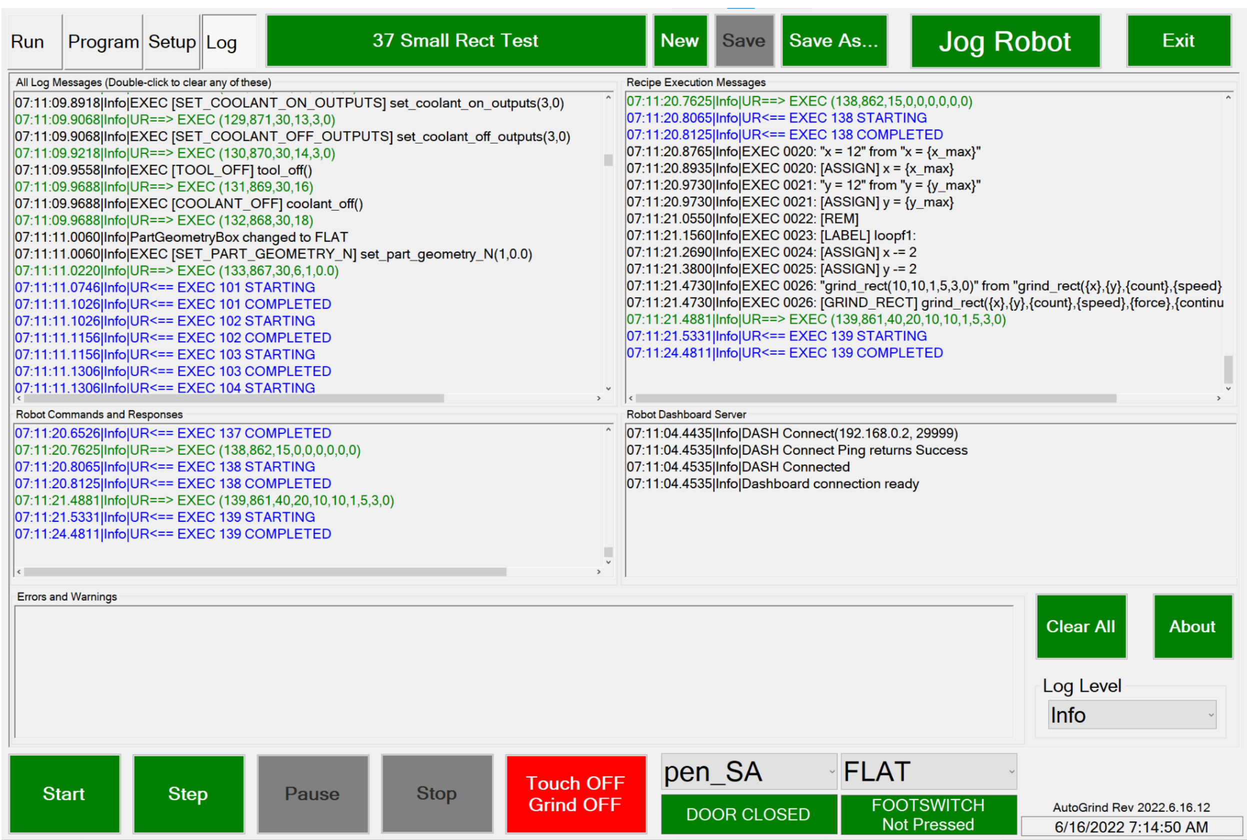
Grind Force Dwell Time: How long robot waits after turning force-on to allow time for tool to settle against part
Grind Max Wait Time: Maximum time system will wait for the next grind command if a grind command ends with 1 (stay in contact with part)
Grind Jog Speed: Linear speed used for all grinding motions that are not in contact with the part other than the actual simulated grind which runs at **Grind Trial Speed.**
Grind Jog Accel: Linear acceleration used for all grinding motions that are not in contact with the part.
Grind Point Frequency: Used as a minimum frequency for points generated for circles and spirals.
Force Damping: May be useful for force mode tuning in the future. Calls the URScript function force_mode_set_damping() with a value between 0 and 1. The default is 0 and that is the only value that has been tested.
Force Gain Scaling: May be useful for force mode tuning in the future. Calls the URScript function force_mode_set_gain_scaling() with a value between 0 and 2. The default is 1.0 and that is the only value that has been tested.

Log Tab

The Log Tab provides five windows where log messages are displayed. The level of detail in the messages is controlled by the Log Level setting:

- Error: only error messages are shown
- Warn: Error messages and Warnings are shown
- Info: All of the above, plus informational messages about execution. Default setting
- Debug: All of the above plus additional information that may be useful for debugging
- Trace: All of the above plus extremely verbose execution tracing

The All Log Messages box gets 100% of the generated messages. These messages are also written to log files in the AutoGrindRoot/Logs directory, where up to forty 25MB files are archived. Information older than this 2GB total is automatically and silently deleted over time.



In addition, some messages are copied for clarity to other boxes. The boxes are labeled with their respective data: Recipe Execution Messages, Robot Commands and Responses, Robot Dashboard (Control and Monitoring) Messages, and Errors/Warnings.
Any of the boxes can be cleared by double-clicking on them. All boxes can be cleared with **Clear All.** All of the messages are appended to the log files and are archived as described above.

Recipe Commands

This is a copy of the recipe commands document that is available from within the software.

```
AUTOGRIND RECIPE COMMANDS  Rev: 2022-06-14

GRINDING COMMAND COMMON PARAMETERS
  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

GRINDING COMMANDS
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.

select_tool(tool_name)
Setup all of the necessary environment to be able to use tool_name.
No motion is performed. Future grinds and position moves will assume
this tool is attached.

set_part_geometry(FLAT|CYLINDER|SPHERE, part_diam_mm)
Future grinds will assume the specified geometry.

The commands below provide a programmatic way to set the grinding
parameters.
grind_touch_retract(touch_retract_mm)
grind_touch_speed(touch_retract_speed_mm/s)
grind_force_dwell(dwell_time_ms)
grind_max_wait(max_time_before_retract_ms)
grind_max_blend_radius(grind_blend_radius_mm)
grind_trial_speed(trial_speed_mm/s)
grind_linear_accel(accel_mm/s^2)
grind_point_frequency(point_frequency_hz)
grind_jog_speed(point_frequency_hz)
grind_jog_accel(point_frequency_hz)
grind_force_mode_damping(damping: 0.0 – 1.0)
grind_force_mode_gain_scaling(scaling: 0.0 – 2.0)
```

VARIABLE MANIPULATION

```
import(filename)
Open up a file and perform any variable assignment (name = value)
lines found in it.

clear or clear()
Delete all variables except ones that are marked in the Variables
Table as system variables. (Variables named robot_* and grind_* are
automatically system variables.)

Update variables using any of these basic operations. Variables can
be inserted in any command using the syntax {var_name}.
var_name = 12.3      var_name = {other_var_name}
var_name++           var_name--
var_name -= 17.5     var_name += 18
```

FLOW CONTROL

label:
Labels a line in a program with a name. Label names are alphanumeric, case-sensitive, and may include the '_' character.

jump(label)
Jumps to the line after the label specified.

jump_gt_zero(var_name, label)
Jumps to the line after the label specified if the var_name is numeric and greater than 0.

end or end()
Terminate execution of a recipe.

prompt(message)
Prompt the operator with a message and pause execution until the dialog is acknowledged.

sleep(seconds)
Pause execution for the specified time. Fractional seconds may be used.

assert(var_name, value)
Testing support. Checks to see if var_name==value and generates an error message if not.

Comments
Blank lines are ignored.
Anything on a line after a '#' character is ignored.

MOTION

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)
set_linear_accel(accel_mm/s^2)
set_joint_speed(speed_deg/s)
set_joint_accel(accel_deg/s^2)
set_blend_radius(blend_radius_mm)
```

ENGINEERING USE ONLY

User Timers

```
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 for internal testing.
enable_user_timers(0=disable|1=enable)
zero_user_timers()
return_user_timers()
```

Low-level Setup Calls

```
These are all called automatically by select_tool(…),
set_part_geometry(..),and the grind_...() commands. They should be
used that way through the high level interface except during testing.
set_part_geometry_N(1=FLAT|2=CYLINDER|3=SPHERE, diam_mm)
set_tcp(x,y,x,rx,ry,rz)
set_payload(mass_kg,cog_x_m, cog_y_m, cog_z_m)
set_door_closed_input(dig_in, value)
set_footswitch_pressed_input(dig_in, value)
set_tool_on_outputs(dig_out, value, …) Can have up to 4 listed
set_tool_off_outputs(dig_out, value, …) Can have up to 4 listed
set_coolant_on_outputs(dig_out, value, …) Can have up to 4 listed
set_coolant_off_outputs(dig_out, value, …) Can have up to 4 listed
set_output(DOUT, 0|1)
tool_on()
tool_off()
coolant_on()
coolant_off()
send_robot(param1, param2, …)
```

Example Recipes

Here are a few recipes that show the kinds of things that can be done in a recipe. The **Testing** subdirectory in the Recipes folder has many more complicated examples that you can examine (and run!)

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 move_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)
```