```
# Copyright 2015-2019 - RoboDK Inc. - https://robodk.com/
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# ----------------------------------------------------
# This file is a POST PROCESSOR for Robot Offline Programming to generate prog
rams
# for a Universal Robot with RoboDK
#
# To edit/test this POST PROCESSOR script file:
# Select "Program"-
>"Add/Edit Post Processor", then select your post or create a new one.
# You can edit this file using any text editor or Python editor. Using a Pytho
n editor allows to quickly evaluate a sample program at the end of this file.
# Python should be automatically installed with RoboDK
#
# You can also edit the POST PROCESSOR manually:
#    1- Open the *.py file with Python IDLE (right click -> Edit with IDLE)
#    2- Make the necessary changes
#    3- Run the file to open Python Shell: Run -> Run module (F5 by default)
#    4- The "test_post()" function is called automatically
# Alternatively, you can edit this file using a text editor and run it with Py
thon
#
# To use a POST PROCESSOR file you must place the *.py file in "C:/RoboDK/Post
s/"
# To select one POST PROCESSOR for your robot in RoboDK you must follow these
steps:
#    1- Open the robot panel (double click a robot)
#    2- Select "Parameters"
#    3- Select "Unlock advanced options"
#    4- Select your post as the file name in the "Robot brand" box
#
# To delete an existing POST PROCESSOR script, simply delete this file (.py fi
le)
#
# ----------------------------------------------------
# More information about RoboDK Post Processors and Offline Programming here:
#     https://robodk.com/help#PostProcessor
#     https://robodk.com/doc/en/PythonAPI/postprocessor.html
# ----------------------------------------------------

DEFAULT_HEADER_SCRIPT = """
```

```
#-------------------------
# Add any default subprograms here in URScript Code
# For example, to drive a gripper as a program call:
# def Gripper_Open():
#   ...
# end
#
# Example to drive a spray gun:
def SprayOn(value):
  # use the value as an output:
  DO_SPRAY = 5
  if value == 0:
    set_standard_digital_out(DO_SPRAY, False)
  else:
    set_standard_digital_out(DO_SPRAY, True)
  end
end

# Example to drive an extruder:
def Extruder(value):
  # use the value as an output:
  if value < 0:
    # stop extruder
  else:
    # start extruder
  end
end

# Example to move an external axis
def MoveAxis(value):
  # use the value as an output:
  DO_AXIS_1 = 1
  DI_AXIS_1 = 1
  if value <= 0:
    set_standard_digital_out(DO_AXIS_1, False)

    # Wait for digital input to change state
    #while (get_standard_digital_in(DI_AXIS_1) != False):
    #  sync()
    #end
  else:
    set_standard_digital_out(DO_AXIS_1, True)

    # Wait for digital input to change state
    #while (get_standard_digital_in(DI_AXIS_1) != True):
    #  sync()
    #end
  end
end
```

```
    def moveExtruder(io_var, io_value):
        if value > 0:
            set_standard_digital_out(4, True)
            while (1 != 0):
                set_standard_digital_out(io_var, True)
                sleep(0.5)
                set_standard_digital_out(io_var, False)
                sleep(0.5)
            end
        if io_value < 0
            set_standard_digital_out(4, False)
            while (1 != 0):
                set_standard_digital_out(io_var, True)
                sleep(0.5)
                set_standard_digital_out(io_var, False)
                sleep(0.5)
            end
    end

  #-------------------------
"""

#SCRIPT_URP = '''<URProgram name="%s">
#  <children>
#    <MainProgram runOnlyOnce="false" motionType="MoveJ" speed="1.047197551196
5976" acceleration="1.3962634015954636" useActiveTCP="false">
#      <children>
#        <Script type="File">
#          <cachedContents>%s
#</cachedContents>
#          <file resolves-to="file">%s</file>
#        </Script>
#      </children>
#    </MainProgram>
#  </children>
#</URProgram>'''

#SCRIPT_URP = '''<URProgram createdIn="3.0.0" lastSavedIn="3.0.0" name="%s" di
rectory="/" installation="default">
#  <children>
#    <MainProgram runOnlyOnce="true" motionType="MoveJ" speed="1.0471975511965
976" acceleration="1.3962634015954636" useActiveTCP="false">
#      <children>
#        <Script type="File">
#          <cachedContents>%s
#</cachedContents>
#          <file resolves-to="file">%s</file>
#        </Script>
#      </children>
#    </MainProgram>
```

```
#  </children>
#</URProgram>'''

#<URProgram createdIn="3.4.3.361" lastSavedIn="3.4.3.361" name="%s" directory=
"." installation="default">
SCRIPT_URP = '''<URProgram createdIn="3.0.0" lastSavedIn="3.0.0" name="%s" dir
ectory="." installation="default">
  <children>
    <MainProgram runOnlyOnce="true" motionType="MoveJ" speed="1.04719755119659
76" acceleration="1.3962634015954636" useActiveTCP="false">
      <children>
        <Script type="File">
          <cachedContents>%s
</cachedContents>
          <file>%s</file>
        </Script>
      </children>
    </MainProgram>
  </children>
</URProgram>'''

def get_safe_name(progname):
    """Get a safe program name"""
    for c in r'-[]/\;,><&*:%=+@!#^|?^':
        progname = progname.replace(c,'')
    if len(progname) <= 0:
        progname = 'Program'
    if progname[0].isdigit():
        progname = 'P' + progname
    return progname

# ----------------------------------------------------
# Import RoboDK tools
from robodk import *

# ----------------------------------------------------
import socket
import struct
# UR information for real time control and monitoring
# Byte shifts for the real time packet:
UR_GET_RUNTIME_MODE = 132*8-4

RUNTIME_CANCELLED = 0
RUNTIME_READY = 1
RUNTIME_BUSY = 2

RUNTIME_MODE_MSG = []
RUNTIME_MODE_MSG.append("Operation cancelled") #0
RUNTIME_MODE_MSG.append("Ready") #1
RUNTIME_MODE_MSG.append("Running") #2 # Running or Jogging
```

```
# Get packet size according to the byte array
def UR_packet_size(buf):
    if len(buf) < 4:
        return 0
    return struct.unpack_from("!i", buf, 0)[0]

# Check if a packet is complete
def UR_packet_check(buf):
    msg_sz = UR_packet_size(buf)
    if len(buf) < msg_sz:
        print("Incorrect packet size %i vs %i" % (msg_sz, len(buf)))
        return False

    return True

# Get specific information from a packet
def UR_packet_value(buf, offset, nval=6):
    if len(buf) < offset+nval:
        print("Not available offset (maybe older Polyscope version?): %i - %i"
 % (len(buf), offset))
        return None
    format = '!'
    for i in range(nval):
        format+='d'
    return list(struct.unpack_from(format, buf, offset))    #return list(struc
t.unpack_from("!dddddd", buf, offset))


ROBOT_PROGRAM_ERROR = -1
ROBOT_NOT_CONNECTED = 0
ROBOT_OK = 1

def GetErrorMsg(rec_bytes):
    idx_error = -1
    try:
        idx_error = rec_bytes.index(b'error')
    except:
        return None

    if idx_error >= 0:
        idx_error_end = min(idx_error + 20, len(rec_bytes))
        try:
            idx_error_end = rec_bytes.index(b'\0',idx_error)
        except:
            return "Unknown error"
    return rec_bytes[idx_error:idx_error_end].decode("utf-8")

def UR_SendProgramRobot(robot_ip, data):
    print("POPUP: Connecting to robot...")
```

```
    sys.stdout.flush()
    robot_socket = socket.create_connection((robot_ip, 30002))
    print("POPUP: Sending program..")
    sys.stdout.flush()
    robot_socket.send(data)
    print("POPUP: Sending program...")
    sys.stdout.flush()
    pause(1)
    received = robot_socket.recv(4096)
    robot_socket.close()

    if received:
        #print("POPUP: Program running")
        #print(str(received))
        sys.stdout.flush()
        error_msg = GetErrorMsg(received)
        if error_msg:
            print("POPUP: Robot response: <strong>" + error_msg + "</strong>")
            sys.stdout.flush()
            pause(5)
            return ROBOT_PROGRAM_ERROR
        else:
            print("POPUP: Program sent. The program should be running on the r
obot.")
            sys.stdout.flush()
            return ROBOT_OK
    else:
        print("POPUP: Robot connection problems...")
        sys.stdout.flush()
        pause(2)
        return ROBOT_NOT_CONNECTED

# Monitor thread to retrieve information from the robot
def UR_Wait_Ready(robot_ip, percent_cmpl):
    RUNTIME_MODE_LAST = -1
    while True:
        print("Connecting to robot %s:%i" % (robot_ip, 30003))
        rt_socket = socket.create_connection((robot_ip, 30003))
        print("Connected")
        buf = b''
        while True:
            more = rt_socket.recv(4096)
            if more:
                buf = buf + more
                if UR_packet_check(buf):
                    packet_len = UR_packet_size(buf)
                    packet, buf = buf[:packet_len], buf[packet_len:]
                    RUNTIME_MODE = round(UR_packet_value(packet, UR_GET_RUNTIM
E_MODE, 1)[0])
                    if RUNTIME_MODE_LAST != RUNTIME_MODE:
```

```
                        RUNTIME_MODE_LAST = RUNTIME_MODE
                        if RUNTIME_MODE < len(RUNTIME_MODE_MSG):
                            print("POPUP: Robot " + RUNTIME_MODE_MSG[RUNTIME_M
ODE] + " (transfer in progress, %.1f%% completed)" % percent_cmpl)
                            sys.stdout.flush()
                        else:
                            print("POPUP: Robot Status Unknown (%.i)" % RUNTIM
E_MODE + " (transfer %.1f%% completed)" % percent_cmpl)
                            sys.stdout.flush()

                        if RUNTIME_MODE == RUNTIME_READY:
                            rt_socket.close()
                            return True

        rt_socket.close()
        return False

def pose_2_ur(pose):
    """Calculate the p[x,y,z,rx,ry,rz] position for a pose target"""
    NUMERIC_TOLERANCE = 1e-8;
    def saturate_1(value):
        return min(max(value,-1.0),1.0)

    angle = acos(  saturate_1((pose[0,0]+pose[1,1]+pose[2,2]-1)/2)   )
    rxyz = [pose[2,1]-pose[1,2], pose[0,2]-pose[2,0], pose[1,0]-pose[0,1]]
    if angle < NUMERIC_TOLERANCE:
        rxyz = [0,0,0]
    else:
        sin_angle = sin(angle)
        if abs(sin_angle) < NUMERIC_TOLERANCE:
            d3 = [pose[0,0],pose[1,1],pose[2,2]]
            mx = max(d3)
            mx_id = d3.index(mx)
            if mx_id == 0:
                rxyz = [pose[0,0]+1, pose[1,0]  , pose[2,0]  ]
            elif mx_id == 1:
                rxyz = [pose[0,1]  , pose[1,1]+1, pose[2,1]  ]
            else:
                rxyz = [pose[0,2]  , pose[1,2]  , pose[2,2]+1]

            rxyz = mult3(rxyz, angle/(sqrt(max(0,2*(1+mx)))))
        else:
            rxyz = normalize3(rxyz)
            rxyz = mult3(rxyz, angle)
    return [pose[0,3], pose[1,3], pose[2,3], rxyz[0], rxyz[1], rxyz[2]]

def pose_2_str(pose):
    """Prints a pose target"""
    [x,y,z,w,p,r] = pose_2_ur(pose)
    MM_2_M = 0.001
```

```
    return ('p[%.6f, %.6f, %.6f, %.6f, %.6f, %.6f]' % (x*MM_2_M,y*MM_2_M,z*MM_
2_M,w,p,r))

def angles_2_str(angles):
    """Prints a joint target"""
    njoints = len(angles)
    d2r = pi/180.0
    if njoints == 6:
        return ('[%.6f, %.6f, %.6f, %.6f, %.6f, %.6f]' % (angles[0]*d2r, angle
s[1]*d2r, angles[2]*d2r, angles[3]*d2r, angles[4]*d2r, angles[5]*d2r))
    else:
        return 'this post only supports 6 joints'

def circle_radius(p0,p1,p2):
    a = norm(subs3(p0,p1))
    b = norm(subs3(p1,p2))
    c = norm(subs3(p2,p0))
    radius = a*b*c/sqrt(pow(a*a+b*b+c*c,2)-
2*(pow(a,4)+pow(b,4)+pow(c,4)))
    return radius

#def distance_p1_p02(p0,p1,p2):
#    v01 = subs3(p1, p0)
#    v02 = subs3(p2, p0)
#    return dot(v02,v01)/dot(v02,v02)

# ----------------------------------------------------
# Object class that handles the robot instructions/syntax
class RobotPost(object):
    """Robot post object"""
    MAX_LINES_X_PROG = 250    # Maximum number of lines per program. If the nu
mber of lines is exceeded, the program will be executed step by step by RoboDK
    PROG_EXT = 'script'        # set the program extension
    SPEED_MS       = 0.3    # default speed for linear moves in m/s
    SPEED_RADS     = 0.75   # default speed for joint moves in rad/s
    ACCEL_MSS      = 3      # default acceleration for lineaer moves in m/ss
    ACCEL_RADSS    = 1.2    # default acceleration for joint moves in rad/ss
    BLEND_RADIUS_M = 0.001  # default blend radius in meters (corners smoothin
g)
    MOVEC_MIN_RADIUS = 1    # minimum circle radius to output (in mm). It does
 not take into account the Blend radius
    MOVEC_MAX_RADIUS = 10000  # maximum circle radius to output (in mm). It do
es not take into account the Blend radius
    USE_MOVEP = False
    #-------------------------------
    REF_FRAME      = eye(4) # default reference frame (the robot reference fra
me)
    LAST_POS_ABS = None # last XYZ position

    # other variables
```

```
    ROBOT_POST = 'unset'
    ROBOT_NAME = 'generic'
    PROG_FILES = []
    MAIN_PROGNAME = 'unknown'

    # 3D Printing Extruder Setup Parameters:
    PRINT_E_AO = 5 # Analog Output ID to command the extruder flow
    PRINT_FLOW_2_SIGNAL = 0.05 # Ratio to convert the flow to an analog signal
    PRINT_FLOW_MAX_SIGNAL = 24 # Maximum signal to provide to the Extruder
    PRINT_ACCEL_MMSS = 1e9 # Acceleration (assume constant speed if we use rou
nding/blending)

    # Internal 3D Printing Parameters
    PRINT_POSE_LAST = None # Last pose printed
    PRINT_E_LAST = 0 # Last Extruder length
    PRINT_E_NEW = 0 # New Extruder Length

    ####Added from manual####
    # 3D Printing Extruder Setup Parameters:
    PRINT_E_AO = 5 # Analog Output ID to command the extruder flow
    PRINT_SPEED_2_SIGNAL = 0.10 # Ratio to convert the speed/flow to an analog
 output signal
    PRINT_FLOW_MAX_SIGNAL = 24 # Maximum signal to provide to the Extruder
    PRINT_ACCEL_MMSS = -1 # Acceleration, -
1 assumes constant speed if we use rounding/blending

    # Internal 3D Printing Parameters
    PRINT_POSE_LAST = None # Last pose printed
    PRINT_E_LAST = 0 # Last Extruder length
    PRINT_E_NEW = None # New Extruder Length
    PRINT_LAST_SIGNAL = None # Last extruder signal
    ##########################

    nPROGS = 0
    PROG = []
    PROG_LIST = []
    VARS = []
    VARS_LIST = []
    SUBPROG = []
    TAB = ''
    LOG = ''

    def __init__(self, robotpost=None, robotname=None, robot_axes = 6, **kwarg
s):
        self.ROBOT_POST = robotpost
        self.ROBOT_NAME = robotname
        for k,v in kwargs.items():
            if k == 'lines_x_prog':
                self.MAX_LINES_X_PROG = v
```

```
    def ProgStart(self, progname):
        progname = get_safe_name(progname)
        self.nPROGS = self.nPROGS + 1
        if self.nPROGS <= 1:
            self.TAB = ''
            # Create global variables:
            self.vars_update()
            self.MAIN_PROGNAME = progname
        else:
            self.addline('# Subprogram %s' % progname)
            self.addline('def %s():' % progname)
            self.TAB = '  '

    def ProgFinish(self, progname):
        progname = get_safe_name(progname)
        self.TAB = ''
        if self.nPROGS <= 1:
            self.addline('# End of main program')
        else:
            self.addline('end')
            self.addline('')

    def ProgSave(self, folder, progname, ask_user = False, show_result = False
):
        progname = get_safe_name(progname)
        progname = progname + '.script'# + self.PROG_EXT
        if ask_user or not DirExists(folder):
            filesave = getSaveFile(folder, progname, 'Save program as...')
            if filesave is not None:
                filesave = filesave.name
            else:
                return
        else:
            filesave = folder + '/' + progname

        self.prog_2_list()

        fid = open(filesave, "w")
        # Create main program call:
        fid.write('def %s():\n' % self.MAIN_PROGNAME)

        # Add global parameters:
        fid.write('  # Global parameters:\n')
        for line in self.VARS_LIST[0]:
            fid.write('  ' + line+ '\n')
        #fid.write('  \n')
        fid.write('  ')

        # Add a custom header if desired:
        fid.write(DEFAULT_HEADER_SCRIPT)
```

```
        fid.write('  \n')

        # Add the suprograms that are being used in RoboDK
        for line in self.SUBPROG:
            fid.write('  ' + line + '\n')
        fid.write('  \n')

        # Add the main code:
        fid.write('  # Main program:\n')
        for prog in self.PROG_LIST:
            for line in prog:
                fid.write('  ' + line + '\n')

        fid.write('end\n\n')
        fid.write('%s()\n' % self.MAIN_PROGNAME)

        fid.close()

        print('SAVED: %s\n' % filesave) # tell RoboDK the path of the saved fi
le
        self.PROG_FILES = filesave

        #---------------------------- SAVE URP (GZIP compressed XML file)-----
--------------------
        filesave_urp = filesave[:-7] #+'.urp'
        fid = open(filesave, "r")
        prog_final = fid.read()
        fid.close()

        try:
            from html import escape  # python 3.x
        except ImportError:
            from cgi import escape  # python 2.x

        prog_final_ok = escape(prog_final)
        self.PROG_XML = SCRIPT_URP % (self.MAIN_PROGNAME, prog_final_ok, self.
MAIN_PROGNAME+'.script')

        # Comment next line to force transfer of the SCRIPT file
        #self.PROG_FILES = filesave_urp

        import gzip
        import os
        with gzip.open(filesave_urp, 'wb') as fid_gz:
            fid_gz.write(self.PROG_XML.encode('utf-8'))

        try:
            os.remove(filesave_urp+'.urp')
        except OSError:
            pass
```

```
        os.rename(filesave_urp, filesave_urp+'.urp')

        #print('SAVED: %s\n' % filesave_urp) # tell RoboDK the path of the sav
ed file
        #----------------------------------------------------------------------
---------------------

        # open file with default application
        if show_result:
            if type(show_result) is str:
                # Open file with provided application
                import subprocess
                p = subprocess.Popen([show_result, filesave])
            elif type(show_result) is list:
                import subprocess
                p = subprocess.Popen(show_result + [filesave])
            else:
                # open file with default application
                os.startfile(filesave)
            if len(self.LOG) > 0:
                mbox('Program generation LOG:\n\n' + self.LOG)

        #if len(self.PROG_LIST) > 1:
        #    mbox("Warning! The program " + progname + " is too long and direc
tly running it on the robot controller might be slow. It is better to run it f
orm RoboDK.")


    def ProgSendRobot(self, robot_ip, remote_path, ftp_user, ftp_pass):
        """Send a program to the robot using the provided parameters. This met
hod is executed right after ProgSave if we selected the option "Send Program t
o Robot".
        The connection parameters must be provided in the robot connection men
u of RoboDK"""
        #UploadFTP(self.PROG_FILES, robot_ip, remote_path, ftp_user, ftp_pass)
        #return

        nprogs = len(self.PROG_LIST)
        for i in range(nprogs):
            # Prepare next program execution:
            send_str = ''
            send_str += ('def %s():\n' % self.MAIN_PROGNAME)

            # Add global parameters:
            send_str += ('  # Global parameters:\n')
            for line in self.VARS_LIST[i]:
                send_str += ('  ' + line + '\n')
            send_str += ('  \n')
```

```
            # Add a custom header if desired:
            send_str += (DEFAULT_HEADER_SCRIPT)
            send_str += ('  \n')

            for line in self.SUBPROG:
                send_str += '  ' + line+ '\n'
            send_str += ('  \n')

            # Add the main code:
            send_str += ('  # Main program:\n')

            for line in self.PROG_LIST[i]:
                send_str += '  ' + line
                send_str += '\n'

            send_str += 'end\n\n'
            send_str += '%s()\n' % self.MAIN_PROGNAME

            send_bytes = str.encode(send_str)

            # Wait until the robot is ready:
            while i > 0 and not UR_Wait_Ready(robot_ip, i*100.0/nprogs):
                print("POPUP: Connect robot to run the program program...")
                sys.stdout.flush()
                pause(2)

            # Send script to the robot:
            #print(send_str)
            #input("POPUP: Enter to continue")
            status = UR_SendProgramRobot(robot_ip, send_bytes)
            while ROBOT_NOT_CONNECTED == status:
                print("POPUP: Connect robot to transfer program...")
                sys.stdout.flush()
                pause(2)
                status = UR_SendProgramRobot(robot_ip, send_bytes)

            if status == ROBOT_PROGRAM_ERROR:
                print("POPUP: Program Error. Running program from the computer
Aborted.")
                sys.stdout.flush()
                pause(2)
                return


    def blend_radius_check(self, pose_abs, ratio_check=0.4):
        # check that the blend radius covers 40% of the move (at most)
        blend_radius = 'blend_radius_m';
        #return blend_radius
        current_pos = pose_abs.Pos()
        if self.LAST_POS_ABS is None:
```

```
            blend_radius = '0'
        else:
            distance = norm(subs3(self.LAST_POS_ABS, current_pos)) # in mm
            if ratio_check*distance < self.BLEND_RADIUS_M*1000:
                blend_radius = '%.3f' % (round(ratio_check*distance*0.001,3))
        #self.LAST_POS_ABS = current_pos
        return blend_radius

    def MoveJ(self, pose, joints, conf_RLF=None):
        """Add a joint movement"""
        if pose is None:
            blend_radius = "0"
            self.LAST_POS_ABS = None
        else:
            pose_abs = self.REF_FRAME*pose
            blend_radius = self.blend_radius_check(pose_abs)
            self.LAST_POS_ABS = pose_abs.Pos()

        if len(joints) < 6:
            self.RunMessage('Move axes to: ' + angles_2_str(joints))
        else:
            self.addline('movej(%s,accel_radss,speed_rads,0,%s)' % (angles_2_s
tr(joints), blend_radius))


####Added from manual####
    def calculate_time(self, distance, Vmax, Amax=-1):
        """Calculate the time to move a distance with Amax acceleration and Vm
ax speed"""
        if Amax <0:
            # Assume constant speed (appropriate smoothing/rounding parameter
must be set)
            Ttot = distance/Vmax
        else:
            # Assume we accelerate and decelerate
            tacc = Vmax/Amax;
            Xacc = 0.5*Amax*tacc*tacc;
            if distance <=2*Xacc:
                # Vmax is not reached
                tacc = sqrt(distance/Amax)
                Ttot = tacc*2
            else:
                # Vmax is reached
                Xvmax = distance - 2*Xacc
                Tvmax = Xvmax/Vmax
                Ttot = 2*tacc + Tvmax
        return Ttot

    def new_move(self, new_pose):
```

```
        """Implement the action on the extruder for 3D printing, if applicable
"""
        if self.PRINT_E_NEW is None or new_pose is None:
            return

        # Skip the first move and remember the pose
        if self.PRINT_POSE_LAST is None:
            self.PRINT_POSE_LAST = new_pose
            return

        # Calculate the increase of material for the next movement
        add_material = self.PRINT_E_NEW - self.PRINT_E_LAST
        self.PRINT_E_LAST = self.PRINT_E_NEW

        # Calculate the robot speed and Extruder signal
        extruder_signal = 0
        if add_material > 0:
            distance_mm = norm(subs3(self.PRINT_POSE_LAST.Pos(), new_pose.Pos(
)))
            # Calculate movement time in seconds
            time_s = self.calculate_time(distance_mm, self.SPEED_MMS, self.PRI
NT_ACCEL_MMSS)

            # Avoid division by 0
            if time_s >0:
                # This may look redundant but it allows you to account for acc
elerations and we can apply small speed adjustments
                speed_mms = distance_mm / time_s

                # Calculate the extruder speed in RPM*Ratio (PRINT_SPEED_2_SIG
NAL)
                extruder_signal = speed_mms * self.PRINT_SPEED_2_SIGNAL

        # Make sure the signal is within the accepted values
        extruder_signal = max(0,min(self.PRINT_FLOW_MAX_SIGNAL, extruder_signa
l))

        # Update the extruder speed when required
        if self.PRINT_LAST_SIGNAL is None or abs(extruder_signal - self.PRINT_
LAST_SIGNAL) > 1e-6:
            self.PRINT_LAST_SIGNAL = extruder_signal
            # Use the built-in setDO function to set an analog output
            self.callExtruder(self.PRINT_E_AO, extruder_signal)
            # Alternatively, provoke a program call and handle the integration
 with the robot controller
            #self.addline('ExtruderSpeed(%.3f)' % extruder_signal)

        # Remember the last pose
        self.PRINT_POSE_LAST = new_pose
########################
```

```
    def MoveL(self, pose, joints, conf_RLF=None):
        """Add a linear movement"""
        # Movement in joint space or Cartesian space should give the same resu
lt:
        # pose_wrt_base = self.REF_FRAME*pose
        # self.addline('movel(%s,accel_mss,speed_ms,0,blend_radius_m)' % (pose
_2_str(pose_wrt_base)))
        self.new_move(pose) # used for 3D printing

        if pose is None:
            blend_radius = "0"
            self.LAST_POS = None
            target = angles_2_str(joints)
        else:
            pose_abs = self.REF_FRAME*pose
            blend_radius = self.blend_radius_check(pose_abs)
            target = pose_2_str(pose_abs)
            self.LAST_POS_ABS = pose_abs.Pos()

        if self.USE_MOVEP:
            self.addline('movep(%s,accel_mss,speed_ms,%s)' % (target, blend_ra
dius))
        else:
            self.addline('movel(%s,accel_mss,speed_ms,0,%s)' % (target, blend_
radius))


    def MoveC(self, pose1, joints1, pose2, joints2, conf_RLF_1=None, conf_RLF_
2=None):
        """Add a circular movement"""
        pose1_abs = self.REF_FRAME*pose1
        pose2_abs = self.REF_FRAME*pose2
        p0 = self.LAST_POS_ABS
        p1 = pose1_abs.Pos()
        p2 = pose2_abs.Pos()
        if p0 is None:
            self.MoveL(pose2, joints2, conf_RLF_2)
            return

        radius = circle_radius(p0, p1, p2)
        print("MoveC Radius: " + str(radius) + " mm")
        if radius < self.MOVEC_MIN_RADIUS or radius > self.MOVEC_MAX_RADIUS:
            self.MoveL(pose2, joints2, conf_RLF_2)
            return

        blend_radius = self.blend_radius_check(pose1_abs, 0.2)
        #blend_radius = '%.3f' % (0.001*radius) #'0'
        #blend_radius = '0'
        self.LAST_POS_ABS = pose2_abs.Pos()
```

```
        #self.addline('movec(%s,%s,accel_mss,speed_ms,%s)' % (angles_2_str(joi
nts1),angles_2_str(joints2), blend_radius))
        self.addline('movec(%s,%s,accel_mss,speed_ms,%s)' % (pose_2_str(pose1_
abs),pose_2_str(pose2_abs), blend_radius))

    def setFrame(self, pose, frame_id=None, frame_name=None):
        """Change the robot reference frame"""
        # the reference frame is not needed if we use joint space for joint an
d linear movements
        # the reference frame is also not needed if we use cartesian moves wit
h respect to the robot base frame
        # the cartesian targets must be pre-
multiplied by the active reference frame
        self.REF_FRAME = pose
        self.addline('# set_reference(%s)' % pose_2_str(pose))

    def setTool(self, pose, tool_id=None, tool_name=None):
        """Change the robot TCP"""
        self.addline('set_tcp(%s)' % pose_2_str(pose))
        #self.addline('set_payload(1.4, [-0.1181, -0.1181, 0.03])')
        #self.addline('set_gravity([0.0, 0.0, 9.82]))')

    def Pause(self, time_ms):
        """Pause the robot program"""
        if time_ms <= 0:
            self.addline('halt() # reimplement this function to force stop')
        else:
            self.addline('sleep(%.3f)' % (time_ms*0.001))

    def setSpeed(self, speed_mms):
        """Changes the robot speed (in mm/s)"""
        #if speed_mms < 999.9:
        #    self.USE_MOVEP = True
        #else:
        #    self.USE_MOVEP = False
        self.SPEED_MMS = speed_mms
        self.SPEED_MS = speed_mms/1000.0
        self.addline('speed_ms    = %.3f' % self.SPEED_MS)

    def setAcceleration(self, accel_mmss):
        """Changes the robot acceleration (in mm/s2)"""
        self.ACCEL_MSS = accel_mmss/1000.0
        self.addline('accel_mss   = %.3f' % self.ACCEL_MSS)

    def setSpeedJoints(self, speed_degs):
        """Changes the robot joint speed (in deg/s)"""
        self.SPEED_RADS = speed_degs*pi/180
        self.addline('speed_rads  = %.3f' % self.SPEED_RADS)

    def setAccelerationJoints(self, accel_degss):
```

```
        """Changes the robot joint acceleration (in deg/s2)"""
        self.ACCEL_RADSS = accel_degss*pi/180
        self.addline('accel_radss = %.3f' % self.ACCEL_RADSS)

    def setZoneData(self, zone_mm):
        """Changes the zone data approach (makes the movement more smooth)"""
        if zone_mm < 0:
            zone_mm = 0
        self.BLEND_RADIUS_M = zone_mm / 1000.0
        self.addline('blend_radius_m = %.3f' % self.BLEND_RADIUS_M)

    def setDO(self, io_var, io_value):
        """Set a Digital Output"""
        print(self) #for testing
        print('io_var= ' + str(io_var)) #for testing
        print('io_value= ' + str(io_value)) #for testing
        if type(io_value) != str: # set default variable value if io_value is
a number
            if io_value > 0:
                io_value = 'True'
            else:
                io_value = 'False'
        print('io_value after manipulation= ' + str(io_value))  #for testing

        if type(io_var) != str:  # set default variable name if io_var is a nu
mber
            newline = 'set_standard_digital_out(%s, %s)' % (str(io_var), io_va
lue)
        else:
            newline = '%s = %s' % (io_var, io_value)

        self.addline(newline)

###Added by jonas###
    def setAO(self, io_var, io_value):
        """Set an Analog Output"""
        print(self) #for testing
        print('io_var= ' + str(io_var)) #for testing
        print('io_value= ' + str(io_value)) #for testing
        if type(io_value) != str:
            io_value = round(io_value, 3)
            print('io_value after manipulation= ' + str(io_value))  #for testi
ng
        if type(io_var) != str and type(io_value) != str:  # set default varia
ble name if io_var and io_value is a number
            newline = 'set_standard_analog_out(%s, %s)' % (str(io_var), io_val
ue)
        else:
            newline = 'Worng Variable Type: %s = %s' % (io_var, io_value)
```

```
        self.addline(newline)

    def callExtruder(self, io_var, io_value):
        newline = 'moveExtruder(%s, %s)' % (io_var, io_value)
        self.addline(newline)

####################

    def waitDI(self, io_var, io_value, timeout_ms=-1):
        """Waits for an input io_var to attain a given value io_value. Optiona
lly, a timeout can be provided."""
        if type(io_var) != str:  # set default variable name if io_var is a nu
mber
            io_var = 'get_standard_digital_in(%s)' % str(io_var)
        if type(io_value) != str: # set default variable value if io_value is
a number
            if io_value > 0:
                io_value = 'True'
            else:
                io_value = 'False'

        # at this point, io_var and io_value must be string values
        #if timeout_ms < 0:
        self.addline('while (%s != %s):' % (io_var, io_value))
        self.addline('  sync()')
        self.addline('end')

    ####Added from manual####
    def RunCode(self, code, is_function_call = False):
        if is_function_call:
            if code.startswith("Extruder("):
              # Intercept the extruder command.
               # if the program call is Extruder(123.56)
               # we extract the number as a string
               # and convert it to a number
                self.PRINT_E_NEW = float(code[9:-1])
                # Skip the program call generation
                return
            else:
                self.addline(code + "()")
        else:
            # Output program code
            self.addline(code)
    ##########################

    def RunMessage(self, message, iscomment = False):
        """Show a message on the controller screen"""
        if iscomment:
            self.addline('# ' + message)
        else:
```

```
            self.addline('popup("%s","Message",False,False,blocking=True)' % m
essage)

# ------------------ private ----------------------
    def vars_update(self):
        # Generate global variables for this program
        self.VARS = []
        self.VARS.append('global speed_ms    = %.3f' % self.SPEED_MS)
        self.VARS.append('global speed_rads  = %.3f' % self.SPEED_RADS)
        self.VARS.append('global accel_mss   = %.3f' % self.ACCEL_MSS)
        self.VARS.append('global accel_radss = %.3f' % self.ACCEL_RADSS)
        self.VARS.append('global blend_radius_m = %.3f' % self.BLEND_RADIUS_M)

    def prog_2_list(self):
        if len(self.PROG) > 1:
            self.PROG_LIST.append(self.PROG)
            self.PROG = []
            self.VARS_LIST.append(self.VARS)
            self.VARS = []
            self.vars_update()

    def addline(self, newline):
        """Add a program line"""
        if self.nPROGS <= 1:
            if len(self.PROG) > self.MAX_LINES_X_PROG:
                self.prog_2_list()

            self.PROG.append(self.TAB + newline)
        else:
            self.SUBPROG.append(self.TAB + newline)

    def addlog(self, newline):
        """Add a log message"""
        self.LOG = self.LOG + newline + '\n'

# ------------------------------------------------
# ------------ For testing purposes ---------------
def Pose(xyzrpw):
    [x,y,z,r,p,w] = xyzrpw
    a = r*math.pi/180
    b = p*math.pi/180
    c = w*math.pi/180
    ca = math.cos(a)
    sa = math.sin(a)
    cb = math.cos(b)
    sb = math.sin(b)
    cc = math.cos(c)
    sc = math.sin(c)
    return Mat([[cb*ca, ca*sc*sb - cc*sa, sc*sa + cc*ca*sb, x],[cb*sa, cc*ca +
 sc*sb*sa, cc*sb*sa - ca*sc, y],[-sb, cb*sc, cc*cb, z],[0,0,0,1]])
```

```
def test_post():
    """Test the post with a basic program"""

    robot = RobotPost('Universal Robotics', 'Generic UR robot')

    robot.ProgStart("Program")
    robot.RunMessage("Program generated by RoboDK", True)
    robot.setFrame(Pose([807.766544, -963.699898, 41.478944, 0, 0, 0]))
    robot.setTool(Pose([62.5, -108.253175, 100, -60, 90, 0]))
    """
    robot.setSpeed(100) # set speed to 100 mm/s
    robot.setAcceleration(3000) # set speed to 3000 mm/ss
    robot.MoveJ(Pose([200, 200, 500, 180, 0, 180]), [-46.18419, -6.77518, -
20.54925, 71.38674, 49.58727, -302.54752] )
    robot.MoveL(Pose([200, 250, 348.734575, 180, 0, -150]), [-41.62707, -
8.89064, -30.01809, 60.62329, 49.66749, -258.98418] )
    robot.MoveL(Pose([200, 200, 262.132034, 180, 0, -150]), [-43.73892, -
3.91728, -35.77935, 58.57566, 54.11615, -253.81122] )
    robot.RunMessage("Setting air valve 1 on")
    robot.RunCode("TCP_On", True)
    robot.Pause(1000)
    robot.MoveL(Pose([200, 250, 348.734575, 180, 0, -150]), [-41.62707, -
8.89064, -30.01809, 60.62329, 49.66749, -258.98418] )
    robot.MoveL(Pose([250, 300, 278.023897, 180, 0, -150]), [-37.52588, -
6.32628, -34.59693, 53.52525, 49.24426, -251.44677] )
    robot.MoveL(Pose([250, 250, 191.421356, 180, 0, -150]), [-39.75778, -
1.04537, -40.37883, 52.09118, 54.15317, -246.94403] )
    robot.RunMessage("Setting air valve off")
    robot.RunCode("TCP_Off", True)
    robot.Pause(1000)
    robot.MoveL(Pose([250, 300, 278.023897, 180, 0, -150]), [-37.52588, -
6.32628, -34.59693, 53.52525, 49.24426, -251.44677] )
    robot.MoveL(Pose([250, 200, 278.023897, 180, 0, -150]), [-41.85389, -
1.95619, -34.89154, 57.43912, 52.34162, -253.73403] )
    robot.MoveL(Pose([250, 150, 191.421356, 180, 0, -150]), [-
43.82111, 3.29703, -40.29493, 56.02402, 56.61169, -249.23532] )
    """
    robot.setAO(5,-2.1234568910111213141516)
    robot.setAO(5,0.0004)
    robot.setAO(5,4.0005)
    robot.setAO(5,4.0006)
    robot.setAO(5,'6.000')
    robot.setAO(5,'test')
    robot.setAO('test',1337)
    robot.ProgFinish("Program")


    # robot.ProgSave(".","Program",True)
    for line in robot.PROG:
```

```
        print(line)
    if len(robot.LOG) > 0:
        mbox('Program generation LOG:\n\n' + robot.LOG)

    input("Press Enter to close...")

if __name__ == "__main__":
    """Function to call when the module is executed by itself: test"""
    test_post()
```