Instructions on How to Use Python Interface for PI Robot Control:

version: 20161027
Edited by Jason

PART 1: Commands

Cartesian Space Motion Commands:

(1) **GO** (x_coord, y_coord, z_coord, u_coord, [v_coord, w_coord]): Point to point motion
Calling function: command_to_robot(shm_buf, cmd_str, param_list, ThreadOrNot=0, error=[2, 2, 1], wait=True)
cmd_str : “GO”
param_list: Target coordinate list, 4 or 6 float numbers.
ThreadOrNot: 0 or 1, determining whether to create a new thread when executing the motion command.
Error: error tolerance for each axis when determining whether it is in target position.

(2) **MOVE** (x_coord, y_coord, z_coord, u_coord, [v_coord, w_coord]): Straight line motion
Calling function: command_to_robot(shm_buf, cmd_str, param_list, ThreadOrNot=0, error=[2, 2, 1], wait=True)
cmd_str : “MOVE”
param_list: Target coordinate list, 4 or 6 float numbers.
ThreadOrNot: 0 or 1, determining whether to create a new thread when executing the motion command.
Error: error tolerance for each axis when determining whether it is in target position.

[3] JUMP (x_coord, y_coord, z_coord, u_coord, [v_coord, w_coord], Up_Limit): Lift-Move-Drop motion for Four-Axis.
Calling function: command_to_robot(shm_buf, cmd_str, param_list, ThreadOrNot=0, error=[2, 2, 1], wait=True)
cmd_str : “JUMP”
param_list: Target coordinate list, 4 or 6 float numbers plus additional Lifting-up limit.
ThreadOrNot: 0 or 1, determining whether to create a new thread when executing the motion command.
Error: error tolerance for each axis when determining whether it is in target position.

[4] JUMP3 (List[Pnt1_6D], List[Pnt2_6D], List[Pnt3_6D], DepartCP = 0, ApproachCP = 0, RotationMode): Jump motion for Six-Axis.
Calling function: command_to_robot(shm_buf, cmd_str, param_list, ThreadOrNot=0, error=[2, 2, 1], wait=True)
cmd_str : “JUMP3”
param_list: 3 coordinate list (float) , plus 3 integer parameters: 0 for now, 0 for now, wethertoRotate orientarion(0 or 1).
ThreadOrNot: 0 or 1, determining whether to create a new thread when executing the motion command.
Error: error tolerance for each axis when determining whether it is in target position.

[5] ARC (x_Mid, y_Mid, [z_Mid,] x_Tar, y_Tar, [z_Tar]): Arc motion specifying middle point and target point.
Calling function: command_to_robot(shm_buf, cmd_str, param_list, ThreadOrNot=0, error=[2, 2, 1], wait=True)

cmd_str : “ARC”
param_list: Middle and Target coordinate (NO ORIENTATION) list, 4 or 6 float numbers.
ThreadOrNot: 0 or 1, determining whether to create a new thread when executing the motion command.
Error: error tolerance for each axis when determining whether it is in target position.

Joint Space Motion Commands:

[6] GO_DEG (J1_deg, J2_deg, J3_deg, J4_deg, [J5_deg, J6_deg]) : Go to an ABSOLUTE joint coordinate, specified in degrees.
Calling function: command_to_robot(shm_buf, cmd_str, param_list)
cmd_str : “GO_DEG”
param_list: Target joint angle list (deg), 4 or 6 float numbers.
ThreadOrNot: 0 or 1, determining whether to create a new thread when executing the motion command.

[7] TGO (x_coord, y_coord, z_coord, u_coord, [v_coord, w_coord]) : Go an RELATIVE joint angles (each axis) from current pose, specified in degrees.
Calling function: command_to_robot(shm_buf, cmd_str, param_list)
cmd_str : “TGO”
param_list: Delta joint angle list (deg), 4 or 6 float numbers.
ThreadOrNot: 0 or 1, determining whether to create a new thread when executing the motion command.

Configurations:

[8] SPEED (Speed_percent, Depart_Speed_percent, Approach_Speed_percent): Max Speed Setting as percentage.
Calling function: command_to_robot(shm_buf, cmd_str, param_list)
cmd_str : “SPEED”
param_list: Speed percentage for Normal/ Depart(Jump) /Approach(Jump) motions, 3 integers.

[9] ACCEL (Accel_percent, Decel_percent, Depart_Accel, Depart_Decel, Approach_Accel, Approach_Decel): Max Acceleration and Deceleration Setting as percentage.
Calling function: command_to_robot(shm_buf, cmd_str, param_list)
cmd_str : “ACCEL”
param_list: Acceleration /Deceleration percentage for Normal/ Depart(Jump) /Approach(Jump) motions, 6 integers.

[10] ELBOW (above / below) : Configure the (6 Axis) solution choice to be Above/Below pose.
cmd_str: “ELBOW”
param_list: 0 for Above, 1 for Below

[11] HAND (Left/Right/Auto) : Configure the (4 Axis) Hand choice to be left/right solution. Or let the algorithm automatically choose the hand, however, this might cause **unexpected** movement during hand transition.
cmd_str: “HAND”
param_list: 0 for Left Hand, 1 for Right Hand, 0 for Auto choice

I/O Control:

[12] ON/OFF (port_Num, Hold_time, WaitOrNot) : Control the Output Port.

Calling function: command_to_robot(shm_buf, cmd_str, param_list)
cmd_str : “ON”/ “OFF”
param_list: out_port number start from 1: integer , Hold status time (seconds): float, WaitOrNot (for the hold time to continue): 0/1, integer.

PART 2: Inquiries

Inquiries WITH arguments:

[1] TARGET_OK: Return whether the input coordinate can be solved by IK.
Calling function: get_status(shm_buf, enq_str, param_list)
enq_str: “TARGET_OK”
param_list: Target coordinate to be verified, 4 or 6 axis.
Returning Result: 0 for OK, non-zero (uint of (-1)) for NOT OK.

[2] SW: Return the status of one input port.
Calling function: get_status(shm_buf, enq_str, param_list)
enq_str: “SW”
param_list: Target in_port number to be read, starting from 1.
Returning Result: 1 / 0 standing for On / Off.

[3] EXEC_TIME: Return the estimated time needed for finishing the “GO” command motion to the target Cartesian pose, this is just for reference, not exactly accurate.
Calling function: get_status(shm_buf, enq_str, param_list)
enq_str: “EXEC_TIME”
param_list: “Go” target coordinate to be estimated, 4 or 6 axis.
Returning Result: an integer representing the number of 2ms instructions, thus the execution time would be this integer multiply by 2ms.

Inquiries WITHOUT arguments:

[4] WHERE: Return the current Cartesian coordinate of robot.
Calling function: get_status(shm_buf, enq_str)
enq_str: “WHERE”
Returning Result: a float list of current coordinates.

[5] INPOS: Return whether the robot is currently in position (Not executing motions)
Calling function: get_status(shm_buf, enq_str)
enq_str: “INPOS”
Returning Result: 1 or 0 indicating whether 5480 had sent all the motion commands.

[6] WHERE_DEG: Return the current joint space coordinate of robot.
Calling function: get_status(shm_buf, enq_str)
enq_str: “WHERE_DEG”
Returning Result: a float list of current joint degree angles.

[7] ELBOW: Return the current setting of Elbow solution preference.
Calling function: get_status(shm_buf, enq_str)
enq_str: “ELBOW”
Returning Result: a one element int list, 0 for ABOVE, 1 for BELOW.

[8] ON/OFF: Report the on/off status of output port.
Calling function: get_status(shm_buf, enq_str)
enq_str: “ON”/ “OFF”
Returning Result: Unsigned integer (32-bit) indicating each port status through each bit.

[9] SPEED: Return the percentage settings of normal/Depart/Approach speed.
Calling function: get_status(shm_buf, enq_str)
enq_str: “SPEED”
Returning Result: list of 3 integers indicating speed settings in percentage. Order refer to the SPEED command.

[10] ACCEL: Return the percentage settings of normal/Depart/Approach Acceleration and Deceleration.
Calling function: get_status(shm_buf, enq_str)
enq_str: “ACCEL”
Returning Result: list of 6 integers indicating acceleration/Deceleration settings in percentage. Order refer to the ACCEL command.