AUBO-i5

Robot Python interface

Original Version 1.0

AUBO (Beijing) Robotics Technology Co., Ltd.

AUBO

Table of contents

1. Import the interface library and instantiate the robot control object

```
from robotcontrol import *
robot = Auboi5Robot()
```

2. Initialization (static method)

Name:	Auboi5Robot.initialize()
Function:	Dynamic library initialization
Parameters:	
Returns:	Success: 0
	Fail: other

3. Deinitialization (static method)

Name:	Auboi5Robot.uninitialize()
Function:	Dynamic library initialization
Parameters:	
Returns:	Success: 0
	Fail: other

4. Create the control context handle

Name:	create_context()
Function:	Create the control context handle
Parameters:	
Returns:	Success: context handle RSHD >= 0
	Fail: -1

5. Remove the control context handle

Name:	destory_context(RSHD)
Function:	Remove the control context handle
Parameters:	**RSHD** context handle
Returns:	Success: 0
	Fail: other

6. Robot system login

Name:	login(IP, PORT)
Function:	Login robot system
Parameters:	**IP** robot IP address **PORT** robot port number
Returns:	Success: 0
	Fail: other

7. Robot system logout

Name:	logout()
Function:	Logout robot system
Parameters:	**RSHD** context handle
Returns:	Success: 0
	Fail: other

8. Initialize the global properties of the robot control

Name:	init_profile()
Function:	Initialize the global properties of the robot control
Parameters:	
Returns:	Success: 0
	Fail: other

9. Set the maximum acceleration of each joint

Name:	set_joint_maxacc(joint_maxacc=(1.0,1.0,1.0,1.0,1.0,1.0))
Function:	Set the maximum acceleration for robot
Parameters:	**Acc** is the joint acceleration, unit: rad/s
Returns:	Success: 0
	Fail: other

10. Get the maximum joint acceleration of the robot

Name:	get_joint_maxacc()
Function:	Get the maximum joint acceleration of the robot
Parameters:	**Acc** is the joint acceleration of six joints, unit: rad/s
Returns:	Success: the maximum acceleration of six joints, unit: rad/s
	Fail: None

11. Set the maximum velocity of each joint

Name:	set_joint_maxvelc(joint_maxvelc=(1.0,1.0,1.0,1.0,1.0,1.0))
Function:	Set the maximum velocity for robot
Parameters:	**Acc** is the joint velocity of six joints, unit: rad/s
Returns:	Success: 0
	Fail: other

12. Get the maximum velocity of each joint

Name:	get_joint_maxvelc()
Function:	Get the maximum velocity for robot
Parameters:	**velc** is the joint velocity of six joints, unit: rad/s
Returns:	Success: the maximum velourity of six joints, unit: rad/s
	Fail: None

13. Set the maximum linear acceleration of the robot end

Name:	set_end_max_line_acc(end_maxacc=0.1)
Function:	Set the acceleration for robot end
Parameters:	**end_maxacc** is the end side acceleration, unit: meter per sec
Returns:	Success: 0
	Fail: other

14. Get the maximum linear acceleration of the robot end

Name:	get_end_max_line_acc()
Function:	Get the acceleration for robot end
Parameters:	
Returns:	Success: the maximum acceleration of the robot end, unit(m/s) Success: 0
	Fail: None

15. Set the maximum linear velocity of the robot end

Name:	set_end_max_line_velc(end_maxvelc=0.1)
Function:	Set the maximum linear velocity of the robot end
Parameters:	**end_maxacc:** the maximum velocity for end side，unit(m/s)
Returns:	Success: 0
	Fail: other

16. Get the maximum linear velocity of the robot end

Name:	get_end_max_line_velc()
Function:	Get the maximum linear velocity of the robot end
Parameters:	
Returns:	Success: the maximum velocity of the robot end，unit(m/s)
	Fail: None

17. Set the maximum angular acceleration of the robot end

Name:	set_end_max_angle_acc(end_maxacc=0.1)
Function:	Set the acceleration for robot end
Parameters:	**end_maxacc:** the maximum end side acceleration，unit: rad/s
Returns:	Success: 0
	Fail: other

18. Get the maximum angular acceleration of the robot end

Name:	get_end_max_angle_acc()
Function:	Get the maximum end side acceleration of the robot
Parameters:	
Returns:	Success: the maximum of the robot end， unit: rad/s
	Fail: None

19. Set the maximum angular velocity of the robot end

Name:	set_end_max_angle_velc(end_maxvelc=0.1)
Function:	Set the velocity for robot end
Parameters:	**end_maxacc:** the maximum end side velocity, unit: rad/s
Returns:	Success: 0
	Fail: other

20. Get the maximum angular velocity of the robot end

Name:	get_end_max_angle_velc()
Function:	Get the maximum end side velocity of the robot
Parameters:	
Returns:	Success: the maximum velocity of the robot end, unit: rad/s
	Fail: None

21. The joint movement of the robot

Name:	move_joint(joint_radian=(0.000000,0.000000,0.000000,0.000000,0.000000,0.000000))
Function:	The joint movement of the robot
Parameters:	**Joint_radian:** the joint angle of six joints, unit: radian
Returns:	Result:
	Success: RobotErrorType.RobotError_SUCC

22. The Linear movement of the robot

Name:	move_line(joint_radian(1.0,1.0,1.0,1.0,1.0,1.0))
Function:	The robot linearly moves to target position form the current position
Parameters:	**RSHD** context handle
Returns:	Joint_radian: the joint angle of six joints, unit: radian
	Success: 0

23. Remove all the set global waypoints

Name:	remove_all_waypoint()
Function:	Remove all the set global waypoints
Parameters:	
Returns:	Joint_radian: the joint angle of six joints, unit: radian

24. Add the global waypoint for track movement

Name:	add_waypoint(joint_radian(1.0,1.0,1.0,1.0,1.0,1.0))
Function:	Add the global waypoint for track movement
Parameters:	**Joint_radian**: the joint angle of six joints, unit: radian
Returns:	Success: 0
	Fail: other

25. The track movement of the robot

Name:	move_track(track_type)
Function:	After adding the waypoint by using add_waypoint, proceed the track movement
Parameters:	**track_type**: type of the track movement Arc movement: RobotMoveTrackType.ARC_CIR Track movement: RobotMoveTrackType.CARTESIAN_MOVEP
Track movement type:	2:ARC_CIR At least 3 points, circle or arc, use **circular** number to separate, 0 is arc, greater than 0 is circle. 3:CARTESIAN_MOVEP At least 3 points, must to set blend radius, The blend radius must be less than 1/2 of the length of the shortest line segment, and the speed should not be high **The following methods are not available:** The following four kinds of three-order spline interpolation curves have the beginning and end point acceleration discontinuities, not available for the new joint drive version 4:CARTESIAN_CUBICSPLINE Cubic spline interpolation (over control point), automatically optimize the track running time. Changing attitude is currently not available. 5:CARTESIAN_UBSPLINEINTP It is necessary to set the time interval of cubic uniform b-spline interpolation (over control points). Changing attitude is currently not available. 6:JIONT_CUBICSPLINE 7:JOINT_UBSPLINEINTP Used for track playback
Returns:	Success: 0
	Fail: other

26. Set the blend radius

Name:	set_blend_radius(blend_radius=0.01)
Function:	Set the blend radius, unit: meter
Parameters:	**blend_radius**: blend radius, unit(m)
Returns:	Result:
	Success: 0
	Fail: other

27. Set the times of the circular movement

Name:	set_circular_loop_times(circular_count=1)
Function:	Set the times of the circular movement
Parameters:	When **circular_count** is greater than 0, the robot is proceeding circular movement **circular_count** times. When **circular_count** is equal to 0, the robot is proceeding arc track movement.
Returns:	Result:
	Success: 0
	Fail: other

28. Set the user coordinate system

Name:	set_user_coord(user_coord)
Function:	Set the user coordinate system
Parameters:	RSHD context handle **user_coord** user coordinate system user_coord = {'coord_type': 2, coordinate system type: must be 2, assign the user coordinate system 'calibrate_method': 0, coordinate system calibration method 'calibrate_points': coordinate system calibration point {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': tool description {"pos": (0.0, 0.0, 0.0), the tool position corresponding to the terminal coordinate system "ori": (1.0, 0.0, 0.0, 0.0)} the tool attitude corresponding to the terminal coordinate system }
Returns:	Result:
	Success: 0
	Fail: other

29. Set the base coordinate system

Name:	set_base_coord()
Function:	Set the base coordinate system
Parameters:	
Returns:	Result:
	Success: 0
	Fail: other

30. Check whether the user coordinate is reasonable

Name:	check_user_coord(user_coord)
Function:	Set the user coordinate system
Parameters:	**user_coord** user coordinate system user_coord = {'coord_type': 2, coordinate system type: must be 2， assign the user coordinate system 'calibrate_method': 0, coordinate system calibration method 'calibrate_points': coordinate system calibration point {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': tool description {"pos": (0.0, 0.0, 0.0), the tool position corresponding to the terminal coordinate system "ori": (1.0, 0.0, 0.0, 0.0)} the tool attitude corresponding to the terminal coordinate system }
Returns:	Result:
	Success: 0
	Fail: other

31. Set the relative offset on base coordinate system

Name:	set_relative_offset_on_base(relative(x,y,z))
Function:	Set the relative offset on base coordinate system
Parameters:	**Relative x,y,z** relative shift, unit: meter
Returns:	Result:
	Success: 0
	Fail: other

32. Set the relative offset on user coordinate system

Name:	set_relative_offset_on_user(relative(x,y,z),user_coord)
Function:	Set the relative offset on user coordinate system
Parameters:	**RSHD:** context handle **Relative x,y,z:** relative shift, unit: meter **user_coord:** user coordinate system user_coord = {'coord_type': 2, coordinate system type: must be 2, assign the user coordinate system 'calibrate_method': 0, coordinate system calibration method 'calibrate_points': coordinate system calibration point {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': tool description {"pos": (0.0, 0.0, 0.0), the tool position corresponding to the terminal coordinate system "ori": (1.0, 0.0, 0.0, 0.0)} the tool attitude corresponding to the terminal coordinate system }
Returns:	Result:
	Success: 0
	Fail: other

33. Forward kinematics

Name:	forward_kin(joint_radian(1.0,1.0,1.0,1.0,1.0,1.0))
Function:	Forward kinematics
Parameters:	**Joint_radian**: the joint angle of six joints unit: radian
Returns:	Forward kinematics result: waypoint {'joint': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0], 'pos':[-0.06403157614989634,-0.4185973810159096, 0.816883228463401], 'ori': [-0.11863209307193756,0.3820514380931854,0.0,0.9164950251579285]}

34. Inverse kinematics

Name:	inverse_kin(joint_radian(1.0, 1.0, 1.0, 1.0, 1.0, 1.0), pos(x,y,z), ori(w,x,y,z))
Function:	Inverse kinematics
Parameters:	**joint_radian** The joint angles of the six joints at the starting point, unit(rad) **Pos** position information **Ori** attitude information
Returns:	Result:
	Success: 0
	Fail: other
Forward kinematics result:	waypoint {'joint':[1.0,1.0,1.0,1.0,1.0,1.0], 'pos': [-0.06403157614989634,-0.4185973810159096,0.816883228463401], 'ori': [-0.11863209307193756,0.3820514380931854,0.0,0.9164950251579285]}

35. Convert the base coordinate system to user coordinate system

Name:	base_to_user(pos(x,y,z), ori(w,x,y,z), user_coord, tool(x,y,z))
Function:	Convert the base coordinate system to user coordinate system
Parameters:	**Pos (x,y,z):** the position information on base coordinate system **Ori (w,x,y,z):** the attitude information on base coordinate system **user_coord:** user coordinate system user_coord = {'coord_type': 2, coordinate system type: must be 2, assign the user coordinate system 'calibrate_method': 0, coordinate system calibration method 'calibrate_points': coordinate system calibration point {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': tool description {"pos": (0.0, 0.0, 0.0), the tool position corresponding to the terminal coordinate system "ori": (1.0, 0.0, 0.0, 0.0)} the tool attitude corresponding to the terminal coordinate system } **Tool (pos(x,y,z), ori(w,x,y,z)):** the position and attitude of the user tool
Returns:	the position and attitude of the user coordinate system: {"pos":(1,2,3), "ori":(1,2,3,4)}

36. Convert the user coordinate system to base coordinate system

Name:	user_to_base(pos(x,y,z), ori(w,x,y,z), user_coord, tool(x,y,z))
Function:	Convert the user coordinate system to base coordinate system
Parameters:	**Pos (x,y,z):** the position information on base coordinate system **Ori (w,x,y,z):** the attitude information on base coordinate system **user_coord:** user coordinate system user_coord = {'coord_type': 2, coordinate system type: must be 2, assign the user coordinate system 'calibrate_method': 0, coordinate system calibration method 'calibrate_points': coordinate system calibration point {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': tool description {"pos": (0.0, 0.0, 0.0), the tool position corresponding to the terminal coordinate system "ori": (1.0, 0.0, 0.0, 0.0)} the tool attitude corresponding to the terminal coordinate system } **Tool (pos(x,y,z), ori(w,x,y,z)):** the position and attitude of the user tool
Returns:	the position and attitude of the base coordinate system:
	{"pos":(1,2,3),
	"ori":(1,2,3,4)}

37. Robot startup

Name:	robot_startup(collision, tool_dynamics)
Function:	Robot startup
Parameters:	**Collision** collision class(0~10) **tool_dynamics** dynamics parameter {"position": (x, y, z), tool's centre of gravity "payload": 1.5 load, (0~5)kilograms "inertia": (xx,xy,xz,yy,yz,zz) inertia}
Returns:	Result:
	Success: 0
	Fail: other
Startup state of robot:	state 0 ROBOT_SERVICE_READY ready 1 ROBOT_SERVICE_STARTING starting 2 ROBOT_SERVICE_WORKING working 3 ROBOT_SERVICE_CLOSING closing 4 ROBOT_SERVICE_CLOSED closed 5 ROBOT_SETVICE_FAULT_POWER power fault 6 ROBOT_SETVICE_FAULT_BRAKE, brake fault 7 ROBOT_SETVICE_FAULT_NO_ROBOT no robot

38. Shut down the robot

Name:	robot_shutdown()
Function:	Shut down the robot
Parameters:	
Returns:	Result:
	Success: 0
	Fail: other

39. Stop the robot movement

Name:	move_stop()
Function:	Stop the movement of robot
Parameters:	
Returns:	Result:
	Success: 0
	Fail: other

40. Suspend the robot movement

Name:	move_pause()
Function:	Suspend the movement of robot
Parameters:	
Returns:	Result:
	Success: 0
	Fail: other

41. Continue the movement of robot

Name:	move_continue()
Function:	Continue the movement of robot
Parameters:	
Returns:	Result:
	Success: 0
	Fail: other

42. Collision recovery

Name:	collision_recover()
Function:	The recovery after collision
Parameters:	
Returns:	Result:
	Success: 0
	Fail: other

43. Get the service state of the robot

Name:	get_robot_state()
Function:	Get the service state of the robot
Parameters:	
Returns:	Result:
	Success: 0
	Fail: other
Startup the state of robot:	state 0 ROBOT_SERVICE_READY ready 1 ROBOT_SERVICE_STARTING starting 2 ROBOT_SERVICE_WORKING working 3 ROBOT_SERVICE_CLOSING closing 4 ROBOT_SERVICE_CLOSED closed 5 ROBOT_SETVICE_FAULT_POWER power fault 6 ROBOT_SETVICE_FAULT_BRAKE, brake fault 7 ROBOT_SETVICE_FAULT_NO_ROBOT no robot

44. Set the working mode of the robot

Name:	set_work_mode(mode)
Function:	Set the working mode of the robot
Parameters:	**Mode** 0 simulation mode 1 real robot
Returns:	Result:
	Success: 0
	Fail: other

45. Get the current working mode of the robot

Name:	get_work_mode()
Function:	Get the current working mode of the robot
Parameters:	
Returns:	Mode 0 simulation mode 1 real robot

46. Set the collision class of the robot

Name:	set_collision_class(collision)
Function:	Set the collision class of the robot
Parameters:	**Collision** collision class(0~10)
Returns:	Result:
	Success: 0

47. Determine whether the real robot has been linked

Name:	is_have_real_robot()
Function:	Determine whether the real robot has been linked
Parameters:	
Returns:	The real robot exists or not
	Not-exist: 0
	Exist :1

48. Get the joint state of the real robot

Name:	get_joint_status()
Function:	Get the joint state of the real robot
Parameters:	
Returns:	Joint state {‘joint1’: {‘current':current(mA),'voltage':voltage(Volts),'temperature':temperature(Celsius)}, 'joint2':{'current':0,'voltage':0.0,'temperature':0}, 'joint3':{'current':0,'voltage':0.0,'temperature':0}, 'joint4':{'current':0,'voltage':0.0,'temperature':0}, 'joint5':{'current':0,'voltage':0.0,'temperature':0}, 'joint6':{'current':0,'voltage':0.0,'temperature':0}}

49. Get the I/O configuration of the robot interface board

Name:	get_board_io_config(iotype)
Function:	Get the I/O configuration of the robot interface board
Parameters:	RSHD context handle **Iotype:** IO type: 0:ControllerDI, 1:ControllerDO, 2:ControllerAI, 3:ControllerAO, 4:UserDI, 5:UserDO, 6:UserAI, 7:UserAO, 8:ToolPower, 9:ToolDI, 10:ToolDO, 11:ToolAI, 12:ToolAO,
Returns:	IO configuration: ({"id": ID "name": “IO name" "addr": IO address "type": IO type "valu": IO current value}, {"id": ID "name": “IO name" "addr": IO address "type": IO type "valu": IO current value}, ...}

50. Set the I/O state of the robot interface board

Name:	set_board_io_status(Iotype, name, value)
Function:	Set the I/O state of the robot interface board
Parameters:	**IoType:** 0:ControllerDI, 1:ControllerDO, 2:ControllerAI, 3:ControllerAO, 4:UserDI, 5:UserDO, 6:UserAI, 7:UserAO, 8:ToolPower, 9:ToolDI, 10:ToolDO, 11:ToolAI, 12:ToolAO **Name:** IO name **Value:** IO state
Returns:	Result:
	Success: 0

51. Get the I/O state of the robot interface board

Name:	get_board_io_status(Iotype, name)
Function:	Get the I/O state of the robot interface board
Parameters:	**IoType:** 0:ControllerDI, 1:ControllerDO, 2:ControllerAI, 3:ControllerAO, 4:UserDI, 5:UserDO, 6:UserAI, 7:UserAO, 8:ToolPower, 9:ToolDI, 10:ToolDO, 11:ToolAI, 12:ToolAO **Name:** IO name
Returns:	Value:IO state

52. Get the tool power state of the robot

Name:	get_tool_power_type()
Function:	Get the power state of the robot on tool
Parameters:	
Returns:	Power_type: OUT_0V = 0, OUT_12V = 1, OUT_24V = 2

53. Set the tool power state of the robot

Name:	set_tool_power_type(power_type)
Function:	Set the tool power state of the robot
Parameters:	**Power_type:** OUT_0V = 0, OUT_12V = 1, OUT_24V = 2
Returns:	Result: Success: 0

54. Set the tool I/O type of the robot

Name:	set_tool_io_type(addr, iotype)
Function:	Set the tool I/O type of the robot
Parameters:	**Addr:** I/O address at the tool TOOL_DIGITAL_IO_0 = 0, TOOL_DIGITAL_IO_1 = 1, TOOL_DIGITAL_IO_2 = 2, TOOL_DIGITAL_IO_3 = 3 **IoType:** IO type 9:ToolDI, 10:ToolDO, 11:ToolAI, 12:ToolAO
Returns:	Result: Success: 0

55. Get the tool power value of the robot

Name:	get_tool_power_voltage()
Function:	Get the tool power value of the robot
Parameters:	
Returns:	Voltage value: voltage

56. Set the tool I/O state of the robot

Name:	set_tool_io_status(name, status)
Function:	Set the tool I/O state of the robot
Parameters:	Addr: I/O address at tool TOOL_DIGITAL_IO_0 = 0, TOOL_DIGITAL_IO_1 = 1, TOOL_DIGITAL_IO_2 = 2, TOOL_DIGITAL_IO_3 = 3 **IoType:** IO type 9:ToolDI, 10:ToolDO, 11:ToolAI, 12:ToolAO status:IO data (0 or 1)
Returns:	Result: Success: 0

57. Set the tool I/O state of the robot

Name:	get_tool_io_status(name)
Function:	Get the tool I/O state of the robot
Parameters:	Addr: I/O address at tool TOOL_DIGITAL_IO_0=0, TOOL_DIGITAL_IO_1=1, TOOL_DIGITAL_IO_2=2, TOOL_DIGITAL_IO_3=3 **IoType:** IO type: 9:ToolDI, 10:ToolDO, 11:ToolAI, 12:ToolAO
Returns:	status:IO data (0 or 1)

58. The robot event callback function

Name:	setcallback_robot_event(robot_event_callback)
Function:	Set the callback function of the robot event
Parameters:	**robot_event_callback** callback function name under the Python
Returns:	Result: Success: 0

Python callback function parameters	{'code': event ID, 'type': event type}
	Event types as shown below:
	typedef enum{
	RobotEvent_armCanbusError, //robot CAN bus error
	RobotEvent_remoteHalt, //robot stop
	RobotEvent_remoteEmergencyStop, //robot remote emergency stop
	RobotEvent_jointError, //joint error
	RobotEvent_forceControl, //force control
	RobotEvent_exitForceControl, //quit the force control
	RobotEvent_softEmergency, //soft scram
	RobotEvent_exitSoftEmergency, //quit the soft emergency stop
	RobotEvent_collision, //collision
	RobotEvent_collisionStatusChanged, //collision status changed
	RobotEvent_tcpParametersSucc, //tool dynamic parameters set successfully
	RobotEvent_powerChanged, //the robot power switch state changed
	RobotEvent_ArmPowerOff, //robot power off
	RobotEvent_mountingPoseChanged, //the installation location has changed
	RobotEvent_encoderError, //encoder error
	RobotEvent_encoderLinesError, //the number of encoders are not equivalent
	RobotEvent_singularityOverspeed, //singularity overspeed
	RobotEvent_currentAlarm, //abnormal robot current
	RobotEvent_toolioError, //robot tool error
	RobotEvent_robotStartupPhase, //robot startup stage
	RobotEvent_robotStartupDoneResult, //the result of completing startup
	RobotEvent_robotShutdownDone, //the result of completing shutdown
	RobotEvent_atTrackTargetPos, //the signal notification of robot moving to the target position (track movement)
	RobotSetPowerOnDone, //Setting power state has completed
	RobotReleaseBrakeDone, //releasing the robot brake has completed
	RobotEvent_robotControllerStateChaned, //the control state of the robot has changed
	RobotEvent_robotControllerError, //robot control error----It usually comes back when there are problems with algorithmic planning
	RobotEvent_socketDisconnected, //socket disconnected
	RobotEvent_overSpeed, //overspeed
	RobotEvent_algorithmException, //robot arithmetic exceptions
	//interface board io event
	RobotEvent_boardIoPoweron, //external power-on signal
	RobotEvent_boardIoRunmode, //linkage/manual
	RobotEvent_boardIoPause, //external pause signal
	RobotEvent_boardIoStop, //external stop signal
	RobotEvent_boardIoHalt, //external halt signal
	RobotEvent_boardIoEmergency, //external emergency stop signal
	RobotEvent_boardIoRelease_alarm, //external all-clear signal
	RobotEvent_boardIoOrigin_pose, //external back-to-original position signal
	RobotEvent_boardIoAutorun, //external autorun signal
	//interface board safety io event
	RobotEvent_safetyIoExternalEmergencyStope, //external emergency stop input 01

	RobotEvent_safetyIoExternalSafeguardStope, //external safeguard stop input 02
	RobotEvent_safetyIoReduced_mode, //reduced mode input
	RobotEvent_safetyIoSafeguard_reset, //reset safeguard
	RobotEvent_safetyIo3PositionSwitch, //three position switch 1
	RobotEvent_safetyIoOperationalMode, //operation mode
	RobotEvent_safetyIoManualEmergencyStop, //demonstrator emergency stop 01
	RobotEvent_safetyIoSystemStop, //system stop input
	//robot working event
	RobotEvent_alreadySuspended, //robot suspended
	RobotEvent_alreadyStopped, //robot stopped
	RobotEvent_alreadyRunning, //robot running
	RobotEvent_exceptEvent = 100,
	//unknown event
	robot_event_unknown,
	//user event
	RobotEvent_User = 1000, // first user event id
	RobotEvent_MaxUser = 65535 // last user event id
	}RobotEventType;

Example	**import** libpyauboi5
	# robot event
	def robot_event(event):
	print(event)
	# set the robot event callback function
	libpyauboi5.setcallback_robot_event(robot_event)

