OzoPython Reference Guide 4.2 Beta

(Note: Some features are not yet implemented.)

bobcook47@hotmail.com

Sample Programs

https://editor.ozobot.com/python?programId=**xxxxxxx** which is the share code

Movement

Description	Methods	Share Code
Simple “hello world”	print	sczii45
Simple Multi-Bot	print	kpfcbim
Move over a rectangle path specified by length x width in cm	amove arotate	o84ry3d
Move over a circle’s circumference path with radius in cm and an elapsed time of seconds	aset_velocity	6mwzhnp

Light Effects

Description	Methods	Share Code
Draw light patterns, tests every light possibility	aset_light_color aset_light_color_rgb sleep	d3rra8t
Draw Blockly light patterns: police car lights to fireworks. Also, a method to set all the front lights to different colors in one call.	aset_light_color aset_light_color_rgb sleep	ec5us8h

Sounds

Description	Methods	Share Code
Speaks or sounds with each possible method.	asay_number asay_direction asay_color emotions.aplay_happy aplay_tone aplay_note aplay_midi	nxn6nrd
Speaks any integer or real number but a high-frequency tone is used for .(dot) in fractions and a low-frequency tone is used for -(minus) in negative fractions less than zero.	asay_number aplay_note math.floor	ui9fmo4
Play the notes/rests of any song described by a list of 2-tuples that specify a (note, duration). Multiple octaves are supported and repeated phrases can be reused without duplicating those note sequences.	aplay_note sleep	dxkpy3u

Navigation

Description	Methods	Share Code
Navigate a black line to a colored intersection, tests every method	dm.aget_value aset_line_following_speed anavigate aget_last_intersection sensors.aline_color	9p29kha
Perform a random walk on any collection of connected lines with standard intersections	aset_line_following_speed anavigate movement.aset_velocity random.randint	gr6rdhb

Perform a random walk but when reaching the end of a line jump across the empty space to find a connection. Display Top red when jumping. Download Map	aset_line_following_speed aset_light_color_rgb random.random anavigate sleep aset_velocity aplay_happy	bkqhoj5

Sensors

Description	Methods	Share Code
Find the Color. Searches inside a red oval for a black square. Download Map	aset_light_color_rgb amove asurface_color aset_light_color sleep	mmibfhb
Avoid Red. Random colored line traversal inside a grid of lines. The Bot reverses path when it encounters a red intersection. Download Map	aset_line_following_speed random.random anavigate aset_velocity asay_number asay_color aline_color	pug4tsb
Skip the Lines. Use fixed moves to navigate to the end of a grid then jump up to a horizontal line with a blue tip. When found, the lights flash blue.	aset_light_color_rgb anavigate amove arotate aline_color aplay_laugh sleep	hj6pdxo

Function Reference

Movement dm.movement

Asynchronous

function amove(distance: float, speed: float) #distance m, speed m/s, should be followed by delay
function arotate(angle: float, speed: float) # distance radians, speed radians/s, should be followed by delay
function aset_velocity(forward_speed: float, angular_speed: float, duration_s: float)
#forward speed m/s, angular speed rad/s, duration s
function aset_wheel_speeds(left_mmps: float, right_mmps: float)
#mm/s 20-90 -90 to -20, followed by wait or other delay
function astop_wheels() #by stopping motors

Synchronous

function move(distance: float, speed: float) #distance m, speed m/s
function move_mm(distance_mm: float, speed_mmps: float) #distance mm, speed mm/s
function rotate(angle: float, speed: float) # distance radians, speed radians/s
function rotate_deg(angle_degrees: float, speed_degreesps: float)
function set_velocity(forward_speed: float, angular_speed: float, duration_s: float)
#forward speed m/s, angular speed rad/s, duration s
function set_wheel_speeds(left_mmps: float, right_mmps: float)
#mm/s 20-90 -90 to -20, follow by wait or other delay
function stop_wheels() #by stopping motors

ozobot

Directions

LEFT RIGHT FORWARD BACK ANY ERROR

Lights

ALL_FRONT ALL_ROBOT FRONT_1 FRONT_2 FRONT_3 FRONT_4 FRONT_5 BACK_RED LEFT_CYAN TOP

LineColor

BLACK BLUE GREEN RED UNKNOWN

Note

A A_flat A_sharp B B_flat C C_sharp D D_flat D_sharp E E_flat F F_sharp G G_sharp G_flat

SurfaceColor

BLACK BLUE GREEN RED WHITE UNKNOWN

ProximitySensorLocation

LEFT_FRONT RIGHT_FRONT LEFT_REAR RIGHT_REAR

function get_robot(uuid: typing.Optional[str]=None, device_name: typing.Optional[str]=None, *,
coro: typing.Optional[bool]=False) -> typing.Union[Bot, BotAsync]
dm = ozobot.get_robot(device_name = "Ari_1", coro=True)
function radians(angle:) -> float
function time.sleep(delay_s: float)

User IO dm.user_io

Asynchronous

function aalert(message: str, cancelable: bool)
function aprint(message: str)
function aprompt(message: str, type: InputTypeLiteral, options: list[InputType], cancelable: bool) -> InputType

Synchronous

function alert(message: str, cancelable: bool)
function print(message: str)
function prompt(message: str, type: InputTypeLiteral, options: list[InputType], cancelable: bool) -> InputType

Line Navigation dm.navigation

Asynchronous

function aset_line_following_speed(0.025) #meters/s
test = (await aget_value('lineNavigationSpeed')) #meters/s from previous aset
function anavigate(ozobot.Directions.FORWARD, follow = False) #pick direction
function anavigate(ozobot.Directions.FORWARD, follow = True) #goto next intersection or line end
if (await dm.navigation.aget_last_intersection())['intersection']['Left']: #if way Left/Right/Straight/End {'intersection':
{'type': 'intersection', 'Backward': True, 'Straight': False, 'Left': True, 'Right': True},
'timestamp': 1749925445083}
function alast_color_code() -> typing.Tuple[LineColor] #last color code sequence

function alast_intersection() -> typing.Tuple[Directions] #all possible directions including BACK
function aposition() -> typing.Tuple[float] #returns x y position in meters/s from start of program
function await_for_next_intersection()

Synchronous

function move(distance: float, speed: float) #distance m, speed m/s
function set_line_following_speed(0.025) #meters/s
test = (get_value('lineNavigationSpeed')) #meters/s from previous set
function navigate(ozobot.Directions.FORWARD, follow = False) #pick direction
function navigate(ozobot.Directions.FORWARD, follow = True) #goto next intersection or line end
if (dm.navigation.get_last_intersection())['intersection']['Left']: #if way Left/Right/Straight/End
  {'intersection': {'type': 'intersection', 'Backward': True, 'Straight': False, 'Left': True, 'Right': True},
    'timestamp': 1749925445083}
function last_color_code() -> typing.Tuple[LineColor] #last color code sequence
function last_intersection() -> typing.Tuple[Directions] #all possible directions including BACK
function position() -> typing.Tuple[float] #returns x y position in meters/s from start of program
function await_for_next_intersection()

Light Effects dm.light_effects

Asynchronous

function aset_light_color(ozobot.SurfaceColor.BLACK, ozobot.Lights.TOP) #top off
function aset_light_color_rgb(127 / 127, 0 / 127, 0 / 127, ozobot.Lights.TOP) #top to red
  aset_light_color_rgb(random.randint(1, 127) / 127, random.randint(1, 127) / 127, random.randint(1, 127) / 127,
      ozobot.Lights.ALL_ROBOT) #all to a random color

Synchronous

function set_light_color(ozobot.SurfaceColor.BLACK, ozobot.Lights.TOP) #top off
function set_light_color_rgb(127 / 127, 0 / 127, 0 / 127, ozobot.Lights.TOP) #top to red
  set_light_color_rgb(random.randint(1, 127) / 127, random.randint(1, 127) / 127, random.randint(1, 127) / 127,
      ozobot.Lights.ALL_ROBOT) #all to a random color

Sounds dm.sounds

Emotions dm.sounds.emotions

Asynchronous

function aplay_happy()
function aplay_laugh()
function aplay_sad()
function aplay_surprised()

Synchronous

function play_happy()
function play_laugh()
function play_sad()
function play_surprised()

Asynchronous

function asay_direction(ozobot.Directions.LEFT)
function asay_color(ozobot.SurfaceColor.RED)
function asay_number(1) #restricted to -199 to +199
function aplay_note(6, ozobot.Note.C, 0.5) #octave 1-10, note, duration s, octave<3 or >7 weird
function aplay_midi_note(midi_note_number: int, duration_s: float) #not implemented
MIDI frequency chart by octave

Synchronous

function say_direction(ozobot.Directions.LEFT)
function say_color(ozobot.SurfaceColor.RED)
function say_number(1) #restricted to -199 to +199
function play_note(6, ozobot.Note.C, 0.5) #octave 1-10, note, duration s, octave<3 or >7 weird

Sensors dm.sensors

Asynchronous

function proximity(location: ProximitySensorLocation) -> float
if ((await dm.sensors.aproximity(ozobot.ProximitySensorLocation.LEFT_FRONT) >= 40) or (await dm.sensors.aproximity(ozobot.ProximitySensorLocation.RIGHT_FRONT) >= 40)) and ((await dm.sensors.aproximity(ozobot.ProximitySensorLocation.LEFT_REAR) >= 40) or (await dm.sensors.aproximity(ozobot.ProximitySensorLocation.RIGHT_REAR) >= 40)): #object in front and rear
test = (await dm.await_next_event({ 'method': 'TimeOfFlight', 'params': {}}))[-1]['distance'] #in meters
[{'distance': 0.253, 'deviation': 0.001, 'ambientRate': 152, 'signalRate': 9872, 'activeCount': 145, 'timestamp': 1749926017647}]
function acolor_sensor()
function aline_color() -> LineColor
function asurface_color() -> SurfaceColor
await dm.sensors.aline_sensors_raw() # [true/false,...] one entry per

Synchronous

function proximity(location: ProximitySensorLocation) -> float
if ((dm.sensors.proximity(ozobot.ProximitySensorLocation.LEFT_FRONT) >= 40) or (dm.sensors.proximity(ozobot.ProximitySensorLocation.RIGHT_FRONT) >= 40))): #object in front
test = (dm.wait_next_event({ 'method': 'TimeOfFlight', 'params': {}}))[-1]['distance'] #in meters
[{'distance': 0.253, 'deviation': 0.001, 'ambientRate': 152, 'signalRate': 9872, 'activeCount': 145, 'timestamp': 1749926017647}]
function color_sensor()
function surface_color() -> SurfaceColor

Asyncio

Asynchronous

function run(main: Awaitable[_T], *, debug: Optional[bool]=...) -> _T
function sleep(delay_s: float, result: _T=..., *, loop: Optional[AbstractEventLoop]=...) -> Future[_T]