QCar2 HAL + PAL

Competition Reference

Curated reference for ACC QCar2 self-driving development, generated from selected HAL/PAL modules.

Modules documented: 15

Functions/methods documented: 247

Classes documented: 46

Each item includes its callable signature and the first paragraph of its docstring. Scope excludes non-QCar products and scaffold-heavy training sections.

Competition Scope

Focused on modules directly relevant to QCar2 autonomy in the ACC competition (vehicle I/O, sensors, state estimation, control, lane/perception processing, path planning, and telemetry tooling).

- hal/content/qcar_functions.py: QCar-specific high-level estimators/controllers.
- hal/products/mats.py: SDCS competition map graph and path generation.
- hal/utilities/control.py: PID and Stanley control primitives.
- hal/utilities/estimation.py: EKF/Kalman filtering base classes.
- hal/utilities/image_processing.py: Image processing routines used in lane/perception pipelines.
- hal/utilities/path_planning.py: Road-map/path-planning internals used by SDCSRoadMap.
- pal/products/qcar.py: Primary QCar2 hardware/sensor interfaces.
- pal/products/qcar_config.py: QCar/QCar2 configuration bootstrap logic.
- pal/utilities/gamepad.py: Gamepad interface for manual control and safety takeover.
- pal/utilities/keyboard.py: Keyboard control helpers used in virtual/manual workflows.
- pal/utilities/lidar.py: Generic lidar device interface used by QCarLidar.
- pal/utilities/math.py: Math/filter helpers used in control and perception loops.
- pal/utilities/probe.py: Probe/Observer telemetry and visualization tools.
- pal/utilities/stream.py: Streaming primitives used for GPS/lidar/observer networking.
- pal/utilities/vision.py: Camera2D/3D streams used by CSI and RealSense.

HAL Library

hal/content/qcar_functions.py

This module contains QCar specific implementations of hal features

Classes

QCarEKF

An EKF designed to estimate the 2D position and orientation of a QCar.

`__init__(x_0, Q_kf=np.diagflat([0.0001, 0.001]), R_kf=np.diagflat([0.001]), Q_ekf=np.diagflat([0.01, 0.01, 0.01]), R_ekf=np.diagflat([0.01, 0.01, 0.001]))`
Initialize QCarEKF with initial state and noise covariance matrices.

`f(x, u, dt)`
Motion model for the kinematic bicycle model.

`J_f(x, u, dt)`
Jacobian of the motion model for the kinematic bicycle model.

`update(u=None, dt=None, y_gps=None, y_imu=None)`
Update the EKF state estimate using GPS and IMU measurements.

QCarDriveController

Implements a drive controller for a QCar that handles speed and steering

__init__(waypoints, cyclic)
Initialize QCarDriveController

reset()
Resets the internal state of the speed and steering controllers.

updatePath(waypoints, cyclic)
Updates the waypoint path for the steering controller.

update(p, th, v, v_ref, dt)
Updates the drive controller with the current state of the QCar.

SpeedController

A feed forward proportional-integral (PI) controller that produces a control signal based on the tracking error and specified gain terms: Kff, Kp, Ki.

__init__(Kp, Ki, Kff, max_throttle=0.2)
Creates a PID Controller Instance

reset()
This method resets numerical integrator.

update(v, vDesire, dt)
Update the controller output based on the current measured velocity, refernce velocity, and time since last update.

hal/products/mats.py

No description provided.

Classes

SDCSRoadMap

A RoadMap implementation for Quanser's Self-Driving Car studio (SDCS)

__init__(leftHandTraffic=False, useSmallMap=False)
Initialize a new SDCSRoadMap instance.

hal/utilities/control.py

This module contains general implementations for various controller types.

Classes

PID

A proportional-integral-derivative (PID) controller.

__init__(Kp=0, Ki=0, Kd=0, uLimits=None)
Creates a PID Controller Instance

reset()
Reset the controller.

update(r, y, dt)
Update the controller output.

StanleyController

A Stanley controller for following a path of waypoints.

__init__(waypoints, k=1, cyclic=True)
No description provided.

updatePath(waypoints, cyclic)
Update the path of waypoints.

update(p, th, speed)
Update the controller output.

hal/utilities/estimation.py

This module contains general implementations for various estimators.

Classes

EKF

A class for performing Extended Kalman Filtering (EKF).

__init__(kwargs)**
Initializes the EKF class instance with provided inputs.

predict(u, dt)
Predicts the next state of the system.

__predict_linear(u, dt)
No description provided.

__predict_nonlinear(u, dt)
No description provided.

correct(y, dt=None)
Corrects the predicted state based on the measurement.

__correct_linear(y, dt=None)
No description provided.

__correct_nonlinear(y, dt=None)
No description provided.

KalmanFilter

Implements a linear Kalman filter.

__init__(kwargs)**
No description provided.

hal/utilities/image_processing.py

No description provided.

Classes

ImageProcessing

No description provided.

__init__()
No description provided.

calibrate_camera(imageCaptures, chessboardDimensions, boxSize)
No description provided.

undistort_img(distImgs, cameraMatrix, distCoefficients)
No description provided.

do_canny(frame)
No description provided.

do_segment(frame, steering=0)
No description provided.

calculate_lines(segment)
No description provided.

average_lines(frame, left, right)
No description provided.

calculate_coordinates(parameters)
No description provided.

driving_parameters(lineParameters)
No description provided.

visualize_lines(frame, lines)
No description provided.

detect_yelllow_lane(frame)
No description provided.

extract_point_given_row(row)
No description provided.

body_to_image(bodyPoint, extrinsicMatrix, intrinsicMatrix)
No description provided.

image_to_body(imagePoint, extrinsicMatrix, intrinsicMatrix)
No description provided.

binary_thresholding(frame, lowerBounds, upperBounds)
This function will automatically detect 3-color (BGR, RGB, HSV) or Grayscale images, and corresponding threshold using the bounds.

image_filtering_close(frame, dilate=1, erode=1, total=1)
This function performs a morphological dilation followed by erosion, useful for filling small holes/gaps in the image.

image_filtering_open(frame, dilate=1, erode=1, total=1)
This function performs a morphological erosion followed by dilation, useful for removing small objects in the image.

image_filtering_skeletonize(frame)
This function performs a morphological skeletonization, useful for retrieving the skeleton of an image while maintaining the Euler # of objects.

mask_image(frame, rowUp, rowDown, colLeft, colRight)
This function masks the provided binary image outside the rectangle defined by input parameters. If any of the row/col parameters are negative, or outside the bounds of the image size, the returned image will be the frame itself.

extract_lane_points_by_row(frame, row)
This function extracts the left most and right most point in provided row in the input frame where a black to white pixel transition is detected.

find_slope_intercept_from_binary(binary)
This function will return the linear polynomial fit coefficients to the lane found in a binary image.

get_perspective_transform(ptsUpperRow, ptsLowerRow)
This function returns a perspective transform from the provided points for a birds-eye-view. Use this function during calibration to retrieve a transform with atleast 2 markings perpendicular to the camera.

circle_pts(frame, pts, radius, color)
This function draws a circle in the input image at the provided points.

extract_lines(filteredImage, originalImage)
No description provided.

hal/utilities/path_planning.py

path_planning.py: A collection of utilities for use in path planning.

Module Functions

hermite_position(s, p1, p2, t1, t2)
Compute the position at point 's' along a cubic Hermite spline.

hermite_tangent(s, p1, p2, t1, t2)
Compute the derivative of a cubic Hermite spline at s.

hermite_heading(s, p1, p2, t1, t2)
Compute the heading angle at s along a cubic Hermite spline.

SCSPath(startPose, endPose, radius, stepSize=0.1)
Calculate the path between two poses using at most one turn.

WHPath(startPose, endPose, radius, startAngle, endAngle, stepSize=0.1)
Calculate the path between two poses using at most one turn.

Classes

RoadMapNode

Class for representing nodes in the graph of a RoadMap

__init__(pose)
Initialize a RoadMapNode instance.

RoadMapEdge

Class for representing edges in the graph of a RoadMap.

__init__(fromNode, toNode)
Initialize a RoadMapEdge instance.

RoadMap

Graph-based roadmap for generating paths between points in a road network.

__init__()
Initialize a RoadMap instance.

add_node(pose)
Add a node to the roadmap.

add_edge(fromNode, toNode, radius)
Add an edge between two nodes in the roadmap.

remove_edge(fromNode, toNode)
Remove an edge between two nodes in the roadmap.

_calculate_trajectory(edge, radius)
Calculate the waypoints and length of the given edge

get_node_pose(nodeID)
Get the pose of a node by its index.

generate_path(nodeSequence)
Generate the shortest path passing through the given sequence of nodes

_heuristic(node, goalNode)
Calculate the heuristic cost between two nodes.

find_shortest_path(startNode, goalNode)
Find the shortest path between two nodes using the A* algorithm.

display(ax=None)
Display the roadmap with nodes, edges, and labels using matplotlib.

initial_check(initPose, nodeSequence, waypointSequence)
No description provided.

get_init_waypoints(pose)
Find the closest waypoint to the given pose.

get_closest_node(pose)
"find the closest node to the given pose. Args: pose (numpy.ndarray): Pose in the form [x, y, th]. Returns: int: Index of the closest node in the roadmap.

Dubins

Class implementing a Dubins path planner with a constant turn radius.
https://github.com/FelicienC/RRT-Dubins/blob/master/code/dubins.py

__init__(radius, point_separation)
No description provided.

all_options(start, end, sort=True)
Computes all the possible Dubin's path and returns them, in the form of a list of tuples representing each option: (path_length, dubins_path, straight).

lsl(start, end, center_0, center_2)
Left-Straight-Left trajectories. First computes the poisition of the centers of the turns, and then uses the fact that the vector defined by the distance between the centers gives the direction and distance of the straight segment.

rsr(start, end, center_0, center_2)
Right-Straight-Right trajectories. First computes the poisition of the centers of the turns, and then uses the fact that the vector defined by the distance between the centers gives the direction and distance of the straight segment.

rsl(start, end, center_0, center_2)
Right-Straight-Left trajectories. Because of the change in turn direction, it is a little more complex to compute than in the RSR or LSL cases. First computes the position of the centers of the turns, and then uses the rectangle triangle defined by the point between the two circles, the center point of one circle and the tangeancy point of this circle to compute the straight segment distance.

lsr(start, end, center_0, center_2)
Left-Straight-Right trajectories. Because of the change in turn direction, it is a little more complex to compute than in the RSR or LSL cases. First computes the poisition of the centers of the turns, and then uses the rectangle triangle defined by the point between the two circles, the center point of one circle and the tangeancy point of this circle to compute the straight segment distance.

lrl(start, end, center_0, center_2)
Left-right-Left trajectories. Using the isocele triangle made by the centers of the three circles, computes the required angles.

rlr(start, end, center_0, center_2)
Right-left-right trajectories. Using the isocele triangle made by the centers of the three circles, computes the required angles.

find_center(point, side)
Given an initial position, and the direction of the turn, computes the center of the circle with turn radius self.radius passing by the intial point.

WHNode

Class for representing nodes in the graph of a WHMap

__init__(pose)
Initialize a WHNode instance.

WHEdge

Class for representing edges in the graph of a WHMap.

__init__(nodeA, nodeB)
Initialize a WHEdge instance.

WHMap

Graph-based warehouse map for generating paths between points in a network.

__init__(stepSize)
Initialize a WHMap instance.

add_node(pose, nodeType)
Add a node to the warehouse map.

add_edge(nodeA, nodeB, startAngle, endAngle, radius=0)
Add an edge between two nodes in the roadmap.

remove_edge(nodeA, nodeB)
Remove an edge between two nodes in the roadmap.

_calculate_trajectory(edge, radius, startAngle, endAngle)
Calculate the waypoints and length of the given edge

_purge_gray_nodes(nodeID)
Only to be used for gray node purge. Every gray node has 2 edges, which are merged by this method.

get_node_pose(nodeID)
Get the pose of a node by its index.

setup()
No description provided.

display()
No description provided.

PAL Library

pal/products/qcar.py

qcar: A module for simplifying interactions with the QCar hardware platform.

Classes

QCar

Class for performing basic QCarIO

__init__(readMode=0, frequency=500, pwmLimit=0.3, steeringBias=0, hilPort=18960)
Configure and initialize the QCar.

_elapsed_time()
No description provided.

_configure_parameters()
No description provided.

_set_options()
No description provided.

_create_io_task()
No description provided.

terminate()
No description provided.

read_write_std(throttle, steering, LEDs=None)
Read and write standard IO signals for the QCar

read()
No description provided.

write(throttle, steering, LEDs=None)
No description provided.

__enter__()
No description provided.

__exit__(type, value, traceback)
No description provided.

QCarCameras

Class for accessing the QCar's CSI cameras.

__init__(frameWidth=820, frameHeight=410, frameRate=30, enableRight=False, enableBack=False, enableLeft=False, enableFront=False, videoPort=18961)
Initializes QCarCameras object.

readAll()
Reads frames from all enabled cameras.

__enter__()

Used for with statement.

__exit__(type, value, traceback)
Used for with statement. Terminates all enabled cameras.

terminate()
Used for with statement. Terminates all enabled cameras.

QCarLidar

QCarLidar class represents the LIDAR sensor on the QCar.

__init__(numMeasurements=384, rangingDistanceMode=2, interpolationMode=0, interpolationMaxDistance=0, interpolationMaxAngle=0, enableFiltering=True, angularResolution=1 * np.pi / 180, lidarPort=18966)
Initializes a new instance of the QCarLidar class.

read()
Reads data from the LIDAR sensor and applies filtering if enabled.

filter_rplidar_data(angles, distances)
Filters RP LIDAR data

QCarRealSense

A class for accessing 3D camera data from the RealSense camera on the QCar.

__init__(mode='RGB&DEPTH', frameWidthRGB=1920, frameHeightRGB=1080, frameRateRGB=30, frameWidthDepth=1280, frameHeightDepth=720, frameRateDepth=15, frameWidthIR=1280, frameHeightIR=720, frameRateIR=15, readMode=1, focalLengthRGB=np.array([[None], [None]], dtype=np.float64), principlePointRGB=np.array([[None], [None]], dtype=np.float64), skewRGB=None, positionRGB=np.array([[None], [None], [None]], dtype=np.float64), orientationRGB=np.array([[None, None, None], [None, None, None], [None, None, None]], dtype=np.float64), focalLengthDepth=np.array([[None], [None]], dtype=np.float64), principlePointDepth=np.array([[None], [None]], dtype=np.float64), skewDepth=None, positionDepth=np.array([[None], [None], [None]], dtype=np.float64), orientationDepth=np.array([[None, None, None], [None, None, None], [None, None, None]], dtype=np.float64), video3dPort=18965)
No description provided.

QCarGPS

A class that reads GPS data from the GPS server.

__init__(initialPose=[0, 0, 0], calibrate=False, gpsPort=18967, lidarIdealPort=18968)
Initializes the QCarGPS class with the initial pose of the QCar.

__initLidarToGPS(initialPose)
No description provided.

__stopLidarToGPS()
No description provided.

__calibrate()
Calibrates the QCar at its initial position and heading.

__emulateGPS()
Starts the GPS emulation using the qcarLidarToGPS executable.

readGPS()
Reads GPS data from the server and updates the position and orientation attributes.

readLidar()
Reads GPS data from the server and updates the position and orientation attributes.

filter_rplidar_data(angles, distances)
Filters RP LIDAR data

terminate()
Terminates the GPS client.

__enter__()
Used for with statement.

__exit__(type, value, traceback)
Used for with statement. Terminates the GPS client.

pal/products/qcar_config.py

No description provided.

Classes

QCar_check

No description provided.

__init__(IS_PHYSICAL)
No description provided.

check_car_type()
No description provided.

create_config()
No description provided.

set_qcar1_params()
No description provided.

set_qcar2_params()
No description provided.

pal/utilities/gamepad.py

gamepad: A module for simplifying interactions with gamepads (controllers).

Classes

LogitechF710

Class for interacting with the Logitech Gamepad F710.

__init__(deviceID=1)
Initialize and open a connection to a LogitechF710 GameController.

read()
Update the gamepad states by polling the GameController.

terminate()
Terminate the GameController connection.

__enter__()
No description provided.

__exit__(type, value, traceback)
No description provided.

pal/utilities/keyboard.py

No description provided.

Classes

PygameKeyboard

No description provided.

__init__()
No description provided.

read()
No description provided.

terminate()

Terminate pygame

PygameKeyboardDrive

No description provided.

__init__(maxThrottle=0.2, maxSteer=0.1)
No description provided.

update(kb)
No description provided.

KeyboardDrive

No description provided.

__init__(mode=0, maxThrottle=0.2, maxSteer=0.1)
No description provided.

update(kb)
No description provided.

KeyBoardDriver

This class supports real-time key press polls for 30 specific keys. The keys included are function keys (Space, Home and Esc), True Arrow keys (Up, Down, Right, Left), False Arrow keys (W, S, D, A), Numpad keys (1 to 9) , Number Keys (1 to 0).

__init__(rate=30)
Valid options for keyboard rate = 10, 30, 100 & 500 Hz.

assign()
This method updates keys from the receive buffer.

poll()
This method checks if any of the 37 supported keys are currently pressed.

update()
This method updates the keyboard variables to the current key press status.

checkConnection()
No description provided.

print()
No description provided.

terminate()
This method terminates the Keyboard and it's client gracefully.

QKeyboard

No description provided.

__init__()
No description provided.

update()
No description provided.

pal/utilities/lidar.py

lidar: A module for simplifying interactions with common LiDAR devices.

Classes

Lidar

A class for interacting with common LiDAR devices.

__init__(type='RPLidar', numMeasurements=384, rangingDistanceMode=2, interpolationMode=0, interpolationMaxDistance=0, interpolationMaxAngle=0)
Initialize a Lidar device with the specified configuration.

read()
Read a scan and store the measurements

terminate()
Terminate the LiDAR device connection correctly.

__enter__()
Return self for use in a 'with' statement.

__exit__(type, value, traceback)
Terminate the LiDAR device connection when exiting a 'with' statement.

pal/utilities/math.py

math.py: A module providing handy functions for common mathematical tasks.

Module Functions

wrap_to_2pi(th)
Wrap an angle in radians to the interval [0, 2*pi).

wrap_to_pi(th)
Wrap an angle in radians to the interval [-pi, pi).

angle(v1, v2)

Compute the angle in radians between two vectors.

signed_angle(v1, v2=None)
Find the signed angle between two vectors

get_mag_and_angle(v)
No description provided.

find_overlap(a, b, i, j)
Finds slices that correspond to overlapping cells of two 2D numpy arrays

ddt_filter(u, state, A, Ts)
No description provided.

lp_filter(u, state, A, Ts)
No description provided.

Classes

SignalGenerator

Class object consisting of common signal generators

sine(amplitude, angularFrequency, phase=0, mean=0)
This function outputs a sinusoid wave based on the provided timestamp.

cosine(amplitude, angularFrequency, phase=0, mean=0)
Outputs a cosinusoid wave based on the provided timestamp.

PWM(frequency, width, phase=0)
This function outputs a PWM wave based on the provided timestamp.

square(amplitude, period)
Outputs a square wave based on the provided timestamp.

Calculus

Class object consisting of basic derivative and integration functions

differentiator(dt, x0=0)
Finite-difference-based numerical derivative.

differentiator_variable(dt, x0=0)
Finite-difference-based numerical derivative.

integrator(dt, integrand=0, saturation=None)
Iterative numerical integrator.

integrator_variable(dt, integrand=0, minSaturation=-np.inf, maxSaturation=np.inf)
Iterative numerical integrator.

Integrator

No description provided.

__init__(integrand=0, minSaturation=-np.inf, maxSaturation=np.inf)
No description provided.

integrate(rate, timestep)
No description provided.

reset(resetValue)
No description provided.

Filter

Class object consisting of different filter functions

low_pass_first_order(wn, dt, x0=0)
Standard first order low pass filter.

low_pass_first_order_variable(wn, dt, x0=0)
Standard first order low pass filter.

low_pass_second_order(wn, dt, zeta=1, x0=0)
Standard second order low pass filter.

complimentary_filter(kp, ki, dt, x0=0)
Complementary filter based rate and correction signal using a combination of low and high pass on them respectively.

moving_average(samples, x0=0)
Standard moving average filter.

Signal

No description provided.

edge_detector()
Rising edge and falling edge detector.

pal/utilities/probe.py

No description provided.

Classes

Probe

Class object to send data to a remote Observer. Includes support for Displays (for video data), Plots (standard polar plot as an image) and Scope (standard time series plotter).

`__init__(ip='localhost')`
No description provided.

`add_display(imageSize=[480, 640, 3], scaling=True, scalingFactor=2, name='display')`
No description provided.

`add_plot(numMeasurements=1680, scaling=True, scalingFactor=2, name='plot')`
No description provided.

`add_scope(numSignals=1, name='scope')`
No description provided.

`check_connection()`
Attempts to connect every unconnected probe in the agentList. Returns True if every probe is successfully connected.

`send(name, imageData=None, lidarData=None, scopeData=None)`
Ensure that at least one of imageData, lidarData, scopeData is provided, and that the type of data provided matches the expected name, otherwise an error message is printed and the method returns False.

`terminate()`
No description provided.

ObserverAgent

No description provided.

`__init__(uriAddress, id, bufferSize, buffer, agentType, properties)`
No description provided.

`receive()`
No description provided.

`process()`
No description provided.

`check_connection()`
Checks if the sendCamera object is connected to its server. returns True or False.

`terminate()`
Terminates the connection.

Observer

Class object to send data to a remote Observer. Includes support for Displays (for video data), Plots (standard polar plot as an image) and Scope (standard time series plotter).

__init__()
No description provided.

add_display(imageSize=[480, 640, 3], scalingFactor=2, name=None)
No description provided.

add_plot(numMeasurements, frameSize=400, pixelsPerMeter=40, scalingFactor=4, name=None)
No description provided.

add_scope(numSignals=1, name=None, signalNames=None)
No description provided.

thread_function(index)
No description provided.

launch()
No description provided.

terminate()
No description provided.

RemoteDisplay

Class object to send camera feed to a device. Works as a client.

__init__(ip='localhost', id=0, imageSize=[480, 640, 3], scaling=True, scalingFactor=2)
ip - IP address of the device receiving the data as a string, eg. '192.168.2.4'

check_connection()
Checks if the sendCamera object is connected to its server. returns True or False.

send(image)
Resizes the image if needed and sends the image added as the input.

terminate()
Terminates the connection.

RemotePlot

No description provided.

__init__(ip='localhost', id=1, numMeasurements=1680, scaling=True, scalingFactor=4)

No description provided.

check_connection()
Checks if the sendCamera object is connected to its server. returns True or False.

send(distances=None, angles=None)
No description provided.

terminate()
Terminates the connection.

RemoteScope

No description provided.

__init__(numSignals=1, id=1, ip='localhost')
No description provided.

check_connection()
Checks if the sendCamera object is connected to its server. returns True or False.

send(time, data=None)
No description provided.

terminate()
Terminates the connection.

pal/utilities/stream.py

stream.py: A module providing utility classes for using Quanser's stream API.

Classes

BaseStream

Base class for StreamServer and StreamClient.

__init__()
No description provided.

timeoutDuration()
Returns the timeout duration in seconds

timeoutDuration(timeout_duration_seconds)
Sets the timeout as an instance of Timeout.

terminate()
Terminate the stream by shutting down and closing the connection.

_check_poll_flags(flags)
Check if the specified are set for self._stream.

__enter__()
No description provided.

__exit__(exc_type, exc_val, exc_tb)
No description provided.

StreamServer

Stream server class for accepting and managing client connections.

__init__(uri, blocking=False)
Initialize the stream server.

status()
int: Get the status of the server (1: listening, 2: closed).

accept_clients()
Accept a client connection.

StreamClient

Stream client class for connecting to a stream server.

__init__(uri, blocking=False)
Initialize the stream client.

_connect()
No description provided.

status()
Returns the status of the client.

receive()
Receive data from the stream.

send(obj)
Send data over the stream.

BasicStream

Class object consisting of basic stream server/client functionality

__init__(uri, agent='S', receiveBuffer=np.zeros(1, dtype=np.float64), sendBufferSize=2048, recvBufferSize=2048, nonBlocking=False, verbose=False, reshapeOrder='C')
This functions simplifies functionality of the quanser_stream module to provide a simple server or client.

checkConnection(timeout=Timeout(seconds=0, nanoseconds=100))
When using non-blocking connections (nonBlocking set to True), the constructor method for this class does not block when listening (as a server) or connecting (as a client). In such cases, use the checkConnection method to attempt continuing to accept incoming connections (as a server) or connect to a server (as a client).

terminate()
Use this method to correctly shutdown and then close connections. This method automatically closes all streams involved (Server will shutdown server streams as well as client streams).

receive(iterations=1, timeout=Timeout(seconds=0, nanoseconds=10))
This functions populates the receiveBuffer with bytes if available.

send(buffer)
This functions sends the data in the numpy array buffer (server or client).

pal/utilities/vision.py

vision.py: A module for simplifying interaction with various camera types.

Classes

Camera3D

No description provided.

__init__(mode='RGB, Depth', frameWidthRGB=1920, frameHeightRGB=1080, frameRateRGB=30.0, frameWidthDepth=1280, frameHeightDepth=720, frameRateDepth=15.0, frameWidthIR=1280, frameHeightIR=720, frameRateIR=15.0, deviceId='0', readMode=1, focalLengthRGB=np.array([[None], [None]], dtype=np.float64), principlePointRGB=np.array([[None], [None]], dtype=np.float64), skewRGB=None, positionRGB=np.array([[None], [None], [None]], dtype=np.float64), orientationRGB=np.array([[None, None, None], [None, None, None], [None, None, None]], dtype=np.float64), focalLengthDepth=np.array([[None], [None]], dtype=np.float64), principlePointDepth=np.array([[None], [None]], dtype=np.float64), skewDepth=None, positionDepth=np.array([[None], [None], [None]], dtype=np.float64), orientationDepth=np.array([[None, None, None], [None, None, None], [None, None, None]], dtype=np.float64))
This class configures RGB-D cameras (eg. Intel Realsense) for use.

terminate()
Terminates all started streams correctly.

read_RGB()
Reads an image from the RGB stream. It returns a timestamp for the frame just read. If no frame was available, it returns -1.

read_depth(dataMode='PX')

Reads an image from the depth stream. Set dataMode to 'PX' for pixels or 'M' for meters. Use the corresponding image buffer to get image data. If no frame was available, it returns -1.

read_IR(lens='LR')
Reads an image from the left and right IR streams based on the lens parameter (L or R). Use the corresponding image buffer to get image data. If no frame was available, it returns -1.

extrinsics_rgb()
Provides the Extrinsic Matrix for the RGB Camera

intrinsics_rgb()
Provides the Intrinsic Matrix for the RGB Camera

extrinsics_depth()
Provides the Extrinsic Matrix for the Depth Camera

intrinsics_depth()
Provides the Intrinsic Matrix for the Depth Camera

__enter__()
No description provided.

__exit__(type, value, traceback)
No description provided.

Camera2D

No description provided.

__init__(cameraId='0', frameWidth=820, frameHeight=410, frameRate=30.0, focalLength=np.array([[None], [None]], dtype=np.float64), principlePoint=np.array([[None], [None]], dtype=np.float64), skew=None, position=np.array([[None], [None], [None]], dtype=np.float64), orientation=np.array([[None, None, None], [None, None, None], [None, None, None]], dtype=np.float64), imageFormat=0, brightness=None, contrast=None, gain=None, exposure=None)
Configures the 2D camera based on the cameraId provided.

read()
Reads a frame, updating the corresponding image buffer. Returns a flag indicating whether the read was successful.

reset()
Resets the 2D camera stream by stopping and starting the capture service.

terminate()
Terminates the 2D camera operation.

extrinsics()
Provides the Extrinsic Matrix for the Camera

intrinsics()
Provides the Intrinsic Matrix for the Camera

__enter__()
Used for with statement.

__exit__(type, value, traceback)
Used for with statement. Terminates the Camera