HAL + PAL

Function Reference

Automatically generated reference from docker/0_libraries/python/hal and docker/0_libraries/python/pal.

Modules documented: 30

Functions/methods documented: 535

Classes documented: 95

Each item includes its callable signature and the first paragraph of its docstring.

HAL Library

hal/__init__.py

hal: high-level utilities for interacting with Quanser products in python.

hal/content/qarm_mini.py

No description provided.

Classes

QArmMiniFunctions

This class contains kinematic functions for the QArm Mini such as Forward/Inverse Position/Differential Kinematics etc.

forward_kinematics(theta)
Implements the forward position kinematics for the QArm Mini.

inverse_kinematics(posEndEffector, gamma, thetaPrevious)
Implements the inverse position kinematics for the QArm Mini.

differential_kinematics(theta)
Implements the differential kinematics for the QArm Mini.

cubic_spline(T, Pf, Pi)
Generate cubic spline coefficients for X, Y, Z.

waypoint_navigator(duration, waypoints, total_time, current_position)
Parameters: duration : Time in seconds between each waypoint transition waypoints : 4xN matrix of waypoints dt : Time step (sample time) total_time : Current time from timer (in seconds) current_position : Current measured position of the robot (4D)

quanser_DH(a, alpha, d, theta)
QUANSER_DH v 1.0 - 26th March 2019 REFERENCE: Chapter 3. Forward and Inverse Kinematics Robot Modeling and Control Spong, Hutchinson, Vidyasagar 2006 (Using standard DH parameters)

QArmMiniKeyboardNavigator

This class provides a range of methods that let you drive the QArm Mini manipulator using a KeyboardDriver class.

__init__(keyboard, initialPose=QArmMini.HOME_POSE)
No description provided.

activate_joint(mode='joint')
No description provided.

move_joints_with_keyboard(timestep, speed=np.pi / 12)
Tap the keyboard 1 through 4 keys to select a joint. Then use the UP and DOWN arrows to increase or decrease the joint's cmd respectively.

move_ee_with_keyboard(timestep, speed=0.05)
Tap the keyboard x, y, z or g keys to select a cartesian x, y, z axis or the end-effector orientation angle gamma. Then use the UP and DOWN arrows to increase or decrease the corresponding cmd value. Note, the speed is multiplied by a factor of 10 for the end-effector angle Gamma.

DataIO

Writes/reads manipulator data to/from a csv file.

__init__(filename='data.csv', newLine='')
No description provided.

write(data, mode='joints')
Writes manipulator data to a csv file.

read()
Reads manipulator data from a csv file.

PlotData

Plotter based on Matplotlib's pyplot specifically for QArm Mini Labs.

plot3D1Curve(data)
Plots a single 3D X/Y/Z curve. No time data required.

plot3D2Curves(dataCmd, dataMeas)
Plots a pair of 3D X/Y/Z curves. No time data required. Use to compare commands vs measured end-effector data.

plot2DJointCurves(data, timeData)
Plots joint data on a time axis.

plot2D2JointCurves(dataCmd, dataMeas, timeData)
Plots a pair of joint data curves on a time axis. Use to compare commands vs measured joint data.

QArmMiniImageProcessing

No description provided.

__init__(resolution=(640, 360, 3))
No description provided.

guassian_blur(img, kernelSize=5, stdDev=1)
No description provided.

convert(img, code=cv2.COLOR_BGR2HSV)
No description provided.

threshold_grayscale(img, upper, lower, code=cv2.THRESH_BINARY)
No description provided.

threshold_color(img, upper, lower)
No description provided.

close(img, kernel, iterations=1)
No description provided.

open(img, kernel, iterations=1)
No description provided.

highlight(imgRGB, mask)
No description provided.

find_objects(image, connectivity, minArea, maxArea)
No description provided.

draw_lines(img_in, color, pts)
Draw lines connecting the points using the provided color

draw_box_on_detection(image, col, row, area)
Helper function to draw bounding box based on find_objects results Assumes the bounding box can be derived from connected components stats

hal/content/qbot_platform_functions.py

No description provided.

Classes

QBPMovement

This class contains the functions for the QBot Platform such as Forward/Inverse Differential Drive Kinematics etc.

__init__()
No description provided.

diff_drive_inverse_velocity_kinematics(forSpd, turnSpd)
This function is for the differential drive inverse velocity kinematics for the QBot Platform. It converts provided body speeds (forward speed in m/s and turn speed in rad/s) into corresponding wheel speeds (rad/s).

diff_drive_forward_velocity_kinematics(wL, wR)
This function is for the differential drive forward velocity kinematics for the QBot Platform. It converts provided wheel speeds (rad/s) into corresponding body speeds (forward speed in m/s and turn speed in rad/s).

QBPVision

No description provided.

__init__()
No description provided.

undistort_img(distImgs, cameraMatrix, distCoefficients)
This function undistorts a general camera, given the camera matrix and coefficients.

df_camera_undistort(image)
This function undistorts the downward camera using the camera intrinsics and coefficients.

subselect_and_threshold(image, rowStart, rowEnd, minThreshold, maxThreshold)
This function subselects a horizontal slice of the input image from rowStart to rowEnd for all columns, and then thresholds it based on the provided min and max thresholds. Returns the binary output from thresholding.

image_find_objects(image, connectivity, minArea, maxArea)
This function implements connected component labeling on the provided image with the desired connectivity. From the list of blobs detected, it returns the first blob that fits the desired area criteria based on minArea and maxArea provided. Returns the column and row location of the blob centroid, as well as the blob's area.

line_to_speed_map(sampleRate, saturation)
No description provided.

QBPRanging

No description provided.

__init__()
No description provided.

adjust_and_subsample(ranges, angles, end=-1, step=4)
No description provided.

correct_lidar(lidarPosition, ranges, angles)
No description provided.

detect_obstacle(ranges, angles, forSpd, forSpeedGain, turnSpd, turnSpeedGain, minThreshold, obstacleNumPoints)
No description provided.

hal/content/qcar.py

This is a modified version of qcar.py for interfacing virtual Qcar2 in a remote machine. Currenly only used by the Lane Keeping lab.

Classes

QCar

Class for performing basic QCarIO

__init__(readMode=0, frequency=500, pwmLimit=0.3, steeringBias=0)
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.

terminate_pc_application()
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)
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)
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))
```
No description provided.

QCarGPS

A class that reads GPS data from the GPS server.

```
__init__(initialPose=[0, 0, 0], calibrate=False)
```
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.

hal/content/qcar_config.py

This is a modified version of qcar_config.py for interfacing virtual Qcar2 in a remote machine.
Currenly only used by the Lane Keeping lab.

Classes

QCar_check

No description provided.

__init__(IS_PHYSICAL, remote_ip='172.16.4.61')
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.

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.

ObjectDetection

An classical object detection algorithm with different options of detection methods, as well as visualization tools.

__init__()
Initializes the object detection class.

color_thresholding(img, lower, upper)
Produce a binary mask based on lower and upper thresholding values.

mask_img(img, mask)
Display color thresholded image.

obj_detect(img, task, mode)
Detect objects in an RGB image and return name, binary mask, and bounding box of each detected object.

find_distance(depth, detectedMask)
Calculate distance of objects based on their binary mask and depth image.

set_tracker(mode='rgb')
Set trackers in CV2 window to assist in color thresholding.

get_tracker()
Get tracker values to be used for creating thresholding masks.

annotate(img, detectedName, detectedBbox, detectedDist)

Add names and bounding boxes of detected objects to the input image.

_nothing(val)
Dummy function used in cv2.createTrackbar().

LaneKeeping

Overarching class containing functions used for the lane keeping skills activity.

__init__(Kdd, ldMin, ldMax, maxSteer, bevShape, bevWorldDims)
Initialize LaneKeeping.

preprocess(lane_marking)
Merge the two edges of the same lane marking and convert all grey pixels to white.

isolate_lane_markings(lane_marking)
Isolate seperate lane marking via connected component analysis.

find_target(isolated_binary_blobs, v)
Compute look-ahead distance from measured velocity, then estimate lane centers (targets for pure pursuit) from isolated lane markings.

find_rot_mat(lane)
Given a lane marking of interest and the its intersection point, a rotation matrix, which creates a vector perpendicular to the lane marking, is computed.

show_debug()
Render debug visualization and display via cv2

_draw_intersect(intersect)
draw the intesection point on the debug output

_draw_angle(intersect, target_point)
visualize the angle for more debug info

_ccw_compare(lane1, lane2)
helper function used for sorting points in CCW order w.r.t. specified center point, based on this answer https://stackoverflow.com/questions/6989100/sort-points-in-clockwise-order

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.

InversePerspectiveMapping

Inverse perspective mapping (IPM) algorithm that creates a bird's-eye view image.

__init__(bevShape, bevWorldDims)
Creates a IPM Instance

get_extrinsics()
This method is used to create the transformation matrices which bring a point from vehicle frame to camera frame.

v2img(XYZ)
Converts input locations XYZ in vehicle frame to pixel coordinates uv in image frame.

get_homography()
This method creates the homography matrix (self.M) which converts. RGB camera feed to BEV in desired dimensions. This is done by: 1.Specifying four arbitrary points in vehicle frame (four corners). 2.Converting them to image coordinates in RGB image frame. 3.Compute the BEV coordinates of the four arbitrary points. 4.Compute the homography matrix between the RGB corners and BEV corners.

create_bird_eye_view(img)
Creates BEV from input camera RGB feed

PurePursuitController

A pure pursuit controller that produces a steering controller based on specified parameters and a target point.

__init__(m_per_pix, rear_wheel_pos, maxSteer)
Creates a pure pursuit controller instance.

target2steer(point)
Calculate output steering angle based on input target point.

LaneMarking

This class stores relevant information for individual isolated lane marking.

__init__(bev_rear_wheel_pos, binary)
Creates a lane marking instance.

find_intersection(ld)
Find the intersection point, and stores it in self.intersection

TargetsTracker

Basic object tracker for target points. Consecutive IDs are assigned to each target point such that the target point to the left has larger ID value.

`__init__(max_frame=2)`
Creates a target point tracker instance.

`update(new_targets)`
Assigned IDs to new target points by comparing their distances to registered target points.

`register(point, id)`
Update self.target_dict with an ID-target pair.

`deregister(id)`
Remove ID-target pair in dictionaries if they disappear for too long.

`_assign_id(all_targets, num_unused)`
Assign consecutive IDs to each target point such that the target point to the left has larger ID value.

LaneSelection

Facilitate the usage of gamepad left and right buttons for selecting detected lane centers.

`__init__()`
Create a Lane Selector instance and initialize signal edge detector.

`select(myLaneKeeping, kb)`
Use gamepad input to select from valid ID of detected target points.

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/products/qarm.py

No description provided.

Classes

QArmUtilities

QArm utilities, such as Forward/Inverse Kinematics, Differential Kinematics etc.

take_user_input_joint_space()
Use this method to take a user input for joint commands. Note that this method pauses execution until it returns. This is elegantly useful with the Position Mode on the QArm, which uses low level trajectory generation.

take_user_input_task_space()
Use this method to take a user input for end-effector position commands. Note that this method pauses execution until it returns. Note that this uses joint level trajectory generation, so the end-effector path will not be linear.

forward_kinematics(phi)
QUANSER_ARM_FPK v 1.0 - 30th August 2020

inverse_kinematics(p, gamma, phi_prev)
QUANSER_ARM_IPK v 1.0 - 31st August 2020

_check_joint_limits(phi)
No description provided.

quanser_arm_DH(a, alpha, d, theta)
QUANSER_ARM_DH v 1.0 - 26th March 2019

differential_kinematics(phi)
Implements the jacobian matrix using theta for the QArm Mini

take_widget_mass()
Use this method to take a user input for widget weight.

QArmKeyboardNavigator

This class provides a range of methods that let you drive the QArm Mini manipulator using a KeyboardDriver class.

__init__(keyboardDriver, initialPose=QArm.HOME_POSE)
No description provided.

activate_joint(mode='joint')
No description provided.

move_joints_with_keyboard(timestep, speed=np.pi / 12)
Tap the keyboard 1 through 4 keys to select a joint. Then use the UP and DOWN arrows to increase or decrease the joint's cmd respectively.

move_ee_with_keyboard(timestep, speed=0.02)
Tap the keyboard x, y, z or g keys to select a cartesian x, y, z axis or the end-effector orientation angle gamma. Then use the UP and DOWN arrows to increase or decrease the corresponding cmd value. Note, the speed is multiplied by a factor of 10 for the end-effector angle Gamma.

hal/products/qcar.py

This module contains QCar specific implementations of the hal.utilities.geometry features

Classes

QCarGeometry

QCarGeometry class for defining QCar-specific frames of reference.

__init__()
Initialize the QCarGeometry class with QCar-specific frames.

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/geometry.py

geometry.py: A module for managing coordinate frames and transformations.

Classes

Geometry

A class for managing frames and transformations in a 3D coordinate system.

__init__()
Initialize a new Geometry instance with a default 'world' frame.

add_frame(name, p=[0, 0, 0], R=np.eye(3), base=None)
Add a new frame, defined relative to the specified base frame.

get_frames()
Get a list of all frame names in the calling instance.

get_transform(frame, base=None)
Get the 4x4 transformation matrix between the frame and base.

get_translation(frame, base=None)
Get the translation vector between the frame and base.

get_rotation_rm(frame, base=None)
Get the rotation matrix between the specified frame and base.

get_rotation_ea(frame, base=None)
Get the Euler angles (extrinsic XYZ) between frame and base.

get_rotation_q(frame, base=None)
Get the quaternion representing the rotation between frame and base.

get_pose_pea(frame, base=None)
Get the pose (position and Euler angles) of frame w.r.t. base.

get_pose_pq(frame, base=None)
Get the pose (position and quaternion) of frame w.r.t. base.

set_transform(T, frame, base=None)
Set the 4x4 transformation matrix for frame relative to base.

set_pose_prm(p, R, frame, base=None)
Set the pose (position and rotation matrix) of frame w.r.t. base.

set_pose_pea(pose, frame, base=None)
Set the pose (position and Euler angles) of frame w.r.t. base.

set_pose_pq(pose, frame, base=None)
Set the pose (position and quaternion) of frame w.r.t. base.

set_translation(p, frame, base=None)
Set the translation vector for frame relative to the base.

set_rotation_rm(R, frame, base=None)
Set the rotation matrix for frame relative to base.

set_rotation_ea(ea, frame, base=None)
Set the Euler angles (extrinsic XYZ) for frame relative to base.

set_rotation_q(q, frame, base=None)

Set the quaternion for the frame relative to base.

MobileRobotGeometry

A class for managing frames and transformations for mobile robots.

__init__()
Initialize a new instance and add the 'body' frame.

get_pose_2d(frame='body', base='world')
Get the 2D pose (x, y, heading) of frame relative to base.

get_heading(frame='body', base='world')
Get the heading angle of frame relative base.

set_pose_2d(pth, frame='body', base='world')
Set the 2D pose (x, y, heading) of frame relative to base.

set_body_from_T(T, frame)
Sets the body frame relative to the world frame using a transformation matrix 'T', defined relative to 'frame'.

set_body_from_Rp(R, p, frame)
Set the body frame relative to the world frame using a rotation matrix R and a translation vector p relative to another frame.

set_body_from_pea(p, ea, frame)
Set the body frame relative to the world frame using a translation vector p and extrinsic Euler angles ea relative to another frame.

set_body_from_pth(p, th, frame)
Set the body frame relative to the world frame using a translation vector p and a heading angle th relative to another frame.

_rm_from_heading(th)
Create a 3x3 rotation matrix from a heading angle.

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/mapping.py

No description provided.

Module Functions

find_overlap(a, b, i, j)
No description provided.

wrap_to_2pi(th)
No description provided.

Classes

OccupancyGrid

No description provided.

__init__(xLength=0, yLength=0, cellWidth=0, sensor=None, pPrior=0.5, pSatLimit=0.001, useGPU=False)
No description provided.

m()
No description provided.

n()
No description provided.

cellWidth()
No description provided.

xLength()
No description provided.

yLength()
No description provided.

pMap()
No description provided.

pSatLimit()
No description provided.

pSatLimit(value)
No description provided.

pPrior()
No description provided.

pPrior(value)
No description provided.

lPrior()
No description provided.

create_grid(xLength, yLength, cellWidth)
No description provided.

reset()
No description provided.

```
update(x, y, th, *args, **kwargs)
```
No description provided.

```
xy_to_ij(x, y)
```
No description provided.

```
xy_to_ij_rounded(x, y)
```
No description provided.

```
ij_to_xy(i, j)
```
No description provided.

```
xy_to_grid(x, y)
```
No description provided.

RollingOccupancyGrid

No description provided.

```
__init__(radius=0, cellWidth=0, sensor=None, pPrior=0.5, pSatLimit=0.001,
useGPU=False)
```
No description provided.

```
update(dx, dy, th, *args, **kwargs)
```
No description provided.

OGSensorModel

No description provided.

```
__init__()
```
No description provided.

```
update_patch()
```
No description provided.

OGExternalModel

No description provided.

```
__init__(m, n)
```
No description provided.

```
update_patch(patch)
```
No description provided.

OGLidarModel

No description provided.

__init__(fov, rMax, phiRes, rRes, cellWidth, filter=True)
No description provided.

fov()
No description provided.

rMax()
No description provided.

phiRes()
No description provided.

rRes()
No description provided.

nPatch()
No description provided.

pPrior()
No description provided.

pPrior(value)
No description provided.

lPrior()
No description provided.

pOcc()
No description provided.

pOcc(value)
No description provided.

pFree()
No description provided.

pFree(value)
No description provided.

create_patches(fov, rMax, phiRes, rRes, cellWidth)
No description provided.

_precompute_patch_accessories()

No description provided.

_update_polarPatch(r)
No description provided.

update_patch(dx, dy, th, angles, distances)
No description provided.

OGRPLidarModel

No description provided.

__init__(patchCellWidth, filter=True)
No description provided.

update_patch(dx, dy, th, angles, distances)
No description provided.

filter_rplidar_data(angles, distances)
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/__init__.py

Low-level libraries for interacting with Quanser product in python

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/products/traffic_light.py

This script provides a user-friendly library to interact with traffic lights. It establishes a connection to a Raspberry Pi Zero W running a micro-controller that controls the traffic light LEDs. The library offers functionalities to control the traffic lights manually (setting them to red, yellow, or green) or switch them to an automatic timed mode. It also allows for turning off the lights and starting or stopping a stream.

Classes

TrafficLight

No description provided.

__init__(ip)
Connection to the lights

status()
Check and return the status of LEDs in use

shutdown()
Shutdown the Traffic Light

auto()
Set the Traffic lights to automatic mode

red()
Set Traffic Light to Red

yellow()
Set Traffic Light to Yellow

green()
Set Traffic Light to Green

color(color)
Function to set one of the traffic light LEDs on. Can only have one color ON at a time.

timed(red=30, yellow=3, green=30)
Set custom timed cycle for the Traffic Lights LEDs

off()
Turn the LEDs off without shutting down

start_stream()
Connect to Matlab/Simulink using QUARC Streaming API

stop_stream()
Reconnects back to Python to use TrafficLight functions

isStreaming()
Check if the Streaming connection is Open

_sendreq(url)
No description provided.

pal/resources/images.py

No description provided.

pal/resources/rtmodels.py

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/scope.py

scope.py: A module providing classes for real-time plotting and visualization.

Classes

_ScopeInfo

No description provided.

__init__(kwargs)**
No description provided.

_AxisInfo

No description provided.

__init__(kwargs)**
No description provided.

_XYAxisInfo

No description provided.

__init__(kwargs)**
No description provided.

_TYAxisInfo

No description provided.

__init__(kwargs)**
No description provided.

_SignalInfo

No description provided.

__init__(kwargs)**
No description provided.

_ImageInfo

No description provided.

__init__(kwargs)**
No description provided.

MultiScope

MultiScope: A class for creating and a scope with multiple axes.

__init__(*args, **kwargs)
Initialize a MultiScope instance.

_cells_available(row, col, rowSpan, colSpan)
No description provided.

_populate_cells(axis)
No description provided.

_addAxis(axisInfo)
No description provided.

addAxis(*args, **kwargs)
Add a time-y axis to the MultiScope instance.

addXYAxis(*args, **kwargs)
Add an x-y axis to the MultiScope instance.

refresh()
Refresh the MultiScope instance, updating contained axes.

refreshAll()
Refresh all active MultiScope instances, updating all scope plots.

Axis

Axis: A base class for creating and managing scope plot axes

__init__(graphicsLayoutWidget, bufferSize, **kwargs)
Initialize an Axis instance.

attachSignal(*args, **kwargs)

Attach a signal to the Axis instance.

sample(t, data)
Sample data for the Axis instance.

clean()
No description provided.

clear()
Clear the signals in the Axis instance.

_initial_refresh(flush)
No description provided.

_post_initial_refresh(flush)
No description provided.

xLabel()
str: The label for the x-axis of the Axis instance.

xLabel(newXLabel)
No description provided.

yLabel()
str: The label for the y-axis of the Axis instance.

yLabel(newYLabel)
No description provided.

xLim()
tuple: The limits of the x-axis in the form (min, max). Set to None to enable auto-ranging.

xLim(newXLimits)
No description provided.

yLim()
tuple: The limits of the y-axis in the form (min, max). Set to None to enable auto-ranging.

yLim(newYLimits)
No description provided.

XYAxis

XYAxis: A class for creating and managing an x-y axis in a MultiScope.

__init__(kwargs)**

Initialize an XYAxis instance.

attachSignal(*args, **kwargs)
Attach a 2D signal to the axis.

attachImage(*args, **kwargs)
Attach an image to the axis.

_post_initial_refresh(flush)
No description provided.

TYAxis

TYAxis: A class for creating and managing a time-y axis in a MultiScope.

__init__(kwargs)**
Initialize a TYAxis instance.

attachSignal(*args, **kwargs)
Attach a signal to the TYAxis instance.

_post_initial_refresh(flush)
No description provided.

Signal

No description provided.

__init__(plot, *args, **kwargs)
No description provided.

add_chunk(x, y)
No description provided.

clear()
No description provided.

Image

Image: A class for containing and managing images to be plotted

__init__(plot, *args, **kwargs)
Initialize an Image instance.

setImage(image)
Set the image data for the Image instance.

scale()
Get the scale of the image.

scale(newScale)
Set the scale of the image.

offset()
Get the offset of the image.

offset(newOffset)
Set the offset of the image.

rotation()
Get the rotation of the image.

rotation(newRotation)
Set the rotation of the image.

levels()
Get the color levels of the image.

levels(newLevels)
Set the color levels of the image.

Scope

Scope: A class for real-time plotting of 1D signals.

__init__(title=None, **kwargs)
Initialize a Scope instance.

attachSignal(kwargs)**
Attach a signal to the Scope instance for plotting.

sample(t, data)
Provide a new data sample for the attached signals.

clean()
No description provided.

clear()
No description provided.

refresh()
Refresh the Scope instance to update the plot with the latest data.

refreshAll()
Refresh all the active Scope instances.

XYScope

XYScope: A class for real-time display of 2D signals and images.

__init__(title=None, **kwargs)
Initialize an XYScope instance.

attachSignal(kwargs)**
Attach a signal to the XYScope instance for plotting.

attachImage(*args, **kwargs)
Attach an image to the XYScope instance for display.

sample(t, data)
Provide a new data sample for the attached signals.

clear()
No description provided.

refresh()
Refresh the display of the XYScope to update the signals and images.

refreshAll()
Refresh all the active Scope instances.

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/timing.py

No description provided.

Classes

Timer

A class for timing your loops using sleep methods that account for both computation time as well as sleep overhead.

__init__(sampleRate, totalTime, verbose=False)
No description provided.

_restart()
Used to restart the simulation time from 0.

check()
Resets internal parameters for next iteration. Use this method as a condition on your while loop.

get_current_time()
Returns the current time (absolute).

get_sample_time()
Returns the sample time (relative).

sleep()
Sleeps as required to catch up to an absolute clock.

QTimer

A class for timing your loops using busy wait methods that account for both computation time as well as sleep overhead.

__init__(sampleRate, totalTime)
No description provided.

_restart()
Used to restart the simulation time from 0.

check()
Resets internal parameters for next iteration. Use this method as a condition on your while loop.

get_current_time()
Returns the current time (absolute).

get_sample_time()
Returns the sample time (relative).

sleep(verbose=False)
Sleeps until the next Timeout expires.

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