MITLL RACECAR Documentation

Harvey Mudd College, MIT

Mar 04, 2021

CONTENTS:

This website contains documentation for the `racecar_core` and `racecar_utils` libraries. These libraries are part of the RACECAR-MN platform.

CHAPTER

ONE

RACECAR_CORE LIBRARY

The `racecar_core` library exposes the intended interface for programming the RACECAR-MN. The Main module handles program execution, and several submodules encapsulate different aspects of the hardware.

1.1 Camera Module

class camera.**Camera**
Returns the color images and depth images captured by the camera.

get_color_image()
Returns a deep copy of the current color image captured by the camera.

Return type NDArray[(480, 640, 3), uint8]

Returns

An array representing the pixels in the image, organized as follows 0th dimension: pixel rows, indexed from top to bottom. 1st dimension: pixel columns, indexed from left to right. 2nd dimension: pixel color channels, in the blue-green-red format.

Note: Each color value ranges from 0 to 255.

This function returns a deep copy of the captured image. This means that we can modify the returned image and it will not change the image returned by future calls to get_color_image().

Example:

```
# Initialize image with a deep copy of the most recent color image captured
# by the camera
image = rc.camera.get_color_image()

# Store the amount of blue in the pixel on row 3, column 5
blue = image[3][5][0]
```

abstract get_color_image_async()
Returns the current color image without the car in "go" mode.

Return type NDArray[(480, 640, 3), uint8]

Returns

An array representing the pixels in the image, organized as follows 0th dimension: pixel rows, indexed from top to bottom. 1st dimension: pixel columns, indexed from left to right. 2nd dimension: pixel color channels, in the blue-green-red format.

Note: Each color value ranges from 0 to 255.

Warning: This function breaks the start-update paradigm and should only be used in Jupyter Notebook.

Example:

```
# Initialize image with the most recent color image captured by the camera
image = rc.camera.get_color_image_async()

# Store the amount of blue in the pixel on row 3, column 5
blue = image[3][5][0]
```

abstract get_color_image_no_copy()

Returns a direct reference to the color image captured by the camera.

Return type NDArray[(480, 640, 3), uint8]

Returns

An array representing the pixels in the image, organized as follows 0th dimension: pixel rows, indexed from top to bottom. 1st dimension: pixel columns, indexed from left to right. 2nd dimension: pixel color channels, in the blue-green-red format.

Warning: Do not modify the returned image. The returned image is a reference to the captured image, so any changes will also affect the images returned by any future calls to get_color_image() or get_color_image_no_copy().

Note: Each color value ranges from 0 to 255.

Unlike get_color_image(), this function does not create a deep copy of the captured image, and is therefore more efficient.

Example:

```
# Initialize image with a direct reference to the recent color image
# captured by the camera
image = rc.camera.get_color_image()

# Store the amount of blue in the pixel on row 3, column 5
blue = image[3][5][0]

# We can safely call crop because it does not modify the source image
cropped_image = rc_utils.crop(image, (0, 0), (10, 10))

# However, if we wish to draw on the image, we must first create a manual
# copy with copy.deepcopy()
image_copy = copy.deepcopy(image)
modified_image = rc_utils.draw_circle(image_copy, (50, 50))
```

abstract get_depth_image()

Returns the current depth image captured by the camera.

Return type NDArray[(480, 640), float32]

Returns A two dimensional array indexed from top left to the bottom right storing the distance of each pixel from the car in cm.

Example:

```
# Initialize image with the most recent depth image captured by the camera
image = rc.camera.get_depth_image()

# Store the distance of the object at pixel row 3, column 5
distance = image[3][5]
```

abstract get_depth_image_async()

Returns the current depth image without the car in "go" mode.

Return type NDArray[(480, 640), float32]

Returns A two dimensional array indexed from top left to the bottom right storing the distance of each pixel from the car in cm.

> **Warning:** This function breaks the start-update paradigm and should only be used in Jupyter Notebook.

Example:

```
# Initialize image with the most recent depth image captured by the camera
image = rc.camera.get_depth_image_async()

# Store the distance of the object at pixel row 3, column 5
distance = image[3][5]
```

get_height()

Returns the pixel height of the color and depth images.

Return type int

Returns The height (number of pixel rows) in the color and depth images.

Example:

```
image = rc.camera.get_color_image()

# Access the top bottom left pixel of the image
bottom_left_pixel = image[rc.camera.get_height() - 1][0]
```

get_max_range()

Returns the maximum distance in cm which can be detected by the depth camera.

Return type float

Returns The maximum range of the depth camera.

Example:

```
depth_image = rc.camera.get_depth_image()
center_distance = rc_utils.get_depth_image_center_distance(depth_image)

# If center_distance is 0.0 (no data), set it to max_range
```

(continues on next page)

(continued from previous page)

```
if center_distance == 0.0:
    center_distance = rc.camera.get_max_range()
```

get_width()
Returns the pixel width of the color and depth images.

Return type int

Returns The width (number of pixel columns) in the color and depth images.

Example:

```
image = rc.camera.get_color_image()

# Access the top right pixel of the image
top_right_pixel = image[0][rc.camera.get_width() - 1]
```

1.2 Controller Module

class controller.**Controller**
Handles input from the controller and exposes constant input state per frame.

class Button(*value*)
The buttons on the controller.

class Joystick(*value*)
The joysticks on the controller.

class Trigger(*value*)
The triggers on the controller.

abstract get_joystick(*joystick*)
Returns the position of a certain joystick as an (x, y) tuple.

Parameters **joystick** (*Joystick*) – Which joystick to check.

Return type Tuple[float, float]

Returns The x and y coordinate of the joystick, with each axis ranging from -1.0 (left or down) to 1.0 (right or up).

Note: The joystick argument must be an associated value of the Joystick enum, which is defined in the Controller module.

Example:

```
# x and y will be given values from -1.0 to 1.0 based on the position of
# the left joystick
(x, y) = rc.controller.get_joystick(rc.controller.Joystick.LEFT)
```

abstract get_trigger(*trigger*)
Returns the position of a certain trigger as a value from 0.0 to 1.0.

Parameters **trigger** (*Trigger*) – Which trigger to check.

Return type float

Returns A value ranging from 0.0 (not pressed) to 1.0 (fully pressed) inclusive.

Note: The trigger argument must be an associated value of the Trigger enum, which is defined in the Controller module.

Example:

```
# Speed will receive a value from 0.0 to 1.0 based on how much the left
# trigger is pressed
speed = rc.controller.get_trigger(rc.controller.Trigger.LEFT)
```

abstract is_down(*button*)

Returns whether a certain button is currently pressed.

Parameters **button** (*Button*) – Which button to check.

Return type bool

Returns True if button is currently pressed.

Note: The button argument must be an associated value of the Button enum, which is defined in the Controller module.

Example:

```
# This update function will print a message for every frame in which
# the A button is held down.  Thus, multiple messages will be printed
# if we press and hold the A button
def update():
    if rc.controller.is_down(rc.controller.Button.A):
        print("The A button is currently pressed.")
```

abstract was_pressed(*button*)

Returns whether a certain button was pressed this frame.

Parameters **button** (*Button*) – Which button to check.

Return type bool

Returns True if button is currently pressed and was not pressed last frame.

Note: The button argument must be an associated value of the Button enum, which is defined in the Controller module.

Example:

```
# This update function will print a single message each time the A
# button is pressed on the controller
def update():
    if rc.controller.was_pressed(rc.controller.Button.A):
        print("The A button was pressed")
```

abstract was_released(*button*)

Returns whether a certain button was released this frame.

Parameters **button** (*Button*) – Which button to check.

Return type `bool`

Returns True if button is currently released and was pressed last frame.

Note: The button argument must be an associated value of the Button enum, which is defined in the Controller module.

Example:

```
# This update function will print a single message each time the A
# button is released on the controller
def update():
    if rc.controller.was_pressed(rc.controller.Button.A):
        print("The A button was released")
```

1.3 Display Module

class display.**Display**(*isHeadless*)

Allows the user to print images to the screen.

abstract create_window()

Creates an empty window into which images will be displayed.

Note: It is not necessary to call create_window before any of the other display methods (show_color_image, show_depth_image, etc.). These methods will automatically create a new window if one was not already created.

Example:

```
# Creates a window
rc.camera.create_window()

# Display an image in this window
image = rc.camera.get_color_image()
rc.display.show_color_image(image)
```

Return type `None`

abstract show_color_image(*image*)

Displays a color image in a window.

Parameters **image** (`NDArray`) – The color image to display to the the screen.

Example:

```
image = rc.camera.get_color_image()

# Show the image captured by the camera
rc.display.show_color_image(image)
```

Return type `None`

show_depth_image(*image*, *max_depth=1000*, *points=[]*)
Displays a depth image in grayscale in a window.

Parameters

- **image** (NDArray[(typing.Any, typing.Any), float32]) – The depth image to display to the screen.
- **max_depth** (int) – The farthest depth to show in the image in cm. Anything past this depth is shown as black.
- **points** (List[Tuple[int, int]]) – A list of points in (pixel row, pixel column) format to show on the image as colored dots.

Example:

```
depth_image = rc.camera.get_depth_image()

# Show the depth_image captured by the camera.
rc.display.show_depth_image(depth_image)

# Show anything that is at most 500 cm away, and show a black cross at
# row 3, column 5
rc.display.show_depth_image(depth_image, 500, [(3, 5)])
```

Return type None

show_lidar(*samples*, *radius=128*, *max_range=1000*, *highlighted_samples=[]*)
Displays a set of LIDAR samples.

Parameters

- **samples** (NDArray[Any, float32]) – A complete LIDAR scan.
- **radius** (int) – Half of the width or height (in pixels) of the generated image.
- **max_range** (int) – The farthest depth to show in the image in cm. Anything past this depth is shown as black.
- **highlighted_samples** (List[Tuple[float, float]]) – A list of samples in (angle, distance) format to show as light blue dots. Angle must be in degrees from straight ahead (clockwise), and distance must be in cm.

Note: Each sample in samples is shown as a red pixel. Each sample in highlighted_samples is shown as a blue pixel. The car is shown as a green dot at the center of the visualization.

Warning: samples must be a complete LIDAR scan. This function assumes that each sample is equal angle appart, and that samples spans the entire 360 degrees. If this is not the case, the visualization will be inaccurate.

Example:

```
lidar_scan = rc.lidar.get_samples()

# Show the lidar scan
rc.display.show_lidar(lidar_scan)
```

(continues on next page)

(continued from previous page)

```
# Show the lidar scan out to 500 cm with the closest point highlighted
closest_point = rc_utils.get_lidar_closest_point(lidar_scan)
rc.display.show_lidar(lidar_scan, 500, [closest_point])
```

Return type None

1.4 Drive Module

class drive.**Drive**

abstract set_max_speed(*max_speed=0.25*)
Sets the maximum throttle in the forward and backward direction.

Parameters **max_speed** (float) – The scale factor applied to speed inputs, ranging from 0.0 to 1.0.

Warning: The RACECAR contains expensive and fragile equipment. Please only increase The max speed if you are in a safe environment without the potential for hard collisions.

Example:

```
# Update the max speed to 0.5
rc.set_max_speed(0.5)
```

Return type None

abstract set_speed_angle(*speed*, *angle*)
Sets the throttle applied to the back wheels and the angle of the front wheels.

Parameters

- **speed** (float) – The amount of throttle (torque) applied to the back wheels from -1.0 (full backward) to 1.0 (full forward).
- **angle** (float) – The amount to turn the front wheels from -1.0 (full left) to 1.0 (full right).

Note: The speed and angle arguments are unitless ratios.

Example:

```
if counter < 1:
    # Drive straight forward at full speed
    rc.drive.set_speed_angle(1, 0)
elif counter < 2:
    # Drive reverse at full speed with the wheels pointing fully to the left
    rc.drive.set_speed_angle(-1, -1)
else:
```

(continues on next page)

(continued from previous page)

```
    # Drive 70% to the right at half speed forward
    rc.drive.set_speed_angle(0.5, 0.7)
```

Return type None

stop()
Brings the car to a stop and points the front wheels forward.

Note: stop is equivalent to rc.drive.set_speed_angle(0, 0)

Example:

```
# Stops the car if the counter is greater than 5
if counter > 5:
    rc.drive.stop()
```

Return type None

1.5 GPIO Module

This module is currently in development.

1.6 LIDAR Module

class lidar.**Lidar**
Returns the scan data captured by the Lidar.

get_num_samples()
Returns the number of samples in a full LIDAR scan.

Return type int

Returns The number of points collected in a complete scan.

Example:

```
scan = rc.lidar.get_samples()

# Access the sample directly behind the car
rear_distance = scan[rc.lidar.get_num_samples() // 2]
```

abstract get_samples()
Returns the current LIDAR scan as an array of distance measurements.

Return type NDArray[720, float32]

Returns An array of distance measurements in cm.

Note: Samples are in clockwise order, with the 0th sample directly in front of the car. Each sample is an equal angle appart.

Example:

```
# Access the most recent lidar scan.
scan = rc.lidar.get_samples()

# Get the distance of the measurement directly in front of the car
forward_distance = scan[0]
```

abstract get_samples_async()

Returns the current LIDAR scan without the car in "go" mode.

Return type NDArray[720, float32]

Returns An array of distance measurements in cm.

Note: Samples are in clockwise order, with the 0th sample directly in front of the car. Each sample is an equal angle appart.

Warning: This function breaks the start-update paradigm and should only be used in Jupyter Notebook.

Example:

```
# Access the most recent lidar scan.
scan = rc.lidar.get_samples_async()

# Get the distance of the measurement directly in front of the car
forward_distance = lidar_ranges[0]
```

1.7 Physics Module

class physics.**Physics**

Returns the linear acceleration and angular velocity measured by the IMU.

abstract get_angular_velocity()

Returns a 3D vector containing the car's angular velocity.

Return type NDArray[3, float32]

Returns The average angular velocity of the car along the (x, y, z) axes during the last frame in rad/s.

Note: The x axis (pitch) points out of the right of the car. The y axis (yaw) points directly up (perpendicular to the ground). The z axis (roll) points out of the front of the car. Rotation sign uses the right hand rule. For example, when the car turns to the left, it has a positive angular velocity along the y axis.

Example:

```
# ang_vel stores the average angular velocity over the previous frame
ang_vel = rc.physics.get_angular_velocity()
```

(continues on next page)

(continued from previous page)

```
# yaw stores the yaw of the car, which is positive when it turns to the left
# and negative when it turns to the right.
yaw = ang_vel[1]
```

abstract get_linear_acceleration()

Returns a 3D vector containing the car's linear acceleration.

Return type NDArray[3, float32]

Returns The average linear acceleration of the car along the (x, y, z) axes during the last frame in m/s^2.

Note: The x axis points out of the right of the car. The y axis points directly up (perpendicular to the ground). The z axis points out of the front of the car.

Example:

```
# accel stores the average acceleration over the previous frame
accel = rc.physics.get_linear_acceleration()

# forward accel stores acceleration in the forward direction.  This will be
# positive when the car accelerates, and negative when it decelerates.
forward_accel = accel[2]
```

1.8 Sound Module

This module is currently in development.

1.9 Racecar Module

class racecar_core.**Racecar**

The top level racecar module containing several submodules which interface with and control the different pieces of the RACECAR hardware.

abstract get_delta_time()

Returns the number of seconds elapsed in the previous frame.

Return type float

Returns The number of seconds between the start of the previous frame and the start of the current frame.

Example:

```
# Increases counter by the number of seconds elapsed in the previous frame
counter += rc.get_delta_time()
```

abstract go()

Starts the RACECAR, beginning in default drive mode.

Note: go idles blocks execution until the program is exited when START + END are pressed simultaneously.

Return type None

abstract set_start_update(*start*, *update*, *update_slow=None*)
Sets the start and update functions used in user program mode.

Parameters

- **start** (Callable[[], None]) – A function called once when the car enters user program mode.
- **update** (Callable[[], None]) – A function called every frame in user program mode. Approximately 60 frames occur per second.
- **update_slow** (Optional[Callable[[], None]]) – A function called once per fixed time interval in user program mode (by default once per second).

Note: The provided functions should not take any parameters.

Example:

```
# Create a racecar object
rc = Racecar()

# Define a start function
def start():
    print("This function is called once")

# Define an update function
def update():
    print("This function is called every frame")

# Provide the racecar with the start and update functions
rc.set_start_update(start, update)

# Tell the racecar to run until the program is exited
rc.go()
```

Return type None

abstract set_update_slow_time(*time=1.0*)
Changes the time between calls to update_slow.

Parameters **time** (float) – The time in seconds between calls to update_slow.

Example:

```
# Sets the time between calls to update_slow to 2 seconds
rc.set_update_slow_time(2)
```

Return type None

CHAPTER

TWO

RACECAR_UTILS LIBRARY

The `racecar_utils` library provides helper functions which aid in performing common robotics tasks. Many of these functions are designed to interface with the inputs and outputs of the `racecar_core` library.

Contains helper functions to support common operations.

class racecar_utils.**ARMarker**(*marker_id*, *marker_corners*)

Encapsulates information about an AR marker detected in a color image.

detect_colors(*color_image*, *potential_colors*)

Attempts to detect the provided colors in the border around the AR marker.

Parameters

- **color_image** (NDArray[(typing.Any, typing.Any), float32]) – The image in which the marker was detected.
- **potential_colors** (List[Tuple[Tuple[int, int, int], Tuple[int, int, int], str]]) – A list of colors which the marker border may be. Each candidate color is formated as (hsv_lower, hsv_upper, color_name).

Example:

```
# Define color candidates in the (hsv_lower, hsv_upper, color_name) format
BLUE = ((90, 100, 100), (120, 255, 255), "blue")
RED = ((170, 100, 100), (10, 255, 255), "red")

# Detect the AR markers in the current color image
image = rc.camera.get_color_image()
markers = rc_utils.get_ar_markers(image)

# Search for the colors RED and BLUE in all of the detected markers
for marker in markers:
    marker.detect_colors(image, [BLUE, RED])
```

Return type None

get_color()

Returns the color of the marker if it was successfully detected.

Return type str

get_corners()

Returns the (row, col) coordinates of the four corners of the marker.

Note: The corners are ordered clockwise with the top-left corner of the pattern appearing first.

Return type NDArray[(4, 2), int32]

get_corners_aruco_format()
Returns the corners of the AR marker formatted as needed by the ArUco library.

Return type NDArray[(1, 4, 2), float32]

get_id()
Returns the integer identification number of the marker pattern.

Return type int

get_orientation()
Returns the orientation of the marker.

Return type *Orientation*

class racecar_utils.**ColorBGR**(*value*)
Common colors defined in the blue-green-red (BGR) format, with each channel ranging from 0 to 255 inclusive.

class racecar_utils.**Orientation**(*value*)
The orientations which an AR marker can face, with the value indicating the index of the corner which is currently oriented in the top-left in the image.

class racecar_utils.**TerminalColor**(*value*)
Colors which can be used when printing text to the terminal, with each value corresponding to the ASCII code for that color.

racecar_utils.**clamp**(*value*, *min*, *max*)
Clamps a value between a minimum and maximum value.

Parameters

- **value** (float) – The input to clamp.
- **min** (float) – The minimum allowed value.
- **max** (float) – The maximum allowed value.

Return type float

Returns The value saturated between min and max.

Example:

```
# a will be set to 3
a = rc_utils.clamp(3, 0, 10)

# b will be set to 0
b = rc_utils.remap_range(-2, 0, 10)

# c will be set to 10
c = rc_utils.remap_range(11, 0, 10)
```

racecar_utils.**colormap_depth_image**(*depth_image*, *max_depth=1000*)
Converts a depth image to a colored image representing depth.

Parameters

- **depth_image** (NDArray[(typing.Any, typing.Any), float32]) – The depth image to convert.
- **max_depth** (int) – The farthest depth to show in the image in cm. Anything past this depth is shown as the farthest color.

Return type NDArray[(typing.Any, typing.Any, 3), uint8]

Returns A color image representation of the provided depth image.

Note: Each color value ranges from 0 to 255. The color of each pixel is determined by its distance.

Example:

```
# retrieve a depth image
depth_image = rc.camera.get_depth_image()

# get the colormapped depth image
depth_image_colormap = rc_utils.colormap_depth_image(depth_image)
```

racecar_utils.**crop**(*image*, *top_left_inclusive*, *bottom_right_exclusive*)

Crops an image to a rectangle based on the specified pixel points.

Parameters

- **image** (NDArray[(typing.Any, Ellipsis), Any]) – The color or depth image to crop.
- **top_left_inclusive** (Tuple[float, float]) – The (row, column) of the top left pixel of the crop rectangle.
- **bottom_right_exclusive** (Tuple[float, float]) – The (row, column) of the pixel one past the bottom right corner of the crop rectangle.

Return type NDArray[(typing.Any, Ellipsis), Any]

Returns A cropped version of the image.

Note: The top_left_inclusive pixel is included in the crop rectangle, but the bottom_right_exclusive pixel is not.

If bottom_right_exclusive exceeds the bottom or right edge of the image, the full image is included along that axis.

Example:

```
image = rc.camera.get_color_image()

# Crop the image to only keep the top half
cropped_image = rc_utils.crop(
    image, (0, 0), (rc.camera.get_height() // 2, rc.camera.get_width())
)
```

racecar_utils.**draw_ar_markers**(*color_image*, *markers*, *color=(0, 255, 0)*)

Draws annotations on the AR markers in a image.

Parameters

- **color_image** (NDArray[(typing.Any, typing.Any, 3), uint8]) – The color image in which the AR markers were detected.

- **markers** (List[*ARMarker*]) – The AR markers detected in the image.
- **color** (Tuple[int, int, int]) – The color used to outline each AR marker, represented in the BGR format.

Warning: This modifies the provided image. If you accessed the image with rc.camera.get_color_image_no_copy(), you must manually create a copy of the image first with copy.deepcopy().

Example:

```
# Detect the AR markers in the current color image
image = rc.camera.get_color_image()
markers = rc_utils.get_ar_markers(image)

# Draw the detected markers an the image and display it
rc_utils.draw_ar_markers(image, markers)
rc.display.show_color_image(color_image)
```

Return type NDArray[(typing.Any, typing.Any, 3), uint8]

racecar_utils.**draw_circle**(*color_image*, *center*, *color=(0, 255, 255)*, *radius=6*)

Draws a circle on the provided image.

Parameters

- **color_image** (NDArray[(typing.Any, typing.Any, 3), uint8]) – The color image on which to draw the contour.
- **center** (Tuple[int, int]) – The pixel (row, column) of the center of the image.
- **color** (Tuple[int, int, int]) – The color to draw the circle, specified as blue-green-red channels each ranging from 0 to 255 inclusive.
- **radius** (int) – The radius of the circle in pixels.

Example:

```
image = rc.camera.get_color_image()

# Extract the largest blue contour
BLUE_HSV_MIN = (90, 50, 50)
BLUE_HSV_MAX = (110, 255, 255)
contours = rc_utils.find_contours(image, BLUE_HSV_MIN, BLUE_HSV_MAX)
largest_contour = rc_utils.get_largest_contour(contours)

# Draw a dot at the center of this contour in red
if (largest_contour is not None):
    center = get_contour_center(contour)
    draw_circle(image, center, rc_utils.ColorBGR.red.value)
```

Return type None

racecar_utils.**draw_contour**(*color_image*, *contour*, *color=(0, 255, 0)*)

Draws a contour on the provided image.

Parameters

- **color_image** (NDArray[(typing.Any, typing.Any, 3), uint8]) – The color image on which to draw the contour.
- **contour** (NDArray) – The contour to draw on the image.
- **color** (Tuple[int, int, int]) – The color to draw the contour, specified as blue-green-red channels each ranging from 0 to 255 inclusive.

Example:

```
image = rc.camera.get_color_image()

# Extract the largest blue contour
BLUE_HSV_MIN = (90, 50, 50)
BLUE_HSV_MAX = (110, 255, 255)
contours = rc_utils.find_contours(image, BLUE_HSV_MIN, BLUE_HSV_MAX)
largest_contour = rc_utils.get_largest_contour(contours)

# Draw this contour onto image
if (largest_contour is not None):
    draw_contour(image, largest_contour)
```

Return type None

racecar_utils.**find_contours**(*color_image*, *hsv_lower*, *hsv_upper*)

Finds all contours of the specified color range in the provided image.

Parameters

- **color_image** (NDArray[(typing.Any, typing.Any, 3), uint8]) – The color image in which to find contours, with pixels represented in the bgr (blue-green-red) format.
- **hsv_lower** (Tuple[int, int, int]) – The lower bound for the hue, saturation, and value of colors to contour.
- **hsv_upper** (Tuple[int, int, int]) – The upper bound for the hue, saturation, and value of the colors to contour.

Return type List[NDArray]

Returns A list of contours around the specified color ranges found in color_image.

Note: Each channel in hsv_lower and hsv_upper ranges from 0 to 255.

Example:

```
# Define the lower and upper hsv ranges for the color blue
BLUE_HSV_MIN = (90, 50, 50)
BLUE_HSV_MAX = (110, 255, 255)

# Extract contours around all blue portions of the current image
contours = rc_utils.find_contours(
    rc.camera.get_color_image(), BLUE_HSV_MIN, BLUE_HSV_MAX
)
```

racecar_utils.**format_colored**(*text*, *color*)

Formats a string so that it is printed to the terminal with a specified color.

Parameters

- **text** (str) – The text to format.
- **color** (*TerminalColor*) – The color to print the text.

Example:

```
# Prints "Hello World!", where "World" is blue
print("Hello " + format_colored("World", rc_utils.TerminalColor.blue) + "!")
```

Return type None

racecar_utils.**get_ar_markers**(*color_image*, *potential_colors=None*)

Finds AR markers in a image.

Parameters

- **color_image** (NDArray[(typing.Any, typing.Any, 3), uint8]) – The color image in which to search for AR markers.
- **potential_colors** (Optional[List[Tuple[Tuple[int, int, int], Tuple[int, int, int], str]]]) – The potential colors of the AR marker, each represented as (hsv_min, hsv_max, color_name)

Return type List[*ARMarker*]

Returns A list of each AR marker's four corners clockwise and an array of the AR marker ids.

Example:

```
# Detect the AR markers in the current color image
image = rc.camera.get_color_image()
markers = racecar_utils.get_ar_markers(image)

# Print information detected for the zeroth marker
if len(markers) >= 1:
    print(markers[0])
```

racecar_utils.**get_closest_pixel**(*depth_image*, *kernel_size=5*)

Finds the closest pixel in a depth image.

Parameters

- **depth_image** (NDArray[(typing.Any, typing.Any), float32]) – The depth image to process.
- **kernel_size** (int) – The size of the area to average around each pixel.

Return type Tuple[int, int]

Returns The (row, column) of the pixel which is closest to the car.

Warning: kernel_size be positive and odd. It is highly recommended that you crop off the bottom of the image, or else this function will likely return the ground directly in front of the car.

Note: The larger the kernel_size, the more that the depth of each pixel is averaged with the distances of the surrounding pixels. This helps reduce noise at the cost of reduced accuracy.

Example:

```
depth_image = rc.camera.get_depth_image()

# Crop off the ground directly in front of the car
cropped_image = rc_utils.crop(
    image, (0, 0), (int(rc.camera.get_height() * 0.66), rc.camera.get_width())
)

# Find the closest pixel
closest_pixel = rc_utils.get_closest_pixel(depth_image)
```

racecar_utils.**get_contour_area**(*contour*)

Finds the area of a contour from an image.

Parameters **contour** (NDArray) – The contour of which to measure the area.

Return type float

Returns The number of pixels contained within the contour

Example:

```
# Extract the largest blue contour
BLUE_HSV_MIN = (90, 50, 50)
BLUE_HSV_MAX = (110, 255, 255)
contours = rc_utils.find_contours(
    rc.camera.get_color_image(), BLUE_HSV_MIN, BLUE_HSV_MAX
)
largest_contour = rc_utils.get_largest_contour(contours)

# Find the area of this contour (will evaluate to 0 if no contour was found)
area = rc_utils.get_contour_area(contour)
```

racecar_utils.**get_contour_center**(*contour*)

Finds the center of a contour from an image.

Parameters **contour** (NDArray) – The contour of which to find the center.

Return type Optional[Tuple[int, int]]

Returns The (row, column) of the pixel at the center of the contour, or None if the contour is empty.

Example:

```
# Extract the largest blue contour
BLUE_HSV_MIN = (90, 50, 50)
BLUE_HSV_MAX = (110, 255, 255)
contours = rc_utils.find_contours(
    rc.camera.get_color_image(), BLUE_HSV_MIN, BLUE_HSV_MAX
)
largest_contour = rc_utils.get_largest_contour(contours)

# Find the center of this contour if it exists
if (largest_contour is not None):
    center = rc_utils.get_contour_center(largest_contour)
```

racecar_utils.**get_depth_image_center_distance**(*depth_image*, *kernel_size=5*)

Finds the distance of the center object in a depth image.

Parameters

- **depth_image** (NDArray[(typing.Any, typing.Any), float32]) – The depth image to process.
- **kernel_size** (int) – The size of the area to average around the center.

Return type float

Returns The distance in cm of the object in the center of the image.

Warning: kernel_size must be positive and odd.

Note: The larger the kernel_size, the more that the center is averaged with the depth of the surrounding pixels. This helps reduce noise at the cost of reduced accuracy. If kernel_size = 1, no averaging is done.

Example:

```
depth_image = rc.camera.get_depth_image()

# Find the distance of the object (in cm) the center of depth_image
center_distance = rc_utils.get_depth_image_center_distance(depth_image)
```

racecar_utils.**get_largest_contour**(*contours*, *min_area=30*)

Finds the largest contour with size greater than min_area.

Parameters

- **contours** (List[NDArray]) – A list of contours found in an image.
- **min_area** (int) – The smallest contour to consider (in number of pixels)

Return type Optional[NDArray]

Returns The largest contour from the list, or None if no contour was larger than min_area.

Example:

```
# Extract the blue contours
BLUE_HSV_MIN = (90, 50, 50)
BLUE_HSV_MAX = (110, 255, 255)
contours = rc_utils.find_contours(
    rc.camera.get_color_image(), BLUE_HSV_MIN, BLUE_HSV_MAX
)

# Find the largest contour
largest_contour = rc_utils.get_largest_contour(contours)
```

racecar_utils.**get_lidar_average_distance**(*scan*, *angle*, *window_angle=4*)

Finds the average distance of the object at a particular angle relative to the car.

Parameters

- **scan** (NDArray[Any, float32]) – The samples from a LIDAR scan
- **angle** (float) – The angle (in degrees) at which to measure distance, starting at 0 directly in front of the car and increasing clockwise.
- **window_angle** (float) – The number of degrees to consider around angle.

Return type float

Returns The average distance of the points at angle in cm.

Note: Ignores any samples with a value of 0.0 (no data). Increasing window_angle reduces noise at the cost of reduced accuracy.

Example:

```
scan = rc.lidar.get_samples()

# Find the distance directly behind the car (6:00 position)
back_distance = rc_utils.get_lidar_average_distance(scan, 180)

# Find the distance to the forward and right of the car (1:30 position)
forward_right_distance = rc_utils.get_lidar_average_distance(scan, 45)
```

racecar_utils.**get_lidar_closest_point**(*scan*, *window=(0, 360)*)

Finds the closest point from a LIDAR scan.

Parameters

- **scan** (NDArray[Any, float32]) – The samples from a LIDAR scan.
- **window** (Tuple[float, float]) – The degree range to consider, expressed as (min_degree, max_degree)

Return type Tuple[float, float]

Returns The (angle, distance) of the point closest to the car within the specified degree window. All angles are in degrees, starting at 0 directly in front of the car and increasing clockwise. Distance is in cm.

Warning: In areas with glass, mirrors, or large open spaces, there is a high likelihood of distance error.

Note: Ignores any samples with a value of 0.0 (no data).

In order to define a window which passes through the 360-0 degree boundary, it is acceptable for window min_degree to be larger than window max_degree. For example, (350, 10) is a 20 degree window in front of the car.

Example:

```
scan = rc.lidar.get_samples()

# Find the angle and distance of the closest point
angle, distance = rc_utils.get_lidar_closest_point(scan)

# Find the closest distance in the 90 degree window behind the car
_, back_distance = rc_utils.get_lidar_closest_point(scan, (135, 225))

# Find the closest distance in the 90 degree window in front of the car
_, front_distance = rc_utils.get_lidar_closest_point(scan, (315, 45))
```

racecar_utils.**get_pixel_average_distance**(*depth_image*, *pix_coord*, *kernel_size=5*)

Finds the distance of a pixel averaged with its neighbors in a depth image.

Parameters

- **depth_image** (NDArray[(typing.Any, typing.Any), float32]) – The depth image to process.
- **pix_coord** (Tuple[int, int]) – The (row, column) of the pixel to measure.
- **kernel_size** (int) – The size of the area to average around the pixel.

Return type float

Returns The distance in cm of the object at the provided pixel.

> **Warning:** kernel_size must be positive and odd.

Note: The larger the kernel_size, the more that the requested pixel is averaged with the distances of the surrounding pixels. This helps reduce noise at the cost of reduced accuracy.

Example:

```
depth_image = rc.camera.get_depth_image()

# Find the distance of the object (in cm) at the pixel (100, 20) of depth_image
average_distance = rc_utils.get_average_distance(depth_image, 100, 20)
```

racecar_utils.**print_colored**(*text*, *color*)

Prints a line of text to the terminal with a specified color.

Parameters

- **text** (str) – The text to print to the terminal.
- **color** (*TerminalColor*) – The color to print the text.

Example:

```
rc_utils.print_colored("This will be black", rc_utils.TerminalColor.black)
rc_utils.print_colored("This will be red", rc_utils.TerminalColor.red)
rc_utils.print_colored("This will be green", rc_utils.TerminalColor.green)
```

Return type None

racecar_utils.**print_error**(*text*)

Prints a line of text to the terminal in red.

Parameters **text** (str) – The text to print to the terminal.

Example:

```
# This text will be printed to the terminal in red
rc_utils.print_error("Error: No image detected")
```

Return type None

racecar_utils.**print_warning**(*text*)

Prints a line of text to the terminal in yellow.

Parameters **text** (str) – The text to print to the terminal.

Example:

```
# This text will be printed to the terminal in yellow
rc_utils.print_warning("Warning: Potential collision detected, reducing speed")
```

Return type None

racecar_utils.**remap_range**(*val*, *old_min*, *old_max*, *new_min*, *new_max*, *saturate=False*)

Remaps a value from one range to another range.

Parameters

- **val** (float) – A number form the old range to be rescaled.
- **old_min** (float) – The inclusive 'lower' bound of the old range.
- **old_max** (float) – The inclusive 'upper' bound of the old range.
- **new_min** (float) – The inclusive 'lower' bound of the new range.
- **new_max** (float) – The inclusive 'upper' bound of the new range.
- **saturate** (bool) – If true, the new_min and new_max limits are enforced.

Note: min need not be less than max; flipping the direction will cause the sign of the mapping to flip. val does not have to be between old_min and old_max.

Example:

```
# a will be set to 25
a = rc_utils.remap_range(5, 0, 10, 0, 50)

# b will be set to 975
b = rc_utils.remap_range(5, 0, 20, 1000, 900)

# c will be set to 30
c = rc_utils.remap_range(2, 0, 1, -10, 10)

# d will be set to 10
d = rc_utils.remap_range(2, 0, 1, -10, 10, True)
```

Return type float

racecar_utils.**stack_images_horizontal**(*image_0*, *image_1*)

Stack two images horizontally.

Parameters

- **image_0** (NDArray[(typing.Any, Ellipsis), Any]) – The image to place on the left.
- **image_1** (NDArray[(typing.Any, Ellipsis), Any]) – The image to place on the right.

Return type NDArray[(typing.Any, Ellipsis), Any]

Returns An image with the original two images next to each other.

Note: The images must have the same height.

Example:

```
color_image = rc.camera.get_color_image()

depth_image = rc.camera.get_depth_image()
depth_image_colormap = rc_utils.colormap_depth_image(depth_image)

# Create a new image with the color on the left and depth on the right
new_image = rc_utils.stack_images_horizontally(color_image, depth_image_colormap)
```

racecar_utils.**stack_images_vertical**(*image_0*, *image_1*)

Stack two images vertically.

Parameters

- **image_0** (NDArray[(typing.Any, Ellipsis), Any]) – The image to place on the top.
- **image_1** (NDArray[(typing.Any, Ellipsis), Any]) – The image to place on the bottom.

Return type NDArray[(typing.Any, Ellipsis), Any]

Returns An image with the original two images on top of eachother.

Note: The images must have the same width.

Example:

```
color_image = rc.camera.get_color_image()

depth_image = rc.camera.get_depth_image()
depth_image_colormap = rc_utils.colormap_depth_image(depth_image)

# Create a new image with the color on the top and depth on the bottom
new_image = rc_utils.stack_images_vertically(color_image, depth_image_colormap)
```

PYTHON MODULE INDEX

r

INDEX

M

O

P

R

S

T

W