BStemAPI-Python Documentation

Release 1.0

Brain Corporation

May 29, 2015

CHAPTER

ONE

BSTEM PACKAGE

1.1 Subpackages

1.1.1 bstem.platform package

Submodules

bstem.platform.ad_cord module

class bstem.platform.ad_cord.**AdCord**(*fpga_image='/usr/share/bstem/adcord_bitmap.bin'*, *reset=True*, *enable_motors=True*, *enable_pwm_out=True*)

Bases: *bstem.platform.bstem_platform.Bstem*

The AdCord is a bStem expansion board reference design developed by Brain Corporation for mobile robots. It is designed for wheeled platforms where precise DC motor control with quadrature feedback is required. AdCord also provides 8 RC servo outputs. To use this class first create an AdCord object:

```
ad = Adcord()
```

All servos and motors are disabled by default, to enable:

```
ad.enable_motors = True
```

For each servo (0-7) set position in the range [-1, 1] using:

```
ad.servo[0].position = 0.5
ad.servo[1].position = -0.5
```

or by setting pulse width in microseconds in the range [600, 2400]:

```
ad.servo[0].pulse_width = 600
```

For each motor (0-3) set the speed in the range [-1, 1] with:

```
ad.motor[0].speed = 0.5
ad.motor[1].speed = -0.5
```

where positive values are forward and negative values are reverse.

To read current motor position in rads:

```
ad.encoder[0].position
```

and the current motor velocity in rads/sec:

```
ad.encoder[0].position
```

Set GPIO (0-6) direction using:

```
ad.gpio[0].direction = "in"   % set GPIO to input
ad.gpio[1].direction = "out"  % set GPIO to output
```

For output GPIOs, set the current value using:

```
ad.gpio[0].value = 0
ad.gpio[1].value = 1
```

Read the current GPIO value using:

```
value = ad.gpio[0].value
```

The current battery voltage can be read using:

```
ad.battery
```

The AdCord also provides access to all features of bStem (accelerometer, gyroscope, barometer, magnetometer, leds) e.g.:

```
(x, y, z) = ad.gyroscope.value
(x, y, z) = ad.accelerometer.value
...
```

Parameters

- **fpga_image** – path to AdCord fpga image
- **reset** – reset board on creation
- **enable_motors** – enable motors on creation

SERVO_DS = 8
(CONSTANT) Device select address for servo/pwm-out

battery
Property that returns battery value.

enable_motors
Interface to writeable property that enables motors.

enable_pwm_out
Interface to writeable property that enables pwm-out.

name
Returns a string reflecting the cord name (e.g., adcord)

reset()
Reset all servos to the mid position and reload FPGA firmware.

bstem.platform.bstem_platform module

class bstem.platform.bstem_platform.**Bstem**
Bases: object

This class provides access to sensors and LEDs on the bStem board. To use first create a Bstem object:

```
b = Bstem()
```

Read the gyroscope using:

```
(x, y, z) = b.gyroscope.value
```

Set gyroscope sensitivity in degrees per second (250, 500 or 2000) using:

```
self.gyroscope.dps = 250
```

Read the accelerometer using:

```
(x, y, z) = b.accelerometer.value
```

Read the magnetometer using:

```
(x, y, z) = b.magnetometer.value
```

Read the barometer using:

```
pressure = b.barometer.pressure
```

Set the LEDS (blue, green, yellow) using:

```
b.blue_led = 0     % turn led off
b.green_led = 1    % turn led on
b.yellow_led = 2   % turn led on (max brightness)
```

bstem.platform.lg_cord module

class bstem.platform.lg_cord.**LgCord**(*fpga_image='/usr/share/bstem/lgcord_bitmap.bin'*, *reset=True*, *enable_motors=True*)

Bases: *bstem.platform.bstem_platform.Bstem*

The LgCord is a beta bStem expansion board reference developed by Brain Corporation for Lego/Mindstorm robots. In combination with a bStem, it can be put in place of Lego's NXT or EV3 controller. It can control up to 6 motors and read from 2 or more sensors (with multiplexing.) Lego cord.

To use this class first create an LgCord object:

```
lg = LgCord()
```

For each motor (0-5) set the speed in the range [-1, 1] with:

```
lg.motor[0].speed = 0.5
lg.motor[1].speed = -0.5
```

where positive values are forward and negative values are reverse.

The LgCord also provides access to all features of bStem (accelerometer, gyroscope, barometer, magnetometer, leds) e.g.:

```
(x, y, z) = lg.gyroscope.value
(x, y, z) = lg.accelerometer.value
...
```

Parameters

- **fpga_image** – path to LgCord fpga image
- **reset** – reset board on creation

- **enable_motors** – enable motors on creation

battery

enable_motors

reset()
Reload FPGA firmware.

bstem.platform.rc_cord module

class bstem.platform.rc_cord.**RcCord**(*fpga_image='/usr/share/bstem/rccord_bitmap.bin'*, *reset=True*, *enable_servo_power=False*, *enable_pwm_out=True*)
Bases: *bstem.platform.bstem_platform.Bstem*

The RCCord is a bStem expansion board reference design developed by Brain Corporation for radio-control robotic applications. RcCord is designed to control traditional RC vehicles, like RC cars, planes, and quad-copters. RcCord can read signals coming from RC receiver units and can output RC servo pulses. To use this class first create an RcCord object:

```
rc = RcCord()
```

All servos are disabled by default, to enable:

```
rc.enable_servo_power = True
```

For each servo (0-8) set position in the range [-1, 1] using:

```
rc.servo[0].position = 0.5
rc.servo[1].position = -0.5
```

or by setting pulse width in microseconds in the range [600, 2400]:

```
rc.servo[0].pulse_width = 600
```

Set GPIO (0-8) direction using:

```
rc.gpio[0].direction = "in"   % set GPIO to input
rc.gpio[1].direction = "out"  % set GPIO to output
```

For output GPIOs, set the current value using:

```
rc.gpio[0].value = 0
rc.gpio[1].value = 1
```

Read the current GPIO value using:

```
value = rc.gpio[0].value
```

The current battery voltage can be read using:

```
rc.battery
```

The RcCord also provides access to all features of bStem (accelerometer, gyroscope, barometer, magnetometer, leds) e.g.:

```
(x, y, z) = rc.gyroscope.value
(x, y, z) = rc.accelerometer.value
...
```

Parameters

- **fpga_image** – path to RcCord fpga image
- **reset** – reset board on creation
- **enable_servo_power** – enable power to servo IO ports

ADC_DS = 24
(CONSTANT) Part of device select address for rcCord

PPM_DS = 2
(CONSTANT) Part of device select address for rcCord

RC_DS = 0
(CONSTANT) Part of device select address for rcCord

RC_MULTI_DS = 1
(CONSTANT) Part of device select address for rcCord

SERVO_DS = [8, 9, 10, 11, 12, 13, 14, 15]
(CONSTANT) Part of device select address for rcCord

battery

enable_pwm_out

enable_servo_power

name
Returns the cord name

reset()
Reset all servos to the mid position and reload FPGA firmware.

Module contents

1.1.2 bstem.test package

Submodules

bstem.test.augmented_disparity module

bstem.test.cord_info module

bstem.test.cord_info.**find_file**(*name*, *path*)
Searches for file name in given path.

bstem.test.cord_info.**get_bstem_cord**(*reset=True*)
Finds the type of cord from file /usr/share/bstem/.adcord or .rccord or .lgcord created by the user and returns a default cord accordingly. User need to override the required cord parameters.

bstem.test.cord_info.**get_cord_property**()
Finds the type of cord based on the file /usr/share/bstem/.adcord or .rccord or .lgcord as created by the user. The function also returns appropriate fpga binary image file. When bstem_fpga folder is available within the bstem folder, then specific binary image is used, else default system binary image is used.

bstem.test.demo_video_encoder module

bstem.test.test_bstem module

class bstem.test.test_bstem.**TestBstem**

test_version()

bstem.test.test_camera module

bstem.test.test_control module

class bstem.test.test_control.**CLTestLoop**(*freq*)
Bases: *bstem.control.ControlLoop*

loop()

class bstem.test.test_control.**TestControlLoop**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

test_multiple_control_loops()

test_single_control_loop()

bstem.test.test_gyro_and_accel_reading_speed module

bstem.test.test_image_wrapper module

bstem.test.test_image_wrapper.**test_YUV_forms**()

bstem.test.test_image_wrapper.**test_color_conversions**()

bstem.test.test_image_wrapper.**test_crop**()

bstem.test.test_image_wrapper.**test_reshape_and_types**()

bstem.test.test_image_wrapper.**validate_crop**(*YUV_im*, *x*, *y*, *width*, *height*)

bstem.test.test_led module

bstem.test.test_log module

bstem.test.test_platform module

bstem.test.test_plot module

class bstem.test.test_plot.**TestPlot**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

test_histogram()

test_histogram_remote()

test_histogram_remote_passive()

test_lineplot()

test_lineplot_remote()

test_lineplot_remote_passive()

bstem.test.test_plot.**plot_rand**()

bstem.test.test_plot.**random**() → x in the interval [0, 1).

bstem.test.test_plot.**simple_fxn**()

bstem.test.test_plot.**sin_tuple_fxn**()

bstem.test.test_ppm module

bstem.test.test_range_sensor module

bstem.test.test_rc_and_adcord module

bstem.test.test_sensor module

bstem.test.test_servo module

bstem.test.test_tracker module

class bstem.test.test_tracker.**TestTracker**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test to ensure that the actual bounding box of the object and the CAMshift calculated bounding box of the object are within the expected delta.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

test_tracker()

bstem.test.test_tracker.**generate_sample_video**()
Generates a video for testing the CAMshift based Tracker for Bstem

bstem.test.test_tutorials module

bstem.test.test_video_encoder module

Module contents

1.1.3 bstem.tutorials package

Submodules

bstem.tutorials.demo_sonar_ir module

bstem.tutorials.demo_sonar_ir.**demo_range_sensor**(*sensor_type*, *num_data=1000*)
read and process various range sensors

bstem.tutorials.example_10_tracking module

class bstem.tutorials.example_10_tracking.**TrackerApp**

run()
Run the tracker app

bstem.tutorials.example_11_disparity module

This tutorial shows how to use the disparity hardware with the stereo cameras.

Before running this tutorial, please confirm that both cameras are plugged in.

bstem.tutorials.example_11_disparity.**hardware_disparity_example**(*num_frames=1000*, *cam_override=None*)
Compute disparity using hardware acceleration. The main pro is that one gets reatively detailed disparity estimates with fairly low cpu cost. However, this method is prone to significant noise and as a result may be difficult to use. Noise can be both large holes where disparity can not be reliably computed as well as erroneously estimating objects as near when they are in fact far due to false matches.

bstem.tutorials.example_11_disparity.**show_disparity**(*disparity*, *size*)

bstem.tutorials.example_11_disparity.**software_disparity_example**(*num_frames=1000*, *cam_override=None*)
Compute disparity using software (OpenCV StereoSGBM). The main pro of this method is that it has low noise and fills in pretty well with cpu load comparable to the hardware example. However, this method is quite blurry and reducing the blurring increases the noise. For more information see: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=stereosgbm#cv2.StereoS

bstem.tutorials.example_12_audio module

bstem.tutorials.example_1_hello_world module

Example: bstem LEDs Updated 1/21/2014

bstem.tutorials.example_1_hello_world.**led_example**()

bstem.tutorials.example_2_sensors module

Example: bstem Sensors Updated 1/21/2014

bstem.tutorials.example_2_sensors.**bstem_alternate_sensor_examples**()
Shows an alternative way to access the sensors, through individual sensor APIs

bstem.tutorials.example_2_sensors.**bstem_sensor_examples**()
Shows the standard way to access the sensors (through Bstem platform class)

bstem.tutorials.example_3a_camera module

Example: bstem Cameras Updated 1/21/2014

bstem.tutorials.example_3a_camera.**camera_example**(*num_frames=50*, *cam=None*)

bstem.tutorials.example_3b_video_recording module

Example: bstem Video Recording Updated 1/21/2014

bstem.tutorials.example_3b_video_recording.**video_recording_example**(*duration*, *file_name*, *cam=None*)

A simple example demonstrates the recording capability.

To play back use the command:

>> gst-launch filesrc location=[filename] ! decodebin ! ffmpegcolorspace ! ximagesink

bstem.tutorials.example_3c_camera_config module

Example: bstem camera config Updated 2/5/2014

bstem.tutorials.example_3c_camera_config.**camera_config_example**()
Configure each of the properties of the camera in turn.

bstem.tutorials.example_3c_camera_config.**show_video**(*cam*)
Show 20ms of video.

bstem.tutorials.example_4_scheduler module

Example: bSTEM Control (Scheduler and ControlLoop) Updated 1/21/2014

class bstem.tutorials.example_4_scheduler.**Acceler**(*freq*)
Bases: *bstem.control.ControlLoop*

Reads and integrates the accelerometer

loop()

class bstem.tutorials.example_4_scheduler.**Blinker**(*freq*)
Bases: *bstem.control.ControlLoop*

Turns on and off an LED

loop()

class bstem.tutorials.example_4_scheduler.**Clunker**(*freq*, *compute_time*)
Bases: *bstem.control.ControlLoop*

simulates slow code by waiting

loop()

bstem.tutorials.example_4_scheduler.**timed_loop_example**()

bstem.tutorials.example_5a_local_logging module

Example: bSTEM Logging Updated 1/21/2014

bstem.tutorials.example_5a_local_logging.**collect_data**()

bstem.tutorials.example_5a_local_logging.**filelogger_example**()

bstem.tutorials.example_5a_local_logging.**logger_writeToCsv_example**()

bstem.tutorials.example_5a_local_logging.**main**()

bstem.tutorials.example_5b_remote_logging_client_server module

Example: bSTEM Remote Logging [on bstem] Updated 1/21/2014

bstem.tutorials.example_5b_remote_logging_client_server.**main**()

bstem.tutorials.example_6a_local_plotting module

Example: bSTEM Remote Plotting Updated 1/21/2014

bstem.tutorials.example_6a_local_plotting.**main**()

bstem.tutorials.example_6a_local_plotting.**plot_local_example**()

bstem.tutorials.example_6a_local_plotting.**plot_rainbow_example**()

bstem.tutorials.example_6b_remote_plot_client_server module

Example: bSTEM Remote Plotting [on bstem] Updated 1/21/2014

bstem.tutorials.example_6b_remote_plot_client_server.**main**()

bstem.tutorials.example_7_parameter_tuning module

bstem.tutorials.example_7_parameter_tuning.**main**()

bstem.tutorials.example_7_parameter_tuning.**variable_server_example**()

bstem.tutorials.logging_client module

Example: bSTEM Remote Logging [on PC] Updated 1/21/2014

bstem.tutorials.logging_client.**collect_data**()

bstem.tutorials.logging_client.**remotelogger_client**(*ip*, *port*)

bstem.tutorials.logging_server module

Example: bSTEM Remote Logging [on PC] Updated 1/21/2014

bstem.tutorials.logging_server.**print_logserver_data**(*logserver*, *fields*)
Helper function to print data from a logserver.

Parameters

- **logserver** (*LogServer instance*) – Server containing data to be printed
- **fields** (*String or Iterarble of Strings*) – Fields (Sensor names) to print

bstem.tutorials.logging_server.**remotelogger_server**(*port*)

bstem.tutorials.remoteplot_client module

bstem.tutorials.remoteplot_client.**remoteplot_client**(*ip*, *port*)

bstem.tutorials.remoteplot_server module

bstem.tutorials.remoteplot_server.**remoteplot_server**(*port*)

Module contents

1.1.4 bstem.utils package

Submodules

bstem.utils.analyze_sonar_dataset module

bstem.utils.compass_calibration module

class bstem.utils.compass_calibration.**CompassCalibration**

Calibration returns axis specific Offset and Magnitude measurements, which are incorporated into the Magnetometer.value() Direct uncalibrated values can still be obtained from the Bstem Sensors .raw_value() or by setting calibrated=False when reading from Magnetometer. Successive Calibrations are appended to the /usr/share/bstem/.compass_calibration.log file The most recent calibration will be used to report calibrated values.

classmethod calibrate(*calibration_time_sec=10*, *plot_results=False*, *wait_for_keyboard=True*)

bstem.utils.demo_ppm_out module

bstem.utils.get_min_max_ppm module

Utility script for determining the maximum and minimum values observed on the bStem PPM inputs. Once the script is running, the user adjust the full range of the control signals going into the PPM input. When the user hits "return", the maximum and minimum values for all 8 channels over the sample period are displayed.

bstem.utils.get_min_max_ppm.**get_min_max_ppm_values**(*poll_user_exit=True*, *cord=None*, *timeout_s=None*)

Polls all 8 PPM input channels and keeps track of the maximum and minimum values of each channel.

The get_min_max_ppm_values function polls all 8 PPM input channels over a period of time to determine the maximum and minimum value for each channel that is observed over the sampling interval. The sampling period is specified by *timeout_s* or until the user hits return. The maximum and minimum values for each channel are then displayed on the screen.

poll_user_exit [{ True, False }, optional] Check for user exit request (return button) on each sample interval

cord [bStem RcCord, optional] The cord that contains the PPM input channels. Default will instantiate a new RcCord object

timeout_s [float, optional] The sampling interval timeout in seconds. The default is no timeout and to wait for the user to hit return to exit

nothing

bstem.utils.get_min_max_ppm.**hit_return**()

Check if user hit return on the keyboard

boolean True if the user hit return, False otherwise

This only works on linux systems and fails on py.test with error: ValueError: redirected Stdin is pseudofile, has no fileno(). *poll_user_exit* added to get_min_max_ppm_values() to avoid execution in test_ppm.py unit test

1.2 Submodules

1.3 bstem.analog_filter module

class bstem.analog_filter.**AnalogFilter**(*cord*, *analog_pin*, *mapping_function*, *filter_type='MEDIAN'*, *buffer_length=5*)

Bases: object

A generic base analogfilter class that supports various types of filtering data read from adc in various cord. Samples can be continiously added and various statistics can be calulated.

Notes: - We need to build support for fixed-frequency polling of sensors.

cord: :class: *RcCord* RcCord object

analog_pin [int] Pin to which the sensor is attached

mapping_function [method] 1D Mapping function: Voltage -> Distance Polynomial and coefficients for mapping voltage to distance

filter_type [str] Filter type for processing samples (MEDIAN or MODE or NONE). When NONE use recent sample

buffer_length [int] size of circular buffer for median filter

distance

out [int] The distance estimate using filtered data and mapping function

get_last_sample()
returns the last collected analog voltage sample

get_samples()
returns entire collected sample for user specific processing

raw_value

out [float] The raw value of analog voltage estimates of recent sample

1.4 bstem.battery_check module

bstem.battery_check.**do_battery_check**(*dev*, *warning_level*, *shutdown_level*, *daemon=False*, *sleep_time=60*)
check the battery level and report warning/shutdown conditions to protect battery. if daemon=True than the program runs in background. sleep_time in seconds enable how often we need to check the battery level and warn the user.

bstem.battery_check.**generate_shutdown_message**(*level*)

bstem.battery_check.**generate_warning_message**(*level*)

bstem.battery_check.**run_daemon**(*dev*, *warning_level*, *shutdown_level*)

1.5 bstem.camera module

class bstem.camera.**Camera**(*output_format='BGR'*, *video_size=None*, *auto_pause_threshold=2*, *number_of_snapshots_after_an_update=10*, *stereo_mode=True*)

Bases: *bstem.sensor.Sensor*

Camera module implements the interface to bSTEM's stereo cameras. It provides interface for taking snapshots and making video recordings.

Interface for configuring and reading properties of the on-board stereo cameras of bStem such as brightness, saturation, contrast and hue. Currently each setting is applied to both cameras. A combination of settings that removes adaptation is available via the remove_adaptation method.

Currently, multiple consecutive snapshots and single recording work.

Some failure modes include:: record; snapshot; snapshot -> second snapshot hangs at starting up snapshot; record -> record hangs at the end record; record -> first record works, the second record hands at the end

Initialize a camera object :param output_format: determine snapshot return image image format. "RGB" or "BGR" :type output_format: string :param auto_pause_threshold: if the interval between calling snapshot is greater than auto_pause_threshold * 40ms then auto-pausing will be triggered to save on CPU time :type auto_pause_threshold: float :param number_of_snapshots_after_an_update: define how many snapshot we will take after changing configuration of the camera (default is 10) :type number_of_snapshots_after_an_update: int

atexit()
helper method to handle python shutdown with extra threads correctly

auto_exposure
Get autoexposure mode

Returns True : auto exposure is enabled False : auto exposure is disabled

Return type bool

auto_white_balance
Get auto white balance mode

Returns True : auto white balance is enabled False : auto white balance is disabled

Return type bool

brightness
Get image brightness

Returns brightness

Return type int

central_auto_exposure
True if auto_exposure is using only the central region of the image.

Returns True : central auto exposure is enabled False : central auto exposure is disabled

Return type bool

cleanup()

contrast
Get image contrast

Returns contrast

Return type int

end_recording()
Ends the recording. Stopping gstreamer pipeline.

fade_to_black
Get fade-to-black dampening factor

Returns dampening factor

Return type int

gain
Get gain, only meaningful when auto exposure is off

Returns gain

Return type int

hue
Get image hue

Returns hue

Return type int

pause()
Call pause() to pause the pipeline. This can substantially reduce the cpu load if snapshot is not going to be immediately called again. If snapshot is going to be called in a tight loop at or near the frame rate of the camera then pause should not be called.

remove_adaptation()
A combination of parameters that removes all automatic adaptation to lighting conditions and sets a reasonable gain

saturation
Get image saturation

Returns saturation

Return type int

sfx
Get SFX control

Returns 'disabled' SFX mode disabled 'monochrome' Monchrome 'sepia' Sepia 'negative' Negative image 'solarization' Solarization with unmodified UV 'solarization_neg_uv' Solarization with negative UV

Return type string

sharpness
Get image sharpness

Returns sharpness

Return type int

snapshot(*img_prefix=None*, *disable_auto_pause=False*)
Take a snapshot. Save to file if flag is set Returns a tuple (left_image, right_image) of ImageWrapper instances. The format of the returned ImageWrapper is determined by the output_format the camera is configured to use. ImageWrapper can be treated like a normal ndarray.

Parameters

- **img_prefix** (*string*) – If specified, snapshots will be saved as prefix_[left|right].jpg.
- **disable_auto_pause** (*bool*) – if True, don't ever pause the pipeline

start_recording (*file_name*, *bitrate=1000000*)
Starts the recording. Starting gstreamer pipeline. :param file_name: the video file path that ends with '.mp4' :type file_name: string

write_images (*img_prefix*, *left_im*, *right_im*)

1.6 bstem.control module

class bstem.control.**ControlLoop** (*freq=100*, *tolerance=0.001*)
Bases: object

A single control loop.

Initialize parameters of control loop.

Parameters

- **freq** (*int*) – frequency in *Hz* that loop function will be called
- **tolerance** (*double*) –
- **tolerance** – acceptable scheduling error e.g. the loop may execute within [-tolerance, tolerance] *ms* of expected time for any iteration of the loop

avg_error

Returns average error in ms of loop iteration

Return type float

avg_exec_time

Returns average execution time in ms of loop iteration

Return type float

control_vars_reset (*start_time*)
Reset all counters.

Parameters **start_time** (*long*) – loop start time

error

Returns error in ms of last loop iteration

Return type float

exec_time

Returns execution time in ms of last loop iteration

Return type float

freq

Returns loop frequency in *Hz*

Return type int

iter

Returns the current loop iteration

Return type int

loop()
This must be overridden by each class that inherits from ControlLoop to implement the main functionality of the loop

period

Returns loop period in *ms*

Return type float

remaining(*cur_time*)

Returns time in *ms* remaining until next iteration of the loop

Return type int

update(*cur_time*)
Called by the controller to schedule loop. Checks if loop needs to run, if so run and update time remaining to next iteration.

Parameters **cur_time** (*float*) – the current time

class bstem.control.**Scheduler**
Bases: object

Container for control loops associated with a single device. Responsible for scheduling executing of all control loops associated with that device.

classmethod add(**args*, ***kwargs*)
Add a new control loop to the device.

Parameters **loop** (*subclass of ControlLoop*) – loop construct defining periodic behavior and frequency

classmethod remove(**args*, ***kwargs*)
Remove an existing control loop.

Parameters **loop** (*subclass of ControlLoop*) – loop already added to Scheduler

classmethod start(**args*, ***kwargs*)
Start execution of all control loops.

Parameters **time_limit** (*int/float*) – length (in seconds) of time to run Scheduler

classmethod suspend(**args*, ***kwargs*)
Suspend execution of all control loops. Call *start()* to resume.

1.7 bstem.decorator module

bstem.decorator.**singletonmethod**(*orig_func*)
Initialize singleton instance when method is called if it has not been initialized already.

Parameters **orig_func** (*function*) – Method intended to affect the singleton instance of object, creating an instance if one does not exist already.

Returns Modified *orig_function*.

Return type function

If **Class** has *function(...)* defined with decorator *@singletonmethod*, calling **Class**.*function(...)* will behave as follows:

```
    if Class._instance is None:
        Class._instance = Class._Impl()
        Class.function(...) # Usually affects Class._instance
    else:
        Class.function(...) # Usually affects Class._instance
```

1.8 bstem.device module

class bstem.device.**AbsEncoder**(*fpga*, *device_select*)
Bases: object

Absolute position encoder calculated via a potentiometer

Absolute position encoder.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

gain

position
Get position.

Returns position in clock ticks

Return type int

class bstem.device.**AdcAdCord**(*fpga*, *device_select*)
Bases: object

Analog to Digital converter for the AdCord

Analog-to-digital encoder.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

ADC_DEV_BATTERY = 144

ADC_DEV_MOTOR_0 = 32

ADC_DEV_MOTOR_1 = 160

ADC_DEV_MOTOR_2 = 0

ADC_DEV_MOTOR_3 = 16

value(*dev=None*, *pin=None*)
Retrieve value from encoder.

Parameters

- **val** (*int*) – encoder value
- **dev** (*float (BATTERY = Voltage, MOTOR = Current)*) – device id (ADC_DEV_BATTERY, ADC_DEV_MOTOR_1, ...)

class bstem.device.**AdcLgCord**(*fpga*, *device_select*)
Bases: object

Analog to Digital converter for the LgCord

Analog-to-digital encoder.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

ADC_DEV_MOTOR_0 = 32

value(*dev*)
Retrieve value from encoder.

Parameters

- **val** (*int*) – encoder value
- **dev** (*float (BATTERY = Voltage, MOTOR = Current)*) – device id (ADC_DEV_BATTERY, ADC_DEV_MOTOR_1, ...)

class bstem.device.**AdcRcCord**(*fpga*, *device_select*)
Bases: object

Analog to Digital converter for the RcCord

Analog-to-digital encoder.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

ADC_DEV_BATTERY = 10

value(*pin*)
Public interface for reading analog input on the RcCord. port: pin number

class bstem.device.**Encoder**(*fpga*, *device_select*, *ticks_per_rev=1200*)
Bases: object

Interface to read motor information - position and velocity

Quadrature encoder.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA
- **ticks_per_rev** (*int (default 1200)*) – number of encoder ticks per wheel revolution

calc_velocity(*vel*, *dt*)
Calculate motor velocity.

Parameters

- **vel** (*float*) – raw encoder postion returned from the FPGA
- **dt** – time step

Returns velocity in rads/s

Return type float

convert_to_clicks(*val*)
Convert encoder rads to clicks.

Parameters **val** (*float*) – angle in radians

Return type int

convert_to_rads(*val*)
Convert encoder clicks to rads.

Parameters **val** (*int*) – angle in clicks

Return type float

position
Encoder position in radians.

Returns position in radians

Return type float

position_clicks
Encoder position in integer number of half-degrees. One full rotation takes 720 ticks.

Returns position in radians

Return type float

velocity
Encoder velocity.

Returns velocity in rads/s

Return type float

class bstem.device.**Gpio**(*fpga*, *device_select_in*, *device_select_out*, *device_select_direction*, *pin_number*)
Bases: object

General purpose input/output pin.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select_in** (*unsigned byte*) – id of device on the FPGA for input
- **device_select_out** (*unsigned byte*) – id of device on the FPGA for output
- **pin_number** (*unsigned byte*) – pin number that is controlled on the FPGA

direction
Return current direction for the gpio pin

Returns 0/1

Return type int

value
Current value of gpio pin.

Returns 0/1

Return type int

class bstem.device.**Led**(*name*)
Bases: object

Interface to control LEDs on the bSTEM

LED.

Parameters **name** (*string*) – Led name on file system e.g. “red”, “yellow”, “blue”

brightness
Led brightness.

Returns Led brightness in range [0,2]

Return type int

class bstem.device.**LgServo**(*fpga*, *device_select*)
Bases: object

Servo interface - position and pulse_width

Pulse width controlled servo.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

position

pulse_width

class bstem.device.**Motor**(*fpga*, *device_select*)
Bases: object

Interface to motors

Pulse-width modulation (PWM) motor driver.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

disable()
Disable motor.

Returns device status

Return type int

reset()
Reset motor after error or disable.

Returns device status

Return type int

speed

class bstem.device.**MulticolorLed**(*fpga*, *device_select*)
Bases: object

Interface to control external multicolor LEDs. Currently supports Pololu #2546, see:

http://www.pololu.com/product/2546

Multi-colored LED.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

color

class bstem.device.**PPMOutput**(*fpga*, *device_select*)
Bases: object

Pulse-width generator using PPM to combine all 8 channels to one line of output. The minimum pulse-width using PPM is fixed at 0.5ms per channel. The interchannel pulse-width stop period is 0.4ms. There will be a total of 8 channels and PPM widht is updated very 18ms corresponding. Currently ppm position based API is not supported and only pure pulse-width based interface is supported.

fpga [FPGA] :FPGA object

device_select [int] device select specific pin used to enable ppm slave

get_pulse_width(*channel*)
Get servo servo pulse-width in millisecond

pulse_width(*channel*, *val*)

channel [int] channel number for calculating the pulse-width [range: 0 upto 7]

value [int] pulse-width that need to be output to specific channel in microsecond Expected range for val is 0.5 to 1.7 (between 0.5ms to 1.7ms) Hard-coded minimum of 0.5ms set in the FPGA code (if counter is zero all the ppm output will be set to 0.5ms pulse per channel).

class bstem.device.**PowerGauge**(*fpga*, *device_select*)
Bases: object

Texas Instruments bq2050H Lithium Ion Power Gauge

Lithium-Ion Power Gauge.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

id
Read battery ID register. Result will be returned by next call to PowerGauge after approximately 4ms.

Returns Result of previous call to PowerGauge

Return type unsigned byte

max_cell_voltage
Read maximum cell voltage register. Result will be returned by next call to PowerGauge after approximately 4ms.

Returns Result of previous call to PowerGauge

Return type unsigned byte

primary_status
Read primary status register. Result will be returned by next call to PowerGauge after approximately 4ms.

bit 7 : Charge status flag, asserted when charge rate detected

bit 6 : Battery replaced flag, asserted whenever bq2050h is reset

bit 5 : Protector status flag, status of the overvoltage protector:

```
0 = PSTAT input low (< 0.5 V)
1 = PSTAT input high (>2.5 V)
asserted when PSTAT high and cleared when low
```

bit 4 [Capacity inaccurate flag, used to warn user when battery has been] been charged a substantial number of times since LMD (Last Measured Discharge):

```
0 when LMD update with full valid discharge
1 after 64th valid charge with no LMD update
```

bit 3 : Valid discharge flag

bit 2 : First end-of-discharge warning, warns user when battery almost empty

bit 1 [Final end of discharge warning, warns user when battery at failure] condition

Returns Result of previous call to PowerGauge

Return type unsigned byte

secondary_status

Read secondary status register. Result will be returned by next call to PowerGauge after approximately 4ms.

bits 6-4 : Discharge rate:

```
0 0 0 : drate < 0.5C
0 0 1 : 0.5C < drate < 2C
0 1 0 : 2C < drate
```

bit 3 : Enable interupt flag, test bit used to determine Vsr activity

bit 2 : Valid charge flag, valid charge condition

bit 0 : Overload flag, asserted when a charge rate in excess of 2C detected

Returns Result of previous call to PowerGauge

Return type unsigned byte

temperature

Read temperature register. Result will be returned by next call to PowerGauge after approximately 4ms.

Returns Result of previous call to PowerGauge

Return type unsigned byte

static to_temp_range(*val*)

Convert temperature register value to temperature range.

Parameters **val** (*unsigned byte*) – Temperature register value

Returns Temperature range

Return type (int, int)

static to_voltage(*val*)

Convert battery voltage register value to voltage.

Parameters **val** (*unsigned byte*) – Battery register

Returns Voltage

Return type float

voltage
Read battery voltage register. Result will be returned by next call to PowerGauge after approximately 4ms.

Returns Result of previous call to PowerGauge

Return type unsigned byte

class bstem.device.**QuadEncoder**(*fpga*, *device_select*, *ticks_per_rev=1200*)
Bases: object

Interface to read motor information from a quadrature encoders - position and velocity

Quadrature encoder.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA
- **ticks_per_rev** (*int (default 1200)*) – number of encoder ticks per wheel revolution

calc_velocity(*vel*, *dt*)
Calculate motor velocity.

Parameters

- **vel** (*float*) – raw encoder postion returned from the FPGA
- **dt** – time step

Returns velocity in rads/s

Return type float

convert_to_clicks(*val*)
Convert encoder rads to clicks.

Parameters **val** (*float*) – angle in radians

Return type int

convert_to_rads(*val*)
Convert encoder clicks to rads.

Parameters **val** (*int*) – angle in clicks

Return type float

position
Encoder position in degrees. One full rotation increments/decrements with 2 x pi.

Returns position in radians

Return type float

position_clicks
Encoder position in integer number of half-degrees. One full rotation takes 720 ticks.

Returns position in clicks

Return type int

velocity
Encoder velocity.

Returns velocity in rads/s

Return type float

class bstem.device.**Rc**(*fpga*, *device_select*, *channel*)

Bases: object

Pulse width generator

FrSky D4R-II eight channel RC controller.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA
- **channel** (*unsigned byte*) – RC channel

pulse_width

Current value of gpio pin.

Returns 0/1

Return type int

class bstem.device.**Servo**(*fpga*, *device_select*, *pin_number*, *pwm_frequency_hz=50*, *pulse_width_min_us=600*, *pulse_width_max_us=2400*)

Bases: object

Servo interface - position and pulse_width

Pulse width controlled servo.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA
- **pin_number** (*unsigned byte*) – pin number of the servo port
- **pwm_frequency_hz** (*integer*) – frequency of pwm signal (default = 50 Hz)
- **pulse_width_min_us** (*float*) – minimum size of the pulse width in microsecond (default = 600 us)
- **pulse_width_max_us** (*float*) – maximum size of the pulse width in microsecond (default = 2400 us)

position

Return the current position value that is used by the servo pwm counter. The value - [0, 1] when PWM pulse is being generated. The value < 0, when PWM pulse is disabled.

pulse_width

Get servo servo pulse-width in microsecond

pwm_frequency

Return the current PWM frequency in hertz

set_position_minmax(*min_val*, *max_val*)

Set the min/max value of the position parameter. Change is servo position is evaluated to ensure that the position stays within the required range specified here. If the user wants the servo position to be in the range 0.3 tp 0.7, then call,

servo.set_position_minmax(0.3, 0.7)

Parameters

- **min_val** (*float*) – position minimum value
- **max_val** (*float*) – position maximum value

class bstem.device.**SmartServo**(*fpga*, *device_select*, *abs_encoder*, *quad_encoder*)

Bases: object

Servo interface for smart servos with absolute position and with standard quad encoder position and velocity

Smart servo.

Parameters

- **fpga** (*bstem.Fpga*) – FPGA object for SPI communication
- **device_select** (*unsigned byte*) – id of device on the FPGA

abs_gain

abs_position

position

Encoder position in degrees. One full rotation increments/decrements with 2 x pi.

Returns position in radians

Return type float

position_clicks

Encoder position in integer number of half-degrees. One full rotation takes 720 ticks.

Returns position in clicks

Return type int

speed

velocity

Encoder velocity.

Returns velocity in rads/s

Return type float

1.9 bstem.disparity_hardware module

class bstem.disparity_hardware.**DisparityHardware**(*width=1280*, *height=720*, *image_change_threshold=2*, *speed_up=False*, *merge_left_right_estimates=False*, *minimum_disparity=0*)

Bases: object

Interface for computing disparity using hardware acceleration

Parameters

- **width** (*int*) – width of the images to be passed in must be a multiple of 8
- **height** (*int*) – height of the images to be passed in must be a multiple of 8
- **image_change_threshold** (*int, default 2*) – how much in pixel values a region (8x8 pixels) of an image must change by to indicate significant change, and thus allow disparity to be computed there.

- **speed_up** (*boolean, default False*) – configure the hardware to run up to 50% faster but with the trade off of not being able to run multiple instances of disparity
- **merge_left_right_estimates** (*boolean, default False*) – Combine the standard disparity estimate (right to left) with an approximate estimate of disparity (left to right) to improve signal to noise. This approximation is only valid when the camera motion is much smaller than the disparity offset.
- **minimum_disparity** (*int, default 0*) – Any disparity value less than minimum_disparity will be set to zero. This is most relevant when merge_left_right_estimates is True.

add_images(*left_im*, *right_im*)
add an image to have motion computed on

Parameters

- **left_im** (ImageWrapper) – left input image
- **right_im** (ImageWrapper) – right input image

The disparity result can be retrieved by calling get_disparity() and will be available after approximately 2 calls to add_images().

static aligned_crop(*left_im*, *right_im*, *crop_x*, *crop_y=0*)
crop (equivelantly translate) left and right images relative to one another so that they are aligned (rectified) to one another at infinity or to exploit the full dynamic range of the hardware.

Parameters

- **left_im** (ImageWrapper) – left input image
- **right_im** (ImageWrapper) – right input image
- **crop_x** (*int*) – how much to remove in pixels from the left side of the left image, if negative then it applies to the left side of the right image instead.
- **crop_y** (*int, default 0*) – how much to remove in pixels from the top of the right image, if negative then it applies to the top of the left image instead.

Returns left_im, right_im

The cropped images will be constructed to have a width a multiple of 32 and height a multiple of 16 as required by the hardware.

get_disparity()

Returns disparity_map, frame_num, left_im, and right_im or None,None,None,None if data isn't available yet

Disparity is aligned to the left image, so as to match OpenCV's convention.

start()
must be called before calling add_images or get_disparity

stop()
should be called when done processing for now, but may start up again

1.10 bstem.display_thread module

class bstem.display_thread.**DisplayThread**(*queue_size=10*, *image_size=(270, 240)*)
Bases: object

Helper class to display images outside the compute thread

atexit_handler()
helper method to handle python shutdown with extra threads correctly

put_window_queue(*name*, *img*)
puts an image into the queue to be eventually displayed name – the window name used by cv2.imshow img – the image to be shown, must be an ndarray or ImageWrapper

waitKey_thread()

1.11 bstem.exception module

exception bstem.exception.**DeviceNotEnabled**(*value*)
Bases: exceptions.Exception

Raised when attempting communication with a device that is not enabled

exception bstem.exception.**DeviceNotFound**(*value*)
Bases: exceptions.Exception

Raised when unable to communicate with driver for the requested device

1.12 bstem.fpga module

class bstem.fpga.**Fpga**(*spi=1*)
Bases: object

Serial Peripheral Interface (SPI) FPGA communication

SPI_1 = 1

SPI_10 = 10

program(*filename*)
Write image file (.bin) to the FPGA.

Parameters **filename** (*string*) – path to image file

spi_send(*device*, *command*, *val*)
Write command to FPGA SPI bus and receive response.

Parameters

- **device** (*unsigned byte*) – device id
- **command** (*unsigned byte*) – the command
- **val** (*int*) – value to written to the device

Returns response from FPGA

Return type int

spi_send_mult(*commands*)
Write multiple commands to the FPGA SPI bus and receive responses.

Parameters

- **device** (*unsigned byte*) – device id
- **command** (*list[list[unsigned byte, unsigned byte, int], ...]*) – list of lists containing (device, command, val)

Returns tuple of responses from the FPGA

Return type tuple(int, ...)

1.13 bstem.hokuyo module

To use this code, you need to setup a static ip address for the lidar.

Add the following:

iface eth8 inet static address 192.168.0.2 gateway 192.168.0.1 netmask 255.255.255.0

to /etc/network/interfaces.

When you boot, run the following command:

sudo ifup eth8

class bstem.hokuyo.**HokuyoDriver**(*ip='192.168.0.10'*, *port=10940*)
Hokuyo 10LX Lidar interface

param ip: ip address of the sensor (default: 192.167.0.100) param port: port of the sensor (default: 10940)

close()
Close socket to sensor

distance(*start_step=0*, *end_step=1080*)
Return a single scan of distance data in mm. A complete scan has range [0,1080] steps. Use parameters to select a sub-region of the scan space.

param start_step: starting step of scan param start_step: end step of scan

returns a numpy array of int, one element per distance sample in mm. If read fails, returns an empty array.

sensor_info()
Return sensor info

sensor_params()
Return sensor param info

start()
Start capturing data on the sensor

stop()
Stop capturing data on the sensor

version()
Return version info

bstem.hokuyo.**main**()

1.14 bstem.image_wrapper module

class bstem.image_wrapper.**ImageWrapper**
Bases: numpy.ndarray

ImageWrapper provides an interface to RGB, BGR and YUV (NV12) data in order to record type, cache and provide the same functionality (crop, resize, convert, etc) for all 3 image types. Further, it subclasses numpy's ndarray, which allows it to be treated just like and ndarray, for whichever image format it was created for. This means if an ImageWrapper is created with BGR data, the ImageWrapper instance can be passed to openCV

methods directly. The shape of the ImageWrapper instance will always be (height,width,3) with the exception for YUV data which will always have size height*3/2*width but may take on multiple different shapes for performance reasons.

Beyond caching, the ImageWrapper class also tries to minimize memory copying and maximizing lazy evaluation. As a result, one can crop and downscale without performing any memory copying, which has a significant performance benefit.

BGR
Returns an ImageWrapper of format BGR from the current data, converted if needed. The shape will be (height,width,3) and can be treated like any ndarray.

RGB
Returns an ImageWrapper of format RGB from the current data, converted if needed. The shape will be (height,width,3) and can be treated like any ndarray.

UV
Returns the U and V channel as an (height/2,width/2,2) ndarray, converts if needed. data[:,:,0] is the U channel and data[:,:,1] is the V.

Y
Returns the Y channel as an (height,width) ndarray, converts if needed.

YUV
Returns an ImageWrapper of format YUV (NV12) from the current data, converted if needed. The shape will be (height*3/2,width) and can be treated like any ndarray:

```
Y = data[0:height,:]
UV = data[height::,:].reshape((height/2,width/2,2))
```

where UV[:,:,0] is U and UV[:,:,1] is V.

crop(*x*, *y*, *width*, *height*)
crop the image :param x: horizontal start position of crop :type x: int, 0 to image width :param y: vertical start position of crop :type y: int, 0 to image height :param width: width of crop, i.e. x:x+width is the range for crop :type width: int, 0 to image width :param height: height of crop, i.e. y:y+width is the range for crop :type height: int, 0 to image height

Returns a new ImageWrapper of the same format cropped.

downscale(*downscale_x*, *downscale_y*)
Computationally fast downscaling:

```
:param downscale_x: downscale size in x (width)
:type downscale_x: int
:param downscale_y: downscale size in y (height)
:type downscale_y: int
```

For YUV width/2 % downscale_x must be 0 and height/2 % downscale_y must be 0

For RGB and BGR width % downscale_x must be 0 and height % downscale_y must be 0

Returns an new ImageWrapper of the same format but downscaled.

has_YUV_data()
Test to see if YUV data can be read without converting first.

height

resize(*width*, *height*, *interpolation=1L*)
resize image, can be slow :param width: the desired image width :type width: int :param height: the desired

image height :type height: int :param interpolation: interpolation method, see cv2.INTER_* constants :type interpolation: int, defaults to cv2.INTER_LINEAR

Returns a new ImageWrapper of the same format resized.

width

1.15 bstem.infrared module

class bstem.infrared.**SharpInfrared2D120X**(*cord*, *analog_pin*, *filter_type='MEDIAN'*, *buffer_length=5*)

Bases: *bstem.analog_filter.AnalogFilter*

Interface to Sharp 2D120X (4-30 cm) IR sensor.

mapping_function implements linear interpolation between distance-voltage measurements from Sharp data sheet: http://www.sharpsma.com/webfm_send/1205

Points extracted using: http://arohatgi.info/WebPlotDigitizer

Linear interpolation outperformed inverse and exponential fits.

See SharpInfrared class for further documentation

class bstem.infrared.**SharpInfrared2Y0A21**(*cord*, *analog_pin*, *filter_type='MEDIAN'*, *buffer_length=5*)

Bases: *bstem.analog_filter.AnalogFilter*

Interface to Sharp 2D120X (10-80 cm) IR sensor.

mapping_function implements linear interpolation between distance-voltage measurements from Sharp data sheet: http://www.sharpsma.com/webfm_send/1208

Linear interpolation outperformed inverse and exponential fits.

Points extracted using: http://arohatgi.info/WebPlotDigitizer

See SharpInfrared class for further documentation

1.16 bstem.log module

class bstem.log.**CsvOutput**

Inherit this class in order to use CsvOutput.writeToCsv

class bstem.log.**FileLogger**(*filename*, *func*, *names*, *freq*)

Bases: *bstem.log.LoggerBase*

Logging variables to a CSV text file. Values are written continuously during execution of the scheduler.

Initialize logger.

Parameters

- **filename** (*string*) – the file
- **func** (*function returning tuple of values*) – function returning a tuple of values each time it is called. Each member in the tuple will be logged as a separate column in the CSV file.
- **names** (*tupe of strings*) – A tuple of names corresponding entries returned by func
- **freq** (*int*) – logging frequency in Hz

close()
Close file and remove logger from the scheduler.

loop()
The control loop

class bstem.log.**LogServer**(*port=1234*)
Bases: *bstem.log.CsvOutput*, collections.defaultdict

Server for variable logging.

Initialize logger.

Parameters **freq** (*int (default 1234)*) – port to listen for incoming connections

clear()
Remove all items from the dictionary.

close()
Set the server to stop listening for data and terminate.

copy()
Return a shallow copy of the dictionary.

default_factory
Return default_factory of the dictionary

get(*key*, *default=None*)

Return the value for *key* if *key* is in the dictionary, else *default*. If *default is not given, it defaults to ''None'', so that this method never raises a ''KeyError''.

items()
Return a copy of the dictionary's list of ''(key, value)" pairs.

iteritems()
Return an iterator over the dictionary's ''(key, value)" pairs.

iterkeys()
Return an iterator over the dictionary's keys.

itervalues()
Return an iterator over the dictionary's values.

keys()
Return a copy of the dictionary's list of keys.

pop(*key*, **default*)

If *key* is in the dictionary, remove it and returns its value, else return *default*. If *default is not given and *key* is not in the dictionary, a ''KeyError" is raised.

popitem()

Remove and return an arbitrary ''(key, value)" pair from the dictionary.

read(*fields*, *default=0*)
Returns tuple most recent _Listener readings

Parameters

- **fields** (*string or string tuple/list*) – string/tuple/list of keys to query
- **default** (*object*) – object to return in event of a no-read

example:

ls.read_fields(('x_pos', 'y_pos')) returns most recent readings of 'x_pos' and 'y_pos'

If there are no readings present for a given label, the respective tuple entry will be logged as 'default'

reset_subscriptions()
Resets the list of observers.

setdefault(*key*, *default=[]*)

If *key* is in the dictionary, return its value. If not, insert *key* with a value of *default* and return *default*.

subscribe(*client*, *fields*)

Registers an observer object with this LogServer. Any time a key is updated, LogServer will call update() on all objects observing that key.

Parameters

- **client** (*object with no-argument update() method*) – observer object
- **fields** (*string tuple/list or string*) – keys to watch

unsubscribe(*client*)
Removes an observer object subscription with this LogServer.

Parameters **client** (*Object with no-argument update() method*) – Observer object

update(**E*, ***F*)
Updates the values of the dictionary (''D'') based on the following:

```
If ''E'' is present and has a ''.keys()'' method, then it does ''for k in E: D[k] = E[k]''.
If ''E'' lacks a ''.keys()'' method, then it does ''for (k, v) in E: D[k] = v''.
All cases are followed by ''for k in F: D[k] = F[k]''.
```

values()
Return a copy of the dictionary's list of values.

viewitems()
Return a new view of the dictionary's items (''(key, value)" pairs).

viewkeys()
Return a new view of the dictionary's keys.

viewvalues()
Return a new view of the dictionary's values.

writeToCsv(*filename=None*, *precision=12*)
Write all stored values to a CSV file.

Parameters

- **filename** (*string*) – the file to log to (**will overwrite**)
- **precision** (*int*) – number of decimal places to output (float/complex)

class bstem.log.**Logger**(*func*, *names*, *freq*)
Bases: *bstem.log.CsvOutput*, *bstem.log.LoggerBase*

Variable logging to text file or remote server. Log data is stored in memory during logging.

Initialize logger.

Parameters

- **func** (*function returning tuple of values*) – function returning a tuple of values each time it is called. Each member in the tuple will be logged as a separate column in the CSV file.
- **freq** (*int*) – logging frequency in Hz

clear()
Clear all stored data.

close()
Remove the object from the Scheduler.

enabled
Logging enabled. When true data recorded whenever the Scheduler is running.

Returns true if logging enabled

Return type bool

loop()
The control loop

writeToCsv(*filename=None*, *precision=12*)
Write all stored values to a CSV file.

Parameters

- **filename** (*string*) – the file
- **precision** (*int*) – number of decimal places to output (float/complex)

writeToServer(*ip*, *port=1234*)
Write all stored values to a remote log server.

Parameters

- **ip** (*string*) – IP address of the remote server
- **port** (*int (default 1234)*) – port of remote server

class bstem.log.**LoggerBase**(*func*, *names*, *freq*)
Bases: *bstem.control.ControlLoop*

Abstract base class for all loggers.

Initialize logger.

Parameters

- **func** (*function returning tuple of values*) – function returning a tuple of values each time it is called. Each member in the tuple will be logged as a separate column in the CSV file.
- **freq** (*int*) – logging frequency in Hz

close()
Remove logger from the scheduler.

class bstem.log.**RemoteLogger**(*ip*, *func*, *names*, *freq*, *port=1234*)
Bases: *bstem.log.LoggerBase*

Initialize logger.

Parameters

- **ip** (*string*) – IP address of the remote server
- **func** (*function returning tuple of values*) – function returning a tuple of values each time it is called. Each member in the tuple will be logged as a separate column in the CSV file.

- **names** (*tuple(string)*) – A tuple of names corresponding to each entry returned by func
- **freq** (*int*) – logging frequency in Hz
- **port** (*int (default 1234)*) – port of remote server

close()
Close connection to remote server and remove logger from the scheduler.

loop()
The control loop

class bstem.log.**VariableClient**(*ip*, *port=1235*)
Bases: object

Client for setting variables remotely.

Initialize client.

Parameters

- **ip** (*string*) – ip address or host name of variable server
- **port** (*int (default 1235)*) – remote port

close()
Close the associated socket.

set(*name*, *val*)
Set a remote variable.

Parameters

- **name** (*string*) – name of the variable as set by VariableServer.register
- **val** – the new value

class bstem.log.**VariableServer**(*freq=2*, *port=1235*)
Bases: *bstem.control.ControlLoop*

Server for setting variables remotely.

Initialize server.

Parameters

- **freq** (*int (default 2 Hz)*) – frequency in Hz
- **port** (*int (default 1235)*) – port to listen for incoming connections

close()
Remove logger from the scheduler.

loop()

register(*obj*, *name*)
Register an object with the server.

Parameters

- **obj** (*a python object*) – the object
- **name** (*string*) – object name as it will be referenced on the client

1.17 bstem.motion_hardware module

class bstem.motion_hardware.**MotionHardware**(*width=1280*, *height=720*, *bitrate=3000000*, *max_im_queued=1*, *max_frames_queued=1*, *image_change_threshold=2*)

Bases: *bstem.video_encoder.VideoEncoder*

Motion specific interface to the Video Encoder

Parameters

- **width** (*int*) – width of the images to be passed in must be a multiple of 8
- **height** (*int*) – height of the images to be passed in must be a multiple of 8
- **bitrate** (*int*) – the bitrate used for encoding
- **max_im_queued** (*int*) – the size of the image queue, used by add_image
- **max_frames_queued** (*int*) – the size of the frame queue, used by get_motion
- **image_change_threshold** (*int, default 5*) – how much in pixels a region of an image must change by to indicate significant change, and thus allow motion to be computed there.

add_image(*im*, *block=True*, *timeout=10*)

add an image to have motion computed on

Parameters

- **im** (ImageWrapper) – input image
- **block** (*int, default 10*) – flag to indicate waiting until there is space in the input queue. If False, images will be dropped when the input queue is full.
- **timeout** – If block==True then timeout is how long to wait while blocking

get_motion(*block=False*, *timeout=1*, *fillin_val=0*)

Gets the computed motion. The motion is normally available after 4-6 add_image() calls. Valid motion estimates are between 7 and 248, which correspond to half pixel resolution motion estimates from the previous frame to the current frame. Regions where motion could not be computed will have an "invalid" 0 value.

Parameters

- **block** (*int, default 1*) – flag to indicate waiting until there is a valid motion measurement.
- **timeout** – If block==True then timeout is how long to wait while blocking

Returns dx, dy, frame_number, frame, prev_frame or None,None,None,None,None if no motion is available

dx is the motion in x

dy is the motion in y

frame_number is the number of the frame being returned, starts at 0 and may not be contiguous because data can get dropped

frame is the reference frame for which motion is computed (and aligned to)

prev_frame is the frame used to compute motions from, motion vectors are

start()

must be called before calling add_image or get_motion

bstem.motion_hardware.**video_encoder_speedup_hack**()
Initialize the hardware in a "faster" configuration.

Run this command before using the hardware for motion processing.

This is a complete hack and has no justification for the parameters except that they work.

The video encoder hardward supports up to 4 simultaneous streams to be compressed. By creating several with different sizes the hardware goes into a faster mode. Use this hack until the kernel driver can be manipulated so that we can always get fast encoding.

1.18 bstem.plot module

class bstem.plot.**Plot**
Bases: object

Live variable plotting.

classmethod clear(**args*, ***kwargs*)

classmethod histogram(**args*, ***kwargs*)
Live histogram.

Parameters

- **func** (*function returning single values OR two-element tuple, (LogServer instance, string OR tuple/list of strings)*) – function returning a single value each time it is called. Each member in the tuple will be plotted as a separate line OR a tuple with two elements, a LogServer and a string or tuple of strings representing keys to observe.
- **title** (*string*) – plot title (Default '')
- **x_label** (*string*) – x axis label (Default '')
- **bg_color** (*string*) – plot background color (Default 'k')
- **color** (*[int,]*) – histogram color in [R,G,B,Alpha] format, where each entry ranges from 0-255 (Default [0, 0, 255, 80])
- **bins** (*int*) – number of bins (Default 20)
- **bin_start** (*int*) – min start value, if None will be minimum of recorded values (Default None)
- **bin_end** (*int*) – min end value, if None will be maximum of recorded values (Default None)
- **windowsize** (*[int, int]*) – window size (Default (600, 350))
- **hist_len** – length of history, if None all values are stored (Default None)
- **is_observer** (*Boolean*) – True if plot's update will be called remotely (e.g. if the plot will registering as a subscriber to a LogServer)

NOTE: When argument 'func' is a tuple (instead of a function), the plot will not update itself periodically (the default behavior). Instead, the tuple's first element is interpreted as a LogServer to observe and its second element is interpreted as a string or tuple/list of strings representing which keys on the LogServer should be observed. When a plot's observed key is updated the LogServer will make the necessary calls to update().

Histogram Example:

```
from bstem.plot import Plot
from time import time
from math import sin

def fxn():
    # Line Plot can plot multiple fields
    return sin(time() % 6.2830)

# Add histograms to Scheduler
Plot.histogram(fxn, 'Local Histogram')
Plot.histogram(fxn, bins=40, bin_start=-0.5, bin_end=1.5, windowsize=(800, 700))
Plot.histogram(fxn, 'title', 'x_label', 'w', [255, 0, 0, 50])

# Run Scheduler for 5 seconds
Scheduler.start(5)

# Remove histograms from Scheduler
Plot.clear()
```

classmethod **lineplot**(**args*, ***kwargs*)

Live line plot.

Parameters

- **func** (*function returning single values OR two-element tuple, (LogServer instance, string OR tuple/list of strings)*) – function returning a single value each time it is called. Each member in the tuple will be plotted as a separate line OR a tuple with two elements, a LogServer and a string or tuple of strings representing keys to observe.
- **title** (*string*) – plot title (Default '')
- **y_label** (*string*) – y axis label (Default '')
- **y_units** (*string*) – y unit label (Default None)
- **bg_color** (*string*) – plot background color (Default 'k')
- **color_map** (*[string,]*) – colors for each line (Default ['r', 'g', 'b', 'c', 'm', 'k'])
- **linewidth** (*float*) – line width (Default 2.0)
- **timewindow** (*int*) – time window of plotted data in seconds (Default 10)
- **windowsize** (*[int, int]*) – window size (Default (600, 350))

NOTE: When argument 'func' is a tuple (instead of a function), the plot will not update itself periodically (the default behavior). Instead, the tuple's first element is interpreted as a LogServer to observe and its second element is interpreted as a string or tuple/list of strings representing which keys on the LogServer should be observed. When a plot's observed key is updated the LogServer will make the necessary calls to update().

LinePlot Example:

```
from bstem.plot import Plot
from time import time
from math import sin

def fxn():
    # Line Plot can plot multiple fields
    return (sin(time() % 6.2830) + .3, sin(time() % 6.2830), sin(time() % 6.2830) - .3)
```

```
        # Add lineplots to Scheduler
        Plot.lineplot(fxn, 'Local Line Plot')
        Plot.lineplot(fxn, timewindow=20, windowsize=(800, 700))
        Plot.lineplot(fxn, 'title', 'y_label', 'y_units', 'w', ['b', 'g', 'r'], 3.0)

        # Run Scheduler for 5 seconds
        Scheduler.start(5)

        # Remove lineplots from Scheduler
        Plot.clear()
```

1.19 bstem.sensor module

class bstem.sensor.**Accelerometer**

Bases: *bstem.sensor.Sensor*

Linear 3D Accelerometer.

enabled

Device enabled.

Returns True if device is enabled

Return type Boolean

value

Linear acceleration.

Returns last recorded linear acceleration along all axis (x, y, z) in g's

Return type tuple (long, long, long)

value_ts(*ts*)

Linear acceleration.

Returns last recorded linear acceleration along all axis (x, y, z) in g's

Return type tuple (long, long, long)

x

Acceleration along the x axis.

Returns last recorded linear acceleration along x axis in g's

Return type long

y

Acceleration along the y axis.

Returns last recorded linear acceleration along y axis in g's

Return type long

z

Acceleration along the z axis.

Returns last recorded linear acceleration along z axis in g's

Return type long

class bstem.sensor.**Barometer**
Bases: *bstem.sensor.Sensor*

Absolute pressure barometer.

enabled
Device enabled.

Returns True if device is enabled

Return type Boolean

pressure
Absolute pressure.

Returns last recorded absolute pressure

Return type long

value
Absolute pressure.

Returns last recorded absolute pressure

Return type tuple (long)

class bstem.sensor.**Gyroscope**
Bases: *bstem.sensor.Sensor*

3 axis Gyroscope.

DPS_2000 = 2000

DPS_250 = 250

DPS_500 = 500

dps
Degrees per Second.

Returns gyro sensitivity (250/500/2000 dps)

Return type Boolean

enabled
Device enabled.

Returns True if device is enabled

Return type Boolean

value
Angular rate.

Returns last recorded angular rate (x, y, z) in rads/s

Return type tuple (long, long, long)

value_ts (*ts*)
Angular rate.

Returns last recorded angular rate (x, y, z) in rads/s

Return type tuple (long, long, long)

x
Angular rate along the x axis.

Returns last recorded angular rate along x axis in rads/s

Return type long

y

Angular rate along the y axis.

Returns last recorded angular rate along y axis in rads/s

Return type long

z

Angular rate along the z axis.

Returns last recorded angular rate along z axis in rads/s

Return type long

class bstem.sensor.**Magnetometer**(*calibrated=False*)

Bases: *bstem.sensor.Sensor*

3D magnetic sensor :param calibrated: returns calibrated readings when True, else returns raw values from sensor. :type calibrated: Boolean

For calibrating magnetometer you can use the script compass_calibration.py in bstem. Calibrated magnetometer data is stored in /usr/share/bstem/.compass_calibration.log

calibration_file_name = '/usr/share/bstem/.compass_calibration.log'

enabled

Device enabled.

Returns True if device is enabled

Return type Boolean

raw_value

Orientation.

Returns last recorded orientation (x, y, z)

Return type tuple (long, long, long)

value

Orientation.

Returns last recorded orientation (x, y, z)

Return type tuple (long, long, long)

x

Orientation along the x axis.

Returns last recorded orientation along x axis

Return type long

y

Orientation along the y axis.

Returns last recorded orientation along y axis

Return type long

z

Orientation along the z axis.

Returns last recorded orientation along z axis

Return type long

class bstem.sensor.**Sensor**
Bases: object

Abstract base class for all sensors.

timestamp(*name*)

value()
Return tuple of parameter values for this sensor.

1.20 bstem.sonar module

class bstem.sonar.**MaxSonar**(*cord*, *analog_pin*, *sonar_class='XL'*, *supply_voltage=5*, *filter_type='MEDIAN'*, *buffer_length=5*)
Bases: *bstem.analog_filter.AnalogFilter*

MaxSonar code running on bstem using the analog input for sampling the range data. PWM mode is not supported currently. The sonar need to be connected to one of the bstem analog pins.

Filter types: MEDIAN, MODE, NONE (last sample only)

Tested on:

1. http://www.maxbotix.com/Ultrasonic_Sensors/MB1200.htm

Currently supported sonar class is XL. Will be extended to other types shortly.

class bstem.sonar.**MaxSonarArray**(*cord*, *analog_pins*, *gpio_strobe_pin=4*, *sonar_class='XL'*, *supply_voltage=5*)
Bases: object

MaxSonarArray triggering mechanism using GPIO pins of Bstem. Only supports all sonars of same type and same supply voltage.

TODO: Allow sonar_class as an array of different type and different voltage

specify the pins to which sonar analog signal is connected. It is recommended to connect the gpio[4] pin to the sonar RX pin to start the convertion process.

distance
returns a list of filtered distance for the sonar array

get_last_sample()
Returns a list of last sample from the sonar array

get_samples()
returns all the recent samples collected from the sonar array. Each element of the list is a buffer or history of current reading. Useful for user specific custom functions.

raw_value
Samples and returns the raw value from sonar adc without doing any cms conversion

1.21 bstem.tracker module

class bstem.tracker.**CAMShiftTracker**(*initial_bb, img_curr, histSize=[16, 16]*)
Tracker Based on Color Histogram and CAMShift. Initialize the tracker by specifying a bounding box on the initial image. This will be the target.

The tracker uses the HSV colorspace and a 2D histogram is constructed for the target. One needs to specify the number of bins in the two axes of this histogram.

Initialize a tracker. :param initial_bb: a list [x,y,w,h] where x and y is the coordinate of the upper-left corner of the bounding box, w and h is the width and height. :type initial_bb: list :param img_curr: the initial image in BGR :type img_curr: Numpy array :param histSize: the number of bins in the histogram :type histSize: list, default to [16,16]

update (*img_curr*)
Update the tracker

Parameters **img_curr** (*Numpy array*) – new image in BGR

1.22 bstem.video module

class bstem.video.**Video** (*callback*, *width=640*, *height=480*)
Bases: object

This class provides streaming video from the bstem to a web page via the Html5 mjpeg video tag (currently supported in Chrome and Firefox). Initialize the video object with a callback function that generates a single frame each time it is called and returns it as a numpy array e.g. to send a video stream from the right camera:

def video_callback(): _, img = cam.snapshot() img = cv2.cvtColor(np.asarray(img), cv2.COLOR_RGB2BGR) return img

vid = Video(video_callback)

to start generating and encoding frames:

vid.setup()

Each video frame can then be retrieved with:

vid.next()

See the video_basic example for complete python and html to stream a single camera.

Initialize streaming video.

Parameters

- **callback** (*numpy array*) – callback to retrieve each video frame
- **width** (*int (default 640)*) – width of video frame
- **height** (*int (default 480)*) – height of video frame

next ()
Return next encoded video frome.

setup ()
Start streaming video

shutdown ()
Stop streaming video

streaming
Return true if currently generating and encoding frames.

1.23 bstem.video_encoder module

class bstem.video_encoder.**VideoEncoder**(*width=1280*, *height=720*, *bitrate=3000000*, *max_im_queued=1*, *max_frames_queued=1*, *capture_type='COMPRESSED'*, *fps=1*, *grayscale=False*)

Bases: object

This class provides convenient access to the Video Encoder. It is able to take aribrary images and return compressed, uncompressed or motion data. It is currently most used for motion information but could be used for streaming compressed data.

Parameters

- **width** (*int*) – width of the images to be passed in must be a multiple of 16
- **height** (*int*) – height of the images to be passed in must be a multiple of 16
- **bitrate** (*int*) – the bitrate used for encoding
- **max_im_queued** (*int*) – the size of the image queue, used by add_image
- **max_frames_queued** (*int*) – the size of the frame queue, used by get_frame
- **capture_type** – if "FILE" the compressed data will be written to an .mp4 file.

Otherwise, this indicates the format of data that will be available when calling get_frame() :type capture_type: string: "FILE", "COMPRESSED", "DECOMPRESSED" OR "MOTION"

add_image(*im*, *block=True*, *timeout=10*)
im is the image to be encoded. It must be an instance of ImageWrapper block indicates if add_image should wait until there is space in the image queue timeout indicates how long to wait if block is True, in seconds and 0 means indefinitely

atexit()
helper method to handle python shutdown with extra threads correctly

get_frame(*block=False*, *timeout=1*)
Gets a frame. block indicates if get_frame should wait until there is a frame in the frame queue timeout indicates how long to wait if block is True, in seconds and 0 means indefinitely

returns frame as a numpy array, timestamp or None,None if no frame is available

set_filename(*filename*)

start()
must be called before calling add_image or get_frame

stop()
should be called when done processing for now, but may start up again

1.24 Module contents

b

A

B

C

D

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

U

V

W

X

Y

Z