. Documentation

Release 2020.2.2

Author

Feb 27, 2020

Robot Programming

This is a python implementation of the kauailabs NavX library.

Note: The RobotPy project is not associated with or endorsed by kauailabs

CHAPTER 1

NavX API

This is not installed on the Robot by default. For installation instructions, see robotpy-navx install docs.

1.1 NavX AHRS Interface

class navx.**AHRS**(**args*, ***kwargs*)

Bases: wpilib.SendableBase, wpilib.ErrorBase, wpilib.interfaces.PIDSource, wpilib.interfaces.Gyro

Overloaded function.

1. __init__(self: navx._navx.AHRS, spi_port_id: wpilib._wpilib.SPI.Port) -> None
2. __init__(self: navx._navx.AHRS, i2c_port_id: wpilib._wpilib.I2C.Port) -> None
3. __init__(self: navx._navx.AHRS, serial_port_id: wpilib._wpilib.SerialPort.Port) -> None
4. __init__(self: navx._navx.AHRS, spi_port_id: wpilib._wpilib.SPI.Port, update_rate_hz: int) -> None
5. __init__(self: navx._navx.AHRS, spi_port_id: wpilib._wpilib.SPI.Port, spi_bitrate: int, update_rate_hz: int) -> None
6. __init__(self: navx._navx.AHRS, i2c_port_id: wpilib._wpilib.I2C.Port, update_rate_hz: int) -> None
7. __init__(self: navx._navx.AHRS, serial_port_id: wpilib._wpilib.SerialPort.Port, data_type: navx._navx.AHRS.SerialDataType, update_rate_hz: int) -> None

class BoardAxis(*arg0: int*) → None

Bases: pybind11_builtins.pybind11_object

Members:

kBoardAxisX

kBoardAxisY

kBoardAxisZ

kBoardAxisX = BoardAxis.kBoardAxisX

kBoardAxisY = BoardAxis.kBoardAxisY

kBoardAxisZ = BoardAxis.kBoardAxisZ

name
(self: handle) -> str

class BoardYawAxis() → None
Bases: pybind11_builtins.pybind11_object

board_axis

up

class SerialDataType(*arg0: int*) → None
Bases: pybind11_builtins.pybind11_object

Members:

kProcessedData : (default): 6 and 9-axis processed data

kRawData : unprocessed data from each individual sensor

kProcessedData = SerialDataType.kProcessedData

kRawData = SerialDataType.kRawData

name
(self: handle) -> str

calibrate() → None
Does nothing

static create_i2c(*port: wpilib._wpilib.I2C.Port = Port.kMXP*, *update_rate_hz: int = 60*) → navx._navx.AHRS
Constructs the AHRS class using I2C communication, overriding the default update rate with a custom rate which may be from 4 to 100, representing the number of updates per second sent by the sensor.

This constructor should be used if communicating via I2C.

Note: Increasing the update rate may increase the CPU utilization.

Parameters **port** (I2C.Port :param update_rate_hz: Custom Update Rate (Hz)) – I2C Port to use

static create_spi(*port: wpilib._wpilib.SPI.Port = Port.kMXP*, *bitrate: int = 500000*, *update_rate_hz: int = 60*) → navx._navx.AHRS
Constructs the AHRS class using SPI communication, overriding the default update rate with a custom rate which may be from 4 to 100, representing the number of updates per second sent by the sensor.

This constructor allows the specification of a custom SPI bitrate, in bits/second.

Note: Increasing the update rate may increase the CPU utilization.

Parameters

- **port** (SPI.Port) – SPI Port to use
- **spi_bitrate** – SPI bitrate (Maximum: 2,000,000)

- **update_rate_hz** – Custom Update Rate (Hz)

enableBoardlevelYawReset(*enable: bool*) → None
Enables or disables board-level yaw zero (reset) requests. Board-level yaw resets are processed by the sensor board and the resulting yaw angle may not be available to the client software until at least 2 update cycles have occurred. Board-level yaw resets however do maintain synchronization between the yaw angle and the sensor-generated Quaternion and Fused Heading values.

Conversely, Software-based yaw resets occur instantaneously; however, Software- based yaw resets do not update the yaw angle component of the sensor-generated Quaternion values or the Fused Heading values.

enableLogging(*enable: bool*) → None
Enables or disables logging (via Console I/O) of AHRS library internal behaviors, including events such as transient communication errors.

getAccelFullScaleRangeG() → int
Returns the sensor full scale range (in G) of the X, Y and X-axis accelerometers.

Returns accelerometer full scale range in G.

getActualUpdateRate() → int
Returns the navX-Model device's currently configured update rate. Note that the update rate that can actually be realized is a value evenly divisible by the navX-Model device's internal motion processor sample clock (200Hz). Therefore, the rate that is returned may be lower than the requested sample rate.

The actual sample rate is rounded down to the nearest integer that is divisible by the number of Digital Motion Processor clock ticks. For instance, a request for 58 Hertz will result in an actual rate of 66Hz (200 / (200 / 58), using integer math.

Returns Returns the current actual update rate in Hz (cycles per second).

getAltitude() → float
Returns the current altitude, based upon calibrated readings from a barometric pressure sensor, and the currently-configured sea-level barometric pressure [navX Aero only]. This value is in units of meters.

Note: This value is only valid sensors including a pressure sensor. To determine whether this value is valid, see *isAltitudeValid()*.

Returns Returns current altitude in meters (as long as the sensor includes an installed on-board pressure sensor).

getAngle() → float
Returns the total accumulated yaw angle (Z Axis, in degrees) reported by the sensor.

Note: The angle is continuous, meaning it's range is beyond 360 degrees. This ensures that algorithms that wouldn't want to see a discontinuity in the gyro output as it sweeps past 0 on the second time around.

Note that the returned yaw value will be offset by a user-specified offset value this user-specified offset value is set by invoking the zeroYaw() method.

Returns The current total accumulated yaw angle (Z axis) of the robot in degrees. This heading is based on integration of the returned rate from the Z-axis (yaw) gyro.

getAngleAdjustment() → float
Returns the currently configured adjustment angle. See *setAngleAdjustment()* for more details.

If this method returns 0 degrees, no adjustment to the value returned via *getAngle()* will occur. :returns: adjustment, in degrees (range: -360 to 360)

getBarometricPressure() → float
Returns the current barometric pressure, based upon calibrated readings from the onboard pressure sensor. This value is in units of millibar.

Note: This value is only valid for a navX Aero. To determine whether this value is valid, see *isAltitudeValid()*.

Returns Returns current barometric pressure (navX Aero only).

getBoardYawAxis() → navx._navx.AHRS.BoardYawAxis
Returns information regarding which sensor board axis (X,Y or Z) and direction (up/down) is currently configured to report Yaw (Z) angle values. NOTE: If the board firmware supports Omnimount, the board yaw axis/direction are configurable.

For more information on Omnimount, please see:

http://navx-mxp.kauailabs.com/navx-mxp/installation/omnimount/

Returns The currently-configured board yaw axis/direction as a tuple of (up, axis). Up can be True/False, axis is 'x', 'y', or 'z')

getByteCount() → float
Returns the count in bytes of data received from the sensor. This could can be useful for diagnosing connectivity issues.

If the byte count is increasing, but the update count (see *getUpdateCount()*) is not, this indicates a software misconfiguration.

Returns The number of bytes received from the sensor.

getCompassHeading() → float
Returns the current tilt-compensated compass heading value (in degrees, from 0 to 360) reported by the sensor.

Note that this value is sensed by a magnetometer, which can be affected by nearby magnetic fields (e.g., the magnetic fields generated by nearby motors).

Before using this value, ensure that (a) the magnetometer has been calibrated and (b) that a magnetic disturbance is not taking place at the instant when the compass heading was generated. :returns: The current tilt-compensated compass heading, in degrees (0-360).

getDisplacementX() → float
Returns the displacement (in meters) of the X axis since resetDisplacement() was last invoked [Experimental].

Note: This feature is experimental. Displacement measures rely on double-integration of acceleration values from MEMS accelerometers which yield "noisy" values. The resulting displacement are not known to be very accurate, and the amount of error increases quickly as time progresses.

Returns Displacement since last reset (in meters).

getDisplacementY() → float
Returns the displacement (in meters) of the Y axis since resetDisplacement() was last invoked [Experimental].

Note: This feature is experimental. Displacement measures rely on double-integration of acceleration values from MEMS accelerometers which yield "noisy" values. The resulting displacement are not known to be very accurate, and the amount of error increases quickly as time progresses.

Returns Displacement since last reset (in meters).

getDisplacementZ() → float
Returns the displacement (in meters) of the Z axis since resetDisplacement() was last invoked [Experimental].

Note: This feature is experimental. Displacement measures rely on double-integration of acceleration values from MEMS accelerometers which yield "noisy" values. The resulting displacement are not known to be very accurate, and the amount of error increases quickly as time progresses.

Returns Displacement since last reset (in meters).

getFirmwareVersion() → str
Returns the version number of the firmware currently executing on the sensor.

To update the firmware to the latest version, please see:

http://navx-mxp.kauailabs.com/navx-mxp/support/updating-firmware/

Returns The firmware version in the format [MajorVersion].[MinorVersion]

getFusedHeading() → float
Returns the "fused" (9-axis) heading.

The 9-axis heading is the fusion of the yaw angle, the tilt-corrected compass heading, and magnetic disturbance detection. Note that the magnetometer calibration procedure is required in order to achieve valid 9-axis headings.

The 9-axis Heading represents the sensor's best estimate of current heading, based upon the last known valid Compass Angle, and updated by the change in the Yaw Angle since the last known valid Compass Angle. The last known valid Compass Angle is updated whenever a Calibrated Compass Angle is read and the sensor has recently rotated less than the Compass Noise Bandwidth (~2 degrees).

Returns Fused Heading in Degrees (range 0-360)

getGyroFullScaleRangeDPS() → int
Returns the sensor full scale range (in degrees per second) of the X, Y and X-axis gyroscopes.

Returns gyroscope full scale range in degrees/second.

getLastSensorTimestamp() → int
Returns the sensor timestamp corresponding to the last sample retrieved from the sensor. Note that this sensor timestamp is only provided when the Register-based IO methods (SPI, I2C) are used; sensor timestamps are not provided when Serial-based IO methods (TTL UART, USB) are used.

Returns The sensor timestamp (in ms) corresponding to the current AHRS sensor data.

Return type int

getPitch() → float
Returns the current pitch value (in degrees, from -180 to 180) reported by the sensor. Pitch is a measure of rotation around the X Axis.

Returns The current pitch value in degrees (-180 to 180).

getPressure() → float
Returns the current barometric pressure (in millibar) [navX Aero only].

This value is valid only if a barometric pressure sensor is onboard.

Returns Returns the current barometric pressure (in millibar).

getQuaternionW() → float
Returns the imaginary portion (W) of the Orientation Quaternion which fully describes the current sensor orientation with respect to the reference angle defined as the angle at which the yaw was last "zeroed".

Each quaternion value (W,X,Y,Z) is expressed as a value ranging from -2 to 2. This total range (4) can be associated with a unit circle, since each circle is comprised of 4 PI Radians.

For more information on Quaternions and their use, please see this definition.

Returns Returns the imaginary portion (W) of the quaternion.

getQuaternionX() → float
Returns the real portion (X axis) of the Orientation Quaternion which fully describes the current sensor orientation with respect to the reference angle defined as the angle at which the yaw was last "zeroed".

Each quaternion value (W,X,Y,Z) is expressed as a value ranging from -2 to 2. This total range (4) can be associated with a unit circle, since each circle is comprised of 4 PI Radians.

For more information on Quaternions and their use, please see this description.

Returns Returns the real portion (X) of the quaternion.

getQuaternionY() → float
Returns the real portion (Y axis) of the Orientation Quaternion which fully describes the current sensor orientation with respect to the reference angle defined as the angle at which the yaw was last "zeroed".

Each quaternion value (W,X,Y,Z) is expressed as a value ranging from -2 to 2. This total range (4) can be associated with a unit circle, since each circle is comprised of 4 PI Radians.

For more information on Quaternions and their use, please see:

https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

Returns Returns the real portion (X) of the quaternion.

getQuaternionZ() → float
Returns the real portion (Z axis) of the Orientation Quaternion which fully describes the current sensor orientation with respect to the reference angle defined as the angle at which the yaw was last "zeroed".

Each quaternion value (W,X,Y,Z) is expressed as a value ranging from -2 to 2. This total range (4) can be associated with a unit circle, since each circle is comprised of 4 PI Radians.

For more information on Quaternions and their use, please see:

https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

Returns Returns the real portion (X) of the quaternion.

getRate() → float
Return the rate of rotation of the yaw (Z-axis) gyro, in degrees per second.

The rate is based on the most recent reading of the yaw gyro angle.

Returns The current rate of change in yaw angle (in degrees per second)

getRawAccelX() → float
Returns the current raw (unprocessed) X-axis acceleration rate (in G).

Note: this value is unprocessed, and should only be accessed by advanced users. This raw value has not had acceleration due to gravity removed from it, and has not been rotated to the world reference frame. Gravity-corrected, world reference frame-corrected X axis acceleration data is accessible via the *getWorldLinearAccelX()* method.

Returns Returns the current acceleration rate (in G).

getRawAccelY() → float
Returns the current raw (unprocessed) Y-axis acceleration rate (in G).

Note: this value is unprocessed, and should only be accessed by advanced users. This raw value has not had acceleration due to gravity removed from it, and has not been rotated to the world reference frame. Gravity-corrected, world reference frame-corrected Y axis acceleration data is accessible via the *getWorldLinearAccelY()* method.

Returns Returns the current acceleration rate (in G).

getRawAccelZ() → float
Returns the current raw (unprocessed) Z-axis acceleration rate (in G).

Note: this value is unprocessed, and should only be accessed by advanced users. This raw value has not had acceleration due to gravity removed from it, and has not been rotated to the world reference frame. Gravity-corrected, world reference frame-corrected Z axis acceleration data is accessible via the *getWorldLinearAccelZ()* method.

Returns Returns the current acceleration rate (in G).

getRawGyroX() → float
Returns the current raw (unprocessed) X-axis gyro rotation rate (in degrees/sec).

Note: This value is un-processed, and should only be accessed by advanced users. Typically, rotation about the X Axis is referred to as "Pitch". Calibrated and Integrated Pitch data is accessible via the *getPitch()* method.

Returns Returns the current rotation rate (in degrees/sec).

getRawGyroY() → float
Returns the current raw (unprocessed) Y-axis gyro rotation rate (in degrees/sec).

Note: This value is un-processed, and should only be accessed by advanced users. Typically, rotation about the T Axis is referred to as "Roll". Calibrated and Integrated Pitch data is accessible via the

getRoll() method.

Returns Returns the current rotation rate (in degrees/sec).

getRawGyroZ() → float
Returns the current raw (unprocessed) Z-axis gyro rotation rate (in degrees/sec).

Note: This value is un-processed, and should only be accessed by advanced users. Typically, rotation about the T Axis is referred to as "Yaw". Calibrated and Integrated Pitch data is accessible via the *getYaw()* method.

Returns Returns the current rotation rate (in degrees/sec).

getRawMagX() → float
Returns the current raw (unprocessed) X-axis magnetometer reading (in uTesla).

Note: this value is unprocessed, and should only be accessed by advanced users. This raw value has not been tilt-corrected, and has not been combined with the other magnetometer axis data to yield a compass heading. Tilt-corrected compass heading data is accessible via the *getCompassHeading()* method.

Returns Returns the mag field strength (in uTesla).

getRawMagY() → float
Returns the current raw (unprocessed) Y-axis magnetometer reading (in uTesla).

Note: this value is unprocessed, and should only be accessed by advanced users. This raw value has not been tilt-corrected, and has not been combined with the other magnetometer axis data to yield a compass heading. Tilt-corrected compass heading data is accessible via the *getCompassHeading()* method.

Returns Returns the mag field strength (in uTesla).

getRawMagZ() → float
Returns the current raw (unprocessed) Z-axis magnetometer reading (in uTesla).

Note: this value is unprocessed, and should only be accessed by advanced users. This raw value has not been tilt-corrected, and has not been combined with the other magnetometer axis data to yield a compass heading. Tilt-corrected compass heading data is accessible via the *getCompassHeading()* method.

Returns Returns the mag field strength (in uTesla).

getRequestedUpdateRate() → int
Returns the currently requested update rate. rate. Note that not every update rate can actually be realized, since the actual update rate must be a value evenly divisible by the navX-Model device's internal motion processor sample clock (200Hz).

To determine the actual update rate, use the *getActualUpdateRate()* method.

Returns Returns the requested update rate in Hz (cycles per second).

getRoll() → float
Returns the current roll value (in degrees, from -180 to 180) reported by the sensor. Roll is a measure of rotation around the X Axis.

Returns The current roll value in degrees (-180 to 180).

getTempC() → float
Returns the current temperature (in degrees centigrade) reported by the sensor's gyro/accelerometer circuit.

This value may be useful in order to perform advanced temperature- correction of raw gyroscope and accelerometer values.

Returns The current temperature (in degrees centigrade).

getUpdateCount() → float
Returns the count of valid updates which have been received from the sensor. This count should increase at the same rate indicated by the configured update rate.

Returns The number of valid updates received from the sensor.

getVelocityX() → float
Returns the velocity (in meters/sec) of the X axis [Experimental].

Note: This feature is experimental. Velocity measures rely on integration of acceleration values from MEMS accelerometers which yield "noisy" values. The resulting velocities are not known to be very accurate.

Returns Current Velocity (in meters/squared).

getVelocityY() → float
Returns the velocity (in meters/sec) of the Y axis [Experimental].

Note: This feature is experimental. Velocity measures rely on integration of acceleration values from MEMS accelerometers which yield "noisy" values. The resulting velocities are not known to be very accurate.

Returns Current Velocity (in meters/squared).

getVelocityZ() → float
Returns the velocity (in meters/sec) of the X axis [Experimental].

Note: This feature is experimental. Velocity measures rely on integration of acceleration values from MEMS accelerometers which yield "noisy" values. The resulting velocities are not known to be very accurate.

Returns Current Velocity (in meters/squared).

getWorldLinearAccelX() → float
Returns the current linear acceleration in the X-axis (in G).

World linear acceleration refers to raw acceleration data, which has had the gravity component removed, and which has been rotated to the same reference frame as the current yaw value. The resulting value represents the current acceleration in the x-axis of the body (e.g., the robot) on which the sensor is mounted.

Returns Current world linear acceleration in the X-axis (in G).

getWorldLinearAccelY() → float
Returns the current linear acceleration in the Y-axis (in G).

World linear acceleration refers to raw acceleration data, which has had the gravity component removed, and which has been rotated to the same reference frame as the current yaw value. The resulting value represents the current acceleration in the Y-axis of the body (e.g., the robot) on which the sensor is mounted.

Returns Current world linear acceleration in the Y-axis (in G).

getWorldLinearAccelZ() → float
Returns the current linear acceleration in the Z-axis (in G).

World linear acceleration refers to raw acceleration data, which has had the gravity component removed, and which has been rotated to the same reference frame as the current yaw value. The resulting value represents the current acceleration in the Z-axis of the body (e.g., the robot) on which the sensor is mounted.

Returns Current world linear acceleration in the Z-axis (in G).

getYaw() → float
Returns the current yaw value (in degrees, from -180 to 180) reported by the sensor. Yaw is a measure of rotation around the Z Axis (which is perpendicular to the earth).

Note that the returned yaw value will be offset by a user-specified offset value this user-specified offset value is set by invoking the zeroYaw() method.

Returns The current yaw value in degrees (-180 to 180).

isAltitudeValid() → bool
Indicates whether the current altitude (and barometric pressure) data is valid. This value will only be true for a sensor with an onboard pressure sensor installed.

If this value is false for a board with an installed pressure sensor, this indicates a malfunction of the onboard pressure sensor.

Returns Returns true if a working pressure sensor is installed.

isBoardlevelYawResetEnabled() → bool
Returns true if Board-level yaw resets are enabled. Conversely, returns false if Software-based yaw resets are active.

Returns true if Board-level yaw resets are enabled.

isCalibrating() → bool
Returns true if the sensor is currently performing automatic gyro/accelerometer calibration. Automatic calibration occurs when the sensor is initially powered on, during which time the sensor should be held still, with the Z-axis pointing up (perpendicular to the earth).

Note: During this automatic calibration, the yaw, pitch and roll values returned may not be accurate.

Once calibration is complete, the sensor will automatically remove an internal yaw offset value from all reported values.

Returns Returns true if the sensor is currently automatically calibrating the gyro and accelerometer sensors.

isConnected() → bool
Indicates whether the sensor is currently connected to the host computer. A connection is considered established whenever communication with the sensor has occurred recently.

Returns Returns true if a valid update has been recently received from the sensor.

isMagneticDisturbance() → bool
Indicates whether the current magnetic field strength diverges from the calibrated value for the earth's magnetic field by more than the currently- configured Magnetic Disturbance Ratio.

This function will always return false if the sensor's magnetometer has not yet been calibrated (see *isMagnetometerCalibrated()*).

Returns true if a magnetic disturbance is detected (or the magnetometer is uncalibrated).

isMagnetometerCalibrated() → bool
Indicates whether the magnetometer has been calibrated.

Magnetometer Calibration must be performed by the user.

Note that if this function does indicate the magnetometer is calibrated, this does not necessarily mean that the calibration quality is sufficient to yield valid compass headings.

Returns Returns true if magnetometer calibration has been performed.

isMoving() → bool
Indicates if the sensor is currently detecting motion, based upon the X and Y-axis world linear acceleration values. If the sum of the absolute values of the X and Y axis exceed a "motion threshold", the motion state is indicated.

Returns Returns true if the sensor is currently detecting motion.

isRotating() → bool
Indicates if the sensor is currently detecting motion, based upon the X and Y-axis world linear acceleration values. If the sum of the absolute values of the X and Y axis exceed a "motion threshold", the motion state is indicated.

Returns Returns true if the sensor is currently detecting motion.

reset() → None
Reset the Yaw gyro.

Resets the Gyro Z (Yaw) axis to a heading of zero. This can be used if there is significant drift in the gyro and it needs to be recalibrated after it has been running.

resetDisplacement() → None
Zeros the displacement integration variables. Invoke this at the moment when integration begins.

setAngleAdjustment(*angle: float*) → None
Sets an amount of angle to be automatically added before returning a angle from the *getAngle()* method. This allows users of the `getAngle` method to logically rotate the sensor by a given amount of degrees.

NOTE 1: The adjustment angle is **only** applied to the value returned from `getAngle` - it does not adjust the value returned from *getYaw()*, nor any of the quaternion values.

NOTE 2: The adjustment angle is **not** automatically cleared whenever the sensor yaw angle is reset.

If not set, the default adjustment angle is 0 degrees (no adjustment).

Parameters **adjustment** – in degrees (range: -360 to 360)

updateDisplacement(*accel_x_g: float*, *accel_y_g: float*, *update_rate_hz: int*, *is_moving: bool*) → None

zeroYaw() → None
Sets the user-specified yaw offset to the current yaw value reported by the sensor.

This user-specified yaw offset is automatically subtracted from subsequent yaw values reported by the getYaw() method.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

K

N

R

S

U

Z