GyroLib 4.0 Documentation

Created by Michael Bartlett, Reza Torbati, and Andrew Zhang for Norman Advanced Robotics

CONTENTS

CHANGE LOG

- Created thread that keeps track of absolute theta

- Added a mode variable that changes all functions between absolute and relative
- Integration now uses seconds() function rather than msleep() which leads to greatly increased accuracy
- Added advanced versions of functions with more arguments
- Reduced the amounts of arguments in most base functions
- Added in a timeout variable used in turn functions to stop the turn if the robot gets stuck
- Changed drive_with_gyro() to use proportional controls rather than a curve fitter to increase simplicity, reduce computational intensity, and increase adjustability
- Added drive_until_analog() and drive_until_digital() functions
- Changed turn_with_gyro() to use PID controls. Renamed the old turn_with_gyro() to turn_with_gyro_overshoot()
- Added abbreviations to almost every function
- Killed “changes” part of header in favor of this change log
- Made the same changes to the create functions, but they are entirely untested and untuned

WALLABY FUNCTIONS

- **declare_motors**(int lmotor, int rmotor)
    - This function is used to declare which two motors are the driving motors at the start of a program
    - lmotor - the motor driving the left wheel
    - rmotor - the motor driving the right wheel
- **declare_degrees**(double degree)
    - This function allows for functions to take normal degrees rather than the kipr units of angles
    - degree - the number of kipr units in 90 normal degrees. Can be found with calibrate_degrees()
- **declare_motors_and_degrees**(int lmotor, int rmotor, double degree)
    - This function combines the declare_motors() and declare_degrees() functions into one shorter function
    - lmotor - the motor driving the left wheel

    - rmotor - the motor driving the right wheel
    - degree - the number of kipr units in 90 normal degrees. Can be found with calibrate_degrees()
- **calibrate_degrees**()
    - This function prints out a list of steps to follow and then returns the number of kipr units in 90 degrees. This measurement should be copied into a variable that can be used in the main program
- **calibrate_gyro_advanced**(int sleep_time, int sample_size)
    - This function calibrates the gyroscope and sets the bias. This should be run at least once at the beginning of every run and should only be run when the robot is not moving.
    - This can be abbreviated as cga()
    - sleep_time - the amount of time, in milliseconds, the robot stops before calibrating. This is not needed if the robot is not moving before the function is called
    - sample_size - how many samples are taken to calibrate the gyroscope. This is usually 5000
- **calibrate_gyro**()
    - This function calibrates the gyroscope and sets the bias variable. This should be run at least once at the beginning of every run and should only be run when the robot is not moving.
    - This can be abbreviated as cg()
    - This is the same as calibrate_gyro_advanced(0, 5000)
- **start_theta_tracker**()
    - This function starts the thread that keeps track of the absolute_theta variable. This should be called at the beginning of every run before the gyroscope is used.
- **stop_theta_tracker**()
    - This function terminates the thread that keeps track of absolute theta. This should only be used in case the number of threads running must be managed or if you don't want absolute_theta to be updated at a certain part of the program.
- **initialize_gyro**(int lmotor, int rmotor, int degree)

    - This function combines the declare_motors_and_degrees(), calibrate_gyro(), and start_theta_tracker() functions all into one function that deals with all the setup needed to use the gyroscope
    - This can be abbreviated as ig()
- **get_axis**()
    - This function returns the axis variable
- **set_axis**(char value)
    - This function sets the axis variable to the given value
    - value - the value, either ‘x’, ‘y’, or ‘z’, to set the axis variable
- **get_absolute_theta**()
    - This function returns the absolute_theta variable in either kipr units or degrees depending on whether degrees have been declared
- **set_absolute_theta**(double theta)
    - This function sets the absolute_theta variable to the given value. This can be used to reset the tracker to prevent drift
    - theta - the angle to set the absolute_theta variable, in kipr units or degrees depending on whether degrees have been declared
- **get_mode**()
    - This function returns the current mode of the gyroscope with 0 being relative and 1 being absolute
- **set_mode**(int value)
    - This function changes the gyroscope mode between absolute and relative
    - value - the mode to switch to, with 0 being relative and 1 being absolute
- **get_timeout**()
    - This function returns the timeout variable in milliseconds
- **set_timeout**(int value)
    - This function changes the timeout variable to the given value
    - value - the value, in milliseconds, to set the timeout variable to
- **get_bias**()
    - This function returns the bias variable that is added to the gyroscope readings
- **arc_with_gyro**(int left_wheel_speed, int right_wheel_speed, double target_theta)
    - This function turns the two motors at a given speed until the desired angle

is reached with positive being counter-clockwise and negative being clockwise. This is less accurate than turn_with_gyro(), but it allows for arcs
    - This can be abbreviated as awg()
    - left_wheel_speed - the speed that the left wheel is set to
    - right_wheel_speed - the speed that the right wheel is set to
    - target_theta - the angle to turn to
- **turn_with_gyro_overshoot**(double correction_constant, double target_theta, double overshoot)
    - This function does a pivot turn to the given angle then continues to correct afterwards for a short time
    - This can be abbreviated as twgo()
    - correction_constant - The proportional constant that controls the speed and accuracy of the turn. This is usually around 12
    - target_theta - the angle to which the robot turns
    - overshoot - the amount of time, in milliseconds, that the turn continues after the desired angle is reached
- **turn_with_gyro_advanced**(double target_theta, double speed_limit, double pk, double ik, double dk)
    - This function does a pivot turn to a desired angle using pid controls
    - This can be abbreviated as twga()
    - target_theta - the angle to which the robot turns
    - speed_limit - the maximum speed, form -1500 to 1500, the robot will turn
    - pk - the proportional constant of correction. This is typically around 12
    - ik - the integral constant of correction. This is typically around 0
    - dk - the derivative constant of correction. This is typically around 0
    - Note: Proportional controls seem to be accurate enough and the remaining constants seem to have minimal effect for rapidly changing systems like a robot
- **turn_with_gyro**(double target_theta)
    - This function does a pivot turn to a desired angle using pid controls with most arguments already filled in
    - This can be abbreviated as twg()
    - target_theta - the angle to which the robot turns

    - This is the same as turn_with_gyro_advanced(target_theta, 1500, 12, 0, 0)
- **drive_with_gyro_advanced**(int speed, int time, double pk, int correction)
    - This function attempts to drive forward while maintaining the starting angle to drive straight.
    - This can be abbreviated as dwga()
    - speed - the speed at which to drive from -1500 to 1500
    - time - the amount of time to drive in milliseconds
    - pk - the proportional constant of correction. Determines how hard the robot tries to correct. Usually around 12
    - correction - a boolean variable that determines if a correction turn is made at the end of a drive. 0 doesn't make a turn and 1 does
- **drive_with_gyro**(int speed, int time)
    - This function attempts to drive forward while maintaining the starting angle to drive straight with some arguments already filled in
    - This can be abbreviated as dwg()
    - speed - the speed at which to drive from -1500 to 1500
    - time - the amount of time to drive in milliseconds
    - This is the same as drive_with_gyro_advanced(speed, time, 12, 0)
- **drive_until_analog_advanced**(int speed, int port, int target_value, double pk, double max_time)
    - This function drives straight using the gyro until an analog sensor reaches a target value
    - This can be abbreviated as duaa()
    - speed - the speed at which to drive from -1500 to 1500
    - port - the port of the analog sensor to check
    - target_value - the value that the sensor must exceed or go under to stop driving
    - pk - the proportional constant of correction. Determines how hard the robot tries to correct. Usually around 12
    - max_time - The amount of time, in milliseconds, the drive will last at max. Use this if you have problems with the sensor not triggering
- **drive_until_analog**(int speed, int port, int target_value)
    - This function drives straight using the gyro until an analog sensor reaches

    - a target value with some arguments already filled in
    - This can be abbreviated as dua()
    - speed - the speed at which to drive from -1500 to 1500
    - port - the port of the analog sensor to check
    - target_value - the value that the sensor must exceed or go under to stop driving
    - This is the same as [drive_until_analog_advanced(speed, port, target_value, 12, 120000)]()
- **drive_until_digital_advanced**(int speed, int port, double pk, double max_time)
    - This function drives straight using the gyro until a digital sensor changes its value
    - This can be abbreviated as duda()
    - speed - the speed at which to drive from -1500 to 1500
    - port - the port of the digital sensor to check
    - pk - the proportional constant of correction. Determines how hard the robot tries to correct. Usually around 12
    - max_time - The amount of time, in milliseconds, the drive will last at max, use this if you have problems with the sensor not triggering
- **drive_until_digital**(int speed, int port)
    - This function drives straight using the gyro until a digital sensor changes its value with some arguments already filled in
    - This can be abbreviated as dud()
    - speed - the speed at which to drive from -1500 to 1500
    - port - the port of the analog sensor to check
    - target_value - the value that the sensor must exceed or go under to stop driving
    - This is the same as [drive_until_digital_advanced(speed, port, 12, 120000)]()
- **drive_with_gyro_distance_advanced**(int speed, int target_distance, int motor, double pk, int correction)
    - This function uses the gyro to drive straight for a given distance
    - This can be abbreviated as dwgda()
    - speed - the speed at which to drive from -1500 to 1500
    - target_distance - the distance to drive with one rotation of the wheel being

    about 1400, but it varies by motor
  - motor - the port of the motor being used to measure the distance
  - pk - the proportional constant of correction. Determines how hard the robot tries to correct. Usually around 12
  - correction - a boolean variable that determines if a correction turn is made at the end of a drive. 0 doesn't make a turn and 1 does
- **drive_with_gyro_distance**(int speed, int target_distance, int motor)
  - This function uses the gyro to drive straight for a given distance with some arguments already filled in
  - This can be abbreviated as dwgd()
  - speed - the speed at which to drive from -1500 to 1500
  - target_distance - the distance to drive with one rotation of the wheel being about 1400, but it varies by motor
  - motor - the port of the motor being used to measure the distance
  - This is the same as drive_with_gyro_distance_advanced(speed, target_distance, motor, 12, 0)

CREATE FUNCTIONS

- **create_calibrate_gyro_advanced**(int sleep_time, int sample_size)
  - This function calibrates the gyroscope and sets the bias variable. This should be run at least once at the beginning of every run and should only be run when the robot is not moving. This functions the exact same as calibrate_gyro_advanced() but uses create_stop(). This should never be used if this is run before the robot begins to move
  - This can be abbreviated as ccga()
  - sleep_time - the amount of time, in milliseconds, the robot stops before calibrating. This is not needed if the robot is not moving before this is run
  - sample_size - how many samples are taken to calibrate the gyroscope. This is usually 5000
- **create_calibrate_gyro**()
  - This function calibrates the gyroscope and sets the bias with some arguments already filled in. This should be run at least once at the

beginning of every run and should only be run when the robot is not moving. This functions the exact same as calibrate_gyro() but uses create_stop(). This should never be used if this is run before the robot begins to move
    - This can be abbreviated as ccg()
    - This is the same as create_calibrate_gyro_advanced(0, 5000)

- **create_arc_with_gyro**(int left_wheel_speed, int right_wheel_speed, double target_theta)
    - This function turns the two motors at a given speed until the desired angle is reached with positive being counter-clockwise and negative being clockwise. This is less accurate than create_turn_with_gyro(), but it allows for arcs
    - This can be abbreviated as cawg()
    - left_wheel_speed - the speed that the left wheel is set to from -500 to 500
    - right_wheel_speed - the speed that the right wheel is set to from -500 to 500
    - target_theta - the angle to turn to
- **create_turn_with_gyro_overshoot**(double correction_constant, double target_theta, double overshoot)
    - This function does a pivot turn to the given angle then continues to correct afterwards for a short time
    - This can be abbreviated as ctwgo()
    - correction_constant - The proportional constant that controls the speed and accuracy of the turn. This is usually around 10
    - target_theta - the angle to which the robot turns
    - overshoot - the amount of time, in milliseconds, that the turn continues for after the desired angle is reached
- **create_turn_with_gyro_advanced**(double target_theta, double speed_limit, double pk, double ik, double dk)
    - This function makes a pivot turn to a given angle using pid controls.
    - This can be abbreviated as ctwga()
    - target_theta - the angle to turn to
    - speed_limit - the maximum speed, from -500 to 500, which the robot will turn

    - pk - the proportional constant of correction. This is typically around x
    - ik - the integral constant of correction. This is typically around x
    - dk - the derivative constant of correction. This is typically around x
- **create_turn_with_gyro**(double target_theta)
    - This function makes a pivot turn to a given angle using pid controls with some arguments already filled in.
    - This can be abbreviated as ctwg()
    - target_theta - the angle to turn to
    - This is the same as create_turn_with_gyro_advanced(target_theta, 500, 10, 0, 0)
- **create_drive_with_gyro_advanced**(int speed, int time, double pk, int correction)
    - This function attempts to drive forward while maintaining the starting angle to drive straight.
    - This can be abbreviated as cdwga()
    - speed - the speed, from -500 to 500, at which the robot drives
    - time - the time, in milliseconds, that the robot drives
    - pk - the proportional constant of correction. Determines how hard the robot tries to correct. Usually around x
    - correction - a boolean variable that determines if a correction turn is made at the end of a drive. 0 doesn't make a turn and 1 does
- **create_drive_with_gyro**(int speed, int time)
    - This function attempts to drive forward while maintaining the starting angle to drive straight with some arguments already filled in.
    - This can be abbreviated as cdwg()
    - speed - the speed, from -500 to 500, at which the robot drives
    - time - the time, in milliseconds, that the robot drives
    - This is the same as create_drive_with_gyro_advanced(speed, time, 10, 0)

VARIABLES

- double **absolute_theta**

    - This variable keeps track of the angle the robot is at relative to the angle it was when start_theta_tracker() was called
    - This can be set with set_absolute_theta() and retrieved with get_absolute_theta()
- char **axis**
    - This variable keeps track of which axis of the gyroscope is used for all gyroscope functions, either ‘x’, ‘y’, or ‘z’
    - This defaults to ‘z’
    - This can be set with set_axis() and can be retrieved with get_axis()
- double **bias**
    - This variable holds a number that is added to the gyroscope readings to help account for the drift of the gyroscope
    - This defaults to 0
    - This can be set with calibrate_gyro() and can be retrieved with get_bias()
- int **mode**
    - This variable keeps track of whether the gyroscope uses relative or absolute theta, with 0 being relative and 1 being absolute
    - This defaults to 0
    - This can be set with set_mode() and retrieved with get_mode()
- int **timeout**
    - This variable holds the time, in milliseconds, that any turning function can last. This can be used to prevent turns from getting stuck and stopping the entire program
    - This defaults to 120000
    - This can be set with set_timeout(int value) and can be retrieved with get_timeout()