Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

C **Encoder**	Class that implements an encoder
C **EngineSpeed**	Header for a class that implements a hall effect sensor to measure engine speed
C **Motor**	Class that implements a motor
C **PIDController**	Class that implements a PID Controller
C **Pin**	Header for a struct that implements a pin
C **WheelSpeed**	Header for a class that implements a hall effect sensor to measure wheel speed

Encoder Class Reference

Class that implements an encoder. More...

`#include <Encoder.h>`

Public Member Functions

	Encoder (**Pin** ENC_A, **Pin** ENC_B) Constructor which creates and initializes an encoder object. More...
uint16_t	**read** () Return the encoder position. More...
void	**zero** () Zero the encoder position. More...

Detailed Description

Class that implements an encoder.

This class allows the user to implement an encoder on either Timer D0, Event Channel 2 or Timer D1, Event Channel 3. Future updates will allow for the timer and event channel to be chosen independently by the user via parameters in the constructor. Each encoder object of this class can read from and written to independently.

Constructor & Destructor Documentation

◆ Encoder()

```
Encoder::Encoder ( Pin  ENC_A,
                   Pin  ENC_B
                 )
```

Constructor which creates and initializes an encoder object.

This constructor creates an encoder object with the given pins. It saves the pins and sets the pins for input. It sets QDPH0 and QDPH1 sening level. It sets up the event system depending on the encoder. Future updates will allow for the the timer and event channel to be chosen independently by the user via parameters in the constructor.

Parameters

ENC_A The first signal pin of the encoder.

ENC_B The second signal pin of the encoder.

Member Function Documentation

◆ read()

```
uint16_t Encoder::read ( )
```

Return the encoder position.

This function retrieves the timer count and returns it as an unsigned 16-bit number representing the encoder position.

Returns

The encoder position in ticks as an unsigned 16-bit number.

◆ zero()

```
void Encoder::zero ( )
```

Zero the encoder position.

This function sets the timer count representing the encoder position to zero.

The documentation for this class was generated from the following files:

- Encoder.h
- Encoder.cpp

EngineSpeed Class Reference

Header for a class that implements a hall effect sensor to measure engine speed. More...

`#include <EngineSpeed.h>`

Public Member Functions

	EngineSpeed (uint8_t triggers) Constructor which creates and initializes an engine speed object. More...
void	**calc** () Update the current time and previous time to calculate the engine speed. More...
uint16_t	**get** () Return the engine speed in rotations per second. More...

Detailed Description

Header for a class that implements a hall effect sensor to measure engine speed.

This class allows the user to implement a hall effect sensor to meausre engine speed. It relies on the user to set up an interrupt service routine to call the calculate function on the rising or falling edge of the signal from a hall effect sensor measuring engine speed. The engine speed is similar to the wheel speed library but also averages the thermodynamic cycle of a four-stroke engine (that is, it averages every two revolutions) to reduce the noise of the output signal.

Constructor & Destructor Documentation

◆ EngineSpeed()

EngineSpeed::EngineSpeed (uint8_t triggers)

Constructor which creates and initializes an engine speed object.

This constructor creates an wheel speed object with the given number of triggers. It saves the number of triggers and initializes the "previous" and "current" times.

Parameters

triggers The number of triggers per revolution of the wheel.

Member Function Documentation

◆ calc()

void EngineSpeed::calc ( )

Update the current time and previous time to calculate the engine speed.

This function stores the "current" time as the previous time and assign the current time to current time. These values can be used to calculate the wheel speed, but this calculation is left until the **get()** function is called to minimize the time spent in the interrupt service routing. The previous time is stored to the respective trigger position, then the position is incremented. If the position increases to a number greater than the number of triggers, the position is reset to zero.

◆ get()

uint16_t EngineSpeed::get ( )

Return the engine speed in rotations per second.

This function calculates the wheel speed using the stored previous time and stored current time. The time between triggers is multipled by the number of triggers to calculate the time required for one revolution in microseconds. The constant 1E6 is divided by this time to get rotations per second. The previous time retrieved is from the current trigger position.

Returns

The wheel speed in rotations per second as a float.

The documentation for this class was generated from the following files:

- **EngineSpeed.h**
- **EngineSpeed.cpp**

Motor Class Reference

Class that implements a motor. More...

`#include <Motor.h>`

Public Member Functions

	Motor (**Pin** INA, **Pin** INB, **Pin** PWM) Constructor which creates and initializes a motor object. More...
void	**init** () Configures the pins. More...
void	**setDutyCycle** (int8_t dutyCycle) Sets the duty cycle. More...

Detailed Description

Class that implements a motor.

This class allows the user to implement a motor. The motor will respond to a duty cycle between -100 and 100, with a duty cycle of 0 causing the motor to coast regardless of direction. Each motor object of this class can controlled independently.

Constructor & Destructor Documentation

◆ Motor()

```
Motor::Motor ( Pin  INA,
               Pin  INB,
               Pin  PWM
             )
```

Constructor which creates and initializes a motor object.

This constructor creates a motor object with the given pins.

Parameters

INA The first direction pin of the motor driver.

INB The second direction pin of the motor driver.

PWM The duty cycle pin of the motor driver.

Member Function Documentation

◆ init()

```
void Motor::init ( )
```

Configures the pins.

This function configures the direction pins (that is, INA and INB) and duty cycle pin (that is, PWM) as output pins.

◆ setDutyCycle()

```
void Motor::setDutyCycle ( int8_t  dutyCycle )
```

Sets the duty cycle.

This function takes a duty cycle between -100 and 100 to write as a PWM signal to the motor driver. It automatically writes to the direction pins depending on the sign of the duty cycle and scales the absolute value of the provided duty cycle to the PWM pin.

Parameters

dutyCycle the duty cycle the motor is set to

The documentation for this class was generated from the following files:

- Motor.h
- Motor.cpp

PIDController Class Reference

Class that implements a PID Controller. More...

`#include <PIDController.h>`

Public Member Functions

	PIDController (float Kp, float Ki, float Kd) Constructor which creates and initializes a motor object. More...
void	**setKp** (float Kp) Updates the proportional gain. More...
void	**setKi** (float Ki) Updates the integral gain. More...
void	**setKd** (float Kd) Updates the derivative gain. More...
void	**setSetpoint** (int32_t setpoint) Updates the setpoint. More...
void	**setLoSat** (int8_t loSat) Updates the low saturation limit. More...
void	**setHiSat** (int8_t hiSat) Updates the high saturation limit. More...
void	**calc** (int32_t measurement) Update the previous error, the error, the integral, and the derivative. More...
float	**get** () Update the saturation status and return the PID output. More...
void	**reset** () Reset the calculated integral and derivative to zero. More...

Detailed Description

Class that implements a PID Controller.

This class allows the user to implement a PID Controller. It is given a setpoint, low saturation, high saturation, and Kp, Ki, and Kd gain values. In addition to providing PID output, the class accounts for integral windup by disabling the integral term if the output is determined as saturated. The class can also serve as a P-Only, PI, or PD controller by setting the appropriate gains to 0, although this does result in some unnecessary calculations.

Constructor & Destructor Documentation

◆ PIDController()

```
PIDController::PIDController ( float  Kp,
                               float  Ki,
                               float  Kd
                             )
```

Constructor which creates and initializes a motor object.

This constructor creates a PID controller object with a given proportional gain, integral gain, and derivative gain. If a P-Only, PI, or PD controller is desired, set the other gains to zero.

Parameters

Kp The proportional gain for the controller.

Ki The integral gain for the controller.

Kd The derivative gain for the controller.

Member Function Documentation

◆ calc()

```
void PIDController::calc ( int32_t  measurement )
```

Update the previous error, the error, the integral, and the derivative.

This function updates the previous error, the error, the integral, and the derivative values. These values can be used to calculate the PID output and saturation status, but this calculation is left until the **get()** function is called to minimize the time spent in the interrupt service routine.

◆ get()

float PIDController::get ( )

Update the saturation status and return the PID output.

This function calculatest the PID output. It then compares the calculated output to the low saturation limit and the high saturation limit. If the PID output is outside of the saturation limits, the saturation status is set to true, else it is set to false. The PID output is then returned.

Returns

get The PID output as a float.

◆ reset()

void PIDController::reset ( )

Reset the calculated integral and derivative to zero.

This function resets the calculated integral and derivative to zero so that the PID controller can be re-entered without an affect from old data.

◆ setHiSat()

void PIDController::setHiSat (int8_t hiSat)

Updates the high saturation limit.

This function updates the high saturation limit to a new value during use.

Parameters

hiSat The new high saturation limit for the controller.

◆ setKd()

void PIDController::setKd (float Kd)

Updates the derivative gain.

This function updates the derivative gain to a new value during use.

Parameters

Kd The new derivative gain for the controller.

◆ setKi()

void PIDController::setKi (float Ki)

Updates the integral gain.

This function updates the integral gain to a new value during use.

Parameters

Ki The new integral gain for the controller.

◆ setKp()

void PIDController::setKp (float Kp)

Updates the proportional gain.

This function updates the proportional gain to a new value during use.

Parameters

Kp The new proportional gain for the controller.

◆ setLoSat()

void PIDController::setLoSat (int8_t loSat)

Updates the low saturation limit.

This function updates the low saturation limit to a new value during use.

Parameters

loSat The new low saturation limit for the controller.

◆ setSetpoint()

void PIDController::setSetpoint (int32_t setpoint)

Updates the setpoint.

This function updates the setpoint to a new value during use.

Parameters

setpoint The new proportional gain for the controller.

The documentation for this class was generated from the following files:

- **PIDController.h**
- **PIDController.cpp**

Pin Struct Reference

Header for a struct that implements a pin. More...

```
#include <Pin.h>
```

Public Attributes

PORT_t *	**PORT**
uint8_t	**PIN_BM**

Detailed Description

Header for a struct that implements a pin.

This struct allows the user to implement a pin containing information about the pin's port and number.

The documentation for this struct was generated from the following file:

- **Pin.h**

WheelSpeed Class Reference

Header for a class that implements a hall effect sensor to measure wheel speed. More...

`#include <WheelSpeed.h>`

Public Member Functions

	WheelSpeed (uint8_t triggers) Constructor which creates and initializes a wheel speed object. More...
void	**calc** () Update the current time and previous time to calculate the wheel speed. More...
float	**get** () Return the wheel speed in rotations per second. More...

Detailed Description

Header for a class that implements a hall effect sensor to measure wheel speed.

This class allows the user to implement a hall effect sensor to meausre wheel speed. It relies on the user to set up an interrupt service routine to call the calculate function on the rising or falling edge of the signal from a hall effect sensor measuring wheel speed.

Constructor & Destructor Documentation

◆ WheelSpeed()

WheelSpeed::WheelSpeed (uint8_t triggers)

Constructor which creates and initializes a wheel speed object.

This constructor creates an wheel speed object with the given number of triggers. It saves the number of triggers and initializes the "previous" and "current" times.

Parameters

triggers The number of triggers per wheel revolution.

Member Function Documentation

◆ calc()

void WheelSpeed::calc ( )

Update the current time and previous time to calculate the wheel speed.

This function stores the "current" time as the previous time and assign the current time to current time. These values can be used to calculate the wheel speed, but this calculation is left until the **get()** function is called to minimize the time spent in the interrupt service routing.

◆ get()

float WheelSpeed::get ( )

Return the wheel speed in rotations per second.

This function calculates the wheel speed using the stored previous time and stored current time. The time between triggers is multipled by the number of triggers to calculate the time required for one revolution in microseconds. The constant 1E6 is divided by this time to get rotations per second.

Returns

The wheel speed in rotations per second as a float.

The documentation for this class was generated from the following files:

- **WheelSpeed.h**
- **WheelSpeed.cpp**

File List

Here is a list of all documented files with brief descriptions:

Encoder.cpp	Source code for a class that implements an encoder
Encoder.h	Header for a class that implements an encoder
EngineSpeed.cpp	Source code for a class that implements a hall effect sensor to measure engine speed
EngineSpeed.h	Header for a class that implements a hall effect sensor to measure engine speed
IO_Config.cpp	Source code for a library that implements the input/output configuration
IO_Config.h	Header for a library that implements the input/output configuration
Motor.cpp	Source code for a class that implements a motor
Motor.h	Header for a class that implements a motor
PIDController.cpp	Source code for a class that implements a PID Controller
PIDController.h	Header for a class that implements a PID Controller
Pin.h	Header for a struct that implements a pin
TC_Config.cpp	Source code for a library that implements the timer/counter configuration
TC_Config.h	Source code for a header that implements the timer/counter configuration
WheelSpeed.cpp	Source code for a class that implements a hall effect sensor to measure wheel speed
WheelSpeed.h	Header for a class that implements a hall effect sensor to measure wheel speed

Encoder.cpp File Reference

Source code for a class that implements an encoder. More...

```
#include "Encoder.h"
#include <stdint.h>
#include "IO_Config.h"
#include "TC_Config.h"
#include "Pin.h"
```

Detailed Description

Source code for a class that implements an encoder.

This class allows the user to implement an encoder on either Timer D0, Event Channel 2 or Timer D1, Event Channel 3. Future updates will allow for the timer and event channel to be chosen independently by the user via parameters in the constructor. Each encoder object of this class can read from and written to independently.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

Encoder.h File Reference

Header for a class that implements an encoder. More...

```
#include <stdint.h>
#include "Pin.h"
```

Go to the source code of this file.

Classes

class **Encoder**
Class that implements an encoder. More...

Detailed Description

Header for a class that implements an encoder.

This class allows the user to implement an encoder on either Timer D0, Event Channel 2 or Timer D1, Event Channel 3. Future updates will allow for the timer and event channel to be chosen independently by the user via parameters in the constructor. Each encoder object of this class can read from and written to independently.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

EngineSpeed.cpp File Reference

Source code for a class that implements a hall effect sensor to measure engine speed. More...

```
#include "EngineSpeed.h"
#include <stdint.h>
#include "TC_Config.h"
```

Variables

const uint32_t **TIMEOUT** = 1000000

Detailed Description

Source code for a class that implements a hall effect sensor to measure engine speed.

This class allows the user to implement a hall effect sensor to meausre engine speed. It relies on the user to set up an interrupt service routine to call the calculate function on the rising or falling edge of the signal from a hall effect sensor measuring engine speed. The engine speed is similar to the wheel speed library but also averages the thermodynamic cycle of a four-stroke engine (that is, it averages every two revolutions) to reduce the noise of the output signal.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

EngineSpeed.h File Reference

Header for a class that implements a hall effect sensor to measure engine speed. More...

```
#include <stdint.h>
#include "TC_Config.h"
```

Go to the source code of this file.

Classes

class **EngineSpeed**
Header for a class that implements a hall effect sensor to measure engine speed. More...

Detailed Description

Header for a class that implements a hall effect sensor to measure engine speed.

This class allows the user to implement a hall effect sensor to meausre engine speed. It relies on the user to set up an interrupt service routine to call the calculate function on the rising or falling edge of the signal from a hall effect sensor measuring engine speed. The engine speed is similar to the wheel speed library but also averages the thermodynamic cycle of a four-stroke engine (that is, it averages every two revolutions) to reduce the noise of the output signal.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

IO_Config.cpp File Reference

Source code for a library that implements the input/output configuration. More...

```
#include "IO_Config.h"
#include <avr/io.h>
#include "Pin.h"
```

Functions

void **IO_Init** ()
Sets up the engine speed and wheel speed interrupt pins. More...

Detailed Description

Source code for a library that implements the input/output configuration.

This library allows the user to configure the inputs/outputs in one place. The library assigns ports and pin numbers via **Pin** objects to each input/output. Additionally, the library sets up interrupts for the engine speed and rear wheel speed hall effect sensors.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

Function Documentation

◆ IO_Init()

void IO_Init ( )

Sets up the engine speed and wheel speed interrupt pins.

This function sets up the engine speed and wheel speed for interrupt functionality.

IO_Config.h File Reference

Header for a library that implements the input/output configuration. More...

```
#include <avr/io.h>
#include "Pin.h"
```

Go to the source code of this file.

Functions

void **IO_Init** ()
Sets up the engine speed and wheel speed interrupt pins. More...

Variables

const **Pin** **P_MOT_INA** = {&PORTA, PIN0_bm}
const **Pin** **P_MOT_INB** = {&PORTA, PIN1_bm}
const **Pin** **P_MOT_PWM** = {&PORTE, PIN2_bm}
const **Pin** **P_ENC_A** = {&PORTA, PIN4_bm}
const **Pin** **P_ENC_B** = {&PORTA, PIN5_bm}
const **Pin** **S_MOT_INA** = {&PORTA, PIN2_bm}
const **Pin** **S_MOT_INB** = {&PORTA, PIN3_bm}
const **Pin** **S_MOT_PWM** = {&PORTE, PIN3_bm}
const **Pin** **S_ENC_A** = {&PORTA, PIN6_bm}
const **Pin** **S_ENC_B** = {&PORTA, PIN7_bm}

Detailed Description

Header for a library that implements the input/output configuration.

This library allows the user to configure the inputs/outputs in one place. The library assigns ports and pin numbers via **Pin** objects to each input/output. Additionally, the library sets up interrupts for the engine speed and rear wheel speed hall effect sensors.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

Function Documentation

IO_Init()

void IO_Init ( )

Sets up the engine speed and wheel speed interrupt pins.

This function sets up the engine speed and wheel speed for interrupt functionality.

Motor.cpp File Reference

Source code for a class that implements a motor. More...

```
#include "Motor.h"
#include "avr/io.h"
#include <stdint.h>
#include "TC_Config.h"
#include "Pin.h"
```

Detailed Description

Source code for a class that implements a motor.

This class allows the user to implement a motor. The motor will respond to a duty cycle between -100 and 100, with a duty cycle of 0 causing the motor to coast regardless of direction. Each motor object of this class can controlled independently.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

Motor.h File Reference

Header for a class that implements a motor. More...

```
#include <stdint.h>
#include "avr/io.h"
#include "Pin.h"
```

Go to the source code of this file.

Classes

class **Motor**
Class that implements a motor. More...

Detailed Description

Header for a class that implements a motor.

This class allows the user to implement a motor. The motor will respond to a duty cycle between -100 and 100, with a duty cycle of 0 causing the motor to coast regardless of direction. Each motor object of this class can controlled independently.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

PIDController.cpp File Reference

Source code for a class that implements a PID Controller. More...

```
#include "PIDController.h"
#include <stdint.h>
```

Detailed Description

Source code for a class that implements a PID Controller.

This class allows the user to implement a PID Controller. It is given a setpoint, low saturation, high saturation, and Kp, Ki, and Kd gain values. In addition to providing PID output, the class accounts for integral windup by disabling the integral term if the output is determined as saturated. The class can also serve as a P-Only, PI, or PD controller by setting the appropriate gains to 0, although this does result in some unnecessary calculations.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

PIDController.h File Reference

Header for a class that implements a PID Controller. More...

```
#include <stdint.h>
```

Go to the source code of this file.

Classes

class **PIDController**
Class that implements a PID Controller. More...

Detailed Description

Header for a class that implements a PID Controller.

This class allows the user to implement a PID Controller. It is given a setpoint, low saturation, high saturation, and Kp, Ki, and Kd gain values. In addition to providing PID output, the class accounts for integral windup by disabling the integral term if the output is determined as saturated. The class can also serve as a P-Only, PI, or PD controller by setting the appropriate gains to 0, although this does result in some unnecessary calculations.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

Pin.h File Reference

Header for a struct that implements a pin. More...

```
#include <avr/io.h>
```

Go to the source code of this file.

Classes

struct **Pin**
Header for a struct that implements a pin. More...

Detailed Description

Header for a struct that implements a pin.

This struct allows the user to implement a pin containing information about the pin's port and number.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

TC_Config.cpp File Reference

Source code for a library that implements the timer/counter configuration. More...

```
#include "TC_Config.h"
#include <avr/io.h>
#include <stdint.h>
#include "IO_Config.h"
#include "Pin.h"
```

Functions

void	**TC_Init** () Sets up the timer configuration. More...
uint32_t	**micros** () Returns the system time. More...
void	**analogWrite** (**Pin** pin, uint8_t dutyCycle) Returns the system time. More...

Detailed Description

Source code for a library that implements the timer/counter configuration.

This library allows the user to configure the timers/counters in one place. System time is retrieved using the **micros()** function, meant to simulate the function on the Arduino but with 1us precision. PWM duty cycle is written on Timer E0 using the **analogWrite()** function, meant to simulate the function on the Arduino. Future updates will allow for the timer and event channel to be chosen independently by the user via parameters in the constructor.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

Function Documentation

◆ analogWrite()

void analogWrite (**Pin** pin,
uint8_t dutyCycle
)

Returns the system time.

This function concatenates the values of two 16-bit timers to return the system time as a 32-bit timer.

Parameters

pin The pin to write the PWM duty cycle to.

dutyCycle The duty cycle to write, between -100 and 100.

◆ micros()

uint32_t micros ( )

Returns the system time.

This function concatenates the values of two 16-bit timers to return the system time as a 32-bit timer.

Returns

The system time in microseconds as an unsigned 32-bit number.

◆ TC_Init()

void TC_Init ( )

Sets up the timer configuration.

This function sets up the system clock timer, timer interrupt, encoder counters, and PWM output.

TC_Config.h File Reference

Source code for a header that implements the timer/counter configuration. More...

```
#include <avr/io.h>
#include <stdint.h>
#include "IO_Config.h"
#include "Pin.h"
```

Go to the source code of this file.

Functions

void	**TC_Init** () Sets up the timer configuration. More...
uint32_t	**micros** () Returns the system time. More...
void	**analogWrite** (**Pin** pin, uint8_t value) Returns the system time. More...

Detailed Description

Source code for a header that implements the timer/counter configuration.

This library allows the user to configure the timers/counters in one place. System time is retrieved using the **micros()** function, meant to simulate the function on the Arduino but with 1us precision. PWM duty cycle is written on Timer E0 using the **analogWrite()** function, meant to simulate the function on the Arduino. Future updates will allow for the timer and event channel to be chosen independently by the user via parameters in the constructor.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

Function Documentation

◆ analogWrite()

void analogWrite (**Pin** pin,
uint8_t dutyCycle
)

Returns the system time.

This function concatenates the values of two 16-bit timers to return the system time as a 32-bit timer.

Parameters

pin The pin to write the PWM duty cycle to.

dutyCycle The duty cycle to write, between -100 and 100.

◆ micros()

uint32_t micros ( )

Returns the system time.

This function concatenates the values of two 16-bit timers to return the system time as a 32-bit timer.

Returns

The system time in microseconds as an unsigned 32-bit number.

◆ TC_Init()

void TC_Init ( )

Sets up the timer configuration.

This function sets up the system clock timer, timer interrupt, encoder counters, and PWM output.

WheelSpeed.cpp File Reference

Source code for a class that implements a hall effect sensor to measure wheel speed. More...

```
#include "WheelSpeed.h"
#include <stdint.h>
#include "TC_Config.h"
```

Variables

const uint32_t **TIMEOUT** = 1000000

Detailed Description

Source code for a class that implements a hall effect sensor to measure wheel speed.

This class allows the user to implement a hall effect sensor to meausre wheel speed. It relies on the user to set up an interrupt service routine to call the calculate function on the rising or falling edge of the signal from a hall effect sensor measuring wheel speed.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09

WheelSpeed.h File Reference

Header for a class that implements a hall effect sensor to measure wheel speed. More...

```
#include <stdint.h>
#include "TC_Config.h"
```

Go to the source code of this file.

Classes

class **WheelSpeed**
Header for a class that implements a hall effect sensor to measure wheel speed. More...

Detailed Description

Header for a class that implements a hall effect sensor to measure wheel speed.

This class allows the user to implement a hall effect sensor to meausre wheel speed. It relies on the user to set up an interrupt service routine to call the calculate function on the rising or falling edge of the signal from a hall effect sensor measuring wheel speed.

Author

KC Egger, Rahul Goyal, Alexandros Petrakis

Date

2019-12-09