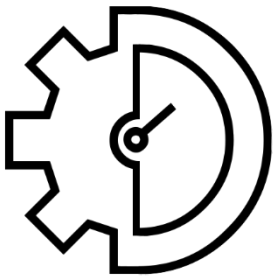


MOTODRIVER

UNIVERSAL ARDUINO LIBRARY

PRODUCT DOCUMENTATION

CONTENTS

PRODUCT REQUIREMENT DOCUMENT

Description

Motodriver is a universal Arduino library that makes simpler DC engine control via popular motor driver shields. Motodriver library includes a number of simple and easy-to-use methods to control your device's engines both separately and as part of a moving group.

If you are brand new to microcontroller programming, or programming in general, this library may help you with your first experience. Examples section will help you get started with this.

Last stable version of library supports the following motor drivers:

- L298N 2X Motor Shield
- L293D Motor Shield

Latest unstable versions can be found in our GitHub repository and can only be installed directly.

Tasks

- Development of a scalable software "skeleton" for easy-to-dev integration of new supported motor drivers and new control methods.
- Development of a several universal control methods suitable for all supported motor drivers.
- Writing complete project documentation for an easy introduction of new developers.

Feedback

You can submit bug report, feature requests or any other feedback about Motodriver library in our feedback section in GitHub repository:

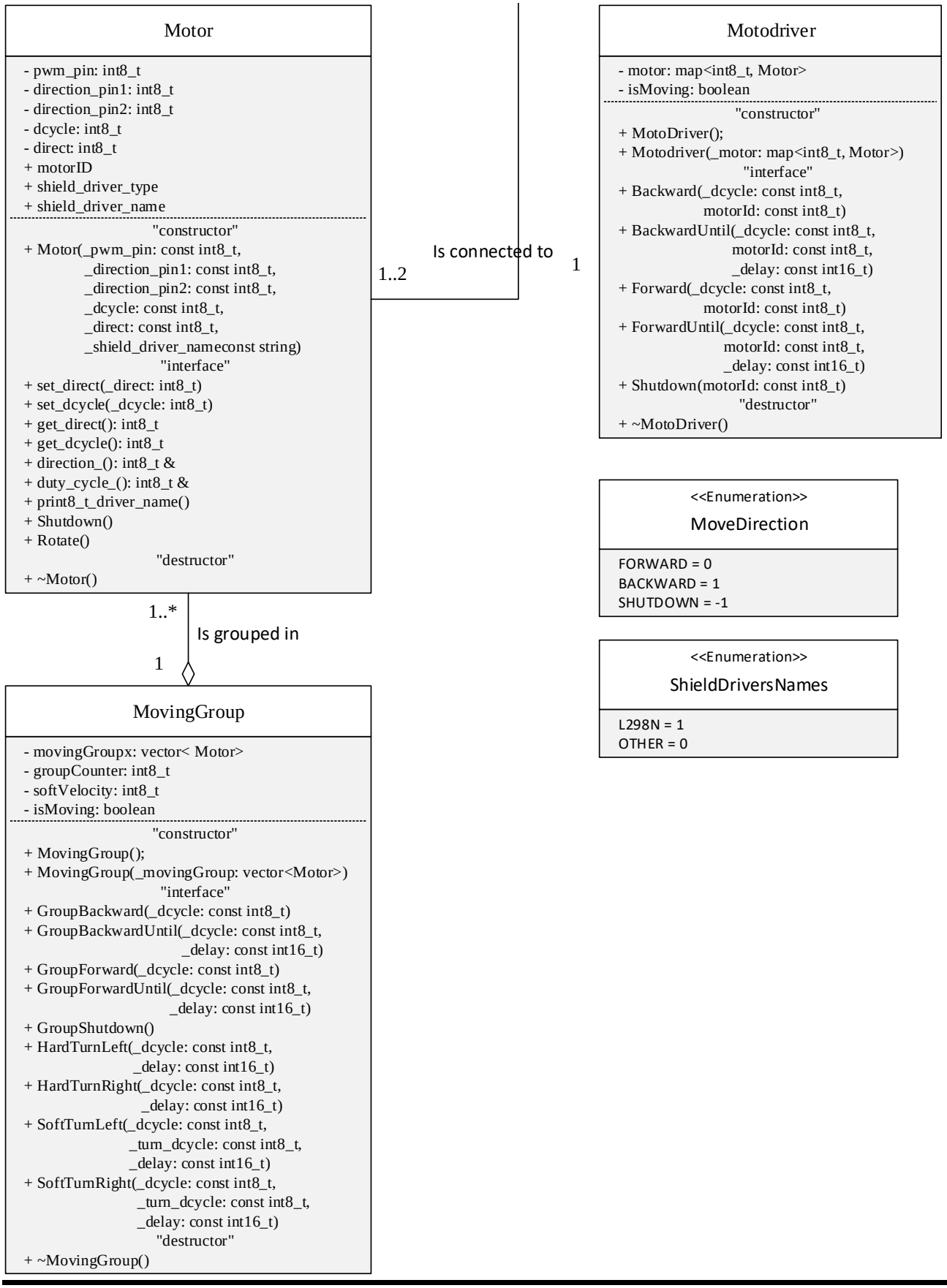
- Bug Report
- Feature Request

DESIGN AND ARCHITECTURE

External libraries

Motodriver uses an external Arduino STL version 1.3.3 library by Mike Matera that contains an implementation of a C++ standard library packaged as an Arduino library.

Architecture



CLASS MOTOR

Syntax

```
class Motor
```

- `Motor()`

  Motor object constructor by default.

- `Motor(const int8_t &_pwm_pin, const int8_t &_direction_pin1,`
  `const int8_t &_direction_pin2,`
  `const ShieldDriversNames &_shield_driver_type)`

  Motor object constructor where:

  pwm_pin - PWM (Pulse-width modulation) controller analog pin number.

  direction_pin1 – First rotating direction digital pin number.

  direction_pin2 – Second rotating direction digital pin number (if exists).

  shield_driver_name – Type of motor driver shield.

- `~Motor()`

  Motor object destructor by default.

Properties

- `private int8_t pwm_pin`

  PWM controller analog pin number.

- `private int8_t direction_pin1`

  First rotating direction digital pin number.

- `private int8_t direction_pin2`

  Second rotating direction digital pin number (if exists).

- `private boolean direction_pin_flag`

  Second direction pin value existence flag.

- `private int8_t dcycle`

  Motor instance duty cycle value.

- `private int8_t direct`

  Motor instance direction value.
- `public int8_t motorID`

  Motor instance ID number.
- `public int8_t shield_driver_type`

  Type of motor driver motor instance connected with.
- `public string shield_driver_name`

  Name of motor driver motor instance connected with.

Accessors

- `public void set_direct(int8_t _direct)`

  Sets new direction value into direct property.
- `public int8_t get_direct()`

  Returns current direct property value.
- `public void set_dcycle(int8_t _dcycle)`

  Sets new direction value into dcycle property.
- `public int8_t get_dcycle()`

  Returns current dcycle property value.
- `public int8_t &direction_()`

  Returns link of direct property value.
- `public int8_t &duty_cycle_()`

  Returns link of dcycle property value.

Methods

- `public void print8_t_driver_name()`

  Prints in console motor driver type motor instance connected with.
- `public void Rotate()`

  Sends signals to digital and analog pins of motor driver to move chosen motor instance with set direction and duty cycle.

- `public void Shutdown()`

  Sends signals to digital and analog pins of motor driver to stop chosen motor instance.

CLASS MOTODRIVER

Syntax

```
class MotoDriver
```

- `MotoDriver()`

  MotoDriver object constructor by default.

- `MotoDriver(std::map<int8_t, Motor> &_motors)`

  MotoDriver object constructor where:

  motors – map container with all connected Motor instances and their IDs.

- `~MotoDriver()`

  MotoDriver object destructor by default.

Properties

- `private std::map<int8_t, Motor> motors`

  Map container with all connected Motor instances and their IDs.

- `private boolean isMoving`

  Flag that takes true value if current motor is moving and false if not.

Methods

- ```
  public void Backward(const int8_t &_dcycle,
                       const int8_t &motorId)
  ```

  Move chosen engine backward with set duty cycle (0-255).

- ```
  public void Forward(const int8_t &_dcycle,
                      const int8_t &motorId)
  ```

  Move chosen engine forward with set duty cycle (0-255).

- ```
  public void BackwardUntil(const int8_t &_dcycle,
                            const int8_t &motorId,
                            const int16_t &_delay)
  ```
  ```

Move chosen engine backward with set duty cycle (0-255) and set delay.

When delay ends, shutdown chosen engine.

- ```
  public void ForwardUntil(const int8_t &_dcycle,
                           const int8_t &motorId,
                           const int16_t &_delay)
  ```

  Move chosen engine forward with set duty cycle (0-255) and set delay.

  When delay ends, shutdown chosen engine.

- ```
  public void Shutdown(const int8_t &motorId)
  ```

  Stops moving chosen engine.

CLASS MOVINGGROUP

Syntax

```
class MovingGroup
```

- `MovingGroup()`

  MovingGroup object constructor by default.

- `MovingGroup(std::vector<Motor> &_movingGroup)`

  MovingGroup object constructor where:

  movingGroup – vector with all connected Motor instances.

- `~MovingGroup()`

  MovingGroup object destructor by default.

Properties

- `private std::vector<Motor> movingGroup`

  Vector with all connected Motor instances.

- `private boolean isMoving`

  Flag that takes true value if joint motor group is moving and false if not.

- `private int8_t groupCounter`

  Counter of the number of motors in joint group .

- `private int8_t softVelocity`

  Duty cycle value for soft turn methods.

Methods

- `public void GroupBackward(const int8_t &_dcycle)`

  Move joint motors group backward with set duty cycle (0-255).

- `public void GroupForward(const int8_t &_dcycle)`

  Move joint motors group forward with set duty cycle (0-255).

- `public void GroupBackwardUntil(const int8_t &_dcycle, const int16_t &_delay)`

  Move joint motors group backward with set duty cycle (0-255) and set delay. When delay ends, shutdown all motors in group.

- `public void GroupForwardUntil(const int8_t &_dcycle, const int16_t &_delay)`

  Move joint motors group forward with set duty cycle (0-255) and set delay. When delay ends, shutdown all motors in group.

- `public void GroupShutdown()`

  Stops moving all motors in group.

- `public void HardTurnLeft(const int8_t &_dcycle, const int16_t &_delay)`

  Hard turn to left with maximum possible angular velocity. Only available for groups from 2-4 joint motors.

- `public void HardTurnRight(const int8_t &_dcycle, const int16_t &_delay)`

  Hard turn to right with maximum possible angular velocity. Only available for groups from 2-4 joint motors.

- `public void SoftTurnLeft(const int8_t &_dcycle, const int8_t &_turn_dcycle, const int16_t &_delay)`

  Soft turn to left with set increase to angular velocity. Only available for groups from 2-4 joint motors.

- `public void SoftTurnRight(const int8_t &_dcycle, const int8_t &_turn_dcycle, const int16_t &_delay)`

  Soft turn to right with set increase to angular velocity. Only available for groups from 2-4 joint motors.

TYPE DECLARATION

ShieldDriverNames

Custom enum type for all supported motor drivers.

- L298N = 1
- L293D = 2
- Other / Undefined = 0

MoveDirection

Custom enum type for all supported directions.

- FORWARD = 0
- BACKWARD = 1
- SHUTDOWN = -1

EXAMPLES

Description

Here you can find several examples of using Motodriver library. They are basic programs and wiring schemas which are meant to introduce you to various aspects of the library. You also can find them in Examples section of our GitHub repository.

These examples were made using:

- Arduino UNO microcontroller board
- Arduino IDE 1.8.19 version
- Motodriver library 2.16 nightly version

Example 1

Introduces basic sketch skeleton for controlling a pair of DC engines via L298N 2X motor driver shield separately.

Source code

```
#include <Arduino.h>
#include "motodriver.h"

void setup() {
     Serial.begin(9600);
}

// After setup function create Motor class instances
// for your DC motors with chosen Arduino pin numbers.

Motor motor1 = Motor(10,9,8,ShieldDriversNames::L298N);
Motor motor2 = Motor(3,5,4,ShieldDriversNames::L298N);

// Create map container with Motor instances and their IDs
// and create MotoDriver class instance as below.

std::map<int8_t, Motor> motorList = { {1, motor1}, {2, motor2} };
MotoDriver motodriver = MotoDriver(motorList);
```

```
void loop() {

   // Move forward first motor with maximum duty cycle during second.

   motodriver.ForwardUntil(255, 1, 1000);

   // Move forward second motor with maximum duty cycle during
   // second.

   motodriver.ForwardUntil(255, 2, 1000);

   // Move backward first motor with low duty cycle during half
   // second.

   motodriver.BackwardUntil(50, 1, 500);
}
```

Example 2

Introduces sketch for controlling a pair DC engines as joint moving group via L298N 2X motor driver shields.

Source code

```
#include <Arduino.h>
#include "motodriver.h"

void setup() {
     Serial.begin(9600);
}

// After setup function create Motor class instances
// for your DC motors with chosen Arduino pin numbers.

Motor motor1 = Motor(10,9,8,ShieldDriversNames::L298N);
Motor motor2 = Motor(3,5,4,ShieldDriversNames::L298N);

// Create list of Motor instances
// and create MovingGroup class instance as below.

std::vector<Motor> motorList = {motor1, motor2};
MovingGroup movingGroup = MovingGroup(motorList);

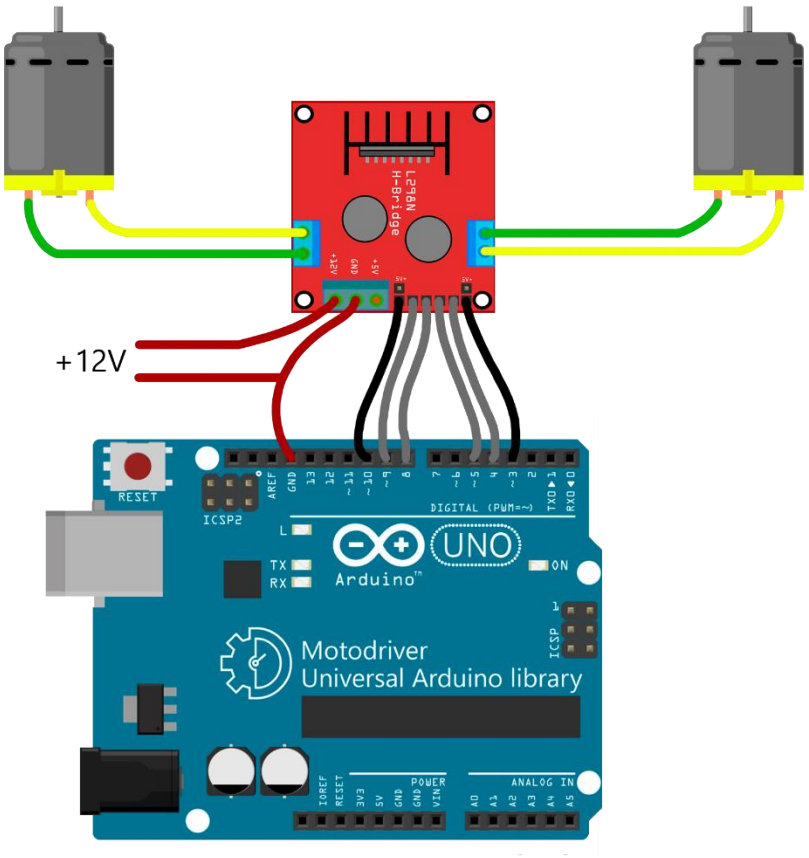
void loop() {

   // Move forward with maximum duty cycle during second.
```

```
    movingGroup.GroupForwardUntil(255, 1000);
    // Move backward with low duty cycle during half a second.
    movingGroup.GroupBackwardUntil(50, 500);
}
```

Example 1 and 2 wiring schema

There each of motor uses two digital pins to control its rotating direction and one analog PWM pin to control its duty cycle.



ENA motor driver pin connected to ~10 pin.

IN1 motor driver pin connected to ~9 pin.

IN2 motor driver pin connected to 8 pin.

IN3 motor driver pin connected to ~5 pin.

IN4 motor driver pin connected to 4 pin.

ENB motor driver pin connected to ~3 pin.

GND motor driver pin connected to GND pin on Arduino and power source.

+12V motor driver pin connected to power source directly.

Example 3

Introduces sketch demonstrates complex movement with turns and U-turns.

Source code

```
#include <Arduino.h>
#include "motodriver.h"

void setup() {
     Serial.begin(9600);
}

// After setup function create Motor class instances
// for your DC motors with chosen Arduino pin numbers.

// Top left
Motor motor1 = Motor(6,5,ShieldDriversNames::L298N);
// Top right
Motor motor2 = Motor(3,4,ShieldDriversNames::L298N);
// Down left
Motor motor3 = Motor(11,10,ShieldDriversNames::L298N);
// Down right
Motor motor4 = Motor(9,8,ShieldDriversNames::L298N);

// Create list of Motor instances
// and create MovingGroup class instance as below.

std::vector<Motor> motorList = {motor1, motor2, motor3, motor4};

MovingGroup movingGroup = MovingGroup(motorList);

void loop() {

   // Move forward with maximum duty cycle during second.

   movingGroup.GroupForwardUntil(255, 1000);

   // Slow down before turn.

   movingGroup.GroupForwardUntil(100, 300);

   // Turn left with maximum possible angular velocity via HardTurn
  // function.

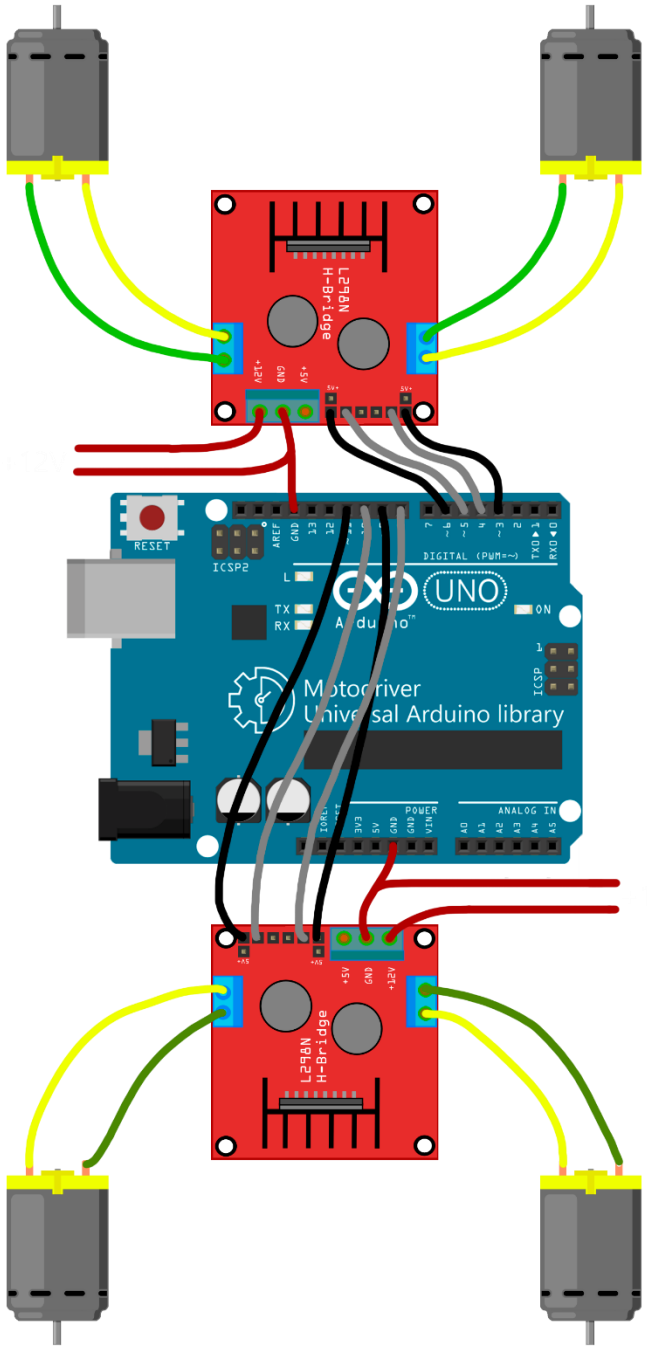
   movingGroup.HardTurnLeft(100, 500);

   // Turn right with slower angular velocity but longer turn time
  // via SoftTurn function.
```

```
    movingGroup.SoftTurnRight(100, 50, 1000);
}
```

Example 3 wiring schema

There each of motor uses only one digital pin to control its rotating direction and one analog PWM pin to control its duty cycle.



First L298N motor shield:

ENA motor driver pin connected to ~6 pin.

IN1 motor driver pin connected to ~5 pin.

IN2 motor driver pin is empty.

IN3 motor driver pin is empty.

IN4 motor driver pin connected to 4 pin.

ENB motor driver pin connected to ~3 pin.

Second L298N motor shield:

ENA motor driver pin connected to ~11 pin.

IN1 motor driver pin connected to ~10 pin.

IN2 motor driver pin is empty.

IN3 motor driver pin is empty.

IN4 motor driver pin connected to 8 pin.

ENB motor driver pin connected to ~9 pin.

Each GND motor driver pin connected to GND pin on Arduino and power source.

Each +12V motor driver pin connected to power source directly.