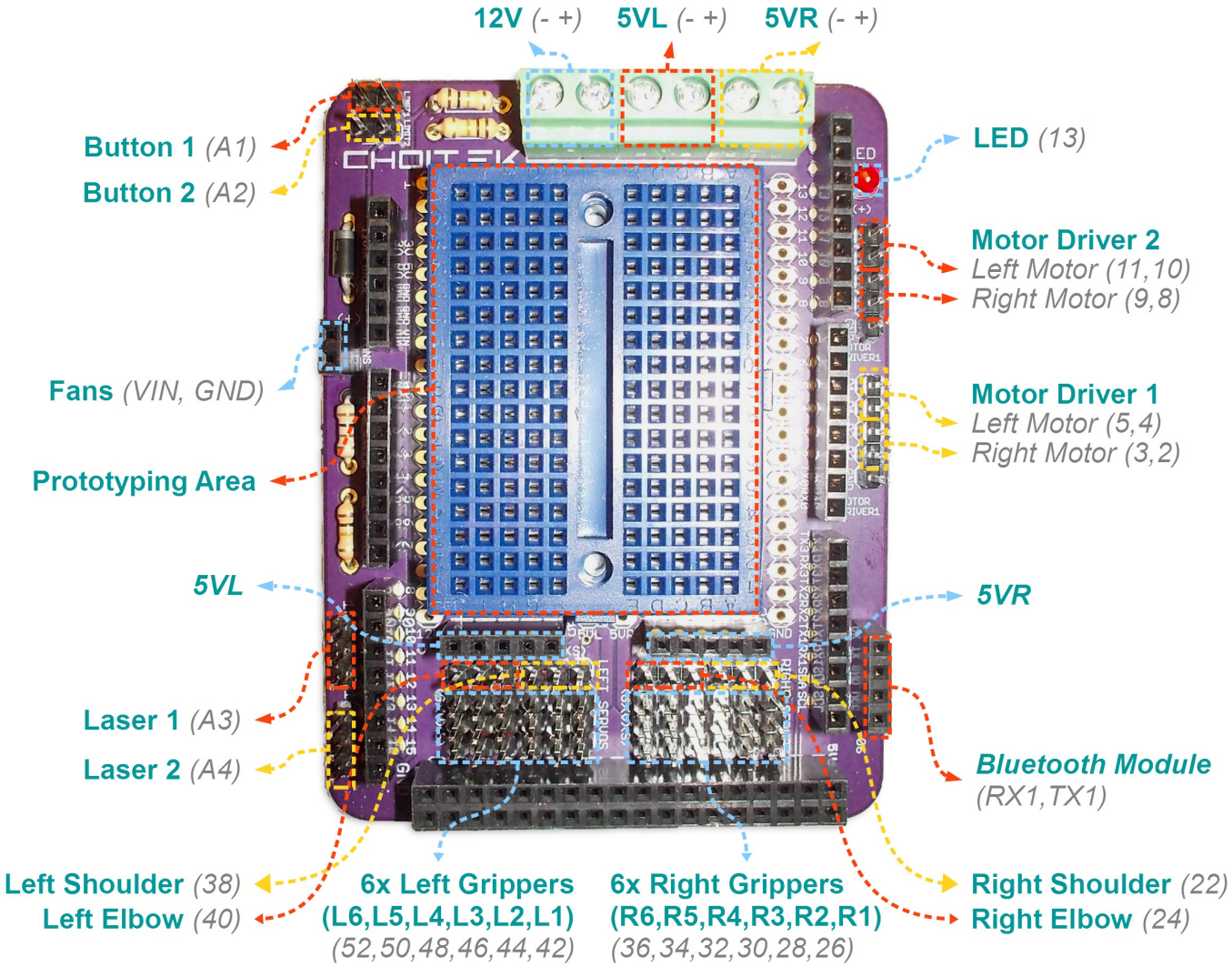
Megamark Arduino Library Documentation

The Choitek Megamark is an advanced full-size multipurpose mobile manipulator robotics platform for students, artists, educators and researchers alike. In our mission to make the platform as open, friendly and compatible as possible, we've made the platform compatible with a variety of different languages and frameworks. This document covers all the commands of the Megamark Arduino Library.

Choitek Robot Shield

Using the Arduino Library is directly tied to the various pin configurations on the Arduino Mega 2560. We've created a robot controller shield specifically for electronics prototyping that can directly connect to various motors, sensors, actuators, and other electronics on the Choitek Megamark Robot. Note the specific pin mappings as labeled below on the robot shield:



Word of Note

The Megamark Arduino Library is lower level than the Python 2.7, Processing 3 and Unity libraries for the Choitek Megamark Robot, and as such is structured differently. Unlike the other software frameworks, creating code using the Arduino IDE will be directly loaded onto the Megamark robot's internal Arduino Mega 2560, where the loaded code can run automatically without the need for an additional laptop to run the Arduino code. However, you will still need a Windows/Mac/Linux machine to write and load the code onto the Arduino Mega 2560 using the Arduino IDE.

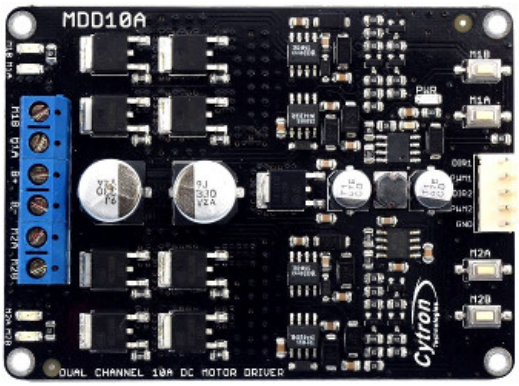
List of Classes

The Megamark Library comes with a variety of plug and play classes that can be used in your Arduino project. Each class comes with its own unique functions and variables.

- `MotorController`
- `ServoMotor`
- `Laser`
- `Voltmeter`
- `SerialHandler`

`MotorController`

The `MotorController` class is used to connect the Arduino Mega with a dual DC motor driver and control each independently with various speed and acceleration. The Choitek Robot Shield has two ports for up to two dual DC motor drivers.



`MotorController(int dir1, int pwm1, int dir2, int pwm2, bool invert1, bool invert2)`

This creates a new instance of a dual DC Motor driver in Arduino with the following parameters:

- `dir1` sets the Arduino pin controlling the first motor's direction. (Default 5)
- `pwm1` sets the Arduino pin controlling the first motor's speed. (Default 4)
- `dir2` sets the Arduino pin controlling the second motor's direction. (Default 3)
- `pwm2` sets the Arduino pin controlling the second motor's speed. (Default 2)
- `invert1` is an optional value that sets whether the first motor is inverted. (Default true)
- `invert2` is an optional value that sets whether the second motor is inverted. (Default false)

`initialize()`

This command will initialize the motor controller and configure the direction and speed pins on the Arduino. Note that this command must be called in Arduino's `setup()` function before using any of the other commands!

`setVelocities(int velocity1, int velocity2)`

This makes a call to set the angular velocities of the motors on the dual DC motor driver. The first argument `velocity1` sets the first motor's velocity, and the second argument `velocity2` sets the second motor's velocity. Both velocities are actually PWM signals and are not tied to any particular unit, and range from `-255` to `255`. A positive value rotates the motor clockwise, whereas a negative value rotates the motor counterclockwise. A value of zero will stop the motor.

`update(float delay)`

This updates the velocities of the dual DC motor driver. Note that the velocities will not change until update is called! We recommend calling `update()` on the motor controller once every loop. The argument `delay` is an optional value that sets the rate at which the velocity changes to the target velocity. (Default 0.5)

`ServoMotor`

The `ServoMotor` class is used to control a single RC hobby servo using a PWM signal. The Choitek Robot Shield has ports for up to 16 servos, where 2 are used for the robot's shoulders, 2 are used for the robot's elbows, 6 are used for the robot's left grippers, and 6 are used for the robot's right grippers.

`ServoMotor(int pin, bool invertDir)`

This creates a new instance of a single RC hobby servo motor connected to `pin` on the Arduino Mega 2560. If `invertDir` is set to `true`, this will invert the direction of the servo's angle.

`on()`

This command will turn the servo on and virtually attach it to the Arduino. Note that this command must be called in Arduino's `setup()` function before using any of the other commands!

`off()`

This command will turn the servo off and virtually detach it from the Arduino. Note that none of the other commands will work unless the servo is turned on again with `on()`.

`rotate(int angle)`

This makes a call to rotate the servo by `angle` degrees, incrementally. A positive value will increase the servo's angle position, whereas a negative value will decrease the servo's angle position. Although `angle` can be any integer value, the servo will clamp its position to a minimum of 0 degrees and a maximum of 180 degrees. Note that the angle representation may not be accurate if a special type of RC servo is used, such as continuous rotation servos.

`rotateTo(int angle, bool onOff)`

This makes a call to rotate the servo to `angle` in degrees, ranging from 0 to 180 degrees. If `onOff` is set to `true`, the servo will toggle its on/off state when `angle` is set to `-1`. (`onOff` is set to `false` by default). Note that the angle representation may not be accurate if a special type of RC servo is used, such as continuous rotation servos.

`update(float delay)`

This updates the angle of the RC servo. Note that the servo's angle will not change until update is called! We recommend calling `update()` on the RC servo once every loop. The argument `delay` is an optional value that sets the rate at which the angle changes to the target angle. (Default 5.0)

`isAttached()`

This checks whether the servo is currently virtually attached to the Arduino Mega. If it is attached, `true` is returned, and if not, `false` is returned. Calling the functions `on()` and `off()` will change the state of attachment.

`getPos()`

This returns the current angle of the RC servo in degrees as an integer.

`Laser`

The `Laser` class is used to connect to and receive incoming range data from infrared laser distance sensors. The Choitek Robot Shield has two ports for up to two infrared laser distance sensors.

`Laser(int pin)`

This creates a new instance of a single infrared laser distance sensor connected to `pin` on the Arduino Mega 2560. Note that `pin` must be an analog input pin on the Arduino Mega!

`initialize()`

This command will initialize the infrared laser sensor and configure the sensor input pin on the Arduino. Note that this command must be called in Arduino's `setup()` function before using any of the other commands!

`range(int mode)`

This gets a current reading of the attached infrared laser sensor in millimeters. Note that this command will return different ranges of values for different types of infrared laser sensors. `Mode` is an optional parameter that sets the units a certain infrared sensor range:

- Set `mode = 0` for raw analog input from the laser sensor (0-1023)
- Set `mode = 1` for 40-300mm laser sensor
- Set `mode = 2` for 100-800mm laser sensor
- Set `mode = 3` for 200-1500mm laser sensor

Voltmeter

The `Voltmeter` class is used to measure battery levels from a simple 12V to 5V voltage divider circuit. This is used to measure the shared voltage of the Megamark robot's two 12V sealed lead acid batteries.

Voltmeter(int pin, int led)

This creates a new instance of a single voltage meter connected to `pin` on the Arduino Mega 2560, used to measure battery level on the Megamark robot. Note that `pin` must be an analog input pin on the Arduino Mega! There is also an optional `led` parameter that sets the pin connected to an LED that lights up when the batteries need to be recharged.

initialize()

This command will initialize the voltage meter and configure the sensor input pin on the Arduino. Note that this command must be called in Arduino's `setup()` function before using any of the other commands!

voltage(int mode)

This gets a current reading of the attached voltage meter in centivolts. Note that this command will return different ranges of values for different combinations of voltage dividers. `Mode` is an optional parameter that sets the units a certain infrared sensor range:

- Set `mode = 0` for raw analog input from the voltmeter (0-1023)
- Set `mode = 1` for 0-15 volts (0-1500 centivolts)

update()

This updates the LED connected to the voltage meter. Note that the LED will not light up when the batteries need to be recharged until `update()` is called! We recommend calling `update()` on the voltage meter once every loop.

SerialHandler

A standard USB A-to-B cable can be plugged into the robot's Arduino Mega 2560 to any USB-compatible computer, such as a Mac, Windows, or Linux laptop (even a Raspberry Pi!). A program running on the main computer can messages back and forth to the Arduino via standard USB communication protocols, as long the Arduino has code ready to receive and send such messages. This is what the Serial Handler (an abstraction of Arduino's lower level `Serial` class) is for.

SerialHandler(int inCnt, int outCnt)

The `SerialHandler` class is formatted to send a list of integers back and forth from the main computer. The `inCnt` parameter sets how many integers should be read from the main computer to the Arduino, and the `outCnt` parameter sets how many integers should be written from the Arduino to the main computer. Note that the hard-coded maximum values of `inCnt` and `outCnt` are both 32.

initialize(int mode)

This command will initialize the `SerialHandler` at a communication speed of 57600 baud. Note that this command must be called in Arduino's `setup()` function before using any of the other commands! An optional `mode` parameter can be set to 0 to initialize the `SerialHandler` according to the Megamark robot's sensors and actuators.

read()

This command will read `inCnt` integers from the incoming serial buffer sent from the main computer to the Arduino. These integers will be stored in a private array called `inData[]`, which values of which can be accessed using `getData(int i)`.

write()

This command will write `outCnt` integers from the incoming serial buffer sent from the main computer to the Arduino. These integers are sent from a private array called `outData[]`, which can be set using `setData(int i, int val)`.

setData(int i, int val)

This command is used to set the data to be sent with the `write()` command. The `i` parameter sets the integer value to be set to `val` at the *ith* index (zero-indexed). For example, if we wanted the third value to be 42, we use the command `setData(2,42)`.

getData(int i)

This command is used to get the data collected from the `read()` command.The `i` parameter will get the integer value from the *ith* index from the read array (zero-indexed). For example, if we wanted to store the third value from the incoming read array to a variable `k`, we use the command `k = getData(2)`.