`ev3devices` – EV3 Devices

LEGO® MINDSTORMS® EV3 motors and sensors.

Motors

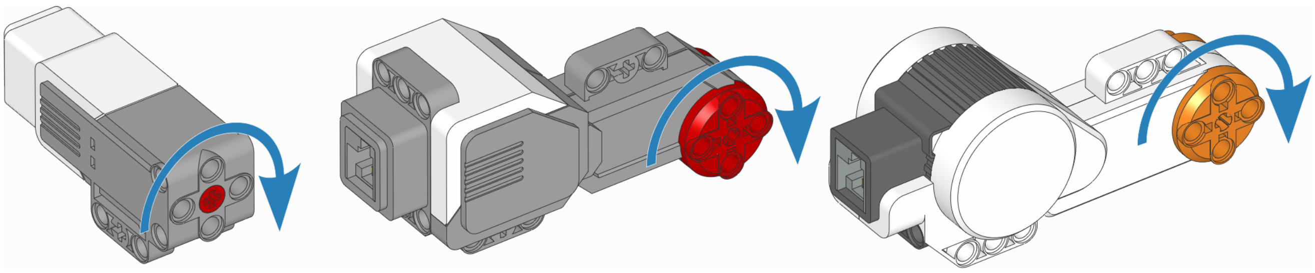


Figure 17 : EV3-compatible motors. The arrows indicate the default positive direction.

class `Motor`*(port, positive_direction=Direction.CLOCKWISE, gears=None)*

Generic class to control motors with built-in rotation sensors.

Parameters:

- **port** (*Port*) – Port to which the motor is connected.
- **positive_direction** (*Direction*) – Which direction the motor should turn when you give a positive speed value or angle.
- **gears** (*list*) –

  List of gears linked to the motor.

  For example: `[12, 36]` represents a gear train with a 12-tooth and a 36-tooth gear. Use a list of lists for multiple gear trains, such as `[[12, 36], [20, 16, 40]]`.

  When you specify a gear train, all motor commands and settings are automatically adjusted to account for the resulting gear ratio. The motor direction remains unchanged by this.

Measuring

speed()

Gets the speed of the motor.

Returns: Motor speed.

Return type: rotational speed: deg/s

`angle`()

Gets the rotation angle of the motor.

Returns: Motor angle.

Return type: angle: deg

`reset_angle`(*angle*)

Sets the accumulated rotation angle of the motor to a desired value.

Parameters: **angle** (angle: deg) – Value to which the angle should be reset.

Stopping

`stop`()

Stops the motor and lets it spin freely.

The motor gradually stops due to friction.

`brake`()

Passively brakes the motor.

The motor stops due to friction, plus the voltage that is generated while the motor is still moving.

`hold`()

Stops the motor and actively holds it at its current angle.

Action

`run`(*speed*)

Runs the motor at a constant speed.

The motor accelerates to the given speed and keeps running at this speed until you give a new command.

Parameters: **speed** (rotational speed: deg/s) – Speed of the motor.

`run_time`(*speed*, *time*, *then=Stop.HOLD*, *wait=True*)

Runs the motor at a constant speed for a given amount of time.

The motor accelerates to the given speed, keeps running at this speed, and then decelerates. The total maneuver lasts for exactly the given amount of `time`.

Parameters:

- **speed** (rotational speed: deg/s) – Speed of the motor.
- **time** (time: ms) – Duration of the maneuver.
- **then** (*Stop*) – What to do after coming to a standstill.
- **wait** (*bool*) – Wait for the maneuver to complete before continuing with the rest of the program.

`run_angle`(*speed*, *rotation_angle*, *then=Stop.HOLD*, *wait=True*)

Runs the motor at a constant speed by a given angle.

Parameters:

- **speed** (rotational speed: deg/s) – Speed of the motor.
- **rotation_angle** (angle: deg) – Angle by which the motor should rotate.
- **then** (*Stop*) – What to do after coming to a standstill.
- **wait** (*bool*) – Wait for the maneuver to complete before continuing with the rest of the program.

`run_target`(*speed*, *target_angle*, *then=Stop.HOLD*, *wait=True*)

Runs the motor at a constant speed towards a given target angle.

The direction of rotation is automatically selected based on the target angle. It does matter if `speed` is positive or negative.

Parameters:

- **speed** (rotational speed: deg/s) – Speed of the motor.
- **target_angle** (angle: deg) – Angle that the motor should rotate to.
- **then** (*Stop*) – What to do after coming to a standstill.
- **wait** (*bool*) – Wait for the motor to reach the target before continuing with the rest of the program.

`run_until_stalled`(*speed*, *then=Stop.COAST*, *duty_limit=None*)

Runs the motor at a constant speed until it stalls.

Parameters:

- **speed** (rotational speed: deg/s) – Speed of the motor.
- **then** (*Stop*) – What to do after coming to a standstill.
- **duty_limit** (percentage: %) – Torque limit during this command. This is useful to avoid applying the full motor torque to a geared or lever mechanism.

Returns: Angle at which the motor becomes stalled.

Return type: angle: deg

`dc`(*duty*)

Rotates the motor at a given duty cycle (also known as "power").

This method lets you use a motor just like a simple DC motor.

Parameters: **duty** (percentage: %) – The duty cycle (-100.0 to 100).

Advanced motion control

`track_target`(*target_angle*)

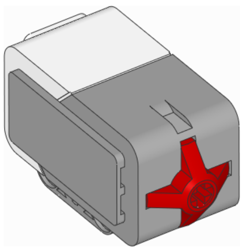
Tracks a target angle. This is similar to `run_target()`, but the usual smooth acceleration is skipped: it will move to the target angle as fast as possible. This method is useful if you want to continuously change the target angle.

Parameters: **target_angle** (angle: deg) – Target angle that the motor should rotate to.

`control`

The motors use PID control to accurately track the speed and angle targets that you specify. You can change its behavior through the `control` attribute of the motor. See The Control Class for an overview of available methods.

Touch Sensor



class `TouchSensor`(*port*)

LEGO® MINDSTORMS® EV3 Touch Sensor.

Parameters: **port** (*Port*) – Port to which the sensor is connected.

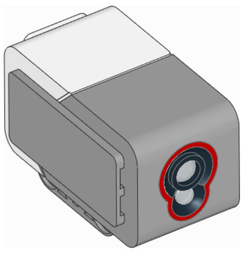
`pressed`()

Checks if the sensor is pressed.

Returns: `True` if the sensor is pressed, `False` if it is not pressed.

Return type: bool

Color Sensor



class **`ColorSensor`**(*port*)

LEGO® MINDSTORMS® EV3 Color Sensor.

Parameters: **port** (*Port*) – Port to which the sensor is connected.

`color`()

Measures the color of a surface.

Returns: `Color.BLACK`, `Color.BLUE`, `Color.GREEN`, `Color.YELLOW`, `Color.RED`, `Color.WHITE`, `Color.BROWN` or `None`.

Return type: `Color`, or `None` if no color is detected.

`ambient`()

Measures the ambient light intensity.

Returns: Ambient light intensity, ranging from 0 (dark) to 100 (bright).

Return type: percentage: %

`reflection`()

Measures the reflection of a surface using a red light.

Returns: Reflection, ranging from 0 (no reflection) to 100 (high reflection).

Return type: percentage: %

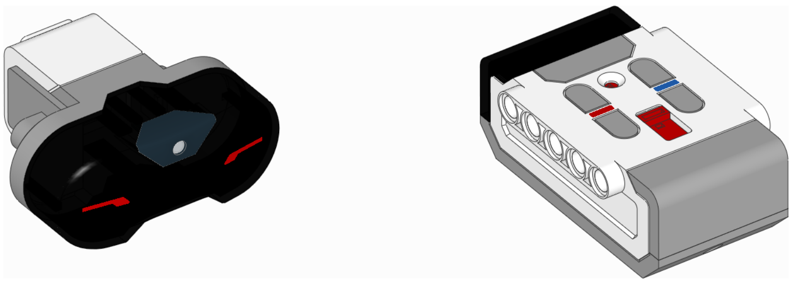
`rgb`()

Measures the reflection of a surface using a red, green, and then a blue light.

Returns: Tuple of reflections for red, green, and blue light, each ranging from 0.0 (no reflection) to 100.0 (high reflection).

Return type: (percentage: %, percentage: %, percentage: %)

Infrared Sensor and Beacon



class `InfraredSensor`(*port*)

LEGO® MINDSTORMS® EV3 Infrared Sensor and Beacon.

Parameters: **port** (*Port*) – Port to which the sensor is connected.

`distance`()

Measures the relative distance between the sensor and an object using infrared light.

Returns: Relative distance ranging from 0 (closest) to 100 (farthest).

Return type: relative distance: %

`beacon`(*channel*)

Measures the relative distance and angle between the remote and the infrared sensor.

Parameters: **channel** (*int*) – Channel number of the remote.

Returns: Tuple of relative distance (0 to 100) and approximate angle (-75 to 75 degrees) between remote and infrared sensor.

Return type: (relative distance: %, angle: deg) or (`None`, `None`) if no remote is detected.

`buttons`(*channel*)

Checks which buttons on the infrared remote are pressed.

This method can detect up to two buttons at once. If you press more buttons, you may not get useful data.

Parameters: **channel** (*int*) – Channel number of the remote.

Returns: List of pressed buttons on the remote on selected channel.

Return type: List of `Button`

`keypad`()

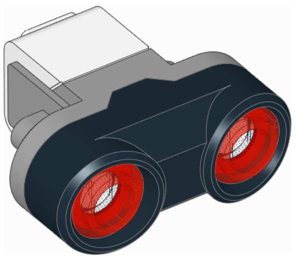
Checks which buttons on the infrared remote are pressed.

This method can independently detect all 4 up/down buttons, but it cannot detect the beacon button.

This method only works with the remote in channel 1.

Returns:	List of pressed buttons on the remote on selected channel.
Return type:	List of `Button`

Ultrasonic Sensor



class `UltrasonicSensor`(*port*)

LEGO® MINDSTORMS® EV3 Ultrasonic Sensor.

Parameters:	**port** (*Port*) – Port to which the sensor is connected.

`distance`(*silent=False*)

Measures the distance between the sensor and an object using ultrasonic sound waves.

Parameters:	**silent** (*bool*) – Choose `True` to turn the sensor off after measuring the distance. This reduces interference with other ultrasonic sensors. If you do this too frequently, the sensor can freeze. If this happens, unplug it and plug it back in.
Returns:	Distance.
Return type:	distance: mm

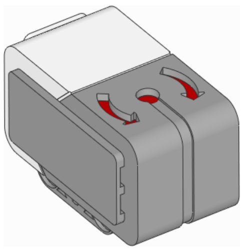
`presence`()

Checks for the presence of other ultrasonic sensors by detecting ultrasonic sounds.

If the other ultrasonic sensor is operating in silent mode, you can only detect the presence of that sensor while it is taking a measurement.

Returns:	`True` if ultrasonic sounds are detected, `False` if not.
Return type:	bool

Gyroscopic Sensor



class **`GyroSensor`**(*port*, *positive_direction=Direction.CLOCKWISE*)

LEGO® MINDSTORMS® EV3 Gyro Sensor.

Parameters:
- **port** (*Port*) – Port to which the sensor is connected.
- **positive_direction** (*Direction*) – Positive rotation direction when looking at the red dot on top of the sensor.

`speed`()

Gets the speed (angular velocity) of the sensor.

Returns: Sensor angular velocity.

Return type: rotational speed: deg/s

`angle`()

Gets the accumulated angle of the sensor.

Returns: Rotation angle.

Return type: angle: deg

If you use the `angle()` method, you cannot use the `speed()` method in the same program. Doing so would reset the sensor angle to zero every time you read the speed.

`reset_angle`(*angle*)

Sets the rotation angle of the sensor to a desired value.

Parameters: **angle** (angle: deg) – Value to which the angle should be reset.