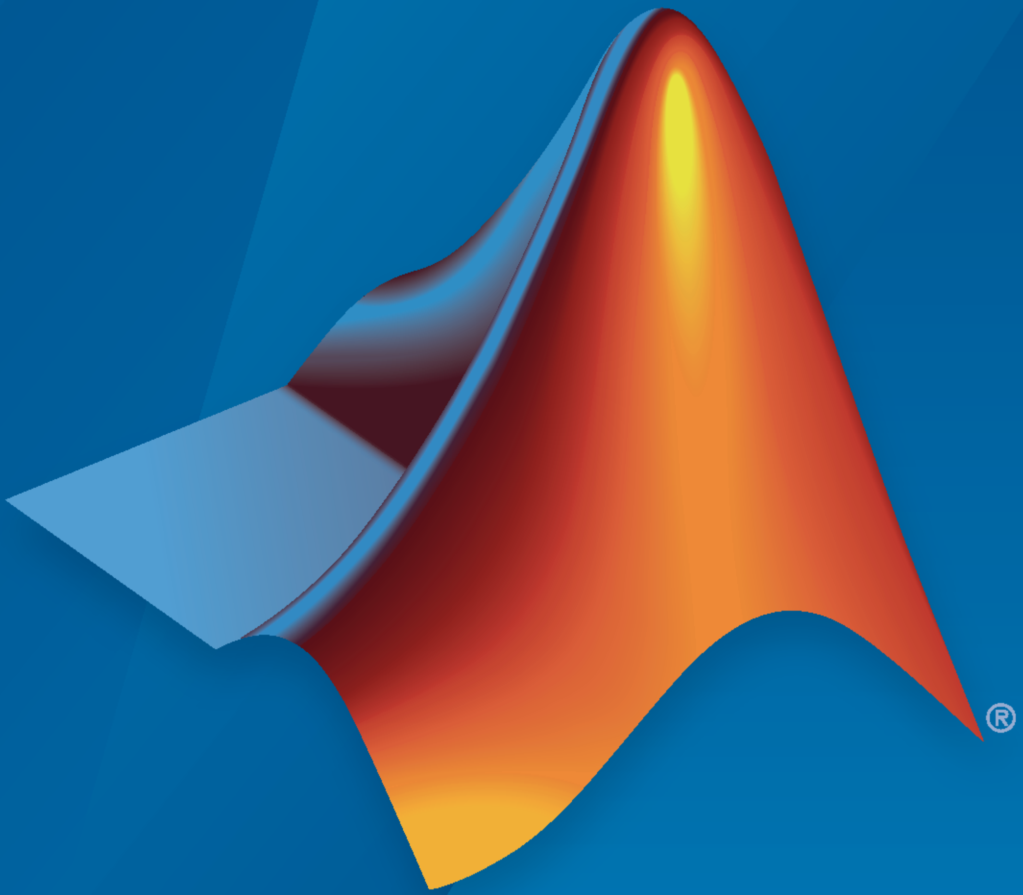
MATLAB® Support Package for LEGO® MINDSTORMS® EV3 Hardware

Reference



MATLAB® & SIMULINK®

R2023b

MathWorks®

MATLAB® Support Package for LEGO® MINDSTORMS® EV3 Hardware Reference

Revision History

September 2014	Online only	Revised for Version 1.0 (R2014b)
March 2015	Online only	Revised for Version 15.1.0 (R2015a)
September 2015	Online only	Revised for Version 15.2.0 (R2015b)
March 2016	Online only	Revised for Version 16.1.0 (R2016a)
September 2016	Online only	Revised for Version 16.2.0 (R2016b)
March 2017	Online only	Revised for Version 17.1.0 (R2017a)
September 2017	Online only	Revised for Version 17.2.0 (R2017b)
March 2018	Online only	Revised for Version 18.1.0 (R2018a)
September 2018	Online only	Revised for Version 18.2.0 (R2018b)
March 2019	Online only	Revised for Version 19.1.0 (R2019a)
September 2019	Online only	Revised for Version 19.2.0 (R2019b)
March 2020	Online only	Revised for Version 20.1.0 (R2020a)
September 2020	Online only	Revised for Version 20.2.0 (R2020b)
March 2021	Online only	Revised for Version 21.1.0 (R2021a)
September 2021	Online only	Revised for Version 21.2.0 (R2021b)
March 2022	Online only	Revised for Version 22.1.0 (R2022a)
September 2022	Online only	Revised for Version 22.2.0 (R2022b)
March 2023	Online only	Revised for Version 23.1.0 (R2023a)
September 2023	Online only	Revised for Version 23.2.0 (R2023b)

Contents

1

Objects and Functions

legoev3

Connection to LEGO MINDSTORMS EV3 brick

Description

This object represents a connection to a LEGO MINDSTORMS EV3 brick. To interact with peripherals on the EV3 brick, such as the speaker, LCD, buttons, or status light, use this object with the functions listed in “Object Functions” on page 1-4. To connect to sensors and motors, use this object with the following sensor objects: `touchSensor`, `irSensor`, `gyroSensor`, `colorSensor`, `sonicSensor`, or `motor`.

Creation

Syntax

```
myev3 = legoev3
myev3 = legoev3(communicationtype)
myev3 = legoev3(communicationtype,ipaddress,id)
myev3 = legoev3(communicationtype,comport)
```

Description

`myev3 = legoev3` creates a connection to an EV3 brick, `myev3`, that uses the same settings as the previous connection.

`myev3 = legoev3(communicationtype)` creates a connection to an EV3 brick using a specified connection type. communicationtype sets the CommunicationType property of the connection object.

`myev3 = legoev3(communicationtype,ipaddress,id)` creates a connection to an EV3 brick using a type of connection that requires the IP address and ID of the EV3 brick.

`myev3 = legoev3(communicationtype,comport)` creates a connection to an EV3 brick using a type of connection that requires a COM port number.

Input Arguments

`comport` — COM port
character vector

COM port or serial device on host computer, specified as a character vector.

Example: `'COM3'`

Example: `'/dev/tty.EV3-SerialPort'`

Data Types: `char`

Properties

`FirmwareVersion` — EV3 firmware version
character vector

This property is read-only.

EV3 firmware version, returned as a character vector.

Example: `'V1.03E'`

Data Types: `char`

`HardwareID` — Unique hardware ID
character vector

This property is read-only.

Unique hardware ID, returned as a character vector. You can set this property when you create the object. After you create the object, this property cannot be changed. This value is available in the EV3 interface under Brick Info.

Example: `'00165340e49b'`

Data Types: `char`

`IPAddress` — IP address
character vector

This property is read-only.

IP address when connecting over a wireless network, returned as a string. You can set this property when you create the object. After you create the object, this property cannot be changed. This value is available in the EV3 interface under Brick Info.

Example: `'192.168.1.2'`

Data Types: `char`

`CommunicationType` — Communication type
`USB` | `WiFi` | `Bluetooth`

This property is read-only.

Connection type, returned as a character vector. You can set this property when you create the object. After you create the object, this property cannot be changed.

Example: `'USB'`

Data Types: `char`

`BatteryLevel` — Percentage of battery power remaining
integer in the range [`0`, `100`]

This property is read-only.

Percentage of battery power remaining, returned as an int8 value.

Example: `68`

Data Types: `int8`

`ConnectedSensors` — List of connected sensors
`''` | `'color'` | `'infrared'` | `'touch'` | `'gyro'` | `'sonic'` | `'unknown'`

This property is read-only.

List of connected sensors, returned as a cell array of character vectors. `''` means no connected sensors.

Example: `{'touch' 'infrared' 'color' 'sonic'}`

Object Functions

clear	Remove items from workspace, freeing up system memory
writeLCD	Write characters to LCD on EV3 brick
clearLCD	Clear characters from LCD on EV3 brick
playTone	Play tones from speaker on EV3 brick
beep	Play beep from speaker on EV3 brick
readButton	Read whether button on EV3 brick is being pressed
writeStatusLight	Control color and mode of status light on EV3 brick

Examples

Reconnect to an EV3 Brick

If you have connected to the EV3 brick previously, you can reconnect without specifying any details.

```
myev3 = legoev3;
```

Connect to EV3 Brick with USB Cable

You can communicate with the EV3 brick over a USB cable.

Using the supplied USB cable, connect the host computer to the EV3 brick.

In the MATLAB Command Window, connect over USB.

```
myev3 = legoev3('USB');
```

Connect to EV3 Brick Over a Wireless Network

You can communicate with the EV3 brick over a wireless network connection.

Set up and configure a wireless network connection.

In the Command Window, connect using a wireless network. Specify the IP address and ID of the EV3 brick.

```
myev3 = legoev3('WiFi','192.168.1.2','00165340e49b');
```

Connect from Windows to EV3 Brick Over Bluetooth

You can communicate with the EV3 brick over a Bluetooth® connection.

Set up and configure a Bluetooth connection.

In the MATLAB Command Window, create a connection from a host computer running Windows® to an EV3 brick over Bluetooth. Specify the COM port number.

```
myev3 = legoev3('Bluetooth','COM3');
```

Connect from macOS to an EV3 Brick Over Bluetooth

You can communicate with the EV3 brick over a Bluetooth connection.

On macOS, Set up and configure a Bluetooth connection to the EV3 brick.

Open Terminal in macOS and enter:

```
ls /dev/tty*
```

```
/dev/tty.EV3-SerialPort
```

The command response gives the name of the Bluetooth connection from macOS to the EV3 Brick

In the MATLAB Command Window, create a connection from a host computer running macOS to an EV3 brick over Bluetooth. Specify the COM port number.

```
myev3 = legoev3('Bluetooth','/dev/tty.EV3-SerialPort');
```

See Also

`touchSensor` | `irSensor` | `gyroSensor` | `colorSensor` | `sonicSensor`

Topics

"Reconnect the Host Computer to an EV3 Brick"
"Connect Host Computer to EV3 Brick over USB Cable"
"Connect Host Computer to EV3 Brick over Wireless Network"
"Connect a Host Computer Running macOS to an EV3 Brick Using Bluetooth"
"Connect a Host Computer Running Windows to an EV3 Brick Using Bluetooth"
"EV3 Brick Hardware"
"Connections to EV3 Hardware"

writeLCD

Write characters to LCD on EV3 brick

Syntax

```
writeLCD(myev3,string)
writeLCD(myev3,string,row,column)
```

Description

`writeLCD(myev3,string)` displays a string near the center of the LCD located on the EV3 brick.

`writeLCD(myev3,string,row,column)` displays a string on a specific row and column of the LCD.

Examples

Display and Clear Text on the LCD

You can display text near the center of the LCD or at a specific location.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3;
```

Display text on the LCD.

```
writeLCD(myev3,'Collision Alert!')
```

The text appears near the center of the LCD.

Clear the text from the LCD.

```
clearLCD(myev3)
```

Display “X” in the lower-right corner of the LCD.

```
writeLCD(myev3,'X', 9, 19)
```

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

`string` — Characters to display
string

Characters to display on LCD, specified as a string.

Example: `'Collision Alert!'`

Data Types: `char`

`row` — Row on LCD
1 to 9

Row number on LCD, specified as a double.

Example: 2

Data Types: `double`

`column` — Column on LCD
1 to 19

Column number on LCD, specified as a double.

Example: 1

Data Types: `double`

See Also

`legoev3` | `clearLCD`

Topics
"Connections to EV3 Hardware"
"EV3 Brick Hardware"

clearLCD

Clear characters from LCD on EV3 brick

Syntax

`clearLCD(myev3)`

Description

`clearLCD(myev3)` clears characters that the `writeLCD` function displays on the LCD located on the EV3 brick.

Examples

Display and Clear Text on the LCD

You can display text near the center of the LCD or at a specific location.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3;
```

Display text on the LCD.

```
writeLCD(myev3,'Collision Alert!')
```

The text appears near the center of the LCD.

Clear the text from the LCD.

```
clearLCD(myev3)
```

Display “X” in the lower-right corner of the LCD.

```
writeLCD(myev3,'X', 9, 19)
```

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

See Also

`legoev3` | `writeLCD`

Topics
"Connections to EV3 Hardware"
"EV3 Brick Hardware"

playTone

Play tones from speaker on EV3 brick

Syntax

```
playTone(myev3)
playTone(myev3,freq,duration,volume)
```

Description

`playTone(myev3)` plays a 500 Hz tone for 1 second at volume level 10 on the speaker that is located on the EV3 brick.

`playTone(myev3,freq,duration,volume)` plays a tone with a specific frequency, duration, and volume.

Examples

Play Tone or Melody from Speaker

Play a tone, and then a short melody.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3;
```

Play a tone that uses the default values.

```
playTone(myev3)
```

Play a series of tones. Specify the frequencies, duration, and volume of each tone.

```
playTone(myev3,412.0,0.5,10)
pause(0.5)
playTone(myev3,550.0,0.75,50)
pause(0.75)
playTone(myev3,925.0,0.25,10)
pause(0.25)
playTone(myev3,734.2,0.80,50)
```

The pauses give each tone time to finish before the next tone starts.

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

`freq` — Frequency, in Hertz
`500` (default) | `250` to `10000`

The frequency of the tone in Hertz, specified as a double.

Example: `500`

Data Types: `double`

`duration` — Duration, in seconds
1 (default) | `0` to `30`

The duration of the tone in seconds, specified as a double.

Example: `1`

Data Types: `double`

`volume` — Relative volume
`10` (default) | `0` to `100`

The relative volume of the tone, specified as a double.

Example: `50`

Data Types: `double`

See Also

`legoev3` | `beep`

Topics
"Connections to EV3 Hardware"
"EV3 Brick Hardware"

beep

Play beep from speaker on EV3 brick

Syntax

```
beep(myev3)
beep(myev3,duration)
```

Description

`beep(myev3)` plays a beep for one-tenth of a second at 500 Hz from the speaker that is located on the EV3 brick.

`beep(myev3,duration)` plays a beep for a specific duration.

Examples

Play a Beep from the Speaker

Play a beep from the EV3 speaker for a short period, and then for a longer period.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3;
```

Play a beep for the default duration of 0.1 second.

```
beep(myev3)
```

Play a beep for 1 second.

```
beep(myev3,1)
```

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

`duration` — Duration, in seconds
`0.1` (default) | `0` to `30`

The duration of the beep in seconds, specified as a double.

Example: `1`

Data Types: `double`

See Also

`legoev3` | `playTone`

Topics

“Connections to EV3 Hardware”
“EV3 Brick Hardware”

readButton

Read whether button on EV3 brick is being pressed

Syntax

```
result = readButton(myev3,button)
```

Description

`result = readButton(myev3,button)` reads whether a specific button on the EV3 is being pressed, and returns the status as a logical value:

- `0` means not pressed.
- `1` means pressed.

Examples

Read the Status of a Button

Read the status of the center button on an EV3 brick.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3;
```

Hold down the center button on the EV3 and read the status of the button.

```
result = readButton(myev3,'center')
```

```
result =

     1
```

Release the center button on the EV3 and read the status of the button.

```
result = readButton(myev3,'center')
```

```
result =

     0
```

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

`button` — Button position
`up` | `down` | `left` | `right` | `center`

Position of the button.

Example: `'up'`

Data Types: `char`

Output Arguments

`result` — Status of button
`0` | `1`

The status of the button, returned as a logical value. `0` indicates not pressed. `1` indicates pressed.

See Also

`legoev3`

Topics
"Connections to EV3 Hardware"
"EV3 Brick Hardware"

writeStatusLight

Control color and mode of status light on EV3 brick

Syntax

```
writeStatusLight(myev3,color,mode)
writeStatusLight(myev3,off)
```

Description

`writeStatusLight(myev3,color,mode)` controls the status light that surrounds the five buttons on the front of the EV3 brick. This function turns the status light on or off, changes the color of the light, and makes the light pulse or remain on continuously.

`writeStatusLight(myev3,off)` turns the status light off.

Examples

Change the Color of the Status Light and Make it Flash

Change the color and mode of the status light on an EV3 brick.

Write a series of different colors and modes to the status light:

```
writeStatusLight(myev3,'orange','pulsing')
pause(3)
writeStatusLight(myev3,'red','solid')
pause(3)
writeStatusLight(myev3,'off')
pause(3)
writeStatusLight(myev3,'green','solid')
```

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

`color` — Color of status light
`off` (default) | `green` | `red` | `orange`

The color of the status light, specified as a string.

Example: `'orange'`

Data Types: `char`

`mode` — Pulsing or continuous light
`solid` (default) | `pulsing`

Pulsing or continuous light, specified as a string.

Example: `'pulsing'`

Data Types: `char`

`off` — Turns the status light off
`off`

Off, specified as a string.

Example: `'off'`

Data Types: `char`

See Also
`legoev3`

Topics
"Connections to EV3 Hardware"
"EV3 Brick Hardware"

colorSensor

Connection to color sensor

Description

This object represents a connection to an EV3 Color Sensor (item number 45506). To read color and light intensity, use this object with the functions listed in “Object Functions” on page 1-19.

Creation

Syntax

```
mycolorsensor = colorSensor(myev3)
mycolorsensor = colorSensor(myev3,inputport)
```

Description

`mycolorsensor = colorSensor(myev3)` creates a connection to a color sensor. You can use this connection with the functions listed in “Object Functions” on page 1-19 to read colors and measure light intensity.

If multiple color sensors are attached to the EV3 brick, this function chooses the sensor that is attached to the EV3 input port with the lowest number.

`mycolorsensor = colorSensor(myev3,inputport)` creates a connection to a color sensor that uses a specific EV3 input port defined by the InputPort property.

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

Output Arguments

`mycolorsensor` — Connection to color sensor
object handle

Connection to color sensor, returned as an object handle.

Properties

`InputPort` — Number of EV3 input port
1 | 2 | 3 | 4

This property is read-only.

Number of the EV3 input port that the sensor uses, returned as a double.

Example: 1

Data Types: `double`

Object Functions

readColor	Read color of object in front of color sensor
readLightIntensity	Read intensity of light that reaches color sensor

Examples

Use a Color Sensor to Read Color and Measure Light Intensity

Read the color and ambient light intensity from a color sensor.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to the color sensor.

```
mycolorsensor = colorSensor(myev3)
```

```
mycolorsensor =

  colorSensor with properties:

    InputPort: 3
```

Read the color in front of the color sensor.

```
color = readColor(mycolorsensor)
```

```
color =

white
```

Read the relative intensity of the ambient light that reaches the color sensor. If not specified, the default mode is `'ambient'`.

```
intensity = readLightIntensity(mycolorsensor)
intensity =

           9
```

Read the relative intensity of the LED light that a nearby object reflects back to the color sensor.

```
intensity = readLightIntensity(mycolorsensor,'reflected')
```

```
intensity =

          53
```

See Also

`legoev3` | `readColor` | `readLightIntensity`

Topics

"Connections to EV3 Hardware"
"Color Sensors"

readColor

Read color of object in front of color sensor

Syntax

`color = readColor(mycolorsensor)`

Description

`color = readColor(mycolorsensor)` reads the color of an object in front of the color sensor, and returns it as a string, such as `red`, `green`, or `blue`.

Examples

Use a Color Sensor to Read Color

Read the color of an object in front of a color sensor.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to the color sensor.

```
mycolorsensor = colorSensor(myev3)
```

```
mycolorsensor = 

  colorSensor with properties:

    InputPort: 3
```

Read the color in front of the color sensor.

```
color = readColor(mycolorsensor)
color =

white
```

Input Arguments

`mycolorsensor` — Connection to color sensor
object handle

Connection to a color sensor, specified as a string that represents the object created using `colorSensor`.

Example: `mycolorsensor`

Data Types: `char`

Output Arguments

`color` — Color name
`none` | `black` | `blue` | `green` | `yellow` | `red` | `white` | `brown`

The name of the color read by the color sensor, returned as a string.

See Also
`legoev3` | `colorSensor` | `readLightIntensity`

Topics
"Connections to EV3 Hardware"
"Color Sensors"

readLightIntensity

Read intensity of light that reaches color sensor

Syntax

`intensity = readLightIntensity(mycolorsensor,mode)`

Description

`intensity = readLightIntensity(mycolorsensor,mode)` measures the intensity of the light that reaches the color sensor, and returns it as value from 0 to 100 (dark to light). The default mode of this function measures ambient light from the environment. When you set the mode to measure reflected light, the sensor emits light from a red LED and measures the amount of light reflected by nearby objects.

Examples

Use a Color Sensor to Measure Ambient and Reflected Light

Measure ambient and reflected light using a color sensor.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to the color sensor.

```
mycolorsensor = colorSensor(myev3)
```

```
mycolorsensor =

  colorSensor with properties:

    InputPort: 3
```

Read the relative intensity of the ambient light that reaches the color sensor. If not specified, the default mode is `'ambient'`.

```
intensity = readLightIntensity(mycolorsensor)
```

```
intensity =

          9
```

Read the relative intensity of the LED light that a nearby object reflects back to the color sensor.

```
intensity = readLightIntensity(mycolorsensor,'reflected')
```

```
intensity =

         53
```

Input Arguments

`mycolorsensor` — Connection to color sensor
object handle

Connection to a color sensor, specified as a string that represents the object created using `colorSensor`.

Example: `mycolorsensor`

Data Types: `char`

`mode` — Read ambient or reflected light
`ambient` (default) | `reflected`

Read ambient or reflected light, specified as a string.

Example: `'reflected'`

Data Types: `char`

Output Arguments

`intensity` — Light intensity
`0` to `100`

Relative light intensity, returned as an int32 value, from 0 to 100 (dark to light).

See Also

`legoev3` | `colorSensor` | `readColor`

Topics
"Connections to EV3 Hardware"
"Color Sensors"

sonicSensor

Connection to ultrasonic sensor

Description

This object represents a connection to an EV3 Ultrasonic Sensor (item number 45504). To measure the distance from the sensor to an object in meters, use this object with the `readDistance` function.

Creation

Syntax

```
mysonicsensor = sonicSensor(myev3)
mysonicsensor = sonicSensor(myev3,inputport)
```

Description

`mysonicsensor = sonicSensor(myev3)` creates a connection to a ultrasonic sensor. You can use this connection with the `readDistance` function to measure the distance from an ultrasonic sensor to an object.

If multiple ultrasonic sensors are attached to the EV3 brick, this function chooses the sensor that is attached to the EV3 input port with the lowest number.

`mysonicsensor = sonicSensor(myev3,inputport)` creates a connection to an ultrasonic sensor that uses a specific EV3 input port defined by the InputPort property.

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

Output Arguments

`mysonicsensor` — Connection to ultrasonic sensor
object handle

Connection to ultrasonic sensor, returned as an object handle.

Properties

`InputPort` — Number of EV3 input port
1 | 2 | 3 | 4

This property is read-only.

Number of the EV3 input port that the sensor uses, returned as a double.

Example: 1

Data Types: double

Object Functions

readDistance Read distance from ultrasonic sensor to object

Examples

Use an Ultrasonic Sensor to Measure Distance

Read the distance, in meters, between an ultrasonic sensor and an object.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Connect to the ultrasonic sensor.

```
mysonicsensor = sonicSensor(myev3)
```

```
mysonicsensor =

  sonicSensor with properties:

    InputPort: 4
```

Read the distance, in meters, from the ultrasonic sensor to the nearest object.

```
distance = readDistance(mysonicsensor)
```

```
distance =

    0.4600
```

See Also

legoev3 | readDistance

Topics

"Connections to EV3 Hardware"
"Sonic Sensors"

readDistance

Read distance from ultrasonic sensor to object

Syntax

`distance = readDistance(mysonicsensor)`

Description

`distance = readDistance(mysonicsensor)` measures the distance from the ultrasonic sensor to an object, and returns the measurement in meters.

Examples

Use an Ultrasonic Sensor to Measure Distance

Read the distance, in meters, between an ultrasonic sensor and an object.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Connect to the ultrasonic sensor.

```
mysonicsensor = sonicSensor(myev3)
```

```
mysonicsensor = 

  sonicSensor with properties:

    InputPort: 4
```

Read the distance, in meters, from the ultrasonic sensor to the nearest object.

```
distance = readDistance(mysonicsensor)
distance =

    0.4600
```

Input Arguments

`mysonicsensor` — Connection to ultrasonic sensor
object handle

Connection to ultrasonic sensor, specified as a string that represents the object created using `sonicSensor`.

Example: `mysonicsensor`

Data Types: `char`

Output Arguments

`distance` — Distance to object
`0` to `2.55`

Distance to object in meters, returned as a double.

See Also

`legoev3` | `sonicSensor`

Topics
"Connections to EV3 Hardware"
"Sonic Sensors"

touchSensor

Connection to touch sensor

Description

This object represents a connection to an EV3 Touch Sensor (item number 45507). To check whether an item is pressing the front of the sensor, use this object with the `readTouch` function.

Creation

Syntax

```
mytouchsensor = touchSensor(myev3)
mytouchsensor = touchSensor(myev3,inputport)
```

Description

`mytouchsensor = touchSensor(myev3)` creates a connection to a touch sensor.

If multiple touch sensors are attached to the EV3 brick, this function chooses the sensor that is attached to the EV3 input port with the lowest number.

`mytouchsensor = touchSensor(myev3,inputport)` creates a connection to a touch sensor that uses a specific EV3 input port defined by the InputPort property.

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

Output Arguments

`mytouchsensor` — Connection to touch sensor
object handle

Connection to touch sensor, returned as an object handle.

Properties

`InputPort` — Number of EV3 input port
1 | 2 | 3 | 4

This property is read-only.

Number of the EV3 input port that the sensor uses, returned as a double.

Example: 1

Data Types: `double`

Object Functions

readTouch	Read touch sensor value

Examples

Use a Touch Sensor to Detect Objects

Determine whether an object is pressing the touch sensor.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Connect to a touch sensor.

```
mytouchsensor = touchSensor(myev3)
```

```
mytouchsensor = 

  touchSensor with properties:

    InputPort: 1
```

Read the state of the touch sensor.

```
pressed = readTouch(mytouchsensor)
```

```
pressed =

     0
```

This result indicates that an object is not pressing the touch sensor.

Press the touch sensor while you take another reading.

```
pressed = readTouch(mytouchsensor)
pressed =

     1
```

This result indicates that an object is pressing the touch sensor.

See Also

`irSensor` | `gyroSensor` | `colorSensor` | `sonicSensor`

Topics

"Connections to EV3 Hardware"
"Touch Sensors"

readTouch

Read touch sensor value

Syntax

`pressed = readTouch(mytouchsensor)`

Description

`pressed = readTouch(mytouchsensor)` reads whether an object is pressing the front of the touch sensor, and returns the status as a logical value:

- `0` means not pressed.
- `1` means pressed.

Examples

Use a Touch Sensor to Detect Objects

Determine whether an object is pressing the touch sensor.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Connect to a touch sensor.

```
mytouchsensor = touchSensor(myev3)
```

```
mytouchsensor = 

  touchSensor with properties:

    InputPort: 1
```

Read the state of the touch sensor.

```
pressed = readTouch(mytouchsensor)
pressed =

     0
```

This result indicates that an object is not pressing the touch sensor.

Press the touch sensor while you take another reading.

```
pressed = readTouch(mytouchsensor)
```

```
pressed =

     1
```

This result indicates that an object is pressing the touch sensor.

Input Arguments

`mytouchsensor` — Connection to touch sensor
object handle

Connection to touch sensor, specified as a string that represents the object created using `touchSensor`.

Example: `mytouchsensor`

Data Types: `char`

Output Arguments

`pressed` — Touch sensor status
`0` | `1`

Touch sensor status, returned as a logical value:

- `0` means not pressed.
- `1` means pressed.

See Also
`legoev3` | `touchSensor`

Topics
“Connections to EV3 Hardware”
“Touch Sensors”

irSensor

Connection to infrared sensor

Description

This object represents a connection to an EV3 Infrared Sensor (item number 45509). To get the relative distance of an object in front of the sensor, or to get data from an EV3 Remote Infrared Beacon (item number 45508), use this object with the functions listed in “Object Functions” on page 1-36.

Creation

Syntax

```
myirsensor = irSensor(myev3)
myirsensor = irSensor(myev3,inputport)
```

Description

`myirsensor = irSensor(myev3)` creates a connection to an infrared sensor. You can use this connection with the functions listed in “Object Functions” on page 1-36 to get the relative distance of an object in front of the sensor, or to get data from an EV3 Remote Infrared Beacon.

If multiple infrared sensors are attached to the EV3 brick, this function chooses the sensor that is attached to the EV3 input port with the lowest number.

`myirsensor = irSensor(myev3,inputport)` creates a connection to an infrared sensor that uses a specific EV3 input port by the InputPort property.

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

Output Arguments

`myirsensor` — Connection to infrared sensor
object handle

Connection to infrared sensor, returned as an object handle.

Properties

`Channel` — IR beacon channel
1 (default) | 2 | 3 | 4

The channel number that the IR beacon uses, specified as a double. The red switch on the beacon has four positions that determine the channel number. The forward position is channel 1.

Example: 2

Data Types: `double`

`InputPort` — Number of EV3 input port
1 | 2 | 3 | 4

This property is read-only.

Number of the EV3 input port that the sensor uses, returned as a double.

Example: 1

Data Types: `double`

Object Functions

readProximity	Read distance from infrared sensor to object
readBeaconProximity	Read distance and heading from infrared sensor to beacon
readBeaconButton	Read number of pressed button on infrared beacon

Examples

Use an Infrared Sensor to Get the Distance and Heading to an IR Beacon

Get data from an infrared sensor, such as the distance to an object, the distance and heading to an IR beacon, or the number of a pressed button on an IR beacon.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 0
     ConnectedSensors: {'touch'  'infrared'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to an infrared sensor.

```
myirsensor = irSensor(myev3)
myirsensor =

  irSensor with properties:

      Channel: 1
    InputPort: 2
```

Read the relative distance from the infrared sensor to the nearest object.

```
proximity = readProximity(myirsensor)
```

```
proximity =

          76
```

Read the relative distance and heading from the infrared sensor to an IR beacon that is using channel 2.

```
[proximity,heading] = readBeaconProximity(myirsensor,2)
```

```
proximity =

 -128


heading =

    0
```

These values indicate that no one is pressing a button on the IR beacon.

Press a button on the IR beacon and try again.

```
[proximity,heading] = readBeaconProximity(myirsensor,2)
```

```
proximity =

   26


heading =

  -14
```

The IR beacon is 26 distance units away from, and 14 heading units to the left of, the sensor.

Check which button someone presses on the IR beacon.

```
beaconbutton = readBeaconButton(myirsensor,2)
```

```
beaconbutton =

           3
```

Someone is pressing button 3.

See Also

`legoev3` | `readProximity` | `readBeaconProximity` | `readBeaconButton`

Topics
"Connections to EV3 Hardware"
"IR Sensor and Remote Infrared Beacon"

readProximity

Read distance from infrared sensor to object

Syntax

`proximity = readProximity(myirsensor)`

Description

`proximity = readProximity(myirsensor)` measures the relative distance from the infrared sensor to a nearby object, and returns it as a value from `0` to `100` (near to far). The maximum range of the sensor is approximately 70 cm (27 inches).

Examples

Use an Infrared Sensor to Measure Distance

Measure the relative distance from an infrared sensor to an object.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 0
     ConnectedSensors: {'touch'  'infrared'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to an infrared sensor.

```
myirsensor = irSensor(myev3)
```

```
myirsensor =

  irSensor with properties:

      Channel: 1
    InputPort: 2
```

Read the relative distance from the infrared sensor to the nearest object.

```
proximity = readProximity(myirsensor)
proximity =

         76
```

Input Arguments

`myirsensor` — Connection to infrared sensor
object handle

Connection to infrared sensor, specified as a string that represents the object created using `irSensor`.

Data Types: `char`

Output Arguments

`proximity` — Relative distance to object
`0` to `100`

The relative distance, returned as an int32 value, from the infrared sensor to a nearby object based on the amount of infrared light that the object reflects.

See Also

`irSensor` | `readBeaconProximity` | `readBeaconButton`

Topics
“Connections to EV3 Hardware”
“IR Sensor and Remote Infrared Beacon”

readBeaconProximity

Read distance and heading from infrared sensor to beacon

Syntax

```
[proximity,heading] = readBeaconProximity(myirsensor,channel)
```

Description

`[proximity,heading] = readBeaconProximity(myirsensor,channel)` measures the relative proximity and heading from the infrared sensor to the infrared beacon. The function returns the proximity as a value from 0 to 100 (near to far), and returns the heading as a value from -25 to 25 (left to right). The maximum range from the sensor to the beacon is approximately 200 cm (79 inches).

Examples

Use an Infrared Sensor to Measure the Distance and Heading to an IR Beacon

Measure the relative distance from an infrared sensor to an EV3 IR beacon.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 0
     ConnectedSensors: {'touch'  'infrared'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to an infrared sensor.

```
myirsensor = irSensor(myev3)
```

```
myirsensor = 

  irSensor with properties:

      Channel: 1
    InputPort: 2
```

Read the relative distance and heading from the infrared sensor to an IR beacon that is using channel 2.

```
[proximity,heading] = readBeaconProximity(myirsensor,2)

proximity =

 -128


heading =

    0
```

These values indicate that no one is pressing a button on the IR beacon.

Press a button on the IR beacon and try again.

```
[proximity,heading] = readBeaconProximity(myirsensor,2)

proximity =

   26


heading =

  -14
```

The IR beacon is 26 distance units away from, and 14 heading units to the left of, the sensor.

Input Arguments

`myirsensor` — Connection to infrared sensor
object handle

Connection to infrared sensor, specified as a string that represents the object created using `irSensor`.

Data Types: `char`

`channel` — IR beacon channel
1 (default) | 2 | 3 | 4

The channel number that the IR beacon uses, specified as a double. The red switch on the beacon has four positions that determine the channel number. The forward position is channel 1.

Example: 2

Data Types: `double`

Output Arguments

`proximity` — Relative proximity of IR beacon
`0` to `100`

Relative distance from the sensor to the IR beacon, returned as an int8 value, from 0 to 100 (near to far).

`heading` — Relative heading of IR beacon
`-25` to `25`

Relative heading from the infrared sensor to the IR beacon, returned as an int8 value, from -25 to 25 (left to right).

See Also

`irSensor` | `readProximity` | `readBeaconButton`

Topics
"Connections to EV3 Hardware"
"IR Sensor and Remote Infrared Beacon"

readBeaconButton

Read number of pressed button on infrared beacon

Syntax

`button = readBeaconButton(myirsensor,channel)`

Description

`button = readBeaconButton(myirsensor,channel)` returns the number of a pressed button or buttons on an infrared beacon, and returns it as a numeric value, from `0` to `11`.

Examples

Use an Infrared Sensor to Read the Number of a Pressed Button on an IR Beacon

Use an infrared sensor to read the number of a pressed button or buttons on an IR beacon.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 0
     ConnectedSensors: {'touch'  'infrared'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to an infrared sensor.

```
myirsensor = irSensor(myev3)
```

```
myirsensor = 

  irSensor with properties:

      Channel: 1
    InputPort: 2
```

Check which button someone presses on the IR beacon.

```
beaconbutton = readBeaconButton(myirsensor,2)
beaconbutton =

           3
```

Someone is pressing button 3.

Input Arguments

`myirsensor` — Connection to infrared sensor
object handle

Connection to infrared sensor, specified as a string that represents the object created using `irSensor`.

Data Types: `char`

`channel` — IR beacon channel
1 (default) | 2 | 3 | 4

The channel number that the IR beacon uses, specified as a double. The red switch on the beacon has four positions that determine the channel number. The forward position is channel 1.

Example: 2

Data Types: `double`

Output Arguments

`button` — Button number
`0` to `11`

The number of a pressed button on the IR beacon, returned as an int32 value.

Zero indicates that no one is currently pressing a button.

There are 5 buttons. The top one is a toggle button. When you press it once, it lights up and this function returns 9.

With the top button toggled on, pressing a combination of 2 buttons outputs a value from 5 to 11:

- 1 & 2 = 10
- 1 & 3 = 5
- 1 & 4 = 6
- 2 & 3 = 7
- 2 & 4 = 8
- 3 & 4 = 11

See Also
`irSensor` | `readProximity` | `readBeaconProximity`

Topics
“Connections to EV3 Hardware”

"IR Sensor and Remote Infrared Beacon"

gyroSensor

Connection to gyroscopic sensor

Description

This object represents a connection to a single-axis EV3 Gyro Sensor (item number 45505). To measure total rotation or rate of rotation, use this object with the functions listed in “Object Functions” on page 1-48.

Creation

Syntax

```
mygyrosensor = gyroSensor(myev3)
mygyrosensor = gyroSensor(myev3,inputport)
```

Description

`mygyrosensor = gyroSensor(myev3)` creates a connection to a gyroscopic sensor. You can use this connection with the functions listed in “Object Functions” on page 1-48 to measure the rotation rate or rotation angle of the sensor, and reset the rotation rate measurement to zero.

If multiple gyroscopic sensors are attached to the EV3 brick, this function chooses the sensor that is attached to the EV3 input port with the lowest number.

`mygyrosensor = gyroSensor(myev3,inputport)` creates a connection to a gyroscopic sensor that uses a specific EV3 input port by the InputPort property.

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

Output Arguments

`mygyrosensor` — Connection to gyroscopic sensor
object handle

Connection to gyroscopic sensor, returned as an object handle.

Properties

`InputPort` — Number of EV3 input port
1 | 2 | 3 | 4

This property is read-only.

Number of the EV3 input port that the sensor uses, returned as a double.

Example: 1

Data Types: `double`

Object Functions

readRotationAngle	Read rotation angle from gyroscopic sensor
readRotationRate	Read rotation rate from gyroscopic sensor
resetRotationAngle	Reset rotation angle to zero

Examples

Use a Gyroscopic Sensor to Measure Rotation

Get rotation measurements from a gyroscopic sensor.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to the gyroscopic sensor called *`mygyrosensor`*.

```
mygyrosensor = gyroSensor(myev3)
```

```
mygyrosensor = 

  gyroSensor with properties:

    InputPort: 2
```

Measure the rotation, in degrees, since the creation of the connection to the sensor.

```
angle = readRotationAngle(mygyrosensor)
```

```
ans =

            59
```

The sensor has rotated 59 degrees clockwise.

Reset the measurement to zero.

```
resetRotationAngle(mygyrosensor)
```

Measure the current rate of rotation, in degrees per second.

```
rate = readRotationRate(mygyrosensor)
```

```
ans =

            12
```

See Also

`legoev3` | `readRotationAngle` | `readRotationRate` | `resetRotationAngle`

Topics

“Connections to EV3 Hardware”
“Gyro Sensors”

readRotationAngle

Read rotation angle from gyroscopic sensor

Syntax

`angle = readRotationAngle(mygyrosensor)`

Description

`angle = readRotationAngle(mygyrosensor)` reads the total amount of rotation since the creation of the connection to the sensor, and returns the measurement in degrees. You can use the `resetRotationAngle` function to reset this value to zero.

Examples

Use a Gyroscopic Sensor to Measure Angle of Rotation

Measure angle of rotation using a gyroscopic sensor.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to the gyroscopic sensor called *`mygyrosensor`*.

```
mygyrosensor = gyroSensor(myev3)
```

```
mygyrosensor =

  gyroSensor with properties:

    InputPort: 2
```

Measure the rotation, in degrees, since the creation of the connection to the sensor.

```
angle = readRotationAngle(mygyrosensor)
```

```
ans =

            59
```

The sensor has rotated 59 degrees clockwise.

Reset the measurement to zero.

```
resetRotationAngle(mygyrosensor)
```

Input Arguments

`mygyrosensor` — Connection to gyroscopic sensor
object handle

Connection to gyroscopic sensor, specified as a string that represents the object created using `gyroSensor`.

Example: `mygyrosensor`

Data Types: `double`

Output Arguments

`angle` — Angle of rotation
angle of rotation

Angle of rotation in degrees, returned as an int32 value. With the sensor upright, positive values indicate clockwise rotation. Negative values indicate counterclockwise rotation.

See Also

`gyroSensor` | `readRotationRate` | `resetRotationAngle`

Topics
“Connections to EV3 Hardware”
“Gyro Sensors”

readRotationRate

Read rotation rate from gyroscopic sensor

Syntax

```
rotationrate = readRotationRate(mygyrosensor)
```

Description

`rotationrate = readRotationRate(mygyrosensor)` reads the rate of rotation of the sensor, and returns the measurement in degrees per second.

Examples

Use a Gyroscopic Sensor to Measure Rate of Rotation

Measure rate of rotation, degrees per second, using a gyroscopic sensor.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to the gyroscopic sensor called *mygyrosensor*.

```
mygyrosensor = gyroSensor(myev3)
```

```
mygyrosensor = 

  gyroSensor with properties:

    InputPort: 2
```

Measure the current rate of rotation, in degrees per second.

```
rate = readRotationRate(mygyrosensor)
```

```
ans =

           12
```

The gyro sensor is rotating at 12 degrees per second.

Input Arguments

`mygyrosensor` — Connection to gyroscopic sensor
object handle

Connection to gyroscopic sensor, specified as a string that represents the object created using `gyroSensor`.

Example: `mygyrosensor`

Data Types: `double`

Output Arguments

`rotationrate` — Rotation rate
`-440` to `440`

The rate of rotation, returned as an int32 value, from -440 to 440 degrees per second.

See Also

`gyroSensor` | `readRotationAngle` | `resetRotationAngle`

Topics
"Connections to EV3 Hardware"
"Gyro Sensors"

resetRotationAngle

Reset rotation angle to zero

Syntax

```
resetRotationAngle(mygyrosensor)
```

Description

`resetRotationAngle(mygyrosensor)` resets the rotation angle measurement to zero. You can use this function to reset the value that the `readRotationAngle` function returns to zero.

Examples

Reset the Angle of Rotation Measurement to Zero

After measuring the angle of rotation using a gyroscopic sensor, you can reset the measurement value to zero.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to the gyroscopic sensor called *`mygyrosensor`*.

```
mygyrosensor = gyroSensor(myev3)
```

```
mygyrosensor =

  gyroSensor with properties:

    InputPort: 2
```

Measure the rotation, in degrees, since the creation of the connection to the sensor.

```
angle = readRotationAngle(mygyrosensor)
```

```
ans =

          59
```

The sensor has rotated 59 degrees clockwise.

Reset the measurement to zero.

```
resetRotationAngle(mygyrosensor)
```

Input Arguments

`mygyrosensor` — Connection to gyroscopic sensor
object handle

Connection to gyroscopic sensor, specified as a string that represents the object created using `gyroSensor`.

Example: `mygyrosensor`

Data Types: `double`

See Also

`gyroSensor` | `readRotationAngle` | `readRotationRate`

Topics
"Connections to EV3 Hardware"
"Gyro Sensors"

motor

Connection to motor

Description

This object represents a connection to an EV3 Large or Medium Motor (item numbers 45502 or 45503). To control the speed and direction of the motor, assign a value to the `Speed` property of the `motor` object. To start, stop, and measure the rotation of a motor, use this object with the functions listed in "Object Functions" on page 1-57.

Creation

Syntax

`mymotor = motor(myev3,outputport)`

Description

`mymotor = motor(myev3,outputport)` creates a connection to a motor that is attached to an EV3 output port by the OutputPort (see Properties), and returns an object handle for that connection. You can use this connection with the functions listed in "Object Functions" on page 1-57 to start, stop, and measure the rotation of a motor.

Input Arguments

`myev3` — Connection to EV3 brick
object handle

Connection to EV3 brick, specified as a string that represents the object created using `legoev3`.

Example: `myev3`

Data Types: `char`

Output Arguments

`mymotor` — Connection to motor
object handle

Connection to a motor, returned as an object handle.

Properties

`OutputPort` — Letter of EV3 output port
A | B | C | D

This property is read-only.

Letter of EV3 output port that the motor uses, returned as a string.

Example: `'A'`

Data Types: `char`

Speed — Motor speed and direction
`0` (default) | `-100` to `100`

The speed and direction of the motor, specified as a double.

To control the speed and direction, assign a value to this property. Input values from `-100` to `100` (full reverse to full forward). You can change this value while the motor is running or stopped.

Example: `50`

Data Types: `double`

Object Functions

start	Start motor
stop	Stop motor
readRotation	Read rotation from motor
resetRotation	Reset rotation count to zero

Examples

Control a Motor and Measure Wheel Rotation

Control a motor and measure wheel rotation.

Create a connection to the EV3 brick called *`myev3`*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to a motor. Specify the letter of the EV3 output port that connects to the motor.

```
mymotor = motor(myev3,'A')
```

```
mymotor = 

  motor with properties:

    OutputPort: 'A'
         Speed: 0
```

By default, the speed is zero.

Configure the motor to reverse at half speed.

```
mymotor.Speed = -50
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: -50
```

Start the motor.

```
start(mymotor)
```

Change the direction and speed of the motor.

```
mymotor.Speed = 70
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: 70
```

Stop the motor.

```
stop(mymotor)
```

Read the rotation value, in degrees, of the motor.

```
rotation = readRotation(mymotor)
```

```
ans =

       -1489
```

Reset the rotation value of the motor to zero.

```
resetRotation(mymotor)
```

Read the rotation again.

```
readRotation(mymotor)
```

```
ans =

       0
```

See Also

`legoev3` | `readRotation` | `resetRotation` | `start` | `stop`

Topics

"Connections to EV3 Hardware"
"Motors"

start

Start motor

Syntax

`start(mymotor)`

Description

`start(mymotor)` starts the motor, using the speed and direction specified by the `Speed` property.

Examples

Control a Motor

Start a motor, change the speed and direction of the motor, and then stop the motor.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to a motor. Specify the letter of the EV3 output port that connects to the motor.

```
mymotor = motor(myev3,'A')
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: 0
```

By default, the speed is zero.

Configure the motor to reverse at half speed.

```
mymotor.Speed = -50
```

```
mymotor =
  motor with properties:

    OutputPort: 'A'
         Speed: -50
```

Start the motor.

```
start(mymotor)
```

Change the direction and speed of the motor.

```
mymotor.Speed = 70
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: 70
```

Stop the motor.

```
stop(mymotor)
```

Input Arguments

`mymotor` — Connection to motor
object handle

Connection to a motor, specified as a string that represents the object created using `motor`.

Example: `mymotor`

Data Types: `char`

See Also

`motor` | `stop` | `readRotation` | `resetRotation`

Topics
"Connections to EV3 Hardware"
"Motors"

stop

Stop motor

Syntax

```
stop(mymotor)
stop(mymotor,mode)
```

Description

`stop(mymotor)` stops supplying power to the motor and lets the motor coast to a stop.

`stop(mymotor,mode)` stops supplying power to the motor and uses electromagnetic braking to stop the motor immediately.

Examples

Control a Motor

Start a motor, change the speed and direction of the motor, and then stop the motor.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 = 

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to a motor. Specify the letter of the EV3 output port that connects to the motor.

```
mymotor = motor(myev3,'A')
```

```
mymotor = 

  motor with properties:

    OutputPort: 'A'
         Speed: 0
```

By default, the speed is zero.

Configure the motor to reverse at half speed.

```
mymotor.Speed = -50

mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: -50
```

Start the motor.

```
start(mymotor)
```

Change the direction and speed of the motor.

```
mymotor.Speed = 70

mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: 70
```

Stop the motor.

```
stop(mymotor)
```

Input Arguments

`mymotor` — Connection to motor
object handle

Connection to a motor, specified as a string that represents the object created using `motor`.

Example: `mymotor`

Data Types: `char`

`mode` — Breaking mode
`0` (default) | `1`

Breaking mode, specified as a logical value. With `0`, the motor coasts to stop gradually. With `1`, the motor brakes to stop immediately.

Example: `1`

Data Types: `logical`

See Also
`motor` | `start` | `readRotation` | `resetRotation`

Topics
"Connections to EV3 Hardware"
"Motors"

readRotation

Read rotation from motor

Syntax

`rotation = readRotation(mymotor)`

Description

`rotation = readRotation(mymotor)` reads the total amount of rotation since the creation of the connection to the motor, and returns the measurement in degrees. You can use the `resetRotation` function to reset this value to zero.

Examples

Measure the Wheel Rotation of a Motor

Control a motor, measure the wheel rotation of the motor in degrees, and clear the measurement.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to a motor. Specify the letter of the EV3 output port that connects to the motor.

```
mymotor = motor(myev3,'A')
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: 0
```

By default, the speed is zero.

Configure the motor to reverse at half speed.

```
mymotor.Speed = -50
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: -50
```

Start the motor.

```
start(mymotor)
```

Read the rotation value, in degrees, of the motor.

```
rotation = readRotation(mymotor)
```

```
ans =

       -1489
```

Reset the rotation value of the motor to zero.

```
resetRotation(mymotor)
```

Input Arguments

`mymotor` — Connection to motor
object handle

Connection to a motor, specified as a string that represents the object created using `motor`.

Example: `mymotor`

Data Types: `char`

Output Arguments

`rotation` — rotation
degrees

Rotation of motor in degrees, returned as an int32 value. Creating a connection to a motor initializes this value to zero. To reset this value to zero, use `resetRotation`.

See Also
`motor` | `start` | `stop` | `resetRotation`

Topics
“Connections to EV3 Hardware”
“Motors”

resetRotation

Reset rotation count to zero

Syntax

`resetRotation(mymotor)`

Description

`resetRotation(mymotor)` resets the rotation measurement to zero. You can use this function to reset the value that the `readRotation` function returns to zero.

Examples

Measure the Wheel Rotation of a Motor

Control a motor, measure the wheel rotation of the motor in degrees, and clear the measurement.

Create a connection to the EV3 brick called *myev3*.

```
myev3 = legoev3
```

```
myev3 =

  legoev3 with properties:

      FirmwareVersion: 'V1.03E'
           HardwareID: []
            IPAddress: []
    CommunicationType: 'USB'
         BatteryLevel: 100
     ConnectedSensors: {'touch'  'gyro'  'color'  'sonic'}
```

The sensor appears in the list of connected sensors.

Create a connection to a motor. Specify the letter of the EV3 output port that connects to the motor.

```
mymotor = motor(myev3,'A')
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: 0
```

By default, the speed is zero.

Configure the motor to reverse at half speed.

```
mymotor.Speed = -50
```

```
mymotor =

  motor with properties:

    OutputPort: 'A'
         Speed: -50
```

Start the motor.

```
start(mymotor)
```

Read the rotation value, in degrees, of the motor.

```
rotation = readRotation(mymotor)
```

```
ans =

       -1489
```

Reset the rotation value of the motor to zero.

```
resetRotation(mymotor)
```

Input Arguments

`mymotor` — Connection to motor
object handle

Connection to a motor, specified as a string that represents the object created using `motor`.

Example: `mymotor`

Data Types: `char`

See Also

`motor` | `start` | `stop` | `readRotation`

Topics
"Connections to EV3 Hardware"
"Motors"

2

Hardware

Getting Started with MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware

This example shows you how to set up communications with the EV3 brick.

Introduction

The MATLAB® Support Package for LEGO® MINDSTORMS® EV3 Hardware enables you to interact with LEGO MINDSTORMS EV3 hardware from within MATLAB.

This example shows you how to set up communications with the EV3 brick over a USB, Wi-Fi®, or Bluetooth® connection. Then, verify that you can use MATLAB commands to interact with the EV3 hardware by playing a tone from speaker on the EV3.

Prerequisites

If you are new to MATLAB, we recommend completing the Interactive MATLAB Tutorial, “Get Started with MATLAB”, and running the Getting Started with MATLAB example.

Required Hardware

You will need the following hardware:

- EV3 Brick
- EV3 USB Cable, or EV3 WiFi Dongle, or Bluetooth Dongle for Host Computer (optional, if no built-in Bluetooth available on your computer)

Task 1 - Set Up LEGO MINDSTORMS EV3 Communication

Set up communications with the EV3 brick using one of the following options.

Option 1: USB

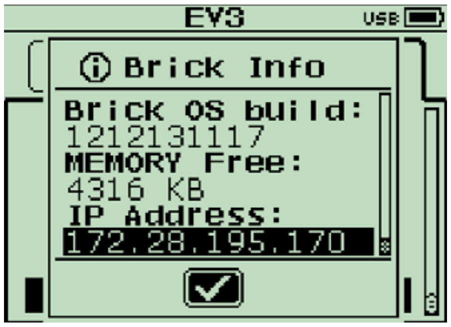
1. Use the USB cable to connect the Mini-USB port on the EV3, labelled 'PC', and USB port on your host computer.

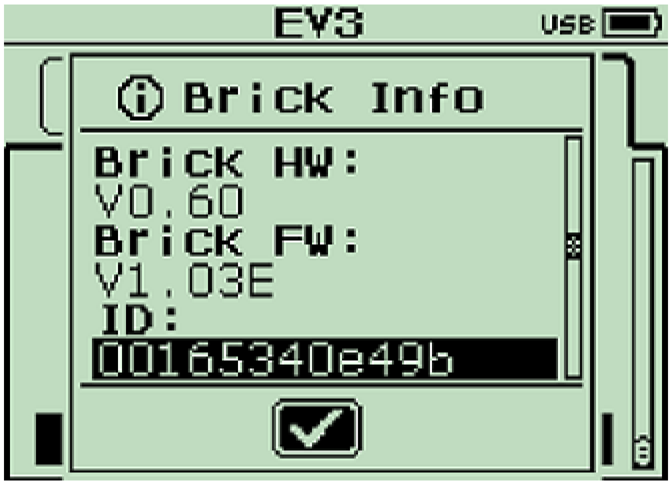
Option 2: WiFi

1. Plug EV3 WiFi Dongle into EV3 Host USB Port, labelled 'USB'.

2. In the EV3 Brick Interface, use **Settings > WiFi** and enable WiFi. Then, search for and connect to a network. For more information, consult the EV3 User Guide.

3. Then, using **Settings > Brick Info**, get the **IP Address** and hardware **ID**. Make a note of these two values for use later on.





4. To verify that the EV3 brick is reachable, use the command line on your host computer to ping the IP address of the EV3 brick. For example, enter:

```
ping 192.28.195.170
```

The ping statistics indicate whether the EV3 brick is reachable from your host computer.

Option 3: Bluetooth

1. Enable Bluetooth on your host computer. If it does not have built-in Bluetooth, use a Bluetooth dongle.

2. In the EV3 Brick Interface, select Settings > Bluetooth and enable Bluetooth.

3. Pair the host computer and EV3 brick. On the host computer, get the number of the serial port for the Bluetooth dongle. Make a note of the name for use later on.

Task 2 - Create a Connection to the EV3 Brick

Create a connection to the EV3 brick called `mylego` using one of the following options.

Option 1: USB

```
mylego = legoev3('usb')
```

Option 2: WiFi

```
mylego = legoev3('wifi',<IP_Address>,<Hardware_ID>)
```

Enter the **IP Address** and hardware **ID** you wrote down during Task 1.

For example:

```
mylego = legoev3('wifi','192.168.1.3','00165340e49b')
```

Option 3: Bluetooth

```
mylego = legoev3('bluetooth',<Serial_Port>)
```

Use the serial port name found in Task 1.

For example:

```
mylego = legoev3('bluetooth','COM3')
```

Option 4: Reconnect Using Settings from the Last Successful Connection

If you use `legoev3` with no arguments, `legoev3` reuses the settings from the last successful connection to an EV3 brick. This is the most effective way to reconnect to a device.

```
mylego = legoev3
```

Task 3 - Beep EV3 Brick

Verify that the connection works. Use the `mylego` connection from Task 2 to play a beep sound from the speaker on the EV3 brick.

```
beep(mylego)
```

Task 4 - Terminate Communication

To terminate the connection, clear the `legoev3` object.

```
clear mylego
```

Summary

This example showed you how to set up communications with the EV3 brick over a USB, WiFi, or Bluetooth connection. It also showed you how to use a MATLAB command to connect to and interact with the EV3 brick. For more information, see “Interact with EV3 Brick Peripherals, Read Sensor Values, and Control Motors” on page 2-5 example.

Interact with EV3 Brick Peripherals, Read Sensor Values, and Control Motors

This example shows you how to interact with the EV3 brick peripherals, read a sensor value, and control a motor.

Introduction

This example shows you how to use MATLAB® command to:

- Interact with the EV3 brick peripherals: display text on the LCD; play a tone on the speaker; read the button status; control the color and state of the status light.
- Read the touch sensor value.
- Control the speed and direction of a motor.

Prerequisites

Create a connection to the EV3 brick called **`mylego`**, as described in "Getting Started with MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware" on page 2-2 example.

Required Hardware

This example requires additional hardware:

- EV3 Touch Sensor
- EV3 Motor

Task 1 - Interact with Brick Peripherals

Use **`mylego`** to interact with the brick peripherals: LCD, speaker, buttons, and status light.

1. Clear the LCD, and then write text on row 2, column 3.

```
clearLCD(mylego)
```

```
writeLCD(mylego,'Hello, LEGO!',2,3)
```

2. Play a 500 Hz tone on the speaker for 3 seconds, with the volume level set to 20.

```
playTone(mylego,500,3,20)
```

3. Read the status of the up button. If the button is pressed, the status is 1. Otherwise, the status is 0.

```
readButton(mylego,'up')
```

4. Illuminate the status light with a red LED, and then turn it off.

```
writeStatusLight(mylego,'red')
```

```
writeStatusLight(mylego,'off')
```

For more information, enter:

```
help legoev3
```

Task 2 - Read a Sensor Value

To interact with sensors that are connected to the input ports on the EV3 brick, create a handle for the sensor. Then, use this handle to perform operations such as reading sensor values.

1. Plug a touch sensor into port #1 on the EV3 brick, and create a handle for it.

```
mytouch = touchSensor(mylego,1)
```

2. Read the value of the touch sensor - pressed (1) and not pressed (0)

```
readTouch(mytouch)
```

For more information, enter:

```
help touchSensor
```

Task 3 - Control the Speed and Direction of a Motor

To interact with motors that are connected to the output ports on the EV3 brick, create a handle for the motor. Use the Speed property to set the speed and direction of the motor. Then, use the handle to start and stop the motor.

1. Plug a motor into port #A on the EV3 brick, and create a handle for it.

```
mymotor = motor(mylego,'A')
```

2. Set the motor speed by assigning a value to the Speed property.

```
mymotor.Speed = 20
```

3. Start the motor.

```
start(mymotor)
```

4. Change the motor speed and reverse its direction.

```
mymotor.Speed = -10
```

5. Stop the motor.

```
stop(mymotor)
```

For more information, enter:

```
help motor
```

Task 4 - Clear Objects

To discard the mylego, mymotor, and mytouch object handles, use the clear function.

```
clear
```

Summary

This example showed you how to:

- Interact with EV3 brick peripherals - LCD, speaker, buttons and status light.

- Read the status of a touch sensor.
- Control the speed and direction of a motor.

Build Collision Alarm Using EV3 Ultrasonic Sensor

This example shows you how to write a MATLAB® script to implement a collision alarm with LEGO® MINDSTORMS® EV3 hardware.

Introduction

The MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware enables you to interact with LEGO MINDSTORMS EV3 hardware from within MATLAB. You can use MATLAB scripts to implement more complex functionality for EV3 hardware.

This example demonstrates a collision alarm implementation with EV3 brick and Ultrasonic sensor. As an object gets closer to the Ultrasonic sensor, the EV3 brick generates a higher-pitched alarm sound.

Prerequisites

Complete “Getting Started with MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware” on page 2-2 and “Interact with EV3 Brick Peripherals, Read Sensor Values, and Control Motors” on page 2-5 examples.

Required Hardware

This example requires additional hardware:

- EV3 Ultrasonic Sensor

Task 1 - Set Up Hardware

1. Follow the instructions in **Getting Started with MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware** example to set up communication between your host computer and EV3 brick.

2. Connect Ultrasonic sensor to an input port of EV3 brick.

Task 2 - Open and Run Collision Alarm MATLAB Script

1. Open the collision alarm script template

```
edit('collision_alarm.m')
```

2. Run script.

Click **Run** button to run the collision alarm script.

Task 3 - Other Things to Try

Reset the detection range by changing

```
RANGE = 0.3
```

the value of RANGE from 0.3 meters to another value, such as 0.5 meters.

Rerun the script to observe the behavior change.

Task 4 - Stop Collision Alarm

Press EV3 UP button to quit while-loop and stop the script, which is implemented as

```
while ~readButton(mylego, 'up')
```

Summary

This example demonstrates using a MATLAB script that to implements a collision alarm. You learned the basic MATLAB script framework to implement more complex functionality for EV3 hardware.

Improve Steering of Two-Wheel Vehicle Using Closed-Loop Control Algorithm

This example shows how to implement a closed-loop control algorithm to make a two-wheel LEGO® MINDSTORMS® EV3 vehicle drive straighter.

Introduction

In a vehicle using independent wheel control, applying the same power to each wheel generally does not result in the vehicle moving straight. This is caused by mechanical and surface differences experienced by each of the wheels. To reduce deviation in the vehicle heading, a better approach is to use a closed-loop controller which adjusts the power applied to two motors based on difference in their rotations. One such controller is a well-known proportional-integral-derivative (PID) controller.

PID control is a basic control loop feedback mechanism. The controller minimizes the difference between the measured and the desired value of a chosen system variable by adjusting the system control inputs.

This example demonstrates the control algorithm implementations, first with no feedback control (Open-Loop Control), and then with feedback control (Closed-Loop Control) based on P controller.

Prerequisites

Complete both “Getting Started with MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware” on page 2-2 and “Interact with EV3 Brick Peripherals, Read Sensor Values, and Control Motors” on page 2-5 examples.

Required Hardware

This example requires extra hardware:

- Two EV3 Large Motors

Task 1 - Set Up Hardware

1. Build a vehicle using two motors to control two independent wheels. Connect one motor to output port **'A'** and the other to output port **'B'**. For example, you can build a vehicle similar to the one described in the printed building instructions in the education core set.

2. Communicating with a moving vehicle is easier with a wireless connection than with a USB cable. Therefore, we recommend setting up WiFi or Bluetooth communications, as described in **Getting Started with MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware** example.

Task 2 - Open and Run Open-Loop Control MATLAB Script

1. Open the open-loop control script template.

```
edit('gostraight_openloop.m')
```

The code sets two motors to same speed and leave them unchanged during execution.

2. Run script.

Click **Run** button to run the open-loop control script. The execution time is 10 seconds defined in

```
EXE_TIME = 10
```

3. Observe deviation with the open-loop system.

The script specifies the same speed for both wheels. However, mechanical and environmental conditions make the wheels to rotate at different speeds, causing the vehicle to deviate from a straight path.

Task 3 - Open and Run Close-Loop Control MATLAB Script

1. Open the close-loop control script template.

```
edit('gostraight_closeloop.m')
```

The code reads the encoders in each wheel, calculates the proportional difference between the rotation speeds of each wheel, and compensates for that difference by adjusting the speed of one motor.

2. Run script.

Click **Run** button to run the close-loop control script. The execution time is 10 seconds defined in

```
EXE_TIME = 10
```

3. Observe the deviation with a closed-loop system.

Observe that, with the closed-loop feedback control system, the vehicle moves straighter than when it was using the open-loop control.

Task 4 - Other Things to Try

1. Change the initial speed setting and adjust the P parameter accordingly

```
SPEED = 20
P = 0.01
```

to make vehicle move straight.

2. Refine the control algorithm with integral and derivative parameters.

Summary

This example demonstrate the implementation of motor control for two-wheel EV3 vehicle. You learned that:

- Open-loop control does not ensure straight driving in a vehicle with independently-powered wheels.
- Closed-loop control uses the difference between two encoder outputs to synchronize the rotation speed of the two wheels.