RenBuggyServo Library Guide

Contents

[1. Introduction 2](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208541)

[2. Features 2](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208542)

[2.1 Constructing a Car 2](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208543)

[2.2 Configuring the Car 3](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208544)

[2.2.1 Servo 3](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208545)

[2.2.2 Motor 4](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208546)

[2.2.3 Encoder 5](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208547)

[2.3 Moving the Car 5](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208548)

[2.4 Changing Car Direction 6](file:///%5C%5Crenishaw.com%5Cglobal%5CGB%5CPLC%5CCommon%5CUserDocs%5CGloucestershire%5CGE%5CIDT%5Cmj138427%5CMy%20Documents%5CRenBuggyServo%20Library%20Documentation.docx#_Toc382208549)

# 1. Introduction

The RenBuggyServo library is intended to be used on the RenBuggy which uses a servo for direction, and a motor to drive the RenBuggy. The servo and motor are controlled using PWM. There is also functionality to allow for the use of an encoder to allow for distances to be specified. The car needs to be configured before it can be driven. The speed of movement can be adjusted.

# 2. Features

## 2.1 Constructing a Car

The car will need a few things when it gets constructed. There are currently 2 constructors.

**Name:** Car()

**Type:** Constructor

**Parameters:**

* PinName servoPin
* PinName motorPin

**Description / Features:**

The car can be driven just using the motor for forward power, and the servo for direction. So, this constructor takes a PinName for the servo, and a PinName for the motor, to act as the pwm output for each. It constructs two new PwmOut objects, one for the servo and one for the motor, passing each their respective PinName.

**Name:** Car()

**Type:** Constructor

**Parameters:**

* PinName servoPin
* PinName motorPin
* int countsPerRevolution
* float wheelCircumference

**Description / Features:**

This constructor does the same as the one above by creating two new PwmOut objects, one for the servo and one for the motor. However, it also accepts two more parameters, to be used in the event where the car is using an encoder to count the stripes of the wheel for measuring distance. The first, an int, called countsPerRevolution, refers to the number of counts required for one full turn of the wheel. The second is a float called wheelCircumference, and refers to the circumference of the wheel that the encoder is reading. This constructor also calls a function to configure the encoder ready for use. This function is described on page 5

## 2.2 Configuring the Car

In order for the car to be driven, it must first have its components configured. There are currently two components that must be configured, the servo and the motor. If the car has an encoder installed, then that will also need configuring. The servo has two functions for configuration, one that accepts data in microseconds and another that accepts it in milliseconds. The motor also has two functions, one accepting microseconds, the other milliseconds. The encoder has a single function for configuration, as it needs some information about the buggy in particular to operate.

### 2.2.1 Servo

The servo uses pulsewidth modulation to operate and so requires some information in order to perform correctly. It needs a pulsewidth, period, range and degrees. The range and degrees parameters refer to the actual output of the movement of the arm.

**Name:** configureServo\_us ()

**Type:** void

**Parameters:**

* int pulsewidth\_us
* int period\_us
* int range
* float degrees

**Description / Features:**

This function takes the pulsewidth and period required for the pwm, in microseconds. It also accepts an int called range, which refers to the range of the pulsewidth to maximum / minimum, from the centre position (e.g. centre position is 1500 microseconds, with + 500 microseconds to turn right, and -500 microseconds to turn left, so range would be 500). The degrees parameter is a float as it is referring to the angle from centre, to maximum left or right turns (e.g. the centre position of the servo is $0^{o}$ and the most it can turn is $\pm 45^{o}$, meaning the degrees is$ 45.0$).

**Name:** configureServo\_ms

**Type:** void

**Parameters:**

* int pulsewidth\_ms
* int period\_ms
* int range
* float degrees

**Description / Features:**

This function takes the pulsewidth and period required for the pwm, in milliseconds. It also accepts an int called range, which refers to the range of the pulsewidth to maximum / minimum, from the centre position (e.g. centre position is 1.5 milliseconds, with + 0.5 milliseconds to turn right, and -0.5 milliseconds to turn left, so range would be 0.5 milliseconds). The degrees parameter is a float as it is referring to the angle from centre, to maximum left or right turns (e.g. the centre position of the servo is $0^{o}$ and the most it can turn is $\pm 45^{o}$, meaning the degrees is$ 45.0$).

### 2.2.2 Motor

The motor requires just a pulsewidth and period in order to operate correctly.

**Name:** configureMotor\_us()

**Type:** void

**Parameters:**

* int pulsewidth\_us
* int period\_us

**Description / Features:**

This function configures the pulsewidth and period for the pwm that is required to run the motor. The format for these is in microseconds.

**Name:** configureMotor\_ms()

**Type:** void

**Parameters:**

* int pulsewidth\_ms
* int period\_ms

**Description / Features:**

This function configures the pulsewidth and the period for the pwm that is requires to run the motor. The format in this instance is in milliseconds.

### 2.2.3 Encoder

**Name:** configureEncoder()

**Type:** void

**Parameters:**

* int countsPerRevolution
* int wheelCircumference

**Description / Features:**

The encoder requires two pieces of information in order for it to be accurate. Firstly, it needs the number of counts it needs to make for one full revolution of the wheel. It also needs to know the circumference of the wheel. These parameters may actually be passed in when you construct a car, as opposed to being manually called later in the application.

## 2.3 Moving the Car

As you probably guessed, the motor will be used to give the car some ‘drive’. There is a function to set the speed. There is an overloaded function, which can accept a distance for the car to move in cm, or just move the car forward, but with no distance specified. Also, the car has a function to make it stop.

**Name:** setSpeed()

**Type:** void

**Parameters:**

* int speed\_us

**Description / Features:**

This function takes in a speed specified by the user and sets the speed the car will move at. It does not move the car forward.

**Name:** forwards()

**Type:** void

**Parameters:**

* float distance

**Description / Features:**

This function takes in a distance in centimetres, and does some calculations using input from the encoder, to make the buggy move the specified distance. If the buggy’s direction is also being altered during the move, it will need to taken in to consideration when specifying a distance.

**Name:** forwards()

**Type:** void

**Parameters:** N/A

**Description / Features:**

This function makes the buggy move, but for no specified distance. Distance can still be calculated by the user, employing the use of wait functions to make the buggy move for specified amounts of time.

**Name:** stop()

**Type:** void

**Parameters:** N/A

**Description / Features:**

This function forces the buggy to stop.

## 2.4 Changing Car Direction

There is only one function for changing the direction the car is facing.

**Name:** setDirection()

**Type:** void

**Parameters:**

* int degrees

**Description / Features:**

This function accepts an angle in degrees. The angle provided should be within the range that the user specified when configuring the servo (see page 3). For example, if the range specified was $45^{o}$, then that would mean that the servo’s full range is $90^{o}$, with $\pm 45^{o}$ being minimum and maximum.