

---

# **noCollide**

**Nico Paller, Manuel Wilke**

**Feb 13, 2021**



**CONTENTS:**

<b>1</b>	<b>NoCollide</b>	<b>3</b>
<b>2</b>	<b>Sensor</b>	<b>5</b>
<b>3</b>	<b>Driver</b>	<b>9</b>
<b>4</b>	<b>Data</b>	<b>11</b>
<b>5</b>	<b>Simulation Integration</b>	<b>13</b>
<b>6</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



The noCollide is a driver warning system to avoid collisions with objects. It can be implemented using LiDAR Sensors and a Raspberry Pi or the CARLA Simulator.



## NOCOLLIDE

```
class lib.nocollide.TtcTimes (too_late: float, brake: float, warning: float)
```

A struct like class to enhance readability when storing Time-to-Collision values. `too_late < brake < warning` must be True

**Parameters**

- **too\_late** (*float*) – The time after which an accident is unavoidable
- **brake** (*float*) – The time after which the driver must be braking to avoid an accident
- **warning** (*float*) – The time after which the driver should be warned

```
class lib.nocollide.NoCollide (driver: lib.driver.Driver, sensors: lib.sensor.SensorGroup,
                               ttc_times: Optional[lib.nocollide.TtcTimes] = None)
```

The class that calculates the acc and regulates the warning.

**Parameters**

- **driver** (*Driver*) – The configuration of the CAN-Bus to initialise
- **sensors** (*SensorGroup*) – The sensor group, on which the calculation should be done
- **ttc\_times** (*Union[TtcTimes, None]*) – The times which define when to warn based on the TTC (Time-to-collision). If None, the default TTC-Times will be used: `TtcTimes(too_late=0.6, brake=1.6, warning=3.0)`

```
calc ()
```

The method that calculates the Time-to-Collision. Takes the Value of the sensor, that measures the closest object and calculates the relative velocity. The TTC results in dividing the distance by the relative velocity :return:

```
run (block: bool = True)
```

The method to start the calculation. Can be run blocking (in an endless loop) or not (1 calculation only)

**Parameters** **block** (*bool*) – whether the method should block or not

```
warn (ttc: float)
```

The method to warn the user based on the Time-to-Collision.

**Parameters** **ttc** (*float*) – the Time-to-Collision





## SENSOR

**class** `lib.sensor.SensorInterface`

An interface to implement a sensor with all the needed methods to function properly

**abstract** `change_addr (new_addr: int)`

A method to change the i2c address to be used with multiple devices on one bus

**Parameters** `new_addr (int)` – the new address that should be set

**abstract** `close ()`

A method to close the sensor/bus

**abstract** `configure (mode: int)`

A method to apply any configuration to the sensor

**Parameters** `mode (int)` – the mode that should be set

**abstract** `measure (rec_bias_corr: bool = True)`

Method to tell the sensor to measure a value

**Parameters** `rec_bias_corr (bool)` – whether to measure with bias correction

**abstract** `read_measurements ()` → float

Method to retrieve the measured data from the sensor

**Returns** the measured data

**Return type** float

**class** `lib.sensor.Sensor (i2c_bus: int = 1, max_range: int = 50)`

The Class that handles a LiDAR Sensor

**Param** `i2c_bus`: the bus number on which the sensor is running, defaults to Bus 1

**change\_addr** (`new_addr: int`)

Method to change the I2C Address of the sensor

**Parameters** `new_addr (int)` – the new address that should be set

**close** ()

Method to close the bus

**configure** (`mode: int = 0`)

Method to initialize the sensor to different modi. Must be done before the sensor can be used

**configuration:** Defaults to 0.

**0:** Default mode, balanced performance

**1:** Short range, high speed. Uses 0x1d maximum acquisition count

- 2: Default range, higher speed short range. Turns on quick termination detection for faster measurements at short range (with decreased accuracy)
- 3: Maximum range. Uses 0xff maximum acquisition count
- 4: High sensitivity detection. Overrides default valid measurement detection algorithm, and uses a threshold value for high sensitivity and noise
- 5: Low sensitivity detection. Overrides default valid measurement detection algorithm, and uses a threshold value for low sensitivity and noise

**Parameters** `mode (int)` – the selected mode

**measure** (`rec_bias_corr: bool = True`)

Method to tell the sensor to take a measurement

**Parameters** `rec_bias_corr (bool)` – Whether the measurement should be taken with or without receiver bias correction; defaults to True

**read\_measurements** () → float

Method to obtain the measured distance in cm

**Returns** the measured value in cm. Returns 0 if timedout

**Return type** int

**class** `lib.sensor.SensorGroup` (`i2c_bus: int`, `sensors: Optional[lib.sensor.SensorInterface] = None`,  
`sensor_names: Optional[List[str]] = None`, `init_mode: int = 0`)

Class to connect to multiple sensors at once.

This class can be used in a context manager (with statement). It returns itself and then all *Sensors* which are being set (default: 3).

**If not used in a with -Statement the bus must be closed manually using the `close()` method**

```
with SensorGroup(i2c_bus=1) as (group, *sensors):  
    ...
```

#### Parameters

- **i2c\_bus** (`int`) – the raspberry pi bus the sensors are running on
- **sensor\_names** (`Union[List[str], None]`) – a list of names, to identify the sensors. Defaults to ["left", "center", "right"] if None
- **init\_mode** (`int`) – the mode the sensors should be initialised with. Defaults to mode 0. See *Sensor.configure()* method

**close** ()

Method do close and delete all sensors from the group. This method is also called when exiting the with-Statement

**set\_mode** (`mode_num: int`, `sensors: Optional[List[str]] = None`)

Method to set the mode for specific or all Sensors in the group

#### Parameters

- **mode\_num** – the mode number, referring to the mode if the *configure()* method of the *Sensor*
- **sensors** – the sensornames of which the mode should be changed

**Raises** **TypeError** – TypeError when the is no such sensor in the group

**start ()**

Method to start the measurements of the sensors. Launches a Thread for each sensor, where it measures continuously in a loop



## DRIVER

**class** `lib.driver.Driver`

An interface to be used by the `py:class: noCollide <.nocollide.NoCollide>` class get and set parameters regarding the vehicle

**abstract** `get_speed()` → *lib.data.Speed*

A method to retrieve the current speed of the car

**Returns** the current speed of the car

**Return type**

`py:class Speed <.data.Speed>`

**abstract** `set_brake(val)`

A method to set the brake amount of the vehicle

**Parameters** `val (float)` – the brake amount

**abstract** `set_throttle(val: float)`

A method to set the throttle of the vehicle

**Parameters** `val (float)` – the throttle amount

**abstract** `warn()`

A method to warn the user for possible collisions

**class** `lib.driver.CanConfig`

Class to better store config details of the can bus if necessary

**class** `lib.driver.CanBus (conf: lib.driver.CanConfig, q_size: int = 1)`

**get\_speed()** → *lib.data.Speed*

Method to retrieve the speed from the queue

**Returns** the latest speed sent via CAN

**Return type** float

**run\_forever()**

Method to keep in touch with the CAN-Bus. Runs in a thread, to avoid blocking

**set\_brake(val)**

A method to set the brake amount of the vehicle

**Parameters** `val (float)` – the brake amount

**set\_throttle(val: float)**

A method to set the throttle of the vehicle

**Parameters** **val** (*float*) – the throttle amount

**warn** ()

A method to warn the user for possible collisions

## DATA

**class** `lib.data.Data` (*value: float, time: float*)

**class** `lib.data.Speed` (*value: float, time: float*)

A class to extend the :py:class: `Data` *<.Data>*. This class stores velocity values and the acceleration can be easily calculated

**acceleration** (*other: lib.data.Speed*) → *lib.data.Data*

A method to calculate the acceleration

**Parameters** **other** – the Speed before with which the Acceleration should be calculated

**Returns** the Acceleration

**Return type**

**py:class** *Data* *<.Data>*

**class** `lib.data.Distance` (*value: float, time: float*)

A class to extend the :py:class: `Data` *<.Data>*. This class stores distance values and the velocity can be easily calculated

**velocity** (*other: lib.data.Distance*) → *lib.data.Speed*

A method to calculate the velocity. Returns Speed with value 1 if the times are the same

**Parameters** **other** – the Distance before with which the speed should be calculated

**Returns** the speed

**Return type**

**py:class** *Speed* *<.Speed>*





## SIMULATION INTEGRATION

**class** lib.sim\_interfaces.**SimNoCollide** (*hud, \*args, \*\*kwargs*)

A class to use the NoCollide in the Carla-Simulator. Simply integrates the HUD of the simulator to warn via the HUD rather than to warn via some kind of Bus

**Parameters**

- **hud** – The HUD of the Simulation
- **args** – Arguments to be passed to the parent class
- **kwargs** – Keyword arguments to be passed to the parent class

**warn** (*ttc: float*)

Overwrites the parent warn Method to warn via the HUD rather than some kind of Bus :param ttc: the Time-to-Collision in seconds :type ttc: float

**class** lib.sim\_interfaces.**SimSensor** (*sensor, max\_range=40*)

A class to implement the Carla-Simulator object detection sensor. Due to the constant pushing nature of the Carla sensor, the last retrieved value and time must be saved to be ready when the NoCollide Algorithm requests the Value.

**Parameters**

- **sensor** – the Carla sensor to use
- **max\_range** – the maximum range of the Carla Sensor

**callback** (*data*)

The callback method that is called, whenever the Carla sensor has measured new data. Simply stores the data, so taht i can be requested any time

**Parameters data** – the data measured by the carla sensor

**change\_addr** (*addr: int*)

A method to change the i2c address to be used with multiple devices on one bus

**Parameters new\_addr** (*int*) – the new address that should be set

**close** ()

A method to close the sensor/bus

**configure** (*mode: int*)

A method to apply any configuration to the sensor

**Parameters mode** (*int*) – the mode that should be set

**destroy** ()

Method to destroy the Sensor to free up memory when the simulation has ended

**listen()**

A Method to tell the Carla sensor to use the `callback()` method whenever a new value is measured

**measure** (*rec\_bias\_corr*: *bool* = *True*)

Method to tell the sensor to measure a value

**Parameters** **rec\_bias\_corr** (*bool*) – whether to measure with bias correction

**read\_measurements** () → *lib.data.Distance*

The method to retrieve the newest Data of the Sensor. If the Carla Sensor hasn't measured anything yet, a *Distance* object will be returned with the maximum range. :return: the newest Distance measured by the Carla sensor :rtype: *Distance*

**stop()**

A method that is needed when stopping the simulation.

```
class lib.sim_interfaces.SimSensorGroup (i2c_bus:          int,          sensors:          Optional[lib.sensor.SensorInterface] = None,  
                                         sensor_names:    Optional[List[str]] = None,  
                                         init_mode: int = 0)
```

Class to overwrite the *SensorGroup* class to use the simulated sensor when retrieving the distance

**get\_closest** () → *lib.data.Distance*

Class to overwrite the `get_closest()` method to use the simulated sensor when retrieving the distance  
:return:

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## A

`acceleration()` (*lib.data.Speed* method), 11

## C

`calc()` (*lib.nocollide.NoCollide* method), 3

`callback()` (*lib.sim\_interfaces.SimSensor* method), 13

`CanBus` (class in *lib.driver*), 9

`CanConfig` (class in *lib.driver*), 9

`change_addr()` (*lib.sensor.Sensor* method), 5

`change_addr()` (*lib.sensor.SensorInterface* method), 5

`change_addr()` (*lib.sim\_interfaces.SimSensor* method), 13

`close()` (*lib.sensor.Sensor* method), 5

`close()` (*lib.sensor.SensorGroup* method), 6

`close()` (*lib.sensor.SensorInterface* method), 5

`close()` (*lib.sim\_interfaces.SimSensor* method), 13

`configure()` (*lib.sensor.Sensor* method), 5

`configure()` (*lib.sensor.SensorInterface* method), 5

`configure()` (*lib.sim\_interfaces.SimSensor* method), 13

## D

`Data`, 10

`Data` (class in *lib.data*), 11

`destroy()` (*lib.sim\_interfaces.SimSensor* method), 13

`Distance` (class in *lib.data*), 11

`Driver`, 7

`Driver` (class in *lib.driver*), 9

## G

`get_closest()` (*lib.sim\_interfaces.SimSensorGroup* method), 14

`get_speed()` (*lib.driver.CanBus* method), 9

`get_speed()` (*lib.driver.Driver* method), 9

## L

`listen()` (*lib.sim\_interfaces.SimSensor* method), 13

## M

`measure()` (*lib.sensor.Sensor* method), 6

`measure()` (*lib.sensor.SensorInterface* method), 5

`measure()` (*lib.sim\_interfaces.SimSensor* method), 14

## N

`NoCollide`, 1

`NoCollide` (class in *lib.nocollide*), 3

## R

`read_measurements()` (*lib.sensor.Sensor* method), 6

`read_measurements()` (*lib.sensor.SensorInterface* method), 5

`read_measurements()` (*lib.sim\_interfaces.SimSensor* method), 14

`run()` (*lib.nocollide.NoCollide* method), 3

`run_forever()` (*lib.driver.CanBus* method), 9

## S

`Sensor`, 3

`Sensor` (class in *lib.sensor*), 5

`SensorGroup` (class in *lib.sensor*), 6

`SensorInterface` (class in *lib.sensor*), 5

`set_brake()` (*lib.driver.CanBus* method), 9

`set_brake()` (*lib.driver.Driver* method), 9

`set_mode()` (*lib.sensor.SensorGroup* method), 6

`set_throttle()` (*lib.driver.CanBus* method), 9

`set_throttle()` (*lib.driver.Driver* method), 9

`SimNoCollide` (class in *lib.sim\_interfaces*), 13

`SimSensor` (class in *lib.sim\_interfaces*), 13

`SimSensorGroup` (class in *lib.sim\_interfaces*), 14

`Simulation Integration`, 11

`Speed` (class in *lib.data*), 11

`start()` (*lib.sensor.SensorGroup* method), 6

`stop()` (*lib.sim\_interfaces.SimSensor* method), 14

## T

`TtcTimes` (class in *lib.nocollide*), 3

## V

`velocity()` (*lib.data.Distance* method), 11

## W

`warn()` (*lib.driver.CanBus method*), 10

`warn()` (*lib.driver.Driver method*), 9

`warn()` (*lib.nocollide.NoCollide method*), 3

`warn()` (*lib.sim\_interfaces.SimNoCollide method*), 13