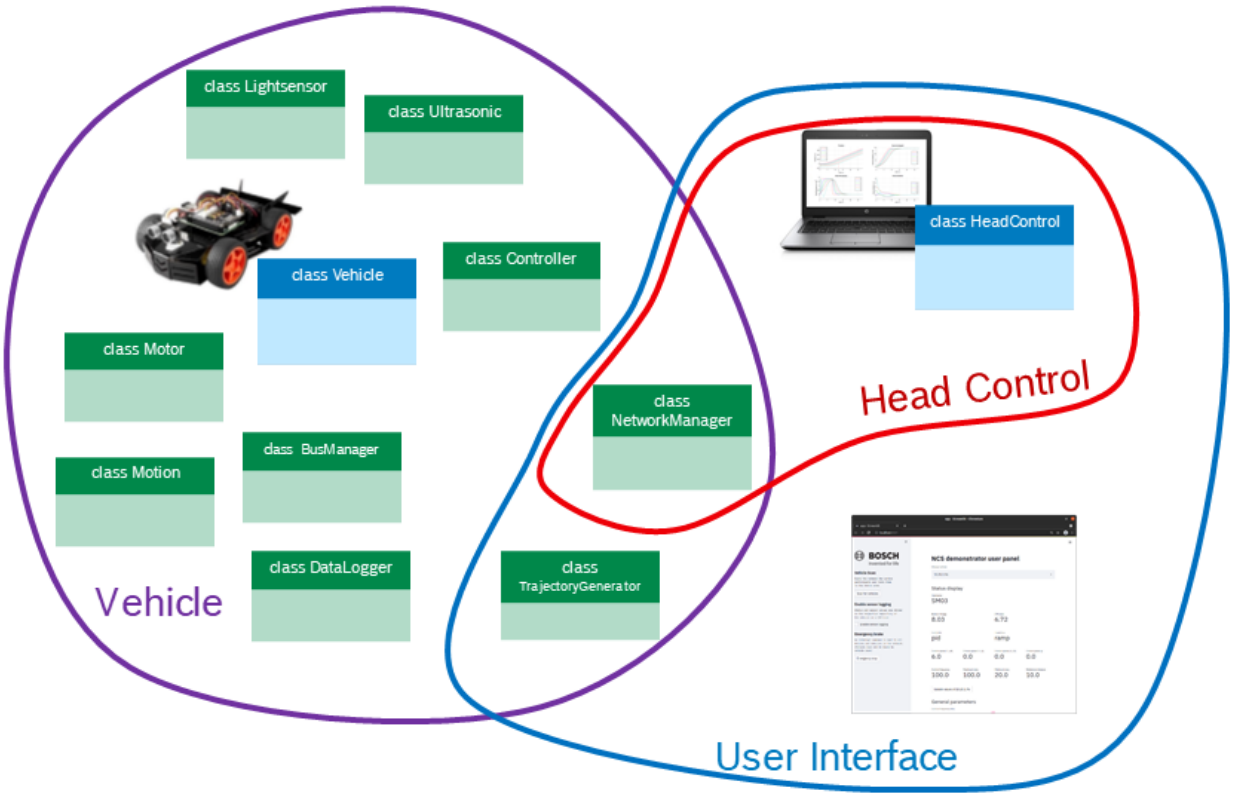
2 Software architecture

The following figure shows the overall class-oriented software architecture. Most of the functions are encapsulated in own classes. The figure shows which scripts instantiate which functions.

For example: The Vehicle object generated from the vehicle class generates a controller, ultrasonic, lightsensor, bus manager, network manager, trajectory generator, data logger, motion and motor object from the respective class. As visible in the figure, the head control shares the same class as the vehicle class. Therefore, the network manager class combines functions necessary for the head control as well as the vehicle class.

Another more class-oriented option would be to generate a vehicle and a head control specific network manager which shares a parent class network manager. Due to lack of time this was not implemented. Also it should be noted, that this increases the complexity of the architecture, which could result in longer training period for inexperienced developers.

The user interface itself does not have its own class. The main reason this is a pure sequential script call, is the structure of the framework streamlit which demands session-states and permanently running the script from top to down for updating changes in the UI.



User interface instantiates trajectory generator for fetching the available trajectory list. Updates regarding the controller list must be changed manually due to the dependence of the controller class on the FileDB class which does not work on the Linux computer.

3 Vehicle Parameters

Vehicle Parameter	Default value	Adaptable by UI?	Adaptable during run?	Description
hostname	*vehicle specific - manually assigned at configuration of Raspberry OS*	no		hostname specifies the accessable network name of a device and is used for communication. The hostname can also be used to identify the vehicles, since all vehicles have their hostname written on it.
vehicle_position	*vehicle specific - manually assigned when vehicle-script is called*	no		vehicle_position specifies the position of the vehicle in the convoy assuming the demonstrator is built up for a startup scenario. The first vehicle is the vehicle standing in front, having no vehicle in front of it. The last vehicle is the vehicle standing at the end of the convoy, having no vehicle behind of it.
reference_distance	10.0	yes	yes	reference_distance specifies the distance the vehicles want to keep from each other. The distance is specified from the end of the ultrasonic sensor until the plastic adapter fixed at the rear of the vehicle.
controller_type	'pid'	yes	yes	controller_type specifies the type of used controller. Each type is defined by a control having different properties, using a different set of parameters. Within a controller type control parameters can change, however the set of parameters is unchanged.
control_frequency	100.0	yes	no	control_frequency specifies the number of runs in the while loop per second. The system tries to finish all tasks before new run is supposed to start and sleeps until new run.

minimum_duty	20.0	yes	no	minimum_duty specifies the lower limit of available regulating control value. The motors will not generate signals with a lower value than minimum duty.
maximum_duty	100.0	yes	no	maximum_duty specifies the upper limit of available regulating control values. Analog to minimum_duty.
control_parameters	*vehicle specific - default value is fetched from the control_param-config-file and padded to the number specified by the variable n_control_param_pad*	yes	yes	control_parameters specify the parameter used for closed loop control, e.g. P-share, I-share and D-share with PID-control.
trajectory	'ramp'	yes	yes	trajectory specifies the generated trajectory profile for the actuation of the car. The ramp profile is time-depended changing the actuation of the vehicle.
battery_value	*vehicle specific - depends on battery charge of vehicle*	no		current battery charge. For further information check the data sheet of the batteries in the hardware section
cpu_value	*vehicle specific - depends on current active processes*	no		current CPU load. CPU load is generated by all running scripts and also by processes run by the general purpose operating system.
enable_communication	True	no		enable_communication can be used for debugging if no communication between vehicles regarding sensor values is desired.
parameter_logging	True	yes	no	if parameter_logging is turned on, after each trajectory or control run a CSV-file is generated and locally stored on the vehicle.
verbose	True	no		if verbose is turned on, the vehicle generates prompt messages in the terminal specifying changes to vehicle

				parameters and current actions of the vehicle.
follow_line	False	no		If follow_line is turned on, the vehicle uses the greyscale sensor to follow a black line. The parameter is set for every vehicle in its config file
use_simulator	False	yes	no	If use_simulator is turned on, the vehicle simulates its movement instead of moving via the motors/actuators. The threads of the sensors and actuators are continued with a low clock rate to reduce computing time
use_centralUnit	False	no		Only useful to use when use_simulator is turned on. If you switch use_centralUnit on, the vehicle will send all of its data to the vehicle with the vehicle_position "0". Make sure this vehicle exists

4 Runtime analysis (< October 2021)

On embedded devices most of the tasks are handled sequentially. One way to enable simultaneous task handling is working with Threads. Process is encapsulated in this threads and can have different states like running, stopped, waiting, inactive and so on. The process management of the operating system dynamically assigns computational resources to each thread. For instance, if one thread is inactive (for example when the program run into time.sleep()), another thread is activated. Usually each thread comes with a own permanent loop and some sleeping time preventing one thread to block all other processes.

In the following all threads are listed on one vehicle, when all implemented components are active:

Thread	Sleeping Time [s]	Task
Main Thread	0.00020	The main thread runs the infinite loop checking and collecting commands from the command storage provided by the network manager thread. If a new command is available it will call the specific functions to carry out the task.
Motor Left-Front	0.00050	Permanently updates the current duty cycle according to current attribute *power* of the motor object. The runtime analysis showed that assigning a thread to each motor is beneficial for the overall runtime.
Motor Left-Rear	0.00050	*(see above)*
Motor Right-Front	0.00050	*(see above)*
Motor Right-Rear	0.00050	*(see above)*
Sensor Ultrasonic	0.00012	Permanently measures distance and updates the attribute distance of the Ultrasonic object. By calling the object the distance attribute is returned.
Network Manager	-	Permanently listens to the bind server address and checks for new messages. By default it listens to all available interfaces at the specified port. If a new message is sent, the network manager processes it and stores it in the corresponding storage or updates current flags. Network Manager runs endless loop without blocking the system. Reason is unclear. I assume that socket and the thread interact with each other.
Status Inspector	0.00100	Permanently measures and updates the attributes CPU load and battery voltage of the status inspector object.

Note: Speed sensors are implemented with interrupts triggered by falling and rising flanks of the sensor signals. Therefore, the fetching of speed values are not threaded.