Hochschule Konstanz
Fakultät Elektrotechnik
und Informationstechnik

System Architecture Vehicle-Emulator

By Mouaad Gssair, Yannick Pauly, Jonas Maier

02.07.2019

Inhalt

1. Installation

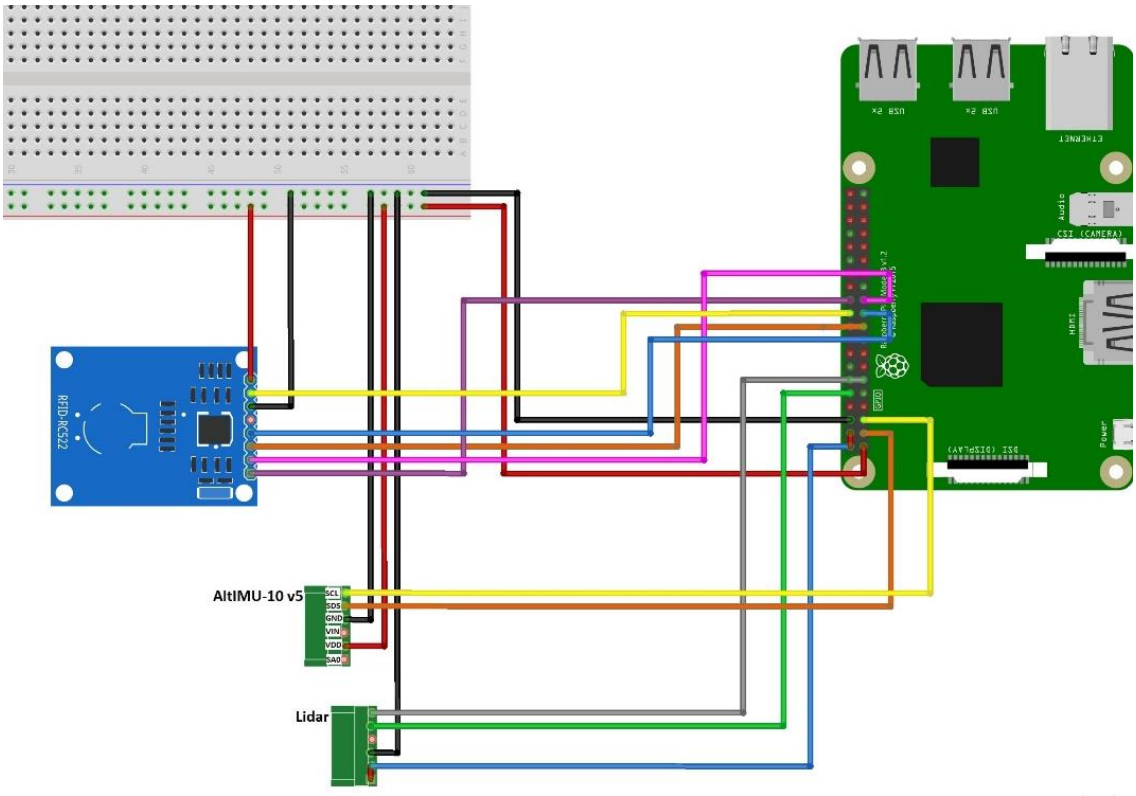
1.1 Hardware

We use five different sensors for measurement in our project.

These Sensors are:

- LIDAR Sensor (xv11)
- RFID RC522
- LSM6DS33 (accelerometer and gyroscope)
- LIS3MDL (magnetometer)
- LPS25H (barometer)

To connect the sensors to the raspberry pi you can do that as shown in this schematic.



1.2 Software

To install all the software to run the system you need Raspbian on the Raspberry Pi and make sure you have internet.

After checking that you can install the software by writing these commands in the command line:

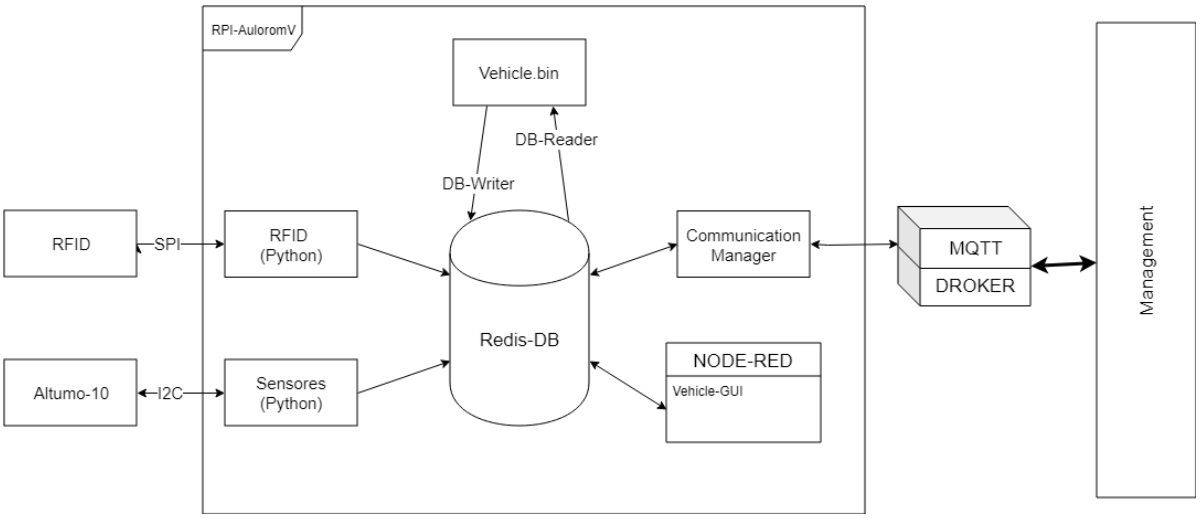
1. `git clone https://github.com/DerZwergGimli/SysArchitecturProject.git`
2. `cd SysArchitecturProject/InstallationScripts`
3. `chmod +x install.sh`
4. `./install.sh`

This will take a fiew minutes to install. After the installation is finished your system should start itself.

2. System Parts

2.1 Overview

In this diagram you can see the system structure.

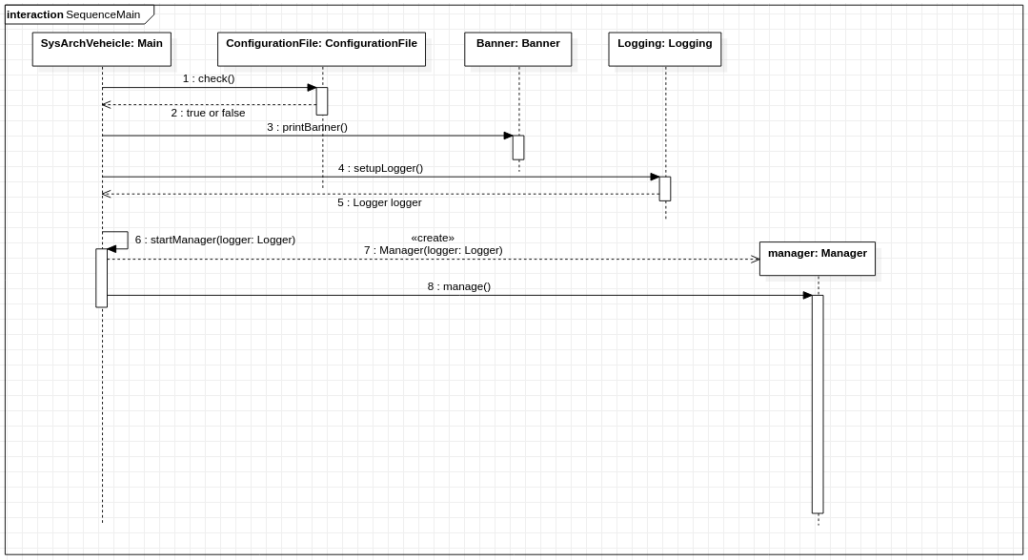


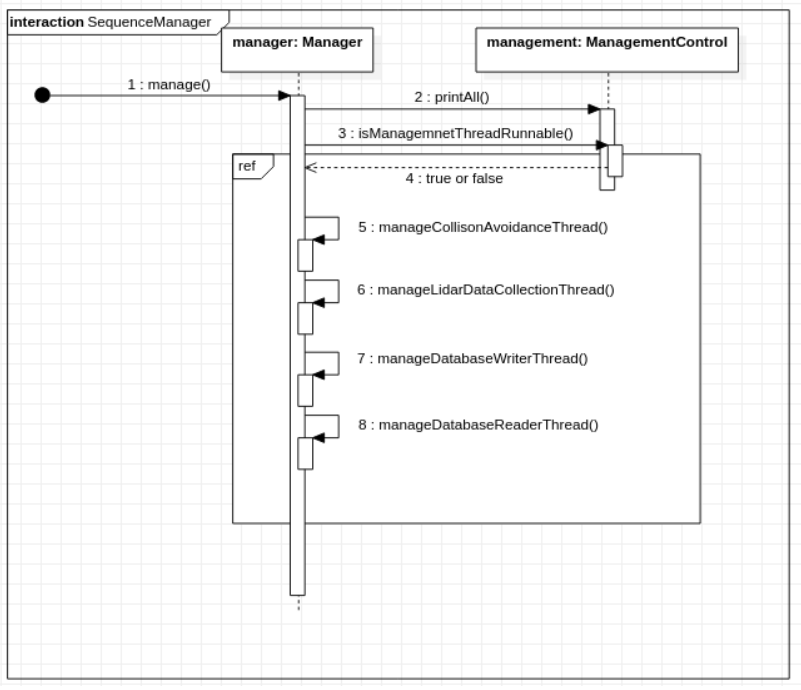
2.2 Vehicle

The vehicle application will emulate the vehicle on a RaspberryPI. The whole application is written in Java with JamicaVM to achieve real-time behavior.
The realtime-vehicle will read in the LIDAR sensor and will perform some collision avoidance. This data should be sent to a motor and steering controller to perform some actions (but this feature is not implemented right now). The collision avoidance now just checks the sections for obstacles that should be avoided.

The whole code is documented in JavaDoc take a look at "Vehicle_Project/doc" folder when you have downloaded our repository.

To get a brief overview of the Realtime-Part of the application there are two sequence diagrams.





The emulated vehicle is realized with Java and compiled with "aicas JamaicaVM" to gain real-time behavior.

Note that: you will have to run this application as superuser/root to create threads that have a higher priority then normal threads.

2.3 RT-System main components

To get a brief overview the main parts of the application are:

1. **Manager:** This will start/stop/manage all threads and mange them. It checks the thread states to make sure all that should run are running.
2. **TLidarDataCollection:** This will collect the data from a LIDAR sensor and passes the data to a queue to analyze the data. Attached to this is a Overrun- and Miss-Hander. This thread will run periodically as a Realtime-Thread.
3. **TCollisionAvoidance:** This will read the data from a queue and performs some checks for obstacles and sends the data further to another queue. Attached to this is a Overrun- and Miss-Hander. This thread will run periodically as a Realtime-Thread.
4. **TDatabaseWriter:** This thread will collect the data and sends it to a database (in our case REDIS). This thread will run periodically as a Realtime-Thread.
5. **TDatabaseReader:** This thread will read from a database. This thread will run periodically as a Realtime-Thread. This thread is right now deactivated because there was at this time no use case!
6. **osInterfaces:** Those classes will provide access to the operating system to read system values like: CPU_temperature or network traffic.
7. **gpioInterface.lidar:** Those classes will provide access to the LIDAR sensor to read the values generated by the sensor.
8. **redisInterface:** This interface is for communicating with a REDIS database.
9. **threads.hander:** Those classes will be called if threads like TLidarDataCollection or TCollisionAvoidance are consuming to much time.
10. **threads.interruptible:** In here is the logic for all threads to make it easier to interrupt them all classes are implementing "Interruptible".

More information you can find in the Java-Doc of this project.

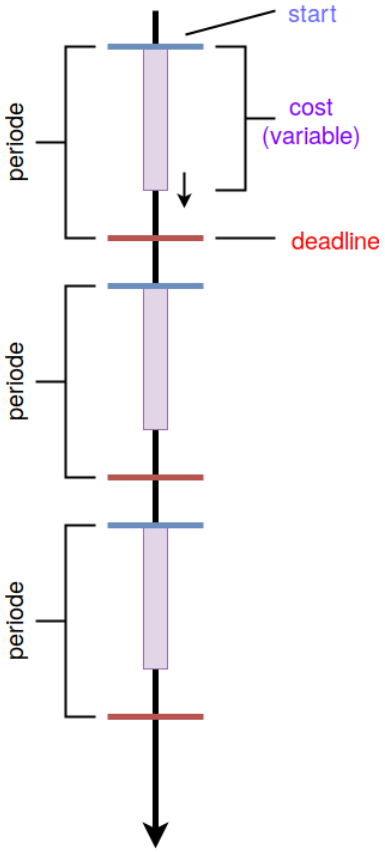
2.4 Overrun- and Miss-Handler

You can set the timing for those to handers for the classes

- LidarDataCollection
- CollisionAvoidance

There are three entries you can make:

1. periode: will define the time in ms when or how often the task will be performed (periodically)
2. cost: will define the time in ms how long the task should take but a cost overrun will only be logged as waring and the time will be increased by one ms.
3. deadline: will define the time in ms how long the task should use cpu time. If the task takes longer than this amount of time it will be interrupted and the thread will be restarted. (Note that this can create a dead-loop).

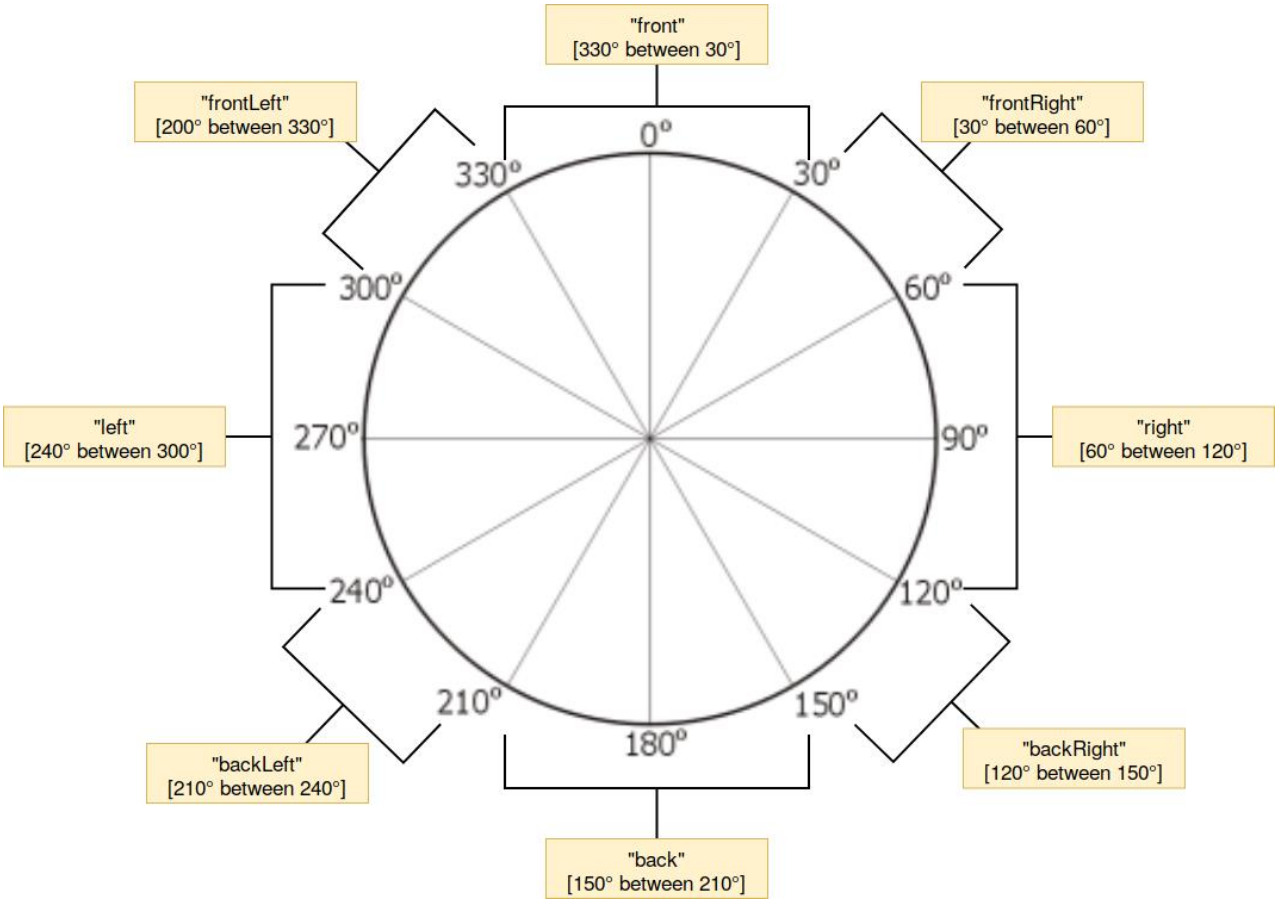


2.5 Collision Avoidance

A feature of our vehicle application is the collision avoidance by simple checking the distances with the allowed minimal distances defined in the configuration file.

It will generate 3 Types of stages the numbers in the brackets may vary:

- objectDetected(0.9)
- ok(1.1)
- error(0.0)



What the collision avoidance will do:

1. Read configuration settings
2. Take a section (for example frontRight).
3. Take relevant LIDAR data
4. Check distances and count distances that are lager then the distance in the configuration file and count them to "OK" and count distances that are smaller than the distance in the configuration file and count them to "NOK".
5. Calculate the difference from "OK"/"NOK" = weighting (of this section).
6. Check if the weighting with the configuration.
7. Define state of this section:
   i. ok: weighting.section > weighting.configuration
   ii. objectDetected: weighting.section < weighting.configuration
   iii. error: weighting.section = 0 or less than zero or any other error
8. Write final state with weighting to the section: objectDetected(0.9)
9. Continue on with the other sections (step 2)
10. If all sections are checked. Send data to Queue and start again (at step 1).

2.6 Logging

This application provides logging abilities the configurations for this you can change in the "config_vehicle.properties" file which is generated when you launch the application for the fist time.

2.7 Configuration-File

The application will generate a configuration file named: "config_vehicle.properties".
To change the behavior of the application you can edit this configuration file.

```
#This is the configuration file for the Vehicle-Emulator

#Added a small delay for the banner (s)
banner.delay=0
console.clearScreen=false #This will clear the screan like if you call "clear" in a linux terminal window!
console.showStatusInConsole=false #This will print out the thread states

#Should threads be started when the application starts (true/false)
#Note that if you kill the manager the application will completely! You must have to execute the application on your platform again, by calling its binary file.
managementControl.managementThreadRunnable=true #This is the main appiclation should always be true
managementControl.collisionAvoidanceThreadRunnable=true #To enable the collion avoidance
managementControl.lidarDataCollection=true #To enable the data collection form a lidar sensor
managementControl.databaseReaderThreadRunnable=false #To read the data to a database
managementControl.databaseWriterThreadRunnable=true #To writre the collected data to the database
managementControl.exposeControlToDatabase=true #To controll the applucation from the databse (start/stop)

#CollisionAvoidance
collisonAvoidance.Front_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.FrontLeft_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.FrontRight_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.Left_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.Right_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.Back_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.BackLeft_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.BackRight_MinDistance=500 #Distance in mm for the obstecte avoidance / detection
collisonAvoidance.weighting=1.0 #difference of OK distances and NOK distances
collisonAvoidance.periode=250 # periode of the collsion avoidance thread
collisonAvoidance.cost=100 # cost of the collsion avoidance thread - not critical
collisonAvoidance.deadline=200 # deadline of the collsion avoidance thread - critila will restart the thread

#Redis-Config
redis.enable=true # To enable/disable the redis connection to the databse
redis.url=localhost # The url of the redis server
redis.port=32769 # Port of the redis server
redis.expireTime=1000 # Time in secounds when the redis values in the daatabse will be deleted
```

```
#Lidar-Config
lidar.enabled=true # To disable the whole lidar sensor if false then are random values generated (for tests)
lidar.initCommand=gpio mode 1 pwm # The command to initialize a gpio pin of a rpi for the lidar sensor
lidar.startRotationCommand=gpio pwm 1 250 # The command to set the rotation speed of the lidar motor
lidar.stopRotationCommand=gpio pwm 1 0 # To stop the rotation of the motor
lidar.scanCommand=/home/pi/vehicle/lidar/xv11test /dev/ttyAMA0 # The command to read the lidar values
lidar.periode=300 # periode of the lidar collection thread
lidar.cost=250 # cost of the lidar collection thread - not critical
lidar.deadline=275 # periode of the lidar collection thread - critila will restart the thread

#Logger-Config
logger.fileName=vehicleLog.log # the name and location of the log file.
logger.maxFileSize=100000 # the maximum file size of one logger file.
logger.maxNumberOfFiles=3 # the maximum file count
logger.appendFiles=true # if the files should be appended or not
logger.debugLevel=ALL # the debug level (used: java.util.logging.Logger)

#NetworkInterface
networkInterface.enabled=true # to controll if the network interface should be read or not
networkInterface.name=wlp2s0 # the name if the network interface that should be read

#Top-Interface
topInterface.enabled=true # to controll if system infos form TOP should be read

#Sensors-Interface
sensorsInterface.enabled=true # to controll if system infos like cpu_temperature should be re
```

2.8 Communication Management

The communication management is made by a separate Java Application running on the JVM with the normal JRE. The following are the important components of the communication application:

- The Manager Class

The Manager (Main Class) will read the configuration from the ***config_ComManager.properties*** file and initialize the properties of the system such as the MQTT Broker URL and port, Redis Host and port, the time interval of the Data Persistency etc. The instances will be initiated with the appropriate configuration and the references to the other instances accordingly, and then start the Threads.

- Database Controller

It is worth to mention that the Redis controller is implemented in the *DBController* Class as a Singleton Pattern, so every instance of the other classes will have the same reference to it, and its method are declared as synchronized to guaranty the Thread-safety.

- Threads

In the communication management application, there two thread running other then the main thread. The first is the Management Thread, where the state machine is running in an infinite loop. The second thread is the Data Persistency thread. It runs an infinite loop with an sleep interval depending on the stat of the Management Thread, in other word, the interval changes depending on whether the driver is present or not.

- Logger

The logger will also be initialized by the Main class, depending on the config properties, and its reference will be passed to the other object.

The logfile containing all the logs of the communication application can be accessed with the following path: /home/pi/vehicle/commManager/ComLog.log.0.

- The State machine

The state machine has basicly 4 state. Where the default state is NO_DRIVER.

- **NO_DRIVER**: In this state, the Management thread will check if the driver is present regularly by checking a key in the DB. The thread will loop infinitely until this is true, then it will switch to the next state, *AUTHENTIFICATION*.
- **AUTHENTIFICATION**: In this case, the Management Thread will send an authentication request to the management system and waits for a response. This response will be handled in the mqttCallback Class, which will update the driver and notify the management thread to switch to the *IS_LOGGED_IN* state if the authentication was successful. Otherwise, it will try again 4 more times. If the counter for the tries reaches 5, the tread will switch back to the *NO_DRIVER* state.
- **IS_LOGGED_IN:** In this state, the thread checks if the driver is still present regularly. If not, it will switch to the *LOGOUT* state.
- **LOGOUT:** In this state, the driver instance in the thread will be set to *null*, and a logout request will be sent to the management system and then switch bac to the *NO_DRIVER* state.

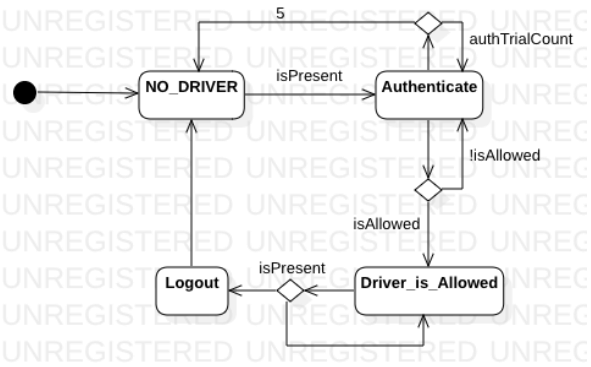


Figure 1 : Communication Management Statemachine

The application is embedded in a jar file and can be found in Git. to starting the application, the correspondent service `communicationManager.service` can be started from the terminal or using the graphical user interface.

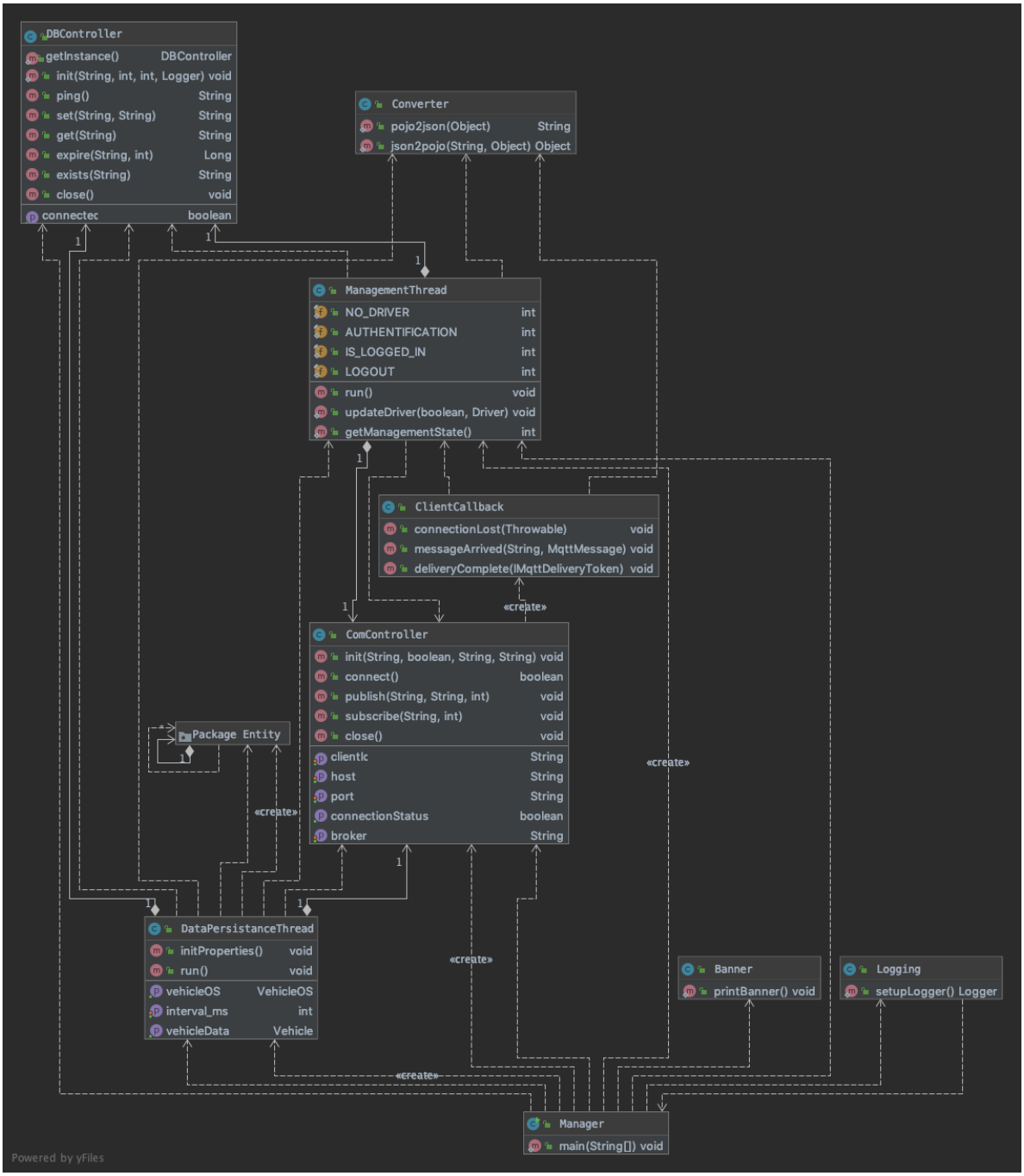


Figure 2 : CommunicationManager Class Diagram

2.9 UI Node-Red

We realized our management UI via Node-Red. The flows you can find here.
Just to get an overview here are some images. You can learn more about node-red here.

2.10 RFID-RF522

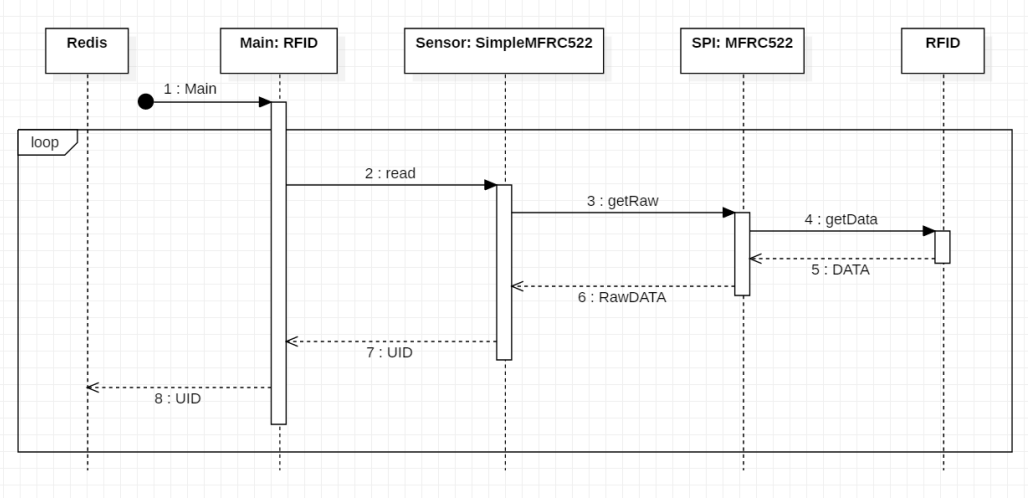
Used Library:

- https://github.com/lthiery/SPI-Py.git
- https://github.com/pimylifeup/MFRC522-python.git

Activate SPI
We use SPI to communicate with the sensor. To enable SPI go to the raspberry configuration tool with `sudo raspi-config`, then to `Interfacing Options` and then enable SPI.
In this case we use a library for the SPI communication. To install this library execute the following steps:

1. `git clone https://github.com/lthiery/SPI-Py.git`
2. `cd SPI-Py`
3. `sudo python setup.py install`

Python
In this sequence diagram is a short overview of the functionality of the python scripts.



To start the sensor separately you can use the virtual environments we built.
1. Activate the environment:
`source ~/vehicle/Sensors/RFID_Sensor/Scripts/activate`

2. Start the python script
`cd ~/vehicle/Sensors/RFID_Sensor/MFRC522-python`

`python RFID.py`

2.11 AltIMU-10 v5

Used library:

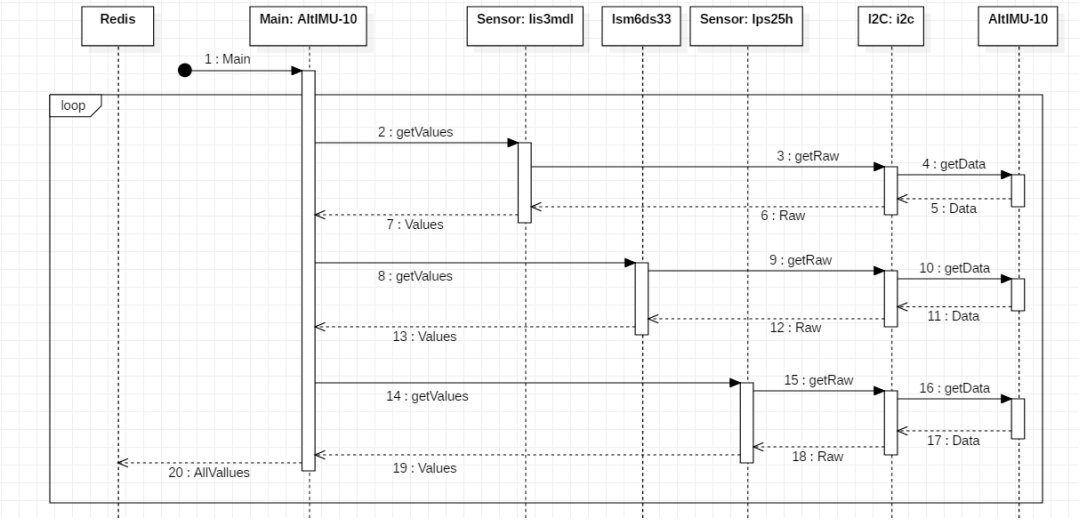
- https://gitlab.com/acrandal/python-AltIMU-10v5

This script uses the sensors:

- LSM6DS33 accelerometer and gyro and temperature
- LIS3MDL magnetometer
- LPS25H barometric

Activate I2C
We use I2C to communicate with the sensors. To enable I2C go to the raspberry configuration tool with `sudo raspi-config`, then to `Interfacing Options` and then enable I2C. You also can check your setup by running `i2cdetect -y N`. The N is the number of I2C-bus you want to use, usually it is 0 or 1. Then t should output a table with a view hex numbers in it.

Python
In this sequence diagram is a short overview of the functionality of the python scripts.
The python script includes 5 different scripts as you can see here.



To start the sensors separately you can use the virtual environments we built.
1. Activate the environment:
`source ~/vehicle/Sensors/AltIMU-10/Scripts/activate`

2. Start the python script
`cd ~/vehicle/Sensors/AltIMU-10/python-AltIMU-10v5`

`python AltIMU-10.py`

2.12 Lidar (XV11)

Used library

- https://github.com/bmegli/xv11lidar-test

Activate UART
For the communication with the Lidar sensor we use UART. To activate UART you need to change 2 files. Open the file config.text with `sudo nano /boot/config.txt`. At the end of the file add the following line `enable_uart=1` and save the file. The next step you have to open the file cmdline.txt with `sudo nano /boot/cmdline.txt`. If in this file there is the entry `console=serial0,115200` you have to remove it and save the change:

Test
You can test your connection and the sensor by following the instructions in this PDF:
file:///C:/Users/Jonas/Downloads/PAE7_Automat_LocherOechslein_nomat%20(3).pdf

Tipp: When you start rotation of the Lidar use a value between 200 and 300 instead of 500.

2.13 Redis Database Topics

To install Redis separately only use `sudo apt install redis-server`.

```
+-- management
|   +-- threads
|       +-- managementRunnable                    (true/false)
|       +-- collisonAvoidanceRunnable             (true/false)
|       +-- databaseReaderRunnable                (true/false)
|       +-- databaseWriterRunnable                (true/false)
|       +-- lidarDataCollectionThreadRunnable     (true/false)
+-- sensors
|   +-- lidar
|       +-- angles               (all values are seperated by a semicolon)
|       +-- distances            (all values are seperated by a semicolon)
|       +-- timestamp           {sample:20190624T005155Z} CET Time
|       +-- timing
|           +-- startTimeNano
|           +-- endTimeNano
|           +-- diffTimeNano
|           +-- timestamp
|   +-- collsionAvoidance
|       +-- status
|           +-- front            {sample:objectDetected(0.0)}
|                                objectDetected/ok/error(distancesOK/distancesNOK)
|           +-- frontLeft
|           +-- frontRight
|           +-- left
|           +-- right
|           +-- backLeft
|           +-- backRight
|           +-- timestamp
|   +-- rfid
|       +-- ID
|       +-- present
|       +-- timestamp
|   +-- gyro
|       +-- gyro_x
|       +-- gyro_y
|       +-- gyro_z
|       +-- unit
|       +-- timestamp
|   +-- accelerometer
|       +-- accelerometer_x
|       +-- accelerometer_y
|       +-- accelerometer_z
|       +-- unit
|       +-- timestamp
```

```
|   +-- magnetometer
|       +-- magnetometer_x
|       +-- magnetometer_y
|       +-- magnetometer_z
|       +-- unit
|       +-- timestamp
|   +-- barometer
|       +-- pressure
|           +-- data
|           +-- unit
|           +-- timestamp
|       +-- metersinheight
|           +-- data
|           +-- unit
|           +-- timestamp
|   +-- temperature
|       +-- data
|       +-- unit
|       +-- timestamp
|   +-- os
|       +-- network
|           +-- eth0
|               +-- rx_bytes
|               +-- rx_dropped
|               +-- rx_errors
|               +-- rx_mcast
|               +-- rx_overrun
|               +-- rx_packages
|               +-- tx_bytes
|               +-- tx_carrier
|               +-- tx_collsns
|               +-- tx_droppped
|               +-- tx_errors
|               +-- tx_packages
|               +-- timestamp
|      +-- temperatur
|          +-- cpu0
|          +-- unit
|          +-- timestamp
|
```

```
|      +-- top
|           +-- systemTime
|           +-- uptime
|           +-- uptimeUnit
|           +-- users_active
|           +-- loadAverage_1min
|           +-- loadAverage_5min
|           +-- loadAverage_15min
|           +-- cpu_user
|           +-- cpu_system
|           +-- cpu_nice
|           +-- cpu_idle
|           +-- cpu_wait
|           +-- cpu_hardwareInterrupts
|           +-- cpu_softwareInterrupts
|           +-- cpu_stolenTimeByHypervisor
|           +-- memory_total
|           +-- memory_free
|           +-- memory_used
|           +-- memory_bufferCache
|           +-- memory_unit
|           +-- swap_total
|           +-- swap_free
|           +-- swap_used
|           +-- swap_bufferCache
|           +-- swap_unit
|           +-- timestamp
```