Chapter 1

Test Results

1.1 Software Tests[IV]

Google Test and Google Mock were used to perform unit testing for the serial module to ensure packet parsing is correct in the serial buffer and that the serial handler behaves as expected. At first, CxxTest was considered and some unit tests were implemented in this framework. However, it was decided to switch to the Google testing framework for its simplicity and ease of use. The CxxTest environment did not provide a direct support to mock class object, unless the mock objects are standard library functions. One way to solve this problem would have been to define the mocked class functions in a separate source file and link it at compile time. Meaning that we would have one .cpp file for the 'real' and one for the 'mock' functionality. This would have resulted in more effort and time spent than estimated with Google Test and Google Mock.

SerialBufferTest.cpp
The tests focus mostly in the parsing of correct packets when data is received from the usb connection. The appendReceiveBuffer() function in the serial buffer class is tested with various fault packets, e.g. wrong delimiters, wrong checksum, etc. Along with those, some tests check the correct behaviour of the checksum() function.

SerialHandlerTest.cpp
The tests are designed to verify the expected behaviour of the readOp() and writeOp() functions of the serial handler class. For this purpose, we mock the serial io interface to simulate correct or false output when performing read and write from the serial connection. In addition, test were included for the functions trying to connect and reconnect with the serial IO.

1.2 Integration Tests[JK]

Our integration tests were mostly done manually by using the Odroid mounted on the car itself. For some parts, like the the LaneFollower interworking with the DecisionMaker, the OpenDaVINCI simulation was used. For integrating the proxy component with the STM32 board, the STM32 board was connected directly to our computers via USB to test the serial connection implementations both in the proxy and on the STM32. The STM32 was programmed to indicate any communication problems through it's LEDs, which helped to identify any communication problems throughout the project.

Most of the software integration went fine straight from the beginning. Nevertheless tweaking of different values was unavoidable (for example for the parking scenario or the overtaking). After the optimal values were found, these were pushed to to the repository as default values (TacoServ user).

While integrating other components with the DecisionMaker, we faced some issues. When the session was up and running with one supercomponent, one proxy and the configuration tool, the DecisionMaker component didn't always receive the SensorBoardData and DecisionMakerMSG as it was supposed to. Sometimes it affected both data types, sometimes just one. We ended up hard coding the state value and restarting the DecisionMaker each time until SBD values were received.
We weren't able to fix this problem. Rechecking with the teacher confirmed that we are using the receiving mechanisms in a correct way. We suspect it may have to do with processing time issues or maybe the Linux environment on our Odroid, and we would have investigated this further if we had more time.