6th Home Assignment – Verification & Validation

Autonomous Vehicle – Adaptive Cruise Control

After the successful simulation of the Adaptive Cruise Control (ACC) module, the team will focus on verification and validation activities.

We are currently in the design phase of adopting this functionality to our intelligent buses. The senior system engineers have finished working on initial requirements and architecture. The goal of this assignment is to perform early verification and validation activities for this functionality to support their work. As before, we are planning to use vehicles with safety drivers. Initially, these drivers can turn on the ACC if they decide that the traffic conditions are appropriate. If the ACC is turned on, then the vehicle is responsible for maintaining a controlled speed.

Requirements

The initial high-level requirements of the ACC components are the followings.

REQ1 The user shall be able to turn on the ACC.
REQ2 When turned on the ACC shall maintain the target speed set by the driver.
REQ3 If the current speed is lower than the target speed and ACC does not detect a car in front in 90 meters, then the ACC shall command the engine to accelerate.
REQ4 If ACC detects a car in front, then it shall decrease the speed of the car.

TASK T1: Reviewing requirements

These requirements may not fulfill all desired quality characteristics (complete, unambiguous, identifiable, consistent, verifiable...) of good requirements.

a. Are there any requirements that are ambiguous?
b. Are there any requirements missing?
c. Are there any questions or issues with the requirements?

Perform the review of the specification and the requirements. You can use the best practices and recommendations from the Requirements presentation as a checklist.

Collect your feedback, and prepare them as a structured list of questions or suggestions that can be discussed with the responsible people. Summarize your findings in the documentation of the assignment.

High-level design

After the simulation results, the architects were pleased with the following architecture for the ACC component. The distance of the cars in front is measured using a radar sensor. The ACC continuously receives the current speed of the car from a speed sensor. The driver uses switches and buttons to command the ACC, and the ACC Commander is responsible to transmit these commands (later the ACC Commander can receive commands from the self-driving component). Finally, the ACC is connected to the Engine Controller to accelerate or decelerate the car. Figure 1 summarizes these blocks.

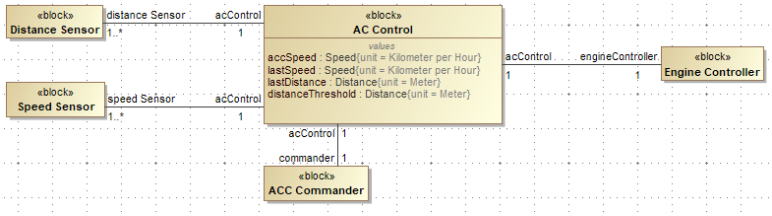

Figure 1: Blocks responsible for adaptive cruise control

Testing the ACC component

To support testing the ACC component in the modeling phase, the systems engineers have designed the test components and test configuration as follows. You can download the detailed test models using the UML 2 Testing Profile on the home assignment page. Be sure to install the "Testing Profile" plug-in in Cameo Systems Modeler if you haven't already done so and check the relevant YouTube videos[1]. The models required for the task are in the *ACC V&V* package.

[1] Test Configurations and Test Cases, Testing with Cameo Simulation Toolkit at `https://www.youtube.com/channel/UCYMdXCyh_e5b1Y7Fwy8zBlQ`

The *TestConductor* implements the distance and speed sensors along with the ACC Commander component. Its task is to provide inputs for the ACC component. The *MockEngineController* is an *EngineController* which captures the signals sent from the ACC component to the *EngineController*.

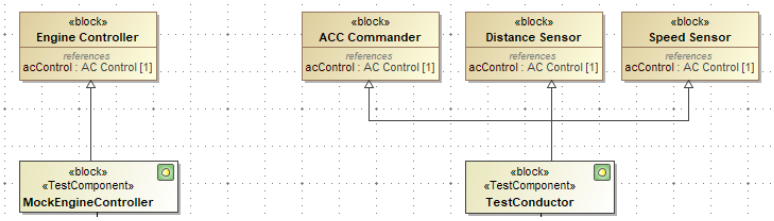

Figure 2: Test architecture

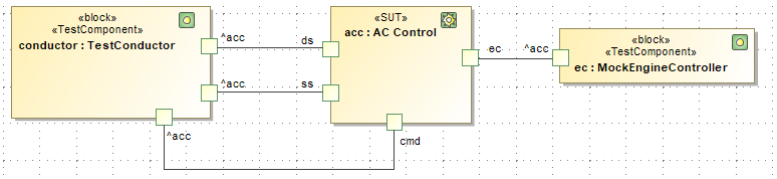

Figure 3: Test configuration

Spend some time with the model and understand the components in the `Test` package. It is probably worth to appreciate the differences compared to the simulation task.

TASK T2: Defining and executing test cases

Your objective is to verify that the ACC component implements its interfaces and works correctly. The ACC component should be able to communicate using all defined ports and should satisfy all the requirements.

a. Refine the above objective into verifiable test objectives. The test objectives should be defined based on the requirements and high-level design. Include the description of the test objectives in the documentation.
b. There are two test scenarios already defined in the model. Execute the test scenarios using the Cameo Simulation Toolkit and get familiar with the different simulation options. You can analyze the outcome of the tests using the *Instance Table* in the `Test Results` package.
c. Design test scenarios for your test objectives and model them using sequence diagrams. At least 3 scenarios are expected.
d. Execute the new test scenarios and evaluate their results.
e. Inspect the internal behavior of the ACC component and check that every important behavior has been covered by your tests. Refine the high-level requirements of the ACC component if there is any uncovered behavior.
f. Define and evaluate the test cases that cover the refined requirements using Cameo Simulation Toolkit.
g. Summarize the results of testing in the documentation (what have you learned about the system under test, have you found any issues, do you have any recommendations, etc.).

The expected outcomes of the assignments are 1) an updated model with the new requirements, test scenarios and test execution results, and 2) documentation with the results of the review phase, new test objectives and test scenarios *with traceability information*, and summary of testing. Keep in mind that this task is not primariliy about simulation – it is just a tool we use to perform the testing activities. The focus should be on testing this time.

Tasks for extra IMSc credits

On real roads and traffic a naive ACC component could misunderstand the situation and make wrong decisions. The above design considers only straight road segments and one car in front of the vehicle.

a. Can you think of situations that can be challenging for the ACC implementation above?
b. Design abstract test cases to illustrate some challenging situations. You do not have to implement them in the model, only specify them in the documentation. But remember that a test case should contain several pieces (objective, execution preconditions, inputs, expected results, postconditions...), use tables to capture this information.