Test Plan

Editor: Julius Kokko Ekholm

Version 0.2

Status

Reviewed		
Approved		

PROJECT IDENTITY

2018/HT, Cascar Group
Linköping University, Dept. Electrical Engineering (ISY)

Group members

Name	**Responsibility**	**Phone**	**Email**
Andreas Lundgren	Project Leader (PL)	070-022 56 44	andlu901@student.liu.se
Julius Kokko Ekholm	Documentation (DOC)	070-304 21 42	julek017@student.liu.se
Adam Ahlgren	Software (SW)	070-796 83 71	adaah731@student.liu.se
Karl Gudmundson	Testing (TST)	070-578 84 82	kargu357@student.liu.se
Anton Kullberg	Software (SW)	073-959 97 71	antku588@student.liu.se
Jonathan Hanses	Design (DES)	076-260 77 47	jonha594@student.liu.se
Andreas Larsson	Information (INF)	076-815 58 03	andla514@student.liu.se
Sten Magnusson	Hardware (HW)	073-766 99 43	stema100@student.liu.se

Email list for the whole group: cascargroup@gmail.com
Web site: *under development*

Customer: Lars Nielsen, lars.nielsen@liu.se
Customer contact: Erik Frisk, erik.frisk@liu.se
Course leader: Daniel Axehill, daniel.axehill@liu.se
Tutor: Pavel Anistratov, pavel.anistratov@liu.se

Contents

Document history

Version	Date	Changes	Sign	Reviewed
0.2	2018-12-	Second draft. Minor adjustments.		
0.1	2018-12-07	First draft.		

1 Introduction

This document provides an overview of the tests that will be conducted in the project *Multiple Autonomous Vehicles in Complex Scenarios* in the course TSRT10 given by Dept. of Electrical Engineering at Linköping University 2018. The test plan contains a description of all tests required to verify the requirements specified in the Requirement Specification [1]. As the Requirement Specification gets updated, some of these tests may be subject to change.

The tests are specified according to the table defined below:

Table 1: Template for the test list

Test nr.	Requirement nr.	Description	Resources	Timeframe

The table defines the number of the test, which requirements that are tested, a description of the test containing purpose, implementation and criteria, a list of all resources needed to perform the test and an approximate time when the test will be performed.

1.1 Test protocol

Whenever a test is performed, a test protocol needs to be completed. The test protocol specifies which test is performed, a result of the test, a date and possible comments. A template of the test protocol can be found in Appendix A.

1.2 Test responsibilities

Ultimately responsible for all tests is the Test Responsible (TST). At a lower level, the person responsible for the module where the test is performed is in charge.

1.3 Failed tests

If the criteria for a test is not fulfilled, the test is deemed failed. It is inevitable that some tests will fail. When a test is failed, there needs to be an evaluation to why this happened and an assessment of possible actions following this. Possible actions include: devoting more time to the problem, redesign the test and if necessary, renegotiate the requirement(s). These evaluations will however not be present in this document.

2 Tests

This section contains all tests needed to confirm that all requirements are fulfilled. The tests are specified according to the template in Table 1.

2.1 Unit tests

Some unit tests are implemented for the code in the form of a test script called `cascar_test.py`. This script executes the files pre-defined in it. These pre-defined files are test classes of a special form. Each such python script must contain a class named "Test..." with methods in it named "test_..." where each method should test something and return true or false depending on if the test was successful. The `cascar_test.py` script is executed like a normal python file.

2.2 Mission Control

Note that this section refers to the section "Route Planning" in [1].

1	9	**Purpose**: Test if the the subsystem works as a ROS node. **Implementation**: Start the subsystem as a ROS node and see that different topics etc. exists. **Criterion**: The subsystem should be able to use ROS communication.	Laptop	w. 45
2	10, 11, 12, 13, 16	**Purpose**: Test if a kinetically feasible path can be created. **Implementation**: On a laptop, start the GUI and Mission Control ROS nodes. Initialize a start position in the MC node to mock the position of a car. Using the GUI, choose a map in the list and set a goal position by clicking a node somewhere on the map. **Criterion**: The car should be able to provide a path, given by a set of global positions, ids and road types. If the path follows the map representation, it is also feasible.	Laptop	w. 46

2.3 Vehicle Perception

3	18, 19	**Purpose**: Test if the subsystem can receive data from VPS **Implementation**: Start the VPS subsystem that has a implemented ROS node on one car and see that it can receive data from VPS. **Criterion**: The car/subsystem should be able to receive information from VPS.	one car	w. 46
4	20	**Purpose**: Test if data from the Lidar can be received from the subsystem. **Implementation**: Start the subsystem and the Lidar and see that the subsystem can process the incoming data stream from the Lidar using ROS communication. **Criterion**: The subsystem should be able to receive data from the LIDAR	one car	w. 47
5	21, 22	**Purpose**: Test if several cars can communicate with each other using ROS communication. **Implementation**: Starting several ROS nodes under different namespaces on two different cars and see if they can communicate with each other, the received data should be processed in a appropriate manner. **Criterion**: The subsystem shall receive data from the other cars and show that it has processed the results.	two cars	w. 47
6	23	**Purpose**: Test if the internal communication publishes data on the correct topics with the correct interval. **Implementation**: Using test code the frequency of the communication of the nodes can be tested and by echoing the topics it can be seen that the correct data is being published. **Criterion**: The subsystem shall publish the received data to the other internal subsystem	one car	w. 47

2.4 Behavioral Decision Making

7	25	**Purpose**: Test if the subsystem can be created with a ROS node. **Implementation**: Create a ROS node and see that it exists. **Criterion**: The subsystem should be implemented in ROS.	Laptop	w. 45
8	26, 27, 28, 30	**Purpose**: Test if one of the cars can make autonomous decisions based on the actions of the other cars. **Implementation**: Set two cars, one behind the other. Set the map to some form of loop and set the paths so the cars will drive endlessly in the loop. Now start the cars at roughly the same time. See that the cars behave in a platooning fashion where the car behind follows the car ahead. Test number 9 must also be completed for the requirements to be fulfilled. **Criterion**: The car being tested (the car behind) should make a decision to platoon.	Laptop, two cars	w. 48
9	26, 27, 28, 30	**Purpose**: Test if one of the cars can make autonomous decisions based on the actions of the other cars. **Implementation**: Set the map to a roundabout scenario where one car drives around in the roundabout endlessly. Now place a car somewhere and give it a path so it will need to enter the roundabout. Start the car when the car in the roundabout has roughly a quarter of the roundabout left. The car entering the roundabout should stop and let the car in the roundabout pass before entering.Test number 10 must also be completed for the requirements to be fulfilled. **Criterion**: The car about to enter the roundabout should wait and let the car in the roundabout pass before entering.	Laptop, two cars	w.48

10	29	**Purpose**: Test if the subsystem can receive information from the other cars. **Implementation**: Start two cars (they don't need to drive) and make sure that each of them gets information about the other cars position, speed and trajectory. **Criterion**: A car should receive information about the other active cars.	Laptop, two cars	w. 46
11	32, 35	**Purpose**: Test if a general structure works for all purposes. **Implementation**: Run the test script cascar_test.py and make sure the tests pass. **Criterion**: The tests defined in all the test scripts defined should pass (given that they are implemented).	Laptop	w. 46
12	31	**Purpose**: Test if the subsystem can output the current state. **Implementation**: Run the node and make sure the topic BDM/current_state outputs a state. **Criterion**: The state should be output to the topic.	Laptop	w. 44

2.5 Motion Planning

13	37	**Purpose**: Test if the subsystem can be created with a ROS node. **Implementation**: Start a MP ROS node. **Criterion**: See that the ROS node exists.	Laptop	w. 45
14	38, 39	**Purpose**: Test if a trajectory can be created. **Implementation**: Place one car in the VPS. Load a map in the GUI. Set a goal point. Observe that a path has been published on the `/cascar/[my_ns]/MC/path` topic. Observe that a goal point has been published on the `/cascar/[my_ns]/BDM/goal_point` topic. `[my_ns]` is the namespace of the car, given by the argument in the launch file. **Criterion**: MP should publish a trajectory on the `/cascar/[my_ns]/MP/trajectory` topic.	Laptop, VPS, one car	w. 45

15	40	**Purpose**: Test if a trajectory is collision free given a static environment. **Implementation**: Place one car in the VPS. Load a map in the GUI. Set one or more goal points, creating a path that does not interfere with any static objects in the map. Start the car. **Criterion**: The car should create a trajectory along its path, not interfering with any objects known when the path was set.	Laptop, VPS, one car	w. 48

2.6 Vehicle Control and Modeling

16	45, 46, 47, 48	**Purpose**: To test if the vehicle controller is able to fulfill the requirement of maximal path deviation. **Implementation**: Place one car in the VPS. Each car shall be tested. A predefined trajectory is sent out to the car which it is supposed to travel. Position and odometry data is logged throughout the test. Afterwards, the data is plotted to see the car's deviations from the path. **Criterion**: The center of the car shall not deviate more than 10 cm in either direction of the path.	Laptop, VPS, three cars	w. 48

17	49	**Purpose**: Test to see that the car stops when the motion planning subsystem does not transmit any new trajectory. Also, this test exists to make sure that the car stops at the end of a trajectory. **Implementation**: This test is divided into two parts: loss of new trajectory from the motion planner and completion of a set trajectory. First part: place one car in the VPS. Let the motion planner send out a trajectory which the car should follow. Then, abort the transmission of a new trajectory to test if the car stops. Second part: place one car in the VPS. Send out a trajectory (predefined or from the motion planner) and see that the car reaches the end point and stops. **Criterion**: If the car has not received a trajectory from the motion planner it should stand still. Also if it reaches the end of the trajectory without receiving a new one it shall stop.	Laptop, VPS, one car	w. 48
18	50	**Purpose**: To test that the car makes use of single-wheel odometry. **Implementation**: For ease of implementation, this can be tested and verified at when any of the tests above in this table are performed. Logged odometry data and the actual speed (both Qualisys and the one calculated by using odometry data) can be plotted. **Criterion**: The subsystem should use single-wheel odometry.	Laptop, one car (VPS can be used but is not required for the test).	w. 48

2.7 Simulator

19	8, 17, 24, 36, 44, 52, 53, 54, 55	**Purpose**: Test if the simulator is working. **Implementation**: Launch two virtual cars on a track. Set their goal points from the GUI and start the cars. **Criterion:** The cars should be able to traverse the track with in a way similar to the physical cars.	Laptop	w.49

2.8 Graphical User Interface

20	58, 61	**Purpose**: Test of being able to display cars position and current action in GUI. **Implementation**: Let two of the cars drive autonomously. Start the GUI and select a map for the cars to be shown in. **Criterion**: The cars movement should be able to be seen in the GUI map and their current action under their name on the side.	Laptop, two cars	w. 47
21	59, 60	**Purpose**: Test of being able to set path and start car from GUI. **Implementation**: Start up one of the cars. Select two points in the GUI and receive a path from Motion Planning. Start the car in the GUI with a button. **Criterion**: The path should be seen in the GUI and the car should follow the path after it has been started from the GUI.	Laptop, one car	w. 48
22	62, 63, 64	**Purpose**: Test of being able to put in path, draw trajectory and display information for all active cars. **Implementation**: Start up two of the cars. Put in a path for both car and start them. **Criterion**: The paths should, their trajectories and their information should be seen in the GUI.	Laptop, two cars	w. 48

2.9 General Requirements

23	1, 5	**Purpose**: Test of manual platooning. **Implementation**: One car drives manually and the other autonomously platoons after the manually driven car. **Criterion**: A car should be able to platoon a manually driven car, for this to be functional the car needs to drive VPS.	Laptop, one car, VPS, joystick	w. 49
24	5, 71, 67, 75	**Purpose**: Test of autonomous platooning. **Implementation**: Two cars drives autonomously towards their respective goal, a car driving behind the other starts platooning/following the first car. The trajectory is then calculated based on the position of the car ahead. **Criterion**: One of the cars should start platooning the other one while following traffic regulations.	Laptop, two cars, VPS	w. 49
25	2, 3, 4, 5, 67	**Purpose**: Test of autonomous roundabout with one car. **Implementation**: One car drives autonomously through a roundabout. **Criterion**: The car should be able to drive through a roundabout using the whole system. This test checks that the subsystems are functional and the general structure is functional and the system works inside VPS. The car should drive according to Swedish traffic regulations and not crash 9 out of 10 times.	Laptop, one car, VPS	w. 49
26	5, 71, 67, 74	**Purpose**: Test of autonomous roundabout with two cars. **Implementation**: Two cars drive autonomously through a roundabout of a similar speed and do not crash. **Criterion**: The cars should be able to drive through a roundabout using the whole system. This test checks that the subsystems and communication between the cars is functional, it also shows that the trajectory of one car can be adjusted by the other. The cars should drive according to Swedish traffic regulations not crash 9 out of 10 times.	Laptop, two cars, VPS	w. 49

3 Requirements Not Tested

Some requirements specified in the requirement specification do not need any explicit tests to be validated, are implied in other tests or have not been tried to be implemented. This sections aims to present all requirements not mentioned in any test as well as an explanation to why they were not tested. In table 2, all these requirements are listed.

Table 2: Requirements not tested

Requirement number	**Requirement specification**	**Reason not to test**
14, 15, 33, 34, 41-43, 51, 56, 57, 65, 68, 73, 77-80	-	Priority level 2.
6, 66	-	Priority level 3.
7	The system shall work with the provided hardware.	Not all provided hardware was explicitly used, but nothing has been removed or added.
69	The work hours for each project member shall not exceed 240 hours.	The requirement is confirmed by the time plan.
70	The active cars should try to avoid collision with known static objects	There is no functionality for online avoidance of static objects. But the objects can be managed if they are known to the user beforehand, as in test 14.
72	The active cars shall handle all implementation situations with a decent safety margin.	The decent safety margin is implied in all tests made.
76	The different scenarios shall only have one lane in each direction	Design requirement.
81	The code written in the project shall be written with the Google code standard.	This has been regularly controlled during the development.

References

[1] *Requirement Specification Version 1.1.* Adam Ahlgren, Julius Kokko Ekholm, Karl Gudmundson, Jonathan Hanses, Anton Kullberg, Andreas Larsson, Andreas Lundgren och Sten Magnusson. LiTH, 2018.

A Test Protocol

Date:

Test number: Iteration:

Participants:

Test Result

☐ Passed

☐ Failed

Notes