# Design and Formal Verification of a Safe Stop Supervisor for an Automated Vehicle

**Jonas Krook**, Lars Svensson, Yuchao Li, Lei Feng, Martin Fabian
**krookj@chalmers.se**, larsvens@kth.se, yuchao@kth.se, lfeng@kth.se, fabian@chalmers.se

## Questions for a safe stop supervisor:

- What safety benefits are achieved by formally verifying the software?
- What requirements can be proven? Which cannot be?
- How is nominal functionality included in the architecture and methods?

## Safe transportation; can we STOP here?

The scenario considered is when an automated vehicle is parked in a spot at parking lot A, and it receives a transport mission where it needs to drive to and park in a goal spot at parking lot B. To do this, it first has to plan a path connecting the two parking lots via the road network. Then it needs to generate a path from a point in A to the road network. When it arrives at parking lot B it needs to construct a path from the road to the goal parking spot.

There is no driver so the vehicle itself needs to ensure safe driving, which means that the vehicle always should be able to reach a safe state while driving, if an error occurs. For driver assistance functions the safe state is usually to cede all control to the driver, but for automated vehicles we cannot let the vehicle continue to move uncontrolled. So the safe state has to include being stationary. However, the vehicle cannot stop just anywhere.

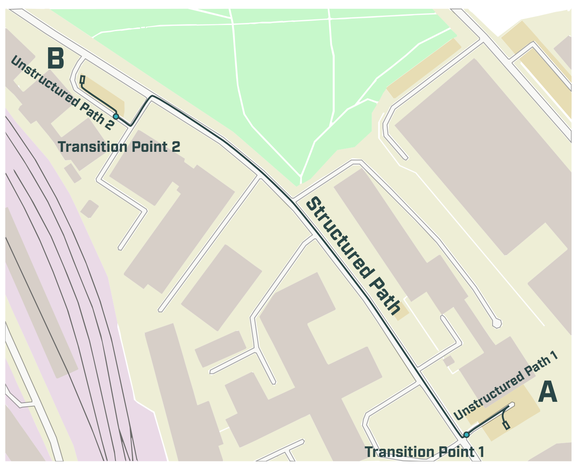


Fig. 2: Example mission with paths and transition points.

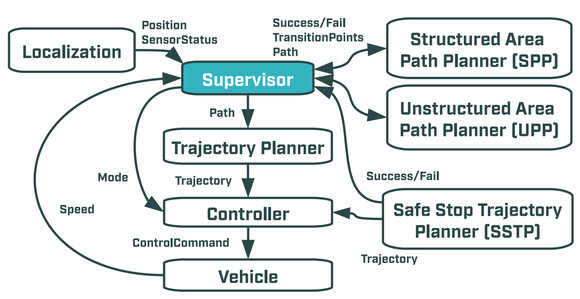


Fig. 1: System architecture.

## One Supervisor to rule them all

To accomplish the transport mission, a supervisor brings together the two nominal path planners UPP and SPP, and makes sure that the SSTP stops the vehicle in a safe spot (e.g. on the shoulder) if and only if an error occurs. The supervisor is implemented using model based design and integrated in a ROS environment.

The proposed supervisor can handle GPS sensor failures and path generation failures. Extensions for additional types of failures can be accommodated with relative ease.

Formal verification requires a model of the vehicle and the software, so a verification model of the supervisor is derived from its implementation. Only key aspects of the other software components in Fig. 1 are modelled. The vehicle is modelled as a standard discrete-time vehicular longitudinal dynamic system along the length of the path.

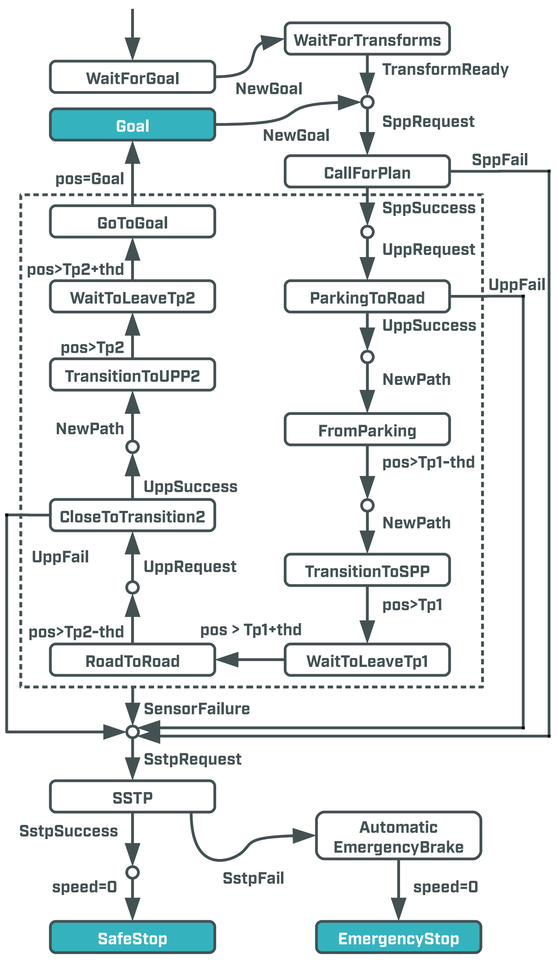


Fig. 3: The proposed supervisor.

## Going for SPIN (What the model checker proves)

The model checker SPIN is used to formally prove the following requirements when the supervisor is interacting with the rest of the system:

- The supervisor and the four concurrent planners shall never stop at invalid end states.
- There is always a future state in which the vehicle is stationary.
- When arriving at the goal position, all paths must have been generated.
- If the vehicle passes the end point of a path then the next path must be known already.
- If the vehicle stops safely then a failure must have occurred and the SSTP cannot have failed.
- If the vehicle stops by emergency braking then a failure must have occurred and the SSTP must have failed.
- If a failure occurs then the vehicle must be stopped safely by SSTP. If SSTP fails then the automatic emergency brake must perform an emergency stop

SPIN also produces a counterexample when trying to prove that it is not possible to reach the goal, showing that the supervisor can successfully coordinate a transport mission.

## Going for a spin

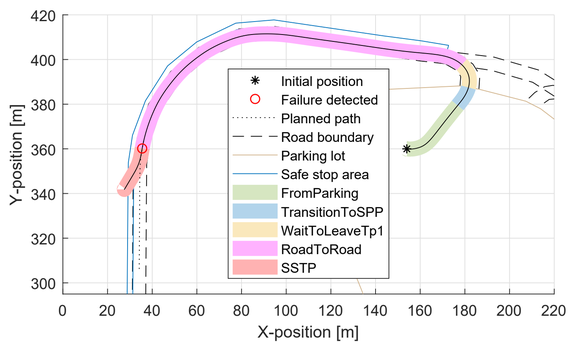


Fig. 4: Position and state during a simulation with a GPS failure.

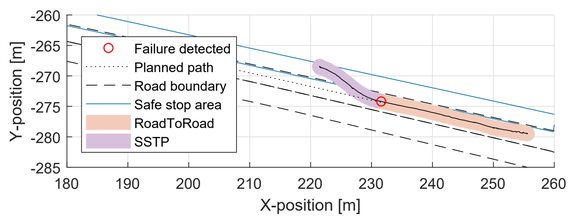


Fig. 5: Position and state during a drive with injected GPS failure.

## Take-aways

- It is possible to formally verify the implementation and not only the design, giving more confidence of correct software.
- Implementation with verification in mind encourages safer coding.
- Only key abstract aspects of the nominal functionality need to be modelled for verification.
- It is difficult to formally verify requirements on nominal functionality with the chosen method.
- Verification requires modelling of software, which is manual and error prone.
- The current design may fall back on automatic emergency braking to stop. Future research should look at strategies that ensures the availability of SSTP.

https://github.com/krooken/wasp-des-rcv https://research.chalmers.se/publication/509469 https://vimeo.com/319427372 https://doi.org/10.5281/zenodo.2651006

WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM