Prototype

Group: Byggare Bob

Linus Berglund[1], Johannes Edenholm[1], Hugo Frost[1], Hamza Kadric[1], Erik Källberg[1], Johan Svennungsson[2], Rikard Teodorsson[2], Timmy Truong[1], Arvid Wiklund[1] and Karl Ängermark[1]

[1] *DAT255 – Software Engineering Project, Bachelor of Science programme in Computer Engineering, Chalmers University of Technology*
[2] *DAT255 – Software Engineering Project, Master of Science programme in Software Engineering, Chalmers University of Technology*

October 27, 2017

Contents

1 Code quality

Using FindBugs returns 0 bugs, here is the link to our generated report.

2 Unit / integration / system tests

The MOPED prototype code is currently not beeing tested with any system tests. The main reasoning behind this was the decision not to use test driven development. If there had been time, then tests could be applied to further ensure that the methods and classes worked as intended.

3 Design rationale

MOPED prototype software design is seperated in to two parts, Hardware and Software, with a defined interface between them. Defining an interface early on enables the writing code that is dependent on the defined methods without them beeing implemented, the major drawback is that once specified the interface might be difficult to change. The reasoning behind dividing the code in to hardware and software blocks was to ensure that there was only one instance of the hardware code running.

3.1 API-level

For the API refer to Java-doc that can be found in the Github repository here.

3.2 Backend

Comunication with the MOPED hardware is handeled with an asynchronous CAN class that reads can-frames aswell as sending frames to control speed and steering.

3.3 External dependencies

Since the ALC was the last User story that was worked on, and we knew that we would need a way to process an image, we swiftly realized that we did not have time to proceed and craft our own code for this. Therefore, the palpable choice was to use a predefined java library for this. OpenCV contained the right methods for filtering out the colors of an image and pointing out to where they were located.

4 Overview

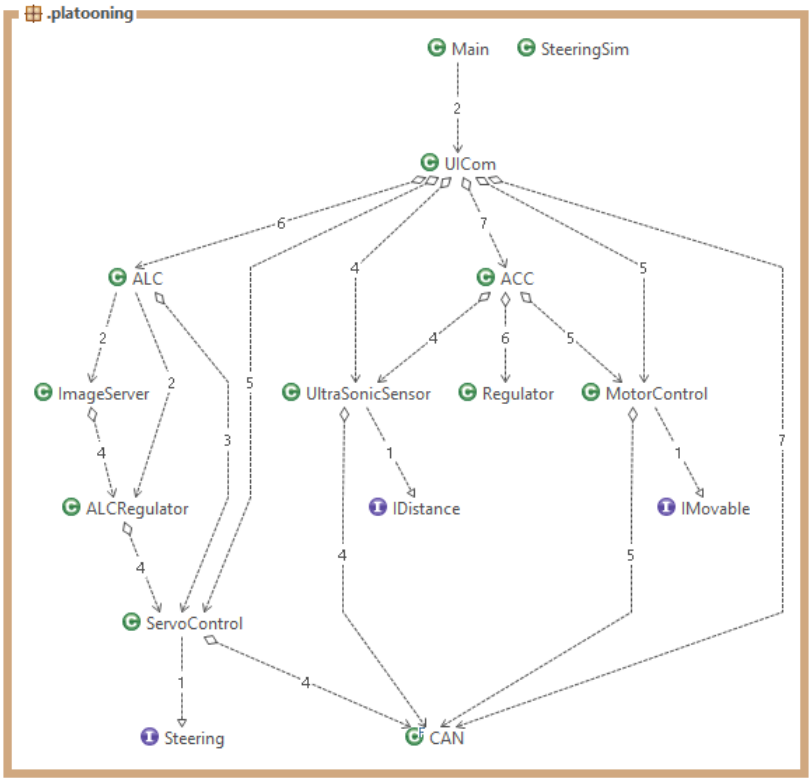

Figure 1: Code structure overview (STAN)

4.1 Behavioural

The intended behaviour of the software was to function as a control system. It would measure the distance to the nearest object in front and set its speed accordingly. Additionally it uses its camera to observe an object in front and follow it by controlling the wheels.

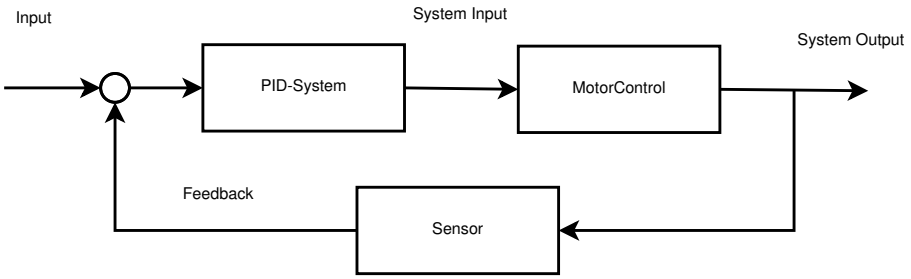

Figure 2: The control system for the ACC

4.2 Structural

MOPED prototype is devided into software on and off the TCU unit. On the TCU the software is devided into threads.

- UICOM - User interface communication server that controls the other threads on TCU
- CAN - CAN bus I/O threads that acts as the mediator/api between the other threads and the VCU and SCU.
- ACC - Adaptive cruise control thread controls VCU through CAN
- ALC - Adaptive lateral control thread, controls steering via image recognition

Off TCU there is an user interface and an OpenCV image recognition server for adaptive longitudinal control. (figure 3).

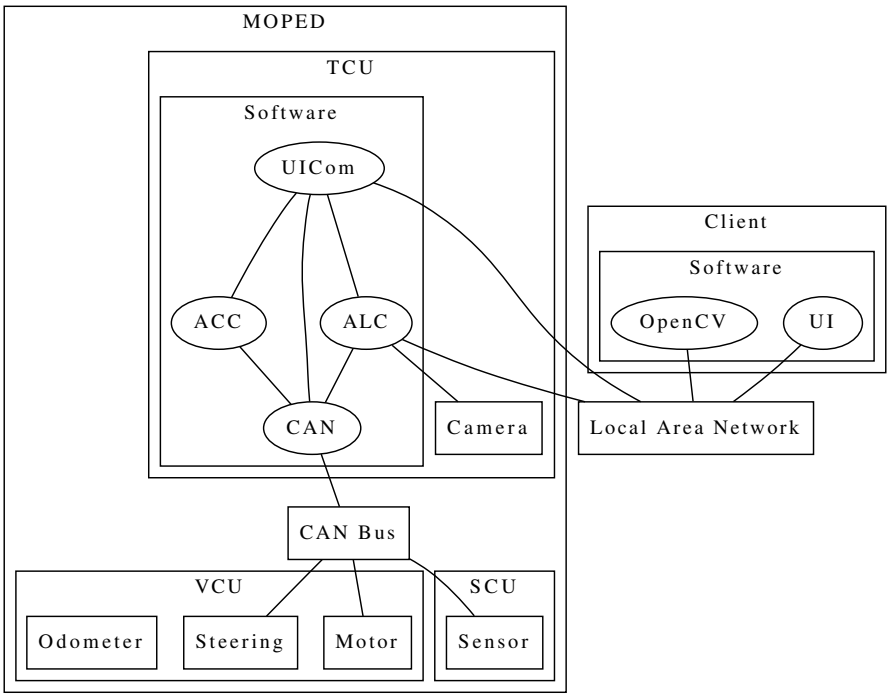

Figure 3: Structural layout

4.3 Protocol

4.3.1 MOPED UI communication

UI control over MOPED was achieved by having the user interface connect to MOPED UICom as a client. Whenever the operator changes the state of the user interface, by interacting with it, a command is sent to UICom. (figure 4)

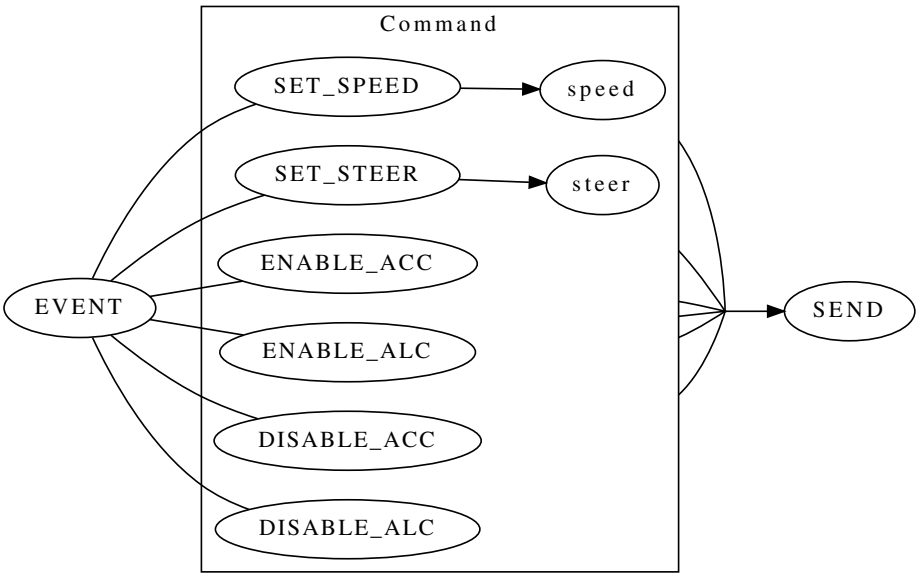

Figure 4: UI event to UICom

The commands are executed by UICom, much like a processor executes instructions, by interpreting the first byte received as command number. Depending on whether the command takes arguments or not the following byte is interpreted as either an argument or next command to be executed. (figure 5 and 6).

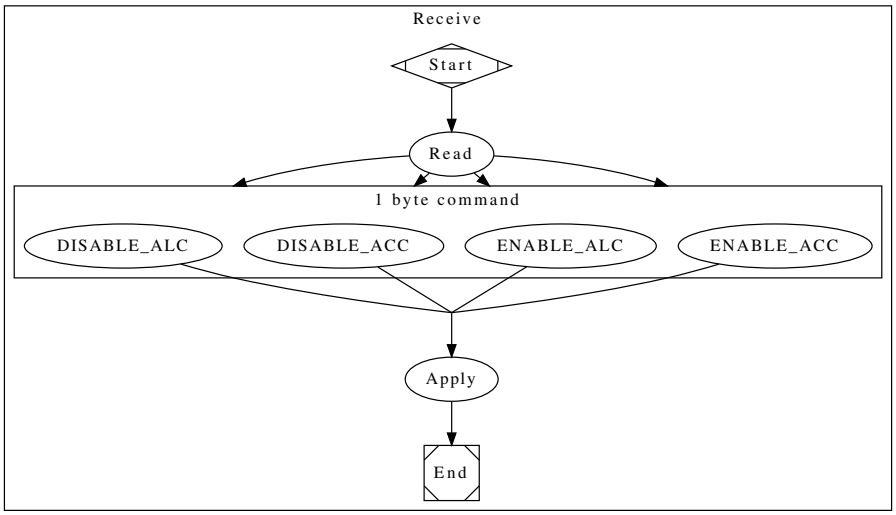

Figure 5: UICom receive with single byte commands

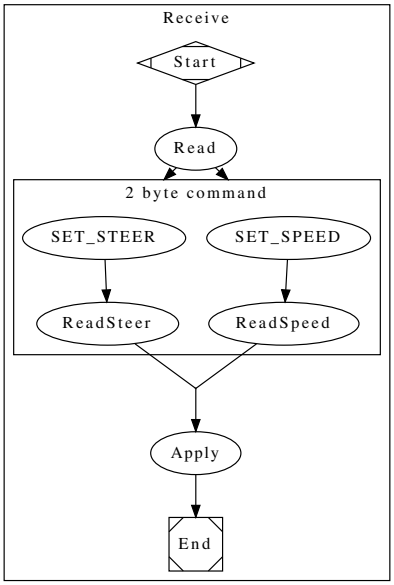

Figure 6: UICom receive with two byte commands

If ACC and/or ALC is active UICom will provide the user interface with 10hz update packets containing the current speed and steering values (figure 7).

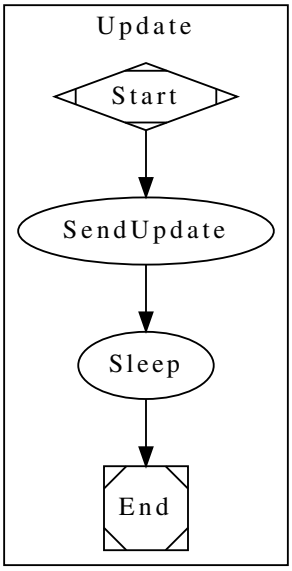

Figure 7: UICom to UI update

4.3.2 ALC

Our approach for implementing adaptive latitudinal control(ALC) was to use OpenCV; an image recognition library. The library was used to identify three red circles located on a piece of paper in front of the MOPED and then calculate the offset by using the red circle in the middle as a point of reference. It works by first removing everything that is not red in the image, then checking if there are three red circles in a row and extracting the center circle's x-coordinate and calculating an offset between -100 (left) and 100 (right) compared to the center point of the image (0).

Since the ARM-processors located on the Raspberry Pis are not optimal for any real-time image recognition, we installed OpenCV on our own laptops and implemented a client-server-model. By remotely starting a video capture on the MOPED's camera and then reading each frame while using OpenCV for image recognition we successfully received an offset. The offset was then sent to the steering servo which made the MOPED turn.

The current solution works quite well except when the camera spots something else that is red outside the paper. Then it can detect three red items in a row even when there are none and as a result start turning in the wrong direction. It would be better if it didn't detect red items outside the paper to prevent wrong turns. A solution could be to check that the circles are all inside the borders of the paper, or to check the height and width of them and see if they are the

same.

The client-server-model could be replaced if the onboard CPU had a higher clock frequency. One potential solution could be replacing the TCU with a x86-motherboard with beefier embedded CPU.

5 Testing

5.1 CAN

Due to the unreliablility of the hardware it was necessary to have a way of testing CAN communication during development without access to the hardware. To meet this requirement two programs of the can-utils for SocketCAN was utilized. The programs were 'candump' and 'canplayer'. 'candump' was utilized to capture can frames and save a session. It was then possible to emulate a scenario by replaying a saved session using 'canplayer' to a virtual CAN port. This proved essential for the development the sensor data parser as it made it possible to iterativly develop and test.

5.2 ACC

Our initial attempts to test the ACC was to create a simulator that would act as a car moving in the front of ours. The idea was to use this to regulate our control system with the Ziegler-Nichols method. However, it proved very time consuming to implement acceleration and deaccelaration in the simulator and as the group made progress with the hardware communication we decided to abandon it and do practical testing on the real MOPED instead.

The practical testing consisted of having the MOPED move towards objects and measure the error from the desired distance it was supposed to keep. When it successfully stopped in front of static objects we tried having it follow objects instead. Between each test we tuned variables to make the deacceleration as smooth as possible. This was somewhat time consuming but ultimately saved time by skipping the simulator step.

5.3 ALC

The ALC feature was one of the last to be completed. When the ALC feature was completed the MOPED we had access to was unfortunately barely able to drive. It was however possible to observe the wheels turning when showing the target to the camera, but we could not test the entire platooning system before the final demonstration. This is why the control system in the ALC is simpler than the ACC.

A previous attempt to simulate the steering had been made so that it could be tested without a MOPED. This had the same problem as the ACC and we weren't sure wether the results would reflect reality.

6 User stories

To organize our User Stories we used Github's Projects feature as our scrum-board. We had six categories; Project backlog, Sprint backlog, In progress, Testing, Done and Abandoned. User Stories would be assigned from the sprint backlog and moved to In progress at the start of each sprint. If a User Story was completed it would be moved to Testing (if testing was applicable) and after being tested moved to Done or back to In Progress. Some ideas we had didn't work out or became irrelevant and their User Stories were put in Abandoned.

Most User Stories were assigned to two or more people, who would complete them together. We approximated the effort of a User Story on a scale of 1 to 5, where 5 would take up the entire sprint. Most User Stories where a 2 or a 3, which meant most team members were working on 2 each sprint.