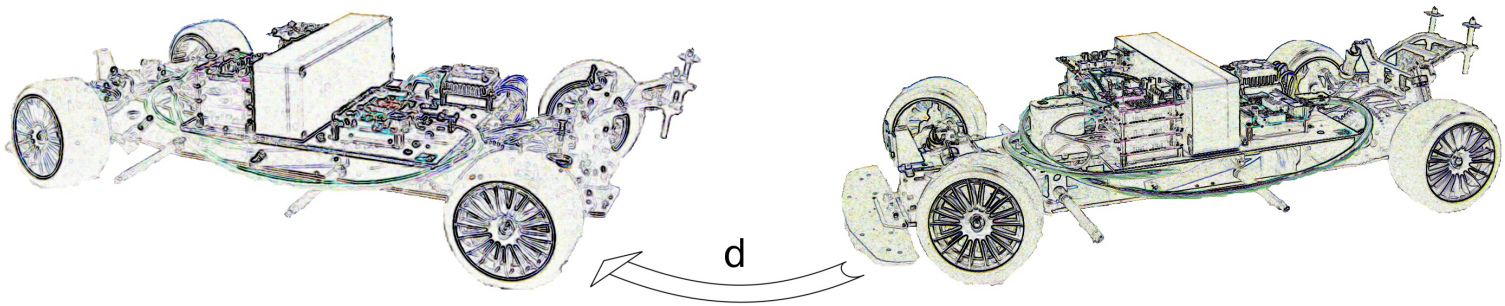
Evaluation of a Model-Based Development Process for Automotive Embedded Systems

Model-Based Development of an Adaptive Cruise Control System

Jonas Mitschang

Fraunhofer Institute for Experimental Software Engineering

September 23, 2009

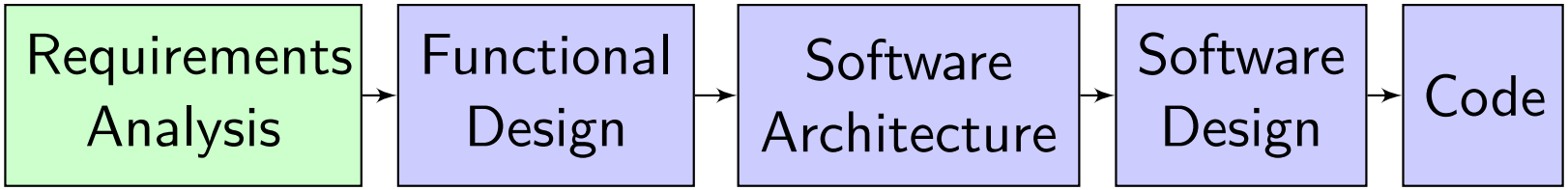


Structure

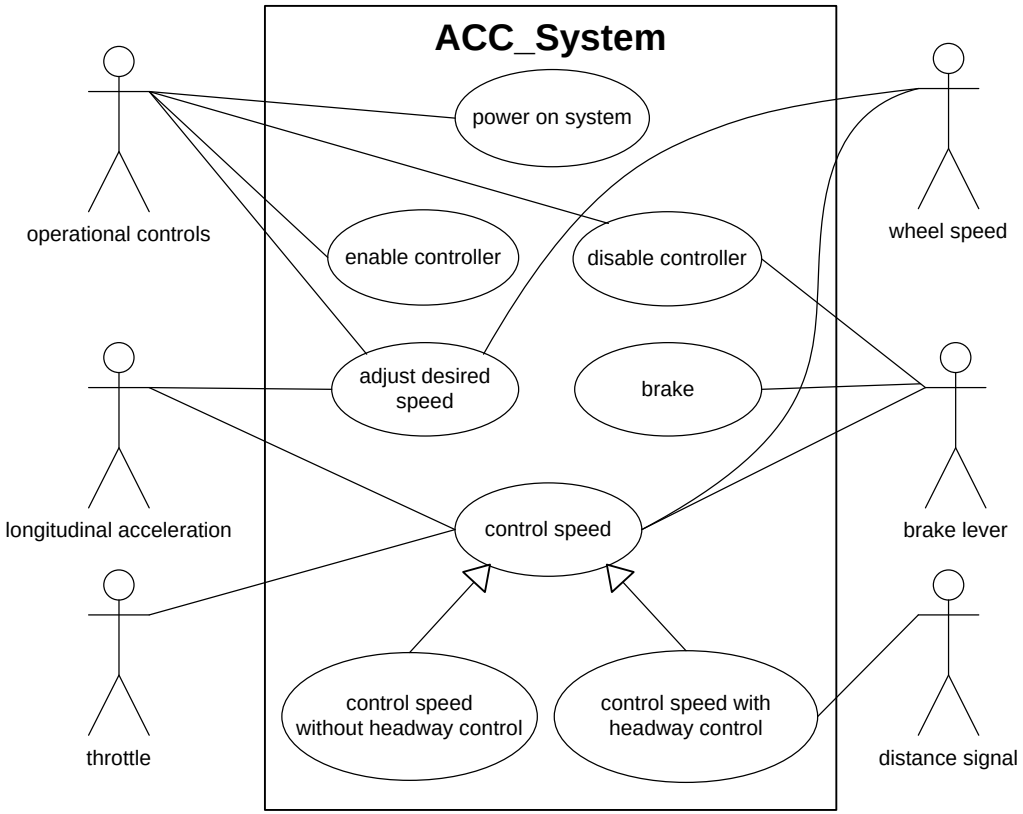
1. Model-Based Software Development Process
   - Requirements Analysis
   - Functional Design
   - Software Architecture
   - Software Design
   - Code

2. Evaluation / Reflection on the Process

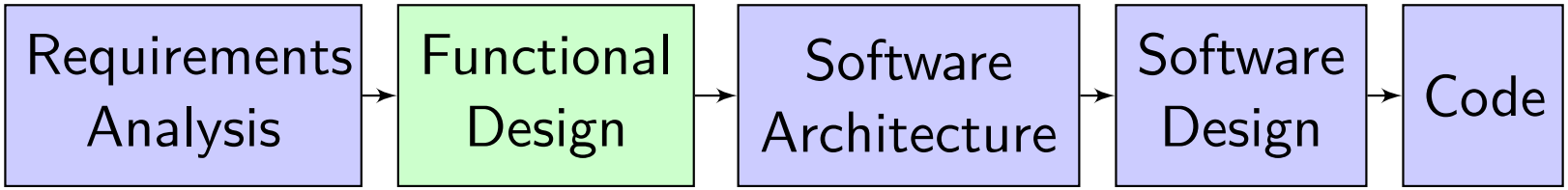
Requirements Analysis



- Powering the system, enabling & disabling the controller
- Adjusting the desired speed
- Braking
- Controlling the speed:
  - Without headway control
  - With headway control: gap 1.5s



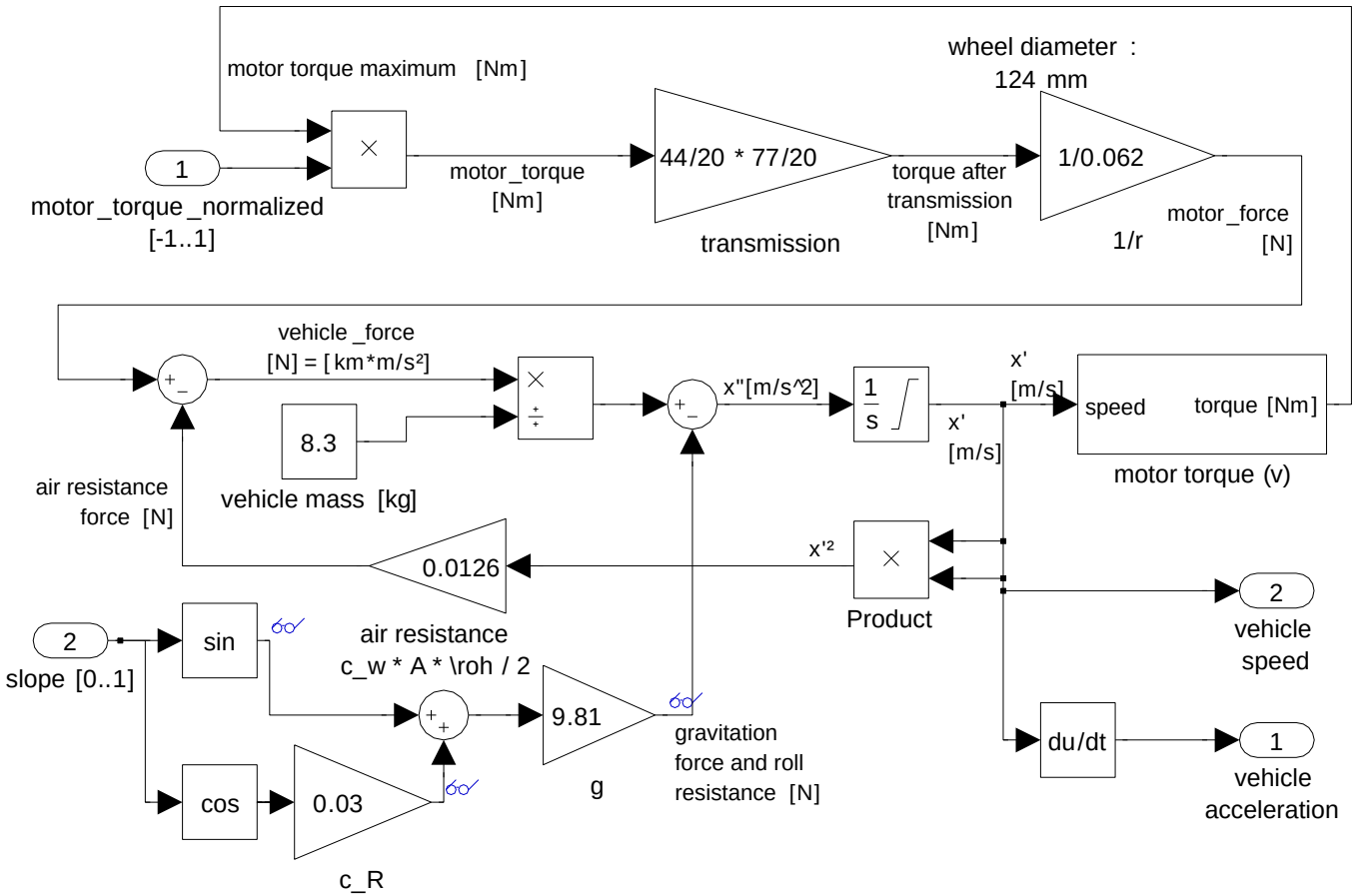
Functional Design



- *Functional Design* done by domain experts
- Main tasks:
  - Vehicle plant
  - Wheel speed sensor filter
  - Speed controller
  - Distance controller

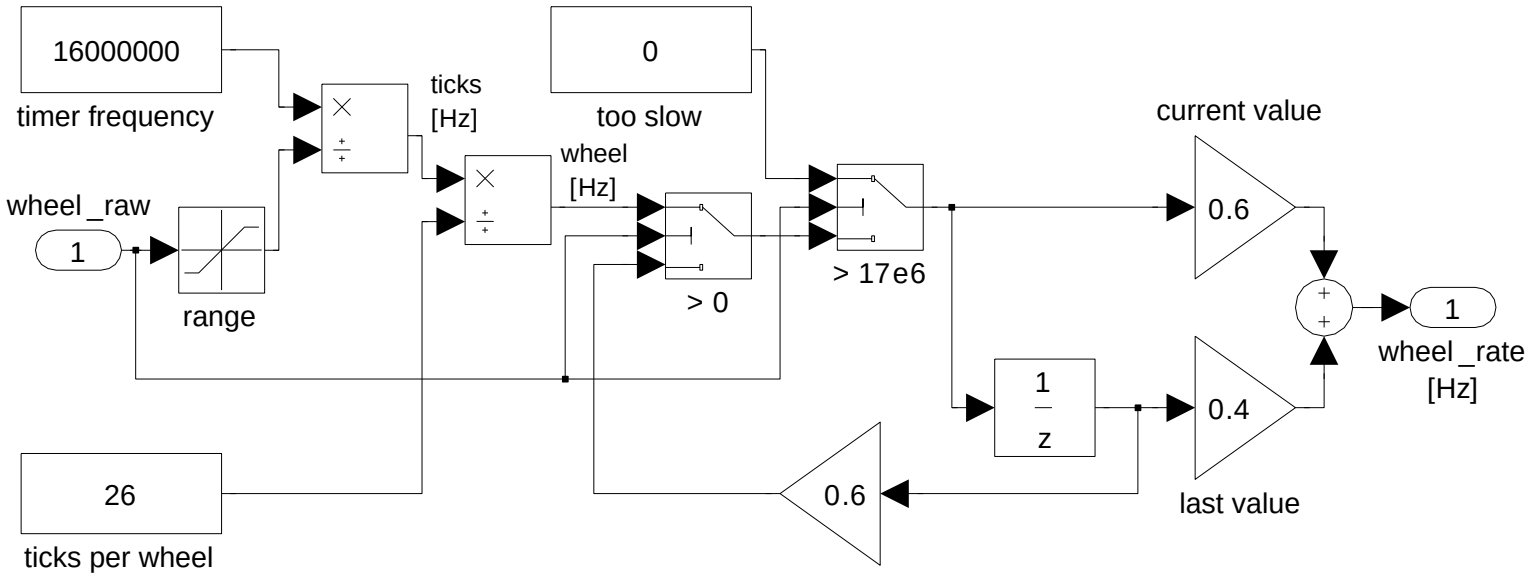
Functional Design - Vehicle Plant

- Model considers: slope, friction, air resistance, and brushless motor
- Not considering: Inertial mass that rotates



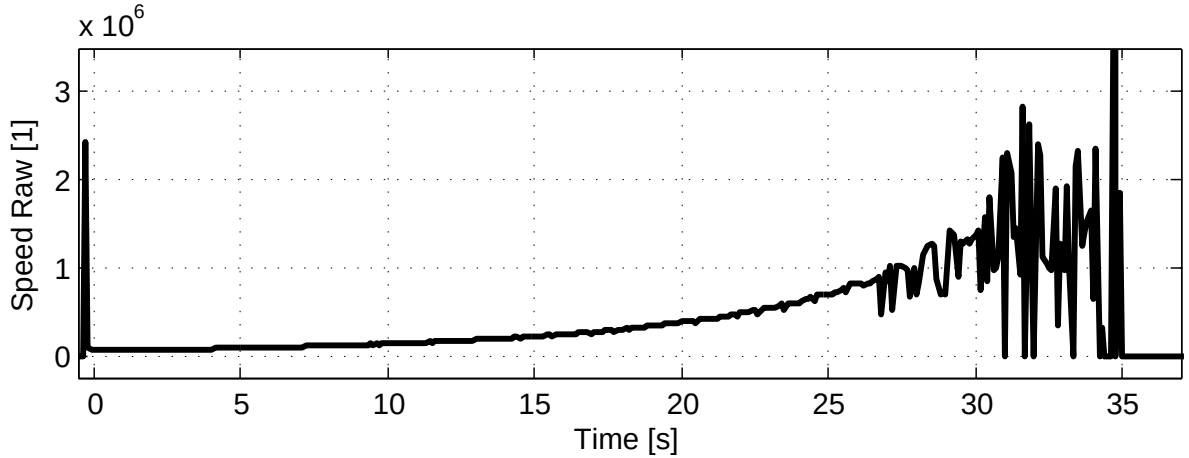
Functional Design - Wheel Speed Sensors

- Sensors generate unprocessed raw data
- IIR filter for converting to SI numbers

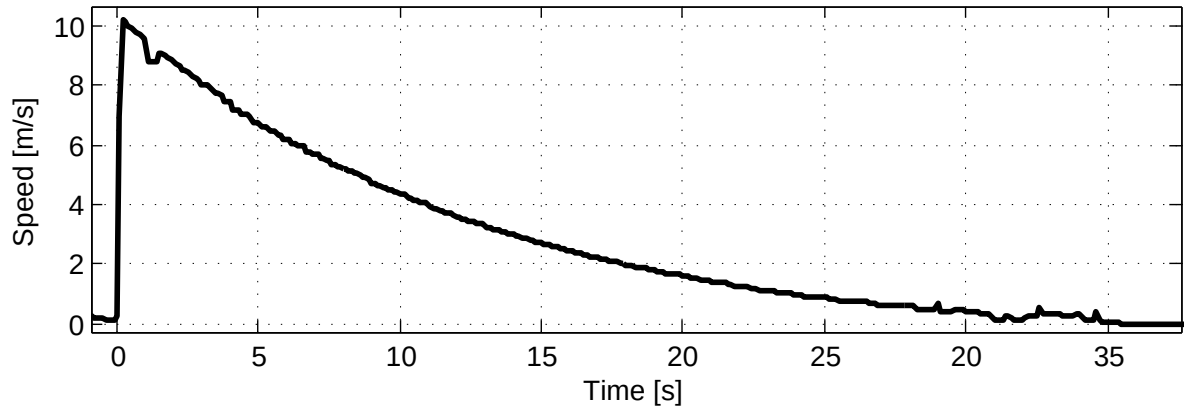


Functional Design - Wheel Speed Sensors

- Unprocessed wheel speed raw data (wheel rolling out):

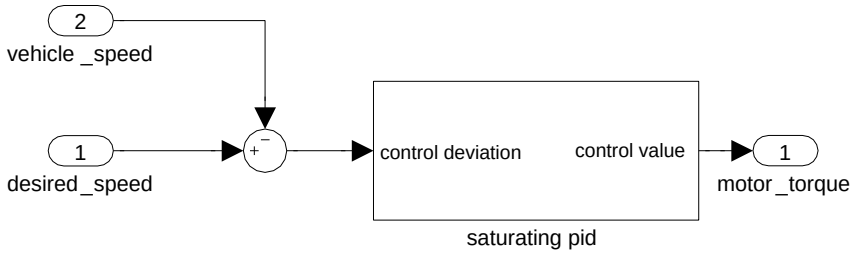


- Processed wheel based vehicle speed:

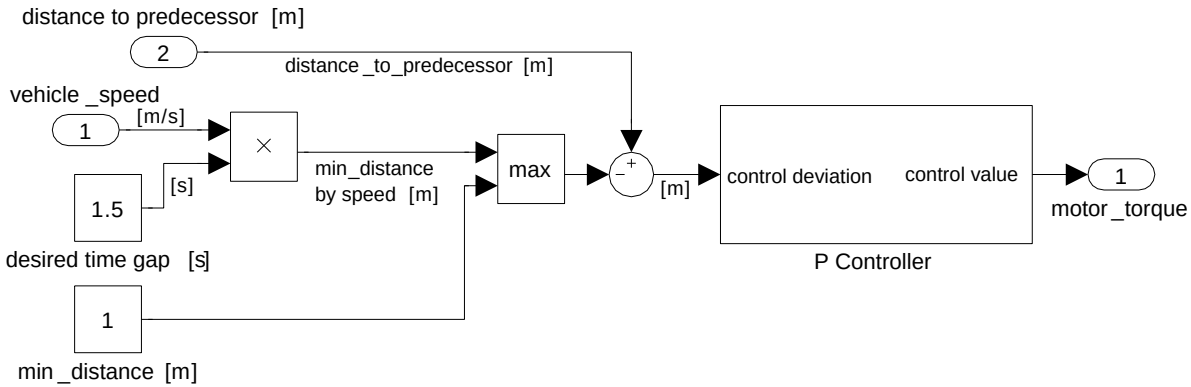


Functional Design - Speed and Distance Controller

- *Speed_Controller*: Control deviation to minimum using a PID controller
- Anti wind up algorithm

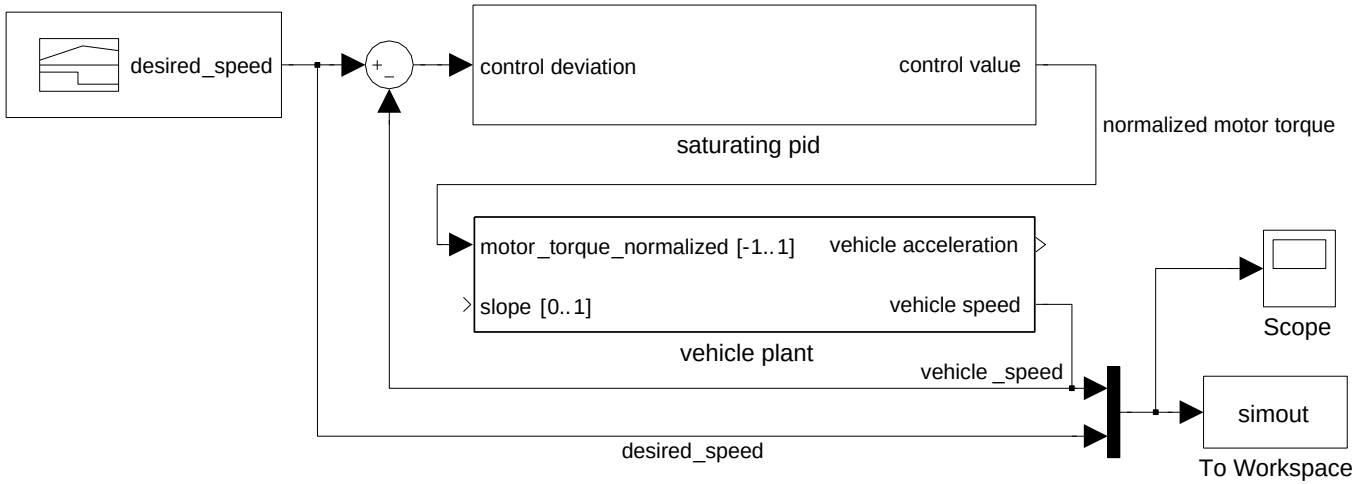


- *Distance_Controller*: Determine the desired distance so that time gap is at least 1.5 seconds



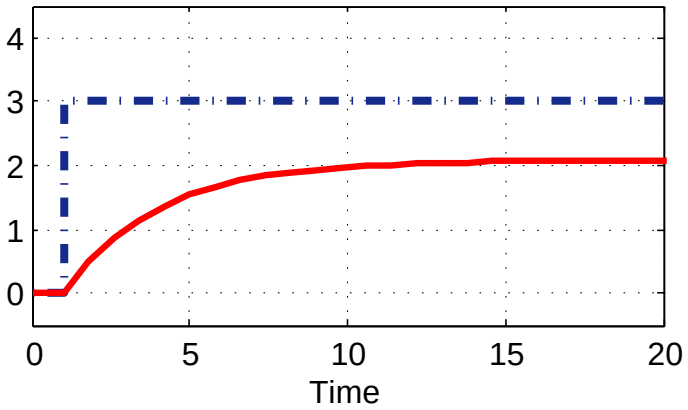
Functional Design - Speed Controller

- The PID controller has three parameters: K_p, K_i, K_d
- Ziegler-Nichols method for parameter tuning
- Block diagram needed for simulating the speed controller



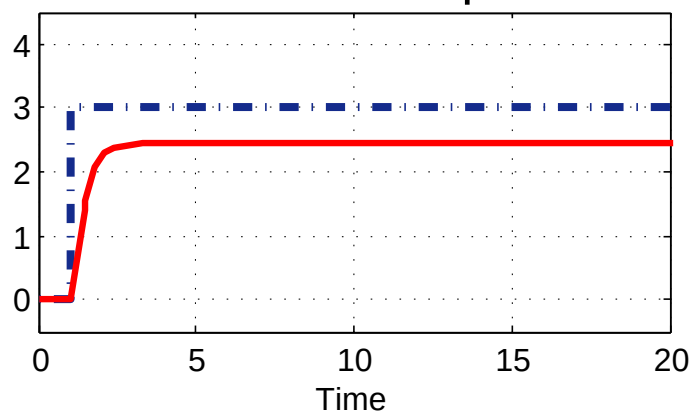
Functional Design - Ziegler Nichols Method

blue: desired speed

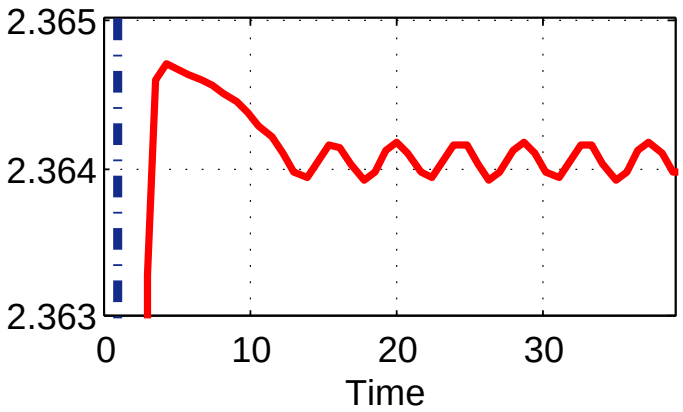


$K_p = 0.1$

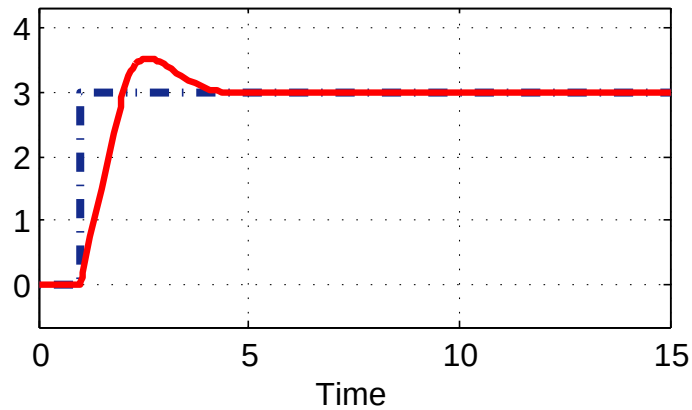
red: current speed



$K_p = 1.0$



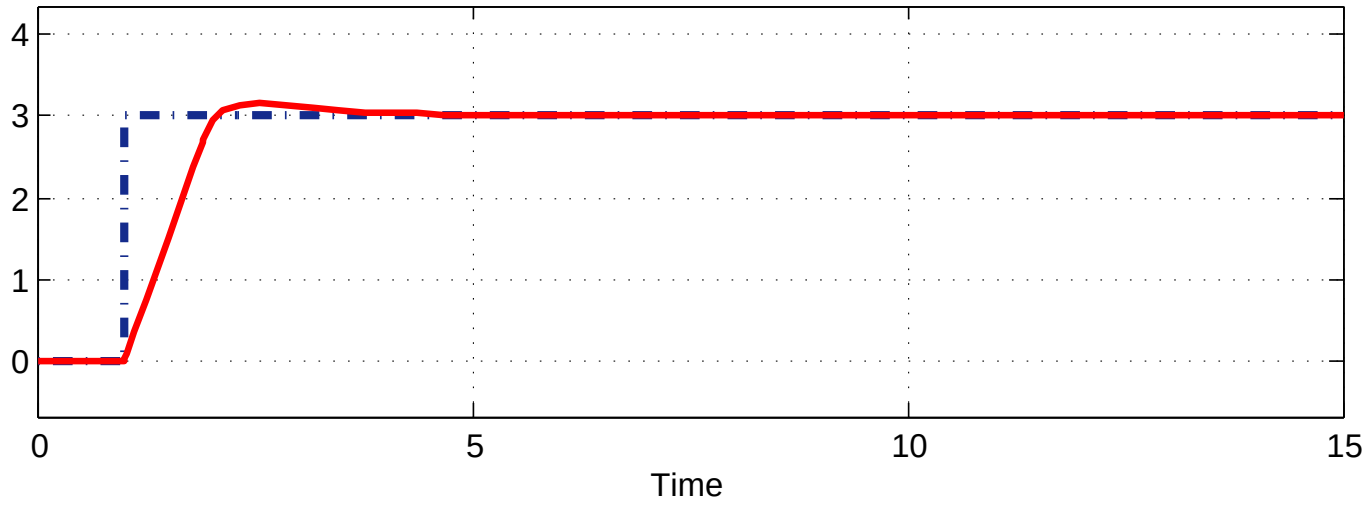
$K_{p_{crit}} = 3.5$, T_{crit} from diagram



Final Ziegler-Nichols response

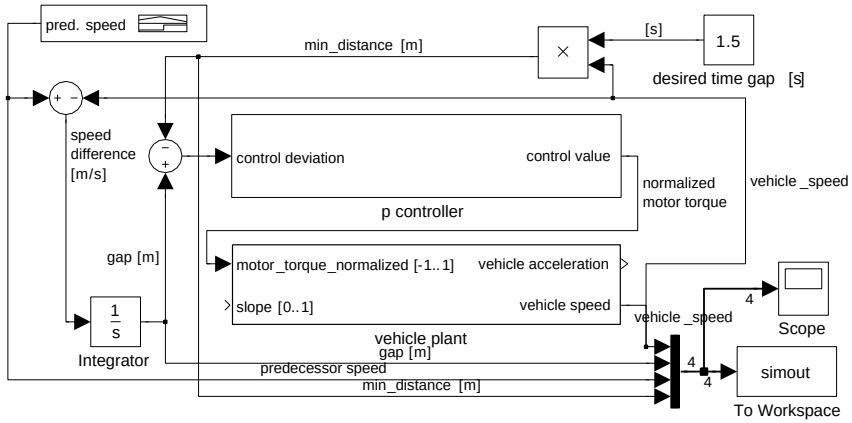
Functional Design - Ziegler Nichols Method

- Overshoot not appropriate (approximate 10%)
- Application-specific parameter tuning:
  - K_p not modified
  - K_i and K_d reduced

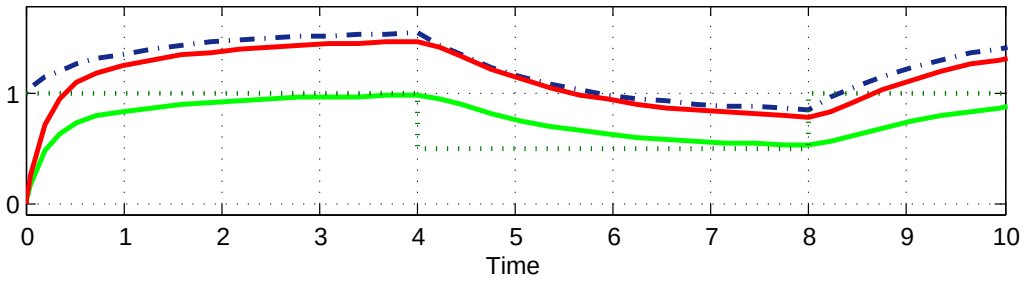


Functional Design - Distance Controller

- Block diagram for simulating the distance controller



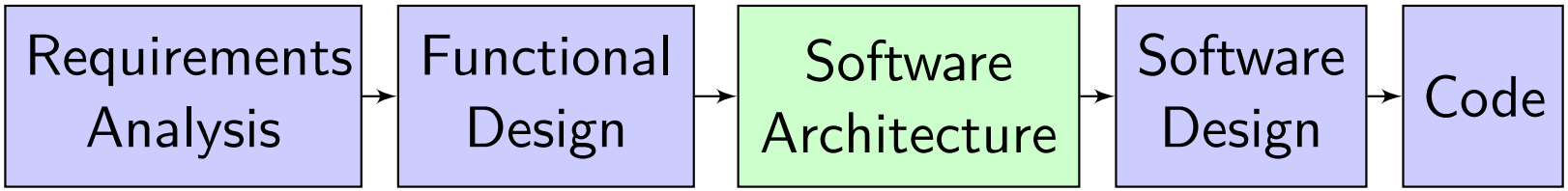
- Resulting controller behavior



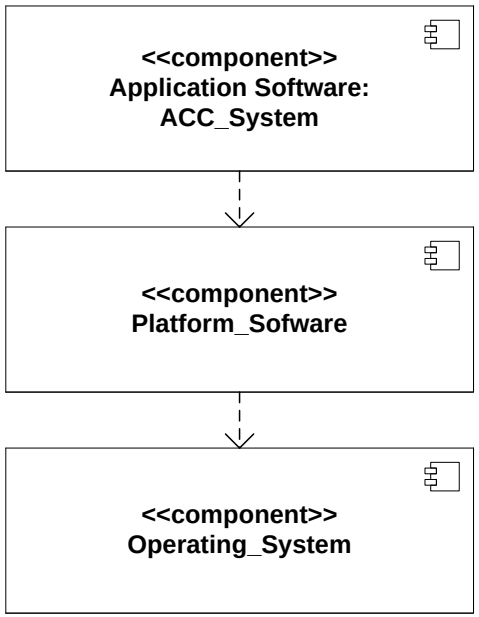
- blue: current distance
- red: desired distance
- dotted green: pred. vehicle speed
- green: vehicle speed

- *Functional Design* finished: Software engineers continue

Software Architecture

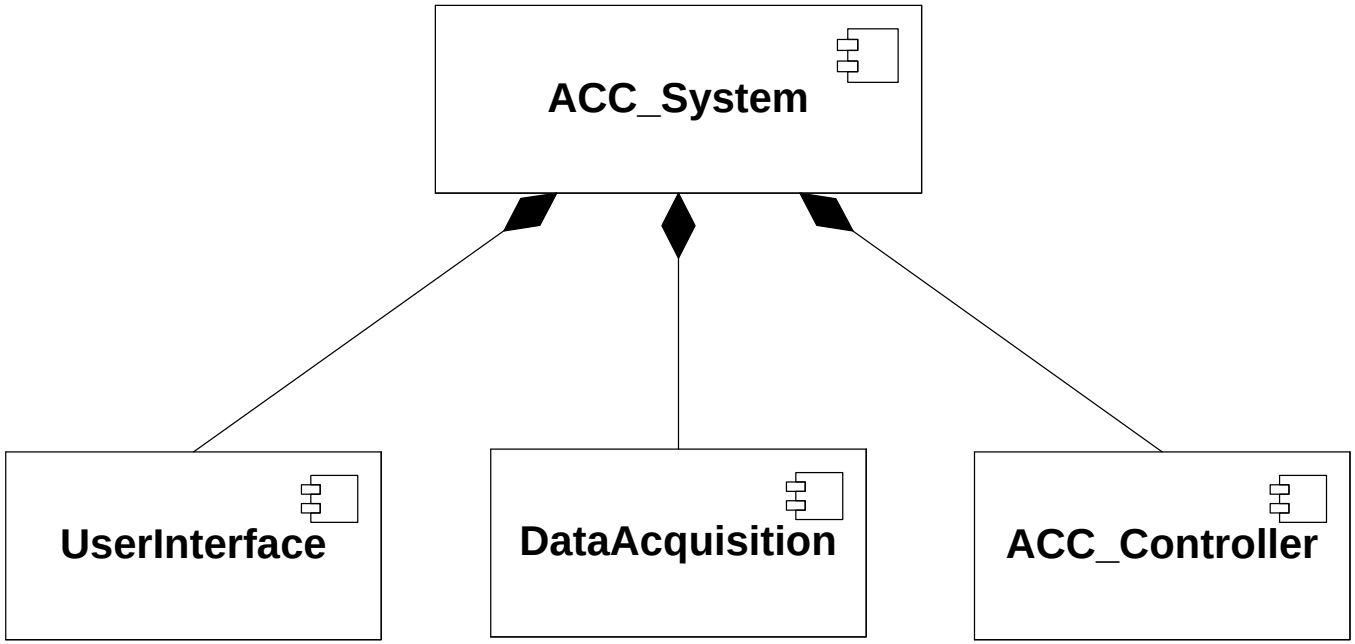


- Structure of the Adaptive Cruise Control System
- *ACC_System* implemented using *Simulink* block diagrams
- *Platform_Software* connects model In/Out Ports to CAN bus
- *Operating_System*: drivers (CAN, SD-Card, UART...)

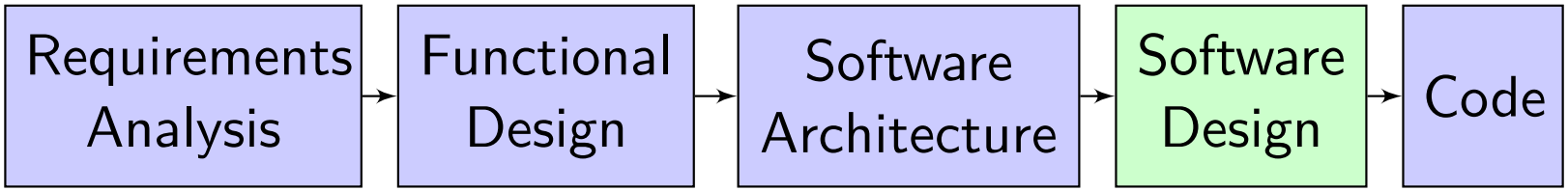


Software Architecture

- Decomposition of the *ACC_System* into sub-components
- *UserInterface*: enable/disable controller, set speed, brake
- *DataAcquisition*: distance to predecessor, wheel speeds, vehicle speed
- *ACC_Controller*: speed/distance controller



Software Design



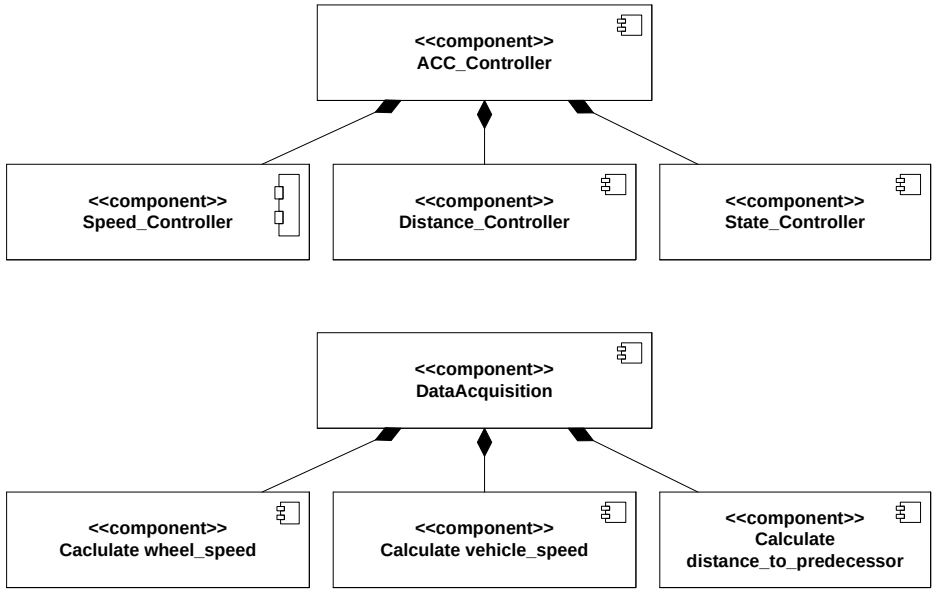
Smooth transition: *Software Architecture* to *Software Design*

- *Structural Refinement*
- *Behavioral Design*
- *Platform-Specific Design*

Software Design: Structural Refinement

Decomposition of the *ACC_System* components:

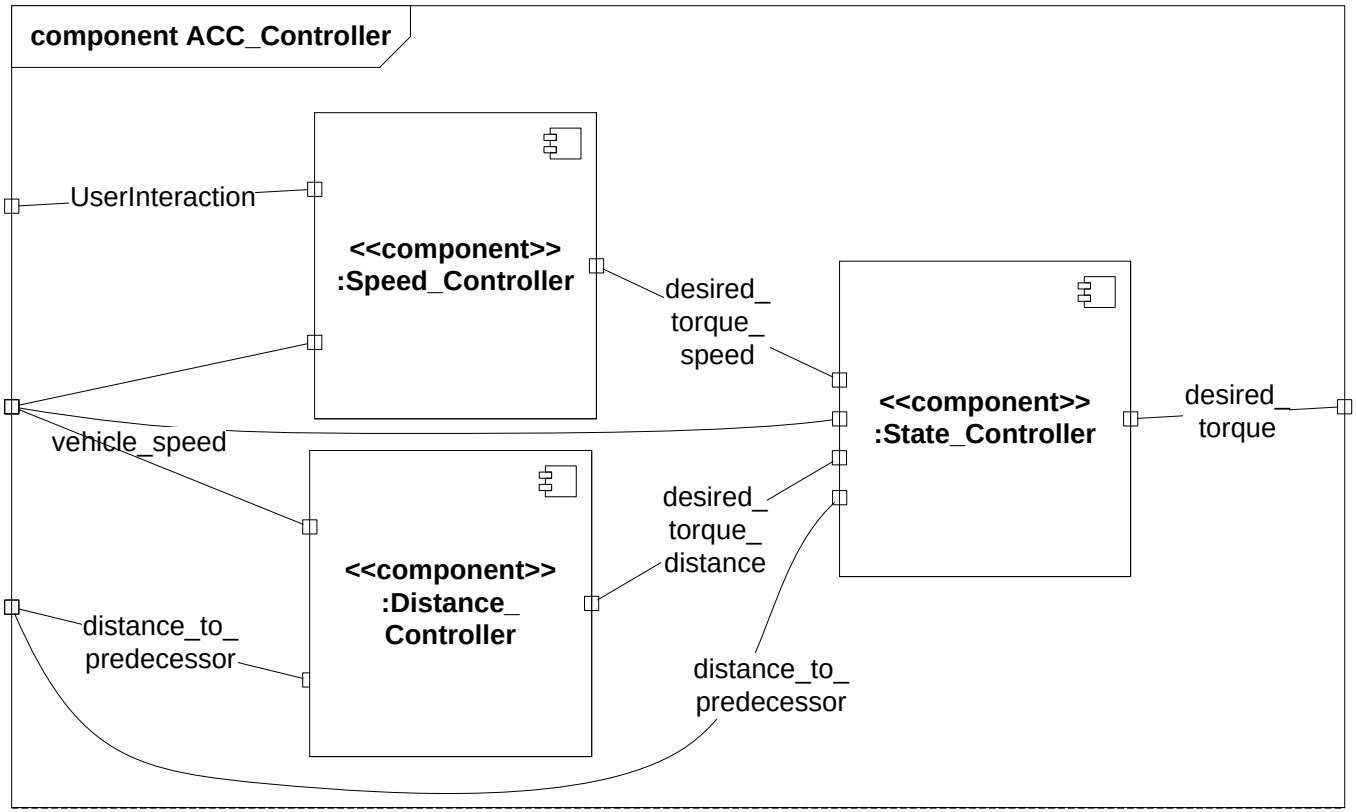
- *ACC_Controller*
  - *Speed_Controller*
  - *Distance_Controller*
  - *State_Controller*
- *DataAcquisition*
  - *Calc Wheel_Speed*
  - *Calc Vehicle_Speed*
  - *Calc Distance*



Software Design: Structural Refinement

Static Interaction:

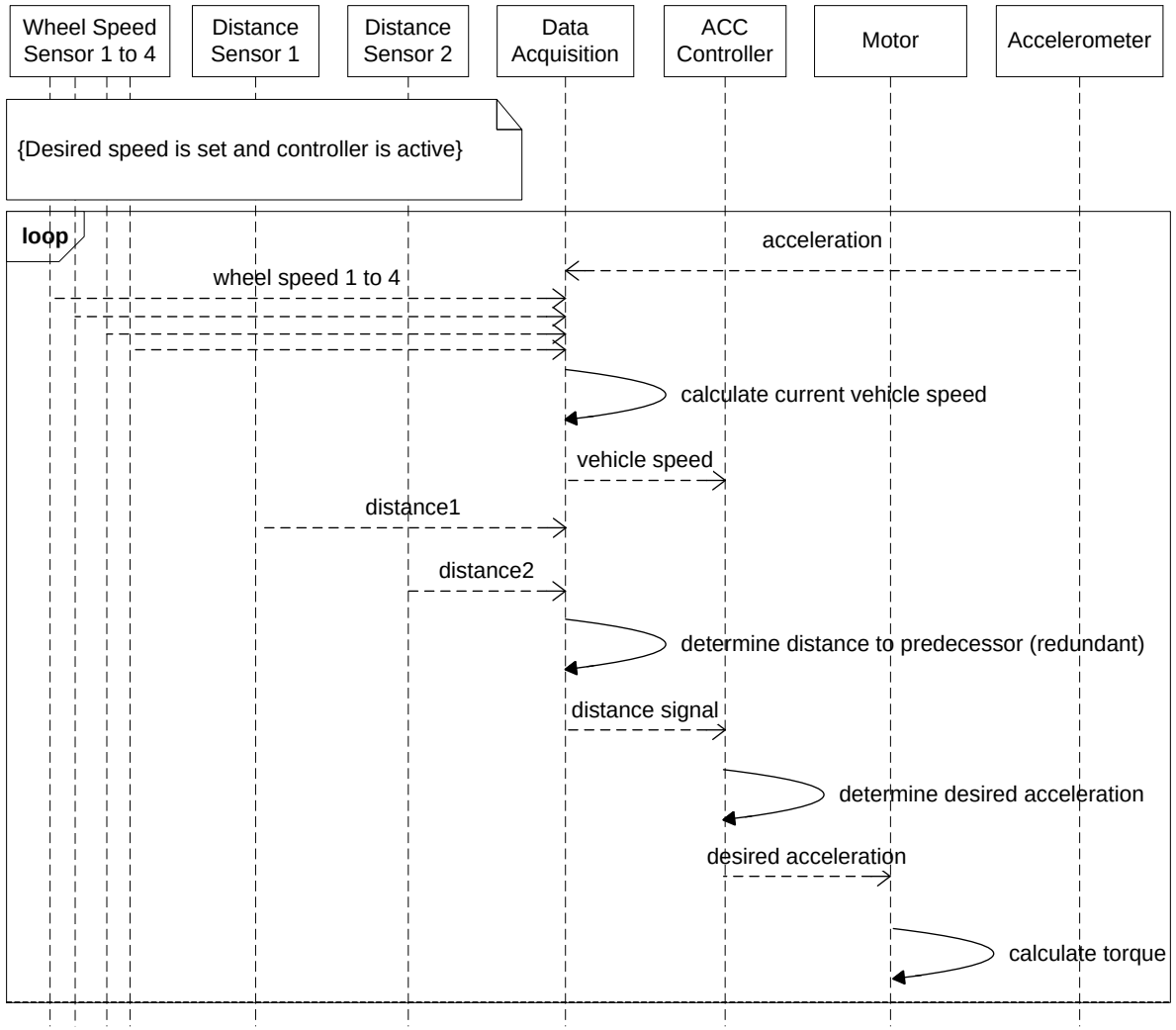
- Refined composite structure diagram of *ACC_Controller*
- Shows data-flow
- And for *DataAcquisition* (not shown here)



Software Design: Structural Refinement

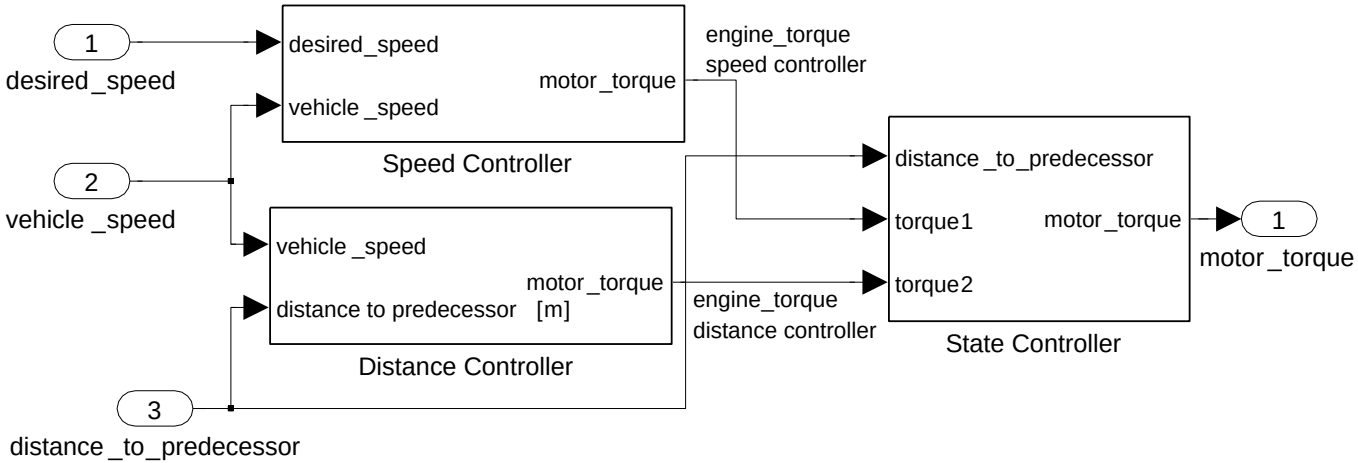
Dynamic Interaction:

- Sequence diagram for scenario *control speed with distance*



Software Design: Behavioral Design

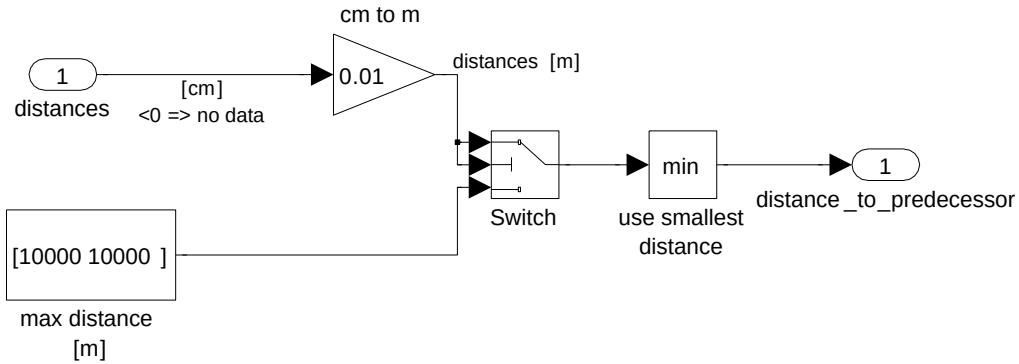
- Behavioral block diagram of *ACC_Controller* like static interaction



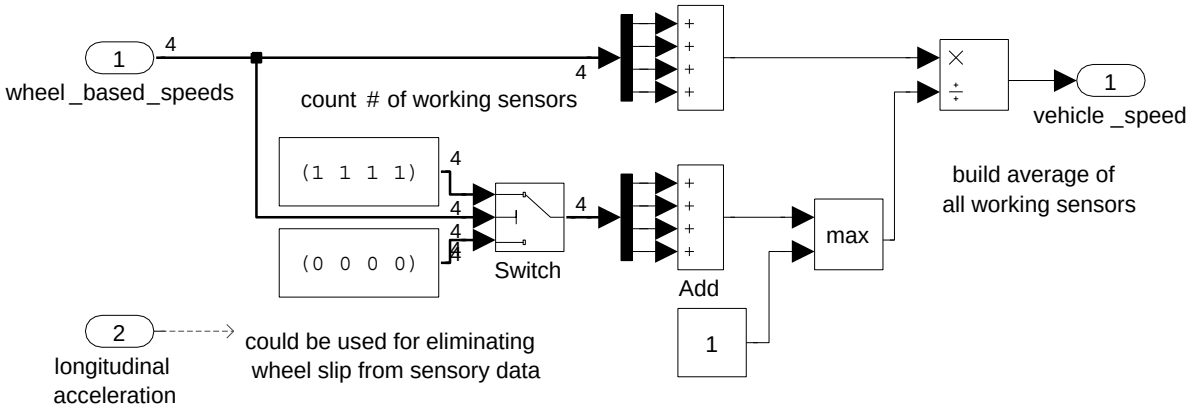
- *Speed-* and *Distance_Controller* from *Functional Design*
- *State_Controller*: Select which controller output to use
  - gets both throttle inputs, selects the appropriate one

Software Design: Behavioral Design

- *Calculate Wheel_Speeds*: From *Functional Design*
- *Calculate Distance_To_Predecessor*:
  - Unit conversion & minimum

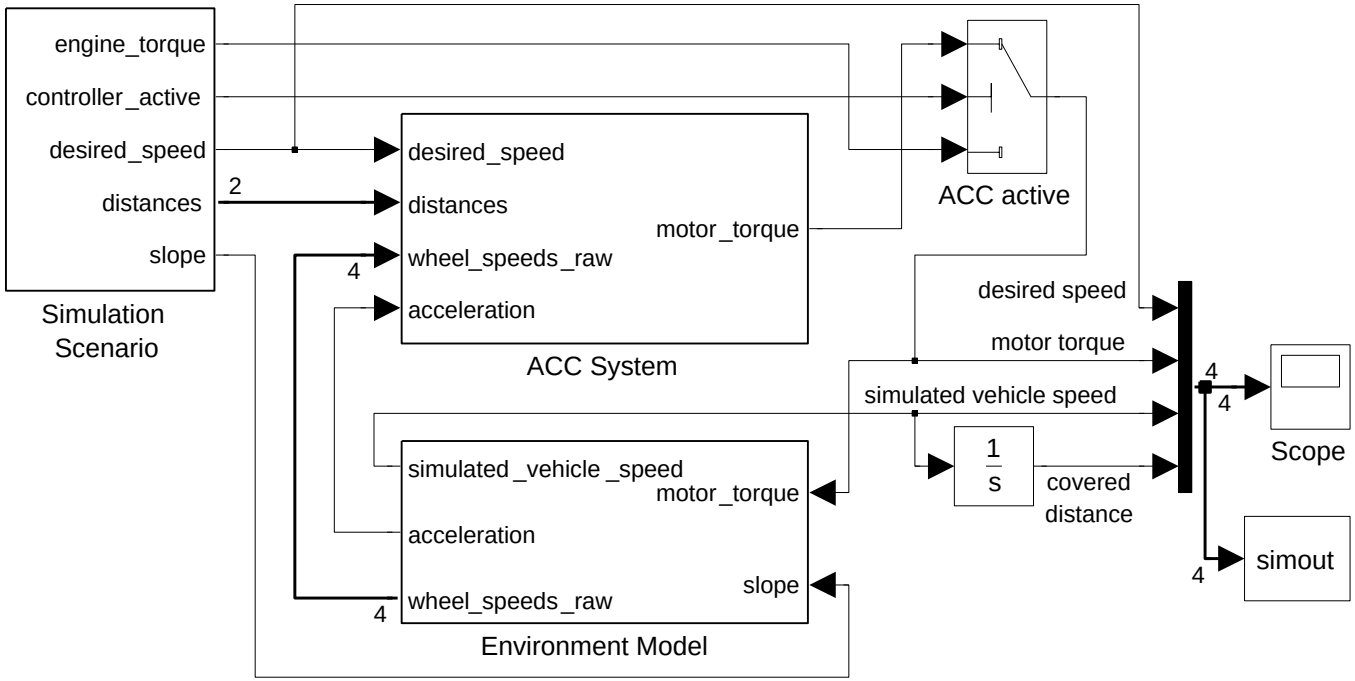


- *Calculate Vehicle_Speed*: Standing wheels are ignored

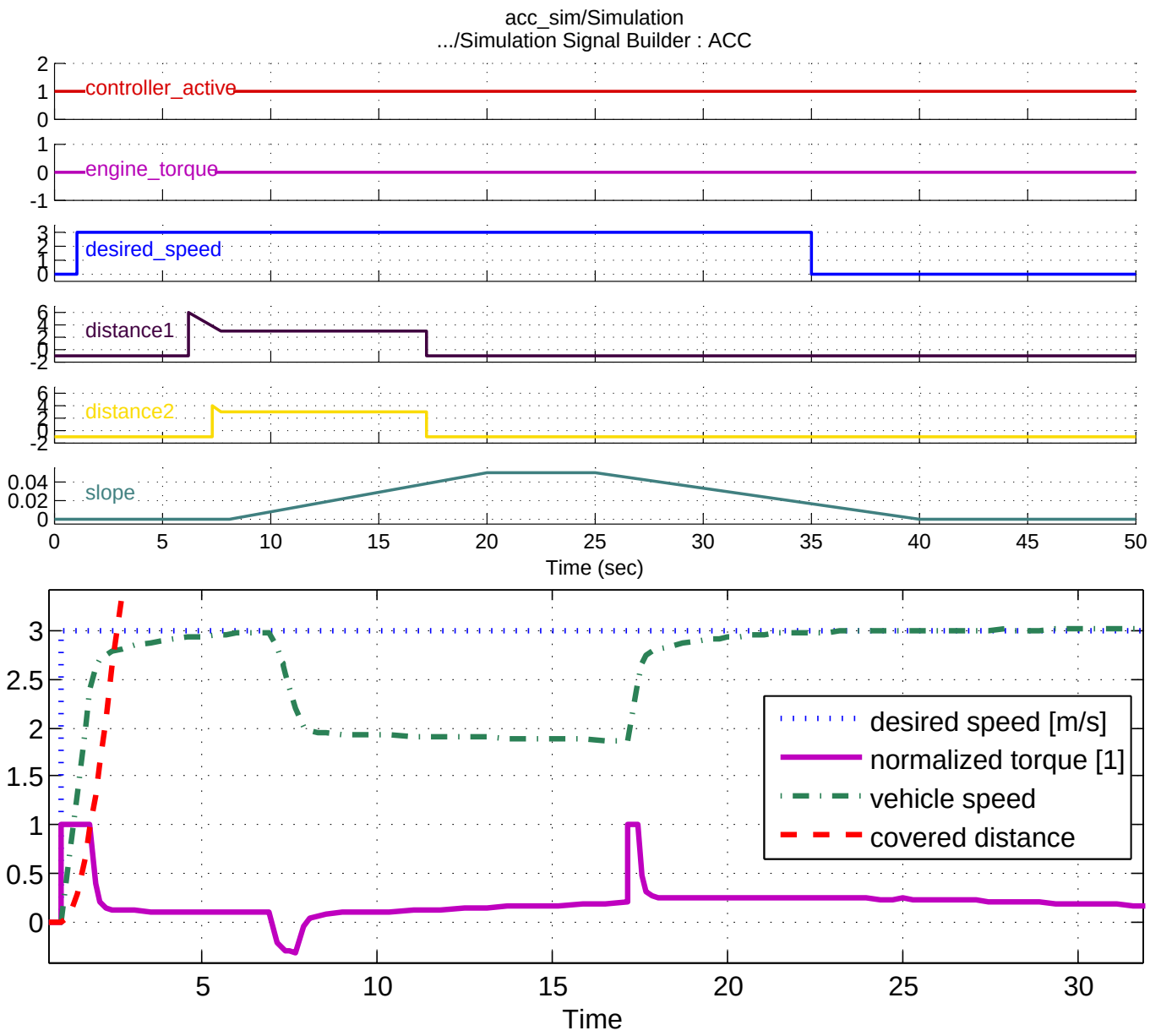


Software Design: Behavioral Design

- All block diagrams completed: Simulation scenarios

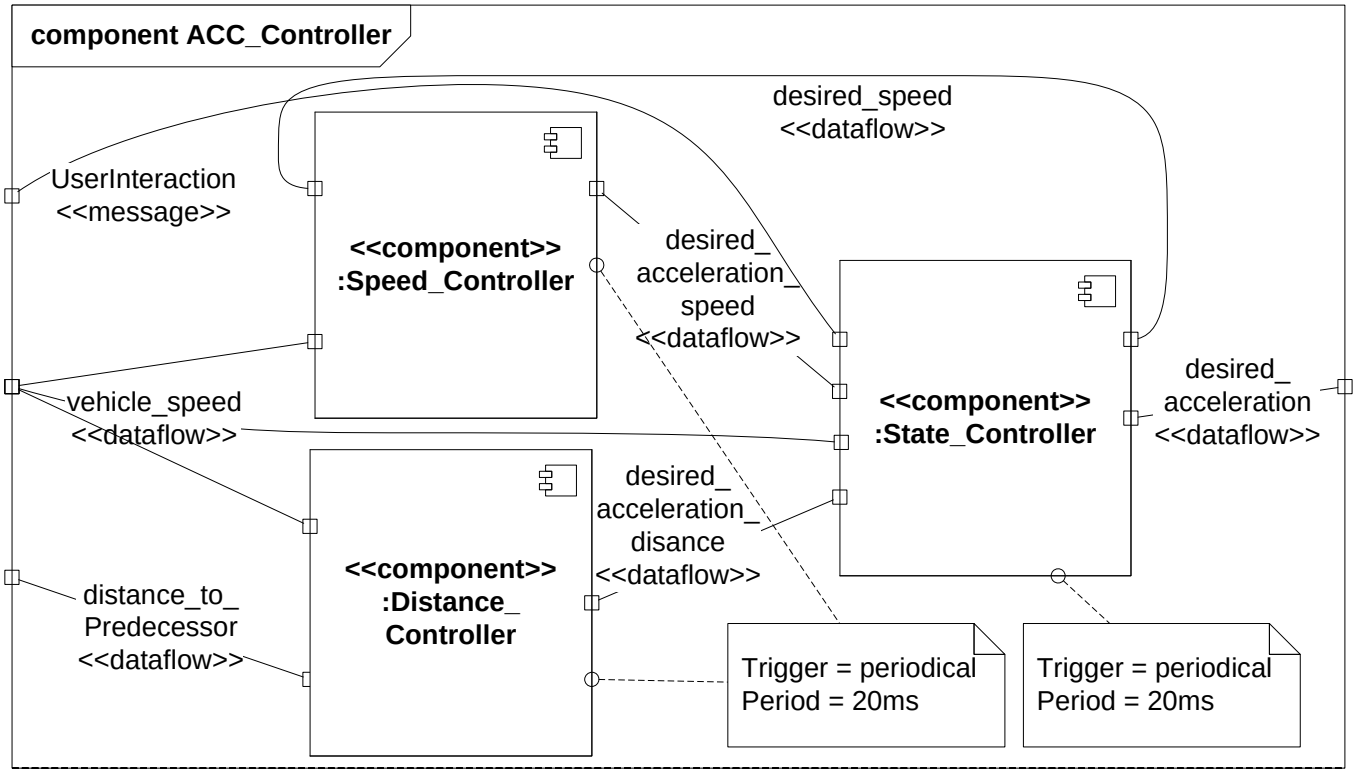


Software Design: Behavioral Design

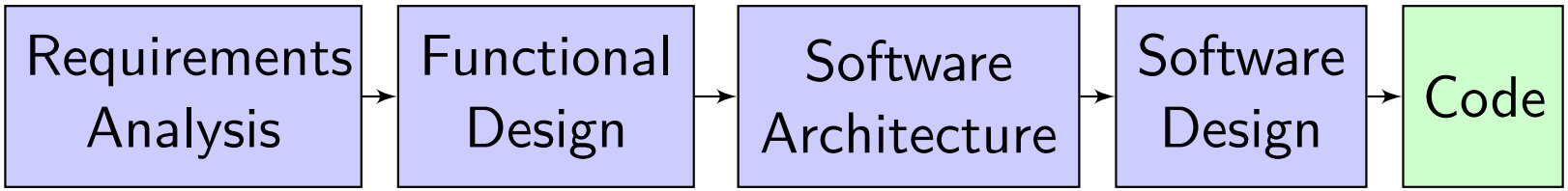


Software Design: Platform-Specific Design

- Platform-specific adaption of the composite structure diagram
- Periodically triggered each 20ms



Code



- *ACC_System* code is automatically generated by *Simulink Realtime Workshop/Embedded Coder* (ANSI C)
- Component model code is extended by platform code and operating system (*SimulinkTarget*)

Listing 1: Embedded Coder example output

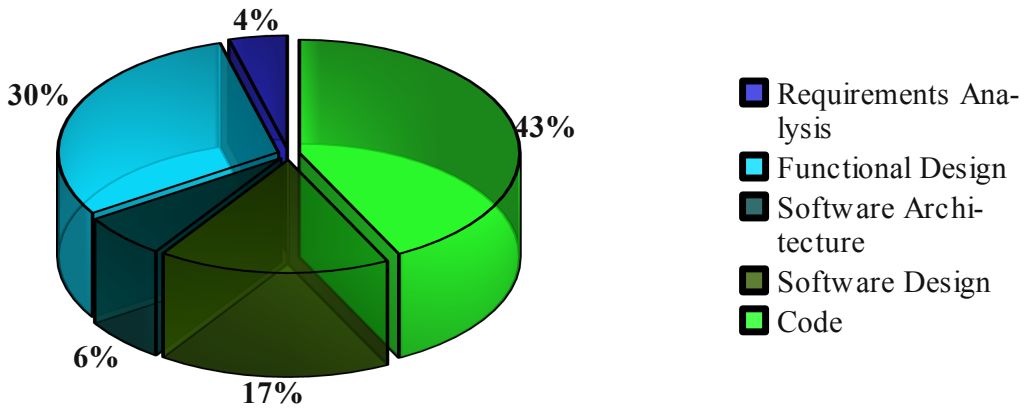
```
/* Model step function */
void acc_step(void) {
  /* local block i/o variables */
  float rtb_Product3;
  float rtb_Sum;
  float rtb_Product2;
  float rtb_Saturation;
  float rtb_throttle01_c;

  /* Sample time: [0.0s, 0.0s] */
  if (rtmIsMajorTimeStep(acc_M)) {
      rate_monotonic_scheduler();
  }
  /* Update absolute time of base rate at minor time step */
  if (rtmIsMinorTimeStep(acc_M)) {
    acc_M->Timing.t[0] = rtsiGetT(&acc_M->solverInfo);
```

Evaluation / Reflection on the Process

- Measured time distribution of the process
- Estimated time distribution for domain experienced personnel
- Estimated time distribution for further developments of vehicle control systems on the same target platform

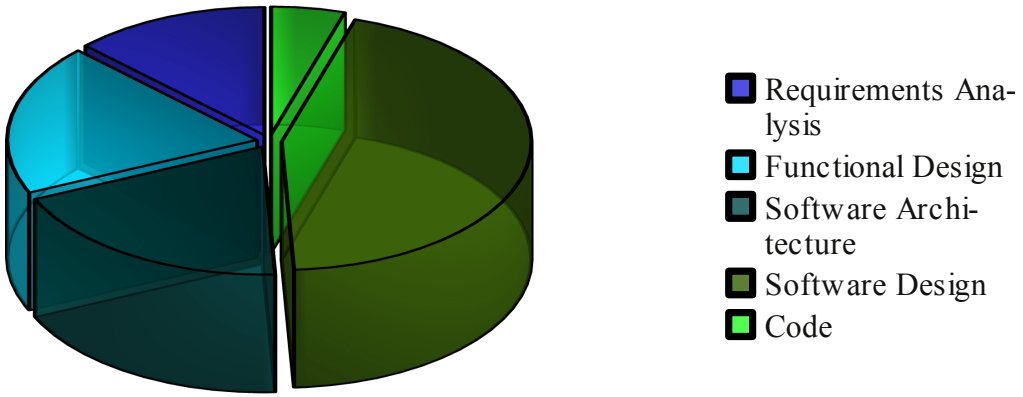
Measured Time Distribution of the Process

- My first model-based development in this domain
- Solving problems included in this chart
- *Requirements Analysis* and *Software Architecture* phases:
  - Short, small system complexity
- *Functional Design* consumed much time:
  - Two feedback loop controllers, vehicle plant, one filter
  - unexperienced in PID parameter tuning
  - learn Matlab details
- *Code* phase
  - Building the toolchain
  - Solving communication problems consumed much time



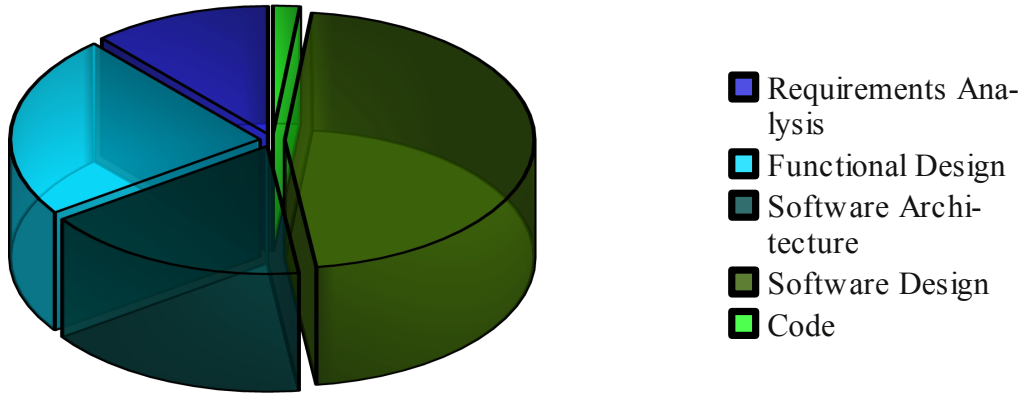
Estimated Time Distribution for Domain Experienced Personnel

- Vast reduction of the *Code* step
  - Activities, that are not directly related to the model-based development are not taken into consideration
  - Platform- and application code combination by a tool
- Reduced *Functional Design* phase
  - Control loop theory (like anti-wind-up, Ziegler-Nichols method)
  - Matlab experience



Estimated Time Distribution for Further Developments of Vehicle Control Systems on the Same Target Platform

- Stable code generation process
  - Completely tool-based
  - Toolchain working
- Proper execution in the target platform
  - Communication issues solved
- *Functional Design* step is reduced:
  - Vehicle plant is fully specified
  - Wheel speed data processing is working



- One of the main advantages is this short *Code* phase

Thank You

- Thank you for your attention