Final Talk LiL4

Group3 - Team speedDreams

Alexander Reisner Alexander Weidinger David Werner

August 01, 2017

Technische Universität München

Table Of Contents

Introduction

Our Task

Autonomous Parking

Our Task

Autonomous Parking

- Start of the parking maneuver by the user
- Autonomously finding a parking spot
- Autonomous execution of the parking maneuver
- Stopping the parking maneuver (in case of failure, ...)
- Parking maneuver itself

Our Task

Autonomous Parking

- Start of the parking maneuver by the user
- Autonomously finding a parking spot
- Autonomous execution of the parking maneuver
- Stopping the parking maneuver (in case of failure, ...)
- Parking maneuver itself

Restrictions

Our Task

Autonomous Parking

- Start of the parking maneuver by the user
- Autonomously finding a parking spot
- Autonomous execution of the parking maneuver
- Stopping the parking maneuver (in case of failure, ...)
- Parking maneuver itself

Restrictions

- Everything starts with less than 1s of latency
- Parking maneuver in real-time...

Our Task

Autonomous Parking

- Start of the parking maneuver by the user
- Autonomously finding a parking spot
- Autonomous execution of the parking maneuver
- Stopping the parking maneuver (in case of failure, ...)
- Parking maneuver itself

Restrictions

- Everything starts with less than 1s of latency
- Parking maneuver in real-time...
- ... on a Pandaboard...

Our Task

Autonomous Parking

- Start of the parking maneuver by the user
- Autonomously finding a parking spot
- Autonomous execution of the parking maneuver
- Stopping the parking maneuver (in case of failure, ...)
- Parking maneuver itself

Restrictions

- Everything starts with less than 1s of latency
- Parking maneuver in real-time...
- ... on a Pandaboard... with Fiasco.OC / Genode OS

Our Task

Autonomous Parking

- Start of the parking maneuver by the user
- Autonomously finding a parking spot
- Autonomous execution of the parking maneuver
- Stopping the parking maneuver (in case of failure, ...)
- Parking maneuver itself

Restrictions

- Everything starts with less than 1s of latency
- Parking maneuver in real-time...
- ... on a Pandaboard... with Fiasco.OC / Genode OS
- Parking without crashing
- Parking in less than 30s total

Project Overview

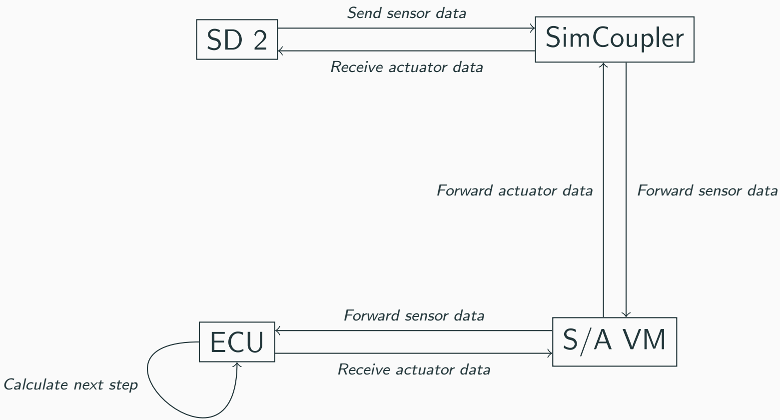

Fig. 1: Overview of components

Task allocation

- Alexander Weidinger
  - Extend SpeedDreams 2 (SD2) by a virtual proximity sensor
  - Build data exchange between SD2 and Simulation Coupler (SimCoupler)
  - Create data exchange between SimCoupler and QEMU S/A VM
- Alexander Reisner
  - Introduce the QEMU S/A VM
  - Exchange data between SimCoupler and QEMU S/A VM
  - Implement mosquitto client to forward data to the ECUs
- David Werner
  - Implement an autonomous parking algorithm
  - Implement mosquitto client to forward calculated control data

Alexander Weidinger

Related Work

- Research Phase: Find implementations of such sensor for TORCS / SD2

Related Work

- Research Phase: Find implementations of such sensor for TORCS / SD2
- Adapt the found sensor implementation for usage in SD2 and test it

Related Work

- Research Phase: Find implementations of such sensor for TORCS / SD2
- Adapt the found sensor implementation for usage in SD2 and test it
- Resignation: The sensor isn't appropriate for our use case

Related Work

- Research Phase: Find implementations of such sensor for TORCS / SD2
- Adapt the found sensor implementation for usage in SD2 and test it
- Resignation: The sensor isn't appropriate for our use case
- **Write our own proximity sensor**

Implementation And Comparison

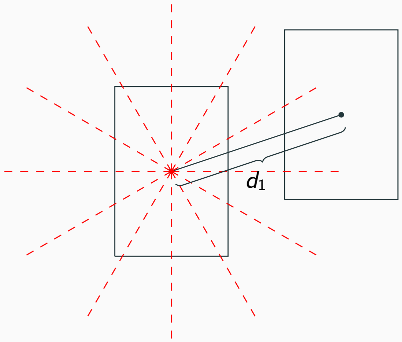

Fig. 2: Proximity sensor implemented by the Simulated Car Racing Championship 2015

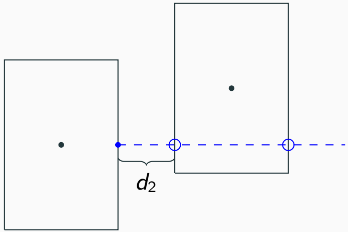

Fig. 3: (Laser) proximity sensors implemented by us

Data Exchange SD2 $\longleftrightarrow$ QEMU S/A VM

Fig. 4: Message exchange between SD2 and QEMU S/A VM

Protocol

- Protocol: Google Protocol Buffers
- Messages: State, Control
- Simple TCP connection
- Simple Protocol: 4 byte message header (length of message) + message itself

```
syntax = "proto3";
package protobuf;

import "sensor.proto";
import "wheel.proto";
import "specification.proto";

message State {
  repeated Sensor sensor = 1;
  repeated Wheel wheel = 2;
  Specification specification = 3;
  float steer = 4;
  float brakeCmd = 5;
  float accelCmd = 6;
}
```

Fig. 5: state.proto

Problems & Limitations

- Protobuf doesn't allow multiple modules to be linked against protobuf

Problems & Limitations

- Protobuf doesn't allow multiple modules to be linked against protobuf

$\rightarrow$ E.g. share one protobuf "module" between all modules

Problems & Limitations

- Protobuf doesn't allow multiple modules to be linked against protobuf

$\rightarrow$ E.g. share one protobuf "module" between all modules

- SimCoupler is currently not used

Problems & Limitations

- Protobuf doesn't allow multiple modules to be linked against protobuf

$\rightarrow$ E.g. share one protobuf “module” between all modules

- SimCoupler is currently not used

$\rightarrow$ should be easily expendable

Problems & Limitations

- Protobuf doesn't allow multiple modules to be linked against protobuf

$\rightarrow$ E.g. share one protobuf "module" between all modules

- SimCoupler is currently not used

$\rightarrow$ should be easily expendable

- Proximity sensor tends to precision errors if obstacle is too close

Problems & Limitations

- Protobuf doesn't allow multiple modules to be linked against protobuf

→ E.g. share one protobuf "module" between all modules

- SimCoupler is currently not used

→ should be easily expendable

- Proximity sensor tends to precision errors if obstacle is too close

→ more or less only after directly crashing into the obstacle :-)

Alex’s ’Would Have Been Nice To Know Before’ Corner

- OSs make “improvements”
- **But:** Tend to interfere with our solutions
- E.g. Nagle’s algorithm (bandwidth efficiency vs. latency)
- **Solution:** `TCP_NODELAY` to disable it

Alexander Reisner

Mosquitto

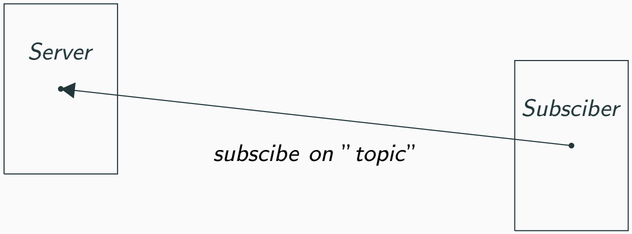

Fig. 6: Subscriber

Mosquitto

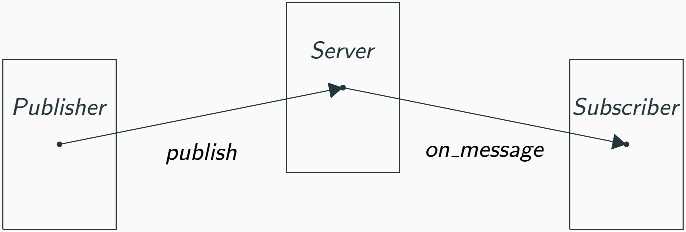

Fig. 7: Publisher

Mosquitto

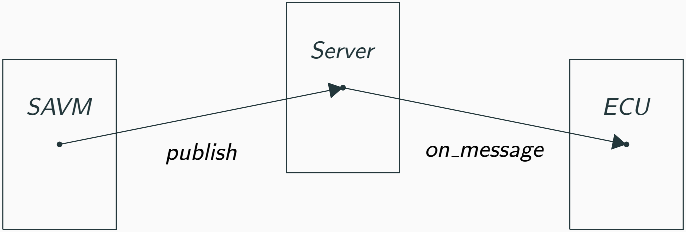

Fig. 8: SAVM/ECU

Mosquitto

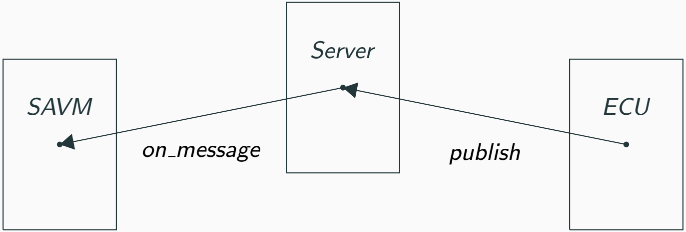

Fig. 9: ECU/SAVM

Full Szenario

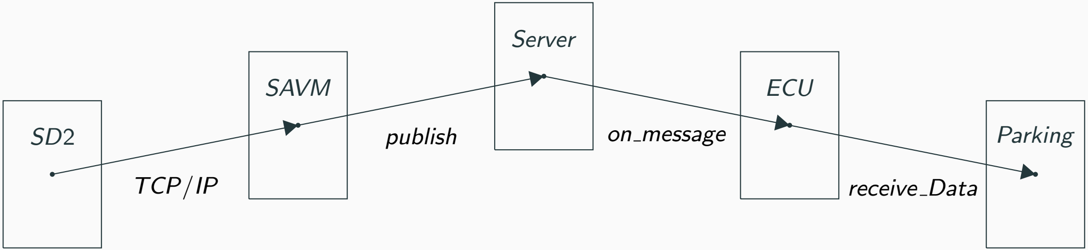

Fig. 10: Full Szenario Forward

Full Szenario

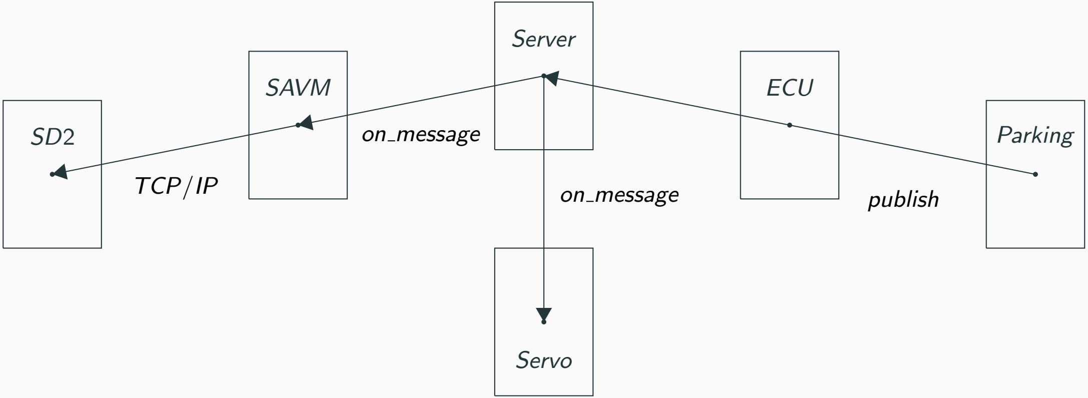

Fig. 11: Full Szenario Backward

David Werner

Problem

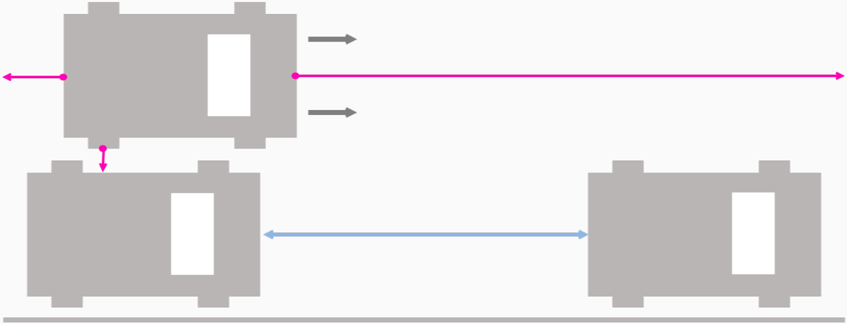

Fig. 12: Car needs to autonomously pass by the parking lot, detect it and perform a parallel parking maneuver

Basics

- calculation of actuator data

Basics

- calculation of actuator data
    - velocity v

Basics

- calculation of actuator data
  - velocity v
  - steering angle ϕ

Basics

- calculation of actuator data
  - velocity v
  - steering angle ϕ
- no computation of an exact path

Basics

- calculation of actuator data
    - velocity v
    - steering angle ϕ
- no computation of an exact path
- actuator data is determined by the evaluation of our 3-phase algorithm

Phase 1 - Searching phase (1)

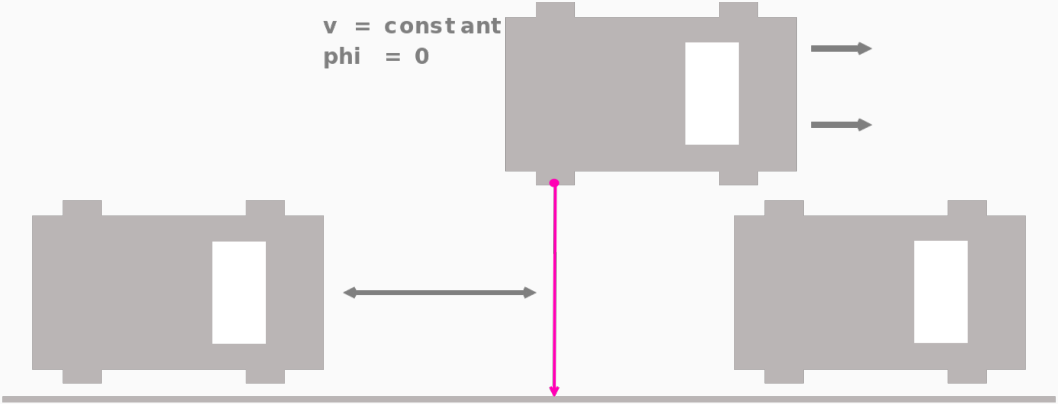

Fig. 13: Car passes the potential parking lot and calculates its size

Phase 1 - Searching phase (2)

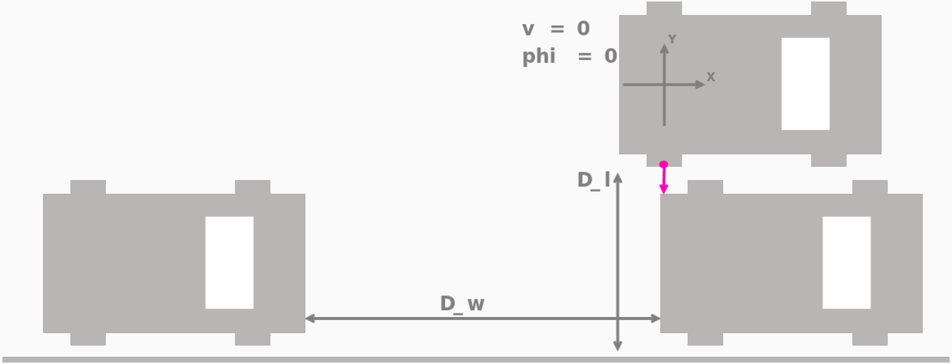

Fig. 14: Algorithm creates environment and position information

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:
    - $\dot{x} = v * cos(\phi) * cos(\theta)$

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:
    - $\dot{x} = v * cos(\phi) * cos(\theta)$
    - $\dot{y} = v * cos(\phi) * sin(\theta)$

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:
  - $\dot{x} = v * cos(\phi) * cos(\theta)$
  - $\dot{y} = v * cos(\phi) * sin(\theta)$
  - $\dot{\theta} = \frac{v}{L} * sin(\phi)$

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:
  - $\dot{x} = v * cos(\phi) * cos(\theta)$
  - $\dot{y} = v * cos(\phi) * sin(\theta)$
  - $\dot{\theta} = \frac{v}{L} * sin(\phi)$
- time dependant formulas for v and ϕ needed

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:
  - $\dot{x} = v * cos(\phi) * cos(\theta)$
  - $\dot{y} = v * cos(\phi) * sin(\theta)$
  - $\dot{\theta} = \frac{v}{L} * sin(\phi)$
- time dependant formulas for v and ϕ needed
  - $\phi(t)$ - steering angle (based on max ϕ)

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:
    - $\dot{x} = v * cos(\phi) * cos(\theta)$
    - $\dot{y} = v * cos(\phi) * sin(\theta)$
    - $\dot{\theta} = \frac{v}{L} * sin(\phi)$
- time dependant formulas for v and ϕ needed
    - $\phi(t)$ - steering angle (based on max ϕ)
    - $v(t)$ - velocity (based on max v)

Phase 2 - Calculation phase (1)

- equations of vehicle's motion:
  - $\dot{x} = v * cos(\phi) * cos(\theta)$
  - $\dot{y} = v * cos(\phi) * sin(\theta)$
  - $\dot{\theta} = \frac{v}{L} * sin(\phi)$
- time dependant formulas for v and ϕ needed
  - $\phi(t)$ - steering angle (based on max ϕ)
  - $v(t)$ - velocity (based on max v)
- time for whole maneuver is estimated and optimized

Phase 2 - Calculation phase (2)

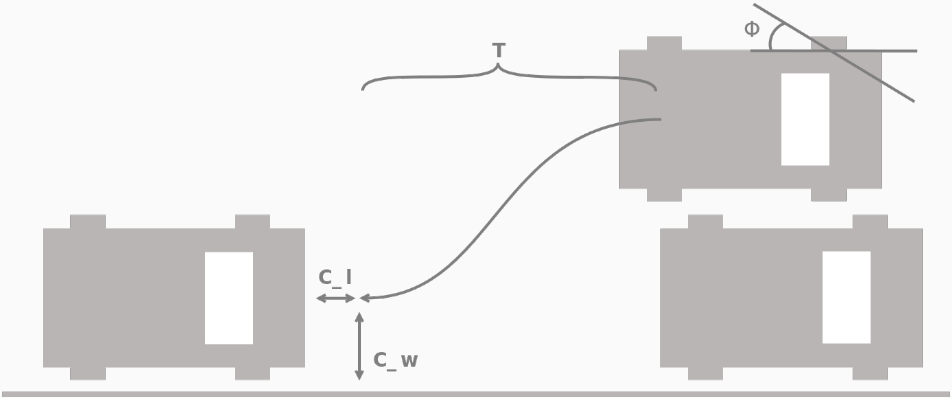

Fig. 15: Algorithm simulates parking maneuver to calculate duration and steering angle

Phase 3 - Parking phase

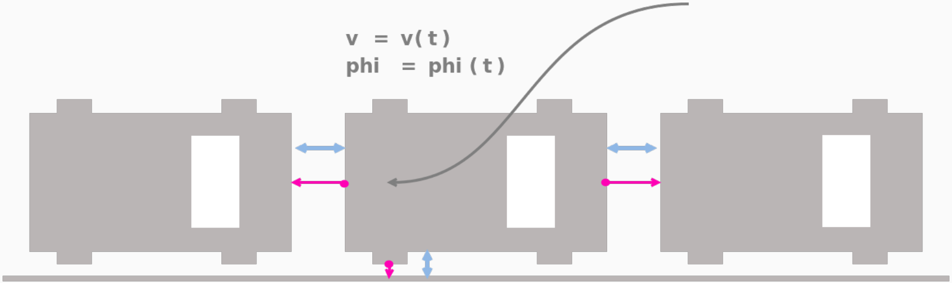

Fig. 16: Algorithm steers and accelerates the car according to calculation until parking position is reached

Problems and Limitations

- calculation of parking duration is based on magic number

Problems and Limitations

- calculation of parking duration is based on magic number
- longitudinal and lateral distance conditions seem to work not properly

Problems and Limitations

- calculation of parking duration is based on magic number
- longitudinal and lateral distance conditions seem to work not properly
- safety distance needs to be higher than necessary

Summary

Did we reach our goals?

Autonomous Parking

- Start of the parking maneuver by the user ✓
- Autonomously finding a parking spot ✓
- Autonomous execution of the parking maneuver ✓
- Stopping the parking maneuver (in case of failure, ...) ✓
- Parking maneuver itself ✓

Restrictions

- Everything starts with less than 1s of latency ✓
- Parking maneuver in real-time... ✓
- ... on a Pandaboard... with Fiasco.OC / Genode OS ✓
- Parking without crashing ✓
- Parking in less than 30s total ✓