University of Stuttgart
Institute of Parallel and Distributed Systems

Distributed Systems

Fachpraktikum / Lab-Course

Software-Defined and Time-Sensitive Networking

Assignment: CAN

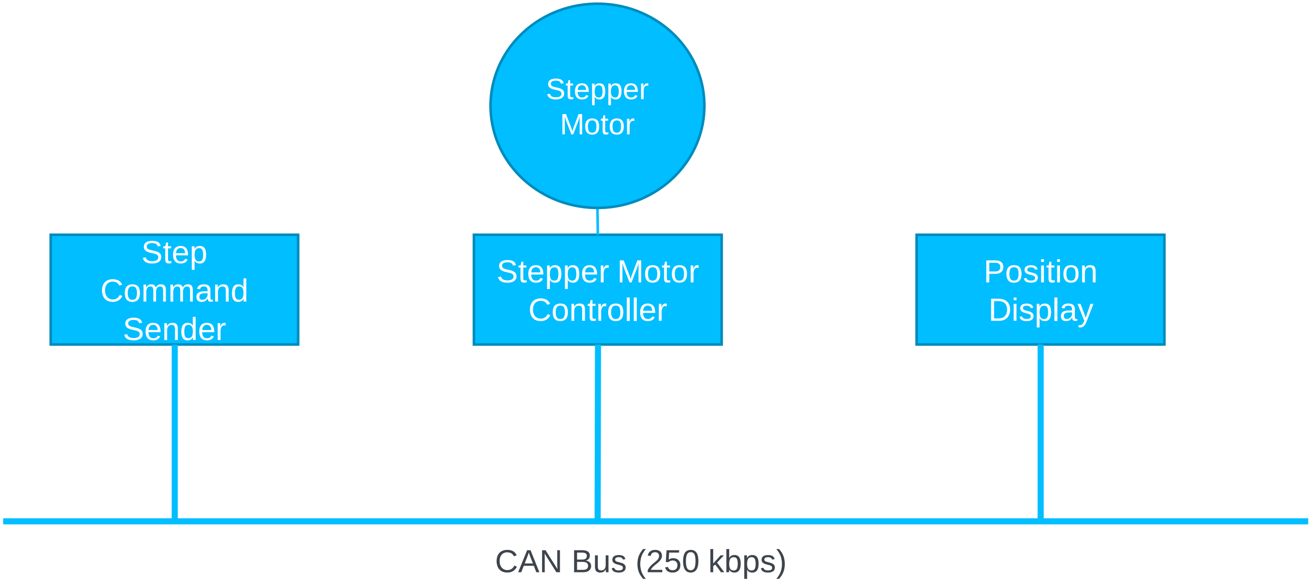
Frank Dürr

Summer Term 2023

IPVS

CAN System to be implemented

System to control stepper motor over CAN Bus



Stepper Motor

Step Command Sender

Stepper Motor Controller

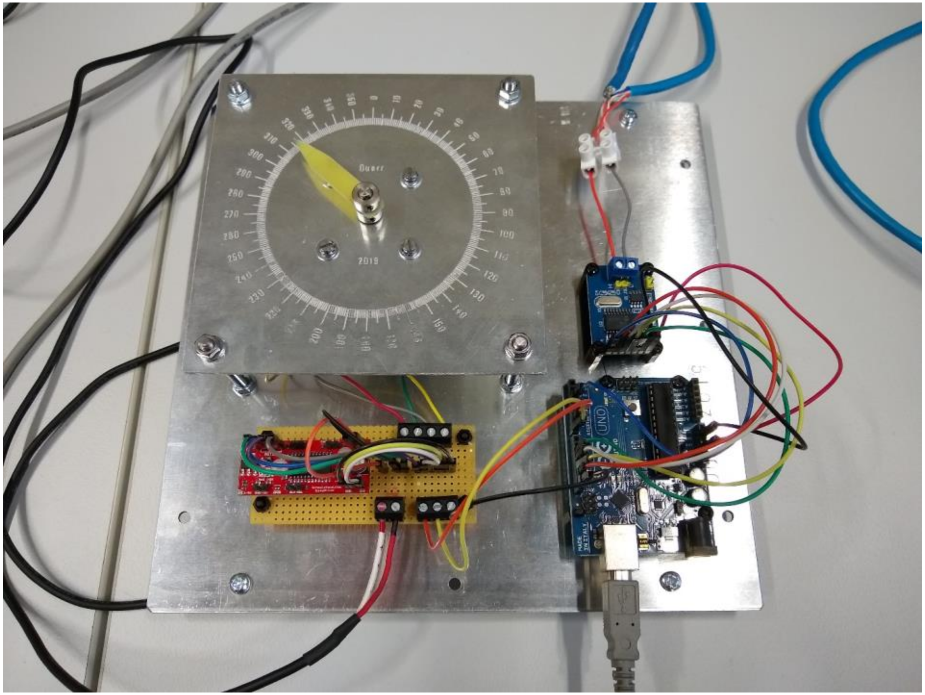
Position Display

CAN Bus (250 kbps)

CAN System to be implemented

Stepper Motor Controller

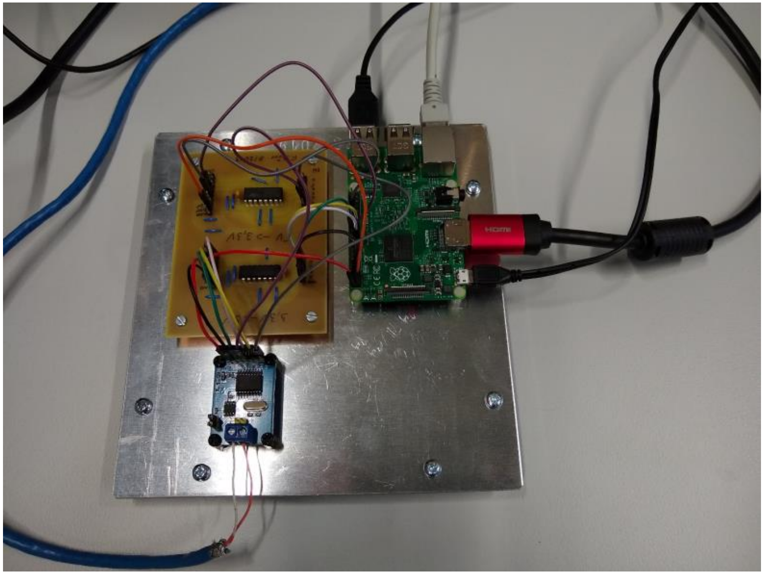
The Stepper Motor Controller is already implemented

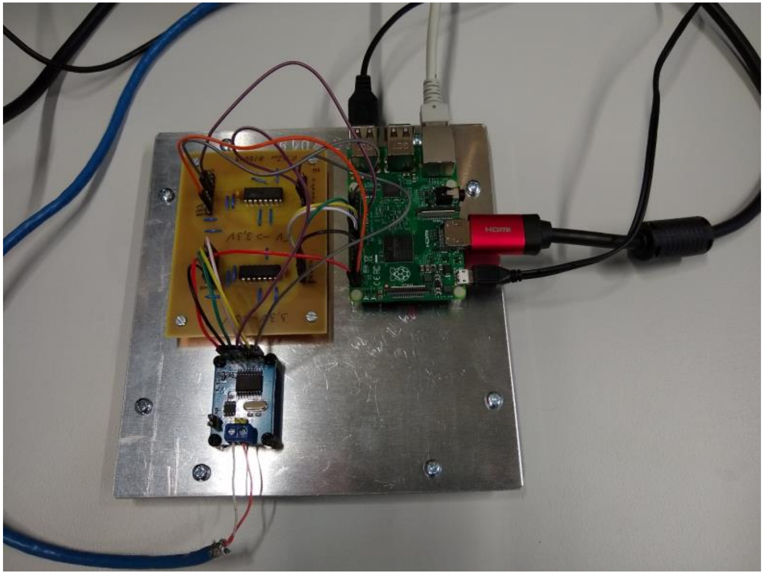


CAN System to be implemented

Step Command Sender & Position Display

You need to implement the Step Command Sender and Position Display





CAN System to be implemented

Step Command Sender

- Cyclic sender
- Sends step commands periodically to Stepper Motor Controller over CAN bus
  - Cycle time (in nano-seconds) directly influences rotation speed
  - One step command = one step
  - Step size: 1.8° / 8
  - Cycle time can be configured via command line parameter
- Payload of step command: one byte
  - 0x01 = one step clock-wise
  - 0x02 = one step counter-clock-wise

CAN System to be implemented

Position Display

- Cyclic sender (and receiver)
- Sends requests for motor position periodically to Stepper Motor Controller via CAN bus
- Receives response with motor position
  - Position in steps
  - unsigned 16 bit integer value

CAN System to be implemented

Priorities of CAN Messages

- Step commands must have higher priority to guarantee smooth rotation speed
- How can this be ensured?

Hints (1)

- We use 11 bit CAN IDs
  - Step command: **000 0000 0001 = 0x01**
  - Position request / response: **000 0000 0010 = 0x02**
- Positions are sent in Big Endian byte order
  - The RPis implement a Little Endian architecture
- If the cycle time of the step command sender is too short, you will get error messages at the sender
  - Try with 1 ms cycle time first

Hints (2)

- Both programs support:
  - Specifying CAN interface (e.g., `-i can0`)
  - CPU pinning (e.g., `-c 1` for pinning process to CPU 1)
  - Setting process priorities (e.g., `-p 30` for priority 30)
    - [0..99] ; higher values mean higher priority
    - Ordinary user processes run at priority 0
    - Don’t set priority too high or you can block important kernel processes
  - Need to run programs as sudo/root
  - These commands should work:

```
$ sudo ./speedcmd-sender  -t 1000000 -c 1 -p 30 -i can0
$ sudo ./positionreq-sender -t 100000000 -c 1 -p 30 -i can0
```

Task (1)

- Add your implementation to the given code skeletons:
  - Folder: `assignment-can/src-rpi/`
  - Step Command Sender
    - file `speedcmd-sender.c`
  - Position Display
    - file `positionreq-sender.c`
- Check TODO comments in source code and add your code there

Task (2)

- Compile with cmake:

```
$ cd assignment-can/src-rpi/
$ mkdir build
$ cd build
$ cmake ..
$ make
```

- After changing source code files, `make` command is sufficient