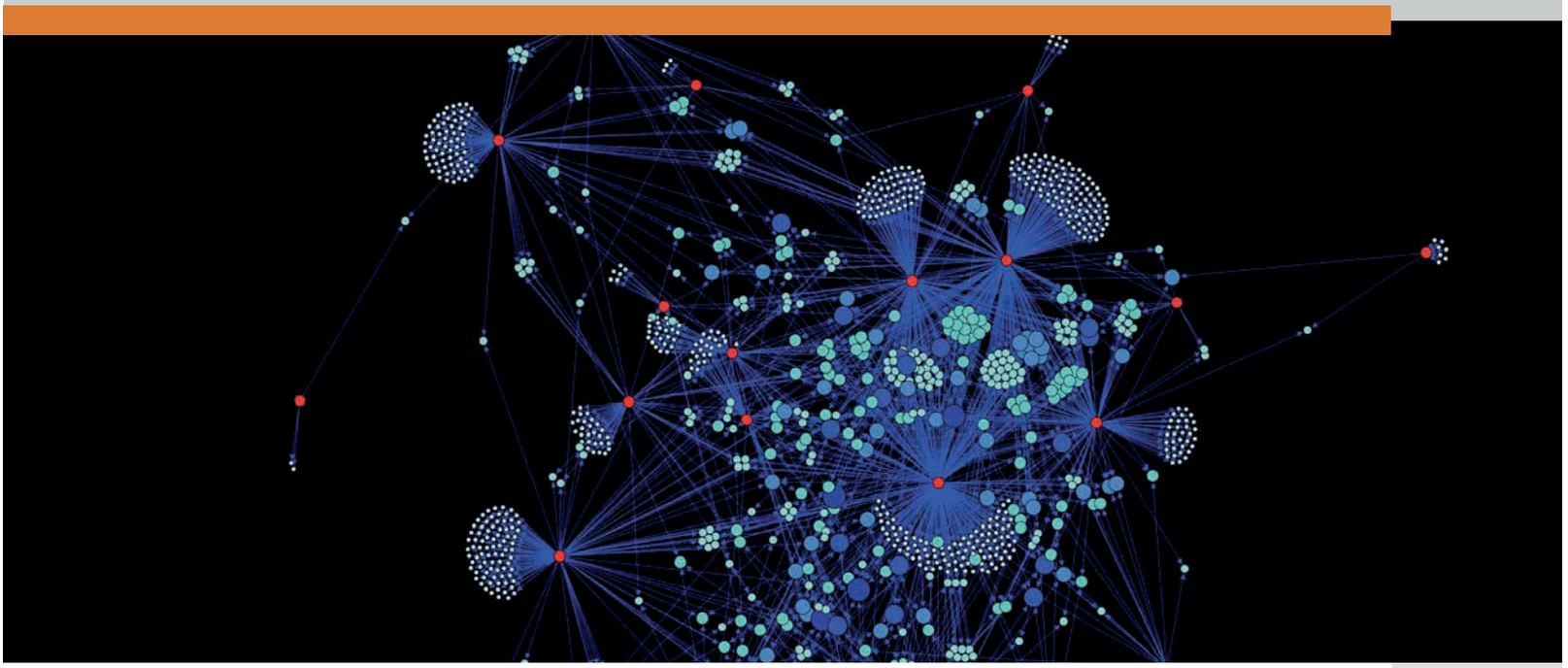
German Research Center for Artificial Intelligence

Document D-15-02



Proceedings of the RIC Project Day

Workgroups 'Electronic Design' and 'Mechatronic Design'

Frank Kirchner (Editor)
Peter Kampmann, Marc Simnofske (Associate Editors)

Document D-15-02
German Research Center for Artificial Intelligence (DFKI) GmbH

Bibliographic information published by the German National Library

The German National Library lists this publication in the German National Biography; detailed bibliographic data are available in the internet at http://dnb.ddb.de

Issue D-15-02 (2015)
ISSN 0946-0098

German Research Center for Artificial Intelligence
Deutsches Forschungszentrum für Künstliche Intelligenz
DFKI GmbH

Founded in 1988, DFKI today is one of the largest nonprofit contract research institutes in the field of innovative software technology based on Artificial Intelligence (AI) methods. DFKI is focusing on the complete cycle of innovation – from world-class basic research and technology development through leading-edge demonstrators and prototypes to product functions and commercialization.

Based in Kaiserslautern, Saarbrücken and Bremen, the German Research Center for Artificial Intelligence ranks among the important 'Centers of Excellence' worldwide. An important element of DFKI's mission is to move innovations as quickly as possible from the lab into the marketplace. Only by maintaining research projects at the forefront of science DFKI has the strength to meet its technology transfer goals.

The key directors of DFKI are Prof. Wolfgang Wahlster (CEO) and Dr. Walter Olthoff (CFO). DFKI's research departments are directed by internationally recognized research scientists:

- Knowledge Management (Prof. A. Dengel)
- Cyber-Physical Systems (Prof. R. Drechsler)
- Robotics Innovation Center (Prof. F. Kirchner)
- Innovative Retail Laboratory (Prof. A. Krüger)
- Institute for Information Systems (Prof. P. Loos)
- Embedded Intelligence (Prof. P. Lukowicz)
- Agents and Simulated Reality (Prof. P. Slusallek)
- Augmented Vision (Prof. D. Stricker)
- Language Technology (Prof. H. Uszkoreit)
- Intelligent User Interfaces (Prof. W. Wahlster)
- Innovative Factory Systems (Prof. D. Zühlke)

In this series, DFKI publishes research reports, technical memos, documents (eg. workshop proceedings), and final project reports. The aim is to make new results, ideas, and software available as quickly as possible.

Prof. Wolfgang Wahlster
Director

Proceedings of the RIC Project Day

Workgroups 'Electronic Design' and
'Mechatronic Design'

Frank Kirchner (Editor)

Peter Kampmann, Marc Simnofske (Associate Editors)

06/2015

Document D-15-02 des

Deutschen Forschungszentrums für Künstliche Intelligenz (DFKI)

Abstract

This document is the current edition of an ongoing series of proceedings to document the workgroups' topics, discussions and efforts at the Robotics Innovation Center of DFKI GmbH. The content of each of these editions represents presentations (talks and posters) of a project day which is organized by two workgroups, respectively.

Workgroups are formed by peers that are dedicated to a specifc topic, so that they provide a platform for cross-project communication and knowledge transfer. In 2008 the workgroups started to present their results and past years work in an open presentation format called brown-bag talk, being a year after moved to more specialized so-called project days. Every year, since 2009, each workgroup presents results and past years work this project day. This format was extended to talk and poster presentations accompanied by the corresponding proceedings as a DFKI Document in 2015.

Zusammenfassung

Dieses Dokument enthält die aktuelle Ausgabe einer laufenden Tagungsband-Serie, welche die Themen, Diskussionen und Bemühungen der Arbeitsgruppen am Robotics Innovation Center der DFKI GmbH dokumentiert. Inhalt einer jeden Ausgabe sind die Vorträge und Poster eines Projekttags, der von jeweils zwei Arbeitsgruppen organisiert wird.

Arbeitsgruppen haben einen Leiter und widmen sich einem bestimmten Themengebiet, in dem sie eine Plattform für Kommunikation und Wissenstransfer über die Projekte hinaus darstellen. Im Jahr 2008 begannen die Arbeitsgruppen ihre Ergebnisse und Arbeiten in einem offenen Vortragsformat (dem sog. Brown-Bag Talk) vorzustellen, welches dann ein Jahr später in eigene Projekttage mündete. Seit 2014, ist dieses Format des Projekttags nochmal zu Vorträgen und Poster-Sitzungen erweitert worden, die seitdem in dem entsprechenden Tagungsband im Format eines DFKI Documents festgehalten werden.

Contents

1 Editorial

This is the second edition of 2015 to document the efforts of the DFKI-RIC thematic workgroups. Workgroups are formed by peers and provide a means for cross-project communication on a deep content level and facilitate knowledge transfer amongst the peers. In 2008 we first started forming workgroups on specific topics around robotics and AI research. Among them were topics as 'system design & engineering', 'machine learning', 'planning & representation' as well as 'frameworks & architectures' and 'man-machine interaction'. These workgroups were established with the intention to provide a platform for interested DFKI-RIC personnel for discussing the start of the art, recent achievements, and future developments in the respective fields.

Over time the workgroups gathered a collection of material in form of presentations, short papers, and posters which were worthwhile to be presented also to the rest of the institute. Due to this development, in 2009, we started to have a project day once every quarter. Each project day provided a platform for two of the workgroups to present their material and to discuss it with the further colleagues of the institute. Nowadays, the project day is organized as a one-day workshop with oral presentations, poster sessions, and a free pizza lunch for everybody who attends. Until now, the talks and posters have only been collected on our servers but were not assembled in a citable document.

This format at present is the next evolutionary step and it aims at eliminating this deficit by compiling the material of the workgroups presented during a project day into a single, citable document of unified format. We will see which steps can be taken in the future to enhance the presentation quality of this material.

Frank Kirchner

This year's third project day presented the material of the workgroups 'Electronic Design' and 'Mechatronic Design'.

Within the workgroup 'Electronic Design' all kinds of electronic developments from sensors to microcontrollers and FPGAs to neuromorphic computing are discussed. Over the years this workgroup has established a powerful library of hardware ip-cores which are used in almost all of our robotic systems speeding up the initial integration significantly in the past time. Developers can build on a continuously growing set of verified building blocks having the possibility to concentrate on the implementation of controllers and behaviors of increasing complexity which is one of the main driving aspects in the future of robotics.

The constant monitoring of the current hardware developments allows the group to rely on state of the art techniques either on hardware itself or on the tools to debug and develop hardware projects. In the past time the constant attentive attitude to whats new and will become future trends allowed the workgroup to handle to increasing complexity of large hardware projects.

This years presentations of the workgroup 'Electronic Design' focuses on the presentation of a communication standard for connecting embedded devices and standard processors in a simple and efficient way. The in-house development NDLCom is presented starting from the basics, the interface for the developers and how end-users can benefit from standardized tooling. A live demonstration of the plug and play approach as well as the efficiency of the developed shows the power of the developed communication solution. The subsequent poster demonstration addresses various topics of the team 'Electronic Design'. Starting from new ideas for decentralized energy storing and harvesting over solutions that simplify the integration of complex robotic systems now and in the future.

The purpose of the workgroup 'Mechatronic Design' is to discuss topics related to the development and assembly of innovative mechatronic systems which take advantage of modern materials and production technologies. Due to a high density of actuators and sensors, these systems are the basis for intelligent and autonomous robotic systems which offer a wide range of applications. At DFKI such systems have been developed for space, underwater, rehabilitation as well as for search and rescue scenarios.

The presentations of the workgroup 'Mechatronic Design' for the project day 2015 explain the current development in several DFKI RIC projects and topics which were discussed within the workgroup. The first

presentation introduce a catheter driver developed in the project CASCADE which allows the handling of the fragile catheter with low pressure. Furthermore the redesigned version of the DFKI Magnet Crawler is presented. The second presentation deal with the development of a elastic element for the next generation of DFKI RIC serial elastic actuators (SEA). The talk outlines the state of the art and explains the development process of the new spring design. The last talk contains self adaptive mechanisms for mobile robots. The author explains the motivation for self adaptive mechanisms followed by two examples of such mechanisms.

Additionally, in the poster-session, the new design of the Coyote III robot and the subsystems are explained. Furthermore the poster motivates the development and describes a reference mission concept. The second poster explains the performance of a diving cell designed for the robot LENG. Within the poster a overview over the requirements is given and the mechanical design is explained. The last poster shows the Teredo IceShuttle developed in the project Europa Explorer. After illustrate the system objectives and requirements the design of the unfolding and deployment mechanism is explained.

We would like to thank the authors of the fourth project day 2015 for their contributions and for the effort to provide their material in a standardized format.

Peter Kampmann, Marc Simnofske

2 ‘Electronic Design’

2.1 ‘NDLCom – Usage, Ecosystem and Tooling’ (ED-T-01)

Martin Zenzes[(1)]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: martin.zenzes@dfki.de

Abstract

This presentation gives an introduction on the Node Level Data Link Communication (NDLCom) which is developed as a communication solution between embedded devices and standard processor units. This talk focuses on the concepts, the payload design and the tooling that is available for NDLCom.

NDLCom – Usage, Ecosystem and Tooling

Projectday *Electronic & Design*

25.6.2015

Contents

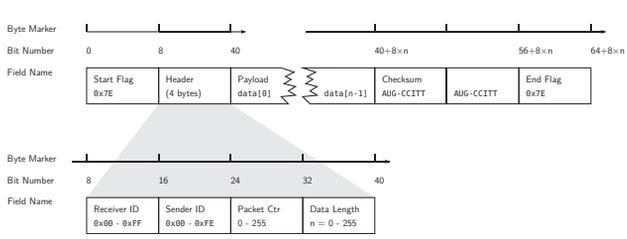
Contents

3 / 24

Roots

NDLCom – **N**ode Level **D**ata **L**ink **Com**munication

- Remnant of iStruct & SeeGrip projects (2010 – 2013)
- Transmission of messages inside a Robot: `ndlcom`
- Plus tooling built on top (Gui, logging, etc...)
- Grew organically based on the needs of the projects



4 / 24

Usage

Where NDLCom is used

Used in Robots:

- Charlie (iStruct)
- Gripper (SeeGrip)
- Sherpa (Rimres)
- SherpaTT, Coyote3 (TransTerrA)
- Aila (Besman)
- Future: *MANTIS (Limes)*

just imagine the pictures of all many impressive robots

OSI-Layers

- OSI – **O**pen **S**ystems **I**nterconnection
- Different Layers – different Functionality
- Settling the lower layers

OSI Model				
Layer		Data unit	Function[3]	Examples
Host layers	7. Application	Data	High-level APIs, including resource sharing, remote file access, directory services and virtual terminals	Mail, Internet Explorer, Firefox
	6. Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption	ASCII, EBCDIC, JPEG
	5. Session		Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes	RPC, PAP, HTTP, FTP, SMTP, Secure Shell
	4. Transport	Segments	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing	TCP, UDP
Media layers	3. Network	Packet/Datagram	Structuring and managing a multi-node network, including addressing, routing and traffic control	IPv4, IPv6, IPsec, AppleTalk
	2. Data link	Bit/Frame	Reliable transmission of data frames between two nodes connected by a physical layer	PPP, IEEE 802.2, L2TP
	1. Physical	Bit	Transmission and reception of raw bit streams over a physical medium	DSL, USB

OSI-Layers

- OSI – **O**pen **S**ystems **I**nterconnection
- Different Layers – different Functionality
- Settling the lower layers

OSI Model				
Layer		Data unit	Function[3]	Examples
Host layers	7. Application	Data	High-level APIs, including resource sharing, remote file access, directory services and virtual terminals	Mail, Internet Explorer, Firefox
	6. Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption	ASCII, EBCDIC, JPEG
	5. Session		Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes	RPC, PAP, HTTP, FTP, SMTP, Secure Shell
	4. Transport	Segments	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing	TCP, UDP
Media layers	3. Network	Packet/Datagram	Structuring and managing a multi-node network, including addressing, routing and traffic control	IPv4, IPv6, IPsec, AppleTalk
	2. Data link	Bit/Frame	Reliable transmission of data frames between two nodes connected by a physical layer	PPP, IEEE 802.2, L2TP
	1. Physical	Bit	Transmission and reception of raw bit streams over a physical medium	DSL, USB

representations

NDLCom

HSCom, LVDS

6 / 24

Contents

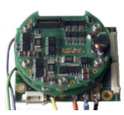
7 / 24

Motivation

Data transmission in a daisychain


- Heterogeneous electronics:
  - `x86`, `Cortex-M3`, `Spartan3/6`, `PowerPC`, (`Cortex-A8`)
- Local control loops, use local computational power
- Simple-to-understand protocol, easy implementation
- Full control, low overhead, small payloads
- Don't care about *some* lost packages






Motivation

Data transmission in a daisychain

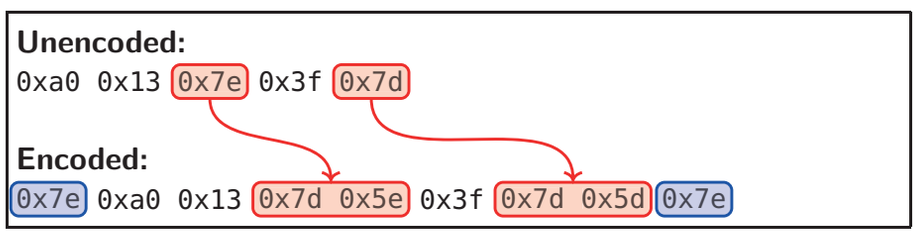

- Each device connected p2p to at least one of it's neighbours
- Simple tree-structure: no circular connections
- Messages get passed from device to device
- Needs some sort of routing on junctions




Message Escaping

Define datagramm in bytestream

- RFC1662: HDLC-like framing
- Escape special characters (`$byte` $\oplus$ `0x20`):
  - `0x7d` -> `0x7d 0x5d`
  - `0x7e` -> `0x7d 0x5e`
- Sorround frame with Flags: `0x7e`



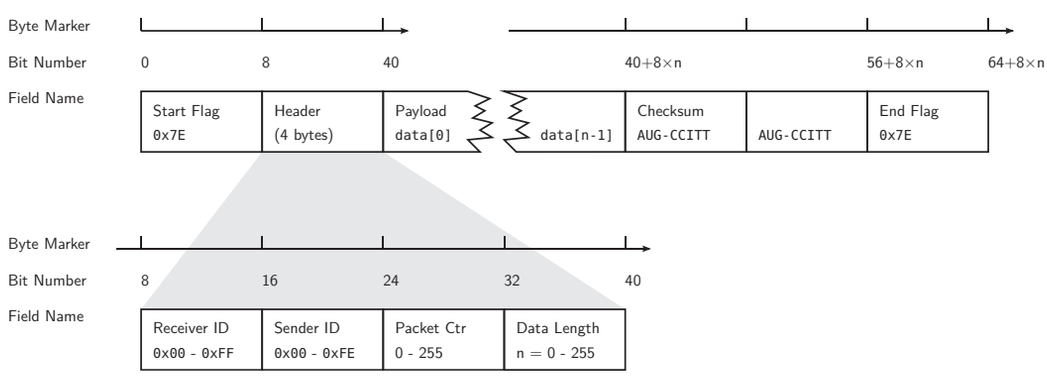
- Stateless Layer2: Single byte determines packet boundaries

RFC1662

Common message format

Header

- Start/Stop flag, CRC16/AUG-CCITT Checksum
- Header: Receiver, Sender, PaketCounter, PayloadLength
- Payload: Representation ID, Timestamp, *actual payload*
- At least 17Byte Overhead (8Byte + 9Byte)

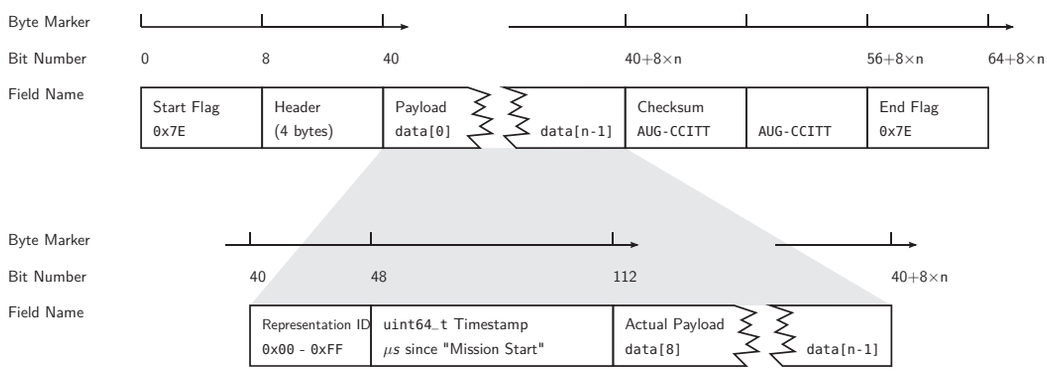


CRC16/AUG-CCITT

Common message format

Payload

- Start/Stop flag, CRC16/AUG-CCITT Checksum
- Header: Receiver, Sender, PaketCounter, PayloadLength
- Payload: Representation ID, Timestamp, *actual payload*
- At least 17Byte Overhead (8Byte + 9Byte)



CRC16/AUG-CCITT

Routing

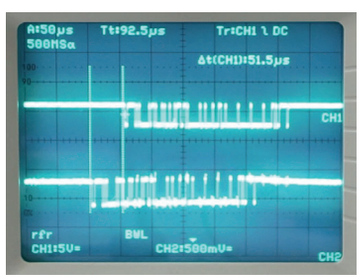
- Receiving a message:
  - store origin in RoutingTable: `senderId->origin`
  - `receiverId==(myId||broadcast)` → handle
  - `receiverId==(!myId||broadcast)` → transmit
- Transmitting a message:
  - get destination from RoutingTable: `origin->receiverId`
  - `receiverId==broadcast` → send `(interfaces != origin)`
  - `destination==unknown` → send `(interface != origin)`
  - `destination!=unknown` → send `destination`

Forwarding

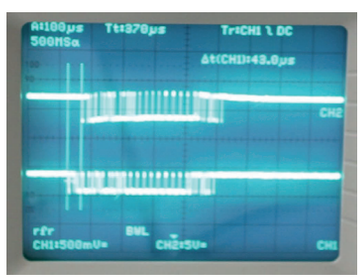
Cut-Through

- Watch for Start Flags, observe second byte: `receiverId`
- If possible: Forward without awaiting the whole package
- Less RAM, less delay
- Cannot filter out broken packages
- Blocks outgoing port if datarate mismatches

Forwarding delay from Rx_{in} to Tx_{out} (921600Baud, $t_{byte} = 20\mu s$)



STM32: $52\mu s$



Spartan3: $43\mu s$

Contents

Registers

- Scalar values of certain type
  `uint16_t, double, ...`
- Addressable via deviceId and registerId
- Static registers are defined in RobotConfig XML
  Outside world has a-priori knowledge
- Dynamic registers can be queried at runtime
  Nothing known in advance
- Also be used directly between devices → local control loops

In System Programming

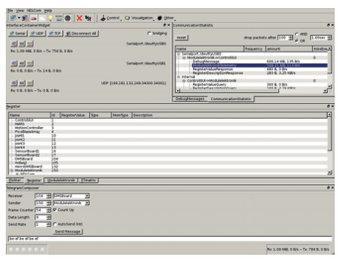
- Supported by most NDLCom devices
- Reprogram hardware *inside* the robot
- Reliable datastream by performing explicit handshaking

Ping

- Receiver has to respond with its local time
- Can be used to set local clock → Clock synchronization
- Broadcast Ping: Get response from all active devices

Contents

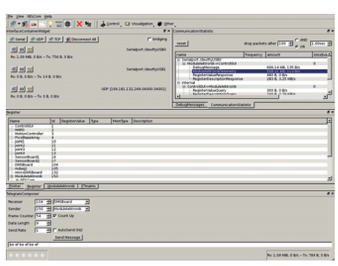
CommonGUI



- Contains most commonly used widgets
  - NDLComQt, ISP, DebugMessages
  - Plotter, Logger, RegisterQt
- Binary symlinked out of build directory
- Some limited commandline options:

```
$ cd gui/CommonGUI
$ make
...
$ readlink -f ./CommonGUI
.../dev/gui/CommonGUI/build/x86_64-linux-gnu/CommonGUI
$ ./CommonGUI
```

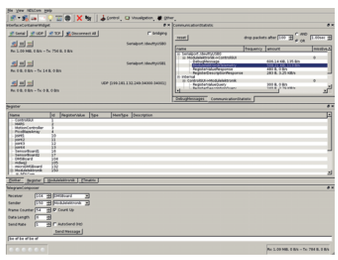

CommonGUI



- Contains most commonly used widgets
  - NDLComQt, ISP, DebugMessages
  - Plotter, Logger, RegisterQt
- Binary symlinked out of build directory
- Some limited commandline options:

```
$ cd gui/CommonGUI
$ make
...
$ readlink -f ./CommonGUI
.../dev/gui/CommonGUI/build/x86_64-linux-gnu/CommonGUI
$ ./CommonGUI --serialport=/dev/ttyUSB0 --baudrate=500000
```

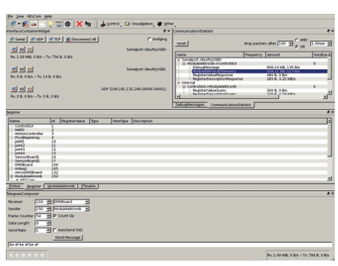
CommonGUI



- Contains most commonly used widgets
  - NDLComQt, ISP, DebugMessages
  - Plotter, Logger, RegisterQt
- Binary symlinked out of build directory
- Some limited commandline options:

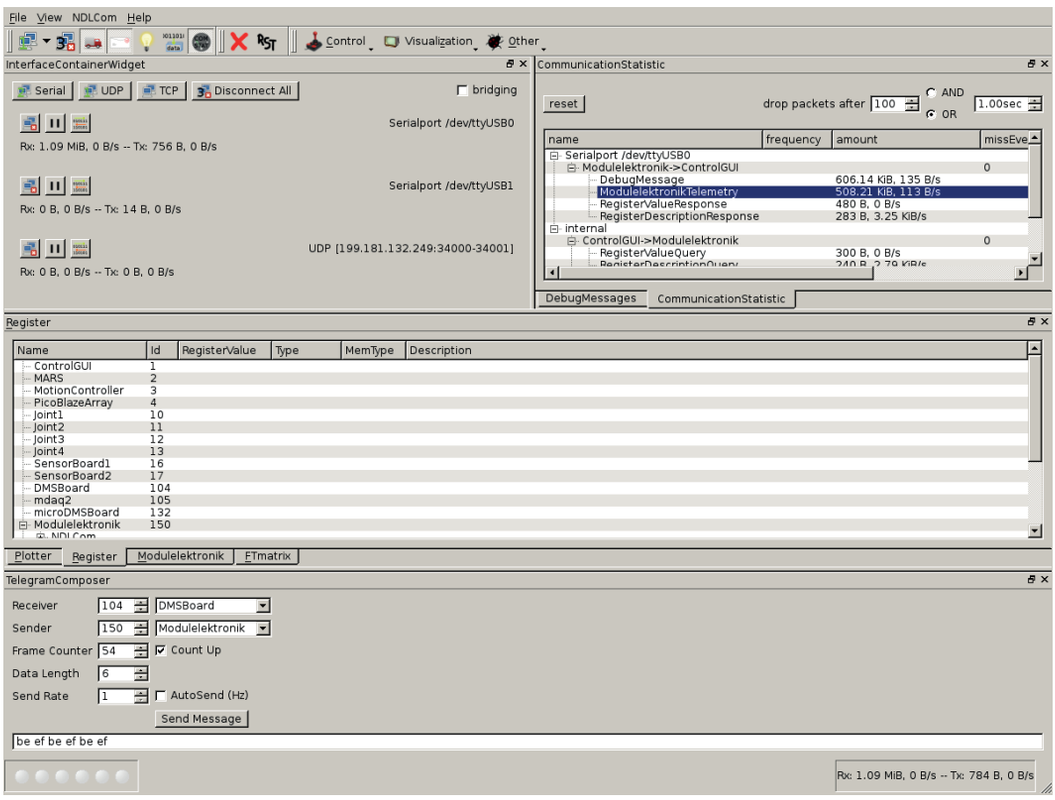
```
$ cd gui/CommonGUI
$ make
...
$ readlink -f ./CommonGUI
.../dev/gui/CommonGUI/build/x86_64-linux-gnu/CommonGUI
$ ./CommonGUI --replayProtolog=myLogFile.protolog
```


CommonGUI



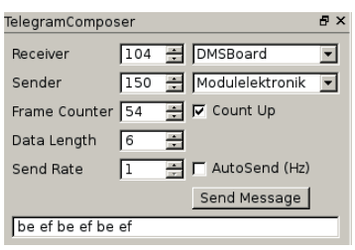
- Contains most commonly used widgets
  - NDLComQt, ISP, DebugMessages
  - Plotter, Logger, RegisterQt
- Binary symlinked out of build directory
- Some limited commandline options:

```
$ cd gui/CommonGUI
$ make
...
$ readlink -f ./CommonGUI
.../dev/gui/CommonGUI/build/x86_64-linux-gnu/CommonGUI
$ ./CommonGUI --xml-file=../../RobotConfig/xml/TestConfig.xml
```



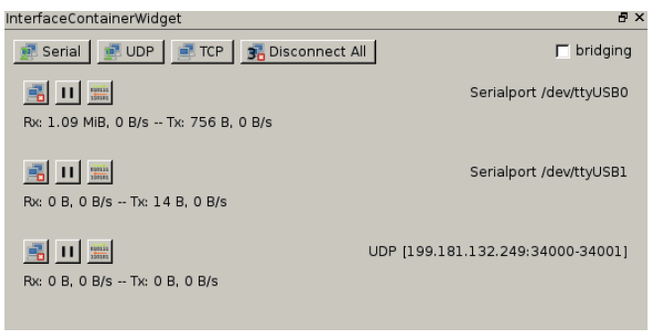
TelegramComposer

- Synthesize messages from scratch
- Fill header via ComboBoxes
- Define arbitrary Payload
- Set sendrate for Loadtesting



ContainerWidget

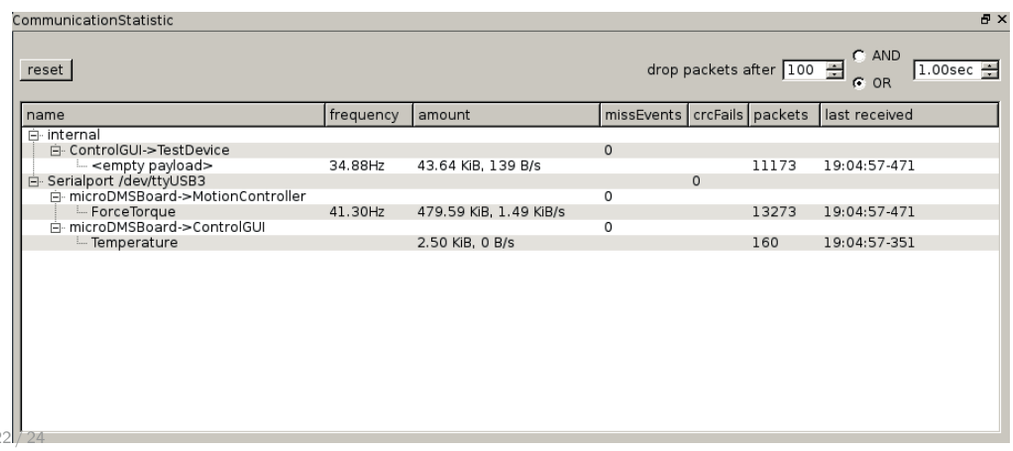
- Create and close multiple connections
- Serial, UDP, and TCP
- See datarate and raw traffic per interface



21 / 24

CommunicationStatistics

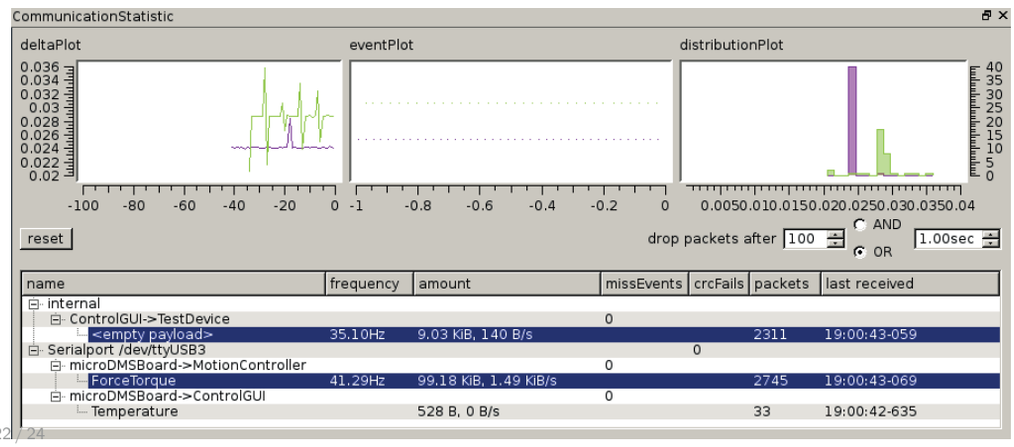
- Watch details about individual datastreams
- Tree structure: Interface → DeviceToDevice → Payload
- Can plots timing information for relative analysis



22 / 24

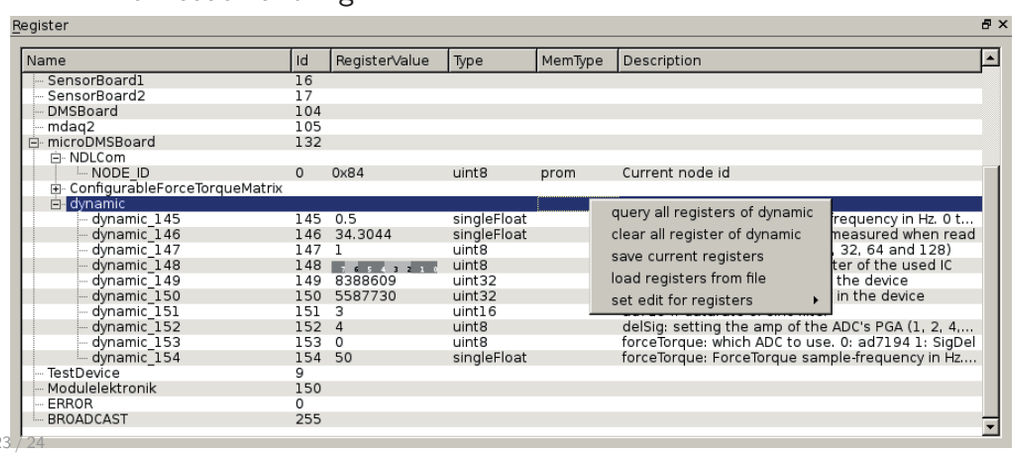
CommunicationStatistics

- Watch details about individual datastreams
- Tree structure: Interface → DeviceToDevice → Payload
- Can plots timing information for relative analysis



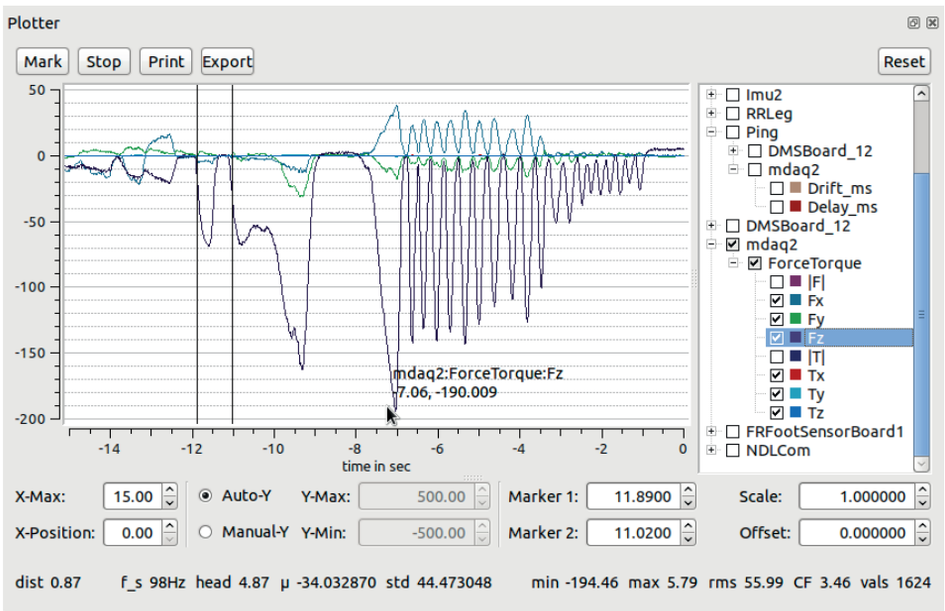
RegisterQt

- Query, View, Edit, Load
- Save and Load registers from textfile
- Tree structure: Device → DeviceClass → Register
- Edits: Hexadecimal, Binary, Decimal
- timeout handling



Plotter

- Live-data viewer with scaling, offset, color, mouse-over, autoscale, start/stop...



24 / 24

2.2 'Advanced introduction on NDLCom' (ED-T-02)

Martin Zenzes [1]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: `martin.zenzes@dfki.de`

Abstract

This second presentation on NDLCom is continuing the introduction to this communication solution. This talk focuses on the access of NDLCom for the embedded systems developer. Ways to obtain the source code, building the libraries and how to interact and extend the ndlcom solution are presented. The talk finishes with a live demo.

NDLCom – Usage, Ecosystem and Tooling

Projectday *Electronic & Design*

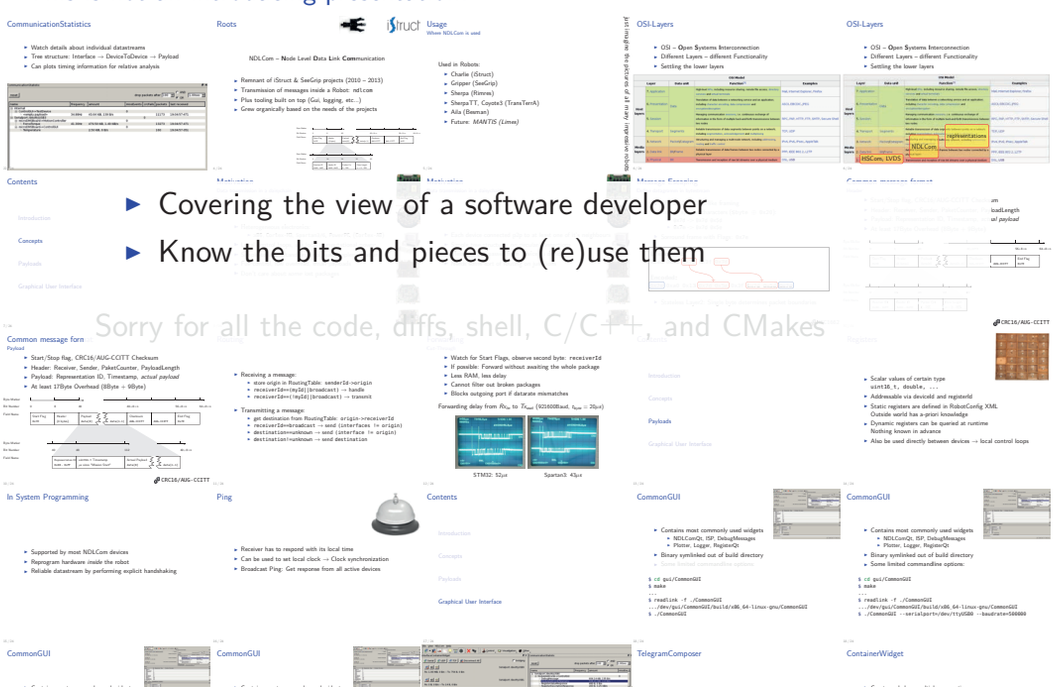
25.6.2015

Contents

Contents

Disclaimer

This is not an introducing presentation!

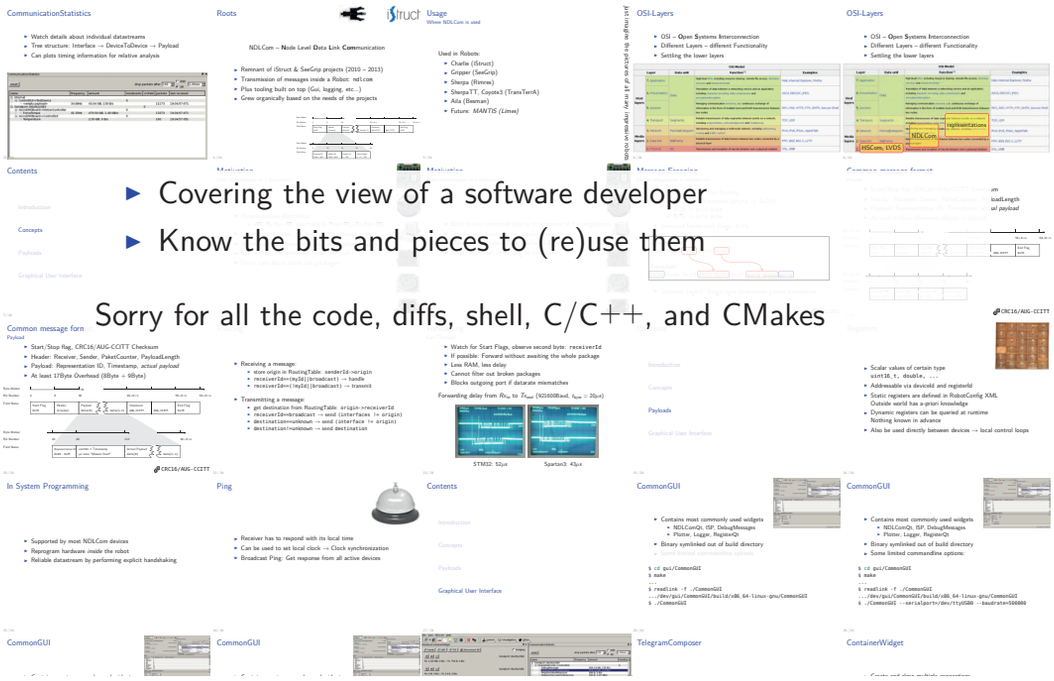


- Covering the view of a software developer
- Know the bits and pieces to (re)use them

Sorry for all the code, diffs, shell, C/C++, and CMakes

Disclaimer

This is not an introducing presentation!



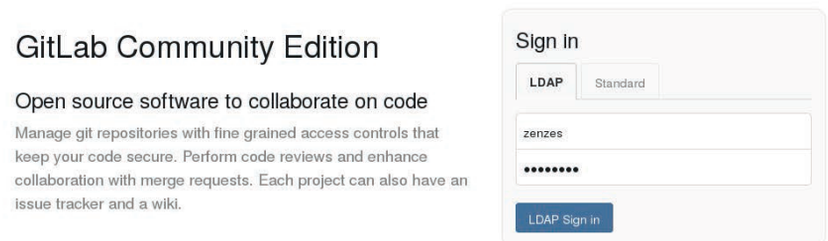
- Covering the view of a software developer
- Know the bits and pieces to (re)use them

Sorry for all the code, diffs, shell, C/C++, and CMakes

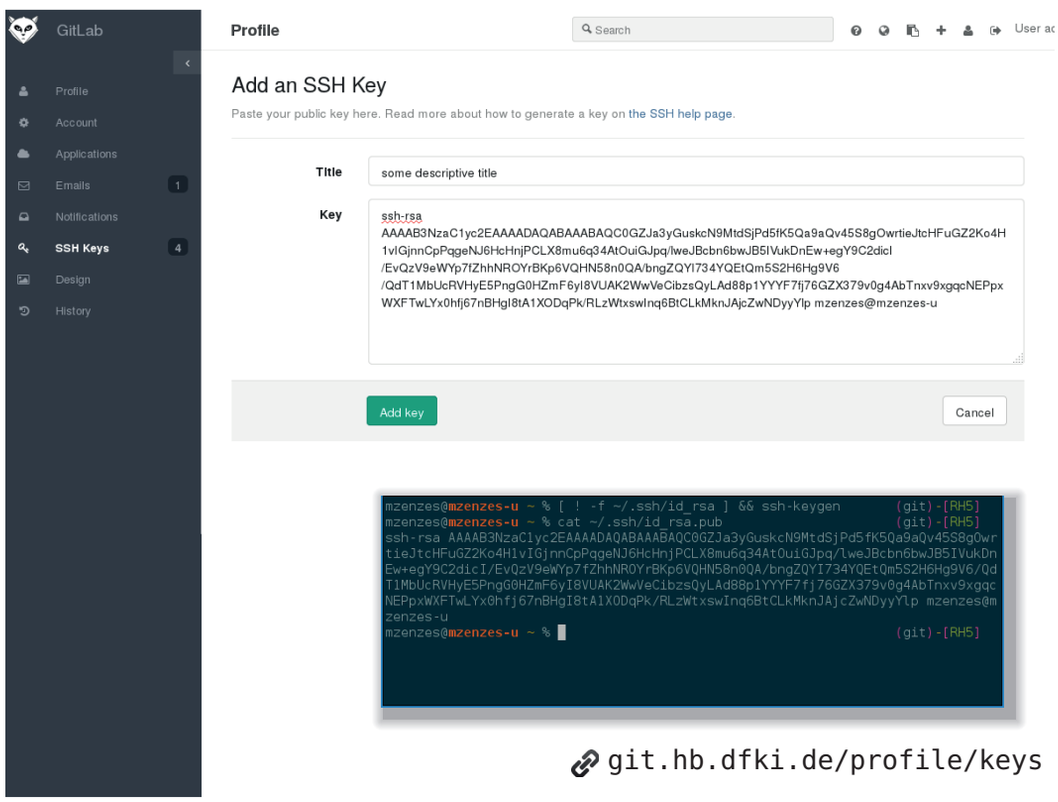
Contents

GitLab

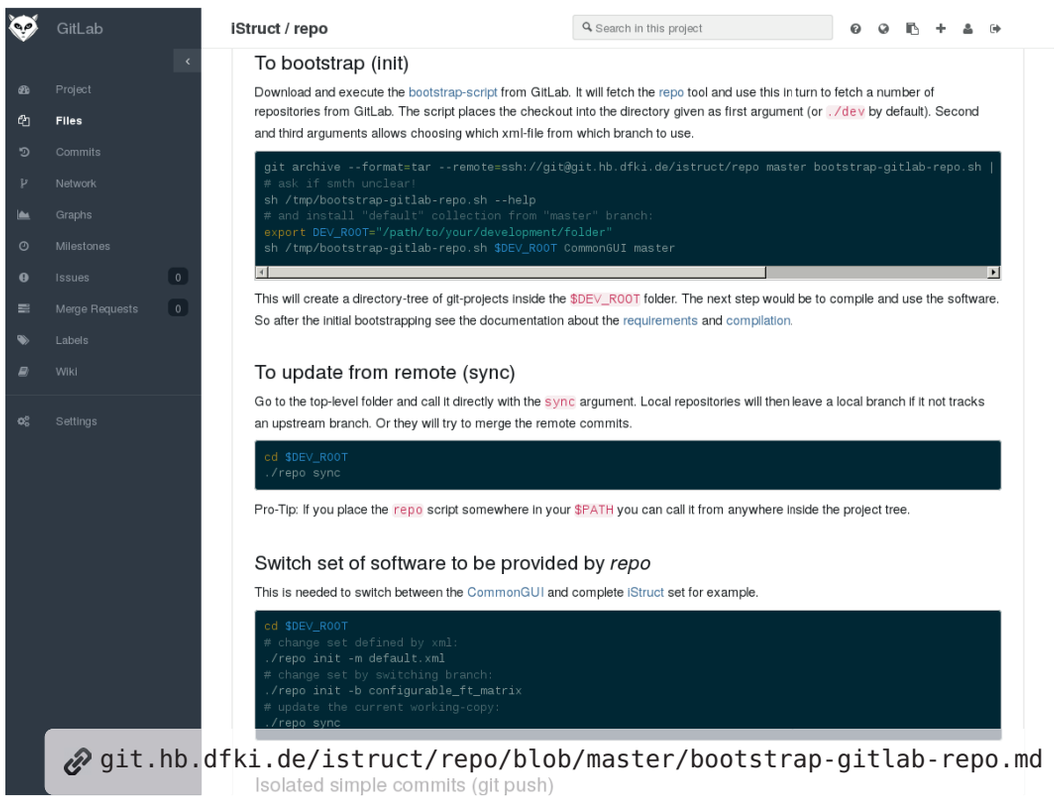
- In-house GitLab instance, similar to GitHub
- Webinterface: Be member of *DFKI Domain*
- Push & Pull: upload public ssh-key
- iStruct-SVN split into 35 git repositories
- Subrepositories managed via *Google repo*



6 / 28



git.hb.dfki.de/profile/keys



Committers

- Some history is lost due to Code Migration
- Distortion due to scripted commits
- Committers sorted by number of commits and printed in columns:

```
$ repo forall -c "git shortlog --numbered --summary" | \
    awk '{n="";for(i=2;i<=NF;i++)n=n $i " "; arr[n]+=$1;} END \
    {for (i in arr) print arr[i],i}' | \
    sort -n | cut -d ' ' -f 1 --complement | column

Malte Römmermann        Zhuowei Wang            Peter Kampmann
vazizi                  Ahmed Rehman Ghazi      Lars Sinda
daniel kuehn            Vahid Azizi             Daniel Kühn
Matthias Jordan         Vinzenz Bargsten        Armin Burchardt
Hendrik Hanff           Ajish Babu              Tobias Stark
Lan Yue Ji              Ivaylo Enchev           Moritz Schilling
Matthias Rosynski       Piotr Twardosz          Martin Zenzes
```

Bootstrapping

- Shell script downloads and initializes `repo`
- `repo` does full checkout of `master` into `./dev`

```
$ git archive --format=tar --remote=ssh://git@git.hb.dfki.de/istruct/repo \
    master bootstrap-gitlab-repo.sh | tar xO > /tmp/bootstrap-gitlab-repo.sh
$ sh /tmp/bootstrap-gitlab-repo.sh
...
$ cd dev
$ ./repo sync
$ cd gui/CommonGUI
...
```

Some `repo` tips:

- `repo init -b $SOMEBRANCH` – switch meta-branch
- `repo sync` – update working copy; rebase local branches
- `repo diff` – be informative
- `repo forall -c 'cmd'` – execute command in each git

Using Repo and Git

Directory Layout

- 35 repositories, 276 directories, 2599 files, 155MB

```
+-- CMake-Modules
+-- documents
+-- external
|   +-- eigen3
+-- gnuplot
+-- gui
|   +-- CharlieGUI
|   +-- CommonGUI
|   +-- TemplateNDLCom
+-- lib
|   +-- autoaccess
|   +-- kinematics
|   |   +-- charlie
|   +-- math
|   +-- ndlcom
|   +-- ndlcom_qt
|   +-- protolog
|   +-- register_qt
|   +-- representations
|   +-- RobotConfig
|   +-- serialcom
|   +-- stm32common
|   +-- stm32fw
|   +-- system
|   +-- tcpcom
|   +-- udpcom
|   +-- walking
|   +-- widgets
|       +-- charlie
|       +-- common
|       +-- controls
|       +-- plotter
+-- mcs
|   +-- CharlieMCS
+-- stm32
|   +-- bootloader
|   +-- DMSBoard
|   +-- mdaq2
|   +-- MicroDMSsupport
+-- tools
    +-- protolog2csv
    +-- ReplayPlotter
```

Directory Layout

- 35 repositories, 276 directories, 2599 files, 155MB

```
+-- CMake-Modules
+-- documents
+-- external
|   +-- eigen3
+-- gnuplot
+-- gui
|   +-- CharlieGUI
|   +-- CommonGUI
|   +-- TemplateNDLCom
+-- lib
|   +-- autoaccess
|   +-- kinematics
|   |   +-- charlie
|   +-- math
|   +-- ndlcom
|   +-- ndlcom_qt
|   +-- protolog
|   +-- register_qt
|   +-- representations
|   +-- RobotConfig
|   +-- serialcom
|   +-- stm32common
|   +-- stm32fw
|   +-- system
|   +-- tcpcom
|   +-- udpcom
|   +-- walking
|   +-- widgets
|       +-- charlie
|       +-- common
|       +-- controls
|       +-- plotter
+-- mcs
|   +-- CharlieMCS
+-- stm32
|   +-- bootloader
|   +-- DMSBoard
|   +-- mdaq2
|   +-- MicroDMSsupport
+-- tools
    +-- protolog2csv
    +-- ReplayPlotter
```

11 / 28

Directory Layout

- 35 repositories, 276 directories, 2599 files, 155MB

```
+-- CMake-Modules
+-- documents
+-- external
|   +-- eigen3
+-- gnuplot
+-- gui
|   +-- CharlieGUI
|   +-- CommonGUI
|   +-- TemplateNDLCom
+-- lib
|   +-- autoaccess
|   +-- kinematics
|   |   +-- charlie
|   +-- math
|   +-- ndlcom
|   +-- ndlcom_qt
|   +-- protolog
|   +-- register_qt
|   +-- representations
|   +-- RobotConfig
|   +-- serialcom
|   +-- stm32common
|   +-- stm32fw
|   +-- system
|   +-- tcpcom
|   +-- udpcom
|   +-- walking
|   +-- widgets
|       +-- charlie
|       +-- common
|       +-- controls
|       +-- plotter
+-- mcs
|   +-- CharlieMCS
+-- stm32
|   +-- bootloader
|   +-- DMSBoard
|   +-- mdaq2
|   +-- MicroDMSsupport
+-- tools
    +-- protolog2csv
    +-- ReplayPlotter
```

11 / 28

Directory Layout

- 35 repositories, 276 directories, 2599 files, 155MB

```
+-- CMake-Modules
+-- documents
+-- external
|   +-- eigen3
+-- gnuplot
+-- gui
|   +-- CharlieGUI
|   +-- CommonGUI
|   +-- TemplateNDLCom
+-- lib
|   +-- autoaccess
|   +-- kinematics
|   |   +-- charlie
|   +-- math
|   +-- ndlcom
|   +-- ndlcom_qt
|   +-- protolog
|   +-- register_qt
|   +-- representations
|   +-- RobotConfig
|   +-- serialcom
|   +-- stm32common
|   +-- stm32fw
|   +-- system
|   +-- tcpcom
|   +-- udpcom
|   +-- walking
|   +-- widgets
|       +-- charlie
|       +-- common
|       +-- controls
|       +-- plotter
+-- mcs
|   +-- CharlieMCS
+-- stm32
|   +-- bootloader
|   +-- DMSBoard
|   +-- mdaq2
|   +-- MicroDMSsupport
+-- tools
    +-- protolog2csv
    +-- ReplayPlotter
```


Buildsystem

Properties

- CMake, augmented with wrapper Makefile
  - `make INSTALLDIR=/tmp/test.install JOBS=1 install`
- `add_subdirectory()` to include components

```
# adding subdirs; testing for TARGET to prevent multiple inclusion:
if(NOT TARGET ndlcom)
    add_subdirectory(${NDLCOM_ROOT_DIR}/lib/ndlcom ndlcom)
endif(NOT TARGET ndlcom)
```

- Independent build *out-of-src* for each component
- `pkg-config` used for propagation of flags
  - `$name.pc` and `$name-uninstalled.pc`
- CMake-Toolchain files in `CMake-Modules`
  - `make ARCH=arm-none-eabi`

Buildsystem

User interface

- Wrapper Makefile: just cd and make

```
$ cd lib/protolog
$ ls
CMakeLists.txt  include   protolog.pc.in              README
doc             Makefile  protolog-uninstalled.pc.in  src
$ make
mkdir -p ".../dev/lib/protolog/build/x86_64-linux-gnu"; \
sh -c "cd .../dev/lib/protolog/build/x86_64-linux-gnu; \
    cmake .../dev/lib/protolog -DCMAKE_EXPORT_COMPILE_COMMANDS=ON \
    -DCMAKE_INSTALL_PREFIX=.../DFKI.install/x86_64-linux-gnu"
-- The C compiler identification is GNU 4.9.2
-- The CXX compiler identification is GNU 4.9.2
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
...
$ ls
build           doc      Makefile        protolog-uninstalled.pc.in  src
CMakeLists.txt  include  protolog.pc.in  README
```


Buildsystem

User interface

- Wrapper Makefile: just cd and make

```
$ cd lib/protolog
$ ls
CMakeLists.txt  include   protolog.pc.in              README
doc             Makefile  protolog-uninstalled.pc.in  src
$ CXX=clang++-3.7 CC=clang-3.7 make
mkdir -p ".../dev/lib/protolog/build/x86_64-pc-linux-gnu;" \
sh -c "cd .../dev/lib/protolog/build/x86_64-pc-linux-gnu; \
    cmake .../dev/lib/protolog -DCMAKE_EXPORT_COMPILE_COMMANDS=ON \
    -DCMAKE_INSTALL_PREFIX=.../DFKI.install/x86_64-pc-linux-gnu"
-- The C compiler identification is Clang 3.7.0
-- The CXX compiler identification is Clang 3.7.0
-- Check for working C compiler: /usr/bin/clang-3.7
-- Check for working C compiler: /usr/bin/clang-3.7 -- works
...
$ ls
build           doc      Makefile        protolog-uninstalled.pc.in  src
CMakeLists.txt  include  protolog.pc.in  README
```

13 / 28

Caveats

No system is perfect...

```
sudo apt-get install \
    build-essential cmake git file \
    xmlstarlet graphviz \
    libqwt5-qt4-dev \
    libdb5.3++-dev \
    libqt4-dev \
    libqt4-opengl-dev \
    libxml++2.6-dev liblzma-dev libffi-dev \
    libxml2-utils \
    libfontconfig-dev \
    libeigen3-dev libboost-dev \
    freeglut3-dev;
```

- Manual handling of system-dependencies
  `software-requirements.md`
- Only static build on Ubuntu/Debian really tested
- Sometimes recreation of build folder neccessary
- `pkg-config` called too often
- Not so easy to create Debian packages
- Some overlinking happens...

`Overlinking`

Contents

Adding new Representation types

- Payload defined via structs in C-headers
- Small database for name->id mapping
- Generates C-defines and VHDL code

To add payload type:

0. *branch* `lib_representations`
1. Create header which describes the datastructure
2. Choose id for the new datatype in `src/names.c`
3. Adopt `include/ndlcom_qt/RepresentationMapper.h`
4. Regenerate `id.h` by recompiling tree

16 / 28

Adding new Representation types

- First member: `struct Representation mBase;`
- Followed by any C datastructure
- Compact layout: `__attribute__((packed))`
- C++ shim for convenience

```
#ifndef _REPRESENTATIONS_TESTDATA_H
#define _REPRESENTATIONS_TESTDATA_H

#include "Representation.h"

struct RepresentationsTestData
{
    struct Representation mBase;
    int theAnswer;
} __attribute__((packed));

#ifdef __cplusplus
namespace representations
{
    struct TestData : public ::RepresentationsTestData {} __attribute__((packed));
};
#endif

#endif/*_REPRESENTATIONS_TESTDATA_H*/
```

17 / 28

Defining Representations Id

- *Nobody has to use representations, is only a payload*
- Mind you: Limited address range!
- Give Id with struct-name directly in `src/names.c`
- C and VHDL defines for compile-time mapping of `name->id`

```
diff --git a/src/names.c b/src/names.c
index 3fdb3f6..dc2f820 100644
--- a/src/names.c
+++ b/src/names.c
@@ -114,6 +114,8 @@ static const struct RepresentationsNamesEntry representations[] =

     {202, "ForwardDynamicsMessage"},
     {203, "BackwardDynamicsMessage"},
+
+    {241, "RepresentationsTestData"},

     // end-of-list
     {0,0}
```


Generated files

- Generated C and VHDL from `src/names.c`

```
--- build/x86_64-linux-gnu/include/representations/id.h        2015-06-12 15:10:29.717823397 +0200
+++ build/x86_64-linux-gnu/include/representations/id.h.back        2015-06-17 10:33:03.166388673 +0200
@@ -81,6 +81,7 @@
 #define REPRESENTATIONS_REPRESENTATION_ID_RepresentationsCapioMotor 0x96
 #define REPRESENTATIONS_REPRESENTATION_ID_ForwardDynamicsMessage 0xca
 #define REPRESENTATIONS_REPRESENTATION_ID_BackwardDynamicsMessage 0xcb
+#define REPRESENTATIONS_REPRESENTATION_ID_RepresentationsTestData 0xf1


 #ifdef __cplusplus
```

generated!

```
--- vhdl/Representations.vhd        2015-06-23 17:20:41.302215816 +0200
+++ vhdl/Representations.vhd.back        2015-06-23 17:15:00.470228481 +0200
@@ -83,4 +83,5 @@
     constant Representation_Id_RepresentationsCapioMotor      : std_logic_vector(7 downto 0) := x"96";
     constant Representation_Id_ForwardDynamicsMessage         : std_logic_vector(7 downto 0) := x"ca";
     constant Representation_Id_BackwardDynamicsMessage        : std_logic_vector(7 downto 0) := x"cb";
+    constant Representation_Id_RepresentationsTestData        : std_logic_vector(7 downto 0) := x"f1";
 end representations;
```

generated!

RepresentationMapper

- Generated manually...
- Qt-Code for mapping different payloads to signals/slots:

```
diff --git a/include/ndlcom_qt/RepresentationMapper.h b/include/ndlcom_qt/RepresentationMapper.h
index 1230330..b98c0c5 100644
--- a/include/ndlcom_qt/RepresentationMapper.h
+++ b/include/ndlcom_qt/RepresentationMapper.h
@@ -83,6 +83,7 @@ namespace representations
     struct MaxTemperature;
     struct WalkingSpeed;
     struct WalkingState;
+    struct TestData;
 };

 struct NDLComHeader;
@@ -175,6 +176,7 @@ namespace ndlcom_qt
         void rxRepresentation(const NDLComHeader&, const representations::Temperature&);
         void rxRepresentation(const NDLComHeader&, const representations::WalkingSpeed&);
         void rxRepresentation(const NDLComHeader&, const representations::WalkingState&);
+        void rxRepresentation(const NDLComHeader&, const representations::TestData&);

     private:

```


RepresentationMapper

- Generated manually...
- Qt-Code for mapping different payloads to signals/slots:

```
diff --git a/src/RepresentationMapper.cpp b/src/RepresentationMapper.cpp
index 25c9742..9333f92 100644
--- a/src/RepresentationMapper.cpp
+++ b/src/RepresentationMapper.cpp
@@ -68,6 +68,7 @@
 #include "representations/valve_control_board.h"
 #include "representations/WalkingSpeed.h"
 #include "representations/WalkingState.h"
+#include "representations/TestData.h"

 #include <QDebug>
 #include <QMetaType>
@@ -147,6 +148,7 @@ RepresentationMapper::RepresentationMapper(QObject* parent) : QObject(parent)
     qRegisterMetaType<representations::MaxTemperature>("representations::MaxTemperature");
     qRegisterMetaType<representations::WalkingState>("representations::WalkingState");
     qRegisterMetaType<representations::WalkingSpeed>("representations::WalkingSpeed");
+    qRegisterMetaType<representations::TestData>("representations::TestData");
     qRegisterMetaType<ndlcom::Message>("ndlcom::Message");

     /* this is the Message-signal emitted by the derived class NDLCom. we connect it to here, so we
@@ -429,6 +431,9 @@ void RepresentationMapper::slot_rxMessage(const ndlcom::Message& msg)
                 case REPRESENTATIONS_REPRESENTATION_ID_RepresentationsWalkingSpeed:
                     emit rxRepresentation(msg.mHdr, *(representations::WalkingSpeed*)repreData);
                     break;
+                case REPRESENTATIONS_REPRESENTATION_ID_RepresentationsTestData:
+                    emit rxRepresentation(msg.mHdr, *(representations::TestData*)repreData);
+                    break;
                 case REPRESENTATIONS_REPRESENTATION_ID_RepresentationsWalkingState:
                     emit rxRepresentation(msg.mHdr, *(representations::WalkingState*)repreData);
                     break;
```

RobotConfig

System model in XML database

- File selected by CMake variable:

```
# this will select which RobotConfig xml-database to use
set(ROBOTCONFIG CommonConfig)
```

- Database split via **<xi:xinclude../>** over multiple files
- Common properties via **<deviceclass../>**, **<metaclass../>**
- Compiletime verification with XSD
- Used to generate C++ and VHDL
- At runtime parsed for detailed Register informations
- *Should* be used to generate even more code...

RobotConfig

Adding a new device

- Define valid **<device../>** in selected XML-file
- C- and VHDL defines for compile-time mapping of `name->id`

```
<!--
xsd-description for a "device" type: they need a name+Id. may contain any
number of registers or isp_regions, additionally to the wrapping classes.

TODO: how to test for uniqueness of registerId per device?
-->
<xs:complexType name="device_t" mixed="true" >
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="register" />
      <xs:element ref="isp_region" />
      <xs:element ref="deviceclass" />
      <xs:element ref="metaclass" />
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="xs:Name" use="required" />
  <xs:attribute name="id" type="ndlcomDeviceId_t" use="required" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

RobotConfig

Adding a new device

- Define valid **<device../>** in selected XML-file
- C- and VHDL defines for compile-time mapping of `name->id`

```
diff --git a/xml/CommonConfig.xml b/xml/CommonConfig.xml
index 2fc6ddf..8ab8008 100644
--- a/xml/CommonConfig.xml
+++ b/xml/CommonConfig.xml
@@ -61,6 +61,9 @@ in einer XML-Datei halbwegs schick unterbringen kann... -->
     <xi:include href="deviceclasses/ConfigurableForceTorqueMatrix.xml" />
   </device>

+  <!-- added new device to xml-database -->
+  <device name="TestDevice" id="9"/>
+
   <device name="Modulelektronik" id="150" >
       <xi:include href="deviceclasses/NDLCom.xml" />
       <xi:include href="deviceclasses/Modulelektronik.xml" />
```


RobotConfig

Adding a new device

- Define valid **<device../>** in selected XML-file
- C- and VHDL defines for compile-time mapping of `name->id`

```
--- build/x86_64-linux-gnu/include/RobotConfig/Devices.h.back        2015-06-12 14:31:56.733909348 +0200
+++ build/x86_64-linux-gnu/include/RobotConfig/Devices.h        2015-06-12 14:32:15.121908665 +0200
@@ -21,6 +21,7 @@
         DMSBoard = 104,
         mdaq2 = 105,
         microDMSBoard = 132,
+        TestDevice = 9,
         Modulelektronik = 150,
         ERROR = 0,
         BROADCAST = 255
@@ -40,6 +41,7 @@
         FIRST_DYNAMIC_REGISTER_ID_DMSBoard = 145,
         FIRST_DYNAMIC_REGISTER_ID_mdaq2 = 145,
         FIRST_DYNAMIC_REGISTER_ID_microDMSBoard = 145,
+        FIRST_DYNAMIC_REGISTER_ID_TestDevice = 0,
         FIRST_DYNAMIC_REGISTER_ID_Modulelektronik = 210,
         FIRST_DYNAMIC_REGISTER_ID_ERROR = 1,
         FIRST_DYNAMIC_REGISTER_ID_BROADCAST = 1
```

generated!

RobotConfig

Adding a new device

- Define valid **<device../>** in selected XML-file
- C- and VHDL defines for compile-time mapping of `name->id`

```
1  --- vhdl/CommonConfig/Devices.vhd.back        2015-06-23 17:39:51.854173061 +0200
2  +++ vhdl/CommonConfig/Devices.vhd        2015-06-23 17:39:59.154172790 +0200
3  @@ -26,6 +26,7 @@
4       constant Device_Id_DMSBoard                     : std_logic_vector(7 downto 0) := x"68";
5       constant Device_Id_mdaq2                        : std_logic_vector(7 downto 0) := x"69";
6       constant Device_Id_microDMSBoard                : std_logic_vector(7 downto 0) := x"95";
7  +    constant Device_Id_TestDevice                   : std_logic_vector(7 downto 0) := x"09";
8       constant Device_Id_Modulelektronik              : std_logic_vector(7 downto 0) := x"96";
9       constant Device_Id_ERROR                        : std_logic_vector(7 downto 0) := x"00";
10      constant Device_Id_BROADCAST                    : std_logic_vector(7 downto 0) := x"ff";
```

generated!

Contents

Demo

Please step back

- Small standalone binary, no purpose
- Task: Fill payload, prepare header, encode package, print content
- Uses `TestDevice` and `struct RepresentationTestHeader`

demo/CMakeLists.txt

demo/ndlcom_demo.cpp

demo/CMakeLists.txt

```
# cmake: boilerplate straight from hell
cmake_minimum_required(VERSION 2.8)
project(ndlcom_demo)

# this will select which RobotConfig xml-database to use
set(ROBOTCONFIG CommonConfig)
# this should point to the root-dir of a "ndlcom" bootstrap
set(NDLCOM_ROOT_DIR "$ENV{HOME}/dev2")

# adding subdirs; testing for TARGET to prevent multiple inclusion:
if(NOT TARGET ndlcom)
    add_subdirectory(${NDLCOM_ROOT_DIR}/lib/ndlcom ndlcom)
endif(NOT TARGET ndlcom)
if(NOT TARGET representations)
    add_subdirectory(${NDLCOM_ROOT_DIR}/lib/representations representations)
endif(NOT TARGET representations)
if(NOT TARGET RobotConfig)
    add_subdirectory(${NDLCOM_ROOT_DIR}/lib/RobotConfig RobotConfig)
endif(NOT TARGET RobotConfig)

# pkg-config to obtain -I and -L/-l flags. search CMAKE_BINARY_DIR for "*-uninstalled.pc" files
SET(ENV{PKG_CONFIG_PATH} "${CMAKE_BINARY_DIR}:$ENV{PKG_CONFIG_PATH}")
find_package(PkgConfig)
pkg_check_modules(${PROJECT_NAME}_PKGCONFIG REQUIRED representations RobotConfig ndlcom)
# use pkg-configs output:
include_directories(${${PROJECT_NAME}_PKGCONFIG_INCLUDE_DIRS})
link_directories(${${PROJECT_NAME}_PKGCONFIG_LIBRARY_DIRS})
add_definitions(${${PROJECT_NAME}_PKGCONFIG_CFLAGS_OTHER})

# and finally do something usefull:
add_executable(${PROJECT_NAME} ndlcom_demo.cpp)
target_link_libraries(${PROJECT_NAME} ${${PROJECT_NAME}_PKGCONFIG_LIBRARIES})
```

demo/CMakeLists.txt

```
1  # cmake: boilerplate straight from hell
2  cmake_minimum_required(VERSION 2.8)
3  project(ndlcom_demo)
4
5  # this will select which RobotConfig xml-database to use
6  set(ROBOTCONFIG CommonConfig)
7  # this should point to the root-dir of a "ndlcom" bootstrap
8  set(NDLCOM_ROOT_DIR "$ENV{HOME}/dev2")
9
10 # adding subdirs; testing for TARGET to prevent multiple inclusion:
11 if(NOT TARGET ndlcom)
12     add_subdirectory(${NDLCOM_ROOT_DIR}/lib/ndlcom ndlcom)
13 endif(NOT TARGET ndlcom)
14 if(NOT TARGET representations)
15     add_subdirectory(${NDLCOM_ROOT_DIR}/lib/representations representations)
16 endif(NOT TARGET representations)
17 if(NOT TARGET RobotConfig)
18     add_subdirectory(${NDLCOM_ROOT_DIR}/lib/RobotConfig RobotConfig)
19 endif(NOT TARGET RobotConfig)
20
21 # pkg-config to obtain -I and -L/-l flags. search CMAKE_BINARY_DIR for "*-uninstalled.pc" files
22 SET(ENV{PKG_CONFIG_PATH} "${CMAKE_BINARY_DIR}:$ENV{PKG_CONFIG_PATH}")
23 find_package(PkgConfig)
24 pkg_check_modules(${PROJECT_NAME}_PKGCONFIG REQUIRED representations RobotConfig ndlcom)
25 # use pkg-configs output:
26 include_directories(${${PROJECT_NAME}_PKGCONFIG_INCLUDE_DIRS})
27 link_directories(${${PROJECT_NAME}_PKGCONFIG_LIBRARY_DIRS})
28 add_definitions(${${PROJECT_NAME}_PKGCONFIG_CFLAGS_OTHER})
29
30 # and finally do something usefull:
31 add_executable(${PROJECT_NAME} ndlcom_demo.cpp)
32 target_link_libraries(${PROJECT_NAME} ${${PROJECT_NAME}_PKGCONFIG_LIBRARIES})
```

25 / 28

demo/CMakeLists.txt

```
1  # cmake: boilerplate straight from hell
2  cmake_minimum_required(VERSION 2.8)
3  project(ndlcom_demo)
4
5  # this will select which RobotConfig xml-database to use
6  set(ROBOTCONFIG CommonConfig)
7  # this should point to the root-dir of a "ndlcom" bootstrap
8  set(NDLCOM_ROOT_DIR "$ENV{HOME}/dev2")
9
10 # adding subdirs; testing for TARGET to prevent multiple inclusion:
11 if(NOT TARGET ndlcom)
12     add_subdirectory(${NDLCOM_ROOT_DIR}/lib/ndlcom ndlcom)
13 endif(NOT TARGET ndlcom)
14 if(NOT TARGET representations)
15     add_subdirectory(${NDLCOM_ROOT_DIR}/lib/representations representations)
16 endif(NOT TARGET representations)
17 if(NOT TARGET RobotConfig)
18     add_subdirectory(${NDLCOM_ROOT_DIR}/lib/RobotConfig RobotConfig)
19 endif(NOT TARGET RobotConfig)
20
21 # pkg-config to obtain -I and -L/-l flags. search CMAKE_BINARY_DIR for "*-uninstalled.pc" files
22 SET(ENV{PKG_CONFIG_PATH} "${CMAKE_BINARY_DIR}:$ENV{PKG_CONFIG_PATH}")
23 find_package(PkgConfig)
24 pkg_check_modules(${PROJECT_NAME}_PKGCONFIG REQUIRED representations RobotConfig ndlcom)
25 # use pkg-configs output:
26 include_directories(${${PROJECT_NAME}_PKGCONFIG_INCLUDE_DIRS})
27 link_directories(${${PROJECT_NAME}_PKGCONFIG_LIBRARY_DIRS})
28 add_definitions(${${PROJECT_NAME}_PKGCONFIG_CFLAGS_OTHER})
29
30 # and finally do something usefull:
31 add_executable(${PROJECT_NAME} ndlcom_demo.cpp)
32 target_link_libraries(${PROJECT_NAME} ${${PROJECT_NAME}_PKGCONFIG_LIBRARIES})
```

25 / 28

demo/CMakeLists.txt

```
# cmake: boilerplate straight from hell
cmake_minimum_required(VERSION 2.8)
project(ndlcom_demo)

# this will select which RobotConfig xml-database to use
set(ROBOTCONFIG CommonConfig)
# this should point to the root-dir of a "ndlcom" bootstrap
set(NDLCOM_ROOT_DIR "$ENV{HOME}/dev2")

# adding subdirs; testing for TARGET to prevent multiple inclusion:
if(NOT TARGET ndlcom)
    add_subdirectory(${NDLCOM_ROOT_DIR}/lib/ndlcom ndlcom)
endif(NOT TARGET ndlcom)
if(NOT TARGET representations)
    add_subdirectory(${NDLCOM_ROOT_DIR}/lib/representations representations)
endif(NOT TARGET representations)
if(NOT TARGET RobotConfig)
    add_subdirectory(${NDLCOM_ROOT_DIR}/lib/RobotConfig RobotConfig)
endif(NOT TARGET RobotConfig)

# pkg-config to obtain -I and -L/-l flags. search CMAKE_BINARY_DIR for "*-uninstalled.pc" files
SET(ENV{PKG_CONFIG_PATH} "${CMAKE_BINARY_DIR}:$ENV{PKG_CONFIG_PATH}")
find_package(PkgConfig)
pkg_check_modules(${PROJECT_NAME}_PKGCONFIG REQUIRED representations RobotConfig ndlcom)
# use pkg-configs output:
include_directories(${${PROJECT_NAME}_PKGCONFIG_INCLUDE_DIRS})
link_directories(${${PROJECT_NAME}_PKGCONFIG_LIBRARY_DIRS})
add_definitions(${${PROJECT_NAME}_PKGCONFIG_CFLAGS_OTHER})

# and finally do something usefull:
add_executable(${PROJECT_NAME} ndlcom_demo.cpp)
target_link_libraries(${PROJECT_NAME} ${${PROJECT_NAME}_PKGCONFIG_LIBRARIES})
```

25 / 28

demo/ndlcom_demo.cpp

```
#include "RobotConfig/Devices.h"
#include "ndlcom/Encoder.h"
#include "representations/id.h"
#include "representations/TestData.h"

#include <iomanip>
#include <iostream>

int main(int argc, char *argv[]) {
    // initialize ndlcom's static datastructure once:
    ndlcomHeaderConfigDefaultSenderId(robotconfig::TestDevice);
    // create c-struct, manually initialize mBase:
    struct RepresentationsTestData data;
    data.mBase.mId = REPRESENTATIONS_REPRESENTATION_ID_RepresentationsTestData;
    // stating the obvious:
    data.theAnswer = 0xabbabeef;

    // prepare paket header, automatically fill pkt-counter:
    struct NDLComHeader hdr;
    ndlcomHeaderPrepare(&hdr, robotconfig::DMSBoard, sizeof(data));
    // encode message into outgoing buffer:
    char buffer[NDLCOM_MAX_ENCODED_MESSAGE_SIZE];
    size_t len = ndlcomEncode(buffer, sizeof(buffer), &hdr, &data);
    // output the data:
    std::cout << "encoded packet: " << std::hex << std::setfill('0');
    for (size_t idx = 0; idx < len; ++idx) {
        std::cout << " 0x" << std::setw(2) << (int)(buffer[idx] & 0xff);
    }
    std::cout << "\n";

    return 0;
}
```

26 / 28

demo/ndlcom_demo.cpp

```
1   #include "RobotConfig/Devices.h"
2   #include "ndlcom/Encoder.h"
3   #include "representations/id.h"
4   #include "representations/TestData.h"
5
6   #include <iomanip>
7   #include <iostream>
8
9   int main(int argc, char *argv[]) {
10      // initialize ndlcom's static datastructure once:
11      ndlcomHeaderConfigDefaultSenderId(robotconfig::TestDevice);
12      // create c-struct, manually initialize mBase:
13      struct RepresentationsTestData data;
14      data.mBase.mId = REPRESENTATIONS_REPRESENTATION_ID_RepresentationsTestData;
15      // stating the obvious:
16      data.theAnswer = 0xabbabeef;
17
18      // prepare paket header, automatically fill pkt-counter:
19      struct NDLComHeader hdr;
20      ndlcomHeaderPrepare(&hdr, robotconfig::DMSBoard, sizeof(data));
21      // encode message into outgoing buffer:
22      char buffer[NDLCOM_MAX_ENCODED_MESSAGE_SIZE];
23      size_t len = ndlcomEncode(buffer, sizeof(buffer), &hdr, &data);
24      // output the data:
25      std::cout << "encoded packet: " << std::hex << std::setfill('0');
26      for (size_t idx = 0; idx < len; ++idx) {
27          std::cout << " 0x" << std::setw(2) << (int)(buffer[idx] & 0xff);
28      }
29      std::cout << "\n";
30
31      return 0;
32  }
```


demo/ndlcom_demo.cpp

```
1   #include "RobotConfig/Devices.h"
2   #include "ndlcom/Encoder.h"
3   #include "representations/id.h"
4   #include "representations/TestData.h"
5
6   #include <iomanip>
7   #include <iostream>
8
9   int main(int argc, char *argv[]) {
10      // initialize ndlcom's static datastructure once:
11      ndlcomHeaderConfigDefaultSenderId(robotconfig::TestDevice);
12      // create c-struct, manually initialize mBase:
13      struct RepresentationsTestData data;
14      data.mBase.mId = REPRESENTATIONS_REPRESENTATION_ID_RepresentationsTestData;
15      // stating the obvious:
16      data.theAnswer = 0xabbabeef;
17
18      // prepare paket header, automatically fill pkt-counter:
19      struct NDLComHeader hdr;
20      ndlcomHeaderPrepare(&hdr, robotconfig::DMSBoard, sizeof(data));
21      // encode message into outgoing buffer:
22      char buffer[NDLCOM_MAX_ENCODED_MESSAGE_SIZE];
23      size_t len = ndlcomEncode(buffer, sizeof(buffer), &hdr, &data);
24      // output the data:
25      std::cout << "encoded packet: " << std::hex << std::setfill('0');
26      for (size_t idx = 0; idx < len; ++idx) {
27          std::cout << " 0x" << std::setw(2) << (int)(buffer[idx] & 0xff);
28      }
29      std::cout << "\n";
30
31      return 0;
32  }
```

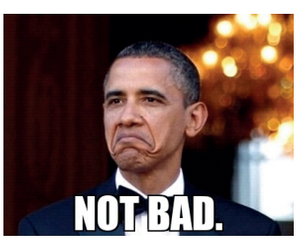
26 / 28

demo/ndlcom_demo.cpp

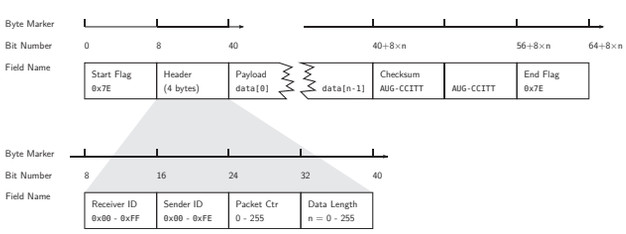
```cpp
#include "RobotConfig/Devices.h"
#include "ndlcom/Encoder.h"
#include "representations/id.h"
#include "representations/TestData.h"

#include <iomanip>
#include <iostream>

int main(int argc, char *argv[]) {
    // initialize ndlcom's static datastructure once:
    ndlcomHeaderConfigDefaultSenderId(robotconfig::TestDevice);
    // create c-struct, manually initialize mBase:
    struct RepresentationsTestData data;
    data.mBase.mId = REPRESENTATIONS_REPRESENTATION_ID_RepresentationsTestData;
    // stating the obvious:
    data.theAnswer = 0xabbabeef;

    // prepare paket header, automatically fill pkt-counter:
    struct NDLComHeader hdr;
    ndlcomHeaderPrepare(&hdr, robotconfig::DMSBoard, sizeof(data));
    // encode message into outgoing buffer:
    char buffer[NDLCOM_MAX_ENCODED_MESSAGE_SIZE];
    size_t len = ndlcomEncode(buffer, sizeof(buffer), &hdr, &data);
    // output the data:
    std::cout << "encoded packet: " << std::hex << std::setfill('0');
    for (size_t idx = 0; idx < len; ++idx) {
        std::cout << " 0x" << std::setw(2) << (int)(buffer[idx] & 0xff);
    }
    std::cout << "\n";

    return 0;
}
```


Output

No wrapper Makefile...

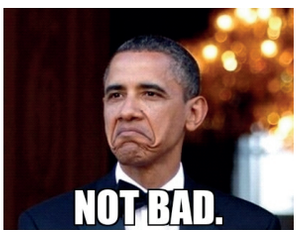



```
$ cd ~/talks/projectday_ndlcom2015/demo
$ ls
CMakeLists.txt   ndlcom_demo.cpp
$ mkdir -p build && cd build
$ cmake ..
...
$ make
...
$ ./ndlcom_demo
encoded packet: 0x7e 0x68 0x09 0x00 0x0d 0xe7 0x00 0x00 0x00 \
    0x00 0x00 0x00 0x00 0x00 0xef 0xbe 0xba 0xab 0xc4 0x71 0x7e
```

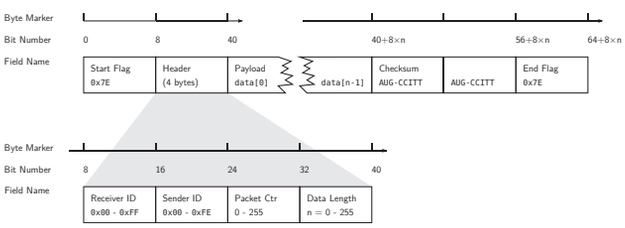


Output

No wrapper Makefile...

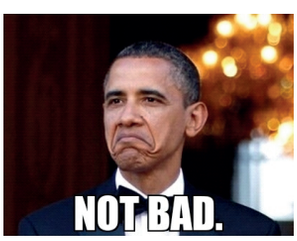



```
$ cd ~/talks/projectday_ndlcom2015/demo
$ ls
CMakeLists.txt   ndlcom_demo.cpp
$ mkdir -p build && cd build
$ cmake ..
...
$ make
...
$ ./ndlcom_demo
encoded packet: 0x7e 0x68 0x09 0x00 0x0d 0xe7 0x00 0x00 0x00 \
    0x00 0x00 0x00 0x00 0x00 0xef 0xbe 0xba 0xab 0xc4 0x71 0x7e
```

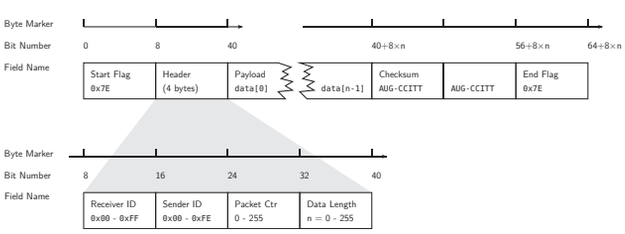


Output

No wrapper Makefile...

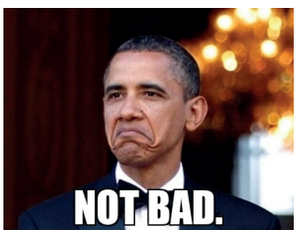



```
$ cd ~/talks/projectday_ndlcom2015/demo
$ ls
CMakeLists.txt   ndlcom_demo.cpp
$ mkdir -p build && cd build
$ cmake ..
...
$ make
...
$ ./ndlcom_demo
encoded packet: 0x7e 0x68 0x09 0x00 0x0d 0xe7 0x00 0x00 0x00 \
    0x00 0x00 0x00 0x00 0x00 0xef 0xbe 0xba 0xab 0xc4 0x71 0x7e
```

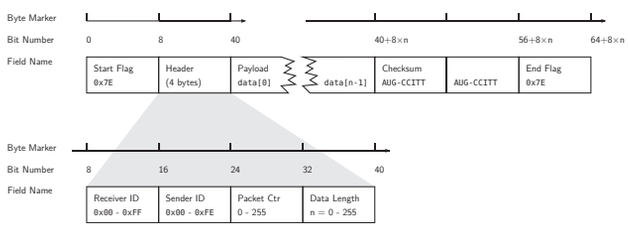


Output

No wrapper Makefile...




```
$ cd ~/talks/projectday_ndlcom2015/demo
$ ls
CMakeLists.txt   ndlcom_demo.cpp
$ mkdir -p build && cd build
$ cmake ..
...
$ make
...
$ ./ndlcom_demo
encoded packet: 0x7e 0x68 0x09 0x00 0x0d 0xe7 0x00 0x00 0x00 \
    0x00 0x00 0x00 0x00 0x00 0xef 0xbe 0xba 0xab 0xc4 0x71 0x7e
```



27 / 28

Questions? Remarks?

28 / 28

2.3 ‘NDLCom on FPGAs’ (ED-T-03)

Tobias Stark[(1)]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: tobias.stark@dfki.de

Abstract

This talk presents the FPGA development side of NDLCom. Its structure as well as the ways to integrate it into own code are presented. The presentation finishes with a demonstration on the effectiveness of routing packets within a hardware architecture that relies on NDLCom.

Gliederung

- HSCom
- VHDL-Implementierung NLDCom
- ZynqBrain
- Hardwaredemo Überblick

Gliederung

- HSCom
- VHDL-Implementierung NLDCom
- ZynqBrain
- Hardwaredemo Überblick

HSCom

- Entwickelt von Florian Hühn
- Motivation
  - Bedarf einer schnellen Datenübertragung mit hoher Bandbreite
  - Möglichst geringer Hardwareaufwand

Universität Bremen

3

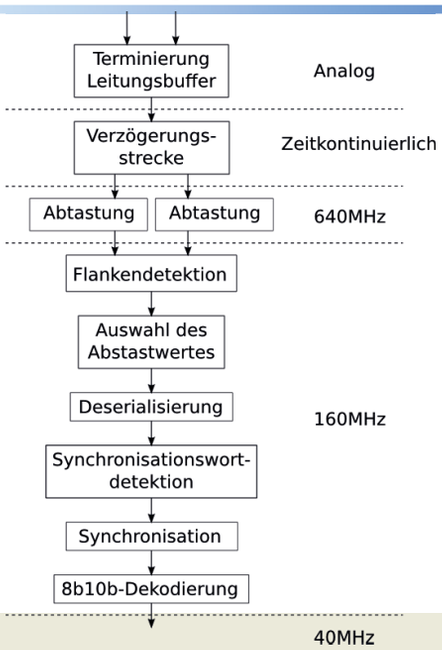
HSCom – Eigenschaften

- Eigenschaften
  - FPGA/VHDL Implementierung
  - Asynchrone, serielle Kommunikation
  - Übertragungsstandard: LVDS
  - DC-freie Leitungskodierung
  - Galvanische Isolation der Datenleitungen
  - 8b10b Kodierung (256 Datenwörter und 11 nutzbare Kontrollwörter)
  - Begrenzte Fehlererkennung
  - Abhängig vom eingesetzten Kabel bis zu 500 Mbit/s (50 MB/s) (wir verwenden 320Mbit/s)

Universität Bremen

4

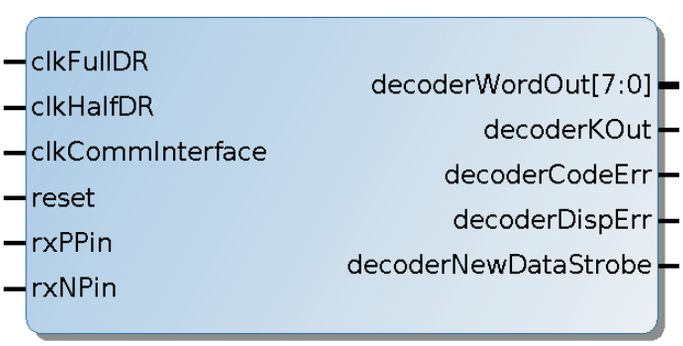
HSCom – Empfänger



Universität Bremen

5

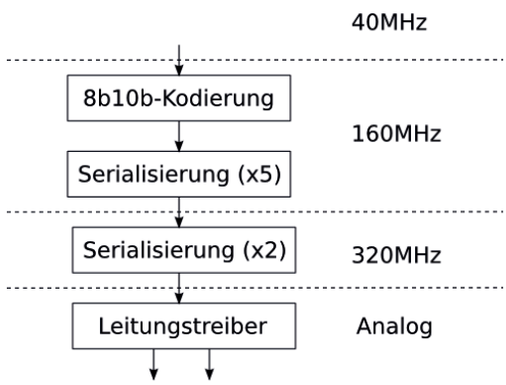
HSCom – Empfänger



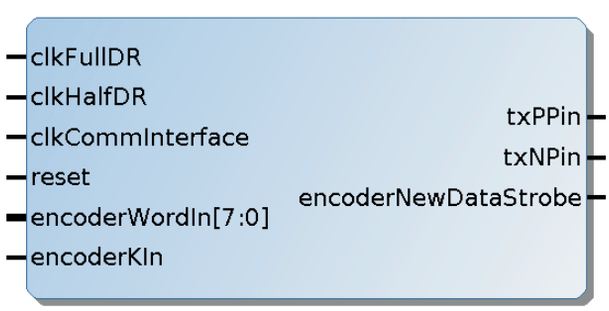
Universität Bremen

6

HSCom – Sender

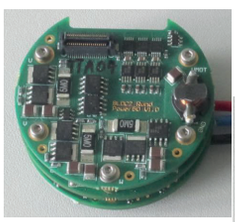


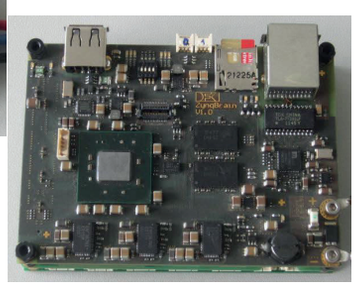
HSCom – Sender

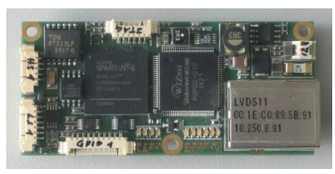


HSCom – Vorhandene Hardware

- Z.Z. verfügbare Hardware, die HSCom verwendet:
  - BLDC V4
  - ZynqBrain (mit HSCom-Extension Board)
  - HSCom-Ethernet Konverter



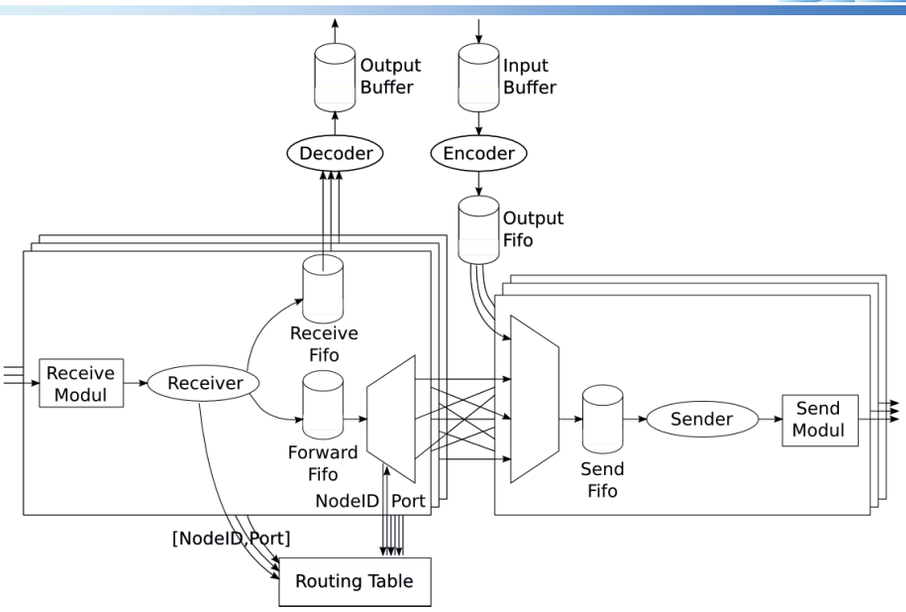




Gliederung

- HSCom
- VHDL-Implementierung NLDCom
- ZynqBrain
- Hardwaredemo Überblick

NDLCom – Funktionsprinzip



Universität Bremen

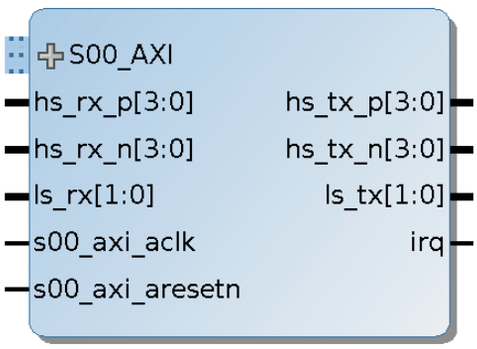
11

NDLCom

Universität Bremen

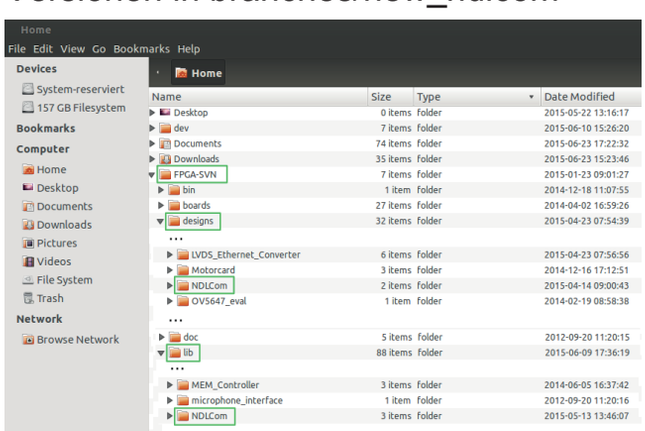
12

NDLCom (ZynqBrain)



NDLCom – SVN

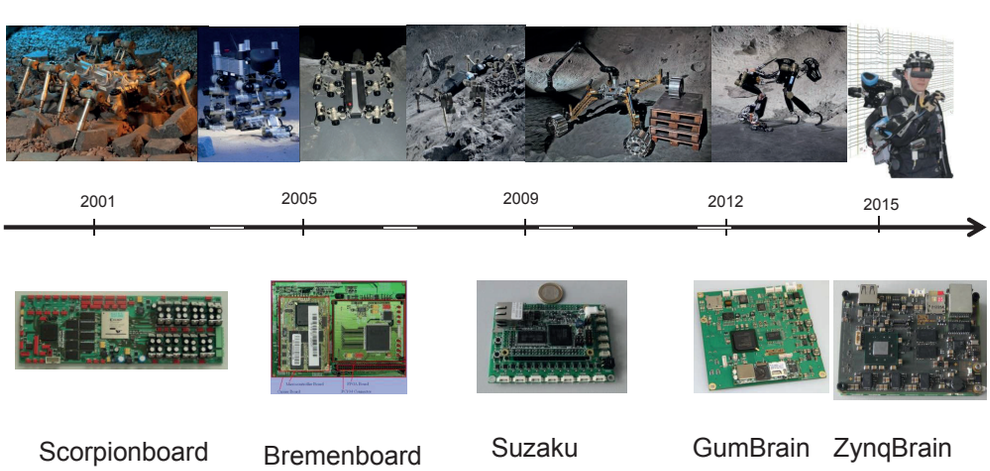
- FPGA-SVN
  - https://svn.hb.dfki.de/FPGA
- Aktuelle Versionen in branches/new_ndlcom



Gliederung

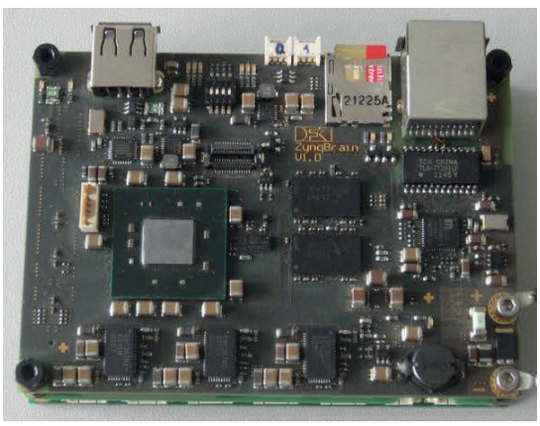
- HSCom
- VHDL-Implementierung NLDCom
- ZynqBrain
- Hardwaredemo Überblick

ZynqBrain – History



ZynqBrain - Übersicht

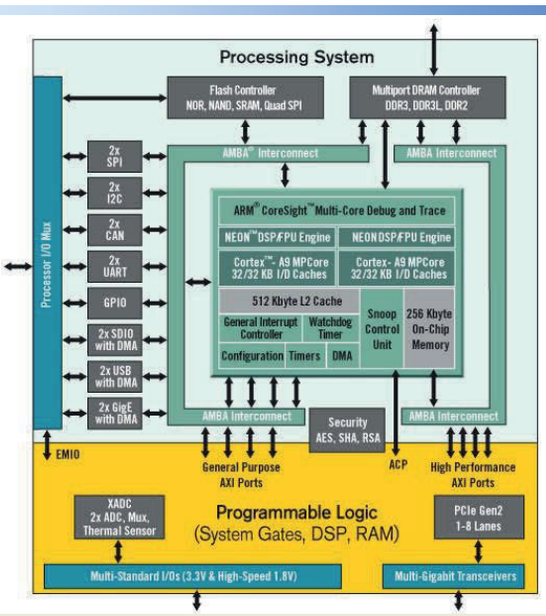
- Xilinx Zynq 7000 SoC
  - ARM A9 DualCore (bis zu 1GHz)
  - Programmable Logic
- Gbit-Ethernet
- DDR3-SDRAM
- SD-Card
- USB
- Debug UARTs
- Zynq-2-Zynq Interconnect
- BrainBus
- 2.5V-5.5V Input



Universität Bremen

17

ZynqBrain – Zynq 7000 Soc

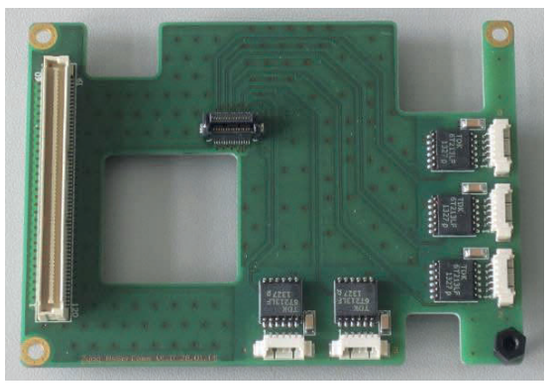


Universität Bremen

18

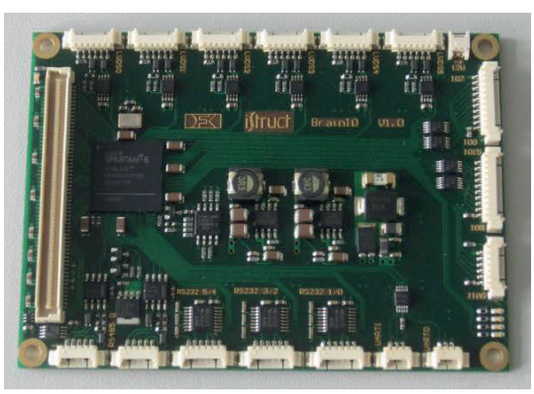
ZynqBrain – HSCom-Platine

- Kommunikation über Zynq2Zynq Interconnect
- Pro ZynqBrain 2 dieser HSCom-Platinen verwendbar (oben und unten)
- 5x HSCom (nur 4 nutzbar)
- Keine aktive Logik
- Keine eigene Stromversorgung



ZynqBrain – iStruct-Extension

- Kommunikation über BrainBus
- 22x LVDS (low-speed)
- 6x RS232
- 2x RS485
- 2x UART
- 16x GPIO
- Spartan6
- 12V Input

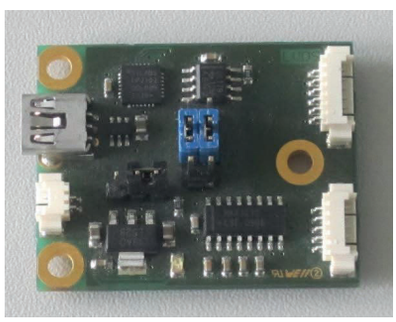


Gliederung

- HSCom
- VHDL-Implementierung NLDCom
- ZynqBrain
- Hardwaredemo Überblick

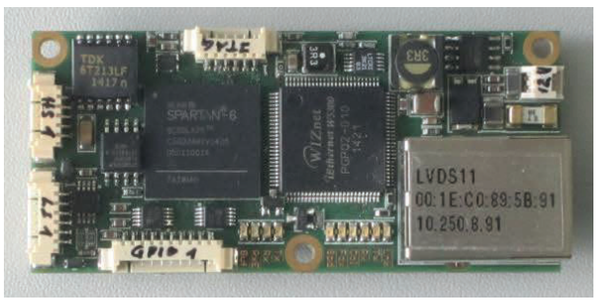
LVDS-USB Konverter

- Konvertierung des Datenstroms zwischen USB, LVDS, RS232 und UART Signalen (über Jumper auswählbar)
- Versorgung über USB oder extern
- Maximale Geschwindigkeit 921600baud



HSCom-Ethernet Konverter

- Konvertierung zwischen HSCom Datenstrom und TCP oder UDP Paketen
- 2x HSCom (z.Z. nur einer verwendbar)
- 2x LVDS (low-speed, z.Z. einer für Konfiguration verwendbar)

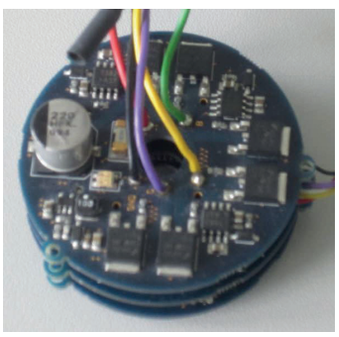


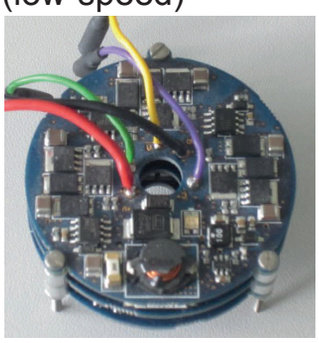
Universität Bremen

23

Motorplatinen BLDC V2 & V3

- Motorplatinen für die DFKI-Robodrive Gelenke
- V2: Spartan3-1000
- V3: Spartan6-45
- 2x LVDS (low-speed)



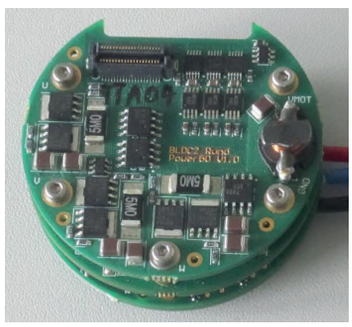


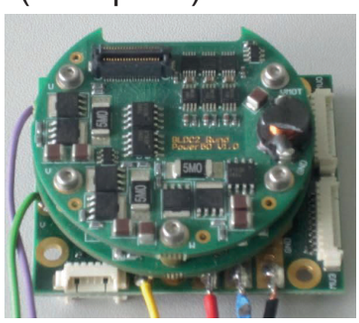
Universität Bremen

24

Motorplatine BLDC V4

- Motorplatinen für die DFKI-Robodrive Gelenke
- Spartan6-45
- 3x HSCom
- 2x LVDS (low-speed)



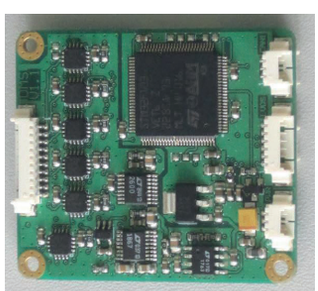


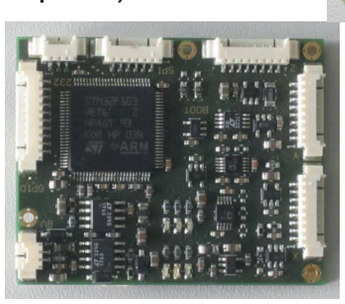
Universität Bremen

25

Sensorboards DMS, MDAQ

- Sensorplatinen zum Auslesen von ForceTorque- oder ähnlichen Sensoren
- STM32-F1
- 2x LVDS (low-speed)



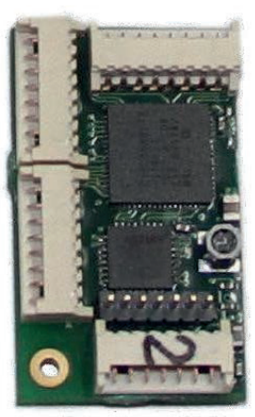


Universität Bremen

26

Sensorboard µDMS

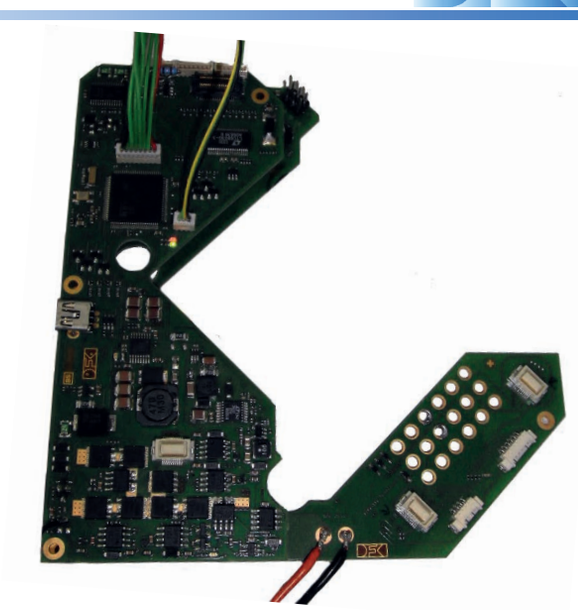
- Sensorplatine zum Auslesen von ForceTorque- oder ähnlichen Sensoren
- PSoC5
- 1x LVDS (low-speed)



Universität Bremen 27

Modulelektronik

- Modulelektronik der TransTerrA-Module
- STM32-F4
- 1x UART
- 2x RS422
- 1x USB



Universität Bremen 28

2.4 'Next Energy Source Bus (NESB)' (ED-P-01)

Patrick Schöberl[(1)]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: patrick.schoeberl@dfki.de

Abstract

The question on how to distribute energy within a robotic system is one of the key questions on the roadmap of the team Hardware Architectures. This poster presents a solution for an energy source bus that takes into account the distribution of energy sources and the acquisition of energy from various distributed energy harvesting methods.

German Research Center for Artificial Intelligence

Next Energy Source Bus (NESB)

A new concept of powering mobile robots by different energy sources

Patrick Schöberl, 2015

The "new" system concepts

- Based on proven concepts of power buses for satellite
- Highly modular
  - Redundancies available
- Platform for new energy sources
  - E.g. solar cells
  - Different kinds of batteries and technologies
- Energy recovery feasible
- Constant regulated bus voltage – unaffected by battery state of charge
- Simplified interface
  - Each module has a CAN-Bus interface and a power connector – that's it!
  - Implementable in future systems to gain interchangeable power sources

The basic Power Modules

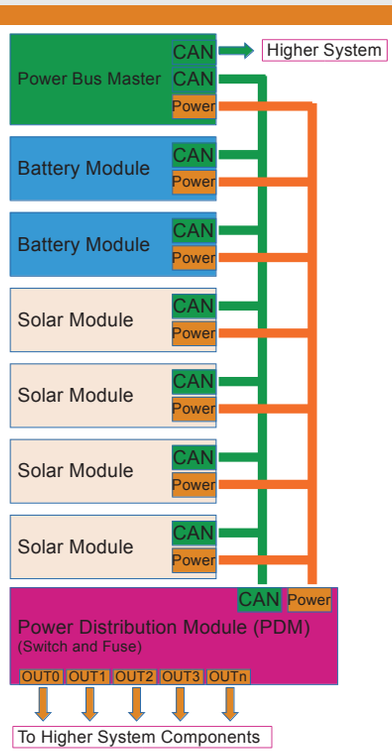
Power Bus Master (PBM)
- Bridge to higher system components
  - E.g. higher system can not control the PDU to activate a consumer load, but it can "ask" the PBM to activate it in case there is enough power
- Control of all connected power modules
- Control of energy flow direction
  - E.g. if more energy is delivered by solar cells than needed, batteries can be charged
- Health management of the entire power system
  - E.g. state of charge, energy consumption

Battery Module (BM)
- Contains battery, charge and discharge regulator
  - Balancing can be implemented if needed
- Electronics tailored for each type of battery or chemistry
- Bus voltage is regulated by a discharge regulator
  - In case the state of charge is low, the bus voltage will remain constant
  - Possibility to apply variable numbers of cells
- Different modules can be connected in parallel to increase e.g. the runtime of the system or reliability
- Controlled by PBM via CAN-Bus

Power Distribution Unit (PDU)
- Acts as an On/Off switch for all loads (motors, sensors, computers)
- Usable as an electronic fuse
- Controlled by PBM via CAN-Bus

Solar Module (SM)
- MPP tracking of solar cells/panels
- Multiple SMs can be connected in parallel
  - E.g. increase the reliability or power
- Controlled by PBM via CAN-Bus

And many more options
- Super capacitor module to increase peak power or energy recovery
- Redundant modules to increase the reliability or runtime
- BMs for different type of batteries
  - E.g. LiPo or LiFePo4 or primary battery for emergency to power the PBM if rechargeable batteries are completely discharged



Example of a NSEB System (Source: P. Schöberl)

A few facts to rethink about

The bus voltage should be decreased from about 48V to a common 24V (Industrial) or 28V (Space and Aerospace up to 1.5kW) - at least for a demonstrator. There is a multitude of interesting highly integrated semiconductors for 24V systems like intelligent and protected high side switches (act as a switch and fuse) and switching regulators. Power supplies for computers and a lot of peripherals are available for 24V as well – no additional DC/DC converter will be needed.

First concepts (like a solar based MPP tracker) are in preparation by students bachelor thesis right now!
A first modular concepts of battery powered systems will be used for project TransTerrA Sherpa.

A sample Application
A solar powered base camp: A base camp equipped with a NESB can harvest sufficient more power than it consumes such that additional energy can be stored in BMs. Mobile robots with NESB can recharge BMs or even change their discharged BMs with recharged BMs. The basecamp could than recharge empty BMs as soon an energy is available.

And finally – a full demonstrator set-up should be designed for further investigations.

Universität Bremen

Contact:
DFKI Bremen & University of Bremen
Robotics Innovation Center

Director: Prof. Dr. Frank Kirchner
E-mail: robotics@dfki.de
Website: www.dfki.de/robotics

2.5 'Cable Tester' (ED-P-02)

Jakob Wehnes [1]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: jakob.wehnes@dfki.de

Abstract

Complex robotic systems consist of a considerable amount of processing units which require cabeling to connect with neighbouring processing units for communication and power transmission. A Cable Tester has been developed based on this scenario that allows the testing of the functionality of cables in an easy and simple way.

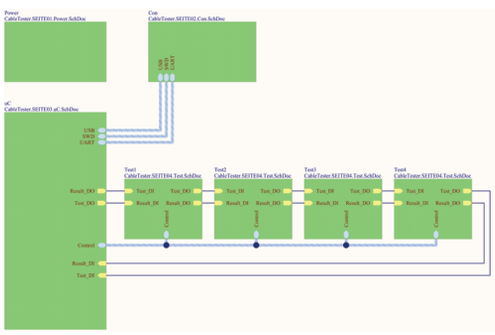
German Research Center for Artificial Intelligence

Cable Tester

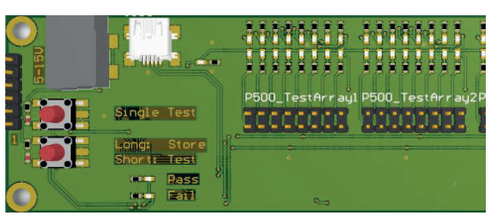
Modular Cable Tester for various purposes.

Main Features

- Modular Design.
  - Multiple Test units can be connected together.
  - Each unit can test 8 strands of wire.
- Connectivity
  - Power; Can be powered by USB or 5-15V external PSU
  - USB
  - UART
- Remote controllable
  - Can be remote controlled over UART or USB
- Detectable faults
  - Shorts
  - Open connection
  - Pin swaps



Block diagram of CableTester with 4 Test units



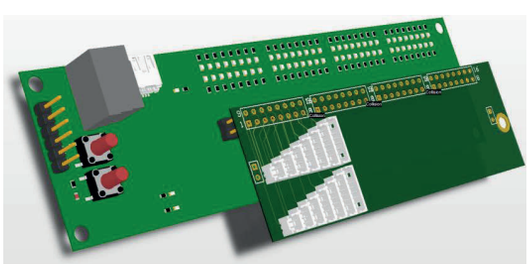
Main PCB with user interface

Functions

- Each cable strand is tested individually
- LED will signal which strand is tested
- Slow test
  - Slow test for human supervision
  - Tests each strand of wire at a slow speed
  - Using LEDs for feedback
- Fast test
  - Save test results for a working cable
  - Tests against recorded results fast
  - Pass/Fail result

Versatile

Main PCB uses 2.54mm header for the test interface.
Using different header modules every possible cable can be tested.



Example of header module with Molex 1,25mm headers

Universität Bremen

Contact:
DFKI Bremen & University of Bremen
Robotics Innovation Center

Director: Prof. Dr. Frank Kirchner
E-mail: robotics@dfki.de
Website: www.dfki.de/robotics

2.6 'Eagle as Tool for wiring Diagrams' (ED-P-03)

Christian Schoo [1]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: christian.schoo@dfki.de

Abstract

The standardization of wiring diagrams for robotic systems is addressed with this poster. A solution based on the layouting software Eagle is presented which can be used as a tool for wiring diagrams as well. The power of this tool is demonstrated at the example of a wiring diagram for the Wally crawler.

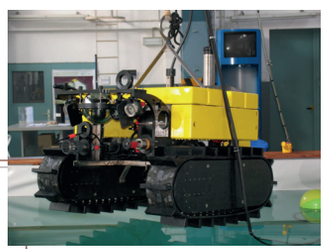
German Research Center for Artificial Intelligence

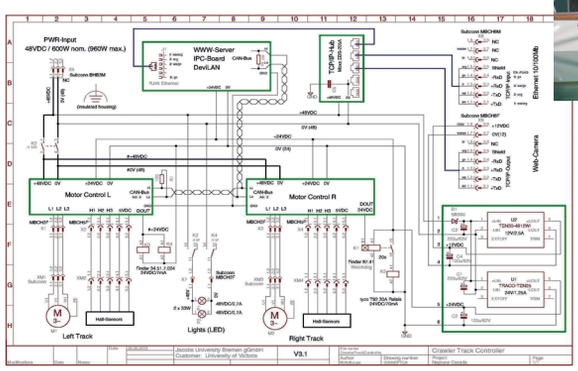
“EAGLE” as Tool for wiring Diagrams

The “EAGLE” schematic editor with specific libraries

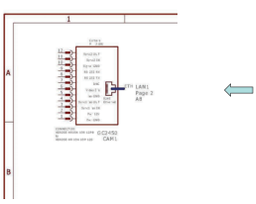
The “EAGLE” schematic editor as tool for wiring diagrams

By using specific libraries for robotic applications the schematic Editor can also be used for cable planning.



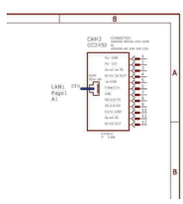


Example of a wiring diagram using Eagle for the Wally-Crawler



Page1

The coordinate system on the sheet allows easy assignment of cable connections.



Page2

Current status of documentation:

Each project is using own tools for cable planning and documentation. Detailed wiring information is done by Excel or only handwritten. Visual overviews do not replace technical documentation.

Future Goals:

- Detailed graphical wiring diagrams including complete information about every connection.
- Only one kind of document is needed for complete technical documentation.
- Standardized cross-project documentation.

Features:

- inexpensive
  - ca. 100€ per license
  - only the schematic editor is needed
  - no need for a maintenance contract
- easy to use
- hierarchical library
  - symbols library containing electrical symbols
  - device library containing symbol elements
- library grows with every new project

Universität Bremen

Contact:
DFKI Bremen & University of Bremen
Robotics Innovation Center

Director: Prof. Dr. Frank Kirchner
E-mail: robotics@dfki.de
Website: www.dfki.de/robotics

3 'Mechatronic Design'

3.1 'A short Introduction of the CASCADE Catheter Driver and the INCASS MagnetCrawler II' (MD-T-01)

Felix Bernhard[(1)]

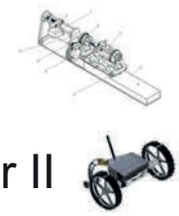
(1) Arbeitsgruppe Robotik, Universität Bremen, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: felix.bernhard@dfki.de

Abstract

In this presentation a catheter-driver developed for the FP7-Project: CASCADE will be introduced to the audience. The catheter-driver is beeing used to test and proove its steering algorythms in artificial test-enviroments.

Also the new design and challenges in developing a new magnet crawler and new wheels for the project INCASS will be presented. The Crawler got updates for autonoumous driving and a new wheel design, which reduces the loss of magnets.



A short Introduction of the CASCADE Catheter Driver and the INCASS MagnetCrawler II

by Felix Bernhard

DFKI Bremen & Universität Bremen
Robotics Innovation Center
Director: Prof. Dr. Frank Kirchner
www.dfki.de/robotics
robotics@dfki.de

Universität Bremen

CASCADE

CASCADE

- *COGNITIVE AUTONOMOUS CATHETERS OPERATING IN DYNAMIC ENVIROMENTS*

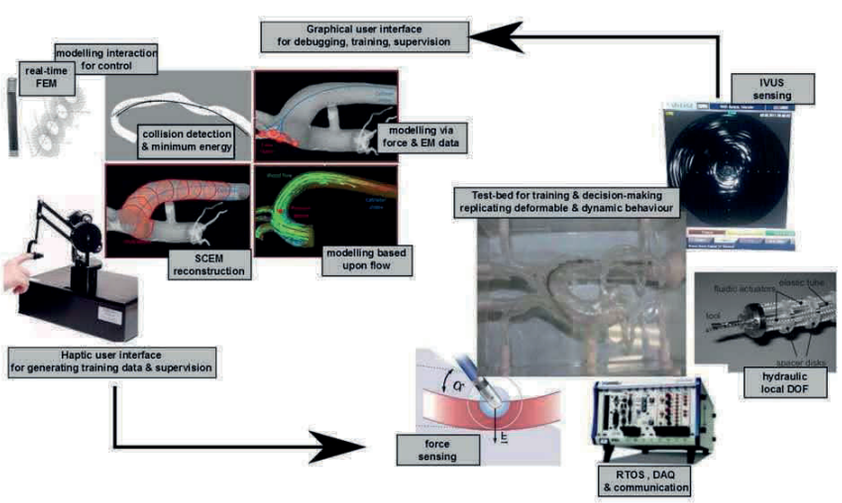
CASCADE

G.A. 601021

Universität Bremen

3

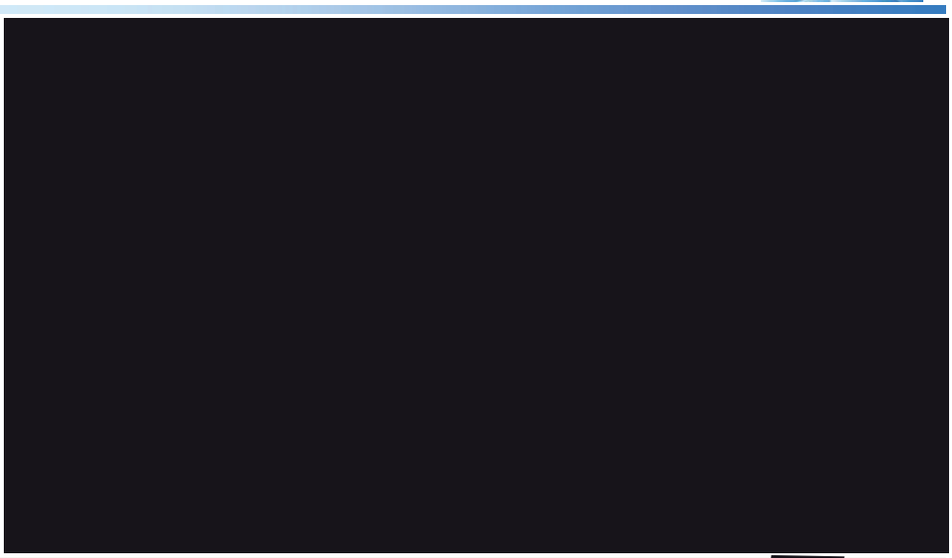
CASCADE



Universität Bremen

4

State of the Art in Catheter Robotics



Universität Bremen

5

State of the Art in Catheter Robotics

(PictureSource: http://www.intechopen.com/source/html/6760)

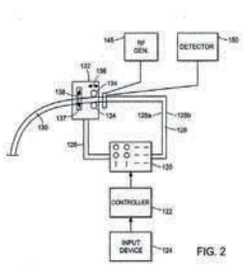


FIG. 2

(Source: Patent US 8684952 B2)

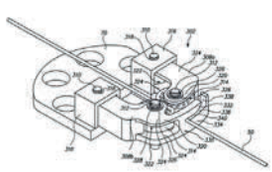
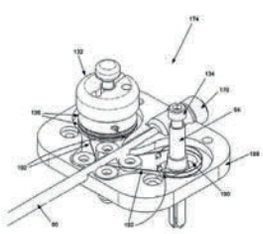


FIG. 31

(Source: Patent US 20140277334 A1)

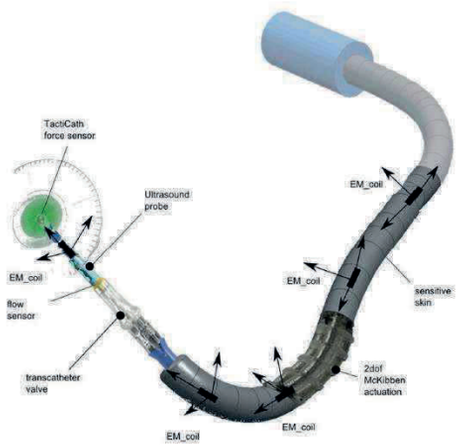


(Source: Patent US 8021326 B2)

Universität Bremen

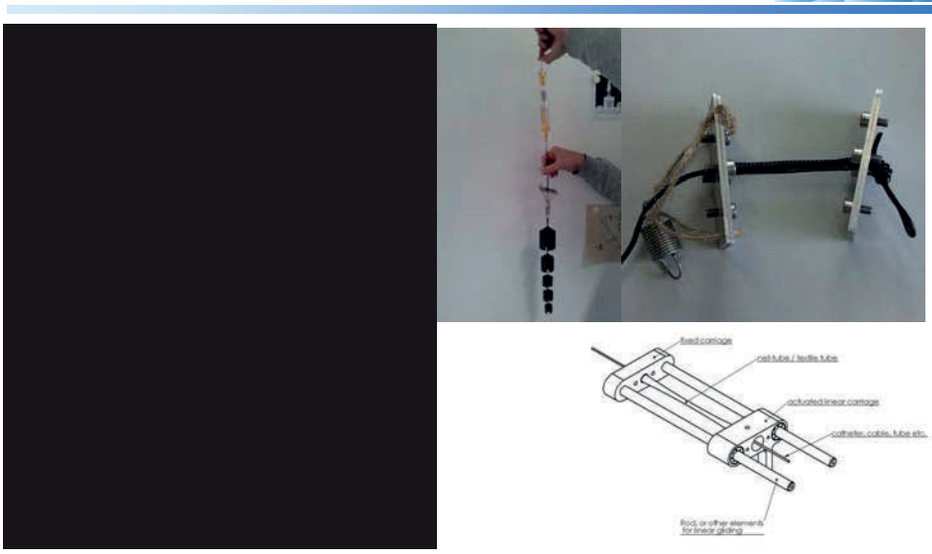
6

Catheters of the Future



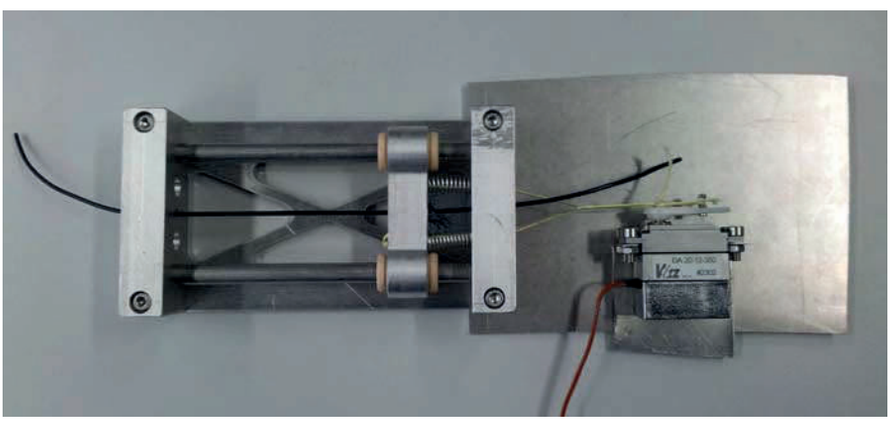
Universität Bremen 7

Different Gripping



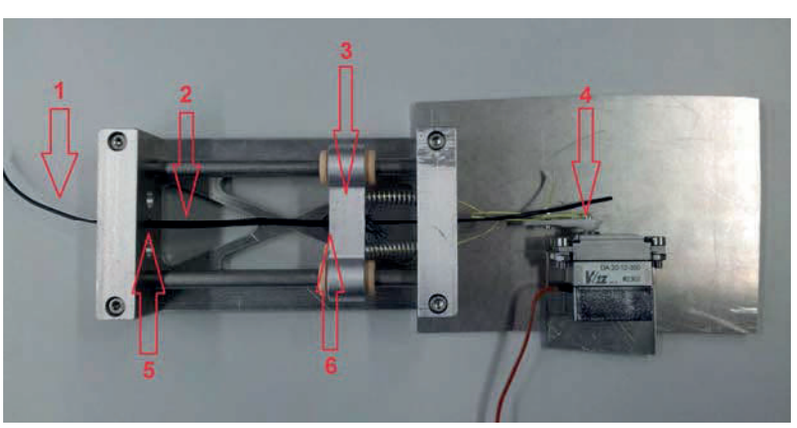
Universität Bremen 8

Different Gripping



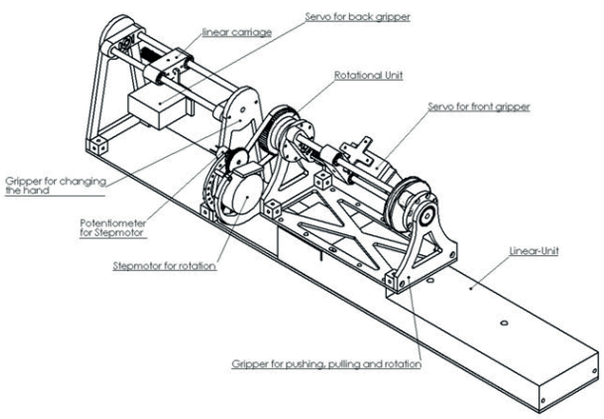
Universität Bremen 9

Different Gripping



Universität Bremen 10

Catheter Driver System

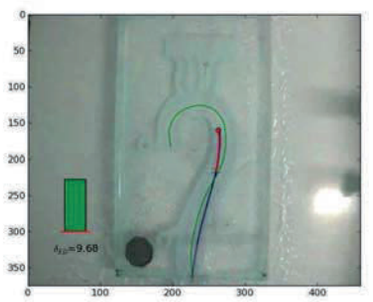


Universität Bremen 11

Catheter Driver System

Universität Bremen 12

Machine Learning with the Catheter Driver



Universität Bremen

13

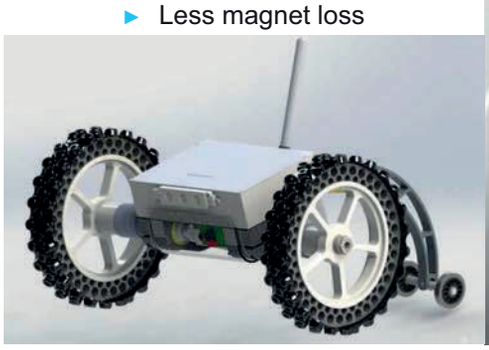
INCASS

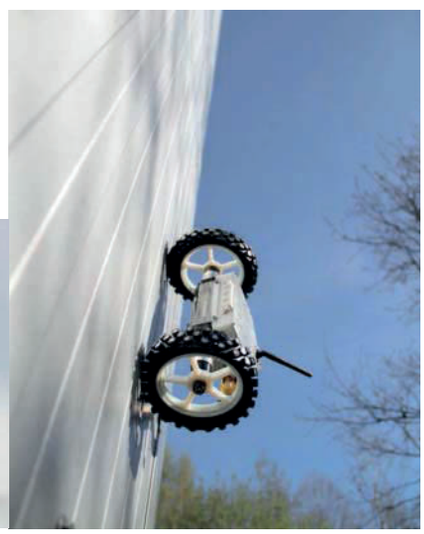
Universität Bremen

14

INCASS INSPECTION CAPABILITIES FOR ENHANCED SHIP SAFETY

- Magnet-Crawler II „MIRA“
  - Onto autonomy
    - Onboard ARM System
  - New wheel design
    - Less magnet loss





Universität Bremen

15

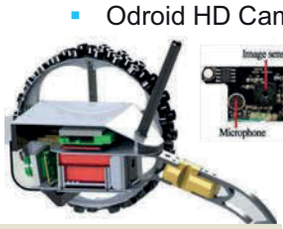
Core-Components of the Crawlers

Magnet Crawler I

- Control:
  - RC-Reciver
  - External Tracking
- Unmovable Camera:
  - 2.4 Ghz Camera module (res: 420 lines)

Magnet Crawler II

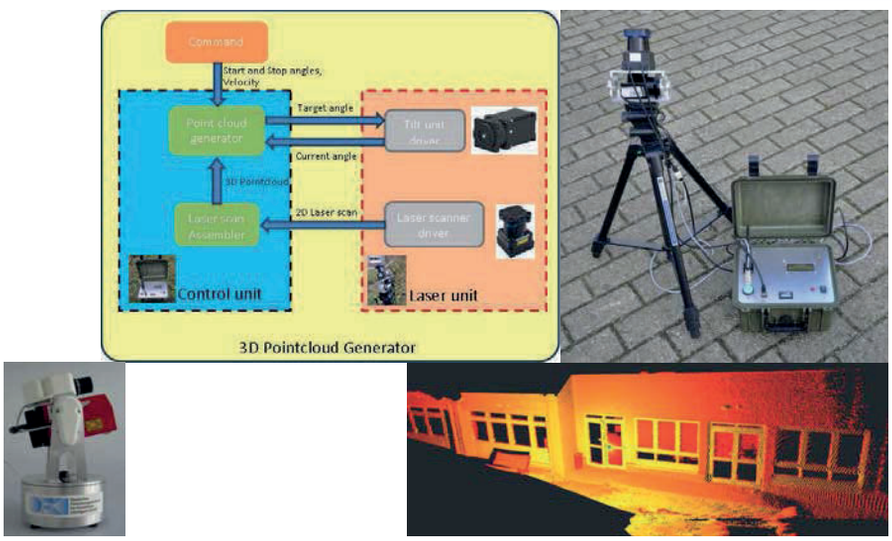
- Control:
  - Odroid U3 – ARM Quadcore
  - Motor-Encoders
  - External Tracking
- Tiltable Camera:
  - Odroid HD Cam 720p.



Universität Bremen

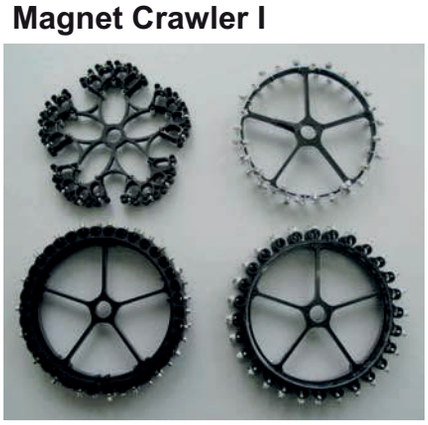
16

External Odroid Scanner



Magnet Wheels

Magnet Crawler I



Magnet Crawler II - A

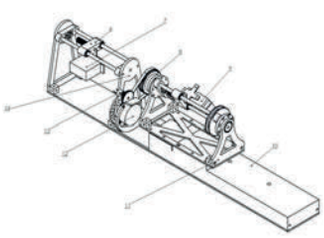
Magnet Wheels

Magnet Crawler II – A

Magnet Crawler II - B

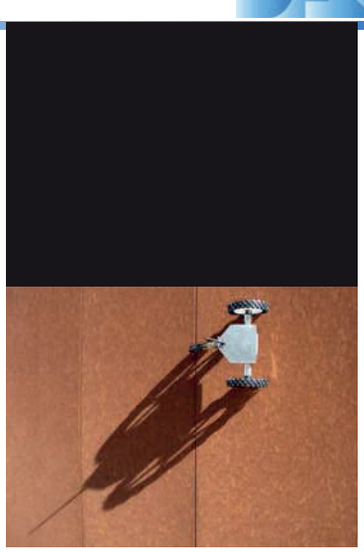
Universität Bremen

19



DFKI Bremen & Universität Bremen
Robotics Innovation Center
Director: Prof. Dr. Frank Kirchner
www.dfki.de/robotics
robotics@dfki.de

Thank you!



Universität Bremen

3.2 'Novel Serial Elastic Actuator Elastic module without friction hysteresis' (MD-T-02)

Martin Mallwitz[(1)]*,Christian Oekermann*[(1)]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: martin.mallwitz@dfki.de

Abstract

A serial elastic actuator (SEA) includes a physical compliance. Thereby the actuator provides the following characteristics: sensible for external mechanical impacts, decoupling loads from the motor side, low-pass filter behavior for loads and it stores energy. This characteristics privilege the actuator for applications with a direct human-robot contact like industrial, rehabilitation or teleoperation scenarios and for walking robots. The talk gives an overview of common applications and designs. It gives a more detailed view on SEA-development at the DFKI project 'Capio'. Generally the implementation of a compliance cause hysteresis problems by friction or material characteristics. Depending of the positioning of elasticity it complicates the design development and increases the actuator size. To solve these problems a customized torsional disc spring is implemented in a serial elastic actuator module. The iterative spring design workflow from the CAD model to the large displacement FEM simulation is described. The DFKI design of a serial elastic module for a 5 Nm actuator is shown beside other research designs.

Novel Serial Elastic Actuator

Elastic module without friction hysteresis

Martin Mallwitz, Christian Oekermann

25.06.2015

DFKI Bremen & Universität Bremen
Robotics Innovation Center
Director: Prof. Dr. Frank Kirchner
www.dfki.de/robotics
robotics@dfki.de

Universität Bremen

Outline

- SEA Motivation
- SEA Applications
- Design of SEAs
- Capio SEA, Design and Usage
- Novel SEA Module, Design and Dimensioning
- Next Steps and Future Work
- References

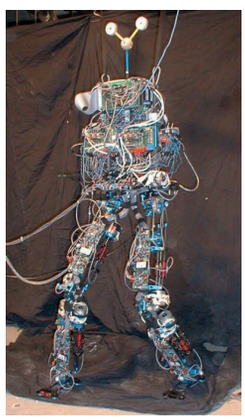
SEA Motivation

- Robot safety principles
  - Supervision, force control, lightweight
  - ISO 13482, ISO 10218
  - Inherent safe design → SEA
- Robot industrial application
  - Usually robot human interaction with safety distance
  - Dangerous situation by robot motion
  - New: robot as cooperative partner in direct contact
- Robot rehabilitation application
  - Direct human robots contact
  - Unhealthy operator/patient

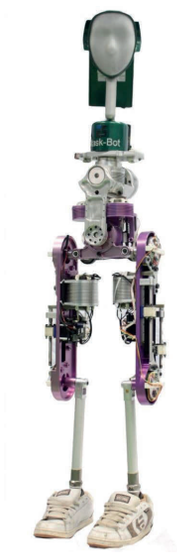
SEA Applications

- Mostly used for legged robots
- No commercial actuators available
- Few industrial application

Baxter

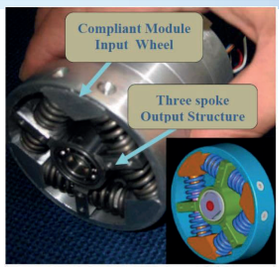


M2

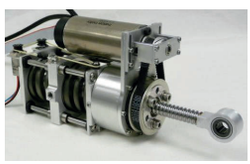


Herbert

Designs of SEAs

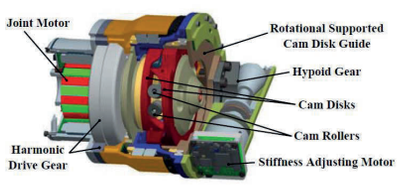


Tsagarakis 2009

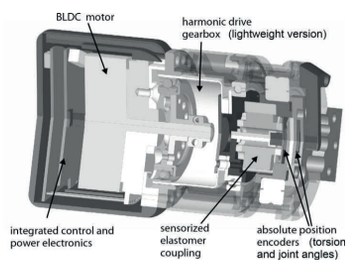


Paine 2012

Pratt 1995



Wolf 2011



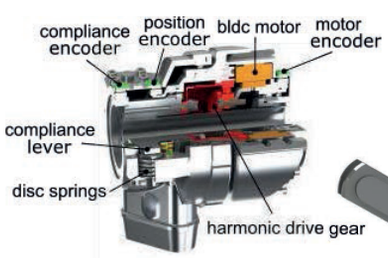
Paskarbeit 2013

Universität Bremen

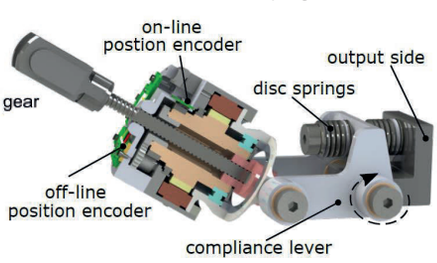
5

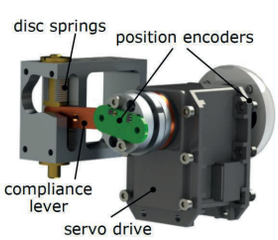
Capio SEAs at the RIC

- Nominal torque 60 Nm
- Overall weight 1kg
- Robodrive ILM 70 x 10, 370 W, 3500 rpm, 1.25 Nm
- Harmonic Drive CPL 20, ratio 1:80
- Variable set of disc springs



- Maximum force 790 N
- 166 mm/s max. travel velocity
- Overall weight 320 g
- Robodrive ILM 50x10, 140 W, 5000 rpm, 0.28 Nm
- Driven nut, Spindle 2 mm lead
- Offline tooth wheel nonius with iC-Haus MH 12 bit sensors
- Online iC-Haus MU 19 bit sensor for BLDC commutation and travel
- Compliance via structure
- Variable set of disc springs





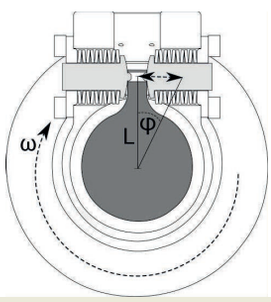
- Dynamixel 24F, 2.6 Nm toque at 12 V, 126 rpm
- Compliance via structure
- Variable set of disc springs
- Compliance measured with two iC-Haus MH sensors

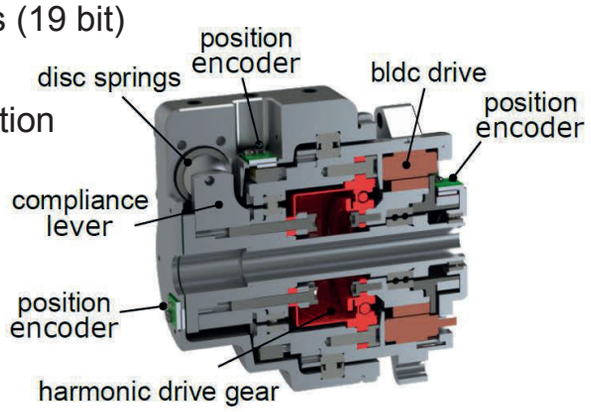
Universität Bremen

6

Capio Design 14 / 28 Nm

- ILM 50x8 robodrive and CPL 14 - 50/100 harmonic drive
- 28 Nm, 140 W, 600g
- 3 x iC-Haus MU sensors (19 bit)
- Variable disc springs
- Hysteresis by spring friction

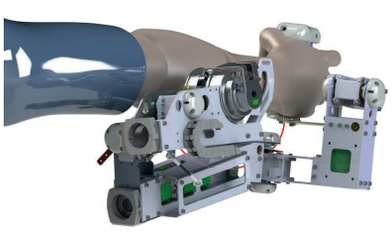




Universität Bremen

7

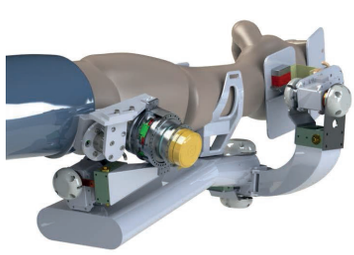
Usage at the Capio Exoskeleton



- Current setting for internal/external rotation of upper arm and ellbow extension/flexion: Capio SEA 14 Nm
- Weight 4 x 600 g =2400 g
- Experimental evaluation showed a max required torque of 4 Nm

Current forearm setup

- Development of adapted torque SEA for weight reduction and safety aspects
- Expected weight reduction about 1000 g



Sketch of new forearm setup

Universität Bremen

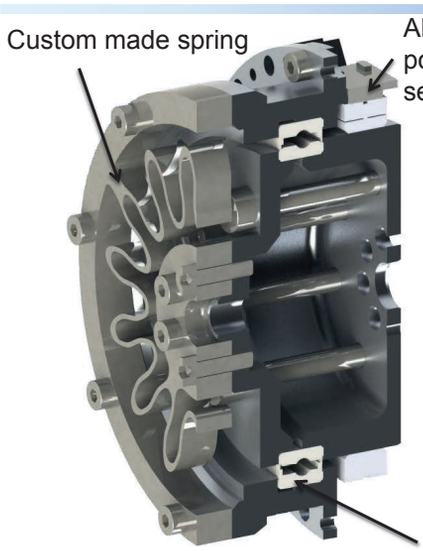
8

Known SEA Problems

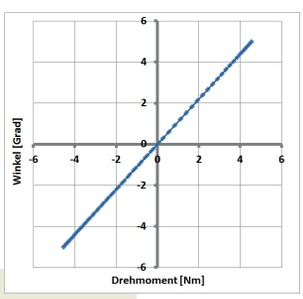
- Hysteresis by spring contact friction
- Hysteresis by elastomer compliant element (nonlinear material characteristics)
- Hysteresis by friction in a sealed bearing
- Short lever arm design if inside housing
- Expensive milled parts necessary
- Clunking and moving of unloaded springs
- Not all torques possible with commercial springs

Universität Bremen 9

Serial Elastic Modul



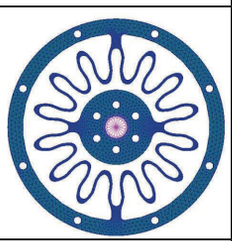
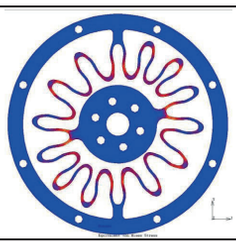
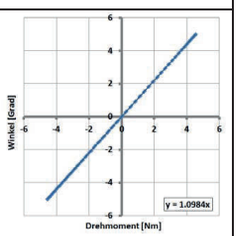
- Goals: 5 Nm, +-5 ° deflection
- Given Diameter by bearing
- Friction free spring application
- Spring package possible
- 300 ° limited working range



Universität Bremen 10

Spring Dimensioning Workflow

- Goals: 5 Nm and +-5 ° deflection
- Iterative process, morphological adaption for stress reduction

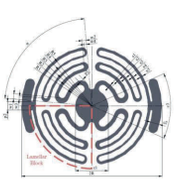
CAD-Design Solidworks	Mesh-Design Patran/Nastran	FEM-Simulation Marc/Mentat	Evaluation
			
Parametric Design	2D- simplification TRIA-Elements	Large-Displacement Simulation	Spring-Stiffness Calculation

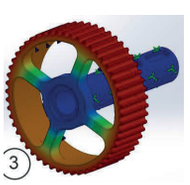
Custom Rotative Spring Designs


Stienen 2008


Lagoda 2010


Carpino 2012

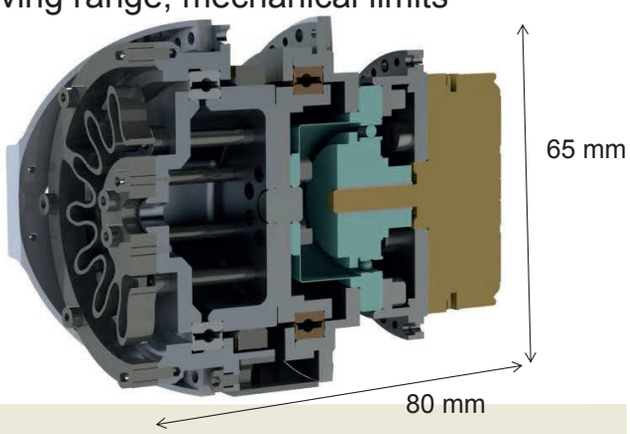

Accoto 2013


Pierce 2014

Year	Name	Stiffness [Nm/rad]	Max Torque [Nm]	Max Deflection [rad]	Outer Diameter [mm]	Thickness [mm]	Weight [g]
2008	Stienen	88,00	22,00	0,25	50,00	10,00	-
2010	Lagoda	219,00	90,00	0,50	~90	-	-
2012	Carpino	98,00	7,68	0,08	85,00	3,00	61,50
2013	Accoto	272,25	30,00	0,11	90,00	23,50	370,00

Serial elastic actuator

- Base: Limes Actuator 5 Nm, Maxon EC flat 45, HFUC11–50
- Module weight 160 g, Overall weight 380 g
- One iC-Haus MU target (19 bit) for deflection and position
- 300 ° moving range, mechanical limits



Universität Bremen

13

Next Steps and Future Work

- Assemble joint, integrate in testbed and Capio forearm
- Spring design
  - Evaluation spring characteristics
  - Optimize design concerning stress, size, weight of spring
  - Safety limits for deflection angle
- Module design
  - Implement strain gauge at spring for redundant torque signal or removing iC-Haus MU sensor
  - Compact design and weight reduction
  - Implement brakes on motor side

Universität Bremen

14

References

- Tsagarakis, N. G., Laffranchi, M., Vanderborght, B., & Caldwell, D. G. (2009, May). A compact soft actuator unit for small scale human friendly robots. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on* (pp. 4356-4362). IEEE.
- Pratt, G., & Williamson, M. M. (1995, August). Series elastic actuators. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on* (Vol. 1, pp. 399-406). IEEE.
- Pierce, B., & Cheng, G. (2014, November). Realising Herbert: An affordable design approach of an anthropometrically correct compliant humanoid robot. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on* (pp. 7-12). IEEE.
- Wolf, S., Eiberger, O., & Hirzinger, G. (2011, May). The DLR FSJ: Energy based design of a variable stiffness joint. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (pp. 5082-5089). IEEE.
- Paskarbeit, J., Annunziata, S., Basa, D., & Schneider, A. (2013). A self-contained, elastic joint drive for robotics applications based on a sensorized elastomer coupling—Design and identification. *Sensors and Actuators A: Physical*, *199*, 56-66.
- Paine, N., & Sentis, L. (2012, December). A new prismatic series elastic actuator with compact size and high performance. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on* (pp. 1759-1766). IEEE.
- Lagoda, C., Schou, A. C., Stienen, A. H., Hekman, E. E., & van der Kooij, H. (2010, September). Design of an electric series elastic actuated joint for robotic gait rehabilitation training. In *Biomedical Robotics and Biomechatronics (BioRob), 2010 3rd IEEE RAS and EMBS International Conference on* (pp. 21-26). IEEE.
- Accoto, D., Carpino, G., Sergi, F., Tagliamonte, N. L., Zollo, L., & Guglielmelli, E. (2013). Design and characterization of a novel high-power series elastic actuator for a lower limb robotic orthosis. *Int J Adv Robot Syst*, *10*, 359.
- Stienen, A. H., Hekman, E. E., Braak, H. T., Aalsma, A. M., van der Helm, F. C., & van der Kooij, H. (2008, October). Design of a rotational hydro-elastic actuator for an active upper-extremity rehabilitation exoskeleton. In *Biomedical Robotics and Biomechatronics, 2008. BioRob 2008. 2nd IEEE RAS & EMBS International Conference on* (pp. 881-888). IEEE.

Universität Bremen

Thank you!

Question?

DFKI Bremen & Universität Bremen
Robotics Innovation Center
Director: Prof. Dr. Frank Kirchner
www.dfki.de/robotics
robotics@dfki.de

Universität Bremen

3.3 'The way to simplicity: Self adaptive mechanisms for robots' (MD-T-03)

Marc Manz[(1)]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: marc.manz@dfki.de

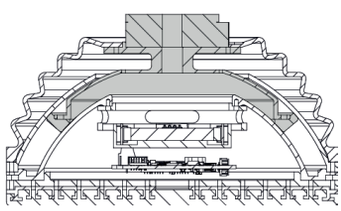
Abstract

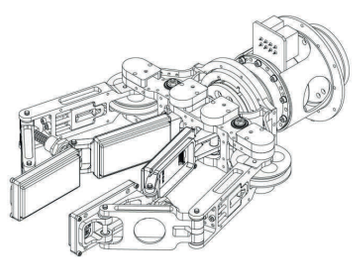
Mobile robots have to deal with complex tasks in real world environments. A lot of information about this environment is required to handle this complexity and to act or react adequate. This information has to be processed to make decisions and therefore a high computing capacity is required. Self adaptive mechanical mechanisms have the potential to reduce the computing complexity by adapting inherently to a given task. This adaptation could be reached with different mechatronic or mechanical approaches. Two examples of such self adaptive mechanical mechanisms are presented in this talk. A self adaptive foot designed for the multi legged robot MANTIS and a self adaptive gripper designed for the COMPI manipulator. The self adaptive foot is basically a walking surface with a ball joint. These ball joint rotates around a virtual centre of rotation which is located below the surface on which the robot is walking. This self adaptive foot thereby always orientates itself with the walking surface towards the ground. The self adaptive Gripper is designed to adapt for two different base types of objects, cylindrical objects and prismatic objects with parallel surfaces.

German Research Center for Artificial Intelligence

The way to simplicity: Self adaptive mechanisms for robots

- Marc Manz -





University of Bremen

Agenda

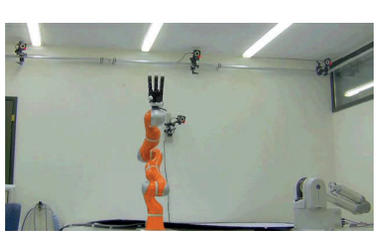
1. Introduction
2. Self adaptive foot (Project LIMES)
   - Motivation and Requirements
   - Concepts
   - Design
3. Self adaptive gripper (Project SpaceBot)
   - Motivation and Requirements
   - Concepts
   - Realization
   - Simulation
4. Conclusion

-Introduction-

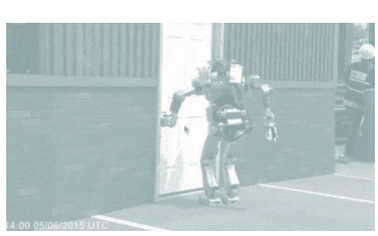
Motivation

- Robots should react fast
  - To fulfill the task
  - To adapt to the environment

- Robots should run without malfunctions
  - For long term autonomy
  - For safety reasons if humans are in the loop



Fast robotic arm catches objekts on the fly, Source: EPFL

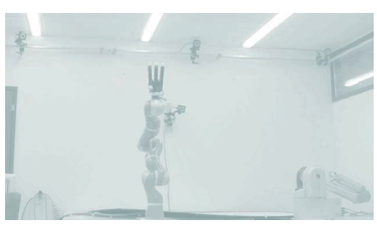


A Compilation of Robots Falling Down at the DARPA Robotics Challange, Source: IEEE Spectrum

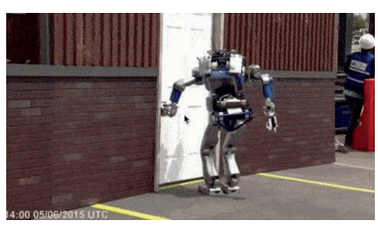
Motivation

- Robots should react fast
  - To fulfill the task
  - To adapt to the environment

- Robots should run without malfunctions
  - For long term autonomy
  - For safety reasons if humans are in the loop



Fast robotic arm catches objekts on the fly, Source: EPFL



A Compilation of Robots Falling Down at the DARPA Robotics Challange, Source: IEEE Spectrum

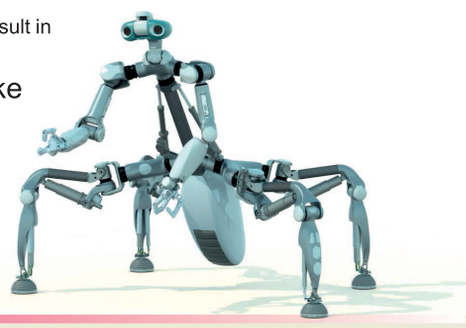
Universität Bremen 5

-Self adaptive foot-

Motivation and Requirements

Locomotion of multi legged robots in uneven terrain

- Insufficient knowledge about the surface
  - Long recording time for detailed point clouds
  - A high computing capacity is required to interpret them
- Fault tolerance
  - Measurements in general are faulty
  - The level of details of each model is limited
  - Higher number of required sensors and motors result in a high probability of failure
- Fast reaction time to adapt in a reflex like manner

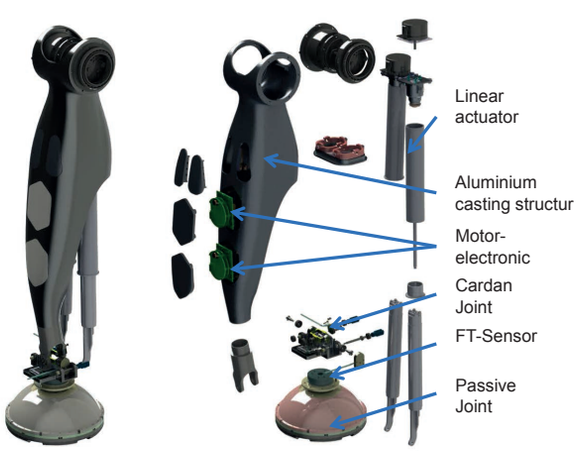


Universität Bremen 7

Concept – Lower leg

Active and passive adjustment

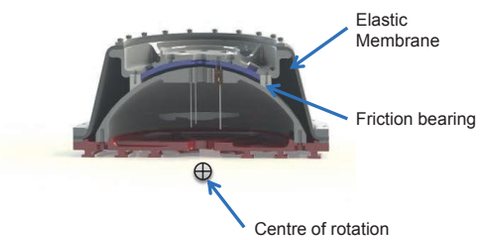
- Active part
  - Two linear actuators to roughly control orientation of the foot relative to the ground
  - Cardan joint with absolute encoder for each axis
  - The linear actuator does not have to provide high speed movements. This results in low power consumption and lower mass of the actuator
- Passive Part
  - Inherent stable passive joint to adapt exactly to the surface
  - Fast response to uneven and unexpected surface variations are covered by the passive joint

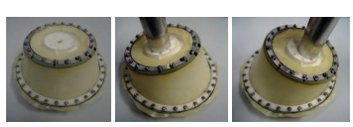


Universität Bremen 8

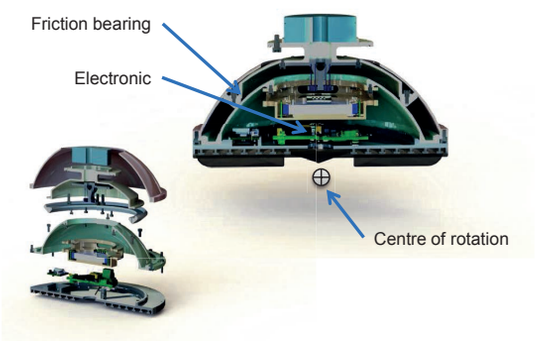
Passive self adaptive joint

Prototyp 1





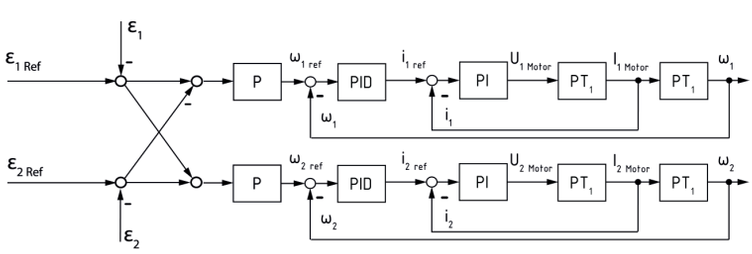
Prototyp 2



Control Scheme

How the lower leg is controlled

- The orientation of the foot is positioned to always be parallel to the ground
- The reference angles are taken from the kinematic module
- No reactive control is necessary due to the adaptive characteristic of the passive adaptive joint



Advantages

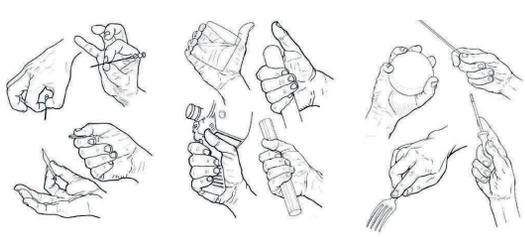
- Sensor nonlinearities (IC Haus MH Magnet) can be neglected
- Fast response of the passive joint
- Lower power consumption

- Self adaptive gripper -

Motivation and Requirements

Grasping of different shaped objects

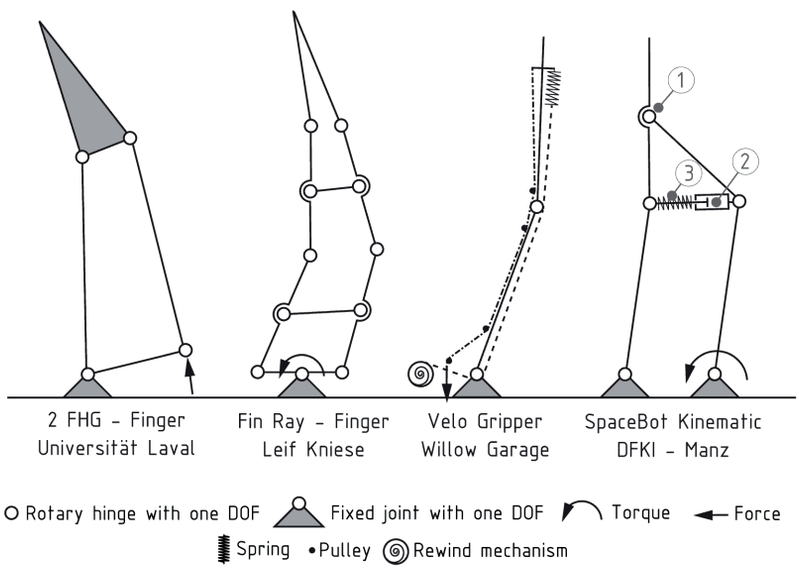
- Insufficient knowledge about the objects
  - Some objects have high similarity of their shape (ovoid and sphere) and therefore they are difficult to differentiate
  - A high computing capacity and good sensor are required to identify the objects
- Low weight is required for mobile robots
- Simple gripper control



Human Hand, Source: I.A. Kapandji, Funktionelle Anatomie der Gelenke. Hippokrates Verlag, 1999

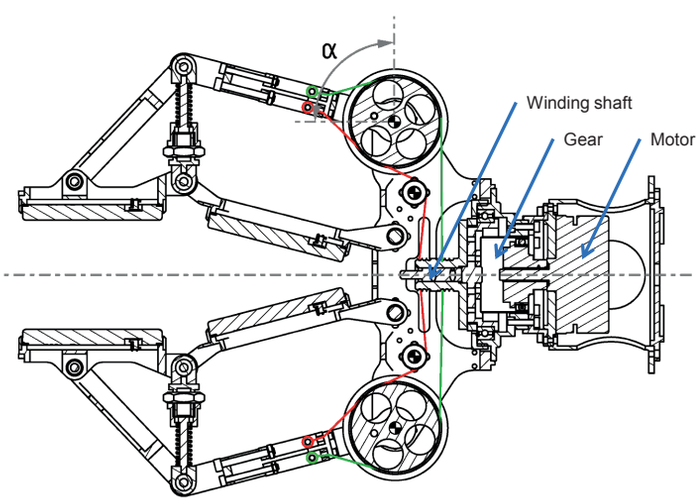
Concept

Self adaptive finger kinematics



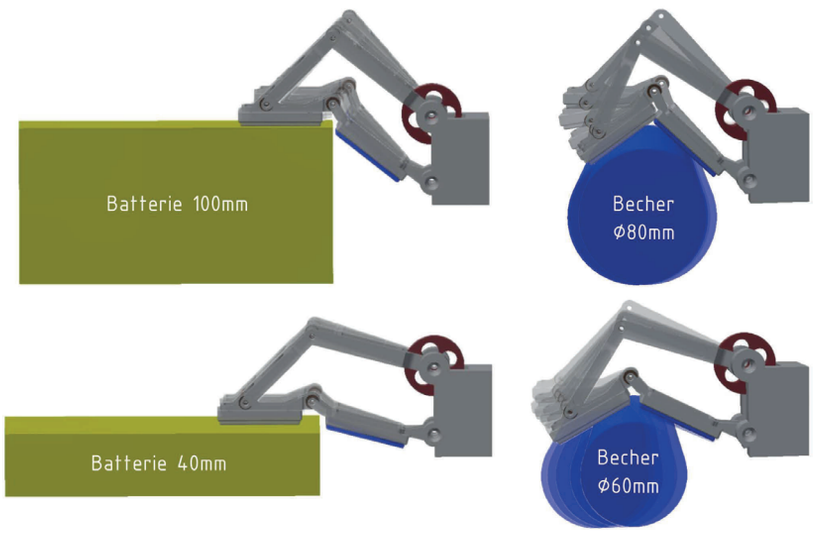
Realization

SpaceBot Gripper II



Adaptation for different shaped objects

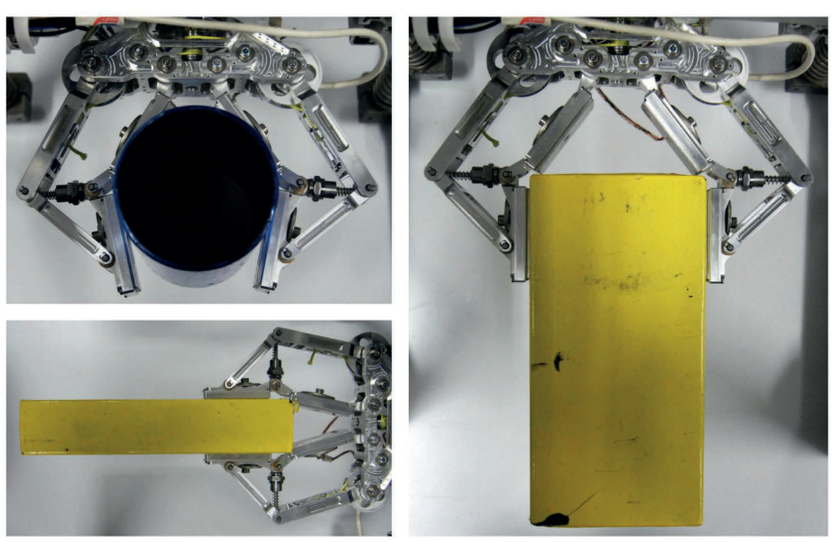
Multi body simulation – Grasping of different objects



Universität Bremen 15

Adaptation for different shaped objects

Grasping test – Grasping of different objects



Universität Bremen 16

-Conclusion-

Conclusion

Self adaptive mechanisms…

- reduce the required computing power
- are inherently fast as they do not react they adapt
- have the potential to save motor's and sensor's
- could reduce the weight of mobile robots
- lower the power consumption

Thank you for
your attention!

3.4 'Coyote III: Highly Mobile and Modular Micro Rover for Cooperative Tasks' (MD-P-01)

Roland Sonsalla [(1)]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: roland.sonsalla@dfki.de

Abstract

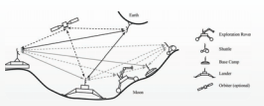
Robotic exploration missions are gaining in importance for the exploration of our solar system. A wide range of different scientific goals have been formulated for future exploration of Moon and Mars. In order to achieve these goals a need arises for robotic systems and mission set-ups with increasing complexity. Coyote III is developed within the scope of the project TransTerrA, which aims to implement a logistics chain to handle complex mission tasks. The poster presents the design considerations for Coyote III as well as the rover's development and over-all modularity concept, including a modular manipulation device. An introduction of the chosen lunar reference mission is given as well as an overview of Coyote III's key technical parameters. Coyote III is a highly mobile modular micro rover platform, able to act as a shuttle rover performing autonomous operations.

German Research Center for Artificial Intelligence

Coyote III: Highly Mobile and Modular Micro Rover for Cooperative Tasks

Roland U. Sonsalla and Joel Bessekon Akpo
DFKI Robotics Innovation Center, Robert-Hooke-Str. 1, Bremen, Germany
Email: Roland.Sonsalla@dfki.de, Joel.Bessekon_Akpo@dfki.de

+ Reference Mission Concept



Future exploration of the solar system is calling for robotic missions with increasing complexity. Coyote III is developed within the scope of the project TransTerrA, aiming to extend the exploration capabilities by implementing a logistics chain based on a (semi-) autonomous and heterogeneous team of cooperating robots. It serves as a shuttle system, cooperating with an exploration rover, stationary base camps and portable payload items (PLI). A reference mission set-up within Amundsen crater provides the baseline for terrestrial tests.

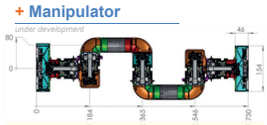
+ Coyote III an Introduction

Coyote III is an enhanced design of the Coyote II rover and is build as a terrestrial testbed. It's design is based on a modular design concept, allowing to implement additional payload and satisfying the following needs:

+ High mobility in unstructured terrain
+ Small and lightweight rover system
+ (Semi-) autonomous operation
+ Carrying and handling of PLIs

+ Manipulator

under development



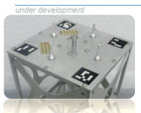
+ optional payload for PLI handling (up to 5 kg)
+ symmetrical design to allow bridging to other robots
+ 5 DoF with Robodrive ILM15x14 bldc-motor and Harmonic Drive gear (160:1) (80/ 224 Nm torque)
+ FPGA based motor diver
+ 2 active EMIs as endeffector with power and data management electronics
+ lightweight aluminum connectors and CFRP tubes

+ Rear Body

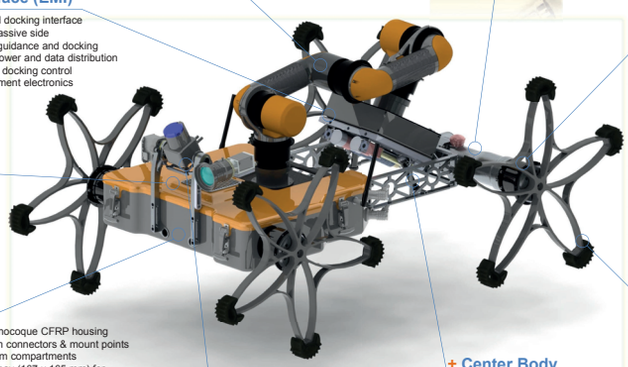
+ 3D-milled aluminum structure
+ central T-link
+ integrated passive roll joint +/-20° to keep 4-wheel ground contact
+ absolute encoder to track the pose of the rovers chassis
+ two tube-like extensions for driving-unit connection

+ Electro-Mechanical Interface (EMI)

under development



+ standardized docking interface
+ active and passive side
+ mechanical guidance and docking
+ 18 pins for power and data distribution
+ vision based docking control
+ PLI management electronics



+ Driving Unit

+ Robodrive ILM 15x08 bldc-motor
+ Harmonic Drive gear (80:1)
+ nominal torque: 22.4 Nm
+ absolute encoder at input and output shaft
+ FPGA based motor driver

+ Subsystem Stacks

+ OBC: IntelCore i7-3517UE, 1.7 GHz
+ IMU: Xsens MTi-300 AHRS
+ mobile access point: 2.4 GHz, 802.11n
+ remote motor stop: 868 MHz Xbee-Pro
+ remote driving control via Bluetooth
+ active thermal control and health monitoring
+ LiPo primary battery (44.4 Vdc, 4.5 Ah)

+ Hybrid Legged-Wheel

+ 5 legs per wheel
+ leg length: ~185 mm

+ Main Housing

+ semi-monocoque CFRP housing
+ aluminum connectors & mount points
+ subsystem compartments
+ payload bay (167 x 165 mm) for passive EMI
+ stainless steel underbody-protection
+ two front bottom cut-outs (35 x 30 mm)
+ mount points and cable bushings for sensor bench connection

+ Sensor Bench

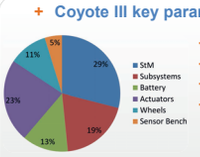
+ static camera: Basler Ace 2048x2048 px, 25 fps, 79.7° FoV
+ tiltable laser range finder (+/- 180°) Hokuyo UST-20LX (20 m x 270° FoV)

+ Center Body

under development

+ lightweight aluminum frame design
+ easy adaptable for payload and/or rover needs
+ independent from front and rear body
+ integrated 2 DoF roll-pitch platform for passive EMI
+ communication and power connection to front body

+ Coyote III key parameters



Coyote III mass distribution

+ Size (l x w x h): 994 x 584 x 380 mm
+ Speed: ~ 1.3 m/s
+ Power consumption: ~ 75 W (driving, average)
+ Mass: 12.5 kg (w/o payload subsystems)
  - EMIs and PMS: ~ 2 kg
  - Manipulator: ~ 6.5 kg
+ Estimated total mass: 21 kg (+ 5 kg PLI)

Supported by:

Federal Ministry for Economic Affairs and Energy

DLR

on the basis of a decision by the German Bundestag.

The work presented is part of the project TransTerrA which is funded by the German Space Agency, Grant number 50 RA 1301.

Universität Bremen

Contact:
DFKI Bremen & University of Bremen
Robotics Innovation Center

Director: Prof. Dr. Frank Kirchner
E-mail: robotics@dfki.de
Website: www.dfki.de/robotics

3.5 'Diving Cell Performance on LENG - Design, Calculation and Performance' (MD-P-02)

Jens Hilljegerdes[(1)]

(1) Arbeitsgruppe Robotik, Universität Bremen, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: jens.hilljegerdes@dfki.de

Abstract

Diving Cell Performance on LENG

To enable a horizontal position control of the AUV system LENG it is equipped with two diving cells. The mechanical concept of this cell is based on a mechanical driven piston. The design in size and shape of this piston is crucial on all subjects in the design process.

The poster describes the mechanical design of the cell, the edge conditions in an underwater environment and the qualitative diving performance based on the diving cells

German Research Center for Artificial Intelligence

Diving Cell Performance on LENG

Design, Calculation and Performance

Design constrains

For the design of a Diving Cell the size and shape is crucial on all subjects in the design process.
In this case the maximum volume difference was the goal on the design. The constrained conditions related to the system are:

- Outside diameter of the cell 180mm
- Minimize in length
- Optional stand alone underwater housing
- Volume difference of 0.8 dm^3 (±0,4dm^3 !)
- Usability for front and rear compartment

Design options

For the mechanical design of the cell the kind of implemented seal is of key importance. The sealing/ bearing in this case are two slide bearings and a Turcon Glyde-Ring as sealing origin from Trelleborg. This seal is rated for 60MPa (600bar) in a PTFE version for water hydraulics.

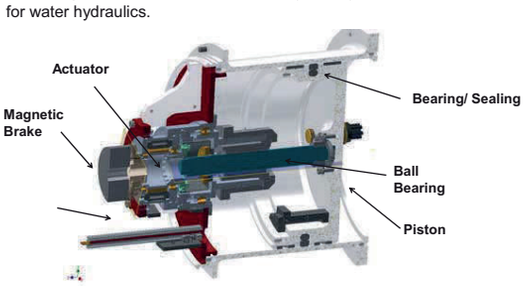


Fig. 1: Diving Cell in 3D - CAD

The cylinder piston is driven by a ball screw spindle, making a brake on the actuator side unavoidable. The spindle is driven by a DFKI actuator with 54Nm / 0,3Hz. With 10mm distance/ turn the speed is 3mm/ second. Due to the limited diameter the plug and cabling is guided trough the piston.

Calculation

The diameter of the piston and the operating diving depth result in the force to be active driven by the linear actuator. In this system the piston is driven by a ball screw spindle that is capable to carry and actuate the load with an efficiency of 90%.

Dimensions and Specifications:
- Piston diameter 150mm
- Mechanical max. actuation 45mm
- Load on the piston 26KN (15bar = 150m diving depth)
- Rated load ball screw: 17kN / need of pressurized compartment
- Volume difference in use of 0.8 dm^3

Diving Cell Diameter

The load on the piston is dependent on its diameter. For an example depth of 150m the relation is shown in figure 2. The load on the piston effects all parts of the design including the actuator, the spindle and the bearing. To archive a greater spindle load capacity a higher thread pitch is common. This effect results in a higher torque of the actuator. Due to this effect a longer but thinner diving cell is preferable.

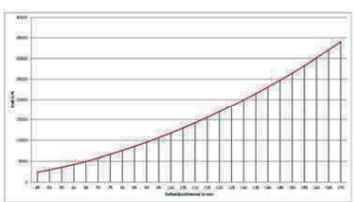


Fig. 2 : Graph on force to diameter of the piston for 15bar (150m)

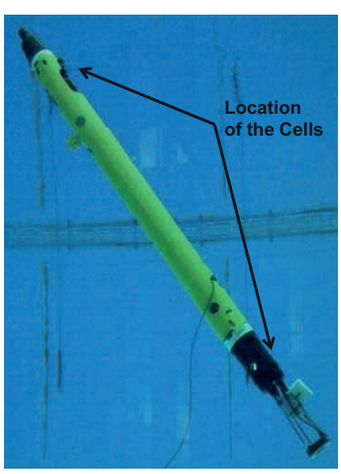


Fig. 3 : LENG, Diving Performance based on Diving Cells

Supported by:

DLR

(grant No. 50NA1217)

Universität Bremen

Contact:
DFKI Bremen & University of Bremen
Robotics Innovation Center

Director: Prof. Dr. Frank Kirchner
E-mail: robotics@dfki.de
Website: www.dfki.de/robotics

3.6 'Teredo IceShuttle - Bringing together under-ice deployment, sensor platform, docking, launch and recovery in a small diameter vehicle design' (MD-P-03)

Marius Wirtz[(1)]

(1) Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Straße 1, 28359 Bremen, Germany

Contact: marius.wirtz@dfki.de

Abstract

The search for life elsewhere in the universe is one of the driving forces for space exploration. Concerning this, during the last decades Jupiter's icy moon Europa gained increasing interest within the scientific community. Sealed below a thick crust of ice the moon harbors a global ocean that carries twice as much water compared to all liquid water on earth. While the exploration of this distant world is of great interest for exobiologists and planetary scientists, the requirements for such a mission are high. Within the Europa Explorer project a first mission concept was created, including the development of an autonomous underwater vehicle (AUV) and an ice drill that is designed to transport the AUV through the ice into the unknown ocean. The ice drill is therefore described as IceShuttle. Since the IceShuttle is also supposed to carry additional equipment, function as a docking station, as well as a sensor carrier for navigation, while having a strong limitation for its cross section, all components of the system have to be lined up in series. This required an unfolding and deployment strategy to be developed, which is a key element of the IceShuttle and is described within the poster.

German Research Center for Artificial Intelligence

Teredo IceShuttle

Bringing together under-ice deployment, sensor platform, docking, launch and recovery in a small diameter vehicle design

System Objective:

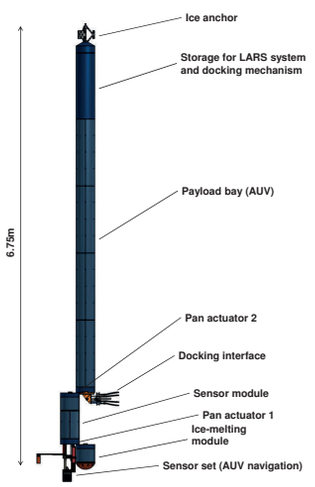
The IceShuttle Teredo is a robotic probe which is capable of transporting a payload through an ice-shield. The mission scenario is Jupiter's icy moon Europa which is covered by an ice crust estimated to be 3 to 30 km thick and suspected to have a 100 km deep ocean below. The system will function as a basecamp and deliver a robot (AUV) which will explore the ocean.

Requirements of Mechatronical Design:

- Drilling through an ice crust
- Transport of an AUV through the ice
- Small diameter (reducing the energy needed to drill through the ice)
- Launch and Recovery (LARS) of the AUV
- Docking interface (energy & data)
- Providing an additional, stationary sensor set (AUV navigation)
- Sustainability (water & ice environment)

System Design:

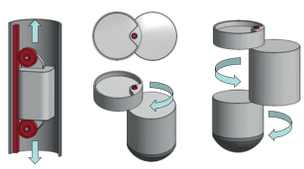
The strong restriction concerning the diameter leads to a serial arrangement of the system components:



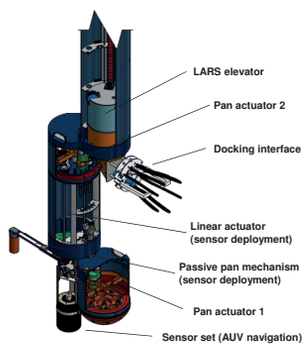
System overview Teredo IceShuttle

Mechanical Principals Chosen for Unfolding and Deployment:

The serial arrangement necessitates mechanisms to unfold the system.



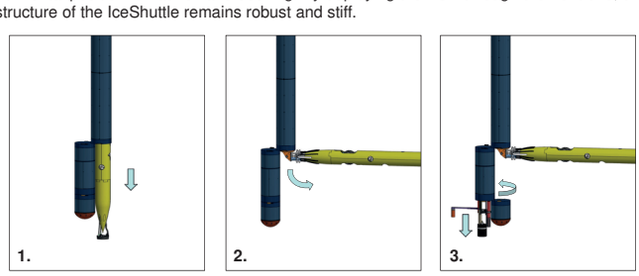
Unfolding & deployment mechanisms



Functional parts (unfolding & deployment)

Payload Deployment:

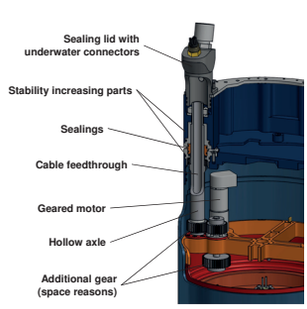
The AUV is deployed through the front end of the payload module and transferred into a horizontal position for launch and docking. By deploying the AUV through the front end, the structure of the IceShuttle remains robust and stiff.



Linear AUV deployment | Transfer of AUV into launch position | Sensor deployment (AUV navigation)

Pan Actuator:

The assembly of the pan actuator marks the most relevant component to unfold the IceShuttle. Its main purpose is to pan the lower modules to clear the openings of the IceShuttle and thus deploy the AUV and the navigation sensors.
The assembly needs to support a load of two eccentrically stored modules, mechanically and electrically connect the modules, as well as seal the pressure hulls. To keep the diameter of the IceShuttle small, the assembly needs to guarantee an extreme eccentricity.



Pan actuator

Universität Bremen

DLR

Grant No: 50 NA 1217

Supported by: Federal Ministry of Economics and Technology on the basis of a decision by the German Bundestag

Contact:
DFKI Bremen & University of Bremen
Robotics Innovation Center

Director: Prof. Dr. Frank Kirchner
E-mail: robotics@dfki.de
Website: www.dfki.de/robotics

German Research Center for Artificial Intelligence (DFKI) GmbH

DFKI Bremen
Robert-Hooke-Straße 1
28359 Bremen
Germany
Phone: +49 421 178 45 0
Fax: +49 421 178 45 4150

DFKI Saarbrücken
Stuhlsatzenhausweg 3
Campus D3 2
66123 Saarbrücken
Germany
Phone: +49 681 875 75 0
Fax: +49 681 857 75 5341

DFKI Kaiserslautern
Trippstadter Straße 122
67608 Kaiserslautern
Germany
Phone: +49 631 205 75 0
Fax: +49 631 205 75 5030

DFKI Projektbüro Berlin
Alt-Moabit 91c
10559 Berlin
Germany
Phone: +49 30 238 95 0

E-mail:
reports@dfki.de

Further information:
http://www.dfki.de