Roboy Vision Documentation

Release 0.1.

Jun 29, 2017

Usage and Installation

1 Relevant Background Information and Pre-Requisits 3

2 Contents 5

This project's goal is to provide Roboy with extensive vision capabilities. This means to recognize, localize and classify objects in its environment as well as to provide data for localization to be processed by other modules. The input will be a realsense camera device, the output should be high-level data about Roboy's environment provided using ROS messages and services.

The most import task in Vision for human interaction is to detect and recognize faces, which is why this was considered the highest priority of this project. The current main tasks of this project are:

- Identification of Roboy Team Members
- Pose estimation of a detected face and Roboy Motor Control
- Tracking of detected objects
- Person Talking detection
- Mood Recognition
- Gender Recognition
- Remebering faces online
- Age classification
- Scene and object classification

CHAPTER 1

Relevant Background Information and Pre-Requisits

Our approach to tackle the given tasks in Vision is to use machine learning methods. Therefore a basic understanding of machine learning, specifically also deep Neural Networks and Convolutional Neural Networks will be necessary.

The following links are to be seen as suggestions for getting started on machine learning:

- Crash Course on Deep Learning in the form of Youtube tutorials: DeepLearning.tv
- Closer Look at the implementation of Neural Networks: The Foundations of deep learning
- An introduction to Convolutional Neural Networks (CNNs): Deep learning in Computer vision
- The machine learning framework used for implementation: Tensorflow
- Furthermore a basic understanding of simple machine learning approaches like Regression, Tree Learning, K-Nearest-Neighbours (KNN), Support Vector Machines (SVMs), Gaussian Models, Eigenfaces, etc. will be helpful.

The papers currently used for implementation should be understood:

- Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks
- FaceNet: A Unified Embedding for Face Recognition and Clustering
- DLIB: Facial landmarks and face recognition
- ‘You Only Look Once: Unified, Real-Time Object Detection <https://pjreddie.com/media/files/papers/yolo.pdf>‘_

Furthermore there are plans to extend the implementation using this paper:

- An All-In-One Convolutional Neural Network for Face Analysis

CHAPTER 2

Contents

Installation

Running all sub modules in realtime requires Ubuntu 16.04 with Kernel version 4.4.x. For getting started a jupyter notebook installation using anaconda will be sufficient. Tutorials in form of jupyter notebooks are provided.

Anaconda

We recommend the use of Anaconda. This allows all python libraries to only be installed in a virtual environment which then won't have any influence on other programs you are running. We will create a virtual environment using python 3.

- Download Anaconda from https://www.continuum.io/downloads#linux:

```
bash ~/Downloads/Anaconda3-4.3.0-Linux-x86_64.sh
```

- Install Anaconda:

```
bash ~/Downloads/Anaconda3-4.3.0-Linux-x86_64.sh
```

- Create a Conda Environment with the name "roboy" and python 3:

```
conda create --name roboy python=3
```

- To work on the created environment it has to be activated:

```
source activate roboy
```

- When you want to leave the environmant you have to use:

```
source deactivate
```

Dependencies

Now you should be working in your virtual environment. We then will install all requirements. We are working with python 3, because of tensorflow requirements.

- First clone the Vision repository and install the necessary python dependencies:

  ```
  cd ~/
  git clone https://github.com/Roboy/Vision
  pip install -r Vision/requirements.txt
  ```

- Install OpenCV:

  ```
  conda install -c menpo opencv3=3.2.0
  ```

- Install Tensorflow:

  ```
  conda install -c conda-forge tensorflow=1.0.0
  ```

- For running the tutorials DLib and jupyter notebook will also be required:

  ```
  conda install -c menpo dlib
  pip install jupyter
  ```

- The Last step is to install ROS Kinetic. Since ROS currently is not running using Python3 we install outside the virtual environment and Python 2. The ROS installation tutorial can be found on: http://wiki.ros.org/kinetic/Installation/Ubuntu.

Todo

Compile ROS with Python 3 to be able to use together with Tensorflow v1.x

Build

To build all ROS message and service files you can use catkin:

```
cd ~/Vision
catkin_make
```

To build doxygen documentation offline for viewing you can run:

```
cd ~/Vision
sphinx-build -b html ./docs ./build/docs
```

Compiling opencv from source

Note: This is required only if you want to work with multiple object/face tracking. This step is needed as this Multiple tracking is part of opencv_contrib module which needs to be compiled along with opencv, as it doesnt gets shipped with openv. The following instructions are for Mac.

First you will need:

1. Mac OSX 10.12
2. XCode

3. Command Line Tools (This is done from inside XCode)
4. CMake(http://www.cmake.org/download/)

Step 1: Download openCV and unzip it somewhere on your computer. Create two new folders inside of the openCV directory, one called StaticLibs and the other SharedLibs.

Step 2a: Build the Static Libraries with Terminal. To build the libraries in Terminal.

- Open CMake.
- Click Browse Source and navigate to your openCV folder.
- Click Browse Build and navigate to your StaticLib Folder.
- Click the configure button. You will be asked how you would like to generate the files. Choose Unix-Makefile from the Drop Down menu and Click OK. CMake will perform some tests and return a set of red boxes appear in the CMake Window.

You will need to uncheck and add to the following options.

- Uncheck BUILD_SHARED_LIBS
- Uncheck BUILD_TESTS
- Add an SDK path to CMAKE_OSX_SYSROOT, it will look something like this "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.9.sdk". (NOTE: make sure your version of SDK is used here)
- Add x86_64 to CMAKE_OSX_ARCHITECTURES, this tells it to compile against the current system
- Uncheck WITH_1394
- Uncheck WITH_FFMPEG

Click Configure again, then Click Generate.

When the application has finished generating, Open Terminal and type the following commands.

- cd <path/to/your/opencv/staticlibs/folder/>
- make (This will take awhile)
- sudo make install

Enter your password. This will install the static libraries on your computer.

Step 2c: Build the Shared Libraries with Terminal.

- Open CMake.
- Click Browse Source and navigate to your openCV folder.
- Click Browse Build and navigate to your SharedLib Folder.
- Click the configure button. You will be asked how you would like to generate the files. Choose Unix-Makefile from the Drop Down menu and Click OK. CMake will perform some tests and return a set of red boxes appear in the CMake Window.

You will need to uncheck and add to the following options.

- Check BUILD_SHARED_LIBS
- Uncheck BUILD_TESTS
- Add an SDK path to CMAKE_OSX_SYSROOT, it will look something like this "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.9.sdk".
- Add x86_64 to CMAKE_OSX_ARCHITECTURES, this tells it to compile against the current system

- Uncheck WITH_1394
- Uncheck WITH_FFMPEG
- Click Configure again, then Click Generate.

When the application has finished generating, Open Terminal.

- cd <path/to/your/opencv/SharedLibs/folder/>
- make (This will take awhile)
- sudo make install

You should see the libraries build in the shared and static libraries folders.

- cd /Users/<Username>/<path-to-installation>/StaticLibs/lib/python3
- ls -s cv2.cpython-36m-darwin.so cv2.so

The above step would help in creating a symbolic link so you can use it with python.

Getting started

Tutorials

As a start run the jupyter notebooks in tutorials section:

```
source activate roboy
cd ~/Vision/tutorials
jupyter notebook
```

There is four different tutorials:

- **Face_Detection** tutorial will show you how to run a face detection an an image or webcam input using the MTCNN neural network. Additionally also DLib face detector is used.
- **Facenet_Embeddings** tutorial shows how to calculate the 128D embeddings given a face using facenet. There exist 2 versions of this tutorial. One is using MTCNN for face detection, the other one using DLib. This tutoial provides the functionality to caluclate and save embeddings on a database of pictures, where all pictures are stored in a folder structure, with the folder name being the person in the picture.
- **Classifier_Training** uses embeddings calculated in the previous tutorial to train a classifier to distinguish between the classes in these embeddings. Currently only SVM and a binary Tree have been implemented.
- **Face_Recognition** tutorial shows how to run the classification on an image or webcam input. It demonstrates this using KNN and the classifiers trained in the previous tutorial.

Real Time

For running face detection in real time you can run the script:

```
source activate roboy
cd ~/Vision
python src/vision_service/scripts/face_detection.python
```

This will show the currently detected faces using MTCNN including the corresponding feature points in an extra window. Integration with ROS is currently missing on this branch because of python 2 vs. python 3 constraint using ROS and Tensorfloe at the same time.

This is why ROS communication was only implemented in **dirty_final_hack** branch (https://github.com/Roboy/Vision/tree/dirty_final_hack). Here an alternative approach for Communication using File I/O between python 2 and python 3 was implemented. Using this approach ROS will run in python 2, whereas face detection with tensorflow will run seperatly using python 3. This branch can be removed as soon as proper Python 3 integration was achieved. In this branch 2 services are offered:

- Face nearby: Query whether a face is nearby
- Recognize face: Recognize name of a given face

Context

The Vision package receives ZED stereo camera input which is then processed internally. Other than that it should also receive data from Roboy motor control about roboy's current position. This can be used to calculate the relative and absolute positions of detected objects.

The main output of the Vision Module will be detected objects and some object properties (e.g. detected face, name of person, mood, gender, ...). The receiver for this will be Natural language processing. This is illustrated in the following contect overview:

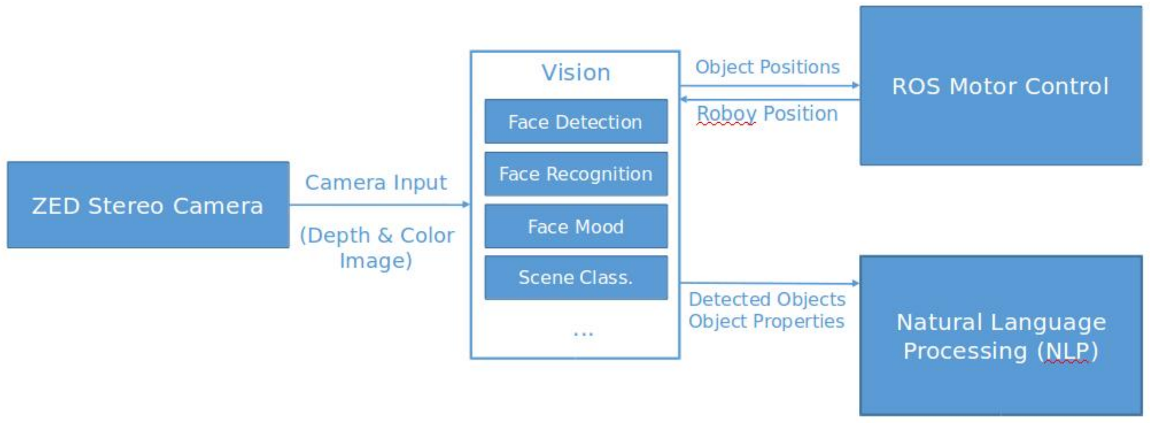


Fig. 2.1: **Context diagram** - shows the birds eye view of the vision system (black box) described by this architecture within the ecosystem it is to be placed in. Shows orbit level interfaces on the user interaction and component scope.

Solution Strategy

Basic decisions for Vision Package:

- Seperation of different tasks into sub-modules (Face Detection, Object Detection, Object tracking, face recognition, mood recognition, age estimation, scene classification, ...)

- Highest priority on face detection and face pose estimation. Recognition of people as second priority. All other properties concerning people with lower priority and general object detection with least importance.
- Face detection using this approach: Joint Face Detection and Alignment using MTCNNs. Good real time performance, other modules to be built on top.
- Face embeddings using FaceNet. These embeddings can be used for recognition.
- Speaker detection using facial landmarks from DLIB
- Ojbect recognition using YOLO

Current implementation:

- **RoboyVision as main, handling all sub-modules:**
  - **Face Detection** using Facenet for calculating embeddings for a given face and SVM for classification. SVM currently trained on pictures of LFW (labelled Faces in the Wild) dataset, using Roboy Team members as next step. Sends coordinates to **Tracker** and facial landmarks to **Speaker Detection**
  - **Speaker Detection** using DLIB's facial landmarks to caluclate specific mouth parameters (width, lip distance) of each face to determine, whether a person is speaking
  - **ROS services** are handled by RoboyVision via websocket
  - **Object recognition** is implemented based on YOLO
  - **Tracking objects/faces** running in realtime. This implementation is based on the MIL(Visual Tracking with Online Multiple Instance Learning). Also part of the Opencv_contrib module.

Plan for this semester with priorities in red (5 being highest priority):

Future plans on current implentation: * Improve tracking by implementing the GOTURN algorithm.

Architecture of the current System:

Public Interfaces

Interfaces to other modules will be realized using ROS Communication. Currently 3 interfaces have been designed for communication with NLP:

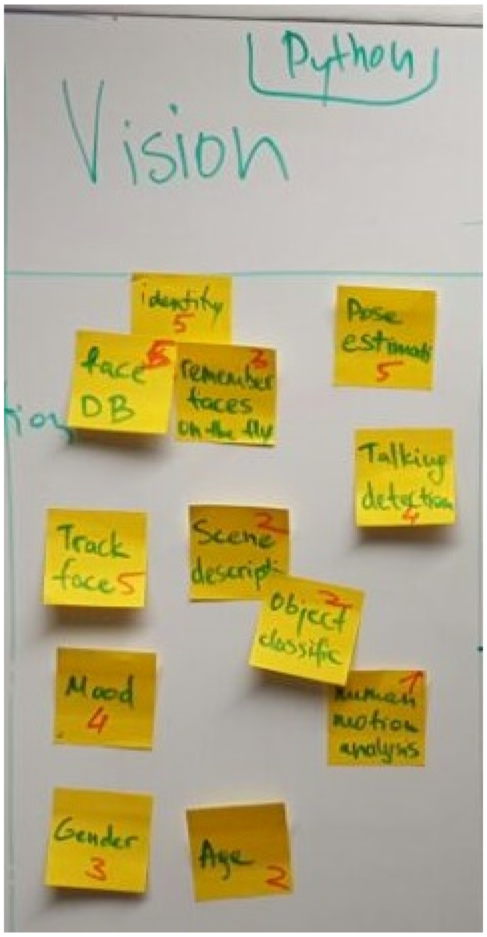
- **wakeup service**: Service to be called to test whether a face is recognized within certain distance. Used by NLP for wakeup:

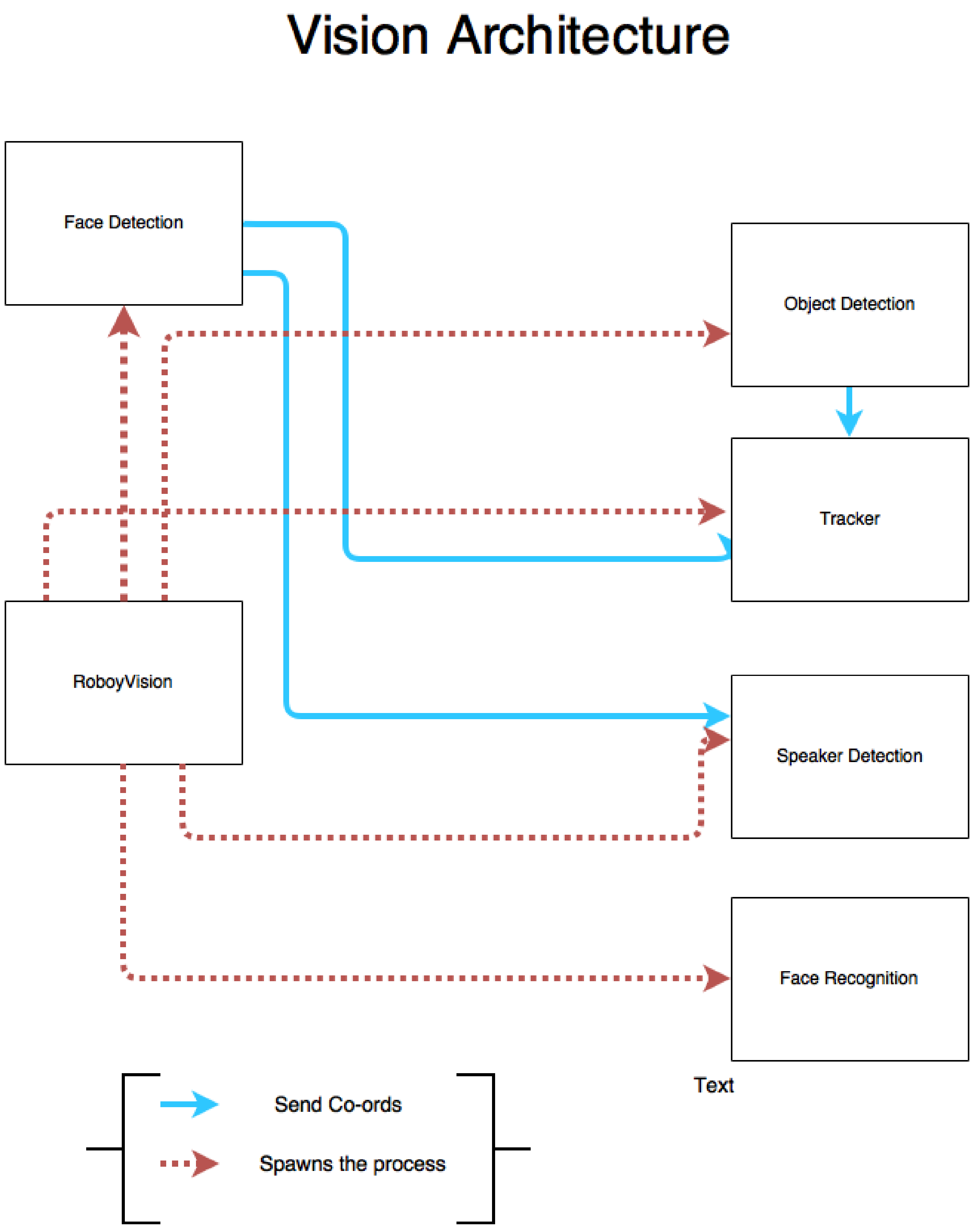
```
# argument: (none)
# returns: Bool face_nearby

rosservice call /detect_face
```

- **recognition service**: Service called to recognize a face. Given a object ID the name of the detected person is returned:

```
# argument: int object_id
# returns: String name

rosservice call /recognize_face *object_id*
```

- **vision_object message**: This message is published for every camera frame an object was detected. The message includes ID type and position of the object.


Python
Vision
identify 5
face DB 5
remember faces on the fly 3
Pose estimati 5
Talking detection 4
Track face 5
Scene descripti 2
Object classific 2
Mood 4
human motion analysis 1
Gender 3
Age 2


Vision Architecture
Face Detection
Object Detection
Tracker
RoboyVision
Speaker Detection
Face Recognition
Text
Send Co-ords
Spawns the process

Currently these interfaces are not implemented because of the constraint that ROS could not run using python 3. Because of this a workaround was implemented in **dirty_final_hack** branch. This will allow running ROS nodes using Python 2.7 and running the Vision module using python 3.6. The two modules then communicate using File I/O. For further details refer to github: https://github.com/Roboy/Vision/tree/dirty_final_hack

Architecture Constraints

Hardware Constraints

Operating System Constraints

Programming Constraints

Constraint Name	Description
Python 3	Tensorflow v1.0 required for facenet implementation which uses Python 3
Python 2	ROS requires Python 2 still

Conventions

We follow the coding guidelines:

Language	Guideline	Tools
Python	https://www.python.org/dev/peps/pep-0008/	pep-format: https://github.com/google/yapf
C++	http://wiki.ros.org/CppStyleGuide	clang-format: https://github.com/davetcoleman/roscpp_code_format

Libraries and external Software

Name	URL/Author	License	Description
MTCNN face detection & alignment	https://github.com/kpzhang93/MTCNN_face_detection_alignment	MIT license	Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Neural Networks
Facenet	https://github.com/davidsandberg/facenet	MIT License	Face recognition using Tensorflow
Object Detection	https://github.com/pjreddie/darknet	GNU General Public License	Object Detection Using YOLO9000
arc42	http://www.arc42.de/template/	Creative Commens Attribution license.	Template for documenting and developing software
DLIB	https://github.com/davisking/dlib	Boost Software License	ML toolkit, mainly used for face detection and facial landmarks
YOLO	https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection		Real-time object recognition system

Presentations

This section lists presentations of Roboy Vision Team.

Midterm/ Final Presentations

- Midterm Presentation WS 2016/17
- Final Presentation WS 2016/17

Other Presentations

- RoadMap 2017

About arc42

Note

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

Literature and references

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6, Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

Examples

- HTML Sanity Checker
- DocChess (german)
- Gradle (german)
- MaMa CRM (german)
- Financial Data Migration (german)

Acknowledgements and collaborations

arc42 originally envisioned by Dr. Peter Hruschka and Dr. Gernot Starke.

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in GitHub under the aim42-Organisation.

Issues We maintain a list of open topics and bugs.

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel
- Ralf D. Müller
- Peter Hruschka
- Jürgen Krey

Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel
- Axel Scheithauer