

# Einführung in ROS und die PSES-Plattform



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Projektseminar Echtzeitsysteme WS2016/17



Robot Operating System



# Was ist ROS ?

- Metabetriebssystem für Roboter
  - Läuft auf Linux (Ubuntu)
  - Hardwareabstraktion
  - Bereitstellung häufig benötigter Funktionen
  - Kommunikation zwischen Prozessen (Nodes)
  - Paketverarbeitung und Building-Tools
- Verteilbar auf mehrere Systeme in einem Netzwerk
- Bietet verschiedene Analysewerkzeuge (Netzwerktopologie und -traffic)
- Open-Source-Bibliothek für typische Probleme in der Robotik
- Unterstützt verschiedene Programmiersprachen (C/C++, Python, Java)



# Was ist ROS nicht?

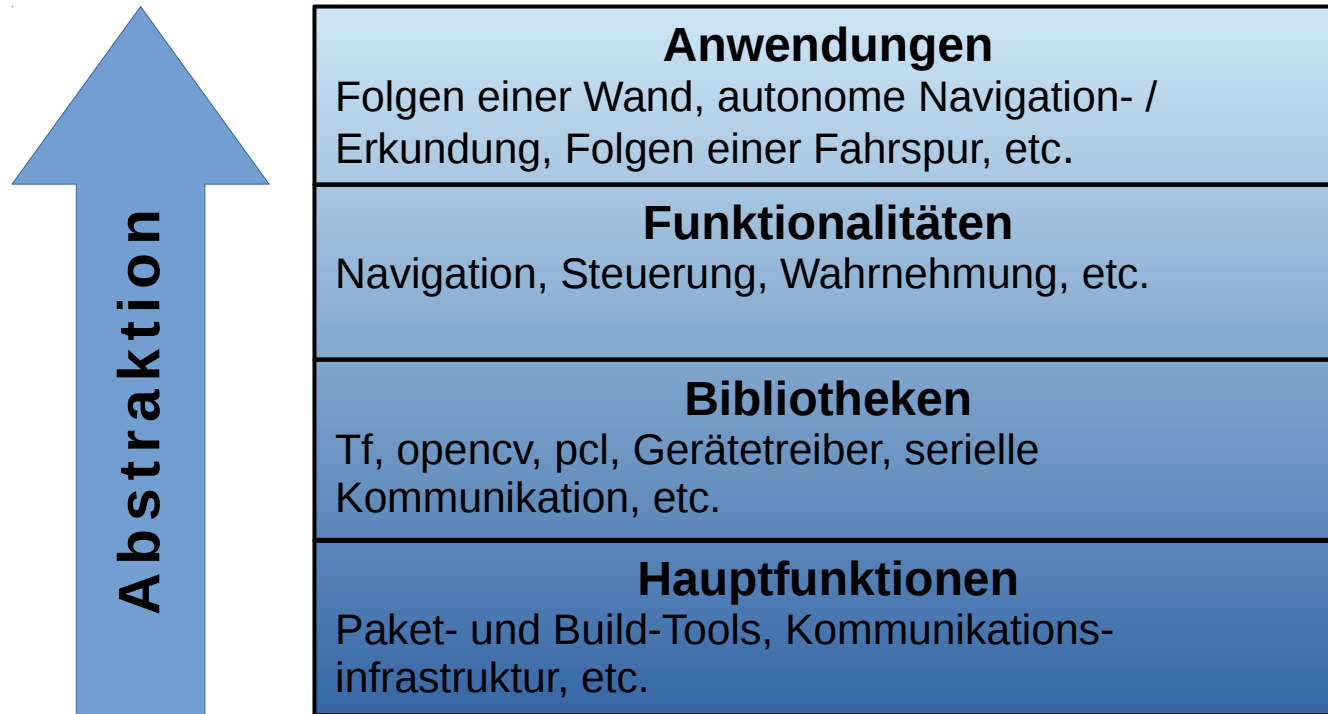
- Ein „echtes“ Betriebssystem
- Eine Programmiersprache
- Eine Entwicklungsumgebung (IDE)
- Eine Architektur die harte Echtzeitbedingungen erfüllt



# Was haben wir von ROS?



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



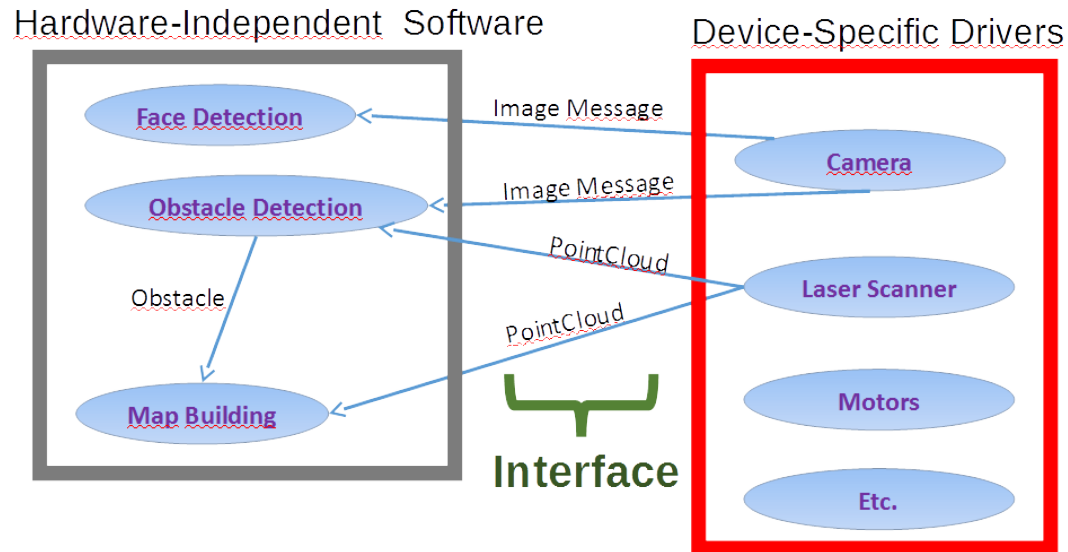
- Verschiedene Abstraktionsniveaus
  - Erleichtert Entwicklung und Wartung
- Einheitliche Schnittstellen zwischen und innerhalb der Schichten
  - Wiederverwendbarkeit von Paketen



# Was haben wir von ROS?



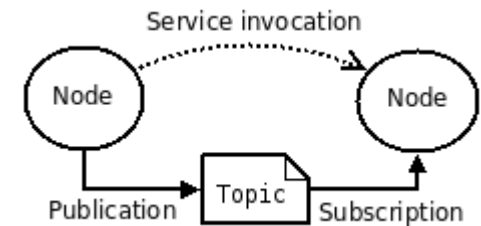
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



- Verschiedene Abstraktionsniveaus
  - Erleichtert Entwicklung und Wartung
- Einheitliche Schnittstellen zwischen und innerhalb der Schichten
  - Wiederverwendbarkeit von Paketen



- ROS-Core (wird immer benötigt)
  - ROS-Master: steuert Kommunikation zwischen Prozessen, registriert und kennt die Namen der ROS-Graph Knoten
  - ROS-Graph (Peer-To-Peer-Netzwerk von Prozessen)
  - Parameter Server (speichert Parameter und andere Daten)
  - Rosout (ein „cout“ für ROS)
- ROS-Graph Komponenten
  - Nodes (übers ROS-Netzwerk verteilte Prozesse)
  - Topics (asynchrone many-to-many Kommunikation mit Publishern und Subscribern)
  - Services (synchrone one-to-many Kommunikation für einfache Funktionen)
  - Parameter (Konfigurations- und Initialisierungseinstellungen auf dem Parameter Server)



- Unabhängige Prozesse im ROS-Netzwerk
- Kommunizieren über Topics
  - Publisher-Objekte zum Senden auf Topics
  - Subscriber-Objekte zum Empfangen von Topics (asynchron über Callbacks)
- Darin werden entwickelte Module ausgeführt
- Für Module gilt: so wenig ROS-Abhängigkeiten wie möglich in eure Algorithmen
- Die Nodes sind nur ein Interface für eure Module ins ROS-Netzwerk



# Erstellen von Nodes



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 #include "ros/ros.h"
2 #include "std_msgs/String.h"
3 #include <string>
4 #include <sstream>
5
6 std::string received;
7
8 void chatterCallback(const std_msgs::String::ConstPtr& call){
9     received = call->data;
10 }
11
12 int main(int argc, char **argv)
13 {
14     ros::init(argc, argv, "node1");
15     ros::NodeHandle n;
16
17     ros::Publisher chatter_pub = n.advertise<std_msgs::String>("node1/chatter", 10);
18     ros::Subscriber chatter_sub = n.subscribe<std_msgs::String>("node2/chatter", 10, chatterCallback);
19
20     ros::Rate loop_rate(10);
21     int count = 0;
22     while (ros::ok())
23     {
24         std_msgs::String msg;
25
26         std::stringstream ss;
27         ss << "Node1: -hello world- count:" << count;
28         msg.data = ss.str();
29         chatter_pub.publish(msg);
30         count++;
31
32         ROS_INFO("%s", received.c_str());
33
34         ros::spinOnce();
35
36         loop_rate.sleep();
37     }
38
39     return 0;
40 }
```





# Compilen der Nodes mit CMake



```
1 cmake_minimum_required(VERSION 2.8.3)
2 project(example)
3
4 ## Find catkin and any catkin packages
5 find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs)
6
7 ## Declare ROS messages and services
8
9 ## Generate added messages and services
10
11 ## Declare a catkin package
12 catkin_package()
13
14 ## Build talker and listener
15 include_directories(include ${catkin_INCLUDE_DIRS})
16
17 add_executable(node1 src/node1.cpp)
18 target_link_libraries(node1 ${catkin_LIBRARIES})
19
20 add_executable(node2 src/node2.cpp)
21 target_link_libraries(node2 ${catkin_LIBRARIES})
```

- CMakeList.txt immer im Hauptverzeichnis eines Paketes



- Manuelle Variante:
  1. ROS-Core starten mit „roscore“
  2. ROS-Node 1 starten mit „roslaunch example node1“
  3. ROS-Node 2 starten mit „roslaunch example node2“
    - Mühsam mit vielen Nodes

- automatische Variante mit einem Launch-File:

```
<?xml version="1.0" encoding="utf-8"?>
<launch>
  <node name="node1" pkg="example" type="node1" output="screen"/>
  <node name="node2" pkg="example" type="node2" output="screen"/>
</launch>
```

- Wird mit „roslaunch example <filename>.launch“ aufgerufen
- Launch-Files werden nach Konvention in „<package\_name>/launch“ abgelegt
- Mit der Endung „.launch“

# Nützliche ROS Core-Bibliotheken

- TF (Transformation) → <http://wiki.ros.org/tf>
  - Hilft bei der Transformation von Punkten in verschiedene Koordinatensysteme
- Navigation Stack → <http://wiki.ros.org/navigation>
  - Voraussetzung für Wegplanung und autonom. Navigation
  - Benötigt Odometrie (Positionsbestimmung)
  - Benötigt eine Karte (z.B. mit dem Paket „map\_server“)
  - Benötigt einen Laserscan o.ä.
- Dynamic\_reconfigure → [http://wiki.ros.org/dynamic\\_reconfigure](http://wiki.ros.org/dynamic_reconfigure)
  - Nutzt den Parameter Server um Variablen in Nodes zu setzen
  - Werte können geändert werden ohne neues Compilen
  - Zur Laufzeit (rqt\_reconfigure) oder beim Start (Launch-File)



# Verteilung auf mehrere Systeme

- System A: ist der ROS-Master
  - System B: ist das neue System
- 1) ROS über die IP-Adresse des Systems A informieren: Dafür muss man bei jedem neuen Terminal-Fenster im System A den folgenden Befehl tätigen `export ROS_IP="ipA"` und `export ROS_HOSTNAME="ipA"` → dabei ist <ipA> der IP-Adresse des Systems A
  - 2) Bei B die URI des Masters in jedem neuen Terminal mit `export ROS_MASTER_URI = "http://<ipA>:11311/"` setzen, sowie `export ROS_IP="ipB"` und `ROS_HOSTNAME="ipB"` → wobei <ip> wiederum der IP-Adresse des Systems A und <ipB> der IP des Systems B entspricht



# Abschließende Empfehlungen

- Tutorials auf der ROS-Wiki lesen → <http://wiki.ros.org/ROS/Tutorials>
  - Besonders wichtig ist dabei:
  - Erstellen/Reparieren eines catkin workspace
  - Erstellen eines ROS-Paketes im workspace
  - Installieren von neuen Paketen
- Weiterführende Tutorials
  - Aufsetzen eines Navigational Stacks → <http://wiki.ros.org/navigation/Tutorials>
  - Visualisierung → <http://wiki.ros.org/rviz/Tutorials>
  - Verteilung auf mehrere Systeme → <http://wiki.ros.org/ROS/Tutorials/MultipleMachines>
  - Generierung von selbstdefinierten ROS-Messages  
<http://wiki.ros.org/ROS/Tutorials/DefiningCustomMessages>
  - Aufnahme und „offline“ Wiedergabe von Topics → <http://wiki.ros.org/rosbag/Tutorials>



# PSES-Plattform



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Kinect v2

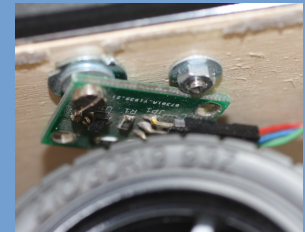
- Tiefenkamera
  - Punktwolke mit Tiefeninformationen
- Farbkamera

## Was man nicht sieht:

- 3-Achsen Gyrosensor ( $^{\circ}/s$ )
- 3-Achsen Beschl.sensor ( $m/s^2$ )
- Magnetometer (Kompass)
  - Noch inaktiv

## Ultraschall Sensor (Entfernung in 0.1mm)

## Hall-Sensor



(Radumdrehung in ms)



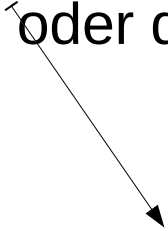
- µController-Board
  - STM32 µController (ARM-Cortex M4 mit FPU)
  - Kommunikation mit dem Mainboard via UART
  - Sensorinformation wird gruppiert verschickt
  - Mit einfachen Kommandos konfigurierbar
  - Quellcode, Doku und Datasheets → <https://github.com/tud-pses/ucboard>
- Mainboard
  - Mitac PD10BI MT Thin Mini-ITX
  - Intel CeleronJ Quadcore 2.0Ghz (2.4Ghz Burst)
  - Intel HD Onboard Grafikkarte (unterstützt OpenCL)
  - 8 GB DDR3-1600 RAM
  - 60 GB SSD
  - Intel Dual Band Wireless AC Netzwerkadapter
  - Betriebssystem Ubuntu 14.04 LTS (64Bit)
    - Username: pses
    - Password: letmein
  - ROS Indigo (Aktualisierung nicht empfohlen)

- **pses\_basis** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Node: **car\_handler**
    - Abstrahiert die Kommunikation mit dem  $\mu$ Controller
    - Published die gesammelten Sensordaten auf dem Topic „pses\_basis/sensor\_data“

```
1 PsesHeader header
2 float32 accelerometer_x
3 float32 accelerometer_y
4 float32 accelerometer_z
5 float32 angular_velocity_x
6 float32 angular_velocity_y
7 float32 angular_velocity_z
8 float32 hall_sensor_dt
9 float32 hall_sensor_dt_full
10 uint32 hall_sensor_count
11 float32 range_sensor_front
12 float32 range_sensor_left
13 float32 range_sensor_right
14 float32 system_battery_voltage
15 float32 motor_battery_voltage
```



- **pses\_basis** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Node: **car\_handler**
    - Abstrahiert die Kommunikation mit dem  $\mu$ Controller
    - Published die gesammelten Sensordaten auf dem Topic „pses\_basis/sensor\_data“
    - Subscribed das Topic „pses\_basis/command“ für Steuerbefehle and die Lenkung oder den Antrieb



|   |                      |
|---|----------------------|
| 1 | PsesHeader header    |
| 2 | bool reset           |
| 3 | int16 steering_level |
| 4 | int16 motor_level    |
| 5 | bool enable_kinect   |

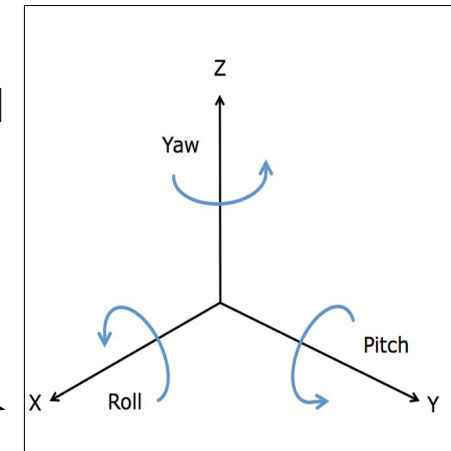


# PSES-Basis

- **pses\_basis** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Node: **car\_odometry**
    - Berechnet aus den Drehraten  $\omega_x$ ,  $\omega_y$ ,  $\omega_z$  die Lagewinkel **Roll, Pitch und Yaw** → Topic „pses\_basis/car\_info“
    - Drehraten sind nur mit einem einfachen Komplementärfilter gefiltert um das Klettern der Winkel zu reduzieren (noch nicht perfekt)
    - Berechnung Geschwindigkeit und gefahrener Strecke anhand des Hall-Sensors → Topic „pses\_basis/car\_info“
    - Berechnung der Odometrie aus Lagewinkeln und zurückgelegter Strecke → **Topic „odom“**

```

1 PsesHeader header
2 float32 roll
3 float32 pitch
4 float32 yaw
5 float32 driven_distance
6 float32 speed
    
```



```

std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
    
```

```

geometry_msgs/Point position
geometry_msgs/Quaternion orientation
    
```

```

geometry_msgs/Twist twist
float64[36] covariance
    
```

```

float64 x
float64 y
float64 z
    
```

```

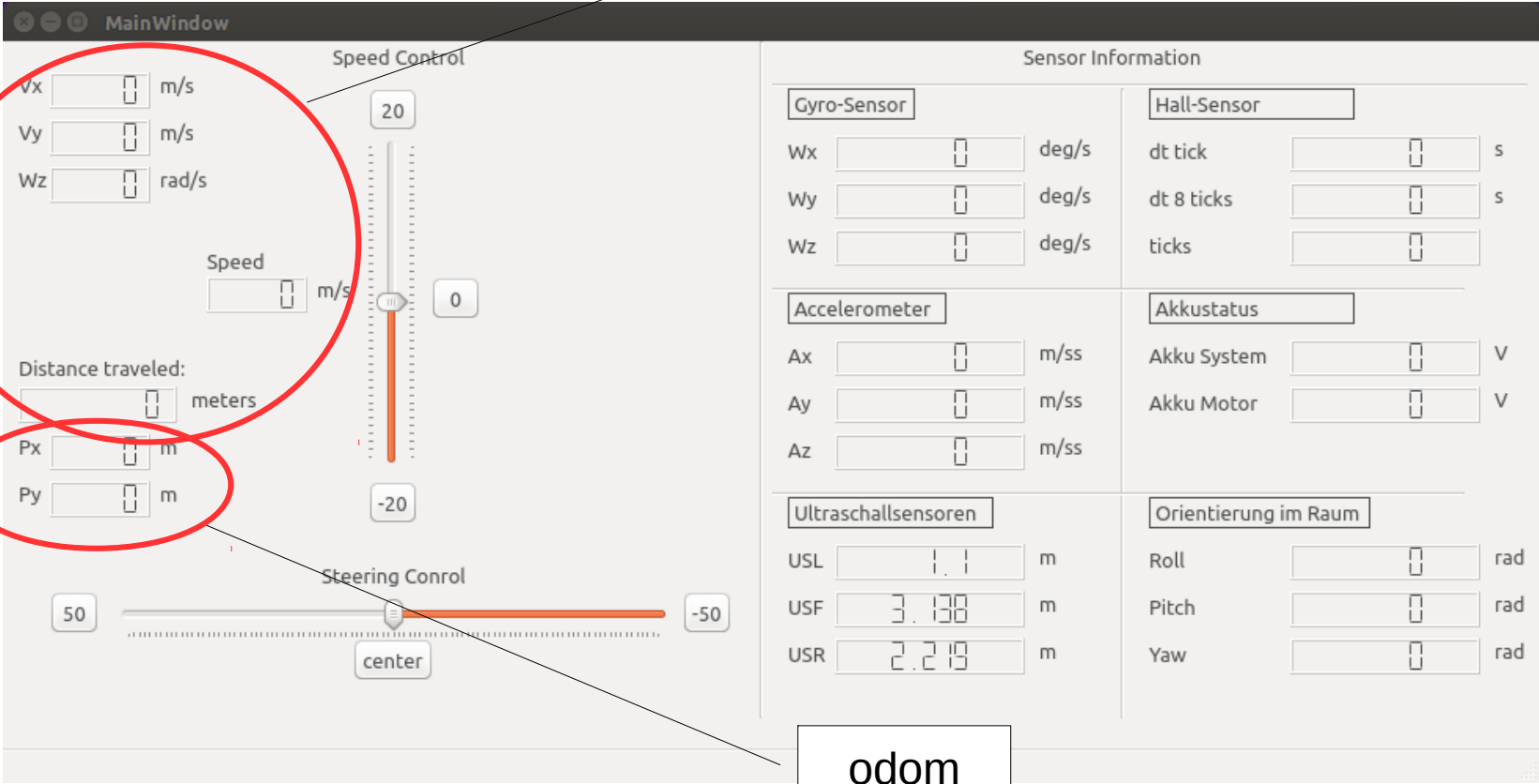
float64 x
float64 y
float64 z
float64 w
    
```

```

geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
    
```



- **pses\_basis** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Node: **car\_dashboard**



The screenshot displays the 'car\_dashboard' ROS node interface, titled 'MainWindow'. It is divided into two main sections: 'Speed Control' on the left and 'Sensor Information' on the right.

**Speed Control Section:**

- Angular velocities: Vx, Vy, Wz (all 0) in m/s and rad/s.
- Linear velocity: Speed (0) in m/s.
- Distance traveled: 0 meters.
- Position: Px, Py (both 0) in meters.
- Steering Control: A slider from -50 to 50, currently at 0 (center).

**Sensor Information Section:**

- Gyro-Sensor:** Wx, Wy, Wz (all 0) in deg/s.
- Hall-Sensor:** dt tick, dt 8 ticks, ticks (all 0) in seconds.
- Accelerometer:** Ax, Ay, Az (all 0) in m/ss.
- Akkustatus:** Akku System, Akku Motor (both 0) in Volts (V).
- Ultraschallsensoren:** USL (1.1), USF (3.138), USR (2.219) in meters (m).
- Orientierung im Raum:** Roll, Pitch, Yaw (all 0) in radians (rad).

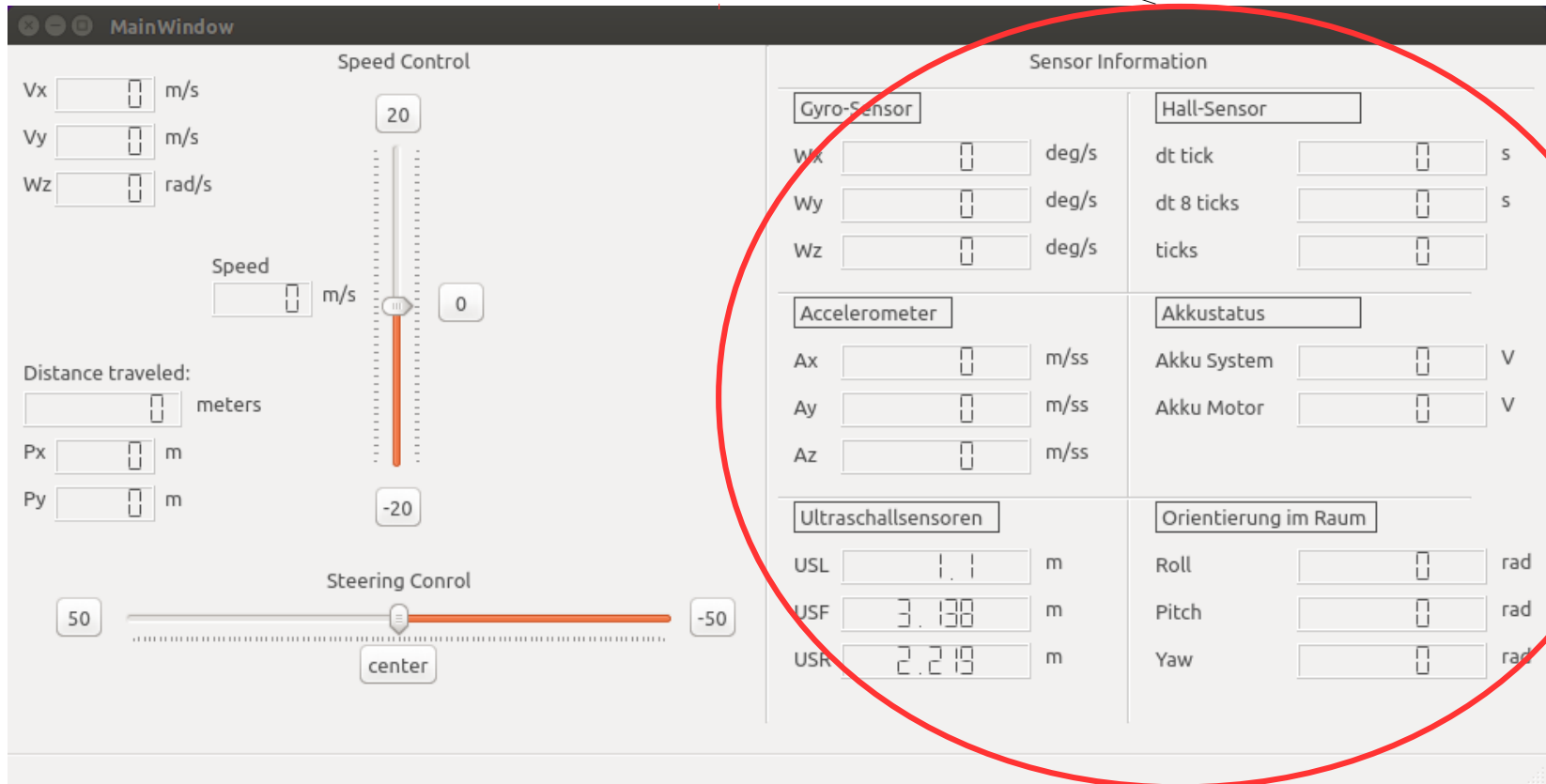
Annotations with red circles and lines:

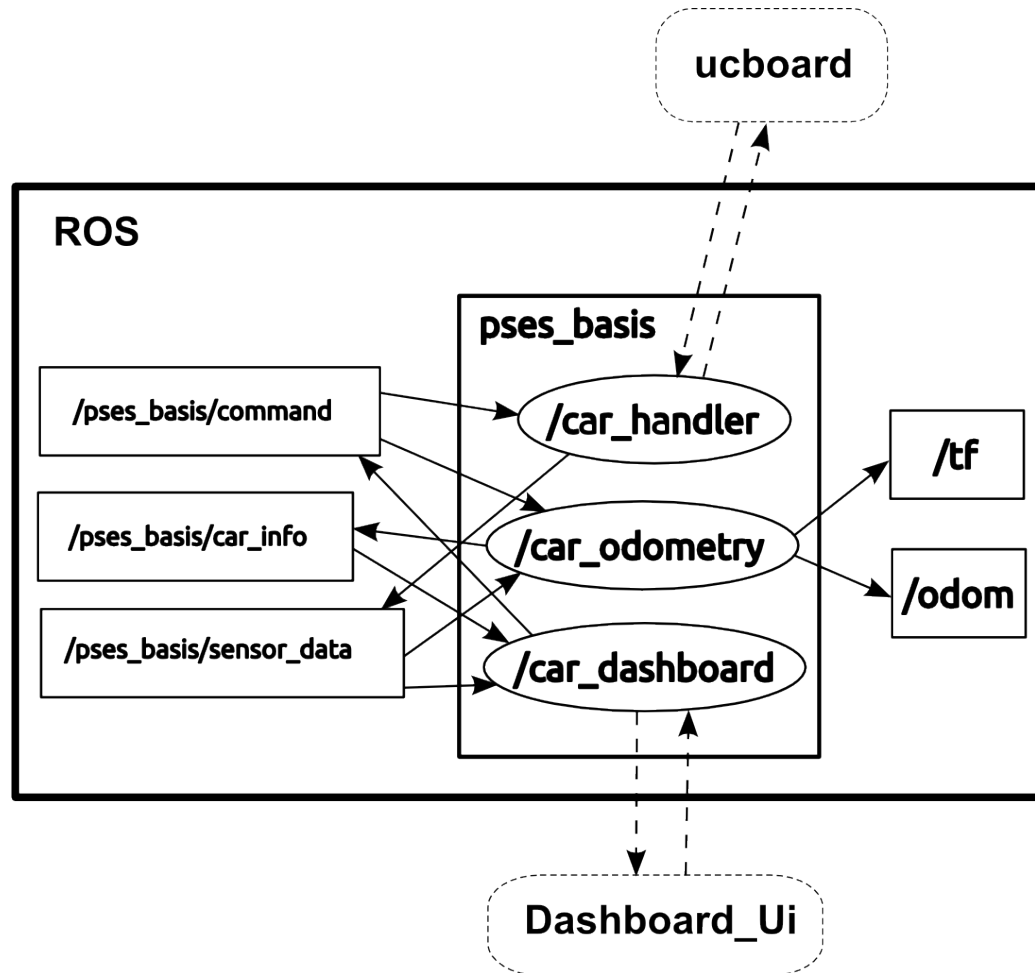
- A red circle highlights the velocity fields (Vx, Vy, Wz, Speed).
- A red circle highlights the position fields (Px, Py).
- A line points from the 'car\_info' label to the 'Speed Control' section.
- A line points from the 'odom' label to the 'Distance traveled' field.



- **pses\_basis** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Node: **car\_dashboard**

sensor\_data





- **pses\_basis** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Bedienung:
    - Zuerst „sudo su“
    - Start via Launchfile → „roslaunch pses\_basis pses\_basis.launch“
    - Beenden mit „strg+c“ im Terminal



- **pses\_simulation** ROS-Paket →  
<https://github.com/tud-pses/PSES-Basis>
  - Node: **simulation\_control**
    - Subscribed das Topic „pses\_basis/command“ für Steuerbefehle and die Lenkung oder den Antrieb
    - Published auf dem Topic „pses\_basis/sensor\_data“ synthetische Sensordaten
    - Published auf dem Topic „pses\_basis/car\_info“ Roll, Pitch, Yaw, gefahrene Distanz und Geschwindigkeit
      - Berechnet aus einem Modell des Autos
    - Berechnet zu jedem Zeitschritt die Odometrie eines Modells anhand von Steuereingaben



- **pses\_simulation** ROS-Paket →  
<https://github.com/tud-pses/PSES-Basis>
  - Node: **simulation\_usscan**
    - Simuliert die Ultraschallsensoren
    - Benutzt dafür Odometrie aus `simulation_control`
    - An der Position des Roboters wird eine Karte in Blickrichtung des Sensors abgetastet
    - Published auf dem Topics „front us range“, „left\_us\_range“ und „right\_us\_range“ synthetische Sensordaten

```
uint8 ULTRASOUND=0
uint8 INFRARED=1
std_msgs/Header header
uint8 radiation_type
float32 field_of_view
float32 min_range
float32 max_range
float32 range
```



- **pses\_simulation** ROS-Paket →  
<https://github.com/tud-pses/PSES-Basis>
  - Node: **simulation\_laserscan**
    - Simuliert einen Laserscan
    - Benutzt dafür Odometrie aus `simulation_control`
    - An der Position des Roboters wird eine Karte in Blickrichtung des Sensors über das komplette Field of View abgetastet
      - wie in `usscan` nur mit mehr Entfernungswerten
  - Published auf dem Topic „scan“ synthetische Sensordaten eines Laserscanners

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

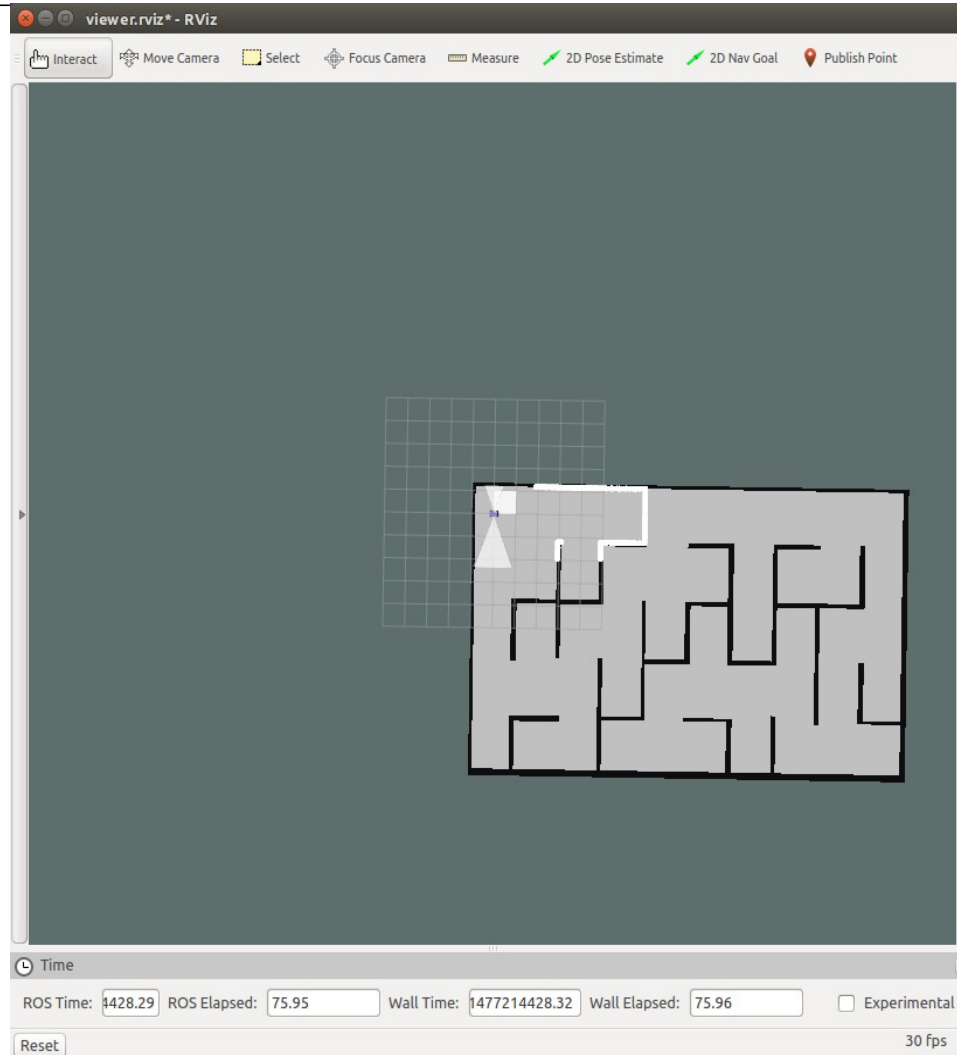
- **pses\_simulation** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Node: **simulation\_dashboard**
    - Einfache GUI zur Kontrolle → wie bei car\_dashboard
  - Node: **simulation\_measurement**
    - Zum Aufzeichnen des zurückgelegten Pfades gedacht
    - Ausgabe in „pses\_simulation/data/output/“ als .png-Datei



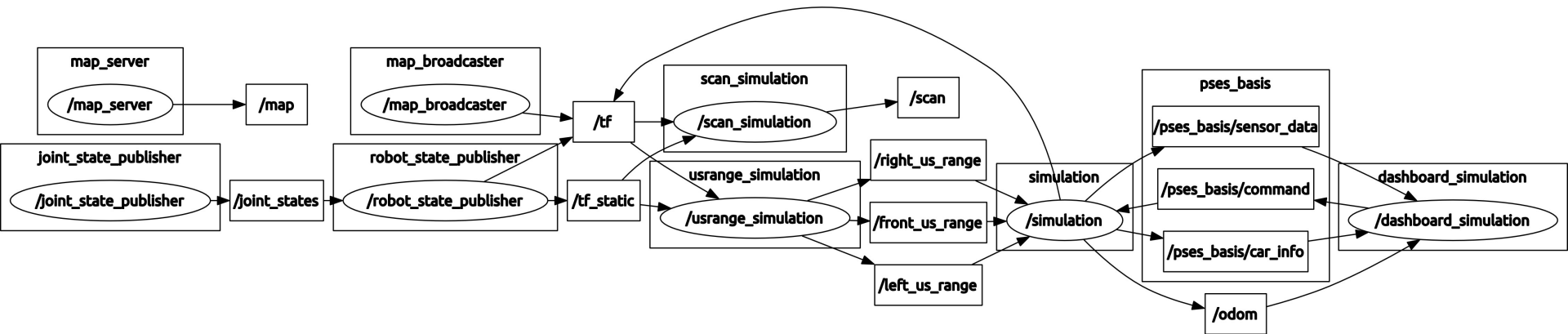
# PSES-Simulation Visualisierung in Rviz



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



# PSES-Simulation Kommunikation



- **pses\_simulation** ROS-Paket → <https://github.com/tud-pses/PSES-Basis>
  - Bedienung:
    - Start via Launchfile → „roslaunch pses\_simulation simulation.launch“
    - Beenden mit „strg+c“ im Terminal
    - Verwenden einer anderen Karte durch ersetzen im Verzeichnis „pses\_simulation/data/map/“
    - Alternative Karten in „pses\_simulation/data/map/alternative maps/“
    - Im Allgemeinen bedient Simulation die selben Schnittstellen wie Basis
      - Pakete die in der Simulation funktionieren sollten auch auf Basis funktionieren



# CarControl-App (1/2)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

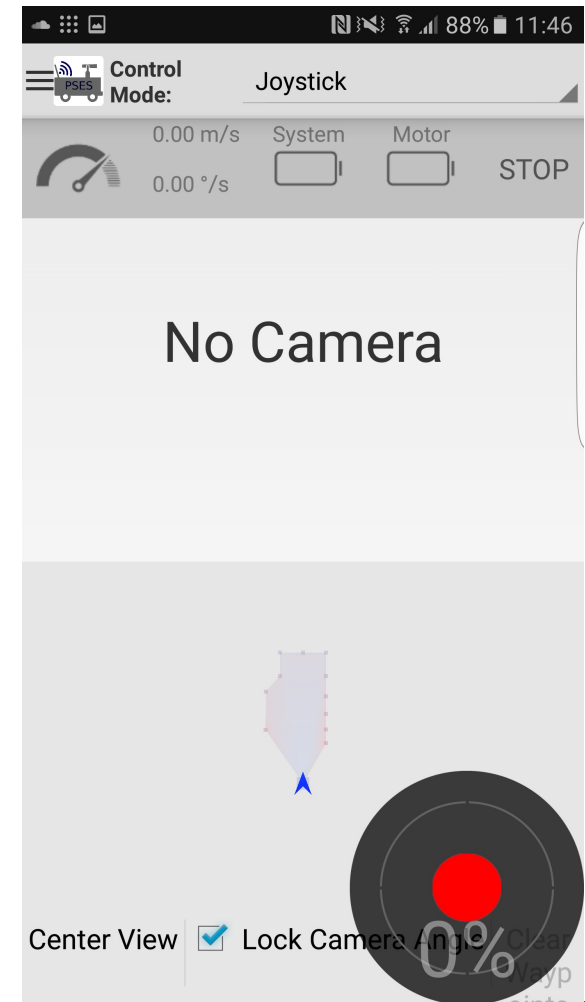
- Eine Android-App zur Fernsteuerung des Roboters und Darstellung der gesammelten Sensordaten wird bereitgestellt

→ <https://github.com/tud-pses/CarControl-App>

- Einige Bugs werden noch beseitigt
  - Updates werden in der Repository hochgeladen

## Voraussetzungen

- Android 4.0+
- Zuverlässige WLAN-Verbindung
  - eduroam :)
  - Mobile Hotspot
  - Verbindung über eigenen Router
  - Ad-hoc-Netz (Nur für gerootete Geräte)

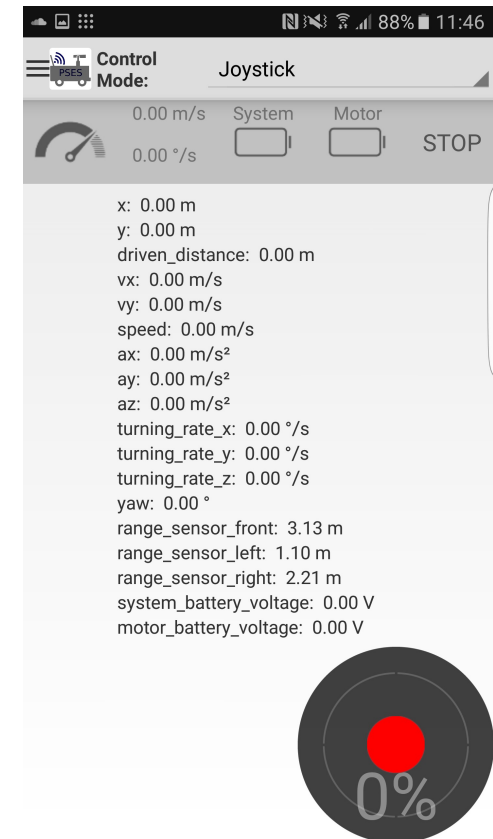


# CarControl-App (2/2)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Die App ist mit ROS integriert und agiert als ein ROS-Node, der über die WLAN-Verbindung mit dem ROS-Master auf dem Mainboard kommuniziert
- Die App lässt sich als Alternative zum ROS-Node „car\_dashboard“ verwenden



[Darstellung der Sensordaten]



# Letzte Hinweise

- Alle Folien, die Pakete und die Doku zum Board
  - <https://github.com/tud-pses>
- Doku zum Basispaket ist noch in Arbeit
- Code noch unkommentiert
- Bei Problemen und Fragen → Email an uns
  - Nicolas: [aceronicolas@hotmail.com](mailto:aceronicolas@hotmail.com)
  - Sebastian: [sebastian.ehmes@gmx.de](mailto:sebastian.ehmes@gmx.de)
- Danke für die Aufmerksamkeit! Fragen ?

