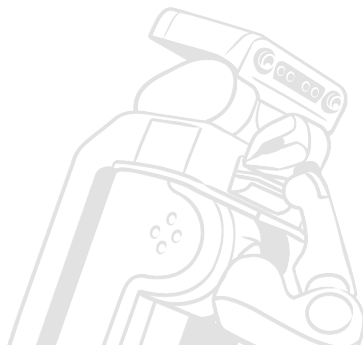


Principles of driver wrapping

Ingo Kresse (kresse@in.tum.de)

Intelligent Autonomous Systems Group
Technische Universität München

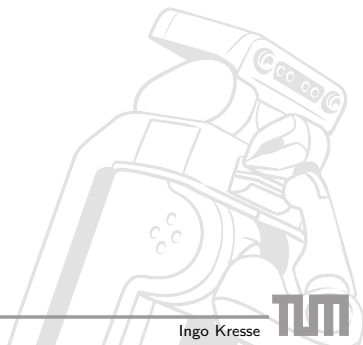
July 18, 2011





Outline

1. Motivation
2. Example Driver
3. Extra Features
4. Conclusion

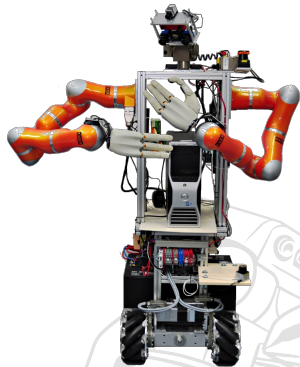




Motivation

Why hardware drivers in ROS?

- ▶ abstract away details of the hardware
- ▶ common interface for a type of hardware
- ▶ share a piece of hardware between programs
- ▶ deal with error conditions



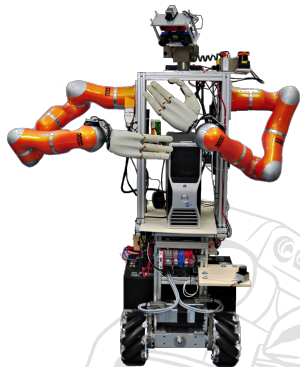


Examples for drivers

ROSIEs hardware:

- ▶ 47 motors (arms, hands, wheels, ...)
- ▶ 11 sensors (lasers, cameras, kinect, ...)

They all can fail independently.





Functionality of a ROS drivers

a good ROS driver should:

- ▶ provide standard ROS topics for sensor data / commands
- ▶ read its configuration data from roscore (roscparam)
- ▶ restart when necessary
- ▶ report current hardware state (rosout, diagnostics)

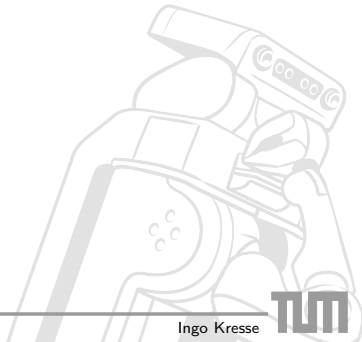
The screenshot displays three ROS diagnostic windows:

- Driver Status:** Shows the driver is running. Full name: /Laser/Laser Front/Driver Status. Component: Driver Status. Hardware ID: H0506272. Level: OK. Message: OK: Streaming. Driver status: RUNNING. Latest status message: Streaming data. Device Status: Sensor works well. Part: (Schunk/Robiq, H0506272). Device ID: H0506272. Scan Thread Load Count: 0. Connected Scan Count: 0. Vendor Name: Holroyo Automatic Co., Ltd. Product Name: SCHUNK Sensor UR3-04LK. Firmware Version: 3.2007163ur/0908. Protocol Version: SCP 2.0. Connected Latency: 99024230. User Time Offset: 0.
- Errors:** Lists several error messages:
 - Actuators: Error
 - Actuators/Arm Left: Dx
 - Actuators/Arm Right: Dx
 - Actuators/Base: Down
 - IC: 2, W: 1) Actuators: Error
 - Arm Left: Dx
 - Arm Right: Dx
 - Roll: Down
 - Hands: left and right hand ok
 - PTU Action Server: We are still alive
 - PTU and Firing Laser: We are still alive
 - Connectors: OK
 - Lasers: OK
 - Wimoto: OK
- ROS Message Log:** Shows a sequence of messages including frame transforms, device connections, calibration progress, and sensor data streaming. It also includes a warning about a C++ action server.



Outline

1. Motivation
2. Example Driver
3. Extra Features
4. Conclusion

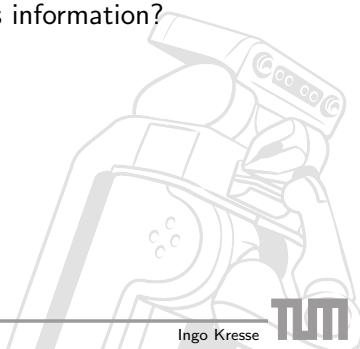




Interface design

Things to consider:

- ▶ Hardware capabilities:
 - ▶ What commands does the driver understand?
 - ▶ What data does the driver publish?
- ▶ What messages exist that convey this information?
(→ `rosmmsg`)
- ▶ How are other drivers doing it?





Example Interface

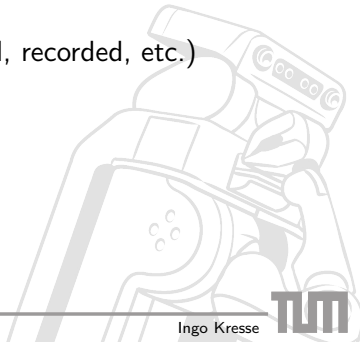
Example: mobile robot base

Interface

- ▶ commanded velocity: `geometry_msgs/Twist`
- ▶ odometry: `tf`

Advantages of this interface:

- ▶ all based on topics (can be monitored, recorded, etc.)
- ▶ used by other drivers as well
- ▶ simple





Starting to code

Let's use roscpp:

manifest.xml:

```
...  
<depend package="roscpp"/>  
<depend package="geometry_msgs"/>  
<depend package="tf"/>  
...
```

src/cool_new_base.cpp:

```
#include <ros/ros.h>  
  
...  
  
int main(int argc, char *argv[])  
{  
    ros::init(argc, argv, "cool_new_base");  
    ros::NodeHandle nh("~");  
    ...  
}
```



Receiving commands

Let's parse the twist command:

src/cool_new_base.cpp:

```
#include <geometry_msgs/Twist.h>

...

void callback(const geometry_msgs::Twist::ConstPtr& msg)
{
    double fwd = msg->linear.x;
    double rot = msg->angular.z;
    ...
}

main(...)
{
    ...
    ros::Subscriber sub = nh.subscribe("/cmd_vel", 1, callback);
    ...
}
```



Publish odometry

We provide the odometry by publishing a tf transform:

src/cool_new_base.cpp:

```
#include <tf/transform_broadcaster.h>

...

tf::TransformBroadcaster transforms;

...

tf::Transform pose(tf::Quaternion(tf::Vector3(0, 0, 1), theta),
                  tf::Point(x, y, 0.0));

transforms.sendTransform(tf::StampedTransform(pose, ros::Time::now(),
                                              "/odom", "/base_link"));

...
```



Make it run

Give ROS a chance to work:

src/cool_new_base.cpp:

```
...  
  
while(true) // main loop  
{  
    ...  
  
    ros::spinOnce();  
}  
  
...
```

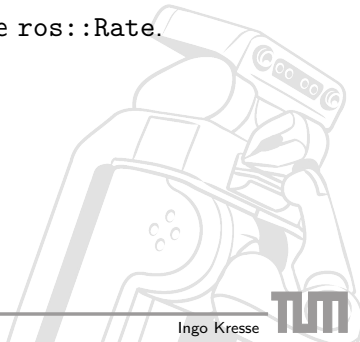
This makes shure that messages are handled and callbacks are called.



Some remarks on ROS

How to organize the main loop:

- ▶ loop rate required by the device: Usually there is a 'wait-for-new-data' anyway.
→ publish every n th data sample.
- ▶ (more or less) arbitrary loop rate: Use `ros::Rate`.
- ▶ Avoid busy waiting (uses 100% CPU)





Some remarks on ROS

Some more things to consider:

- ▶ Callbacks should not block for a long time (everything else will be stalled as well)
- ▶ Keep computational load within bounds (also for the consumers)
- ▶ Set the data rate 'right': (rule of thumb in ROS: 10-200 Hz. Best option: have a rotparam!)
- ▶ Make 'good' time stamps (calibrate delays, if possible)



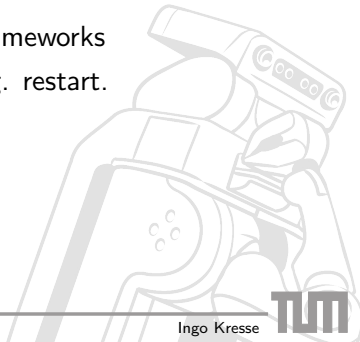
A word on code organization

What about the rest of the code?

- ▶ Encapsulate hardware code into separate class (separate file)
- ▶ Make separate ROS wrapper class.

Advantages:

- ▶ hardware code is reusable in other frameworks
- ▶ ROS wrapper has an easier job to e.g. restart.

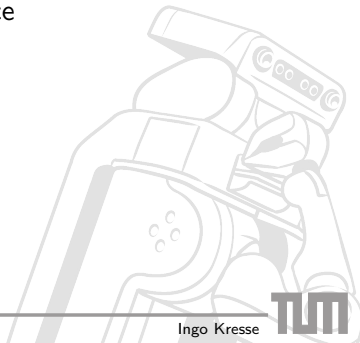




Robustness

How can we make a driver robust?

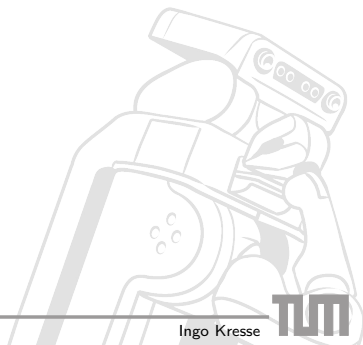
- ▶ check whether data is arriving
- ▶ check whether data is valid
- ▶ when something is wrong: reset device
- ▶ keep trying!





Outline

1. Motivation
2. Example Driver
- 3. Extra Features**
4. Conclusion





ROS Parameters

- ▶ do the job of reading parameters from file or command line
- ▶ stored in a central place (roscore)
- ▶ usually primitive types, but can be structs and lists
- ▶ there are tools to change them online

Usage:

src/cool_new_base.cpp:

```
...  
double max_speed;  
nh.param("max_speed", max_speed, 2.0);  
...
```



How can we set a ROS Parameter?

- ▶ When starting from the command line:

command line:

```
# ./cool_new_base _max_speed:=0.5
```

- ▶ Using the rosparam tool:

rosparam tool:

```
# rosparam set cool_new_base/max_speed 0.5
```

- ▶ In a launch file:

cool_new_base.launch

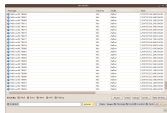
```
...  
<param name="max_speed" type="double" value="0.5" />  
...
```



ROS Logging system



vs.



- ▶ has severity levels (DEBUG, INFO, WARN, ERROR)
- ▶ reports file name and line number
- ▶ travels over the network



ROS Logging system

- ▶ powerful console

Message	Severity	Node	Time
1 hello world 78866	info	talker	1.259701549.869195000
1 hello world 78867	info	talker	1.259701550.089209000
1 hello world 78868	info	talker	1.259701550.169192000
1 hello world 78872	info	talker	1.259701550.269198000
1 hello world 78873	info	talker	1.259701550.369194000
1 hello world 78874	info	talker	1.259701550.469195000
1 hello world 78875	info	talker	1.259701550.569196000
1 hello world 78876	info	talker	1.259701550.669191000
1 hello world 78877	info	talker	1.259701550.769192000
1 hello world 78878	info	talker	1.259701550.869246000
1 hello world 78879	info	talker	1.259701550.869351000
1 hello world 78880	info	talker	1.259701551.069208000
1 hello world 78881	info	talker	1.259701551.169190000
1 hello world 78882	info	talker	1.259701551.269192000
1 hello world 78883	info	talker	1.259701551.369193000
1 hello world 78884	info	talker	1.259701551.469194000
1 hello world 78885	info	talker	1.259701551.569194000
1 hello world 78886	info	talker	1.259701551.669190000
1 hello world 78887	info	talker	1.259701551.769207000
1 hello world 78888	info	talker	1.259701551.869196000
1 hello world 78889	info	talker	1.259701551.969193000
1 hello world 78890	info	talker	1.259701552.069209000
1 hello world 78891	info	talker	1.259701552.169190000
1 hello world 78892	info	talker	1.259701552.269193000
1 hello world 78893	info	talker	1.259701552.369192000

- ▶ all message in one place
- ▶ extensive filtering capabilities



ROS Logging system

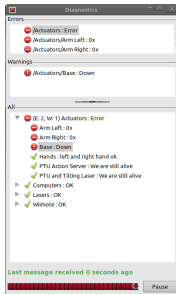
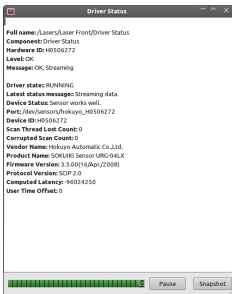
Usage example:

src/cool_new_base.cpp:

```
...  
ROS_INFO("initialization took %f seconds", t_elapsed);  
...
```



Diagnostics



Purpose:

- ▶ keep track of hardware status
- ▶ quickly find out “why it’s not working”

Regularly (every 1s) send some status message, containing drivers state.



Diagnostics

Code example:

src/cool_new_base.cpp:

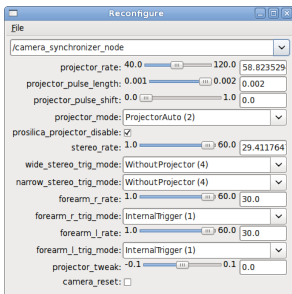
```
void diagsUpdate(diagnostic_updater::DiagnosticStatusWrapper &s)
{
  s.summary(diagnostic_msgs::DiagnosticStatus::OK, "Operational");
  s.addf("important number", "%d", 42);
}

...

diagnostic_updater::Updater diags;
diags.setHardwareID("cool_new_base");
diags.add("Base", diagsUpdate);
...
diags.update();
```




Dynamic reconfigure

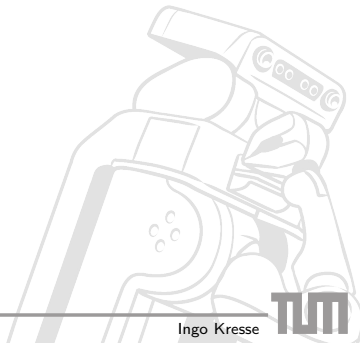


- ▶ Drivers publish details about their parameters (min/max values, etc.) on a topic.
- ▶ From this description a GUI is generated.
- ▶ The changed configuration is sent using a service call.



Outline

1. Motivation
2. Example Driver
3. Extra Features
- 4. Conclusion**



Well, every presentation should have one, right?

- ▶ Philosophy is geared towards systems with a lot of hardware
→ robustness is a must
- ▶ ROS code interface is easy (only a few lines of code)
- ▶ There are many ROS tools that help with common tasks
(rospams, rosout, diagnostics, etc.)

